Universidad de la República

Facultad de Ingeniería

Gestión de la privacidad al interoperar blockchain

Informe de Proyecto de Grado presentado por

Agustín Ameigenda, Braian de Barros y Santiago Martinez

como requisito de graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisor

Guzmán Llambías

Montevideo, 21 de julio de 2023

Agradecimientos

En lo personal, yo: Santiago, quiero agradecer principalmente a mi madre y a mi padre que no solo me han apoyado y acompañado durante la realización de este documento, sino que siempre han creído en mí a lo largo de toda la carrera. Agradezco también a mi familia por su apoyo y a mi novia por sus valiosos consejos. Por último, agradezco a todos mis amigos por brindarme su motivación y aliento para escribir este documento. Muchas gracias a todos por acompañarme a lo largo de la carrera.

Yo, Agustín, quiero en primer lugar agradecerles a mis padres y hermana por haberme apoyado y alentado a lo largo del proyecto y toda mi carrera. Además, agradezco al resto de mi familia por el apoyo brindado. También quiero agradecer a todos mis amigos por haberme acompañado y animado a lo largo de todo este proceso. Finalmente, pero no de menor importancia, quiero agradecer a mis compañeros con los que he compartido este proceso. Les agradezco mucho a todos.

Yo, Braian, quiero agradecer a mis padres y hermanos, quienes han sido mi mayor apoyo a lo largo de este proyecto y toda mi trayectoria en la Universidad. Asimismo, quiero reconocer y agradecer la colaboración de todos los compañeros de estudio con quienes he compartido este emocionante recorrido.

Finalmente, los tres miembros del equipo queremos agradecerle a nuestro tutor Guzmán, por su dedicación durante el desarrollo del trabajo de grado. Los consejos y la motivación que nos brindó, sin duda alguna, fueron invaluables, muchas gracias. Además, queríamos agradecer a una amiga del equipo que nos brindó consejos muy valiosos sobre gramática española, su valiosa información, sin duda alguna, fue de mucha ayuda, gracias.

Resumen

En los últimos años, el área de interoperabilidad dentro de la tecnología *blockchain* ha obtenido cada vez más protagonismo y hoy en día son varias las soluciones que permiten interoperar dos o más *blockchain*, independientemente de si estas son públicas o privadas. Las *blockchain* públicas son redes abiertas y descentralizadas en las que cualquier persona puede participar y acceder a los datos. A su vez, los usuarios son pseudoanónimos y la información es transparente para todos los usuarios. Por otro lado, las *blockchain* privadas son redes restringidas a un grupo selecto de participantes, con permisos y niveles de acceso controlados. En estas redes los usuarios están plenamente identificados y su información es confidencial. Interoperar *blockchain* públicas y privadas presenta ciertos desafíos, en particular se deben resolver algunos desafíos asociados a la privacidad. El primer desafío, conocido como privacidad de la identidad, consiste en mantener de forma oculta o protegida la identidad de los participantes en una transacción *cross-blockchain*. En particular, cuando un usuario pseudoanónimo de una *blockchain* pública inicia una transacción *cross-blockchain* hacia una *blockchain* privada que requiere autenticación. El segundo desafío, conocido como privacidad del mensaje, consiste en que solo los involucrados en la transacción *cross-blockchain* puedan interpretar la información de esta. Por ejemplo, que solo los participantes de la transacción puedan interpretar la información que se almacenó en la *blockchain* pública. Tanto la privacidad de la identidad como la privacidad del mensaje son requisitos de privacidad al interoperar.

En el presente proyecto se diseñó y construyó una solución que soporta los requisitos de privacidad al interoperar mencionados anteriormente. En particular, se buscó interoperar una *blockchain* pública con una *blockchain* privada, en este caso *Ethereum* con *Hyperledger Fabric* respectivamente. La solución construida extiende una solución de interoperabilidad preexistente de tipo *Gateway*, a la que se le agregó la capacidad de preservar la privacidad de la identidad de los participantes y la confidencialidad del mensaje. La solución utiliza credenciales anónimas para preservar la privacidad de la identidad y hace uso de pruebas de conocimiento cero para mantener la confidencialidad del mensaje. Se desarrolló un prototipo basado en *Idemix* y *Zokrates*. *Idemix* se utilizó para gestionar las credenciales anónimas y *Zokrates* para la construcción y desarrollo de las pruebas de conocimiento cero. Fue necesaria la utilización de la herramienta *Zokrates* debido a la dificultad que posee la construcción y manipulación de las pruebas

de conocimiento cero.

Para validar la propuesta, se instanció la solución a un escenario de prescripciones médicas. A su vez, se realizaron pruebas de performance y un análisis de costos. Las pruebas de performance mostraron que la solución es factible en términos de tiempos de procesamiento, pero existe un cuello de botella al generar la prueba de conocimiento cero que debe ser tenido en cuenta. A partir de los datos proporcionados por el análisis de costos, se concluyó que la solución es factible en términos económicos si los medicamentos son de alto costo.

A modo de conclusión, se alcanzó el objetivo de diseñar e implementar un prototipo que satisfaga los requisitos de privacidad al interoperar una *blockchain* publica con una privada, en particular *Ethereum* con *Hyperledger Fabric*.

Palabras clave: Blockchain, Interoperabilidad, Privacidad, Ethereum, Hyperledger Fabric, Idemix, Zokrates, Gateway

Índice general

Capítulo 1

Introducción

En este capítulo se presenta una visión general del contenido del documento, así como una introducción al trabajo realizado. En la sección 1.1 se postula la motivación que impulsó la realización de este trabajo. Luego en la sección 1.2 se listan los objetivos del proyecto. Posteriormente, en la sección 1.3 se describen concretamente los aportes realizados por el proyecto. Finalmente, en la sección 1.4 se describe la estructura del documento.

1.1. Motivación

En el año 2008 se publicó el *paper*: *"Bitcoin: A Peer-to-Peer Electronic Cash System"*, el cual proponía la creación de *Bitcoin*, un activo digital basado en un registro inmutable que funciona sobre una red de nodos descentralizada (Nakamoto, 2008). Dicho registro inmutable se conocería posteriormente como *blockchain*, una de las tecnologías disruptivas más importante de los últimos años. Principalmente debido a qué *blockchain* demostró su utilidad en escenarios de confianza cero, por ejemplo, pagos electrónicos. Dado que la confianza en las instituciones financieras y la reserva de valor han sido el problema por excelencia a lo largo de la historia, es que la tecnología *blockchain* (junto a las criptomonedas) es utilizada principalmente como reserva de valor, es decir, dinero. Sin embargo, no solo se limita a este tipo de escenarios, también se ha visto aplicada en otros como: salud, IoT, gobierno, entre otros (Llambías y cols., 2019). A esto se le suma que mes a mes nacen nuevas *blockchain* que brindan una amplia variedad de soluciones como, por ejemplo: contratos inteligentes, almacenamiento descentralizado, pruebas de propiedad digital, por mencionar alguna de ellas.

Sin embargo, la interoperabilidad entre las *blockchain* no es una característica que posean en su diseño. Las *blockchain* funcionan como grandes silos de información aislados y no es posible compartir información entre ellas (R. Belchior y Correia, s.f.). Es por tal motivo que en los últimos años han aparecido varias soluciones que permiten la interoperabilidad entre diversas *blockchain*

(Guzman Llambias, s.f.). Entendiéndose a la interoperabilidad como la capacidad para conectarse, comunicarse y colaborar entre sí de manera efectiva y segura. Sin embargo, estas soluciones aún poseen varias limitaciones dependiendo del tipo de *blockchain* a interoperar, ya que estas pueden ser públicas o privadas. Las *blockchain* públicas son redes abiertas y descentralizadas, accesibles para cualquier persona. A su vez, los usuarios son pseudoanónimos y pueden visualizar toda la información en la red, dado que es transparente. En contraposición, las *blockchain* privadas son redes restringidas a un grupo selecto de participantes con niveles controlados de permisos y accesos a la información. En este tipo de *blockchain* los usuarios están plenamente identificados y su información es confidencial.

Entre las limitaciones que presentan las soluciones de interoperabilidad al interoperar *blockchain* públicas con privadas, existen desafíos asociados a la privacidad. El primer desafío, conocido como privacidad de la identidad, consiste en mantener de forma oculta o protegida la identidad de los participantes en una transacción *cross-blockchain*. En particular, cuando un usuario pseudoanónimo de una *blockchain* pública inicia una transacción *cross-blockchain* hacia una *blockchain* privada que requiere autenticación. El segundo desafío, conocido como privacidad del mensaje, consiste en que solo los involucrados en la transacción *cross-blockchain* puedan interpretar la información de esta. Por ejemplo, que solo los participantes de la transacción puedan interpretar la información que se almacenó en la *blockchain* pública. Tanto la privacidad de la identidad como la privacidad del mensaje son requisitos de privacidad al interoperar.

Al día de hoy no existen o son muy pocas las soluciones que abordan el problema de la privacidad al interoperar, en particular una *blockchain* publica con una privada (Bernal Bernabe, Canovas, Hernandez-Ramos, Torres Moreno, y Skarmeta, 2019). Lo que limita el uso y adopción de la tecnología.

1.2. Objetivos

El objetivo principal del proyecto es diseñar una solución de interoperabilidad entre plataformas *blockchain* público-privada, que conserve las características de privacidad al interoperar. La lista de objetivos del proyecto de forma específica es la siguiente:

- Estudiar el concepto de privacidad en *blockchain* y relevar qué soluciones existen actualmente para preservarla.
- Realizar un estudio de la privacidad al interoperar plataformas *blockchain* y que soluciones existen para conservar las características de privacidad al interoperar.
- Relevar y clasificar las soluciones existentes de interoperabilidad según las características de privacidad.
- Diseñar e implementar una solución de interoperabilidad que conserve las características de privacidad al interoperar *blockchain* públicas y privadas.

- Evaluación de la propuesta mediante su aplicación en un escenario de prescripciones médicas, análisis de costos y pruebas de performance.
- Documentar el trabajo realizado en un informe técnico.

1.3. Aportes del proyecto

Los aportes realizados en el proyecto son los siguientes:

- Relevamiento de la privacidad en *blockchain* y en interoperabilidad entre plataformas *blockchain*.
- Clasificación de las soluciones existentes de interoperabilidad entre plataformas *blockchain* con base en las características de privacidad.
- Evolución de la solución de interoperabilidad desarrollada en el trabajo de grado (Mathías Castro, 2023) incorporándole el diseño de una solución para satisfacer los requisitos de privacidad de la identidad y de privacidad del mensaje al interoperar.
- Extensión del prototipo propuesto por el proyecto de grado (Mathías Castro, 2023), incorporando la propuesta de solución diseñada y mejorando la portabilidad del prototipo junto a las nuevas incorporaciones «*dockerizandolo*».
- Evaluación del prototipo mediante su aplicación a un escenario de prescripción médica, pruebas de performance, análisis de costos y comparación con proyectos existentes.
- Documentación técnica del trabajo realizado.

1.4. Estructura del documento

El documento se divide en los capítulos descritos a continuación. El capítulo 2 desarrolla los conceptos teóricos necesarios para comprender el trabajo realizado. El capítulo 3 analiza los requerimientos del proyecto. El capítulo 4 desarrolla la conceptualización de la solución, explicando cómo se llegó al diseño de la solución. El capítulo 5 describe el diseño de la solución. El capítulo 6 ahonda en la implementación del prototipo. En el capítulo 7 se exponen las evaluaciones llevadas adelante para validar la propuesta. En el capítulo 8 se muestra cómo fue la gestión del proyecto. Finalmente, en el capítulo 9 se desarrollan las conclusiones y se proponen posibles mejoras en la solución como trabajo a futuro.

Capítulo 2

Marco conceptual

En este capítulo se explicarán los conceptos utilizados a lo largo del documento. El capítulo se divide en cuatro grandes secciones: *blockchain*, privacidad en *blockchain*, interoperabilidad en *blockchain* y trabajo relacionado. La sección 2.1 introduce el concepto de *blockchain*: desarrolla sus características, explica su clasificación y brinda algunos ejemplos. La sección 2.2 presenta soluciones de privacidad existentes, desarrolla conceptos y herramientas de privacidad relevantes para el desarrollo del proyecto. La sección 2.3 comienza definiendo interoperabilidad, presenta cuáles son los requisitos de privacidad al interoperar, los tipos de operaciones de interoperabilidad que existen, los retos que presenta interoperar *blockchain*, y, por último, se desarrollan cuáles son las soluciones de la industria para interoperar actualmente. Finalmente, en la sección 2.4 se presentan algunos trabajos relacionados.

2.1. Blockchain

Los primeros indicios del nacimiento de esta tecnología se dieron a través del *paper* denominado: *"Bitcoin: A Peer-to-Peer Electronic Cash System"* (Nakamoto, 2008) bajo la autoría del enigmático Satoshi Nakamoto, persona de la cual se desconoce su identidad real. El enfoque del artículo consiste en una solución a los problemas existentes en el comercio en línea, en el que a menudo se necesita la intervención de una institución financiera como intermediaria en las transacciones entre dos personas. Al utilizar estas instituciones, se corre el riesgo de que las transacciones sean revertidas. Estas instituciones también deben mediar en las posibles disputas que puedan surgir en las transacciones, lo que requiere un alto nivel de confianza en la institución. Además, el costo de la mediación incrementa el costo total de la transacción, ya que estas instituciones cobran por actuar de intermediario entre las partes. Las limitaciones impuestas por las instituciones financieras también hacen que las transacciones en línea sean menos accesibles para aquellos que desean realizar transacciones pequeñas y casuales. Esto se debe a que las instituciones financieras suelen establecer montos mínimos

de transferencia que limitan el tamaño mínimo práctico de las transacciones.

Las *blockchain*[1] son un tipo de DLT (*Distributed Ledger Technology*) (Adviser, s.f.). Una DLT es una base de datos que se puede compartir en una red de múltiples sitios. Todos los participantes de esta red pueden tener una copia de esta base de datos. La seguridad y precisión de estos sistemas se mantienen criptográficamente a través del uso de claves y firmas para controlar las acciones dentro de la misma. Las entradas pueden ser actualizadas por un participante y el cambio se vería reflejado en toda la red. Aunque una *blockchain* es un tipo de DLT, una DLT no es necesariamente una *blockchain*, ya que la *blockchain* agrega otras características como su estructura de datos basada en bloques, *smart contract*, entre otras.

Blockchain es una tecnología conformada por un conjunto de máquinas con capacidad de procesamiento y almacenamiento denominados nodos, que permite mantener una estructura de datos, llamada *ledger*, compuesta por una cadena de bloques en donde se almacenan los datos. Cada usuario de la *blockchain* tiene su propia clave pública y privada. Cuando este se registra en la red, su clave pública se agrega a la cadena de bloques y se asocia con su identidad. De esta manera, logra cierto grado de anonimato al estar asociado a un identificador y no dar a conocer su identidad real, similar a como funciona el número de cuenta en un sistema bancario. Otros usuarios pueden utilizar la clave pública para cifrar información confidencial, y el destinatario utilizará su propia clave privada, que está vinculada a su clave pública única, para descifrar el mensaje y procesar la transacción.

Las operaciones que se realizan para agregar información al *ledger* son denominadas transacciones y su contenido puede ser variado dependiendo del tipo de *blockchain* que se esté operando. Existen *blockchain* orientadas esencialmente a la reserva de valor, otras orientadas al almacenamiento de archivos digitales, entre otras. Cada una de estas transacciones son enviadas a la red de nodos, luego de ser validadas se ≪*hashean*≫[2] y se combinan con otras transacciones para crear un bloque y así agregarlo a la cadena por medio de un algoritmo de consenso, la transacción queda incorporada a la *blockchain* y se considera como completa.

2.1.1. Características

Si bien existen diferentes arquitecturas de *blockchain*, todas ellas cumplen con un determinado conjunto de características que hacen que se diferencien de los sistemas centralizados que abundan hoy en día. (Hong-Ning Dai y Xie, 2018)

[1] A lo largo del documento el término *blockchain* se utilizará tanto para el plural como para el singular de *blockchain*.

[2] El verbo *hashear* no existe en español, pero dado que es un término ampliamente utilizado en la jerga informática, se utilizara en el informe con el significado de: ≪aplicar una función de *hash* a un elemento y obtener su *hash*≫.

Datos descentralizados

Al estar distribuida en múltiples nodos de la red, la información se vuelve más segura y transparente, ya que no depende de una entidad central para su verificación y validación. Esto permite una mayor confianza entre los participantes de la red y reduce la posibilidad de fraudes y manipulaciones. De todos modos, el grado de descentralización dependerá del tipo de *blockchain* que se esté tratando, si es pública o privada.

Trazabilidad

Cada transacción es validada y registrada con una marca de tiempo en la *blockchain*, lo que permite a los usuarios rastrear fácilmente los registros anteriores accediendo a cualquier nodo de la red distribuida. Esto mejora la trazabilidad y la transparencia de los datos almacenados en la *blockchain*, lo que a su vez aumenta la confianza en la red y facilita la toma de decisiones informadas.

Persistencia

Dado que cada transacción en una red *blockchain* debe ser confirmada y registrada en bloques distribuidos en toda la red, es prácticamente imposible manipularla. Cada bloque emitido es validado por otros nodos y transacciones, lo que garantiza que cualquier intento de falsificación pueda ser detectado fácilmente. Esto proporciona un alto nivel de seguridad y confianza en la integridad de los datos almacenados en la cadena de bloques. Además, la descentralización de la red *blockchain* significa que no existe un punto central de falla, lo que hace que sea muy difícil de atacar o interrumpir la red en su conjunto.

Inmutabilidad

La inmutabilidad es una característica que garantiza que una vez registrada una transacción en la red, no pueda ser alterada sin que otros nodos se enteren. Esto se debe a que cada transacción es validada por múltiples nodos de la red y almacenada en un bloque enlazado a otros bloques anteriores, lo que hace que cualquier intento de modificar una transacción anterior implique la modificación de todos los bloques posteriores, lo cual resulta prácticamente imposible de realizar.

2.1.2. Mecanismos de consenso

Dado que la estructura de datos no se encuentra en un lugar centralizado, sino que cada nodo de la red mantiene una copia, es fundamental que exista coordinación entre todos ellos para garantizar que la información sea consistente en todo momento. De lo contrario, se corre el riesgo de que algunos nodos intenten introducir información corrupta o que la falta de sincronización entre los nodos genere incoherencias en la red. Para evitar estas situaciones, se utiliza un algoritmo de consenso que determina qué bloques forman parte de la red

y cuándo se puede agregar un nuevo bloque, lo que garantiza la integridad y la coherencia de la información en la red descentralizada. Para obtener más información sobre los distintos algoritmos de consenso, ver el anexo A, donde se presentan los algoritmos de consenso utilizados en la actualidad y que tienen influencia en los entornos de desarrollo que se manejaron durante el proyecto.

2.1.3. Clasificación

A continuación, se proporciona una clasificación de las *blockchain* basada en el nivel de acceso y participación requerido para interactuar con los datos y otros participantes de la red.

Públicas

Este tipo de *blockchain* son redes descentralizadas donde no existe una autoridad central que regule el funcionamiento. Cualquier usuario u organización del mundo puede tener libre acceso a la lectura del *ledger*, ejecución de transacciones y participación del proceso de consenso, siendo estos usuarios pseudoanónimos, ya que utilizan un seudónimo como identificador y no su verdadera identidad. La hipótesis de funcionamiento en este tipo de *blockchain* se basa en que estos usuarios anónimos pueden llegar a ser poco fiables y pueden tener malas intenciones para con la red. Por esto, la importancia del algoritmo de consenso en este tipo de *blockchain*, para mantener las características de inmutabilidad y seguridad.

Privadas

Como contraparte a las *blockchain* públicas, las privadas tienen el acceso restringido al *ledger*, ya que no cualquier usuario puede acceder al mismo. Este tipo de *blockchain* suele ser usada por organizaciones donde la red es administrada por unos pocos usuarios, quienes determinan el nivel de privilegios que pueden tener los demás a la hora de consultar y/o modificar el *ledger*. Dado que esta clase de *blockchain* está orientada a organizaciones, su funcionamiento suele ser más centralizado que las públicas. Por lo tanto, no necesita de un algoritmo de consenso tan sofisticado para mantener la inmutabilidad del *ledger*, ya que se asume como hipótesis que se está ejecutando en un entorno de confianza. Sus principales aplicaciones están orientadas a mejorar los procesos internos de una organización y reducir costos.

Consorcio

Por último, las *blockchain* de consorcio comparten muchas características con las *blockchain* privadas. Estas *blockchain* se diferencian en que no son manejadas por una única organización, sino por un conjunto de organizaciones que juntas conforman un consorcio y estos definen las reglas que se deben de cumplir en cada una de las diferentes transacciones. Las organizaciones solo verán las transacciones en las que participan, otorgando cierto nivel de confidencialidad

entre las mismas. El uso principal de este tipo de *blockchain* es el de poder mejorar los procesos entre los participantes y resolver sus conflictos. Es por esto que si bien las transacciones tienen un bajo costo en comparación con las *blockchain* públicas, son más elaboradas que el de las *blockchain* privadas. Esto se debe a que el consenso se realiza entre varias organizaciones y esto posiblemente genere conflictos de intereses.

2.1.4. Smart contract

El concepto de *Smart contract* (Gans, s.f.) fue introducido, por primera vez, por el científico informático Nick Szabo y consistía en un programa que agrupaba un conjunto de promesas a cumplir entre dos partes y estipulaba las acciones a seguir en caso del no cumplimiento de alguna de ellas. La integración de este concepto en la tecnología *blockchain* permitió la ejecución de código con características propias de un contrato, permitiendo a los nodos interactuar entre sí bajo ciertas reglas ya establecidas. En el caso de *Ethereum*, un *smart contract* (Buterin, s.f.-b) es un simple programa compuesto por funciones y datos (estado) que reside en una dirección específica de la *blockchain*. Por lo tanto, una vez que son desplegados en esta no se pueden eliminar de forma predeterminada y las interacciones con ellos son irreversibles.

2.1.5. Ejemplos de blockchain

A continuación, se da un breve detalle de las dos implementaciones de *blockchain* utilizadas durante el proyecto.

Hyperledger Fabric

Hyperledger Fabric (*Fabric*) (Androulaki y cols., 2018) es una plataforma modular para operar y desplegar *blockchain* del tipo privada permisionada. Se caracteriza por permitir la preservación de la confidencialidad tanto en las transacciones como en la ejecución, permitiendo el acceso a dicha información mediante un robusto sistema de control de acceso. Además, su modularidad permite variar protocolos de consenso, métodos de autenticación y otras variables internas, haciéndola una plataforma muy flexible. A su vez, a diferencia de otras tecnologías, no posee una criptomoneda propia.

Dado que *Fabric* es una plataforma de *blockchain*, permite la creación de múltiples *ledgers* en una sola instancia, cada uno de estos es conocido como un canal. De cada canal puede participar un subdominio de todas las organizaciones que integran esa instancia de *Fabric*, esto es en parte lo que garantiza la privacidad.

Para operar *Fabric* la infraestructura requerida consta de nodos interconectados que pueden ser de tipo *orderer* o *peer* y un MSP (*Membership Service Provider*). Los nodos *orderer* se encargan de establecer el orden de las transacciones y transmitirlo a todos los nodos *peer*. Además, se encargan de manejar

modificaciones en la configuración de un canal y, en algunos casos, del control de acceso.

Por otra parte, los nodos *peer* que son los que realizan las transacciones en la *blockchain*, poseen una base de datos que guarda el estado de los canales en los que están incluidos. Cada organización que sea parte del sistema deberá contar con al menos uno de estos nodos. Finalmente, el MSP es el encargado de persistir y otorgar las credenciales a cada nodo y cliente del sistema. Además, posee un pequeño componente en cada nodo que permite autenticar los mensajes generalmente mediante una firma digital.

Ethereum

Ethereum (Buterin, s.f.-a) es una plataforma de código abierto basada en la tecnología *blockchain* que funciona mediante la utilización de *smart contract* 2.1.4. Los *smart contract* pueden ser utilizados para la creación de aplicaciones descentralizadas (dApps) y están basados en el lenguaje de programación denominado *Solidity*. Estos programas serán ejecutados automáticamente en la *blockchain* una vez que son desplegados.

La red de *Ethereum* también utiliza un sistema de consenso denominado *Proof of Work*, ya explicado en la sección A.1. En este sistema, los mineros que resuelven los problemas matemáticos y generan los bloques recibirán una recompensa en *Ether*, el activo nativo de la red.

Para ejecutar un *smart contract* o realizar una transacción en la red de *Ethereum*, es necesario gastar una cierta cantidad de *Ether*. Esto se hace mediante el sistema de tarifas de gas, que se utiliza para medir el costo computacional de ejecutar una transacción o un *smart contract* en la red. El gas se paga en *Ether* y su precio fluctúa dependiendo de la congestión de la red.

2.2. Privacidad en blockchain

Vistos los principales conceptos de *blockchain*, a continuación, se desarrollan algunos conceptos de privacidad, necesarios para la comprensión del documento. Primero, se aborda el concepto más importante de privacidad, las pruebas de conocimiento cero. Luego se describen soluciones de privacidad existentes que abordan el problema de la privacidad en *blockchain* públicas. Posteriormente, se desarrollan conceptos y herramientas de privacidad necesarios para la realización del proyecto.

2.2.1. Zero Knowledge Proofs

Las pruebas de conocimiento cero, ZKP (*Zero Knowledge Proofs*) (Ethereum.org, January 17, 2023), son una forma de probar la validez de una afirmación sin revelar la afirmación en sí ni ningún tipo de información relevante de esta. De la prueba participan dos partes: el probador y el verificador, el primero intenta probar la afirmación, mientras que el segundo intentara validar dicha prueba.

Este tipo de pruebas representaron una revolución para la criptografía, ya que permiten mejorar la privacidad de los usuarios en los sistemas donde se aplican. Por ejemplo, si un individuo desea comprar una bebida alcohólica, debe probarle al vendedor que es mayor de edad, para esto normalmente muestra su identificación, pero la identificación tiene mucha información sensible que el vendedor no tendría por qué saber. Utilizando una ZKP, el individuo que va a comprar a la tienda podría presentar la prueba de que es mayor de edad y el vendedor verificaría que dicha prueba es correcta, validando así su mayoría de edad. Como se puede observar en esta última interacción, la información sensible del cliente queda protegida, ya que no hay necesidad de que muestre su identificación.

Los protocolos que generan las ZKP cumplen las siguientes propiedades:

- **Completitud:** Si el *input* es válido, el protocolo siempre va a retornar *true*.
- **Robustez:** Si el *input* es inválido, entonces es teóricamente imposible engañar al protocolo y construir una prueba que retorne *true*. Por lo tanto, un probador deshonesto no puede engañar a un verificador honesto con una prueba que valida una afirmación falsa.
- **Conocimiento cero:** El verificador no conoce nada acerca de la afirmación más allá de si es verdadera o falsa. Esta propiedad evita también que el verificador descubra o derive cuál fue la afirmación original a partir de la prueba.

Una forma de entender cómo funcionan las ZKP es con el ejemplo de la cueva de Alibaba, en la figura 2.1 se puede observar el ejemplo ilustrado. En la historia tenemos 2 personajes, Victor el *verifier* y Peggy la *prover*. Peggy conoce la palabra mágica para abrir la puerta de la cueva de Alibaba y quiere demostrarle a Victor que ella conoce la palabra mágica, pero sin rebelársela a él (a Peggy no le gusta compartir). Para esto, Peggy entra por alguna de las 2 entradas a la cueva A o B, mientras Victor espera sin observar que camino elige Peggy. Luego Victor entra a la cueva y le indica a Peggy el camino por el cual debe de salir. Por último, Peggy sale por el camino indicado por Victor y de ser necesario abre la puerta.

Observando el ejemplo se puede apreciar que Peggy puede o no tener necesidad de abrir la puerta de la cueva. Si al inicio ella adivina qué camino le va a indicar Victor para salir, ella podría entrar por dicho camino desde un inicio. De esta forma puede cumplir con la indicación sin abrir la puerta y por ende demostrar que sabe la palabra mágica sin conocerla realmente. La probabilidad de que esto ocurra es del 50 %, pero si la historia se repite una segunda vez, la probabilidad de que Peggy adivine 2 veces el camino que le permite salir sin abrir la puerta es del 25 %. De esta forma, repitiendo la prueba n veces, la probabilidad de que Peggy no pueda abrir la puerta tiende a 0. (Jean-Jacques Quisquater, 1990)

Este tipo de pruebas se llaman pruebas ZKP interactivas, ya que se necesita de una interacción entre el *prover* y el *verifier* para llevar a cabo el protocolo

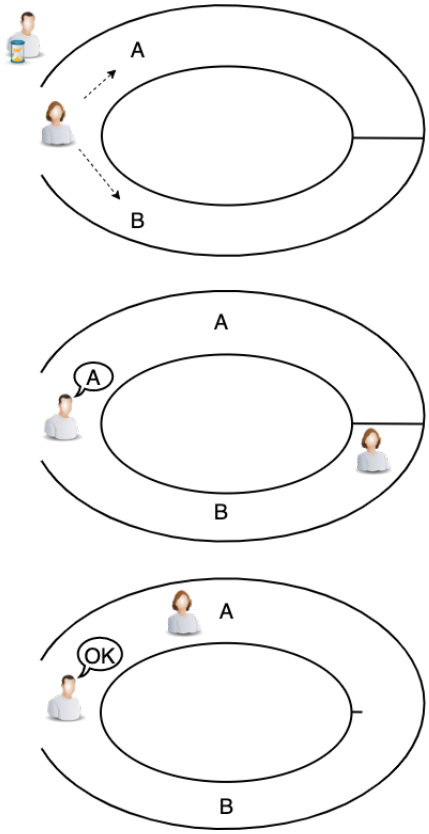


Figura 2.1: Representación del ejemplo de la cueva de Alibaba

de conocimiento cero. Sin embargo, existe otro tipo de pruebas que son no interactivas, en estas no es necesaria una interacción constante entre el *prover* y el *verifier*. En las pruebas no interactivas basta con que el *prover* le envíe la prueba al *verifier*, debido a que este último tiene toda la información que necesita en la prueba para asegurarse al 100 % que la afirmación es verdadera.

2.2.2. Soluciones de privacidad existentes

Se analizaron soluciones de privacidad existentes para estudiar cómo resolvían los problemas de privacidad en una *blockchain* pública. En particular, se investigaron qué técnicas o herramientas utilizaron para atacar la privacidad del mensaje y la privacidad de la identidad. Como este análisis era simplemente exploratorio para dimensionar el problema a atacar, se estudiaron las dos *blockchain* públicas más utilizadas que solucionan los problemas de privacidad: *Monero* y *ZCash*.

Monero

Monero (SerHack y the Monero Community, s.f.) es una criptomoneda que se enfoca en la privacidad y el anonimato de las transacciones. A diferencia de otras criptomonedas, *Monero* utiliza técnicas avanzadas de cifrado y anonimización para mantener la privacidad de las transacciones. Esto se logra mediante el uso de tres características: firmas en anillo, direcciones *stealth* y *ringCT*. En la figura 2.2 se puede apreciar en qué momento de la transacción actúa cada característica.

Figura 2.2: Participación de las características de seguridad de *Monero* en una transaccion

Las firmas en anillo (*ring signatures*) son una técnica de criptografía utilizada en *Monero* para proteger la privacidad de los usuarios en las transacciones. En lugar de firmar una transacción con la clave privada del remitente, se crea un grupo de claves públicas de otros usuarios de la red (llamados "firmantes"), junto con la clave pública del remitente. Esto forma un "anillo" de firmas posibles, de las cuales solo una es válida. De esta manera, la transacción es validada por la red sin que sea posible rastrear la identidad del remitente.

Las direcciones *stealth* en *Monero* son utilizadas para proteger la privacidad del destinatario de una transacción. Estas direcciones se generan a partir de la clave pública del destinatario y una clave aleatoria, y luego se cifran utilizando la clave pública del remitente. De esta manera, solo el remitente que posee la clave privada correspondiente puede descifrar la dirección y enviar los fondos. Además, *Monero* utiliza un protocolo de mezcla para ocultar aún más la dirección del destinatario en la transacción, lo que hace que sea más difícil de rastrear la transacción completa y la identidad del destinatario. En resumen, las direcciones *stealth* garantizan que las transacciones sean privadas y anónimas, haciendo difícil de rastrear la identidad del destinatario.

Monero utiliza el protocolo *RingCT* (*Ring Confidential Transactions*) para ocultar la cantidad de transacciones en la cadena de bloques. *RingCT* utiliza un sistema de cifrado avanzado para mostrar un rango de valores posibles en lugar de la cantidad exacta de una transacción, lo que aumenta el nivel de privacidad de las transacciones en *Monero*. Además, *RingCT* también utiliza técnicas de mezcla para ocultar la dirección del remitente y del destinatario en la transacción, lo que hace más difícil de rastrear la transacción completa

y la identidad de los usuarios. En resumen, *Monero* combina una variedad de herramientas y técnicas de privacidad, como las firmas en anillo, direcciones "*stealth*" y *RingCT*, para proteger la privacidad y el anonimato de los usuarios en sus transacciones.

