



UNIVERSIDAD DE LA REPÚBLICA

PROYECTO DE GRADO

---

# Manejo de la privacidad en permissioned blockchain

---

*Autores:*
Matías LEAL
Maximiliano MONTIGLIO

*Tutores:*
Dr. Alfredo VIOLA
Ing. Octavio PEREZ KEMPNER

Instituto de Computación
Facultad de Ingeniería

Montevideo - Uruguay
19 de noviembre de 2021

«*En un sólo día podemos encontrar más genios escondidos en el anonimato, que todos los oficialmente reconocidos en la historia de la humanidad.* »

José Luis Rodríguez Jiménez

UNIVERSIDAD DE LA REPÚBLICA

# *Resumen*

Facultad de Ingeniería
Instituto de Computación, InCo

Tesis de Grado

**Manejo de la privacidad en permissioned blockchain**

por Matías LEAL y Maximiliano MONTIGLIO

Una *blockchain* es un registro inmutable de transacciones (al que llamamos *ledger*) compartido entre un conjunto de nodos distribuidos. En particular, en una *blockchain permissioned* los usuarios deben ser identificados para permitirles realizar ciertas acciones. Por ejemplo, en una *blockchain permissioned* se conoce la identidad de todos los nodos que validan las transacciones. Las blockchain de tipo *permissioned* se han vuelto bastante populares entre las empresas que forman convenios, ya que prioriza la confianza sobre la privacidad.

En este proyecto se estudian dos técnicas para mejorar la privacidad en blockchains con una configuración *permissioned*. Una de estas estrategias está basada en el uso de una firma ciega, en donde se utiliza un algoritmo criptográfico para el cual se consigue firmar una transacción sin leer su contenido, y en la otra estrategia se utilizan pruebas de conocimiento cero no interactivas (o también conocidas como *Non Interactive Zero-Knowledge Proof*) para demostrar determinadas sentencias que impiden vincular a un usuario con determinada transacción.

Además, se presenta Hyperledger Fabric (o simplemente Fabric), una *blockchain* de tipo *permissioned*, en la cual se verán las instanciaciones de estos dos mecanismos de anonimato acompañado de un caso de estudio, al que llamamos *AleaChain*, una blockchain *permissioned* de juegos de azar, con la cual se busca ejemplificar la anonimidad obtenida así como los mecanismos utilizados, así como poder investigar la capacidad de esta tecnología desde el punto de vista del anonimato.
Respecto a la firma ciega, se presentan los cambios requeridos a realizarse en una red blockchain en concreto (Hyperledger Fabric), para que estas puedan incorporarse al *Framework*.

# *Agradecimientos*

Primeramente quisiéramos agradecerle al Tuba, por muchísimas cuestiones, entre ellas el haber sido el primero en darnos para adelante, haber aceptado nuestra propuesta inicial, abrirnos las puertas de su casa, haber dedicado tiempo de lectura en conjunto, por las múltiples charlas enriquecedoras, y el haber articulado este proyecto para que podamos elegir abiertamente el tema, y habernos ubicado con las personas adecuadas para poder realizarlo.

Por segundo, y no menos importante, quisiéramos agradecer a Octavio por haber aceptado participar en este trabajo con gran disposición y entusiasmo, por toda su ayuda, sus valiosos comentarios, y haber podido hacer todo esto desde Francia.

Por tercero, quisiéramos agradecer a Mirko Koscina, que desde Francia, tuvo la disposición de aclararnos dudas, y de manera muy amable y clara.

Por cuarto, a todos aquellos amigos (dentro o fuera de la facultad) que han aportado en este camino para volvernos profesionales.

Matías (a título personal): Contando toda la trayectoria hasta este punto, sin dudas no fue un camino trivial, y si tuviera que agradecer, sería primero a mi madre, por apoyarme para que estudiara desde un comienzo y poder convertirme en profesional. A Maxi por haber transitado todo este camino con él desde la UTU. A mí novia por ser el respaldo y compañía casi a diario. A mis demás amigos. Y dedicárselo a todos aquellos que te cruzan y te preguntan "¿en qué año estás?".

Maxi (a título personal): Realmente que ha sido un camino largo hasta este momento pero que ha valido la pena y que he podido disfrutar ampliamente. Este camino ha sido posible por el apoyo incondicional de mi familia, amigos y compañeros de facultad. En especial, gracias al apoyo de mis padres y mi hermano. Por último, gracias a Mati por compartir este camino desde el primer día.

# Índice general

# Índice de figuras

# Lista de siglas

| | |
|---|---|
| **DLT** | **D**istributed **L**edger **T**echnology |
| **CFT** | **C**rash **F**ault **T**olerance |
| **BFT** | **B**yzantine **F**ault **T**olerance |
| **PBFT** | **P**ractical **B**yzantine **F**ault **T**olerance |
| **PoW** | **P**roof-**o**f-**W**ork |
| **UTXO** | **U**nspent **T**ransaction **O**utput |
| **MSP** | **M**embership **S**ervice **P**roviders |
| **P2P** | **P**eer **t**oo **P**eer |
| **ZKP** | **Z**ero **K**nowledge **P**roof |
| **CA** | **C**ertificate **A**uthority |
| **zk-SNARK** | **z**ero-**k**nowledge **S**uccinct **N**on-**I**nteractive **A**rgument of **K**nowledge |

# Capítulo 1

# Introducción

## 1.1. Motivación

El tema fundamental de este proyecto es el anonimato y más específicamente el anonimato en el ámbito de *permissioned blockchains*. Con respecto al tema de anonimato las preguntas que nos inquietaron inicialmente fueron: ¿cómo se asegura el anonimato en una red que es compartida por una gran cantidad de usuarios?, ¿cómo se hace para hacer una transferencia a alguien en un sistema distribuido en donde son los mismos nodos de la red los que validan esos envíos, sin que quede ningún registro de quien fue el que envió y quien el que recibió esa transacción?, ¿cómo demuestra alguien que es el dueño de un elemento sin revelar su identidad?. Pues el hecho de contestar estas preguntas son la motivación que tiene este proyecto, y no solamente tratando el tema desde el punto de vista de las criptomonedas, sino de transacciones genéricas. No se verá una respuesta genérica para esta pregunta, pero sí se hará aportando el estudio a dos mecanismos que permitirán dar ejemplos de cómo se pueden resolver los problemas planteados por las preguntas. Es importante resaltar que este campo se encuentra en pleno desarrollo. Por lo tanto, es difícil saber cuales de todas las ideas presentadas en esta tesis son las que en el futuro sean ampliamente aceptadas o implementadas en la vida cotidiana de las personas.

## 1.2. Blockchain

Una *blockchain* es una estructura de datos que permite crear un registro digital de datos (llamado *ledger*) y compartirlo entre partes independientes. Este término se hizo famoso en el año 2008 cuando Satoshi Nakamoto publicó el artículo "P2P (peer to peer) Electronic Cash System"[1]. En dicho documento, el autor define al Bitcoin, una moneda digital [1] descentralizada que puede ser enviada entre los participantes de una red sin que exista una autoridad centralizada o capa intermedia que se vea involucrada. De hecho, el objetivo del desarrollo de Bitcoin fue resolver el problema de doble gasto al implementar el concepto de atomicidad [2, 3]. El doble gasto o *double spending* es un problema que ocurre cuando se intenta gastar una moneda en más de una transacción, como se puede ver en la figura 1.1.

1 Moneda digital: es un tipo de moneda o medio de intercambio disponible en forma digital, y no en forma física como pueden ser los billetes o las monedas de bronce, que permiten transacciones instantáneas y transferencia de propiedad sin fronteras

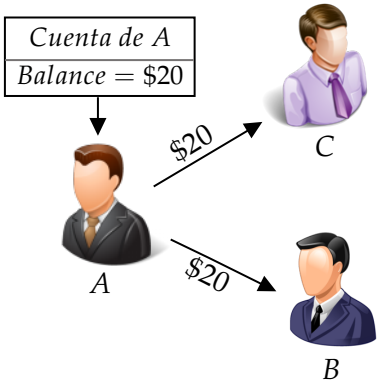


FIGURA 1.1: Problema del doble gasto

La atomicidad garantiza que toda transacción ocurre antes o después de otras transacciones o directamente no ocurre. Por ejemplo, *A* envía una transacción de $20 a *B*, que debe ocurrir en una marca de tiempo (*timestamp*) específica y que debe estar antes o después de cualquier otra transacción. La idea es evitar gastar los $20 más de una vez en transacciones simultáneas.

### 1.2.1. Plataformas Blockchain

En esta sección, veremos brevemente algunas plataformas que utilizan *blockchain*: Bitcoin, Ethereum, Monero, Zcash e Hyperledger Fabric.

#### 1.2.1.1. Bitcoin

En el artículo publicado por Nakamoto donde describe por primera vez Bitcoin [1] se integran de forma ingeniosa soluciones existentes a diferentes problemas de la ciencia de la computación, entre ellas está el algoritmo de *Proof-Of-Work (PoW)* (4.6.3) que sincroniza el estado del sistema cada 10 minutos mediante una elección global que soluciona el problema del doble gasto. Esta es una mejora significativa de las monedas digitales anteriores que intentaron resolver el problema del doble gasto a través de una entidad centralizada.

La red Bitcoin fue descrita el 31 octubre de 2008, mientras que la misma entró en funcionamiento el día 3 de enero de 2009 cuando Satoshi Nakamoto minó el primer bloque, cosechando una recompensa de 50 bitcoins. La red desde ese entonces ha sido mantenida y mejorada de forma continua por otros desarrolladores siguiendo un enfoque de proyecto de código abierto con participantes de todo el mundo.

#### 1.2.1.2. Ethereum

En 2015, Vitalik Buterin desarrolló Ethereum [4, 5] como una plataforma de código abierto para hacer cosas que Bitcoin no podía ni quería hacer. Ethereum fue concebido con la idea de una plataforma *blockchain* de propósito general, equipada con un poderoso lenguaje Turing-completo [2] capaz de escribir *smart contracts* (contratos inteligentes) 4.9 complejos para desarrollar cualquier tipo de aplicación. Al

2 Turing-Completo: es a aquel lenguaje que tiene un poder computacional equivalente a lo que se denomina Máquina de Turing Universal.

usar este enfoque, se abstraen los mecanismos de las redes entre pares, *blockchain* y consenso. Los desarrolladores pueden centrarse en el diseño y desarrollo de sus aplicaciones sin preocuparse por estos detalles.

Después del nacimiento de Bitcoin, las personas comenzaron a tomar conciencia del poder de *blockchain* e intentaron aplicarlo a otras aplicaciones que no sean criptomonedas. Estos intentos incluían construir sobre la red de Bitcoin o comenzar una nueva *blockchain*. Desarrollar Bitcoin tiene la ventaja de la potencia informática disponible de los participantes actuales, pero las aplicaciones estarían limitadas por las restricciones internas de la red. La implementación de *smart contracts* en Bitcoin es muy restringida, así como los tipos de transacción, los tipos de datos y el tamaño del almacenamiento de los datos, lo que limita severamente el tipo de aplicaciones que pueden ejecutarse directamente en la *blockchain*. Ethereum fue creado como una plataforma *blockchain* que resuelve todos estos problemas.

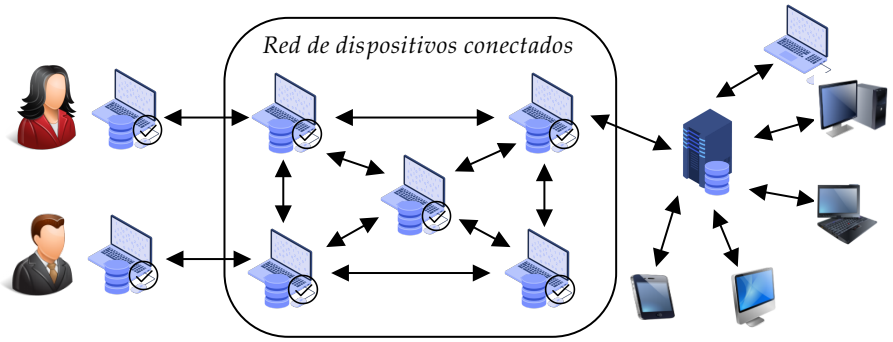


FIGURA 1.2: Sistema de *ledger* distribuido

#### 1.2.1.3. Monero

Monero [6] es una criptomoneda que se basa en el protocolo CryptoNote [7] el cual es un protocolo de capa de aplicación que fue desarrollado por Nicolas van Saberhagen en 2013. CryptoNote es una tecnología diseñada para ser una mejora con respecto a las criptomonedas como Bitcoin y Ethereum. La tecnología que hay detrás de CryptoNote es muy similar a la de muchas criptomonedas estándar. Aun así, existen diferencias importantes entre CryptoNote y la tecnología que hay detrás de otras criptomonedas. La más importante de estas diferencias es que el protocolo CryptoNote está diseñado para tener un mayor grado de privacidad y anonimato que las criptomonedas tradicionales. Actualmente existen diversas criptomonedas que utilizan este protocolo como núcleo de su plataforma *blockchain* además de Monero, por ejemplo: Bytecoin, Dashcoin, entre otras, siendo Monero la más exitosa.

Las criptomonedas basadas en CryptoNote se ejecutan en cadenas de bloques idénticas a la de *Bitcoin*, sin embargo, existe una diferencia notable: el origen de las operaciones queda oculto bajo el manto de anonimato que provee CryptoNote y no puede ser determinado. Para garantizar esto, se utilizan firmas de anillo [8] y claves únicas para cada transacción. Esto permite que solamente el emisor y el receptor, conozcan los detalles del intercambio. El resto de los usuarios únicamente pueden ver el monto aproximado de los fondos transmitidos.

Las características fundamentales que están relacionadas con la privacidad que implementa CryptoNote son:

- **Untraceable payments:** Esta característica implica que no es posible saber cual es la identidad de un usuario que ha creado una transacción. Para garantizar esta característica CryptoNote hace uso de firmas de anillos el cual es una primitiva de firma digital mas sofisticada. En esta primitiva se tiene un grupo de individuos llamado *anonymity-set* [3], donde cada uno de los participantes tiene su propia clave pública y privada. Lo que garantiza esta primitiva es que el firmante de un mensaje dado es un miembro del grupo y que está en condiciones de realizar la transacción. La principal distinción con las primitivas de firma digital estándar es que el firmante necesita una sola clave secreta, pero un verificador no puede establecer la identidad exacta del firmante a partir de una firma. Esto lleva a que no se pueda conocer con certeza quién fue el creador de la transacción sino que se sabe que es alguien que pertenece al *anonymity-set*.

- **Unlinkable transactions:** Esta características implica que dadas dos transacciones cualesquiera, debería ser imposible probar que fueron enviadas a la misma persona. Para garantizar esta característica CryptoNote, crea de forma automática múltiples claves únicas de un solo uso, derivadas de una única clave pública. En este protocolo, el remitente utiliza la clave pública del receptor, y agrega sus propios datos aleatorios, para calcular una clave única para el pago. Lo que se garantiza con esta primitiva es que la generación de claves será única para cada transacción, aunque el emisor y el receptor sean los mismos. Esto lleva a que no se pueda establecer una clave pública que identifique a un usuario y lograr vincular las transacciones que recibe.

Kumar at el. [9] realizaron un análisis sobre la trazabilidad de las transacciones en Monero. En primera instancia se logró determinar por medio de un análisis pasivo (solamente leyendo las transacciones en la *blockchain*) que el 65 % de las transacciones tenían un *anonymity-set* de tamaño uno, lo que significa que las entradas de estas transacciones eran rastreables. Además, este error de utilizar un *anonymity-set* de ese tamaño disparaba un efecto cascada que permitía que otras transacciones que tenían *anonymity-set* de mayor tamaño también se vieran comprometidas, y terminaran siendo rastreables. Como resultado final, se concluye que el 87 % de las transacciones se pueden rastrear. Es importante aclarar que los resultados listados en ese artículo han sido conseguidos con una versión de Monero que no hacía uso de RingCT [10]. Actualmente, monero utiliza RingCT que permite ocultar de cada transacción los montos enviados, los montos recibidos, el emisor y el receptor. La implementación de RingCT ha ayudado a disminuir la probabilidad de éxito a largo plazo de algunas de las heurísticas utilizadas en [9]. Las otras heurísticas aprovechan las malas prácticas utilizadas por los usuarios por lo que su solución depende de la divulgación de buenas prácticas hacia los usuarios.

#### 1.2.1.4. ZCash

Zcash es una criptomoneda destinada a proporcionar un método más avanzado de privacidad a sus usuarios, basada en el protocolo Zerocash [11]. Zerocash es un

3 Anonymity-set: es el conjunto de entidades que pueden tener los mismos atributos , haciéndolos indistinguibles entre sí dentro de un contexto específico desde la perspectiva de un atacante u observador en particular.

protocolo que proporciona una versión de Bitcoin que preserva la privacidad (aunque también es aplicable a monedas similares a Bitcoin).

En general, para que una criptomoneda sea confiable, un usuario debe poder verificar la correctitud de las transacciones. En Bitcoin, esta capacidad se logra gracias a la *ledger*, con lo que las transacciones son auditables públicamente. Cada transacción es totalmente transparente, las direcciones de origen y de destino, así como el monto de la transacción están en texto claro. Si una identidad del mundo real está asociada con una dirección de Bitcoin, entonces puede ocurrir una pérdida grave de privacidad. Las soluciones para resolver los problemas de privacidad de Bitcoin van desde la utilización de muchas direcciones, a estrategias de *mixing*, donde varios usuarios envían monedas a una dirección que posteriormente se agrupan y combinan con las de otros usuarios antes de ser enviadas a los receptores.

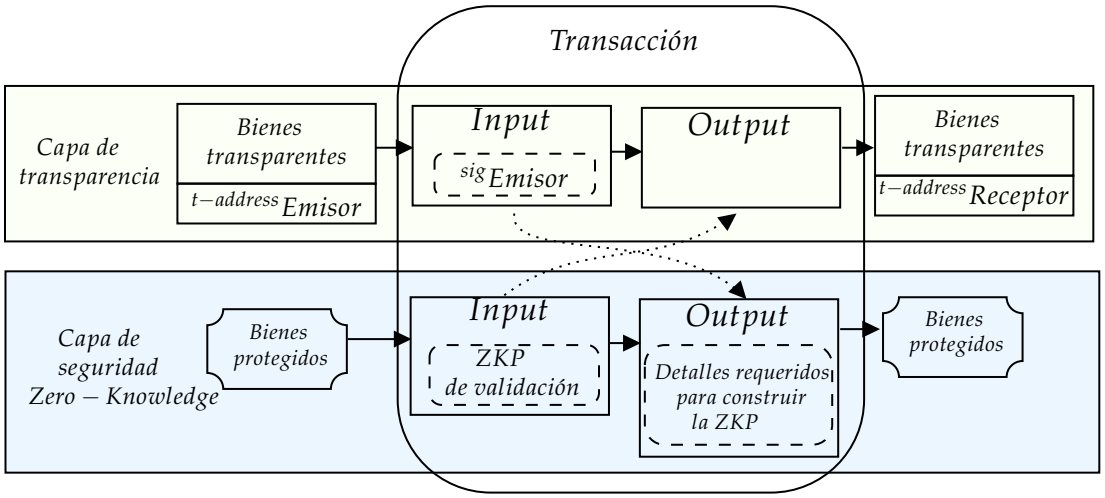


FIGURA 1.3: Representación de transacciones en ZCash

La operación esencial de una criptomoneda es la transferencia, donde se envía una cierta cantidad de "monedas" de Alicia hacía Bob. Luego, Bob puede enviar estas monedas a otra persona, pero Alicia no debe ser capaz de "volver a gastar" las monedas. Zerocash [12] proporciona privacidad y fungibilidad [4] al codificar estas restricciones como un circuito aritmético con una prueba zk-SNARK (*Zero-Knowledge Succinct Non-Interactive Argument of Knowledge*) para el circuito adjunto a la *blockchain* como un registro de la transacción, que puede ser verificado de manera eficiente. Zerocash define el concepto de un "esquema de pago descentralizado", y luego muestra cómo se pueden utilizar zk-SNARK para construir dicho sistema.

