Escuela Técnica Superior de Ingenieros
Industriales y de Telecomunicación

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Máster

Desarrollo de un Marketplace para la Internet de las Cosas basado en Blockchain

(Development of a Blockchain-based Marketplace for the Internet of Things)

Para acceder al Título de

Máster Universitario en Ingeniería de Telecomunicación

Autor: Iván González Takmovtseva

Septiembre - 2020

Resumen

En un mundo en el que todo está interconectado gracias al Internet de las Cosas, el volumen de información que se intercambia es enorme. Actualmente, con el auge de las técnicas de análisis de datos existentes está información puede resultar atractiva para multitud de empresas para conseguir una ventaja competitiva frente al resto. En este mercado, se necesita que el intercambio de información entre productores y consumidores sea totalmente seguro y transparente. Las características inherentes de la tecnología Blockchain hacen que sea una herramienta muy a tener en cuenta en este tipo de situaciones. Por lo tanto, este TFM estudia el posible nicho de mercado que puede tener la tecnología Blockchain en el ecosistema del Internet de las Cosas para la venta de medidas. Para ello, se desarrolla una plataforma basada en tecnología Blockchain donde los proveedores de infraestructuras de IoT pueden publicar las medidas que producen sus sensores para que aquellos clientes que estén interesados puedan comprarlas.

Abstract

In a world where everything is connected thanks to the Internet of Things technology, the volume of information exchanged is enormous. Currently, with the rise of data science, this information can be very attractive for several companies to achieve a competitive advantage over their competitors. In this market, the information exchanged between the producers and the consumers must be secure and transparent. The inherent properties of Blockchain technology makes it a tool to take into account in this kind of situations. Therefore, this TFM studies the possible marketplace of Blockchain technology in the Internet of things environment. To do so, a Blockchain-based platform is developed where the IoT infrastructure providers can publish the measurements of their sensors so that the customers who are interested in these measurements can buy them.

Índice

Índice de figuras

Lista de Acrónimos

AES Advanced Encryption Standard

API Application Programming Interface

BFT Byzantine Fault Tolerance

CA Certification Authority

CAP Consistency Availability Partition Tolerance

CSS Cascading Style Sheets

DAPP Decentralized APPlication

DB DataBase

ECIES Elliptic Curve Integrated Encryption Scheme

EVM Ethereum Virtual Machine

GCM Galois Counter Mode

HTML Hypertext Markup Language

HTTP HyperText Transfer Protocol

HTTPS HyperText Transfer Protocol Secure

IBFT Istanbul Byzantine Fault Tolerance

IoT Internet of Things

IPC Inter-Process Communication

JSON JavaScript Object Notation

NGSI Next Generation Service Interfaces

OCB Orion Context Broker

PoA Proof of Authority

PoS Proof of Stake

PoW Proof of Work

SQL Structured Query Language

TFM Trabajo de Fin de Master

TTP Third Trusted Party

URL Uniform Resource Locator

Capítulo 1

Introducción

La evolución que ha tenido Internet desde su creación ha sido exponencial, tanto desde el punto de vista de la infraestructura que la soporte como de la información y contenido que circula por ella. En este sentido de los orígenes en que solo unos pocos producían la información que muchos consumían, se ha pasado a la proliferación de productores de información.

Uno de los aspectos más importantes a tener en cuenta en este escenario consiste en mantener la transparencia, gobernanza e integridad de la información en Internet. Es decir, poder garantizar que una entidad o persona ha dicho o ha hecho algo a través de la red en un determinado instante temporal.

Hasta hace poco, era necesario utilizar terceras partes de confianza o TTPs para asegurar que una entidad había realizado una determinada acción. Como por ejemplo una transacción bancaria. Esto implicaba que estas TTP tenían que ser totalmente confiables y por lo tanto siempre tenían la verdad absoluta.

Con la de tecnología de Blockchain se pretende poner fin a este tipo de estrategias, en las cuales se hace uso de terceras partes de confianza para garantizar que se ha realizado una determinada acción, dado que esta tecnología permite que cualquier usuario interesado pueda comprobarlo por si mismo. Al no depender de estas TTPs, se permiten aumentar la velocidad de las transacciones y abaratar costes. Por ejemplo, según una publicación de Oliver Wyman en colaboración con el banco Santander [1], se estima que los bancos podrán reducir entre 15 y 20 billones de dolares en costes de infraestructura asociados a las transacciones internacionales para el año 2022 gracias a la implementación de Blockchain.

Cada vez más empresas están adoptando el uso de Blockchain en el día a día. Algunos ejemplos son el Banco Santander, el cual introdujo en el año 2016 esta tecnología para realizar pagos internacionales [2]. Otro ejemplo de gran compañía que decide adoptar esta tecnología es Walmart. Esta, la utiliza para la trazabilidad de los productos en toda la cadena de suministro [3].

Además, de los dos ejemplos anteriores existen muchas otras compañías, en diversos ámbitos, que hacen uso de Blockchain como BBVA, British Airways, el gobierno de Dubai, Ford, Siemens, HSBC etc [4].

Una de las tecnologías en las que la proliferación de productores de información se agudiza, es la Internet de las Cosas o IoT. Esta tecnología permite la comunicación entre diferentes

dispositivos sin necesidad de ningún tipo de interacción humana. Se trata, por lo tanto, de una tecnología habilitadora de un escenario en el que todo lo que nos rodea, podría llegar a generar y consumir información de una manera desatendida y transparente, ofreciendo y consumiendo nuevos servicios basados en dicha información.

Algunos ejemplos de escenarios IoT son las smart grids o redes eléctricas inteligentes, hogares inteligentes, comunicaciones entre vehículos, etc.

Se estima que para el año 2025 el número de dispositivos IoT interconectados será cerca de 22 000 millones [5]. Además, se prevé que para el año 2026 la cuota de mercado de esta tecnología será cerca de 13 billones americanos de dolares [6]. Por lo tanto, como se puede observar, es una tecnología con un enorme impacto en el futuro.

1.1. Motivación

En este contexto, el presente trabajo de Fin de Master se plantea una serie de motivaciones que subyacen a su realización y a los objetivos que persigue.

En primer lugar, se busca comprender de una manera más detallada el funcionamiento de la tecnología Blockchain, y con ello, a continuación, plantear su uso en un ecosistema IoT en el que participen múltiples productores y consumidores de información y en el que se puedan generar un mercado de datos seguro y confiable sin la intermediación de terceras partes de confianza que gobiernen el mercado y los datos de una manera centralizada.

1.2. Objetivos

Para poder alcanzar la visión que se ha descrito en el apartado anterior, se han definido una serie de objetivos específicos que se detallan a continuación:

- Diseñar una plataforma, que mediante tecnología Blockchain, permita la compra-venta de medidas en un entorno IoT. De tal manera, que una entidad que controla diferentes productores o sensores puedan utilizar esta tecnología para vender las medidas que estos producen.
- Garantizar, mediante el uso de Blockchain, que el intercambio de información sea totalmente transparente tanto para los clientes como para el proveedor de las medidas. Es decir, en la Blockchain se debe reflejar quién ha comprado un dato, cuándo ha realizado la compra y que el cliente ha recibido dicho dato. Todo esto, de manera pública para que cualquier usuario o entidad pueda comprobarlo.
- Garantizar que el cliente se asegura que la medida que ha comprado proviene de un sensor autorizado por el vendedor. Por lo tanto, el cliente sabe en todo momento que la medida que está comprando es veraz.
- Desarrollar una interfaz que permita a los clientes observar las medidas disponibles y comprarlas de manera intuitiva.

1.3. Resumen ejecutivo del trabajo

En el Capítulo 2 se describen los diferentes aspectos teóricos básicos que son necesarios conocer para poder entender el desarrollo del proyecto.

A continuación, en el Capítulo 3, se expone el diseño y desarrollo de la plataforma. En la parte de diseño, se describen los diferentes elementos que componen la plataforma y sus roles dentro de esta. En la sección de desarrollo, se detalla el desarrollo de los diferentes componentes que integra la plataforma.

En el Capítulo 4 se valida el correcto funcionamiento de la plataforma mediante una prueba de concepto. Para ello, se despliega la plataforma en un entorno controlado y se emula el funcionamiento de los clientes y de la infraestructura IoT.

Finalmente, en el Capítulo 5, se analiza el resultado del proyecto y las posibles líneas de desarrollo que se pueden seguir en el futuro.

Capítulo 2

Marco teórico y práctico del proyecto

A lo largo de este capítulo se van a exponer los aspectos más relevantes que conforman el marco de desarrollo de este trabajo de Fin de Máster (TFM). Este repaso es fundamental para permitir al lector seguir la descripción de las tareals llevadas al cabo durante la realización del TFM que se describe en los siguientes capítulos.

2.1. Sistemas distribuidos

Antes de explicar el concepto de Blockchain, es necesario entender qué es un sistema distribuido dado que, en su núcleo, Blockchain es básicamente un sistema distribuido. De manera más precisa, es un sistema distribuido y descentralizado [7].

Un sistema distribuido es un paradigma computacional en el que dos o más nodos colaboran entre ellos para conseguir un determinado fin.

El desafío principal del diseño de este tipo de tecnologías es la coordinación entre los nodos y la tolerancia de fallo. A pesar de que algunos nodos colaboran de manera defectuosa o pierden la conexión con otros nodos, el sistema distribuido debería de ser capaz de tolerar esto y de conseguir el resultado deseado.

El problema de este tipo de sistemas es el teorema de CAP o teorema de Brewer [8]. Este teorema estipula que en un sistema distribuido no pueden cumplir las siguientes propiedades de manera simultanea.

- Consistencia: Todos los nodos en el sistema distribuido tienen una copia individual del último dato.
- Disponibilidad: El sistema responde a las peticiones de los usuarios de manera correcta a pesar de que haya uno o más nodos caídos.
- Tolerancia al particionado: Si un grupo de nodos pierden la conexión con el sistema, este sigue funcionado de manera correcta.

A pesar de que ha sido demostrado que un sistema distribuido no puede tener las tres propiedades funcionando de manera simultanea [9], Blockchain de alguna manera si que lo consigue. Más adelante se explicará cómo.

2.2. Blockchain

Desde un punto de vista tecnológico, "una cadena de bloques (Blockchain), también conocida como libro de contabilidad distribuido (distributed ledger), es una base de datos distribuida que registra bloques de información y los entrelaza para facilitar la recuperación de la información y la verificación de que ésta no ha sido cambiada" [10].

Simplificándolo mucho, se puede definir como una base de datos distribuida en modo *append*. Es decir solo se puede añadir contenido, no se puede modificarlo o elimnarlo. Esta tecnología tiene una serie de propiedades que la hacen muy adecuadas en diversos escenarios [11]:

- Inmutabilidad: Una vez que se añade contenido a la Blockchain, este no puede ser modificado o eliminado.
- Transparencia: Cualquier usuario que tenga acceso a la Blockchain puede observar su contenido en cualquier momento. De tal manera, que un usuario que introduce un dato en la Blockchain puede verificar su estado y comprobar en que operaciones se ha utilizado.
- Eficiencia: Permite a los negocios llevar al cabo operaciones de manera eficiente y económica. No hay necesidad de utilizar terceras partes de confianza.
- Trazabilidad: Permite observar el registro de manera continua. Esto permite, evitar actividades fraudulentas
- Seguridad: Utiliza diversas técnicas criptográficas que garantizan la inmutabilidad de los datos y su proveniencia.

Por otro lado, desde una perspectiva de negocio, Blockchain se puede definir como una plataforma en la que los usuarios intercambian valores utilizando transacciones sin que haya ninguna TTP que controle estas transacciones. Esto permite que la Blockchain sea un mecanismo descentralizado de consenso donde no existe ninguna autoridad que controle la base de datos.

2.2.1. Elementos que componen una Blockchain

El elemento principal de una Blockchain es el bloque. Un bloque es, simplemente, una agrupación de información digital que correspondería a una página del libro de contabilidad. En estos se almacenan las diferentes transacciones que se producen en la Blockchain. Cada bloque, excepto el genesis[1], incluye una referencia del bloque anterior (su hash). De tal manera, que si un individuo malintencionado desea modificar el contenido de un bloque antiguo, tendría que cambiar, además, todos los bloques posteriores.

En la Figura 2.1 se puede observarla relación, mediante el hash, entre los diferentes bloques de la Blockchain.

[1]Primer bloque que se creó durante la creación de la Blockchain.

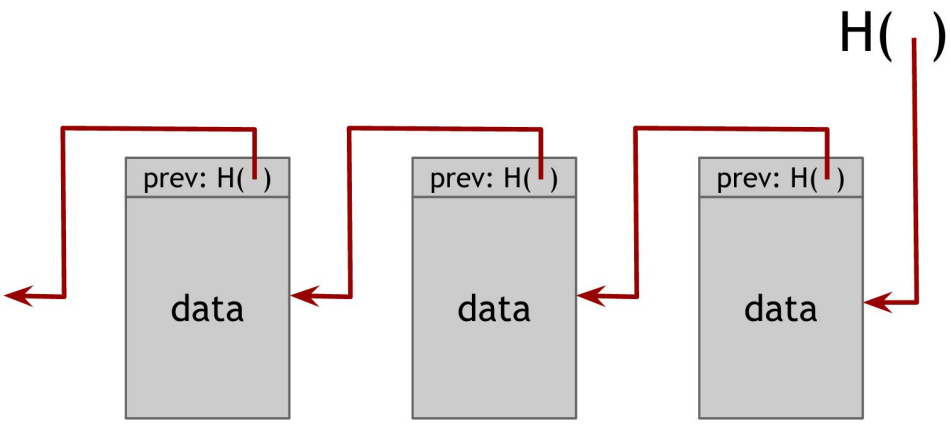

Figura 2.1: Esquema de la relación entre los bloques. [12]

La estructura de los bloques depende principalmente del tipo y el diseño de la Blockchain, pero generalmente existen una serie de atributos que son esenciales en cualquier diseño. Estos son: cabecera del bloque, puntero al bloque anterior, sello temporal o time-stamp, nonce, contador de transacciones y transacciones.

En la Figura 2.2 se puede observar los diferentes campos que componen un bloque en la Blockchain de Bitcoin [13].

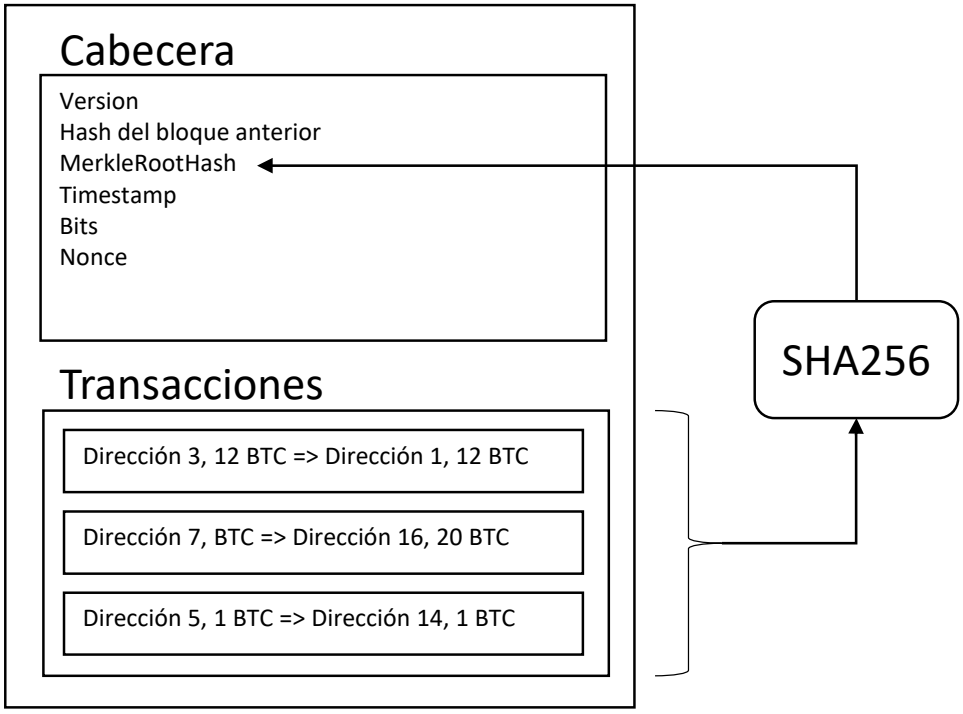

Figura 2.2: Bloque en Bitcoin

Como se ha comentado previamente cada Blockchain se puede diseñar de distinta forma dependiendo de su funcionalidad. Sin embargo, existen una serie de elementos genéricos que componen una Blockchain:

- Direcciones: Identificadores únicos que se utilizan en las transacciones para representar al transmisor y al receptor. Habitualmente se derivan las direcciones de los usuarios a partir de sus claves públicas. Normalmente, realizando una operación de hash sobre esta. Por lo tanto, las direcciones son únicas. En la práctica, sin embargo, un usuario no tiene porque utilizar siempre la misma dirección sino que puede generar nuevas direcciones. De echo, Blockchain es un sistema pseudoanónimo.
- Transacción: Unidad fundamental de Blockchain que representa la transferencia de información de una dirección a otra.
- Bloque: Elemento básico de almacenamiento de la información.
- Scripting o lenguaje de programación: Conjunto de comandos o instrucciones que permiten realizar diferentes operaciones básicas sobre las transacciones.
- Máquina virtual: Permite correr programas, denominados smart contracts, sobre la Blockchain para realizar diferentes funciones. Cabe destacar que no todas las Blockchains tienen una máquina virtual como por ejemplo Bitcoin.
- Máquina de estados: Una Blockchain se puede entender como un mecanismo de transición de estados, en el que un estado cambia su forma inicial a su siguiente forma como resultado de una transacción.
- Nodos: La tarea básica de un nodo dentro de la Blockchain consiste en la validación y verificación de las transacciones. Además, dependiendo del algoritmo de consenso de la Blockchain y su rol dentro de este, puede realizar diferentes funciones. Más adelante se detallarán los distintos mecanismos de consenso que se utilizan hoy en día.
- Contratos inteligentes o Smart Contracts: Programas que se ejecutan sobre la máquina virtual de la Blockchain cuando se cumplen unas determinadas condiciones. Este paradigma es una de las características principales de Blockchains como Ethereum, Hyperledger, etc.

2.2.2. Funcionamiento básico de las Blockchain

A continuación se va a explicar de manera resumida, en cinco pasos, el proceso de almacenamiento de los bloques que utilizan las Blockchain.

1. Un nodo genera una transacción en la Blockchain y la firma con su clave privada.
2. La transacción se propaga por los otros nodos, los cuales validan la transacción en función de un determinado criterio. Normalmente, es necesario que más de un nodo validen dicha transacción.
3. Una vez que se ha validado la transacción, se incluye en un bloque y se propaga por la red. En este punto, se considera que la transacción ha sido confirmada con éxito.
4. El bloque generado se convierte en un nuevo bloque dentro del ledger de la Blockchain y se enlaza con el próximo bloque se vaya a generar. Como se ha comentado antes, este puntero se realiza mediante el hash del bloque. En este momento, la transacción obtiene su segunda confirmación y el bloque su primera.

5. Las transacciones requieren múltiples confirmaciones para que se consideren transacciones finales. Por ejemplo, en Bitcoin se considera que una transacción es la final cuando se ha confirmado seis veces.

En la Figura 2.3 se muestran los puntos comentados anteriormente de manera gráfica.

El aspecto clave en este proceso es el de la validación de los bloques. Este proceso requiere que los nodos de la Blockchain alcancen un acuerdo sobre el bloque que se va a incluir y las transacciones que lo componen. Estos acuerdos se basan en distintos mecanismos de consenso. En el siguiente apartado se va a explicar qué es un algoritmo de consenso y cuáles son los más utilizados hoy en día.

