uc3m | Universidad Carlos III de Madrid

Grado Ingeniería Telemática

2018 - 2019

Trabajo Fin de Grado

APLICACIONES DE CONTRATOS INTELIGENTES EN ETHEREUM

José Romero Solís

Tutor: Marcelo Bagnulo Braun

Leganés 2019

RESUMEN

Hasta ahora, se parte de un concepto de sistemas centralizados, donde las webs se encuentran alojadas en servidores, los cuales controlan y proveen de servicios a los usuarios. A partir del nacimiento de Bitcoin, gracias a la tecnología blockchain, se han desarrollado varios sistemas descentralizados, sin la intervención de terceras partes, dando lugar en el año 2014 a Ethereum.

Ethereum es una criptomoneda como Bitcoin, diferenciándose en que aparte de realizar transacciones, está diseñada para almacenar y ejecutar código, dando comienzo a los contratos inteligentes. Estos novedosos contratos son acuerdos entre dos o más partes, ejecutándose de manera autónoma en un sistema no controlado, ganando tiempo y evitando costes de la intervención de terceros.

En este trabajo se ha hecho uso de la tecnología blockchain para desplegar un contrato inteligente desarrollado en el lenguaje Solidity, gestionando el alquiler de viviendas, pudiendo añadir y alquilar propiedades. Para manipular y facilitar el uso de dicho contrato, se usa una aplicación web. Además, se implementa un dispositivo IoT para la simulación de apertura de la puerta. Con este contrato se garantiza que el alquiler sea efectivo, ahorrando tiempo y evitando que inquilino y propietario se personen en la firma.

Con todo esto, se evalúan los costes de despliegue y de transacción existentes en la red de Ethereum. Además, se valoran los problemas de seguridad y privacidad y sus posibles soluciones, determinando la viabilidad del proyecto.

Palabras clave: Cadena de bloques; Ethereum; Contratos inteligentes, Solidity

AGRADECIMIENTOS

Sin darme cuenta, después de varios años de duro trabajo, estoy redactando las últimas líneas de lo que me va a convertir en titulado de ingeniería telemática. Por ello, no puedo irme sin agradecer a las personas que me ha acompañado durante este tiempo y han hecho que esto sea posible.

Antes de nada, agradecer a todos los profesores, que han conseguido que adquiera todos los conocimientos para acabar satisfactoriamente mis estudios de grado. Mencionar también a mi tutor Marcelo, por permitir desarrollarme en un campo nuevo y que tanta curiosidad me causaba.

También me gustaría agradecer a mis padres, por el apoyo durante este largo tiempo y que me han dado la oportunidad de estudiar lo que me gustaba. Además, agradecer a mis abuelos, que tan felices estarían de verme con este título.

No me olvido de mis amigos de la universidad, que han conseguido que durante estos años todo sea más fácil; sus consejos, motivaciones y risas han sido fundamentales para conseguir todo esto.

A todos, mil gracias.

Jose

ÍNDICE DE CONTENIDOS

RESUMEN ..... **iii**
AGRADECIMIENTOS ..... **vi**
ÍNDICE DE CONTENIDOS ..... **viii**
ÍNDICE DE FIGURAS ..... **xi**
ÍNDICE DE TABLAS ..... **xiii**
1. INTRODUCCIÓN **1**
1.1. Motivación 1
1.2. Objetivos del proyecto 2
1.3. Estructura de la memoria 2
2. ESTADO DEL ARTE **5**
2.1. Blockchain 5
2.1.1. Arquitectura de blockchain 7
2.1.2. Tipos de Blockchain 8
2.1.3. Mineros 9
2.1.4. Criptografía 10
2.1.5. Consenso 11
2.1.6. Actualización de la cadena 12
2.1.7. Monedero y direcciones 14
2.1.8. Clientes 15
2.1.9. Escalabilidad 16
2.2. Ethereum 16
2.2.1. Máquina Virtual de Ethereum 18
2.2.2. Cuentas Ethereum 18
2.2.3. Transacciones y mensajes 20
2.2.4. Contratos inteligentes sobre Ethereum 20
2.3. Solidity 22
3. DISEÑO E IMPLEMENTACIÓN **27**
3.1. El contrato inteligente 28
3.1.1. Estructura de la vivienda 28
3.1.2. Estructura de datos 29
3.1.3. Funciones del contrato 29
3.2. Interfaz de usuario 31

3.2.1. Librería Web3.js 31
3.2.2. Descripción 32
3.3. Dispositivo IoT 34
3.4. Despliegue 37
3.4.1. Entorno de desarrollo 37
3.4.2. Cliente Ethereum 37
3.4.3. Red de pruebas 37
3.4.4. Cuentas de Ethereum 38
4. ANÁLISIS Y CONCLUSIONES 40
4.1. Valoración 40
4.2. Costes 40
4.3. Seguridad 42
4.4. Privacidad 42
4.5. Líneas futuras 43
4.6. Marco regulador 44
4.6.1. Leyes aplicables 44
4.7. Propiedad intelectual 44
ANEXO A – ENTORNO SOCIOECONÓMICO 46
A.1. Impacto socioeconómico 46
A.2. Planificación del proyecto 47
A.3. Presupuesto del proyecto 48
ANEXO B – RESUMEN DE CONTENIDOS EN INGLES 51
B.1. Abstract 51
B.2. STATE OF ART 52
B.2.1. Blockchain 52
B.2.2. Ethereum 53
B.2.3. Solidity 54
B.3 DESIGN AND IMPLEMNENTATION 55
B.4. CONCLUSIONS 58
BIBLIOGRAFÍA 60

ÍNDICE DE FIGURAS

Fig. 2.1. Topologías de red - [3] 5
Fig. 2.2. Funcionamiento de la cadena de bloques - [4] 6
Fig. 2.3. Secuencia de bloques [5] 7
Fig. 2.4. Estructura del bloque [5] 7
Fig. 2.5. Estructura de los bloques simplificada [2] 9
Fig. 2.6. Ejemplo función hash [9] 10
Fig 2.7. Árbol de Merkle [9] 11
Fig. 2.8. Fork por minado simultáneo [11] 13
Fig. 2.9. Soft-fork [12] 13
Fig. 2.10. Hard-fork [12] 14
Fig. 2.11. Ejecución de un contrato inteligente en la blockchain [9] 21
Fig. 3.1. Visión general de la arquitectura 27
Fig. 3.2. Interfaz de usuario – Pantalla Principal 33
Fig. 3.3. Interfaz de usuario – Ventana Emergente 33
Fig. 3.4. Microcontrolador ESP8266. Fuente: Amazon 34
Fig. A.1. Diagrama de Gantt 48
Fig. B.1. Simplified block structure 52

ÍNDICE DE TABLAS

TABLA 2.1. COMPARATIVA BLOCKCHAIN PÚBLICA VS PRIVADA 9
TABLA 2.2.. UNIDADES ETHER 19
TABLA 3.1. ESTRUCTURA QUE MODELA LA VIVIENDA 28
TABLA 3.2.. FUNCIONES DEL CONTRATO 30
TABLA 4.1. COSTE EMPLEADO EN EL DESPLIEGUE 41
TABLA 4.2. COSTE EMPLEADO EN LAS TRANSACCIONES DE FUNCIONES FUNDAMENTALES 41
TABLA A.1. PLANIFICACIÓN DEL PROYECTO 47
TABLA A.2. PRESUPUESTO TOTAL 49
TABLA B.1. MODELING HOUSING STRUCTURE 56
TABLA B.2. CONTRACT FUNCTIONS 56
TABLA B.3. COST EMPLOYED 58

1. INTRODUCCIÓN

1.1. Motivación

Hoy en día los contratos son caros documentos escritos, los cuales están sujetos a leyes y necesitan costosos intermediarios para hacer cumplir dicho contrato. Los contratos inteligentes o *Smart Contracts* evitan los costes económicos y temporales ya que no requieren la intervención humana para llevarse a cabo, ejecutándose de manera autónoma y automática.

Siempre se ha oído hablar de App para referirnos a las clásicas aplicaciones, pero todas estas tienen un problema común: son centralizadas**.** Las aplicaciones centralizadas están en manos, por lo general, de una empresa que realiza cambios sin preguntar a los usuarios. Actualmente, se habla de DApp, que son aplicaciones que no dependen de una entidad central, sino de todos los usuarios que la componen.

Los contratos inteligentes están basados en la tecnología *blockchain*, que ofrece las propiedades de descentralización, inmutabilidad y transparencia, marcando el atractivo fundamental de la tecnología, ya que, al no existir ningún ente controlador, se consigue que los propietarios de la red sean todos los usuarios que participan en ella, permitiendo que los datos sean visibles y la información no sea falseada o modificada.

Antes del nacimiento de la red blockchain, la confianza estaba monopolizada por abogados, por notarios o por bancos. Sin embargo, a partir de la aparición de la tecnología y como consecuencia del desarrollo de contratos inteligentes, se pueden tener acuerdos y transacciones, evitando terceras personas, pudiendo ahorrar costes monetarios y de tiempo.

Blockchain es una de las tecnologías que mayor huella va a dejar en la historia, no sólo por su potente tecnología, sino también por la revolución que va a suponer en el modelo económico actual, haciendo cambiar el sistema financiero global; creando una nueva forma de transacciones, permitiendo una mejor repartición del capital global y un aumento de oportunidades.

1.2. Objetivos del proyecto

Una vez encuadrados en el ámbito del proyecto, se propone una idea de contrato inteligente, para gestionar el alquiler de viviendas usando la tecnología blockchain, evitando que propietario e inquilino se personen en la firma y entrega de llaves, empleando dicho contrato y un dispositivo IoT para la apertura de la propiedad, conectándose directamente con la red.

Para ello, cualquier usuario de la red puede publicar viviendas, proporcionando la dirección, el precio que desea que cueste por segundo y el tiempo máximo que quiere que su vivienda se encuentre alquilada. Por otra parte, un inquilino puede arrendar una propiedad suministrando, en concepto de fianza, el coste que tiene alquilar la misma durante su tiempo máximo. Si al activar el cierre de la vivienda, no se consume el total del tiempo, una de las partes, el inquilino, recibe la parte de dinero que no ha disfrutado, mientras que a la otra le pertenece la cantidad consumida. De la misma manera, un propietario puede cerrar su propiedad, si el inquilino ha consumido el total del tiempo máximo. Además, cuando la transacción del alquiler se produzca, se simula la apertura de la puerta de acceso a la vivienda gracias a un dispositivo IoT.

La meta de este trabajo es diseñar una aplicación totalmente descentralizada, es decir, sin la intervención de ningún ente central. La aplicación debe estar compuesta por un contrato inteligente que medie entre las partes interesadas, inquilino y propietario, además de implementar el dispositivo IoT con el entorno y desarrollar una interfaz de usuario, para poder interactuar de manera fácil e intuitiva.

1.3. Estructura de la memoria

En la presente memoria, se expone el desarrollo e implementación de una aplicación descentralizada, intentando exponer el tema tratado de manera estructurada, marcando una lectura clara y comprensiva. Al finalizar dicho documento, se debe haber comprendido con claridad las bases de la tecnología, su uso y lo que ello supone.

Además de esta introducción, donde se exponen los motivos por los cuales ha sido elegido este proyecto y sus objetivos funcionales. Se añade la estructura de la memoria, que se puede consultar en el índice de este trabajo, aunque aquí, se expone una descripción sobre el contenido de cada sección:

- **Estado del arte**: Se exponen las bases teóricas sobre las cuales de desarrolla este trabajo, comenzando por la tecnología donde se asienta Ethereum y sus contratos inteligentes. Finalizando con los cimientos del lenguaje que permite el desarrollo e implementación de los contratos inteligentes.
- **Desarrollo e implementación:** Finalizada la parte de análisis del capítulo anterior, en esta sección, se explica la parte técnica del proceso para crear una aplicación descentralizada, incluyendo el desarrollo del contrato y la manipulación con el dispositivo IoT, además de exponer las diferentes librerías usadas.
- **Evaluación y conclusiones:** Para finalizar y concluido el desarrollo, se destaca el resultado obtenido de la aplicación y el análisis de los objetivos iniciales. Además, se tienen en consideración, las diferentes prestaciones de la aplicación, así como los costes, la seguridad y la privacidad que posee. Por último, incluye el marco regulador y las leyes aplicadas en el proyecto para verificar su viabilidad.
- **Anexo A – Entorno socioeconómico:** Este primer anexo contiene el estudio del impacto del proyecto para la sociedad, como el uso que se puede hacer de la tecnología y de los contratos inteligentes. Además, se incluye el presupuesto y la planificación para abordar el trabajo.
- **Anexo B – Resumen de contenidos en inglés:** Se incluye el contenido resumido en ingles de las partes más notables de la memoria.
- **Bibliografía:** Finalmente, en este apartado, se aportan las fuentes bibliográficas en las cuales se apoya el trabajo. Cabe destacar, que se ha empleado el formato IEEE para citar las referencias.

2. ESTADO DEL ARTE

2.1. Blockchain

El concepto de cadena de bloques o blockchain, se remonta por primera vez a un artículo publicado en el año 2008, con el seudónimo de Satoshi Nakamoto y que actualmente se conoce por ser la plataforma base de la criptomoneda Bitcoin. Aunque "Bitcoin" y "Blockchain" a veces son utilizados como sinónimos, hay que saber que Bitcoin es sólo una aplicación de la tecnología Blockchain [Fig. 2.1]. La primera implementación de blockchain, fue pensada para el intercambio de dinero digital entre dos partes, evitando hacer partícipe a instituciones fiscales, para ello, se incluyen sistemas y mecanismos para que ambas partes confíen en el intercambio.

Hoy en día, la mayor parte de los servicios que se usan, tienen una estructura centralizada, manteniendo todos los datos en una única ubicación conocida como servidor. En este caso, la cadena de bloques tiene una estructura distribuida, marcando la diferencia respecto a las otras convencionales: la información esta replicada en todos los nodos que componen la blockchain, los datos una vez registrados y validados no se pueden modificar; de hecho, las transacciones se conservan en un histórico desde el nacimiento de la tecnología. Por todo esto, sólo hace falta un único nodo para conservar los datos de la red, evitando la pérdida de la información [2].