#### 1.2.1.5. Hyperledger Fabric

Hyperledger se inició en 2015 por la Linux Fundation [13] como una plataforma *blockchain* de código abierto con soporte para *ledgers* distribuidas basados en blockchain. Se centra en el desarrollo de *blockchains* con el objetivo de mejorar el rendimiento y la confiabilidad de las transacciones comerciales globales de las principales compañías tecnológicas, financieras y de cadenas de suministros. El proyecto

4 Bien fungible: Son aquellos bienes que pueden sustituirse los uno por los otros de la misma calidad y en igual cantidad, debido a su equivalencia recíproca.

tiene como objetivo permitir que múltiples equipos individuales colaboren para desarrollar protocolos y estándares abiertos, proporcionando un marco modular que admite diferentes componentes para diferentes usos. Este enfoque puede admitir una variedad de *blockchains*, cada una con su propio algoritmo de consenso, modelos de almacenamiento, servicios de identidad, control de acceso y contratos.

El proyecto Hyperledger consta de muchas *blockchains* individuales aportadas por diferentes compañías, e Hyperledger Fabric es una de ellas. Al igual que Bitcoin y Ethereum, tiene una *ledger*, usa *smart contracts*, y es un sistema mediante el cual los participantes gestionan sus transacciones. Pero aquí es donde terminan las similitudes. Hyperledger Fabric se diferencia de los sistemas públicos de *blockchain* (como Bitcoin o Ethereum) en que es *permissioned*. Si bien los sistemas abiertos *permissionless* permiten que entidades desconocidas participen en la red y utilizan protocolos como *Proof-of-Wook* para validar las transacciones y asegurar la red, los miembros de una red Hyperledger Fabric se inscriben a través de un Proveedor de Servicios de Membresía (MSP) de confianza. Los nodos en Hyperledger Fabric se pueden dividir en varios tipos: *committer*, *endorser* y *orderer*. Existe la posibilidad de que un nodo cumpla con las tareas de mas de un tipo al mismo tiempo.

### 1.2.2. Ventajas y desafíos de *Blockchain*

La tecnología *Blockchain* parece una gran solución para aumentar la seguridad e integridad de nuestras bases de datos, pero no está exenta de debilidades. Si bien *blockchain* puede proporcionar descentralización, transparencia, inmutabilidad y disponibilidad a sus datos, una gran parte de las *blockchains* como Bitcoin y Ethereum sufren problemas de pérdida de privacidad y bajo rendimiento.

#### 1.2.2.1. Ventajas

- **Descentralización**: La ventaja más notable de *blockchain* es la de eliminar la necesidad de un tercero de confianza. Todos pueden verificar la validez y la autorización de las transacciones de *blockchain*. Las bases de datos tradicionales deben ser alojadas y administradas por una entidad, y no importa cuán confiable sea esa entidad, todavía existe la posibilidad de que alguien que tenga acceso al sistema y pueda alterar los datos. El factor humano es siempre el eslabón más débil de un sistema de seguridad. Al eliminar a este tercero de escena, no solo aumenta la seguridad del sistema, sino que también reduce el costo para las organizaciones y simplifica enormemente los procesos.
- **Transparencia**: *Blockchain* es una *ledger* distribuida, lo que significa que los nodos de la red tienen acceso a una copia de la base de datos, por lo que el sistema es transparente y no se puede modificar la *ledger* sin que todos lo sepan.
- **Inmutabilidad**: Es muy difícil modificar los datos después de que se escriben en la *blockchain*, y cuanto más tiempo haya estado presente una transacción en la cadena de bloques, más difícil será modificarla. Para cambiar un bloque, el atacante necesita volver a calcular cada bloque posterior, lo cual no es posible sin la colusión de la mayoría de los nodos en el sistema.
- **Disponibilidad**: Debido a que la *ledger* se replica y actualiza en cada nodo de la red, no existe un único punto de falla. A medida que crece el número de nodos, la disponibilidad se vuelve cada vez mayor, y si algunos nodos no están disponibles, la red sigue funcionando normalmente.

#### 1.2.2.2. Desafíos

- **Privacidad**: Debido a que *blockchain* es totalmente transparente, todos pueden saber lo que hacen los demás, y aunque las partes son anónimas en la *blockchain*, la información confidencial aún puede filtrarse. Un atacante puede rastrear fácilmente transacciones del usuario y obtener información confidencial. Esto es importante en el sector financiero, donde los usuarios generalmente no quieren que su situación financiera sea conocida por todos. En este aspecto, un sistema con una autoridad central puede tener la ventaja de tener control de acceso a los datos, por lo que es más fácil ocultar información.

- **Performance**: La performance siempre ha sido un problema con *blockchain*. Las dos criptomonedas más populares, Bitcoin y Ethereum, admiten menos de 20 transacciones por segundo, una cantidad muy pequeña en comparación con las admitidas por los sistemas de bases de datos transaccionales modernos. Si bien el sistema de consenso de *Proof-of-Work* ayuda a garantizar la inmutabilidad de los datos, también supone una carga para la red. El proceso de minería requiere una gran cantidad de potencia informática y cuanto más grande sea la red, más potencia se necesita para generar un solo bloque.

### 1.2.3. Aplicaciones de *Blockchain*

En esta Sección se verán algunas de las aplicaciones de *blockchain* en diferentes ámbitos que han sido propuestas hasta el momento. Estas propuestas abarcan ámbitos como las finanzas, Internet de las cosas (IoT), cuidados de la salud y manejo de cadenas de suministros. A continuación, se describe como es que se ha utilizado *blockchain* en cada una de estas actividades:

- **Finanzas**: La aparición de sistemas *permissionless* de *blockchain* como Ethereum y sistemas *permissioned* como Fabric despertó el interés del público al estar revolucionando los servicios financieros y comerciales tradicionales.
  *Blockchain* puede potencialmente alterar muchas áreas, incluida la banca, la gestión de riesgos, la transformación empresarial o el mercado financiero. Desde 2016, grandes compañías de software como IBM y Microsoft también comenzaron a ofrecer *blockchain* como servicio. Alex Tapscott y Don Tapscott explican en [14] como a partir de la utilización de *blockchain* compañías de cualquier tamaño recaudan dinero directamente de los inversores a través de ofertas de globales sin tener que incurrir en los costos de intermediación de los banqueros. Esto permite a las compañías bajar los costos operativos haciendo que estas compañías sean más rentables.

- **Internet de las cosas (IoT)**: la seguridad de los dispositivos IoT está en una etapa muy prematura, especialmente en lo que refiere a anonimato y autenticación. La seguridad y la privacidad son las dos principales preocupaciones en los sistemas que involucran dispositivos IoT. La utilización de *blockchain* en IoT puede proporcionar seguridad y privacidad, dos preocupaciones principales en los sistemas informáticos relacionados con IoT. Por ejemplo, Rifi *et al.* [15] propuso una arquitectura descentralizada basada en *blockchain* que utiliza contratos inteligentes para implementar el acceso seguro a datos en IoT.

- **Cuidados de la salud**: *Blockchain* puede potencialmente crear una gran revolución en el ecosistema de atención médica al brindar seguridad y privacidad a

la gestión de los datos clínicos de los pacientes. Los individuos serán responsables de manejar sus propios registros y de obtener el control general de sus propios datos con el apoyo de esta tecnología. Al utilizar una *ledger* compartida, todas las entidades involucradas en el proceso podrían acceder al sistema sin importar qué sistema médico electrónico utilicen. Esto no solo ofrece mayor seguridad y transparencia, sino que también permite a los médicos dedicar más tiempo a la atención y el tratamiento del paciente. Además, las estadísticas de las investigaciones serán más fáciles de compartir, lo que, a su vez, facilitaría los ensayos clínicos y las terapias de tratamiento para enfermedades poco comunes. En un sistema de atención médica, el intercambio de datos entre los proveedores de soluciones de atención médica es tan importante que puede conducir a una mayor precisión en el diagnóstico, tratamientos efectivos y un ecosistema rentable. A medida que los datos de los pacientes crecen cada día más, se requiere la utilización adecuada de los recursos para recopilar y aprovechar de manera efectiva los conocimientos de esos datos. Una propuesta de este tipo de sistemas es la de "My Health My Data" (MHMD) [16].

- **Cadenas de suministros**: Debido a que la tecnología *blockchain* permite rastrear de forma segura y transparente todo tipo de transacciones, puede proporcionar una mejora en los sistemas de cadena de suministros. Cada vez que un producto cambia de manos, la transacción puede documentarse, creando un historial permanente, desde la fabricación hasta la entrega al consumidor final. Esto podría reducir los retrasos, los costos adicionales y los errores humanos. Específicamente, *blockchain* puede mejorar las siguientes tareas:

  - Registrar la cantidad y la transferencia de activos, como los remolques, contenedores, etc., a medida que se mueven entre los intermediarios de la cadena de suministro.
  - Seguimiento de órdenes de compra, órdenes de cambio, recibos, notificaciones de envío u otros documentos relacionados con el comercio.
  - Asignar o verificar certificaciones o ciertas propiedades de productos físicos; por ejemplo, determinar si un producto alimenticio es orgánico.
  - Compartir información sobre el proceso de fabricación, montaje, entrega y mantenimiento de productos con proveedores y vendedores.

  En el artículo [17] los autores investigan cuales son los requisitos y funcionalidades necesarias para la integración de los sistemas informáticos con la cadena de suministros y explican como la integración de *blockchain* puede cumplir con ellos, logrando una mayor eficiencia y un menor costo para todos los participantes de la cadena. Un ejemplo de integración de *blockchain* en cadenas de suministros es [18] donde los autores utilizan *blockchain* para registrar el proceso de fabricación de cajas de cartón, construyendo un historial desde la fabricación hasta el reciclaje. Los autores detallan los beneficios potenciales para los consumidores, los proveedores y el impacto en el medio ambiente.

- **Voto electrónico**: Construir un sistema de votación electrónica seguro que ofrezca la confianza y la privacidad de los esquemas de votación tradiciones, al tiempo que proporciona la transparencia y flexibilidad ofrecidas por los sistemas electrónicos ha sido un desafío durante mucho tiempo. La integración de *blockchain* en los sistemas de voto electrónico permite que los votos se registren de forma precisa, permanente, segura y transparente. Por lo tanto, nadie podrá modificar o manipular los votos. Además, *blockchain* proporciona cierto nivel

de anonimidad que permite que no se pueda saber que es lo que ha votado cada ciudadano mientras que también se puedan realizar auditorías por parte de entidades externas. Aunque nada es totalmente seguro, la manipulación es casi imposible con *blockchains*. En [19] los autores proponen un sistema de voto electrónico descentralizado basado en *blockchain* que permite a los usuarios votar desde cualquier dispositivo con acceso a internet, mientras que los resultados serán verificables y no podrán ser corrompidos por estar replicados en los nodos de la red.

## 1.3. Trabajos relacionados

El estudio de la privacidad en plataformas que hacen uso de *blockchain* ha sido de gran interés para los investigadores desde hace varios años. Los primeros trabajos desarrollados en esta área fueron enfocados en el análisis de la privacidad de las plataformas que hacían uso de *blockchain* para su funcionamiento. Bitcoin, al ser la plataforma que revolucionó el uso del dinero digital por medio de la utilización de *blockchain*, fue la primera en recibir particular atención con respecto a los niveles de privacidad que mantenía. Uno de los argumentos muy usados en la comunidad para fomentar el uso de Bitcoin fue el de ser un medio de pago totalmente anónimo, lo cual no es cierto ya que por diseño Bitcoin no es anónimo sino que es *psuedoanonimo* lo cual es un nivel de anonimato menor. Por otro lado, los primeros trabajos en el análisis de privacidad de Bitcoin dejaron a la plataforma en un nivel de vulnerabilidad mayor, ya que estos demostraban que era posible vincular a los usuarios del mundo real con las transacciones dentro de la plataforma de Bitcoin dejando al descubierto todo su historial de pago en la plataforma.

A partir de estos resultados es que se comienzan a desarrollar nuevas propuestas que aumenten el nivel de anonimato en Bitcoin, las cuales en principio eran integrables en el protocolo de Bitcoin sin que este sea modificado pero que iban en contra de una de las premisas de la plataforma como lo es la descentralización, hasta llegar a *CoinShuffle* donde se describe una nueva propuesta que utiliza un sistema de mixing pero que no depende de terceros de confianza. Por otro lado, algunos desarrolladores tomaron muy en serio el comentario de los desarrolladores del núcleo de Bitcoin donde argumentaban que tener una plataforma totalmente anónima no era el objetivo de Bitcoin. Tomando en cuenta esto, decidieron diseñar *Zerocoin* la cual es una nueva plataforma que se deriva de Bitcoin pero que no es compatible con la misma, la cual por medio de primitivas criptográficas busca tener transacciones totalmente anónimas sin depender de terceros de confianza.

### 1.3.1. Análisis de seguridad

A partir de la irrupción de Bitcoin en el mercado de las monedas privadas, se comienza a investigar la seguridad de estas monedas y puntualmente cual es el nivel de privacidad de las mismas. Por un lado, se ha dado particular atención al análisis de los datos que se guarden en la *blockchain* y cómo esto repercute en la privacidad de los usuarios. Mientras que por otro lado, se ha analizado cuanta información se puede obtener por medio de un análisis de tráfico de red y cuanto compromete la privacidad de los usuarios.

#### 1.3.1.1. Análisis de transacciones

Los primeros análisis de privacidad en plataformas que hacen uso de blockchian han sido enfocados en Bitcoin. Para esta tarea se han tomado dos caminos, el primero donde se hace un análisis pasivo de la información pública de la plataforma[20-22], mientras que en el segundo se hace un análisis activo donde se realizan pagos desde direcciones de Bitcoin propias a servicios conocidos para identificarlos en el futuro o donde se hace trabajar a un nodo como intermediario en la creación de transacción para obtener información sobre los usuarios. A continuación se describen algunos de los trabajos de mayor interés que hacen uso del análisis de transacciones.

El primer artículo de investigación sobre privacidad en Bitcon fue escrito en 2011 por Reid y Harrigan [20] donde a partir de la información publicada en la *blockchain*, los autores construyen una red de transacciones y una red de usuarios. El primero representa el flujo de *bitcoins* entre transacciones, donde cada vértice representa una transacción y cada arista dirigida indica si existe o no una dirección de entrada/-salida que vincule las transacciones. El segundo representa el flujo de usuarios de *Bitcoin* a lo largo del tiempo, para construir la red de usuarios, los autores agrupan las direcciones del mismo usuario suponiendo que todas las direcciones de entrada de una transacción pertenecen al mismo usuario. A partir de la información recolectada los autores concluyen que es posible asociar muchas direcciones *Bitcoin* entre sí y también asociar direcciones *Bitcoin* con información de identificación externa a la plataforma.

En el artículo escrito por Androulaki et al. [21] los autores se plantean realizar un análisis de la privacidad de la plataforma Bitcoin utilizando la información pública de la misma. Para evaluar la privacidad en esta plataforma, construyen un simulador de una plataforma Bitcoin pero restringido a usuarios de una universidad. Por medio de la utilización de dos heurísticas simples que explotan dos características del protocolo Bitcoin se llega a que el 40 % de los perfiles de los usuarios de este entorno cerrado ha podido ser reconstruido aun cuando estos siguen las medidas de privacidad recomendadas por Bitcoin. Una de las heurísticas se basa en el análisis de las transacciones con múltiples direcciones de entrada, haciendo uso de que no se pueden realizar transferencias donde varios usuarios hagan una contribución al pago sino que un único usuario puede pagar con diferentes direcciones las cuales es propietario, esta heurística es difícil de dejar sin efecto por la forma en la que está construido el protocolo. Mientras que la otra heurística se basa en el análisis de las salidas de una transacción, más puntualmente en la dirección donde se redirige el sobrante del pago de una transacción la cual también debe estar en poder del usuario que efectuó el pago. Esta heurística si es posible de evitar haciendo una reutilización de algunas de las direcciones de entrada para redirigir el sobrante de la transacción.

En el artículo escrito por Meiklejohn et al. [23] se hace un análisis activo. Al realizar pagos desde direcciones de Bitcoin propias a servicios conocidos (como grupos de minería, billeteras en línea, sitios de intercambio, etc.) pueden identificar dichos servicios en el futuro por medio de las transacciones alojadas en la *blockchain*. Además, también navegan por internet para obtener la identificaciones de usuarios de otras direcciones. Luego, utilizaron las dos mismas heurísticas que en [21] para descubrir vínculos entre direcciones bitcoins. Con su análisis, los autores concluyen que para grandes transacciones de Bitcoin, es posible rastrear sus movimientos y la red de Bitcoin no ofrece suficiente anonimato, por ejemplo, para el lavado de dinero. Tal

trazabilidad es aún más aguda en caso de que el analizador esté (o tenga acceso) a un servicio central, como un grupo de minería, un proveedor de billetera electrónica o un sitio de intercambio de bitcoins.

#### 1.3.1.2. Análisis de tráfico

Otro aspecto a considerar cuando se está analizando la privacidad de un sistema informático que mantiene comunicaciones a través de la red es la seguridad a lo largo del protocolo TCP/IP. En este sentido, lo que se busca analizar es si por medio del tráfico creado por uno de los nodos hacia otro de los nodos de la red se puede obtener alguna información sobre quienes son los emisores y los receptores de las transacciones vinculando las direcciones de las transacciones con direcciones IP.

Uno de los primeros en tener en cuenta este nivel de seguridad para las plataformas de *blockchain* fueron Reid y Harrigan en [20], donde hacen una pequeña descripción del problema pero sin ser muy profundos en su análisis.

Uno de los primeros artículos en tratar esta problemática es el desarrollado por Koshy at el. [24] en donde realizan un estudio de anonimato basado en el tráfico de transacciones en tiempo real recopilado durante 5 meses. Para ello, los autores desarrollan CoinSeer, un cliente de Bitcoin diseñado exclusivamente para la recopilación de datos. Para más de 5 millones de transacciones, recopilaron información sobre la dirección IP de donde CoinSeer recibió dicha transacción y, en el caso general, asignaron como la IP correspondiente a la transacción la que transmitió la transacción por primera vez. Para realizar un análisis de red puro, los autores no aplican ningún proceso de agrupación de direcciones, por lo que solo las transacciones con una única dirección de entrada (casi cuatro millones) se han podido tener en cuenta. Después de su análisis, los autores concluyen que es difícil vincular direcciones IP con direcciones Bitcoin realizando un análisis de tráfico si los peers de Bitcoin actúan correctamente. Además, los autores también indican que algunas configuraciones de red, como servicios de mixing o billeteras electrónicas, pueden conducir a suposiciones erróneas al vincular direcciones IP y direcciones *Bitcoin*, por lo que se puede decir que estas técnicas son exitosas contra estos ataques.

### 1.3.2. Nuevas propuestas

A partir de las problemáticas encontradas en la plataforma Bitcoin se han desarrollado variantes para solucionar los problemas de privacidad. Algunas de las propuestas pueden ser integradas a la plataforma Bitcoin sin que esta deba de ser modificada como el caso de las propuestas basadas en mixing, mientras que otras propuestas no son compatibles con el estado actual de la plataforma, siendo necesario crear un fork de la misma para poder adaptar los cambios. Aunque las propuestas basadas en mixing sean fácilmente integrables a la plataforma de Bitcoin, algunas de ellas van totalmente en contra de la descentralización planteada originalmente en Bitcoin ya que incorporan entidades centralizadas en las cuales se debe confiar.

#### 1.3.2.1. CoinShuffle

Una propuesta que busca eliminar los servicios centralizados de mixing es la desarrollada por Ruffing et al. [25]. Esta propuesta utiliza los principios definidos en

CoinJoin [26] para garantizar la seguridad contra el robo y utiliza un protocolo de comunicación de grupos anónimos para garantizar el anonimato. Esta propuesta también proporciona compatibilidad total con el estado actual de la plataforma Bitcoin al igual que Mixcoin [27]. Mientras que las características innovadores que agrega esta propuesta son la eliminación de terceros de confianza, siendo únicamente ejecutado por los usuarios interesados en lograr la desvinculación de sus transacciones, esta desvinculación esta garantizada si existen al menos dos usuarios que actúan de forma honesta. Al eliminar estos terceros de confianza también se eliminan las comisiones que recibían los mismos, bajando los costos a los usuarios. A pesar de la incorporación de estas características, la sobrecarga introducida es insignificante.