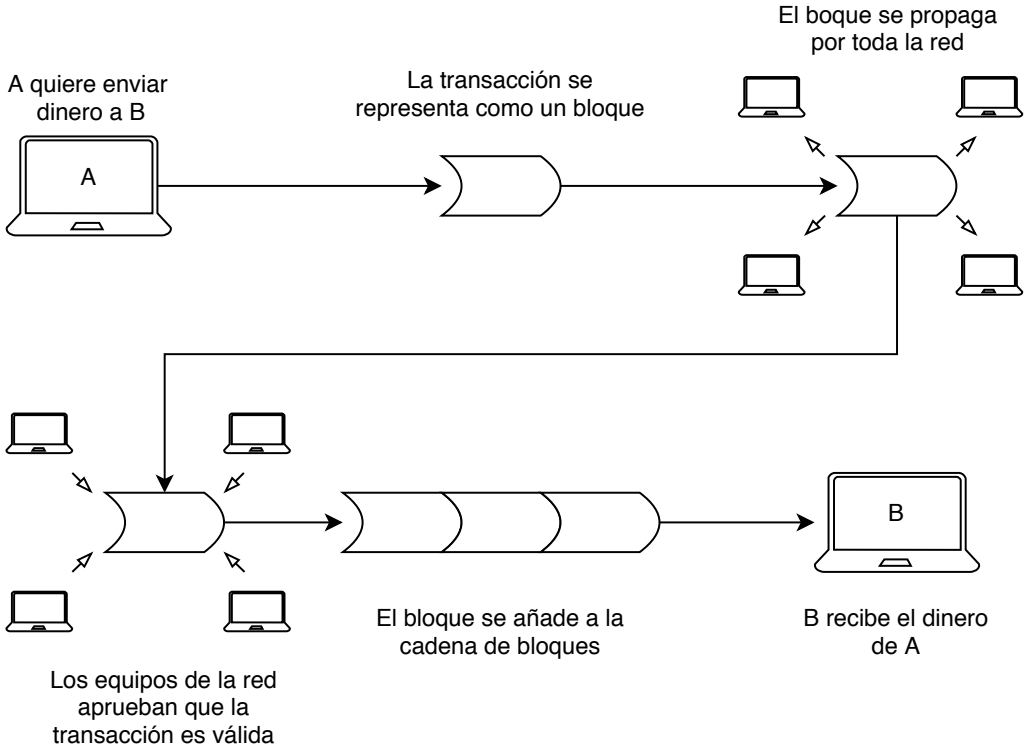

Figura 2.3: Funcionamiento de una Blockchain.

2.3. Algoritmos de consenso

Un algoritmo de consenso se puede definir como un mecanismo, que utilizan las Blockchain, que establece un conjunto de reglas y procesos para alcanzar un acuerdo entre un conjunto de nodos de manera distribuida. Cabe destacar que para llegar al acuerdo final no es necesario que todos los nodos estén de acuerdo, con que lo esté la mayoría es suficiente.

Un ejemplo de un mecanismo de consenso, en el día a día, podría ser un sistema de votación en el que un grupo de personas decide hacer una tarea o no. Todo el mundo sugiere una idea y se votan las diferentes ideas. La que obtenga más votos es la que se realiza, a pesar de que haya gente que haya votado en contra.

En un mecanismo de consenso, existen múltiples requisitos que se deben cumplir para

alcanzar el resultado deseado. Estos son:

- Acuerdo: Todos los nodos honestos deciden sobre el mismo valor.
- Terminación: Todos los nodos terminan la ejecución del proceso de consenso y alcanzan una decisión.
- Validez: El valor acordado por todos los nodos honestos debe ser el mismo que el valor inicial propuesto por uno de los nodos honestos.
- Tolerancia al fallo: El algoritmo de consenso debe ser capaz de funcionar en presencia de nodos erróneos o maliciosos.
- Un nodo no pude tomar más de una decisión sobre un mismo valor. Es decir, para un determinado valor, los nodos solo pueden decidir una vez.

De manera simplificada, se pueden agrupar los diferentes algoritmos de consenso en dos grandes grupos:

1. Basados en un esfuerzo (Proof-based) o basados en un líder (Leader-based), los cuales se basan en la elección de un líder que es el encargado de proponer el valor.
2. Basados en el criterio de tolerancia de fallo bizantina (Byzantine fault tolerance-based), el cual se trata de un enfoque más tradicional basado en votaciones.

La selección del tipo mecanismo de consenso depende de las necesidades de la Blockchain. Existen tres tipos básicos de Blockchain: públicas (ninguna entidad las gobierna), de consorcio (gobernadas por múltiples entidades) y privadas (gobernadas por una única entidad). Cada uno de estos tipos tiene diferentes aplicaciones. El mecanismo de consenso adoptado se tiene que ajustar a la aplicación específica para la que va a ser utilizado [14].

A continuación se detallan algunos de los mecanismos más utilizados.

2.3.1. Proof of Work (PoW)

Este tipo de mecanismo fue originalmente adoptado por Bitcoin y se utiliza en otras Blockchain como por ejemplo en la Blockchain pública de Ethereum (Mainnet). En este algoritmo se utilizan los recursos computacionales de los participantes para la generación de bloques. De tal manera que, el usuario que primero resuelva un puzzle criptográfico complejo es el encargado de generar el nuevo bloque. Este concepto se denomina minar.

En la Figura 2.4 se puede observar el flujo de la creación de bloques en este algoritmo.

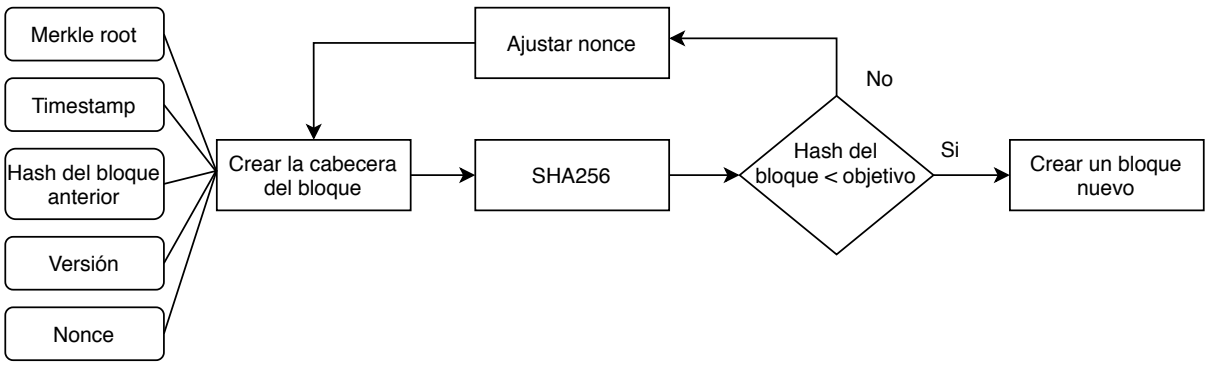

Figura 2.4: Mecanismo PoW. [14]

Para que un usuario malicioso sea capaz de hacerse con el control de la red, es decir se el encargado de decidir qué bloque se introduce y cuál no, es necesario que tenga más del 51 % de la fuerza computacional de la red. Debido a que muchos usuarios están utilizando baterías de tarjetas gráficas para minar bloques, muchas criptomonedas están pensando en dejar de utilizar este consenso y empezar a utilizar Proof of Stake. Como por ejemplo la mainnet the Ethereum [15].

2.3.2. Proof of Stake (PoS)

Este algoritmo, a diferencia del anterior, selecciona al usuario que crea el nuevo bloque en función de la cantidad de criptomonedas que haya invertido para minar dicho bloque.

Al igual que en el caso de PoW, los usuarios también tienen que resolver un rompecabezas criptográfico. Salvo que en este caso, los usuarios no tienen que ajustar el nonce, sino que la clave para resolver el puzzle es la cantidad de criptomonedas que posee.

Un factor fundamental a la hora de resolver dicho puzzle es la antigüedad de la criptomoneda. Esta se deriva a partir de la cantidad de tiempo que el usuario ha tenido la moneda sin gastarla. Por ejemplo, si un usuario posee 10 monedas durante un total de 20 días, entonces la antigüedad de la moneda es de 200. Una vez que el usuario crea un nuevo bloque, la edad de su moneda se pone a 0.

PoS es un mecanismo de consenso que está pensado para que los nodos ahorren energía, dado que en vez de gastar muchos recursos computacionales en resolver el puzzle se hace uso de las criptomonedas que tiene el usuario.

En la Figura 2.5 se puede observar el flujo de creación de bloques en PoS.

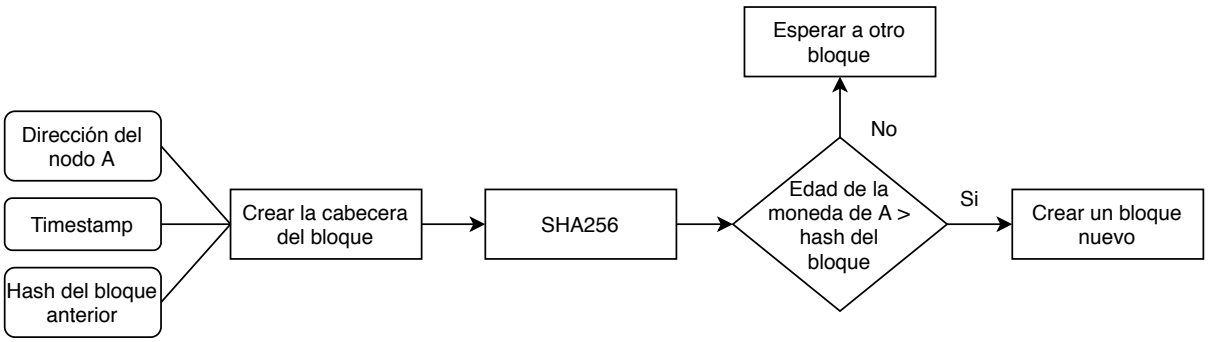

Figura 2.5: Mecanismo PoS. [14]

2.3.3. Proof of Authority (PoA)

PoA es un un algoritmo de consenso que provee un alto rendimiento y tolerancia de fallos. En este mecanismo, solo generan nuevos bloques aquellos nodos que tengan la autoridad para hacerlo.

Comparado con los algoritmos de consenso anteriores, PoA tiene multitud de ventajas:

- PoA no necesita grandes recursos computacionales en la resolución de puzzles criptográficos complejos.

- Se puede definir el intervalo temporal que pasa en la generación de nuevos bloques. En los anteriores mecanismos, el tiempo entre la generación de nuevos bloques era aleatorio, ya que dependía de la dificultad del rompecabezas y el número de mineros.
- Alta velocidad de transacciones. Los bloques se generan durante el intervalo de tiempo especificado, incluso es posible el minado a demanda. Es decir, los bloques se generan en el momento que haya una transacción, de tal manera, que no es necesario esperar. En caso de que no se produzcan transacciones durante este intervalo de tiempo, se generarán bloques vacíos.
- Mayor tolerancia frente a la existencia de nodos maliciosos. En este caso, para que un usuario maligno pueda controlar la generación de bloques es necesario que se apropie de más del 51 % de los nodos que conforman la Blockchain.

Existen múltiples implementaciones de este mecanismo de consenso, algunos de los ejemplos más usados son Clique e IBFT.

Clique

Clique es un mecanismo de consenso, de tipo PoA, utilizado en las redes privadas de Ethereum donde no es necesario el uso de recompensas a la hora de minar los bloques.

En este algoritmo se diferencian dos tipos de nodos según su relación con el minado de bloques. Nodos validadores y nodos no validadores.

El primer grupo está compuesto por aquellos nodos capaces de generar nuevos bloques en la Blockchain. Su principal tarea consiste en comprobar la validez de las transacciones e insertarlas en un nuevo bloque. Para que un bloque sea aceptado en la Blockchain tiene que haber sido validado por más del 50 % de los nodos validadores. Al igual que ocurre con otros tipos de PoA, se puede definir el intervalo de tiempo entre generación de bloques nuevos. Incluso, se puede definir que dicho intervalo sea 0 (minado a demanda).

Por otro lado, los nodos no validadores no pueden introducir nuevos bloques en la Blockchain. Su única funcionalidad consiste en la generación de transacciones y en la proposición de nuevos nodos validadores.

Clique permite que todos los miembros de la Blockchain elijan los nodos validadores que pueden autorizar las transacciones. Para añadir un nuevo nodo de este tipo, el resto de los nodos validadores deben de aceptar la propuesta. Lo mismo ocurre en el caso contrario, es decir, para que un nodo deje de ser validador.

Istanbul Byzantine Fault Tolerance (IBFT)

Istanbul BFT o IBFT es una implementación práctica del algoritmo Byzantine Fault Tolerance (BFT) con algunas ligeras modificaciones. En este mecanismo, para que un nuevo bloque se inserte en la Blockchain, es necesario que múltiples nodos validadores acepten o voten a favor de este bloque y, además, que lo firmen.

Por definición, BFT significa que la red debe de continuar a pesar de la existencia de nodos maliciosos que pretenden incluir bloques inválidos. En particular, este algoritmo, puede tolerar hasta F nodos maliciosos en una red compuesta por 3F + 1 nodos. Esto implica que la red puede tolerar, aproximadamente, que 1/3 de sus nodos sean deshonestos.

Al igual que ocurre en otos mecanismos de tipo PoA, en IBFT el intervalo de tiempo

entre la generación de nuevos bloques también es constante.

Resumiendo, este algoritmo tiene las siguientes ventajas e inconvenientes [16].

Ventajas

- Protege a la red frente a un 30 % de nodos deshonestos.
- Muy difícil de modificar su contenido puesto que, para que esto ocurra, un usuario maligno necesitaría todas las claves privadas de los nodos validadores que firmaron el bloque.

Inconvenientes

- Los bloques se minan en intervalos de tiempo constantes, lo que puede dar lugar a la generación de bloque vacíos.
- Gran número de mensajes de cabecera, los cuales aumentan a medida que se incrementan el número de nodos validadores.

2.4. Ethereum

Ethereum es una Blockchain de código libre, pública y cuya moneda principal es el ether. A diferencia de otras Blockchain, como Bitcoin, Ethereum no solo permite el envío de transacciones sino que además es programable. Esto implica que los desarrolladores pueden crear programas que funcionen sobre la Blockchain.

2.4.1. Aplicaciones descentralizadas

Las aplicaciones descentralizadas o DAPPs (Decentralized APPlication) son recursos programables que se cargan en la Blockchain y funcionan siempre de manera predecible. Esto se debe a que una vez que una DAPP se carga en la Blockchain, no es posible su modificación ni eliminación. El elemento principal de este tipo de aplicaciones es el denominado contrato inteligente o smart contract.

Un smart contract es un recurso programable que se ejecuta directamente sobre la maquina virtual de la Blockchain y que interactúa con esta. Para la creación de este tipo de programas, generalmente se usa el lenguaje de programación Solidity. En la Figura 2.6 se muestra un ejemplo de contrato inteligente cuyo objetivo consiste en almacenar el valor de una variable en la Blockchain.

El contrato consta de dos funciones y de una variable (`storedData`). La función `set` toma como parámetro de entrada un entero sin signo y almacena dicho valor en la variable `storeData`.

Por otro lado, la función `get`, no recibe ningún valor de entrada y devuelve el valor almacenado en la variable `storeData`.

Estos programa se ejecutan sobre la máquina virtual de Ethereum o (EVM). Esta máquina está bastante delimitada y completamente aislada, lo que significa que el código que se ejecuta dentro de ella no tiene ningún acceso a la red, ficheros del sistema o otros procesos. Incluso los contratos inteligentes tienen un acceso limitado a otros contratos.

```
pragma solidity ^0.4.0;

contract SimpleStorage {
 uint storedData;

 function set(uint x) public {
     storedData = x;
 }

 function get() public view returns (uint) {
     return storedData;
 }
}
```

Figura 2.6: Ejemplo de Smart Contract. [17]

2.4.2. Arquitectura de la Blockchain de Ethereum

En Etherum existen tres tipos de nodos según su funcionalidad: nodos completos, nodos ligeros y nodos archivo.

Nodos completos

Los nodos completos realizan las siguientes tareas:

- Almacenan todo el contenido de la Blockchain y pueden ofrecer dicho contenido a la red.
- Reciben nuevas transacciones y validan sus orígenes.
- Almacenan únicamente el estado actual de la Blockchain.
- Cualquier estado de la Blockchain se puede derivar a partir de este tipo de nodos.

Nodos ligeros

Los nodos ligeros tienen las siguientes características:

- Únicamente almacenan las cabeceras de los bloques. En el caso de que sea necesario obtener también su contenido, lo solicitan a algunos de los nodos completos.
- Verifican el origen de las transacciones.

Este tipo de nodos son útiles en dispositivos con baja capacidad de cómputo o que requieran poco consumo, como por ejemplo teléfonos móviles, los cuales no pueden permitirse almacenar demasiada información.

Nodos Archivo

Este tipo de nodos tienen las siguientes características:

- Almacenan el mismo contenido que los nodos completos.
- Construyen un archivo histórico con todos los estado de la Blockchain.

Este tipo de nodos solo son necesarios si se desea comprobar el estado de un cuenta en un determinado instante temporal/bloque. Habitualmente, este tipo de nodos son utilizados

por servicios de tipo Blockchain explorer, cuya tarea consiste en monitorizar el contenido de la Blockchain.

Ethereum utiliza principalmente dos tipos de mecanismos de consenso. Para redes públicas, ethereum utiliza PoW. Sin embargo, para redes privadas, aquellas que no están conectadas a la red pública, se utiliza Clique.

2.4.3. Características singulares de Ethereum

A parte de los contratos inteligentes, la Blockchain de Ethereum tiene una serie de elementos que la hacen particular. A continuación se detallan algunos de ellos.

Cuentas

En Ethereum existen dos tipos de cuentas: Cuentas externas controladas por una pareja de claves pública-privada y cuentas de contratos controladas por el código que se almacena con una cuenta.

La dirección de una cuenta externa de Ethereum se extrae a partir de la clave pública del usuario. Este tipo de direcciones se asocian con cada uno de los usuarios de la Blockchain.

Cabe destacar que para que un usuario pueda acceder a su cuenta de Ethereum necesita una contraseña. Esto se debe a que las claves privadas de los usuarios se almacenan cifradas con AES en el nodo en un fichero. En la Figura 2.7 se puede observar el proceso de obtención de una clave privada.

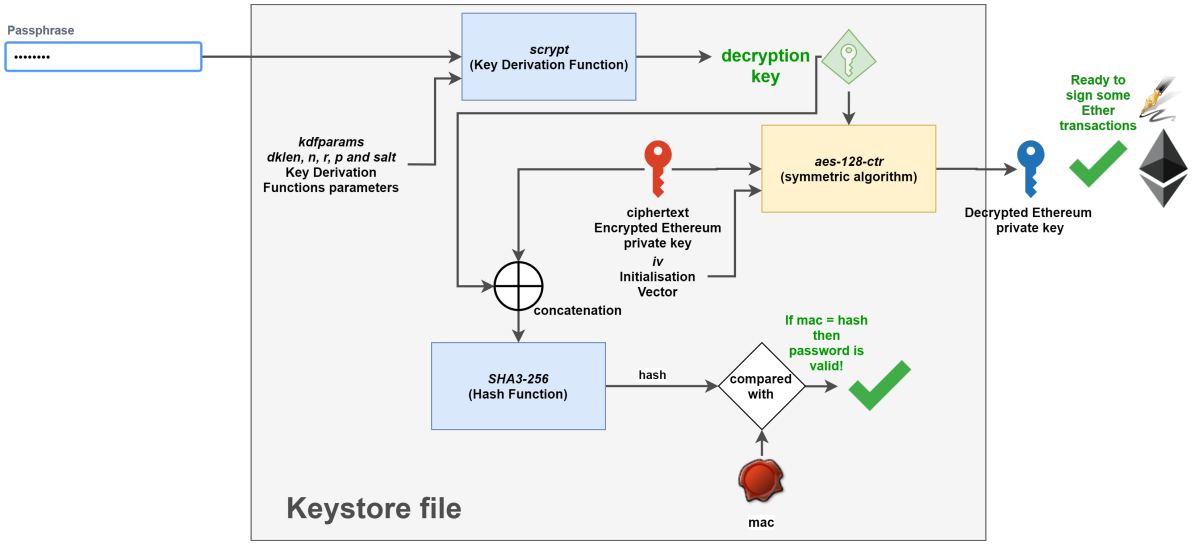

Figura 2.7: Proceso de login de un usuario para obtener su clave privada.

