Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Desarrollo de un prototipo Layer 2 para Ethereum

Development of a Layer 2 prototype for Ethereum

Florentín Pérez González

10 de junio de 2021

D. **José Luis Roda García**, con N.I.F. 43356123L profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

“Desarrollo de un prototipo Layer 2 para Ethereum”

ha sido realizada bajo su dirección.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de junio de 2021

Agradecimientos.

Agradecer a mi familia por su apoyo y confianza incondicional, así como a mis amigos. Agradecer también a José Luis Roda Garcia y a Benito José Cuesta Viera de la ULL; y a Antonio Manuel Estévez Garcia y Yeray Rodríguez Rodríguez de Open por sus consejos, supervisión y asesoría a lo largo del desarrollo del proyecto.

Licencia.

Resumen.

Blockchain es una tecnología de reciente aparición que, debido a sus características, levanta el interés de numerosos sectores que investigan distintas alternativas para lograr adoptar la tecnología a sus intereses. Dentro de las redes existentes, se encuentra Ethereum, la segunda cadena de bloques más importante (en valor de mercado) solo por detrás de Bitcoin y una de las más usadas por los desarrolladores debido al concepto de "contrato inteligente".

No obstante, Ethereum posee ciertas limitaciones que, en la actualidad, provocan que ciertos escenarios de uso y aplicaciones no puedan hacer uso efectivo de su red pública. Ethereum posee, pues un problema de escalabilidad consecuencia fundamentalmente de dos factores distintos: transacciones por segundo y coste por transacción, ambos inadecuados para escenarios de uso que requieren de cientos de interacciones con la red.

La comunidad ha propuesto numerosas alternativas para solucionar el problema de escalabilidad de Ethereum. Dentro de ellas se encuentran los protocolos Layer 2, cuya base funcional se basa en relegar de cálculos a la red principal de Ethereum.

El experimento realizado ha consistido en el desarrollo de un prototipo Layer 2 basado en el protocolo ZK Rollup con objeto de demostrar la viabilidad de estas soluciones para el problema de escalabilidad de Ethereum. El prototipo se ha desarrollado con la pretensión de lograr los mejores resultados posibles y de acuerdo a un escenario de uso de economía colaborativa.

Los resultados obtenidos con la ejecución del prototipo han demostrado una mejora de escalabilidad tanto en transacciones por segundo como en el coste de por transacción, siendo estos hasta 288 y 49 veces mejores respectivamente. Estos resultados demuestran la viabilidad de los protocolos Layer 2 como propuestas para mejorar la escalabilidad de Ethereum.

Palabras clave: Blockchain, Ethereum, Layer 1, Layer 2, ZK Rollup, Optimistic Rollup, Validium, Plasma, Escalabilidad.

Abstract.

Blockchain is a recently emerged technology that, due to its characteristics, raises the interest of numerous sectors, which are investigating different alternatives to adopt the technology to their interests. Among the existing networks is Ethereum, the second most important blockchain (in market value) only behind Bitcoin and one of the most used by developers due to the concept of "smart contract".

However, Ethereum has certain limitations that currently prevent certain usage scenarios and applications from making effective use of its public network. Ethereum therefore has a scalability problem resulting mainly from two different factors: transactions per second and cost per transaction, both of which are inadequate for usage scenarios that require hundreds of interactions with the network.

The community has proposed numerous alternatives to solve Ethereum's scalability problem. Among them are Layer 2 protocols, whose functional basis is based on relegating computation from the main Ethereum network.

The experiment carried out consisted of the development of a Layer 2 prototype based on the ZK Rollup protocol in order to demonstrate the viability of these solutions for Ethereum's scalability problem. The prototype has been developed with the pretension of achieving the best possible results and according to a collaborative economy usage scenario.

The results obtained with the execution of the prototype have demonstrated a scalability improvement in both transactions per second and cost per transaction, being these up to 288 and 49 times better respectively. These results demonstrate the viability of Layer 2 protocols as proposals to improve the scalability of Ethereum.

Key words: *Blockchain, Ethereum, Layer 1, Layer 2, ZK Rollup, Optimistic Rollup, Validium, Plasma, Scalability.*

Índice general.

Índice de figuras.

Índice de tablas.

1. Introducción.

1.1. Antecedentes.

Ethereum [1], lanzado oficialmente al público en 2015, supuso una revolución en el mundo de las tecnologías Blockchain al incorporar la capacidad de almacenar programas ejecutables en la red, conocidos como contratos inteligentes [2]. Esto provocó que, en escaso tiempo, llamase la atención de distintos sectores que veían en la tecnología diversas utilidades. No obstante, la tecnología, especialmente su red principal, posee ciertas limitaciones que suponen un impedimento para su adopción por parte de otros sectores. Es el caso, por ejemplo, de la cantidad de transacciones por segundo que, de media, soporta la red, alrededor de 15, insuficiente para escenarios de uso caracterizados por cientos o miles de interacciones por segundo. A ello, habría que añadir el hecho de que la realización de estas transacciones tienen un coste asociado en unidades de Ether cuyo valor económico está ligado al mercado de criptodivisas, donde se le asigna un valor que, en la actualidad, además de elevado, es alcista.

Por tanto, el desarrollo de aplicaciones que interactúen masivamente con la red se ve ralentizado debido a la incapacidad de esta de satisfacer sus necesidades además del elevado coste económico que supondría. Con objeto de corregir esta situación, existen diversas propuestas que pretenden aumentar la escalabilidad de Ethereum. Dentro de estas están los protocolos Layer 2 (a partir de ahora L2) que intentan resolver el problema relegando de carga a la red, realizando cálculos en entornos externos y comunicando los resultados de los mismos a la red principal (a partir de ahora L1).

1.2. Objetivo.

El objetivo de este proyecto es la recopilación de información sobre los distintos protocolos Layer 2 existentes destacando sus características fundamentales. Recabada dicha información, se elaborará el prototipo de uno de ellos definiendo previamente los requisitos básicos de sus componentes así como una posible arquitectura. El prototipo deberá mostrar resultados que justifiquen como, en efecto, esta serie de soluciones permite aumentar la escalabilidad de Ethereum.

1.3 Estado del arte.

Los protocolos L2 generan una capa adicional que se interpone entre el usuario y la red de Ethereum que este esté utilizando y que se encarga de realizar gran parte de las operaciones resultantes de las interacciones del usuario. Cada cierto tiempo, una versión resumida de las operaciones realizadas se transmite a la L1 con objeto de dejar constancia de las mismas. De esta manera, se relega a la red principal de la realización de una gran parte de los cálculos, alcanzando nuevas cotas de escalabilidad.

No obstante, existen numerosas formas de realizar este proceso, razón que justifica la existencia de numerosos protocolos L2. Cada uno de ellos posee sus respectivas características, así como sus ventajas y desventajas.

1.3.1 Plasma.

Plasma [3] es una de las soluciones Layer 2 pioneras y que más investigación han recibido a lo largo del tiempo. Funciona a través de la combinación de contratos inteligentes y árboles Merkle[4], utilizados para la gestión de metainformación; así como con el apoyo de un número elevado de cadenas ocultas. Estas se estructuran en forma de árbol y su función es disminuir la carga de su red padre, a la que periódicamente se reportan para comunicar información. Resulta relevante destacar que cada cadena es independiente en funcionalidad, implementación y diseño.

La comunicación entre cadenas se verifica y securiza a través de pruebas criptográficas, en concreto, *fraud proofs* [5], cuya implementación puede variar en cada una de las distintas cadenas. Estas pruebas garantizan que los usuarios puedan denunciar cualquier actividad deshonesta que detecten, activando los mecanismos de disputa necesarios.

El enfoque de este protocolo, aunque permite solucionar el problema de Ethereum posee ciertos inconvenientes. En primera instancia, la infraestructura que requiere es significativa, pues se deben mantener numerosas cadenas auxiliares así como los mecanismos comunicativos entre ellas. Por otro lado, la utilización de *fraud proofs* supone que se debe asumir una elevada finalidad para las operaciones, llegando a ser, en ocasiones, de semanas.

No obstante, el principal inconveniente de Plasma está ligado a su propio diseño, el cual dificulta proteger el sistema ante ciertas situaciones, perjudicando la seguridad general del mismo. Debido a ello, se han creado numerosas propuestas focalizadas, en ocasiones, a intentar solucionar los problemas inherentes al protocolo y, en otros, a adaptarlo a ciertos escenarios de uso. Sin embargo, ninguna de estas propuestas es perfecta, pues no son capaces de corregir enteramente los defectos de este tipo de soluciones, por lo que con cada iteración que se realiza al protocolo aparecen más y más de las mismas, a la par que también se descubren nuevos defectos. Ello ha provocado que, en la actualidad, Plasma sea el protocolo más extenso y complejo de todos debido a la importante cantidad de propuestas que contiene, siendo considerado a día de hoy una familia de alternativas.

1.3.2 State Channels.

La mejora de escalabilidad de los State Channels [6] se logra a través del uso de contratos multifirma con la capacidad de bloquear el estado de una sección de la cadena de bloques (podría, por ejemplo, ser el valor de una variable) durante la ejecución del mismo, creando

en el acto un canal para las transacciones entre los usuarios involucrados. Una vez el canal se encuentra establecido, los usuarios realizan transacciones o actualizaciones del estado bloqueado off-chain. Una vez las operaciones entre ellos hayan finalizado, deben comunicar de manera conjunta dicha decisión al contrato gestor del canal, que procederá a su cierre y a actualizar el estado bloqueado con el presente en la última transacción realizada entre los agentes involucrados.

La utilización de este protocolo supone el pago de dos transacciones en la L1, una para la apertura del canal y otra para el cierre del mismo, siendo todas las transacciones intermedias gratuitas. No obstante, estas transacciones están limitadas a actualizaciones de estado, independientemente de si estas están o no vinculadas a fondos económicos.

Este tipo de soluciones ofrecen prestaciones interesantes. Por un lado, la utilización de canales entre una serie concreta de usuarios ayuda a la seguridad y privacidad del sistema, además de lograr que las transacciones tengan una finalidad instantánea. No obstante, el protocolo requiere la participación activa de los usuarios del canal para el correcto funcionamiento del mismo, así como una predisposición de los mismos a actuar de manera honesta. En caso contrario, se debe recurrir a mecanismos de disputa que empeoran la experiencia de uso y la mejora de escalabilidad. En última instancia, la utilización de esta solución no siempre resulta conveniente a los usuarios debido al pago de las transacciones para la apertura y cierre del canal. A efectos prácticos, solo si los usuarios van a realizar numerosas interacciones entre ellos resulta conveniente recurrir al sistema.

1.3.3 Side Chains.

Estas soluciones se basan en la ideación y creación de nuevas Blockchains con la capacidad de comunicarse con la cadena principal, es decir, aquella que conforma la Layer 1. De implementarse, es trabajo del programador encargado gestionar cualquier aspecto de su funcionamiento interno, como pueden ser el mecanismo de consenso o los parámetros de los bloques. Esto también implica la necesidad de permitir la existencia de nodos que gestionan la cadena de manera descentralizada y que se encarguen de las funciones que típicamente se les asignan en una blockchain, como lo son la creación de bloques o la confirmación y procesamiento de transacciones.

La ventaja de las Side Chains [7] reside en el importante grado de libertad que existe para los responsables de su creación. Pueden implementar cualquier método o protocolo que consideren adecuado para aumentar las características que más quieran favorecer. En un contexto de lograda innovación y optimización, se pueden obtener rendimientos considerables. No obstante y como se puede suponer es,

junto con Plasma, la solución que más conocimiento y recursos requiere para su correcta planificación, dada la necesidad de crear y mantener una infraestructura considerable.

1.3.4 ZK Rollups.

Este protocolo pertenece a una familia de soluciones L2 conocida como “Soluciones Rollups”. Todos los miembros de esta categoría gestionan la información de la misma manera, siendo la principal diferencia entre ellos los datos que transmiten a la L1 y las pruebas criptográficas que emplean para validar los datos. Así pues, distinguimos la existencia de uno o varios árboles Merkle que controlan la información de los usuarios L2, además de unos usuarios especializados denominados “validadores”. Estos se encargan de recoger las transacciones realizadas por los usuarios convencionales y en función de los datos de las mismas, modificar los estados de los árboles Merkle. Acto seguido transmite una versión simplificada de las operaciones captadas a la red principal con objeto de dejar constancia de ellas, así como las raíces Merkle resultantes de la ejecución de las transacciones. En otras palabras, realizan una única transacción equivalente a varias.

