Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTULO DEL TFG: Un protocolo seguro para el intercambio de datos.

TITULACIÓN: Grau en Enginyeria Telemàtica

AUTOR: Jesús Ligero Martínez

DIRECTOR: Juan Hernández Serrano

FECHA: 5 de septiembre de 2019

Título: Un protocolo seguro para el intercambio de datos.

Autor: Jesús Ligero Martínez

Director: Juan Hernández Serrano

Fecha: 5 de septiembre de 2019

Resumen

El proyecto se basa en un protocolo de intercambio de datos sin necesidad de una previa confianza entre ambos sujetos ni una tercera parte de confianza, dado que se encarga de promover un comercio justo, protegiendo tanto a consumidores como proveedores, basándose en una verificación de los datos aleatoria y en teoría de juegos, para que nadie pueda lucrarse ni hostigar a nadie. Para implementarlo se usará la tecnología Blockchain, más concretamente Ethereum, usando su naturaleza de inmutabilidad, transparencia y ejecución determinista, garantizando a ambas partes un intercambio justo.

Title: A secure payment channel protocol for data exchange

Author: Jesús Ligero Martínez

Directors: Juan Hernández Serrano

Date: 5 de septiembre de 2019

Overview

The project is based on a data exchange protocol without the need for prior trust between both subjects or a trusted third party (TTP), since it promoves fair trades, protecting both consumers and suppliers, based on a random data verification and game theory, so that nobody can profit or harass anyone. To implement it will be used Blockchain technology, more specifically Ethereum, using its nature of immutability, transparency and deterministic execution, guaranteeing both parties a fair exchange.

ÍNDICE

INTRODUCCIÓN

El uso de datos dentro de una empresa se ha convertido en un aspecto decisivo para el éxito empresarial. El uso adecuado de las técnicas de big data resulta de ayuda en prácticamente todos los ámbitos de una empresa, por ejemplo: en identificar nuevas ideas, optimizar los procesos operativos, mejorar la comunicación de la empresa, evaluar, personalizar sus productos adaptándose a los clientes, tomar mejores decisiones más rápidamente.... En este contexto, los ecosistemas han crecido para satisfacer las necesidades de datos de las empresas, dando lugar a diversos actores en el mercado: proveedores de datos, custodios de datos, agregadores de datos y muchos más.

Por lo tanto, en la actualidad las empresas no se limitan únicamente a recopilar y analizar datos, si no que cada vez más confían en los datos de terceros para mejorar su valor comercial.

Esta compra de datos no es trivial, pues presenta muchos desafíos, desde la regulación y privacidad de los datos hasta la dificultad de que el cliente pueda valorar los datos sin que el proveedor se los revele.

La creación de mercados online aborda muchos de estos problemas. Ayuda a unificar intereses entre proveedores y consumidores en una plataforma donde ambas partes pueden conocerse e intercambiar información, resuelve el problema de integración de conectar usuarios y proveedores; sin embargo, todavía quedan algunos inconvenientes. Uno de los problemas más difíciles de solucionar es el de convencer a los consumidores del valor de los datos sin revelarlos, esto se debe a una falta de confianza entre consumidores y proveedores, pues en un mercado online sin terceras partes de confianza puede parecer muy fraudulento. Esto se traduce principalmente como una potente barrera de entrada para los proveedores en el mercado, pues no gozan de la confianza de los clientes, perjudicando la competencia y reduciendo así la utilidad para los consumidores.

El proyecto se centra en un protocolo para dar una solución a este problema sin necesidad de una tercera parte de confianza al mismo tiempo que se preservan las características de seguridad, privacidad y equidad necesarias que debe tener un mercado.

En segundo lugar, el protocolo trabaja en una verificación aleatoria de los datos, junto con la Teoría de Juegos para garantizar que nadie actúe fraudulentamente. Para implementarlo se usará la tecnología Blockchain, más

concretamente Ethereum. Este tipo de tecnologías son ideales cuando se trata de realizar aplicaciones descentralizadas eliminando terceras partes de confianza. Se usarán los Smart Contracts de Ethereum por su naturaleza de inmutabilidad, transparencia y ejecución determinista, para garantizar a ambas partes un intercambio justo.

El trabajo está dividido en los siguientes capítulos:

1. Fundamentos tecnológicos: donde se explican todas las bases tecnológicas con el objetivo de ofrecer claridad al proyecto. para que alguien que no tenga ninguna noción de estos pueda entender el proyecto en su totalidad.
2. Tecnologías utilizadas: donde se detallan aquellas tecnologías específicas que se han usado para la realización del proyecto.
3. Proyecto: Donde se explica en profundidad el proyecto, detallando cómo funciona y las decisiones de diseño. Se propondrán posibles ataques y qué mecanismos tiene para contrarrestarlos. Finalmente se enseñará una aplicación desarrollada implementando este protocolo,
4. Conclusiones: Se explicarán los resultados, líneas futuras y opiniones personales

CAPÍTULO 1. FUNDAMENTOS TECNOLÓGICOS

El proyecto se sustenta sobre ciertas tecnologías algunas de las cuales no son de uso común como Blockchain, Ethereum, Merkle Tree,... Para entender el proyecto es importante comprenderlas, o por lo menos tener una noción de ellas.

1.1 Inicios de Blockchain: Bitcoin

Bitcoin es una conjunto de conceptos y tecnologías que forman la base de un ecosistema de moneda digital. El concepto surgió como una alternativa a las monedas convencionales, con la idea de retirar las terceras partes de confianza o intermediarios. En palabras de Satoshi Nakamoto, creador -o creadores- de Bitcoin: *"El problema raíz con las monedas convencionales es la confianza que se requiere para que funcione. Se debe confiar en el Banco Central para no degradar la moneda, pero la historia de las monedas fiduciarias está llena de violaciones de esa confianza".*
Actualmente las ciudadanos son dependientes, casi exclusivamente, de las instituciones financieras como terceros de confianza. Se basa en un sistema, centralizado e inaccesible al público, donde el ciudadano no tiene otra opción que confiar en estos entes y no tiene control sobre su dinero, únicamente puede pedir que el banco haga transacciones por él, pero todo está basado en una confianza, que como ya se ha mencionado ha sido quebrantada en muchas ocasiones (corralitos, quiebras del banco, corrupciones,…). La idea de Bitcoin es librar a los ciudadanos de los intermediarios, dándoles la soberanía sobre su dinero.
Para ello se crea un sistema descentralizado donde nadie tiene más poder. Básicamente es una base de datos distribuida y pública donde se guardan las cuentas y el balance de los usuarios, un registro de transacciones ordenadas y un sistema para validar las transacciones, para evitar anomalías, por ejemplo, que nadie pueda pagar con un dinero del que no dispone. Las transacciones se agrupan en bloques y estos están enlazados criptográficamente, de aquí surgió el nombre de Blockchain.

Utilizando criptografía asimétrica los usuarios tienen claves públicas y privadas. De las públicas derivan las direcciones donde los usuarios pueden recibir dinero y, las privadas, permiten autorizar una transacción del dinero almacenado en la dirección correspondiente a su clave pública. Esto significa que los usuarios son dueños de sus cuentas y por ende de su dinero. Esto tiene beneficios pues no depende de una tercera parte de confianza (TTP) como un banco, y solo el usuario tiene el poder de usar su dinero, sin embargo

conlleva más responsabilidades, pues si el usuario pierde su clave privada pierde su dinero también.

A diferencia de las monedas tradicionales, los bitcoins son completamente virtuales. No hay monedas físicas, ni oro en el que se sustente su valor, así que el valor se basa únicamente en la confianza en la tecnología y en mera especulación.

Pese a que la idea surge para crear simplemente una moneda virtual, la revolución está en la tecnología en sí, extrapolable a muchos otros campos y con un amplísimo potencial.

1.1.1. Blockchain

Blockchain se puede definir como una máquina de estados, replicada en todos los participantes, donde se guardan las cuentas y cierta información de ellas, como el balance, y todas las transiciones de ésta que son las transacciones.

El término blockchain representa una combinación de componentes y tecnologías. Sin embargo, hoy en día hay una gran variedad de blockchains con diferentes propiedades. Por ello se intentará definir los componentes básicos de las blockchains públicas y no permisionadas, que son aquellas con las que se desarrolla este trabajo. Dichos componentes se alistan a continuación:

- Una red P2P que propaga transacciones y bloques de transacciones verificadas, basada en un protocolo estandarizado de “gossip”.
- Transacciones que representan las transiciones de estado.
- Un conjunto de reglas de consenso que rige lo que constituye una transacción y lo que hace que sea válida.
- Una máquina de estado que procesa las transacciones de acuerdo con las reglas del consenso.
- Una cadena de bloques, criptográficamente asegurados, que actúa como un registro de todas las transiciones de estado verificadas y aceptadas.
- Un algoritmo de consenso que descentraliza el control sobre la cadena de bloques, que incentiva a los participantes a cooperar en la aplicación de las reglas de consenso.
- Un esquema de incentivos, basado en teoría de juegos, que asegura el funcionamiento de la máquina de estado en un entorno abierto.

Como ya se ha mencionado, estos son los componentes básicos de una blockchain pública y abierta, hay otros tipos de blockchains enfocadas con otros propósitos, pero tanto Bitcoin como Ethereum son públicas y no permisionadas.

La mayoría de blockchains privadas y/o permisionadas tienen unos componentes bastante similares, pero sobre todo varían en el consenso (decidir que es válido y que no y sobre todo quién puede validar las transacciones) y en el esquema de incentivos pues al ser privada, no hace falta incentivar a los participantes a que mantengan la red, los creadores de ésta ya se ocuparan de mantenerla.

1.1.2. Ethereum

Ethereum es una implementación de la tecnología de Blockchain, que tiene las características anteriormente descritas y se describe a menudo como "computadora mundial".

Desde una perspectiva informática, Ethereum es una máquina de estados determinista e ilimitada, que consta de un estado único, globalmente accesible, y una máquina virtual que aplica cambios a ese estado. Cada transacción contiene la lógica para hacer una transición del estado, y el estado global cambia al validar un nuevo bloque, compuesto de diversas transacciones. Desde una perspectiva más práctica, Ethereum es una infraestructura informática descentralizada de código abierto que ejecuta programas llamados smart contracts. Utiliza una cadena de bloques para sincronizar y almacenar los cambios de estado del sistema, junto con una criptomoneda llamada **ether** para medir y restringir los costos de recursos de ejecución. La plataforma permite crear aplicaciones descentralizadas muy potentes con funciones económicas integradas. La blockchain proporciona a las aplicaciones ciertos beneficios, tales como:

- Disponibilidad: siempre está hábil pues siempre hay gente manteniendo la red y en caso de que se caigan algunos nodos, la información está replicada en el resto.
- Transparencia: pues todo el mundo puede ver lo que hay en la blockchain.

- Neutralidad: como todo el mundo participa en la red, nadie sale favorecido.

1.1.3. Ethereum vs Bitcoin

Ethereum comparte muchos elementos con otras cadenas de bloques públicas como Bitcoin: una red P2P que conecta a los participantes, un algoritmo de consenso bizantino, PoW (Proof of Work), el uso de primitivas criptográficas como la las firmas digitales o hashes y una moneda digital, el ether.
Sin embargo en muchos ámbitos es muy diferente. Tanto el propósito como la construcción de Ethereum difiere de las blockchains públicas que lo precedieron, como Bitcoin.

El propósito de Ethereum no es principalmente ser una red de pago de moneda digital como Bitcoin. Es cierto que Ethereum tiene su propìa moneda virtual, el ether, y ésta es vital para su funcionamiento,pero no está destinada únicamente a hacer pagos sino que es un recurso para pagar el uso de la “computadora mundial” de Ethereum.

Bitcoin tiene un lenguaje de script muy limitado, restringido simplemente a evaluaciones de verdadero/falso sobre las condiciones de gasto. En contraste, Ethereum está especialmente diseñado (femenino) para ser una blockchain programable, que ejecuta una máquina virtual capaz de ejecutar códigos de una gran complejidad.

Obviamente hay límites computacionales, no puede ser que una computadora mundial ejecutase un simple bucle infinito, para ello existe el **gas.** El **gas** es la cuantificación de recursos que requiere ejecutar el código de una transacción. Hay un límite de gas en una transacción lo que significa que hay un límite en los recursos que se pueden consumir. Además el usuario que ha hecho la transaccion debera pagar con ethers el gas consumido, de esta manera paga el uso de la “computadora mundial” .

1.2. Ethereum en profundidad

En este apartado se explican aquellos conceptos de Ethereum indispensables para comprender la tecnología que se ha utilizado en el proyecto. En caso de que se quiera profundizar, se recomienda el libro “*Mastering Ethereum“* como fuente de información.

1.2.1. Unidades monetarias de Ethereum

La moneda de Ethereum es el **ether**, éste se subdivide en unidades más pequeñas, lo que le confiere mayor versatilidad. La unidad subdividida menor es el **wei,** un ether equivale a 10^18 weis.

El valor del ether se representa internamente como un entero sin signo en weis, por ejemplo cuando enviamos 1 ether internamente se codifica 1,000,000,000,000,000,000 como resultado.

Valor (en wei)	Exponente	Nombre común	Nombre en SI
1	1	wei	Wei
1,000	10^3	Babbage	Kilowei o femtoether
1,000,000	10^6	Lovelace	Megawei o picoether
1,000,000,000	10^9	Shannon	Gigawei o nanoether
1,000,000,000,000	10^{12}	Szabo	Microether o micro
1,000,000,000,000,000	10^{15}	Finney	Milliether o milli
1,000,000,000,000,000,000	*10^{18}*	*Ether*	*Ether*
1,000,000,000,000,000,000,000	10^{21}	Grand	Kiloether
1,000,000,000,000,000,000,000,000	10^{24}		Megaether

Fig.1.1. Tabla de denominaciones a las unidades de Ethereum

1.2.2. Criptografía en Ethereum

La criptografía es una rama de las matemáticas usada ampliamente en la seguridad informática, resulta esencial para el funcionamiento de Ethereum y, de hecho, para cualquier blockchain.

Como Ethereum es una blockchain pública, no tiene sentido cifrar información, pues todo el mundo debe ser capaz de leerla. En cambio, las herramientas criptográficas tienen otros objetivos, como demostrar un conocimiento sin revelarlo, como en el caso de una firma digital, o dar pruebas fehacientes de la autenticidad de los datos, como el caso de *hashes.*

En un futuro se plantea usar herramientos de criptografía más complejas como pruebas de *“zero proof knowledge”* y cifrado homomórfico, que permitirán registrar transacciones parcialmente encriptadas y, aún así, que los nodos sean capaces de verificar si la transacción ha sido válida. Pese a que aportan grandes ventajas, también supone un gran desafío implementarlas con la tecnología actual.

En este apartado se introducen las herramientas criptográficas claves para entender el funcionamiento de Ethereum, no se analizarán en detalle, en el sentido de que no se mostrarán los cálculos, simplemente se presentan las ideas generales que permitan comprender las herramientas.

1.2.2.1. “Hash”

Se puede definir como una función que asigna datos de tamaño arbitrario a datos de un tamaño fijo. Las funciones de *hash* son esenciales en los sistemas criptográficos actuales y, por supuesto, en Ethereum.

A continuación se detallan sus propiedades:

- **Determinismo:** Dado un input siempre debe producir el mismo output.
- **Verificabilidad :** Calcular el *hash* de un mensaje es eficiente (complejidad lineal). Quizás calcula si el hash… es eficiente?
- **Difusión :** Un pequeño cambio en el input, por ejemplo, un simple bit diferente cambia el output del hash de manera tan extensa que no se puede correlacionar con el hash del mensaje original.
- **Irreversibilidad :** Calcular el input original desde el output es imposible, la única manera es a fuerza bruta *(un interminable ensayo y error),* o en otras palabras, probando al azar inputs.
- **Protección contra colisiones:** Es prácticamente inviable calcular dos inputs que tengan el mismo output.
- **Rango definido:** El input de un hash puede tener cualquier tamaño, pero el output siempre debe tener el mismo.

Estas propiedades hacen al hash idóneo para muchas aplicaciones, una de las más utilizadas es la creación de identificadores únicos. La mejor manera de entender las utilidades de estas funciones es a partir de ejemplos.