ZCash

Zcash es una criptomoneda descentralizada que ofrece transacciones privadas y seguras mediante la utilización de criptografía de última generación. La privacidad en *Zcash* (Daira Hopwood, s.f.) se logra mediante el uso del protocolo *zk-SNARK* (ZCASH, s.f.), el cual permite a los usuarios realizar transacciones privadas sin revelar información personal o de transacciones a terceros. Cuando un usuario realiza una transacción en *Zcash*, la información se cifra y se envía a la red. Luego, se utiliza el protocolo *zk-SNARK* para verificar la validez de la transacción sin revelar detalles adicionales.

El protocolo *zk-SNARK* es una técnica de prueba de conocimiento cero que permite a los usuarios demostrar que tienen derecho a realizar una transacción sin tener que revelar información adicional, por ejemplo, el monto de dinero que se está transfiriendo o el saldo disponible del emisor. De esta forma, *zk-SNARK* permite que las transacciones se realicen de forma privada y segura.

Es importante mencionar que, aunque *Zcash* proporciona privacidad en las transacciones, las direcciones de las billeteras no son completamente anónimas. Las direcciones de las billeteras de *Zcash* son pseudónimos, lo que significa que los usuarios pueden tener varias direcciones para mantener su privacidad. Sin embargo, si una dirección de la billetera está vinculada a una identidad real, puede ser posible rastrear las transacciones realizadas con esa dirección.

En resumen, la privacidad en *Zcash* se logra mediante el uso del protocolo *zk-SNARK*, que permite a los usuarios realizar transacciones privadas sin tener que revelar información personal o de transacciones a terceros. Aunque las direcciones de las billeteras no son completamente anónimas, los usuarios pueden utilizar varias direcciones para mantener su privacidad.

2.2.3. Idemix

Idemix o *Identity Mixer*(Camenisch y Herreweghen, 2002), es una suite de protocolos desarrollados por IBM con el objetivo de crear un sistema de autenticación anónimo. En un sistema tradicional de autenticación que use certificados (por ejemplo X.509) en los que se guarda toda la información del usuario, automáticamente descalifica al sistema como anónimo. Además, en el caso de que solo se guarde una cantidad limitada de información, si el certificado es reutilizable, entonces varios usos del mismo certificado pueden ser vinculados, revelando la identidad del usuario.

Idemix implementa la capacidad de que la autoridad que se encarga de la autenticación reconozca a los usuarios solo a través de pseudónimos. Cada uso de la credencial es identificado con un seudónimo distinto, logrando así que no

se pueda encontrar una relación entre los mismos. Para ello hace uso de *Zero Knowledge Proofs* 2.2.1.

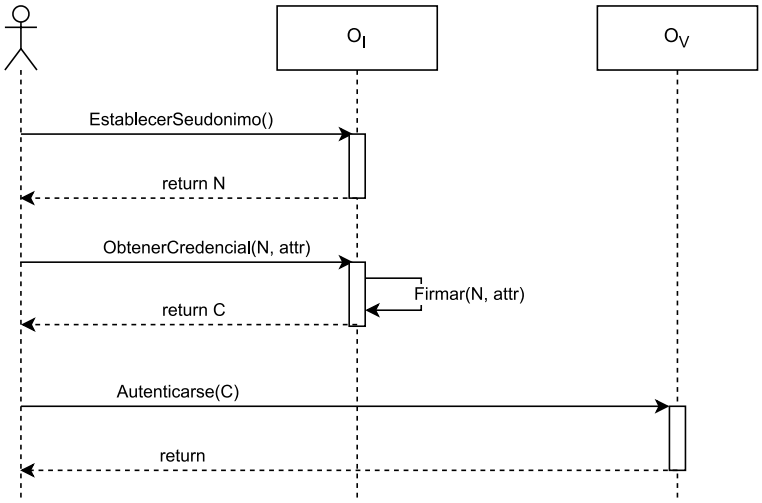


Figura 2.3: Funcionamiento de Idemix

Como se observa en la figura 2.3 el funcionamiento de *Idemix* consta de tres actores: un usuario U, una organización *issuer* O_I que se encarga de otorgar las credenciales y una organización verificadora O_V. Se hace uso de dos protocolos, un protocolo para pedir la credencial y otro para mostrarla. Para ello, U y O_I establecen un seudónimo N, y si U bajo el seudónimo N es capaz de pedir una credencial con un atributo $attr$, entonces O_I produce una credencial C constituida por una firma de N y $attr$, y es enviado a U. Luego, U para lograr la autenticación debe presentar su credencial obtenida ante O_V. Para ello, no mostrará en ningún momento su credencial, sino que utilizará un protocolo interactivo para dar una prueba *ZKP* con la cual se le probará a O_V que se tiene una credencial C. De esta manera, al no presentar la credencial, se logra que no pueda relacionarse este uso, con ningún otro futuro.

El uso de credenciales anónimas permite al usuario corroborar que cumple con un requisito sin revelar información innecesaria, permitiéndole realizar una acción en un sistema conservando su anonimato.

2.2.4. Private Smart contracts

Los *Private Smart Contract*, son un objeto de estudio al día de hoy y, aunque hay varias propuestas teóricas al respecto, las implementaciones prácticas son muy escasas actualmente. Por tal motivo, es difícil encontrar información en común entre las propuestas independientes presentadas para acuñar el término.

Así como los *smart contract* heredan las ventajas de las *Blockchain* públicas, como la disponibilidad, la inmutabilidad y otras características, también

heredan sus problemas de privacidad. Por tal motivo, existen propuestas para mejorar estas carencias de los *smart contract*. Aunque las diferentes soluciones propuestas siguen diferentes caminos y acercamientos al problema de privacidad, en general coinciden en que un *Private Smart Contract* cumple las siguientes características: (Ramsés Fernández-Valéncia, Jul 2, 2021)

- **Privacidad de la especificación:** El código de un *Private Smart Contract* debe de estar oculto durante el despliegue y los subsecuentes procesos de ejecución.
- **Privacidad de la ejecución:** El proceso de ejecución del *Private Smart Contract* en un nodo de la red verificador debe mantenerse oculto, así como sus argumentos y el valor de retorno generado.
- **Privacidad de estado:** El estado interno de un *Private Smart Contract* puede contener secretos de los usuarios.

Existen muchos proyectos teóricos abordando este problema, muchos basándose en ZKP 2.2.1 y operaciones *off-chain* como solución, pero también existen otros que utilizan criptografía homomórfica para asegurar la privacidad de las operaciones. (Solomon y Almashaqbeh, 2021)

Dentro de los trabajos que atacan este problema, las 3 principales soluciones encontradas al día de hoy son: Zether(Benedikt Bünz, 2020), Zkay(Steffen y cols., 2019) y Zexe (Bowe y cols., 2020). Dado que estas soluciones son teóricas e iniciales, donde no hay herramientas o tecnologías para aplicarlas, no se entra en detalle en su descripción.

2.2.5. zkEVM

Las zkEVM (*Zero Knowledge Ethereum Virtual Machine*) (José Segura, 5 octubre, 2022) son una evolución de la *Ethereum Virtual Machine*, que mejoran la eficiencia en la ejecución de los *smart contract* y la privacidad de los usuarios. Las zkEVM nacieron como necesidad de mejorar la experiencia de desarrollo de los programadores de *smart contract* a la hora de trabajar con pruebas de conocimiento cero 2.2.1.

Los programadores con la idea de mejorar la escalabilidad de la red de *Ethereum* desarrollaron los *Rollups*. Estos son una forma de escalabilidad que ejecuta transacciones en otra *blockchain* (denominada *Layer* 2), y posteriormente las agrupa y las manda a ejecutar en la *Layer* 1. Esto conlleva varios problemas de centralización y seguridad que para abolirlos se comenzaron a utilizar ZKP. Con ellas, se eliminaban estos inconvenientes, pero se agregabann nuevos. Los *smart contract* con estas mejoras eran difíciles de ejecutar con ZKP en la EVM y además era imposible o inviable portar viejas dApp a este nuevo enfoque. Las zkEVM nacen como solución a toda la problemática, brindando un entorno de ejecución compatible con las ZKP y al mismo tiempo con la EVM. Lo que permite ejecutar y portar viejas dApp a esta nueva tecnología. Como las zkEVM nacen de una necesidad concreta de la industria, casi no hay desarrollo

académico de este tema particular y en general es un avance de la industria misma.

Las zkEVM por su funcionamiento agregan características de privacidad sobre la red de *Ethereum*. Estas ejecutan los *smart contract* de forma privada, por lo tanto, los nodos no tienen conocimiento alguno de lo que sucede durante la ejecución. Además, los argumentos de entrada vienen cifrados y lo único que necesita la zkEVM es de un *witness* para generar el resultado y la prueba ZKP de que la ejecución del contrato fue válida. En la figura 2.4 se puede ver una ilustración de los datos de entrada de una zkEVM y los datos de salida que retorna la misma.

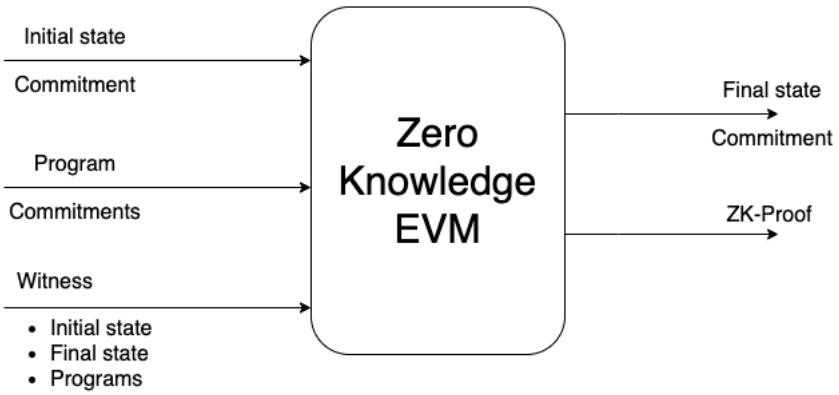


Figura 2.4: I/O de una zkEVM, Imagen basada en: (José Segura, 5 octubre, 2022)

2.2.6. Zokrates

Zokrates se autodefine como una *toolbox* para ayudar a los desarrolladores a construir pruebas ZKP sin tener demasiados conocimientos de criptografía. En otras palabras, su propósito es bajar la barrera de entrada a esta tecnología. Para esto ofrece un CLI, un lenguaje de programación y una librería para *JavaScript*, de modo que el desarrollador pueda programar pruebas y con el CLI o la librería generar las llaves de prueba y verificación necesarias para generar ZKP y el contrato en *Ethereum* que las verifique.

Como se puede observar en la figura 2.5, el primer paso para usar Zokrates es definir con el lenguaje de programación la prueba (1). Una vez la misma es definida, se compila a una representación intermedia (2) de la cual se puede computar el testigo (6) y generar las llaves de prueba (5) y de verificación (4). Por un lado, el encargado de generar las pruebas para demostrar la afirmación (el probador), necesita generar el testigo (6) con la afirmación más la llave de prueba (5), para así construir la ZKP (7). Por el otro lado, el encargado de verificar que la prueba es correcta (el verificador) va a desplegar en *Ethereum*

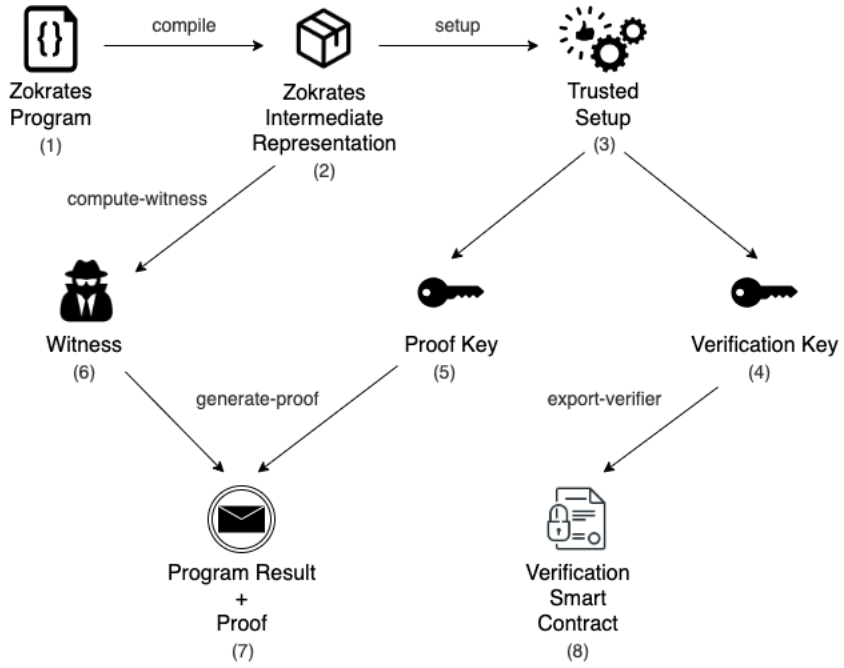


Figura 2.5: Ciclo de vida de Zokrates

el contrato de verificación (8) generado a partir de la llave de verificación (4). Una vez desplegado, con la prueba generada por el Probador y los argumentos de la prueba (7), el Verificador puede utilizar el contrato desplegado (8) para efectivamente verificar la prueba.

2.3. Interoperabilidad en blockchain

En las siguientes secciones se presentan conceptos orientados a la interoperabilidad entre *blockchain*. Se comienza comentando que se entiende por interoperabilidad según la visión de diferentes autores, los tipos de operaciones que se pueden efectuar al interoperar y sus principales limitaciones. También se da una breve explicación de las principales soluciones de la industria al problema de interoperar dos *blockchain* con la posibilidad de profundizar más en estas soluciones en el anexo C. Para finalizar, se dan los requerimientos de privacidad que se tienen que cumplir para garantizar una interoperabilidad segura

2.3.1. Que entendemos por Interoperabilidad

El IEEE define la interoperabilidad como “la capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información

intercambiada". Esta definición se puede aplicar al caso concreto de la interoperabilidad entre dos sistemas de *blockchain* ("IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries", 1991).

Existen muchos tipos de interoperabilidad en el ecosistema de *blockchain*, ya que las *blockchain* además de interactuar entre ellas, también pueden interactuar con componentes externos. Por lo tanto, se contemplarán los tres tipos de interoperabilidad más generales (R. Belchior y Correia, s.f.). Por un lado, la interoperabilidad entre dos o más *blockchain*, como se muestra en la figura 2.6 (B), tanto homogéneas, es decir, dos *blockchain* del mismo tipo, como heterogéneas en donde ambas *blockchain* son diferentes. Para este caso en concreto, el instituto nacional de estándares y tecnologías (NIST) nos brinda una definición para la interoperabilidad aplicada a *blockchain* y lo define como "Una composición de sistemas de cadenas de bloques distinguibles, cada uno de los cuales representa un libro mayor de datos distribuidos único, donde la ejecución de transacciones atómicas puede abarcar múltiples sistemas heterogéneos de cadenas de bloques, y donde los datos registrados en una *blockchain* son accesibles, verificables y referenciales por otra transacción posiblemente externa en una manera semánticamente posible". Por otro lado, existe la interoperabilidad entre *dApps* (también denominadas *businnes application*) con *blockchain* como se muestra en la figura 2.6 (A). Finalmente, también está la opción en que las *blockchain* envíen y/o consulten información a una dApp como se muestra en la figura 2.6 (C).

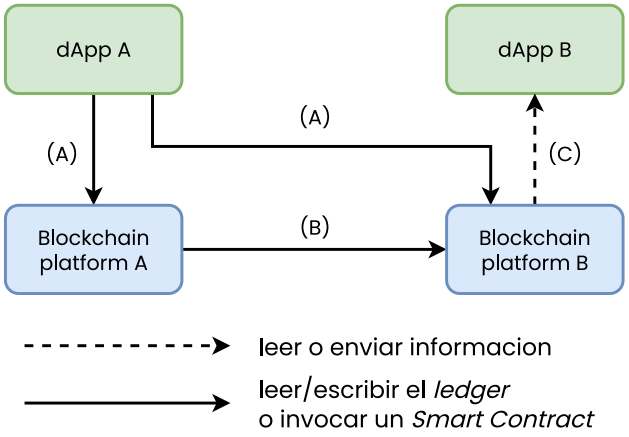


Figura 2.6: Tipos de interoperabilidad. Imagen basada en: (Guzman Llambias, s.f.)

2.3.2. Tipo de operaciones

Una *blockchain* de origen es una *blockchain* que emite una operación contra una *blockchain* de destino. Teniendo esto en cuenta podemos definir tres

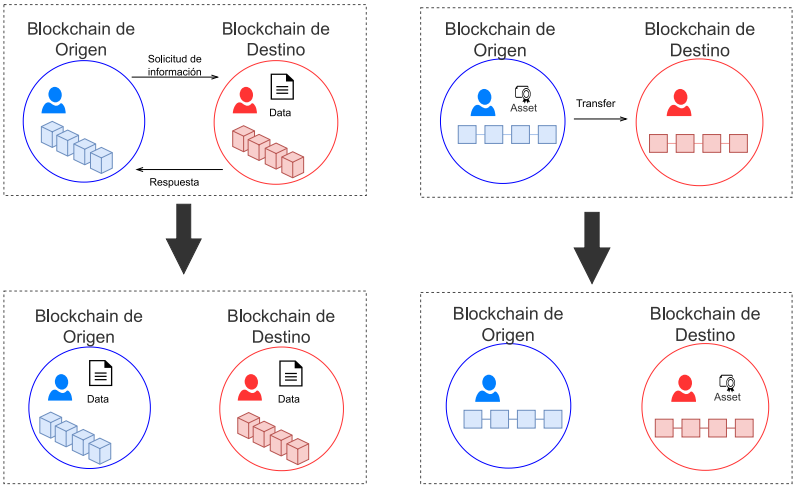
tipos de operaciones entre dos *blockchain*: *Transfer*, *Data sharing* y *Exchange*(Hyperledger.org, February 16, 2023).

Data sharing

Como se puede observar en la figura 2.7a, la operación de *Data sharing* hace referencia a una operación de consulta de información desde una *blockchain* origen a una *blockchain* destino. Este tipo de consulta puede ser tan variada como una consulta de una transacción monetaria entre dos usuarios o la existencia de un *asset* especifico.

Transfer

Como se puede observar en la figura 2.7b, este tipo de operación es unidireccional, desde una *blockchain* de origen a una *blockchain* de destino. La operación incluye algún dato que debe de enviarse a la *blockchain* de destino con alguna prueba correspondiente de que ha pasado el protocolo de consenso en la *blockchain* de origen. En la *blockchain* de destino se validará la prueba y se procederá, en caso de que sea necesario, a guardar esta información en su propio *ledger*.



(a) Operación de *Data sharing* (b) Operación de *Transfer*

Figura 2.7: Operaciones de Data sharing y Transfer

Exchange

En este caso se requiere realizar una tarea coordinada entre ambas *blockchain* para intercambiar objetos de información, cumpliendo con las propiedades de atomicidad y de consistencia de las transacciones ACID. Se realiza una acción

desde la *blockchain* de origen enviando un mensaje a la de destino, luego ambas *blockchain* deben de ejecutar las acciones. Si la ejecución de la *blockchain* de destino falla, entonces la *blockchain* de origen debe revertir sus cambios para garantizar la consistencia de datos. En la figura 2.8 se puede observar como dos usuarios que se encuentran en dos *blockchain* distintas realizan una operación de *Exchange* para intercambiar sus *assets*. En este caso, el usuario X le da su *asset* M al usuario Y en la *blockchain* de origen y al mismo tiempo el usuario Y en la *blockchain* de destino le entrega su *asset* N.

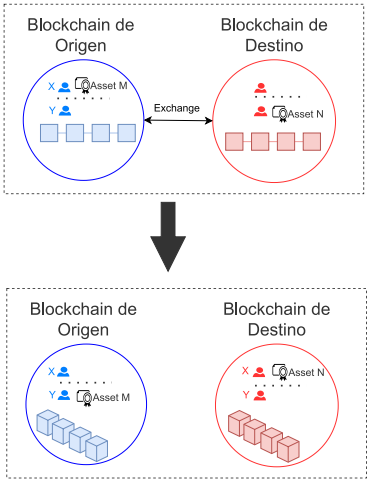


Figura 2.8: Operacion de *Exchange*

2.3.3. Soluciones actuales en la industria

A continuación, se da un breve panorama de las soluciones que existen hoy en día para poder interoperar dos *blockchain*. (Guzman Llambias, s.f.). Para más información sobre estas soluciones, ver el anexo C.

Notary Scheme

El *Notary Scheme* es un método que se centra en la participación de un tercero encargado de coordinar las *blockchain* que desean interoperar. De esta forma, el *Notary Scheme* monitorea los eventos de una *blockchain* que involucra modificaciones en otra *blockchain* y lleva a cabo las operaciones necesarias en ambas *blockchain* para lograr la interoperabilidad.

Atomic Swaps

Atomic Swap es un método que permite el intercambio seguro y confiable de activos entre diferentes *blockchain* sin la necesidad de un intermediario centralizado. En esencia, un *Atomic Swap* es una transacción atómica entre dos

partes que se lleva a cabo de forma instantánea y segura, garantizando que ambas partes reciban el activo que acordaron intercambiar sin riesgo de fraude o incumplimiento. Este método funciona mediante la creación de contratos inteligentes (*smart contract*) entre dos *blockchain* diferentes, donde se establecen las condiciones y términos del intercambio. Los contratos inteligentes están diseñados para asegurar que cada parte reciba su activo acordado en el intercambio, de manera que si una parte no cumple con los términos del contrato, el intercambio se cancela y se devuelve el activo a cada parte original.

Sidechain/Relays

El mecanismo de *Sidechain/Relay* es una solución para interoperar, escalar y actualizar *blockchain*. Consiste en dos *blockchain*, una *main chain* que mantiene el *ledger* principal y una *Sidechain* que maneja su propio *ledger* y algoritmo de consenso. La comunicación puede ser unidireccional o bidireccional, y se logra a través de un protocolo de comunicación y un *Relay*, que verifica las transacciones recibidas por agentes externos. Esto puede usarse para implementar un canal de pagos o agregar funcionalidad a una *blockchain*, como los *smart contract* en Bitcoin.

Blockchain of blockchains

La solución de *blockchain of blockchains* consiste en un protocolo de consenso que organiza y valida los bloques de varias *blockchain* interconectadas. La solución se parece a la de *Sidechain/Relay*, pero en este caso hay una cadena principal *(Relay chain)* que conecta varias *blockchain* secundarias *(Parachains)* personalizadas y optimizadas para diferentes casos de uso. La solución se basa en soluciones de la industria como *Polkadot* y *Cosmos*, y se necesitan bloques de construcción como *Relay*, Validadores, *Connectors* y *Routers*.

Enterprise connectors

En el artículo (Guzman Llambias, s.f.) se presenta la evolución de la categoría *Hybrid Connectors* a *Enterprise Connectors*, que se aplica más a empresas con cierto grado de confianza. Estos se dividen en subcategorías como *Gateway*, *Generic Interface* y *Enterprise Relays*, que tienen propiedades, arquitectura y características similares. El *Gateway* es un software que actúa como intermediario entre dos *blockchain*, transformando los mensajes enviados por la *blockchain* de origen para que sean comprensibles por la *blockchain* de destino. *Generic Interface* permite a un sistema de origen comunicarse con múltiples *blockchain* utilizando una API normalizada, y *Enterprise Relays* es una solución que puede aplicarse de manera descentralizada y redirige las transacciones de una *blockchain* de origen a una de destino, utilizando mecanismos de adaptación de mensajes, conectores y enrutamiento.

2.3.4. Requisitos de privacidad al interoperar

A partir de trabajos académicos y literatura relevada, se concluyó que los requisitos de privacidad deseados en *blockchain* al momento de interoperar son la privacidad de la identidad, la privacidad del mensaje y la privacidad del canal utilizado (Rui Zhang, s.f.) (Dan Wang, s.f.) (SerHack y the Monero Community, s.f.) (Bernal Bernabe y cols., 2019).

Privacidad de la identidad

La privacidad de la identidad hace referencia a mantener la identidad de los actores de una transacción de manera oculta o protegida. De modo que un actor ajeno al intercambio no sea capaz de distinguir a través de la observación la verdadera identidad de las partes involucradas. Para conseguir esto, es vital que las transacciones posean la facultad de ser intrazables, siendo la *intrazabilidad* la facultad de no poder vincular de ninguna manera dos entidades observadas, en este caso, dos o más transacciones. Esto se debe a que comúnmente en las *blockchain* públicas los ataques usados para revelar el *pseudoanonimato* de los usuarios, es utilizando algoritmos que evalúan su comportamiento. Estos analizan las distintas direcciones de la red hasta descubrir comportamientos que eventualmente deriven en la obtención de información sensible, revelando así la identidad del usuario en la red. En caso de tener las dos facultades, es decir, el *pseudoanonimato* en la red y la *intrazabilidad* de las transacciones, podemos garantizar la privacidad de la identidad. Ya que, de esta manera, los actores de la transacción se mantienen ocultos bajo el *pseudoanonimato* de la red y nadie puede trazar su identidad.

Privacidad del mensaje

La privacidad del mensaje hace referencia no solo a la necesidad de mantener el contenido del mensaje grabado en el *ledger* privado, sino también a las partes involucradas en la transmisión de dicho mensaje. Es decir, la información transferida, así como los participantes en la comunicación, tienen que ser confidenciales. Solo los actores involucrados deben de poseer la capacidad de acceder al contenido del mensaje grabado en el *ledger* de la *blockchain*. Pudiendo ser dicho contenido la información sensible del dominio de la aplicación descentralizada, un *smart contract*, etc. La privacidad del mensaje se enfoca en impedir que se vincule la información sensible transferida y la dirección emisora/receptora del mensaje. De modo que, un tercero no pueda utilizar la información sensible transmitida para descubrir la identidad del usuario real detrás de la dirección pseudoanónima de este en la red.

Se decidió además incluir en la definición, en el contexto del proyecto realizado, la ejecución de los *smart contract*. En aquellas *blockchain* que soporten *smart contracts* y estos sean el mensaje de la transacción, puede ocurrir que un nodo malicioso de la red que ejecute el *smart contract* también lo depure, accediendo al valor de las variables o de los parámetros en un momento dado. Por tal motivo, es importante prestar atención a no manejar información sensible en

el *smart contract* en ningún momento, o en caso de hacerlo, que la misma esté ofuscada.

Con esta modificación, la definición de privacidad del mensaje utilizada en el contexto del proyecto es: la privacidad del mensaje consta en que solo los involucrados en la transacción puedan acceder a la información almacenada en el *ledger* de la *blockchain*. Además, no debe de ser posible vincular la información sensible relacionada a la transacción con las direcciones del emisor/receptor de la transferencia. Por último, los *smart contract* no deben de manipular información sensible en su ejecución o la misma debe ser ofuscada, de modo que no se pueda acceder a ella depurando el *smart contract*.

Privacidad del canal

Todo dispositivo conectado a internet tiene asignada una dirección IP que puede fácilmente ser trazada para obtener la localización geográfica del dispositivo y descubrir así la identidad del usuario que lo utiliza. Cuando un nodo de una red *blockchain* (véase Bitcoin) envía un *broadcast* para que su transacción sea ejecutada, dicha petición lleva consigo la dirección IP del usuario. Aunque la misma no sea grabada en el *ledger* ni utilizada de forma activa en la lógica de la *blockchain*, un nodo malicioso podría eventualmente monitorear las direcciones IP y generar un *log* donde vincule la dirección de la *blockchain* con la dirección IP del usuario. Por tal motivo, es importante que la *blockchain* garantice la privacidad del canal, de modo que la dirección IP del usuario y, por ende, su localización geográfica sea confidencial.

2.4. Trabajo relacionado

Solo se encontró un trabajo relacionado que abarca privacidad e interoperabilidad al mismo tiempo, por tal motivo la mayoría de los artículos mencionados se enfocan en uno de los dos problemas a resolver.

El artículo *"Security and Privacy on Blockchain"* (Rui Zhang, s.f.) presenta una visión general de los desafíos en temas de privacidad y seguridad en *blockchain*. Expone cuáles son las propiedades básicas esenciales que debería de poseer una *blockchain* en términos de seguridad. También lista las propiedades de privacidad que son deseables en muchas aplicaciones de *blockchain*. Asimismo, el artículo *"A survey on privacy protection in blockchain system"*(Feng, He, Zeadally, Khan, y Kumar, 2019) se centra en el análisis de los desafíos de la privacidad en *blockchain*. Presenta los requisitos de privacidad de la identidad y de la transacción/mensaje, exponiendo formas de quebrantar dichos requisitos en las *blockchain* mayormente utilizadas como bitcoin. Finalmente, el artículo *"Privacy-Preserving Solutions for Blockchain: Review and Challenges"*(Bernal Bernabe y cols., 2019) realiza un análisis exhaustivo sobre la privacidad en *blockchain*. Este presenta las vulnerabilidades que presentan las *blockchain* en tema de privacidad. Siendo este capaz de identificar el requisito de la privacidad del canal. También presenta una taxonomía de soluciones para

preservar la privacidad, incluyendo un análisis de sus ventajas y desventajas.

Estos trabajos previos permitieron reconocer los problemas que existen con la privacidad en *blockchain*. En particular, se extrajeron los requisitos de privacidad que deben de poseer las *blockchain* y se analizaron y consideraron las soluciones sugeridas para los mismos.

Por el lado de interoperabilidad, se analizaron varios artículos que relevan el estado del arte de las soluciones existentes. El artículo *"A Survey on Blockchain Interoperability: Past, Present, and Future Trends"* (R. Belchior y Correia, s.f.) hace un relevamiento sobre un corpus de 402 documentos. Los autores identifican 67 soluciones de interoperabilidad existentes, además discuten las tecnologías, estándares de soporte, los casos de uso, los desafíos abiertos y las direcciones futuras de investigación. En la misma línea, el artículo *"Blockchain Interoperability: a Feature-based Classification Framework and Challenges Ahead"* (Guzman Llambias, s.f.) realiza una clasificación en categorías y subcategorías de 62 documentos científicos que abordan la interoperabilidad en *blockchain*. El artículo también discute el panorama actual de la interoperabilidad en *blockchain*, identificando los desafíos y sugiriendo direcciones futuras de investigación. Por último, el artículo *"Security and Privacy Challenges in Blockchain Interoperability - A Multivocal Literature Review"*(Haugum, Hoff, Alsadi, y Li, 2022) realiza una revisión bibliográfica sobre los desafíos de privacidad al interoperar. Este, discute posibles mitigaciones de los problemas de privacidad y se destacan los desafíos abiertos que surgieron a partir de las soluciones analizadas.

Estos análisis facilitaron la realización del proyecto, ya que permitieron contextualizar el estado actual del campo de la interoperabilidad en *blockchain*. Además, aquellos que también abordaron los problemas de privacidad en su análisis brindaron puntos de inicio para continuar su investigación. Y así poder abordar la interoperabilidad en *blockchain* pública-privada propuesta en este proyecto.

Finalmente, se analizó el artículo *"A gateway based interoperability solution for permissioned blockchains"*(Bruno Bradach, s.f.) el cual describe una implementación de la solución de interoperabilidad de tipo *gateway*. Dicha implementación fue utilizada como base por el proyecto de grado: "Interoperabilidad entre plataformas de blockchain"(Mathías Castro, 2023). Este extiende y mejora la implementación, otorgándole la capacidad para interoperar una *blockchain* pública con una privada, en particular *Ethereum - Hyperledger Fabric*.

Capítulo 3

Análisis de requerimientos

El capítulo inicia con una descripción de los requerimientos del proyecto en la sección 3.1. Luego, en la sección 3.2 se establece cuál fue el alcance logrado de los requerimientos establecidos. A continuación, en la sección 3.3 se presenta una descripción del escenario a implementar. Finalmente, en la sección 3.4, se listan los casos de uso obtenidos de la descripción del escenario.

3.1. Requerimientos del proyecto

En la tabla 3.1 se listan los requerimientos iniciales del proyecto.

Identificador	Descripción
REQ 1	Proveer una solución de interoperabilidad entre *Ethereum* y *Fabric* que tenga en cuenta los requerimientos de privacidad de *blockchain* definidos en la sección 2.3.4.
REQ 1.1	Realizar una operación de tipo *Data sharing*, lo que implica ejecutar una consulta de información a *Fabric* desde un *smart contract* en *Ethereum* y guardar esa información en la *blockchain* de *Ethereum*.
REQ 1.2	Realizar una operación de tipo *Exchange* que permita a partir de la ejecución de un *smart contract* en *Fabric*, desencadenar la ejecución de un *smart contract* en *Ethereum* que le asigne a un usuario de esta *blockchain* un NFT.
REQ 2	Implementar el escenario propuesto descrito en la sección 3.3, utilizando una solución de interoperabilidad que cumpla con los requerimientos de privacidad señalados en REQ 1.