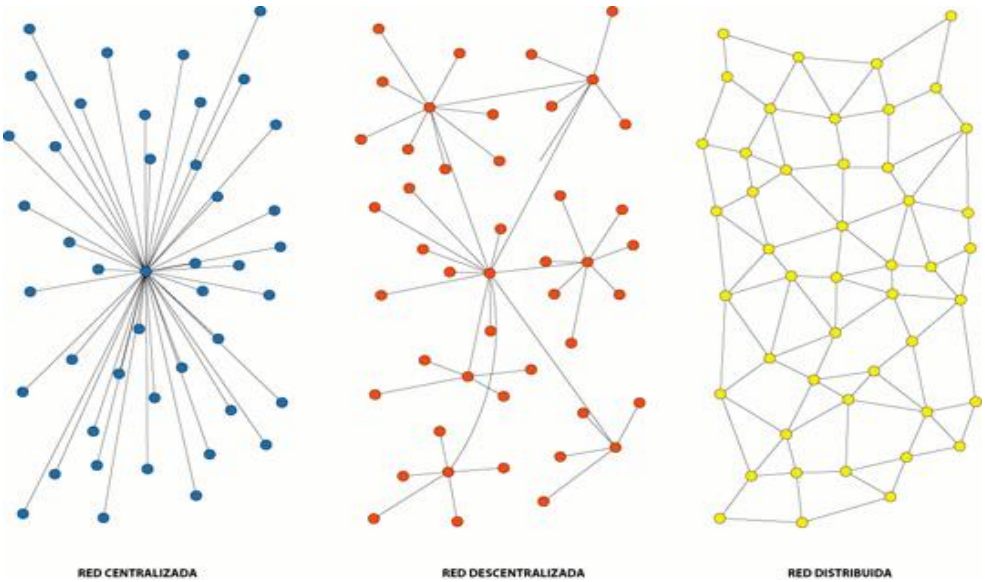

Fig. 2.1. Topologías de red - [3]

Por tanto, se puede decir que la red *blockchain* o cadena de bloques, es una base de datos compartida de manera P2P donde existe una lista creciente de registros o bloques que están enlazados unos con otros de forma cronológica, caracterizada porque la información es inmutable, se encuentra ordenada y únicamente puede ser añadida a la red si hay un acuerdo entre la mayoría de los nodos. Dicho de otro modo, es una especie de "libro" donde se van anotando todos los registros de cualquier tipo de transacción y donde cada transacción es verificada en consenso por todos los participantes.

Fig. 2.2. Funcionamiento de la cadena de bloques - [4]

Blockchain permite administrar datos y transacciones gracias al sistema descentralizado y distribuido, donde los bloques de información se encadenan de manera secuencial generando registros no modificables. Todo el conjunto de los bloques es compartido en los nodos de la red, verificando que las transacciones emitidas en el sistema sean correctas.

El funcionamiento básico, siguiendo la Fig. 2.2, es el siguiente: si un usuario A quiere mandar dinero a B, encapsula la información en una transacción, el resto de los nodos la reciben y verifican que el balance de A es superior a la cantidad establecida. Una vez verificado mediante consenso, el bloque puede añadirse a la red.

Resumiendo, todos los nodos de la red tienen una copia de toda la información, evitando así, que estos datos sean falseados, modificados o eliminados. Toda la información es verificada mediante consenso por los miembros de la red. Este consenso criptográfico,

basado en funciones hash, asegura que exista una única versión de los datos almacenados y de cada transacción. Por todo lo anterior, se introduce el concepto de descentralización, basado en la confianza de conexiones P2P, en el cual no se necesita la regencia de un miembro central. Por lo que se puede decir que, la cadena de bloques marca la diferencia respecto a las habituales bases de datos, ya que no existe un ente central o servidor, además, tendremos una base de datos descentralizada, compartida y replicada.

2.1.1. Arquitectura de blockchain

Como ya se ha mencionado, blockchain es una secuencia de bloques que alberga una lista de transacciones. La Fig. 2.3, muestra una secuencia de bloques de la cadena. En la cabecera de cada bloque, existe una referencia del bloque padre, es decir, el resultado de la función hash del bloque anterior. De esta manera, todo bloque queda unido con su antecesor, formando una cadena. Hay que destacar que, el primer bloque de una cadena es conocido como bloque génesis y no posee referencia del bloque padre.

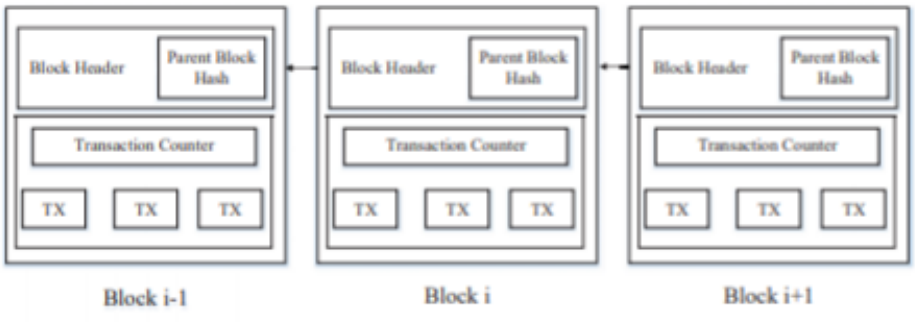

Fig. 2.3. Secuencia de bloques [5]

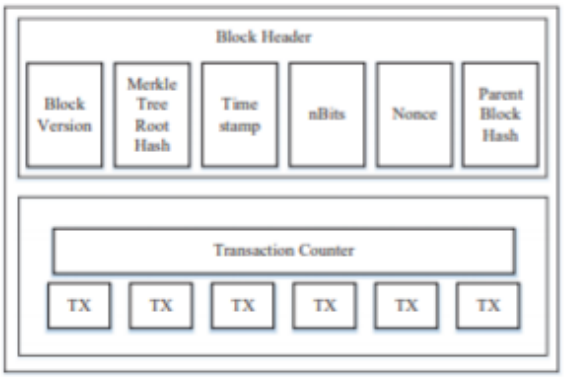

Fig. 2.4. Estructura del bloque [5]

Una vez detallado como se encuentran unidos los bloques, se explica el interior del bloque. Básicamente, un bloque se compone de una cabecera y un cuerpo, como se muestra en la Fig. 2.4. La cabecera del bloque este compuesto por [5]:

- Versión del bloque: indica las reglas de validación a seguir para ese conjunto de bloques.
- Hash del nodo raíz del árbol de Merkle: el resultado de aplicar la función hash a todas las transacciones del bloque.
- Marca de tiempo: número de segundos transcurridos desde el 1 de enero de 1970.
- nBits: número de ceros que tiene al principio el resultado hash de la cabecera. Regula la dificultad.
- Nonce: campo de 4 bytes arbitrarios, para ajustarse con el nBits.
- Hash del bloque padre: valor de hash de 256 bits que apunta al bloque anterior.

Por otra parte, el cuerpo del bloque está compuesto por un contador y las transacciones propiamente dichas. Cabe destacar, que el número máximo de transacciones que puede tener un bloque depende del tamaño del bloque y del tamaño de cada transacción.

2.1.2. Tipos de Blockchain

A pesar de que cuando se piensa en Blockchain se suele referenciar erróneamente con Bitcoin, donde cualquier persona puede participar en dicha red, se han creado y desarrollado diferentes tipos de redes según las necesidades. Según las características que posea, puede realizarse una clasificación según el acceso a los datos para controlar la ejecución de transacciones. Existiendo cadenas públicas o privadas [6] [7].

En una blockchain pública o sin permisos, la cadena de bloques es accesible libremente por cualquier usuario que desee participar en ella, ajustándose a los mecanismos de consenso [2.1.5] de la comunidad. Este tipo de cadenas, se caracterizan por necesitar tokens nativos (Ej. Ether) para poder incentivar a los usuarios y mantener el sistema. Destacan las famosas redes de Bitcoin o Ethereum.

Aunque en las redes privadas el funcionamiento es similar a las otras, se distinguen de las públicas, en que el acceso es sólo para aquellos que han requerido una autorización previa y han sido admitidos. Es decir, en esta red existe una lista de sujetos, con identidades

conocidas, habilitados para llevar a cabo transacciones. Es cada vez más común, que las empresas creen sus propias redes para el almacenamiento interno de datos.

TABLA 2.1. COMPARATIVA BLOCKCHAIN PÚBLICA VS PRIVADA

	PÚBLICAS	**PRIVADAS**
Acceso	• Sin restricciones	• Organización
Participantes	• Sin permisos • Anónimos	• Con permisos • Identidad conocida
Seguridad	• Mecanismos de consenso	• Aprobación previa de los participantes
Velocidad de transacción	• Baja	• Alta

2.1.3. Mineros

Muchos participes del sistema blockchain no se encuentran conectados a la red para hacer uso de los servicios, sino para crear nuevos bloques. A medida que se van formando transacciones y/o contratos hay que almacenar esa información en nuevos bloques.

Básicamente, para poder escribir y crear nuevos bloques es necesaria la existencia de nodos mineros que participen en la validación de transacciones. Los mineros compiten en la resolución de un "enigma" muy complejo que requiere mucha potencia de computación. Por ello, los nodos obtienen una recompensa económica por prestar el servicio de computación al resto de usuarios. Una vez el minero cree haber resuelto el problema matemático, el resto de la comunidad verifica que la solución es correcta. Si es así, el nuevo bloque es añadido a la cadena, quedando la información consolidada, y, recibiendo el minero su recompensa económica.

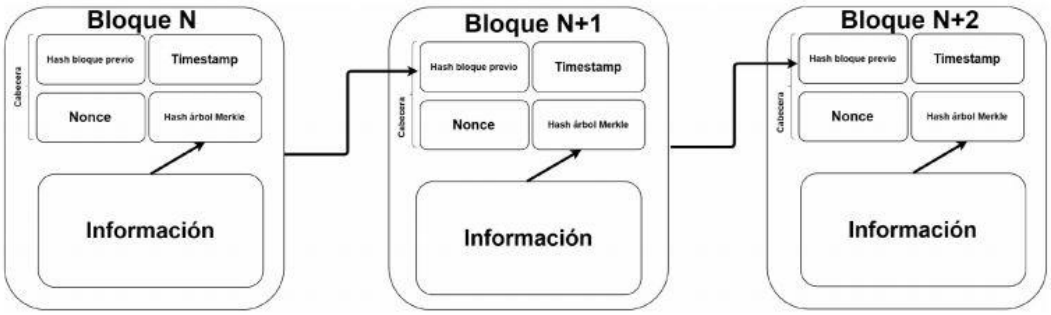

Fig. 2.5. Estructura de los bloques simplificada [2]

En las redes de Bitcoin o Ethereum, la lucha que permite alcanzar un acuerdo entre los mineros para confirmar las transacciones es la prueba de trabajo o *Proof-of-work* (*PoW*). Dicho de otra manera, un bloque es aceptado e introducido en la *blockchain* cuando hay un minero que ha completado la prueba de trabajo.

En cada bloque se incluye la función hash del bloque anterior, así la cadena se encuentra unida consecutivamente. Siguiendo la Fig. 2.5, para añadir un nuevo bloque (n+3), se incluye el resultado de la función *hash* del bloque anterior, que se introduce en la cabecera del nuevo bloque. A continuación, se añaden todas las transacciones, formando un árbol de Merkle [2.1.4] y se busca por fuerza bruta un parámetro (nonce) que, al hacer hash sobre el nuevo bloque, el resultado comience con tantos ceros como haya establecido la comunidad [8].

2.1.4. Criptografía

Partiendo que la criptografía es el arte y la ciencia de ocultar un mensaje, esta sección estará centrada en la verdadera esencia de la cadena de bloques: la función hash y el árbol de Merkle, teniendo como finalidad verificar la integridad de ésta.

En cada bloque se incluye la función hash del anterior, de esta manera quedan enlazados unos con otros. Por ello, es importante saber qué hace la función hash: teniendo una entrada con una secuencia de bits, al hacer la función hash sobre la misma, se obtiene a la salida una secuencia de bits fija, que no depende del número que haya a la entrada, si se cambia alguno de estos a la entrada, se obtiene un resultado totalmente diferente. La función hash se caracteriza porque dada una salida, no hay manera de reconstruir la entrada.

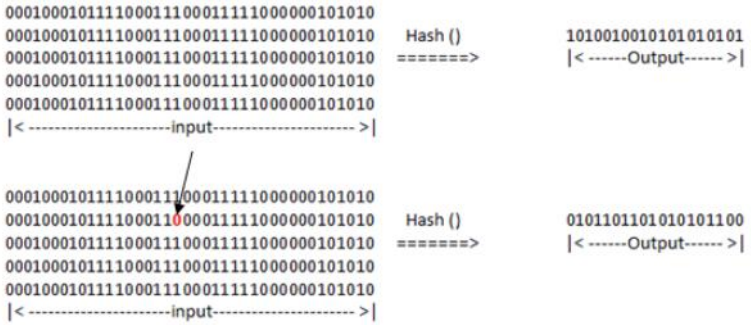

Fig. 2.6. Ejemplo función hash [9]

En cambio, para almacenar todas las transacciones en un bloque se usa una estructura llamada árbol de Merkle. Es una estructura de datos que está basada en combinaciones de funciones hash. Típicamente el árbol de Merkle es representado como árbol binario, donde cada nodo tiene como máximo dos hijos. Este algoritmo permite que las transacciones no sean alteradas, garantizando que, una vez formado el bloque, las transacciones no se puedan modificar. Cualquier intento de manipulación provocaría un cambio en los hashes hasta llegar al nodo raíz (*hash root*). Cabe destacar, que en bitcoin la función hash utilizada es SHA_{256}.