#### 1.3.2.2. ZeroCoin

Zerocoin es un sistema de dinero electrónico distribuido propuesto por Miers et al. [28] para el protocolo de Bitcoin que utiliza técnicas criptográficas para romper el vínculo entre las transacciones individuales de Bitcoin sin agregar terceros en quien confiar. Este sistema esta basado en la transformación de Zerocoins en Basecoins y viceversa. Basecoin es una altcoin [5] similar a Bitcoin, y Zerocoin es una extensión de esta altcoin. En este sistema, Basecoin es la moneda con la que se realizan transacciones, y Zerocoin solo proporciona un mecanismo para cambiar sus basecoins por otras nuevas que no se pueden vincular a las antiguas.

Para poder realizar transferencias de monedas es que se debe por un lado crear una nueva moneda zerocoin por medio del proceso llamado *minting* pero para que la misma tenga valor se debe de renunciar a un basecoin del mismo valor. El proceso de minting consiste en la creación de un *commitment* criptográfico donde el usuario publica en la *blockchain* el *commitment* de una tupla de valores aleatorios que mantendrá en secreto, al publicar este *commitment* de forma instantánea se destruye una basecoin y crea una zerocoin. Para poder gastar un zerocoin uno debe de demostrar que conoce los valores correspondientes al *commitment* que publico, pero esto no se puede realizar de forma directa ya que sino se dejaría en evidencia la relación con la basecoin que se destruyó, por lo cual es necesario realizar una prueba *zero-knowledge* donde dice conocer los valores aleatorios correspondiente a uno de los *commitments* guardados en la *blockchain*. Para que la transacción sea válida, se verificará que la prueba de *zero-knowledge* realmente abre uno de los *commitment* guardados en la *blockchain* y también verifican que el mismo no ha sido abierto anteriormente para evitar el doble gasto.

La utilización de *commitments* permite romper la vinculación entre la basecoin original y la zerocoin generada en el proceso de *minting*, mientras que la utilización de pruebas *zero-knowledge* hace que los verificadores no aprendan nada más que la correctitud de la misma, manteniendo a los dueños de las monedas la utilización de las mismas.

5 Altcoin: El término Altcoin se refiere a criptomonedas que derivan del código fuente de Bitcoin también conocidas como bifurcación.

## 1.4. Objetivos propuestos

Este proyecto aborda como eje, el manejo de la privacidad en blockchain, puntualmente en las blockchains de tipo *permissioned* las cuales tienen ciertas premisas de trabajo que hacen que el manejo de la privacidad en ellas sea diferente al que se hace en las blockchains *permissionless* (como lo son Bitcoin o Ethereum). En un comienzo, se presentan los conocimientos teóricos necesarios para la compresión de este trabajo, los cuales abarcan conocimientos teóricos sobre criptografía y *blockchain*. Posteriormente se plantea el estudio de la plataforma Fabric la cual implementa una *blockchain* de tipo *permissioned* de uso empresarial, que tiene un nivel de anonimato bajo y que será descrito. Tomando esta plataforma como base se plantea el estudio de dos alternativas las cuales realizan modificaciones al flujo normal de la plataforma en pos de lograr un mayor nivel de anonimato. Se busca entender en profundidad los esquemas criptográficos que se implementan en cada unas de las alternativas, pero fundamentalmente se busca entender por que es que se utiliza cada uno de ellos, qué ventajas se obtienen, que desventajas y en presentar posibles alternativas a las utilizadas. Además, se plantea una comparativa entre el esquema original de Hyperledger Fabric y las dos alternativas planteadas en post de explicitar las ventajas y desventajas de cada uno de ellos. Para poder lograr esto, se plantea la definición de un caso de estudio que sea aplicable a las tres variantes, dando en cada uno cuales son los actores que están involucrados, como son sus interacciones, los roles de los usuarios, etc.. Luego de dar una descripción completa del caso de uso en cada una de las plataformas se presenta un análisis comparativo entre las tres variantes, y describe las fortalezas y debilidades de la aplicación del caso de estudio de cada una. Como último objetivo se propone la implementación e integración de una de las alternativas propuestas (BlindCons) en la plataforma de Hyperledger Fabric.

## 1.5. Organización del documento

El trabajo se encuentra organizado de la siguiente manera:

- El Capítulo 2 define el caso de estudio que se utilizará como ejemplo para facilitar las ideas utilizadas en el marco de las distintas estrategias que se verán para atacar la problemática.
- El Capítulo 3 contiene los fundamentos teóricos criptográficos definidos para el desarrollo de las estrategias que se utilizan en los Capítulos posteriores.
- El Capítulo 4 explica los fundamentos de *blockchain* que serán esenciales para comprender el resto de este proyecto, así como dar una descripción del potencial que tiene la tecnología *blockchain*.
- El Capítulo 5 se centra exclusivamente en la plataforma Hyperledger Fabric, una *blockchain* de tipo *permissioned*, la cual se escoge como la *blockchain* a la cual expresar las estrategias de los Capítulos 6 y 7, que abordan el problema de la privacidad.
- En el Capítulo 6 se desarrolla la estrategia BlindCons, la cual aplica en *blockchains* de tipo *permissioned*, en las cuales los usuarios que realicen transacciones conservarán su identidad gracias a un mecanismo de firma ciega. A su vez se explica una posible instanciación de dicha estrategia en Hyperledger Fabric.

- En el Capítulo 7 se desarrolla una estrategia para preservar la privacidad en un sistema de tokens distribuidos, como podría ser Hyperledger Fabric, a su vez que se explican los fundamentos teóricos de este tipo de sistemas, y una propuesta para obtener privacidad y auditabilidad en estos esquemas.
- En el Capítulo 8 se finaliza con un resumen sobre los temas desarrollados junto con las conclusiones principales del trabajo. Además se plantean otras líneas de trabajo relacionadas con los temas expuestos en este proyecto en los cuales se puede seguir profundizando.

## 1.6. Terminología en inglés

Es importante aclarar que la literatura fundamental de los temas tratados en este proyecto está en inglés, y hay poco material en español. Por tal motivo, no es sencillo encontrar traducciones adecuadas al español. Por claridad, y debido al hecho de que la nomenclatura aceptada internacionalmente está en inglés, se utilizarán estos términos en su lenguaje original. Se toma como convección para todo el proyecto, la escritura de estos términos pertenecientes al idioma inglés haciendo uso de la tipografía itálica.

# Capítulo 2

# Caso de estudio

A la hora de entender en detalle los problemas estudiados en este proyecto se encuentran dos grandes dificultades. La primera dificultad es que son problemas donde existen diversos participantes con objetivos y acciones diferentes. Mientras que la segunda dificultad es el uso de diversas primitivas criptográficas donde el motivo de su uso no se infiere de forma directa. Teniendo en cuenta estas dificultades, se ha optado por presentar un caso de estudio que sirva de guía al lector a la hora de entender los problemas planteados. El caso de estudio elegido es un casino en línea.

En particular esta problemática va centrada en las soluciones existentes, que utilizan tecnologías *blockchain*. El escenario de estudio estará basado en un sistema de casinos online, al que se nombra AleaChain (Alea en latín quiere decir fortuna o suerte, de la que proviene aleatorio. Alea es por tanto, sinónimo de azar, y chain refiere a la estructura de cadena de bloques en la que se almacenarán los datos la cual será una *blockchain*) basado en *blockchain* para garantizar un nivel de transparencia más alto del que los casinos en línea tienen actualmente.

## 2.1. Realidad planteada

Es normal que día a día, más casinos amplíen su cartera de negocios haciendo uso de TICs [1], debido a que a la gente le gusta apostar utilizando sus computadoras o teléfonos celulares, desde la seguridad y comodidad de sus hogares. Todos los que han concurrido a jugar en un casino saben que además de las apuestas se incurre en otros gastos, por ejemplo en bebidas, en comida, en transporte, etc., pero al jugar en línea se gasta únicamente el dinero realizando apuestas. A grandes rasgos, un casino en línea que utiliza *blockchain* tiene las mismas características que un casino en línea y además agrega otras que los casinos en línea tradicionales no pueden garantizar. En la sección 2.2.1 se describen cuales son los beneficios que otorga el uso de *blockchain* en un sistema de casino en línea.

## 2.2. Estado del arte

A continuación, se describen las principales problemáticas de los sistemas de apuestas online, en base a los artículos [29] y [30].

El problema de casi todos los casinos en línea es la falta de confianza entre el jugador y la casa de apuestas. Los jugadores no saben que hay detrás de la interfaz del juego, es decir, se enfrentan a una caja negra, en la que deberán de confiar

1 **Tecnologías de la información y la comunicación (TIC)**: término que enfatiza el papel de las comunicaciones, la integración de las telecomunicaciones y las computadoras, así como el software que permiten a los usuarios acceder, almacenar, transmitir y manipular información

a regañadientes, incluso si se trata de un casino de gran reputación. Todas las casas de apuestas, no importa si se ejecutan en línea o fuera de línea, están sujetas a las mismas regulaciones las cuales deben cumplir.

Por ejemplo, todas las máquinas de juego automáticas, están programadas para devolver un cierto porcentaje del conjunto de dinero apostado a los jugadores, el cual debe estar entre un 80 % a un 99 % de dicho dinero. Algunos sitios de juegos de azar, un tanto "grises" violan estas normativas, ya que no les importa la reputación. En estos casos simplemente al cabo de un tiempo cambian de dominio, redefinen su interfaz y comienzan desde cero pero ahora con un nuevo nombre.

Esto último es algo que ocurre con bastante frecuencia ya que se puede realizar con relativa facilidad. Solamente es necesario abandonar el dominio que identifica a la casa de apuestas mientras que sus propietarios pueden desaparecer con todo el dinero que han recaudado.

La defensa que tienen los usuarios contra estas actitudes es la generación de listas negras de casinos fraudulentos, pero no pueden evitar que aparezcan nuevos.

Surge naturalmente la siguiente pregunta: ¿cuáles son las principales causas que llevan a estos casinos a encontrarse en una lista negra?, y a continuación se listan las distintas problemáticas que presentan estas plataformas que las llevan a ingresar en listas negras:

- **Falta de pago**: las personas no solo juegan para divertirse, sino también para obtener un rédito económico. Por eso es importante que los casinos sean honestos con los pagos. Algunos casinos permiten retirar el dinero ganado de forma rápida, pero en otros los pagos pueden demorar 30 días o incluso más. Los casinos tienen que pagar para no caer dentro de listas negras. Comúnmente un método para no pagar es mentir sobre el envío de cheque de pago, que nunca llega a destino, o simplemente decirle al jugador que no cumple con las normas para realizar retiros.

- **Difícil acceso**: hacer un depósito para jugar puede ser difícil, debido a que los bancos no les gustan las instituciones de juego. Hay muchas historias sobre personas que les han cerrado sus cuentas bancarias después de hacer un depósito en un casino en línea.

- **Falta de aleatoriedad**: cada casino tiene que usar un generador de números aleatorios para que todas las simulaciones utilicen secuencias de valores con una distribución uniforme generados independientemente. Todos los jugadores deberían poder tener una larga racha perdedora o ganadora, pero basándose en la "misma suerte" que si tuvieran en un casino de manera presencial. Sin embargo, el mecanismo de generación de números aleatorios (*Random Number Generator* [2]) puede ser manipulado para alterar las posibilidades, sesgando de forma que el jugador se le haga imposible ganar. En caso de detectarse este comportamiento se pondría al casino en una lista negra de forma automática.

2 ***Random Number Generator*** (RNG): es un proceso que genera una secuencia de números o símbolos que no se pueden distinguir polinomialmente de una secuencia generada con valores uniformes independientes.

- **Publicidad engañosa**: algunos casinos intentan atraer a jugadores ofreciendo muchos bonos imposibles de obtener debido a que solicitan muchos requisitos (como añadir tarjeta de crédito o cuenta de PayPal, y además enviar un pequeño pago inicial). Por lo que muchos usuarios no pueden reclamarlos.
- **Casinos spammeadores**: a la gran mayoría de los usuarios de casillas de correo les ha llegado una gran cantidad de correo basura con ofertas inútiles, algunos ni siquiera entienden por que les han llegado. Un tipo de oferta muy común es la realizada por los casinos, que recurren a llenar las bandejas de entrada de usuarios de casillas de correo con promociones vinculadas con su plataforma.
- **Otras prácticas cuestionables**: mal soporte técnico, falta de respuesta a las solicitudes de los usuarios, falta de seguridad y de cifrado adecuadas, poner en riesgo a sus usuarios, o simplemente software lento, cualquiera de estos ítems son suficientes para que un casino termine en una lista negra. También sucede que a los ganadores se les impide volver a usar el sitio si han ganado mucho dinero. La gente afortunada es percibida como un riesgo comercial.

### 2.2.1. Beneficios de utilizar tecnologías *blockchain* en sistemas de casinos

Se detallan, a continuación los beneficios de utilizar tecnologías *blockchain* para afrontar esta problemática, en base a los artículos [30], [31] y [32].

Además de las facilidades que ofrece un casino en línea para que los jugadores no tengan que moverse de sus hogares, la integración de *blockchain* en los mismos nos provee las siguientes ventajas que hasta el momento eran problemas que los casinos en línea no tenían forma de demostrar que realmente resolvían:

- **Seguridad**: al utilizar un *smart contract* como intermediario entre los participantes de una apuesta se garantiza que mientras que se está realizando la simulación de la apuesta los fondos apostados quedan bloqueados hasta finalizar la simulación. Esto lleva a que nadie pueda realizar una transacción involucrando las monedas apostadas ni tampoco nadie podrá decir que no quería realizar esa apuesta. Cuando la simulación termine se verificará el resultado y se enviaran las recompensas a el/los ganador/es.
- **Transparencia**: al utilizarse *blockchain* como registro inmutable de los resultados de cada apuesta, es posible realizar una trazabilidad de los resultados de cada juego para verificar que las simulaciones de los mismos han sido honestas y cumplen con las distribuciones de probabilidad que el juego dice tener.
- **Eficiencia**: al utilizar oráculos de confianza para realizar la simulación de los números pseudoaleatorios, se elimina el tiempo empleado en la ejecución de algoritmos que certifiquen la confiabilidad de estos oráculos a la hora de generar números pseudoaleatorios. Este es un requisito excluyente en un juego de apuestas en tiempo real, que lleva a desechar algunas plataformas de *blockchain* por no poder confirmar los resultados de una apuesta en segundos.

### 2.2.2. Plataformas en ejecución

Desde la irrupción de *blockchain* en el mundo de las criptomonedas, han existido muchos ámbitos que han visto que las características que tiene esta estructura de

datos puede ayudar a solucionar problemas que hasta el momento no habían podido ser solucionados. Se puede decir que el ámbito de los casinos online ha sido uno de ellos, ya que existen una gran cantidad de plataformas que han adoptado la tecnología *blockchain*. Lo que caracteriza a este tipo de plataformas es la necesidad de obtener los resultados de forma instantánea y sin costos extras, lo cual no se acopla del todo bien con:

- **Tiempos de confirmación**: todas las transferencias creadas en una plataformas que haga uso de una *blockchain* de tipo *permissionless* necesitan tiempo (a veces segundos, minutos o hasta horas) para confirmar su resultado. Por ejemplo en el caso de Bitcoin se necesitan de diez minutos para que un bloque para considerar como válido el resultado de una transacción.
- **Pago de tarifa por transacción**: cada transacción creada en una plataforma que haga uso de una *blockchain* de tipo *permissionless* tendrá que pagar una tarifa como compensación a los participantes de la red que se encargan de verificar que esta sea válida.

Una solución a estos problemas han sido los *State Channels* [33]. Estos canales permiten crear una cantidad potencialmente ilimitada de transacciones por fuera de la *blockchain* al costo de únicamente dos transacciones en la *blockchain*. El ciclo de vida de un *State Channel* esta determinado por tres eventos:

1. **Bloqueo de fondos de los participantes**: en esta etapa un conjunto de participantes acuerda bloquear fondos que sirven como garantía en las transacciones que van a ser creadas en la siguiente etapa. En esta etapa, todos los participantes deben de estar de acuerdo en participar para que la misma sea exitosa. Se puede realizar por medio de una firma digital 3.5 o por medio de un *smart contract* 4.9.
2. **Actualización de los fondos bloqueados**: los participantes del canal actualizan de forma incremental los fondos bloqueados mediante la creación de una cantidad potencialmente infinita de transacciones sin costo.
3. **Confirmación final de los fondos bloqueados**: luego de realizados los cambios a los fondos bloqueados se actualizan de forma permanente en la *blockchain* con el último estado en el cual los participantes se han puesto de acuerdo.

Únicamente las primeras dos etapas involucran modificaciones en la *blockchain*, minimizando los costos de creación de transferencias y el tiempo de confirmación de las mismas. A continuación se presentan las características principales de dos plataformas de casino online que hacen uso de *blockchain* y de *State Channels* para mejorar la experiencia de los usuarios:

- **FunFair**: es una plataforma y un protocolo de código abierto basados en *Ethereum* para juegos de casino online [34]. En la plataforma se integran juegos desarrollados por los integrantes del proyecto con juegos creados por desarrolladores independientes. Estos juegos se integran a la plataforma para que los jugadores los utilicen, mientras que los desarrolladores son recompensados por la utilización que tengan sus juegos. Las características tecnológicas que hacen atractiva esta plataforma en el ámbito de los casinos online son las siguientes:

- Utilización de *Fate Channels* que permiten generar números aleatorios en tiempo real. Y también ejecuta la simulación del *smart contract* que corresponde a la apuesta, de forma independiente sin que deba existir interacción entre los usuarios ni tampoco con terceros de confianza (son una implementación de *State Channels* en FunFair).
- Tiene una moneda propia representada en forma de token llamada FUN la cual es de fácil acceso.
- Utilización de un algoritmo de resolución de disputas a utilizarse cuando uno de los participantes no esta de acuerdo con los resultados.
- Persistencia de los resultados de las apuestas en la *blockchain*.
- Flexibilidad en las decisiones de los jugadores a la hora de apostar, permitiendo aumentar las apuestas, disminuir las apuestas o volver a apostar los montos ganados.

– **DaoBet**: es una plataforma y protocolo de juegos que utiliza *blockchain* [35]. El protocolo permite la cooperación entre *peers* desconocidos en el contexto de los juegos de azar, como los desarrolladores de juegos, los operadores de casinos, los propios jugadores y también los auditores. El protocolo DaoBet ofrece transparencia a través de la completa apertura del código fuente. Además utiliza el algoritmo Signidice [36] para la generación de números aleatorios de forma descentralizada entre dos participantes. Esto permite al jugador recibir los resultados de la apuesta mucho más rápido. Las características tecnológicas que hacen atractiva esta plataforma en el ámbito de los casinos online son las siguientes:

- Tiene bajos costos de comisiones ya que elimina los intermediarios por medio de la utilización de *State Channels*.
- Implementación de un mecanismo de resolución de disputas en caso de que los participantes de una apuesta no esté de acuerdo con el resultado que el otro participante pública.
- Tiene una moneda propia representada en forma de token llamada BET la cual es posible adquirir directamente o mediante una migración de tokens provenientes de la plataforma Ethereum.
- Persistencia de los resultados de las apuestas en la *blockchain*, los cuales pueden ser vistos por cualquier individuo utilizando la plataforma de DaoBet.

## 2.3. AleaChain

En esta sección se define un caso de uso concreto de un sistema de casinos basado en tecnología *blockchain*. Para su descripción se hará uso de conceptos tales como *ledger* 4.2 y *smart contract* 4.9.

Se plantea el problema describiendo los requisitos de la plataforma de un casino online. Por simplicidad se omite todo diseño o diagrama de diseño como sería el enfoque de ingeniería de software debido a que no es el objetivo de este proyecto. Y dado que este caso es simplemente para ejemplificar las estrategias y para resolver este problema concreto, se detalla cómo se resuelve cada requisito.

La idea central es utilizar la tecnología de *blockchain* para atacar los problemas históricos de los sistemas de casinos online, discutidos en la sección 2.2.1.

Como requisito principal se tiene la necesidad de que la plataforma otorgue un mayor nivel de transparencia que el otorgado por los casinos online típicos. Esto va a ser posible por medio de la utilización de *smart contracts* que permitirán que ninguno de los jugadores robe el dinero apostado o se niegue a pagar luego de obtenidos los resultados. Por otro lado, los usuarios podrán leer el resultado de todas las transacciones realizadas en el sistema haciendo consultas a la *blockchain*.