La peculiaridad de ZK Rollup [8] reside en los métodos criptográficos que utiliza. En lugar de realizar *Fraud Proofs*, genera y entrega a la capa principal una prueba de conocimiento cero[9], en concreto, una versión no interactiva de las mismas[10]. A diferencia de las primeras, estas pruebas pueden ser verificadas fácilmente, demostrando con ello la veracidad de la información entregada, siendo innecesario, en consecuencia, recurrir a mecanismos de disputas así como a la comprobación activa por parte de los usuarios con objeto de detectar actividades maliciosas. El resultado es un protocolo con unas elevadas prestaciones de transacciones por segundo, reducida finalidad (ya que no se pueden dar disputas), seguro y con unas altas cotas de privacidad gracias a las características subyacentes a las pruebas de conocimiento cero. No obstante, presenta un inconveniente principal, siendo este la complejidad de las pruebas criptográficas, que se traduce en un significativo consumo de recursos. Además, también son difíciles de diseñar y generalizar, lo que ha relegado fundamentalmente el protocolo a casos de uso muy concretos caracterizados por la realización en exclusiva de transferencias entre usuarios.

1.3.5 Validium.

Perteneciente a la familia de los Rollups, Validium [11] se diferencia de ZK Rollup en que solo se tramita la prueba de

conocimiento cero generada, nunca ninguna expresión de los datos. Esto le permite lograr incrementos mayores en escalabilidad, pero a costa de perjudicar la seguridad del sistema debido a la falta de disponibilidad de datos, haciendo imposible acceder a dicha información en caso de error o de ataque. En el peor de los casos, los fondos de los usuarios podrían quedar bloqueados o perderse para siempre [12].

1.3.6 Optimistic Rollup.

El objetivo de Optimistic Rollup *[13]* es ofrecer una alternativa a ZK Rollup que permita computación general. Es decir, que no esté ligada a transferencias. Para ello, eliminan las pruebas de conocimiento cero y las reemplazan por “fraud proofs” al igual que las que emplea Plasma. Además de permitirle cubrir un espacio amplio de escenarios de uso, también supone un menor coste de recursos a la par que conservar las prestaciones de escalabilidad. No obstante, y como se ha mencionado en secciones anteriores, la utilización de estas pruebas criptográficas obliga asumir cierto grado de confianza y, ante la posibilidad de fraude, disponer de un método que permita la apertura de disputas, con la consecuente necesidad de que los usuarios comprueben las pruebas entregadas para garantizar la veracidad de las mismas. En caso de que se abra una disputa, el sistema verá su rendimiento empeorado, viéndose especialmente afectada la finalidad de las transacciones.

2. Escenario de uso.

2.1 Justificación.

Debido al amplio espectro de posibles implementaciones, se decide definir un escenario de uso con objeto de establecer una serie de requisitos fundamentales que el prototipo L2 debe ser capaz de satisfacer. De igual manera, su pretensión es, también, acercar el sistema que se diseñe a un contexto más real y que influya en las decisiones de diseño del prototipo.

2.2 Definición.

Existen numerosas empresas que interactúan entre ellas a través de Ethereum y que padecen los problemas de escalabilidad de la red. Realizar dichas interacciones a través de un protocolo L2 podría, en consecuencia, beneficiarles en gran medida. Así pues, para la elaboración del prototipo se ha decidido trabajar con un escenario de economía colaborativa. En este entorno, las empresas interesadas se registran y transaccionan con otras que también lo hayan hecho, obteniendo los beneficios de escalabilidad inherentes al protocolo en el proceso, representados estos tanto en una mayor cantidad de interacciones por segundo como, y especialmente, una reducción significativa en las tarifas asociadas.

En consecuencia, y a través de este entorno, las organizaciones serían capaces de reducir los costes funcionales de su infraestructura sin renunciar a las prestaciones que les ofrece Ethereum como tecnología.

2.3 Objetivos.

Permitir a las organizaciones poder interactuar entre ellas en un entorno Ethereum más adaptado a sus necesidades:

- Reducir las tasas a pagar por transacción, pudiéndose alcanzar valores de decenas de veces menores.
- Aumentar las transacciones por segundo que se podrían realizar en la Layer 1 de Ethereum. Se podrían alcanzar cerca de miles de transacciones por segundo.
- La reducción efectiva del valor de las tasas permitiría abstraerse en mayor medida del carácter volátil del Ether como cripto-divisa.

2.4 Actores humanos.

Los usuarios, presumiblemente organizaciones registradas, se definen como los distintos participantes del proyecto. Estos se corresponden con los usuarios finales y su propósito sería únicamente operar entre ellos realizando transacciones económicas de manera análoga a cómo se realiza en la Layer 1 de Ethereum. La interacción se realizaría a través de una API diseñada con tal fin.

3. Prototipo Layer 2.

3.1 Justificación.

Se ha escogido que el prototipo a desarrollar sea del protocolo ZK Rollup previamente expuesto. Esta decisión se ha tomado teniendo en cuenta varios factores.

- Es una implementación que no requiere el mantenimiento de una infraestructura tan extensa como Plasma o Side Chain.
- Gracias al uso de las pruebas de conocimiento cero, ofrece un importante grado de seguridad y de privacidad a los usuarios.
- No requiere establecer mecanismos de disputas.
- Se adapta correctamente al escenario de uso ideado.
- Es, junto con Optimistic Rollup, el protocolo Layer 2 que más interés ha suscitado en la comunidad.
- La eventual salida de Ethereum 2.0 beneficiará a los protocolos de la familia de los Rollups, que podrán alcanzar, incluso, mejores resultados [14].

El objetivo del prototipo será satisfacer las necesidades del caso de uso por un lado y, dada su naturaleza experimental, obtener la mayor mejora de escalabilidad posible.

3.2 Requisitos.

ZK Rollup es un protocolo con un destacable grado de personalización, puesto que no existe una serie de pautas concretas sobre cómo satisfacer los distintos requisitos que caracterizan al protocolo. Ello supone que la estructura y diseño recaen completamente en manos del programador interesado, que deberá elegir aquella que le resulte más conveniente. Independientemente de la implementación, todo prototipo ZK Rollup debe satisfacer los siguientes requisitos.

- Los usuarios deben haberse registrado en el sistema antes de poder utilizarlo.
- La gestión de la información de la Layer 2 debe realizarse a través de un árbol Merkle (o incluso varios).
- Las pruebas de conocimiento cero se deben de realizar sin interacción entre el verificador y el testigo. Existen, pues, dos alternativas: pruebas SNARK [15] y STARK [16].
- Las transacciones de los usuarios deben ser recogidas por, al menos, un validador.
- Las transacciones inválidas de los usuarios deben ser rechazadas (ya sea por insuficiencia de capital o por falsa autoría).
- Además de la prueba de conocimiento cero, también se debe tramitar datos de la Layer 2 a la Layer 1 con objeto de hacerlos accesibles a los usuarios. Ello en base a varias razones, siendo una de ellas disponer de un medio a través del cual comprobar el estado de la Layer 2, detectando con mayor facilidad acciones fraudulentas en caso de que no se de una correspondencia con el estado esperado. A modo

de ejemplo, un usuario podría no tener asociado el balance que, teóricamente, le corresponde.

- Junto a los datos, se deben almacenar las distintas raíces Merkle que se van generando al procesar el conjunto de operaciones tramitadas. A efectos prácticos, esto genera un histórico de hashes que, en combinación con la información anterior debería ser suficiente para reconstruir el árbol Merkle de resultar necesario.
- Debe existir, como mínimo, un contrato inteligente en la Layer 1 que haga de punto de acceso al sistema y que gestione la información necesaria.
- Debe existir, como mínimo, un mecanismo en la Layer 1 que verifique las pruebas de conocimiento cero.
- El sistema debe evitar cualquier acción de verificación si se entregan pruebas inválidas.
- La entrega de datos, así como de las pruebas criptográficas, solo puede ser posible para usuarios validadores.
- El árbol Merkle no debe ser modificado por ningún usuario sin autorización.

3.3 Adaptación al escenario de uso.

Registro de organizaciones:

1. Realizar transferencia de Ether al contrato comunicador on-chain.
2. Esperar por la realización de la transacción.
3. Adquirir cliente de la plataforma colaborativa.

Resultado: Organización registrada permanentemente en la plataforma colaborativa. Podrá operar en la misma con los fondos depositados o con los que reciba de otros usuarios.

Atendiendo al protocolo escogido, se deben dar soporte a las siguientes acciones para satisfacer las necesidades del escenario de uso propuesto.

Realización de transferencias:

1. Comprobar que la organización a la que se desea transferir forma parte de la plataforma colaborativa.
2. Realizar transferencia a través del cliente y con el wallet asociado.

Conflicto:

- En 1: Si la organización no forma parte de la plataforma.
  - Registrar la organización y asignarles los fondos que se desean transferir.
  - Impedir la operación.

Resultado: Transferencia efectiva de fondos entre dos organizaciones miembros.

Tabla 1: Escenario de uso - Registro de organizaciones

Tabla 2: Escenario de uso - Realización de transferencias

4. Arquitectura y diseño.

4.1 Diagrama de la arquitectura.

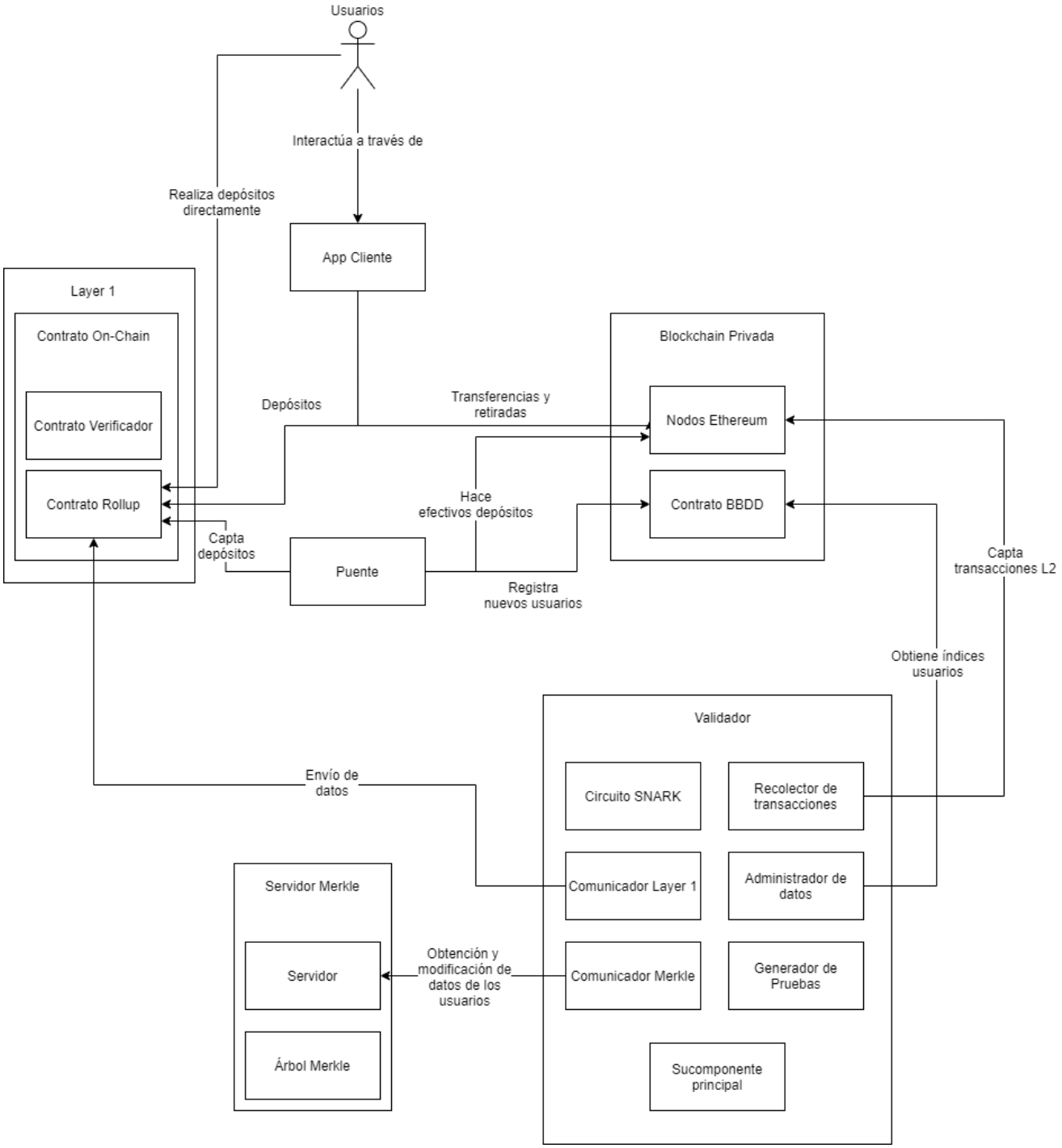

Figura 1. Diagrama arquitectura

Los usuarios interactúan con el sistema a través de una API dirigida a ello. Esta puede ser una aplicación adecuada al escenario de uso o una serie de métodos o pautas para poder utilizar el protocolo ZK Rollup. Desde la misma, los usuarios deberían ser capaces de realizar las distintas operaciones que admite el prototipo: transferencias, depósitos y retiradas, estas últimas con respecto a un contrato inteligente en la L1.