Como se ha mencionado anteriormente, los bloques de la Blockchain están enlazados criptográficamente, en realidad, simplemente cada bloque contiene el Hash del anterior. Es una aplicación de la función de Hash muy conveniente ya que obtenemos un identificador único para cada bloque, acotado en tamaño e inviable de “falsificar”, en el sentido de que no se puede crear otro bloque con

el mismo hash como output por la propiedad de protección contra colisiones. Si se pudiese falsificar un bloque cualquiera podría manipular la información de la Blockchain, pues nadie podría saber qué bloque es el “original”.

Entonces, ¿hasta qué punto es inviable encontrar una colisión? El rango de inputs es infinito pues podemos poner cualquier tamaño, pero el output siempre tiene el mismo tamaño, eso significa que si pueden existir colisiones, pero ¿qué probabilidad hay de encontrar una?.

La respuesta reside en el tamaño de la salida de la función de Hash utilizada (siempre que esté bien diseñada con las propiedades anteriormente descritas), en Ethereum por ejemplo se utiliza el Keccak-256. Esta función tiene como salida 256 bits siempre, eso significa que hay 2^{256} posibles outputs. Debido al Teorema de Cumpleaños se entiende que una función de hash está rota (99.99% de encontrar una colisión) cuando calculamos $2^{n/2}$ hashes, donde **n** (negrita) es el número de bits del output.

Para poner un ejemplo con números reales, se estima que la red de Bitcoin calcula aproximadamente 300×10^{15} hashes por segundo, por lo tanto:

(1.1.)

$$2^{256/2}\text{hashes}/(300*10^{15}\,\text{hashes/s} * 86400\ \text{s/día} * 365\ \text{días/año}) \approx 3.6\times10^{13}\ \text{años}$$

La red de Bitcoin tardaria 3.6×10^{13} años en romper un algoritmo de hash de 256 bits. Con este ejemplo práctico el lector puede imaginarse que actualmente no se ha encontrado ninguna colisión en los sistemas de hash de 256 bits como sha256 o keccak-256.

Un apunte interesante es que en documentos y código de Ethereum se menciona que usan el algoritmo de hash SHA-3 en vez del keccak-256, ésto puede llevar a confusión. Si es cierto que el keccak-256 es el SHA-3, pero no se trata de la última versión aceptada de este algoritmo y ambas versiones no obtienen los mismos resultados, en el sentido de que para el mismo input se obtienen outputs diferentes. Por lo tanto es importante verificar qué versión de SHA-3 o keccak-256 se está usando.

1.2.2.2. Criptografía asimétrica

La criptografía asimétrica, es un sistema criptográfico que utiliza un par de claves, una pública la cual puede difundirse y otra privada, que solo debe

conocer el propietario. Se trata de dos números que tienen la siguiente propiedad: lo que se cifra con una clave se descifra con la otra, de ahí viene el nombre de “asimétrica”.

La generación de estas claves se obtiene a partir de funciones matemáticas denominadas “*trapdoor*”, como su nombre indica se trata de funciones fáciles de calcular pero cuyo inverso es de muy difícil cálculo, hasta tal punto que cuando son de cierta envergadura, como en la implementación, de Ethereum se convierte en un problema computacionalmente imposible.

Un ejemplo simple de una función sencilla de “trapdoor” podría ser la multiplicación de dos números primos. Si bien es muy sencillo multiplicar dos números primos como 2459*4261= 10477799 , resulta muy difícil factorizar el número 10477799, pues se deben ir probando uno a uno todos los números primos. Para la generación de las claves, Ethereum usa el problema del logaritmo discreto como “trapdoor”, optimizado utilizando operaciones basadas en curvas elípticas.

Volviendo al tema del apartado, el par de claves es lo que se denomina una cuenta en Ethereum. De la pública se deriva la dirección, y la privada proporciona al usuario el control sobre su cuenta. Se puede hacer una analogía a una cuenta bancaria, donde la clave pública es el número de cuenta bancaria y el PIN es la clave privada. Cualquier usuario puede dar su número de cuenta bancaria para que le ingresen dinero, pero solo él debe saber su PIN y por tanto tener el control de su cuenta.

Con la propiedad de las claves mencionadas se efectúa una firma digital, que no consiste más que en cifrar, normalmente, un hash del mensaje que se quiere firmar (en este caso una transacción), con la clave privada del usuario. Cualquier usuario puede comprobar que la firma es correcta, solo necesita hacer uso de la clave pública para “desencriptar” esta firma y comprobar que corresponde al hash de la transacción. Esta operación no revela en ningún momento la clave privada del usuario y da una prueba fehaciente de que el usuario que envió esa transacción conoce la clave privada.

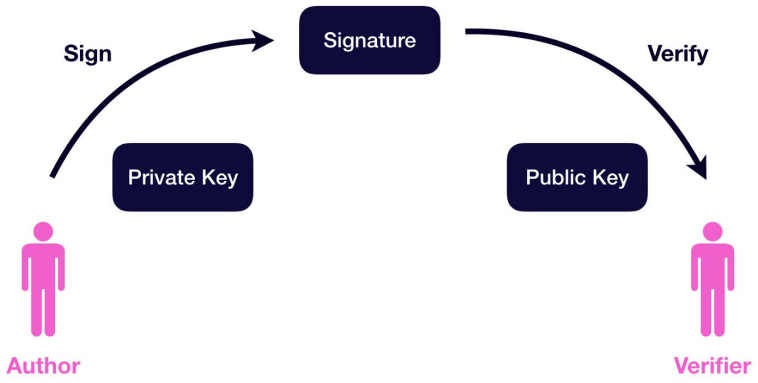

Fig.1.2. Esquema de verificación de una firma digital

1.2.3. Wallets

Se usa el término “Wallet” para referirse a una aplicación software que ayuda a administrar las cuentas de Ethereum. Se encargan tanto de crear y guardar las cuentas como de firmar transacciones con ellas

Existen dos tipos de wallets dependiendo si las claves que crea están relacionadas o no. Si las claves no están relacionadas, se llaman wallets no deterministas. Por otro lado, los Wallets deterministas derivan todas las claves de una “*seed*”. Estas pueden ser generadas siempre que se disponga de la *seed.* Esto los hace más seguros ante accidentes, pues si se pierde o estropea el ordenador, si se guarda la *seed* siempre se podrán recalcular las claves y por tanto el dinero.

La *seed* se trata de un número hexadécimal, por ejemplo: “0xFCCF1AB3329FD5DA3DA9577511F8F137”. Como se acaba de mencionar, es importante guardar la *seed*, pues en caso de accidentes siempre se pueda recuperar las cuentas, pero guardar tales números es muy poco práctico. Por ello el wallet genera el “mnemonic”. Esto es un conjunto de palabras, a partir de las cuales un wallet puede calcular la *seed*, y por tanto nuestras cuentas. Por lo tanto en vez de tener que recordar o escribir un número largo hexadécimal se puede guardar un conjunto de palabras, siendo mucho más fácil de recordar, escribir y menos propenso a errores.

Se ha mencionado escribir pues es recomendable escribir la clave en un papel y no almacenarla en ningún recurso electrónico, para evitar posibles ataques de hackers, pues si consiguen la seed, consiguen el dinero.

Ejemplo de mnemonic: “sunny welcome rainrare fold satisfy vote green grid noodle olive frown captain frog cruel again stomach organ hat town alien evidence glue gorilla”

Dentro de los wallets deterministas, los HD wallets (Hierarchical Deterministic Wallets) son una implementación avanzada y muy utilizada de los wallets deterministas. Se diferencia de su antecesor en que las llaves se derivan en una estructura de árbol, como se ve en la figura

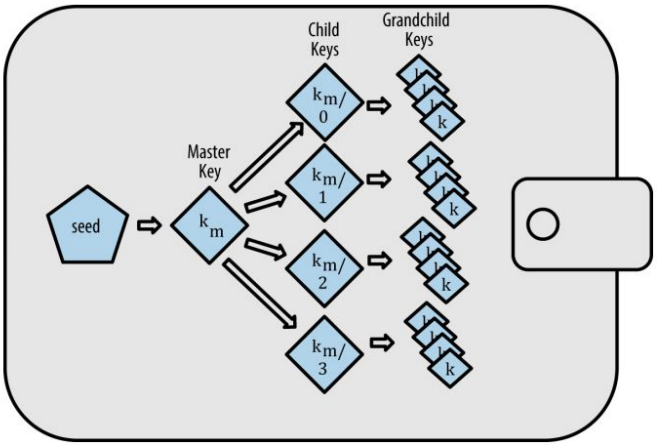

Fig.1.3. Derivación de las claves de un HD wallet

Existe otra distinción entre wallets. Aquellos que requieren conexión a internet para funcionar, llamados hot wallets, y aquellos que no la requieren, llamados cold wallets. En general, los cold wallets se consideran más seguros, pero es más laborioso firmar transacciones.

1.2.4. Transacciones

Las transacciones son mensajes firmados con cierto formato, originados por una cuenta de Ethereum, transmitidos por la red Ethereum y almacenadas en la blockchain. Las transacciones también pueden ser comprendidas como la única cosa que puede producir un cambio del estado, cambiando el balance de las cuentas o ejecutando/creando Smart Contracts. Como se ha mencionado anteriormente Ethereum se puede ver como una máquina de estados y las transacciones son las transiciones de ésta.

Las transacciones están constituidas por:

- **Nonce:** un valor escalar igual al número de transacciones enviadas desde esta dirección.
- **Gas Price** : el precio del gas en weis que el remitente está dispuesto a pagar .
- **Gas Limit:** el máximo gas que el remitente está dispuesto a pagar para la realización de esta transacción.
- **Value: l**a cantidad de ethers que se envía.
- **Data :** Datos binarios de longitud variable, son los datos de la interacción o creación de un Smart Contract. .
- **v,r,s: l**os tres componentes necesarios para la firma digital.

Los campos Value y Data son opcionales, si una transacción no tiene campo de Data es un pago y si no tiene campo de Value es una invocación. Si tiene ambos campos es ambas cosas y si no tiene ninguna es una transacción inútil pero válida al fin y al cabo.

1.2.4.1.¿Por qué el Nonce?

El nonce es una parte fundamental de la transacción, simplemente es un contador del número de transacciones han sido enviadas desde una dirección concreta, si bien es cierto que cada número solo se usa una vez, es posible que un nombre como *counter* sería más esclarecedor.

El nonce es un atributo de la dirección origen; sin embargo, el nonce no se almacena explícitamente como parte del estado de una cuenta, se calcula dinámicamente cada vez, contando el número de transacciones confirmadas que se han originado desde esa dirección.

Hay dos escenarios en los que la existencia del nonce es esencial, se describen a continuación junto a un ejemplo del escenario en cuestión.

- Permite dar una orden de ejecución a las transacciones

Supongamos que un usuario quiere hacer 2 transacciones, una de 6 ethers que es muy importante que se lleve a cabo y otra de 5 ethers. El usuario firma primero la importante y la envía a la blockchain y a continuación hace el mismo proceso con la otra transacción. El usuario no ha tenido en cuenta que solo tiene 8 ethers en su cuenta, así que solo una transacción se podrá llevar a cabo, aún así el usuario puede confiar que como ha enviado primero la importante es la que se llevará a cabo. Sin embargo en un sistema descentralizado los nodos pueden recibir las transacciones en cualquier orden,

no hay ninguna garantía, eso significa que el orden en que se ejecute estas transacciones será completamente aleatorio, si no fuera por el nonce. Los nodos al validar la transacción recuentan dinámicamente el número de transacciones que lleva el usuario, y si no es coherente con el nonce ignoran la transacción, por lo tanto como el usuario ha firmado primero la transacción importante tendrá el nonce idóneo y se procesará primero. En resumen esta característica aporta utilidad a los usuarios, permitiendo dar orden a las transacciones sin tener que esperar que estas se procesen.

- Protege contra transacciones duplicadas (ataques de replay)

Supongamos otro escenario donde un usuario quiere comprar un artículo, para ello hará una transacción con valor de 1 ether a la cuenta correspondiente del vendedor.

Como ya hemos visto antes, el único capaz de autorizar esa transacción es el usuario, pues solo él sabe su clave privada y es capaz de firmar esa transacción y por tanto autorizarla. Si no existiera el nonce, el vendedor del artículo podría simplemente volver a enviar la transacción en la que se le ingresaba dinero. No sabe cual es la clave pero sabe que esa transacción es válida y que le ingresa dinero. Esto se denomina un *ataque de replay*, y se soluciona con el nonce. Como la transacción tiene un nonce todas las transacciones son diferentes y por ende tienen como resultado una firma diferente.

1.2.4.2. El gas en las transacciones

Se ha mencionado anteriormente el gas, como una cuantificación de recursos necesarios para ejecutar la transacción, se puede entender como el “combustible “ de Ethereum. El gas no es directamente ether sino que tiene un valor separado, tasado, eso sí, en ethers. El gas está separado del valor del ether para proteger al sistema de la volatilidad que pueda surgir de los cambios de valor del ether, por ejemplo en caso de que el ether subiese mucho su valor, los costes de hacer transacciones y sobre todo de ejecutar smart contracts subirían linealmente, esto podría ser un problema, haciendo inservible la plataforma para ciertas aplicaciones. Por ello en todas las transacciones hay un campo llamado gasPrice, que permite al remitente de la transacción establecer el precio que está dispuesto a pagar por el gas. El precio se mide en weis por unidad de gas.

El usuario puede poner el parámetro de gasPrice que desee, entonces, ¿qué le impide poner 0 o un valor muy bajo . Para entender mejor esta respuesta se puede encontrar más información en el apartado de Mineria/consenso.

Por ahora, es suficiente con saber que un minero tiene la capacidad de validar un bloque de transacciones a cambio de un incentivo. El punto clave es que el precio que se paga por el gas en la transacción no se quema (expresión que significa que el dinero se pierde para siempre) si no que se lo lleva minero que haya minado esa transacción como recompensa. Por ello los mineros ordenan las transacciones de más a menos gasPrice, dando prioridad a las que ofrecen más incentivos, o en otras palabras, que tienen un mayor gasPrice. Por este motivo el valor del gasPrice está inversamente relacionado con el tiempo que tarda en validarse una transacción, una transacción con un alto gasPrice tardará muy poco en validarse, mientras que si el usuario pone un valor muy bajo ó 0 a su transacción es posible que se descarte o que tarde muchísimo en ser validada. También es cierto que todo depende de la demanda, han habido periodos de muy baja demanda en los que se han validado transacciones gratuitamente, pero actualmente es prácticamente imposible.

A modo de ejemplo se presenta la figura 1.4. de una web que proporciona datos actuales sobre el gas, valores típicos, cálculos estimados, etc…

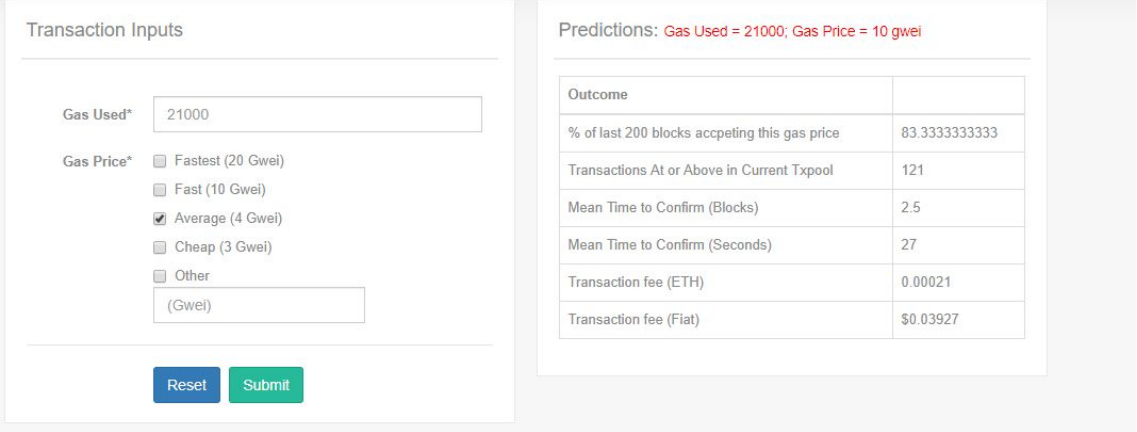

Fig.1.4. Información extraída de: ETH Gas Station [https://ethgasstation.info/] 28/08/2019

Usando la información extraída de cada una de los opciones se ha hecho la siguiente tabla para mostrar los resultados.