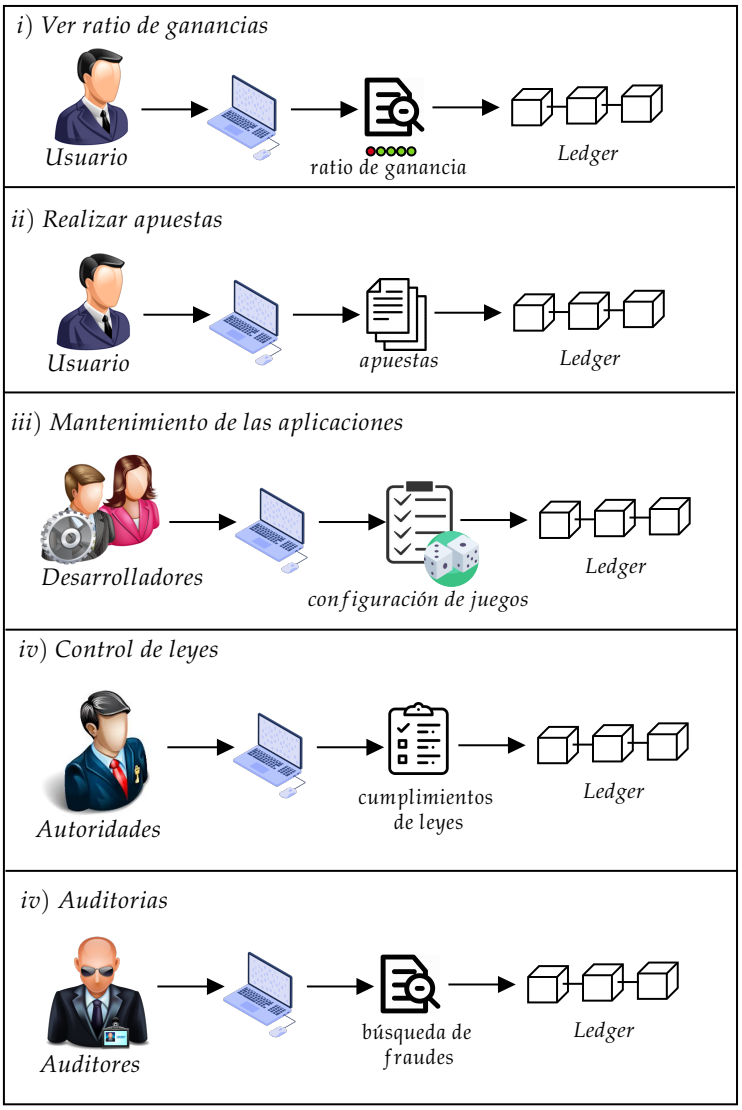


FIGURA 2.1: Escenarios de uso en AleaChain

En particular, se hará uso de una *blockchain permissioned*, esto implica que todos los usuarios deberán autenticarse en el sistema. En esta plataforma, los jugadores podrán participar en diferentes juegos de azar, en los que podrán apostar fichas a las que se nombran como ***AleaToken***. Estas fichas representarán algún valor monetario, lo cual incentivará a los jugadores a aumentar su cantidad de fichas realizando apuestas.

En la figura 2.1 se muestran de forma gráfica todas las interacciones de los usuarios con el sistema. Las cuales se pueden describir como:

- Ver ratio de ganancias: los usuarios pueden verificar cual es el ratio de ganancia histórico de cada juego por medio de consultas a la *blockchain*.
- Realizar apuestas: los usuarios pueden apostar en un juego de su elección.
- Mantenimiento de las aplicaciones: los desarrolladores pueden realizar modificaciones a los juegos que tiene desplegados en la plataforma. Esto se logra por medio de modificaciones a los *smart contracts*.
- Control de las leyes: las autoridades del casino pueden verificar que todos los usuarios estén siguiendo las reglas por medio de lecturas de la *blockchain*. En caso de encontrar usuarios con comportamientos maliciosos, puede aplicar las sanciones pertinentes.
- Auditorías: los auditores externos pueden verificar que el casino esta cumpliendo las regulaciones por medio de lecturas de la *blockchain*. En caso de no hacerlo, puede aplicar las sanciones pertinentes.

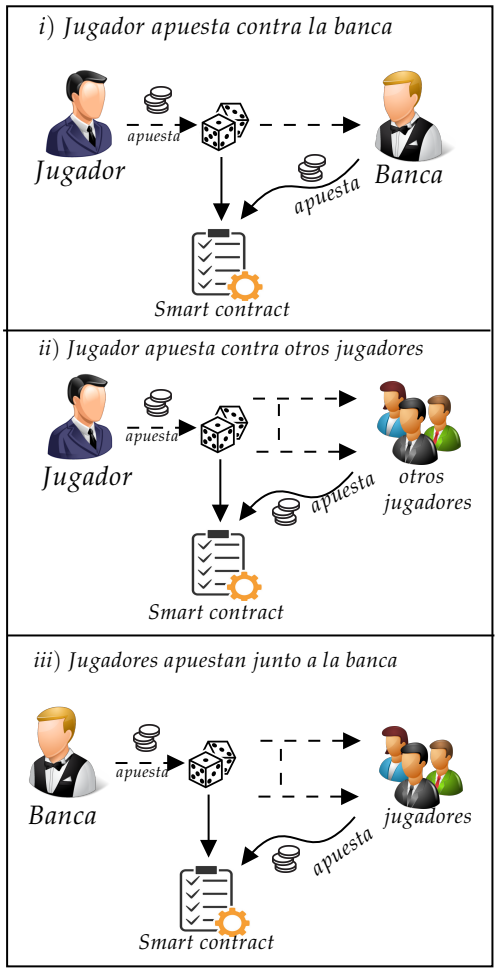


FIGURA 2.2: Tipos de apuestas en AleaChain

Como es evidente, un jugador no podrá apostar si no cuenta con el saldo necesario ni tampoco apostando dos veces las mismas fichas de forma simultanea.

*AleaChain* tiene el requisito de que cada apuesta sea completamente anónima. Esto implica que no se puede vincular con la apuesta a ninguno de los participantes ni tampoco saber la cantidad apostada por cada uno de ellos.

Los jugadores podrán ver las apuestas que se han realizado en los distintos tipos de juegos de azar disponibles. Esto permite poder calcular el ratio de retorno de cada juego al realizar un seguimiento de los resultados obtenidos en el juego. Aun así no podrán obtener ninguna otra información más que esa, es decir, no se podrá saber quién o quiénes han estado realizando apuestas, ni la cantidad apostada ni ningún otro tipo de vinculación.

Se podrán llevar a cabo tres tipos de apuestas, como se muestran visualmente en el figura 2.2, las cuales se describen de la siguiente forma:

- **Apuesta contra la banca**: el jugador y la banca crearán un *smart contract* indicando el tipo de juego en el que desea apostar. En ese *smart contract*, el jugador le establecerá a la banca la cantidad de fichas que desea apostar. Después de ponerse de acuerdo en los valores del *smart contract*, el contrato procederá a solicitar un número aleatorio a algún oráculo de la plataforma. Luego de recibido el número aleatorio se determinará quien es el ganador de la apuesta. Posteriormente se creará una transacción adjudicando los fondos al ganador de la apuesta.
- **Apuesta entre jugadores**: los jugadores que participan en un juego crearán un *smart contract*. En este *smart contract* indicarán el tipo de juego en el que desean apostar y la cantidad de fichas que apostarán. Después de ponerse de acuerdo en los valores del *smart contract*, el contrato procederá a solicitar un número aleatorio a algún oráculo de la plataforma. En base a este número se determinará quién o quiénes son los ganadores de la apuesta. Posteriormente se creará una transacción adjudicando los fondos a los ganadores de la apuesta.
- **Apuestas entre la banca y varios jugadores**: la banca siempre tendrá a disposición juegos de azar, por medio de *smart contracts* públicos que generará por cada apuesta. Esto permite entrar a los jugadores, y que puedan apostar las cantidades que deseen y en el juego de su elección. Los jugadores establecerán la cantidad de fichas que apostará cada uno. Después de que todos los participantes de la apuesta se ponen de acuerdo en los valores del *smart contract*, el contrato procederá a solicitar un número aleatorio a algún oráculo de la plataforma. En base a este número se determinará quién o quiénes son los ganadores de la apuesta. Posteriormente se creará una transacción por cada ganador de la apuesta adjudicando los fondos ganados a cada uno de los ganadores.

**Aclaración sobre el mecanismo de apuestas:** La idea de utilizar un *smart contract*, por apuesta, tiene como objetivo bloquear los montos apostados, de forma similar a la estrategia de *state channel* que tiene DaoBet o de *fate channel* de FunFair.

La plataforma contará con usuarios especiales que estarán destinados a la realización de tareas específicas, las cuales se detallan a continuación:

- **Reguladores**: tendrán la tarea de verificar que no existan jugadores realizando trampas o realizando apuestas sospechosas. También tendrán la tarea de verificar que se estén cumpliendo las normas establecidas (por ejemplo las tasas de

retorno de los juegos, o que los oráculos que proporcionan números aleatorios sigan la distribución que han especificado). Teniendo a su vez, la posibilidad de expulsar jugadores del sistema, o de invalidar juegos que se encuentren en el sistema.

- **Oráculos**: tendrán la tarea de recibir las solicitudes de números aleatorios de los *smart contracts* generados en las apuestas. Cada *smart contract* le indicará al oráculo el rango donde quiere que se genere el número aleatorio y el oráculo responderá con un número aleatorio uniforme en ese rango. La generación de números aleatorios podrá ser obtenida de algún sitio de confianza externo a la plataforma (por ejemplo: random.org o oraclize.it). La otra posibilidad es que sea el mismo oráculo el encargado de generar los números aleatorios por medio de algún generador de números pseudoaleatorios. Estas entidades obtendrán un incentivo económico por cada número aleatorio que retornen a los *smart contract* para motivar a que actúen de forma honesta. En caso de que un oráculo no actúe de forma honesta, los reguladores los reportaran a los administradores de la plataforma para que sean eliminados.

En la figura 2.2 se describe lo que sería un escenario típico de apuesta entre dos jugadores. Luego de interactuar y ponerse de acuerdo en el tipo de apuesta y la cantidad de fichas a apostar crearan un *smart contract* que registra estos datos. Este *smart contract* sera guardado en la *ledger*. Cuando el *smart contract* se ejecute cada uno recibirá el monto que le corresponda según gane o pierda la apuesta.

Para las distintas estrategias se flexibilizan los requisitos en caso de que no puedan cumplirlos todos. No será estrictamente necesario cumplir con cada uno de ellos, dado que el motivo de definir este caso de estudio es para ejemplificar con las distintas estrategias que serán descritas en este proyecto, y hacer una comparativa entre sí. En la Secciones 5.6, 6.6, 7.6 (que son las secciones que finalizan los capítulos 5, 6, 7, respectivamente) se describe la implementación de este caso de uso en cada una de las estrategias. Finalizando con la comparativa de cada estrategia al final del Capítulo 7, en la sección 8.2 en donde se realiza un análisis comparativo de las ventajas y desventajas que presenta la implementación de este caso de uso en cada una de las estrategias.

# Capítulo 3

# Fundamentos teóricos

En este Capítulo se presentan los fundamentos teóricos necesarios para la comprensión de las diferentes propuestas planteadas en los capítulos 5, 6 y 7. Se comienza presentando algunas nociones de seguridad 3.1 las primitivas criptográficas utilizadas a lo largo del trabajo. Se finaliza el capitulo definiendo formalmente algunos términos que serán utilizados ampliamente a lo largo de esta tesis, como por ejemplo *anonymity* o *unlinkability*.

## 3.1. Introducción

Este proyecto emplea algunas definiciones, y modelos de seguridad para especificar características de algunas de las primitivas a utilizar. Primero se definen nociones de seguridad que explican los objetivos del adversario, y cuánto poder e información tiene a su alcance. También es necesario definir los posibles modelos de prueba, que definen el conjunto de supuestos que ayudan a probar la seguridad de los algoritmos o primitivas utilizadas. La definiciones de esta Sección son las expuestas en [37]

### 3.1.1. Nociones de seguridad

En términos de su poder computacional, un adversario se puede clasificar en uno de los siguientes tipos:

- **Computationally-bounded**: Un adversario que solo es capaz de realizar cálculos en tiempo polinomial. Un modelo con este tipo de adversario proporciona seguridad computacional.
- **Computationally-unbounded**: Un adversario que tiene cantidades desconocidas (no necesariamente infinitas) de recursos. Por lo tanto, cualquier algoritmo en este modelo no puede asumir ningún límite en los poderes computacionales del adversario y debe mantener su seguridad con cualquier cantidad de poder computacional que el adversario pueda tener.

En términos de acciones que un adversario puede tomar, estas se pueden clasificar en uno de los siguientes tipos:

- **Passive (semi-honest o honest-but-curious)**: El adversario puede leer el estado interno de los participantes corruptos y su comunicación, tratando de obtener alguna información a la que no tiene derecho. A esto se le llama *eavesdropping* (escuchar a escondidas). El modelo *passive* es el modelo adversario más débil.

- **Active (malicious o Byzantine)**: Adicionalmente el adversario puede hacer que los participantes malintencionados se desvíen de la especificación del protocolo para falsificar el resultado del protocolo. El modelo *active* es el modelo de adversario más fuerte.
- **Semi-malicious**: El adversario sigue la ejecución del protocolo (similar a un adversario *passive*), pero puede elegir sus monedas (y entradas) aleatorias de cualquier manera arbitraria.

En términos de cuándo se elige el conjunto de participantes malintencionados, un adversario puede clasificarse en uno de los siguientes tipos:

- **Static**: El adversario está restringido a elegir su conjunto de participantes deshonestos al comienzo del protocolo y no puede cambiar este conjunto más adelante.
- **Adaptive**: El adversario puede elegir su conjunto de participantes deshonestos en cualquier momento durante la ejecución del protocolo.

### 3.1.2. Modelos de prueba de seguridad

- **Standard model**: Un algoritmo es seguro en el *Standard model* si la única suposición que hace para su prueba de seguridad es el poder computacional del adversario (llamado suposición *standard*). Las construcciones criptográficas se basan generalmente en supuestos de complejidad, que establecen que algún problema, por ejemplo, la factorización o el problema del logaritmo discreto, no se pueden resolver en tiempo polinomial. Si bien existen varias pruebas de seguridad en el modelo estándar, las pruebas de seguridad son notoriamente difíciles de lograr en el *Standard model* [38].
- **Random oracle model**: Un *random oracle* (óraculo aleatorio) es una caja negra abstracta que genera números verdaderamente aleatorios. Un algoritmo es seguro en el *random oracle model* si asume la existencia de un oráculo aleatorio como prueba de seguridad [39]. Se expone con mayor detalle en 3.3.
- **Common reference string (CRS) model**: Un algoritmo es seguro en el *CRS model* si supone que todas las partes tienen acceso a un string aleatorio en común tomado de una distribución predeterminada [40].
- **Public key infrastructure (PKI) model**: Un algoritmo es seguro en el modelo PKI si asume que las identidades de todas las partes pueden verificarse de manera confiable utilizando técnicas de criptografía de clave pública (PKC) 3.4.

### 3.1.3. Modelos de prueba de seguridad

Algunas de las propiedades de las distintas primitivas que serán presentadas en futuras secciones se las describe siguiendo el paradigma de *game based proof*.
En un *game based proof* la seguridad de un algoritmo criptográfico se expresa como un *game* (juego) entre un *adversary* (adversario) y una entidad hipotética llamada *challenger* (retador). Tanto el *adversary* como el *challenger* son procesos probabilísticos que se comunican entre sí, por lo que se puede modelar un *game* como un espacio de probabilidad. Para introducirse en las *game based proof* se recomienda el siguiente artículo [41].

Generalmente, la definición de seguridad se vincula a algún evento particular $\mathcal{S}$. En ese contexto, seguridad significa que para cada adversario eficiente, la probabilidad de que el evento $\mathcal{S}$ ocurra, es muy cercana a alguna probabilidad objetivo específica: típicamente, 0, $\frac{1}{2}$, o la probabilidad de algún evento en algún otro juego en el que el mismo *adversary* está interactuando con un *challenger* diferente [42]. Este modelo es utilizado para complementar las definiciones de *hiding* y *binding* 3.7, *blindness* 3.4 y la propiedad *unforgeability* 3.2

## 3.2. Funciones de Hash

Una función de hash $H$ es un función que toma como entrada un string de largo arbitrario y retorna un string de largo fijo. Formalmente, una función de hash es una función

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

donde $n$ es un entero de largo fijo.

Dentro de la familia de las funciones de hash existe un subconjunto llamado funciones de hash criptográficos. Estas funciones de hash cumplen con ciertas propiedades especiales. Estas propiedades son muy útiles a la hora de utilizarlas dentro de otras primitivas criptográficas. Las propiedades que caracterizan a los hash criptográficos son:

1. *Preimage Resistant*: para todas las salidas $y \in \{0,1\}^n$ es computacionalmente difícil [1] encontrar un $x \in \{0,1\}^*$ tal que $H(x) = y$. Es decir, dado un elemento perteneciente al codominio es difícil encontrar otro elemento del dominio que tenga el mismo valor de hash.

2. *Second preimage resistance*: dado $x \in \{0,1\}^*$ es computacionalmente difícil encontrar un $x' \in \{0,1\}^*$ con $x \neq x'$ tal que $H(x) = H(x')$. Es decir, dado un elemento cualquiera del dominio, es difícil de encontrar otro elemento diferente en el dominio que tenga el mismo valor de hash.

3. *Collision resistant*: es computacionalmente difícil encontrar cualquier pareja $x, x' \in \{0,1\}^*$ con $x \neq x'$ tal que $H(x) = H(x')$. Es decir, es difícil de encontrar cualquier pareja de elementos en el dominio que tengan el mismo valor de hash. La diferencia con la propiedad anterior es que no hay restricción en la elección de los elementos del dominio mientras que estos sean diferentes.

4. *Puzzle friendliness*: no existe una estrategia para resolver un rompecabezas computacional basada en la función hash que sea más eficiente que simplemente probar valores aleatorios de un conjunto de posibles soluciones.

En este proyecto se verán las funciones de hash criptográficas en la construcción de los punteros entre bloques de la *blockchain*, en la construcción de algoritmos de firma digital, en la construcción de algoritmos de firma ciega, en la construcción de la estructura de datos que contiene las transacciones dentro de los bloques de la *blockchain* llamada *Merkle trees* (árboles de Merkle), entre otras aplicaciones.

1 Computacionalmente difícil: informalmente este término refiere que, a menos que $P = NP$ no existe ningún algoritmo que resuelva el problema en tiempo polinomial para todas las entradas posibles.

## 3.3. Modelo del oráculo aleatorio

Antes de describir el modelo del oráculo aleatorio es necesario explicar qué es un oráculo aleatorio. A continuación, se describen las particularidades de este artefacto que lo hacen útil a la hora de probar la seguridad de primitivas criptográficas.

### 3.3.1. Oráculo aleatorio

Un oráculo aleatorio es la entidad encargada de proveer a los participantes de una primitiva criptográfica tal que dado un rango de valores de entrada entre 0 y $n$, devuelve cada vez que es invocada un valor uniforme en $0..n$ e independiente de otras llamadas. Tiene la particularidad de que si se le pregunta dos veces con el mismo parámetro de entrada $x$ entonces responde en ambas ocasiones con el mismo valor. Por estos motivos, dado un rango entre 0 y $n$, un oráculo aleatorio se puede considerar como una función de *hash* perfecta, en el sentido de que para cada elemento $x$ de entrada devuelve un elemento aleatorio y uniforme en dicho rango e independiente del resto de los valores de entrada.

Para dar una descripción mas amigable sobre el oráculo aleatorio, se puede afirmar que un oráculo es una entidad que tiene en su poder una tabla de valores (potencialmente infinita) que comienza vacía y a medida que es consultado asigna valores aleatorios de largo fijo a las entradas consultadas. La particularidad que tiene esta tabla es que el oráculo la construye a demanda, es decir, a medida que los participantes le consultan por los valores de entrada el oráculo les asigna valores de salida aleatorios. En caso de que el elemento ya este ingresado en la tabla, el oráculo simplemente responde su valor de salida. Por lo que tiene una forma perezosa de construir la función aleatoria.