Un nuevo usuario que llegue al sistema debe realizar primeramente un depósito para poder interactuar con el mismo. Esto es así, ya que esta operación, además de asignar capital al usuario, lo registra en el protocolo. Se trata pues, de la inversión inicial que todo usuario debe realizar. Los usuarios registrados pueden interactuar entre ellos a través de la realización de transferencias en la L2. Estas, que a efectos prácticos son transacciones, son procesadas por la infraestructura L2 construida, que en el caso que nos atañe se corresponde con una blockchain privada. Uno o varios usuarios especiales denominados validadores recogen dichas transacciones y dejan constancia de los efectos de las mismas en un árbol Merkle ubicado en la L2. Este es una estructura de datos que contiene información sobre cada usuario registrado, por ejemplo, el capital que tienen disponible. Acto seguido, genera una prueba de conocimiento cero que acredite la validez de las transacciones realizadas y la tramita, junto a una versión simplificada de las transacciones, a la L1, concretamente al mismo contrato inteligente empleado por los usuarios para realizar depósitos.

A efectos prácticos, el validador está realizando una transacción L1 equivalente a cientos o miles de transacciones L2. El contrato utiliza la información de las transacciones para generar un histórico de las mismas, además de verificar la validez de la prueba criptográfica entregada. Si esta es incorrecta, el validador no debería poder realizar cambios en el estado del contrato. Si la prueba es correcta y alguna de las transacciones entregada se corresponde con una retirada, el contrato realiza el envío correspondiente de Ether al usuario que ha solicitado la operación.

4.2 Decisiones de diseño.

Además de decisiones de diseño que afectan a la definición y estructura de ciertos componentes, se han tomado otras cuyo efecto afecta al sistema en su conjunto.

En primera instancia, se encuentra la decisión de utilizar una blockchain privada como infraestructura L2. En términos teóricos, la tecnología empleada para construir este componente puede ser cualquiera siempre que sea capaz de rechazar transacciones L2 inválidas y actualizar adecuadamente los balances de los usuarios registrados en el sistema. Una blockchain privada cumple estos requisitos de una manera sencilla. Los

usuarios no podrán realizar transacciones con un valor mayor que el capital que posean y, la necesidad de firmar cada transacción garantiza la autoría de estas. Además, la actualización de los balances se gestiona por la propia cadena de bloques siendo innecesario recurrir a elementos adicionales. También, otorga otras ventajas que benefician especialmente a la naturaleza prototípica del proyecto. Por ejemplo, se puede utilizar como base de datos de resultar necesario, evitando así recurrir a componentes adicionales que implementen algún modelo de base de datos. Desde el punto de vista de los usuarios del sistema también puede resultar conveniente, pues pueden emplear las mismas herramientas que utilizaban para operar con la L1 fácilmente en la L2. En el caso de *wallets*, como Metamask [17], bastaría con incluir la nueva red, mientras que con posibles scripts sería necesario cambiar la red con la que operan. No obstante, la blockchain privada también presenta ciertos inconvenientes, especialmente desde el punto de vista de la seguridad. En términos generales, y dada la necesidad de que la red sea accesible al público, resulta necesario securizar adecuadamente la cadena de bloques con objeto de evitar posibles ataques. Sin embargo, la naturaleza experimental del proyecto nos permite ignorar esta necesidad al no exponerse al público y no haber activos reales.

El nivel de centralización del sistema, al igual que el grado de confianza de los validadores que se presupone son dos de los factores más significativos a definir en todo protocolo ZK Rollup. Un elevado nivel de centralización garantiza a la entidad responsable un mayor control sobre su sistema y el uso que se le da al mismo, evitando en la mayoría de los casos la posibilidad de que se confíe en usuarios deshonestos. No obstante, ello puede suponer la desconfianza de ciertos sectores de usuarios e incluso, puede quien considere que el enfoque va en contra de los principios de las cadenas de bloques. Un elevado nivel de descentralización permitiría el ahorro de recursos al relegar las acciones del validador a usuarios externos. Por su parte, suponer un elevado grado de confianza reduce la cantidad de comprobaciones que se tienen que realizar en la Layer 1 para comprobar la integridad de la información que se tramita por parte de los validadores, obteniendo en el proceso mejores resultados de escalabilidad. No obstante, por el contrario, expone el sistema a posibles ataques si un usuario no se comporta como se confía que haga. Por lo general, suponer un elevado grado de confianza solo es posible si se trabaja en un entorno centralizado. Para el caso que nos atañe, el prototipo será centralizado y se confiará en el buen hacer de los validadores, reduciendo las comprobaciones únicamente a restringir el acceso a los métodos de los componentes de la Layer 1 a los usuarios válidos, además de todas aquellas básicas y fundamentales del protocolo.

Existen dos tipos de pruebas de conocimiento cero válidas: SNARK y STARK. Para el prototipo se ha decidido operar con SNARK dada la mayor rapidez y coste reducido de verificar las pruebas generadas, cuestión que, al realizarse en la L1, supondría un mayor coste de gas por transacción de cara a los usuarios.

4.3 Listado de componentes.

A continuación se expondrá un listado de los diferentes componentes del sistema, dividiéndolos en subcomponentes de ser necesario. Se corresponde con una visión detallada de los elementos presentes en el diagrama de la arquitectura presentada en el apartado *4.1 y Figura 1.* Este listado incluye ciertos aspectos técnicos que serán adecuadamente explicados en el apartado *5*.

Contrato On-Chain			
Categoría	Layer 1	**Lenguaje**	Solidity
Subcomponente	**Requisitos**		
Contrato Rollup	Permitir realizar depósitos a los usuarios.		
	Permitir el registro de nuevos usuarios.		
	No debe permitir más registros del máximo permitido		
	Gestiona el acceso a los métodos a los usuarios válidos.		
	Debe mantener una lista de los validadores actuales.		
	Se encarga de generar el histórico de los datos.		
	Debe gestionar las raíces Merkle.		
	Debe permitir la entrega de datos y de la prueba SNARK.		
	Debe realizar retiradas cuando corresponda.		
Verificador	Debe aceptar pruebas SNARK válidas.		
	Debe rechazar pruebas SNARK inválidas.		

Tabla 3: Componentes - Contrato On-Chain

Puente			
Categoría	Layer 2	**Lenguaje**	NodeJS
Requisitos			
Capta los depósitos de la Layer 1.			
Recrea los depósitos de la Layer 1 en la Layer 2.			
Dependencias técnicas		**Detalles**	

Web3js	Permite la interacción con Ethereum. Usado para detectar eventos de depósitos en la Layer 1 y operar en la Layer 2.

Tabla 4: Componentes - Puente

Validador			
Categoría	Layer 2	**Lenguaje**	NodeJS
Subcomponente	**Requisitos**		
Recolector de transacciones	Captar transacciones Layer 2 directamente de la Blockchain privada.		
	Comprobar que los usuarios estén registrados (ante posible ataque).		
	Formatea las transacciones L2 a una versión resumida.		
	Obtiene los índices de las hojas Merkle asociados a cada usuario.		
	Almacena las transacciones formateadas hasta que sean requeridas por el subcomponente central.		
	Agrupa las transacciones L2 en conjuntos.		
Generador de pruebas	Debe generar pruebas SNARK dada una entrada válida.		
	Debe obtener la expresión hexadecimal necesaria para la verificación On-Chain de la entrada y la prueba generada.		
Administrador de datos	Debe obtener los índices asociados a los usuarios registrados.		
Circuito SNARK	Define el procedimiento para la obtención de la prueba SNARK.		
Comunicador Layer 1	Permite enviar los datos de las transacciones L2 al contracto On Chain.		
	Permite enviar las pruebas SNARK generadas por el validador al contrato On-Chain.		
Comunicador Merkle	Permite comunicar al contrato On-Chain de las retiradas a realizar.		
	Debe contactar con el servidor Merkle para modificar el árbol Merkle.		
Subcomponente principal	Debe contactar con el servidor Merkle para obtener las raíces del árbol.		
	Debe contactar con el servidor Merkle para obtener el estado de las hojas.		

	Debe gestionar al resto de subcomponentes.
	Debe comunicar a los subcomponentes entre ellos.
	Recoge las transacciones del “Recolector de transacciones”
	Decide qué datos entregar a la L1.
	Obtiene las retiradas a realizar en función de las transacciones procesadas.
	Modifica el árbol Merkle haciendo uso de “Comunicador Merkle”.
Dependencias técnicas	**Detalles**
Web3js	Permite la interacción con Ethereum. Usado para la comunicación con la Layer 1 y para la recolección de transacciones de la Layer 2, así como para la obtención de los índices de los usuarios.
Circom	Permite la construcción de circuitos SNARK a través de un DSL.
SnarkJS	Permite la generación de pruebas SNARK dada una entrada y un circuito escrito en Circom compilado.
Axios	Permite la realización de peticiones HTTP. Empleado para comunicarse con el “Servidor Merkle”.

Tabla 5: Componentes - Validador

Blockchain privada			
Categoría	Layer 2	**Lenguaje**	Go y Solidity
Subcomponente	**Requisitos**		
Nodos (2) Geth de Ethereum	Deben constituir una red PoA.		
	Deben generar nuevos bloques cada segundo.		
	Deben permitir la interacción de los usuarios habilitando la interfaz HTTP.		
Contrato BBDD	Debe almacenar los usuarios registrados		
	Debe asignar a cada usuario registrado una hoja del árbol Merkle.		
	Debe permitir la adición de nuevos usuarios.		
	Debe permitir la lectura de datos.		

	Debe impedir más registros de los permitidos.
Dependencias técnicas	**Detalles**
Geth	Cliente de Ethereum escrito en Go. Necesario para la creación de los nodos de las cadenas de bloques, creando en consecuencia la red.

Tabla 6: Componentes - Blockchain privada

APP Cliente			
Categoría	No corresponde	**Lenguaje**	NodeJS y JS
Requisitos			
Debe permitir a los usuarios realizar depósitos en el contrato On-Chain.			
Debe permitir a los usuarios interactuar entre ellos en la Layer 2.			
Dependencias técnicas		**Detalles**	
Web3js		Permite la interacción con Ethereum. Utilizado para la comunicación con la Layer 1 y la Layer 2.	
Webpack		Utilizado para permitir la utilización de módulos npm en el navegador.	
HTML		Estructura de la web.	
CSS		Diseño de la web.	