Por otro lado, la dirección del contrato se genera a partir de la dirección del usuario que lo creó, el número de la transacción en el que se envío y un número aleatorio (nonce).

Gas

Cuando se envía una transacción desde un nodo a otro, esta se carga con un determinado gas, cuyo objetivo consiste en limitar la cantidad de trabajo necesaria para ejecutar la transacción y para pagar por dicha ejecución. A mayor gas, más dinero tiene que pagar el usuario para garantizar el impuesto de transacción.

El precio del gas lo estipula el creador de la transacción. Este, en total tiene que pagar como impuesto: $tax = gas\_price \cdot gas$.

Cabe destacar que el gas puede ser 0, pero el problema es que los nodos que estén minando transacciones para conseguir Ether lo más probable es que no se interesen por dicha transacción. Lógicamente, los nodos intentan minar aquellas transacciones cuyo impuesto es mayor para así conseguir mayor recompensa.

Almacén, memoria y stack

Cada cuenta tiene asociado un área de memoria en la EVM denominado Almacén (Storage). En este, se almacenan las direcciones de los contratos que generó el usuario.

Otro área de la memoria se denomina memoria (memory). En este área se almacenan las variables de los contratos.

La EVM es una máquina de tipo pila, por lo tanto todas las operaciones computacionales se ejecutan sobre un stack. Este puede contener como máximo 1024 elementos y su tamaño de palabra es de 256 bits.

Registros o logs

En Ethereum es posible la generación de eventos, en un smart contract, cuando se ha realizado una determinada operación. Un evento es un tipo de dato estructurado, que es fácilmente analizable fuera de la Blockchain. Los eventos pueden ser filtrados en función de múltiples parámetros como por ejemplo dirección que originó el evento, hash de la transacción, índices personalizados, etc.

Los registros o logs se utilizan para almacenar este tipo de variables.

2.5. Quorum

Quorum es una Blockchain, basada en Ethereum, desarrollada con el fin de proveer servicios en finanzas, cadenas de suministro, etc. A diferencia de Ethereum. Quorum es una Blockchain privada o permisionada, por lo que es capaz de soportar contratos y/o transacciones privadas.

Quorum soporta dos modos de funcionamiento simultáneos:

- Modo público: Accesible por todos los usuarios de la Blockchain.
- Modo Privado: Solo pueden acceder los nodos que tengan permiso.

Esta Blockchain está compuesta por dos elementos: Nodos quorum y gestores de privacidad. Los primeros, son modificaciones de los nodos completos Ethereum para el soporte de transacciones y contratos privados.

Por otro lado, los gestores de privacidad se encargan de gestionar las claves asociadas a los nodos. De tal manera que hacen que el intercambio de información confidencial sea posible.

En la Figura 2.8 se puede observar la arquitectura de Quorum. En esta aparecen dos elementos dentro del gestor de privacidad. El gestor de transacciones y el enclave. El primero es el responsable de la privacidad de las transacciones. Almacena y permite

el acceso a los datos de las transacciones cifradas y permite el envío de transacciones privadas a otros nodos de la red.

Por otro lado, el enclave se encarga de realizar el cifrado/descifrado de los datos y la generación de claves simétricas para dicho cifrado.

También se puede observar un elemento denominado go-ethereum. Este es simplemente el cliente que usa Ethereum para interactuar con la Blockchain. En este caso, hace uso del cliente programado en Golang.

Esta Blockchain permite el uso de dos algoritmos de consenso: IBFT y Raft.

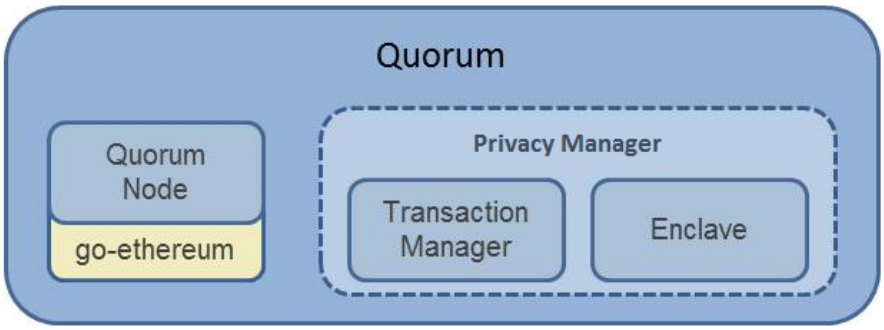

Figura 2.8: Arquitectura de Quorum. [18]

2.6. FIWARE

Como se ha descrito en el capitulo 1, una de las motivaciones de este trabajo es la integración de la tecnología Blockchain con el mundo IoT. En este ecosistema, aparte del despliegue de infraestructura hardware en el entorno, uno de los aspectos clave es la capacidad de acceder a la información generada desde esa infraestructura. Para ello, es necesario disponer de una plataforma software que facilite dicho acceso. Una de las plataformas que se utilizan en este ecosistema es FIWARE. Este, es una iniciativa de código libre que define una serie de estándares para la gestión y el intercambio de información proveniente de diversas fuentes que facilitan el desarrollo de soluciones inteligentes para diferentes ámbitos de aplicación de la IoT como, por ejemplo, ciudades inteligentes, industria, smart grid, etc [19].

El elemento fundamental de la plataforma de IoT de FIWARE es el Orion Context Broker (OCB). Este componente se encarga de administrar la información de contexto, consultarla y actualizarla. “El Context Broker está rodeado por un conjunto de componentes adicionales, que permiten suministrar datos de contexto de diversas fuentes (Internet de las Cosas, robots y sistemas de terceros) y brindan soporte para el procesamiento, análisis y visualización de datos, así como para control de acceso a datos, publicación o monetización” [20].

El OCB permite la publicación de información de contexto por entidades (llamados productores de contexto) como por ejemplo sensores, de manera que la información de contexto publicada se encuentre disponible para otras entidades (consumidores de contexto) los cuales están interesados en procesar la información, por ejemplo una aplicación para smartphones que usa la información de los sensores.

El OCB permite la publicación de información por parte de los productores de contexto, por ejemplo sensores y asociarla a un determinado tema (topic). Otras entidades, denominadas consumidores de contexto, pueden recibir la información publicada en Orion mediante un sistema de suscripciones en las que los topic se utilizan como mecanismo de enrutado y filtrado de la información. Por ejemplo, si un sensor pública información en Orion cuyo contexto es *el tiempo en Santander*, los consumidores interesados podrían obtener dicha información suscribiéndose a dicho tema.

En la Figura 2.9 se puede el esquema de funcionamiento del OCB. En esta figura se puede ver como los consumidores lo primero que hacen es realizar una petición en la que se suscriben a uno o varios contextos. Después, es el OCB el encargado de notificar la nueva información a las que los consumidores estén suscritos, cada vez que alguno de los productores se actualice.

También se puede comprobar que Orion está conectado a una base de datos. Esta se encarga de almacenar la información de contexto más actualizada que existe, la última medida de cada entidad que integra un determinado contexto y las suscripciones de los consumidores. Es importante destacar que esta base de datos no almacena información histórica. Es decir, solo guarda la última medida relativa a cada entidad. Los consumidores son los responsables de almacenar las medidas que les interesen.

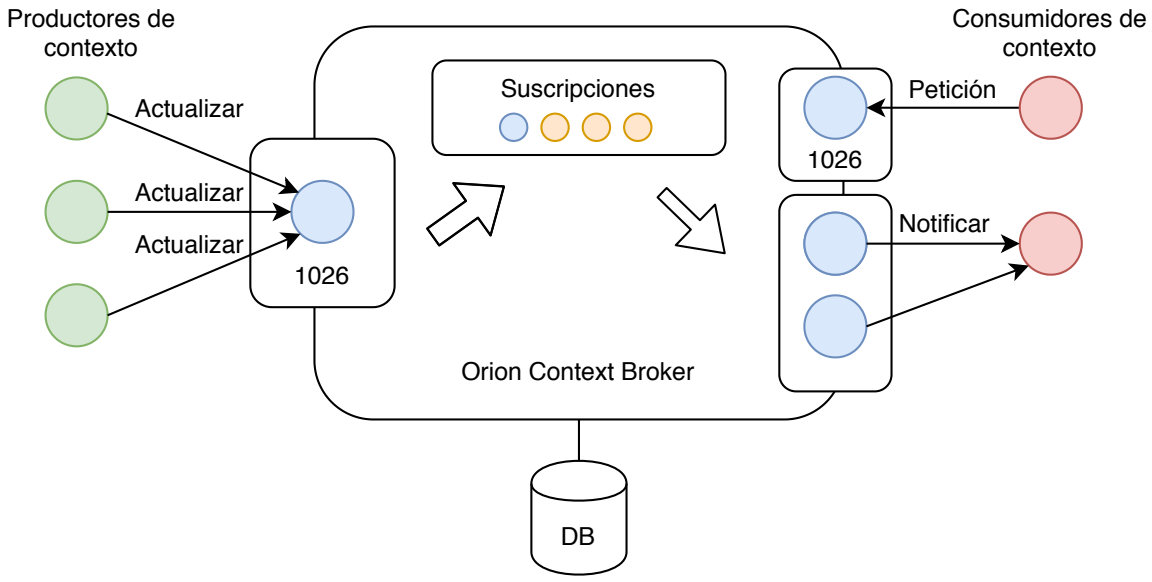

Figura 2.9: Esquema de funcionamiento del OCB. [21]

Toda la comunicación entre los distintos componentes de la arquitectura de alto nivel del OCB se realiza a través de API FIWARE NGSI v2. El API NGSIv2 es una especificación que define:

- Un modelo de datos para la información de contexto.
- Una interfaz para el intercambio de la información por medio de operaciones de consulta, suscripción y actualización. Este intercambio se produce mediante peticiones HTTP (HyperText Transfer Protocol) cuyo cuerpo se encuentra en formato JSON.

Respecto al modelo de datos definidos para NGSIv2, la información de contexto se representa a través de estructuras de datos denominadas Elementos de Contexto o ContextElements (Figura 2.10), las cuales tienen asociadas los siguientes campos:

- Un EntityId y un EntityType único que se utiliza para identificar el contexto.
- Un conjunto de uno o más atributos. Por ejemplo, en el caso de un sensor el atributo correspondería con el valor medido.
- Metadatos opcionales vinculados a cada uno de los atributos. Por ejemplo, la precisión de la medida de dicho sensor.

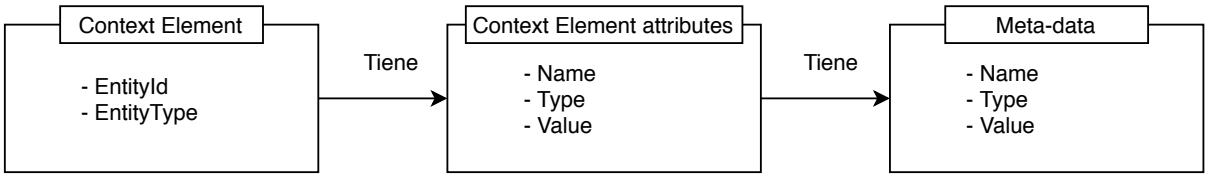

Figura 2.10: Estructura de un elemento de contexto. [21]

En lo que se refiere al intercambio de información, en NGSIv2 se detalla el ciclo de vida completo de Entidades y Context Elements. En este ciclo de vida, destacan, por ser los años habituales, las siguientes funcionalidades:

Para que un productor publique sus medidas en el OCB, lo primero que debe hacer es crear una Entidad (identifcada por su *id* y su *type* del elemento de contexto) en el OBC. La creación de una Entidad se implementa mediante una petición al OCB, encapsulada en una petición HTTP de tipo `POST` con destino al path `/v2/entities`. El contenido de dicha petición debe incluir un ContextElement.

Una vez que se ha creado la entidad en el OCB, el productor puede actualizar el valor de dicha entidad mediante una petición HTTP de tipo `PATCH` con destino al path `/v2/entities/{id}/attrs` con los valores de los atributos que cambian.

Por otro lado, los consumidores se suscriben a las entidades mediante una petición HTTP de tipo `POST` con destino al path `/v2/subscriptions`. En el cuerpo del mensaje de esta petición, los consumidores deben incluir el `id` de la entidad a la que se suscriben, los atributos que desea obtener y la dirección en donde se deben notificar las medidas. Una vez que un consumidor se ha suscrito a una entidad, cada vez que esta actualicé su valor el OCB notificará al consumidor con el valor de la entidad actualizada.

Capítulo 3

Diseño y desarrollo del Marketplace

En este capitulo se describe el diseño y el desarrollo de la plataforma para la provisión de un entorno seguro para el intercambio de información proveniente de una infraestructura IoT, para lo que se detallarán las funcionalidades de los distintos elementos que componen dicha plataforma, así como los pasos que se han llevado al cabo para completar su implementación.

3.1. Principios de diseño y Arquitectura del sistema

El proyecto consiste en la integración de una plataforma de Blockchain en un ecosistema IoT FIWARE para la compra-venta de medidas. De esta manera, los productores de información pueden hacer públicas las medidas o eventos a través de la Blockchain y los clientes acceder a dicha Blockchain para comprar información. El objetivo de usar la tecnología Blockchain en este tipo de redes consiste en garantizar los siguientes elementos:

- Garantizar que la proveniencia del dato es confiable.
- Garantizar que el cliente ha pagado por un determinado dato.
- Garantizar que el cliente ha recibido dicho dato.

Aplicando los puntos anteriores se consigue un sistema totalmente transparente tanto para el cliente como para el proveedor de los datos. De esta forma, es imposible que un cliente diga que no ha recibido un dato o que no lo ha comprado, puesto que todo lo que ocurre queda constancia en la Blockchain.

En la Figura 3.1 se muestra el esquema de alto nivel de la plataforma. El componente principal para el proveedor de la información (entidad que controla la infraestructura IoT) es el elemento denominado *Market*. Se trata del componente que, de hecho, integra el ecosistema IoT con la blockchain.

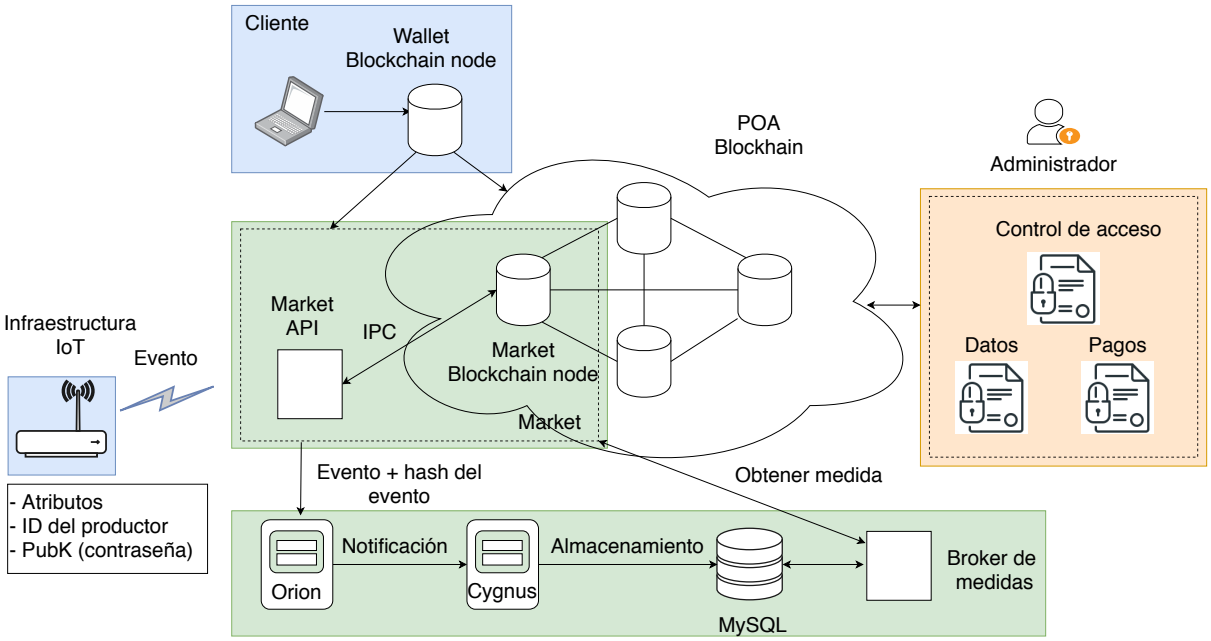

Figura 3.1: Arquitectura funcional de la plataforma

La plataforma está compuesta por diversos elementos que se pueden catalogar según el rol del usuario que los controla. Así, el código de colores de la Figura 3.1 representa lo siguiente:

- Los elementos de color verde están controlados exclusivamente por el administrador de la plataforma (por ejemplo el componente *Market*).
- Los elementos que no tienen color (por ejemplo la Blockchain) son aquellos que pueden estar controlados por la plataforma o no. Por ejemplo, en este caso, la plataforma puede utilizar una Blockchain pública o desplegar una serie de nodos validadores y formar su propia Blockchain privada.
- Los azules son elementos controlados por entidades que no pertenecen a la plataforma (por ejemplo la *infraestructura IoT* o los clientes)
- Los elementos naranja son aquellos que son públicos, cualquier persona/entidad puede acceder a ellos, pero están controlados por el administrador de la plataforma (Por ejemplo los contratos inteligentes).

3.1.1. Descripción funcional de los componentes

A continuación se describen los diferentes componente que se muestran en la Figura 3.1.

Market

Las funciones básicas de este componente son:

1. Escuchar las medidas provenientes de la infraestructura IoT.
2. Introducir nuevas medidas en la Blockchain.
3. Actualizar el ContextElement con la información adicional (Hash del dato) que permitirá localizar la medida tanto en la Blockchain como en la base de datos.

4. Procesar las peticiones provenientes de los consumidores (clientes).

Para realizar las funciones anteriores se implementan dos interfaces: *Market API* y *Market Blockchain node*.

La interfaz *Market API* actúa como elemento de conexión entre el mundo exterior (ecosistema IoT y clientes) y la Blockchain. La función de esta interfaz consiste en escuchar y procesar las peticiones HTTP que recibe la plataforma. Dependiendo del origen de estas peticiones, la plataforma realiza una tarea u otra:

- **Infraestructura IoT**: La interfaz escucha las medidas, en formato NGSIv2, provenientes de la infraestructura IoT e introduce la información necesaria para localizar estas medidas. Para ello, almacena la URL en la que se encuentra la medida.
- **Clientes**: Los clientes usan esta interfaz para indicar a la plataforma, mediante peticiones HTTP, que han comprado una medida. La plataforma procesa dichas peticiones y devuelve el valor de las medidas a los clientes a través de la Blockchain.
- **Administrador**: El administrador de la plataforma utiliza esta interfaz para gestionar los diferentes sensores que están autorizados en la plataforma. El administrador puede añadir o eliminar sensores de la plataforma.

Además de escuchar peticiones HTTP de los elementos anteriores, esta interfaz envía las medidas procesadas junto con su hash al OCB (*Orion*) para que sean almacenadas en una base de datos. El hash, al ser único por cada medida, se utiliza como atributo para diferenciar las medidas tanto en la Blockchain como en la base de datos.

Por otro lado, el *Market Blockchain node* se trata del nodo de la Blockchain que utiliza el administrador de la plataforma para conectarse con la Blockchain. Este componente se utiliza de manera exclusiva para interactuar con la Blockchain y con los contratos inteligentes que se ejecutan sobre esta.

La comunicación entre ambas interfaces se realiza mediante un enlace IPC (Inter-process communication). Este tipo de enlaces permite el intercambio de mensajes entre dos procesos de un sistema operativo.