Cuadro 3.1: Requerimientos del proyecto

Dado los requerimientos REQ 1.1 y REQ 1.2, se debe de tomar en cuenta ciertas consideraciones para poder cumplir con los requerimientos de privacidad al interoperar mencionados en el requerimiento REQ 1:

- **Privacidad del mensaje:** La solución deberá contemplar una forma de ofuscación de los parámetros para guardar la información en el *ledger*. Esto se debe a que la privacidad del mensaje será quebrantada en dos oportunidades. La primera es cuando se llama al contrato en *Ethereum* y la información de los parámetros queda expuesta. Siguiendo en la misma línea, durante la ejecución del *smart contract* en los nodos de *Ethereum* se podría exponer información sensible. Por lo tanto, el código en los *smart contract* no debería manipular información sensible sin ser previamente ofuscada. La segunda oportunidad donde se quebranta la privacidad del mensaje es cuando *Ethereum* guarda en el *ledger* la información recibida de *Fabric*.

- **Privacidad de la identidad:** Se debe de encontrar una forma de autenticarse desde la *blockchain* pública en la *blockchain* privada sin dar a conocer información de la identidad del usuario. De modo que al autenticarse en *Fabric* no se revele la identidad del usuario en *Ethereum*. En el escenario propuesto, la privacidad de la identidad será quebrantada cuando el usuario intente autenticarse desde la *blockchain* pública (*Ethereum*), en la *blockchain* privada (*Fabric*). Ya que al autenticarse en la red de *Fabric* se conocerá la identidad del usuario en *Ethereum* y su identidad quedará expuesta.

- **Privacidad del canal:** Este tipo de privacidad no será tomado en cuenta en la interoperabilidad. Dado que dicha vulnerabilidad está asociada al uso de internet y diseñar e implementar una solución de dicha magnitud, se escapa completamente del alcance del proyecto.

Además, con el fin de que el escenario pueda ser implementado para cumplir con una lógica de negocio que justifique su implementación con *blockchain* es que se decidió extender la operación de *Data sharing* definida en la sección 2.3.2. De este modo, la operación de *Data sharing* no solamente consistirá en realizar una consulta de información sobre la *blockchain* de *Fabric*, sino que también se realizará una acción sobre esta.

3.2. Alcance del proyecto

El diseño de la solución y la investigación realizada sobre la privacidad demandaron una gran cantidad de tiempo y de recursos. Por tal motivo, y en conjunto con el tutor, se decidió dejar fuera del alcance el requerimiento REQ 1.2 que consistía en la implementación de la operación de interoperabilidad de *Exchange*. El alcance logrado al finalizar el proyecto fue del cumplimiento de los requerimientos REQ 1, REQ 1.1 y REQ 2.

3.3. Descripción del escenario

Con el fin de tener mejor trazabilidad en un formato electrónico, las mutualistas necesitan tener almacenada la información médica de los usuarios de manera confiable como: los historiales médicos, las consultas realizadas, las prescripciones otorgadas, entre otros. Es por esto que las mutualistas utilizan una *blockchain* privada como *Fabric* para guardar de forma confiable esta información. Por otro lado, las farmacias utilizan una *blockchain* pública como *Ethereum* para transparentar el uso de las prescripciones emitidas por las mutualistas y de esta manera poder brindar otro método de pago para los medicamentos. Al ser una *blockchain* pública se permite garantizar la trazabilidad de las prescripciones utilizadas y evitar su reutilización. De esta manera, se logra una mayor transparencia en el proceso de emisión y uso de las prescripciones entre farmacias y mutualistas. Además de las *blockchain*, tanto las mutualistas como las farmacias cuentan con sus propias terminales de autoservicio que les permiten a los usuarios interactuar con las *blockchain*. Como se puede ver en la figura 3.1, el escenario comienza con el doctor ingresando una prescripción para el paciente en el sistema de la mutualista. Para esto, ingresa el identificador del paciente y la descripción del medicamento (1) en la terminal de autoservicio de su consultorio. Esto permitirá agregar una nueva prescripción a la *blockchain* de la mutualista y asociarla a un paciente. Luego, el paciente ingresa a una terminal de autoservicio en la mutualista con sus credenciales y esta le brinda un comprobante de la prescripción recetada por el médico (3). Posteriormente, el usuario se dirige a la farmacia y manipulando una terminal de autoservicio ingresa los datos para solicitar el medicamento (5). Los datos que ingresa son sus credenciales básicas para autenticarse en la mutualista, y los datos de la prescripción. De esta forma, el sistema de la farmacia consulta por medio de un *smart contract* a la *blockchain* de la mutualista si la prescripción es válida y si la misma está asignada a ese paciente (7). En caso de que la prescripción no sea válida o no esté asignada al paciente, se le mostrará un mensaje de error en la terminal. En caso contrario, la solicitud será válida y se le mostrará un mensaje en la terminal indicando que su medicamento será despachado en el mostrador. Además, en el sistema de la mutualista se consumirá la prescripción, es decir, se marcará la misma como consumida para que ya no se pueda volver a utilizar.

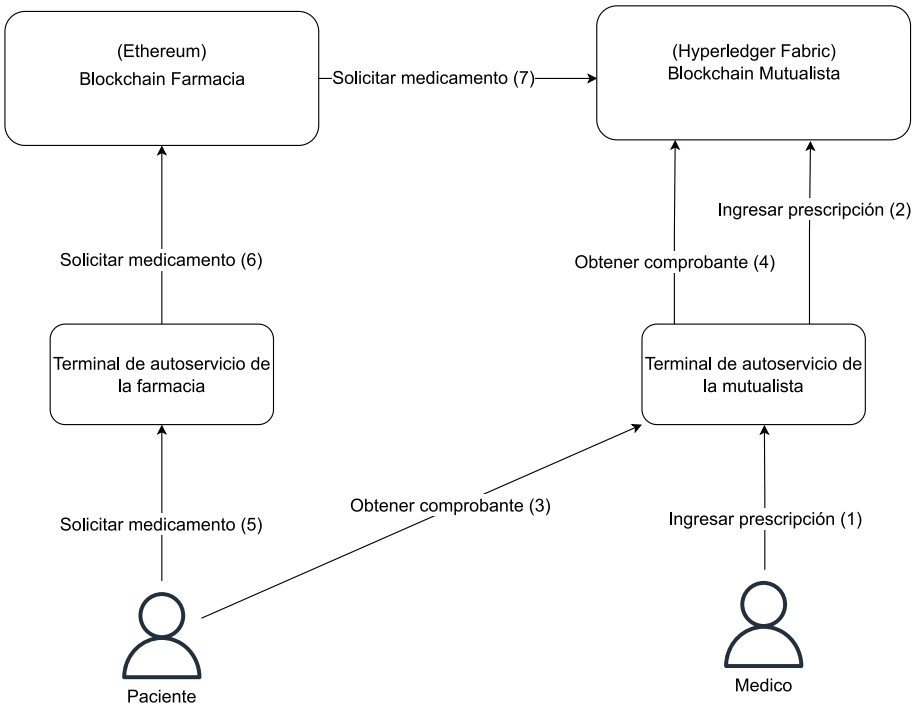


Figura 3.1: Descripción del escenario

3.4. Casos de uso

A continuación, se presentan los actores del escenario y como estos interactúan con los casos de uso obtenidos de la descripción del escenario de la sección 3.3.

3.4.1. Actores

En el escenario a implementar se logran distinguir dos tipos de actores diferentes

- **Pacientes**: Son los usuarios que están asociados a una mutualista y pueden hacer uso de sus servicios.
- **Médicos**: Son quienes trabajan en las mutualistas y atienden a los pacientes en las consultas. Pueden recetar prescripciones médicas a los pacientes.

3.4.2. Casos de uso

Basándose en la descripción del escenario de la sección 3.3 y los requerimientos de privacidad que tiene que contemplar la solución, se pueden identificar tres

casos de uso específicos: 1) Ingresar prescripción, 2) Obtener datos de prescripción para farmacia y token de autenticación y 3) Usar prescripción. Cada uno de estos casos de uso representa una parte esencial del proceso de interoperabilidad deseado. Los actores interactúan con los casos de uso como se muestra en la imagen 3.2.

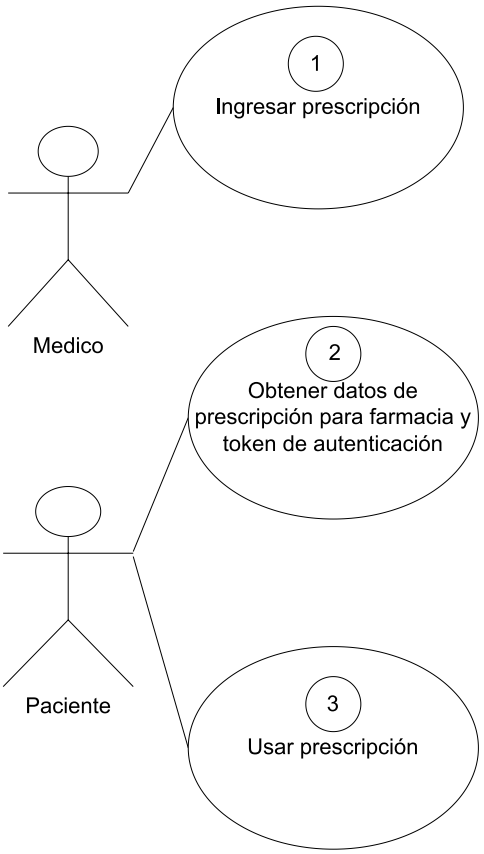


Figura 3.2: Interacción de actores y casos de uso

A continuación, se detallará cada uno de los casos de uso mencionados anteriormente.

Nombre	Ingresar prescripción
Actores	Médico
Actividades	Dar de alta una prescripción
Descripción	El médico que atiende al paciente en su consulta carga en la *blockchain* de la mutualista los datos necesarios para dar de alta una prescripción. Para esto, ingresa en una terminal de la mutualista el identificador del paciente y el nombre del medicamento. Dicha operación concluye con el retorno de un identificador de la prescripción creada y se le da al paciente para que pueda utilizarla en una farmacia.
Pre-condiciones	Existe el usuario paciente al que se le quiere dar la prescripción.
Post-condiciones	Se crea una prescripción en la *blockchain* de la mutualista y esta se la asocia a un paciente.

Nombre	Obtener datos de prescripción para farmacia y token de autenticación
Actores	Paciente
Actividades	Obtener información asociada al método de autenticación que no expone la identidad del usuario y datos de la prescripción
Descripción	El paciente debe ingresar sus credenciales en la terminal de la mutualista para obtener la información de la prescripción y un token de autenticación para presentar en la farmacia.
Pre-condiciones	Existe el usuario paciente en la mutualista.
Post-condiciones	El usuario recibe un token de autenticación e información de la prescripción recetada por el médico. La prescripción contiene un ID y una descripción.

Nombre	Usar prescripción
Actores	Paciente
Actividades	Se consume la prescripción asignada en caso de que exista
Descripción	Para hacer uso de la prescripción, el paciente ingresar los datos de la misma, junto con un token de autenticación. Si la prescripción es válida y no ha sido usada previamente, se otorga al paciente el medicamento. En caso contrario, si la prescripción ya ha sido utilizada o no existe en el sistema, se le mostrará un mensaje de error indicando que no se puede hacer uso de ella.
Pre-condiciones	Existe el usuario paciente en la mutualista.
Post-condiciones	Se marca como utilizada la prescripción.

Capítulo 4

Conceptualización de la solución

En este capítulo se abordan los estudios realizados para delimitar el problema y conceptualizar una posible solución para el mismo. En la sección 4.1 se desarrolla cuál fue la metodología de trabajo utilizada. Posteriormente, en la sección 4.2 se expone el análisis realizado sobre las soluciones existentes de interoperabilidad y se brinda una conclusión de este en la sección 4.3. A continuación, en la sección 4.4 se realiza un análisis sobre la solución de *Gateway* para ver dónde se quebrantan los requisitos de privacidad en esta. Luego, se enumeran los diferentes abordajes utilizados para conceptualizar la solución al problema de privacidad en la sección 4.5. Cerrando dicha sección con el abordaje seleccionado para la solución final. Por último, en la sección 4.6 se brinda una conclusión final sobre la conceptualización de la solución.

4.1. Metodología de trabajo

A continuación, se describen los pasos que siguió el equipo para conceptualizar la solución, estos se pueden observar en la figura 4.1.

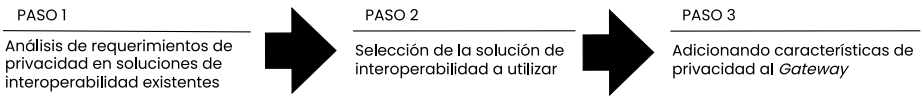


Figura 4.1: Pasos de la metodología de trabajo utilizada en la conceptualización de la solución.

Análisis de requerimientos de privacidad en soluciones de interoperabilidad existentes

Una vez definidos los requisitos de privacidad al interoperar, se procedió a relevar las soluciones preexistentes para atacar el problema de interoperabilidad entre dos *blockchain*. Las soluciones se pueden dividir según los siguientes criterios: si son soluciones descentralizadas o no, los tipos de operaciones que permiten realizar, y las propiedades que se mantienen al interoperar. Dado que el objetivo es interoperar dos *blockchain* cumpliendo los requisitos de privacidad, es necesario ver si alguna solución preexistente los conserva por defecto. Los análisis se realizaron solución a solución, estos consistieron en analizar cómo interactúan las *blockchain* con cada una de las posibles operaciones a realizar. En particular, se observó si la ejecución de las operaciones respetaban los requisitos de privacidad.

Selección de la solución de interoperabilidad a utilizar

Una vez terminado el análisis sobre las soluciones, se decidió cuál de estas utilizar para diseñar e implementar la solución final. El primer paso fue estudiar si ya existían implementaciones de las soluciones que implementaran los requisitos de privacidad al interoperar. Además, se investigó cuáles de estas eran funcionales para interoperar las *blockchain* de *Ethereum* e *Hyperledger Fabric*. Dado que se trata de un proyecto de grado de Ingeniería, la complejidad de la solución y la posibilidad de optimizar los recursos disponibles se vuelven elementos de especial valor a la hora de evaluar las soluciones. Teniendo esto en mente, la posibilidad de utilizar la implementación de *Gateway* construida en el proyecto de grado «Interoperabilidad entre plataformas de blockchain» (Mathías Castro, 2023) de la UDELAR, se tornó particularmente interesante. Aun así, se evaluaron más soluciones con el objetivo de contrastarlas contra la solución de *Gateway*. Finalmente, luego de estudiar las alternativas, se terminó eligiendo la solución de *Gateway* mencionada.

Adicionando características de privacidad al Gateway

Una vez seleccionada la solución para interoperar las *blockchain*, el siguiente paso fue evolucionar su arquitectura para cumplir con los requisitos de privacidad a la hora de interoperar. Con esto en mente se identificaron las diferentes etapas que se suceden en el procesamiento de una transacción entre las *blockchain* utilizando la solución de *Gateway*.

Al finalizar el reconocimiento de las etapas que conlleva procesar una transacción, se pudieron identificar en qué etapas se infringen los requisitos de privacidad. La privacidad de la identidad se abordó estudiando *Idemix* 2.2.3, herramienta que fue sugerida por el tutor para, luego del estudio, agregarlo a la arquitectura del *Gateway* y solucionar la privacidad de la identidad. La privacidad del mensaje requirió varias iteraciones y se exploraron varias alternativas hasta conseguir una solución utilizando pruebas de conocimiento cero 2.2.1.

4.2. Análisis de privacidad en soluciones de interoperabilidad

En esta sección se encuentra la tabla 4.1 con las soluciones de interoperabilidad preexistentes relevadas. Para cada solución se indica qué requisitos de privacidad satisface y cuales no, según el tipo de operación a realizar (*Data Sharing*, *Exchange*, *Transfer*). Además, se brinda una breve explicación de por qué satisface o no los requisitos. Dado que en el objetivo del proyecto se plantea interoperar una *blockchain* pública con una privada, en este análisis solo se tuvo en cuenta dicho par. Más detalles de este análisis se encuentra en el anexo E.

Solución	Requisitos de privacidad satisfechos	Requisitos de privacidad no satisfechos.
Notary Scheme	Se cumple el requisito de privacidad de la identidad para las operaciones de *Transfer* y *Data Sharing*, siempre que el *Notary* no sea un agente malicioso. Para la privacidad del mensaje, con las operaciones de tipo *Data Sharing*, la privacidad se va a mantener siempre que el *Notary* no sea un agente malicioso.	Para todas las operaciones y requisitos de privacidad, si el *Notary* es un agente malicioso, va a poder filtrar información sensible y quebrantar cualquiera de los requisitos. En caso de que el *Notary* no sea malicioso, aún existen casos donde se quiebren los requisitos de privacidad. La mayoría se quiebran cuando la información interactúa con la *blockchain* pública.
Atomic Swap	La solución no satisface ningún requisito de privacidad.	Todos los requisitos de privacidad se ven quebrantados cuando la información interactúa con la *blockchain* pública, ya que esta no satisface ningún requisito.
Sidechain /Relay	La solución no satisface ningún requisito de privacidad.	Esta solución no contempla ningún tipo de ofuscación sobre las transacciones. Además, es necesario que los usuarios de la *blockchain* pública se autentifiquen en la privada exponiendo su identidad. Teniendo todo lo anterior en cuenta, la solución no satisface ningún requisito de privacidad.
Gateway/ Generic Interface/ Enterprise Relay	El análisis para los *Enterpise Connectors* es similar al *Notary*. Cuando el conector es de confianza y no es un agente malicioso, se satisface el requisito de privacidad de la identidad para las operaciones *Transfer*.	En el caso de la operación de *Data Sharing*, la privacidad de la identidad se ve quebrantada cuando el usuario de la *blockchain* pública se autentifica en la *blockchain* privada. Las operaciones de *Exchange* exponen la identidad del usuario en la *blockchain* pública cuando su información es manipulada. Por el mismo motivo no se satisfacen los requisitos de privacidad del mensaje ni de privacidad del canal para todos los tipos de operación.

Cuadro 4.1: Tabla con el análisis de privacidad realizado sobre las soluciones de interoperabilidad existentes.

Vale la pena mencionar que los *Enterprise Connectors* fueron estudiados en

conjunto, ya que son muy similares entre ellos y las diferencias entre los mismos no afectan al análisis realizado. Asimismo, los *Enterprise Connectors* son muy similares al *Notary Scheme* por lo que comparten muchas de sus propiedades. Por más detalles de esto último ver anexo E. Por otro lado, es importante mencionar que para la solución de *blockchain-of-blockchain* solo se analizó el par *público-público*, debido a la complejidad de este tipo de soluciones y que solo existen implementaciones utilizando *blockchain* públicas. Finalmente, el análisis anterior se realizó sobre el par público-privada, pero el mismo estudio también se realizó para el par público-público (ver anexo F), y privado-privado (ver anexo G).

4.3. Conclusión del análisis

El caso de uso propone una operación de tipo *Data Sharing* como ejemplo de interoperabilidad. Por tal motivo, se descartaron las soluciones de *Atmoic Swap*, ya que estas solo soportan operaciones de tipo *Exchange*. Por otro lado, la implementación de *Atomic Swap* requiere que haya sincronización entre las partes al realizar el intercambio de información. Esta complejidad también aportó a la decisión de descartar este tipo de soluciones.

Las soluciones basadas en *Blockchain of Blockchains* se descartaron por su alta complejidad. Teniendo en cuenta el tiempo y recursos acotados del proyecto, incursionar en este tipo de soluciones se escaparía totalmente del alcance.

Las soluciones de tipo *Sidechain/Relay* fueron descartadas porque no existe una implementación a la fecha del proyecto que contemple la interoperabilidad entre una *blockchain* pública y una privada. Además, no es viable construir una implementación desde cero que contemple dicho caso. Esto se debe a la complejidad de este tipo de soluciones y que no es el objetivo del proyecto. Tampoco se cuenta con los recursos para fabricar una *Sidechain* para *Etherum* con *Fabric*.

Por último, quedan analizar las soluciones de tipo *Notary Scheme* y *Enterprise Connectors*. Como se mencionó en la sección 4.2, estas soluciones son muy parecidas, sin embargo, se seleccionó la solución *Enterprise Connector*, en particular *Gateway*, dado que ya se contaba con una implementación de referencia (Mathías Castro, 2023).

El *Enterprise Relay* fue descartado debido a que agrega varias complejidades a la solución que no son requeridas. Por ejemplo, no se quería tener lógica de negocio asociada al caso de uso en la solución de integración. Por otro lado, *Generic Interface* se descartó debido a que las soluciones existentes no sirven para interoperar una *blockchain* directamente con otra *blockchain*.

4.4. Análisis de extensión del Gateway

Es importante observar que realizar una consulta de información sobre una *blockchain* pública, no deja rastro de la información del usuario y no requiere

autenticación. Por lo tanto, conservar la privacidad es trivial en una operación de tipo *Data Sharing* desde la *blockchain* privada. Es por esto que se identificaron únicamente los pasos de una transacción desde una *blockchain* pública hacia una privada.

Los pasos identificados de una transacción desde *Ethereum* sobre *Fabric* utilizando la solución de *Gateway* son los siguientes:

1. El Usuario ingresa en la dApp los datos que esta solicite para ejecutar la transacción.
2. La dApp llama al *smart contract* de *Ethereum* con dichos datos como entrada.
3. El *smart contract* en *Ethereum* emite un evento solicitando ejecutar una función de *Fabric* con todos los datos necesarios para ello.
4. El *Gateway* escucha el evento de *Ethereum*, inicia sesión en *Fabric* y le pide a este último ejecutar la función con los datos que recibió a través del evento.
5. *Fabric* ejecuta la función solicitada y envía el resultado hacia el *Gateway*.
6. El *Gateway* toma el resultado de *Fabric* y ejecuta una función de respuesta en el *smart contract* de *Ethereum* con el resultado como parámetro.
7. Ahora, con toda la información, el *smart contract* de *Ethereum* termina de procesar la solicitud de la dApp.
8. El *smart contract* de *Ethereum* guarda el resultado en el *ledger* de *Ethereum*.

Analizando los pasos identificados anteriormente, la privacidad del mensaje se vulnera en el paso número 2, ya que la función en *Ethereum* al procesar los datos de entrada los expuso debido a que estos no tienen ningún tipo de ofuscación. Además, también se quebranta la privacidad del mensaje en el paso número 8, ya que tampoco se ofuscan los datos de ninguna forma al guardarlos en el *ledger*. Por otro lado, la privacidad de la identidad se vulnera en el paso número 4, ya que el evento que emite *Ethereum* y escucha el *Gateway* contiene información privada que expone la identidad del usuario.

4.5. Desarrollo de la solución

En la sección anterior, se identificaron los pasos donde se vulneran los requisitos de privacidad en una transacción entre *Ethereum* y *Fabric* utilizando una solución de *Gateway*. Este análisis, a su vez, se puede dividir en tres etapas como se muestra en la figura 4.2. A continuación, se procederá a analizar en detalle las mismas y cómo se podrían mejorar de cara a soportar ambos requisitos.

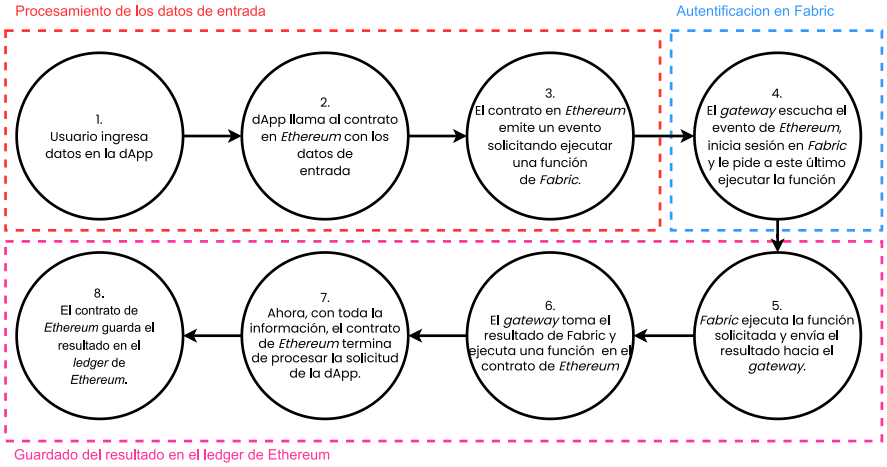


Figura 4.2: Agrupación de etapas en una transacción utilizando la solución de *Gateway*

4.5.1. Privacidad de la identidad

Dado lo anterior, se puede observar que es necesario conseguir un mecanismo que pueda autenticar al usuario en *Fabric* sin relacionarlo con el usuario de *Ethereum*. Para cumplir con el requisito de privacidad de la identidad, se decidió utilizar el concepto de un sistema de credenciales anónimas, como lo es *Idemix* descrito en la sección 2.2.3, ya que *Fabric* cuenta con una implementación de este. Para ello, se utiliza un esquema muy similar al visto en la sección 2.2.3, donde se identifica a la autoridad certificadora de *Fabric* como la organización *issuer* y a la *Fabric blockchain* como la organización verificadora.

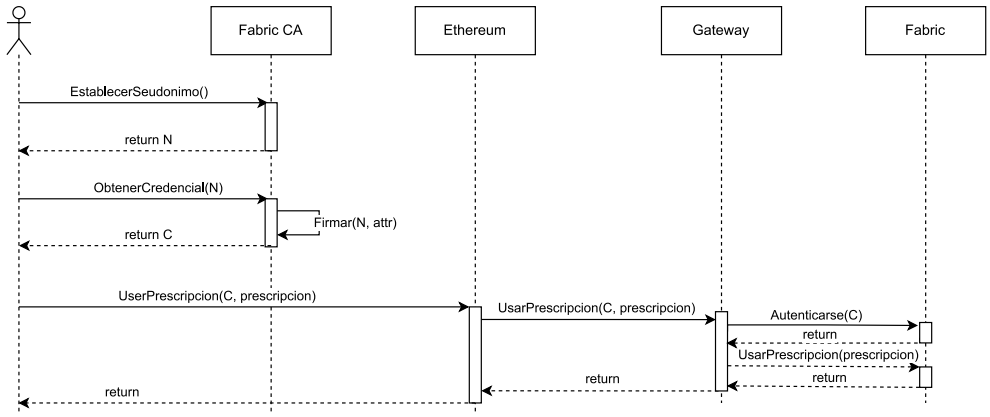


Figura 4.3: Aplicación de *Idemix*

Esta asociación deriva en el diagrama presentado en la figura 4.3. A es-

te diagrama, además, se le suman las entidades de *Ethereum* y el *Gateway* a través de los cuales el usuario debe enviar su credencial. Entonces, al usuario primero se le debe de otorgar una credencial, comunicándose con la *Fabric CA*, como se muestra en el diagrama con las operaciones, $EstablecerSeudonimo$ y $ObtenerCredencial$. Para luego enviarla mediante *Ethereum* y *Gateway*, en la operación $UsarPrescripcion$. Finalmente, autenticarse en *Fabric* para lograr, de esta forma, ejecutar la operación deseada sobre la *blockchain*, en este caso, $UsarPrescripcion$.

4.5.2. Privacidad del mensaje

A continuación, se exponen las distintas propuestas analizadas para solucionar el requisito de privacidad del mensaje. Las alternativas son: encriptación asimétrica, un protocolo desarrollado por el equipo que utiliza IPFS (ver anexo D), zkEVM 2.2.5 o *smart contract* Privados 2.2.4 y pruebas de conocimiento cero 2.2.1. Los abordajes descartados se encuentran detallados en el anexo H. Finalmente, para resolver el requisito se empleó la solución que utiliza pruebas de conocimiento cero.

Abordajes descartados

El primer abordaje consiste en encriptar los datos de entrada del *smart contract* en *Ethereum* y que el *Gateway* los desencripte con su clave privada, para luego enviar la *request* desencriptada a *Fabric*. De forma similar se encriptarían los elementos a ser guardados en el *ledger* de *Ethereum* y de esta forma se solucionaría el problema de privacidad del mensaje. Sin embargo, este abordaje fue descartado por ser vulnerable, ya que en el futuro, con el avance tecnológico, se podría romper la encriptación y los datos serían expuestos, vulnerándose el requisito de privacidad. El segundo abordaje se enfoca en el problema del almacenamiento de los elementos en el *ledger* de *Ethereum*. Como alternativa de encriptar los datos, se construyó un protocolo que permite guardarlos de forma segura y descentralizada. Este hace uso de IPFS, una tecnología de almacenamiento descentralizado. A su vez, también se construyó otro protocolo para recuperar los elementos previamente almacenados. Finalmente, este abordaje fue descartado por su alta complejidad de implementación, en particular, no se pudo resolver un problema asociado a la comunicación entre el *Gateway* y el usuario de forma descentralizada. El tercer y último abordaje descartado consiste en utilizar *Private Smart Contract* y *zkEVM*. Estas tecnologías revolucionarias prometen resolver los problemas de privacidad en su totalidad. Sin embargo, se encuentran en etapas muy tempranas de desarrollo. De todos modos, su estudio permitió mejorar el entendimiento del problema de privacidad en interoperabilidad y así llegar al cuarto abordaje: las pruebas de conocimiento cero.

Abordaje utilizado: Zero Knowledge Proofs con Zokrates

Investigando diferentes formas de construir pruebas de conocimiento cero, se encontró la tecnología Zokrates 2.2.6. Esta tecnología autodescrita como *toolkit* permite generar todo tipo de pruebas ZKP. En particular se construyó un programa con *Zokrates* que genere ZKP. Estas a su vez permitirían probar que un individuo posee la preimagen de un *hash* sin revelar nada acerca de la preimagen. En la figura 4.4 se puede observar como un Probador utilizaría este tipo de pruebas para demostrarle a un Verificador, que posee cierta información, de forma que el Verificador solo conozca la ZKP y el *hash* de la información.

Este tipo de pruebas permitirían resolver todos los problemas que puedan extrapolarse a probar que se posee la preimagen de un *hash*. En el escenario propuesto, la información en texto plano se extrapolaría a la información de la prescripción del medicamento en *Fabric*. Además, el *hash* de la información sería lo que presenta el paciente en la farmacia, es decir, el *hash* de la prescripción. Por lo tanto, el problema se reduce a que el *smart contract* en *Ethereum* consulte a *Fabric* si este posee la preimagen del *hash*. Para contestar esta pregunta, *Fabric* va a generar una ZKP que permita verificar si este efectivamente posee la preimagen.

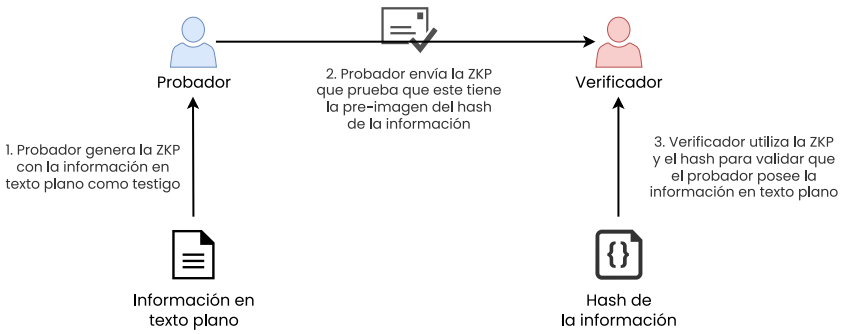


Figura 4.4: Interacción entre el Probador y el Verificador con la ZKP construida con Zokrates

El abordaje con estas ZKP generadas resuelve el requisito de la privacidad del mensaje, ya que la ZKP generada se puede almacenar sin problemas en el *ledger* de *Ethereum*. Esto se debe a que a partir de la ZKP no se puede inferir nada de la información en *Fabric*, por lo tanto, se va a mantener la privacidad del mensaje. En la figura 4.5 se puede observar como serían las interacciones con el abordaje planteado instanciado a la solución de *Gateway* propuesta.

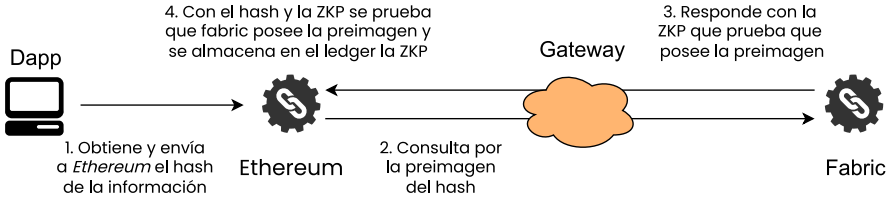


Figura 4.5: Interacción entre los componentes de la solución propuesta en una operación de *Data Sharing* que utiliza ZKP

4.6. Conclusiones

Las ZKP resultaron ser vitales para evolucionar la solución de *Gateway* y solucionar sus problemas de privacidad. La privacidad de la identidad fue subsanada utilizando credenciales anónimas con *Idemix*, las cuales utilizan a su vez ZKP para su implementación. Para la privacidad del mensaje también se utilizaron ZKP, pero en este caso fueron construidas específicamente para una clase de problemas que incluyen al escenario propuesto. En la construcción de la ZKP fue clave la herramienta *Zokrates*, permitiendo abstraer la complejidad de construcción de la ZKP en un lenguaje de programación propio. En conclusión, las ZKP fueron necesarias a lo largo de la solución para agregar características de privacidad a la misma y cumplir con los requisitos de privacidad. Esto se alinea con la dirección que está tomando la industria, ya que esta tiene presente la importancia de este tipo de pruebas. Por esa razón, todos los esfuerzos están puestos en su desarrollo y mejora.