Tabla 7: Componentes - APP Cliente

Servidor Merkle			
Categoría	Layer 2	**Lenguaje**	Rust
Subcomponente	**Requisitos**		
Árbol Merkle	Debe permitir la construcción de un árbol Merkle vacío dada la profundidad del mismo.		
	Debe permitir la obtención de las raíces del árbol.		
	Debe permitir que las hojas del árbol contengan datos.		
	Debe permitir la modificación de las hojas del árbol.		
	Debe calcular adecuadamente los hashes del árbol.		
	Debe permitir obtener el estado de sus hojas.		

Servidor	Debe crear una instancia de árbol Merkle de profundidad 16.
	Debe establecer una manera de obtener las raíces del árbol a través de HTTP.
	Debe establecer una manera de obtener el estado de las hojas del árbol a través de HTTP.
	Debe establecer una manera de modificar las hojas del árbol.
	Antes de finalizar su ejecución debe finalizar las peticiones HTTP pendientes.
	Debe definir el algoritmo a emplear para la creación de los hashes del árbol (se ha escogido Mimc7).
	Debe definir la estructura de datos que contendrán las hojas del árbol.
Dependencias técnicas	**Detalles**
Warp	Servidor HTTP escrito en Rust y basado en Tokyo.
Mimc7	Algoritmo empleado para la generación de Hashes. Optimizado para la generación de pruebas SNARK.

Tabla 8: Componentes - Servidor Merkle

5. Desarrollo de componentes.

5.1 Servidor Merkle.

5.1.1 Árbol Merkle.

El protocolo ZK Rollup emplea árboles Merkle, en lugar de otras estructuras de datos, para gestionar la información referente a sus usuarios debido a la propiedad de los mismos de ligar una gran cantidad de datos a un único hash, resultando sencillo comprobar la integridad de la información. Si se posee una raíz Merkle, y los datos que, en teoría, conforman el árbol, se puede construir el mismo y comprobar si las raíces coinciden. En caso negativo, se puede afirmar que ha habido una modificación de los datos. Esta característica es especialmente útil en las pruebas de conocimiento cero para garantizar que el validador está trabajando con los datos correctos, así como para el proceso de verificación que se realiza en la L1.

Con respecto a la implementación del prototipo, este se ha construido desde cero, sin hacer uso de librería alguna. Esto es así, ya que se le quería otorgar ciertas funcionalidades que, en librerías de Rust ya existentes, no estaban presentes en su totalidad. El enfoque seguido ha sido generalista: se ha diseñado el árbol Merkle para que pueda emplearse con múltiples estructuras de datos y con distintos algoritmos de generación de hashes. Para ello, se han recurrido a los *traits* de Rust, funcionalidad similar a las clases abstractas de otros lenguajes como C++. Fundamentalmente se define una serie de métodos que toda estructura que implemente el *trait* debe declarar. Si un tipo de datos no implementa los *traits* correspondientes, el árbol lo rechazará. En concreto se han definido dos *traits*.

- **Element**: Toda estructura de datos que desee almacenarse en los nodos del árbol debe implementar este *trait*. Define métodos para expresar el tipo de datos en un *array* de 32 bytes, así como para, a partir del *array*, obtener el dato correspondiente. Se han escogido bytes al resultar la manera estándar en la que trabajan la mayoría de los algoritmos de encriptación.
- **Algorithm**: Debe implementarlo cualquier estructura cuya función sea generar los hashes del árbol Merkle. Define métodos para la carga de datos y la obtención del hash correspondiente a los mismos.

Definidos dos tipos de datos que implementen los traits anteriores, estos pueden utilizarse para crear una instancia del árbol válida. En el caso de la estructura a contener en los nodos, es necesaria que esta también implemente el *trait Default*, con objeto de aportar un valor válido a las hojas tras la construcción del árbol, que se

considerará vacío.

Las funcionalidades del árbol son más limitadas, ya que están dirigidas exclusivamente a satisfacer las necesidades de un ZK Rollup. Así pues, este puede construirse indicando la profundidad del mismo, generando en el proceso un árbol vacío del tamaño deseado. Adicionalmente, define métodos para la obtención de la raíz Merkle y de los datos que almacenan las hojas, así como su modificación. La raíz Merkle está disponible en todo momento y no es necesaria calcularla con cada petición. Esto es así, ya que el árbol está programado para actualizar los hashes de los nodos tras cada modificación de las hojas. La actualización solo afecta a los nodos afectados por el cambio, evitando así cálculos innecesarios.

5.1.2 Servidor.

Con objeto de permitir y controlar el acceso al árbol desde el resto de componentes se ha construido un servidor Rust encargado de la interacción con el mismo. Esta se ha hecho a través de la librería *warp* [18], basada en *Tokyo* [19].

A efectos prácticos el servidor es un proceso multihilo. Por un lado, *warp* se encarga de generar una *piscina* de hilos para tratar con las distintas peticiones HTTP que se reciban. Por otro, el árbol Merkle se instancia desde un segundo hilo originado en el principal. Para la comunicación de los hilos de la *piscina* con este último se ha implementado un sistema de paso de mensajes. Fundamentalmente se han definido dos canales de comunicación. Uno permite el envío de mensajes de los hilos que atienden peticiones HTTP al del árbol y, el otro, permite el proceso contrario. Cuando se recibe una petición, se envía un mensaje al hilo del árbol con la operación que debe realizar. Este responderá con un mensaje de confirmación si la acción solicitada se ha realizado con éxito con el resultado de la petición si esta ha sido de lectura; o bien, con un error en caso de que se haya intentado realizar una operación inválida. En este último caso, el hilo que atiende la petición informa del error al quien la haya realizado.

Cuando el servidor va a suspenderse, se notifica desde el hilo principal al del árbol a través de un mensaje para que detenga su actividad. Este resolverá todas las peticiones pendientes hasta que detecte la solicitud de parada. Tras esto, el proceso finaliza.

Adicionalmente, es también responsabilidad del servidor definir la estructura de datos que contendrán las hojas del árbol así como el algoritmo que se empleará para la generación de hashes. En el primer caso, se ha escogido una estructura simple, que contendrá solamente la información sobre el balance de cada usuario, cuyo valor máximo se ha decidido de acuerdo al escenario de uso, fijándolo en 2^{80}wei (1.208.926 unidades de Ether). Se considera que este valor es superior al que las organizaciones mantendrán activamente en el sistema. Por su parte, se ha escogido Mimc7 [20] como algoritmo de encriptación. Se trata de un algoritmo optimizado para la generación de pruebas SNARK que, no obstante, define sus hashes a través de

BigInt, por lo que resulta necesario encargarse de la conversión de estos a bytes al implementar el *trait Algorithm*. Es fundamental que el árbol emplee el mismo algoritmo que el circuito encargado de la generación de pruebas. Al ser este último factor el más limitante, se ha escogido un algoritmo que le sea más favorable.

El servidor responde a las siguientes peticiones HTTP:

Petición	Ruta	Método HTTP	Acción
Root	/root	GET	Obtiene la raíz Merkle en bytes.
RootString	/rootString	GET	Obtiene la raíz Merkle en formato string. Este contiene el hash BigInt generado por el algoritmo.
Index	/index/<usize>	GET	Obtiene los datos de la hoja con índice <usize>
Modify	/modify/<usize>/<string>	POST	Modifica una hoja con índice <usize>. <string> contiene el nuevo balance de la hoja (el balance puede tener un valor mayor que el máximo que admiten los datos numéricos de Rust).

Tabla 9: Servidor Merkle - Peticiones HTTP

5.1.3 Consideraciones - Tamaño del árbol.

La profundidad del árbol Merkle establece la cantidad máxima de usuarios que pueden registrarse en el sistema. Este parámetro está íntimamente ligado al escenario de uso. Se debe escoger un valor lo suficientemente elevado como para satisfacer la demanda prevista, pero no en exceso pues afectaría negativamente a las mejoras de escalabilidad. Atendiendo a estos hechos, se ha optado por un árbol de 16 de profundidad. Ello establece un máximo de 2^{16}hojas, es decir, 65536 usuarios máximos. Suponiendo que el escenario de economía colaborativa está especialmente dirigido a organizaciones (además interesadas en Ethereum), dicho valor parece improbable de alcanzarse, al menos, en un futuro cercano.

Una posible pregunta que puede surgir es por qué la necesidad de especificar límites, tanto a la cantidad de usuarios como al balance que estos pueden tener. Aunque es cierto que sería interesante admitir un número infinito (en su defecto muy grande) de ellos con objeto de evitar situaciones en las que alcanzar el límite establecido evite ciertas interacciones con el sistema, ello perjudicaría la mejora de escalabilidad. Esto es así, puesto que el diseño del árbol influye en el formato de las transacciones L2, aquellas que se procesan en amplios conjuntos a la L1 en forma de una única transacción. Estas

reemplazan los campos de las transacciones L1 [21] por equivalentes de menor tamaño (en bytes) de la L2. Al ocupar un menor peso cada transacción, su coste en gas [22] también se reduce. Así pues, y a modo de ejemplo, las transacciones L2 poseen un campo “from” que indica quien ha realizado la operación. En la L1, su valor sería una dirección de 32 Bytes. En la L2, en cambio, es un número que indica la hoja del árbol Merkle asociada al usuario en sí. En el caso que nos atañe, para expresar todas las posibles hojas de un árbol de profundidad 16 solo es necesario emplear un máximo de 2 bytes, significativamente menos que los empleados en la L1. Si modificáramos el árbol para que aceptase la mayor cantidad posible de usuarios podríamos acabar generando, incluso, transacciones con un peso superior a las generadas en la L1 para operaciones equivalentes. El mismo razonamiento se puede hacer para los límites del balance. Así pues, en la práctica interesa que los límites sean lo más reducidos posible, pero siempre satisfaciendo las necesidades del escenario de uso.

A continuación se expone una tabla comparativa entre ambos tipos de transacciones.

Parámetro	Layer 1 (Bytes)	Layer 2 (Bytes)	Explicación
Nonce	~3	0	En la L2 no es necesario dicho campo debido a la gestión interna de la blockchain privada.
GasPrice	~8	0	Se gestiona internamente por la blockchain privada. Igualmente, el sistema no tendrá coste asociado dado a su naturaleza experimental.
Gas	3	0	Este parámetro no expresa ninguna información relevante sobre los efectos de la transacción, por lo que puede ser removido.
To	20	2	En la L2 podemos identificar cada usuario en función de su índice asociado en el árbol Merkle. Siendo este de 16 de profundidad, son necesarios 2 bytes.
Value	~9	10	Son necesarios 80 bits (10 bytes) para representar el balance máximo que se podrá transferir.
Signature	65	0	La autoría de las transacciones se gestiona por la blockchain privada.
From	0	2	Al igual que “To”, se puede identificar a cada usuario por su índice en el árbol Merkle. En la L1 se obtiene a partir de la firma.
Total	~108	14	Esta reducción es el factor determinista, junto con el diseño del contrato inteligente de la L1, de la mejora de escalabilidad.

Tabla 10: Servidor Merkle - Comparativa transacciones

5.2 Blockchain privada.

5.2.1. Nodos Geth de Ethereum.

La red privada puede parametrizarse como más conveniente resulte al programador. En el caso que nos concierne se corresponde con una red que implementa el mecanismo de consenso PoA [23], pues evita el gasto energético y apenas requiere recursos al no implementar minería. Es necesario establecer un mínimo de dos nodos, pues en este tipo de redes el nodo que se encarga de generar nuevos bloques no puede mantener su interfaz RPC abierta dado el peligro que ello supondría para los usuarios validadores del protocolo, que deben desbloquear sus cuentas para crear nuevos bloques. Así pues, existe un primer nodo encargado de tal actividad y un segundo que supone el punto de entrada a la red a través del protocolo JSON-RPC. Dada la naturaleza experimental del proyecto, el gas no tiene coste asociado y el límite por bloque es de 15.000.000 de unidades. Además, se ha configurado al nodo que genera bloques para que lo haga cada segundo.

Como se mencionó con anterioridad, el empleo de una cadena de bloques como infraestructura Layer 2 permite automatizar muchos de los procesos de verificación que se deben realizar en esta capa. No obstante, también supone otros inconvenientes, especialmente en cuanto a seguridad se refiere y en consumo de recursos (cada bloque, vacío o no, requiere memoria secundaria). Debido a ello, sería oportuno abrir una línea de investigación sobre posibles alternativas de cara al desarrollo de un producto no experimental.