Contratos inteligentes

Dentro de la Blockchain existen tres contratos inteligentes: *Control de acceso*, *Datos* y *Pagos*. Todos ellos controlados por un superusuario. Este, corresponde al usuario administrador encargado de gestionar la plataforma de compra-venta de medidas y su uso consiste fundamentalmente en el control de acceso de los sensores en la Blockchain. Por lo tanto, solamente los sensores que han sido previamente autorizados por el administrador pueden introducir contenido en la Blockchain.

El contrato de *Control de acceso* se encarga de realizar las siguientes funciones:

1. Controlar y gestionar los productores de información que tienen acceso a la blockchain.
2. Gestionar las claves públicas de los clientes que participan en la Blockchain. Esta función es necesaria para transmitir las medidas compradas por los clientes de manera segura.

Por otro lado, el contrato inteligente *Datos* se encarga de almacenar la información necesaria, para que el administrador de la plataforma sea capaz de localizar el dato, en la Blockchain. Para hacer esto, se almacena en la Blockchain el hash de la medida y la URL en la que se localiza. El hash de la medida se utiliza como elemento diferenciador para localizar las medidas en la blockchain. Además de almacenar el hash y la URL, también se guarda una breve descripción de la medida para que los clientes sepan el producto que están comprando.

El contrato inteligente *Pagos* realiza dos funciones:

1. Fijar el precio de las medidas que se introducen en la Blockchain. Esta tarea solamente la puede realizar el administrador de la plataforma.
2. Controlar y gestionar las compras que realizan los clientes. Se tiene que garantizar que un cliente ha comprado una medida y que un cliente no compre la misma medida dos veces.

Orion

Orion es un OCB privado controlado exclusivamente por el administrador de la plataforma.

Este componente, recibe las medidas de los sensores junto con su hash provenientes del *Market*. Las medidas provienen de este elemento y no de la infraestructura IoT debido a que el administrador de la plataforma debe asegurarse que la información necesaria para localizar la medida ha sido previamente introducida en la Blockchain. Además, siguiendo este orden, el administrador puede introducir información adicional a la medida para que su localización sea más sencilla, como por ejemplo su hash.

Cygnus y MySQL

Cygnus es un componente, que forma parte de FIWARE, controlado por el administrador de la plataforma. La tarea de este componente consiste en suscribirse en el OCB a las medidas que producen los sensores autorizados en la plataforma y almacenarlas en una base de datos para aportar persistencia a la información.

En este caso, la base de datos elegida es MySQL.

Broker de medidas

Este componente, controlado por el administrador de la plataforma, actúa como intermediario entre la base de datos y el componente *Market*. En este caso la única entidad que tiene acceso a este componente, y por ende a la base de datos, es el administrador de la plataforma.

Este elemento lo único que hace es escuchar peticiones HTTP de tipo `GET` a los paths de las medidas y como resultado retorna el valor de la medida. Para que esto sea posible, las URLs de las medidas tienen que ser únicas. Para garantizar esto, la URL de la medida incluye el hash de esta. Las URLs se compondrán de la siguiente manera:

`https://direcciónIP:puerto/fiware-service/fiware-servicepath/hash`

Cabe destacar, que antes de mandar la medida, la plataforma necesita comprobar que el usuario que esta accediendo está autorizado. El proceso de autenticación se realizará

mediante el mecanismo `basic authentication` de HTTP. En este caso, como se ha comentado antes, solamente el administrador de la plataforma tendrá acceso a esas medidas.

Wallet Blockchain node

Para que un cliente pueda comprar medidas en el *Market* necesitan hacer uso del elemento *Wallet Blockchain node*. Este componente, mediante un servidor web, permite que los clientes observen las diferentes medidas disponibles en la plataforma, así como las medidas que han comprado.

Adicionalmente, el componente actúa como un nodo de la Blockchain para interactuar con esta a la hora de comprar medidas.

3.1.2. Procedimientos del Marketplace

Como se ha descrito, la plataforma desarrollada en este TFM tiene como principal objetivo el dar soporte al intercambio seguro de datos en escenario similar al de un mercado en el que productores y consumidores tengan seguridad en el proceso de compra.

Para ello, la plataforma permite una serie de procedimientos que se describen en los siguientes apartados. En ellos se utiliza el siguiente código de colores para describir las distintas operaciones que tienen lugar en cada uno de ellos.

- Las operaciones de color azul son aquellas que introducen nuevo contenido a la Blockchain en forma de transacción,
- Las funciones de color verde son aquellas que recogen información de la Blockchain pero no introducen ningún contenido a esta. Por lo tanto, no generan nuevas transacciones.
- Las operaciones que no tienen asociadas ningún color son peticiones HTTP que no tienen ninguna interacción con la Blockchain.

Esta gama de colores se va a utilizar en los esquemas posteriores, salvo que se indique lo contrario.

Registro del cliente en el Marketplace

En primer lugar, para que un cliente pueda acceder a la plataforma para comprar medidas, es necesario que el cliente registre su clave pública en la Blockchain. El cliente necesita hacer esto para que la plataforma sea capaz de enviarle las medidas que compre cifradas a través de la Blockchain.

Para registrar la clave pública en la Blockchain, el cliente realizará una transacción, desde su nodo de la Blockchain, a la función del contrato de control de acceso encargada de almacenar la clave asociada a un usuario en la Blockchain.

Una vez que el cliente ha almacenado la clave pública en la Blockchain, ésta se almacenará de manera permanente salvo que el cliente la cambie. Esto implica que para futuras compras, la plataforma usará esta clave pública para enviar las medidas cifradas al cliente.

Registro de los productores de medidas en el Marketplace

El administrador de la plataforma es el único usuario que tiene permisos para registrar productores en la Blockchain. Por lo tanto, si un proveedor de datos desea introducir un

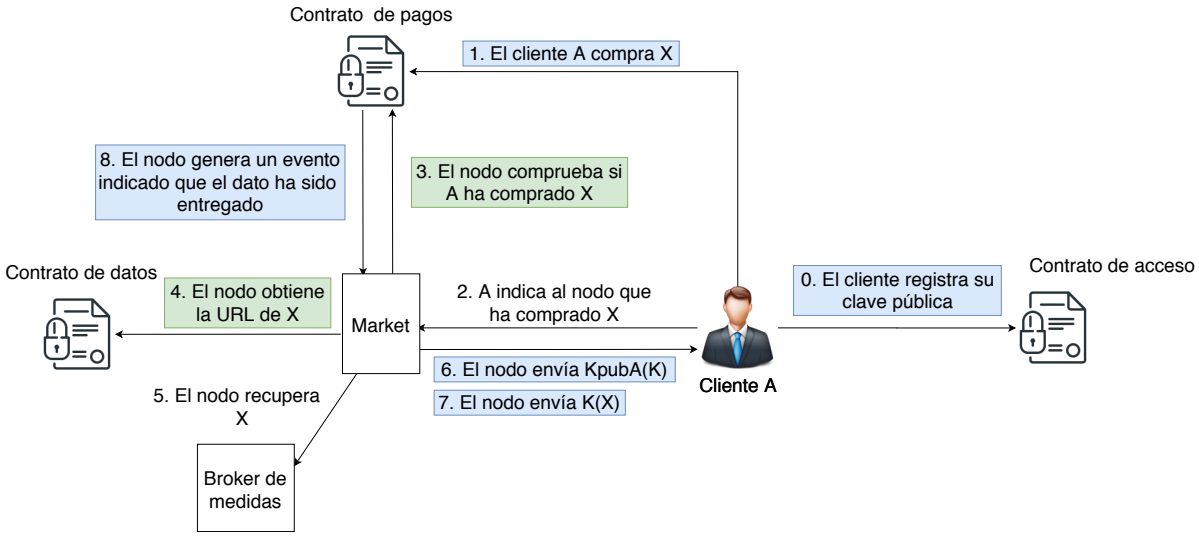

Figura 3.2: Procedimiento de compra de datos

sensor en la plataforma, es necesario que llegue a un acuerdo con el administrador del Marketplace para que éste permita la inserción de medidas de dicho sensor.

Para registrar los sensores que tienen permisos en la plataforma, el administrador almacenará la dirección de Ethereum de los sensores autorizados en una variable dentro de la EVM de Ethereum.

Para ello, se habilitará, dentro de la plataforma, un punto de acceso para que el administrador pueda añadir o eliminar sensores de la plataforma.

Inclusión de un dato en la Blockchain

En la Blockchain se almacenará la información necesaria para localizar la medida. De tal manera, que si un cliente desea comprar una medida, el administrador será capaz de obtener la localización de dicha medida únicamente a través de la Blockchain.

En la Blockchain, se van a almacenar dos atributos por cada medida:

1. **URL**: La URL en la que el administrador puede recuperar el valor de la medida.
2. **Descripción**: Este campo contiene una breve descripción de la medida. La inclusión de este elemento en la Blockchain se realiza para indicar al cliente el propósito de la medida. De tal forma que los clientes, viendo esta descripción, deciden si comprar o no la medida. Un ejemplo de descripción podría ser: *Medida de la temperatura del salón en la casa 1.*

Para localizar las medidas dentro de la Blockchain se utiliza su hash.

Compra de un dato en el Marketplace

En la Figura 3.2 se puede observar los diferentes procesos involucrados durante la compra de una medida.

1. Los clientes realizan una transacción a un método del contrato que gestiona los pagos indicando el elemento que desean comprar.
2. El cliente indica, mediante una petición HTTP, al nodo *Market* que ha comprado

un dato. Dentro del cuerpo de esta petición, el cliente indica el hash de la medida que ha comprado, su dirección de la Blockchain y la firma de estos dos atributos. Este último atributo se incluye para que la plataforma sea capaz de comprobar que el mensaje lo ha enviado la dirección que ha realizado la compra.

3. El nodo *Market* comprueba si el usuario, que se indica en la petición HTTP, ha comprado la medida o no.
4. El nodo *Market* obtiene la URL de la medida a partir de la Blockchain.
5. El nodo *Market* recupera el valor de la medida que se encuentra en la dirección obtenida en el punto anterior.
6. El nodo *Market* envía la clave de cifrado simétrico, que se utiliza para cifrar la medida, cifrada con la clave pública del cliente a través de la Blockchain.
7. El nodo *Market* envía la medida, cifrada con la clave de cifrado simétrico, al usuario que compró dicha medida a través de la Blockchain.
8. El nodo *Market* genera un evento en la Blockchain indicando que la entrega de la medida se ha realizado con éxito.

3.2. Desarrollo de los smart contracts

Como se ha comentado previamente, en la Blockchain se van a desplegar tres contratos inteligentes. Uno para gestionar el acceso de los clientes y de los productores (control de acceso), otro para gestionar la información que se almacena en la Blockchain (control de datos) y un último contrato para gestionar la compra-venta de las medidas (control de pagos).

A lo largo de este apartado se van a explicar de manera más detallada las funciones de cada uno de los contratos y como se han programado.

3.2.1. Contrato para el control de acceso

Este contrato debe proveer dos servicios:

1. Controlar que solamente los sensores autorizados por el proveedor de la información tengan acceso al contrato de *Datos*.
2. Almacenar las claves públicas de los clientes.

Para resolver la primera de estas funcionalidades, el contrato dispone de dos funciones para añadir o eliminar cuentas de un registro de usuarios autorizados que se mantienen en la EVM de la Blockchain. La Figura 3.3 muestra en detalle el código fuente del contrato. En ambas funciones se opera sobre la variable `whitePages`. Un mapping que asocia identificadores con cuentas de Ethereum.

```
// Add a producer account to the list of producers
function addAccountToRegister(bytes32 id, address account) public
{
 assert (msg.sender == admin);
 whitePages[id] = account;

 emit newAddrRegistered(id);
}

// Remove a producer from the list of producers
function removeAccountFromRegister(bytes32 id) public
{
 assert(msg.sender == admin);
 delete whitePages[id];

 emit newAddrRemove(id);
}
```

Figura 3.3: Control de los productores en el contrato de control de acceso

Para garantizar que solamente el administrador es el único que puede modificar dicho registro, se utiliza la función `assert`. Esta función continua con la ejecución del código si se cumple la condición que aparece entre paréntesis, pero en caso contrario, aborta la ejecución del contrato. En este caso, se tiene que cumplir que la cuenta de Ethereum del administrador sea la que acceda a estas funciones del contrato. Para obtener la dirección que está accediendo al contrato se usa la variable `msg.sender`. Cabe destacar que previamente se ha asignado el valor de la variable `admin` a la cuenta de Ethereum del administrador.

El contrato, además, genera un evento en la Blockchain cada vez que se añade o elimina un sensor haciendo uso de la función `emit`. Estos eventos, resultan de utilidad, sobre todo, en escenarios en los que hubiera varios administradores. Si alguno añadiera un sensor que no debería, se podría comprobar en la información del evento quién lo añadió.

El contrato también incluye una función para obtener la dirección asociada a un productor. La Figura 3.4 muestra el código fuente de dicha función.

```
// Gets the address of the producer
function getAddress(bytes32 id) public view returns(address)
{
    return whitePages[id];
}
```

Figura 3.4: Obtención de la dirección asociada a un sensor

Para resolver la funcionalidad de almacenar las claves públicas de los clientes en la Blockchain, así como para acceder posteriormente a ellas, se utilizan, respectivamente, las funciones `addPubKey` y `getPubKey` que aparecen en la Figura 3.5. La primera recibe como parámetros de entrada la clave pública del cliente y la almacena en formato string en un mapping que asocia direcciones con su correspondiente clave. Se emplea nuevamente la variable `msg.sender`, por lo que el cliente solo puede asociar su clave pública. Un cliente

malicioso no podría asociar claves públicas a direcciones que no son la suya ya que la Blockchain garantiza quién interactúa con el contrato evitando posibles suplantaciones de identidad.

La segunda función (`getPubKey`), recibe como parámetro de entrada una dirección Ethereum y devuelve la clave pública asociada a esa dirección. En este caso, cualquier usuario puede obtener la clave pública de otros usuarios que se encuentren registrados en el mapping `clientsPubKeys`.

```
// Stores the public key of the client
function addPubKey(string memory pubKey) public
{
    clientsPubKeys[msg.sender] = pubKey;
}

// Retrieves the value of the public Key
function getPubKey(address addr) public view returns(string memory)
{
    return clientsPubKeys[addr];
}
```

Figura 3.5: Lectura y escritura de las claves públicas en el contrato

Más adelante se describirá la forma en que cada una de estas funcionalidades se emplean en los diferentes procedimientos que se llevan al cabo en la plataforma.

El código completo de este contrato se puede observar en [22].

3.2.2. Contrato para el almacenamiento de medidas

Este contrato está compuesto principalmente por dos funciones. Una función para que los sensores almacenen la información asociada a la medidas y otra para poder recuperar dicha información.

En la Figura 3.6 se puede observar la función encargada de almacenar la URL y la descripción del dato en la Blockchain. Esta recibe como parámetros de entrada el hash de la medida, la URL para acceder a dicha medida, el identificador del sensor que produjo la medida y la descripción de esta.

Lo primero que hace esta función es comprobar que la dirección de Ethereum que quiere almacenar la información en la Blockchain pertenece a uno de los dispositivos IoT autorizados. Esto se realiza llamando a la función `getAddress` del contrato de control de acceso [1]. En este caso no se ha utilizado `assert` como en casos anteriores sino que se ha utilizado `require`. A efectos prácticos, ambas se comportan de la misma manera. La única diferencia entre una función y la otra es que en el caso del `require` si no se cumple la condición se envía un mensaje explicando el motivo y se le devuelve todo el Ether al nodo que genero la transacción. En este caso, dado que se trata de una red de PoA en la que el precio del gas es 0 se puede utilizar las dos funciones de manera indiferente.

[1] En Ethereum es posible llamar a un contrato desde otro contrato

```
// Stores information in the Blockchain
function storeInfo(bytes32 hash, string memory uri, bytes32 id, string memory
    description) public
{
    require(msg.sender == getAddress(id), "The ID that you are using is not registered
        ");
    dataStruct memory dataToStore;

    dataToStore.uri = uri;
    dataToStore.description = description;

    ledger[hash] = dataToStore;

    // Emit an event once the data has been stored in the Blockchain
    emit evtStoreInfo(hash, uri);
}
```

Figura 3.6: Función para almacenar la información de la medida en la Blockchain

Una vez que se ha comprobado que el sensor que quiere introducir la medida está autorizado, se almacena la información en un mapping que asocia el hash de la medida con el objeto que contiene la información relativa a dicha medida. En la Figura 3.7 se puede observar la estructura que almacena la información relativa a una medida.

```
struct dataStruct
{
    string uri;
    string description;
}
```

Figura 3.7: Estructura del objeto utilizada para almacenar medidas en la Blockchain

Por otro lado, para obtener el dato se utiliza la función se muestra en la Figura 3.8. La función recibe como único parámetro de entrada el hash del dato y retorna dos valores. La URL y la descripción del dato asociado a ese hash.

```
// Retrieves information from the Blockchain
function retrieveInfo(bytes32 hash) view public returns (string memory, string memory)
{
    return (ledger[hash].uri, ledger[hash].description);
}
```

Figura 3.8: Función para recuperar la información de la medida en la Blockchain

En [23] se puede consultar el código completo del contrato.

3.2.3. Contrato para la gestión de los pagos

Este contrato incluye, principalmente, dos funciones. La primera comprueba que se ha realizado el pago por una medida. La segunda se encarga de fijar el precio que tienen que pagar los clientes por cada medida.

La Figura 3.9 presenta el detalle de la función `payData`. Esta función establece los controles para garantizar que un usuario ha pagado por una medida.

Se puede observar que la función recibe como parámetros de entrada dos elementos: el hash del dato que desea adquirir el cliente y el número de tokens que paga. A continuación, antes de procesar el pago, se comprueban una serie de condiciones:

1. Se confirma que el número de tokens que paga el cliente es igual o mayor que el precio del producto.
2. Se confirma que el usuario no está pagando algo que ya ha pagado y que está en proceso de pago. Esto se comprueba mediante la variable `hasPaid`. Esta es un mapping que asocia usuarios a otro mapping de hashes a booleanos (`mapping(address =>mapping(bytes32 =>hash))`). De tal manera, que a partir de la dirección del usuario y el hash del dato se puede comprobar el valor de esta variable.

   El valor de esta variable es siempre false y se activa a true cuando el usuario paga por la medida. En el momento en el que el administrador se la envía al cliente, esta se vuelve a poner a false. De esta forma, se puede comprobar si un usuario ha pagado por un dato, pero aún está a la espera de recibirlo. Por lo tanto, gracias al uso de esta variable se evita que un usuario pague dos veces por el mismo dato.
3. Se confirma que el dato existe en la Blockchain. Para ello, se utiliza la función `retrieveInfo` del contrato que gestiona el almacenamiento de la información.
4. Se confirma que el precio del dato es distinto de 0. Esto se hace para evitar que el usuario compre datos que aún no tienen un precio asignado.

Una vez que se han comprobado que se cumplen las condiciones anteriores, se actualiza la variable `hasPaid` a true para indicar que el usuario está a la espera de de que el pago a ese hash sea procesado por el nodo *Market* y se emite un evento que notifica esta compra. En este último, se almacena la dirección de la cuenta que almacena el dato, el hash del dato y el número de tokens que ha pagado el usuario.

Por otro lado, para garantizar que el usuario recibe el dato se utiliza la función `sendToClient` que aparece en la Figura 3.10. En esta, se puede observar como esta función recibe como parámetros de entrada la dirección del cliente, el hash de la medida, el hash de la transacción que contiene el secreto que se usa para cifrar la información y el hash de la transacción que contiene la medida cifrada.

Este contrato se utiliza también para fijar el precio de las medidas. Esto se realiza mediante la función `setPriceData`. En la Figura 3.11 se puede observar su contenido. La función recibe como parámetros de entrada el hash del dato y un entero cuyo valor es el precio de la medida.