Tabla 1.1. Tabla extraída de la información de ETH Gas Station

Inputs		Predicciones	
Precio del gas (Gweis)	Gas utilizado	Tiempo medio para confirmación	Precio de la transacción ($)
0.11	21000	7h 28m15s	0.00043
3(barato)	21000	13m 19s	0.01178
4 (promedio)	21000	5m 58s	0.01571
10(rápido)	21000	27s	0.03927
20(muy rápido)	21000	22s	0.07854

Para su mejor comprensión, es preciso contextualizar algunos datos:

- Gweis significa 10^9 weis, que vendría a ser 1/10^9 ethers, como está expresado en la tabla de “unidades monetarias de Ether”.
- El coste de hacer una transacción simple sin llamar ni crear un Smart Contract es de 21.000 qué es el valor que se ha puesto para ilustrar el ejemplo.
- ETH Gas Station tiene en cuenta las últimas transacciones para estimar estos valores.
- El ether está a 185$.

Tal como se muestra en la tabla 1.1. pagando el valor promedio actual se tardaría unos 6 minutos y el precio es mínimo. Al pagar más del doble el tiempo baja hasta 27s.

Se ha puesto el mínimo de Gweis que permitía la web que són 0.11 para ilustrar y responder la pregunta anterior. Los números hablan por sí solos, se tardarian estimadamente 7h y media para validar esta transacción, por lo que actualmente parece imposible que se ejecuten transacciones gratuitas debido a la demanda.

El segundo parámetro de la transacción relativo al gas es el gas Limit. En éste el usuario indica la máxima cantidad de gas que está dispuesto a pagar. Este campo tiene mucho sentido en diversas ocasiones. Por ejemplo puede ser que la transacción ejecuta un Smart Contract y éste acabe en un bucle infinito, gracias al Gas Limit el usuario puede poner un tope y no pagar más de cierta cantidad.

Aún así, normalmente se puede estimar la cantidad de gas necesaria al llamar o crear un contrato, de hecho el wallet de Metamask estima el coste y lo muestra por pantalla /imagen antes de hacer la transacción, pero no siempre las estimaciones son correctas ya que las llamadas de un contrato pueden

depender del estado de éste, y el estado del contrato puede cambiar por otras transacciones de otros usuarios, y como se ha comentado anteriormente no se puede saber con certeza qué transacciones se ejecutan primero. En resumen, el Gas Limit protege al usuario de pagar en exceso de ether si hay algún problema o imprevisto.

En caso de que una transacción requiera más gas del Gas Limit la transacción se cancelará pero el usuario deberá pagar el gas igualmente, pues el cómputo se ha realizado.

Propagación de las transacciones.

La red Ethereum utiliza un protocolo de "enrutamiento de gossip". Cada cliente Ethereum actúa como un nodo en una red punto a punto (P2P), formando una red mallada. El término "nodo" se usará para referirse a un cliente Ethereum que está conectado y participe en la red P2P.

El proceso empieza con una transacción firmada y válida, luego se transmite a todos los demás nodos de Ethereum que están directamente conectados al nodo de origen. En promedio, cada nodo Ethereum mantiene conexiones con al menos otros 13 nodos, llamados *peers*. Cada nodo vecino valida la transacción tan pronto como la recibe. Si están de acuerdo en que es válida, almacenan una copia y la propagan a todos sus *peers*.

En unos segundos, una transacción de Ethereum se propaga a todos los nodos de Ethereum de todo el mundo. Desde la perspectiva de cada nodo, no es posible discernir el origen de la transacción. El vecino que lo envió al nodo puede ser el originador de la transacción o puede haberlo recibido de uno de sus *peers*. Para poder rastrear los orígenes de las transacciones, o interferir con la propagación, un atacante tendría que controlar un porcentaje significativo de todos los nodos. Esto es parte del diseño de seguridad y privacidad de las redes P2P, especialmente cuando se aplica a las redes blockchain.

1.2.5. Consenso/Minería

En grandes sistemas descentralizados, poner de acuerdo a todos los participantes en un único estado del sistema es una ardua tarea, para ello está el consenso. El consenso trata de regir unas reglas para que el sistema pueda funcionar de manera distribuida y determinista.

Cuando se trata de sistemas descentralizados un nodo no puede simplemente confiar en cualquier transacción que le llegue, debe haber algún mecanismo que valide que la actualización del estado es correcta.

Además, hay otros problemas que se deben solucionar, como: ¿quién pone orden a las transacciones?, ¿qué transacciones van en cada bloque?, y en caso de que sean varios usuarios, ¿por qué estarían dispuestos a validar y ordenar transacciones?. Siendo realistas nadie deja encendido todo el dia su PC, gastando electricidad, para validar transacciones ajenas sin ningún incentivo, es por eso que uno de los elementos clave de las Blockchains es un sistema de incentivos, como ya se ha mencionado anteriormente.

Todos estos problemas son los que intentan resolver los algoritmos de consenso, que se describen a continuación.

1.2.5.1. "Proof of Work" (PoW)

Éste es el algoritmo de consenso más conocido y típico gracias a Bitcoin.

La idea surge de cómo hacer un sistema de "votación" para validar una transacción cuando se trata de un sistema descentralizado. Podría ser un sistema de votación por cuentas, como votación convencional, si la mayoría de personas votan que una transacción es válida lo será; pero dado que las cuentas son gratuitas, alguien podría hacerse 2 billones de cuentas y tener el control sobre la red, ésto es lo que se denomina un ataque de Sybil .

Por este motivo se creó el algoritmo de consenso, cambiando el paradigma de una identidad un voto. La capacidad de voto se distribuirá entre aquellos que "trabajen", aquellos que gasten ciertos recursos computacionales serán los que podrán decidir qué transacciones son válidas, a éstos usuarios se les denominan mineros. A cambio de que mantengan la red con su capacidad de cómputo, requiriendo cierto equipamiento y energía se les otorgan incentivos en forma de criptomonedas.

Pero, ¿qué significa "trabajar" exactamente?, ¿es acaso tan costoso validar una transacción?

La respuesta es no. Comprobar que una transacción sea válida, como por ejemplo, si una cuenta tiene suficientes fondos para hacer una transacción, no cuesta prácticamente recursos, de hecho todos los nodos autentifican que las transacciones que les llegan sean válidas, incluso los que no minan, en caso de que no sean válidas no las propagan por la red.

En este contexto “trabajar” significa resolver un puzzle criptográfico muy costoso computacionalmente. Con tal de poder comprender mejor su funcionamiento, se explicará paso por paso:

1. El nodo se baja la Blockchain para poder empezar a minar.
2. El nodo recibe transacciones y las valida, una vez dispone de suficientes, les ponen un orden según el GasPrice y junto con algunos metadatos como el *Hash* del bloque anterior crea un bloque y empieza a minarlo.
3. Minar el bloque implica resolver un puzzle criptográfico, entonces empieza una carrera entre todos los mineros a ver quien es capaz de minarlo primero.
4. Si consigue minar el bloque primero, enviará a todos su *peers* el bloque validado rápidamente para poder ganar el incentivo. En caso de que reciba un bloque válido ya minado antes de acabar, dejará los cálculos y empezará a minar el siguiente bloque.

Pero, ¿qué significa, exactamente, “minar” un bloque?.

Básicamente, consiste en tomar un bloque creado con las transacciones y los metadatos, añadiendo una transacción especial, donde la cuenta del minero recibe dinero. Esta cantidad será la suma de un **dinero base** más todo el gas que se haya pagado en las transacciones, por este motivo se minan primero las transacciones que tienen un mayor gasPrice pues este parámetro determina el coste de las comisiones que gana el minero.

El dinero base es la moneda que se “crea” en el proceso de minado, esto puede llevar a confusión, pues puede parecer que minar sirva para crear las criptomonedas de una blockchain, sin embargo la verdadera finalidad es validar las transacciones, o en otras palabras, dar un consenso a una red distribuida, y el medio para conseguirlo es crear una moneda para incentivar a los mineros.

Una vez se ha creado el bloque se debe resolver el puzzle criptográfico ya mencionado. Se trata de hacer el *Hash* (concepto ya explicado en previos apartados) del bloque y que el resultado sea más pequeño que cierto número, dando así un intervalo de posibles soluciones. Ya que el Hash utilizado es de 256 bits de salida, se especifica que los primeros 10 bits deben ser 0 por ejemplo, si el Hash cae en el intervalo correcto el puzzle estará resuelto y el minero podrá difundir el bloque para ganar el incentivo.

Pero el hash es una función determinista, si el input no cambia, el resultado tampoco, y el minero no tiene poder sobre que pone en el bloque *(no se entiende)*, ¿de qué forma puede el minero probar alternativas a su bloque si el hash del bloque no está incluído en el intérvalo?.

Por ello hay un campo especial en cada bloque llamado *Nonce.* Este valor es el único que el minero puede y debe variar hasta conseguir el resultado deseado. Como se ha explicado anteriormente el Hash es una función de un solo sentido, no se puede calcular el inverso, la única manera de resolver este puzzle es usando la fuerza bruta, o en otras palabras probar números hasta encontrar uno que valga, por eso se le llama Proof of Work.

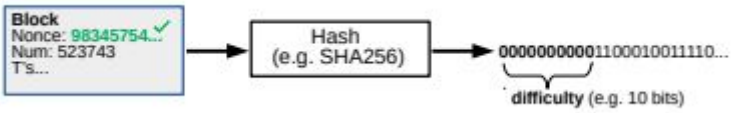

Fig.1.5. Ejemplo de una transacción minada

La dificultad del minado depende de cuán grande sea el intervalo de números posibles, por ejemplo, si aumenta la dificultad, se especificarán más bits a 0 y por tanto habrá un espacio de soluciones mucho más pequeño, siendo más difícil ganar el incentivo. Esta dificultad es un parámetro intrínseco en la blockchain y se ajusta automáticamente, el objetivo es que se mine un bloque cada 20 segundos en Ethereum, si de media se tarda más se reduce la dificultad y viceversa.

El proceso de minar un bloque es costoso, requiere ciertos recursos, lo cual es un incentivo para que los mineros les interese actuar honestamente. Por ejemplo, todos los nodos comprueban que todas las transacciones sean válidas, si un minero mina un bloque con algún error o añadiendo transacciones falsas, el bloque será rechazado por la red y éste perderá su incentivo.

Otro punto interesante es que la red siempre aceptará como buena la cadena de bloques más larga, por este motivo, los mineros siempre minaran sobre la cadena más larga, incluso dejando a medias un bloque que estaban minando, pues en caso contrario estarían compitiendo contra el resto de la red para conseguir la cadena más larga, y no solo la red le lleva un bloque de ventaja en este ejemplo, si no que además tiene mucha más capacidad de cómputo que un simple nodo (imagen?). Por este motivo no vale la pena que los usuarios vayan "en contra" de la red, solo se pierde capacidad de cómputo, mientras que si sigues las "normas" la red te incentiva por ello

Una disputa bastante común es que dos mineros acaben el bloque prácticamente a la vez, en ese caso difundirán su bloque y los nodos aceptarán

como correcto el primer bloque que hayan recibido. Esto desemboca en que aproximadamente la “mitad” de la blockchain continuará desde un bloque y la otra mitad desde el otro. Cada minero minará el bloque que haya recibido antes. Este fenómeno se denomina ***fork*** y ocurre a menudo, dividiendo la Blockchain en dos ramas. Se soluciona en el momento en que un minero de la red mina el siguiente bloque, convirtiendo así su cadena en la más larga y por ende la aceptada por al red, dejando en vía muerta la otra rama. Por ello se recomienda que si se planea efectuar una transacción de gran valor se espere hasta que se hayan minado unos 5 bloques por delante, de esta manera se asegura de que la transacción se ha efectuado y minado correctamente y no se trata de un *fork*.

El ataque más obvio contra este sistema es el denominado *ataque del 51%*. Se podría producir si alguien lograra poseer el 51% o más del poder computacional de la red, ganando un control absoluto sobre ésta. Sería capaz de reescribir la historia, deshacer transacciones o controlar qué transacciones se efectúan, ya que siempre podría ganar la “carrera” de la minera frente al resto de la red, consiguiendo ser la cadena más larga y por ende la aceptada.

Por ello, este consenso asume que nunca nadie pueda llegar a ese 51% de ***hash power***. Si ocurriese esa Blockchain perdería la confianza de los usuarios ya que se convertiría en un sistema centralizado donde un usuario gobierna la red. Si se pierde la confianza de los usuarios en el sistema, se pìerde también el valor de la criptomoneda, dejando así la red obsoleta. Por esa razón a ningún minero le interesa llegar al 51%, si tiene tal equipamiento para llegar a esa capacidad computacional le interesa seguir ganando incentivos de la red y, por tanto, que funcione y los usuarios confíen en esa tecnología.

Pese a que la idea es brillante, también tiene sus fallos. Debido a la fuerte revalorización de las criptomonedas, los incentivos son mucho más valiosos, habiendo más demanda por éstos y por tanto más competencia. Esto se traduce en una fuerte subida de la dificultad del minado, dado que ésta existe para mantener el flujo de bloques por tiempo y si la capacidad de cómputo de la red sube drásticamente debido a la competencia, la dificultad debe subir con ella.

Todos estos factores conllevan que actualmente se necesite un hardware específico para minar, en caso contrario no sale a cuenta el incentivo frente a toda la energía eléctrica empleada para el minado. A su vez significa que hay algunas personas que poseen la mayoría de ***Hash power*** de la red, pues son las propietarias de centrales especializadas para minar. Esto converge en dos problemas: la mayoría del ***Hash power*** está en manos de pocas personas, y recordemos que blockchain existe para ser descentralizada, y el segundo problema es que hay un enorme gasto de energía simplemente para resolver

puzzles criptográficos. Por ello existen otros algoritmos de consenso como el “Proof of Stake”. Actualmente Ethereum utiliza “Proof of Work”, pero en un futuro planean migrar a este otro.

1.2.5.2. Proof of Stake”(PoS)

En general, un algoritmo PoS funciona de la siguiente manera: la cadena de bloques realiza un seguimiento de un conjunto de validadores. Un validador puede ser cualquiera que tenga criptomonedas de la blockchain, como es el caso del ether en Ethereum, mediante el envío de una transacción especial que bloquea su criptomoneda en un depósito. Los validadores se turnan para votar el siguiente bloque válido. El peso del voto del validador depende del tamaño de su depósito, cuanto más ether tiene bloqueado un usuario más valdrá su voto. En este caso el peso del voto no está en el poder computacional si no en la cantidad de criptomoneda que se posea, y se entiende que las personas que disponen más criptomonedas son las más interesadas en que el sistema funcione correctamente, que no se pierda la confianza en esa blockchain y de ese modo ganar más dinero.

Los validadores obtienen una pequeña recompensa, proporcional a la participación depositada, por cada bloque que sea aceptado por la mayoría; pese a ello un validador corre el riesgo de perder su depósito si el bloque al que apostó es rechazado por la mayoría de los validadores. Por lo tanto, PoS obliga a los validadores a actuar honestamente y seguir las reglas de consenso, por un sistema de recompensa y castigo. La principal diferencia entre PoS y PoW es que el el castigo en PoS es intrínseco a la blockchain, les quita ether, mientras que en PoW el castigo es extrínseco, pérdida de fondos gastados en suministro eléctrico.

1.2.5.3. Consenso para Blockchain no Descentralizados

Pese a que Ethereum se creó como una Blockchain completamente descentralizada y pública, también podemos usarla para crear redes privadas y/o permisionadas, simplemente tienen otros fines y básicamente difieren de la red original de Ethereum en el algoritmo de consenso.

Privadas significa que no todo el mundo puede visualizar que hay en la Blockchain, es una red solo para algunos usuarios, por ende es una red privada.

Permisionadas hace referencia a quién puede escribir en la Blockchain, solo podrán aquellas cuentas con permisos privilegiados.