5.2.2. Contrato BBDD.

Se puede emplear la cadena de bloques privada para almacenar información sin necesidad de recurrir a bases de datos adicionales. En este caso, se ha desplegado un contrato inteligente que almacena el conjunto de usuarios registrados en el sistema y asocia a cada uno de ellos el índice de la hoja del árbol Merkle que le corresponde. Cuando se detecta un depósito en la L1, el componente “Puente” lo detecta y comprueba si el usuario responsable se encuentra ya registrado examinando la información de este contrato. En caso negativo, lo registra, siéndole asignado un índice automáticamente. Si por el contrario el usuario ya se encontraba registrado, no se realiza ninguna interacción adicional con el contrato.

De resultar necesario, otros componentes del sistema también pueden consultar la información registrada, aunque su modificación está restringida únicamente a usuarios administradores al tratarse de información de vital importancia.

5.3 App Cliente.

La funcionalidad de este componente es ofrecer a los usuarios una manera de poder interactuar con el sistema, tanto con la parte del mismo ubicada en la Layer 1, como con la infraestructura Layer 2. Se trata de una web de navegador ofrecida a través de un servidor Express. La web permite a los usuarios realizar depósitos, transferencias a otros usuarios y retiradas de capital. Para cada una de las opciones, el usuario debe introducir los datos pertinentes. Además, el sistema requerirá en todo momento que el usuario posea un *wallet* disponible. Si en algún momento de la interacción del usuario con la web se diera algún problema relacionado con este último, el usuario sería notificado apropiadamente de los mismos.

Debido a las limitaciones actuales de los *wallets*, es responsabilidad del usuario cambiar, cuando corresponda, la red a la que está conectado. Esto es así debido a la no existencia de un método oficial que permita cambiar las redes de manera automática. El cambio es necesario debido a la distinción de la blockchain de cada capa como redes independientes. Sin estar conectadas a las mismas las operaciones no se podrán realizar.

La operación de depósito supone realizar una llamada al método correspondiente del contrato On-Chain, enviando además una cantidad de Ether (expresada en wei) válida (mayor que 0 pero menor que 2^{80}). Para realizarla, el usuario debe conectarse primero a la L1.

Por su parte, transferencias y retiradas se corresponden con operaciones de la L2. La primera supone el envío de una cantidad arbitraria de Ether a un segundo usuario. Se debe tener en cuenta que, al ser la infraestructura una blockchain privada, esta permitirá enviar ether a cualquier dirección, se corresponda la misma con la de un usuario o no. Si se enviase fondos a una dirección no registrada, dichos fondos serían irrecuperables. Dada la naturaleza prototípica del proyecto, no se han establecido sistemas de control y/o recuperación de ningún tipo. Por su parte, las retiradas son interpretadas internamente como transferencias que se producen del usuario a una dirección privilegiada. Esta última se especifica por parte de los desarrolladores del sistema. Cuando el recolector de transacciones del validador detecta una transacción cuyo destino es esta dirección, la interpreta como una operación de retirada.

5.4 Puente.

Los depósitos se deben realizar a nivel de la Layer 1, pero por definición no existe manera alguna de que un contrato inteligente se pueda comunicar con el exterior. Ello supone que, tras un depósito, no existiría manera de comunicarlo directamente al encargado en la Layer 2. Para evitar esta situación se crea este componente, que se encarga de escuchar constantemente a los eventos de depósito que emita el contrato On-Chain cada vez que se realiza esta operación. De los mismos extraerá el usuario y la cantidad que ha depositado. Acto seguido realizará una transferencia en la L2 equivalente al depósito. La dirección que realiza la transacción será la de un usuario privilegiado, en este caso la misma a la que los usuarios deben enviar fondos para efectuar retiradas. Cuando el recolector de transacciones

del validador capte transacciones en las que el emisor es esta dirección sabe que deben interpretarse como depósitos.

El empleo de este sistema aporta, además, ventajas importantes. Los eventos que se producen por el contrato On-Chain quedan permanentemente registrados en la cadena de bloques y pueden recuperarse en cualquier momento. Ello supone que, si en algún momento se produce un fallo en la L2, los depósitos que los usuarios hayan realizado mientras tanto no se perderían, ya que podrían recuperarse una vez el sistema vuelva a estar operativo.

5.5 Validador.

5.5.1. Recolector de transacciones.

Este subcomponente se comunica con la cadena de bloques privada y detecta cada nuevo bloque que esta origina, comprobando en el proceso si este posee transacciones registradas. En caso afirmativo, las recupera de la cadena de bloques y les da el formato de 14 bytes expuesto en la sección *5.1.3.* Estas se almacenan en un *array* dinámico de NodeJS. Durante el formateo de las transacciones, el sistema comprueba de qué tipo son. En caso de que una de las direcciones de las transacciones sea la dirección del usuario privilegiado, entonces pasará a considerarse como un depósito o una retirada en función de si esta aparece en el campo de origen o de destino.

Con objeto de formatear adecuadamente las transacciones, el subcomponente requiere acceder al registro de usuarios registrados en la Layer 2. Este acceso no se realiza directamente, sino que utiliza la funcionalidad del “Administrador de datos” para tal fin. Además, se ha implementado una funcionalidad similar a un caché. Cada vez que se solicita el índice asociado a un usuario, el sistema lo registra con objeto de, en caso de volver a ser necesario, no recurrir de nuevo al “Administrador de datos”, reduciendo en consecuencia la latencia. Dicho objeto se almacena en memoria secundaria en un fichero JSON en caso de que se interrumpa al validador. Cuando este vuelva a funcionar, carga los datos de ese mismo fichero. Ello supone que, en un principio, si el sistema alcanza su máximo de usuarios, eventualmente se poseería un caché con la información de cada uno de ellos, no volviendo a ser necesario recurrir al “Administrador de datos”.

Antes de proceder con su actividad, el “Recolector de transacciones” carga, de un segundo fichero JSON, el último bloque del que se recogieron transacciones. Acto seguido, comprueba si este se corresponde con el bloque actual de la cadena de bloques. Si la respuesta es negativa, analiza todos los bloques existentes hasta alcanzarlo recolectando y formateando todas las transacciones que encuentre en los mismos. Antes de suspender al validador, se modifica el fichero JSON con el índice del último bloque procesado. Este comportamiento garantiza que, en caso de que falle el validador, las

transacciones L2 que los usuarios hayan realizado mientras tanto sigan siendo posibles de recolectar.

Así mismo, es este subcomponente quien se encarga de gestionar las condiciones de parada. Existe una estructura de datos, a efectos prácticos un vector, donde cada índice contiene un valor equivalente a cuántas transacciones puede recoger el "subcomponente central". Si, por ejemplo, se alcanza el límite de transacciones (más información sobre las condiciones de envío en el apartado *5.5.4*), supongamos 100, entonces la primera posición del vector tendría asignado este valor. Si, por el contrario se alcanza el límite de retiradas, supongamos 10, pero solo después de 20 transacciones, entonces en lugar de 100 aparecerá un 20 en el mismo índice del vector. Cada vez que el mencionado subcomponente recolecta transacciones, el primer elemento del vector se pierde y el primer *array*, que contiene las transacciones en su conjunto, ve su tamaño reducido convenientemente. En caso de que no se haya alcanzado aún alguna de las condiciones de envío, entonces el vector estará vacío. Por el contrario, si se alcanzan varias veces antes de que se recojan las transacciones, entonces el vector tendrá una longitud mayor.

5.5.2. Administrador de datos.

Su funcionalidad es muy reducida. Se dedica exclusivamente a realizar consultas al "Contrato BBDD" presente en la blockchain privada. Se corresponde, pues, con el único método que posee el validador para poder obtener la información contenida en el mismo. En caso de que se solicite la obtención del índice asociada a una dirección no registrada, el subcomponente notificará de dicho hecho. Dependerá de quien haya solicitado los servicios del administrador determinar cómo actuar. En concreto, en la implementación actual, solamente el "Recolector de transacciones" hace uso del "Administrador de datos".

En caso de darse, esta situación puede deberse a dos motivos. El primero es que un usuario no deseado haya conseguido infiltrarse en el sistema. El segundo, es que un usuario haya realizado una transferencia a una dirección no registrada. Aunque ambos casos son diferentes, el "Recolector de transacciones" responde de igual manera a ambos: emitiendo un error que, en el "subcomponente central" se procesa provocando el cierre del proceso.

5.5.3. Comunicador Merkle.

Permite al validador comunicarse con el árbol Merkle con objeto de obtener información del mismo y/o modificarlo. Se trata, a efectos prácticos, de un módulo con numerosos métodos definidos que permiten realizar las peticiones HTTP correspondientes haciendo uso de los métodos que el módulo "Axios" de NodeJS ofrece para ello. En concreto, se implementan métodos para solicitar la raíz del árbol Merkle y los datos de las hojas. También se puede modificar estas últimas.

5.5.4. Subcomponente central.

Este subcomponente se encarga de gestionar a los anteriores logrando así la funcionalidad deseada. No obstante, resulta necesario destacar ciertas decisiones de diseño para poder explicar adecuadamente el funcionamiento de este componente, así como determinados aspectos fundamentales del sistema en su conjunto que aún no han sido especificados.

El validador es el encargado de enviar la información a la L1. Dicha información, como se ha destacado hasta ahora, se corresponde con las pruebas criptográficas pertinentes y el conjunto de transacciones L2. Aunque este enfoque es correcto, el prototipo se comporta de manera diferente con objeto de lograr mejores resultados de escalabilidad y dado el elevado nivel de confianza que se supone. Así pues, no se envían las transacciones L2, sino los deltas de estado de los usuarios afectados por dichas transacciones. Es decir, el estado resultante para un usuario X después de efectuarse Y cantidad de transacciones donde al menos una de ellas le afectaba. Trabajar con deltas nos permite ampliar en gran medida la cantidad máxima de transacciones que podemos procesar. Este enfoque permite que, en situaciones en las que solamente un subconjunto reducido de los usuarios están activos en el sistema, se pueda procesar una cantidad mayor de transacciones. A modo de ejemplo, supóngase dos usuarios que solamente interactúan entre ellos y que realizan X transacciones. En el enfoque original, se enviarían a la L1 X transacciones L2 equivalentes. Con el enfoque de deltas, solo se enviarían dos, obteniendo la misma cantidad de transacciones por segundo con un coste de gas mucho más reducido. A efectos teóricos, sería incluso posible ralentizar el envío de información hasta que no se alcance una cantidad concreta de deltas. Estos, sin embargo, no son siempre ventajosos frente a las transacciones simplificadas, pues presentan dos problemas principales. El primero solo se da en ciertas situaciones, en concreto en aquellas en las todas las transacciones se dan entre usuarios sin que ninguno de ellos se repita. En dicho caso, X de estas transacciones generarían 2X deltas, obteniéndose una mejora de escalabilidad más reducida. El segundo inconveniente no es circunstancial y se corresponde con la pérdida de información de cara al usuario. Mientras que las transacciones L2, una vez publicadas en la L1, permiten a cualquier persona comprobar que operaciones exactas han ocurrido, con los deltas solo serían capaces de ver los resultados de dichas operaciones, siéndoles desconocido cómo se ha llegado a ellos. Esta situación, que podría no resultar conveniente en ciertos escenarios de uso, no resulta relevante para el prototipo debido a la pretensión del mismo de alcanzar la mayor cantidad posible de transacciones por segundo, hecho que se considera más probable con la utilización de deltas. Con respecto al formato de los deltas, estos solo están formados por el índice identificador del usuario (2 bytes) y por su balance en el árbol Merkle (10 bytes).