A continuación, se comprueba que el usuario que accede a esta función es el administrador ya que este es el único que tiene en su poder el fijar el precio de la medida. Una vez que se cumple esta condición, se pone a false el valor de la variable `hasPaid` para indicar que el dato ya ha sido enviado y se emite un evento que notifica a la Blockchain del pago del dato. Este evento almacena la cuenta del cliente, el hash del dato, el hash de la transacción que contiene el secreto y el hash de la transacción que contiene el dato cifrado.

```
// Function to pay the data of the hash
function payData(bytes32 hash, uint256 tokens) public
{
    // Confirm that the amount of money sent by the client
    // is enough to buy the data
    assert(tokens >= catalogue[hash]);

    // Check that the user is not trying to buy something that
    // is already bought
    assert(hasPaid[msg.sender][hash] != true);

    // Check if the hash exists
    (string memory pubDate, string memory uri) = dataContract.retrieveInfo(hash);
    assert (bytes(uri).length != 0);
    assert(bytes(pubDate).length != 0);

    // Check that the element is available to buy (price != 0)
    assert(getPriceData(hash) != 0);

    // Update the balance of the owner
    balance += catalogue[hash];

    // Update the hasPaid function to true
    hasPaid[msg.sender][hash] = true;

    // Emitting the event
    emit purchaseNotify(msg.sender, hash, catalogue[hash]);
}
```

Figura 3.9: Función `payData`

```
// Send the response to the clientAccount
function sendToClient(address clientAccount, bytes32 hash, bytes32 txHashExchange,
    bytes32 txHashData) public
{
    // The only user that can send information to the client is the admin
    assert(msg.sender == admin);

    // Set to false the hash
    hasPaid[clientAccount][hash] = false;

    // Emitting the event to assure that the response has been sent
    emit responseNotify(clientAccount, hash, txHashExchange, txHashData);
}
```

Figura 3.10: Función `sendToClient`

Una vez que se comprueban estas dos condiciones, se fija el valor del precio al valor introducido por el administrador. Para hacer esto se hace uso de la variable `catalogue`. Esta, es un mapping que asocia hashes a precios.

Finalmente, como resultado se retorna un booleano indicando que el precio se ha fijado con éxito.

```
// Function used by the admin user to set the prices of the data
function setPriceData(bytes32 hash, uint256 price) public returns(bool)
{
    assert (msg.sender == admin);

    // Check if the hash exists
    (string memory pubDate, string memory uri) = dataContract.retrieveInfo(hash);
    assert (bytes(uri).length != 0);
    assert(bytes(pubDate).length != 0);
    catalogue[hash] = price;

    return true;
}
```

Figura 3.11: Función `setPriceData`

En [24] se puede consultar el código completo de este contrato.

3.3. Desarrollo del Broker de medidas

En este apartado se va a describir el desarrollo del Broker de medidas que se encargará de gestionar las peticiones provenientes del *Market* y responderle con la medida correspondiente.

Como se ha descrito en el apartado 3.1.1, las URLs de cada una de las medidas se generan en base al hash de la medida de tal forma que para que el nodo *Market* pueda obtener una determinada medida, lo único que tiene que hacer es hacer un `GET` a la URL de dicha medida. Esta URL es la que se encuentra almacenada en la Blockchain.

Este elemento solo puede ser accedido por el *Market*. Por lo tanto, es necesario utilizar un mecanismo de autenticación que sea capaz de diferenciar las peticiones provenientes de este componente del resto. Se utiliza, para ello, un mecanismo de API key que se valida utilizando una autenticación básica (cabecera `Authorization`) de HTTP.

El Broker de medidas solo es accesible mediante HTTPS (HyperText Transfer Protocol Secure). De esta forma, la información que se envía tanto en la petición como en la respuesta está cifrada. Por lo tanto, aunque haya un usuario malicioso escuchando la conversación entre el nodo `Market` y el Broker de medidas no puede obtener dicha información. Para ello, en primer lugar se ha creado una CA (Certification Authority) para firmar el certificado del servidor mediante la siguiente instrucción:

```
openssl req -x509 -nodes -days 1024 -newkey rsa:4096 -out ca.root.pem -
    keyout ca.key.pem
```

A continuación, se ha creado el certificado del servidor y se ha firmado con la clave privada de la CA mediante las siguientes instrucciones:

```
$ openssl req -newkey rsa:2048 -nodes -keyout private-key.pem -days 1024
   -out public-cert.csr
$ CAFOLDER=/home/ivan/Desktop/demoPOA2/ca
$ openssl x509 -req -in public-cert.csr -CA $CAFOLDER/ca.root.pem -CAkey
   $CAFOLDER/ca.key.pem -CAcreateserial -out public-cert.pem -days 1024 -
   sha256
```

Para la implementación de este componente se eligió hacerlo utilizando el paquete `express` del lenguaje nodejs.

El Pseudocódigo 1 describe el funcionamiento del servidor. Básicamente este componente devuelve el valor de las medidas al nodo *Market*.

Además del servicio de acceso a las medidas, este servidor permite listar todas las medidas en la base de datos y borrar una medida de ésta. En todos estos casos, estos servicios solo están disponibles para el usuario administrador.

El código completo de este componente se puede consultar en [25].

Pseudocódigo 1: Funcionamiento del Broker de medidas

```
Iniciar el servidor y escuchar peticiones;
Comprobar que la petición proviene de una función autorizada;
if Tiene acceso then
    Obtener la medida de la base de datos;
    Responder con la medida al nodo que generó la petición;
else
    Responder al nodo que generó la petición con un mensaje de error;
```

3.4. Desarrollo del *Market API*

Para la implementación de este componente, el lenguaje de programación utilizado fue Go. Se ha elegido por su rapidez y por su compatibilidad con la red de Ethereum, dado que esta se encuentra programada en este lenguaje.

Este componente exporta tres servicios básicos:

1. Escuchar los eventos que generan los sensores, reenviarlos al OCB e introducir la URL de estas medidas en la Blockchain. En este caso, el servidor va a escuchar las peticiones de los sensores en el path `/notify`.
2. Escuchar las peticiones de compra de los clientes para proporcionarles la medida. En este caso, el servidor escucha las peticiones de los clientes en el path `/buydata`.
3. Hacer de punto de acceso para que el administrador pueda gestionar los productores que tienen acceso a la Blockchain. Este punto de acceso se encuentra en el path `/adminap`

En Go para poder acceder a las funciones de los smart contracts es necesario compilarlos y crear un código Go asociado a dichos contratos. Para ello se utiliza la herramienta `abigen`. Por ejemplo, para generar el archivo Go del contrato de control de acceso se utiliza la siguiente orden:

```
$ abigen -sol accessContract/accessControl.sol -pkg accessControlContract
    -out accessContract/accessControlContract.go
```

Mediante el comando anterior se ha convertido el fichero fuente, que emplea el lenguaje Solidity, a una interfaz de Go. Este proceso se realiza con los tres contratos.

Una vez que se han compilado los contratos, lo primero que hace el servidor antes de arrancar, es conectarse al nodo de la Blockchain e inicializar los tres contratos. La conexión con el nodo de la Blockchain se hace mediante IPC.

El nodo de la Blockchain del administrador solo tiene activado como punto de acceso esta conexión IPC. La idea detrás de esto, es que este nodo solamente pueda ser accedido por procesos que se ejecutan dentro del mismo dispositivo físico. Por lo tanto, si un usuario malintencionado desea registrarse en ese nodo tiene que tener acceso al dispositivo físico.

Previamente, se ha mencionado la existencia de tres paths: `/notify`, `/buydata` y `/adminap`. A continuación se muestra que ocurre en cada uno de estos.

3.4.1. `/notify`

Como se ha comentado previamente, en este path se procesan los eventos generados por los sensores. Para ello, en primer lugar, se comprueba que los sensores que quieren introducir contenido en la Blockchain están autorizados por el administrador. A continuación se reenvía la medida junto con su hash a Orion. Finalmente, se almacena en la Blockchain la descripción y la URL de la medida en la Blockchain.

En el Pseudocódigo 2 se describen los procesos que se han implementado para habilitar el servicio de recepción de medidas desde los sensores.

Pseudocódigo 2: `/notify`

```
Obtener la cabecera del mensaje;
Obtener el cuerpo del mensaje;
Comprobar que el sensor tiene acceso a la Blockchain;
if Tiene acceso then
    Preparar y envíar la medida a Orion;
    Preparar e introducir los datos en la blockcahin;
    Responder al sensor con un 200 OK;
else
    Responder al sensor con un mensaje de error;
```

Una vez que se ha recuperado el contenido de la cabecera y del cuerpo del mensaje, el siguiente paso consiste en comprobar que se trata de un sensor autorizado por el administrador. Para ello, el cuerpo del mensaje que se recibe del sensor está compuesto por tres campos: la medida, el identificador del sensor y una contraseña cifrada con la clave pública del administrador. Esta contraseña es la que se utiliza para obtener la clave privada de Ethereum del sensor, la cual se encuentra almacenada cifrada en el `Market`.

Cabe destacar que esta clave cifrada no se encuentra almacenada en la Blockchain sino que se encuentra almacenada en un fichero del sistema.

En la Figura 3.12, se puede observar los pasos que sigue el Market para comprobar si la petición la envía un sensor autorizado o no.

Al recibir la notificación del sensor, obtiene la dirección asociada con la id de este utilizando el contrato de control de acceso. Al mismo tiempo, el servidor descifra, con su clave privada, la contraseña que utiliza el sensor para autenticarse frente a Ethereum.

Una vez que el servidor ha recuperado la contraseña, obtiene la clave privada del sensor. Si se puede recuperar la clave privada a partir del password implica que el sensor es quien dice ser. En caso de que no sea posible recuperar dicha clave, significa que el sensor no tiene acceso a la Blockchain. De esta forma es posible comprobar si el sensor tiene permisos de acceso a la Blockchain o no.

Tras comprobar que el sensor está autorizado para acceder a la Blockchain, el siguiente paso consiste en reenviar la medida en formato NGSIv2 al OCB, añadiéndole la metainofrmación necesaria para su posterior recuperación en caso de que algún consumidor la demande.

A la hora de conformar la petición, se descartan el campo descripción y el campo que contiene la contraseña del sensor cifrada con la clave pública del administrador. Además de borrar estos campos, se añade un atributo de tipo metadata con el hash de la medida. Éste es el que se utiliza como índice para encontrar una determinada medida.

En la cabecera de la petición también se añaden dos campos: *Fiware-Service* y *Fiware-Servicepath*. Estos campos, hacen referencia a la base de datos y la tabla de MySQL donde se debe almacenar la medida.

La Figura 3.13 muestra un ejemplo del cuerpo de una medida que se envía al OCB. En esta se puede ver como dentro del campo `attributes` se encuentra el metadato y un campo `value` con la medida, el id del sensor y el instante temporal en el que se generó la medida.

En el Pseudocódigo 3 se puede observar las operaciones necesarias para enviar la medida al OCB. En primer lugar se realiza una petición POST a Orion. En el caso de que el status code de la respuesta sea un `2xx` implica que la petición se ha enviado con éxito. Por el contrario, en el caso de que devuelva un `422` implica que se está volviendo a crear una entidad que ya existía. Por lo tanto, como se comento en el apartado de NGSIv2 de los aspectos teóricos, para añadir una nueva medida a una entidad que ya existe es necesario realizar un PATCH al atributo que se desea modificar.

Por ejemplo, para crear una nueva entidad se realiza un POST a la URL:

```
http://orionhost:1026/v2/entities
```

Mientras que para añadir una nueva medida a una entidad existente, se realiza un PATCH a la siguiente URL:

```
http://orionhost:1026/v2/entities/id-del-topic/attrs
```

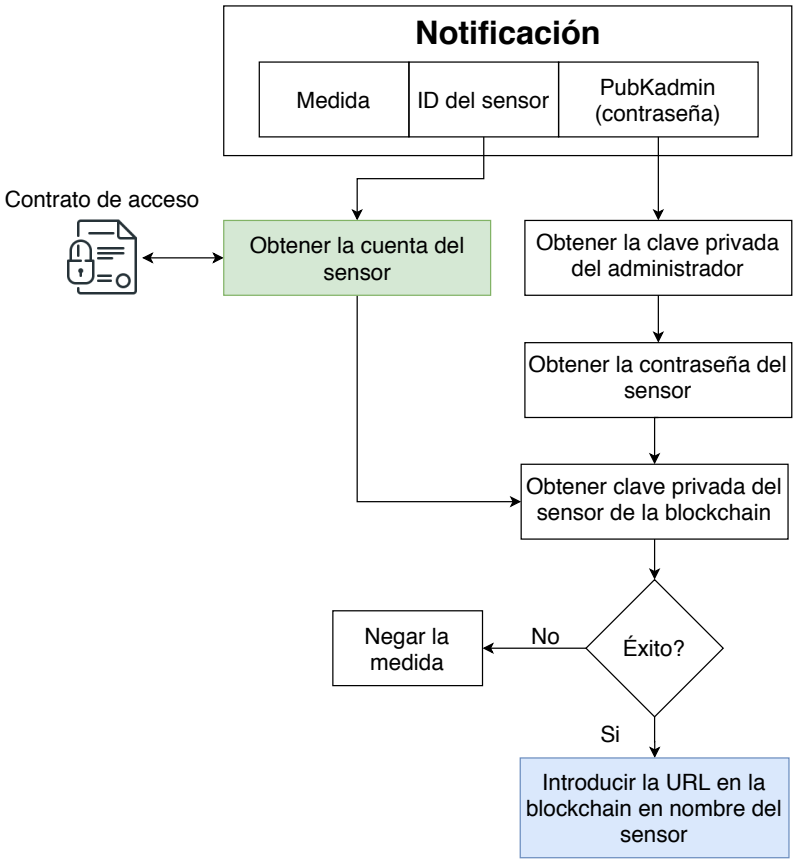

Figura 3.12: Control de acceso de los sensores

```
{
  "id": "Room1",
  "type": "Room",
  "attributes": {
    "metadata": {
      "hash": {
        "type": "String",
        "value": "8050fd47b9fd757360609d623a525987c3335d1c83f57a31f340d3e46455febb"
      }
    },
    "type": "Object",
    "value": {
      "temperature": {
        "value": 25,
        "type": "Number"
      },
      "sensorID": {
        "value": "sensor1",
        "type": "String"
      },
      "timestamp": {
        "value": 1588050810892,
        "type": "Number"
      }
    }
  }
}
```

Figura 3.13: Ejemplo del cuerpo de una medida que se envía a Orion

Pseudocódigo 3: `Función PostData`

Convertir el cuerpo del mensaje a JSON;
Hacer un POST a Orion con la petición y esperar el status code;
if *status code = 422* **then**
    Cambiar la URL de Orion parta modificar un atributo;
    Borrar los campos `id` y `type`;
    Hacer un PATCH a Orion con el nuevo cuerpo del mensaje;

Una vez que se ha enviado el dato con éxito al OCB, el siguiente paso consiste en introducir el dato en la Blockchain. Como se ha comentado en apartados anteriores, para reducir la cantidad de información que se almacena en la Blockchain, se introduce sólo el hash, la URL y la descripción de la medida. Esto se realiza mediante una transacción a la función `storeInfo` del contrato que se encarga de almacenar los datos. En este caso, la transacción la envía el sensor con su cuenta de Ethereum.

Después de introducir la información necesaria para recuperar la medida en la Blockchain, se establece el precio de dicha información. Para ello se accede a la función `setPriceData` del contrato que se encarga de gestionar las compras de los clientes. En este caso, el encargado de fijar el precio del dato es el administrador.

3.4.2. `/buydata`

En esta ruta, el servidor escucha las peticiones de compra que realizan los clientes. Es decir, cada vez que un cliente compre un dato debe de enviar una petición HTTPS a este path.

El cuerpo de dicha petición contiene un objeto JSON con los siguientes datos:

- El hash del dato que el cliente ha comprado.
- La dirección de la cuenta de Ethereum que ha comprado el dato. Cada cliente, lógicamente, envía su dirección.
- La firma de los dos campos anteriores. Este campo se utiliza para que la plataforma verifique que la petición HTTPS la ha realizado el cliente que ha comprado la medida en la Blockchain y así evitar suplantaciones de identidad en el proceso.

En la Figura 3.14 se muestra un ejemplo del cuerpo del mensaje que se envía en dicha petición HTTPS.

```
{
  "_hash": "0x06efec25918d4d73ca593b999f3e65af835e27ad9532e11e200b221fe667f64c",
  "_account": "0x5bab040bc593f57eda64ea431b14f182fe167f3f",
  "_signature": "0x39fe823be78a637a7e999273ca7f268b4944a84243eefdc68262b4b20
     e4ce5625d921e37ab6b95c4fed1362a3b47c3066bf6ecdbfeaf998a5db00421a1504d09"
}
```

Figura 3.14: Ejemplo del cuerpo de una petición a `/buydata`

Una vez que el `Market` ha recibido una petición del cliente, realiza los pasos que se indican

en la Figura 3.15. En ésta, X representa el hash del dato y A la dirección de Ethereum del cliente.

En primer lugar, el `Market`, nada más recibir la petición del cliente, comprueba que este ha realizado la compra en la Blockchain. Para ello, comprueba mediante la firma la identidad del cliente.

Una vez que ha verificado que el cliente tiene su clave pública de Ethereum publicada en la Blockchain, comprueba que el cliente ha realizado la compra del dato. Para ver esto, chequea si existe algún evento de tipo `purchaseNotify` en la Blockchain cuyo hash y cuenta son las que se indican en la petición HTTP.

Después de verificar que el cliente ha realizado la compra, el `Market` busca la URL del dato en la Blockchain. Tras recuperarla, realiza una petición GET a esta dirección autenticándose mediante su API key. Como respuesta a esa petición, el `Market` obtiene el valor de la medida que ha comprado el cliente.

El siguiente paso que realiza el `Market` consiste en cifrar la medida para que solo el cliente la pueda recuperar. Una vez que cifra la medida, ésta se envía en el cuerpo de una transacción de la Blockchain. Gracias a esto, puede garantizar que el cliente ha recibido la medida que ha comprado.

En la Figura 3.16 se puede observar el proceso que realiza el `Market` para cifrar y enviar la medida al cliente.

La clave de cifrado simétrico se cifra, en paralelo, tanto con la clave pública del cliente como con la del administrador. De esta forma se evita que el cliente pueda reclamar que la clave simétrica no es válida ya que el administrador de el `Market` puede, en todo momento, comprobar dicha validez utilizando su propia clave privada.

Los algoritmos de cifrado utilizados son AES-256-GCM y ECIES para el cifrado simétrico y asimétrico respectivamente.

Finalmente, una vez que se han realizado las dos transacciones en la Blockchain, el `Market` responde a la petición que envío el cliente con los hashes de las transacciones en las que se encuentra la clave simétrica cifrada y la medida cifrada.