Este tipo de Blockchains no necesitan algoritmos de consenso del tipo PoW o PoS, pues usualmente hay entidades de confianza y no están pensadas para que cualquier usuario pueda intervenir, por ello tienen algoritmos de consenso propios como el Proof of Authority.

1.2.5.4. Proof of authority (PoA)

Este algoritmo es para blockchains permisionadas. Es mucho más simple y menos costoso que los anteriores, se eligen unas autoridades que son las únicas que participan en el consenso, y éstas se turnan para validar los bloques. Los clientes privados de Ethereum acostumbran a usar este consenso, pues no requiere ningún gasto de electricidad adicional como el PoW y es mucho más sencillo.

1.2.6. Clientes de Ethereum

Un cliente o nodo de Ethereum es una software que implementa la tecnología de éste y se comunica a través de la red P2P con otros clientes de Ethereum. Existen varias implementaciones del cliente, hechas por diferentes equipos en lenguajes de programación distintos, pero todos “hablan” el mismo protocolo y siguen las mismas reglas, por lo que pueden interoperar en una misma red de Ethereum. En la red pública se considera una ventaja frente a ataques, ya que en caso de que un cliente sea vulnerable o tenga fallos, mientras se arregla esa implementación, el resto sigue funcionando igual.

A continuación se enumeran las 6 implementaciones principales del cliente de Ethereum, siendo las dos primeras las más comunes:

- Parity, escrito en Rust
- Geth, escrito en Go
- cpp-ethereum, escrito en C++
- pyethereum, escrito en Python
- Mantis, escrito en Scala
- Harmony, escrito en Java

Para cualquier blockchain es vital tener muchos nodos, independientes y dispersos geográficamente, dotando al sistema de las ventajas de ser un

sistema descentralizado extenso, con una gran capacidad de recuperación de la red, alta disponibilidad, protección contra ataques, etc…

Cualquier usuario puede desplegar un nodo de la blockchain en su su casa, independientemente del tipo de cliente que utilice, puede desplegarlo de 3 maneras distintas:

- **Archival node**: Se baja toda la Blockchain, desde el primer bloque de la cadena , también llamado *"génesis block"*, y reproduce todas las transacciones que han habido hasta la fecha, llegando al estado actual. Este tipo de nodo además guarda cada uno de los estados por los que ha pasado la Blockchain, ocupando más de 1.5 Tb de memoria. Por este motivo es un tipo de nodo muy poco utilizado, pues ocupa mucha memoria y no ofrece ventajas frente el full node.
- **Full node:** Este nodo realiza un proceso similar al anterior, a excepción de que no guarda todos los estados que han habido, de hecho los "poda", guardándose uno de cada 1024 estados, por lo que disminuye exponencialmente el tamaño almacenado, llegando a unos 120 Gb.
- **Light node:** Este nodo no vuelve a recalcular toda la Blockchain como hacen los anteriores, de hecho solo se baja los últimos bloques y hace ciertas verificaciones usando un *"Merkle roots"* (concepto que se introducirá en un apartado posterior). Es más inseguro que el Full node, pero su punto fuerte es que puede desplegarse incluso en dispositivos embebidos o móviles, debido a sus bajos requisitos.

Llegados a este punto cabe preguntarse, ¿qué nodo debería desplegar un desarrollador o un usuario?

La respuesta puede ser perfectamente, ninguno. No es necesario desplegar nodos para interactuar con la Blockchain, hay alternativas como clientes remotos o "light clients", que no almacenan nada de la Blockchain y simplemente usan un Full node para resolver las peticiones del usuario, tanto para recoger información como para enviar transacciones.

La diferencia entre los dos tipos de clientes es que los "light clients" efectúan ciertas validaciones con la información recibida de los full nodes usando "Merkle roots" , dando así un nivel de seguridad similar al de desplegar un Full node. Por su parte, los clientes remotos no hacen validaciones y confían plenamente en el cliente al cual están conectados.

Para desarrolladores existen alternativas, como **Ganache**, software utilizado en el proyecto y más profundizado posteriormente. Se trata de un cliente de Ethereum, que no se sincroniza con nada, simplemente sirve como entorno de

pruebas para testar, por ejemplo, despliegues de Smart Contracts antes de usarlos en la red principal.

1.2.7. Smart Contract

El nombre Smart contract a menudo da lugar a confusión, pues ha sido un término que ha ido evolucionando, pero actualmente en Ethereum un Smart Contract no es ni un contrato ni es inteligente.

En Ethereum existen dos tipos de direcciones: aquellas que pertenecen a un usuario, el cual controla con su clave privada, y las direcciones de los Smart Contracts. Se diferencian básicamente en dos aspectos:

1. No tiene una clave privada asociada, por lo tanto nadie tiene el control sobre él.
2. Además de tener un balance, tiene un código asociado. Este código dictará la lógica que sigue el *smart contract* pudiendo llegar a una gran complejidad, desde hacer simples transacciones hasta crear nuevos *smart contract*.

En Ethereum un smart contract se puede entender como un programa informático inmutable que se ejecuta de manera determinista en la blockchain. A continuación se detallan las características de éste:

- **Programa informático:** Los Smart Contracts son, simplemente, programas informáticos. La palabra "contrato" no tiene significado legal en este contexto.
- **Inmutable:** Una vez creado, el código de un *smart contract* no puede cambiar. Si se requiere una actualización o cambio, se debe desplegar un nuevo *smart contract.*
- **Determinista:** El resultado de la ejecución de un *smart contract* es el mismo para todos los nodos que lo ejecutan.
- **Ejecución “síncrona”** La ejecución de cualquier función de un smart contract siempre empieza con una transacción a éste. Nunca empezará a ejecutarse por sí mismo, en el sentido que no se puede programar un smart contract para que ejecute cierta lógica *a*l cabo de 3 días. Si es necesario hacer una operación al cabo de 3 días, un usuario deberá iniciar la ejecución con una transacción.

Para crear un smart contract primero se debe escribir el código definiendo su lógica, en éste puede aparecer funciones y variables que se guardarán en la Blockchain. Después hace falta compilarlo y por último se crea con una

transacción especial que incluye el código compilado. Un smart contract puede crear a otro siempre que disponga del código que necesite compilar.

Haciendo una similitud con la programación orientada a objetos, se puede ver como una clase, la cual tiene sus atributos y sus métodos, al crear el smart contract se “instancia la clase” pudiendo a partir de este momento interactuar con ésta.

Una vez desplegado el contrato en la blockchain cualquier usuario puede interactuar con este. Si únicamente se requiere información de alguna variable y no se pretende ejecutar ningún código entonces se hace una “query”, este tipo de llamadas son gratuitas.

En cambio si se requiere ejecutar alguna función o cambiar alguna variable es necesario hacer una transacción y por tanto pagar por el coste correspondiente.

Los smarts contracts operan en un contexto de ejecución muy limitado. Esto significa que no pueden acceder a cualquier información de la Blockchain, solo pueden acceder a su propio estado, información de la transacción que los invocó y cierta metainformación de los últimos bloques, como por ejemplo el número de bloque actual.

Como ya se ha mencionado los contratos son inmutables, con la única excepción de que tengan una función especial llamada “*self destruct”*, ésta borra todo el código y datos guardados de este smart contract.

Actualmente hay diferentes lenguajes para programar smart contracts, si bien **Solidity** es, prácticamente, el único utilizado y en el que se basará el proyecto que se presenta.

Programar un Smart Contract es más complejo de lo que puede parecer a simple vista. Si bien solidity es un lenguaje de alto nivel hay algunas consideraciones que se deben tener en cuenta.

1. El contrato debe estar muy bien optimizado, pues todo el computo que se optimiza es gas que se ahorra, que a su vez se traduce en dinero. Hay algunas consideraciones que se deben tomar al programar un Smart contract, pues hay operaciones que gastan más gas que otras, por ejemplo guardar algo en el almacenamiento de Ethereum es mucho más costoso que en una memoria temporal fuera del estado de la Blockchain.
2. Es vital que un Smart contract sea muy robusto, esté bien auditado y se haya comprobado exhaustivamente que es resistente a bugs y a ataques pues no deja de ser un programa el cual todo el mundo tiene

acceso, el menos error o bug puede resultar en la pérdida de todos los fondos de este contrato, como ya ha ocurrido muchas veces y con grandes cantidades de dinero.

1.2.8. DAPPs

Un DAPP es una aplicación descentralizada parcial o completamente. Frecuentemente se considera una Dapp un smart contract con un front end en web para interactuar con él. Pero en el sentido más estricto puede tratarse de toda la infraestructura de una aplicación descentralizada por ejemplo, veamos los componentes típicos de una app:

• Software de backend (lógica de la aplicación)

• Software frontend(vistas de la aplicación)

• Bases de datos

• Comunicaciones de mensajes.

• Resolución de nombres

Cada uno de estos componentes puede llegar a estar descentralizado, por ejemplo, el back end puede tratarse de un Smar Contract en Ethereum, el front end puede ser una aplicación descargada que se ejecuta en cada dispositivo, y la base de datos puede usar un almacenamiento basado en P2P. Las Dapp puramente descentralizadas son muy poco comunes actualmente, suponen un gran desafío y son muy difíciles de implementar, pero es la visión de futuro de los fundadores de Ethereum.

Aunque la Dapp no sea completamente descentralizada aporta ventajas que las Apps normales no tienen:

Disponibilidad: Al tratarse de sistemas distribuidos, siempre se podrá acceder a la aplicación, a diferencia de una app convencional, pues los servidores necesitan horas de mantenimiento, desastres naturales...

Transparencia: La naturaleza de una blockchain es que todos puedan inspeccionar su contenido, y por ende el código de la Dapp, dando seguridad a los usuarios, pues saben exactamente la lógica que hay detrás de su aplicación.

Protección contra ataques: Al estar en un sistema distribuido nadie puede cambiar la lógica de la Dapp y nadie puede intervenir para que un usuario interactúe con esta.

1.3. Merkle Tree

Es una herramienta criptográfica cuyo objetivo es autenticar y verificar una estructura de datos. Es una herramienta clave para mi proyecto como ya se analizará en los siguientes apartados. Como su nombre indica tiene estructura de árbol, en el caso del proyecto que se presenta y para ilustrar el ejemplo más sencillo cada rama se bifurca en otras dos, por ello se denomina árbol binario.

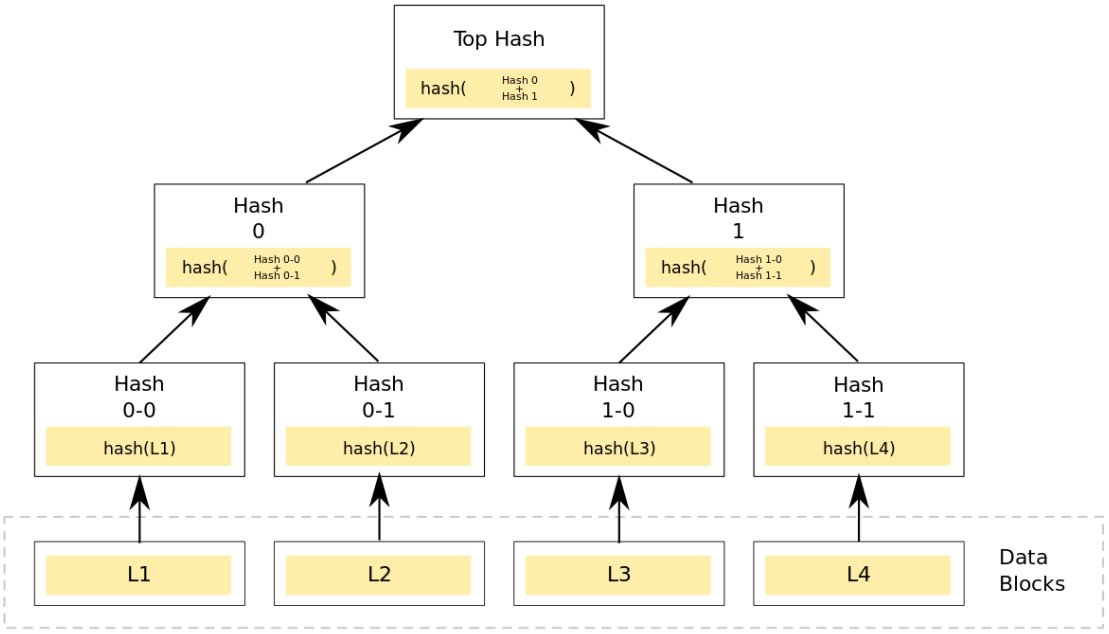

Fig.1.6. Ejemplo de Merkle Tree binario

Para empezar se dispone de los datos que se quiere autenticar, éstos se dividen en bloques. El hash de estos bloques será el primer eslabón del árbol u “hojas” del árbol.

En el siguiente eslabón del árbol se trata de concatenar 2 resultados del anterior eslabón (ya que es un árbol binario), y hacer el hash de ese resultado, dando lugar a un nuevo eslabón, hasta que el árbol confluye en un único hash, llamado “root” como podemos ver en la figura.

Esta estructura de datos es muy útil, pues permite verificar fácilmente cualquier bloque de datos con una mínima información, esta comprobación se denomina *“Merkle proof”* .

A continuación se pondrá un ejemplo práctico:

Volvamos a la Figura 1.6, supongamos que tenemos el Merkle root de los datos y queremos verificar que el bloque L1 forma parte de este conjunto de bloques del Merkle tree. Simplemente lo que necesitamos son los Hashes 0-1 y el 1. Con estos dos hashes podremos hacer todas las comprobaciones necesarias: primero haremos el Hash de nuestra hoja, después lo concatenamos con el hash 0-1 y haremos el hash resultante. Finalmente lo concatenamos con el Hash 1 y comprobaremos que el resultado es igual al Merkle Root.

Otra propiedad muy útil de los Merkle tree es que asegura que todos los datos son correctos, pues si hay un bit diferente en uno de los ficheros o hay una falsificación de un posible atacante el Merkle Root será completamente diferente, pues toda la rama de hashes cambiará drásticamente gracias a las propiedad de difusión del Hash ya mencionada.

CAPÍTULO 2. TECNOLOGÍAS UTILIZADAS

Para la implementación del proyecto se han utilizado las siguientes tecnologías.

React:

Se ha usado ReactJs para implementar el front-end. ReactJs es una librería de JavaScript opensource , desarrollada por Facebook, que se utiliza básicamente para interfaces de usuario tanto de web como de aplicaciones de móvil.

Normalmente se utiliza para hacer SPA(single page applications). Las principales ventajas de React Js frente a otros frameworks es que es escalable, simple y rápido.

Express:

Se ha utilizado Express para implementar el back-end. Express es un framework de Node.js minimalista y flexible para crear aplicaciones web. Se ha utilizado este framework por su simplicidad al trabajar, pues la idea de este proyecto no es un producto final, si nó una prueba de concepto sobre todo centrada en la parte de Blockchain.

Truffle:

Se ha utilizado truffle para compilar y testear el Smart Contract. Este es un framework específico para desarrolladores de ethereum. Sirve para compilar, probar y testear Smart contracts. Incluye herramientas muy útiles, como test automatizados, gestión de redes(para elegir la red Blockchain en que se desplegará el Smart contract), una consola interactiva para comunicarse directamente con el contrato desplegado....

Ganache:

Framework que permite simular una blockchain de ethereum, para uso personal. Además proporciona ciertas herramientas, como un control del minado, un explorador de bloques y genera 10 cuentas de Ethereum y les añade 100 ethers. Herramienta ideal para probar Smart Contracts antes de desplegarlos en redes públicas.

Metamask

Se trata de un HD hot wallet para Ethereum, conceptos que ya se ha introducido previamente. Resumiendo, se trata de un wallet del cual crea todas las claves a partir de un seed, y necesito conexión internet para poder firmar las transacciones. Metamask encripta la seed junto a una contraseña (que se le pide al usuario la primera vez que lo instala). Así si un atacante mira los archivos del ordenador solo encontrada un bloque encriptado y le será imposible descifrar la seed.