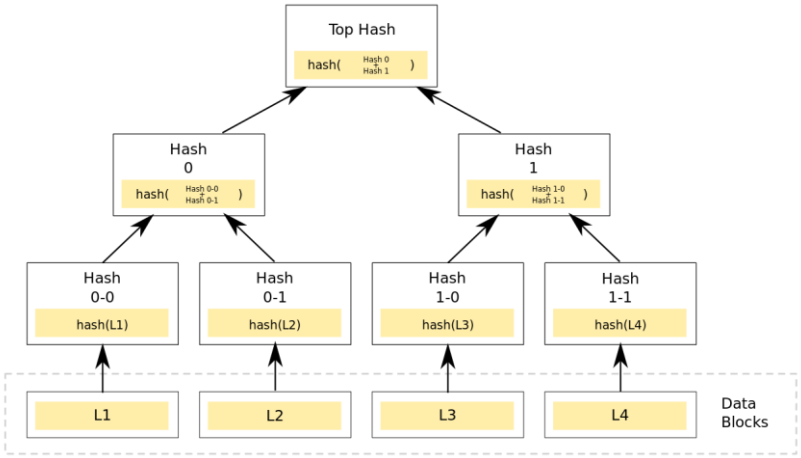

Fig. 2.7. Árbol de Merkle [9]

2.1.5. Consenso

A pesar de que el algoritmo usado en Ethereum y Bitcoin es el famoso *PoW (proof-of-work),* existen otros tipos de mecanismos de consenso, es decir, alternativas a la creación de bloques para su posterior adhesión a la cadena de bloques. Se explican algunos de ellos [2]:

- Proof-of-Work (PoW): Este mecanismo de consenso, es usado en redes Bitcoin o Ethereum. Como ya sabemos, en las redes descentralizadas, alguien debe ser el elegido para registrar y validar las transacciones. Aunque la manera más sencilla es que se eligiese de manera aleatoria, puede traer consigo problemas de ataques al sistema. Por tanto, la manera de evitarlos es que, si un nodo quiere registrar un bloque, necesite emplear mucho trabajo computacional para demostrarlo. En PoW, cada minero calcula el resultado de aplicar la función de hash a la cabecera

del bloque. Como ya se sabe, la cabecera del bloque contiene el campo "nonce", siendo el único campo modificable de éste. Los mineros pueden variar este valor para obtener diferentes valores de hash, hasta conseguir que el resultado comience con tantos ceros como estén establecidos. Una vez el minero cree haber conseguido este valor, transmite el bloque al resto de nodos para que se lo confirmen. Si el bloque es validado, el resto de los nodos añaden el bloque a sus cadenas.

- Proof-of-Stake (PoS): Mecanismo alternativo a PoW, pensado para ahorrar energía. En este caso, los mineros demuestran la cantidad de monedas que poseen. Por tanto, la idea principal es que los mineros con más monedas no van a tener intención de corromper la cadena y van a ser los elegidos para minar el siguiente bloque. Esta selección, basada en el balance de las cuentas de los mineros es injusta, porque, aquel más "rico" es el que domina la red.
- Proof-of-Importance (PoI): Es un algoritmo similar al PoS, pero con ciertas modificaciones. En vez de asignar a los mineros que posea el mayor número de tokens, en PoI el que más cantidad de transacciones mueva, es el que mayor probabilidad tiene de minar el siguiente bloque.

2.1.6. Actualización de la cadena

Ante una estructura descentralizada, y a pesar de existir diferentes mecanismos de consenso, pueden darse bifurcaciones en la cadena (*forks*).

En el minado, pueden darse ambigüedades y producirse bifurcaciones, por ejemplo, no saber qué bloque ha sido generado antes, que se cambien las reglas o que se actualice la tecnología [10].

- **Bifurcación por minado simultaneo**: partiendo de que cualquier nodo puede llegar a minar un bloque, podría ocurrir que dos nodos lleguen a resolver la prueba de trabajo casi de manera simultánea. Por este motivo y problemas de latencia, algunos nodos recibirán el bloque producido por el primer minero y otros por el segundo. La ambigüedad se resuelve cuando se recibe otro nuevo bloque, considerándose válida la cadena donde se ha generado este nuevo bloque.

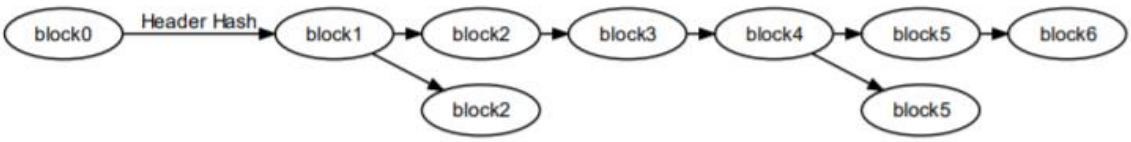

Fig. 2.8. Fork por minado simultáneo [11]

- **Bifurcación por cambios en las reglas:** a pesar de que es muy común la bifurcación por minado simultáneo, existen *forks* usados por los desarrolladores para llevar a cabo modificaciones del protocolo, ya sea para agregar actualizaciones o proteger la red de ataques. La nueva versión que se lleve a cabo dependerá de la cantidad de mineros que la implementen. Existen dos tipos de forks: fuertes y suaves.

  Suaves: también conocido en inglés como *soft-fork*. En este tipo de bifurcación, existe una compatibilidad entre los bloques generados por la versión nueva y la antigua, es decir, los nodos que aún no han actualizado a la nueva versión también reconocen los bloques de la nueva, pero los nodos actualizados no aceptan los bloques de la versión antigua, de esta manera se facilita que los nodos actualizados creen la cadena más larga. Una vez que la mayoría de los nodos hayan actualizado, la cadena estará formada únicamente por bloques de la versión renovada y los bloques desactualizados no serán reconocidos por la mayor parte de los nodos.

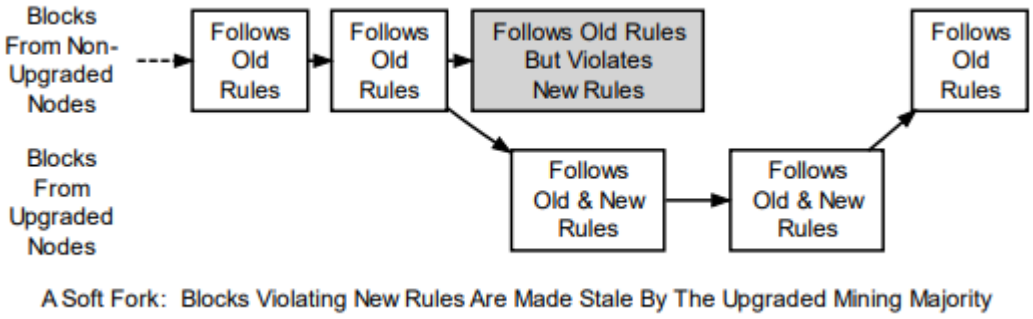

Fig. 2.9. Soft-fork [12]

Fuertes: o *hard-fork*, consiste en cambiar radicalmente el protocolo usado hasta entonces. En este tipo de bifurcaciones, no existe compatibilidad alguna con las versiones anteriores. Un bloque producido por un nodo actualizado es aceptado por uno renovado y rechazado por un nodo desactualizado. El grupo que tenga los nodos actualizados seguirá las nuevas reglas, mientras que el grupo de nodos no actualizados conservará la cadena con las reglas antiguas.

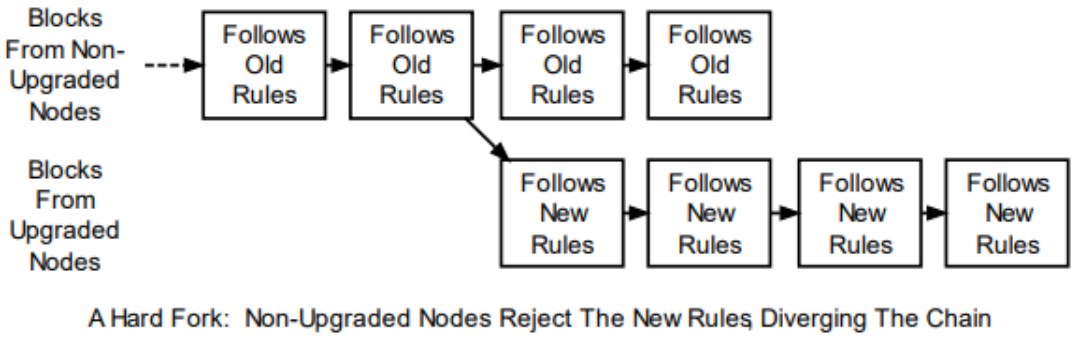

Fig. 2.10. Hard-fork [12]

2.1.7. Monedero y direcciones

Para almacenar las criptomonedas, se emplean direcciones de cuentas que constan de 20 bytes. Para ello, cada participante de la red tiene que crear un fichero donde almacene las claves usadas para firmar todas las transacciones. Este archivo no es almacenado en la blockchain, sino que debe ser almacenado en el disco duro local de cada usuario.

Las claves usadas para firmar las transacciones están compuestas de una parte pública y otra privada.

- La clave privada es cualquier número de 256 bits generado de forma aleatoria.

$$K_{priv} \in \{0,1\}^{256} \tag{2.1}$$

- En cambio, la clave pública está matemáticamente relacionada con la clave privada, es decir, la calve pública es generada a partir de la privada mediante la criptografía de curva elíptica ECDSA [13]. Cabe destacar, que es imposible obtener la clave privada dada una pública.

$$K_{pub} = ECDSA_{512}(K_{priv})$$

(2.2)

Cuando un propietario de una dirección lleva a cabo una transacción, se crea una firma usando la clave privada y se revela al mismo tiempo la clave pública. La dirección usada públicamente para las transacciones está formada por 160 bits (20 bytes) y es generada mediante las funciones *hash* $RIPMED_{160}$ y SHA_{256} [14].

$$K_{direccion} = RIPEMD_{160}\left(SHA_{256}(K_{pub})\right)$$

(2.3)

Por lo general, los usuarios almacenan sus direcciones y sus claves públicas y privadas en *wallets* o monederos, custodiando todas las direcciones, visualizando la cantidad de criptomonedas que tienen.

2.1.8. Clientes

Para acceder a la red es necesario un proveedor que te de acceso al sistema descentralizado, que son los llamados nodos o clientes. Existen dos tipos de clientes o nodos:

- **Los clientes completos o *full nodes*:** descargan todos los bloques, validando cada uno de los bloques y transacciones. Además, los nodos completos suelen generar el monedero donde se almacenan las direcciones de las cuentas [2.1.7]. Uno de los problemas de estos clientes, es que al descargar todo el histórico de transacciones, necesita mucho espacio en disco. Actualmente, la red principal de Ethereum tiene una magnitud de unos 60 GB y va en aumento.
- **Los clientes ligeros o *light nodes***: no almacenan las transacciones completamente, pero si se apoyan en servidores externos para obtener el resto de la información. Al igual que los clientes completos, generan el monedero para las cuentas.

2.1.9. Escalabilidad

En todo momento se están generando nuevas transacciones, haciendo que la cadena de bloques vaya aumentando su tamaño. Como ya se sabe, cada nodo almacena todas las transacciones, para posteriormente, verificar y validar los nuevos bloques generados. Además, actualmente, la cadena de bloques de Bitcoin sólo procesa aproximadamente siete transacciones por segundo, incumpliendo el requisito de que millones de usuarios procesen transacciones. También hay que destacar que, como la capacidad de los bloques es muy pequeña, los mineros prefieren transacciones con altas comisiones a transacciones pequeñas, pudiendo existir retrasos en éstas. [5]

Por todo esto, se han propuesto algunas soluciones a estos problemas:

- **Optimizar el almacenamiento de blockchain**: para un nodo, cada vez es más complicado gestionar el histórico completo de transacciones, por tanto, se ha propuesto un nuevo esquema: eliminar los registros antiguos de la red y mantener una base de datos con las cuentas de direcciones que no se encuentren vacías.
- **Rediseñar la blockchain**: se ha propuesto crear una nueva generación de Bitcoin (Bitcoin-NG), cuya idea principal es dividir el bloque actual en dos partes. Un bloque, *key block*, que elija un líder, es decir, se escoja un minero que decida las transacciones que van a ser añadidas a la red. Otra parte, sería el llamado microbloque, donde se incluirían las transacciones.

2.2. Ethereum

De manera introductoria, Ethereum es una plataforma distribuida y descentralizada, introducida por Vitalik Buterin, que al igual que Bitcoin, utiliza como tecnología base la red blockchain.

Esta cadena de bloques, además de que sus nodos descarguen todo el histórico de transacciones, como en bitcoin, también almacenan el estado actual de las cuentas. En esta base de datos también se almacena código, permitiendo ejecutar programas, dando lugar al desarrollo de contratos inteligentes. Para entender lo que realmente significa Ethereum, se puede decir que, Ethereum es un gran sistema computacional, el cual puedes usar libremente, jugar con él o incluso modificarlo para satisfacer cualquier necesidad. A

parte, puedes crear tu propia moneda personalizada y ofrecerla a quien esté dispuesto a consumirla.

Ethereum tiene como finalidad construir un sistema alternativo para desarrollar DApps o aplicaciones descentralizadas, es decir, una plataforma abierta donde cualquier persona puede crear un sistema descentralizado, que funcione con la tecnología *blockchain*. De la misma manera que Bitcoin, Ethereum no es controlado por nadie, aunque se diferencia de Bitcoin que Ethereum fue construido para ser configurable [15].

La filosofía que está detrás del diseño de Ethereum es la siguiente:

- **Simplicidad**: el protocolo de Ethereum debe ser lo más simple posible, tanto en el almacenamiento de los datos como en la eficiencia del tiempo. Un programador con conocimientos medios debería ser capaz de implementar toda la especificación.
- **Universalidad**: Ethereum proporciona un lenguaje interno a alto nivel, en el que cualquier programador puede usarlo para construir cualquier contrato inteligente o inventar su propia moneda.
- **Modularidad**: las partes del protocolo de Ethereum deben diseñarse para que sean lo más modulares y separables posibles, es decir, durante el desarrollo, el objetivo principal es que, si se hiciera una modificación en el protocolo, la pila siguiese funcionando. El desarrollo de Ethereum debe hacerse para beneficiar a todo el sistema, no sólo a sí mismo.
- **Agilidad**: a pesar de que con las modificaciones a alto nivel hay que ser muy cautos, si en las pruebas posteriores al proceso de desarrollo, se encuentran ciertas modificaciones y oportunidades, se deben explotar.
- **Sin discriminación y sin censura**: el protocolo de Ethereum no debe restringir características específicas de uso. Los mecanismos reguladores que lo componen deben diseñarse para regular el daño y no oponerse a aplicaciones indeseables específicas. Cualquier programador puede ejecutar un bucle infinito, siempre que esté dispuesto a pagar la tarifa computacional.