Es válido preguntarse cuál es la razón para no usar directamente una función aleatoria como la que implementa el oráculo aleatorio, en vez de una función de hash criptográfica. La respuesta a esta pregunta es simple: **no es viable en el mundo real**. Si se tomara la decisión de guardar toda la tabla donde se especifica los valores de entrada y de salida, entonces, se necesitaría un espacio de almacenamiento exponencial en el largo de los valores de entrada (se tendrían $2^n$ entradas, siendo $n$ el largo de la entrada). Aún teniendo el espacio para guardar esta tabla, el tiempo necesario para recorrer dicha tabla en busca de un valor de entrada, tendría un orden exponencial.

### 3.3.2. Pruebas de seguridad en el modelo del oráculo aleatorio

El modelo del oráculo aleatorio (*Random Oracle Model (ROM)*) es una heurística utilizada para demostrar la seguridad de primitivas criptográficas bajo la premisa de que las funciones de hash utilizadas se comportan como una fuente perfecta de aleatoriedad uniforme e independiente. Bajo este modelo es habitual construir primitivas criptográficas "seguras.[en] dos pasos. En el primer paso se diseña la primitiva haciendo uso de un oráculo que tiene acceso a una función aleatoria con el cual los participantes interactúan. Luego se prueba que dada esa realidad, el sistema es seguro. En el segundo paso se hace un reemplazo de este oráculo por una función de hash criptográfica (por ejemplo $SHA-1$, $SHA-3$), en la cual todos los participantes tienen acceso. Luego de este segundo paso, se termina obteniendo una implementación de una primitiva criptográfica utilizable en el "mundo real"donde los oráculos

aleatorios no existen. Esta metodología de construcción de primitivas criptográficas fue introducida por Bellare y Rogaway en [43].

### 3.3.3. Controversias del modelo del oráculo aleatorio

Se puede notar que este cambio de un oráculo aleatorio hacia una función de hash criptográfica no permite asegurar que la primitiva que se ha demostrado que es seguro en el ROM lo siga siendo cuando se haga uso de la función de hash. Esta particularidad ha despertado en la comunidad científica grandes debates sobre la utilidad del modelo [44-47].

Por un lado, se han publicado resultados muy negativos para este modelo, como por ejemplo, el artículo escrito por Canetti *et.al.* [44] donde se ha demostrado que existen primitivas de cifrado y de firma digital que son seguras en el ROM pero que son inseguras para cualquier función que se ejecute en tiempo polinomial. En las conclusiones finales, Canetti argumenta que el ROM es una mala abstracción para analizar la seguridad de primitivas criptográficas. Mientras que Goldreich dice que una primitiva que se haya probado segura en el ROM no puede tomarse como evidencia de la seguridad de las implementaciones de esta primitiva.

Por otro lado, Koblitz y Menezes aseguraron que aunque la propuesta presentada en [44] era ingeniosa y de interés teórica, no era una preocupación para las implementaciones prácticas ya que se basaban en violaciones de los principios básicos de una práctica criptográfica sólida. También argumentan que la falta de ejemplos seguros (que no violen principios básicos) en el ROM, pero no en la práctica, apunta a que cada vez más se debe de confiar en él ROM.

A pesar de la controversia generada alrededor de la utilización de ROM, las primitivas que han dado una prueba de su seguridad haciendo uso del ROM han tenido un gran éxito en el mundo real, hasta el punto de que casi todas los primitivas de clave pública utilizadas en la práctica lo utilizan. Además de utilizarlo como modelo de prueba también es el ROM utilizado como principio de diseño a la hora de crear una nueva primitiva criptográfica. Aun así, se encontraron ataques realizables en el mundo real a primitivas que habían probado su seguridad haciendo uso de ROM pero el problema estaba vinculado a la función de hash específica que utilizaban en lugar del ROM (en particular las funciones MD4 y MD5).

## 3.4. Criptografía de clave pública

A continuación se define una primitiva de clave pública.

**Definición 1** (Primitiva de clave pública)**.** *Una Primitiva*[2] *de clave pública es una tripleta* $(Gen, Enc, Dec)$ *de algoritmos en tiempo polinomial* [3]*, definidos de la siguiente manera:*

- $Gen(1^k)$*: Es un algoritmo generador de claves. Toma como entrada un elemento* $1^k$ *donde k es un parámetro de seguridad y retorna la tupla* $(K_p, K_s)$*, donde* $K_s$ *es la clave privada y* $K_p$ *es la clave pública.*

2 Primitivas: conjunto de algoritmos que se comportan de una manera relacionada.

3 Algoritmo de tiempo polinomial: es un algoritmo el cual cumple que para toda entrada la cantidad de operaciones ejecutadas está acotada por un polinomio. La variable de este polinomio está ligada con el largo de la instancia del problema a resolver.

- $Enc(m, K_p)$ : *Es un algoritmo de cifrado. Toma como entrada un mensaje $m$ y una clave pública $K_p$ y retorna un texto cifrado $c$.*
- $Dec(c, K_s)$ : *Es un algoritmo de descifrado. Toma como entrada un mensaje cifrado $c$, y una clave secreta $K_s$ y retorna $m$ o $\perp$. Donde $\perp$ simboliza "texto cifrado inválido".*

Las *blockchain* usan criptografía de clave pública para validar transacciones en la red. El uso de estas claves deben seguir las siguientes dos especificaciones:

- Los datos cifrados con una clave pública puede ser descifrados utilizando únicamente la clave privada asociada a dicha clave pública.
- Los datos cifrados con una clave privada deben de poder ser verificados utilizando la clave pública asociada a la clave privada.

Una propiedad importante que deben de cumplir todos los pares de claves generados por $Gen(1^k)$ es que derivar la clave privada con solo tener acceso la clave pública debe de ser costoso computacionalmente. Mientras que derivar la clave pública a partir de la clave privada debe de ser posible mediante un algoritmo eficiente. La clave pública está diseñada para ser compartida a través de la red, y por lo tanto puede estar expuesta a cualquier persona [48].

## 3.5. Firma digital

Las primitivas de firma digital, son la versión electrónica de las firmas manuscritas para documentos digitales: la firma de un usuario sobre un mensaje $m$ es un *string* que depende de $m$, de datos públicos y de datos privados específicos del usuario, y (posiblemente), también de datos elegidos aleatoriamente, de tal manera que cualquiera pueda verificar la validez de la firma usando sólo datos públicos. Los datos públicos de un usuario son conocidos como clave pública, mientras que los datos privados es lo que se conoce como clave privada. Obviamente, es fundamental poder evitar la falsificación de la firma de un usuario sin conocer su clave privada. Las siguientes definiciones están basadas en [49].

**Definición 2** (Primitiva de Firma digital)**.** *Una primitiva de firma digital se define de la siguiente forma (Figura 3.1).*

- $\mathcal{G}(1^k)$*: Es el algoritmo generador de claves.¿ Que dado un valor de entrada $1^k$, donde $k$ es un parámetro de seguridad, $\mathcal{G}$ produce el par $(K_p, K_s)$, que coinciden con la clave pública y privada. Se denota con el valor $n$ a la longitud de la clave pública.*
- $\Sigma(m, K_p, K_s)$*: Es el algoritmo de generación de firma. Que dado un mensaje $m$, y el par que coincide con la clave pública y privada $(K_p, K_s)$, $\Sigma$ produce una firma $\sigma$. En algunos esquemas puede recibir otras entradas también.*
- $\mathcal{V}(m, K_p)$*: Es el algoritmo de verificación de firma. Que dado una firma $\sigma$, un mensaje $m$, y una clave pública $K_p$, $\mathcal{V}$ verifica si $\sigma$ es una firma válida sobre $m$ con respecto a $K_p$.*

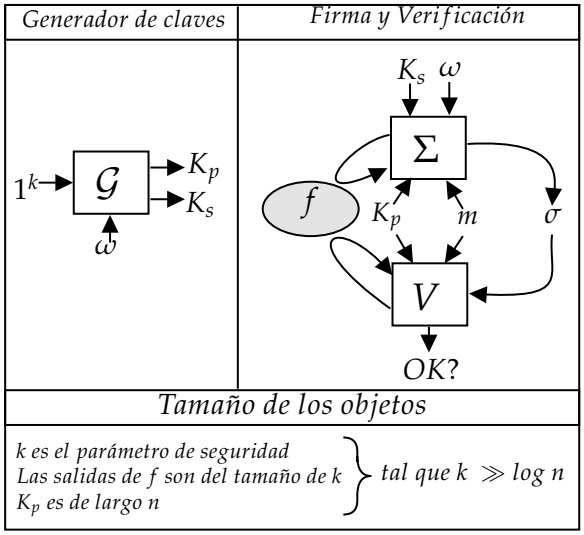


FIGURA 3.1: Esquema de firma digital

Uno de los criterios básicos de seguridad de un esquema de firma digital, es la propiedad de *Unforgeability* (no falsificación), que en palabras textuales, es la propiedad que implica que dado un documento firmado, al momento de ver la firma de ese documento, no es posible generar otro documento con la misma firma, a continuación se presenta la definición de *unforgeability*, basado en [50].

**Definición 3** (Propiedad Unforgeability (no-falsificación))**.** *La noción de seguridad estándar para firmas digitales exige que ningún usuario, después de interactuar arbitrariamente muchas veces con un firmante, y que k de estas interacciones fueron consideradas exitosas por el firmante, pueda producir más de k firmas. Además, el adversario puede programar e intercalar sus sesiones con el firmante de cualquier manera arbitraria.*

Para ayudar al lector, a entender un poco más dicha definición, a continuación se presenta el *game* de *unforgeability* [51]:

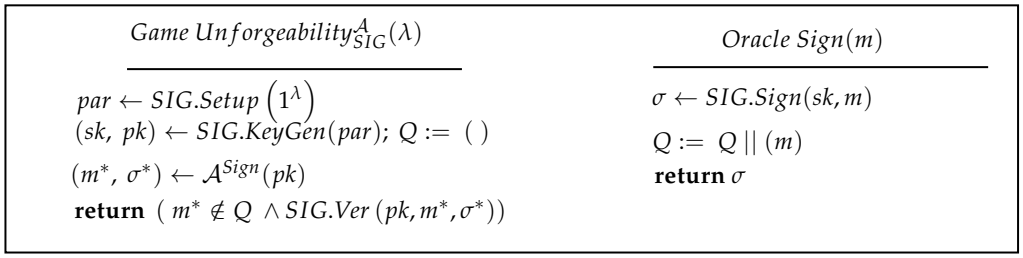


FIGURA 3.2: Game de la propiedad *unforgeability* para firmas digitales

### 3.5.1. Firma ciega

A continuación, se presenta otra primitiva criptográfica: los esquemas de firma ciega. En esta Sección se introduce particularmente el esquema en que se utilizará, en la Sección 3.5.3. Pero primero se ven algunas definiciones específicas sobre las propiedades de una firma ciega, relacionada con una configuración de dinero electrónico.

#### 3.5.1.1. Motivación

En el año 1982, David Chuam [52], fue pionero en un trabajo dirigido a crear una versión electrónica de dinero. Para alcanzar dicha meta, introduce la noción de *"coin"* y de *"randomized blind signatures"* (firma ciega aleatoria). Afirmó que esta era la única forma de garantizar el anonimato requerido: en la vida real, una moneda no se puede rastrear fácilmente desde que sale del banco hasta llegar a alguna tienda, además de que dos gastos de una misma persona no pueden vincularse entre sí. Estas son dos propiedades principales de las monedas reales que Chaum quería imitar: *untraceability* (imposibilidad de rastreo) y *unlinkability* (desvinculabilidad).

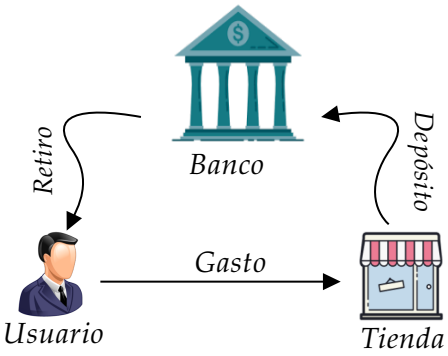


FIGURA 3.3: Moneda electrónica

Propuso definir una *electronic coin* (moneda electrónica) como un número con un certificado (una firma) producido por el banco. La cual es retirada del banco, gastado por el usuario y depositado en la tienda (ver Figura 3.3).

Describe luego *on-line electronic cash*, un esquema donde Chuam utiliza firmas ciegas para la producción de monedas. El usuario hace que el banco firme ciegamente una moneda. Entonces el usuario está en posesión de una moneda válida que el propio banco no puede reconocer ni vincular con el usuario. Cuando el usuario gasta la moneda, la tienda la devuelve inmediatamente al banco. Si la moneda ya se ha gastado, el banco detecta el hecho e informa a la tienda para que se niegue el pago. Es un contexto online: hay una comunicación continua entre la tienda y el banco para verificar la validez de las monedas. Para definir el esquema, Chaum introdujo el primer esquema de firma ciega, basado en RSA.

### 3.5.2. Esquema de firma ciega

Un criterio de calidad importante de los esquemas de firma ciega, es el número de interacciones necesarias que ocurren durante el protocolo de firma. En ellas se asume el *random oracle* (RO) y/o *common reference string* (CSR) configuradas a través de una *trusted third party* (TTP)[4], al que todas las partes deben tener acceso, o si sus pruebas se mantienen en el modelo estándar [5]. Los esquemas que solamente poseen dos movimientos de interacción son llamados como *round-optimal* [53]. Esta *round-optimality*, por un lado implica inmediatamente seguridad concurrente (que de lo contrario debe tenerse en cuenta por separado), y por otro lado, implica un

4 Tercera parte de confianza (TTP): En criptografía es una entidad que facilita las interacciones entre dos participantes, donde ambas partes confían en dicho tercer participante.

5 Modelo estándar: en criptografía se refiere a modelo estándar al modelo de computación en el que el adversario sólo está limitado por la cantidad de tiempo y el poder computacional disponible.

criterio crucial para la eficiencia de un esquema. Se puede ver un poco más de esto en [53, 54], donde se comenta un poco más sobre otros esquemas de firma ciega. A continuación se verá la definición de un esquema de firma ciega, basados en la definición de [55].

**Definición 4** (Firma ciega)**.** *Un esquema de firma ciega se define de la siguiente forma:*

- $\mathcal{G}(1^k)$*: Es el algoritmo generador de claves. Que dado un valor de entrada* $1^k$*, donde* $k$ *es un parámetro de seguridad,* $\mathcal{G}$ *produce el par* $(K_p, K_s)$*, que coinciden con la clave pública y privada.*
- $\Sigma(m, K_p, K_s)$*: Es el algoritmo de firma. Que es un protocolo interactivo, entre, el usuario que utiliza como entrada un mensaje* $m$*, y la clave pública del firmante, y el firmante que generó el par que coincide con su clave pública* $(K_p, K_s)$*. Al final de este protocolo, el firmante tendrá como salida "no-completado" o "completado", mientras que el usuario tendrá como salida el valor "fallido" o una firma* $\sigma$*.*
- $\mathcal{V}(m, K_p)$*: Es el algoritmo de verificación. Que dado una firma* $\sigma$*, un mensaje* $m$*, y una clave pública* $K_p$*,* $\mathcal{V}$ *prueba si* $\sigma$ *es una firma válida sobre* $m$ *con respecto a* $K_p$*.*

Un esquema de firma ciega es un proceso interactivo en donde participan un usuario y un firmante. En el primer paso, el usuario toma el mensaje $d$, realiza un cálculo a partir de ese mensaje para obtener un nuevo mensaje $d^*$, el cual es enviado al firmante. Luego el firmante utiliza su clave privada para generar una firma $\sigma^*$ para el mensaje $d^*$ y lo envía de vuelta al usuario.
El criterio básico de seguridad de un esquema de firma ciega son las propiedades de *Blindness* y de *Unforgeability* (esta última aplica de igual forma que con la firma digital convencional vista en 3). A continuación se introduce la definición de *Blindness* basado en [50].

**Definición 5** (Propiedad Blindness (cegado))**.** *Esta propiedad requiere que el firmante no pueda vincular un par mensaje/firma a una ejecución particular del protocolo de firma. Formalmente, el adversario escoge dos mensajes* $m_1$ *y* $m_2$*, y el experimento ejecuta el protocolo de firma actuando como el usuario con el adversario, obteniéndose así la firma* $\sigma_b$ *sobre el mensaje* $m_b$ *y la firma* $\sigma_{1-b}$ *sobre el mensaje* $m_{1-b}$*, siendo* $b$ *un valor aleatorio entre 0 y 1. Si ambas firmas son válidas, se le otorga al adversario las firmas* $(\sigma_0, \sigma_1)$ *y debe poder determinar el valor de* $b$*.*

Para ayudar al lector, a entender un poco más dicha definición, a continuación se verá el *game* de blindness [51]:

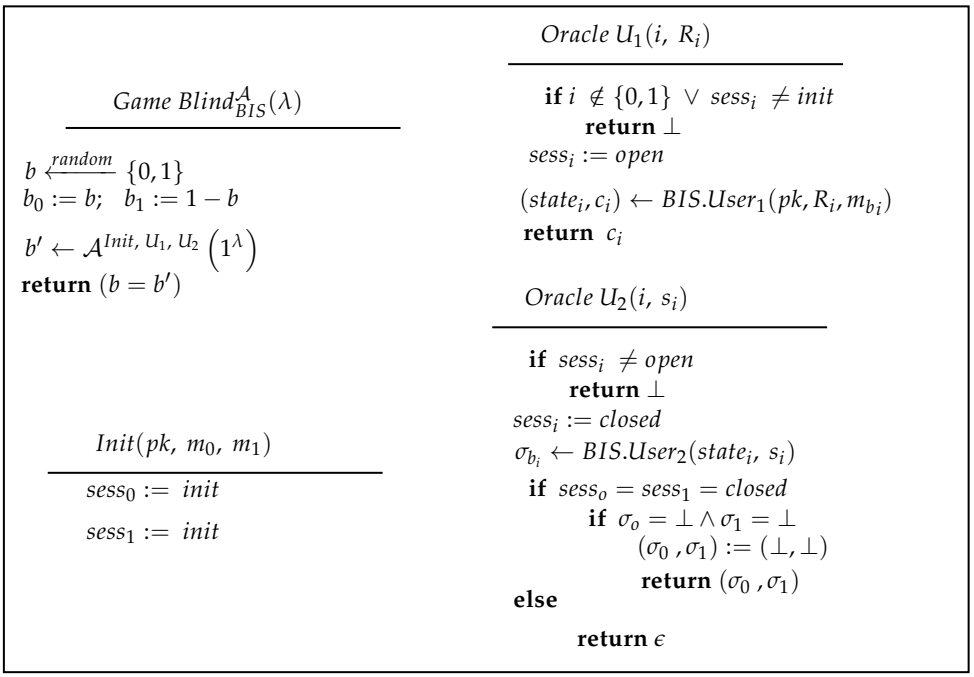


FIGURA 3.4: Game de la propiedad de *blindness*

### 3.5.3. Firma ciega de Okamoto-Schnorr

Esta primitiva fue introducida en 1992 por Okamoto [56] como una variante de la primitiva de firma ciega de Schnorr [57].

#### 3.5.3.1. Primitiva de firma ciega de Okamoto-Schnorr

Para un parámetro de seguridad $k$, se eligen los primos $p$, $q$ elegidos de manera tal que $q|(p-1)$ y $2^{k-1} < q \leq 2^k$. Por propiedad de cuerpos se tiene que $\mathbb{Z}_p^*$ es de orden $p-1$ y que existe un único subgrupo $G \subseteq \mathbb{Z}_p^*$ de orden $q$. Posteriormente se seleccionan dos elementos $g, h \in G$, generadores de $G$.

Sea $(r, s) \in (\mathbb{Z}_p^* \times \mathbb{Z}_p^*)$ la clave secreta del firmante, y sea $y := g^{-r}h^{-s}$ la correspondiente clave pública, generados mediante un algoritmo de generación $\mathcal{G}$.
El protocolo (Figura 3.5) por el cual un usuario obtiene una firma ciega de una autoridad firmante sobre un mensaje $m$ es el siguiente:

- La autoridad firmante selecciona dos elementos aleatorios $(t, u) \in (\mathbb{Z}_q^* \times \mathbb{Z}_q^*)$, computa y envía al usuario el *commitment* $a := g^t h^u$ mód $p$.
- El usuario selecciona $\beta, \gamma, \delta \in \mathbb{Z}_q$ de forma aleatoria y oculta $a$ en $\alpha := ag^\beta h^\gamma y^\delta$ mód $p$. Luego computa el desafío $\epsilon := \mathcal{H}(m||\alpha)$ y envía $e := \epsilon - \gamma$ mód $q$ a la autoridad.
- La autoridad firmante computa los valores $R := t + er$ mód $q$ y $S := u + es$ mód $q$ y envía el par $(R, S)$ que satisface $a := g^R h^S y^e$ mód $p$.
- El usuario computa los valores $\rho := R + \beta$ mód $q$ y $\sigma := S + \gamma$ mód $q$.
- Por último la firma ciega sobre el mensaje $m$, será la tupla $(m, \epsilon, \rho, \sigma)$.