Otra cuestión de interés es determinar las condiciones bajo las

cuales los validadores envían la información a la L1. La idea más básica sería realizar el trámite al alcanzarse cierta cantidad de transacciones en la L2 y, aunque este planteamiento es correcto, no cubre todas las distintas situaciones que se pueden dar. Esto es así, porque no todas las transacciones se pueden considerar iguales. Depósitos, transferencias y retiradas requieren todos que se realicen acciones distintas y, aunque a nivel de la L2 un depósito y una transferencia se pueden considerar como iguales, no ocurre así con las retiradas. Cuando el contrato inteligente de la L1 recibe una orden de retirada desde la L2, este debe realizar una transacción L1 (21000 de gas) que envíe fondos del contrato al usuario en cuestión. A ese coste de gas habría que sumarle los pertinentes a las computaciones internas que realice el contrato para procesar la información. Ello supone que una retirada es bastante más costosa que las otras dos operaciones. El orden dependerá del diseño del sistema y de la implementación del contrato inteligente. Como el prototipo trabaja con deltas, la primera restricción, en teoría, no resulta útil, ya que puede acabar reduciendo la capacidad máxima del sistema. En su lugar, habría de reemplazarse por una cantidad concreta de deltas. Dicha cantidad X podría alcanzarse, como mínimo, tras X/2 transacciones. En otros casos, podría no alcanzarse nunca. Debido a este factor de incertidumbre y a la imposibilidad de trabajar con él, se debe seguir considerando un máximo de transacciones, que no obstante podrá ser varias veces superior al máximo posible en un sistema convencional; para el caso que nos concierne, se ha decidido la cantidad de 65000. Por tanto, en el prototipo desarrollado existen tres criterios de envío; se alcanza el límite de transacciones, deltas o retiradas. El valor de los últimos depende de la implementación del contrato inteligente, en concreto, de cuantos de estos elementos y operaciones puede gestionar de manera consecutiva sin sobrepasar el límite de gas establecido en la L1. Experimentalmente, a través de la realización de un *benchmark* durante la fase de pruebas del prototipo, se ha calculado que estos límites son de 48000 y 120 respectivamente.

Presentadas estas cuestiones, a continuación se detallará el funcionamiento del subcomponente. Primeramente, inicializa instancias del “Recolector de transacciones” y del “Comunicador L1 (más información en el apartado *5.5.6*)”. Acto seguido, y en función de una función temporal, consulta al primer subcomponente si se ha llegado a una de las condiciones de envío y, en caso afirmativo, qué cantidad de transacciones debe recoger. La función temporal devuelve la cantidad a esperar comprendida entre un mínimo y un máximo, ambos incluidos. El valor obtenido dependerá de la cantidad de transacciones que existan en el recolector. Cuanta más cercana sea esta a la cantidad de transacciones máximas que soporta el sistema, menor será la cantidad de tiempo de espera. De esta manera se obtiene un sistema que, ante una elevada demanda, actúa más rápido.

Cuando se posee un conjunto de transacciones con el que operar, el subcomponente, a través del “Comunicador Merkle” se encarga de obtener la información necesaria para el circuito y, además, modificar las hojas del árbol en función de las transacciones

recogidas. Así pues, si, por ejemplo, se posee una transferencia del usuario 1 al 4, se modificará el árbol dos veces: una para decrementar el balance del usuario 1 en X unidades, y otra para incrementar el del 4 en esa misma cantidad. Realizadas las modificaciones, obtiene la raíz que resulta en el árbol. Paralelamente genera los deltas de los usuarios afectados. Estos, en este momento, poseen un formato de clave-valor, siendo la clave el índice del usuario y el valor su balance final. También se generan las retiradas de la misma manera.

Con el conjunto de transacciones, además del estado inicial del árbol y junto con las raíces del mismo antes y después de las transacciones, el subcomponente tiene todas las entradas necesarias para poder ejecutar el circuito que genera la prueba SNARK (más información en el apartado *5.5.5*). Para ello, simplemente solicita dicha acción al subcomponente correspondiente que se encargará del proceso siempre que se le suministren los datos. El circuito solo funciona con una cantidad concreta de transacciones, definida en el momento de su creación, siendo esta el límite superior de transacciones anteriormente mencionado, 65000. Sin embargo, la existencia de múltiples condiciones de envío supone que no siempre se alcanzará dicha cantidad, haciendo imposible la generación de pruebas correctas con el circuito. Ante estas situaciones, el "subcomponente central" se encarga de generar transacciones vacías y de añadirlas al conjunto antes de generar la prueba SNARK. Estas transacciones son acciones que no tienen efecto ninguno, pues se tratan de transferencias de valor nulo entre dos usuarios cualesquiera.

Con la prueba generada, el siguiente paso es el envío de la información a la Layer 1, proceso del que se encargará la instancia del "Comunicador Layer 1" creada. La información se envía en un orden concreto debido al diseño del contrato que responde a cuestiones de optimización y, también, al enfoque que se ha perseguido, conocido como "Commit-Verify". Se trata de un planteamiento bastante extendido en el desarrollo de protocolos ZK Rollup que consiste en enviar primero los datos (deltas) y, posteriormente, la prueba SNARK, esto debido a que, por lo general, la generación de esta última requiere de una importante inversión de tiempo. Siguiendo este enfoque sería posible crear un sistema en el que un validador envía los datos, y otro, la prueba, incluso sería posible lograr la paralelización, aumentando aún más las prestaciones del sistema. Así pues, el orden de envío se corresponde con primero los deltas, acto seguido las retiradas en caso de que hubiera alguna y, por último, la prueba SNARK. Resulta importante destacar que antes de enviar los datos, estos son serializados, puesto que el contrato inteligente se ha definido para que trabaje con bytes. Así pues, los deltas y las retiradas se expresan en palabras de 12 bytes cada una que se concatenan entre ellas. Como se mencionó en la sección *5.5.4* de este documento, estas palabras de 12 bytes están formadas por el índice del usuario (2 bytes) y su balance (10 bytes); idem en el caso de las retiradas, pero reemplazando balance por cantidad total a retirar. Si las operaciones han resultado exitosas, el subcomponente central vuelve a realizar su actividad desde cero.

5.5.5. Circuito SNARK.

Se trata de uno de los subcomponentes más importantes del validador. Permite la generación de pruebas SNARK. Se ha elegido *circom [24]* como herramienta para la construcción del circuito por varios motivos. En primera instancia, es de las librerías dedicadas con mayor documentación disponible en internet; en segunda instancia, permite al programador abstraerse en gran medida de las complejidades subyacentes de trabajar con pruebas de conocimiento cero; y en última instancia, posee una sintaxis muy básica y similar a lenguajes como JavaScript.

Para que el circuito pueda realizar su actividad requiere una serie de entradas, las cuales se pueden definir como públicas o privadas. Si se declaran como públicas, cualquier usuario podrá verlas y serán necesarias para la verificación de las pruebas. El circuito funciona mediante restricciones. Estas son una serie de condiciones que tienen que darse para que la prueba que se genere se considere válida. De igual manera, también añaden complejidad al sistema, haciéndolo más resistente a técnicas de fuerza bruta. Si una de estas restricciones no se cumple, la prueba se genera de igual manera, pero se rechazará si se intenta verificar. En el caso que nos atañe, el circuito se va a encargar de recrear internamente y de manera aislada, sin intervención del exterior, el estado que debería tener el árbol Merkle después de procesar X cantidad de transacciones. Computado, comprueba el resultado obtenido con el que el validador pasa al circuito como entrada. Si este último ha intentado manipular el sistema a su favor, la restricción del circuito no se cumplirá generando en el acto una prueba inválida. Para poder realizar estos cálculos, el circuito requiere el estado inicial del árbol Merkle, es decir, el valor interno de cada una de sus hojas además de todas y cada unas de las transacciones que han sido procesadas por el validador en el formato dado por el “Recolector de transacciones”. Con dicha información es capaz de recrear la raíz inicial y, más tarde, la resultante de procesar las transacciones. Adicionalmente a la restricción mencionada, el circuito posee una adicional que comprueba que la raíz inicial que computa coincide con la que el validador afirma haber usado (también pasada como entrada).

El circuito no se puede emplear con cualquier cantidad de transacciones ni con árboles de tamaño arbitrario. Esto es así debido a que no se puede trabajar directamente con el circuito, sino con su versión compilada para una instancia concreta de árbol y para una cantidad dada de transacciones. En el caso del primero, basta con indicar la profundidad del mismo, 16, mientras que en el de las transacciones se debe indicar el límite superior de transacciones, que una vez alcanzado, supondría el envío de información del validador a la L1.

5.5.6. Comunicador Layer 1.

Su funcionalidad reside en permitir el trámite de información del validador a la Layer 2. En concreto, implementa métodos para el envío de los deltas, las retiradas (de haberlas) y la prueba SNARK generada. Cada envío se realiza a través de métodos diferentes debido al enfoque empleado a la hora de construir el contrato On-Chain. Dado que en el mismo se mantiene un histórico de la información asignando a cada bloque de datos un índice, el subcomponente realiza un seguimiento de dicho índice con objeto de que los envíos resulten satisfactorios. Si se intenta enviar información con un índice que no corresponde, ya sea por ya haberse utilizado o por ser mayor que el que espera el contrato, este rechaza la operación.

5.5.7. Generador de pruebas.

Dada una entrada válida, es capaz de emplear el circuito compilado para la generación de una prueba SNARK. La entrada la recibirá del “subcomponente central”, siendo esta, a efectos prácticos, el conjunto de transacciones L2 formateadas. Además de generar la prueba, realiza un sencillo proceso para expresarla en formato hexadecimal, siendo cada dato de 64 cifras hexadecimales. Esto es necesario puesto que es el formato que espera recibir el verificador de la L1. Si la prueba es correcta, pero no está expresada de la manera esperada, la verificación resultará inválida.

5.6 Contrato On-Chain.

5.6.1. Verificador.

Se trata de un contrato inteligente que se instancia dentro del “contrato Rollup”. Su función es verificar las pruebas SNARK que entrega el validador. Si se entrega una prueba inválida, el sistema no permitiría la realización de retiradas ni la entrega de nueva información. En efecto, los datos deben ser verificados en el enfoque actual antes de poder enviar otros.

Este contrato se ha obtenido a través de la herramienta utilizada para crear el circuito SNARK, *circom*. Esta posee una funcionalidad que permite generar el contrato inteligente con la capacidad de verificar las pruebas que dicho circuito puede generar.

5.6.2. Contrato Rollup.

Se trata del contrato principal, encargado de la generación del histórico de datos. Su diseño es fundamental pues es el que determina la cantidad de transacciones por segundo que se puede llegar a alcanzar. Este contrato posee cuatro métodos importantes. El primero se corresponde con el que los usuarios utilizan para realizar depósitos. Si el usuario es nuevo, se le asigna un ID y se registran ambos, ID y dirección asociada, en el contrato. Si el usuario ya está registrado, este proceso se salta. En ambos casos, se emite un evento que será captado por el componente “Puente” para recrear el

depósito en la L2. Los tres métodos restantes son de uso exclusivo de los validadores y permiten la entrega de datos, retiradas y pruebas criptográficas. Resulta relevante recordar que la pretensión del prototipo es obtener la máxima cantidad de transacciones posibles partiendo de un sistema centralizado y con un elevado grado de confianza. Esto quiere decir, que la información que tramitan los validadores no será comprobada por parte del contrato, reduciendo así la operatoria de los métodos.

El método que permite tramitar los deltas será el que determine la cantidad de transacciones por segundo del sistema. Resulta importante, pues, determinar cuántos deltas se pueden incluir en una única transacción. Para ello, debemos tener en cuenta dos factores: el límite de gas máximo al que aspiramos y los costes de gas de las distintas operaciones que se realicen. El primero es, en este caso, de 15.000.000 de unidades, el mismo que los límites de los bloques actuales en la red principal de Ethereum. La segunda cuestión, por su parte, requiere de una mayor explicación. En Ethereum, además de los cálculos, el simple paso de datos tiene un coste de gas asociado. Además, existen múltiples maneras de conseguir el histórico. A continuación se expondrán los costes aproximados de almacenar información.

- **Almacenamiento dentro del contrato**: Se requieren 20000 de gas por cada 32 Bytes de información. Además, una operación de escritura tiene un coste adicional de 5000 de gas. Se emplea cuando la información necesita ser accesible desde la EVM.
- **Parámetros**: 4 de gas por cada Zero Byte (un byte con todos sus bits a cero) y 68 de gas por cada byte convencional. Se emplea cuando la información solo es necesaria para la ejecución de un método.
- **Logs**: 375 de gas por cada operación que origina un Log y 8 de gas adicional por cada byte de datos. Se emplea cuando la información no necesita ser accesible desde la EVM.