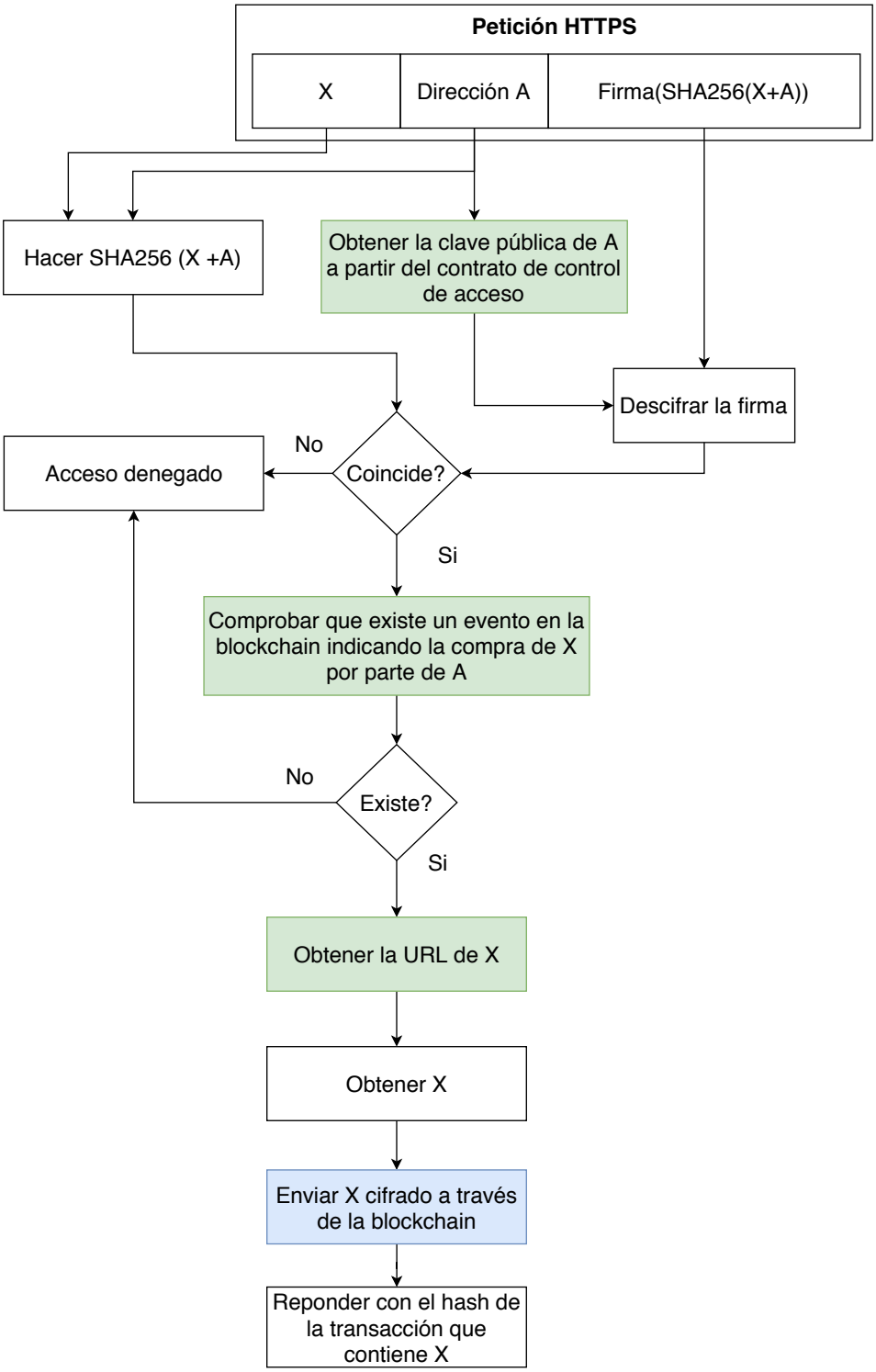

Figura 3.15: Esquema de la ruta `/buydata`

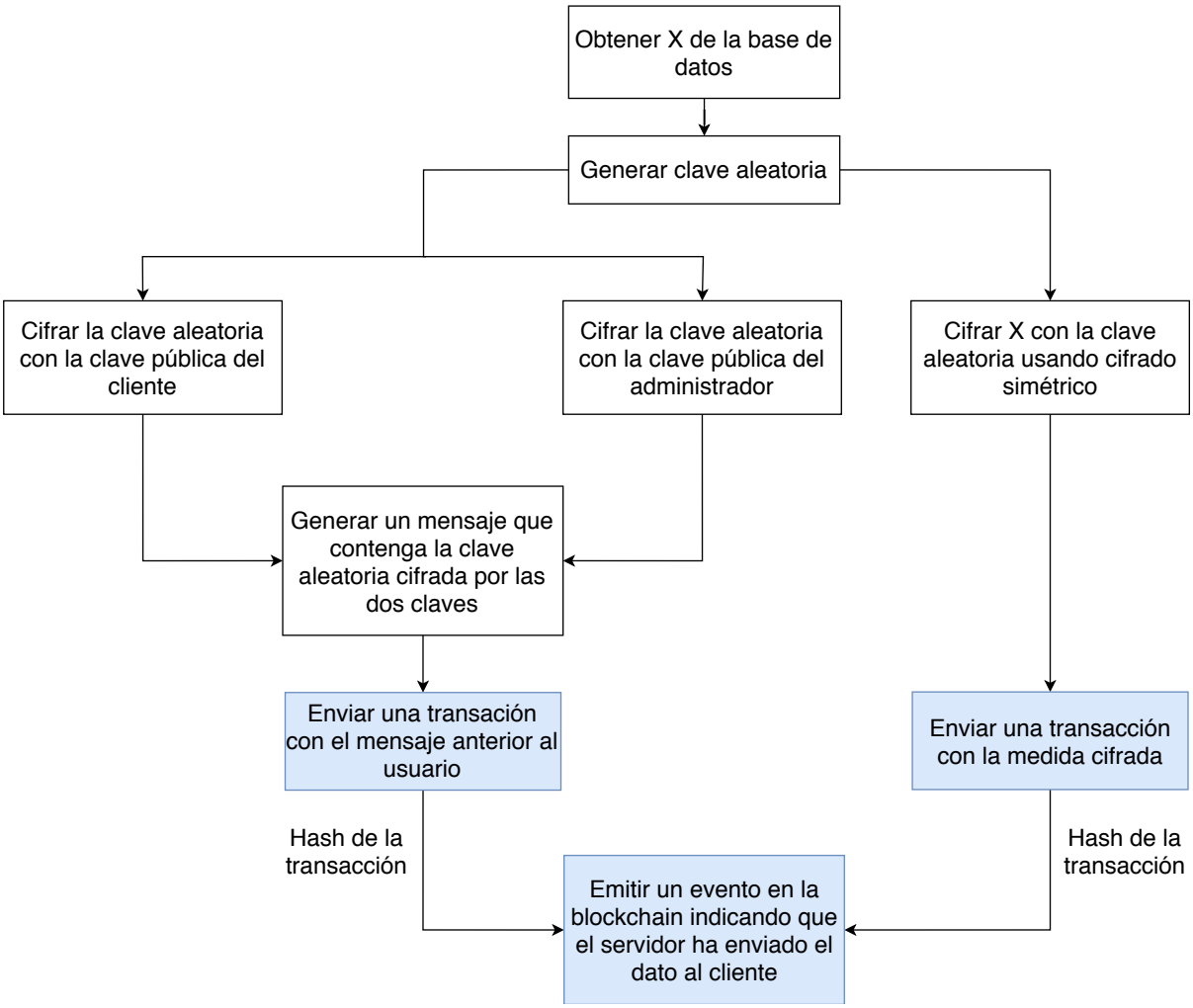

Figura 3.16: Proceso de cifrado de la medida y transferencia

3.4.3. `/adminap`

Además de las dos rutas mencionadas anteriormente, la plataforma también escucha peticiones en la ruta `/adminap`. En esta, se encuentra el punto de acceso del usuario administrador con la Blockchain. A través de este servicio, el administrador de datos puede añadir nuevos productores para que tengan acceso a la Blockchain o eliminarles el acceso a esta. Adicionalmente, también puede modificar los precios de las medidas.

Para autenticar al administrador se utiliza el mecanismo `basic authentication` de HTTP. En esta, como usuario se utiliza la dirección de Ethereum del administrador y como contraseña la clave para descifrar la clave privada de su cuenta.

El cuerpo del mensaje que envía el administrador en la petición HTTP varía dependiendo de la función a la que quiera acceder. A continuación se muestran las diferentes posibilidades

Añadir un sensor o eliminarlo

En este caso, si el administrador desea añadir un nuevo sensor a la Blockchain o eliminar uno ya existente, el cuerpo del mensaje contiene tres campos:

1. `action`: Este campo indica la acción que quiere realizar el usuario. Puede tomar tres posibles valores: `addProducer`, `removeProducer` o `setPrice`.
2. `producerAddr`: String que indica la dirección de Ethereum en formato hexadecimal del sensor que se quiere añadir o eliminar.
3. `producerID`: String que indica el identificador único del sensor.

Fijar el precio de una medida

En este caso, para fijar o modificar el precio de una medida, el cuerpo del mensaje que envía el administrador contiene los siguientes campos:

1. `action`: Este campo indica la acción que quiere realizar el usuario. Puede tomar tres posibles valores: `addProducer`, `removeProducer` o `setPrice`.
2. `hash`: String que indica el hash de la medida a la que se quiere poner precio.
3. `price`: Entero que indica el precio que se quiere fijar.

El administrador realiza estas funciones mediante un `POST` a la dirección:

`https://direccionIPMarket:puerto/adminap`

Desde el punto de vista del el `Market`, lo primero que hace nada más recibir una petición a esta ruta es observar el valor de la cabecera `Authorization`. En el caso de que el usuario que realiza la petición no tenga acceso, le devuelve un mensaje de error. En caso contrario, se comprueba el valor `action` del cuerpo del mensaje. En función de este parámetro, el servidor realiza una tarea u otra.

El código de este componente se puede consultar en [26].

3.5. Desarrollo del cliente

El objetivo del cliente que se ha desarrollado, es poder observar los datos publicados en la plataforma Blockchain y permitir la compra de aquellos que le interesen. En este sentido, permite la validación funcional del resto de componentes del sistema.

En el Pseudocódigo 4 se pueden observar los pasos que sigue el cliente para comprar las medidas.

Lo primero que hace el cliente es conectarse al punto de acceso de su nodo. En este caso, en el nodo del cliente se ha habilitado un punto de acceso websocket, además del IPC. Por lo tanto, el cliente puede acceder de manera remota a su propio nodo siempre y cuando tenga la dirección y el puerto de este.

El siguiente paso que debe realizar el cliente para comprar un dato es autenticarse en la Blockchain para poder enviar transacciones. El proceso de autenticación del cliente en la Blockchain consiste en obtener su clave privada de Ethereum a partir de la contraseña que conoce.

Una vez que el cliente ha sido autenticado con éxito en la Blockchain y ha elegido que dato comprar, publica su clave pública en el caso de que no lo haya hecho y comprueba si ya ha comprado esa medida o no. En caso de que la haya comprado, simplemente busca

Pseudocódigo 4: Proceso de compra del cliente

```
Conectarse al nodo del cliente;
Identificarse en la Blockchain;
Elegir un dato para comprar;
Comprobar si su clave pública está en la Blockchain;
if No está then
    Publicar la clave pública en la Blockchain;
Comprobar si el dato que quiere comprar ya lo ha comprado;
if No comprado then
    Comprar la medida;
    Notificar al servidor del Market que ha comprado la medida;
    Esperar la respuesta del Market;
Descifrar la medida de la Blockchain;
```

en la Blockchain las transacciones que contienen la medida cifrada, y la clave con la que se ha cifrado, y la desencripta. En caso contrario, paga el dato mediante la función `payData` del smart contract de pagos.

Tras pagar la medida, notifica al *Market* el pago. Para ello, le envía la petición que se comentó en el apartado 3.4.2. Como respuesta, en el caso de que la medida exista, recibe el hash de la transacción que contiene la clave, cifrada con la clave pública del cliente, y el hash de la transacción que contiene la medida, cifrada con dicha clave.

Para descifrar la medida, el cliente desencripta con su clave privada la clave de cifrado simétrico y utiliza esta última para descifrar la medida.

Se ha desarrollado una interfaz gráfica basada en tecnología web para poder hacer demostraciones y pruebas funcionales de manera ágil e intuitiva. La parte backend de este cliente se ha implementado en código Nodejs y el frontend en HTML, CSS y Javascript.

Los códigos de este componente se pueden encontrar en [27].

Capítulo 4

Despliegue y prueba de concepto del Marketplace

En este capítulo se describe el conjunto de actividades que se han llevado al cabo para la validación y pruebas del sistema que se ha implementado.

4.1. Preparación del entorno de desarrollo

Una vez completado el diseño y desarrollo de la plataforma, es necesario desplegar un entorno de desarrollo para validar el funcionamiento de los distintos componentes que se han desarrollado. En este apartado se describe qué aplicaciones y servicios forman parte de ese entorno y como se ha llevado a cabo su despliegue. En este sentido, cabe destacar que la plataforma se ha desarrollado, por completo, sobre el sistema operativo Ubuntu 19.0.4.

Instalación de las aplicaciones relacionadas con FIWARE

En primer lugar se instalaron los elementos correspondientes al ecosistema FIWARE. Estos son: *Orion* y *Cygnus*.

Para facilitar la portabilidad de la plataforma, ambos componentes se desplegaron mediante contenedores Docker. Para ello, en primer lugar es necesario instalar el entorno Docker en el sistema ejecutando los siguientes comandos.

```
$ sudo apt-get install docker.io
$ sudo apt-get install docker-compose
```

Una vez que se ha instalado Docker en el sistema, para la instalación de *Orion* se crea un fichero `docker-compose.yml` con el siguiente contenido:

```
mongo:
  image: mongo:3.6
  command: --nojournal
orion:
  image: fiware/orion
  links:
    - mongo
  ports:
    - "1026:1026"
```

```
command: -dbhost mongo
```

Para desplegar el contenedor de *Orion* se utiliza el comando `$ docker-compose up` desde el path en el que se haya generado el fichero `.yml`. Una vez que el comando se ha ejecutado, se inicia el OCB y comienza a atender las peticiones que llegan al puerto `1026` (parámetro configurable en el fichero `.yml`).

Durante el desarrollo del proyecto se observó que había ciertos problemas de interconexión entre los componentes Docker y elementos externos. Por lo tanto, se decidió ejecutar los contenedores directamente en la red del anfitrión y no en una subred docker. Por lo tanto, para iniciar el OCB, sin ningún problema, se utilizan los siguientes comandos:

```
$ docker run -d --network host mongo:3.6
$ docker run --network host --name fiware-orion fiware/orion -dbhost
   127.0.0.1:27017
```

Como se puede apreciar en los comandos anteriores, para hacer funcionar *Orion* es necesario utilizar una base de datos (opción `-dbhost`) que controle todos los consumidores que estén suscritos a este elemento. En este caso, la base de datos que se utiliza es `MongoDB`. Se trata de una base de datos no SQL, en la cual la información se encuentra almacenada en formato JSON.

Una vez que se ha instalado el OCB, el siguiente paso consiste en la instalación de *Cygnus*. Para ello, se utiliza el siguiente comando:

```
$ docker pull fiware/cygnus-common
```

Después de que se haya descargado el contenedor Docker de *Cygnus*, es necesario cambiar la configuración de este nodo para que funcione recibiendo tramas en formato `NGSIv2` provenientes del OCB e introduzca la información de estas tramas en la base de datos MySQL. Para que esto sea posible, se configura el fichero `agent.conf` como el que aperece en el Anexo A.1.1.

Una vez que se ha configurado de manera apropiada *Cygnus*, se inicia mediante el siguiente comando:

```
$ docker run -it --network host --entrypoint bash cygnus-ngsi:mysql /opt/
   apache-flume/init.sh
```

En este momento, el contenedor de *Cygnus* se encuentra funcionando en el puerto `5050` (parámetro configurable en el fichero `agent.conf`).

Instalación y configuración de MySQL

Para instalar MySQL se ejecuta el siguiente comando:

```
$ sudo apt install mysql-server
```

Tras ejecutar el comando anterior, MySQL se encuentra funcionado en el puerto `3306` (opción por defecto que debe ser coherente con la configuración de *Cygnus*). A continuación, se añade un usuario con privilegios para modificar contenido en la base de datos. Para ello, desde la interfaz de línea de comandos de MySQL, se ejecutan los siguientes comandos:

```
mysql> CREATE USER  'cygnus'@'%' IDENTIFIED BY 'telematica';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'cygnus'@'%';
mysql> FLUSH PRIVILEGES;
```

Estos comandos crean un usuario cuyo nombre es `cygnus` y cuya contraseña es `telematica`. Este usuario tendrá privilegios para crear bases de datos y tablas dentro del servidor MySQL. El administrador de la plataforma utilizará este usuario, que solo debe conocer él, para introducir contenido en la base de datos y para extraerlo.

Despliegue de la Blockchain

A la hora de realizar el despliegue de la Blockchain que representa la base del marketplace desarrollado en este TFM, se ha decidido utilizar Ethereum Clique como tecnología de Blockchain. A pesar de que la plataforma se ha realizado para esta tecnología, también se ha comprobado que funciona de manera correcta sobre Quorum.

A continuación, se muestran los pasos que se han seguido para desplegar la Blockchain.

En primer lugar, para desplegar Ethereum se han instalado los paquetes necesarios mediante las siguientes instrucciones:

```
$ sudo add-apt-repository -y ppa:ethereum/ethereum
$ sudo apt-get update
$ sudo apt-get install ethereum
```

También, dado que Ethereum está desarrollada sobre el lenguaje de programación Golang, es necesario instalar go en el sistema. Para ello se utiliza la siguiente instrucción:

```
$ sudo snap install --classic go
```

Una vez que se han instalado todas las dependencias que necesita Ethereum, el siguiente paso consiste en el despliegue de los nodos que conforman la red. En este caso se van a desplegar cuatro nodos. Dos que actuarán como nodos validadores y otros dos nodos que no son validadores. De estos dos últimos, uno será un nodo controlado por el administrador (*Market Blockchain Node*) y el otro será un nodo asociado a un cliente (*Wallet Blockchain Node*). Es importante destacar que esta configuración obedece a los objetivos del TFM en que el resultado final debe ser una aprueba de concepto de la plataforma desarrollada, pero nada evitaría escalar esta solución a una Blockchain con más nodos pertenecientes al administrador o a los clientes.

Además de los cuatro nodos que conforman la Blockchain, se va a introducir un quinto nodo que actuará como Bootnode. Cabe destacar que este tipo de nodos no interactúan con la Blockchain. Es decir no proponen ni validan bloques, su única tarea es el descubrimiento de la red.

Cabe destacar que la Blockchain se podría haber desplegado perfectamente sin este nodo de encaminamiento asignando a cada nodo las direcciones de sus vecinos. Sin embargo, se ha preferido hacer uso de este Bootnode para automatizar y simplificar el proceso de búsqueda de vecinos y para facilitar la escalabilidad del sistema en el caso de que fuera necesario añadir más nodos cliente, proveedor y/o validador.

En la Figura 4.1 se puede observar los diferentes nodos que conforman la red Blockchain que se ha implementado.

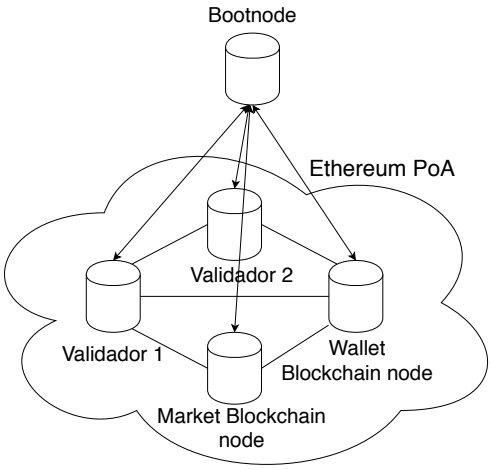

Figura 4.1: Esquema de la red Blockchain desplegada para la prueba de concepto de la plataforma

Para desplegar los dos nodos validadores, en primer lugar, se han creado dos carpetas. Dentro de estas se ha creado una cuenta de Ethereum para cada uno de los nodos. Esto se ha realizado mediante las siguientes instrucciones. En estas, los nodos `node0` y `node1` corresponden con los validadores 1 y 2 respectivamente.

```
$ geth --datadir node0/ account new
$ geth --datadir node1/ account new
```

A continuación, se ha modificado un bloque génesis como el que aparece en el anexo A.1.2. En este, se ha substituido el valor del campo `extraData` por el siguiente valor:

```
0x0000000000000000000000000000000000000000000000000000000000000000006a946a
0fba9e3961d4c193354dc2bc18895fc6ab7e2d1c2bceaecbd8951ff7825b813276434e0d
9300000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
```

Esta linea esta compuesta por los siguientes campos:

1. *Vanity* [32 bytes]: Todo ceros.
2. *Direcciones* [X bytes]: Direcciones de los usuarios validadores, que se han creado previamente, concatenadas.
3. *Proposer Seal* [65 bytes]: Todo ceros.