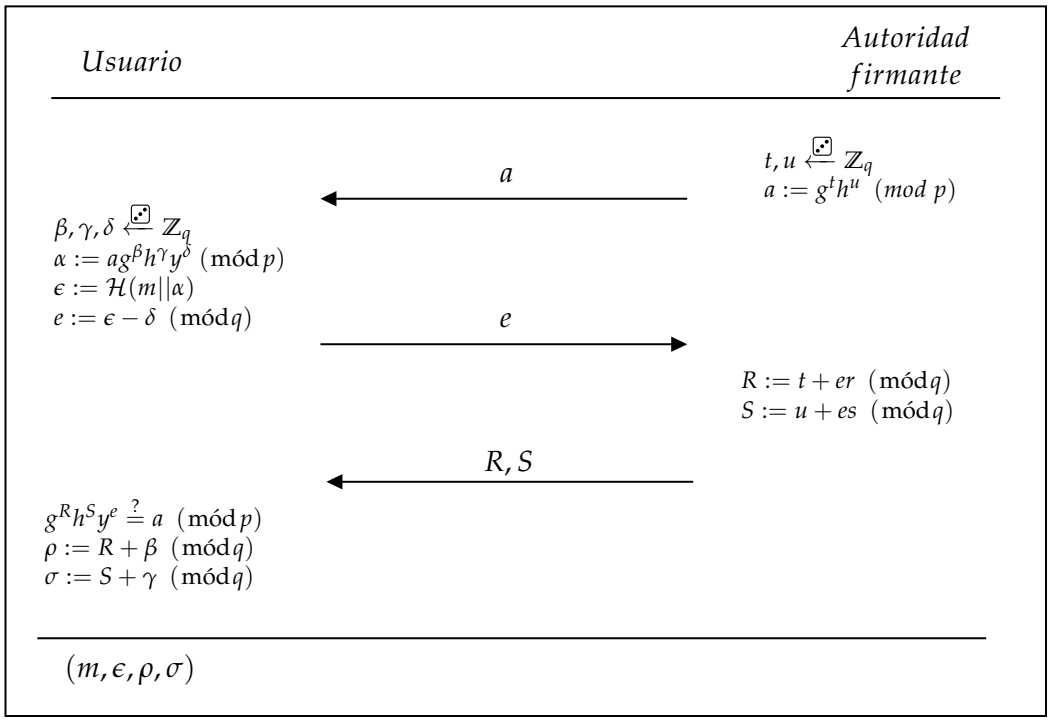


FIGURA 3.5: Primitiva de firma ciega de Okamoto-Schnorr

Para verificar la firma dado un mensaje $m \in \{0,1\}^*$ y una firma $(\epsilon, \rho, \sigma)$, se calcula que $\alpha = g^\rho h^\sigma y^\epsilon$ mód $p$ y luego se verifica que $\epsilon = \mathcal{H}(m||\alpha)$.

Si un firmante conoce otra representación $(r', s')$ de $y$ con respecto a $(g, h)$ y ejecuta el protocolo un vez $(r, s)$, y una vez usando $(r', s')$, entonces las dos vistas del verificador serán indistinguibles.

En [49], se prueba la propiedad de *unforgeability* y de *blindness* para este esquema.

## 3.6. Zero-knowledge

En esta sección se definen los esquemas de commitment y *zero knowledge proofs* (ZKP), que se utilizan en la estrategia empleada en el Capítulo 7.

### 3.6.1. Zero Knowledge Proof

El concepto de *Zero Knowledge Proof* (ZKP) (prueba de conocimiento cero), fue propuesto en 1989 por Goldwasser, Micali y Rackoff [58]. Esta primitiva criptográfica permite mostrar que cierta afirmación es verdadera respecto a datos (secretos), sin revelar ninguna otra información acerca de dichos datos más allá de la afirmación en sí misma. Desde entonces, el protocolo ZKP se convirtió en un importante campo de investigación, ya que proporciona una nueva caracterización de la clase de complejidad NP, usando los llamados "programas interactivos", y también porque es muy útil construir muchas primitivas criptográficas. Dado un elemento $x$ de un lenguaje $\mathcal{L} \in NP$, una entidad llamada *prover* es capaz de convencer a un *verifier* (verificador) de que $x$ de hecho pertenece a $\mathcal{L}$, es decir, existe un *witness* (testigo) $w$ para $x$. Una característica deseable de tales sistemas de prueba es la propiedad de

*succintness* (sucinticidad), lo que significa informalmente que el tamaño de la prueba es pequeño y, por lo tanto, permite que las pruebas puedan ser guardadas de forma completa en una *blockchain*. Al estar guardadas de forma completa en la *blockchain*, cualquier usuario podrá leer la prueba y verificar la correctitud de la misma.

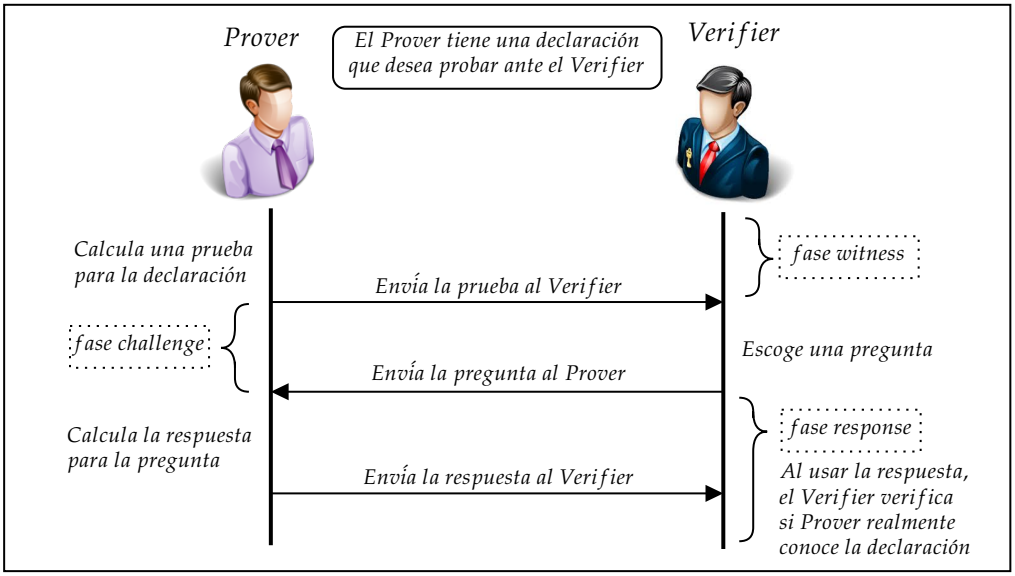


FIGURA 3.6: Esquema de una prueba de conocimiento cero interactiva (Σ-protocol 3-rondas)

Una prueba de conocimiento cero debe cumplir las siguientes propiedades:

- **Completeness**: Dado un *witness* $w$ que satisface la instancia $x$, se tiene que $Verify(Prove(x,w)) = 1$ $P[< Prove(w,z), Verify(z) > (x)] = 1$, donde $z$ es un input auxiliar tal que $z \in \{0,1\}^*$ [59].
- **Soundness**: Si el *witness* $w$ no satisface a $x$, entonces la probabilidad de $P([Verify(Prove(x,w)) = 1])$ es suficientemente chica.
- **Zero-Kwnoledge**: Dada una interacción entre el *prover* y el *verifier*, se nombra a esta interacción como *view*. Para capturar la propiedad de conocimiento cero, se utiliza un simulador de tiempo polinómico, que tiene acceso a la misma entrada dada al *verifier* (incluida su aleatoriedad), pero sin acceso a la entrada del *prover*, para generar una *view simulada*. Por lo que el esquema ZKP cumple ser:
  - *perfect zero knowledge* si la view sumulada, bajo la suposición de $x \in \mathcal{L}$, tiene la misma distribución que la view original.
  - *statical zero knowledge* si esas distribuciones son *estadísticamente cercanas*.
  - *computational zero knowledge* si no existe un distintivo en tiempo polinómico para esas distribuciones.

  Intuitivamente, la existencia de un simulador de este tipo significa que, independientemente de lo que el *verifier* pueda calcular a partir de la interacción con el *prover*, ya era posible calcular antes de dicha interacción, por lo tanto, el *verifier* no aprendió nada de él.

  En la figura 3.6 se ve una prueba de zero knowledge basada en Σ-protocol,

que para no entrar en detalles, se comenta que son protocolos sencillos en que las dos partes interactúan en tres etapas. Hay métodos bien estudiados y que son estándars para convertir un Σ-protocol en ZKP [60] [61].

### 3.6.2. Commitment

Un esquema de *commitment* criptográfico le permite a alguien calcular un valor que oculta un mensaje sin ambigüedad, en el sentido de que nadie podrá argumentar que este valor corresponde a un mensaje diferente luego de creado el *commitment*. En otras palabras, dada la imposibilidad de cambiar el mensaje oculto, se dice que el usuario se comprometió (committed) con ese mensaje.
El propósito de usar un esquema de compromiso es permitir al *prover* calcular pruebas de conocimiento cero donde el mensaje oculto es el *witness* subyacente $w$.

**Definición:** Un esquema de *commitment* se define por los algoritmos *Commit* y *Open* de la siguiente manera:

- $c = Commit(m, r)$. Dado un mensaje $m$ y un valor aleatorio $r$, se calcula como salida un valor $c$, que informalmente oculta el mensaje $m$ tal que es difícil calcular el mensaje $m'$ y el valor aleatorio $r'$ de forma que se satisfaga $Commit(m', r') = Commit(m, r)$. En particular, es difícil invertir la función *Commit* para obtener $m$ o $r$.
- $b = Open(c, m, r)$. Dado un *commitment* $c$, un mensaje $m$ y un valor aleatorio $r$, el algoritmo retorna *true* sí y solo sí $c = Commit(m, r)$.

Un esquema de *commitment* poseé dos propiedades:

- **Binding**: Dado un *commitment* $c$, es difícil calcular un par diferente de mensajes $m$ de valor aleatorio $v$ cuyo *commitment* es $c$. Esta propiedad garantiza que no hay ambigüedad en el esquema de *commitment*, y por lo tanto, después que $c$ se publica, es difícil abrirlo con un valor diferente. Esta propiedad protege al resto de los usuarios en caso de que el creador del *commit* quiera cambiar el mensaje original $m$.
- **Hiding**: Es difícil calcular cualquier información acerca de $m$ dado $c$. Esta propiedad protege al creador del *commit* del resto de los usuarios, ya que dado $c$, el resto de los usuarios no pueden reconstruir parcial o totalmente el mensaje $m$.

Para ayudar al lector, a entender un poco más dichas definiciones, a continuación se verán los *games* de la propiedad binding y de la propiedad hiding [59]:

| $Game\ Hiding^{\mathcal{A}}_{SIG}(\lambda)$ | $Game\ Binding^{\mathcal{A}}_{SIG}(\lambda)$ |
|---|---|
| $par \leftarrow SIG.Setup\left(1^{\lambda}\right)$<br>$(m_0,\ m_1,\ state) \leftarrow \mathcal{A}_1(par)$<br>$b \xleftarrow{random} \{0,1\};\ (c,d) \leftarrow Commit_{par}(m_b)$<br>$b' \leftarrow \mathcal{A}_2(c,\ state)$<br>**return** $(b = b')$ | $par \leftarrow SIG.Setup\left(1^{\lambda}\right)$<br>$(c,\ d,\ d') \leftarrow \mathcal{A}_1(par)$<br>$m \leftarrow Open(c,d);\ m' \leftarrow Open(c,d')$<br>**if** $(m = \perp\ or\ m' = \perp)$<br>**return** 0<br>**else**<br>**return** 1 |

FIGURA 3.7: Games de las propiedades de *hiding* y *binding* respectivamente

#### 3.6.2.1. Pedersen commitment

Un esquema muy conocido de *commitment* es el *Pedersen commitment* [62].
Dado un grupo $\mathbb{Z}_p$, de orden primo $p$, donde el problema de resolver el logaritmo discreto es difícil, el *commitment* se calcula de la siguiente forma:

$$c = Commit(m, r) = g^m h^r \;\; (g, h \;\; son \;\; generadores \;\; de \;\; \mathbb{Z}_p)$$

Para abrir dicho *commitment*, dado un mensaje $m$ y un valor aleatorio $r$, simplemente se recalcula y se compara el resultado con el valor de $c$. Una interesante propiedad que cumple el *commitment* de Pedersen es la de ser homomórfica (*homomorphic*). Significa que, si se toman dos mensaje arbitrarios $m_1$ y $m_2$ y dos valores aleatorios $r_1$ y $r_2$, tal que $c_1 = Commit(m_1, r_1)$ y $c_2 = Commit(m_2, r_2)$ respectivamente, entonces

$$c_1 c_2 = Commit(m_1 + m_2, r_1 + r_2)$$

El *commitment* de Pedersen es comúnmente implementado utilizando grupos bajo curvas elípticas en lugar de $\mathbb{Z}_p$. Además, es importante observar que si se conoce el logaritmo discreto de $h$ con respecto a $g$ entonces es realmente fácil generar $m'$ y $r'$ tal que $Commit(m', r') = Commit(m, r)$, rompiendo así la propiedad de binding (también, se puede decir que en ese sentido que el *commitment* de Pedersen es un *trapdoor commitment scheme* [63]).

### 3.6.3. Non-Interactive Zero Knowledge (NIZK)

Existen sistemas de zero knowledge proof que consisten en transmitir un único mensaje del *prover* al *verifier*, a estos se les llama no interactivos.

Un algoritmo no interactivo de conocimiento cero está definido por las fases *Setup*, *Prove* y *Verify*:

- *Setup*: es el algoritmo responsable de la generación de parámetros. Concretamente se tiene $params = Setup(\lambda)$, donde la entrada es el parámetro secreto $\lambda$ y la salida son los parámetros del algoritmo del sistema ZKP.
- *Prove*: es la sintaxis dada por $proof = Prove(x, w)$. El algoritmo recibe como entrada una instancia $x$ de algún lenguaje NP $\mathcal{L}$, el testigo $w$ que sirve para verificar que $x$ pertenece a $\mathcal{L}$. Se obtiene como salida la prueba de conocimiento zero.
- *Verify*: es el algoritmo que toma a la prueba ($proof$) como entrada y retorna como salida un bit $b$, el cual vale 1 en caso de que el *verifier* acepte la prueba.

Es importante remarcar que no todas los esquemas de ZKP son no interactivos. Al contrario, la mayoría de los protocolos ZKP descritos en la literatura son de hecho interactivos. En general, el *prover* debe responder los mensajes de desafío enviados por el *verifier* para convencerlo de que la prueba es válida. Esto requiere múltiples rondas de comunicación. En el contexto de aplicaciones *blockchain*, sería preferible evitar tantas interacciones, debido a que *(i)* los nodos validadores no pueden acordar adecuadamente cómo elegir esos desafíos, ya que en muchas construcciones se tienen que elegir aleatoriamente, mientras que el algoritmo de verificación debe ser determinista para llegar a un consenso; o *(ii)* porque haría muy pobre la complejidad de la comunicación del sistema.

## 3.7. Verifiable Random Function (VRF)

El concepto de *Verifiable Random Function* (función aleatoria verificable) fue introducido por Micali, Rabin y Vadhan [64]. Esta primitiva criptográfica es una función pseudoaleatoria que proporciona pruebas en donde sus resultados son verificables de forma pública. En resumen, asigna entradas a salidas pseudoaleatorias que son verificables. Las VRF se pueden usar para proporcionar *precommitments* deterministas que se pueden revelar más tarde utilizando pruebas. Las VRF son resistentes a los ataques de pre-imagen, a diferencia de la firma digital tradicional. Una VRF poseé los siguientes tres algoritmos:

- $KeyGen(r) \rightarrow (V_K, S_K)$. Este algoritmo de generación de clave, genera una clave de verificación ($V_K$) y una clave secreta ($S_K$) de una entrada aleatoria $r$.
- $Eval(S_K, m) \rightarrow (O, \pi)$. Este algoritmo de evaluación, toma de entrada una clave secreta ($SK$) y un mensaje $m$, y produce como salida un string pseudoaleatorio $O$ y una prueba $\pi$.
- $Verify(V_K, m, O, \pi) \rightarrow 0/1$. La función de verificación, toma de entrada la clave de verificación ($V_K$), un mensaje $m$, un string pseudoaleatorio $O$ y una prueba $\pi$. Y producirá la salida 1 si y sólo si verifica que $O$ es la salida producida por el algoritmo de evaluación asociado a la clave secreta de entrada $S_K$ y el mensaje $m$, de lo contrario, genera 0 como salida.

La construcción original propuesta era bastante ineficiente, pero más tarde Yevgeniy Dodis y Aleksandr Yampolskiy propusieron una VRF eficiente y realizable en la práctica, conocida como Dodis-Yampolskiy [65].

## 3.8. Terminología

En esta sección se brindan las definiciones referentes a la terminología utilizada sobre *anonymity*, *pseudonymity* y otros términos vinculados con la privacidad. La terminología que se define y las figuras presentadas son extraídas del artículo de Pfitzmann y Hansen [66]. Otras técnicas que no serán vistas en esta tesis pueden encontrarse de Zhang y Xue [67].

### 3.8.1. Terminología de privacidad

En esta sección se definen los términos *anonymity, unlinkability, unobservability y pseudonymity*.

Esta terminología está basada en la configuración habitual en la que los remitentes envían mensajes a los destinatarios utilizando una red de comunicación (como se muestra en la figura 3.8). Pero a efectos prácticos, otras configuraciones, por ejemplo, usuarios que consultan una base de datos o una *blockchain* por ejemplo, o clientes comprando en una tienda de comercio electrónico, se puede derivar la misma terminología abstrayendo nombres especiales "remitente", "destinatario" y "mensaje". Pero para facilitar la explicación, se utiliza la configuración habitual.

Más específicamente, se lo puede definir como un sistema. Para el propósito de este proyecto, dicho sistema tiene las siguientes propiedades relevantes:

1. El sistema tiene un entorno, es decir, partes del mundo están "fuera" del sistema. La conjunción del sistema y su entorno forman el universo.
2. El estado del sistema puede cambiar únicamente por acciones que ocurren dentro del sistema.

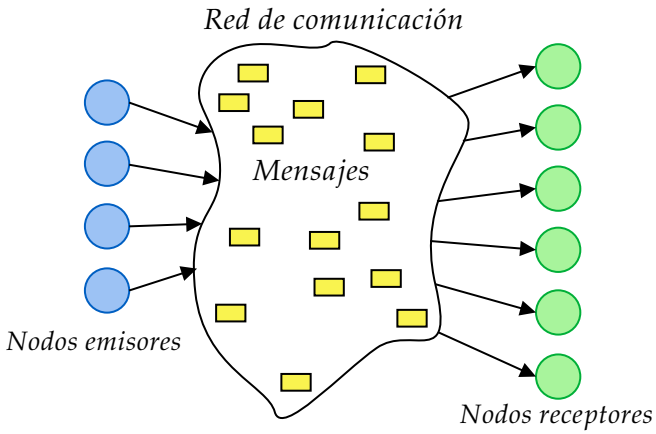


FIGURA 3.8: Red de comunicación

Todas las declaraciones se hacen desde el punto de vista de un atacante que puede estar interesado en monitorear qué comunicación está ocurriendo, qué patrones de comunicación existen, o incluso en manipular la comunicación. No solo se asume que el atacante puede ser un intruso que interviene en las líneas de comunicación, sino también un intruso participativo, es decir que participa en comunicaciones normales y controla al menos algunas estaciones. Se supondrá que el atacante utiliza todos los datos disponibles para inferir (probabilidades de) sus elementos de interés (por ejemplo quién envió o recibió qué mensajes).