En nuestro caso, la información solo la necesitamos para generar el histórico, pero los datos del mismo nunca serán utilizados por ningún método del contrato inteligente. Ello quiere decir que no se requieren para la EVM, lo que nos permite utilizar el tercer método de almacenamiento. Adicionalmente, el segundo se empleará de igual manera pues se requiere para obtener la información que el validador tramita. Al eliminar la necesidad de comprobaciones, el método únicamente se encargará de generar el Log con el índice del bloque de datos tramitado y el conjunto de todos los deltas. Adicionalmente, se genera un segundo histórico con ciertos datos de interés. Se trata de una estructura con el índice del bloque de datos, la raíz Merkle asociada (se asigna solo si la fase de verificación es exitosa), y las direcciones de los validadores que han tramitado el bloque y, más tarde, validado. Este sistema permite tener un registro de quienes han sido responsables de ciertas operaciones, pudiendo remitir a los mismos en caso de fraude. Aunque es cierto que los validadores se consideran de confianza, siendo en principio imposible la posibilidad de un fraude, esta información se considera útil de cara a los usuarios y, en parte debido a la sencillez de su implementación, se ha incluído en el sistema final. Resulta relevante destacar que, al contrario que los deltas, esta información no se ha almacenado en forma de logs, sino dentro del contrato, en un diccionario. Esto es debido a la

necesidad de otros métodos de acceder a dicha información para modificarla (es el caso de la raíz Merkle y de la dirección del validador que ha realizado la verificación, cuyos valores se asignan en fases posteriores).

La tramitación de retiradas se debe realizar después del método anterior, pero antes de proceder a la verificación. El método correspondiente se encarga de recibir el conjunto de retiradas a realizar y de almacenar dicha información en el bloque de datos generado por el método que genera el histórico de datos, empleando, en consecuencia, la primera alternativa de almacenamiento. Nótese que la realización de las retiradas no se realiza en este paso. Esto es así, porque se debe impedir toda acción que afecta al balance interno del contrato hasta que se haya verificado la información entregada a través de la prueba SNARK. Si este método hiciera efectiva las retiradas, se podrían retirar fondos a partir de bloques de datos incorrectos.

Por último, tenemos el método que se encarga de verificar los bloques de datos entregados. Este recibe como parámetros la prueba SNARK junto con las entradas públicas del circuito, las cuales se corresponden con la raíz Merkle antes y después de procesar las transacciones. Internamente, el método hace uso de una instancia del contrato “Verificador” previamente explicado para comprobar la veracidad de la prueba entregada. De igual manera, también comprueba que la raíz inicial se corresponda con la misma que el contrato tiene registrada en el último bloque de datos validado. Si ambas comprobaciones son exitosas, el método pasa a hacer efectivas todas y cada una de las retiradas solicitadas y registradas en el bloque actual. Cada retirada genera un evento que los usuarios pueden después comprobar para obtener un histórico de todas las efectuadas. También se emite un segundo tipo de evento que notifica sobre el hecho de que el bloque ha sido verificado. Es en este momento cuando los validadores podrían enviar otro conjunto de datos, repitiendo el proceso descrito.

Resulta importante destacar que el sistema está pensado para la existencia de un único validador. En caso de que hubiera varios se deberían implementar mecanismos de espera en los validadores con objeto de evitar el envío de bloques en situaciones en las que no es posible. De igual manera exigiría la creación de un nuevo componente que se encargue de gestionar la actividad de los validadores con objeto de que las actuaciones de uno no afecte a la del resto. Dichas acciones, aunque interesantes a nivel comercial, aumentarían la complejidad general del sistema, motivo por el cual han sido descartadas en el prototipo.

6. Resultados obtenidos.

6.1. Transacciones por segundo.

La cantidad de transacciones por segundo que se pueden alcanzar depende del método encargado de generar el histórico de datos. En concreto, de la cantidad de deltas que sea capaz de aceptar bajo la implementación actual. Existe una excepción, que es el caso de que el trámite se haya realizado por alcanzar el máximo de retiradas. En dicho caso el método que define la cantidad máxima de transacciones que se pueden procesar es el encargado de la verificación.

Resulta necesario atender a las distintas situaciones que se pueden dar, detallando las transacciones por segundo que se obtendrían en cada una de ellas. Destacar que los resultados que aquí se indican son producto de la experimentación.

Con respecto a la cantidad de deltas que el sistema es capaz de procesar sin sobrepasar el límite de gas establecido, esta depende en gran medida de cuántos bytes a cero poseen los datos tramitados. Tal y como se mencionó en el apartado anterior, los parámetros de entrada poseen un coste en gas asociado que dependerá de cuántos bytes a cero estos tengan. Atendiendo a la realidad de que nunca debería darse un caso en el que todos los deltas están a cero, supongamos los dos siguientes escenarios:

- Cada delta tiene 5 o menos *No-Zero* bytes: En este caso, se ha obtenido que la cantidad máxima de deltas que se podrían tramitar en cada transacción sería de 70000.
- Ningún byte a cero: Si todos los bytes poseen información, entonces la cantidad anterior se reduce a 48000 deltas máximos.

Como se puede comprobar, los resultados claramente varían. Resulta importante, no obstante, determinar bajo qué escenario trabajará nuestro sistema. A pesar de que el primero aporta mejores prestaciones, tiene asociado un factor de riesgo demasiado elevado. Si desarrollamos el validador para que tramite las transacciones al generar 70000 deltas, el sistema podría fallar si todos, o simplemente una elevada proporción de estos poseen más de 5 *No-Zero* bytes. Si el contrato rechaza la llamada por sobrepasar el límite de gas, el sistema requeriría de intervención manual para poder seguir operando. Esta situación provoca que, al trabajar con ZK Rollups, resulta necesario operar siempre en el peor escenario, por muy improbable que este sea. Así pues, podemos afirmar que la cantidad máxima de deltas que soportará nuestro contrato será de 48000.

No obstante, ahora se plantea la pregunta de a cuántas transacciones se corresponden dichos deltas. De nuevo, se nos vuelven a presentar dos situaciones.

- En el peor de los casos, se alcanzan después de 24000 transacciones.
- En el mejor de los casos, no se alcanzan nunca. En esta situación la cantidad de transacciones que se enviarán sería el límite de las mismas establecido a nivel del validador y fijado en la instancia del circuito compilada, que en nuestro caso se

corresponde con 65000.

Atendiendo al hecho de que los bloques en Ethereum se generan cada, aproximadamente, 15 segundos, en el peor de los casos obtendríamos $24000\,t\,/\,15\,s\,=\,1600\,t/s$. En el segundo, $65000\,t\,/\,15\,s\,=\,43333\,t/s$. Por tanto, el prototipo desarrollado sería capaz de generar entre 1600 y 4333 transacciones por segundo, en función de la actividad de los usuarios en la L2.

No obstante, en el caso de que la condición de envío sea el haber alcanzado la cantidad máxima de retiradas, los resultados cambiarían drásticamente. En dicha situación, y atendiendo al hecho de que cada retirada supone un coste de 21000 de gas y que estas se procesan junto con la verificación de la prueba SNARK, unos 377594 de gas, si todas las transacciones fueran retiradas, el límite se alcanzaría con la 120. Ello supondría un rendimiento de $120\,t\,/\,15\,s\,=\,8\,t/s$, menor que las 15 de las Layer 1. No obstante, esta no es una situación realmente preocupante. Las retiradas están pensadas para ser las operaciones menos comunes de todas, por lo que se den 120 de ellas seguidas es ciertamente improbable. De igual manera, las retiradas suponen siempre un cuello de botella en la mayoría de sistemas Layer 2 debido a la necesidad de realizar siempre una operación de 21000 de gas para transferir los fondos. A modo de ejemplo del impacto que tienen estas operaciones, si usáramos todo el límite de gas para procesar retiradas, solo podríamos efectuar con éxito 714, cifra reducida que se correspondería con 47 transacciones por segundo.

6.2. Mejora de escalabilidad.

Si solo se realizan transferencias en Ethereum (L1), se podrían efectuar hasta 714 por bloque. Con el prototipo desarrollado, entre 24000 y 65000. Ello supone una mejora en un factor de 33 y 91 respectivamente.

6.3. Reducción de costes.

Aunque en el prototipo se han excluido las tasas, para este ejercicio se va a determinar cuánto gas deberían pagar los usuarios por cada una de sus operaciones. Supongamos, primero, un escenario en el que no se da ninguna retirada, únicamente transferencias entre usuarios. Para calcular el coste, lo mejor es trabajar con los dos peores escenarios que se pueden dar. El primero ya es conocido, alcanzar el máximo de deltas en 24000 transacciones. El segundo es alcanzar el límite de deltas al mismo tiempo que se alcanza el límite de 65000 transacciones. Este último se considera el segundo peor caso porque se sigue alcanzando el límite de deltas, que es lo que determina el coste de gas del método. Si por algún motivo se alcanzarán 65000 transacciones generando solamente 2 deltas, entonces el coste de gas estaría lejos del límite, pues rondaría alrededor de las 70000 unidades.

- Si se alcanza el límite de deltas en deltas en 24000 transacciones, entonces, y considerando el coste de la prueba SNARK, cada transacción tendría un coste en gas de $15000000\,g\,/\,24000\,t\,=\,625$ unidades.

- Si se alcanza el límite tras las 65000 transacciones, $15000000\ g\ /\ 65000\ t\ =\ 231$unidades.

Haciendo la media entre ambos valores obtendremos un valor de 428 de gas por transacción Layer 2. Costando una transacción Layer 1 21000 de gas, el usuario tendría que realizar 49 transacciones Layer 2 para alcanzar el coste que le supondría una única transacción Layer 1.

No obstante, los datos ofrecidos son solo vinculantes en caso de que realicen transferencias. Si la operación realizada es un depósito o una retirada los resultados cambian. En el primer caso, el usuario además del coste de 428 de gas, debe pagar lo que le cueste la operación L1 del depósito. El coste de la misma dependerá fundamentalmente de si ya se encuentra registrado o no. En el peor de los casos, que es aquel en el que debe registrarse, el coste en gas de la llamada al método del contrato le supondría un coste de 75386 de gas adicionales.

En el caso de las retiradas hay que tener en cuenta más variables. Además de los 428 de gas, se debe tener en cuenta los 21000 de gas necesarios para realizar la transferencia. Así mismo, tramitar el dato de las retiradas supone una operación de escritura que, como se indicó en su momento, tiene un coste de 20000 de gas por cada 32 bytes. Teniendo en cuenta que cada retirada son 12 bytes, al usuario le correspondería pagar 7500 de gas adicionales, a los que habría que sumar, por último, el coste asociado de sobrescribir datos en la EVM, aproximadamente 5000 de gas. Ello supone un coste total de 33928 de gas, un 61% de gas más que una operación L1.

Estos datos vienen a demostrar que, en efecto, tanto los depósitos como las retiradas son operaciones que escasamente se van a realizar. Los usuarios van a dedicarse, casi en exclusiva, a la realización de transferencias y únicamente recurrirán a los depósitos y retiradas cuando sea necesario.

Atendiendo a los resultados expuestos, resulta interesante plantear cuántas operaciones L2 deben realizar los usuarios para amortizar el coste de un depósito y el de una retirada. En el caso del depósito se corresponde con la ecuación $21000x\ =\ 75386\ +\ 428x$, que se resuelve cuando X toma el valor, aproximadamente, de 3.66. En el caso de las retiradas, la ecuación sería $21000x\ =\ 33928\ +\ 428x$, donde X debe valer alrededor de 1.65 para poder resolverla. Esto significa que los usuarios amortizarán la inversión inicial del primer depósito con la cuarta transferencia L2. Cada retirada, por su parte, requiere de dos transferencias L2 para amortizarse.