Metamask se ha implementado como una extensión del navegador de internet como chrome o mozilla, haciéndolo muy sencillo de instalar y utilizar. Se ha utilizado metamask para poder interactuar con la Blockchain des del front end, el front end se conecta con metamask, y este firma las transacciones que el usuario decida y las envía al nodo. Metamask puede conectarse a diferentes nodos, por defecto utiliza nodos de Infura, pero también puede conectarse a cualquier red que se le configure. Como entorno de pruebas se ha conectado Metamask a la Blockchain simulada con Ganache, desplegado contratos con truffle y efectuando transacciones des del front end usando librerías de Web3js y Metamask para la gestión de cuenta y el envío de las transacciones y des del Back end usando librerías de Web3js para ambas cosas.

Web 3js

Se trata de un conjunto de librerías en javascript que sirven para interactuar con la Ethereum ya sea desde un nodo local o remoto..

CAPÍTULO 3. PROTOCOLO DE INTERCAMBIO DE DATOS

3.1. Intercambio de datos con verificación aleatoria

Lograr un intercambio de productos virtuales justo entre varios participantes minimizando los riesgos es el objetivo principal del comercio virtual. Para lograrlo, es esencial tanto que el consumidor obtenga el producto como que el proveedor sea pagado.

Esto se consigue, muchas veces, sin necesidad de un protocolo estricto, simplemente por la confianza entre ambos. Las contrapartes se conocen y tienen confianza por experiencias anteriores, por futuros intereses o por popularidad en el sector, dando por hecho que ninguna parte planea estafar al otro.

Cuando una mayor seguridad es necesaria, pues por ejemplo no se conocen mutuamente, hay mucho dinero en juego… se hacen necesarios las terceras parte de confianza TTP(trusted third party) en quien ambas partes confían plenamente garantizando un correcto intercambio, un ejemplo cotidiano de esto podría ser Amazon.

Esto resuelve el aspecto de minimizar riesgos pero a cambio se debe pagar un coste adicional a la TTP, haciendo menos óptimo el proceso, más caro y por tanto añadiendo una nueva barrera de entrada para proveedores pequeños. Además en el fondo todo se basa en la confianza en las TTP.

Aquí es donde las tecnologías Blockchain pueden llegar para salvar el dia, pues son ideales cuando se trata de quitar terceras partes de confianza incluso en procedimientos críticos donde hay dinero de por medio.

El uso de la criptografía asimétrica proporciona una prueba fehaciente con cierta facilidad en la verificación al atribuir acciones (pagos) a participantes específicos (propietarios de criptomonedas).

Blockchain proporciona integridad, equidad, disponibilidad y transparencia sin necesidad de una autoridad central (TTP).

El protocolo presentado para el intercambio de datos comparte los objetivos previamente indicados para el comercio online además de proporcionar una solución a la falta de confianza en proveedores de datos, ofreciendo una

muestra aleatoria al consumidor, para que él mismo pueda valorarlos. Los objetivos de este protocolo són:

1. Que el consumidor obtenga una muestra justa de los datos que se comercializan antes de comprometerse. El protocolo debe garantizar que todo los datos son estadísticamente consistentes con esta muestra y que la muestra no está influenciada por el proveedor. Así el consumidor puede evaluar su valor de modo realista.

2. Que se paga al proveedor, si y sólo si, el consumidor tiene acceso a los datos. Por lo tanto que el consumidor no puede obtener los datos sin pagar y el proveedor no puede recibir el pago sin dar los datos.

3. Que la solución sea rentable. O en otras palabras que el coste de ejecutar el protocolo sea mucho menor que el valor de los datos que puede intercambiar. Debido a los altos costes que tienen ciertas operaciones en Blockchain se debe procurar que el coste del protocolo sea prácticamente fijo, independiente de la cantidad de datos intercambiados.

El protocolo ofrece un intercambio de datos anónimo, basándose en una verificación aleatoria de los datos y no en una confianza al proveedor o a una TTP, resolviendo el problema de la confianza y por tanto quitando barreras de entrada.

3.2. Diseño del protocolo de intercambio de datos

En este apartado se definirán los pasos por los que pasa el protocolo, y cada uno de los actores justificando las decisiones de diseño.

Se supondrá una plataforma de marketplace para que ambos puedan ponerse en contacto además de que ésta asegurará que crean los Smart Contracts adecuados, para evitar fraudes o revisiones exhaustivas de éstos, eliminando barreras de entrada.

Para asegurar que los Smart Contracts son exactamente como se han propuesto en el protocolo, el MarketPlace tendrá un Smart Contract, el cual tendrá una función especial que como resultado creará un Smart Contract “hijo” determinado. Esto sirve básicamente para que los proveedores sigan la “plantilla” establecida por el marketplace, pues los Smart Contracts resultantes serán exactamente iguales exceptuando algunos parámetros propios de los proveedores.

Los proveedores podrán realizar una transacción con sus parámetros a esa función para crear su Smart Contract. A este patrón de diseño se le suele llamar “factory”.

Se supondrá que los proveedores quieren realizar la tarea automáticamente, así pues habrá un servidor que gestionará todo el intercambio de datos, haciendo las comprobaciones necesarias y contactando tanto con el consumidor como con la Blockchain.

Paso 1: Registro en el Marketplace del proveedor

Un proveedor posee ciertos datos que desea vender, éste decide ponerse en contacto con el marketplace pues es una manera de ganar visibilidad y la confianza de los consumidores.

Para ello, deberá hacer una transacción al Smart Contract del marketplace añadiendo ciertos parámetros, siendo:

n : el número de muestras que desea vender.

p: el precio de las muestras.

Collateral: Será tanto un parámetro del Smart Contract como el precio que deberá pagar tanto el proveedor como el consumidor y se usará en caso de que alguien actúe fraudulentamente.

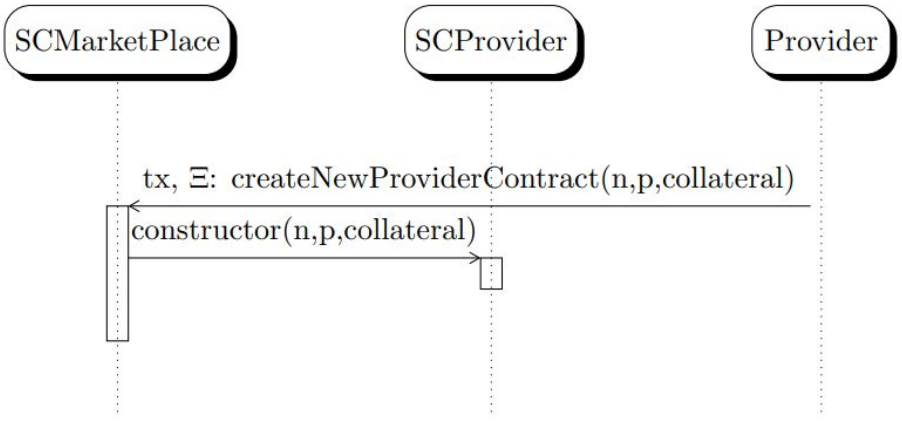

Fig.3.1. Diagrama de flujo del paso 1

Como resultado se creará un nuevo Smart Contract para el proveedor, como se ve en la figura 3.1. Éste es el patrón de diseño de “factory” antes mencionado, y se ha elegido para estandarizar el proceso de intercambio de datos, dando así confianza a los consumidores. El símbolo Ξ indica que la transacción contiene un pago.

Una vez complete este proceso podrá registrarse en el marketplace, poniendo los datos que quiere vender, el precio de estos, alguna descripción, la Url a su servidor para que más adelante el consumidor se pueda poner en contacto con él y la dirección del Smart Contract creado. El marketplace hará una “query” a la Blockchain para comprobar que ese Smart Contract se ha creado del suyo y aceptará el registro del proveedor. Por ese motivo el Smart Contract del marketplace, no solo crea los Smart Contract de los proveedores si no que se guarda una lista con las direcciones de éstos, haciendo muy fácil este último proceso.

Paso 2: Creación del Smart Contract por el consumidor

El siguiente paso es que un consumidor entre en el Marketplace y decida que unos datos pueden ser de su interés. entonces empezará el protocolo de intercambio de datos. En ese momento el consumidor se descargará un javascript del marketplace con la Url del servidor del proveedor, conteniendo toda la lógica necesaria para realizar el protocolo.

Para empezar el consumidor hará una transacción al Smart Contract del Proveedor como se ve en la figura 3.2. creando un nuevo Smart Contract, éste será el que usarán ambos para el intercambio de datos. El Smart Contract del proveedor sigue el mismo patrón de factory que su antecesor, se usa como “plantilla” y el consumidor añade ciertos parámetros Se ha decidido este diseño para que el proveedor solo deba crear una instancia del Smart Contract para todos los clientes que estén interesados en esos datos, pues el Smart Contract es irreutilizable una vez se ha realizado un intercambio de datos con él, más adelante se explicará por qué. Además de una comodidad para el proveedor representa un desincentivo para ataques de replay para el consumidor, como ya se hablará más adelante en apartado de “posibles ataques”.

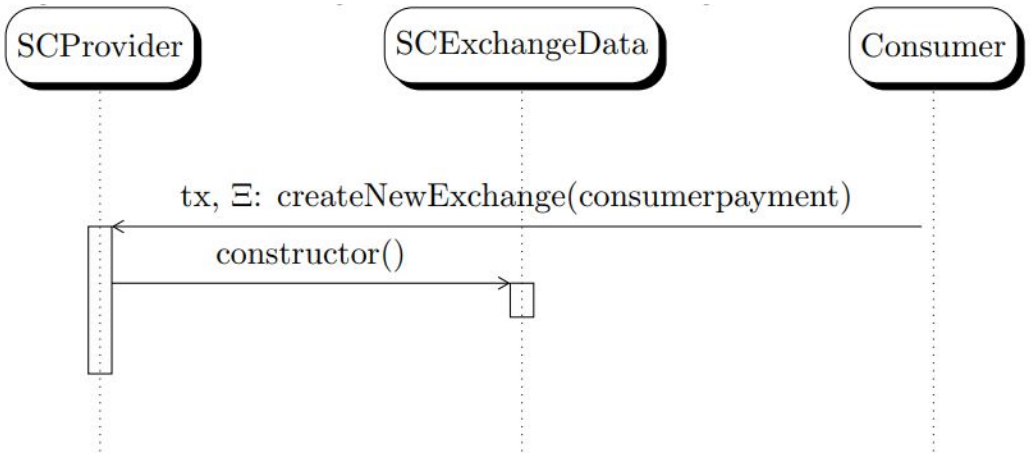

Fig.3.2. Diagrama de flujo del paso 2

En la creación del nuevo Smart Contract el consumidor deberá poner todo el dinero que valen los datos que está comprando, junto a un dinero colateral en caso de que haya problemas/cancelaciones, el mismo que puso el proveedor en el paso anterior.

Por supuesto, este dinero no lo podrá recibir el proveedor hasta finalizar el protocolo y el consumidor podrá recuperarlo prácticamente todo en caso que no quiera continuar con la compra. Se ha decidido de esta manera para desincentivar una vez más ataques de replay por parte del consumidor. Se le reintegrará todo el dinero al consumidor en caso de que el proveedor cancele la venta o no responda en el plazo de una semana. Además el proveedor le pagará un pequeño importe, pues el consumidor se ha gastado algo de ether en la creación del Smart Contract y ha sido “fallo del proveedor”. Ese pequeño importe sale de colateral que ambos han pagado.

En caso de que sea el consumidor quien cancele la venta, se le reintegrará todo el dinero pagado de los datos, pero parte del colateral se le dará al proveedor, dependiendo del estado del Smart Contract (ver diagrama de estados Figura 3.3.) .

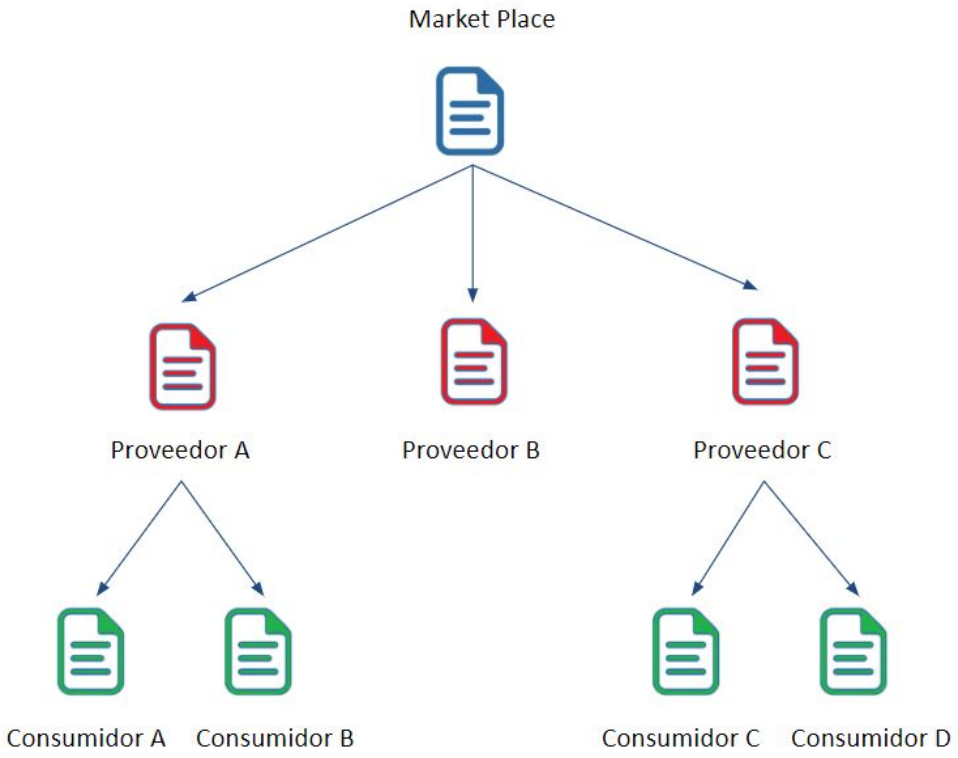

Fig.3.3. Jerarquia de Smart Contracts

Recapitulando, como se puede ver en la Figura 3.3., habrá una jerarquía de 3 niveles de Smart Contracts.

1. El Smart Contract del Marketplace, que básicamente servirán para asegurarse que los proveedores usan su “plantilla” para evitar fraudes y revisiones exhaustivas con tal de comprobar que el contrato es correcto.Es responsabilidad del consumidor comprobar que el Smart Contract es del proveedor, pero es mucho más sencillo ver que éste pertenece al MarketPlace que tener que leer, testear y revisar un Smart contract para comprobar que las funciones son las que dicen ser y no hay trampas.
2. Los Smart Contracts creados por los proveedores, provienen del Smart Contract del MarketPlace y servirán como “plantilla” para los Smart contracts que implementarán el protocolo.
3. Los Smart Contracts creados por los consumidores, estos serán los que implementen el protocolo de intercambio de datos.

Los dos primeros Smart Contracts básicamente son “factorys” y llevan una lista de los Smart Contracts que han creado y el tercero es el que se hará cargo del protocolo de intercambio de datos.