2.2.1. Máquina Virtual de Ethereum

Como se ha dicho anteriormente, Ethereum es una cadena de bloques que se puede programar, diferenciándose de otras cadenas de bloques que, en vez de ofrecer un conjunto de operaciones predefinidas, Ethereum permite a los usuarios construir sus propias operaciones y aplicaciones.

Esto es posible gracias a la máquina virtual de Ethereum o EVM (Ethereum Virtual Machine), que puede gestionar código de cualquier complejidad y puede ser programado en varios lenguajes de programación existentes, como Solidity (similitudes con C y JavaScript) o Serpent (similar a Python). La máquina virtual de Ethereum es considerada el corazón de Ethereum, ya que es el centro neurálgico de la plataforma. Es el sistema que permite tener conectado todos los nodos y actúa como el motor de la plataforma.

La EVM ejecuta contratos inteligentes (*smart contracts*), considerados los acuerdos del futuro. Un contrato inteligente es un código ejecutable que corre en la cadena de bloques, facilitando y haciendo cumplir los términos del contrato. Estos contratos digitales se encuentran automatizados, además de ser rápidos y efectivos. Un contrato inteligente está compuesto por un balance, un almacenamiento y un código ejecutable. El balance y el almacenamiento forman parte del estado del contrato, que son guardados en la *blockchain* y únicamente son modificados cada vez que el contrato sea llamado o invocado [16].

En todo momento, la base de datos de Ethereum es actualizada por cientos de nodos conectados en la misma red, que se encuentran ejecutando la EVM y las mismas instrucciones. Por ello, podemos decir que Ethereum podría ser un gran ordenador mundial, donde cada nodo ejecuta la EVM con el fin de mantener el mismo consenso en la cadena de bloques.

La plataforma de Ethereum no tiene ningún valor en sí misma, sino que al igual que otros lenguajes de programación depende de empresarios y programadores para exprimir al máximo la plataforma.

2.2.2. Cuentas Ethereum

Como Ethereum es una red pública y sin permisos [2.1.2], existe un token asociado a la plataforma, el “ether”, utilizado para realizar transacciones y/o pagar el “impuesto” llamado GAS. El ether es almacenado en las cuentas, existiendo dos tipos de cuentas:

- Cuentas de propiedad externa (EOAs): Cuenta controlada de manera externa por un usuario y mediante claves privadas.
- Cuentas contrato: Cuenta que está controlada por un código de contrato asociado, y únicamente puede ser activada por una EOA.

Las cuentas contrato sólo realizan operaciones cuando se lo indica una cuenta de propiedad externa. Por tanto, no es posible que una cuenta contrato realice operaciones nativas, como la generación de números aleatorios o las llamadas a API, sólo pueden hacer estas cosas si lo solicita una EOA.

Aunque el ether es el token predeterminado, existe un sistema métrico con unidades más pequeñas al ether. A nivel interno, las transacciones se suelen hacer en "Wei". A continuación, se exponen las unidades:

TABLA 2.2.. UNIDADES ETHER

UNIDAD	**VALOR WEI**	**WEI**
wei	1 wei	1
Kwei	10^3wei	1000
Mwei	10^6wei	1000000
Gwei	10^9wei	1000000000
microether	10^{12}wei	1000000000000
milliether	10^{15}wei	1000000000000000
ether	10^{18}wei	1000000000000000000

2.2.3. Transacciones y mensajes

El termino transacción se usa en Ethereum para referirse al paquete de datos firmado que almacena un mensaje y se envía desde una cuenta de propiedad externa. Las transacciones contienen:

- Cuenta destinatario.
- Cuenta del remitente.
- La cantidad de Ether que desea transferir el emisor (*value*).
- Un campo de datos opcional.
- Un valor de *STARTGAS*, cantidad máxima de gas que se puede consumir.
- Un valor de *GASPRICE*, que simboliza la cantidad de ether que el emisor está dispuesto a pagar por la unidad de gas.

Los tres primeros son campos estándares, que están presentes en cualquier criptomoneda, mientras que los campos STARTGAS y GASPRICE son críticos para evitar bucles infinitos indeseados. Aunque se ha mencionado anteriormente, aquí se introduce el concepto de GAS, utilizado para pagar las tarifas de transacción en Ethereum. La unidad de gas representa el coste para ejecutar una operación en la EVM. El precio del gas puede ser definido por el que origina la transacción, siendo los mineros los que deciden si la incluyen en sus bloques o no. Cabe destacar, que cuanto mayor sea el costo de computación más alto será el gas.

En cambio, los mensajes son similares a las transacciones, diferenciándose en que los mensajes son generados por otros contratos y no por una cuenta externa. Al igual que las transacciones, los mensajes pueden enviarse una cuenta externa, a otro contrato para ejecutar sus funciones o para generar un nuevo contrato.

2.2.4. Contratos inteligentes sobre Ethereum

Aunque el concepto de contrato inteligente fue mencionado teóricamente por Nick Szabo en 1996 [17], no ha sido posible hasta el nacimiento y el desarrollo de la tecnología blockchain. Un contrato inteligente es un programa almacenado en la cadena de bloques, guardando cierta analogía con los contratos tradicionales. Mientras que los contratos tradiciones están sujetos a leyes, los otros son capaces de ejecutarse de manera autónoma, haciendo cumplir el acuerdo entre las partes, sin necesidad de mediadores [18].

Gracias a las propiedades de inmutabilidad y transparencia que tiene su tecnología, los contratos inteligentes son seguros y confiables, sin existir riesgo de manipulación y falsificación.

Volviendo a Ethereum, los contratos inteligentes almacenan en la cadena de bloques el código ejecutable del programa, además de los datos asociados a él y el balance del contrato. Al igual que las cuentas de los usuarios tienen una dirección, los contratos también la tienen, pudiendo ejecutar funciones o transferir fondos.

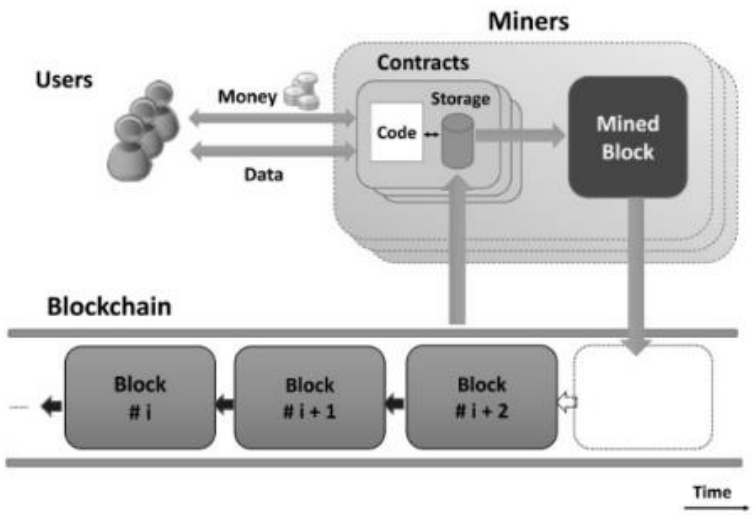

Fig. 2.11. Ejecución de un contrato inteligente en la blockchain [9]

La Fig. 2.11 ilustra la interacción que existe entre las cuentas de los usuarios con el contrato. Un usuario puede realizar una transacción a un contrato para enviar ether o ejecutar una de sus funciones. De la misma manera, un contrato puede enviar dinero desde su propia cuenta balance a otras cuentas o incluso ejecutar funciones de otros contratos.

Cabe destacar, que la modificación de las variables almacenadas es bastante costosa, aproximadamente 20000 unidades de gas por cada 256 bits.

2.3. Solidity

A pesar de que Ethereum soporta diferentes lenguajes para desarrollar contratos inteligentes, el más popular y usado de todos es Solidity. Un lenguaje a alto nivel cuya sintaxis es muy similar a JavaScript. A continuación, se exponen las diferentes características de Solidity.

Antes de todo, para iniciar el flujo en Solidity, se hace con la palabra reservada "*contract"*, similar a una clase, que puede contener funciones, modificadores y diferentes tipos de datos como cualquier lenguaje de programación. Cabe destacar que este lenguaje permite la herencia de contratos [19].

Variables y tipos de datos

Los valores de las variables se encuentran alojados en el almacenamiento del contrato. En Solidity hay que especificar el tipo de dato cuando declaras una variable. Solidity proporciona los siguientes tipos de datos elementales:

- Booleanos: pueden representar dos valores, *true* o *false*.
- Enteros: se pueden diferenciar dos subtipos de enteros, enteros con signo (*int*) o sin signo (*uint*).
- Direcciones: representan una dirección de una cuenta de Ethereum. Posee 20 bytes (*address*).
- Tipos de tamaño fijo: su longitud puede variar desde 1 a 32 bytes (*bytes1, bytes2, ..., bytes32*).
- Arrays de tamaño dinámico: existen tipos de datos que ajustan su tamaño de manera dinámica. S*tring.*

Además, Solidity proporciona algunas funciones y variables especiales que están siempre presentes en el espacio de nombres global, usadas para obtener información acerca del estado del *blockchain* o del contrato.

Visibilidad de las funciones y variables

Como en otros lenguajes de programación, aunque con algunas diferencias, existen cuatro visibilidades diferentes a la hora de definir funciones y variables en Solidity:

- **public:** las funciones públicas pueden ser llamadas de manera interna o a través de mensajes externos. Las variables públicas crean un *getter* automático.
- **private:** las funciones privadas y el estado de las variables son únicamente visibles internamente, es decir, desde el propio contrato y no desde contratos heredados o derivados.
- **external:** este tipo de visibilidad es similar a *public,* diferenciándose en que sus funciones no pueden ser llamadas internamente como "funcion()", sino que deben ser invocadas como "this.funcion()". Las funciones *external* son más eficientes cuando se trabaja con grandes arrays de datos.
- **internal:** las variables y funciones internas se comportan como las privadas, es decir, sólo se pueden llamar internamente, ya sea desde el mismo contrato o desde derivados, sin poder usarse la palabra reservada *this.*

Cabe destacar que, aunque una función o variable sea atribuida como "*private*", esto no significa que no sea visible para el resto de los miembros de la blockchain, sólo quiere decir que no puede ser accesible por otro contrato. Puesto que como se ha dicho anteriormente, todas las transacciones y funcionalidades de un contrato son visibles para todos los participantes de la red blockchain.

Variables y funciones especiales

Existen algunas variables y funciones que están siempre presentes en el espacio de Solidity, es decir, pueden ser usadas en cualquier momento sin necesidad de importar ninguna librería. Se describen algunas de las más usadas a continuación:

- **block.timestamp()** ó **now()**→ Devuelve el número de segundos transcurridos desde el 1 de enero de 1970 del bloque en curso.
- **block.number()** → Devuelve el número actual del bloque
- **msg.sender()** → Devuelve la dirección de la cuenta ejecutora del mensaje.
- **msg.value()** → Devuelve la cantidad de wei que trae el mensaje consigo.
- **gasleft()** → Devuelve el gas restante de la transacción.

- **<address>.transfer(uint cantidad)** → Se envía desde el balance del contrato la cantidad de Wei a la dirección especificada.
- **<address>.send(uint cantidad)** → Otra manera de mandar wei desde el balance del contrato a la dirección especifica. Actualmente en desuso.

Llamadas a funciones

Las funciones en Solidity son las unidades ejecutables de un contrato, que pueden ser ejecutadas de manera externa o interna. Se ejecutan de manera interna cuando se hace la llamada o *call* desde dentro del mismo contrato, por otra parte, se puede ejecutar un método externamente, cuando la llamada se realiza desde una instancia ajena al contrato. Por lo general, para llamar a una función externamente, la máquina virtual de Ethereum (EVM) espera el nombre del método y sus argumentos convertidos mediante la función de hash Keccak-256.

Función de *fallback*

Cuando un contrato es desplegado y compilado en la EVM, las funciones que lo componen son reconocidas mediante su nombre y sus argumentos. Si un método es invocado y no existe coincidencia mediante nombre y/o argumentos, es ejecutada la función de *fallback.*

La función de *fallback* se caracteriza por no tener argumentos y no poder retornar nada. Si un contrato recibe un *message call* sin nombre de función y sin argumentos y, además, la función de *fallback* no está definida, entonces se lanza una excepción. La función de *fallback* no debe contener muchas instrucciones ya que sólo hay disponible 2300 unidades de gas para ejecutar esta función. Hay que asegurarse que esta función no supere el gas establecido, ya que podría lanzar una excepción.

Manejo de excepciones

Las excepciones pueden ser provocadas debido a que el mensaje no contiene el nombre y/o argumentos correctos, que la llamada a la función se quede sin gas que consumir o simplemente que una función la lance intencionadamente. Cabe destacar, que las excepciones finalizan la ejecución en curso y revierten todos los cambios.

En este momento, capturar las excepciones en Solidity no es posible, aunque no existen demasiados problemas reconocidos. Un inconveniente notable que se ha observado hasta la fecha es cuando se llama a un contrato y la llamada se queda sin *GAS*. Otro, es cuando se usa la función *send* (método para enviar Ether) y existe algún fallo, no se revierten los cambios, por ello, actualmente se usa el método *transfer* que soluciona este problema. Estas irregularidades pueden afectar a la seguridad de los contratos.

Modificadores

Las funciones modificadoras tienen la capacidad de cambiar el comportamiento de una función, es decir, si una función tiene un modificador asociado, se asegura que cumpla unas condiciones específicas antes de ser ejecutada y se evita que se llegue a consumir GAS.

El modificador *payable*, ya implementado en Solidity, especifica que la función va a recibir ether. Si se envía ether a una función que carece de este modificador, el método lanzará una excepción.

Eventos

Otro concepto para destacar en Solidity son los eventos, interfaces capaces de llevar a cabo un registro del contrato, es decir, aplicaciones externas a éste, usan los eventos para monitorizar el estado de un contrato, sin tener que interactuar directamente con él.

Por lo general, los eventos suelen ser usados por la interfaz de usuario, es decir, la parte del *front-end* del contrato, reflejando y actuando ante una modificación del éste.