7. Conclusiones y líneas futuras.

7.1. Conclusiones.

Los resultados del prototipo mostrados en el apartado anterior demuestran cómo, en efecto, ZK Rollup permite aumentar la escalabilidad de Ethereum, tanto en transacciones por segundo como en el gas que se ha de pagar por cada una de ellas, permitiendo a los usuarios reducir costes.

Sin embargo, dichos resultados son consecuencia de la naturaleza experimental del prototipo, presente esta en ciertas decisiones de diseño que se han tomado. Así pues, adaptar el proyecto a uno comercial supondría una disminución de las mejoras de escalabilidad debido a las adaptaciones que se tendrían que realizar, especialmente si se quiere disponer de un sistema con un mejor grado de centralización, que exigiría incrementar el número de comprobaciones a realizar en el contrato On-Chain para garantizar la viabilidad de la información. También sería preciso adaptar al sistema de ciertas características que ahora mismo carece, como pueden ser utilidades para garantizar las salidas de los usuarios en caso de ataque o de fallo de funcionamiento en la L2, evitando así que los fondos queden congelados.

Los resultados también son consecuencia del tipo de transacciones con las que se opera, únicamente transferencias. Adaptar el sistema para que permita todo tipo de transacciones supondría una disminución general de la escalabilidad debido, de nuevo, a una mayor cantidad de cálculos y por los efectos subyacentes de estas en la L1. También se debe atender al hecho de que cada transacción no estándar requeriría de tramitar distintos tipos de datos a la L1, en la mayoría de los casos de una mayor cantidad de bytes.

7.2. Líneas futuras.

Habiendo demostrado la viabilidad del protocolo para aumentar la escalabilidad de Ethereum, los siguientes pasos deberían ser la apertura de líneas de investigación dirigidas a mejorar los resultados obtenidos y adaptarlo a un sistema más comercial. Para esto último, es necesario establecer mecanismos que incrementen la seguridad del sistema en su conjunto y que además no partan de un nivel de confianza inverificable. También sería preciso estudiar cómo lograr la paralelización para combatir los efectos negativos que tendría el incremento de operaciones. Simultáneamente debería investigarse y plantearse distintas infraestructuras Layer 2 a la blockchain privada implementada en el protocolo. A efectos prácticos sería interesante proponer una lista de posibles alternativas y sus respectivos casos de uso.

En los ZK Rollups la acción efectuada por el validador es la más costosa en recursos de todas. Puede, sin demasiada dificultad, tardar minutos en realizar su actividad debido, por un lado, a la complejidad general de las pruebas de conocimiento cero y, por otro, a la modificación del árbol Merkle, donde el rendimiento del algoritmo de encriptación es vital. En el caso del prototipo, el factor que más ralentiza la actividad del validador es la modificación del árbol Merkle que este debe realizar. Esta puede deberse a la

implementación del árbol en sí o, al algoritmo de encriptación utilizado (Mimc7). Independientemente de la situación, sería preciso abrir varias líneas de investigación al respecto. Por un lado, estudiar la posibilidad de que se implementen mejoras en el código existente y, por otro y más importante, estudiar distintas alternativas al algoritmo empleado. Este se seleccionó frente a otras opciones debido a su principal característica, estar optimizado para pruebas de conocimiento cero. Esta optimización supone que su uso en circuitos requiere de una mayor cantidad de recursos, siendo necesario para el caso que nos atañe debido a las limitaciones existentes al respecto y por el diseño del circuito construido. Una posible alternativa sería estudiar diseños alternativos que permitan las mismas garantías consumiendo menos recursos y sin tener que modificar el algoritmo de encriptación. En consecuencia, una línea de investigación debería enfocarse en analizar en profundidad las pruebas de conocimiento cero, tanto la manera más adecuada para construirlas como distintos frameworks para su creación (recordar que en el prototipo se ha empleado *circom*).

Por último, dependiendo del escenario de uso que se quiera satisfacer, puede ser necesario construir un ZK Rollup que admita todo tipo de transacciones, no únicamente transferencias. Este requisito, de plantearse, sería el más costoso de todos los hasta ahora presentados, ya que exigiría cambios en la mayoría de los componentes del sistema. Por un lado, debería construirse una nueva API para los usuarios que permita esta realidad y ajustar la infraestructura L2 a esta. También sería preciso realizar modificaciones en el validador así como en el contrato On-Chain. El primero debería atender a la naturaleza de las nuevas transacciones y preparar distintos tipos de formatos, posiblemente renunciando a la posibilidad de emplear deltas. El segundo, por su parte, debería preparar distintos métodos para recrear en la L1 los efectos que los usuarios desean. No obstante, la mayor dificultad radicaría en el diseño del circuito que genera las pruebas de conocimiento cero, que debería admitir todo tipo de transacciones, no únicamente transferencias. Tal y como se mencionó al principio de este documento, es la dificultad de lograr pruebas de conocimiento cero generales lo que ha restringido los escenarios de uso para los que se emplea ZK Rollup. Además, también fue el motivo de aparición de Optimistic Rollup. No obstante, es una cuestión que sigue siendo posible y que los miembros de Matter Labs lograron (después de años de investigación) implementar con éxito en su propuesta ZkSync, ideando, en parte, una VM similar a la de Ethereum (aunque con sus diferencias).

8. Summary and conclusions.

Ethereum has a scalability problem which, for users, is evident in its rate of transactions per second and the gas cost of each transaction. In order to solve these problems, a Layer 2 prototype based on the ZK Rollup protocol has been studied and developed. Given the experimental nature of the project, both the architecture and the design of the prototype have been conceived with the objective of obtaining the best possible results. In general terms, the project can be divided into seven major components present in both layers. These allow interaction among the users, the update of the system state and the communication between layers, which is necessary to data transfers.

The results obtained demonstrate the viability of Layer 2 protocols as solutions to the Ethereum scalability problem. The numbers of transactions per second have been expanded from 15 to a minimum of 1600 and a maximum of 4333, although, if the system is fully controlled by withdrawals, a type of operation consisting of the extractions of funds from a Layer 1 smart contract, the number is reduced to 8 transactions. In terms of the gas needed for the different operations, each Layer 2 transaction cost 428 units. If the operation requires some kind of action from Layer 1, then the cost is increased for the user depending on the required operation. If this is a withdrawal, the resulting value is 33928 and, if it is an initial deposit in the same smart contract, 75386. Both operations are amortised with 2 and 4 conventional Layer 2 transfers. These results demonstrate, in addition, how the protocol is aimed towards user interaction, avoiding as much as possible the execution of any action that requires any kind of interaction in Layer 1.

9. Presupuesto.

Para la realización del prototipo únicamente se ha utilizado un ordenador, en el que todos los sistemas funcionan de manera coordinada.

Elemento	Coste (€)	Descripción
Ordenador	62.49	El ordenador tiene un coste aproximado de 1000 €, y su duración estimada es de 4 años. Habiendo durado el proyecto 3 meses, el coste correspondiente es el indicado.
Formación	4500	Se han dedicado aproximadamente 300 horas a formación, a un precio de 15 € la hora.
Análisis y diseño	600	Se han dedicado aproximadamente 40 horas a análisis y diseño, a un precio de 15 € la hora.
Desarrollo del prototipo	2760	Se han dedicado aproximadamente 184 horas al desarrollo del prototipo, a un precio de 15 € la hora.
Extracción de resultados y conclusiones	180	Se han dedicado aproximadamente 12 horas a la extracción de resultados y conclusiones, a un precio de 15 € la hora.
Total	8102.49	

Tabla 11: Presupuesto

10. Referencias y bibliografía.

[1] “Cadena de bloques”, Wikipedia [Online]. Disponible en: https://es.wikipedia.org/wiki/Cadena_de_bloques

[2] “Smart Contracts”, Wikipedia [Online]. Disponible en: https://en.wikipedia.org/wiki/Smart_contract

[3] “Plasma”, EthHub [Online]. Disponible en: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/plasma/

[4] “Merkle Tree”, Wikipedia [Online]. Disponible en: https://en.wikipedia.org/wiki/Merkle_tree

[5] I. Hagopian (2019, 14 Enero), “Fraud proofs—Secure on-chain scalability” [Online]. Disponible en: https://hackernoon.com/fraud-proofs-secure-on-chain-scalability-f96779574df

[6] “State Channels”, EthHub [Online]. Disponible en: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/state-channels/

[7] “SideChains”, EthHub [Online]. Disponible en: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/sidechains/

[8] “ZK-Rollups”, EthHub [Online]. Disponible en: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/

[9] T. Tokens24 (2018, 25 Abril), “La prueba de conocimiento cero explicada” [Online]. Disponible en: https://www.tokens24.com/es/cryptopedia/basics/la-prueba-del-conocimiento-cero-explicada

[10] C. Reitwiessner (2016, 5 Diciembre), “zkSNARKs in a nutshell” [Online]. Disponible en: https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/

[11] A. Gluchowski (2020, 6 Junio), “zkRollup vs. Validium” [Online]. Disponible en: https://medium.com/matter-labs/zkrollup-vs-validium-starkex-5614e38bc263

[12] “StarkEx Validium ransom attack”, Ethereum.org [Online]. Disponible en: https://notes.ethereum.org/DD7GyItYQ02d0ax_X-UbWg

[13] “Optimistic Rollups”, EthHub [Online]. Disponible en: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/optimistic_rollups/

[14] “Soluciones de capa 2 (L2), rollups y el futuro de Ethereum con Jordi Baylina”, Investing [Online]. Disponible en: https://es.investing.com/news/cryptocurrency-news/soluciones-de-capa-2-l2-rollups-y-el-futuro-de-ethereum-con-jordi-baylina-2112648

[15] “zk-SNARKs y zk-STARKs Explicadas”, Binance [Online]. Disponible en: https://academy.binance.com/es/articles/zk-snarks-and-zk-starks-explained

[16] “zk-SNARKs y zk-STARKs Explicadas”, Binance [Online]. Disponible en: https://academy.binance.com/es/articles/zk-snarks-and-zk-starks-explained

[17] Metamask [Online]. Disponible en: https://metamask.io

[18] “warp”, GitHub [Online]. Disponible en: https://github.com/seanmonstar/warp

[19] “Tokyo”, Tokyo [Online]. Disponible en: https://tokio.rs

[20] M. Albrecht, L. Grassi, C. Rechberger, *et al,* “MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity” [Online]. Disponible en: https://eprint.iacr.org/2016/492.pdf

[21] K. Ziechmann (2021, 30 Marzo), “Transactions” [Online]. Disponible en: https://ethereum.org/en/developers/docs/transactions/

[22] ryancreatescopy (2021, 24 Mayo), “Gas and fees” [Online]. Disponible en: https://ethereum.org/en/developers/docs/gas/

[23] “Proof-of-Authority consensus”, readthedocs [Online]. Disponible en: https://apla.readthedocs.io/en/latest/concepts/consensus.html

[24] Circom [Online]. Disponible en: https://docs.circom.io

Otras referencias

(2021, 5 Enero), “An Incomplete Guide to Rollups” [Online]. Disponible en: https://vitalik.ca/general/2021/01/05/rollup.html

Interdax (2020, 29 Octubre), “Scaling Ethereum on L2: Optimistic Rollups and ZK-Rollups” [Online]. Disponible en: https://medium.com/interdax/ethereum-l2-optimistic-and-zk-rollups-dffa58870c93

vbuterin (2018, 22 Septiembre), “On-chain scaling to potentially ~500 tx/sec through mass tx validation” [Online]. Disponible en: https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477

ZKSwap “ZKSwap Whitepaper: a Layer-2 Token Swap Protocol based on ZK-Rollups Technology” [Online]. Disponible en: https://medium.com/zkswap/zkswap-whitepaper-a-layer-2-token-swap-protocol-based-on-zk-rollup-113671ef3e6d

“Introduction to ZK Rollup”, DevelopPaper [Online]. Disponible en: https://developpaper.com/introduction-to-zk-rollup/

A. Gluchowski (2020, 12 Junio) “Evaluating Ethereum L2 Scaling Solutions: A Comparison Framework” [Online]. Disponible en: https://medium.com/matter-labs/evaluating-ethereum-l2-scaling-solutions-a-comparison-framework-b6b2f410f955