Paso 3: Preparación de datos y envío de los criptobloques

Hasta ahora han sido pasos preparatorios para empezar con el protocolo de intercambio de datos por ello todo el flujo de comunicaciones restante se refleja en la Figura 3.4

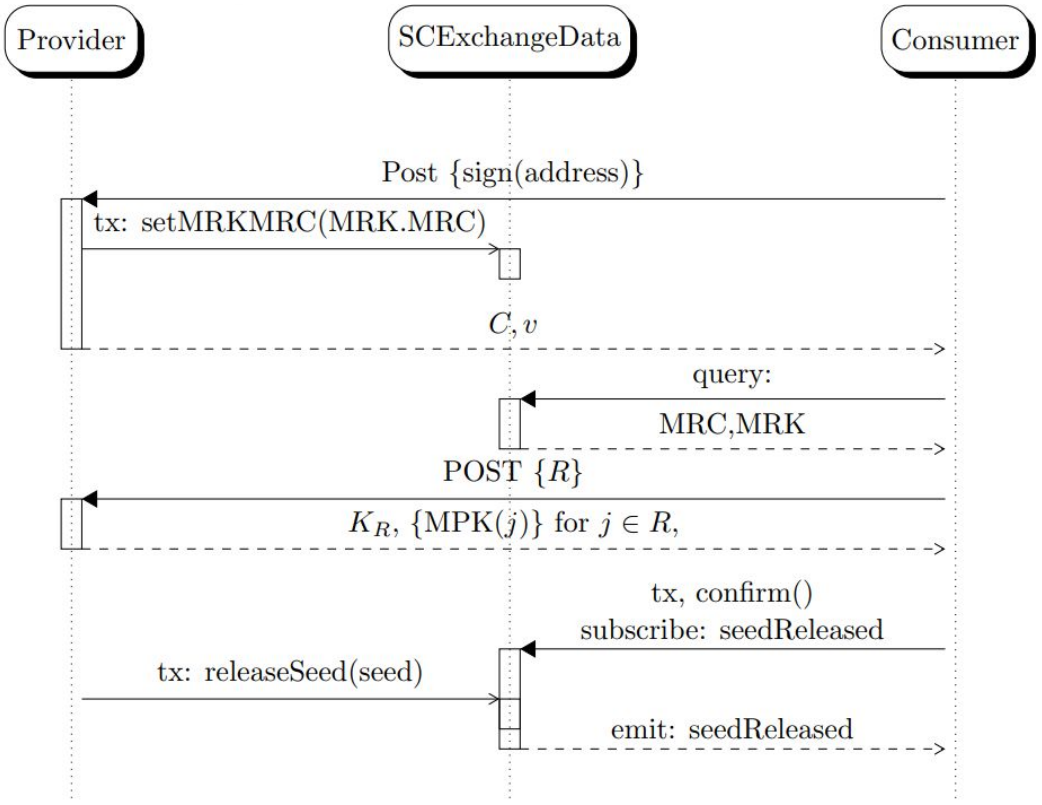

Fig. 3.4. Diagrama de flujo del protocolo de intercambio de datos

Una vez el consumidor cree el Smart Contract, se pondrá en contacto con el proveedor, ver en la Figura 3.4. el primer mensaje (usará la interfaz proporcionada por el marketplace o con su propia implementación si así lo prefiere), enviando la dirección de su Smart Contract. El servidor del proveedor comprobará que efectivamente en la Blockchain hay un nuevo Smart contract (verde, véase Figura 3.4) que se ha creado a partir de suyo (rojo) con esa dirección (pues en su Smart Contract tiene un recuento de los SC que se han creado a partir de este). Si todo es correcto, el proveedor procederá a la preparación de los datos.

El proveedor debe preparar los datos dividiéndolos en bloques, debe hacerlo en un orden diferente cada vez que empiece el proceso para evitar ataques de replay (ver en ataques de replay para más información). Habrá casos en los que los datos se dividen "naturalmente", por ejemplo, en registros de una base de datos donde cada registro es una muestra, en esos casos tendrá que agrupar registros aleatoriamente para crear bloques de estos y otras situaciones donde las muestras deben generarse a partir de una fuente compacta de datos, por ejemplo un video, en ese caso deberá tomar trozos del video y reordenarlos.

Cuando ya tenga creados los bloques de datos deberá crear una clave diferente para cifrar cada bloque, para ello creará una “seed” siendo éste un número aleatorio hexadecimal del cual se derivarán todas las claves. En caso que la compra se efectúe con éxito se acabará revelando la seed para que el consumidor pueda desencriptar todos los bloques de datos.

Es muy importante que con el conjunto de claves que obtiene el consumidor no pueda deducir el resto de claves pero que usando la “seed” sea fácil derivar todas las claves. Para resolver este problema se ha decido usar el “hash” del índice del cripotobloque concatenado con la “seed”, siguiendo el ejemplo como se ve en la Figura 3.4.

Hash(00seed) → clave bloque 0

Hash(01seed) → clave bloque 1

Hash(02seed) → clave bloque 2

Gracias a las propiedades del “hash” estas operaciones no se pueden invertir y son muy óptimas de calcular en Blockchain, más que sumar un entero por ejemplo, por lo que son perfectas para nuestro protocolo.

Hay que tener en cuenta que la “seed” se hará pública al final del proceso, es por eso que todo el tráfico de datos debe realizarse en un canal seguro, y que no se puede reutilizar un Smart Contract, pues una vez revelada la “seed” ya no se pueden utilizar esas claves, dado que tanto la seed como la manera de calcular las claves es pública. Por ello por cada consumidor se deberá encriptar los datos con “seeds” diferentes.

Además el proveedor hará un Merkle Tree (concepto explicado en el apartado 1.3. de Merkle Tree) tanto de los bloques cifrados (MRC: Merkle Root Crypto) como de las claves (MRK Merkle Root Key), dando lugar a dos merkle roots. Éstos se subirán a la Blockchain como se puede ver en la Figura 3.4. “SetMRKMRC”.

Seguidamente el proveedor enviará el conjunto de criptogramas al consumidor por un canal “ordinario”, off-blockchain. La blockchain nunca debe ser un lugar donde almacenar datos. No solo resulta inviable por sus costes, además se trata de datos que podían ser sensibles, y en caso de subirlos a la Blockchain y que se descubriese la clave para desencriptarlos, la información sensible se descubriría, dando acceso a millones de usuarios a ésta, además de que sería, en un principio, permanente e inmutable. Por estas razones se ha decidido usar un canal off-Blockchain, por supuesto deberá estar cifrado pues nadie más debe acceder a esta información. Véase en la Figura 3.4. (C,v) donde:

C: representa todos los criptobloques

v: representa el número de muestras que puede pedir el consumidor

El consumidor podrá comprobar que los criptobloques que le han entregado pertenecen al MRC subido a la Blockchain. El consumidor podrá obtener los merkle roots de la Blockchain enviando una “query”, véase en la Figura 3.4. (query: MRC,MRK). Se ha usado la tecnología de merkle tree pues aporta una gestión eficiente y segura en la verificación de grandes volúmenes de datos. Esto permite asegurar al consumidor que el proveedor no va a alterar el conjunto de datos prometido.

Pero, *¿cómo nos aseguramos que quien inicia la comunicación es el “verdadero” consumidor?* El Smart Contract puede comprobar que cuentas intervienen en él, teniendo guardada tanto la del proveedor como la del consumidor en el momento en que se creó, y ambos solo pueden interaccionar con él de ciertas maneras, por ejemplo, el proveedor no puede confirmar un pago. De hecho el Smart contract funcionará como una máquina de estados, y en cada estado sólo una de las dos partes podrá interactuar con éste, véase la Figura 3.5.

Solo las personas con las claves privadas correspondientes podrán interaccionar con el Smart Contract, pero entonces, *¿cómo el proveedor sabe que el consumidor es quien dice ser en la comunicación off-Blockchain, o sea, el creador del Smart Contract?* Si un atacante logra suplantar al consumidor podría ver una muestra de los datos gratuitamente además de entorpecer el proceso. Para ello la comunicación se firmará con las claves que se usan en la Blockchain, más concretamente, en el primer mensaje del consumidor, este firma con su clave privada la dirección del Smart Contract del intercambio de datos.

El proveedor puede comprobar que la firma pertenece a la cuenta del consumidor, que a su vez es es la que ha creado el Smart Contract del intercambio de datos, vinculando ambas comunicaciones.

Paso 4 : Selección de las muestras que serán reveladas

El siguiente paso es que el Consumidor compruebe que los criptobloques recibidos concuerdan con el merkle root colgado en la Blockchain. Una vez esta comprobación se ha llevado a cabo satisfactoriamente el consumidor puede pedir algunas claves que cifran los bloques, para así descifrarlos y obtener una muestra de los datos que va a comprar, pudiendo valorar la calidad y utilidad que le aportan.

Para pedir las claves el consumidor elegirá los índices de los criptobloques que desea desencriptar, este será un porcentaje de los criptobloques especificados por el proveedor, en el paso 3, el parámetro **v.** Es importante que esta muestra sea suficientemente pequeña para que el consumidor no le salga a cuenta hacer ataques de replay. Pero debe ser algo significativa, ya que si no, el proveedor podría intentar vender datos falsos o de muy poco valor sin que el consumidor se de cuenta. Se hablará más extensamente de ambos casos en posibles ataques.

Como el consumidor no conoce qué información hay en cada bloque, se entiende que escogerá “al azar”, aunque en el fondo lo que ocurre es que escoge a ciegas, de aquí el nombre de verificación aleatoria.

Gracias a las propiedades del Merkle tree el consumidor tendrá la total confianza de que el proveedor no podrá cambiar ningún dato ni clave, en otras palabras, que los datos que sean revelados serán verdaderamente un subconjunto de los datos reales a vender.

Es importante que los índices se escojan al azar o a ciegas, pues si el proveedor tuviese el poder de elegir la muestra que enseña, podría estar sesgada, dando muchas facilidades para engañar sobre la calidad de sus datos. Por ello se decidió que el consumidor fuera quien decidiera a ciegas los índices que desea. La otra posibilidad sería que el Smart Contract decidiese los índices usando algoritmos de números aleatorios, pero estos son bastante arduos dentro de la Blockchain, pues como se ha mencionado anteriormente el código ejecutado en blockchain debe ser determinista, todo el mundo debe tener el mismo resultado, así pues no se pueden calcular números completamente aleatorios, deben ser pseudoaleatorios, pero entonces dependiendo del valor de estos los mineros pueden llegar trucar la “aleatoriedad” hasta cierto punto (por ejemplo en caso de coger el tiempo exacto en que se mina un bloque), por ello se ha decidido no entrar en estos problemas y que el consumidor escoja a ciegas los índices.

Como ya se ha mencionado el proveedor cambiará el orden de los datos cada vez que reciba una petición nueva de un consumidor y, por tanto deberá calcular un nuevo Merkle tree, etc... para que no puedan poco a poco sacarle los datos, una vez más se explicará con más detalle en la sección de ataques.

El javascript del marketplace proporciona un generador de índices aleatorios, pero si el consumidor lo prefiere los puede elegir él. Una vez elegidos se enviarán al proveedor con el post{R}, véase la Figura 3.4. donde R es el conjunto de índices enviados.

Paso 5 : Revelación de las muestras escogidas

Es este punto el proveedor envía las claves correspondientes a los índices escogidos como respuesta de la última petición, véase en la Figura 3.4. (Kr, {MPK (j)}$\forall j \in R$) donde:

Kr: son las claves que descifran los cripto bloques que tienen como índices **R** (del anterior paso).

{MPK (j)}$\forall j \in R$: Son las Merkle Proofs para cada una de las claves **Kr.**

El consumidor deberá usar las claves junto a las merkle proofs y comprobar que con todas llega al MRK (Merkle Root Keys) que se ha subido anteriormente en la Blockchain en el paso 3. De esta manera tiene una prueba fehaciente de que ninguna clave y ningún bloque se ha alterado, por tanto los datos que desencripte no podrán haber sido falsificados.

Paso 6 : Confirmación

En cualquier punto hasta aquí, tanto el proveedor como el consumidor pueden haberse echado para atrás, por ejemplo si el proveedor tiene algún problema con los datos, o el consumidor no queda satisfecho con la muestra o bien ha hecho las comprobaciones anteriormente descritas y no han sido satisfactorias, por ejemplo si el proveedor ha actuado de manera fraudulenta o errónea, dando keys que no funcionan o criptobloques que no desencadenan en el mismo merkle root.

También puede haber ocurrido un “time out”, todos los estados del Smart Contract están “timeados”, en este caso se da una semana para actuar en cada paso. En caso de que uno de los dos no interactúe cuando debe y se pase el plazo del tiempo o explícitamente llame a la función de cancelar se les devolverá el dinero a ambas partes, dependiendo de quién haya cancelado el

intercambio y de cuándo, se usará el dinero colateral para compensar la parte que haya salido perdiendo, o quemarlo para desincentivar ataques, se explicará con más énfasis en su sección.

Si todas las comprobaciones son correctas y la muestra de datos satisface las necesidades del consumidor, éste podrá confirmar la compra mediante una transacción (**confirm()**), véase la Figura 3.4. Una vez confirme ya no podrá desdecirse, solo podrá recuperar su dinero en caso de que haya algún error o fraude en el intercambio de datos (se explicará a continuación) o el proveedor cambie de opinión.

Paso 7: Revelación de la *seed*

El proveedor en respuesta revelará la "seed" mediante la Blockchain, (**tx:releaseSeed(seed))**, veáse Figura 3.4.

El consumidor irá haciendo peticiones a la Blockchain a intervalos de tiempo, o bien se suscribirá a un evento para obtener la "seed". Una vez el consumidor obtenga la "seed" le permitirá derivar todas las claves y descifrar todos los datos. Si todo es correcto el protocolo habrá acabado y el proveedor podrá retirar el dinero cuando acabe un time out, en este caso una semana, que es el tiempo que se le da al consumidor para realizar las comprobaciones.

Pero es posible que el proveedor actúe de forma fraudulenta o errónea al no revelar la "seed" correcta y el consumidor se quede sin los datos y sin el dinero.

Recapitulando, hasta ahora el consumidor ha podido comprobar que existen una serie de datos encriptados, que pertenecen a MRC, y que ciertas claves pertenecen a MRK y éstas desencriptan correctamente los criptobloques. Dado que se ha hecho una verificación aleatoria de ambos merkle roots, significa que existen tanto los datos que se quieren vender como las claves que los desencriptan, y ambas son correctas.

Por lo tanto, la única manera que tiene el proveedor de estafar al consumidor es revelar una "seed" falsa o errónea. En ese caso el proveedor ganaría el dinero sin revelar los datos. Por ello el protocolo agrega un paso adicional opcional para resolver este conflicto y proteger al consumidor. En caso de que la "seed" sea incorrecta, el consumidor podrá llamar a una función del Smart contract llamada Conflict. Esta función generará a partir de la "seed" todas las claves y calculará el MRK con éstas. En caso de que no coincida con la anterior se le dará la razón al consumidor y éste recuperará todo su dinero. Es importante hacer hincapié en el gran coste computacional que tiene esta última función, que se traduce en un coste monetario. Por ello esta función se

entiende como un caso excepcional que el consumidor solo pagará en caso de que esté seguro de que tiene razón, en caso contrario el proveedor será pagado y el consumidor habrá gastado dinero inútilmente en la ejecución de esa función. Para pagar los costes de esta función y desalentar a ambas partes de usar el protocolo de manera fraudulenta se usará el dinero colateral ya mencionado anteriormente. En caso de que el consumidor tenga razón se le devolverá todo el dinero bloqueado en el Smart Contract además de pagarle todo el dinero colateral del proveedor, probablemente resultando en una pequeña ganancia.

Diagrama de Flujo Smart Contract

Este es el diagrama de flujo por el cual pasa el Smart Contract durante todo el proceso anteriormente descrito, se puede ver que hay las mismas transacciones que en la Figura 3.5 si simplemente los usuarios hacen las transacciones siguiendo con normalidad el protocolo, trazando una línea “horizontal” en el diagrama. Este diagrama también tiene en cuenta el caso de que alguno quiera cancelar, o no responda en un plazo de tiempo “timeout”, por eso hay las bifurcaciones. Dependiendo de cuando el consumidor o proveedor cancelen, se llevarán una parte del colateral diferente, pero siempre el consumidor recuperará su dinero. Por ejemplo, si el proveedor no responde cuando el consumidor crea el contrato y se pasa el timeout (una semana) el consumidor podrá recibir todo el colateral que ha puesto él y el proveedor por las molestias generadas. En caso de que sea el Consumidor quien no responda en el paso siguiente (wait forConfirm) y se pase el timeout, el consumidor recuperará su dinero pero el Proveedor se llevará ambos colaterales por las molestias ocasionadas.

El caso crítico es si el consumidor cancela. Ambos han cumplido y respetado los timeouts, que el consumidor cancele se puede deber o bien a que no le interese los datos o bien por que los datos son fraudulentos, falsos, replicados, o porque el proveedor le ha enviado muy pocas muestras de prueba. Pueden haber muchos factores. Para desincentivar a ambos de hacer posibles fraudes, tanto el proveedor vender datos fraudulentos como el consumidor hacer ataques de replay donde hace miles de peticiones para sacar los datos a muestras gratuitas, se quemarán ambos colaterales en este caso. Quemar ether significa que nadie se lo queda y queda en un “vacío”. En vez de quemar una solución interesante sería donarlo a una ONG, pero en cualquier caso que ni el consumidor ni el proveedor se lo queden, pues es la única opción de diseño que ecuánime para todos y desincentiva va a hacer ataque a ambos.