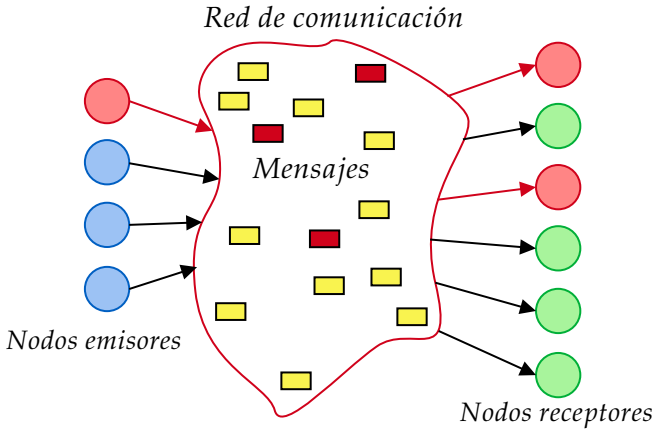


FIGURA 3.9: Red de comunicación (ej. de dominio de un atacante)

Para las siguientes definiciones se toma el supuesto de que el atacante podrá obtener información sobre el emisor o el receptor usando el contenido de los mensajes.

#### 3.8.1.1. Términos

– **Anonymity**: Para permitir que exista el anonimato de un sujeto, siempre tiene que haber un conjunto apropiado de sujetos que potencialmente poseen los

mismos atributos. Por lo tanto el anonimato es el estado de no ser identificable dentro de un conjunto de sujetos, el conjunto de anonimato.
El conjunto de anonimato es el conjunto de todos los sujetos posibles. Con respecto a las entidades actuantes, el conjunto de anonimato consiste en los sujetos que podrían causar una acción. Con respecto a los destinatarios, el conjunto de anonimato consiste en los sujetos que podrían abordarse. Por lo tanto, un remitente puede ser anónimo solo dentro de un conjunto de remitentes potenciales, su conjunto de anonimato de remitente, que puede ser un subconjunto de todos los sujetos en todo el mundo que pueden enviar mensajes de vez en cuando. Lo mismo es cierto para el destinatario, que puede ser anónimo dentro de un conjunto de destinatarios potenciales, que forman su conjunto de anonimato de destinatario. Ambos conjuntos de anonimato pueden ser disjuntos, ser los mismos o pueden superponerse. Además, se debe tener en cuenta que los conjuntos de anonimato pueden variar con el tiempo.

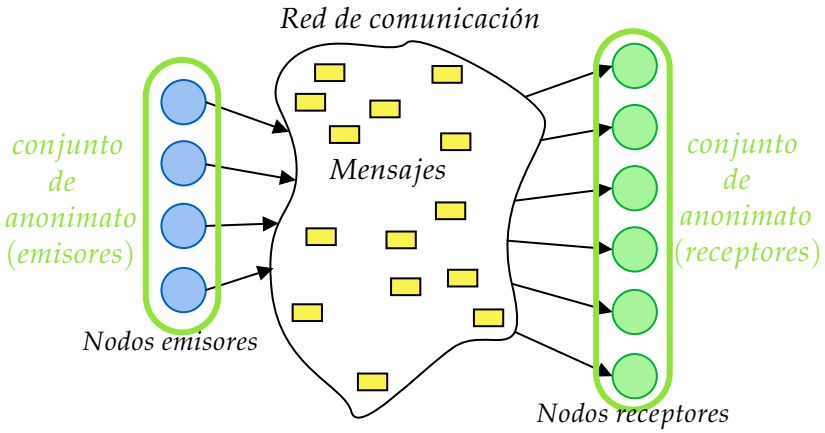


FIGURA 3.10: Red de comunicación (ej. de conjuntos de anonimato)

En igualdad de condiciones, el anonimato es más fuerte cuanto más grande es el conjunto de anonimato respectivo y más uniformemente distribuido está el envío o la recepción de mensajes y de los sujetos dentro de esos conjuntos.

De la discusión anterior se deduce que el anonimato en general, así como el anonimato de cada sujeto en particular, es un concepto que depende mucho del contexto (por ejemplo, población de sujetos, atributos, marco de tiempo, etc.). Para cuantificar el anonimato dentro de situaciones concretas, habría que describir el sistema con suficiente detalle, lo que prácticamente no es (siempre) posible para sistemas abiertos grandes (pero tal vez para algunas bases de datos pequeñas, por ejemplo). Además de la cantidad de anonimato proporcionada dentro de un entorno particular, hay otro aspecto del anonimato: su robustez. La solidez del anonimato caracteriza la estabilidad de la cantidad de anonimato frente a los cambios en el entorno particular (por ejemplo, considerando a un atacante más fuerte o con una probabilidad de distribuciones diferente) se podría utilizar la calidad del anonimato como un término que comprende tanto cantidad como robustez del anonimato.

– **Unlinkability**: La desvinculación de dos o más elementos de interés (por ejemplo, temas, mensajes, eventos, acciones) significa que dentro del sistema (que comprende estos y posiblemente otros elementos), desde la perspectiva del

atacante, estos elementos de interés no están más ni menos relacionados después de su observación de lo que están relacionados con su conocimiento anterior.

Esto significa que la probabilidad de que esos elementos estén relacionados desde la perspectiva del atacante permanece igual antes (conocimiento a priori) y después de la observación del atacante (conocimiento a posteriori del atacante)

- **Anonymity en términos de unlinkability**: Si consideramos enviar y recibir mensajes como elementos de interés, el anonimato puede definirse como la imposibilidad de vinculación de un elemento de interés y cualquier identificador de un sujeto (ID). Más específicamente, se puede describir el anonimato de un elemento de interés de modo que no se pueda vincular a ninguna ID, y el anonimato de una ID como no vinculable a ningún elemento de interés.

  Por lo tanto, se tiene que el anonimato del emisor como las propiedades de que un mensaje en particular no se puede vincular a ningún emisor, y que dado un emisor en particular no se le pueda vincular ningún mensaje. Lo mismo es cierto con respecto al anonimato del destinatario, lo que significa que un mensaje en particular no puede vincular a ningún destinatario y que dado un destinatario en particular no se le pueda vincular ningún mensaje.
  El anonimato de la relación significa que no se puede rastrear quién se comunica con quién. En otras palabras, el emisor y el destinatario (o los destinatarios en caso de una multidifusión) no se pueden vincular entre sí. Por lo tanto, el anonimato de la relación es una propiedad más débil que el anonimato del emisor y el anonimato del destinatario: puede rastrearse quién envía qué mensajes y también es posible rastrear quién recibe qué mensajes, siempre y cuando no haya vinculación entre ningún mensaje enviado y cualquier mensaje recibido y, por lo tanto, no se conoce la relación entre el remitente y el destinatario.

- **Unobservability**: La *unobservability* es el estado de los elementos de interés que no se pueden distinguir de ningún elemento de interés (del mismo tipo).

  Esto significa que los mensajes no son discernibles, es decir poseen "ruido aleatorio". Al igual que se tienen conjuntos de sujetos de anonimato con respecto al anonimato, se tienen conjuntos de objetos de *unobservability* con respecto a la *unobservability*.

  La *unobservability* del emisor significa que no se nota si algún emisor dentro del conjunto de *unobservability* envía mensajes. Análogamente la *unobservability* de los destinatarios refiere a que no se nota si algún receptor del conjunto de *unobservability* recibe mensajes.

  La *unobservability* de una relación significa que no se nota si se envía algo de un conjunto de posibles emisores a un conjunto de posibles destinatarios. En otras palabras, no se nota si dentro del conjunto de *unobservability* de la relación de todos los posibles pares de emisor-receptor, se cambia un mensaje en cualquier relación.

- **Relación entre términos**: Con respecto al mismo atacante, la *unobservability* revela siempre solo un verdadero subconjunto de la información que revela el

anonimato. Se puede usar la notación abreviada:

$$\text{unobservability} \rightarrow \text{anonimato}$$

Esta notación $\rightarrow$ se leé "implica que".
Utilizando esta mismo argumento y notación, se tiene que:

unobservability del emisor $\rightarrow$ anonimato del emisor
unobservability del destinatario $\rightarrow$ anonimato del destinatario
relación de unobservability $\rightarrow$ relación de anonimato

Como se señaló anteriormente, se tiene también que:

anonimato del emisor $\rightarrow$ relación de anonimato
anonimato del destinatario $\rightarrow$ relación de anonimato

unobservability del emisor $\rightarrow$ relación de unobservability
unobservability del destinatario $\rightarrow$ relación de unobservability

– **Pseudonymity**: Refiere al uso de pseudónimos. Los pseudónimos son identificadores de sujetos, en nuestro entorno de emisor y destinatario. El sujeto al que se refiere el seudónimo es el titular del seudónimo. Ser seudóanonimo es el estado de usar un seudónimo como ID.

  En nuestro entorno habitual se supondrá que cada seudónimo se refiere a exactamente un titular, invariable con el tiempo, y que no se transfiere a otros sujetos. Tipos específicos de pseudónimos pueden extender esta configuración: un seudónimo grupal se refiere a un conjunto de titulares, es decir, puede referirse a múltiples titulares; un seudónimo transferible puede transferirse de un titular a otro y convertirse en su titular.

  Un pseudónimo grupal puede inducir un conjunto de anonimato: utilizando la información provista únicamente por el pseudónimo, un atacante no puede decidir si una persona específica del grupo realizó una acción.

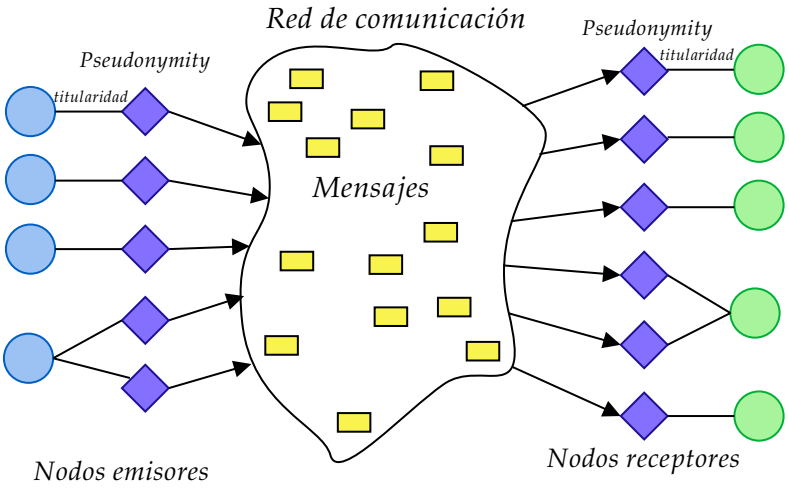


FIGURA 3.11: Red de comunicación (ej. de *pseudonymity*)

Los pseudónimos transferibles pueden, si el atacante no puede monitorear completamente todas las transferencias de titularidad, cumplir el mismo propósito, sin disminuir la responsabilidad como lo ve una autoridad que controla todas las transferencias de titularidad.

# Capítulo 4

# Fundamentos de Blockchain

En este Capítulo se presentan los fundamentos sobre los cuales se construye *blockchain*, y que serán indispensables para la comprensión de este proyecto. A modo de introducción, se toma la implementación de *blockchain* utilizada en *Bitcoin*. Posteriormente, se presenta el concepto más general de las tecnologías de registros de transacciones distribuidas (DLT), incluyendo una clasificación de esta tecnología. Después se detalla el ciclo de vida de las transacciones en estas tecnológicas, y que incluye la descripción de los algoritmos de consensos, que determinan cuándo una transacción se considera válida.

## 4.1. Visión general

*Blockchain*, es un término que está generando mucha expectativa en los últimos tiempos. Se puede decir que su popularidad encuentra su origen en el año 2008, año en el que se envía un mensaje a una lista de correo firmado por Satoshi Nakamoto y titulado "Bitcoin: A Peer-to-Peer Electronic Cash System"[1]. A este artículo se lo conoce como la piedra fundacional de la criptomoneda *Bitcoin* y de todas las criptomonedas posteriores. Sin embargo, la creación de *blockchain* se remonta al año 1991 donde Herber y Stornetta [68] presentaron una solución computacionalmente práctica para los documentos digitales con *timestamp* (sello de tiempo) para que no puedan ser modificados o manipulados.

La idea original de Bitcoin era la de resolver la pregunta de “¿Es posible implementar un sistema electrónico de dinero usando una arquitectura *peer-to-peer*[1] sin intermediarios, de forma tal que sea equivalente al dinero en efectivo?"[70].

Si bien el concepto de la tecnología *Blockchain* comienza a tomar fuerza con el desarrollo de criptomonedas, el rango de aplicaciones posibles se ha ampliado con el paso del tiempo, por ejemplo, la tecnología se ha utilizado para atacar otros tipos de problemas (tales como la salud, gestión de stock y propiedades, verificación de identidad y transparencia en la administración pública [71], entre otros.).

En el esquema tradicional las transacciones se centralizan a través de organizaciones que consideramos confiables, por ejemplo, cuando alguien se recibe o gradúa, su empleador solicita una certificado oficial (o título) a la universidad en donde dicha persona estudió como prueba de que verdaderamente ha culminado sus estudios. En este caso, se considera a la universidad como un intermediario confiable entre el

1 *Peer-to-Peer*: Es una arquitectura de red, en donde existe una mínima (o ninguna) dependencia de una infraestructura de servidores que deban mantenerse siempre activos. En su lugar, la aplicación explota la comunicación directa entre parejas de *hosts* conectados de forma intermitente, conocidos como *peers* (pares) [69].

graduado y el empleador, en donde su función es la de garantizar que la información que contiene el certificado es precisa y veraz. ¿Y por qué el empleador no le pide directamente al estudiante que proporcione una copia de su registro escolar? La razón es la confianza, ya que el candidato puede modificar el contenido.

## 4.2. Distributed Ledger Technology

Un registro de transacciones distribuido o en inglés *Distributed Ledger Technology (DLT)* es un tipo de estructura de datos que se replica, comparte y sincroniza entre múltiples dispositivos que pueden operar geográficamente distantes entre sí. La principal característica que comparten estos tipos de estructura de datos, es la posibilidad de almacenar de forma pública, permanente y simultánea los datos introducidos en un programa que comparten un grupo de personas o instituciones, haciendo accesible la información para todos ellos.

Además, estas tecnologías pueden registrar activos virtuales de forma descentralizada, ayudándose de la criptografía para realizar y verificar las modificaciones que se producen en la base de datos y efectuar de manera segura transacciones entre dos participantes.

La sincronización de los datos, se logra mediante el uso de algoritmos de consenso entre los participantes que tienen una copia de la *ledger* (registro inmutable de transacciones). En este contexto, el consenso refiere a un protocolo que garantiza que los distintos participantes acuerdan un estado específico del sistema como válido [72].

Podemos decir que una *ledger* distribuida se basa en tres componentes [73]:

- Un modelo de datos que capture el estado actual de la *ledger*.
- Un lenguaje de transacciones que puede cambiar el estado de la *ledger*.
- Un protocolo usado para generar consenso entre los participantes sobre qué transacciones serán aceptadas, y en qué orden se registran en la *ledger*.

Se han definido diferentes tipos de *ledger* distribuidas en los últimos años, siendo *Blockchain* el más destacado. Una *Blockchain* es una *ledger* de *append-only* [2] organizada como una cadena de bloques que se basa en una red P2P para realizar su gestión, actualización y sus operaciones. En términos generales, los bloques simplemente funcionan de contenedores para las transacciones, y se pueden vincular a una cadena de bloques ya existente por medio de punteros.

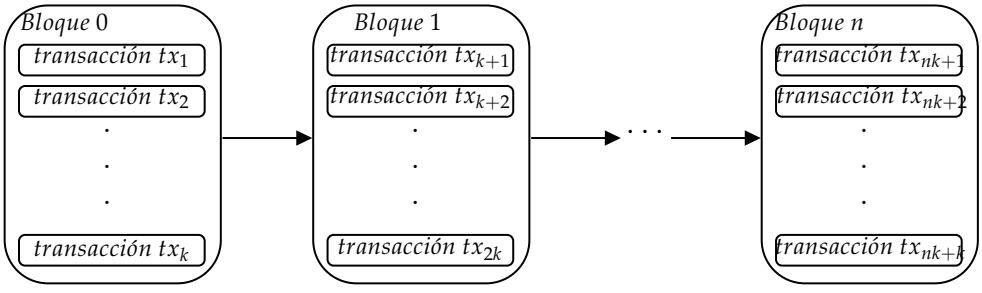


FIGURA 4.1: Diagrama de Blockchain

2 Apped-only: Refiere al hecho de que únicamente concatena o agrega cosas, ni quita ni modifica cosas que ya han sido agregadas.

El primer bloque se le llama bloque génesis (En la ilustración 4.1, es el bloque 0). Cada nuevo bloque se agrega encima de los bloques existentes formando una estructura de lista encadenada. Antes de ser agregado, el bloque es validado por la red por medio del algoritmo de consenso. El algoritmo de consenso define el proceso que permite decidir qué nodo propondrá el nuevo bloque, y si es válido o no. Dependiendo de un algoritmo de consenso, algunos nodos pueden no estar de acuerdo, y dicho no determinismo de los algoritmos utilizados lleva a que en cierto instante de tiempo dos participante de la red tengan una visión diferente del orden aceptado de las transacciones, por lo que la blockchain podrá tener una o varias bifurcaciones [3], como se ilustra en la Figura 4.2. Luego de elegido el próximo bloque a agregar en la blockchain, se comunica el mismo a todos los nodos de la red por medio de un mensaje de *broadcast*. Los nodos de la red seguirán la regla de la cadena más larga [4] para decidir en cuál de las bifurcaciones agregar el nuevo bloque. Al existir estas bifurcaciones no se puede considerar como válida a una transacción de forma inmediata. Lo habitual es esperar seis confirmaciones luego de añadido el bloque que contiene una transacción para considerarla como válida.

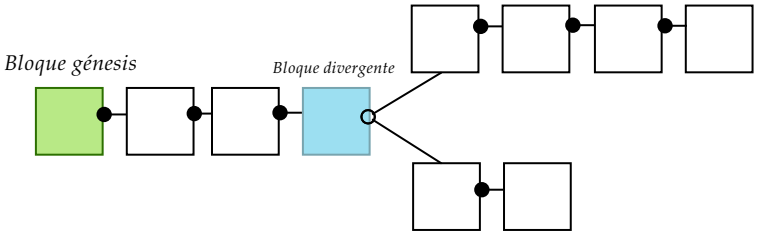


FIGURA 4.2: Bifurcación en Blockchain

Como estructura de datos, *blockchain* tiene dos características distintivas que son los *timestamp* (marca de tiempo), explicados en la sección 4.8 y la vinculación que hay mediante un puntero de hash entre cada bloque de la cadena y el anterior, de tal manera que cualquier modificación realizada en un bloque obliga a la regeneración de los siguientes bloques de la cadena. Haciendo uso de los *timestamp* y los punteros de hash, se proporciona la capacidad de detectar las manipulaciones en una *blockchain*, esta propiedad conocida como inmutabilidad es lo que hace a la estructura de datos tener un alto nivel de seguridad. A su vez esto hace que *blockchain* sea adecuada muchos escenarios donde es necesario compartir datos de forma pública pero sin que sea posible editarlos, por ejemplo para la transferencia de propiedades de activos.

## 4.3. Tipos de blockchains

Otro aspecto relevante que define una *blockchain* son los permisos que son necesarios para que nuevos participantes se unan a la red, o no. En este sentido, una

3 Las bifurcaciones ocurren cuando el software de diferentes mineros se desalinea. Depende de los mineros decidir qué blockchain seguir usando. Si no hay una decisión unánime, esto puede resultar en la creación de dos versiones de la cadena de bloques.

4 Regla de la cadena mas larga: Refiere a la cantidad de cómputo requerido para la creación de los bloques que contienen a la bifurcación. Para cada bloque existe una dificultad asociada que determina la cantidad de computo que es necesaria para encontrar el *nonce* que hace válido al bloque. Debido a esto, puede llegar a darse el caso de que bifurcaciones con menos bloques sean consideradas las más largas.

*ledger* distribuida puede ser *permissionless* (privada o pública, en donde todos pueden unirse sin restricciones) o *permissioned* (privada o pública, donde se necesitan de permisos para unirse) [72].