3. DISEÑO E IMPLEMENTACIÓN

Este capítulo describe el diseño y la implementación del contrato inteligente, la aplicación que interactúa con el usuario y el dispositivo IoT.

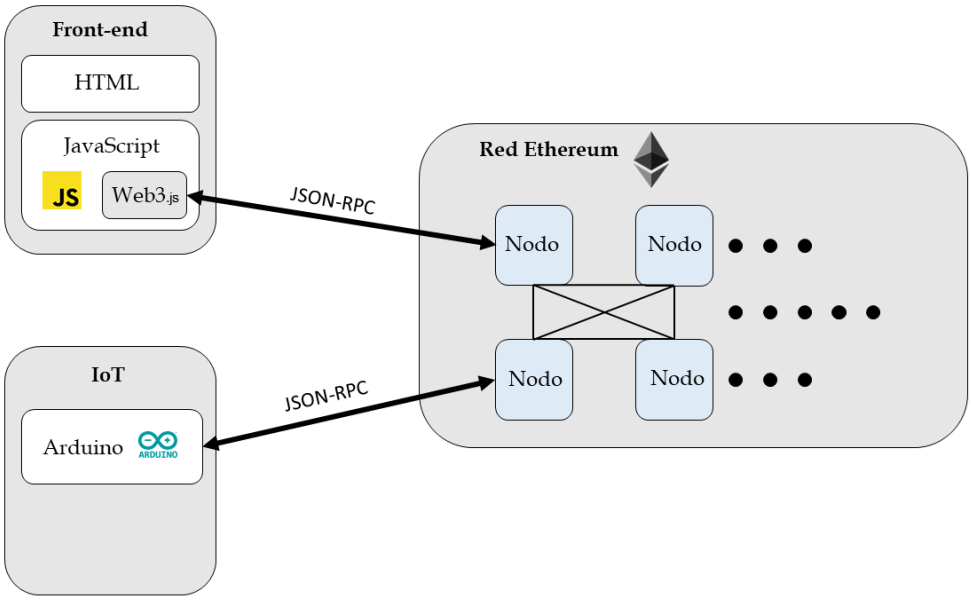

Fig. 3.1. Visión general de la arquitectura

La Fig. 3.1, muestra un resumen general del funcionamiento de la arquitectura del sistema. La aplicación usa un contrato inteligente desplegado en la red blockchain de Ethereum, donde se almacenan los datos pertenecientes al alquiler y gestiona el intercambio con el usuario. En la sección [3.1], se expone el desarrollo del contrato inteligente donde se describen variables, estructura de almacenamiento usada y funciones de interacción. Además, en el apartado [3.2] se presenta la interfaz de usuario y la manera de comunicarse con la blockchain mediante su librería. En [3.3] se explica la implementación del dispositivo IoT y cómo se transmiten las peticiones con los nodos. Para finalizar, en la sección [3.4] se expone el despliegue del contrato, creando los nodos para participar en la red y el entorno de desarrollo usado.

3.1. El contrato inteligente

Uno de los diseños iniciales, fue el de hacer un despliegue nuevo cada vez que se alquila una vivienda, es decir, generar un nuevo contrato y dirección cada vez que se arrienda una nueva vivienda. Finalmente, el contrato del alquiler está diseñado para que una vez desplegado, únicamente existan interacciones sobre el mismo contrato, es decir, todos los usuarios se comuniquen con una misma dirección de contrato, pudiendo modificar el estado de éste. Un despliegue nuevo por contrato se descartó ya que los despliegues implican un mayor coste monetario.

Básicamente, la estructura del contrato es la siguiente:

- Estructura que modela una única vivienda [3.1.1]
- Estructura de datos dónde se almacenan las viviendas [3.1.2].
- Funciones para realizar los cambios de estado del contrato [3.1.3].

3.1.1. Estructura de la vivienda

En primer lugar, se ha definido el objeto que conforma una única vivienda, para ello se ha hecho uso de las estructuras (*structs*) de Solidity. A continuación, se exponen en la siguiente tabla los tipos de datos empleados y un breve resumen.

TABLA 3.1. ESTRUCTURA QUE MODELA LA VIVIENDA

struct vivienda		
TIPOS DE DATO	**NOMBRE**	**DESCRIPCIÓN**
address	inquilino	Dirección pública del inquilino.
address	propietario	Dirección pública del propietario.
string	direccion	Dirección física de la propiedad
bool	alquilada	Estado actual de la vivienda.
uint	precioSeg	Precio por segundo
uint	tiempoInicio	Marca de tiempo inicial.
uint	tiempoFinal	Marca de tiempo final.
uint	tiempoMaximoAlq	Tiempo máximo de alquiler

Análogamente a los contratos tradicionales de alquileres de viviendas, se tiene un propietario y un inquilino, que en este caso es la dirección pública de la cuenta de ambos. Además, las propiedades tienen una situación geográfica, representada mediante una cadena de texto. Asimismo, para comprobar si la vivienda se encuentra alquilada, se realiza con un tipo de dato lógico que marque verdadero o falso en cada caso. Y, por último, los contratos de alquiler tienen un precio, una duración y una fecha de inicio y fin.

3.1.2. Estructura de datos

Una vez definido el objeto que modela la vivienda, se tiene la necesidad de almacenarlas según se vayan creando. A pesar de que Solidity tiene varias estructuras de datos, se ha hecho uso de la estructura de datos *mappings*, que asocia claves con diferentes valores, es decir, puede ser visto como una tabla hash donde se permite acceder directamente a los objetos o valores mediante una clave.

```
mapping ( TipoDatoClave => TipoDatoObjeto) nombreMapping
```

En este caso, la clave es un tipo de dato *uint*, que representa el identificador único que tiene cada vivienda que se añade al sistema. En cambio, el tipo de dato del objeto es el *struct* que define la vivienda.

```
mapping (uint => vivienda) viviendas;
```

3.1.3. Funciones del contrato

Conocido el objeto que define la vivienda y la estructura de datos para almacenarlas, se explican las funciones utilizadas en el contrato para realizar cambios en su estado y/o hacer consultas desde el dispositivo IoT e interfaz. En la siguiente tabla se muestran las cinco funciones desarrolladas, así como sus argumentos de entrada y salida:

TABLA 3.2.. FUNCIONES DEL CONTRATO

NOMBRE	DATOS ENTRADA	DATOS SALIDA
nuevaVivienda ()	- s*tring* direccion - *uint* precio - *uint* tiempoMax	*uint* viviendaID
alquilar ()	- *uint* viviendaID	___
cerrar ()	___	*bool*
estaAlquilada ()	- *uint* viviendaID	*bool*
getVivienda ()	- *uint* viviendaID	-*string* direccion -*bool* alquilada -*uint* precio -*uint* tiempoMax

Como podemos ver en la tabla, el contrato está compuesto por cinco funciones. En primer lugar, tenemos la función para añadir la nueva vivienda, que como argumentos de entrada tiene la dirección donde se encuentra, el precio por segundo y el tiempo máximo que puede estar alquilada. Esta función, añade a la estructura de datos una nueva vivienda con los datos proporcionados. Como el "*struct vivienda*" posee más variables que no se pasan como parámetro, se considera que la vivienda no se encuentra alquilada, no tiene inquilino y que el ejecutor de la función es el propietario de éste.

Otra función para destacar es la encargada de alquilar la propiedad, que recibe el identificador de la vivienda como argumento. Esta función, implementa el modificador ***payable***, permitiendo agregar ether al contrato. Accede a la estructura y modifica la dirección del inquilino, cambia el estado del alquiler a *true* y marca el tiempo de inicio mediante el *timestamp* del bloque con ***now()*.** También se agrega la condición de que una única cuenta pueda alquilar la vivienda, es decir, una cuenta no puede arrendar más de una propiedad.

Siguiendo la tabla, está la función encargada de cerrar la vivienda, que simplemente busca entre todas las viviendas, alguna en la que el inquilino coincida con la cuenta ejecutora de la función (***msg.sender***). Una vez encontrada, se calcula el tiempo transcurrido desde que se inicia el alquiler hasta que se cierra, se reparte la parte proporcional de ether y se

cambia el estado lógico del alquiler. De la misma manera, un propietario puede cerrar su vivienda si un inquilino ha sobrepasado el límite de tiempo.

Las dos últimas funciones, no tienen coste de gas, ya que únicamente consultan el contrato sin modificar las variables. La primera, "estaAlquilada(uint viviendaID)", es usada por el dispositivo IoT, para preguntar al contrato en qué estado se encuentra la propiedad, es decir, si esta alquilada en ese momento o no. La segunda, es utilizada por la interfaz de usuario, que mediante el identificador de la vivienda, obtiene la cadena de texto de la dirección, el precio, el tiempo máximo establecido y si ésta se encuentra alquilada.

Para resumir, se tiene el *struct vivienda* donde se modela un único alojamiento con las variables definidas en [3.1.1], y cada uno de estas, se almacena en una estructura de datos (*mapping*). El contrato está compuesto de diferentes funciones, capaces de crear nuevas viviendas, añadirlas a la estructura de datos y acceder a ellas para cambiar su estado. Alguno de los métodos definidos, no cambian el estado, sino que simplemente hacen una copia en el nodo local y devuelven un valor, este tipo de funciones no tienen ningún coste de gas.

3.2. Interfaz de usuario

El medio usado para que el usuario pueda comunicarse con el contrato, se ha llevado a cabo mediante *HTML* para crear la estructura básica de la página web, *CSS* para los estilos y la librería *Web3.js* de *JavaScript* para las comunicaciones entre la interfaz de usuario y la *blockchain.*

3.2.1. Librería Web3.js

Esta librería de JavaScript es compatible con Ethereum y lleva a cabo todas las especificaciones del sistema RPC e IPC, permitiendo realizar acciones como enviar ether, leer y/o escribir datos de los contratos.

Esta librería, contiene varios módulos con algunas características particulares, es decir, que tienen funcionalidades diferentes. Son usados dos de ellos [20]:

- web3-eth: paquete que permite interactuar con la blockchain de Ethereum y con los contratos inteligentes.

- web3-utils: paquete que contiene funciones y métodos para facilitar el trabajo al desarrollador.

Para leer datos de los contratos, se crea una instancia de Web3 y se indica a que proveedor debe conectarse, es decir, a que nodo tiene que hablar para llevar a cabo las lecturas y escrituras.

```
web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"))
```

También hay que tener una representación del contrato Ethereum mediante la función *web3.eth.contract(),* esta función espera dos argumentos: el ABI del contrato y la dirección de éste. El ABI es la representación en formato JSON de la estructura del contrato. Por tanto, quedaría de la siguiente manera:

```
var contrato = new web3.eth.contract(abi , direccionContrato)
```

De esta manera, ya se puede empezar a leer y escribir los datos del contrato llamando a sus funciones: directamente ejecutando "*contrato.miFuncion()"*. Si el método lleva argumentos o ether, quedaría de la siguiente manera:

```
contrato.miFuncion(arg1, arg2,...,{value: 1000})
```

En esta sección, quedan reflejadas las funciones básicas de la librería de Web3.js, aunque toda la documentación oficial se encuentra en [20].

3.2.2. Descripción

Aunque la interfaz de usuario no es el motor de la aplicación, si es un punto para destacar, ya que con ella se puede interactuar con los contratos alojados en la blockchain de forma sencilla e intuitiva, es decir, como si se tratase de una web convencional.

Como se ha mencionado, para llevar a cabo el desarrollo de la interfaz, se ha empleado HTML y la librería de JavaScript Web3.js que ofrece Ethereum.

Para que el consumidor de la aplicación disfrute de mayor usabilidad y experiencia, se ha decidido que sólo exista una única ventana principal apoyada de una ventana flotante. En la Fig. 3.2, se muestra el resultado de la interfaz, en ella se han colocado referencias numéricas para mostrar las diferentes oportunidades de interacción que posee el usuario.

Fig. 3.2. Interfaz de usuario – Pantalla Principal

Siguiendo el punto número uno, obtenemos un desplegable con las diferentes cuentas que se tienen alojados en el cliente Ethereum [3.4.4], pudiendo elegir la cuenta deseada para realizar la transacción. Se obtiene mediante el uso de una función "*web3.eth.accounts*" alojada en la librería Web3.js.

El número dos se muestra a través de una ventana emergente [Fig. 3.3], permitiendo que un propietario pueda añadir una nueva vivienda a la red de Ethereum. Se facilita un formulario que incluye: la dirección donde se encuentra la vivienda, el precio (mWei) por segundo y el número máximo de días/horas que desea tener alquilada su vivienda. Para ello, se obtienen los datos del formulario mediante JQuery y se ejecuta la función correspondiente utilizando la librería como se ha visto en [3.2.1].

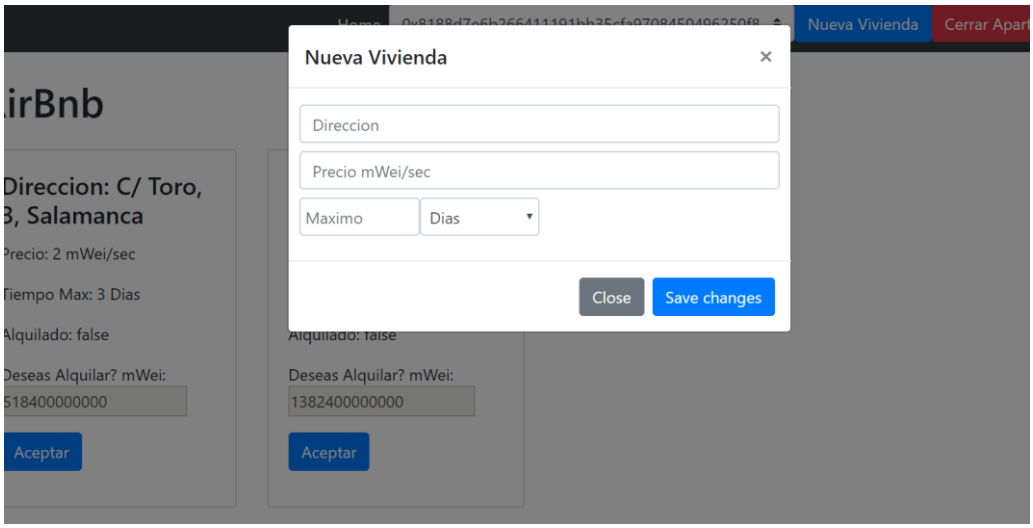

Fig. 3.3. Interfaz de usuario – Ventana Emergente

Volviendo a la Fig. 3.2, con el número tres, tenemos el botón para cerrar la vivienda que se encuentra habilitado independientemente si se tiene alguna propiedad en alquiler. Por tanto, es el contrato el encargado de verificar si se tiene alguna vivienda arrendada, y en caso afirmativo, que se lleve a cabo el cierre de ésta.