Capítulo 5

Diseño de la solución

Este capítulo inicia con una descripción conceptual de la solución en la sección 5.1. Luego se detalla la arquitectura de la misma mediante las vistas 4 + 1 en la sección 5.2. Para finalizar, en la sección 5.3 se da una descripción de las principales decisiones de diseño.

5.1. Descripción conceptual de la solución

La solución propuesta está compuesta por la *blockhain* de *Ethereum*, la *blockchain* de *Fabric*, el *Gateway* y dos dApps, una de la mutualista para interactuar con *Fabric* y otra de la farmacia para interactuar con *Ethereum*. La solución y sus componentes pueden observarse en la figura 5.1. En la *blockchain* de *Ethereum* se encuentran dos *smart contract*, el *Pharmacy Smart Contract* que se comunica con la dApp de la farmacia y con el *Gateway*. Además, también está el *Verifier Smart Contract*, el cual es responsable de validar las ZKP al ser invocado por el *Pharmacy Smart Contract*. El *Gateway* está compuesto por el *Ethereum Connector*, el *Router* y el *Fabric Connector*. Todos estos componentes fueron heredados del proyecto (Mathías Castro, 2023). También es parte del *Gateway* el *Fabric-Idemix Connector* que fue desarrollado para que se comunique con la *Fabric CA*. Esta última es una componente de *Fabric*. Por otro lado, en la *blockchain* de *Fabric* se encuentra el *Prescriptions Smart Contract* que interactúa con la dApp de la mutualista y con el *Zokrates adapter*. Este último es responsable de generar las ZKP.

La solución consta de tres etapas, donde cada una concuerda con un caso de uso. La primera etapa comienza cuando el médico carga en el sistema una prescripción para el paciente utilizando la dApp de la mutualista. Esta primera etapa se corresponde con el caso de uso 3.4.2. La segunda etapa consiste en que el paciente obtenga los datos necesarios para usar la prescripción en la farmacia. Para esto, a través de la dApp de la mutualista obtendrá los datos listados en el caso de uso 3.4.2. En la tercera y última etapa el paciente concurre a una terminal de autoservicio de la farmacia. Luego, ingresa los datos obtenidos en

la etapa anterior en la dApp de la farmacia y si los datos son válidos, canjea la prescripción. Esta tercera etapa se corresponde con el caso de uso 3.4.2.

Las tres etapas descritas anteriormente requieren de una etapa previa de configuración o etapa de *setup*. En esta etapa se coordinan la mutualista con la farmacia para generar la llave de prueba (*proof key*) y la llave de verificación (*Verify key*). Este encuentro de coordinación podría ser auditado, por ejemplo, por una agencia nacional que regularice el uso de la solución general. Esta etapa de *setup* es necesaria para que ambas entidades consigan sus llaves tanto para generar pruebas ZKP (la mutualista) como para verificar que las mismas son válidas (la farmacia).

A continuación, se detallan las interacciones entre los componentes al ejecutarse las distintas etapas.

En la primera etapa el médico carga una nueva prescripción en el sistema. Para esto usa la dApp de *Fabric* e ingresa la prescripción con los detalles del medicamento y los datos del paciente. La dApp entonces genera una prescripción con los datos ingresados y la almacena en el *smart contract* de prescripciones (*Prescriptions Smart Contract*). Además, la prescripción se asocia a un *hash* que identifica al usuario en el sistema, con el fin de autorizar más adelante a una credencial anónima que quiera acceder a la prescripción almacenada.

Una vez completada la primera etapa, el paciente ingresa con su usuario en la dApp de la mutualista y solicita los datos necesarios para poder hacer uso de la prescripción en la farmacia. La dApp accede al *Prescriptions Smart Contract* y obtiene los datos de la prescripción ingresados por el médico para el paciente. Además, el *smart contract* retorna el *hash* almacenado junto a la prescripción, para autorizar al usuario en el sistema de la mutualista. Por último, la dApp se comunica con la *Fabric CA*, para generar una credencial anónima que permita al sistema de la farmacia verificar la prescripción sin exponer la identidad del paciente. Finalmente, la dApp le muestra al paciente todos los datos obtenidos anteriormente.

Teniendo todos los datos de la prescripción y su credencial anónima, el paciente puede concurrir a la farmacia y dar inicio a la etapa tres. Para esto ingresa los datos obtenidos en la dApp de la farmacia (*Pharmacy dApp*) con el fin de utilizar la prescripción. La *Pharmacy dApp* $\ll$*hashea*$\gg$ la prescripción y llama al *smart contract* de la farmacia (*Pharmacy Smart Contract*). Esta llamada tiene como argumentos los datos ingresados por el paciente (credencial anónima y *hash* del usuario para autorización), más el *hash* de la prescripción generado en la dApp. El *Pharmacy Smart Contract* emite un evento en el cual envía: todos los datos proporcionados por la dApp, el nombre de la función de *Fabric* que se desea ejecutar, el *smart contract* de *Fabric* donde se encuentra la función y una función *callback* que reciba la respuesta de *Fabric* como parámetro. El evento emitido es escuchado por el *Ethereum connector* en el *Gateway*, este decodifica el mensaje y lo envía al *Router*. Este último toma el mensaje y verifica su destino. Dado que el destino es la *blockchain* de *Fabric* con credenciales anónimas, el *Router* redirige el mensaje hacia el *Fabric-Idemix Connector*. Este hace una solicitud a la *Fabric CA* y válida la credencial anónima del paciente autenticándose sin exponer la identidad del usuario en *Fabric*. Una vez autenti-

cado, el conector se comunica con el *Prescriptions Smart Contract* en *Fabric* y ejecuta la función para validar la prescripción del usuario. Para esto, el *smart contract* busca la prescripción asociada al *hash* de autorización del paciente, y si la encuentra, verifica que el *hash* de la misma sea igual al *hash* de la prescripción recibido en el mensaje desde *Ethereum*. En caso de ser igual, el *smart contract* utiliza el *Zokrates Adapter* para generar una ZKP que valide la prescripción. Una vez que la ZKP se encuentre generada, el *Prescriptions Smart Contract* emite un evento solicitando ejecutar la función de *callback* obtenida desde *Etehereum* con la ZKP como argumento. El *Fabric Connector* escucha el evento, traduce el mensaje, lo envía a través del *Router* hacia el *Ethereum connector* y este último invoca la función de *callback* en el *Pharmacy Smart Contract*. El *Pharmacy Smart Contract* invoca al *Verifier Smart Contract* con la ZKP recibida para validarla. Una vez validada, el *Pharmacy Smart Contract* emite un evento informando el resultado de la operación, este es escuchado por la *Pharmacy dApp* y le informa al paciente si puede o no utilizar la prescripción.

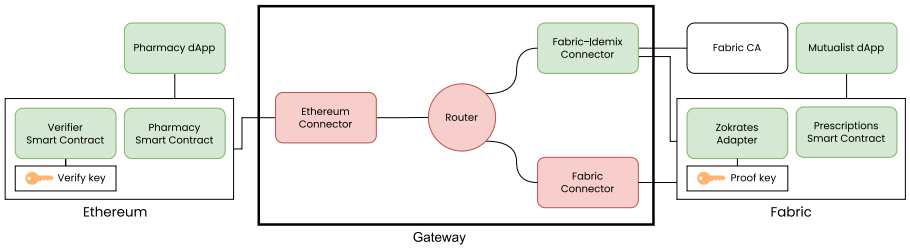


Figura 5.1: Componentes de la solución. En verde los desarrollados durante el transcurso del proyecto y en rojo los heredados de la solución del *Gateway*. La *Fabric* CA es provista por *Fabric*.

5.2. Arquitectura de la solución

A continuación, se detalla la arquitectura del sistema mediante la representación de las vistas 4 + 1 sin tomar en cuenta la vista de casos de uso debido a que ya fue descrita en la sección de análisis de requerimientos 3.

5.2.1. Vista de implementación

En la figura 5.2 se muestran los principales componentes de la aplicación, donde se presenta de lado izquierdo los componentes de acceso público relacionados con *Ethereum*.

DApp Farmacia

Es una aplicación web la cual consume una interfaz de *Ethereum* que le permite la comunicación bilateral con el *Pharmacy Contract* para hacer uso de una prescripción.

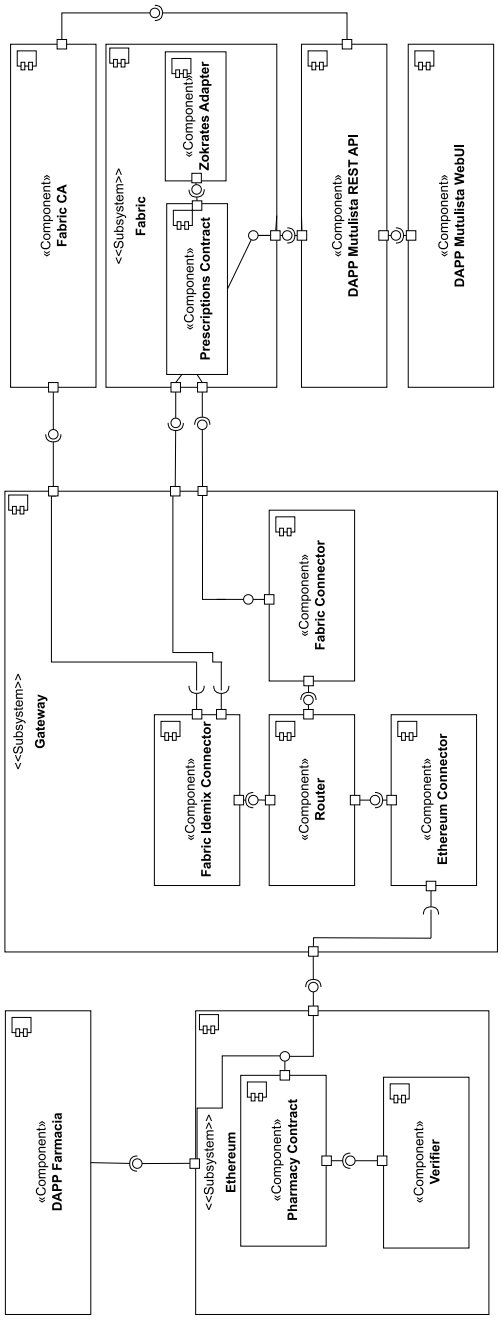


Figura 5.2: Diagrama de componentes

Pharmacy Contract y Verifier

El *Pharmacy Contract*, un *smart contract* en *Ethereum*, tiene como objetivo solicitar la utilización de una prescripción y recibir como respuesta si la prescripción es válida. Para ello, debe consumir una interfaz provista por otro *smart contract* llamado *Verifier* con el cual verifica la validez de una prescripción. Además, se comunica a través de una interfaz expuesta por *Ethereum* con el *Ethereum Connector* a través del cual solicita la prescripción, y luego recibe una prueba como respuesta.

Ethereum Connector

El *Ethereum Connector* envía la solicitud de utilizar la prescripción a través del *Router* hacia el *Fabric Idemix Connector*. Además, utiliza la interfaz que consume de *Ethereum* para realizar una llamada a la función de verificación del *Pharmacy Contract*.

Router

Dentro del subsistema *Gateway*, todos los conectores consumen la interfaz del *Router*, el cual es encargado de redirigir los mensajes al *connector* correspondiente.

Fabric Idemix Connector

El *Fabric Idemix Connector* es encargado de solicitar, en potestad del usuario, la prescripción al *Prescriptions Contract*, un *smart contract* en *Fabric*. Para llevar a cabo dicha labor debe consumir la interfaz provista por *Fabric* y *Fabric CA*, para la validación de las credenciales anónimas.

Fabric Connector

El *Fabric Connector* se encarga de escuchar cambios en *Fabric* a través de la interfaz a la que está suscrito y enviarlos a través del *Router* hacia *Ethereum*.

Fabric CA

La *Fabric CA* es un componente utilizado para realizar la autenticación sin revelar la identidad del usuario, expone una interfaz consumida por el *Fabric Idemix Connector* y la *dApp Mutualista*.

Prescriptions Contract

El *Prescriptions Contract* es el encargado de crear, actualizar y borrar prescripciones. Este expone una interfaz para que los demás componentes puedan acceder a las prescripciones. Además, consume la interfaz provista por el *Zokrates Adapter* con el objetivo de crear una prueba ZKP que luego retorna al utilizar una prescripción.

Zokrates Adapter

El *Zokrates Adapter* es un componente *off-chain* que opera dentro de la misma red que *Fabric*. Se encarga de crear una ZKP correspondiente a la prescripción solicitada.

dApp Mutualista WebUI y REST API

La *dApp Mutualista WebUI* es una interfaz gráfica web que consume la interfaz de la *dApp Mutualista REST API*, de la cual hace uso para crear usuarios tanto médicos como pacientes, crear prescripciones, entre otros. Para poder realizar estas tareas, la *dApp Mutualista REST API* debe comunicarse con el *Precriptions Contract* a través de *Fabric*, ya que las prescripciones se encuentran almacenadas en ese lugar.

5.2.2. Vista lógica

A continuación, se presenta el flujo que se realiza para llevar a cabo los casos de uso definidos en la sección 3.4.2 por medio de diagramas de secuencia.

Ingresar prescripción

El flujo de la figura 5.3 comienza con el médico agregando el identificador del paciente y el nombre del medicamento en el componente *dApp Mutualista WebUI*. Una vez que seleccione la opción de ingresar se dispara una petición *createPrescription(userId,description_prescripcion)* al componente *dApp Mutualista REST API* el cual se encarga de ejecutar el *smart contract* de *Fabric* en el componente *Prescriptions Contract*. La ejecución del *smart contract* crea una prescripción en el sistema para el paciente y retorna su identificador *prescriptionId*.

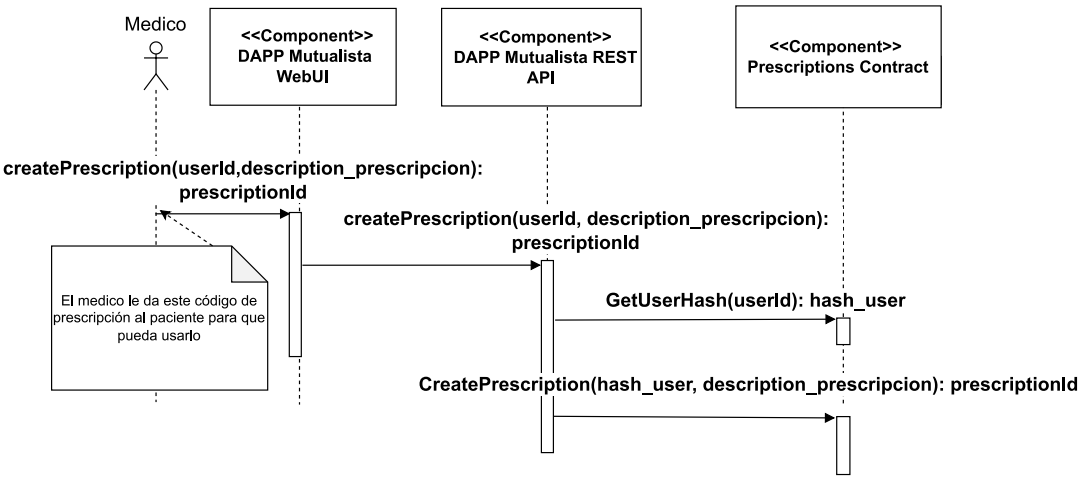


Figura 5.3: Diagrama de secuencia para Ingresar prescripción

Obtener credenciales

El flujo de la figura 5.4 comienza con el paciente ingresando el identificador de su prescripción en el componente *dApp Mutualista WebUI* para así obtener los elementos necesarios que le permitirá usar su prescripción en una farmacia. Al solicitar estos elementos, se realiza una petición al compo nente *dApp Mutualista REST API* mediante *getPrescriptionAndUserAndIdemixCredentials(prescriptionId)*. Este se encarga de obtener los diferentes datos a retornar comunicándose con el componente *Prescriptions Contract* y el componente *Fabric CA*. Primero se ejecuta *GetPrescription(prescriptionId)* en el componente *Prescripcions Contract* para poder obtener la prescripción. Una vez finalizada la ejecución del *smart contract* se obtienen las credenciales que necesita el usuario para usar en la farmacia. Para esto, se llama al componente *Fabric CA* mediante *getIdemixEnrollmentJson(enrollment, mspId)*, siendo el *enrollment* obtenido a partir del identificador del usuario en *Fabric* y el *mspId* el identificador de la *Fabric CA*.

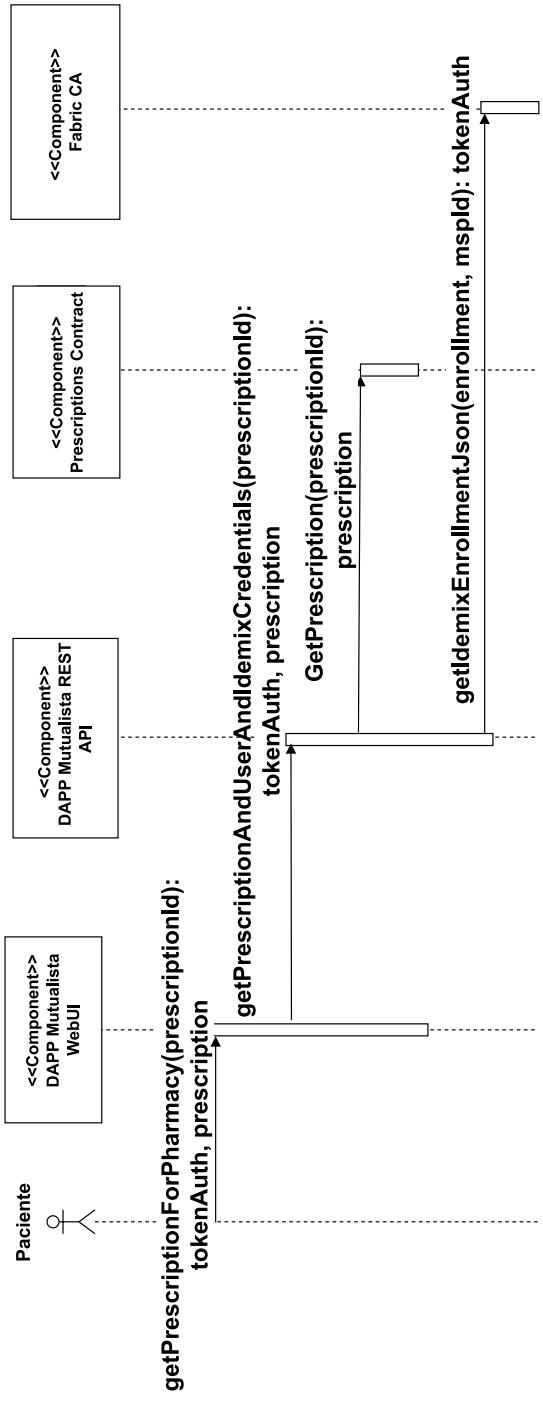


Figura 5.4: Diagrama de secuencia para Obtener credenciales

Usar prescripción

El flujo de la figura 5.5 comienza con el paciente agregando sus credenciales anónimas de *Idemix* representadas como un tokenAuth, su *hash* de usuario y su prescripción en el componente *dApp Farmacia*. Una vez que seleccione la opción de ingresar se dispara la petición *confirmAndBurnPrescription(tokenAuth, hash_user, hash_prescripcion)*. La invocación de UsarPrescripcion de la *dApp Farmacia* ejecuta el *smart contract* de *Ethereum* representado por el componente *Pharmacy Contract* y este a su vez se conecta con el *Gateway* mediante *ExistAndBurnPrescription(hash_user, hash_prescription, tokenAuth, OperationResponseEth)* para intentar usar la prescripción. Además, se envía el parámetro *OperationResponseEth*, este es una función de *callback* que será invocada junto a la respuesta generada en *Fabric*. El *Gateway* ejecuta el *smart contract* representado por el componente *Prescriptions Contract* utilizando la operación *BurnPrescription(tokenAuth, hash_user, hash_prescription, OperationResponseFabric)*. El *Prescriptions Contract* válida si se puede usar o no la prescripción y genera la prueba correspondiente mediante el componente *Zokrates Adapter*. Una vez ejecutada la validación de la prescripción, se devuelve la respuesta al *Gateway* utilizando la función de *callback OperationResponseFabric(proof)*. Si bien no quedo implementado y se menciona más adelante como una mejora a futuro, el *Gateway* se encargará de revocar las credenciales anónimas para que no puedan ser usadas nuevamente. Para esto se comunicaría con el componente *Fabric CA* llamando a la función *revokeCredentials(tokenAuth)*. Al obtener la prueba enviada por *Fabric*, el *Gateway* se la retorna al *Pharmacy Contract* mediante el *callback OperationResponseEth(proof)*. Una vez que el *Pharmacy Contract* obtiene la prueba, la verifica en el *smart contract Verifier* para obtener el resultado de la operación y finalmente emitir un evento al cliente con el resultado final.

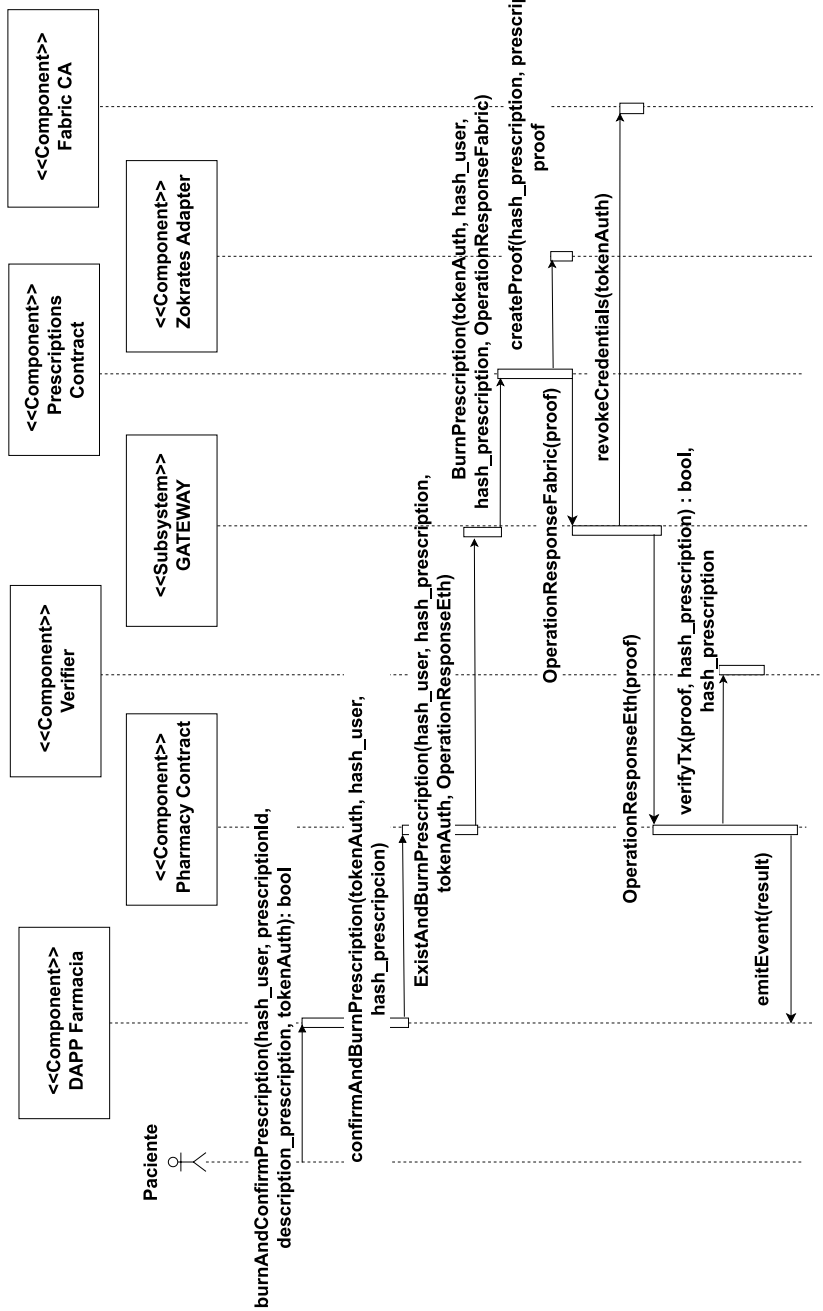


Figura 5.5: Diagrama de secuencia para usar una prescripción

5.2.3. Vista de distribución

En la figura 5.6, se observa el despliegue en un *host Docker* de la solución, y algunos componentes externos. Los componentes externos constan de dos *smart contracts*, *PharmacyContract* y *Verifier* que deben ser desplegados en la red de *Ethereum*. La *dApp Farmacia*, por otro lado, es una aplicación web estática que puede ser servida desde cualquier CDN o similar. Dentro del *host* que ejecuta *Docker* se crean tres redes separadas. Esto es con el objetivo de simular un ambiente de producción en donde cada sección estaría alojada por una entidad distinta. El *Gateway* estaría alojado por una entidad, mientras que la Red *Fabric* de las mutualistas es alojada por otra. Además, cada una de las mutualistas tiene su propia infraestructura para su aplicación. Dentro del *Gateway* se encuentran cinco contenedores separados que alojan las distintas aplicaciones. Una de ellas es el *Ethereum Connector* que además de estar en la red interna del *Gateway* debe estar expuesto a internet para comunicarse a través de *web sockets* con la red *Ethereum*. Luego se encuentra el *Router*, este no necesita exposición a internet, ya que solo se comunica con los tres *connectors* en su red local. Mientras tanto, *Fabric Idemix Connector* y *Fabric Connector* son las últimas dos aplicaciones que corren en contenedores dentro de la red del *Gateway*. Estas también deben estar expuestas a internet para comunicarse con la red *Fabric* de las mutualistas.

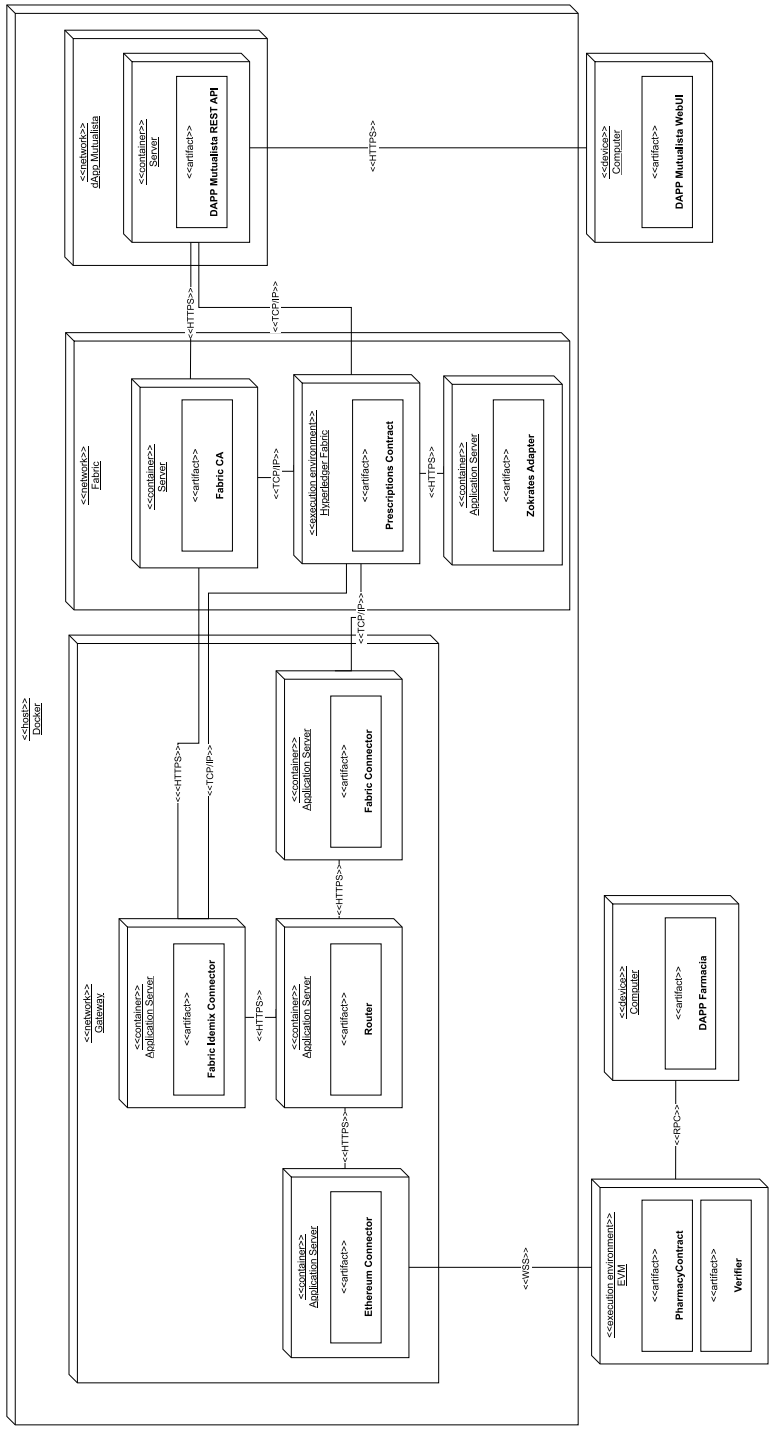


Figura 5.6: Diagrama de despliegue

La red *Fabric* contiene tres componentes. Un contenedor aloja la *Fabric CA*, que es una REST API que puede ser consumida por entidades externas, por lo que además de estar en la red *Fabric* está expuesta a internet. Además, se tiene la propia *blockchain* de *Fabric* en la que se debe desplegar un *smart contract*, el *Prescriptions Contract*, *Fabric* también se encuentra expuesto a internet, ya que debe ser consultado por servicios externos. El contenedor que tiene desplegado al *Zokrates Adapter*, una aplicación que expone una REST API consumida por el *Prescriptions Contract*, por lo cual solo se encuentra expuesta a la red interna y no al internet. Finalmente, en la red de la mutualista se aloja un contenedor que tiene desplegado el componente REST de la dApp de la mutualista llamado *dApp Mutualista REST API*, y un contenedor que tiene desplegado la interfaz de usuario de la dApp de la Mutualista llamada *dApp Mutualista WebUI*. Ambos expuestos a internet, ya que son usados por los usuarios de la mutualista.

5.3. Principales decisiones de diseño

En esta sección se describen algunas de las consideraciones que se tomaron al diseñar el sistema. A su vez, se detallan las razones del abordaje tomado para resolver algunos inconvenientes que surgieron a lo largo del diseño de la prueba de concepto.

5.3.1. Revocación de credenciales

Como al realizar cualquier comunicación con el *Gateway* el mensaje debía de ser publicado en *Ethereum*, las credenciales quedarían expuestas y podrían ser extraídas de la *blockchain* y reutilizadas. Por lo que se tomó la decisión de agregar un mecanismo de revocación de las mismas una vez hecha la consulta en *Fabric*. Pero esto no puede ser implementado debido a una limitación tecnológica de *Idemix* en *Fabric* que actualmente se encuentra en una fase temprana de implementación.

5.3.2. Mecanismo de autorización

Se utilizó el *hash* del usuario para autorizar las peticiones. Un mecanismo alternativo sería el uso de atributos personalizados, pero aún no es posible generar una credencial anónima con este tipo de atributos.

5.3.3. Doble hashing

Dado que la preimagen puede tener un tamaño considerable e impactar en el desempeño del cálculo de la prueba ZKP, se decidió realizar un doble *hashing* de la prescripción. Con el objetivo de acelerar el desempeño, se planteó la posibilidad de realizar la prueba a partir del *hash* del texto inicial, en lugar del texto plano inicial, desembocando en un *double-hashing*. Observar que, aunque la prueba es distinta, se sigue obteniendo el mismo resultado, ya que ahora

se prueba que el usuario posee un *hash* del texto inicial que corresponde a la preimagen del *hash* que posee la farmacia. Esto se mantiene siempre que el algoritmo de *hash* seleccionado tenga una probabilidad de colisiones suficientemente baja para ser despreciable. Visto en el escenario planteado, ahora la farmacia posee un *hash* del *hash* de la prescripción, es decir, la prescripción se ≪*hashea*≫ dos veces, y la mutualista debe generar una ZKP con el *hash* de la prescripción, ya que ahora este es la preimagen buscada.