La modificación del campo `extraData` es necesaria para reflejar los nodos validadores que contiene la red. Al ser un algoritmo de tipo PoA, se necesita que durante el despliegue inicial de la red haya al menos un nodo que actúe como nodo validador. El valor de la dirección de Ethereum de este nodo, se debe de indicar en el bloque génesis. En este caso, dado que se trata de una red que tiene dos validadores, se han añadido las direcciones de ambos nodos en el campo `extraData` para evitar un proceso de votación. En el caso de que solo se hubiera añadido un único validador en este campo, sería necesario realizar un proceso de votación para incluir al otro nodo como validador.

Otros campos a tener en cuenta en el bloque génesis son:

- *chainID*: Identificador de la Blockchain. El número elegido es uno completamente aleatorio.
- *period*: Periodo de tiempo, en segundos, que pasa entre la generación de bloques. En el caso de que el valor elegido sea 0, los bloques se minan a demanda. Es decir, cada vez que se genera un nuevo bloque, se inserta en la Blockchain.
- *gasLimit*: Máximo gas que puede utilizar una transacción. Se ha elegido un número alto.
- *gasUsed*: Precio del gas. Dado que es un Blockchain PoA, se ha decidido que el precio del gas sea 0. De esta manera, se pueden generar transacciones de manera gratuita. Entre los objetivos del TFM no estaba el de definir un modelo de negocio propio para el marketplace, esto es, el actor que opera la propia plataforma, por lo que el hecho de que las transacciones sean gratuitas resulta apropiado. En otros supuestos de modelo de negocio, se podrían estudiar otras opciones para el precio del gas.

El siguiente paso consiste en inicializar los validadores con el bloque génesis que se ha modificado en el paso anterior (`genesis.json`). Para ello se ejecutan los siguientes comandos:

```
$ geth --datadir ./node0 init genesis.json
$ geth --datadir ./node1 init genesis.json
```

Antes de arrancar estos nodos, se va a crear y arrancar el nodo bootnode. Para ello se utilizan las siguientes ordenes:

```
$ mkdir bootnode
$ bootnode --genkey=bootnode/boot.key
$ bootnode -nodekey bootnode/boot.key -verbosity 9 -addr :30310 2>/dev/
   null 1>bootnode/enodeAddr.txt &
```

El último comando de la secuencia anterior, se utiliza para arrancar el bootnode y escribir el arranque en el fichero `enodeAddr.txt`. Una de las líneas que aparece en este fichero es la dirección Ethereum del nodo. Esta dirección es la que van a utilizar los nodos que conforman la Blockchain para encontrarse unos a otros. En el caso concreto del despliegue realizado durante el TFM es:

```
enode://4c6257ff3ab47c204549b8ad4b11acb769b9dddf5eefe5142c2ae9317fd046b05
597a2dbe71eeac8202f506f4de83f69010008932e45e11e86e1a2497fb8c861@127.0.0.1
:30310
```

Finalmente, se arranca el validador 1 mediante la siguiente línea:

```
$ geth --datadir ./ --syncmode full --nat=extip:127.0.0.1 --mine --miner.
   threads 1--verbosity 3 --networkid 56424 --port 30300 --gasprice '0'
   --rpccorsdomain "*" --bootnodes "enode://4c6257ff3ab47c204549b8ad4b11
   acb769b9dddf5eefe5142c2ae9317fd046b05597a2dbe71eeac8202f506f4de83f6901
   0008932e45e11e86e1a2497fb8c861@127.0.0.1:30310" --unlock 6
   a946a0fba9e3961d4c193354dc2bc18895fc6ab --password password.txt
```

Mientras que el validador 2 se arranca mediante este comando:

```
$ geth --datadir ./ --syncmode full --nat=extip:127.0.0.1 --mine --miner.
   threads 1--verbosity 3 --networkid 56424 --port 30301 --gasprice '0'
   --rpccorsdomain "*" --bootnodes "enode://4c6257ff3ab47c204549b8ad4b11
   acb769b9dddf5eefe5142c2ae9317fd046b05597a2dbe71eeac8202f506f4de83f6901
   0008932e45e11e86e1a2497fb8c861@127.0.0.1:30310" --unlock 7e2d1
   c2bceaecbd8951ff7825b813276434e0d93 --password password.txt
```

Cabe destacar las siguientes opciones de estos comandos:

- `syncmode`: Indica el modo de sincronismo que tiene el nodo. En este caso, el nodo almacena los bloques completos.
- `mine`: Indica al nodo que nada mas arrancar tiene que empezar a minar.
- `unlock`: Desbloquea la cuenta que aparece a continuación para que el nodo sea capaz de minar nuevos bloques.
- `gasprice`: Indica que el precio del gas es 0.
- `bootnodes`: Dirección del Bootnode. Su valor, debe coincidir con la dirección asignada al Bootnode.

En este punto, la red Blockchain tiene dos nodos validadores y un bootnode funcionando. El siguiente paso consiste en añadir los nodos del cliente (*Wallet Blockchain Node*) y del administrador de la plataforma (*Market Blockchain Node*). Esto se realiza de manera muy similar al proceso anterior, salvo que ahora ya no es necesario modificar el bloque génesis.

Para crear estos dos nuevos nodos simplemente se tienen que ejecutar los siguientes comandos. En estos, `new-node` corresponde al nodo del administrador y `client-node` al nodo del cliente.

```
# Creación de cuentas asociadas a los nodos
$ geth --datadir new-node/ account new
$ geth --datadir client-node/ account new

# Inicialización de los nodos con el mismo bloque genesis que el
   utilizado con los nodos validadores
$ geth --datadir new-node/ init genesis.json
$ geth --datadir client-node/ init genesis.json
```

Una vez que se han creado los nodos, el siguiente paso consiste en iniciarlos. Para poner en funcionamiento el nodo `new-node` se utiliza el siguiente comando:

```
$ geth --datadir ./ --syncmode full --verbosity 3 --networkid 56424 --
   port 30318 --nat=extip:127.0.0.1 --gasprice '0' --bootnodes "enode://4
   c6257ff3ab47c204549b8ad4b11acb769b9dddf5eefe5142c2ae9317fd046b05597
   a2dbe71eeac8202f506f4de83f69010008932e45e11e86e1a2497fb8c861@127
   .0.0.1:30310"
```

Mientras que para arrancar el nodo del cliente se utiliza este otro:

```
$ geth --datadir ./ --syncmode full --verbosity 3 --networkid 56424 --
   port 30319 --nat=extip:127.0.0.1 --ws --wsport 33019 --wsaddr="
```

```
127.0.0.1" --wsorigins "*" -allow-insecure-unlock --gasprice '0' --
bootnodes "enode://4c6257ff3ab47c204549b8ad4b11acb769b9
dddf5eefe5142c2ae9317fd046b05597a2dbe71eeac8202f506f4de83f69010008932
e45e11e86e1a2497fb8c861@127.0.0.1:30310"
```

Como se puede apreciar, este último comando es ligeramente diferente a los anteriores. Esto se debe a que se incluye una interfaz de acceso al nodo mediante websockets. Gracias a esto, un usuario podrá comunicarse con dicho nodo de manera remota. Los clientes podrán utilizar esta interfaz para interactuar con su nodo en la Blockchain a través de aplicaciones web.

Para comprobar el correcto funcionamiento y despliegue de la Blockchain, se puede establecer una conexión con alguno de los nodos, y comprobar que el número de nodos *peers* o nodos conectados es el que se espera. La Figura 4.2 muestra esta validación para el caso de la Blockchain que se desplegó. Como se puede comprobar el número de *peers* del nodo `new-node` es, para una red de cuatro nodos, precisamente tres.

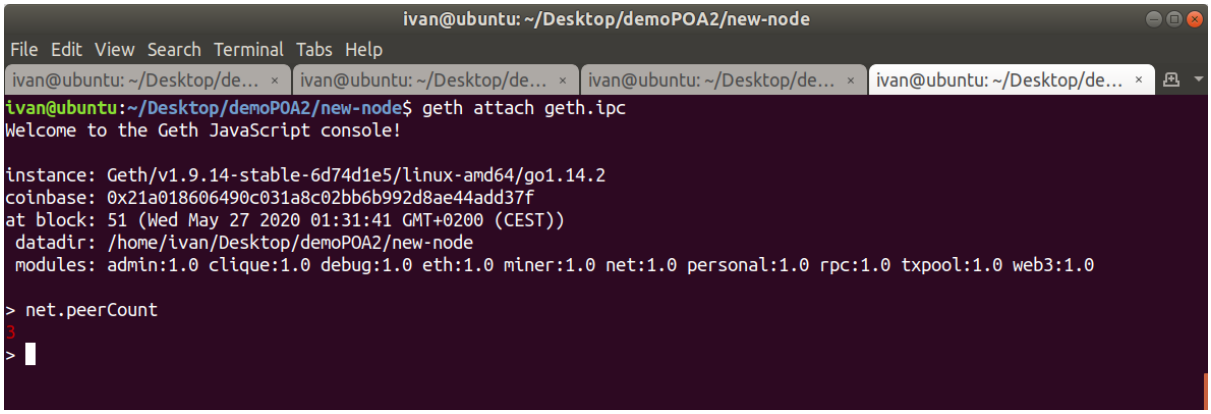

Figura 4.2: Número de nodos conectados a `new-client`

En este punto, la red Blockchain está desplegada por completo. Por lo tanto, el siguiente paso consiste en el desarrollo de los contratos inteligentes que correrán sobre esta.

Para automatizar el despliegue de la red y el despliegue de nuevos nodos se han realizado los scripts que aparecen en [28].

4.2. Puesta en marcha de la plataforma

En primer lugar, se arrancan todos los componentes salvo el *Market API* y el *Wallet Blockchain Node* del cliente. Esto se debe a que primero es necesario desplegar los smart contracts en la Blockchain. Para ello se ejecuta un pequeño código escrito en Go, el cual se puede encontrar en [29]. Como resultado de la ejecución de este script, se despliegan los tres contratos en la Blockchain y se obtienen las direcciones en los que estos se han desplegados. Éstas, son necesarias para configurar tanto el *Market* como el del cliente y que para así puedan interactuar con la Blockchain. Para modificar los valores de las direcciones de los contratos simplemente basta con asignar a las variables que almacenan dichas direcciones, en ambos elementos, los valores que se obtienen en el paso anterior.

Una vez que se han configurado las direcciones de los contratos tanto en el *Market* como el *Wallet Blockchain Node* del cliente, se pueden arrancar estos elementos.

A continuación, el administrador podrá comenzar a realizar peticiones `POST` al path `/adminap` del *Market* para otorgar acceso a la Blockchain a los sensores que hayan solicitado acceso a la plataforma. De esta forma, el productor autorizado ya puede insertar contenido en la Blockchain.

En este instante, la plataforma está desplegada por completo y es operativa.

4.3. Validación mediante prueba de concepto

En este apartado se va a describir la validación funcional que se ha hecho de la plataforma. Para ello, se va a usar como situación: un sensor que mide la temperatura de la habitación de una casa y un cliente que desea comprar estas medidas.

En la Figura 4.3 se puede observar el escenario planteado durante la prueba de concepto de la plataforma.

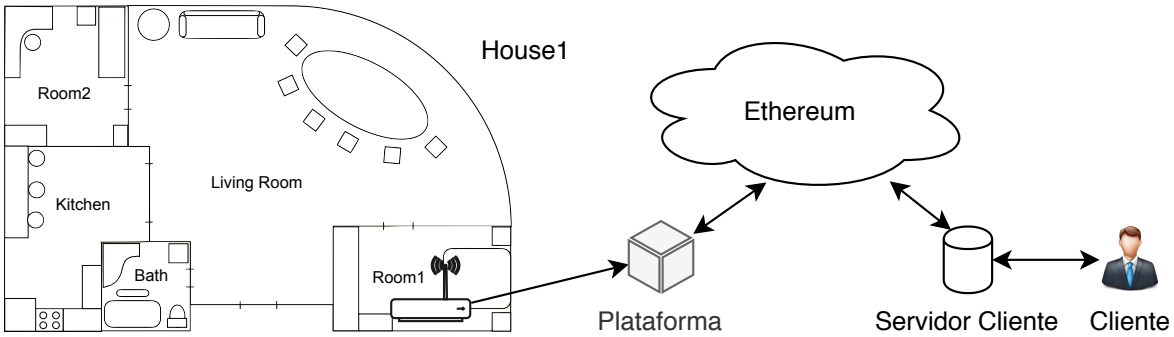

Figura 4.3: Escenario planteado durante para la prueba de concepto

El sensor situado en la habitación *Room1* de la casa *House1* monitoriza la temperatura de la habitación y envía esta información al Marketplace.

Por otro lado, los clientes dispondrán de su propio nodo en dicho Marketplace para poder adquirir, cuando lo deseen, las medidas generadas por dicho sensor.

Con el fin de tener un entorno controlado, el sensor se va a emular utilizando un comando de Linux(i.e. el comando `curl`) que permite enviar peticiones HTTP. En este caso, se enviarán al path correspondiente de la plataforma (`localhost:5051/notify`).

En la Figura 4.4 se puede observar el uso de este comando. El sensor especifica en formato NGSIv2 el valor de la medida, así como la descripción de esta y su clave de Ethereum cifrada con la clave pública del administrador de la plataforma. Además, se especifican también las cabeceras `Fiware-Service` y `Fiware-Servicepath` para identificar el topic de la medida.

```
curl localhost:5051/notify -s -S --header 'Content-Type: application/json' --header '
    Accept: application/json' --header 'Fiware-Service:RoomsControl' --header 'Fiware-
    ServicePath:/house1' -X POST -d @- <<EOF
{
  "id": "Room1",
  "type": "Room",
  "attributes": {
    "type": "Object",
    "value": {
      "temperature": {
        "value": 27,
        "type": "Number"
      },
      "sensorID": {
        "value": "sensor1",
        "type": "String"
      },
      "timestamp": {
        "value": 1588005723790,
        "type": "Number"
      }
    }
  },
  "description": {
    "value": "Measurement of the temperature in the living room of house 1",
    "type": "String"
  },
  "cipher": {
    "type": "String",
    "value": "0x0435c015c0a1076894ba8f862db6c356dab54ddfdee61160daf8ba795b758180
        c18ee2ee014c8ddc31419df932e57eefa37f72193760210
        a479d345e7d9f9bb2d4e6decff266eeb7148aa7d8a31523e4ce74149e6e18eab5bde5462f12c1bda749
        01"
  }
}
EOF
```

Figura 4.4: Envío de una medida tomada por el sensor

Para que Orion envíe las medidas a Cygnus, es necesario que éste, previamente, se suscriba al topic de la medida. Al igual que ocurría con el caso del sensor, la suscripción se realiza también con el comando `curl`. Para ello se envía una petición con el formato adecuado a Orion.

En la Figura 4.5 se puede observar el formato de la suscripción. Cygnus se suscribe a la entidad `Room1` asociada al tema *Fiware-Service: Roomscontrol* en el path *Fiware-Servicepath:/house1*. Estos valores, deben coincidir con los que el sensor incorpora en la cabecera de su petición HTTP.

```
curl -v localhost:1026/v2/subscriptions -s -S --header 'Content-type: application/json
    ' --header 'Accept: application/json' --header 'Fiware-Service:RoomsControl' --
    header 'Fiware-ServicePath:/house1' -d @- <<EOF
{
  "description": "A subscription to get the temperature info about Room1",
  "subject": {
    "entities": [
      {
        "id": "Room1",
        "type": "Room"
      }
    ],
    "condition": {
      "attrs": [ "attributes"]
    }
  },
  "notification": {
    "http": {
      "url": "http://localhost:5050/notify"
    },
    "metadata": ["hash"]
  },
  "expires": "2040-01-01T14:00:00.00Z",
  "throttling": 5
}
EOF
```

Figura 4.5: Suscripción a las medidas tomada por el sensor

Como resultado de la ejecución de los dos comandos, se ha almacenado la medida en la base de datos y la plataforma ha publicado esta medida en la Blockchain con el hash `0xd64100cdb775b42c39f06b7384e06acaade5ff5351eceb82fe1d559273f3f03a` para que cualquier cliente pueda comprarla.

El cliente puede observar las medidas que publican los sensores en la interfaz gráfica que implementa su servidor, tal y como se puede ver en la Figura 4.6. En esta, se puede ver las medidas disponibles que tiene el cliente para comprar. En este caso solo hay una única medida. El cliente puede observar la fecha en la que se insertó la medida en la Blockchain, su precio y la descripción de esta. Como se puede comprobar, la descripción que aparece coincide con la que ha enviado el sensor a la plataforma.

Para que el cliente sea capaz de comprar alguna medida, lo primero que tiene que hacer es registrarse en la Blockchain. Esto lo realiza accediendo a la pestaña que pone login en la interfaz gráfica. El cliente debe utilizar la dirección de su cuenta de Ethereum y su contraseña. En la Figura 4.7 se puede observar la ventana de login que utiliza el cliente para autenticarse frente a la Blockchain.

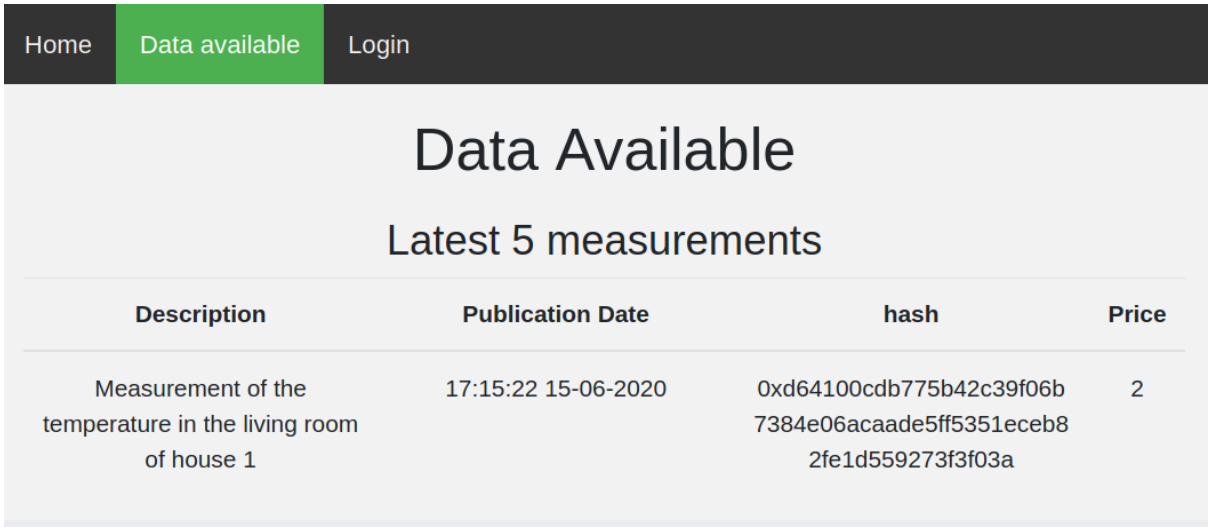

Figura 4.6: Medidas publicadas en el frontend del cliente

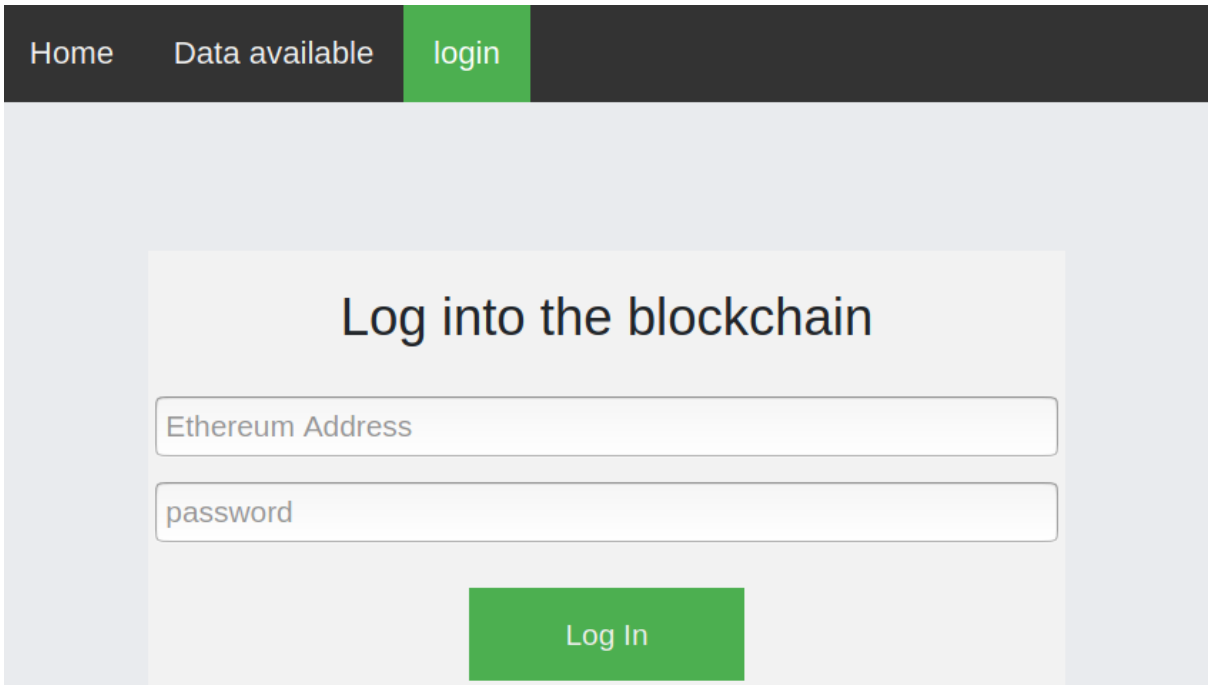

Figura 4.7: Interfaz de login para el cliente

Una vez que el cliente se ha autenticado correctamente frente a la Blockchain, para comprar el dato el cliente solamente tiene que situarse en la pestaña cuyo nombre es Wallet, introducir el hash la medida y darle al botón `buy Data`. Tras darle al botón la medida que ha comprado se muestra en `Purchases`.