Cuando una nueva vivienda es añadida al sistema, se van generando las diferentes propiedades en la ventana. Además, por cada vivienda, se genera un botón de "aceptar" para que el inquilino dé el consentimiento del alquiler con el ether descrito. Una vez formalizado, será el dispositivo IoT el encargado de la apertura.

Para resumir, estos cuatro puntos son las posibles interacciones que tiene el usuario para alquilar o añadir viviendas al sistema a través de la interfaz, empleando las funciones del contrato descritas en [3.1] con ayuda de la librería Web3.js.

3.3. Dispositivo IoT

Para llevar a cabo la simulación, se ha empleado el microcontrolador ESP8266 de MCU. Este micro destaca por su bajo coste con módulo de WIFI integrado. Su pila de TCP/IP y su desarrollo en Arduino facilita el trabajo para realizar consultas al nodo de Ethereum. Para evitar costes de comprar en un primer momento una cerradura electrónica, se han empleado los leds integrados en la placa, para poder simular que la vivienda se encuentra alquilada y abierta. En desarrollos futuros se podría hacer uso de una cerradura conectada al mismo módulo.

Fig. 3.4. Microcontrolador ESP8266. Fuente: Amazon

Aunque el ESP8266 puede ser usado como cliente o servidor, en este caso será usado como cliente. La librería WIFI de Arduino que éste implementa, facilita la conexión con el router y las peticiones de HTTP (GET y POST).

Este dispositivo llevará a cabo peticiones POST a la función *estaAlquilada(uint)* del contrato [3.1]. Esta función recibe como parámetro el identificador único que posee cada vivienda y devuelve true o false dependiendo del estado del alquiler. La consulta se hace en intervalos de tiempo especificados por el desarrollador, en este caso son 10 segundos, es decir, se pregunta cada cierto tiempo, si la vivienda con dicho identificador se encuentra alquilada.

En primer lugar, se importan las librerías del ESP8266:

```
#include < ESP8266HTTPClient.h >
#include < ESP8266WiFi.h >
```

La librería facilita la conexión con el router con una única sentencia, quedando de la siguiente manera:

```
WiFi.begin("ESSID", "PASSWORD");
```

Una vez establecida la conexión, se crea un objeto de la clase *HTTPClient*, especificando la IP del nodo Ethereum y la cabecera *Content-Type* para especificar el formato que va a ser transmitido por la red. En este caso, el formato es de tipo JSON.

```
http.begin("http://IP:8545") //IP donde se encuentra ejecutando el nodo y puerto por defecto
http.addHeader("Content-Type", "application/json") //Especifica la cabecera Content-Type
```

Siguiendo la API de JSON-RPC [20], para ejecutar una llamada a una función del contrato (*eth_call*), se hace una petición POST en formato JSON, mediante la siguiente estructura:

```
{"jsonrpc":"2.0", //versión de JSON-RCP
"method":"eth_call", //Especifica la llamada a una función del contrato
"params": [{"to": "0xdireccionContrato", "data": "Datos"}}
```

Para llevar a cabo la petición, es necesario especificar la dirección del contrato mediante el campo ***to***.

El valor ***data*** especifica la función del contrato que se quiere ejecutar, para ello se pone como ejemplo que se quiere preguntar si la vivienda con identificador 10, se encuentra

alquilada. Es decir, se quiere conseguir llamar "*estaAlquilada(10)*", para ello se hace de la siguiente manera:

- 4 primeros bytes de aplicar la función hash keccak-256 de la forma ASCII del nombre de la función y del tipo de dato de sus argumentos.

  $KECCAK_{256}(estaAlquilada(uint256)) = 0xf12b7459$

- A continuación, el argumento es codificado en hexadecimal, añadiendo tantos ceros hasta completar 32 bytes.

  $HEX(10) =$

  $0x000000000000000000000000000000000000000000000000000000000000000A$

  32 bytes

- Se adhieren ambos resultados para formar un total de 4+32 = 36 bytes. Quedando de la siguiente forma:

```
"data":
"0xf12b7459000000000000000000000000000000000000000000000000000000000000000A"
```

Por último, se ejecuta **http.**POST() poniendo como argumento lo detallado anteriormente. Una vez enviado el POST, el nodo contesta en formato JSON. Por ejemplo:

```
{"jsonrpc": "2.0",
 "id": 1,
 "result": "0x01"}
```

El campo que nos da realmente la información es ***result,*** donde se encuentra el retorno de la función *estaAlquilada()*, codificado en hexadecimal, siendo true = 0x01 y false = 0x00. En este caso, se comprueba si el alojamiento se encuentra alquilado o no, encendiendo el led en caso afirmativo.

Resumiendo, para llevar a cabo la implementación del dispositivo IoT, se ha usado un microcontrolador programable en lenguaje Arduino que destaca por su precio y facilidad de uso. Siguiendo los documentos del micro y de la API de JSON-RPC, se importan las librerías y se hacen peticiones POST a la función del contrato cada cierto intervalo de tiempo, comprobando el estado del alquiler de la vivienda en todo momento.

3.4. Despliegue

3.4.1. Entorno de desarrollo

En primer lugar, es necesario un entorno de desarrollo o *IDE* (*Integrated Development Environment*) que facilita programar los contratos y llevar a cabo el despliegue, poseyendo un conjunto de herramientas de ayuda para los desarrolladores.

El entorno usado en este proyecto es **Remix** [21], un IDE ejecutable desde el navegador web. Remix es un IDE de Solidity para escribir, compilar, depurar y desplegar los contratos. Para acceder, únicamente es necesario acceder a https://remix.ethereum.org. Posee un completo editor de código y varias pestañas donde se puede compilar, desplegar en el nodo seleccionado, probar y ejecutar los contratos.

3.4.2. Cliente Ethereum

Para hacer el despliegue del contrato e integrar la interfaz de usuario y el dispositivo IoT con la *blockchain* es necesario un cliente Ethereum. TestRPC o Geth son los clientes más populares, ejecutados a través de línea de comandos. TestRPC, destaca en las primeras fases del desarrollo, por su rapidez y facilidad de uso. Al ejecutar el comando *testrpc*, se crea un nodo local y diez cuentas monedero con ether de prueba. En cambio, Geth es un cliente más completo con varias funcionalidades, pudiendo ejecutar el nodo en la red principal o en una red de pruebas con características similares a la principal.

Ambos clientes, pueden comunicarse desde la misma u otra máquina, usando el protocolo JSON-RPC [22]. Protocolo que lleva a cabo los procedimientos de llamadas a las funciones (RPC – *Remote Procedure Call*) codificado en JSON.

3.4.3. Red de pruebas

Para el desarrollo inicial, se emplea una red local (*testrpc*), que es mucho más rápida que cualquier otro entorno. En cambio, existen *testnets* o redes de pruebas que simulan la red principal de Ethereum.

Las redes de pruebas son utilizadas antes de lanzar los contratos y aplicaciones a la red real, evitando poder perder activos, es decir, dinero. Tienen las mismas propiedades que la red principal:

- Existen mineros para firmar las transacciones.
- Puedes recibir y transferir Ether.
- Hay que pagar un coste de GAS para llevar a cabo las transacciones.

La principal se diferencia respecto a la red de pruebas, en que su moneda tiene valor cero y se puede conseguir ether de manera gratuita.

Existen tres redes de pruebas muy similares al entorno de producción de Ethereum, cuya elección va a gustos del desarrollador.

- Ropsten.
- Kovan.
- Rinkeby.

Siguiendo la documentación de Geth [23], para ejecutar un nodo en la red de pruebas Rinkeby, se especifica la red, se habilita el uso de HTTP-RPC mediante el argumento –rpc y se específica con –rpcaddr la IP donde escucha el servidor. Por defecto, el servidor escucha en el puerto 8545, aunque puede modificarse mediante –rpcport. Queda de la siguiente manera:

```
Geth --rinkeby --rpc --rpcaddr "ip_local"
```

3.4.4. Cuentas de Ethereum

Crear una nueva cuenta de usuario en Ethereum conlleva crear una pareja de claves, una pública y otra privada, almacenado en un fichero [2.2.2]. Con el cliente Go-Ethereum, es posible crear nuevas cuentas con el comando "*Geth account new*" [23]. A continuación, pide una contraseña para salvar la información, generando un fichero con las claves en el directorio del cliente Ethereum. Aunque la cuenta esta creada, hay que desbloquearla para poder realizar transacciones con el comando "*Geth --unlock <dirCuenta>"*.

Cabe destacar, que para que el nodo se comunique con el dispositivo IoT no es necesario la creación de cuenta alguna, pues únicamente lleva a cabo consultas del estado del contrato.

4. ANÁLISIS Y CONCLUSIONES

4.1. Valoración

En el presente trabajo, se ha desarrollado una aplicación descentralizada utilizando los contratos inteligentes de la plataforma Ethereum, basada en la tecnología blockchain. En concreto, se ha desarrollado una aplicación descentralizada enfocada en el alquiler de viviendas, donde los participantes de la red puedan añadir viviendas, ordenar el alquiler de alguna de ellas o confirmar su cierre. Además de la interfaz de usuario necesaria para realizar la firma de alquileres de manera visual, se ha implementado un dispositivo IoT conectado a la cadena de bloques simulando la cerradura electrónica de la puerta de la vivienda. También, la aplicación es totalmente escalable, pudiendo manejar gran cantidad de viviendas y detectar errores, evitando que alguna de las partes pierda dinero. Por tanto, de esta manera, quedarían cubiertos todos los objetivos iniciales del proyecto.

A parte de la aplicación propiamente dicha, hay que destacar los conocimientos técnicos adquiridos de la tecnología blockchain, pieza fundamental de este trabajo. Actualmente, esta tecnología se encuentra en auge, debido a las características de inmutabilidad y transparencia que posee. Cada vez más compañías están poniendo el foco en la tecnología, necesitando gente con conocimientos del entorno.

Aunque se haya finalizado el desarrollo e implementación de la aplicación, en las secciones siguientes se analizan los costes de despliegue y los asociados a la interacción con el contrato. Se describen los puntos críticos existentes en cuanto a seguridad y privacidad, y sus posibles soluciones. También, se exponen las leyes aplicables al presente trabajo, que todo proyecto debe cumplir. Para finalizar, se detallan las líneas de trabajo futuras, limitando los puntos donde se debe incidir para un progreso del proyecto.

4.2. Costes

Como se ha mencionado en capítulos anteriores, la interacción con la cadena de bloques de Ethereum tiene un coste asociado, directamente proporcional al trabajo computacional empleado. Por tanto, la creación de contratos Ethereum y la interacción con éstos, tiene un consumo de gas que debe ser pagado por el emisor de la transacción [2.2.3]. Para tener

una aproximación de costes más reales, similares al de la red principal, las medidas se han realizado en la red de pruebas Rinkeby.

Simplemente, para calcular el coste en euros, el gas usado es multiplicado por el precio de Gas/Eth y por el tipo de cambio Eth/Eur, como aparece en (4.1).

Aunque el precio de gas es variable según demanda, actualmente el precio medio es de 18 gigaWei (gWei) [24], por tanto, se asume este valor. Actualmente, el valor de cambio de ether en euros es de 133€. El gas consumido en cada transacción es visible a través de la página web EtherScan.

$$C_{eur} = C_{gas} * P_{gas} * \ P_{divisa} \tag{4.1}$$

En la TABLA 4.1, se muestra el gas empleado en el despliegue y su coste en euros.

TABLA 4.1. COSTE EMPLEADO EN EL DESPLIEGUE

	GAS EMPLEADO	**COSTE (€)**
DESPLIEGUE	1020702	2.44

Por tanto, el despliegue tendría un coste único de 2.44€, esta cantidad únicamente habría que abonarlo la primera vez que se lanza el contrato en la *blockchain.*

En cuanto a la ejecución de las funciones del contrato [3.1], los costes medidos asociados, son los detallados en la TABLA 4.2:

TABLA 4.2. COSTE EMPLEADO EN LAS TRANSACCIONES DE FUNCIONES FUNDAMENTALES

	GAS EMPLEADO	**COSTE (€)**
nuevaVivienda()	188778	0,44€
alquilar()	131421	0,31€
cerrar()	41071	0,10€

4.3. Seguridad

En esta sección, se analizan los aspectos de seguridad más importantes, los cuales incluyen la seguridad de las cuentas, evitando transacciones indeseadas que no han sido ejecutadas voluntariamente por el usuario.

En primer lugar, el nodo que está habilitado para el dispositivo IoT, no tiene ninguna cuenta-monedero habilitada, ya que el dispositivo únicamente necesita hacer consultas sobre el estado del contrato cada cierto intervalo de tiempo. Por tanto, un posible atacante podría conocer sólo el estado del alquiler, es decir, si la vivienda se encuentra alquilada o no, por lo cual, no existen problemas de seguridad para el nodo conectado con el dispositivo IoT.

En cambio, el cliente Ethereum que realiza las transacciones de alquiler, sí tiene cuentas-monedero asociadas. Al ejecutar el nodo, puede ocurrir que algún atacante "proponga" aceptar solicitudes en la máquina local. Si se confirman, cualquier proceso puede conectarse con el nodo y hacer un uso fraudulento del cliente y de las cuentas. Para poder amortiguar el problema, se puede llevar a cabo [25]:

- Restringir el tráfico del router a la IP local y el puerto específico (por defecto 8545) dónde se ejecute el nodo.
- Elegir una contraseña segura para las cuentas.
- Activar un proxy inverso.
- Configurar un firewall.