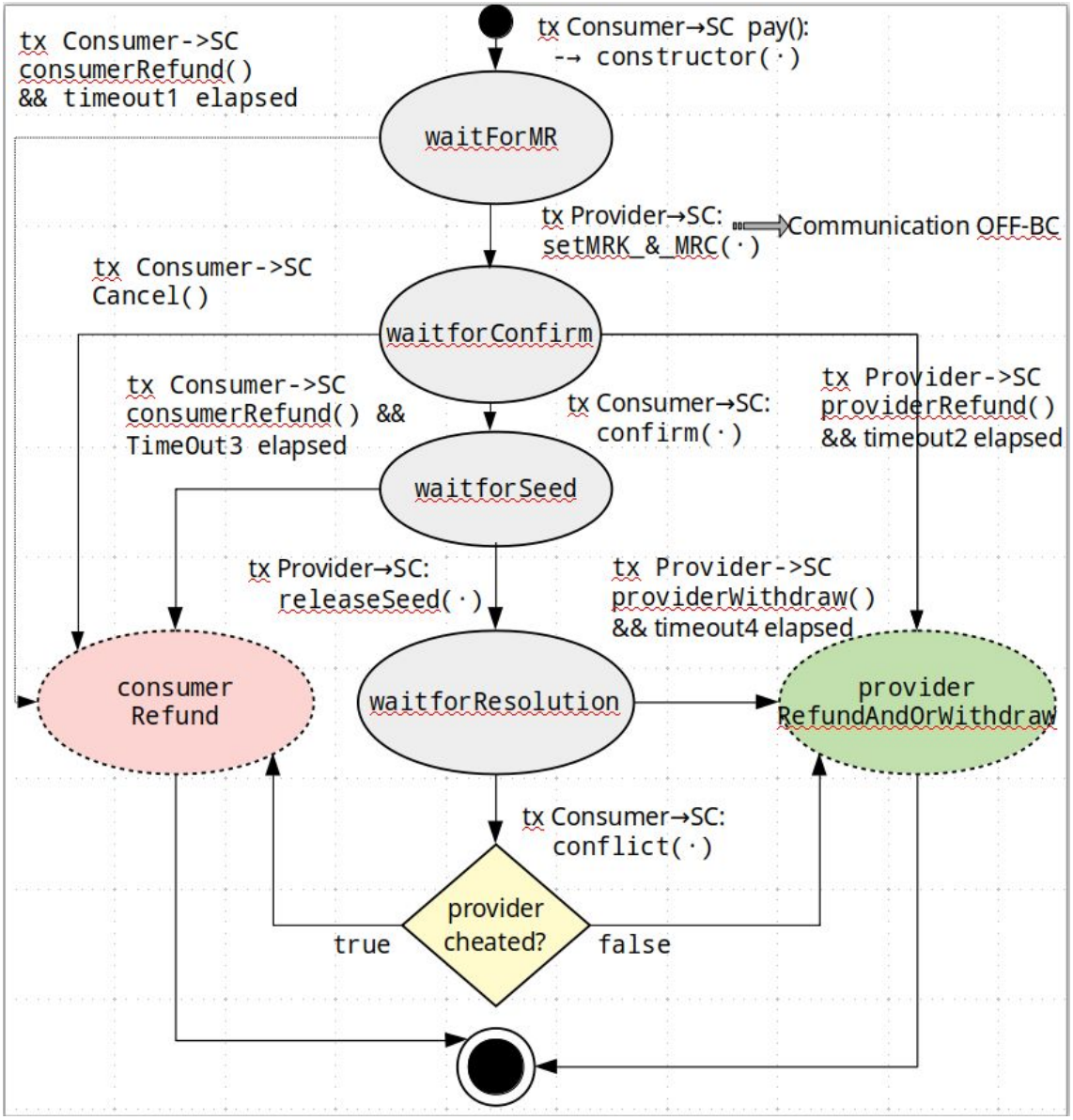

Fig. 3.5. Diagrama de flujo del Smart Contract

3.3. Posibles ataques al protocolo

En este apartado se explicarán tanto los posible ataques que puede sufrir el protocolo, por el proveedor, por el consumidor o por un agente externo, como los mecanismos de que dispone el protocolo para contrarrestarlos.

Eavesdropping por un atacante externo

Se trata de que un atacante pueda acceder a la comunicación privada off-Blockchain que tienen el consumidor y el proveedor, este concepto de escuchar una comunicación privada se llama “Eavesdropping” en inglés. En ese caso podría guardarse los criptogramas y en caso de que la compra de los datos se realizará, se revelaría la seed y el atacante también conseguiría los datos. Por esta razón el canal off-Blockchain debe estar encriptado de principio a fín.

Datos falsos o replicados por parte del proveedor.

El proveedor podría intentar vender datos replicados o falsos, estos no tendrían ningún valor para el consumidor como es obvio. La idea de este ataque es vender más información de la que actualmente se dispone y que el consumidor no llegue a ver los datos replicados o falsos con la muestra gratuita aleatoria que se le da.

Dado que esta comunicación ocurre Off-blockchain el Smart Contract no puede actuar al respecto. Solo se le pueden dar directrices al consumidor. El consumidor deberá coger una muestra suficientemente grande de datos para que sea representativa con los datos totales que hay. También se le recomienda que coja conjuntos de índices consecutivos para encontrar posibles repetidas. En caso de que el proveedor le de una muestra muy pequeña de datos se le recomienda que no confirme la compra.

Oferta replicada por el proveedor

Parecido al ataque anterior, simplemente un proveedor tiene un set de datos donde un pequeño porcentaje son útiles/correctos. Este intentará vender muchas veces su conjunto, hasta que un consumidor le toque una muestra no representativa que los datos que haya escogido sean buenos. Para evitar este ataque se quemará una parte del colateral del proveedor cuando un consumidor no confirma una compra. De esta manera el consumidor irá

perdiendo dinero cada vez que le rechacen un intercambio, haciendo inviable un ataque por probabilidades pues habrá perdido más dinero que si hubiese vendido sus datos honestamente.

Cambio de la seed por parte del proveedor

El proveedor podría revelar una “seed” incorrecta, engañando al consumidor, quedándose con el dinero sin dar los datos a cambio.

Como se ha explicado anteriormente la función conflict del SC comprueba que con las claves derivadas se llega al MRK (Merkle Root Key) si el proveedor intentase realizar este ataque perdería el conflict y el consumidor recuperaría su dinero. La única manera de quedarse el dinero y ganar el conflict sería “romper” la función de Hash, encontrando una colisión. Como se ha explicado en el apartado de Hash, romper una función de Hash de 256 bits como la que se ha utilizado para la implementación es computacionalmente imposible, ver ecuación 1.1. (página 14).

Ataque de replay por parte del consumidor

El proveedor da una muestra aleatoria al consumidor de sus datos antes de realizar el pago, este paso puede ser aprovechado por un atacante para hacer miles de peticiones, obteniendo los datos sin llegar a pagar al proveedor.

Existen 4 puntos en este proyecto que desincentivan los ataques de replay:

1. Los datos están desordenados, por lo que si se piden muchas muestras aleatorias, lo más probable es que salgan datos repetidos, y por tanto fútiles para el atacante.
2. El atacante debe bloquear un dinero como se especifica en el paso 2, así que si quiere hace muchas peticiones concurrentes deberá tener una gran cantidad de dinero, más aún teniendo en cuenta el punto 1.
3. El atacante pierde el dinero colateral y el dinero gastado en la creación del Smart Contract en cada petición.
4. La muestra de datos es suficientemente pequeña para que el coste de hacer tantas peticiones sea mucho mayor que el de pagar los datos directamente al proveedor.

Ataque de suplantación de identidad del consumidor por una atacante externo

Habiendo creado el smart contract una atacante externo podría quererse hacer pasar por el consumidor, esto podría servir para:

- Conseguir quedarse con la muestra gratuita de datos
- Entorpecer el proceso, haciendo inservible los contratos del proveedor (ataque a la competencia).
- Quedarse con los criptobloques y esperar, en caso de que el consumidor los acabé comprando y se revele la seed él también tendrá acceso a los datos.

Para evitar este ataque se toman dos precauciones.

a. El consumidor debe firmar en el primer mensaje la dirección del Smart contract que creó con esa misma cuenta. Este proceso da una prueba fehaciente de que el usuario que se pone en contacto con el proveedor conoce la clave privada del consumidor que creó el Smart Contract, vinculando ambas comunicaciones. Un atacante nunca conocerá la clave privada del consumidor, haciendo imposible la suplantación de este.
b. El proveedor sólo enviará los cripto bloques una vez, quitando la posibilidad de que un atacante pudiera quedarse con estos y “esperar” a que el consumidor los compre, dándole acceso a él también

3.5. Aplicación desarrollada del protocolo descrito

La aplicación que se ha montado implementa el protocolo de intercambio de datos entre el consumidor y el proveedor. El consumidor tendrá el javascript bajado del Marketplace con los datos adecuados, y el proveedor un dataset para vender y un Smart Contract creado.

Infraestructura de la aplicación

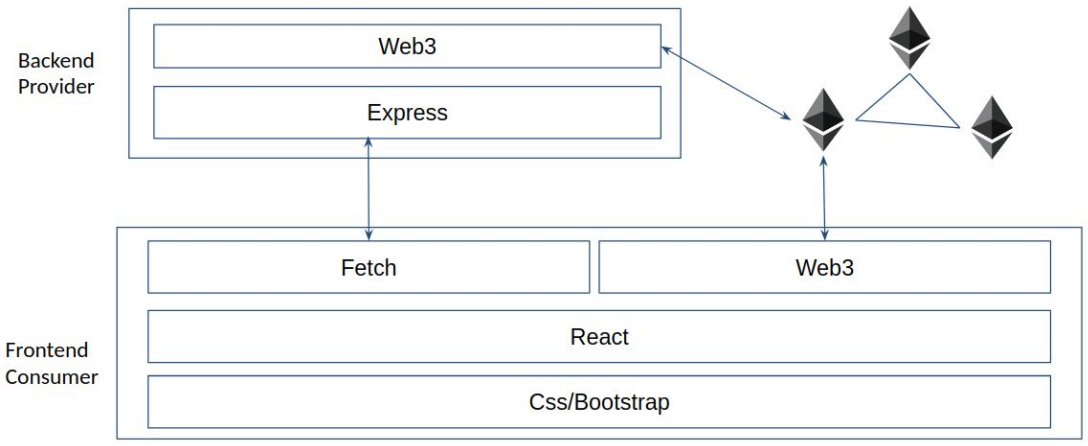

Fig. 3.6. Stack de protocolos de la aplicación desarrollada

Se ha implementado un Front End, hecho con react y las vistas y css sacados de la librería Bootstrap. Este cliente se conecta con la Blockchain con Web3 con ayuda del Metamask, pues este hace la gestión de las cuentas, firmando las transacciones. Además interactúa con el proveedor con API calls.

El proveedor tendrá un Backend hecho con express. Como se ha hecho a modo de prueba, no tendrá una base de datos sino que tendrá los datos que quiere vender en un fichero, este se leerá y encriptará como se ha descrito anteriormente. El proveedor se conecta con la Blockchain con Web3 directamente sin pasar por un Wallet como Metamask. Tiene una cuenta con ether hardcodeada y usa esta para firmar las transacciones.

Ejemplo de intercambio de datos usando la aplicación

Para la implementación de la aplicación se ha usado Ganache (simulador de Ethereum personal), como entorno de pruebas. Metamask (el wallet o gestor de las cuentas), se conectará a Ganache directamente, se le añadirán dos cuentas de Ganache, pues éstas son las que tienen ether en este Ethereum simulado y por tanto las que pueden hacer transacciones. Una cuenta será del proveedor y la otra del consumidor. el provider y otra el consumer.

El proyecto puede encontrarse en la dirección https://github.com/invocamanman/TFGExchangeData.

Para usarlo se deberá tener instalado Npm, Truffle, Ganache y Metamask. Se deberá primero hacer un “truffle migrate” con el Ganache abierto, para instanciar los Smart Contracts en la Blockchain simulada, por defecto en este proyecto Truffle se conectará a Ganache. Esto resultará en algo como:

```
2 deploy contracts.js
=====================

   Replacing 'Factory'
   -------------------
   > transaction hash:    0x55a26df50d6d7c4393c83836e0f13d1cd9b7892eb52f2d105a2e8eb5bc0e5cb1
   > Blocks: 0            Seconds: 0
   > contract address:    0x1A90A2EF90D19f10d693A3BfAcEA929423Cf92D0
   > block number:        3
   > block timestamp:     1567794909
   > account:             0x53878D5E54C6A8d115853cBD663bEfD07b5b118D
   > balance:             99.9602841
   > gas used:            1710879
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.03421758 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.03421758 ETH
```

Fig. 3.7. Instanciar Smart Contract con truffle

Después se deberá entrar en las carpetas react y express en dos terminales diferentes y en ambas hacer un “npm start”, para inicializar tanto el front end como el back end. Finalmente se deberá importar una cuenta de Ganache al Metamask, ya que éstas son las únicas con ether, o sea con capacidad para interactuar con la Blockchain. Para importar las cuentas tendremos que pegar la clave privada de una de estas en el Metamask como se muestra en la figura 3.8

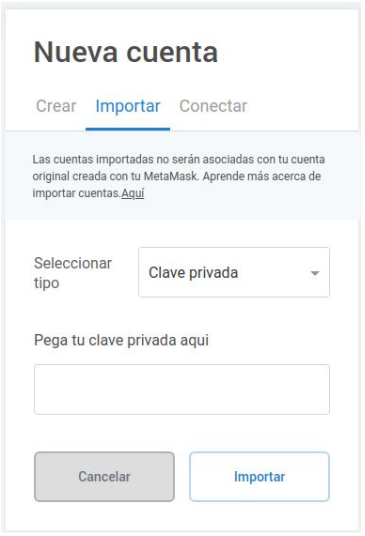

Fig. 3.8. Importación de cuenta con clave privada en Metamask

Con este set up ya funcionará la aplicación. En caso de que se quiera usar para una red pública de Ethereum, supongamos que seguimos en un entorno de test, se usará una de las redes de pruebas de Ethereum como puede ser Robsten, donde se puede pedir Ether gratuitamente en https://faucet.ropsten.be/. Obviamente este ether no se traducen en valor monetario. Una vez la cuenta esté cargada con fondos se deberá implementar un HDwallet en el truffle para que pueda crear el Smart Contract usando las clave correspondientes a la cuenta que se ha pedido el ether además de cambiar la red que se conecta por defecto. También deberemos cambiar las direcciones de los Smart Contracts guardadas en los ficheros config.js tanto del express como del react a las que resulten una vez desplegados..

Ahora se seguirán los mismos pasos que en el apartado de diseño del protocolo.
Como partimos del Smart Contract del proveedor que hemos instanciado con truffle, el primer paso del protocolo ya está hecho.