5.3.4. Creación de Zokrates Adapter

Referente al cálculo de la prueba, debido a una limitación tecnológica que prohibía el uso de la librería para la generación de pruebas, se optó por hacer ese cálculo *off-chain*. Aunque en un futuro cercano se espera que haya soporte para realizar este paso dentro de la ejecución del *smart contract* y así remover este componente del sistema.

Capítulo 6

Desarrollo

En este capítulo se analizan los detalles más relevantes de implementación de la solución. En la sección 6.1 se enumeran las tecnologías utilizadas, se menciona como interactúan los componentes y se muestra un diagrama que ilustra las tecnologías y sus interacciones. En la sección 6.2 se describen los detalles más relevantes de implementación del escenario de estudio. La sección 6.3 describe como se configuró, integró y utilizó *Zokrates* en la solución. En la sección 6.4 se listan las decisiones de implementación más relevantes tomadas durante el desarrollo de la solución. La sección 6.5 describe las limitaciones tecnológicas encontradas durante la implementación y como fueron solventadas. Por último, la sección 6.6 describe cuáles fueron los desafíos de implementación y cómo se resolvieron.

6.1. Tecnologías

En la implementación del caso de estudio planteado, fue necesaria la utilización de varias tecnologías. En la figura 6.1 se puede ver un diagrama de la solución, donde se muestra para cada componente la tecnología en la que fue desarrollado.

La dApp de la farmacia fue construida utilizando HTML, CSS y JS para el *frontend*. En el *backend* se utilizó *JavaScript* junto con la librería *Web3* para interactuar con la *blockchain* de *Ethereum*. En *Ethereum* se encuentran dos *smart contract* desplegados, el *PharmacySmartContract* escrito en *Solidity*[1], y el *smart contract Verifier* que fue generado utilizando *Zokrates* 2.2.6.

En lo que respecta a *Fabric*, se utilizó *Spring Framework*[2] para el desarrollo del *backend* de la dApp de la mutualista y *Javascript Svelte*[3] para el desarrollo de su *frontend*. El contrato desplegado en *Fabric* (*PrescriptionsContract*) fue

[1] https://soliditylang.org/
[2] https://spring.io/
[3] https://svelte.dev/

desarrollado en *GO*[4]. El *Zokrates-Adapter* fue construido utilizando *Node.js*[5] junto al *framework Express*[6] y el SDK de *Zokrates*. La *Fabric CA* es una REST API en *GO*, la misma fue utilizada sin realizarle modificación alguna.

Por último, y en lo que refiere al *Gateway*, se mantuvo la tecnología con la cual fue desarrollado anteriormente en el proyecto de grado (Mathías Castro, 2023), es decir, *Node.js*. A su vez, se extendió agregándole el componente *Fabric-IdemixConnector* el cual fue desarrollado utilizando *Java*[7].

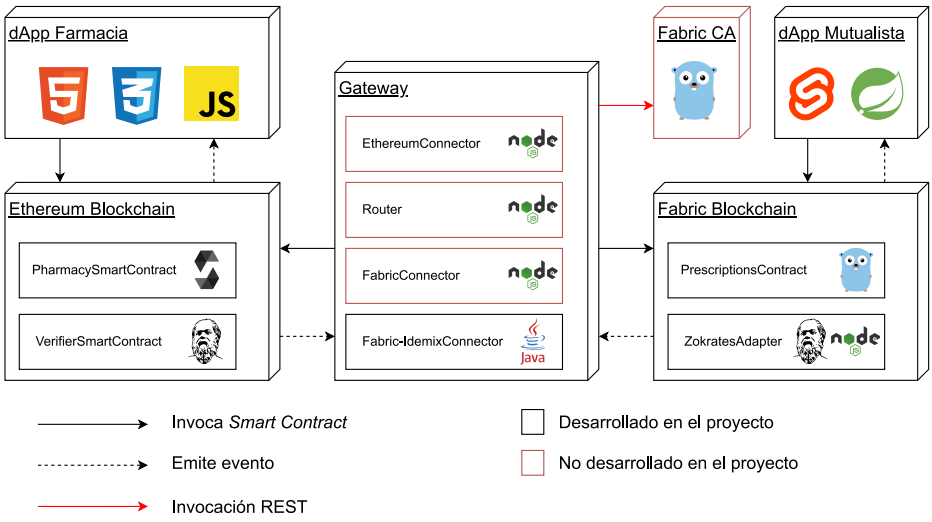


Figura 6.1: Tecnologías utilizadas en la solución

6.2. Implementación del escenario

A continuación se describen los componentes implementados.

6.2.1. Componentes Fabric

En *Fabric* existen dos componentes importantes, el *smart contract Prescriptions* y la *dApp Mutualista*, compuesta de una REST API y una interfaz gráfica web. Por su parte, el *smart contract Prescriptions* se encarga de interactuar con el *ledger* de *Fabric*, creando, listando y removiendo prescripciones. En la figura 6.2 se muestran las firmas de las funciones del *PrescriptionsContract*. También se define un tipo prescripción (Prescription) con un ID, una descripción y un dueño (*owner*) que se puede ver en la figura 6.3.

[4]https://go.dev/
[5]https://nodejs.org/en
[6]https://expressjs.com/
[7]https://www.java.com/en/

```go
type SmartContract struct {
  contractapi.Contract
}
// Inicializa el ledger
func (s *SmartContract) InitLedger() {}
// Crea una nueva prescripcion y la añade al estado del ledger
func (s *SmartContract) CreatePrescription(
    ctx contractapi.TransactionContextInterface,
    id string, description string, owner string
) (string, error) {}
// Obtiene una prescripcion para un usuario
func (s *SmartContract) GetPrescription(
    ctx contractapi.TransactionContextInterface, prescriptionID string
) (string, error) {}
// Obtiene el hash del usuario
func (s *SmartContract) GetUserHash(
    ctx contractapi.TransactionContextInterface
) (string, error) {}

// Usa una prescripcion y genera una ZKP si todas las condiciones se cumplen
// Las condiciones son: La prescripcion  EXISTE y PERTENECE AL USUARIO
func (s *SmartContract) BurnPrescription(
    ctx contractapi.TransactionContextInterface, args string
) (string, error) {}

// Obtiene todas las prescripciones para un usuario
func (s *SmartContract) GetPrescriptions(
    ctx contractapi.TransactionContextInterface
) (string, error) {}

// Crea un Remote Event y llama a sendRemoteEvent con los argumentos correctos
// Esta funcion es utilizada por BurnPrescription
func notify(
    ctx contractapi.TransactionContextInterface,
    info map[string]interface{},
    bcTarget string,
    scTarget string,
    opTarget string,
    responseMessageId string,
    replyTo string,
) {}
```

Figura 6.2: Métodos en el PrescriptionsContract

```
type Prescription struct {
    ID              string `json:"id"`
    Description     string `json:"description"`
    Owner           string `json:"owner"`
}
```

Figura 6.3: Estructurado de la prescripción

Para interactuar con el *smart contract* existe la dApp de la Mutualista. La dApp consta de una REST API que se encarga de manejar el acceso y creación de usuarios y prescripciones en *Fabric*. Esta cuenta con dos *controllers* uno encargado del CRUD de usuarios y el otro encargado de las prescripciones. Este último cuenta con funcionalidades como listar las prescripciones del usuario y obtener los datos de la prescripción de la farmacia. Además, también se encarga de la obtención de credenciales anónimas. Esto se realiza a partir del ID del usuario (un nombre ingresado por el usuario al momento de registrarse en la dApp), e invocando el método correspondiente. Luego, se crea un objeto de tipo *Enrollment* con la credencial *Idemix* como se muestra en la figura 6.4.

```
public String getCredentialsIdemix(String userId) throws Exception {
    X509Identity identity = (X509Identity) this.wallet.get(userId);
    NetworkConfig.OrgInfo info = this.config.getClientOrganization();
    AppUser user = new AppUser(userId, info.getName(), info.getMspId(), identity);
    return caClient.getIdemixEnrollmentJson(
        user.getEnrollment(),
        env.getProperty("msp.idemix")
    ).toString();
}
```

Figura 6.4: Obtener credenciales anónimas en dApp Mutualista

6.2.2. Fabric Connector Idemix

El *Fabric Connector Idemix* es una REST API con un único *endpoint* encargado de ejecutar en *Fabric* una función del *smart contract* utilizando credenciales *Idemix*. Como se observa en la figura 6.5 se encarga de transformar un JSON recibido en una credencial utilizable por la API de Fabric. Luego de ser utilizada, la credencial debería ser revocada, pero en la actualidad la revocación de credenciales *Idemix* no se encuentra aún implementada en la *Fabric* CA, por lo cual la implementación de este paso no se llevó a cabo.

```java
@RestController
@RequestMapping("/")
public class FabricConnectorController {
    @Autowired
    private FabricClient fabricClient;

    @PostMapping
    public ResponseEntity<Object> callBlockchain(@RequestBody Map<String, Object> body) {
        Map<String, Object> header = (Map<String, Object>) body.get("header");
        Map<String, Object> data = (Map<String, Object>) body.get("data");
        Map<String, Object> target = (Map<String, Object>) header.get("target");
        Map<String,Object> enrollment = gson2.fromJson((String) data.get("idemix"), Map.class);
        JsonObjectBuilder builder = Json.createObjectBuilder();

        //Obtener propiedades del request body ...

       CompletableFuture.runAsync(() -> {
            try {
                String event = gson.toJson(body);
                String response = (String) this.fabricClient.callBlockchainIdemix(
                    builder.build(), (String) target.get("operation"),
                    (String) target.get("contract"), event
                );
            } catch (Exception e) {
                logger.error("An error occured: " + e.getMessage());
            }
        });

        return new ResponseEntity<>(HttpStatus.OK);
    }
}

@Service
public class FabricClient {
    // Inicializacion del servicio...

    public Object callBlockchainIdemix(JsonObject idemixEnrollment, String functionName,
    String chaincodeName, String event) throws Exception {
        AppUser appUser = new AppUser(
                "anon", idemixEnrollment.getString("ou"),
                idemixEnrollment.getString("mspID"),
                caClient.getIdemixEnrollmentFromJson(idemixEnrollment)
        );
        return invokeBlockChain(appUser, functionName, chaincodeName, event);
    }
}
```

Figura 6.5: Función para utilizar credenciales anónimas en *Fabric Connector Idemix*

6.2.3. DApp Farmacia

Del lado de la farmacia, las credenciales son utilizadas a través de la dApp web. El usuario ingresa su credencial anónima, su *hash* de autenticación, el ID de la prescripción y su descripción. La dApp de la farmacia procesa los datos y emplea la técnica de *doble hashing*, explicada en la sección 6.4.1, sobre los datos ingresados por el usuario. Luego, la dApp invoca la función *confirmAndBurn-*

Prescription del *PharmacyContract* en *Ethereum*. Finalmente, la dApp espera por un evento de respuesta desde *Ethereum* que le indique si la operación fue exitosa o no. Los detalles de implementación de la dApp de la farmacia se encuentran en el anexo I.

6.2.4. Ethereum Smart Contract

Las funciones más importantes de los *smart contracts* en *Ethereum* y que se analizarán en esta sección son: *confirmAndBurnPrescription*, *resultAndBurnPrescription* y *verifyTx*. Para más detalles de implementación ver anexo J.

La función *confirmAndBurnPrescription* es la que va a ser invocada por la dApp de la farmacia y es responsable de emitir el evento *ExistAndBurnPrescription*. Este evento es el que escucha el *Gateway* y contiene toda la información necesaria para que la mutualista pueda validar o rechazar la prescripción. La función y el evento mencionados pueden observarse en la figura 6.6.

```
function confirmAndBurnPrescription(string memory _certificate,
    string memory _user, string memory _prescription) public
{
    emit ExistAndBurnPrescription(
        _user,
        _prescription,
        "fabric-idemix",
        "prescriptions",
        "burnPrescription",
        "resultBurnPrescription",
        _certificate
    );
}
```

Figura 6.6: Función *confirmAndBurnPrescription* y evento *ExistAndBurnPrescription* del *PharmacyContract*

Por otro lado, la función *resultAndBurnPrescription* es responsable por validar si una ZKP 2.2.1 es correcta o no. Recordar que *Fabric* enviará, además de la ZKP y del doble *hash* de la prescripción, un *booleano* que indica si la operación fue exitosa. En caso afirmativo, el contrato validará la prueba ZKP utilizando la función *verifyTx* del contrato *Verifier*. Para finalizar, el contrato emite el evento *PrecriptionResult* que indica el resultado de la verificación. La función y el evento se pueden observar en la figura 6.7.

6.3. Uso de Zokrates

Esta sección describe como se utilizó la *toolbox* de *Zokrates* en la implementación. En líneas generales se utilizó la herramienta para generar dos llaves a través de su CLI. La primera llave es la de verificación, esta se utilizó para generar un *smart contract* en *Ethereum* que permite verificar las pruebas de conocimiento cero. De esta forma, el contrato de la farmacia va a ser capaz de

```
function resultBurnPrescription(bool _result,
Verifier.Proof memory _proof, uint[8] memory _hashPrescription) public
{
    if (!_result)
   {
        emit PrescriptionResult(false, _hashPrescription);
        return;
   }

    emit PrescriptionResult(
        verifier.verifyTx(_proof, _hashPrescription),
        _hashPrescription
    );
}
```

Figura 6.7: Función *confirmAndBurnPrescription* y evento *ExistAndBurnPrescription* del *PharmacyContract*

verificar si las ZKP generadas en la mutualista son correctas utilizando el contrato de verificación. La segunda llave es la de prueba y junto al programa que define las ZKP a generar, se pueden construir pruebas de conocimiento cero. Tanto la llave de prueba como el programa están integrados en la infraestructura de *Fabric*. De esta forma, la mutualista puede generar ZKP que prueben la existencia y pertenencia de las prescripciones a un paciente.

Habiendo descrito el uso general de la herramienta, a continuación, se profundizará en su implementación en la solución. Para esto es necesario construir el programa que defina las ZKP, las genere con la llave de prueba y las verifique con la llave de verificación.

Definición de las ZKP

Como se explicó anteriormente en la sección 2.2.6, el primer paso para utilizar *Zokrates* es definir la prueba de conocimiento cero utilizando su lenguaje de programación. Esta definición es un programa denominado *main.zok*, el cual puede observarse en la figura 6.8. El programa comienza importando la función de *hash Sha256* que provee *Zokrates*. Esta recibe un arreglo Uint de 32 bits de tamaño 8. La función concatena a este arreglo otro arreglo *dummy*, también de tamaño 8. Por último, el arreglo resultante es $\ll$*hasheado*$\gg$ entendiendo que su contenido representa valores hexadecimales. Finalmente, el programa evalúa que el *hash* resultante de la operación sea igual al *expected_hash* recibido como parámetro de entrada en el programa.

```
import "hashes/sha256/256bitPadded" as sha256;

def main(private u32[8] input, u32[8] expected_hash) {
    u32[8] hash = sha256(input);
    assert(hash == expected_hash);
    return;
}
```

Figura 6.8: Programa "main.zok" que define las ZKP generadas por Zokrates

Generación de las ZKP

Tanto el programa construido como la llave de prueba generada a partir del mismo, se almacenan en el *Zokrates-Adapter* en la red de *Fabric*. El *Zokrates-Adapter* haciendo uso de los dos recursos puede generar pruebas de conocimiento cero con su función *createProof* (ver figura 6.9).

```
/**
   * Gets a prescription and creates it's ZKP
   * @param {string} prescriptionId
   * @returns ZKP of prescription if exists
   */
const createProof = async (prescription, hash) => {
  // Transform prescription to Uint8Array
  const processedPrescription = CryptoJS.SHA256(prescription).toString(CryptoJS.enc.Hex);
  const prescriptionBytes = convertHashToU32(processedPrescription);
  const imageBytes = convertHashToU32(hash)
  const preimage = Array.from(prescriptionBytes).map(String);

  // Compute witness
  const { witness } = zokratesProvider.computeWitness(program, [preimage, imageBytes]);

  // Generate and return proof in event
  return zokratesProvider.generateProof(program.program, witness, key);
}
```

Figura 6.9: Función del *Zokrates-Adapter* para generar pruebas ZKP

Verificación de las ZKP

La llave de verificación es utilizada junto al CLI de *Zokrates* para generar el *Smart Contract Verifier*. Este es desplegado en la red de *Ethereum* y define la función pública *verifyTx*, que permite validar una prueba ZKP generada a partir del programa *main.zok*. Esta función es de tipo *View*, lo que significa

que el costo de gas por utilizar dicha función es cero. En la figura 6.10 puede observarse la firma de la función. El *smart contract* de la farmacia llamará a la función *verifyTx* para validar si la prescripción es válida.

```
function verifyTx(
    Proof memory proof, uint[8] memory input
) public view returns (bool r) { }
```

Figura 6.10: Firma de la función *verifyTx* que se encuentra en el *Smart Contract Verifier*

6.4. Decisiones de implementación

A continuación, se desarrollan las decisiones de implementación más importantes. Primero, se analiza la técnica de *doble hashing* que logró mejoras en la performance y permitió reducir los costos de la solución. Luego, se analizan consideraciones de seguridad en *Ethereum* a la hora de escribir los *smart contract* dentro de la red. Por último, se analiza la necesidad de autorizar usuarios en *Fabric* a través de un ID.

6.4.1. Lógica del doble hashing

Una vez implementada la solución con *Zokrates* y realizadas algunas pruebas de performance, se observó un cuello de botella a la hora de generar las ZKP. Antes de utilizar la técnica de doble *hashing*, se tomaba el texto de la prescripción y solamente se ≪*hasheaba*≫ una vez, sin embargo, esto hacía que Zokrates demorara alrededor de 5 minutos para generar la ZKP. La mejora del *doble hashing* tuvo como objetivo reducir el tamaño del arreglo de *bytes* que recibe el programa *main.zok* y que este pasara a recibir un arreglo de tamaño fijo con los bytes de un *hash*. De esta forma, se reduce el tiempo de procesamiento y el costo de gas. A su vez, se aumenta la cantidad de texto que se puede procesar. Esto se debe a que antes se procesaba un arreglo de tamaño fijo, donde cada carácter se correspondía con un *byte*. Ahora el *hash* puede representar cualquier texto con un arreglo de bytes mucho menor.

Dentro de la dApp de *Ethereum* se encuentra la lógica para obtener y procesar los datos proporcionados por el usuario. En particular, posee la función *burnAndConfirmPrescripcion* la cual se muestra en la figura 6.11. La función procesa los datos proporcionados por el usuario y concatena el ID de la prescripción y su descripción para luego ≪*hashear*≫ el texto concatenado dos veces. En la primera ejecución de la función de *hash* se obtiene como resultado el *hash*

del texto plano, luego en la segunda ejecución se obtiene el *hash* resultante de ≪*hashear*≫ los hexadecimales que representan al *hash* del texto concatenado.

```
burnAndConfirmPrescription: async (hash_user, idPrescription, description_prescription, certificate) => {
    const concat = idPrescription.concat(description_prescription);
    const hash_prescription = CryptoJS.SHA256(CryptoJS.SHA256(concat)).toString(CryptoJS.enc.Hex);
    await App.pharmacyContract.methods
    .confirmAndBurnPrescription(certificate, hash_user, hash_prescription)
    .send({ from: App.account });
    App.hash_prescription = hash_prescription;
},
```

Figura 6.11: Función *confirmAndBurnPrescription* de la dApp de *Ethereum*

Con el doble *hashing* realizado se consigue una mejora considerable en términos de performance, ya que se redujo dicho tiempo a menos de un minuto. También mejoró el consumo de gas, ya que ≪*deployar*≫ el contrato *Verifier* en *Ethereum* consumía el máximo de gas permitido por la red[8]. Con el cambio implementado se bajo el consumo a 1603539 de gas[9].

6.4.2. Consideraciones de seguridad en Ethereum

Existe un problema de privacidad a la hora de ejecutar los *smart contract* en los nodos de *Ethereum*. Debido a que un nodo de la red al ejecutar el contrato podría depurar el valor de las variables y acceder a los datos privados durante su procesamiento. Este problema se debe a que actualmente existe la limitante tecnológica de no poder ofuscar el procesamiento de los *smart contract* en los nodos de la red de *Ethereum*, para más detalles ver anexo H.3. Por lo tanto, se tuvo especial cuidado al desarrollar el *Pharmacy Smart Contract*, escribiéndolo con la precaución de no procesar información sensible del usuario. Como se pudo observar en la sección 6.2.4 el código del contrato desarrollado no tiene lógica de negocio alguna. Simplemente, es un intermediario entre el *Gateway* y la dApp de la farmacia, con la salvedad de que también verifica las pruebas ZKP usando el contrato *Verifier*. Para ver más detalles del código de los *smart contract* en *Ethereum* ver el anexo J.

6.4.3. Autorización de usuarios en Fabric

Dado que la autenticación en *Fabric* además de credenciales convencionales utiliza credenciales anónimas, lo único que se sabe del usuario al momento de utilizarlas es que tiene acceso al sistema. Por lo tanto, para saber si la acción de un usuario sobre una prescripción está autorizada, es necesario tener un mecanismo que garantice que está autorizado a acceder a dicha prescripción, sin

[8] El consumo máximo de la red en gas se valuaba en 78.000 USD al tiempo de realización del proyecto

[9] Al tiempo de realización del proyecto esta cantidad de gas se traduce a poco más de 40 USD

revelar la identidad del usuario fuera del sistema. Por tales motivos, se pensó en el *hash* del usuario. Este *hash* se genera para cada usuario de forma única a partir de su identidad en *Fabric* como se muestra en la figura 6.12. Al momento de solicitar una prescripción para ser utilizada, el *hash* del usuario se combina con el ID de la prescripción y se aplica una función de *hash* sobre ambos. Luego, al utilizar esta prescripción se realiza el chequeo presentado en la figura 6.13. Allí se comprueba que concatenando el ID y el *owner* (*hash* del usuario) de la prescripción, y aplicando la función de *hash* se obtiene como resultado el valor hallado en la solicitud de la prescripción.

```go
func (s *SmartContract) GetUserHash(
    ctx contractapi.TransactionContextInterface
) (string, error) {
    id, err := ctx.GetClientIdentity().GetID()
    if err != nil {
        return "", err
    }
    h := sha256.New()
    h.Write([]byte(id))
    userHash :=  hex.EncodeToString(h.Sum(nil)) fmt.Println(userHash)

    return userHash, err
}
```

Figura 6.12: Generación del *hash* del usuario

```go
func (s *SmartContract) BurnPrescription(
    ctx contractapi.TransactionContextInterface, args string
) (string, error) {

    // Recibir el evento y procesarlo...

    auth := incomingEvent.Body["user"].(string)
    authVerification := prescription.Owner + prescription.ID
    h := sha256.New()
    h.Write([]byte(authVerification))
    if auth != hex.EncodeToString(h.Sum(nil)) {
        return "", fmt.Errorf("User is not authorized to access the prescription")
    }

    // Próximos pasos...
}
```

Figura 6.13: Chequeo de autorización

6.5. Limitaciones tecnológicas

En esta sección, se expondrán algunas de las limitaciones tecnológicas que se han enfrentado y cómo se encontraron soluciones para superarlas.

6.5.1. Persistencia de credenciales anónimas en Fabric

Se realizó un cambio en la SDK de *Fabric* para que las credenciales anónimas puedan persistirse. El estado actual de la implementación de la SDK solo permite crearlas y utilizarlas en tiempo de ejecución, pero para lograr la implementación del escenario era necesario poder utilizarlas más tarde. Para ello, se llevó a cabo una modificación que permite realizar lo requerido, aunque no siga el patrón que utiliza la SDK para los tipos tradicionales de credenciales. Actualmente, esta funcionalidad se encuentra en desarrollo por la *Hyperledger Foundation*. En la figura 6.14 se muestra en primer lugar la firma de la función existente en la SDK *idemixEnroll* que fue separada en dos funciones, una para poder persistir la respuesta de la *Fabric CA* y otra para obtener el objeto *Enrollment*.

```java
// Consulta a la Fabric CA y crea un IdemixEnrollment
// Funcion ya existente en la SDK
public Enrollment idemixEnroll(Enrollment enrollment, String mspID) {}

// Retorna la respuest de Fabric CA como un Json
public JsonObject getIdemixEnrollmentJson(Enrollment enrollment, String mspID) {}

public Enrollment getIdemixEnrollmentFromJson(JsonObject json) throws Exception {

    String mspID = json.getString("mspID");
    IdemixIssuerPublicKey ipk = getIssuerPublicKey(json.getString("ipk")); // Issuer Public Key
    PublicKey rpk = getRevocationPublicKey(json.getString("rpk")); // Revocation Public Key
    BIG k = BIG.fromBytes(Base64.getDecoder().decode(json.getString("k")));

    // Deserialize idemix credential
    String credential = json.getString("cred");
    byte[] credBytes = Base64.getDecoder().decode(credential.getBytes(UTF_8));
    Idemix.Credential credProto = Idemix.Credential.parseFrom(credBytes);
    IdemixCredential cred = new IdemixCredential(credProto);

    String ou = json.getString("ou");
    int role = json.getInt("role"); // Encoded IdemixRole from Fabric-Ca

    // Deserialize idemix cri (Credential Revocation Information)
    String criStr = json.getString("cri");
    byte[] criBytes = Base64.getDecoder().decode(criStr.getBytes(UTF_8));
    Idemix.CredentialRevocationInformation cri = Idemix.CredentialRevocationInformation.parseFrom(criBytes);

    return new IdemixEnrollment(ipk, rpk, mspID, k, cred, cri, ou, role);
}
```

Figura 6.14: Modificación Realizada en la *Fabric* CA SDK

6.5.2. Fabric Idemix Connector

El *Fabric Idemix Connector* es un nuevo conector dentro del *Gateway* desarrollado en Java utilizando el *framework Spring boot*. Es una REST API al igual que otros conectores, con un único método. El objetivo de este conector es enviar a ejecutar una operación de un *smart contract* en *Fabric* pero autenticándose usando credenciales anónimas. Se debió separar del conector existente, ya que *Idemix* no se encuentra implementado en la SDK *Node* de *Fabric* y se tuvo que implementar en Java. El nuevo conector solo permite enviar mensajes hacia *Fabric*. Para recibir mensajes se sigue empleando el conector *Fabric* original, ya

que no es necesario hacer uso de credenciales anónimas para escuchar un evento de *Fabric*.

6.6. Desafíos de implementación

Uno de los desafíos de implementación se dio al comienzo de la etapa de desarrollo, al intentar usar *Idemix* en *Fabric*. Para este paso es requerido configurar que el MSP de *Fabric* acepte credenciales anónimas y además, configurar los permisos adecuados para un usuario *Idemix*. Pero la documentación no lo deja claro y no especifica en detalle qué cambios de configuración son necesarios, entre otras cosas. Por lo que se realizó una búsqueda de implementaciones de *Fabric* configuradas con *Idemix* donde se logró encontrar un ejemplo que tenía las configuraciones necesarias. Luego de unas pequeñas modificaciones para adaptarse al uso requerido por el proyecto, fue posible desplegar un ambiente *Fabric* con soporte para *Idemix*.

Otro de los desafíos encontrados fue al momento de realizar la integración con *Zokrates*. La documentación de ciertas funciones no era clara o directamente no estaban documentadas. Por un lado, faltaba información acerca de las implementaciones de *sha256* dentro del lenguaje de programación de *Zokrates*. Para solventar esto se analizaron y ≪*testearon*≫ todas las implementaciones que ofrece *Zokrates* de *sha256*, hasta encontrar la indicada para la técnica de *doble hashing*. Por otro lado, hubo un contratiempo al intentar utilizar las claves y el programa *main.zok* junto a la librería de *Zokrates* para *Node.js*. Este contratiempo se dio en el desarrollo del *Zokrates Adapter*. El problema radicaba en que la documentación de esta librería es muy limitada y básica, por lo que al realizar una búsqueda por mejor documentación o soporte de la *toolbox*, se encontró un *chat*[10]. En este existían varias discusiones acerca de la librería de *Node.js*. Luego de una búsqueda exhaustiva en el chat, se encontró un código que incluía un ejemplo de cómo utilizar las claves en la librería *Node.js*. Aunque el ejemplo no se ajustaba del todo a la necesidad de uso, se pudo llegar a una implementación correcta y funcional, lo cual permitió finalmente configurar *Zokrates* e integrarlo a la solución.

[10] `https://app.gitter.im/#/room/#ZoKrates_Lobby:gitter.im`

Capítulo 7

Evaluación de la solución

En esta sección se presentarán los resultados obtenidos luego de una serie de pruebas realizadas sobre la solución implementada. Primero, en la sección 7.1, se evaluará la performance de la solución sobre el caso de uso Usar Prescripción. Luego, en la sección 7.2, se muestra un estudio de los gastos en los que se incurre por el uso de *Ethereum* en la solución. En la sección 7.3, se mencionarán brevemente algunos de los pros y contras hallados al evaluar la solución. Finalmente, en la sección 7.4, se realizará una comparación con un trabajo similar.

7.1. Pruebas de performance

A continuación, se presentan las pruebas realizadas mediante la herramienta *JMeter*[1] que permite simular el uso de la aplicación por varios usuarios en simultáneo.

7.1.1. Escenario

Para la prueba se utilizó como escenario el caso de uso Usar Prescripción descrito en la subsección 5.2.2, ya que es el único que involucra una operación de interoperabilidad. A continuación, se reitera en la figura 7.1 el diagrama para facilitar la lectura.

[1] https://jmeter.apache.org/

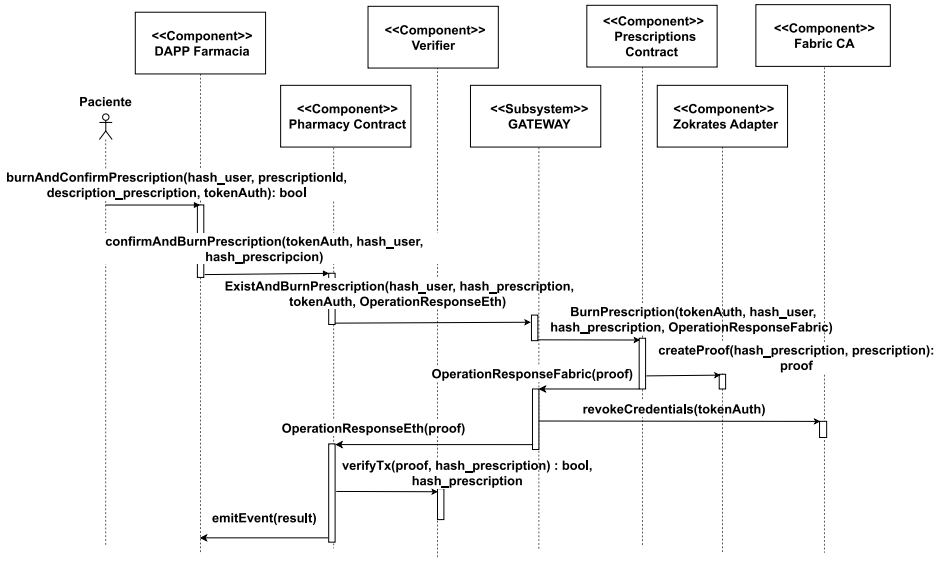


Figura 7.1: Diagrama de secuencia para usar una prescripción

7.1.2. Hardware

Todo el ambiente se ejecutó en una PC de escritorio usando *Docker* a través de WSL[2]. La PC cuenta con las especificaciones presentadas en el cuadro 7.1.

CPU	AMD Ryzen 5 3600
RAM	32GB DDR4 3200MHz
SSD	Sabrent Rocket Q NVMe
GPU	Nvidia RTX 3070

Cuadro 7.1: Especificaciones PC

7.1.3. Topología

En la PC descrita anteriormente corren tanto el *software* de prueba *JMeter* como las redes de *Ethereum* y *Hyperledger Fabric*, las dApps, y el *Gateway*. Además, se creó un pequeño servicio REST que permite consultar la *blockchain* de *Ethereum* y esperar por la respuesta proveniente del evento. Desde *JMeter* se actúa simulando ser el actor paciente en el caso de uso especificado en la subsección 7.1.1, mientras que este servicio REST simula ser la dApp Farmacia. A continuación, se presenta en la figura 7.2 lo descrito anteriormente.