4.4. Privacidad

En este apartado, se explican los problemas de privacidad que aparecen por almacenar los detalles del contrato en red blockchain. Como se sabe, las transacciones en la blockchain no son confidenciales, ya que todo el mundo puede acceder y ver toda la información que se envía y se almacena en un contrato.

En cuanto a las transacciones relativas al alquiler, quedan totalmente visibles las direcciones del propietario e inquilino. Además, es accesible la dirección de la vivienda, su precio y si se encuentra arrendada.

Aunque una de las ventajas de la tecnología blockchain es la transparencia, en algunos casos sí puede ser un problema. Toda transacción es visible por toda la comunidad y por ahora no tiene solución alguna. Sin embargo, en este proyecto, se considera que la información visible, no es confidencial para ninguna de las partes.

4.5. Líneas futuras

Como en todo proyecto, hay que acotar y ceñirse a los objetivos iniciales para obtener el máximo rendimiento y claridad del trabajo. En contraposición, se producen nuevas ideas generadas una vez finalizado el trabajo. A continuación, se exponen algunas líneas que pueden ser de provecho para mejorar el trabajo expuesto:

- En este proyecto, las viviendas únicamente poseen, mediante una cadena de texto, la dirección donde se encuentra la misma. Por tanto, se podrían añadir imágenes para que posibles inquilinos, puedan ver el interior de la vivienda. Como el almacenamiento en la blockchain de Ethereum es caro, aproximadamente 2,5€ el KB. En la comunidad de Ethereum, se utiliza un sistema descentralizado de transferencia de archivos como IPFS [26] o Swarm [27].
- Actualmente, es el propietario el que tiene que cerrar la vivienda si el inquilino sobrepasa el tiempo máximo. Por tanto, otra posible mejora de la especificación, si se sobrepasa el tiempo establecido, la vivienda se cierre automáticamente.
- En vez de ejecutar el nodo del dispositivo IoT en un ordenador, lo cual es innecesario, se puede emplear una Raspberry. De esta manera, se ahorran costes de material y de consumo.
- Para este proyecto, se han utilizado los leds integrados del microcontrolador ESP8266, para realizar la simulación de la apertura de la puerta. A partir de ahora, se puede hacer uso de una cerradura electrónica, conectada al microcontrolador.
- Actualmente, no se pueden eliminar las viviendas que se encuentran en el sistema, por tanto, se puede añadir una función que realice esta especificación.

4.6. Marco regulador

Las criptomonedas son monedas virtuales usadas para el intercambio de capital o servicios sin necesidad de la intervención de ningún organismo ni entidad financiera, además, las transacciones son totalmente anónimas sin saber a qué usuario con nombre y apellidos corresponde la transacción. A causa de esto, los gobiernos y entidades internacionales se han puesto en alerta con este novedoso sistema debido a varios factores: especulación, estafas y blanqueo de capital. Aunque, por el momento, en España no existe regulación alguna.

4.6.1. Leyes aplicables

Aunque en las leyes españolas no hay ninguna regulación existente para los contratos inteligentes y/o criptomonedas, en este proyecto pueden aplicarse algunas leyes fundamentales del territorio nacional:

- LEY 7/1998, de 13 de abril, sobre las condiciones generales de contratación [28].
- REAL DECRETO 1/2007, de 16 de noviembre, por el que se aprueba el texto refundido de la Ley General para la defensa de los consumidores y usuarios y otras leyes complementarias [29].
- LEY ORGANICA 15/1999, de 13 de diciembre, de protección de datos de carácter personal [30].
- LEY 29/1994, de 24 de noviembre, de arrendamientos urbanos. [31]

4.7. Propiedad intelectual

La propiedad intelectual se define como los bienes intangibles o físicos cuyos derechos pertenecen al autor que ha realizado el trabajo. En lo que a desarrollo software se refiere, existe una regulación para obtener la autoría de los derechos. En España se tramita a través de la Oficina Española de Patentes y Marcas (OEPM), perteneciente al Ministerio de Industria, Comercio y Turismo. Aunque las patentes en estos casos no son fáciles, en caso de ser aceptada la solicitud, el autor se reserva el derecho exclusivo de la idea durante un tiempo. La última legislación vigente en España se recoge bajo la Ley 24/2015 [32].

ANEXO A – ENTORNO SOCIOECONÓMICO

A.1. Impacto socioeconómico

Actualmente, vivimos en una época de era digital, donde la innovación supone una revolución para todos, además, el mundo necesita producir, gestionar y almacenar una enorme cantidad de información certificada en todo momento, que hasta ahora han hecho humanos. Mediante el avance tecnológico, hemos cambiado las maneras de hacer las cosas, que hasta este momento era una rutina. Aunque en este proyecto se ha realizado un contrato inteligente de alquiler, se pueden desarrollar otros contratos. Por tanto, en este apartado, se explica cómo influye la sustitución de los contratos tradicionales por los contratos inteligentes.

Antes de todo, para construir un contrato inteligente, hay que plantearse si las cláusulas de un contrato convencional se pueden transcribir a código, si es así, el contrato inteligente se puede poner en circulación en la red de Ethereum.

Los contratos de arrendamiento, como se ha desarrollado en este proyecto, evitan que inquilino y/o propietario modifiquen el contrato, evitando conflictos, y como posible consecuencia, problemas judiciales. Siguiendo la misma línea, tenemos los contratos de compraventa, que ahorran costes notariales en la transferencia de la propiedad, verificando mediante firma digital que la propiedad cambie de propietario.

Otra posible aplicación, es evitar falsear documentos como pueden ser los diplomas o certificados universitarios. Algunas instituciones ya lo están haciendo para evitar certificaciones fingidas [33]. Esto se posible gracias a una de las características fundamentales de la tecnología, la inmutabilidad, permitiendo que no se puedan falsear datos.

Básicamente, el contrato de papel del que venimos pasa por un proceso en el que intervienen terceras personas, como pueden ser abogados o notarios. Además, actualmente es común que algunas personas modifiquen documentos y/o contratos para beneficio propio. Por ello, cabe destacar que, aunque la tecnología aún es muy reciente, gracias a las propiedades de inmutabilidad y transparencia que ofrece la red, los contratos inteligentes sean una revolución en un futuro muy próximo.

A.2. Planificación del proyecto

En este apartado, se adjunta la planificación seguida para la elaboración del trabajo [TABLA A.1], detallando las tareas y la duración de éstas, en semanas. También, se añade el diagrama de Gantt [Fig. A.1], donde se puede ver como alguna de las actividades son realizadas en paralelo.

TABLA A.1. PLANIFICACIÓN DEL PROYECTO

PLANIFICACIÓN DEL PROYECTO	
Estudio teórico sobre redes Blockchain	1 semana
Estudio teórico sobre Ethereum y contratos inteligentes	1 semana
Objetivo del proyecto	1 semana
Estado del arte	2 semanas
Análisis de soluciones al problema propuesto	1 semana
Diseño del contrato inteligente	5 semanas
Diseño de la interfaz de usuario	2 semanas
Diseño del dispositivo IoT	3 semanas
Despliegue del sistema en la red	3 semanas
Prueba de la aplicación	5 semanas
Evaluación de los resultados	5 semanas
Realización de la memoria	7 semanas
TOTAL DURACIÓN	
	36 semanas

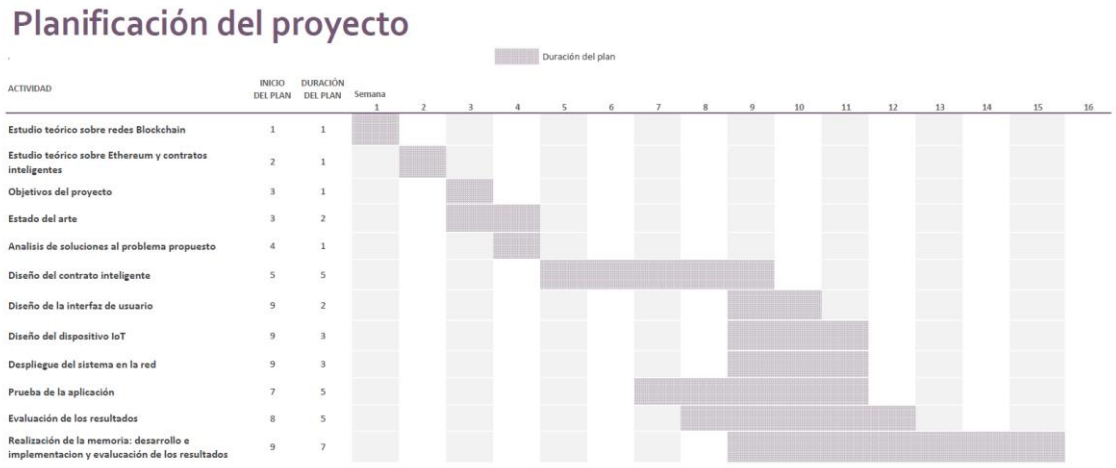

Fig. A.1. Diagrama de Gantt

En el diagrama de Gantt, se exponen los plazos de los procesos mediante barras horizontales. Se visualiza la duración y lo plazos de inicio y fin de cada una de las actividades. Aunque la duración total es de 36 semanas, como se muestra en la tabla anterior, [TABLA A.1], algunas actividades pueden realizarse de manera simultánea. Por ello, el tiempo final de realización es de 15 semanas.

A.3. Presupuesto del proyecto

Con el fin de estimar el entorno económico que impacta sobre el proyecto, se hace un análisis de los costes totales que afectan al proyecto, teniendo los costes de personal, los importes del material y equipamiento, y, por último, los costes del software.

En primer lugar, se evalúan los costes del alumno, donde se incluyen las horas invertidas y el coste por hora de un recién graduado. Se estima que el alumno invierte 400 horas a la elaboración del trabajo con un coste de 6€ la hora. Además, se estima el tiempo invertido por parte del tutor, con varios años de experiencia en el sector, tiene una duración aproximada de 50 horas, y se le atribuye un salario de 23€ por hora.

Por otra parte, se tiene el material utilizado para el desarrollo de la aplicación, detallado en la **¡Error! No se encuentra el origen de la referencia.**TABLA A.2. Por último, se añaden los costes de equipamiento y de las licencias de los programas empleados.

TABLA A.2. PRESUPUESTO TOTAL

PRESUPUESTO DEL PROYECTO	
Personal	
Alumno	2.400 €
Tutor	1.150 €
Total de personal	3.550 €
Material	
ESP8266 NodeMCU	12 €
Total de material	12 €
Equipamiento y licencias	
Ordenador	1.100 €
Windows 10 Home	145 €
Microsoft Office - Licencia Universidad	0,00 €
Arduino IDE	0,00 €
Remix IDE	0,00 €
Total de equipamiento y licencias	1.245,00 €
TOTAL DEL PRESUPUESTO	**4.807 €**

ANEXO B – RESUMEN DE CONTENIDOS EN INGLES

B.1. Abstract

So far, it is part of a concept of centralized systems, where webs are hosted on servers, which control and provide services to users. From the birth of Bitcoin, thanks to the blockchain technology, several decentralized systems have been developed, without the intervention of third parties, giving place in the year 2014 to Ethereum.

Ethereum is a cryptocurrency like Bitcoin, differing in that apart from making transactions, it is designed to store and execute code, birthing smart contracts. These new contracts are agreements between two or more parties, running independently in an uncontrolled system, gaining time and avoiding costs of the intervention of third parties.

In this work has been made use of blockchain technology to deploy an intelligent contract developed in Solidity language, managing the rental of homes, can add and rent properties. A Web application is used to manipulate and facilitate the use of such a contract. In addition, a IoT device is implemented to simulate door opening. This agreement ensures that the rent is effective, saving time and preventing tenants and landlords from being in the firm.

With all this, the existing deployment and transaction costs are assessed on the Ethereum network. In addition, the problems of security and privacy and their possible solutions are valued, determining the viability of the project.

Keywords: Blockchain; Ethereum; Smart Contracts, Solidity

B.2. STATE OF ART

B.2.1. Blockchain

Nowadays, services with a centralised structure are employed keeping data in a unique location known as server. The structure distributed, which characterises blockchain, has advantages respect to another conventional. In blockchain, all information is retorted in every node which sets up blockchain avoiding losing information. Recorded and authenticated data cannot change. Since technology was born, transactions preserve in a historical. Due to this fact, a unique node is required to conserve data on the net avoiding the loose of information. To sum up, all nodes have a copy of information preventing they can be falsified, modified or removed. Information is always verified through common consensus between the members of the net. The consensus is based on hash functions ensuring a single version of data exists. As a result, the concept of decentralization is introduced basing on confidence of P2P connections.

Most of the participants of blockchain are not connected with the net to employ its services, but to create new blocks. Throughout the formation of transactions, that information must be collected in new blocks. For that, the existence of mining nodes is required so that they take part in the approval of transactions. Miners compete on the solve of a mathematical problem which requires much power of computation.

In Bitcoin and Ethereum, the struggle is called proof-of-work. This agreement demands the effort of miner's machines to deal with the problem. As a consequence, when miners resolve the issue, they receive compensation. Therefore, when a mathematical puzzle is solved by a miner, it has to be verified by the rest of community. If it is right, the new block is added to the chain.

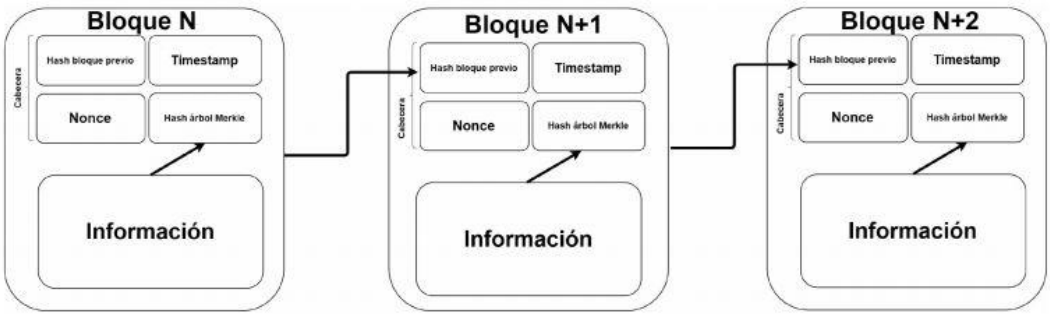

Fig. B.1. Simplified block structure

Each block includes the hash function from the previous one, so the chain is unified consecutively. Following Fig. B.1, in order to put in a new block (n+3), the result of the hash function from the previous block which introduces into the head of the new block. Next, all transactions are thrown in constituting Merkle tree and a parameter is found by gross power (nonce) which produces hash on the new block, the outcome begins with as zeros as community has established.