En la Figura 4.8 se puede observar la compra de la medida por parte del cliente. En esta se puede observar, la fecha en la que se ha realizado la compra, el hash de la medida que se ha comprado, el hash de la transacción que contiene el proceso de compra, el precio de la medida y el estado de la compra. Este último parámetro se pone de color verde en el caso de que la medida se haya entregado con éxito al cliente.

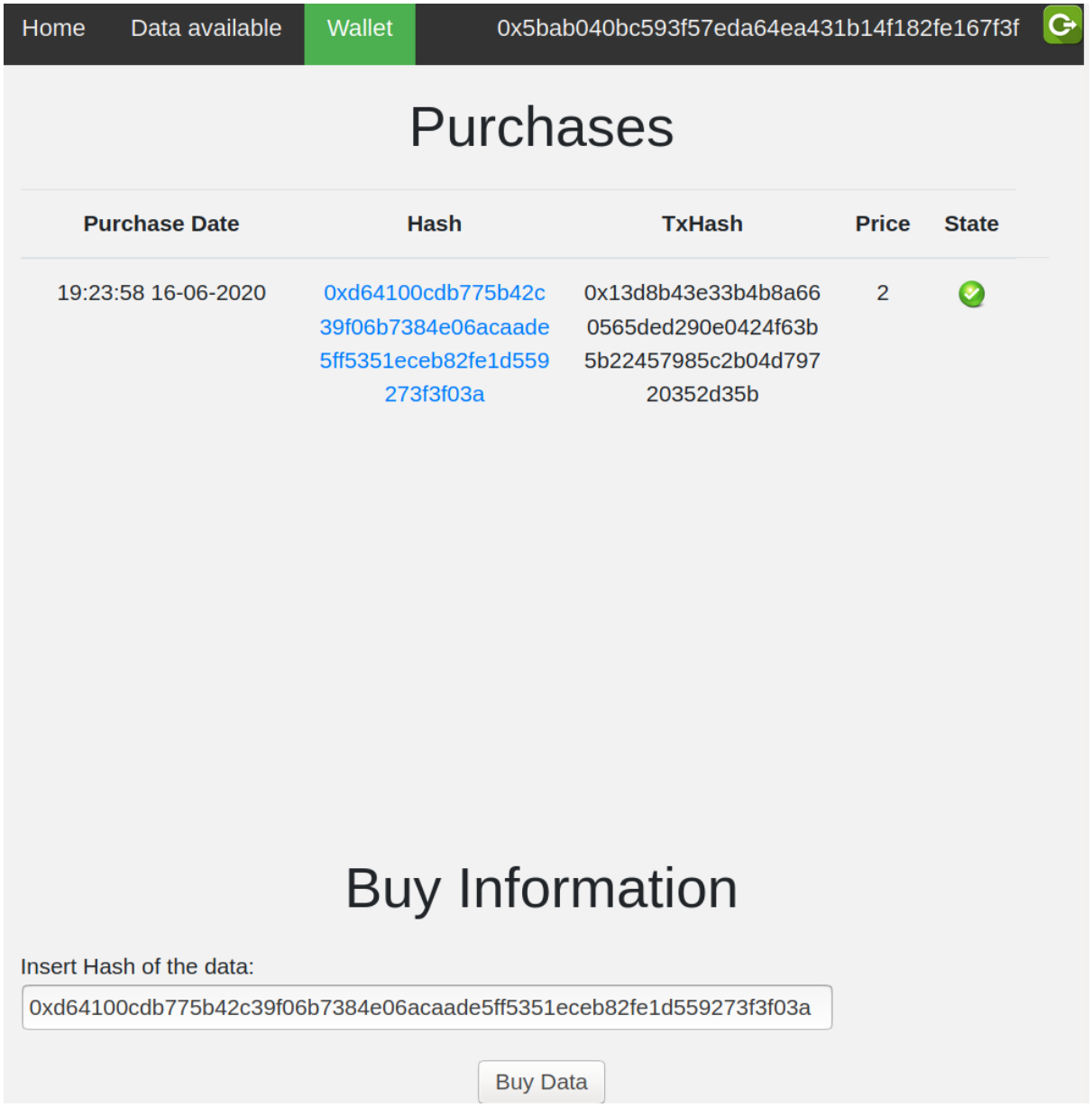

Figura 4.8: Proceso de compra

Adicionalmente, en la parte superior izquierda de la Figura 4.8 se puede ver una serie de números en hexadecimal. Este valor se trata de la dirección de Ethereum del cliente.

Para observar el valor de la medida el cliente solamente tiene que pulsar sobre su hash (parámetro que aparece de color azul). Tras pulsar sobre este valor se abre una ventana en la que se muestra el hash de la transacción que contiene la clave secreta, el hash de la transacción que contiene el dato cifrado con esa clave y la medida en claro. Esto se puede observar en la Figura 4.9.

En el caso de que se publiquen más medidas y el cliente deseé comprarlas, simplemente tiene que repetir este proceso.

Figura 4.9: Observación de la medida comprada

Capítulo 5

Conclusiones y líneas futuras

En este trabajo se ha desarrollado una plataforma que integra la tecnología Blockchain sobre un entorno IoT para habilitar un mercado de datos distribuido que no requiere el uso de terceras partes que centraliza el proceso. Como se ha podido observar, el uso de esta tecnología tiene la gran ventaja de no depender de una tercera parte que confirme que el cliente ha comprado o recibido un dato. De esta forma se puede hacer el intercambio de información de una manera rápida, sin intermediarios y totalmente transparente.

Adicionalmente, el cliente sabe exactamente lo que está comprando y a quién se lo está comprando dado que solamente aquellos sensores que estén autorizados por el proveedor de la información pueden introducir contenido en la Blockchain. De esta forma, el cliente se asegura que los datos que está comprando son fiables dado que proceden de fuentes que son completamente trazables. A partir de esto, es posible aportar al sistema elementos de reputación y calidad de la información.

Además, dado que el dato se almacena en la Blockchain mediante su hash, el cliente se asegura de que no está comprando datos repetidos.

A pesar de las ventajas que se han comentado anteriormente, Blockchain tiene un problema: la escalabilidad. Para que un nodo pueda introducir contenido a la Blockchain tiene que estar totalmente sincronizado con esta. Esto implica que un cliente que deseé poner un nodo para comprar medidas, necesita previamente sincronizarlo con la blokchain lo cual no es un proceso rápido, aunque solo debe realizarse una vez.

Una desventaja que se ha observado durante el desarrollo de la plataforma es que las medidas se envían a través de la Blockchain, a pesar de estar cifradas. Esto supone una sobrecarga de transacciones bastante grande, dado que cada vez que un cliente compre una medida, esta se envía a través de la Blockchain. Esto puede provocar un crecimiento muy rápido de esta.

La principal línea futura de trabajo que se debe seguir es la búsqueda de un método que garantice las ventajas de esta plataforma pero sin que las medidas se transmitan a través de la Blockchain. Sin embargo, garantizar que un cliente reciba una medida no es algo trivial.

Además de la línea de trabajo anterior, se pueden realizar otras como:

- Conectarse a la Blockchain pública de Ethereum, en vez de utilizar una Blockchain

privada como la que se usa a lo largo del trabajo.

- Introducir la plataforma, el servidor REST y el servidor del cliente en contenedores Docker para aumentar la portabilidad del sistema.
- Observar la implementación de esta plataforma en la nueva versión de Ethereum: Ethereum 2.0, la cual se va a desplegar en el próximo año [30].
- Implementar una plataforma de pago como Paypal o tarjeta de crédito para que el cliente pueda comprar esos datos.
- Evaluar el rendimiento de la plataforma cuando se somete a condiciones de uso más exigentes de forma que se pueda llegar a conclusiones más precisas en cuanto a la escalabilidad del sistema.
- Integrar la plataforma con un entorno IoT real y analizar en detalle la escalabilidad del sistema diseñando mecanismos que alivien los problemas que se puedan plantear.

Apéndice A

Anexo

A.1. Ficheros de configuración

A.1.1. Fichero de configuracion `agent.conf` de Cygnus

```
1  cygnus-ngsi.sources = http-source
2  cygnus-ngsi.sinks = mysql-sink
3  cygnus-ngsi.channels = mysql-channel
4
5  cygnus-ngsi.sources.http-source.type = org.apache.flume.source.http.
       HTTPSource
6  cygnus-ngsi.sources.http-source.channels = mysql-channel mongo-channel
       sth-channel ckan-channel hdfs-channel cartodb-channel postgresql-
       channel orion-channel postgis-channel elasticsearch-channel
7  cygnus-ngsi.sources.http-source.port = 5050
8  cygnus-ngsi.sources.http-source.handler = com.telefonica.iot.cygnus.
       handlers.NGSIRestHandler
9  cygnus-ngsi.sources.http-source.handler.notification_target = /notify
10 cygnus-ngsi.sources.http-source.handler.default_service = default
11 cygnus-ngsi.sources.http-source.handler.default_service_path = /
12 cygnus-ngsi.sources.http-source.interceptors = ts gi
13 cygnus-ngsi.sources.http-source.interceptors.ts.type = timestamp
14 cygnus-ngsi.sources.http-source.interceptors.gi.type = com.telefonica.
       iot.cygnus.interceptors.NGSIGroupingInterceptor$Builder
15 cygnus-ngsi.sources.http-source.interceptors.gi.
       grouping_rules_conf_file = /opt/apache-flume/conf/grouping_rules.
       conf
16 cygnus-ngsi.sources.http-source.interceptors.nmi.type = com.telefonica.
       iot.cygnus.interceptors.NGSINameMappingsInterceptor$Builder
17 cygnus-ngsi.sources.http-source.interceptors.nmi.
       name_mappings_conf_file = /opt/apache-flume/conf/name_mappings.conf
18
19 cygnus-ngsi.sinks.mysql-sink.type = com.telefonica.iot.cygnus.sinks.
       NGSIMySQLSink
20 cygnus-ngsi.sinks.mysql-sink.channel = mysql-channel
21 cygnus-ngsi.sinks.mysql-sink.mysql_host = 192.168.199.141
22 cygnus-ngsi.sinks.mysql-sink.mysql_port = 3306
23 cygnus-ngsi.sinks.mysql-sink.mysql_username = cygnus
24 cygnus-ngsi.sinks.mysql-sink.mysql_password = telematica
25 cygnus-ngsi.sinks.mysql-sink.attr_persistence = row
26 cygnus-ngsi.sinks.mysql-sink.attr_native_types = false
```

```
27 cygnus-ngsi.sinks.mysql-sink.batch_size = 1
28 cygnus-ngsi.sinks.mysql-sink.batch_timeout = 30
29 cygnus-ngsi.sinks.mysql-sink.batch_ttl = 10
30
31 cygnus-ngsi.channels.mysql-channel.type = com.telefonica.iot.cygnus.
      channels.CygnusMemoryChannel
32 cygnus-ngsi.channels.mysql-channel.capacity = 100000
33 cygnus-ngsi.channels.mysql-channel.transactionCapacity = 10000
```

A.1.2. Bloque Genesis utilizado como modelo

```
{
    "config": {
      "chainId": 56424,
      "homesteadBlock": 0,
      "eip150Block": 0,
      "eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000
          0",
      "eip155Block": 0,
      "eip158Block": 0,
      "byzantiumBlock": 0,
      "constantinopleBlock": 0,
      "petersburgBlock": 0,
      "istanbulBlock": 0,
      "clique": {
        "period": 0,
        "epoch": 30000
      }
    },
    "nonce": "0x0",
    "timestamp": "0x5ecafd13",
    "extraData": "0x0000000000000000000000000000000000000000000000000000000000000002
        fe0aa42988ef1ac0da1040cd333ea3980b4be320000000000000000000000000000000000000000
        0000000000000000000000000000000000000000000000000000000000000000
        0000000000000",
    "gasLimit": "0x47b760",
    "difficulty": "0x1",
    "mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "coinbase": "0x0000000000000000000000000000000000000000",
    "number": "0x0",
    "alloc": {},
    "gasUsed": "0x0",
    "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000"
  }
```

Bibliografía

[1] O. Wyman, "The Fintech 2.0 Paper." Disponible en: https://www.oliverwyman.com/our-expertise/insights/2015/jun/the-fintech-2-0-paper.html. [Accedido: 24-jun-2020].

[2] "Santander becomes first UK Bank to introduce blockchain technology for international payments with the launch of a new app | Santander UK." Disponible en: https://www.santander.co.uk/about-santander/media-centre/press-releases/santander-becomes-first-uk-bank-to-introduce-blockchain. [Accedido: 05-abr-2020].

[3] "Walmart Joins Hyperledger Along With Seven Other New Members." Disponible en: https://www.forbes.com/sites/robertanzalone/2020/03/08/walmart-joins-hyperledger-along-with-seven-other-new-members/.

[4] "What Companies Are Using Blockchain Technology?." Disponible en: https://101blockchains.com/companies-using-blockchain-technology/.

[5] "LPWAN: The fastest growing IoT communication technology," Oct. 2018. Disponible en: https://www.iot-now.com/2018/10/29/89895-lpwan-fastest-growing-iot-communication-technology/. [Accedido: 6-jun-2020].

[6] V. Reports, "IoT Platform Market Size to Reach USD 13.310 Billion by 2026 - Valuates Reports." Disponible en: https://www.prnewswire.com/news-releases/iot-platform-market-size-to-reach-usd-13-310-billion-by-2026---valuates-reports-301059317.html. [Accedido: 6-jun-2020].

[7] I. Bashir, *Mastering blockchain: distributed ledgers, decentralization, and smart contracts explained*. Birmingham: Packt Publishing, 2. edition, fully revised and updated ed., 2018. OCLC: 1037843721.

[8] "Cap theorem." Disponible en: https://www.ibm.com/cloud/learn/cap-theorem. [Accedido: 10-jul-2020].

[9] "An Illustrated Proof of the CAP Theorem." Disponible en https://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/. [Accedido: 10-jul-2020].

[10] "Qué es una cadena de bloques (block chain)," Mar. 2020. Disponible en: https://www.criptonoticias.com/criptopedia/que-es-una-cadena-de-bloques-block-chain/. [Accedido: 07-abr-2020].

[11] "Why Blockchain is Important in 2020 and Beyond," Dec. 2019. Disponible en: https://101blockchains.com/why-blockchain-is-important/.[Accedido: 07-abr-2020].

[12] A. Narayanan, E. W. Felten, J. Bonneau, A. Miller, S. Goldfeder, and J. Clark, "Bitcoin and cryptocurrency technologies: a comprehensive introduction," 2016. OCLC: 1145890545.

[13] S. Lee and S. Kim, "Countering Block Withholding Attack Efficiently," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, (Paris, France), pp. 330–335, IEEE, Apr. 2019. Disponible en: https://ieeexplore.ieee.org/document/8845116/. [Accedido: 07-abr-2020].

[14] S. Zhang and J.-H. Lee, "Analysis of the main consensus protocols of blockchain," *ICT Express*, p. S240595951930164X, Aug. 2019. Disponible en: https://linkinghub.elsevier.com/retrieve/pii/S240595951930164X. [Accedido: 08-abr-2020].

[15] "HIVE Blockchain Expanding Its Ethereum Mining Operations by 20 Percent | BTCMANAGER," Mar. 2020. Disponible en: https://btcmanager.com/hive-blockchain-expanding-ethereum-mining-20-percent/. [Accedido: 08-abr-2020].

[16] S. Nolan, "Quorum Blockchain Consensus Algorithms," Nov. 2018. Disponible en: https://medium.com/coinmonks/quorum-blockchain-consensus-algorithms-ab38790091. [Accedido: 09-abr-2020].

[17] "Introduction to Smart Contracts — Solidity 0.4.24 documentation." Disponible en: https://solidity.readthedocs.io/en/v0.4.24/introduction-to-smart-contracts.html#a-simple-smart-contract. [Accedido: 09-abr-2020].

[18] "Quorum." Disponible en: http://docs.goquorum.com/en/latest/. [Accedido: 10-abr-2020].

[19] "Developers." Disponible en: https://www.fiware.org/developers/. [Accedido: 6-jun-2020].

[20] "La plataforma FIWARE - Aprende FIWARE en Español." Disponible en: https://fiware-training.readthedocs.io/es_MX/latest/ecosistemaFIWARE/plataformaFIWARE/. [Accedido: 6-jun-2020].

[21] "Orion Context Broker (OCB) - Aprende FIWARE en Español." Disponible en: https://fiware-training.readthedocs.io/es_MX/latest/ecosistemaFIWARE/ocb/#orion-context-broker. [Accedido: 6-jun-2020].

[22] I. González, "Contrato control de acceso." Disponible en: https://github.com/igonzaleztak/blockchain-golang/tree/master/contracts/accessContract/accessControl.sol. [Accedido: 9-jun-2020].

[23] I. González, "Contrato gestión de datos." Disponible en: https://github.com/igonzaleztak/blockchain-golang/blob/master/contracts/dataContract/data_ledger.sol. [Accedido: 9-jun-2020].

[24] I. González, "Contrato gestión de pagos." Disponible en: https://github.com/igonzaleztak/blockchain-golang/blob/master/contracts/balanceContract/balance.sol. [Accedido: 9-jun-2020].

[25] I. González, "Código del broker de medidas." Disponible en: https://github.com/igonzaleztak/sql-rest-server/blob/master/main.js. [Accedido: 9-jun-2020].

[26] I. González, "Código del market." Disponible en: https://github.com/igonzaleztak/blockchain-golang. [Accedido: 9-jun-2020].

[27] I. González, "Códigos del servidor del cliente." Disponible en: https://github.com/igonzaleztak/client-nodejs/tree/alternative1-frontend. [Accedido: 10-jun-2020].

[28] I. González, "Scripts para el despligue de la blockchain," June 2020. Disponible en: https://github.com/igonzaleztak/scripted-ethereum. [Accedido: 9-jun-2020].

[29] I. González, "Código para desplegar los contratos." Disponible en: https://github.com/igonzaleztak/blockchain-golang/tree/deployContracts-alt-1. [Accedido: 10-jun-2020].

[30] "Ethereum 2.0 FAQ by ConsenSys." Disponible en: https://consensys.net/knowledge-base/ethereum-2/faq/. [Accedido: 16-jun-2020].