[2]WSL o *Windows Subsystem For Linux*, en este caso en su versión 2, es una máquina virtual liviana que permite correr un kernel completo de Linux en Windows

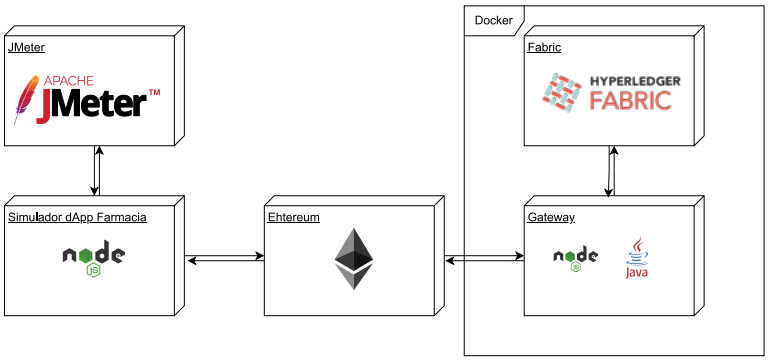


Figura 7.2: Diagrama de topología de la prueba de performance

7.1.4. Resultados

Se realizaron cinco ejecuciones consecutivas de diez peticiones cada una, donde se obtuvieron los resultados de latencia de peticiones resumidos en el cuadro 7.2. Con estos resultados y la observación del sistema durante las pruebas, se determinó un cuello de botella en la solución. Este viene dado por la librería usada para la creación de las ZKP incluso después de las mejoras realizadas (doble *hashing* 6.4.1). El tiempo en la creación (**createProof** en la figura 7.1) y verificación de la ZKP (**verifyTx** en la figura 7.1) es el principal responsable de la latencia en las peticiones. Como información adicional, en el cuadro 7.3 se presentan los datos obtenidos para **createProof** y **BurnPrescription** del **Prescriptions Contract**[3].

Mediana	9947 ms
Promedio	33441 ms
Min.	2214 ms
Máx.	133064 ms
Throughput	0.65 requests/s

Cuadro 7.2: Resultados

	Duración de *createProof*	Duración Total *BurnPrescription*
Promedio	7333 ms	7335 ms
Mediana	6055 ms	6056 ms
Máximo	17637 ms	17645 ms
Mínimo	4818 ms	4818 ms

Cuadro 7.3: Latencia al generar la ZKP con Zokrates

[3]Recordar que **createProof** es una operación del Zokrates Adapter y BurnPrescription es del Prescriptions Contract

Del cuadro 7.3 se puede deducir entonces que la mayor parte del tiempo que conlleva una petición se debe a la creación de la ZKP, y que el resto de la operación **BurnPrescription** del **PrescriptionsContract** no agrega una latencia significativa a la petición. Siendo esta en la mayoría de los casos no superior a 1 ms, y en promedio 2 ms.

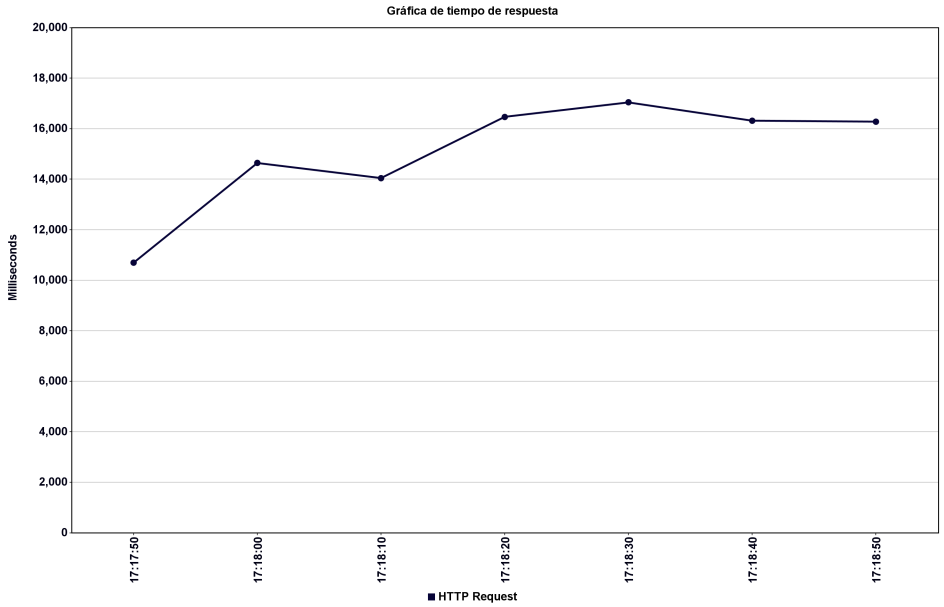


Figura 7.3: Latencia de las peticiones a lo largo del tiempo[4]

Además, en la figura 7.3, se observa que las peticiones parecen tomar el doble de los tiempos documentados en el cuadro 7.3. Esto se debe a que la verificación de la prueba conlleva una cantidad de cómputo similar a la creación, y esto no se ve contemplado en el cuadro 7.3.

Otro problema observado con menor impacto, se debe al uso de *NodeJS* para el desarrollo del *Gateway*, ya que esta tecnología es monohilo. Por ejemplo, en el *Ethereum connector* si se encolan muchos mensajes de respuesta hacia *Ethereum*, los mensajes de ida hacia el *Router* tienen una demora extra debido a que no puede procesar estas operaciones como hilos de ejecución separados.

7.2. Estudio de consumo de gas

El estudio de consumo de gas tiene como objetivo ver los costos incurridos al utilizar la solución por cada una de sus partes. Para evaluar este costo se ejecutó el caso de uso *Usar Prescripción*, al igual que en la sección 7.1. En particular,

[4]Los valores del eje X hacen referencia a la hora en la que fueron realizadas las peticiones, tomando peticiones realizadas cada 10000 ms

este caso de uso es el único que implica costos en *Ethereum*. Además, se tomó en cuenta el costo inicial de desplegar los *smart contract* en *Ethereum*.

Para el despliegue inicial de los *smart contract*, el costo obtenido en el caso del *PharmacyContract* son 841,386 unidades de gas que equivalen a $0,01ETH$ [5]. Mientras que para el *Verifier* el costo alcanza las 1,603,539 unidades de gas equivalentes a $0,03ETH$ [6]. El costo elevado de este último se debe en gran parte a que posee código no optimizado debido a que fue generado automáticamente por una herramienta 6.4.1. Estos costos serán asumidos una única vez por la Farmacia, para el *PharmacyContract*, y la organización que posea la responsabilidad del *Gateway* para el *Verifier*.

Por otro lado, recordando el caso de uso, la primera operación (**confirmAndBurnPrescription** del **PharmacyContract**) consiste en enviar un evento por parte del usuario (a través de la dApp Farmacia) a *Ethereum*. Como este evento escribe en la *blockchain*, tiene un costo asumido por el usuario que alcanza las 97.304 unidades de gas, que equivalen a 0,0019 ETH [7], que abonará como costo de procesar cada transacción. Por lo cual, no sería eficiente si el medicamento a comprar es de un valor cercano al costo de procesar la transacción.

Finalmente, la validación de la prueba ZKP (**verifyTx** en el **Verifier**) es el último paso en el que se incurren gastos en *Ethereum*. Estos gastos deben ser costeados por el *Gateway* en cada validación realizada. Estos gastos alcanzan las 427.646 unidades de gas, equivalentes a 0,0085 ETH [8]. Esto se debe a la ejecución del contrato *Verifier* el cual posee código no optimizado, lo que lo vuelve costoso. Una posible forma de reducir este costo es guardar directamente la prueba en *Ethereum* y verificarla luego por separado. A continuación, se resume en el cuadro 7.4 los valores mencionados.

7.3. Discusión de la solución

Dados los resultados obtenidos en las pruebas, es pertinente evaluar algunos puntos claves positivos y negativos de la solución.

Por un lado, como se vio en capítulos anteriores, la solución logra correctamente cumplir con preservar la confidencialidad y privacidad de los datos del usuario, y es una solución que aunque implementada específicamente para un caso de uso, es posible generalizar para otra diversidad de escenarios que requieran hallar una prueba de la posesión de la pre imagen de un *hash*. Otra virtud de la solución es también su versatilidad para ser desplegada utilizando contenedores de manera sencilla que simplifica el uso de la solución.

Por otro lado, la solución no está exenta de debilidades, entre ellas se encuentra la baja performance y los costos elevados de cada transacción. En cuanto a la performance, la creación de ZKP incurre en un tiempo de respuesta alto para un usuario que desea obtener su medicamento de forma rápida, por lo cual sería

[5] $1,55USD$ al 3/2/2023
[6] $49,66USD$ al 3/2/2023
[7] 3.5 USD al 3/2/2023
[8] 14.07 USD al 3/2/2023

Operación	**Costo en Miles Gas**	**Costo ETH**	**Costo USD Usuario**	**Costo USD Gateway**
Deploy PharmacyContract	841	0.01	0	16,55 única vez
Deploy Verifier	1603	0.03	0	49,66 única vez
confirmAndBurnPrescription de *PharmacyContract*	97	0.0019	3.5	0
verifyTx de *Verifier*	428	0.0085	0	14.07
Costo total de solicitar un medicamento	525	0.0104	3.5	14.07

Cuadro 7.4: Resumen resultados de estudio de gas

uno de los primeros aspectos a tratar para mejorar la solución. En lo que refiere a los costos elevados, esto es relativo al precio del bien que se está adquiriendo. En caso de medicamentos de bajo costo y disponibilidad al público general, la solución no resulta de gran utilidad, ya que por adquirir un medicamento de por ejemplo valor 2 USD el usuario pagaría 3.5 USD en comisiones. Sin embargo, en el caso de medicamentos más costosos, por ejemplo, de 500 USD o más, la comisión fija de 3.5 USD ya no sería un problema tan grande, ya que solo aportaría un pequeño costo extra.

7.4. Comparación con otros proyectos

Sobre el final del presente proyecto, se presenta un trabajo (Schlatt, Sedlmeir, Traue, y Völter, 2022) cuyo escenario es muy similar al utilizado en este proyecto. La diferencia más grande con el escenario del artículo es la presencia de la entidad aseguradora, que es la poseedora de la información del seguro de salud del usuario paciente del sistema. Esta tiene la responsabilidad activa de validar si el medicamento solicitado en la prescripción por parte del paciente está cubierto en el seguro. En caso afirmativo, cubre el costo de este, pagándole directamente a la farmacia. Aun teniendo en cuenta esta diferencia entre los escenarios, la realidad es que son muy similares. Sin embargo, los proyectos y sus objetivos son distintos. Su trabajo se enfoca en resolver desafíos asociados a la gestión de prescripciones médicas: la descentralización del sistema de gestión, la prevención de la doble utilización de las prescripciones y el manejo de datos confidenciales.

El trabajo comparado propone la utilización de dos *blockchain* como solu-

ción: *Quorum*[9] e *Hyperlegder Indy*[10]. *Quorum* es un *fork* privado de *Ethereum* que agrega características de privacidad, mientras que *Hyperledger Indy* es una *blockchain* privada que se caracteriza por su madurez en la gestión descentralizada de identidades. Además de que ambas *blockchain* son privadas, lo cual ya es una diferencia significativa, el tipo de interoperabilidad buscado entre ellas es distinto al que se busca resolver en nuestro proyecto. El tipo de interoperabilidad que se busca en nuestro proyecto es entre las *blockchain* (tipo B en la figura 2.6). Sin embargo, en el trabajo comparado, la interoperabilidad se da a través de las aplicaciones (tipo A en la figura 2.6).

Diferencia	Nuestro Proyecto	Proyecto Comparado
Entidad aseguradora en el escenario planteado	El escenario planteado no contempla a una entidad aseguradora.	La entidad aseguradora es una pieza clave en el escenario planteado.
Desafíos atacados	El objetivo es cumplir con los requisitos de privacidad al interoperar una *blockchain* publica con una privada	Se centra en las prescripciones médicas y sus desafíos asociados: respetar la confidencialidad de la información de los pacientes y evitar el doble gasto de las prescripciones médicas.
Blockchain interoperadas	*Ethereum - Hyperledger Fabric* (publica - privada)	*Quorum - Hyperledger Indy* (privada - privada)
Tipo de interoperabilidad	Se trabaja con una interoperabilidad directa entre una *blockchain* y otra.	Se trabaja con una aplicación la cual interopera una *blockchain* con otra, en otras palabras las *blockchain* no interoperan directamente entre ellas
Solución de interoperabilidad utilizada	*Gateway*	Ninguna

Cuadro 7.5: Tabla comparativa con las principales diferencias entre nuestro proyecto y el trabajo presentado.

En la tabla 7.5 se puede observar de forma concisa las diferencias mencionadas. La presencia de la aseguradora en el trabajo presentado hace que su escenario y la solución general sea más fiel a la realidad, pero considerando los objetivos planteados en este proyecto, su ausencia no impacta negativamente a este. Esta diferencia en el escenario se corresponde con las diferencias en los desafíos atacados. El trabajo comparado tiene como objetivo final solucionar los desafíos asociados a las prescripciones médicas y la tecnología *blockchain* simplemente es un medio para dicho fin. En contrapartida, nuestro proyecto tiene como objetivo principal la tecnología *blockchain* y en particular la interoperabilidad respetando los requisitos de privacidad. Esta gran diferencia impacta negativamente en nuestro proyecto con respecto al trabajo comparado, ya que este último tiene más libertad a la hora de elegir tecnologías y arquitecturas, como por ejemplo las *blockchain* utilizadas fueron seleccionadas porque eran la mejor opción para el escenario planteado; sin embargo, en nuestro proyecto estas fueron dadas como parte del problema a resolver. Finalmente, la última gran diferencia observada es el tipo de interoperabilidad utilizado. Esta diferencia también impacta negativamente a nuestro proyecto, ya que la interoperabilidad entre una *blockchain* y otra de forma directa (tipo B) es un tipo de arquitectura

[9] https://consensys.net/quorum/
[10] https://www.hyperledger.org/use/hyperledger-indy

poco utilizado en la industria, lo que hace que su desarrollo esté en etapas de menor madurez.

Dejando las diferencias entre las soluciones a un lado, ambas resuelven un escenario similar. Por lo tanto, es interesante comparar el rendimiento y costo de las soluciones. En el artículo del trabajo comparado se realiza una evaluación de cuáles serían los costos de gas y de performance si en vez de usar la *blockchain* de *Quorum* se utilizara la red pública de *Ethereum*. Utilizando esta nueva configuración, el costo de cada prescripción emitida por el médico tiene un costo de 85.000 de gas[11]. Por otro lado, utilizar dicha prescripción tiene un costo de 15.000 de gas[12]. Comparándolo con las pruebas realizadas en la sección 7.2, los resultados indican que la solución más conveniente por costo de gas es la del trabajo comparado.

En el caso de la performance, el trabajo comparado soporta unas 10 transacciones por segundo, donde cada una de estas reporta una demora de al menos un minuto, siendo este tiempo además muy volátil. Esta métrica utilizada para medir la performance de la solución difiere en la utilizada por nuestro proyecto en la sección 7.1. Por tal motivo, no es posible realizar una comparación en este apartado entre ambos proyectos.

Por último, hay que tener en cuenta que tanto los valores de gas como los de performance mejoran notoriamente en el trabajo comparado utilizando la configuración original. Es decir, la *blockchain* de *Quorum* en lugar de la de *Ethereum*. En la tabla 7.6 se pueden observar los resultados comparados.

	Nuestro Proyecto	Proyecto Comparado
Costo de gas por desplegar el *Smart Contract* en *Ethereum*	2.444.925 unidades de Gas	?
Costo de gas por prescripción utilizada	524.950 unidades de Gas	100.000 unidades de Gas
Transacciones por segundo	0,65 tx/s (transacciones completadas por segundo)	10 tx/s (transacciones emitidas por segundo)

Cuadro 7.6: Tabla comparativa con los resultados obtenidos de las soluciones

En conclusión, los resultados muestran que la solución del trabajo comparado es más conveniente en términos de costo. Sin embargo, hay que considerar que desde su concepción dicha solución tuvo en cuenta este factor y el de performance. En el caso de nuestro proyecto, aunque estos factores fueron tomados en cuenta, la prioridad fue llegar a un prototipo funcional. Como se explicó en la sección 6.4.1, la técnica de *doble hashing* es una de las mejoras realizadas para mejorar la performance y el costo de la solución construida. También impacto negativamente en nuestra solución que las tecnologías del *Gateway* fueron

[11] Lo que equivale a 7,25 USD al 20/4/2023.
[12] lo que equivale a 1.28 USD al 14/2/2023

escogidas previamente. Por ejemplo, como se mencionó en la sección 7.1.4, la naturaleza *single thread* de *Node.js* supone un cuello de botella que afecta a la performance. Finalmente, por una cuestión de tiempo no se pudieron agregar otras mejoras a la solución, como reducciones sobre el costo de gas, esta y otras mejoras se mencionan en la sección 9.2.

Capítulo 8

Gestión de proyecto

En las siguientes secciones se detalla la gestión general del proyecto. Primero, en la sección 8.1 se detalla cuál fue la planificación inicial del proyecto y los mecanismos de comunicación a seguir. Luego, en la sección 8.2 se brinda el cronograma final y se detallan los motivos que llevaron a aplazarlo. Finalmente, en la sección 8.3 se describen las herramientas utilizadas para la gestión del proyecto y los usos dados para cada una de ellas.

8.1. Organización del proyecto

Al iniciar el proyecto se estableció un plan de trabajo que acompañaría al grupo en el transcurso del mismo. El cual se compone de una serie de tareas a realizar y sus tiempos estimados como se muestra en la Figura 8.1.

Tareas	Abr	May	Jun	Jul	Ago	Sep	Oct	Nov
Plan de trabajo	X							
Estudio de conceptos	X	X						
Definir requerimientos de proyecto	X							
Definir casos de uso		X						
Relevar soluciones existentes			X					
Análisis y diseño de la solución final			X	X				
Implementación de la POC			X	X				
Implementación de la solución final				X	X	X		
Evaluación de la solución final						X	X	X
Test de performance							X	
Test general								X
Documentación						X	X	X

Figura 8.1: Cronograma inicial

A la hora de planificar el proyecto se acordó realizar entregables de cada una de las tareas establecidas para presentarle al tutor y así obtener una retroalimentación de lo trabajado hasta el momento. Los entregables, en su gran mayoría, fueron documentos y diagramas asociados a las tareas realizadas, como por ejemplo un documento con todas las soluciones de interoperabilidad existentes y analizadas, un diagrama del diseño de la solución, entre otras.

Para la organización del proyecto se decidió mantener de manera constante dos reuniones semanales entre todos los integrantes del grupo. De esta forma, el equipo sabía en qué estaba trabajando cada uno, además de intercambiar ideas y establecer los objetivos a cumplir en el transcurso de los próximos días hasta la siguiente reunión. Por otro lado, se mantuvieron reuniones periódicas cada dos semanas con el tutor del proyecto. Las mismas tenían el objetivo de poder evacuar dudas, mostrar el avance de lo realizado hasta el momento y lo que se intentaría mostrar en la siguiente reunión.

8.2. Cronograma final

A continuación, se muestra en la Figura 8.2 el cronograma final del proyecto.

Tareas	Abr	May	Jun	Jul	Ago	Sep	Oct	Nov	Dic	Ene	Feb	Mar	Abr	May	Jun	Jul
Plan de trabajo	X															
Estudio de conceptos	X	X														
Definir requerimientos de proyecto	X															
Definir casos de uso		X														
Relevar soluciones existentes			X	X												
Análisis y diseño de la solución final			X	X	X											
Implementación de la POC				X	X	X										
Implementación de la solución final					X	X	X									
Evaluación de la solución final							X	X	X	X	X	X				
Test general								X	X	X	X	X				
Test de performance									X	X	X	X				
Documentación							X	X	X	X	X	X	X	X	X	X

Figura 8.2: Cronograma final

Hubo diferentes contratiempos a lo largo del proyecto que provocaron un cambio en el cronograma inicial de la Figura 8.1. Al comparar el cronograma inicial y el final de las figuras 8.1 y 8.2 respectivamente, se puede observar cómo se terminaron extendiendo los tiempos en las tareas de *Relevar soluciones existentes* y *Análisis y diseño de la solución final*. Para el primer caso, dicha extensión de tiempo se debió a una mala estimación al inicio del proyecto, ya que la tarea llevó más tiempo de lo esperado. El relevamiento de las soluciones traía consigo un análisis de la privacidad al interoperar las *blockchain*. Dicho análisis se dividió de acuerdo a si la interoperabilidad se daba entre *blockchain* públicas, *blockchain* privadas con *blockchain* públicas y entre *blockchain* privadas. Por otro lado, la tarea de *Análisis y diseño de la solución final* tuvo una extensión de tiempo mayor de la prevista debido a la complejidad de la misma. Como se menciona en la sección 4.5.1, se empezó abordando este punto con una solución distinta a la solución lograda y se necesitó de más tiempo para finalizarla.

La documentación fue otro punto donde la estimación al inicio del proyecto falló. Esto, sumado a los retrasos sufridos en las etapas anteriores, hicieron que no se pudiera llegar con la documentación adecuada para el mes de diciembre, como se tenía estimado originalmente.

Otra demora fue la necesidad de contar con un equipamiento mínimo de hardware que soporte la ejecución de todo el sistema en su conjunto para poder realizar el *testing* de performance adecuado. Hizo falta la compra de memoria RAM para poder realizar el *testing* correspondiente. Si bien esto no fue un inconveniente monetario debido a que uno de los integrantes ya tenía planeado hacerlo, agregó un tiempo de espera para realizar dicha tarea.

8.3. Herramientas

Para poder organizar el desarrollo del proyecto fue necesario utilizar las siguientes herramientas.

GitLab

GitLab, conocido por sus características de control de versiones de código, utiliza Git, que es un sistema de control de versiones distribuido, para llevar un registro de todos los cambios realizados en el código de un proyecto. Esto permite a los desarrolladores trabajar de manera colaborativa en un proyecto y revertir fácilmente a versiones previas si es necesario.

Google Drive

Google Drive es un servicio de almacenamiento en la nube de Google que permite a los usuarios guardar y acceder a sus archivos en línea. Si bien permite la colaboración en tiempo real para trabajar en documentos, el uso que le dimos fue exclusivamente para el almacenamiento de trabajos académicos útiles que se iban encontrando para el proyecto.

Overleaf

Overleaf es una plataforma en línea para la creación y la colaboración de documentos LaTeX. LaTeX es un sistema de formato de texto que se utiliza ampliamente en la comunidad académica para crear documentos con un formato preciso y profesional, y en el cual el grupo tenía experiencia obtenida en cursos anteriores. Overleaf simplifica el proceso de escribir y formatear documentos en LaTeX, y ofrece herramientas de colaboración en tiempo real que le permitió al equipo trabajar al mismo tiempo en la construcción de este documento.

Discord

Discord es una plataforma de comunicación en línea que ofrece una variedad de características de comunicación, incluyendo: chat de texto, voz y vídeo. Además, permite a los usuarios crear y unirse a servidores de Discord para comunicarse con otros miembros. El grupo venía utilizando esta herramienta desde hace varios semestres y ya se manejaba con un servidor de comunicación propio.

Meet

Google Meet es una plataforma de videoconferencia en línea de Google que permite a los usuarios realizar reuniones, llamadas de voz y vídeo en línea, de manera segura y fácil. Se utilizó esta herramienta para realizar las reuniones con el tutor de una manera organizada, ya que se agendaban con anticipación.

ClickUp

ClickUp es una solución de gestión de proyectos en línea fácil de usar y accesible que permitió al equipo planificar, organizar y monitorear todas las tareas de manera efectiva. También se utilizó esta plataforma para la elaboración de documentación a modo de entregables mencionada en la sección 8.1.

Diagrams.net

Diagrams.net es una herramienta en línea para crear diagramas y gráficos de forma colaborativa. Es una opción popular para la creación de diagramas de flujo, mapas mentales, diagramas de redes y muchos otros tipos de diagramas y gráficos. Esta fue una de las herramientas más utilizadas durante el proyecto, debido a la facilidad que ofrece para diagramar y los beneficios que esto traía a la hora de conceptualizar problemas, debatir ideas y diseñar soluciones.

Capítulo 9

Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones del trabajo realizado y se discute el trabajo a futuro. La sección 9.1 presenta las conclusiones, mientras que la sección 9.2 el trabajo a futuro.

9.1. Conclusiones del proyecto

Como principal conclusión del proyecto, fue posible realizar el diseño e implementación de una solución de *blockchain* que permite preservar dos de los requisitos de privacidad relevados: privacidad de la identidad y privacidad del mensaje. Para llevar adelante este objetivo se estudiaron los requerimientos de privacidad en *blockchain*, así como en interoperabilidad. De acuerdo a los requerimientos relevados, se clasificaron las soluciones de interoperabilidad existentes, teniendo en cuenta si las mismas conservaban dichos requisitos de privacidad o no. A partir de esta clasificación se decidió extender la solución de *Gateway*, de modo que satisficiera los requisitos de privacidad de la identidad y privacidad del mensaje. Con base en este diseño se implementó un prototipo extendiendo una implementación previa de *Gateway* desarrollada por el proyecto de grado (Mathías Castro, 2023). Dicho prototipo se validó con: el escenario de uso planteado basado en prescripciones médicas, las pruebas de performance y los análisis de costos realizados. Finalmente, se documentó el trabajo realizado, junto con el resultado de las pruebas y las conclusiones tomadas a partir de estas.

Luego de realizado el proyecto se observa que las soluciones de interoperabilidad en *blockchain* continúan evolucionando a alta velocidad. Al día de hoy, son varias las soluciones existentes y funcionales para interoperar *blockchain* públicas. Por el contrario, los avances en interoperabilidad entre *blockchain* públicas con *blockchain* privadas son muy pocos. Esto se debe principalmente a que los esfuerzos de la industria siguen centrándose en otros aspectos como mejorar la

performance, los costos de la red, la capacidad de procesamiento y la velocidad de las transacciones. Sin embargo, en el caso de la academia es diferente, la cantidad de artículos y estudios relacionados con interoperabilidad en *blockchain* crece mes a mes, incluso para el par público-privado. Este aumento en la popularidad está intrínsecamente relacionado con la fama que están adquiriendo las pruebas de conocimiento cero. Aunque la industria por el momento solo ve en las ZKP el futuro de la escalabilidad y la performance de la red, es cuestión de tiempo que comiencen a ser utilizadas como parte de soluciones de privacidad, probablemente con un enfoque similar al utilizado en este proyecto.

Personalmente, el equipo concluye que el ecosistema *blockchain* está falto de madurez en algunas áreas, entendiéndose al ecosistema como *frameworks* de desarrollo, tamaño de la comunidad, material académico, documentación, entre otros. Algunas áreas del ecosistema como el material académico y la documentación necesitan mejorar. Si bien el manejo y el desarrollo básico en las *blockchain* está debidamente documentado, a medida que se comienza a complejizar la tarea a realizar, la documentación escasea al punto de tener que leer directamente el código fuente de la tecnología. Esto se debe a que es una tecnología que crece principalmente por el impulso de la industria, siendo la academia la que sigue el avance un paso por detrás. La academia aún tiene mucho por investigar y formalizar, lo cual es crucial para un avance sólido de la tecnología. A su vez, dentro de *blockchain*, el área de interoperabilidad aún tiene mucho margen de mejora. Aunque existen varias soluciones para interoperar *blockchain* públicas, estas aún son difíciles de utilizar. Finalmente, la privacidad aplicada a interoperabilidad está lejos de ser el campo más popular, sin embargo, poco a poco comienza a ganar relevancia y la industria nota la necesidad de progresar en este aspecto.

9.2. Trabajo a futuro

En esta sección se desarrollan algunas propuestas para continuar con el trabajo realizado y mejorarlo en el futuro.

Soportar otros tipos de operaciones de interoperabilidad

Dentro de los objetivos del proyecto se encontraba la idea de soportar las operaciones de tipo *Exchange*, sin embargo, este objetivo quedo fuera del alcance del proyecto. Entonces, la primera propuesta consiste en agregar a la solución soporte a este tipo de operaciones e incluso agregar soporte también para las operaciones de tipo *Transfer*.

Mejora de la performance

Actualmente, generar la prueba ZKP automáticamente con *Zokrates* es lo que consume mayor cantidad de tiempo y degrada la performance de la solución. Esto podría subsanarse dejando de utilizar pruebas ZKP autogeneradas y

construirlas de forma personalizada y especifica al problema. Por ejemplo, desarrollar un código que construya una ZKP específica para el escenario, de esta forma la generación de la prueba no sufre degradaciones de performance debido a generalizaciones.

Reducción del costo de gas

El costo de gas de ejecutar el caso de uso de utilizar la prescripción podría reducirse. Para esto, el equipo propone almacenar la ZKP generada en *Ethereum* y que posteriormente de forma *off-chain* se valide que esta sea correcta. Con este cambio se omite el costo de ejecutar la función *VerifyTx*, ya que ejecutar esta función desde una dApp no tiene costo porque no realiza cambios en el *ledger* de *Ethereum*.

Descentralización de la solución

La solución actualmente contradice de alguna manera el espíritu de la comunidad *blockchain*. Es decir, que la toma de decisiones sea descentralizada y las operaciones sean validadas. Esto se debe a que el *Gateway* se encuentra centralizado y las transacciones entre las *blockchain* no son validadas. Queda pendiente para futuros trabajos mejorar estos aspectos de la solución.

Referencias

Adviser, U. G. C. S. (s.f.). Distributed ledger technology: beyond block chain. Descargado de https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf (Accedido: 2023-4-09)

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., ... Yellick, J. (2018). Hyperledger fabric: A distributed operating system for permissioned blockchains. En *Proceedings of the thirteenth eurosys conference*. Descargado de https://dl.acm.org/doi/pdf/10.1145/3190508.3190538 (Accedido: 2023-01-19)

Benedikt Bünz, M. Z. . D. B., Shashank Agrawal. (2020). Zether: Towards privacy in a smart contract world. En *Financial cryptography and data security* (p. 423-443). Cham.

Bernal Bernabe, J., Canovas, J. L., Hernandez-Ramos, J. L., Torres Moreno, R., y Skarmeta, A. (2019). Privacy-preserving solutions for blockchain: Review and challenges. *IEEE Access*, *7*, 164908-164940. doi: 10.1109/ACCESS.2019.2950872

Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., y Wu, H. (2020). Zexe: Enabling decentralized private computation. En *2020 ieee symposium on security and privacy (sp)* (p. 947-964).

Bruno Bradach, G. L. L. G. R. u. R., Juan Nogueira. (s.f.). A gateway based interoperability solution for permissioned blockchains. Descargado de Aunnopublicado (Accedido: 2023-03-21)

Buterin, V. (s.f.-a). Ethereum. Descargado de https://ethereum.org/en/ (Accedido: 2023-4-3)

Buterin, V. (s.f.-b). Introduction to smart contracts. Descargado de https://ethereum.org/en/developers/docs/smart-contracts/ (Accedido: 2022-4-09)

Buterin, V. (s.f.-c). Proof-of-stake (pos). Descargado de https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/ (Accedido: 2022-2-27)

Buterin, V. (s.f.-d). Proof-of-work (pow). Descargado de https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/ (Accedido: 2022-2-27)

Camenisch, J., y Herreweghen, E. V. (2002). Design and implementation of the idemix anonymous credential system. *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, 21–30. Descargado de `http://www.csl.mtu.edu/cs6461/www/Reading/Camenisch02.pdf` (Accedido: 2023-01-16)

Castro, M., y Liskov, B. (1999). Practical byzantine fault tolerance. En *Proceedings of the third symposium on operating systems design and implementation* (p. 173–186). USA: USENIX Association.

Cosmos. (s.f.). `https://cosmos.network/`. (Accessed: 2023-3-09)

Daira Hopwood, T. H. N. W., Sean Bowe. (s.f.). Zcash protocol specification. Descargado de `https://github.com/zcash/zips/blob/main/protocol/protocol.pdf` (Accedido: 2023-3-21)

Dan Wang, Y. W., Jindong Zhao. (s.f.). A survey on privacy protection of blockchain: The technology and application. Descargado de `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9093015` (Accedido: 2023-3-3)

Ethereum.org. (January 17, 2023). *Zero-knowledge proofs.* `https://ethereum.org/en/zero-knowledge-proofs/`. (Accedido: 2023-01-19)

Feng, Q., He, D., Zeadally, S., Khan, M. K., y Kumar, N. (2019). A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications*, *126*, 45-58. Descargado de `https://www.sciencedirect.com/science/article/pii/S1084804518303485`

Gans, J. S. (s.f.). The fine print in smart contracts. Descargado de `https://www.nber.org/system/files/working_papers/w25443/w25443.pdf` (Accedido: 2022-4-09)

Guzman Llambias, R. R., Laura Gonzales. (s.f.). Blockchain interoperability: a feature-based classification framework and challenges ahead. Descargado de `Aunnopublicado` (Accedido: 2023-01-10)

Haugum, T., Hoff, B., Alsadi, M., y Li, J. (2022). Security and privacy challenges in blockchain interoperability - a multivocal literature review. En *Proceedings of the international conference on evaluation and assessment in software engineering 2022* (p. 347–356). New York, NY, USA: Association for Computing Machinery. Descargado de `https://doi.org/10.1145/3530019.3531345` doi: 10.1145/3530019.3531345

Hewett, N. (s.f.).