As it is said before, every block is included in hash function of the previous one being connected between them. For that, it is important knowing how hash function acts: having an entrance of bits sequence which reacts itself. For that, it is important to know how hash function acts: having an entrance of bits sequence which reacts hash function itself, the bits sequence is gained in the release. In spite of quantity of bytes in the entrance, the same number of bits are obtained in the dismissal. If there is any change of bytes in the entrance, the outcome will be different. The hash function is characterised by being a dismissal, an entrance cannot be restored.

On the other hand, in order to collect all transactions in a block, a structure called tree of Merkel is employed. It is a data structure which is based on combinations of hash functions. This algorithm allows transactions are not modified although the block has already been constructed. Furthermore, in bitcoin hash function is SHA256 for its blocks and transactions.

B.2.2. Ethereum

Firstly, Ethereum is a platform which uses the blockchain net as basis technology. In this blockchain, not only accumulates the balance state of accounts, but also the code. This permits to execute the code in a decentralized platform provoking development of smart contracts. Ethereum aims at developing DApps or decentralized applications. As the same as Bitcoin, Ethereum is not controlled by anyone although it is carried out in order to be adjustable.

The Ethereum focus is the virtual machine which arranges the code of any complexity. Moreover, it is the system which allows to be connected with all nodes and it acts as platform motor. EVM executes Smart contracts which is executable code running in the blockchain. A Smart contract is composed by economic balance, a storage and an executable code with functions to change its state. Ethereum database is updated by

numerous nodes connected in the same net that they are running EVM. Ethereum is a significant computer where each node runs EVM keeping the same consensus.

Transactions are block of data which collects a message sending from external account. Transactions contain the following fields:

- Sender
- Sign of sender identification
- Field of optional data.
- STARTGAS, quantity of gas that the issuer gets to pay.
- GASPRICE, quantity of Ether that the sender gets to pay for an unit of gas. The gas, which is employed for paying rates of transaction in Ethereum. The gas shows the cost of an operation in Ethereum.

The quantity of gas can be defined by the creator of the transaction and miners decide if they include into blocks or not. In addition, STARTGAS and GASPRICE fields are important to avoid endless loops. On the other hand, messages are like transactions distinguishing in messages are generated by other contracts and not by an external account. As the same as transactions, messages can send from external account, to other contract so as to execute its functions or to create a new contract.

B.2.3. Solidity

Although Ethereum supports different languages to develop intelligent contracts, the most popular and used of all is Solidity. A high-level language whose syntax is very similar to JavaScript. The different features of solidity are explained below.

First, to indicate the start of the flow in solidity, is made with the reserved word "*contract"*, similar to a class, which can contain functions, modifiers, data types *struct* or *enum,* like almost any programming language. It is noteworthy that this language allows the inheritance of contracts.

In solidity you have to specify the type of data when you declare a variable. This language has these four elementary types of data:

- Bool
- Integer
- Addresses
- Fixed size types
- Arrays of dynamic size.

Exist some variables and special functions in the space of solidity useful when developing contracts:

- **block.timestamp()** or **now()**→ Returns the number of seconds elapsed since January 1st 1970 of the current block.
- **block.number()** → Returns the current number of the block
- **msg.sender()** → Returns the address of the executing account of the message.
- **msg.value()** → Returns the amount of Wei that brings the message with him.
- **gasleft()** → Returns the remaining gas from the transaction.
- **<address>.transfer(uint amount)** → The amount of Wei is sent from the balance of the contract to the specified address.

B.3 DESIGN AND IMPLEMNENTATION

As an introductory summary, the decentralized application developed uses a smart contract deployed on the blockchain network where the rental data is stored. In addition, a user interface is included to manage properties and an IoT device connected to the network to simulate the opening and closing of houses.

The basic structure of the rental contract is as follows:

- Object that model the house
- Data structure where store housing
- Functions to make contract status changes

For modeling the object of housing are used Solidity structs. The following table displays the data types used and a brief summary.

TABLA B.1. MODELING HOUSING STRUCTURE

Struct housing		
Data types	**Name**	**Description**
address	tenant	Owner's public address.
address	owner	Tenant's public address
string	location	Physical address of the property
bool	rented	Current state of the house.
uint	priceSec	Price per second
uint	startTime	Initial timestamp.
uint	endTime	Final timestamp.
uint	maxTimeRent	Maximum rental time

The data structure used are the so-called mappings of solidity, which are data structure similar to hash tables that associate keys with objects.

mapping (uint = > housing) housing;

To interact with the contract are necessary functions that consult and/or change the status of the contract. The functions used are the following.

TABLA B.2. CONTRACT FUNCTIONS

Name	**Input Data**	**Output data**
newHousing ()	- s*tring* Address - *uint* Price - *uint* maxTimeRent	*uint* housingID
rent ()	- *uint* housingID	___
close ()	___	*bool*
isRented ()	- *uint* housingID	*bool*
getHousing ()	- *uint* housingID	-*string* Address -*bool* Rented -*uint* Price -*uint* maxTimeRent

The function to add housing includes to the structure a new dwelling with the arguments provided. In addition, for the rest of the variables not provided, is considered that housing is not rented, it has not any tenant and the executor of the function is the owner.

The rent function will be responsible for the contract to receive the ether by the tenant. Basically, it accesses to the structure with the identifier provided, modifying the status of the rental and setting an initial timestamp.

To close the house, is sought if sender message is a tenant in any of the housing system, if so, the elapsed time is determined, and the amount not enjoyed is sent to the tenant and the consumed to the owner.

On the other hand, the function of "*isRented(uint id)*", indicates using the identifier provided, if this It is rented or not which is used by the IoT device. By last, the function "*getHousing(uint id)*", returns relevant information about it and is used by the user interface.

Now, the user interface has been developed in HTML with the help of the Web3.js library of JavaScript, it connects to the Ethereum node making transactions with the contract. For it, there is to import the library and create an instance of this, indicating the address IP of the node with which it must talk. In addition, you have to Insert the structure JSON of the contract (ABI) and the address of the contract to interact with. Therefore, following the documentation web3. js, it is still In the following way to run a function.

```
web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"))
Was Contract = new web3.eth.contract(abi , contractAddress)
Contract.MyFunction(arg1, arg2,...,{value: 1000})
```

To simulate the door opening, the ESP8266 microcontroller has been used, which is programmed under the Arduino language and has a WIFI module that allows make HTTP requests (GET And POST).

The device asks about the function *isRented(uint)* of the contract in a certain interval of time. As the node follows the JSON-RPC protocol, it makes the query following the standard making POST request:

```
{"jsonrpc":"2.0",//JSON-RCP version
"method":"eth_call",//Specific the call to a function of the contract
"params": [{"to": "0xcontractAddress", "data": "Data"}}
```

B.4. CONCLUSIONS

In the present work, a decentralized application has been developed using the intelligent contracts of the Ethereum platform, based on the blockchain technology. A decentralized application has been developed focused on the rental of houses, where the participants of the network can add houses, order the rent of some of them or confirm their closure. In addition to the user interface required to make the rental signature visually, a IoT device has been deployed connected to the blockchain simulating the electronic door lock of the dwelling. With all this, all the initial objectives of the project would be covered.

The costs involved in carrying out transactions are always paid by the emitter

TABLA B.3. COST EMPLOYED

	GAS used	**Cost (€)**
Deployment	1020702	2.44
newHousing()	188778	0,44€
Rent()	131421	0,31€
Close()	41071	0,10€

it's important to talk about the safety, the node that uses the IoT device, has no security issues, because this node has no wallet enabled. On the other hand, the node that performed rental transactions have a wallet enabled, so it could happen that some attacker wants to make a use fraudulent.

To avoid this, it can:

- Restrict router traffic from the default port 8545 to the local IP where the node is running.
- Choose a secure password for the accounts.
- Activate an inverse proxy.
- Set up a firewall.

On the other hand, the privacy of the contract is also important, although in blockchain one of the main philosophies is transparency, you have to note that all contract information is visible to all network participants.

BIBLIOGRAFÍA

[1] Equisoft, “La cadena de bloques (blockchain). Una tecnología disruptiva con el poder de revolucionar el sector financiero.” Marzo 2017.

[2] C. Dolader, J. Bel, J. L. Muñoz, “La blockchain : fundamentos, aplicaciones y relación con otras tecnologías disruptivas” *Economía industrial,* nº 405, pp. 33-40, 2017.

[3] D. de Ugarte, *El poder de las redes.* Madrid: Cobre Ediciones [En línea]. Disponible en: https://ciccentro.files.wordpress.com/2013/07/el_poder_de_las_redes-1.pdf. [Último acceso: Diciembre 2018].

[4] “Bitcoin: Qué es, cómo funciona y por qué ha llegado para quedarse” *EL PAÍS FINANCIERO,* 17 Diciembre 2018.

[5] Z. Zheng, S. Xie, H. Dai, X. Chen y H. Wang, “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends” de *2017 IEEE International Congress on Big Data (BigData Congress)*, Honolulu, 2017.

[6] P. Jayachandran, “Blogs: IBM” Mayo 2017. [En línea]. Disponible en: https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-public-and-private-blockchain/. [Último acceso: Diciembre 2018].

[7] Bitfury Group, “Public versus Private Blockchains” 20 Octubre 2015. [En línea]. Disponible en: https://bitfury.com/content/downloads/public-vs-private-pt1-1.pdf. [Último acceso: Diciembre 2018].

[8] R. J. Palomo, “'Blockchain': la descentralización del poder y su aplicación en la defensa” *bie3: Boletín I.E.E.E.*, nº 10, pp. 885-904, 2018.

[9] M. Mukhopadhyay, *Ethereum Smart Contract Development*, Packt Publishing, 2018.

[10] M. Echebarria, “Contratos electronicos autoejecutables (smart contract) y pagos con tecnología blockchain” *Revista de estudios europeos,* nº 70, pp. 69-97, 2017.

[11] J. Aguirre, “Cadena de bloques: potencial aplicación a Historias Clínicas”, Trabajo Final, 2017.

[12] B. Asolo, “Blockchain Soft Fork & Hard Fork Explained”, Septiembre 2017. [En línea]. Disponible en: https://www.mycryptopedia.com/hard-fork-soft-fork-explained/. [Último acceso: Diciembre 2018].

[13] H. Mayer, “ECDSA Security in Bitcoin and Ethereum: a Research Survey”, 2016.

[14] A. Preukschat, *Blockchain: la revolución industrial de internet*, Gestión 2000, 2017.

[15] “Ethereum Homestead Documentation — Ethereum Homestead 0.1 documentation” [En línea]. Disponible en: http://www.ethdocs.org/. [Último acceso: Enero 2019].

[16] M. Alharby y A. van Moorsel, “Blockchain Based Smart Contracts : A Systematic Mapping Study”, Medina, 2017.

[17] N. Szabo, “Smart Contracts: Building Blocks for Digital Markets”, 1996.

[18] “Smart contracts, ¿Qué son, cómo funcionan y qué aportan?”, Agosto 2016. [En línea]. Disponible en: https://academy.bit2me.com/que-son-los-smart-contracts/. [Último acceso: Diciembre 2018].

[19] F. Schüpfer, “Design and Implementation of a Smart Conctract Application”, Master Thesis, Agosto 2017. [En línea]. Disponible en: https://files.ifi.uzh.ch/CSG/staff/Rafati/Florian-Schupfer-MA.pdf.

[20] “Ethereum JavaScript API” [En línea]. Disponible en: https://web3js.readthedocs.io/en/1.0/. [Último acceso: Enero 2019].

[21] “Remix Documentation” [En línea]. Disponible en: https://remix.readthedocs.io/en/latest/. [Último acceso: Enero 2019].

[22] “Go-Ethereum JSON-RPC API” [En línea]. Disponible en: https://github.com/ethereum/wiki/wiki/JSON-RPC. [Último acceso: Enero 2019].

[23] Arash, “Go-Ethereum GETH Documentation”, Diciembre 2018. [En línea]. Disponible en: https://github.com/ethereum/go-ethereum/. [Último acceso: Enero 2019].

[24] “Ethereum (ETH) Blockchain Explorer” [En línea]. Disponible en: https://etherscan.io/.

[25] X. Wang, X. Zha, G. Yu y W. Ni, “Attack and Defence of Ethereum Remote APIs” 2018.

[26] Protocol Labs, “IPFS is the Distributed Web”. [En línea]. Disponible en: https://ipfs.io/. [Último acceso: Enero 2019].

[27] “Swarm Documentation - Swarm 0.3” [En línea]. Disponible en: https://swarm-guide.readthedocs.io/. [Último acceso: Enero 2019].

[28] “Ley 7/1998, de 13 de abril, sobre condiciones generales de la contratación” Mayo 1998.

[29] “Real Decreto Legislativo 1/2007, de 16 de noviembre, por el que se aprueba el texto refundido de la Ley General para la Defensa de los Consumidores y Usuarios y otras leyes complementarias”, Diciembre 2007.

[30] “Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal”, Enero 2000.

[31] “Ley 29/1994, de 24 de noviembre, de Arrendamientos Urbanos”, Enero 1995.

[32] “Ley 24/2015, de 24 de julio, de Patentes”, Abril 2017.

[33] EFE, “Las universidades adoptan la tecnología 'blockchain' para luchar contra las vidas académicas ficticias”, *20 Minutos*, 11 Septiembre 2018.

[34] J. Arias, "Estado del arte de la tecnología Blockchian: ¿Burbuja o Consolidacion?", Trabajo fin de grado. Enero 2018. [En línea]. Disponible en: https://es.slideshare.net/jordicabral/estado-del-arte-de-la-tecnologa-blockchain-burbuja-o-consolidacin.

[35] M. Luis, "Blockchain, los smart contracts y un caso de uso", Trabajo fin de grado, Universidad de La Laguna, España. 2017. [En línea]. Disponible en: https://riull.ull.es/xmlui/handle/915/6250