Paso 2: Creación del Smart Contract por el consumidor

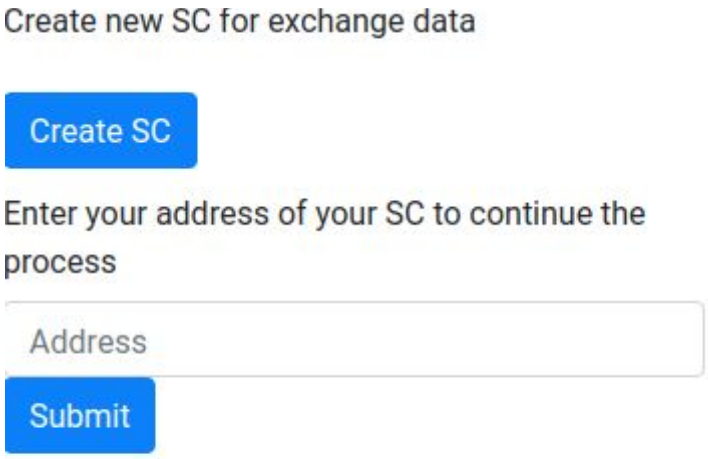

Fig. 3.9. Implementación del paso 2

En un principio se podrá tanto crear un nuevo Exchange data como continuar el proceso de uno creado anteriormente al introducir la dirección del SC

Si se clica en Create SC:

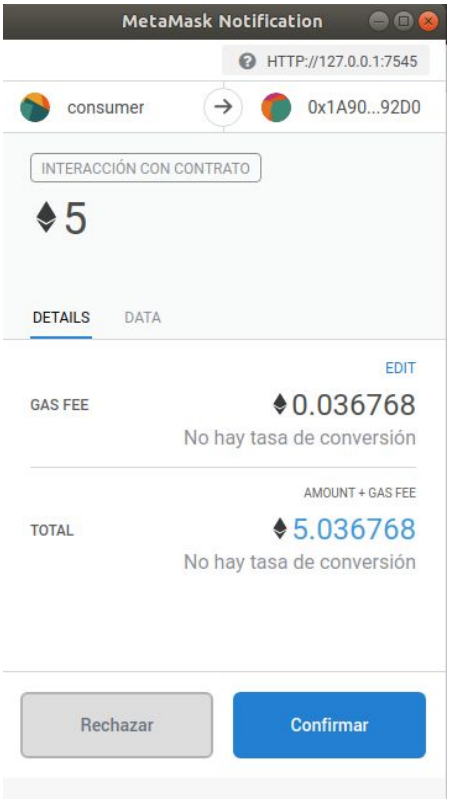

Fig. 3.10. Pop up del metamask

Nos aparecerá una pop up de metamask preguntando si queremos realizar esta transacción, en este caso el value+data cuesta 5 ethers y el coste de crear el nuevo Smart Contract 0.036 ethers. Siempre nos saldrá este popUp cuando queramos hacer una transacción des del Front End

Cuando se haya creado satisfactoriamente saldrá lo siguiente.

Fig. 3. 11. Implementación del paso 3

La stage nos indicará en qué estado del proyecto estamos, véase la figura 3.5, ahora estamos al 0 pues éste se acaba de crear.

Paso 3: Preparación de datos y envío de los criptobloques

Le damos al botón de la Figura 3.11, enviando la dirección del SC firmada (aunque sea un mensaje de texto y no una transacción también saldrá un popup del metamask conforme lo queremos firmar).

Fig. 3.12. Pop up Metamask firmas mensaje

Desgraciadamente no podemos ver el contenido del mensaje por que esta en hexadecimal (es una dirección de Ethereum), y debido a un intento de conversión del Metamask a string acaba siendo ilegible.
Al firmar el mensaje recibiremos todos los criptobloques en un fichero txt.

Paso 4 : Selección de las muestras que serán reveladas

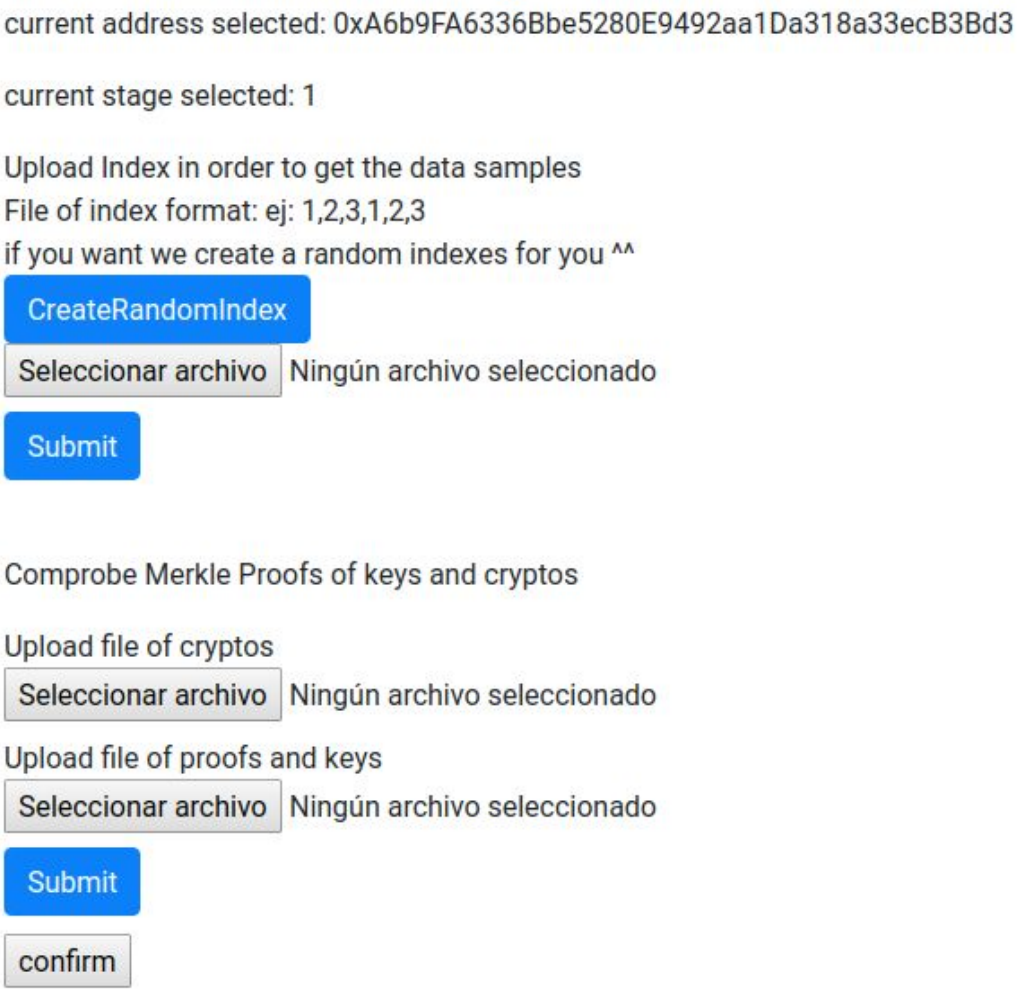

Figura 3.13. Implementación del paso 4

En este formulario podremos hacer diversas cosas, para empezar vemos que hemos cambiado de Stage pues el Smart Contract ha pasado al siguiente estado. En este paso podemos enviar al servidor unos índices separados por comas, los puede crear tanto el proveedor como el propio cliente clicando al botón de CreateRandomIndex generando un fichero de índices aleatorios.
Justo abajo de este botón podemos introducir el fichero creado para que el proveedor nos envíe las keys junto con su Merkle Proof de los índices seleccionados, que también nos bajaremos en formato txt.

Paso 5 : Revelación de las muestras escogidas

Una vez tenemos el fichero de criptobloques y el fichero de proofs y keys podemos introducirlos en el formulario de abajo para comprobar que los merkle

roots coinciden con los que estan el la Blockchain y para desencriptar la muestra de pruebas, en un fichero txt llamado result.

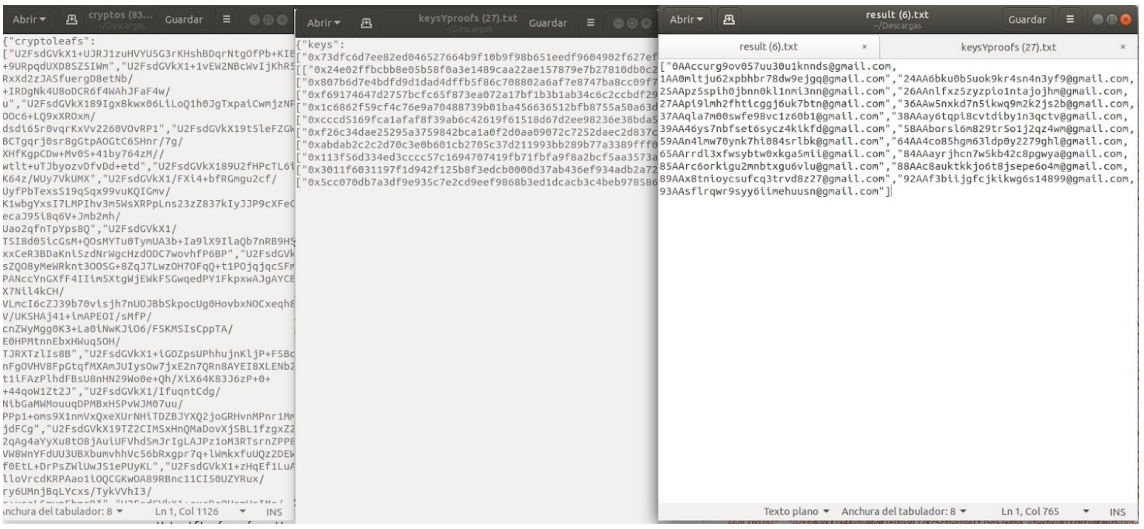

Fig. 3.14. Ficheros resultantes de ejecutar el paso 4 y 5

Véanse los 3 ficheros. El resultado son los datos desencriptados. Para hacer visible que claramente son datos, se han puesto como muestras correos electrónicos y todos acaban en @gmail.com.

En caso de que el consumidor esté de acuerdo con los datos revelados del fichero result, podrá clicar en confirm para continuar con la compra

Paso 6 : Confirmación

Al clicar el botón confirm el SC cambiará de estado, eso provocará que el front end muestre:

Fig. 3.15.Implementación del paso 6

Ahora deberíamos esperar a que el proveedor revelará la “seed”, para hacer este proceso más síncrono, se ha creado una comunicación que avisa al proveedor que el consumidor ha confirmado la cuenta y quiere la seed.

Paso 7: Revelación de la *seed*

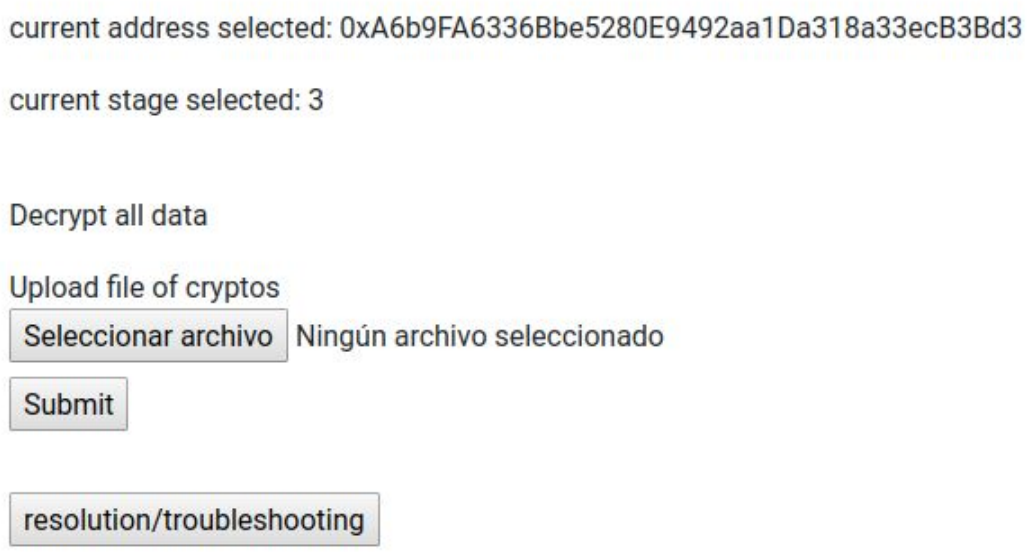

Fig.3.16. Implementación del paso 7

Ahora podremos desencriptar todos los datos si ponemos el fichero de criptobloques que hemos recibido al principio del protocolo. El front end coge la seed de la Blockchain, calcula todas las claves y desencripta los bloques, resultando en un fichero llamada "*alldata"* con todos los datos desencriptados, ver la figura 3.16

Fig. 3.16. Datos desencriptados

Conflict()

En caso de que no se puedan desencriptar los datos, el consumidor puede darle al botón de conflict /resolution.
Como hemos sido sinceros el programa detectará que la seed es correcta y el consumidor no recuperará su dinero. Este apartado ha sido el más arduo de implementar pues se tenía que conseguir que tanto en javascript (servidor/cliente) como en Ethereum se consiguiera las mismas funciones de Hash y concatenación. Aunque parece trivial no lo es, requiere una propia implementación del merkle tree tanto de Javascript como de solidity, pues las librerías existentes de estas no son compatibles. El problema se solucionó gracias a una librerías de Ethereum para Javascript que hacían exactamente la misma función de hash que en este. Como se ha explicado en el apartado del Hash, Ethereum usa una función de Hash algo especial, pues es el SHA-3 pero no la última implementación de este. Esto lleva muchas confusiones en las librerías de Hash, dando como resultado hashes que funcionan ligeramente diferente, pero el resultado es completamente distinto. Ha sido de una gran ayuda una branch de Open Zeppelin (librería más conocida de Smart Contracts) que habían empezado un proyecto de cómputo de Merkle Tree en Ethereum. Pues es típico validar pruebas de Merkle Tree en Ethereum pero no calcular un árbol entero como se realiza en este proyecto.

Es curioso que la implementación de esta ha sido tan óptima que gasta solo una media de 130000 de gas, que es algo menos que la creación del mismo Smart Contract. Esta es directamente proporcional al número de bloques que se encripta y se han encriptado 50 bloques. Pero tiene una fácil solución, simplemente hacer los bloques contengan más datos, reduciendo así el número de bloques y por tanto el coste de esta función, pues solo depende del número no de su tamaño. Aún así se recuerda que esta función sirve solo para desalentar a proveedores de actuar fraudulentamente, por lo que no hace falta tener en cuenta estas directrices.

CONCLUSIONES

El objetivo personal que impulsó el proyecto fue aprender y comprender las tecnologías Blockchain en profundidad, entendiendo casos de uso tanto sencillos como complejos, y de ese modo ser capaz de desarrollar una aplicación útil y de cierta complejidad con estos conocimientos. Personalmente estoy contento con el resultado, he aprendido muchísimo, he consolidado muchos conocimientos realizando este trabajo, entiendo tanto como casos de uso, como problemáticas tanto en la teoría como en la implementación.

En el proyecto he tratado de explicar, en profundidad, las tecnologías utilizadas sobre todo Blockchain, pues hay un gran desconocimiento de ésta y es vital tanto para entender el proyecto.

Se ha conseguido idear e implementar un protocolo de intercambio de datos muy robusto, se han pensado en todos los posibles ataques a éste y la protección contra ellos. Se ha implementado tanto un cliente como un servidor que se conectan a la Blockchain y continuamente extraen información y realizan transacciones. Pese a que las vistas de la aplicación son bastante pobres, la lógica que hay detrás tienen cierta complejidad, así pues me siento satisfecho con el resultado.

Se podría haber calculado, más en detalle, el dinero colateral que deben poner ambas partes, he optado por dejar abierta la elección al proveedor. También se podrían haber dado directrices más específicas de qué porcentaje de muestras debe dar el proveedor para que éstas sean representativas pero no incentive al consumidor a realizar ataques de replay. El desconocimiento de este mercado como el valor típico de los datos, el volumen que se suelen venderse... han dificultado una estimación realista de estos parámetros. Hace falta un gran conocimiento del mercado para poder implementar correctamente esta Teoría de Juegos, pero los conceptos y las ideas de cómo se debe implementar sí se han pensado.

En conclusión, con una buena auditoría al Smart Contract del protocolo, mejorando algunos aspectos, dando confianza a los usuarios creo que podría ser de gran utilidad para muchas empresas.

BIBLIOGRAFÍA

[1] Antonopoulos, A.M.,Wood, G., *Mastering Ethereum: Building Smart Contract and DApps.* Ed. O' Reilly (2019).

[2] Antonopoulos, A.M.*Mastering Bitcoin: Programming the open Blockchain*. Ed. O'Reilly (2017).

[3] https://www.nytimes.com/2013/12/15/sunday-review/the-bitcoin-ideology.html

[4] https://bitcoin.org/bitcoin.pdf

[5] OpenZeppelin ttps://github.com/OpenZeppelin/openzeppelin-contracts

[6] Rama de Openzepelin para el cómputo de Merkle Tree

https://github.com/dwardu/openzeppelin-solidity/tree/feature/merkle-root

[7] Documentacion oficial de web3 https://web3js.readthedocs.io/en/v1.2.1/

[8] Documentación oficial de reactjs https://reactjs.org/docs