Hong-Ning Dai, H. W. Z. Z., Xiangping Chen, y Xie, S. (2018). Blockchain challenges and opportunities: a survey. *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, 6-7. Descargado de `https://www.henrylab.net/wp-content/uploads/2017/10/blockchain.pdf` (Accedido: 2023-04-21)

Hyperledger.org. (February 16, 2023). *Interoperability modes.* `https://labs.hyperledger.org/weaver-dlt-interoperability/docs/external/interoperability-modes/#asset-transfer`. (Accedido: 2023-02-16)

Ieee standard computer dictionary: A compilation of ieee standard computer

glossaries. (1991). *IEEE Std 610*, 1-217. doi: 10.1109/IEEESTD.1991.106963

Jean-Jacques Quisquater, M. Q. M. Q. L. G. M. A. G. G. G. A. G. G. G. . S. G., Myriam Quisquater. (1990). How to explain zeroknowledge protocols to your children. En *Advances in cryptology — crypto' 89 proceedings* (p. 628-631). New York, NY: Springer New York.

José Segura. (5 octubre, 2022). *¿qué es la zkevm?* `https://academy.bit2me.com/que-es-la-zkevm/`. (Accessed: 2023-01-22)

Lamport, L., Shostak, R., y Pease, M. (1982, jul). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, *4*(3), 382–401. doi: 10.1145/357172.357176

Llambías, G., Barreiro, J., Villar, P., Toscano, M., Pereira, M., y González, L. (2019). Towards a requirement-driven identification and selection process for blockchain platforms. En *2019 xlv latin american computing conference (clei)* (p. 1-10). doi: 10.1109/CLEI47609.2019.235053

Mathías Castro, E. G. y. S. P. (2023). Interoperabilidad entre plataformas de blockchain. , 109.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, 1-6. Descargado de `https://bitcoin.org/bitcoin.pdf` (Accedido: 2022-11-15)

Ongaro, D., y Ousterhout, J. (2014). In search of an understandable consensus algorithm. En (p. 305–320). USA: USENIX Association.

Polkadot. (s.f.). `https://polkadot.network/`. (Accessed: 2023-3-09)

Ramsés Fernández-Valéncia. (Jul 2, 2021). *Private smart contract execution.* `https://medium.com/iovlabs-innovation-stories/private-smart-contract-execution-7df89e28eb30`. (Accessed: 2023-01-21)

R. Belchior, S. G., A. Vasconcelos, y Correia, M. (s.f.). A survey on blockchain interoperability: Past, present, and future trends. Descargado de `https://dl.acm.org/doi/10.1145/3471140` (Accedido: 2023-05-03)

Rui Zhang, L. L., Rui Xue. (s.f.). Security and privacy on blockchain. Descargado de `https://dl.acm.org/doi/pdf/10.1145/3316481` (Accedido: 2023-3-3)

Schlatt, V., Sedlmeir, J., Traue, J., y Völter, F. (2022, dec). Harmonizing sensitive data exchange and double-spending prevention through blockchain and digital wallets: The case of e-prescription management. *Distrib. Ledger Technol.*. Descargado de `https://doi.org/10.1145/3571509` (Just Accepted) doi: 10.1145/3571509

SerHack, y the Monero Community. (s.f.). Mastering monero. Descargado de `https://masteringmonero.com/book/Mastering%20Monero%20First%20Edition%20by%20SerHack%20and%20Monero%20Community.pdf` (Accedido: 2023-3-3)

S. Ghaemi, R. B. R. S. C. H. K., S. Rouhani, y Musilek, P. (s.f.). A pub-sub architecture to promote blockchain interoperability. Descargado de `https://arxiv.org/pdf/2101.12331.pdf` (Accedido: 2023-05-03)

Solomon, R., y Almashaqbeh, G. (2021). smartfhe: Privacy-preserving smart

contracts from fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, *2021*, 133.

Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P., y Vechev, M. (2019). Zkay: Specifying and enforcing data privacy in smart contracts. En *Proceedings of the 2019 acm sigsac conference on computer and communications security* (p. 1759–1776). New York, NY, USA: Association for Computing Machinery.

Welcome to the ipfs. (s.f.). Descargado de `https://docs.ipfs.tech/` (Accedido: 2022-1-19)

Yang, G., Lee, K., Lee, K., Yoo, Y., Lee, H., y Yoo, C. (2022). Resource analysis of blockchain consensus algorithms in hyperledger fabric. *IEEE Access*, *10*, 74902-74920. doi: 10.1109/ACCESS.2022.3190979

ZCASH. (s.f.). *What are zk-snarks?* `https://z.cash/technology/zksnarks/`. (Accessed: 2022-12-06)

Apéndice A

Mecanismos de consenso

En las siguientes secciones se presentan los algoritmos de consenso más utilizados en la actualidad.

A.1. Proof of Work (PoW)

Una *Proof of Work* (Buterin, s.f.-d) (prueba de trabajo) es un dato difícil de producir pero fácil de verificar para otros. Poder producir este tipo de prueba es un proceso aleatorio con baja probabilidad de éxito, por lo que requiere de mucho esfuerzo de tiempo para poder generar una prueba válida. En las *blockchain* que manejan este tipo de algoritmo como, por ejemplo, *Bitcoin* o *Ethereum*, existen nodos denominados mineros que se especializan en el armado de bloques. Estos reciben un conjunto de transacciones, las validan y en caso de que todas sean válidas, se arma un bloque con todas ellas. Además, se agrega al bloque el *hash* correspondiente al bloque anterior. Para culminar con la creación del bloque, se debe de completar una prueba de trabajo que integre todos los datos del bloque y este pueda ser validado por los demás nodos de la red. Esta prueba de trabajo no es más que una función de *hash* que recibe como parámetros los datos del bloque que se quiere agregar a la cadena y un parámetro extra denominado *nonce*. Este último es el que requiere ser calculado a base de prueba y error para que el resultado de la función de *hash* tenga una cantidad definida de ceros al inicio. Mientras más ceros se exija al inicio del *hash*, más difícil será el cálculo del *nonce*; y, por lo tanto, se precisará una gran cantidad de cómputo para lograrlo.

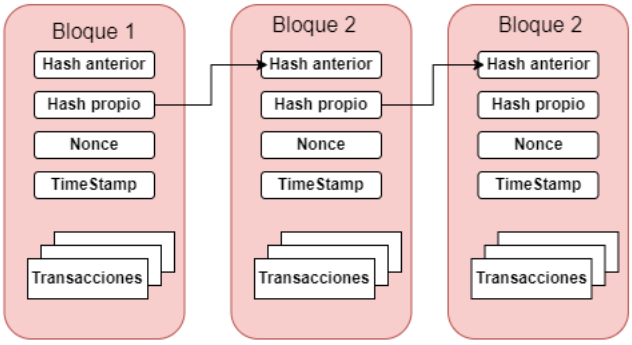


Figura A.1: Cadena de bloques

Una vez logrado el cálculo del *nonce*, se le agrega al bloque y este se distribuye a todos los nodos de la red. Una vez que los nodos reciban al bloque, lo validan y lo agregan a su cadena en caso de que este sea válido como se muestra en la figura A.1.

Ahora bien, ¿qué pasa cuando se generan dos bloques con transacciones distintas y estos son recibidos por la red? Como se muestra en la figura izquierda A.2. En PoW, la red aceptará como cadena válida aquella que sea la más larga (mayor cantidad de bloques) y en caso de que un nodo reciba dos bloques válidos, elegirá de manera arbitraria por uno de ellos y lo agregará a la cadena como se muestra en la figura derecha A.2.

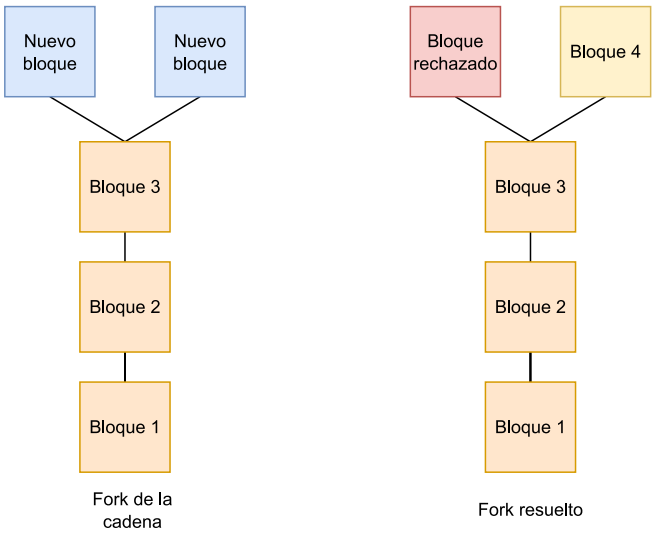


Figura A.2: Fork

Dado este consenso de que la cadena más larga es la cadena válida en la red,

entonces para que un nodo o grupo de nodos pueda tomar el control total de la *blockchain* es necesario que posea por lo menos 51 % del poder de cómputo de la red. Si esto ocurre, estos nodos pueden generar cadenas más largas que contengan solo las transacciones que ellos deseen y descarten las restantes. Este tipo de ataque a una *blockchain* con PoW es denominado Ataque del 51 %.

A.2. Proof of Stake (PoS)

La *Proof of Stake* (Buterin, s.f.-c) (prueba de participación) es un mecanismo alternativo a PoW que fue adoptado por *Ethereum* en una de las últimas actualizaciones de 2022. En lugar de necesitar poder de cómputo para validar un bloque, los validadores deben apostar un bien de valor de la *blockchain*, típicamente la moneda de la misma. De esta manera se logra reducir la cantidad de cómputo y, por lo tanto, la energía necesaria consumida por la red. En las *blockchain* con este algoritmo de consenso, se seleccionan nodos que tienen como cometido validar bloques. Para ello, cada validador tiene que apostar cierta cantidad de monedas (dependiendo de la *blockchain*) para informar a la red de que el bloque es válido. Si alguno de estos nodos llegara a mentir, entonces perdería todos los activos que apostó previamente para ser un nodo validador. Un ataque del 51 % también puede ser fácil de lograr en una *blockchain* de pocos participantes. El ataque del 51 % ocurre cuando un grupo de validadores controla la mayoría de las monedas de la *blockchain* y, por lo tanto, la mayoría de los votos en la validación de bloques. Si este grupo malintencionado llega a controlar la *blockchain*, podría manipularla para su propio beneficio, como invalidar transacciones legítimas o agregar transacciones no válidas.

A.3. Raft

Hyperledger Fabric permite la implementación y uso de distintos protocolos de consenso, pero hace uso de uno en particular por defecto. Este protocolo se conoce como *Raft*(Ongaro y Ousterhout, 2014) y es un protocolo del tipo *Crash Fault Tolerant* o CFT. Este protocolo trabaja en modalidad "líder y seguidor", donde un líder es elegido entre todos los nodos a base de votación, y posteriormente este se encarga de manejar completamente las transacciones entrantes, darles un orden y avisar a los demás nodos cuando es seguro comprometerse a dichas transacciones. En la figura A.3 se muestra la operación del protocolo *Raft* en *Hyperledger Fabric*. Este proceso comienza cuando una transacción es presentada ante un nodo(1), luego este envía la transacción al líder(2). El líder ordena ambas transacciones y crea el bloque (3), y finalmente, envía a los otros nodos para ser replicado(4)(Yang y cols., 2022).

Raft puede continuar funcionando aunque el nodo líder pierda la conexión, garantizando cierta tolerancia a fallos. El protocolo establece que en este caso los demás nodos se colocan como candidatos, y empieza un proceso de votación. Este se repite hasta que un nodo llegue a la mayoría de los votos. Por lo tanto,

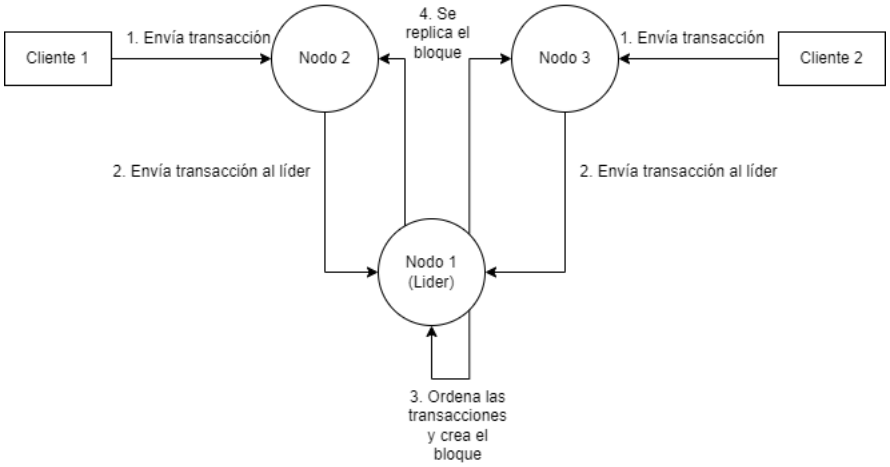


Figura A.3: Diagrama de ejemplo protocolo *Raft*

la tolerancia a fallos se da siempre y cuando más de la mitad de la red continúe activa(Yang y cols., 2022). Otra característica a tener en cuenta es que *Raft* no es resistente a fallos Bizantinos (Lamport, Shostak, y Pease, 1982), este tipo de fallos se dan cuando un nodo de la red falla, pero el resto no pueden constatar si este falló o no. Para ello existen otros protocolos como el Practical Byzantine Fault Tolerance.

A.4. Practical Byzantine Fault Tolerance

El protocolo *practical Byzantine Fault Tolerance* (o por sus siglas en inglés pBFT)(Castro y Liskov, 1999) trabaja seleccionando un nodo líder el cual puede cambiar en cualquier momento, este es elegido por votación entre los nodos. Una vez es seleccionado este nodo, un cliente puede solicitar una transacción, la cual será enviada al líder. Posteriormente, el nodo líder enviará a validar la transacción a cada uno de los demás nodos, esta fase es conocida como *pre-prepare*. A medida que los nodos validen la transacción, enviarán un mensaje a todos los otros nodos, conocido como fase *prepare*. Luego cada nodo nuevamente enviara un mensaje de *commit* a cada otro nodo, si cada nodo recibe $n + 1$ mensajes válidos de *commit*, donde n es la máxima cantidad de nodos maliciosos o fallidos, se valida la transacción. Luego cada nodo enviara un mensaje al cliente, si el mismo recibe $n + 1$ respuestas, esto quiere decir que la transacción se ha concretado correctamente. A continuación, se presenta en la figura A.4 un diagrama de un ejemplo de este algoritmo con cuatro nodos en donde uno de ellos tiene un fallo.

Este tipo de algoritmos generalmente permite soportar el fallo de un tercio de la red, y no $50\,\% + 1$ como en los casos vistos anteriormente, por eso es

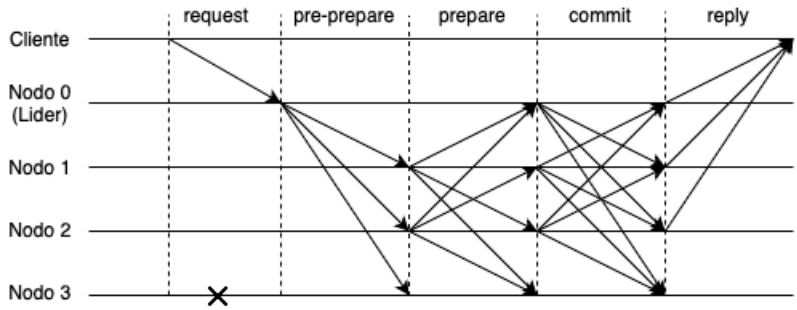


Figura A.4: Diagrama de un Practical Byzantine Fault Tolerance

mucho más común que se utilice en *blockchain* de tipo privada o consorcio.

Apéndice B

Desafíos de interoperabilidad en blockchain

A la hora de integrar dos soluciones *blockchain* se presentan algunos desafíos a ser evaluados y manejados para lograr una interoperabilidad correcta entre estos dos sistemas (Hewett, s.f.).

B.1. Estandarización de datos

Cada ecosistema de *blockchain* define estándares para la estructura y representación de datos de entidades, como contratos y participantes. No obstante, al buscar la interoperabilidad entre distintas plataformas de *blockchain*, pueden surgir desafíos debido a la falta de atributos estandarizados en dichos estándares.

B.2. Mecanismo de consenso

Los diferentes mecanismos de consenso, como la Prueba de Trabajo (*Proof of Work, PoW*) y la Prueba de Participación (*Proof of Stake, PoS*), no son interoperables de forma nativa. Aun así, aunque dos plataformas de *blockchain* utilicen el mismo mecanismo de consenso, puede resultar complicado sincronizar los datos entre ellas y llegar a un consenso sobre el orden de las transacciones.

B.3. Autenticación y autorización

Las *blockchain* pueden admitir transacciones con firmas múltiples, lo que permite que varios participantes firmen digitalmente la misma transacción. Sin embargo, esto se implementa de manera diferente en distintas plataformas de

blockchain. La autenticación y la autorización no son interoperables en algunas *blockchain* a pesar de tener mecanismos de consenso similares. En consecuencia, los métodos de interoperabilidad deben basarse en mecanismos de autenticación cruzada. Estos mecanismos pueden variar desde el almacenamiento simple de contraseñas encriptadas hasta la superposición de una autenticación de usuario en las plataformas de *blockchain*.

Apéndice C

Soluciones actuales en la industria

A continuación, se da una descripción de las soluciones que existen hoy en día para poder interoperar dos *blockchain*. (Guzman Llambias, s.f.).

C.1. Notary Scheme

Una de las maneras más conocidas al día de hoy de interoperar varias *blockchain* es por medio de un *Notary Scheme*, método que se utiliza principalmente para el intercambio o transferencia de criptomonedas. La idea central de este método involucra la participación de un tercero que oficia de coordinador entre las *blockchain*. Su labor consta de monitorear constantemente las múltiples *blockchain* y realiza transacciones en ellas ante el desencadenamiento de un evento específico, como se muestra en la figura C.1. Por ejemplo, dado la ejecución de un *smart contract* en la *blockchain* A, el *Notary* podrá ejecutar una transacción en la *blockchain* B.

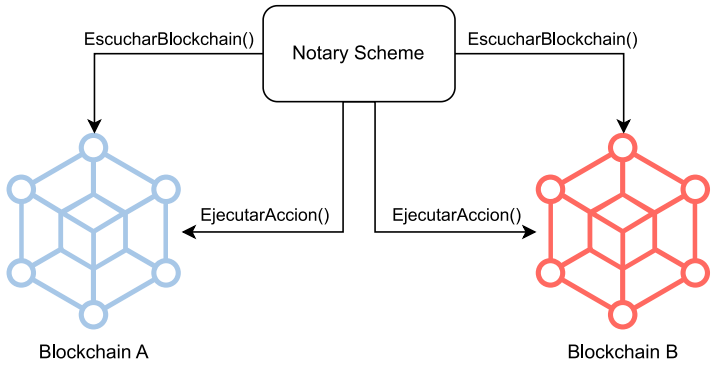


Figura C.1: Notary Scheme

En la práctica, esta entidad puede operar en los intercambios de dos maneras distintas, de manera centralizada (CEX) o descentralizada (DEX). Los *Notary* centralizados CEX son los más populares al día de hoy. Un ejemplo de esto es la plataforma de *Binance*. En este tipo de funcionamiento, el *Notary* maneja las claves privadas de los usuarios y ejecuta transacciones en su nombre. Por otro lado, los *Notary* descentralizados DEX permiten el intercambio de criptomonedas sin la necesidad de un tercero. En este caso, los usuarios conservan sus propias claves privadas e intercambian sus actvivos mediante *Atomic Swaps* o soluciones de *smart contract* y el rol del tercero en discordia es el de mantener un registro de las transacciones entre las partes. Los dos protocolos DEX más conocidos son *0x* y *Uniswap*, ambos basados en soluciones de *smart contract*.

C.2. Atomic Swaps

Atomic Swap surgió como una alternativa a los intercambios centralizados, ya que permiten operaciones atómicas entre *blockchain* utilizando los conceptos de *Hashlock* y *Timelocks* por medio de la ejecución de un *smart contract* denominados *Hashed Time Lock Contracts* (HTLC). Un *Hashlock* es un mecanismo que garantiza que una transacción solo se puede desbloquear si se proporciona una clave que coincida con un valor *hash* previamente acordado. En el contexto de un *Atomic Swap*, cada parte bloquea sus activos en una dirección única de la *blockchain* utilizando un *Hashlock*. Esto significa que el otro participante no puede acceder a los activos bloqueados sin proporcionar el valor de la clave correcta. Por otro lado, un *Timelock* es un mecanismo que permite que una transacción se desbloquee solo después de que se cumpla un cierto período de tiempo. En el contexto de un *Atomic Swap*, se utilizan *Timelocks* para asegurarse de que cada parte solo pueda acceder a los activos bloqueados después de un tiempo acordado. Esto asegura que, si alguna de las partes no cumple con su parte del contrato, la otra parte pueda recuperar sus activos después de un tiempo determinado. En conjunto, el uso de *Hashlocks* y *Timelocks* garantiza que un *Atomic Swap* sea seguro y confiable, ya que las transacciones solo se pueden desbloquear si se cumplen ciertas condiciones previamente acordadas por ambas partes. Además, estos mecanismos permiten que el intercambio se realice sin necesidad de confiar en terceros y sin riesgo de que los activos se pierdan o se desvíen.

Si bien hay muchos protocolos disponibles para el funcionamiento de un *Atomic Swap* a continuación, se presentan los pasos para que Alice y Bob puedan intercambiar activos entre *Bitcoin* y *Ethereum* utilizando un *smart contract* HTLC.

1. Alice y Bob acuerdan intercambiar sus activos y establecen los detalles del intercambio

2. Alice crea un *smart contract* en la *blockchain* de *Bitcoin* y bloquea sus 2 BTC en el contrato utilizando el *hashlock* generado a partir del secreto S

que ella eligió y definiendo un *timelock* especifico.

3. Alice le da a Bob su valor de *hash* obtenido al realizar el *hashlock*, la función de *hash*, el *timelock* definido y los detalles de la publicación del contrato. Con esto, Bob verifica que el *smart contract* de Alice este publicado en la *blockchain* de *Bitcoin*.

4. Bob crea un*smart contract* en la *blockchain* de *Ethereum* y bloquea sus 28 ETH en el contrato utilizando el *hashlock* generado a partir del secreto X que él eligió y el *timelock* de Alice.

5. Bob le da a Alice su valor de *hash* obtenido al realizar el *hashlock*, la función de *hash* y los detalles de la publicación del contrato. Con esto, Alice verifica que el *smart contract* de Bob este publicado en la *blockchain* de *Ethereum*.

6. Bob le envía el valor de su secreto X correspondiente al *hashlock* de sus fondos en *Ethereum* a Alice por medio de un canal privado

7. Si el valor obtenido al aplicar la función de *hash* al secreto X proporcionado por Bob coincide con el valor *hash* del *hashlock* utilizado para bloquear sus fondos, el *hashlock* se desbloquea y los 32 ETH de *Ethereum* bloqueados en la dirección única de Bob se envían a la dirección de *Ethereum* de Alice luego de cumplido el tiempo del *timelock*.

8. Luego, Alice le envía el valor de su secreto S correspondiente al *hashlock* de sus fondos en *Bitcoin* a Bob y este lo utiliza para acceder a los BTC de Alice luego del cumplimiento del tiempo del *timelock* en caso de que coincida con el *hash* del *hashlock* anteriormente revelado a Bob.

9. Si Bob o Alice no proporcionan el valor secreto correspondiente dentro del límite de tiempo establecido por los *Timelocks* o proporcionan un secreto falso, el HTLC se anula y ambos recuperan sus fondos.

C.3. Sidechain/Relays

La solución de *sidechain/relay* es un mecanismo para interoperar, escalar y/o actualizar; que consiste en la existencia de dos *blockchain*. Una se considera la *main chain* la cual mantiene el *ledger* principal y la otra se considera la *sidechain* la cual es una extensión de la *main chain*, pero que maneja su propio *ledger* y algoritmo de consenso. La comunicación entre ambas puede ser unidireccional, de la *main chian* a la *sidechain* o también puede ser bidireccional, es decir, que ambas *blockchain* se comuniquen entre sí, como se muestra en la figura C.2. La comunicación entre ambas *blockchain* requiere de dos componentes. Por un lado, se requiere de un protocolo de comunicación que permita la correcta transferencia de mensajes entre ambas *blockchain* por medio de conectores y adaptadores de mensajes. Por otro lado, se necesita de la implementación de un *Relay*, el cual es una pieza de software alojada dentro de una *blockchain* que

permite verificar las transacciones recibidas por agentes externos. En este caso, la *sidechain* utilizaría este componente para realizar verificaciones de la información obtenida de la *main chain*. Una de las utilidades de esta implementación puede ser la aplicación de un canal de pagos, donde los usuarios interactúan en la *sidechain* y cuando terminan sus operaciones se publican los resultados en la *main chain* logrando así mayores niveles de eficiencia. También puede ser usada para agregar funcionalidad a una *blockchain*, como es el caso de Bitcoin que utiliza una *sidechain* para implementar *smart contract*, ya que no soporta esta funcionalidad de manera nativa.

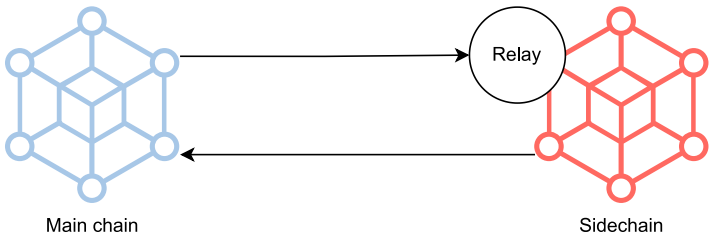


Figura C.2: Sidechain/Relay

C.4. Blockchain of blockchains

La solución de *blockchain of blockchains* es de las soluciones menos estudiadas a nivel teórico y su definición no está del todo clara. Se basa principalmente en soluciones aplicadas por la industria como son los casos de *Polkadot* (*Polkadot*, s.f.) y Cosmos (*Cosmos*, s.f.) más que en investigaciones académicas. En (R. Belchior y Correia, s.f.) se describe esta solución como un marco que proporciona características específicas para interoperar *blockchain*. La idea de esta solución es la de tener un protocolo de consenso como los vistos en el anexo A que organice y valide los bloques que contienen un conjunto de transacciones pertenecientes a las *blockchain* que se interconectan y además se pueda proporcionar una visión actualizada de cada una de las *blockchain* interconectadas. Las implementaciones suelen ser similares a las de *sidechain/relay*, ya que suele haber una cadena principal denominada *relay chain*, la cual se encarga de conectar las *blockchain* secundarias denominadas *parachains* (este nombre es propio de *Polkadot*). Las *parachains* pueden ser personalizadas y optimizadas para diferentes casos de uso y ofrecen un alto nivel de interoperabilidad con otras *blockchain*. En (Guzman Llambias, s.f.) se enumeran los bloques de construcción que debería tener este tipo de soluciones y son similares a los ya explicados en la sección anterior C.3. Estos son: un *relay* para recibir las transacciones de *blockchain* de origen, *validadores* que puedan validar las transacciones entrantes según un protocolo de consenso y por último, *Connectors* y *Routers* que puedan enviar las transacciones a la *blockchain* destino.

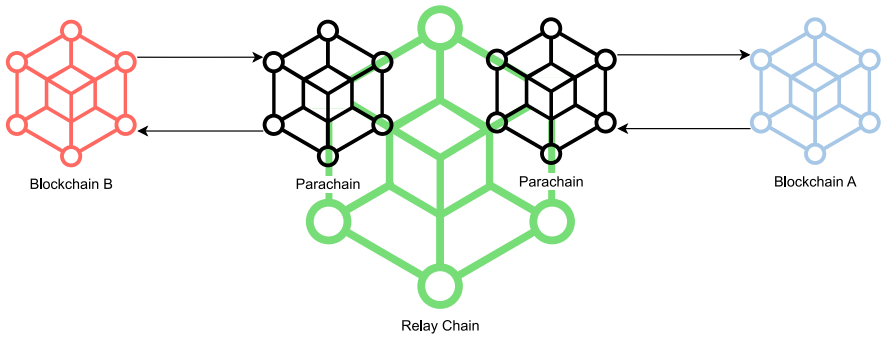


Figura C.3: Blockchain of blockchains

C.5. Enterprise connectors

Según la definición expuesta en (Guzman Llambias, s.f.), los *Enterprise connectors* son una evolución de la categoría *Hybrid connectors* presentada en (R. Belchior y Correia, s.f.). La razón de dicha evolución es que esta categoría se aplica más a empresas donde existe cierto grado de confianza. También se definen nuevas subcategorías para los *Enterprise Connectors* tales como *Gateway*, *Generic interface* y *Enterprise relays*. Los *Enterprise Connectors* tienen muchas similitudes con los *Notary Scheme*. En general, sus propiedades, arquitectura y características son similares, como se puede observar en la descripción de las subcategorías a continuación, y en la figura C.4.

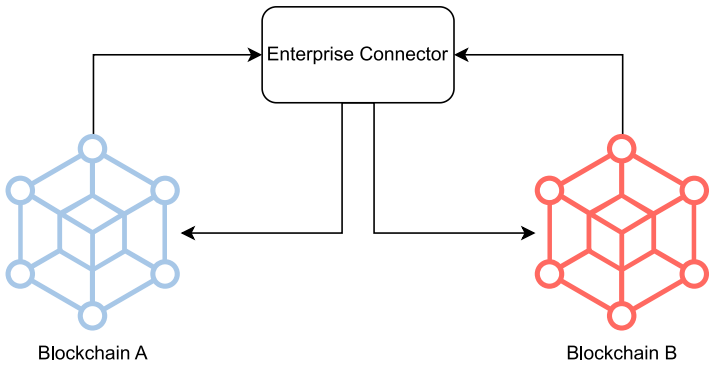


Figura C.4: Enterprise connector

Gateway: Bajo el contexto de interoperabilidad de *blockchain*, un *Gateway* es un *software* que actúa como intermediario entre dos *blockchain*, una de origen y otra de destino. El objetivo de este *software* es manipular los mensajes enviados por la *blockchain* de origen y transformarlos de modo que estos sean comprensibles y manipulables por la *blockchain* de destino. Para lograr este objetivo, el

Gateway se implementa utilizando conectores (*Connectors*) y adaptadores de mensajes (*Message adapters*). Los conectores son las partes del *Gateway* que actúan como *middleware* proporcionando un protocolo de comunicación entre las *blockchain* que se van a interoperar. Siendo estos los puntos de entrada y salida del *Gateway*. Por otro lado, el adaptador de mensaje es lo que brinda la capacidad de transformación de los datos. Esta solución es la más básica en los *Enterprise connector* por lo que puede evolucionar a una solución de *Generic interface* o *Enterprise Relays*.

Generic interface: Las *Generic interface* permiten a un sistema de origen comunicarse con múltiples *blockchain* utilizando una API normalizada provista por un tercero. Estos sistemas de origen pueden ser un sistema de aplicación o una *blockchain*. Es el sistema de origen el responsable de aplicar las transformaciones necesarias sobre los datos, de modo que estos sean compatibles con el formato de la API y por ende con el sistema destino. Al igual que en el caso del *Gateway*, la solución de *Generic interface* utiliza conectores y adaptadores de mensajes para transformar los datos recibidos y aplicar el protocolo de comunicación correspondiente según la *blockchain* de destino. En caso de que la operación por parte del sistema de origen requiera de una respuesta, es el *software* intermediario quien se encarga de transformar los datos y redirigir el mensaje de respuesta. Esta solución, a diferencia de la solución de *Gateway* permite la interoperabilidad con múltiples *blockchain*, ya que a diferencia del *Gateway* esta solución posee características de registro para descubrir nuevas plataformas *blockchain* con las que interoperar y, por lo tanto, posee funciones de enrutamiento.

Enterprise Relays: Los *Enteprise Relays* como se definen en (Guzman Llambias, s.f.), son una evolución conceptual de los *Trusted Relays*, ya que también abarcan soluciones descentralizadas y menos fiables como (S. Ghaemi y Musilek, s.f.). Los *Trusted Relays*, definidos como una subcategoría de los *Hybrid Connectors* en (R. Belchior y Correia, s.f.), son piezas de *software* centralizadas que se encargan de redirigir las transacciones de una *blockchain* de origen a una de destino. Permitiendo así hacer uso de la información de una *blockchain* en otra, siempre y cuando se esté trabajando sobre entornos permisionados. Para esto se utilizan mecanismos de adaptación de mensajes, conectores, enrutamiento y registro, al igual que las soluciones anteriores. Además, utiliza mecanismos de retransmisión de la información y nodos validadores en la *blockchain* de destino que se encargan de validar esas transacciones utilizando el algoritmo de consenso de la *blockchain* de origen.

Apéndice D

IPFS

IPFS ("Welcome to the IPFS", s.f.), o InterPlanetary File System, es una tecnología de almacenamiento y distribución de archivos que utiliza una red de igual a igual (peer-to-peer). En lugar de almacenar archivos en un solo servidor centralizado, IPFS utiliza una red distribuida de nodos para almacenar fragmentos de archivos y compartirlos entre los usuarios de la red. Esto permite que los archivos se almacenen de forma más eficiente y segura, sin depender de un único punto de falla.

IPFS utiliza una tecnología llamada *Content Addressing* para identificar de forma única cada archivo almacenado en la red. Cada archivo se divide en fragmentos, y cada fragmento se identifica mediante su *hash* criptográfico. Esto significa que cada vez que se solicita un archivo en la red, se puede buscar en la red a través de su *hash*, lo que hace que el proceso de búsqueda y descarga de archivos sea más rápido y eficiente. Además de ser utilizado para compartir y almacenar archivos, IPFS también se puede utilizar para alojar aplicaciones web descentralizadas. Estas aplicaciones se ejecutan completamente en la red IPFS y no requieren un servidor centralizado para funcionar. Esto aumenta la eficiencia y la seguridad de la aplicación al no depender de servidores centralizados.

IPFS utiliza una red descentralizada de nodos para almacenar y compartir archivos. Los nodos de la red pueden ser, desde usuarios regulares hasta organizaciones o empresas que deseen alojar contenido en la red IPFS. La red utiliza una tecnología llamada *Swarm Routing* para enrutar los datos de manera eficiente a través de la red. Esto significa que los datos se distribuyen automáticamente entre los nodos de la red, lo que garantiza que siempre haya múltiples copias disponibles.

Apéndice E

Análisis detallado de privacidad en soluciones de interoperabilidad pública-privada

En este anexo se detalla el análisis de la sección 4.2. Dicho análisis estudia solución a solución los requisitos de privacidad al ejecutar cada una de las posibles operaciones al interoperar: *Transfer*, *Exchange*, *Data Sharing*. Como no existen implementaciones actuales de soluciones de tipo *Blockchain of Blockchains* que permitan interoperar una *blockchain* publica con una *blockchain* privada, dicha solución quedo fuera del estudio realizado.

E.1. Notary Scheme

En este tipo de solución, tanto la conservación de las características de privacidad como el cumplimiento de los requisitos de privacidad al interoperar, dependen del *Notary*. De ser un agente malicioso, podrá exponer información privada a terceros no participantes de la transacción entre las *blockchain*. Sin embargo, dado que es común en el ecosistema *blockchain* encontrar soluciones de esta índole que funcionan y son aceptadas por los usuarios, se podría asumir que el *Notary* es confiable. Bajo esta hipótesis se conservarían algunas características de privacidad.

- **Privacidad de la identidad**, en las operaciones de tipo ***Transfer***, la privacidad de la identidad depende de que el *Notary* no revele la trazabilidad completa de la operación. Si lo hiciera, un tercero que tenga permisos de lectura en la *blockchain* privada podría ver la traza de la operación desde el origen en la *blockchain* privada, hasta el destino en la *blockchain*

pública, identificando de esta forma al emisor y al receptor de la transacción.

Por otro lado, los intercambios de activos en las operaciones de tipo ***Exchange***, expondrán la identidad de los involucrados en la transacción que ocurre sobre la *blockchain* pública. Un tercero cualquiera podría ver el origen de la transacción, seguirla hasta el *Notary* y como este último expone la traza (suponiendo de que es malicioso), el usuario podrá seguir la transacción hasta el destinatario. De este modo quedaron expuestos los dos participantes en la operación quebrantando el requisito de privacidad.

Por último, para la operación de tipo ***Data Sharing*** si se consulta información desde la *blockchain* pública sobre la privada, se expondrá la identidad de los participantes, quebrando el requisito de privacidad. Esto se debe a que la consulta pasa por el *Notary*, y si este último es un agente malicioso, podría difundir la identidad del usuario que realiza la consulta. También podría ocurrir que el *Notary* utilicé las credenciales propias del usuario para autenticarse en la *blockchain* privada. En dicho caso, aunque el *Notary* no fuera malicioso, el acceso a la *blockchain* quedara registrado y la identidad del usuario quedara expuesta.

- **Privacidad del mensaje**, sin importar si el *Notary* es malicioso o no, el requisito de privacidad del mensaje no se va a cumplir para las operaciones de ***Transfer*** y ***Exchange***. Esto se debe a que por defecto el *Notary* no ofusca/encripta la información que se envía a través de la *blockchain* pública, lo que permite que cualquiera vea el contenido del mensaje. En el caso de las operaciones de tipo ***Data Sharing*** se quebrantará el requisito de privacidad si y solo si el *Notary* es malicioso y difunde públicamente el contenido del mensaje.

- **Privacidad del canal**, dado que una parte de la transacción entre las *blockchain* ocurre sobre la *blockchain* pública, algún usuario malicioso podría estar registrando las direcciones IP que acceden a dicha *blockchain*, quebrantando así este requisito de privacidad. Por otro lado, si el *Notary* es un actor malicioso, podría ser el quién lleve un registro de las direcciones IP de los usuarios que realizan solicitudes al mismo y vulnerar así el requisito de privacidad.

E.2. Atomic Swap

Se ha realizado el análisis de privacidad en esta solución de interoperabilidad solo para las operaciones de tipo ***Exchange***. Esto se debe a que es el único tipo de operación que soporta esta solución.

- **Privacidad de la identidad**: en las operaciones de tipo ***Exchange*** no se mantendrá la privacidad de la identidad. Aunque las transacciones se

realicen de forma independiente en cada una de las *blockchain*, la transacción de la *blockchain* pública expondrá las identidades de los participantes en la misma.

- **Privacidad del mensaje**: en este caso solo es posible mantener la privacidad del mensaje en el intercambio que se realiza sobre la *blockchain* privada. Ya que la transacción es independiente a la *blockchain* pública. Sin embargo, en la transacción pública el contenido del mensaje podrá ser visto por cualquier usuario. Por tal motivo, el requisito de privacidad del mensaje no sé satisface.
- **Privacidad del canal**: en las operaciones de tipo ***Exchange*** no se mantendrá esta característica. Puesto que en la *blockchain* pública se podrían estar registrando/≪*logueando*≫[1] las direcciones IP de los usuarios que realizan solicitudes al *smart contract* ≪*deployado*≫[2].

E.3. Sidechain / Relay

Según la definición de la sección C.3 este tipo de solución es extremadamente versátil. Además, dado que su implementación se realiza pensando en la interoperabilidad con la *Mainchain*, esta puede conseguir satisfacer los requisitos de privacidad si se lo propone. Sin embargo, el estudio de esta solución no tendrá en cuenta las implementaciones que se desvíen de las características básicas de la definición.

- **Privacidad de la identidad**: este tipo de solución no preserva la privacidad de la identidad. Ya que para realizar las operaciones de ***Transfer*** y ***Data Sharing*** es necesario que el usuario de la *blockchain* pública se autentique en la *blockchain* privada. De esta forma su identidad quedará expuesta. En el caso de las operaciones de tipo ***Exchange*** donde se realizan intercambios de forma separada en cada una de las *blokchain*. Las transferencias realizadas (intercambio de activos) en la *blockchain* pública, expondrán las identidades de los participantes.
- **Privacidad del mensaje**: La definición de *Sidechain* no contempla ningún tipo de ofuscación sobre las transacciones, ni de la información que se almacena de las mismas. Por lo tanto, se puede afirmar que la solución no cumple con el requisito de privacidad del mensaje.
- **Privacidad del canal**: Similar a los casos anteriores, si existe un agente malicioso que esté registrando las conexiones a cualquiera de las *blockchain*, podría rastrear la dirección IP del usuario que realiza la consulta y exponer su identidad. Por lo tanto, no se cumple con el requisito de privacidad del canal.

[1] La palabra *logueando* no existe en el español, pero el término es altamente utilizado en la jerga informática con el significado de anotar o guardar una lista de elementos o acciones

[2] La palabra deployado no existe en el español, pero el término es altamente utilizado en la jerga informática con el significado de desplegar un sistema en un entorno de ejecución

E.4. Enterprise connectors: Gateway, Generic Interface, Enterprise Relay

Los *Enterprise connectors* fueron estudiados en conjunto debido a su similitud con la solución de *Notary Scheme*. La diferencia entre los mismos es la capacidad para realizar acciones que involucren lógica de negocio. Por lo tanto, a efectos del análisis realizado, la diferencia entre los mismos no cambia sus capacidades para satisfacer los requisitos de privacidad.

- **Privacidad de la identidad**: En el caso de las operaciones ***Transfer*** se da una situación similar a la del ***Notary Scheme*** dado que la privacidad se va a mantener si el *Enterprise Connector* no expone la identidad de los participantes. Para esto debe de contar con algún tipo de ofuscación sobre la identidad de los usuarios participantes en la operación. Además, este tipo de solución va a depender de la confianza que tengan las partes en el *Enterprise connector*, ya que este va a conocer la identidad de los participantes y podría exponerlas en un futuro. Para las otras operaciones no se va a mantener la privacidad, en el caso de las operaciones ***Exchange*** la información completa queda dentro de cada *blockchain* y en consecuencia en la pública se podría trazar dicha información. En el caso del ***Data Sharing*** al igual que en escenarios anteriores, el usuario de la *blockchain* pública va a tener que autenticarse en la *blockchain* privada para poder consultar información exponiendo así su identidad.

- **Privacidad del mensaje**: Mismo caso que la solución de *Notary Scheme*. Si el *Enterprise connector* que media en la operación no agrega un método de ofuscación al mensaje, la información del mismo será pública. Por lo tanto, no se cumplirá con el requisito de privacidad. A su vez, vale destacar que si el método de ofuscación lo agrega el conector, las partes deberán de confiar en que este no exponga dicha información confidencial en un futuro.

- **Privacidad del canal**: Mismo caso que la solución de *Notary Scheme*

Apéndice F

Análisis detallado de privacidad en soluciones de interoperabilidad público-público

En este anexo se abordará el estudio realizado sobre la privacidad en soluciones de interoperabilidad entre dos *blockchain* públicas. Para este estudio se tomaron como referencia *blockchain* públicas como *Bitcoin* o *Ethereum* que no implementan técnicas de privacidad como podrían ser *Monero* o *Zcash*.

En todos las soluciones de interoperabilidad para el caso de privacidad de la identidad, las operaciones de *Data Sharing* van a mantener la privacidad de la identidad. Por un lado, debido a que en este tipo de operaciones solo se consulta información y dichas consultas no dejan rastros en el *ledger*. Por otro lado, los usuarios no requieren de autenticarse en las *blockchain* públicas para consultar información, evitando así revelar su identidad y, por lo tanto, el requisito de privacidad no se va a quebrantar. Para el resto de operaciones, la tabla F.1 muestra los resultados del análisis realizado para la privacidad de la identidad sobre las soluciones de interoperabilidad estudiadas.

En el caso de la privacidad del mensaje al interoperar dos *blockchain* públicas, el requisito va a ser quebrantado. Esto se debe a que las *blockchain* interoperadas son públicas y que en estas los mensajes son públicos. Con respecto a la privacidad del canal, en el caso del *Notary Scheme* y de los *Enterprise Connectors*, va a depender de la confianza que se tenga en el intermediario (el *Notary* o el *Connector*). Si el mismo es un agente malicioso, la privacidad del canal se va a quebrantar dado que este podría registrar las direcciones IP y así descubrir la identidad de los participantes en la transacción. En el resto de soluciones, la privacidad del canal no se va a mantener, ya que un usuario podría estar registrando las conexiones a cualquiera de las *blockchain* y guardando las direcciones

Solución	Requisitos de privacidad satisfechos	Requisitos de privacidad no satisfechos.
Notary Scheme	En el caso de una operación *Transfer* se va a mantener la privacidad siempre que el *Notary* no difunda la información de la traza completa, ya que es el único que la conoce. Sin embargo, si se trata de un token no fungible, dado que estos son únicos en ambas redes un usuario podría hacer uso de dicha característica para descifrar la traza completa.	Para el caso de la operación *Exchange* como los *assets* sé intercambian en cada *blockchain* respectivamente, la traza total queda registrada en cada una de estas y, por lo tanto, la privacidad de la identidad no se mantiene.
Atomic Swap	La solución no satisface ningún requisito de privacidad.	Como la operación *Transfer* no es soportada por *Atomic Swap* solo se contemplará el caso de la operación *Exchange* y para la misma no se mantiene la privacidad de la identidad, dado que los contratos serían públicos por las características de las *blockchain* y, por lo tanto, cualquiera podría ver la información asociada a los mismos pudiendo así trazar toda la información.
Sidechain/ Relay	La solución no satisface ningún requisito de privacidad.	Tanto para la operación de *Transfer* como para la de *Exchange* no se va a mantener la privacidad de la identidad, dado que las *Sidechain* básicas mantienen la trazabilidad y cualquier usuario puede consultarla.
Blockchain of blockchains	La solución no satisface ningún requisito de privacidad.	A la fecha de realización del proyecto, las pocas soluciones implementadas basadas en *blockchain of blockchains*, todas mantienen la trazabilidad de los elementos y la misma puede ser consultada por cualquier usuario, por lo tanto, no se mantiene la privacidad de la identidad.
Gateway/ Generic Interface/ Enterprise Relay	El analisis para los *Enterpise Connectors* es similar al del *Notary*. Cuando el conector es de confianza y no es un agente malicioso, se satisface el requisito de privacidad de la identidad para las operaciones *Transfer*.	Al igual que en el caso del *Notary Scheme*, en las operaciones de tipo *Exchange* no se satisface el requisito de privacidad de la identidad.

Cuadro F.1: Tabla con el análisis de privacidad realizado sobre las soluciones existentes de interoperabilidad entre dos *blockchain* publicas
.

IP, pudiendo eventualmente rastrear la dirección física de los participantes de la transacción.

Apéndice G

Análisis de privacidad en soluciones de interoperabilidad privado-privado

En este anexo se abordará el estudio realizado sobre la privacidad en soluciones de interoperabilidad entre dos *blockchain* privadas. Como no existen implementaciones actuales de soluciones de tipo *Blockchain of Blockchains* que permitan interoperar dos *blockchain* privadas, dicha solución quedo fuera del estudio realizado.

En primer lugar, para todas las soluciones de interoperabilidad analizadas, la privacidad de la identidad no se va a preservar al interoperar. Esto se debe a que en las *blockchain* privadas los usuarios deben de estar debidamente identificados para poder operarlas. Tampoco se va a preservar la privacidad del mensaje al interoperar las *blockchain* privadas. Esto se debe a que el agente privado que «*hostea*»[1] cualquiera de las *blockchain* a interoperar, podría estar registrando las direcciones IP de los usuarios.

Sin embargo, la privacidad del mensaje sí se va a preservar al interoperar las *blockchain* siempre que se mantenga el mismo mecanismo de cifrado/descifrado entre ambas *blockchain*. En caso contrario, no se va a preservar la privacidad del mensaje.

[1] El verbo hostear no existe en el español, pero el término es altamente utilizado en la jerga informática con el significado de alojar un sistema y mantenerlo en una ubicación dada.

Apéndice H

Propuestas alternativas para resolver la privacidad del mensaje

En la sección 4.5.2 se encuentra un resumen de los abordajes alternativos para solucionar la privacidad del mensaje y al final se detalla el abordaje utilizado. En este anexo se desarrollan con mayor detalle las propuestas alternativas descartadas.

H.1. Primer abordaje: Encriptación Asimétrica

Como primer abordaje se decidió utilizar encriptación asimétrica con el *gateway*. De esta forma, la dApp de *Ethereum* que quisiera llamar una función de *Fabric* a través del *gateway* debería encriptar los datos de entrada con la clave pública del *gateway*. Esto considerando que el *gateway* sea ≪*hosteado*≫ por la misma organización que *Fabric*. En caso contrario, *Fabric* es el que debería poseer la clave pública.

El *gateway* utilizaría su clave privada para desencriptar los datos. Como el *gateway* es ≪*hosteado*≫ por la misma organización que *Fabric* los datos entre estos podrían transmitirse sin encriptar. De esta forma, todo registro que quede almacenado o deje rastros en *Ethereum* estaría encriptado. Esto se puede observar en el diagrama de la figura H.1.

Este abordaje de encriptación asimétrica efectivamente soluciona el problema de privacidad, y así como fue utilizado para los datos de entrada, podría utilizarse de forma similar para encriptar los elementos guardados en el *ledger* de *Ethereum*. Sin embargo, este método tiene un punto a considerar y es que todo lo que quede guardado en *Ethereum* es inmutable, es decir, con el paso del tiempo esa información no va a poder ser eliminada ni modificada. Por lo tanto, eventualmente en el futuro podría suceder que la encriptación sea quebrantada

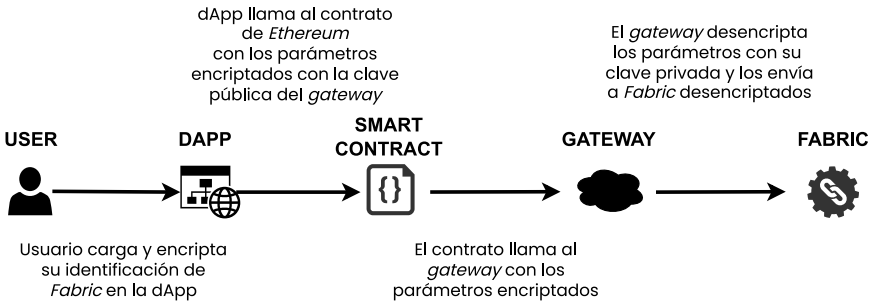


Figura H.1: Envío de parámetros respetando privacidad del mensaje

y un agente malicioso acceda a los datos privados. Es por tal motivo que este método fue descartado como posible solución.

Observar que si solo se encriptasen los datos de entrada, no sería del todo un problema si la información almacenada se hiciera pública en un futuro al romperse la encriptación. Esto se debe a que el procesamiento de dichos datos se realiza en *Fabric*. Si en un futuro lejano se descubrieran los parámetros de entrada, teniendo algunos recaudos en los métodos de autentificación, el usuario malicioso se vería imposibilitado de procesar dichos datos al no poder iniciar sesión en *Fabric*.

H.2. Segundo abordaje: IPFS

Encriptar los datos a guardar en el *ledger* no era una opción válida por lo visto anteriormente. Por lo tanto, el segundo abordaje tuvo como punto de partida relajar el problema. Aceptando encriptar los datos de entrada, pero teniendo que buscar una alternativa al registro de datos en el *ledger* de *Ethereum*. Para ello se decidió utilizar la tecnología *InterPlanetary File System* (IPFS)D. Esta permite almacenar cualquier tipo de archivos de forma descentralizada y ubicarlos en la red a través de un identificador único llamado CID. Para poder utilizar esta tecnología, el equipo construyo un protocolo a su alrededor. El objetivo era que esta se pueda utilizar para almacenar y recuperar información de forma privada y descentralizada. El protocolo consistía en los pasos enumerados a continuación, que también se pueden apreciar en la figura H.2:

- Paso 0. Luego de que *Ethereum* solicitará información de *Fabric* a través del *gateway*. El protocolo se ejecutaría una vez que esta información fuera generada por *Fabric* y enviada al *gateway* con la intención de ser almacenada en *Ethereum*.
- Paso 1. Encriptar la información que se desea almacenar.
- Paso 2. Dividir el documento encriptado en partes.

- Paso 3. Almacenar cada parte en IPFS obteniendo un CID por cada parte almacenada.
- Paso 4. Listar todos los CID en un archivo que se llamará llave.
- Paso 5. ≪*Hashear*≫ la llave y guardar el *hash* en *Ethereum*.
- Paso 6. Otorgar la llave al usuario.

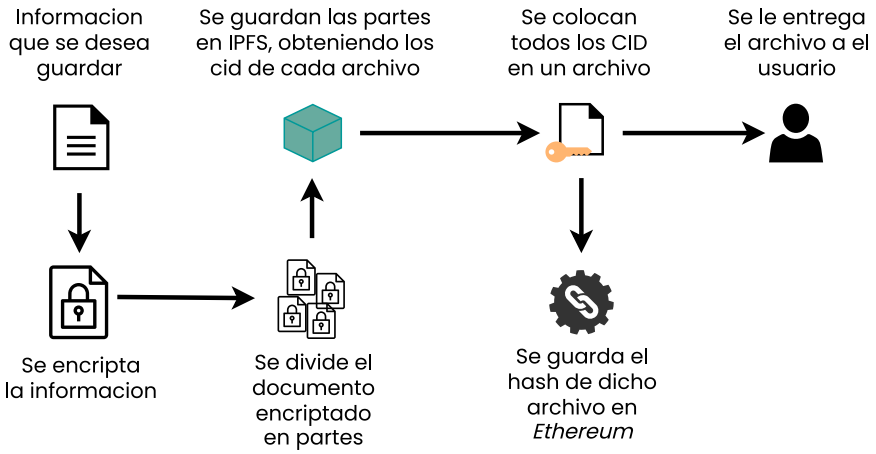


Figura H.2: Procedimiento para guardar archivos en IPFS

De esta forma, si el usuario quisiera recuperar la información almacenada, tendría que seguir los siguientes pasos. Primero, con su llave (la lista de CID) buscaría las partes del archivo almacenado en IPFS. Luego uniría todas las partes en un solo documento, y por último lo desencriptaría, recuperando así la información original. Además, como medida adicional de seguridad, la llave presentada por un usuario puede ser ≪*hasheada*≫ y contrastada con el *hash* almacenado en *Ethereum*. Corroborando así que la información que es recuperada se corresponde con el resultado procesado por *Fabric*. En la figura H.3 se puede observar el protocolo paso a paso.

Este enfoque utilizando IPFS cumple con el objetivo deseado: almacenar información de forma descentralizada, sin necesidad de encriptarla en *Ethereum*. Además, tiene la ventaja de que la información es dispersada en partes a través de la red de IPFS. Por lo tanto, sin la llave es casi imposible recuperar el archivo original. Por otro lado, gracias al *hash* almacenado en *Ethereum* es posible validar que la llave presentada por el usuario es la asociada al resultado.

Sin embargo, la solución presenta varias desventajas. En primer lugar, es necesario pagar una cantidad de dinero cada cierto tiempo para que los archivos almacenados en IPFS no sean removidos de la red. A su vez, el no pagar no garantiza que los archivos sean borrados. En otras palabras, utilizar la red de IPFS provoca la pérdida del control sobre los datos, ya que no hay una heurística o serie de reglas que garanticen como estos van a ser manejados.

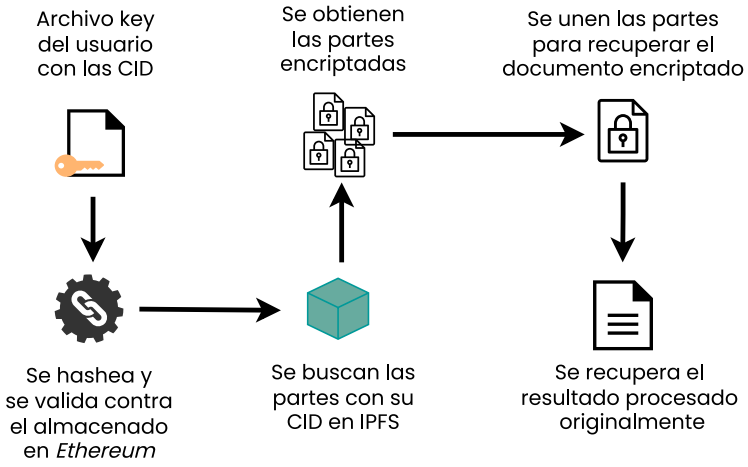


Figura H.3: Procedimiento para recuperar archivos en IPFS

Finalmente, la solución fue descartada, debido a la dificultad técnica de implementar los protocolos de almacenamiento y recuperación de información en el *gateway*. En particular, no fue posible resolver el envío de la llave con los CID al usuario desde el *gateway* de forma descentralizada. Por otro lado, otorgarle tantas responsabilidades al *gateway*, sumado al costo de usar IPFS, comprometía la descentralización de la solución global.

H.3. Tercer abordaje: Private Smart Contract y zkEVM

Las *zkEVM* 2.2.5 y los *Private Smart Contract* 2.2.4 son soluciones que prometen resolver los problemas de privacidad con un enfoque más general y homogéneo. Es decir, desde que la transacción se genera, pasando por el procesamiento y validación en los nodos de la red, hasta el guardado y la completitud de la operación.

Estas herramientas se analizaron como posibles soluciones al requisito de privacidad del mensaje. Sin embargo, ambas tecnologías están en etapas muy tempranas de desarrollo y por este motivo fueron descartadas. Por un lado, recién durante el final del desarrollo del proyecto de grado aparecieron las primeras versiones operativas de algunas *zkEVM*. No obstante, las mismas se enfocaban completamente en mejorar la performance y cantidad de transacciones, dejando de lado el apartado de privacidad. Por otro lado, los *Private Smart Contract* todavía están en etapa de desarrollo a nivel académico. Aunque existen algunas propuestas de implementación, aún no existe una solución operativa funcional.

Finalmente, el estudio de ambas tecnologías tuvo importantes implicancias en el desarrollo del proyecto. En primer lugar, permitió dimensionar de forma más precisa la dificultad que conlleva solucionar el requisito de privacidad del

mensaje. En segundo lugar, y más importante, se observó que varias implementaciones de ambas tecnologías utilizaban ZKP, lo que reorientó los esfuerzos del equipo hacia intentar solucionar el requisito con este tipo de pruebas.

Apéndice I

DApp Farmacia

La dApp de la farmacia está compuesta por cuatro archivos, el *index.html* y el *main.css*, que contienen la estructura y los estilos respectivamente de la aplicación web. Luego existen dos archivos *Javascript*, por un lado, está el *app.js* que contiene la lógica de negocio de la aplicación y la lógica para comunicarse con el *Pharmacy Smart Contract* en *Ethereum*. Por otro lado, está él *ui.js* cuyo propósito es conectar la UI con la lógica del *app.js*.

I.1. App.js

El archivo *app.js* utiliza varias funciones para interactuar con la *wallet* de *Ethereum* instalada en el navegador. Entre estas funciones encontramos *loadEthereum*, esta función carga la variable *web3* que se utilizara para interactuar futuramente con *Ethereum*. La misma puede observarse en la figura I.1

```
loadEthereum: async () => {
    if (window.ethereum)
   {
        App.web3Provider = window.ethereum;
        window.web3 = new Web3(window.ethereum);
        await window.ethereum.request({method: 'eth_requestAccounts'});
    } else {
        alert('ERROR: You have to install Metamask')
   }
}
```

Figura I.1: Función *loadEthereum* del app.js

Una vez cargada la variable de entorno *web3*, hay que cargar un objeto que represente al *smart contract* con el que se quiere interactuar. Para esto se utilizará el archivo ABI[1] *PharmacyContract.json*, el mismo fue generado al

[1]El archivo ABI, es un archivo JSON que describe el *smart contract* implementado y sus funciones.

compilar el *Pharmacy Smart Contract*. Esta tarea es realizada por la función *loadContracts*, la misma obtiene los datos del contrato desde su archivo JSON y utiliza la librería *web3* para cargar la variable *App.pharmacyContract* que representara al contrato en la dApp. Esta funcion puede observarse en la figura I.2

```
loadContracts: async () => {
    const res = await fetch("PharmacyContract.json")
    const pharmacyContractJSON = await res.json();
    const networkId = await web3.eth.net.getId();

    App.pharmacyContract = await new web3.eth.Contract(
        pharmacyContractJSON.abi,
        pharmacyContractJSON.networks[networkId].address,
    );
}
```

Figura I.2: Función *loadContracts* del app.js

Teniendo la variable *pharmacyContract* correctamente cargada, es hora de suscribirse a los eventos del *Pharmacy Smart Contract*. En particular, interesa suscribirse al evento *PrescriptionResult*, ya que este se lanzara cuando la prescripción haya sido aprobada o rechazada. La función *listenResultEvent* es la responsable de escuchar el evento y de especificar la función de *callback* asociada. Como se puede observar en la figura I.3 cuando el evento envía el estado "*data*" se verifica que sea el asociado a la prescripción procesada para el usuario actual de la terminal y posteriormente se ejecuta la función de *callback riseResult* que envía el resultado a la UI.

```
listenResultEvent: async () => {
    await App.pharmacyContract.events.PrescriptionResult()
    .on('data', function(event){
        if (
            event.event !== "PrescriptionResult" ||
            App.hashToHexString(event.returnValues.hash_prescription) !== App.hash_prescription
        ) {
            return;
        }

        App.riseResult(event.returnValues.result);
    })
    .on('changed', changed => console.log('EVENT CHANGED:', changed))
    .on('error', err => console.log('EVENT ERROR:', err))
    .on('connected', str => console.log('EVENT CONNECTED:', str))
}
```

Figura I.3: Función *listenResultEvent* del app.js

La última función relevante del *app.js* es la función *burnAndConfirmPrescription*, esta es analizada en la sección 6.4.1.

I.2. ui.js

Este archivo es el responsable de comunicar la UI con el *app.js*. A continuación, se describirá a grandes rasgos que hacen sus funciones, las cuales se pueden observar en la figura I.4. Cuando el DOM termina de cargar el contenido se invoca la función *App.init* del *app.js* y se envía como parámetro la función de *callback* a ser invocada cuando se termine de evaluar la validez de una prescripción. Por último se define el evento de *submit* del formulario, el cual válida que se haya ingresado el certificado con las credenciales anónimas y envía todos los datos ingresados en la función *App.burnAndConfirmPrescription* descrita en la sección 6.4.1.

```
const prescriptionForm = document.querySelector("#prescriptionForm");
const certificateInput = document.querySelector("#certificateInput");
var certificate = '';

document.addEventListener("DOMContentLoaded", () => {
    App.init((result) => {
        alert(result ? "Su prescripcion fue aprobada" : "Su prescripcion no fue aprobada");
    });
})

certificateInput.addEventListener("change", e => {
    var reader = new FileReader();
    reader.onload = event => {
        certificate = event.target.result;
    };
    reader.readAsText(e.target.files[0]);
})

prescriptionForm.addEventListener("submit", e => {
    e.preventDefault()

    if (certificate == '')
   {
        alert("You have to upload a certificate")
        return;
   }

    console.log("sending", prescriptionForm["id_prescription"].value);

    App.burnAndConfirmPrescription(
        prescriptionForm["hash_user"].value,
        prescriptionForm["id_prescription"].value,
        prescriptionForm["description_prescription"].value,
        certificate
   )

    console.log("sended");

    prescriptionForm["hash_user"].value = ''
    prescriptionForm["id_prescription"].value = ''
    prescriptionForm["description_prescription"].value = ''
    certificateInput.value = null;
    certificate = '';
})
```

Figura I.4: Funciones definidas en ui.js

Apéndice J

Smart Contract en Ethereum

Existen dos *smart contract* en *Ethereum*, el *PharmacyContract* y el *Verifier*. Este último es autogenerado con Zokrates a partir de la clave de verificación. Por tal motivo, la única función de relevancia que posee el contrato fue analizada en la sección 6.3.

Para poder utilizar el contrato *Verifier* desde el *PharmacyContract* es necesario importarlo y para esto hay que describir las estructuras que van a ser utilizadas del primero. Como se puede observar en la figura J.1, primero se describe la Liberia *Pairing*, de esta interesan los *struct G1Point* y *G2Point*, que son una representación de puntos en el plano X, Y. Luego se describe el contrato *Verifier*, en este son de interés el *struct Proof*, que es la representación de una prueba ZKP, y la función *verifyTx* que es la responsable de validar las ZKP.

Luego el contrato *Pharmacy* define una variable que es de tipo *Verifier*. Para cargarla es necesario que el *owner* del *PharmacyContract* (es decir, la *wallet* que lo desplegó en *Etehereum*) invoque la función *setVerifierContract* con la dirección del contrato *Verifier* desplegado en la red de *Ethereum*. Esta función además hace uso del modificador *_ownerOnly* el cual controla que solo el *owner* pueda ejecutar dicha función del contrato. La figura J.2 muestra las funciones y las variables definidas.

```
library Pairing {
    struct G1Point {
        uint X;
        uint Y;
    }
    // Encoding of field elements is: X[0] * z + X[1]
    struct G2Point {
        uint[2] X;
        uint[2] Y;
    }
}

contract Verifier {
    struct Proof {
        Pairing.G1Point a;
        Pairing.G2Point b;
        Pairing.G1Point c;
    }

    function verifyTx(Proof memory proof, uint[8] memory input) public view returns (bool r) { }
}

```

Figura J.1: Estructuras necesarias para importar el contrato *Verifier*

```
address owner;

constructor(address _verifierAddress) {
    owner = msg.sender;
    verifier = Verifier(_verifierAddress);
}

modifier _ownerOnly() {
  require(msg.sender == owner);
  _;
}

function setVerifierContract(address _verifierAddress) public _ownerOnly {
    verifier = Verifier(_verifierAddress);
}
```

Figura J.2: Función *setVerifierContract* y variables relacionadas