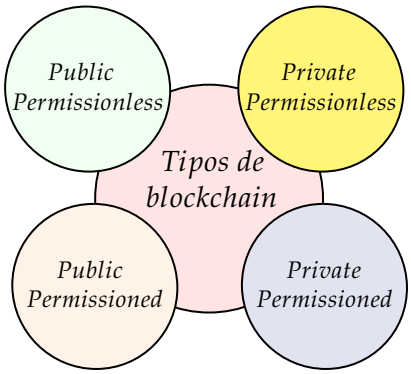


FIGURA 4.3: Tipos de blockchain

En la figura 4.3 se ven los cuatro tipos de *blockchain* existentes las cuales se caracterizan por dos aspectos, el primero la necesidad de permisos para ingresar a la plataforma y el segundo la necesidad de permisos para participar del algoritmo de consenso. A continuación, se describen las características fundamentales de cada uno de los tipos de *blockchain* existentes:

- *Public Permissionless*: en este tipo de *blockchain* cualquier participante puede unirse o irse de la red en el momento que desee, como también puede participar en la ejecución del algoritmo de consenso. Todos los participantes de la plataforma tienen permiso de leer o escribir en la *blockchain*. Esto hace que no pueda asumirse honestidad en el accionar de los participantes. Bitcoin y Ethereum son ejemplos de plataformas que hacen uso de este tipo de *blockchain*.

- *Private Permissionless*: es una *blockchain* donde cualquier participante puede leer su contenido pero solo un conjunto autorizado de nodos pueden crear nuevas transacciones o participar del algoritmo de consenso. En este tipo de plataformas existe un administrador que es el encargado de otorgar los permisos de escritura o participación del algoritmo de consenso, esto hace que la red no sea totalmente descentralizada. Ripple [74] o Libra [75] son ejemplos de plataformas que hacen uso de este tipo de *blockchain*.

- *Public Permissioned*: son una implementación híbrida entre una *ledger permissionless* y una privada. Esto funciona a nivel de consenso: en una *public permissioned*, un grupo preseleccionado de participantes controla el proceso de consenso, pero también otros participantes pueden participar en la creación de nuevas transacciones y/o revisarlas. La configuración específica de cada *consortium blockchain* (es decir, qué participantes tienen el poder de autorizar transacciones a través del proceso de consenso, que participantes puede revisar el historial de la cadena, que participantes pueden crear nuevas transacciones y más) es una decisión de cada consorcio individual. La plataforma de *blockchain* de este tipo que ha tenido mayor trascendencia es Fabric [76].

- *Private Permissioned*: es una *blockchain* donde cada participantes de la red debe estar inscrito en una CA antes de unirse a la *blockchain*. Los permisos de lectura pueden ser públicos o restringidos de forma arbitraria. Las aplicaciones relacionadas incluyen gestión de bases de datos, auditoría, etc., internas de una

sola empresa, por lo que la legibilidad pública puede no ser posible en muchos casos. Aunque existen casos donde se desea que la auditabilidad sea pública. Un ejemplo de plataforma *blockchain* que sigue este enfoque es Corda [77].

## 4.4. Transacciones en Bitcoin

Dentro de una plataforma descentralizada que utiliza *blockchain*, una transacción es el proceso de transferir la propiedad de los activos que se manejan en esa plataforma entre dos participantes. El proceso de validación está a cargo de los nodos que ejecutan el algoritmo de consenso. Los nodos reciben nuevas transacciones y verifican que sean válidas, en caso de ser válidas, las agregan al nuevo bloque de transacciones que están construyendo como se ilustra en la figura 4.4. Al llegar al tamaño máximo de bloque que permite la plataforma, propondrán a la red este bloque como el próximo a agregar a la *blockchain*.

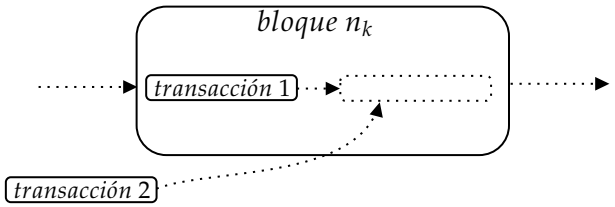


FIGURA 4.4: Cadena de transacciones

En la figura 4.5 se ilustra cómo se inicializan transacciones en Bitcoin:

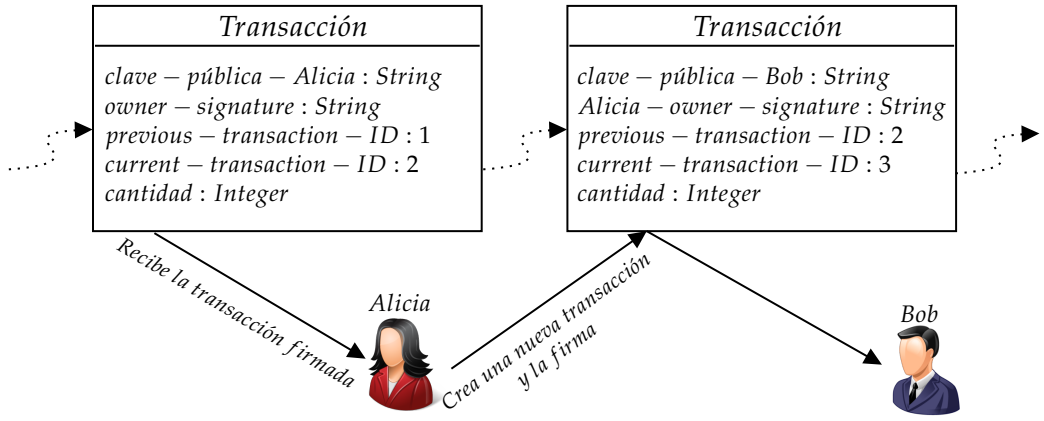


FIGURA 4.5: Mecanismo de una transacción

En la figura 4.5, Alicia recibe una cantidad de monedas de *Bitcoin* como resultado de una transacción previa. Entonces, ella decide transferir esas monedas a Bob. Él debe validar que Alicia fue quien envió este crédito, para mantener el no repudio de la red. Además, Bob necesita rastrear la fuente del crédito de Alice para mantener la autenticidad de la red. Para lograr estos procesos, la plataforma utiliza criptografía de clave pública. Alicia crea el *payload* (carga útil) de la transacción la cual contiene los siguientes campos:

- *clave − pública − Bob*: es un campo para designar a Bob como el nuevo propietario auténtico de las monedas transferidas.

- $Alicia - owner - signature$: Permite designar a Alicia como la anterior propietaria auténtica de las monedas transferidas.
- $previous - transaction - ID$: es el orden de la transacción anterior que contiene el origen de las monedas de Alicia.
- $current - transaction - ID$: indica el orden de la transacción actual dentro de la lista de transacciones.
- $cantidad$: es el monto que indica cuántas monedas deben ser transferidas.

A grandes rasgos, cada transacción apunta a la anterior utilizando los campos $previous - owner - signature$ y $previous - transaction - ID$. Esto crea una lista de transacciones rastreables y vinculables.

## 4.5. Saldos de cuentas

Al igual que en cualquier sistema bancario, en cualquier plataforma que utilice *blockchain* como base de datos, es necesario devolver al usuario la cantidad de elementos que le pertenecen. Por ejemplo, en una plataforma de criptomonedas que hace uso de *blockchain* para registrar las transacciones del sistema, es necesario presentarle al usuario cuantas monedas posee en todo momento. Para resolver este problema sean propuestos dos métodos, el primero llamado **Modelo UTXO** y el segundo llamado **Modelo de cuentas**.

### 4.5.1. Modelo UTXO

En el modelo UTXO (*unspent transaction outputs*) cada transacción consiste en una lista de entradas y una lista de salidas, donde las salidas representan un valor específico que está disponible para ser utilizado como entrada de transacciones posteriores. Cada salida puede utilizarse en exactamente una entrada.

Además, no se admiten ciclos en estas conexiones y, por lo tanto, podemos considerar una colección de transacciones que se utilizan entre sí como un grafo acíclico dirigido, donde una transacción con $m$ entradas y $n$ salidas está representada por un nodo en el grafo con $m$ aristas hacia adentro y $n$ arista hacia afuera. La suma de los valores consumidos por las entradas de una transacción debe ser igual a la suma de los valores proporcionados por sus salidas. Por lo que el valor total se conserva (es decir, sale la misma cantidad que la que entra) [78]. *Bitcoin* sigue este modelo para mantener el balance de las cuentas de los usuarios.

### 4.5.2. Modelo de Cuentas

En el modelo basado en cuentas se manejan los saldos de cada participante de la misma forma que en un banco, donde un usuario es el titular de una cuenta, y en esa única cuenta se guardan todos sus activos. Es decir, este modelo maneja una estructura de datos (que es mantenida en memoria principal) donde cada participante de la red tiene asignada una cuenta en la cual puede retirar o depositar fondos. Las transacciones en este modelo se representan mediante acciones de transferir una cantidad de una cuenta hacia otra. Por otro lado, al almacenarse el saldo en un único registro en lugar de múltiples *UTXO* implica que el modelo de cuentas terminará necesitando menos espacio en memoria para mantener el saldo de los participantes

y menos espacio de almacenamiento. Ya que el registro de transacciones únicamente representa el movimiento de una cuenta hacia otra sin tener que especificar múltiples direcciones de entrada y salida. *Ethereum* sigue este modelo para mantener el balance de las cuentas de los usuarios.

## 4.6. Consenso

Cachin *et al.* [79] definen a *blockchain* como una "ledger distribuida para registrar transacciones, mantenida por muchos nodos y sin una autoridad central a través de un sistema criptográfico distribuido". En la ilustración 4.5, Alicia le envía a Bob una transacción, sin embargo, el sistema utilizado debe asegurarse de que Alicia sea un participante honesto antes de procesar esta transacción. En un sistema con cientos de participantes, *blockchain* debe aplicar un protocolo de consenso de tolerancia a fallas para garantizar que los participantes sean honestos para eliminar el efecto negativo de los nodos maliciosos o deshonestos. Por ejemplo, generalmente se asume que para una *ledger* con $n$ nodos en total sea confiable se establece que no más de un tercio o la mitad del total de los nodos están defectuosos ($f$) en una *ledger* [80] como se explica en la ecuación 4.1:

$$f < \frac{n}{k} \;\; para \;\; quórum \;\; k = 2, 3 \tag{4.1}$$

Podemos ver que en la ecuación 4.1, cuando el quórum, $k = 2$, entonces $f < \frac{n}{2}$, el número de nodos fallidos es menos que la mitad del total de nodos. Además, cuando el quórum, $k = 3$, entonces $f < \frac{n}{3}$, y el número de nodos fallidos es menos que un tercio del total de nodos. Por lo tanto, el número de nodos honestos ($h$) en una *ledger* puede ser calculado con la ecuación 4.2.

$$h = n - f \;\; (nodos \;\; que \;\; son \;\; honestos) \tag{4.2}$$

Un protocolo de consenso es una serie de procesos realizados de forma distribuida y asincrónico para lograr un acuerdo entre los nodos para cualquier transacción presentada. Es un proceso de toma de decisiones en donde los nodos respaldan a la mejor decisión para la *blockchain*. Más precisamente, un protocolo de consenso debe de asegurar las siguientes propiedades:

- Validez (*validity*): si un nodo honesto $h$, envía un broadcast de un mensaje $m$, entonces $h$ entrega eventualmente el mensaje $m$ a sí mismo.
- Acuerdo (*agreement*): si algún nodo honesto entrega un mensaje $m$, entonces $m$ es eventualmente avalado por todos los demás nodos honestos.
- Integridad (*integrity*): un nodo honesto realizará un duplicado del broadcast de un mensaje y, por lo tanto, el mensaje recibido, $m$, es idéntico al mensaje que se envió.
- Seguridad (*safety*): cada nodo de la *ledger* tiene garantizado la misma secuencia de valores de entrada y obtiene como resultado los mismos valores de salida que el resto de los nodos. Esta propiedad promueve un servicio de replicación donde eventualmente cada nodo tiene una copia actualizada de todos los mensajes para preservar la consistencia en la *ledger* [81].

Esto lleva a que este tipo de blockchains se basan en algoritmos de consenso probabilísticos. Esto algoritmos garantizan la coherencia de la *ledger* con un alto grado

de probabilidad siempre y cuando la cantidad de nodos maliciosos no superen el 50 % [5].

### 4.6.1. Crash fault tolerance (CFT)

Para comprender esta sección, y las secciones 4.6.2 y 4.6.2.1 es importante comprender que en una red de comunicación abierta, existirán nodos que se pueden considerar honestos o maliciosos. Un nodo malicioso es un nodo que puede desviarse arbitrariamente del protocolo. Dicho nodo podría realizar acciones que perjudican al sistema, por ejemplo, reteniendo mensajes que debiera enviar, enviando mensajes con el objetivo de obtener información que no debiera obtener si siguiera lo establecido por el protocolo. Mientras que se dice que un nodo es honesto siempre que actúa de acuerdo al protocolo establecido.
Hay un híbrido entre honesto y malicioso que se llama *"honest-but-curious"* (honesto pero curioso) que son aquellos que siguen el protocolo pero sí pueden aprovechar algún tipo extra de información para hacer algo malo lo hacen.
*Crash fault tolerance* (CFT), en términos de *blockchain*, es una propiedad que cumplen algunos de los algoritmos de consenso, y que indica que existe algún grado de resistencia a la caída de los nodos (es decir un nodo que se desconecta de la red), o nodos con datos corruptos, pero esta propiedad no agrega ninguna resistencia a la aparición de nodos maliciosos en la red. Un algoritmo que cumple la propiedad de CFT, puede realizar correctamente el proceso a pesar de que determinada cantidad de componentes fallen sin que sean maliciosos, e igualmente llegar a un consenso. Al ejecutarse dentro de una misma empresa, en donde los nodos suelen tener objetivos similares, un algoritmo de consenso basado en la propiedad de CFT es una buena solución cuando falla uno o varios componentes del sistema. Aunque es bueno destacar que esta idea cambia cuando se trata de un ámbito empresarial con distintas empresas que tengan objetivos diferentes entre sí, o yendo más lejos, que tengan objetivos contrapuestos.

### 4.6.2. Byzantine fault tolerance (BFT)

*Byzantine fault tolerance* (tolerancia a faltas bizantinas) es otra propiedad, que cumplen aquellos algoritmos que se defienden ante fallas bizantinas, mitigando la influencia que los posibles nodos maliciosos puedan tener sobre el funcionamiento correcto del sistema. Este tipo de fallas fue descrito en 1982 por Lamport *et al.* [82]. BTF es un derivado del problema de los generales bizantinos [6].

#### 4.6.2.1. Practical byzantine fault tolerance (PBFT)

El algoritmo de consenso *Practical byzantine fault tolerance (PBFT)* fue presentado por Castro *et al.* [83] en 1999. Este algoritmo está diseñado para funcionar en sistemas asincrónicos y está desarrollado para ser de alto rendimiento y con un tiempo de ejecución elevado además de un ligero aumento en la latencia, mejorando algunos de los problemas nativos de los algoritmos basados en BFT. Entre ellos su ineficiencia en la práctica, o del hecho de que asumen sincronización (debido a que se basan en

5 *Ataque del 51 %*: Este ataque permite que uno o varios agentes maliciosos que controlen al menos el 51 % de la red puedan hacer con ella lo que deseen.

6 *Problema de los generales bizantinos*: Es un problema de lógica, el cual plantea que en un asedio hay un número indeterminado de generales que debe de coordinarse para la conquista. Sólo uno cursa la orden (que es binaria, atacar o retirarse), el comandante, el resto son tenientes. Puede haber uno o más traidores (comandante incluido), con el objetivo de que no se cumpla la orden.

los límites conocidos de velocidad y de delay de los mensajes). Todos los nodos en el esquema PBFT se ordenan en una secuencia con un nodo que es el nodo *leader* (líder, o también llamado primario).

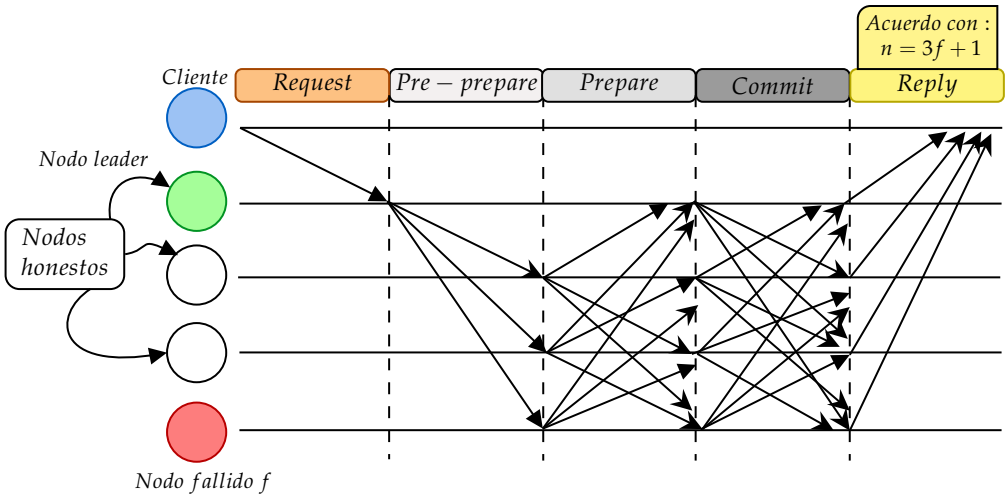


FIGURA 4.6: Algoritmo de consenso PBTF

La ilustración de la figura 4.6, muestra que cada ronda del consenso en PBFT consiste en cinco fases que se ejecutan en secuencia de la siguiente manera:

- **Request**: el cliente envía a los nodos *leaders* un pedido para invocar una operación de servicio.
- **Pre-prepare**: los nodos *leaders* realizan un multicast de request a todos los demás nodos.
- **Prepare**: los nodos realizan un multicast de la request para garantizar que un nodo *leader* no defectuoso esté de acuerdo con el total de la orden de solicitud.
- **Commit**: los nodos *leaders* realizan el commit de los request y se la envían a todos los demás nodos para lograr el acuerdo de propiedad de consenso.
- **Reply**: sí los nodos llegan a un acuerdo, responden al cliente con su aprobación.

El cliente espera por $f + 1$ respuestas de diferentes nodos con el mismo resultado ($f$ representa el número máximo de nodos que pueden estar defectuosos). Esto elimina efectivamente los intentos de defraudar al sistema ya que el cliente espera que al menos un nodo honesto vote sobre la integridad de la decisión que toman todos los nodos defectuosos.

### 4.6.3. Proof of work (PoW)

Este algoritmo de consenso fue desarrollado por Dwork y Naor en 1993 [84]. Como se describe en ese artículo, la idea original de esta algoritmo de consenso era prevenir los abusos en la utilización de la red por parte de los usuarios, como por

ejemplo el envió de *spam*[7] o los ataques de denegación de servicios (DoS) [8]. Lo que se buscaba con este nuevo algoritmo era que al enviar un nuevo mail los usuarios tengan que resolver un problema que era altamente demandante de capacidad de cómputo, y así para luego enviar el mensaje al servidor de correo. Cuando este lo recibía se encargaba de verificar que el problema se haya resuelto correctamente para enviar el mensaje al destinatario. Se estimaba que el cálculo de este problema consumía un centésimo de dólar, lo cual era despreciable para el usuario promedio, pero que era difícil de afrontar para los usuarios que desean realizar *spam* enviando millones de mensajes.

#### 4.6.3.1. Proof of work en Bitcoin

La implementación de *proof of work* que se utiliza en *Bitcoin* busca elegir de forma aleatoria a un nodo que pertenece a un conjunto de nodos que han demostrado que han resuelto un problema que insume el gasto de un recurso que ningún individuo de la red puede monopolizar. Esto lleva a concluir que los nodos serán seleccionados de forma proporcional a su capacidad de cómputo.

En *Bitcoin* el problema que se le plantea resolver a los nodos es el de encontrar un número llamado *nonce* de tal forma que el cálculo del hash de la concatenación de ese número con el hash del bloque anterior y todas las transacciones incluidas en el nuevo bloque cumple ser menor a un valor específico. Esto restringe de forma considerable la cantidad de hash válidos. Es decir, se busca un valor de *nonce* que cumpla la siguiente inecuación:

$$HASH(nonce \,||\, prev\_hash \,||\, tx_1 \,||\, tx_2 \,||\, ... \,||\, tx_k\,) \;<\; target \tag{4.3}$$

La dificultad de resolver esta inecuación por parte de los nodos estará totalmente ligada a la propiedad *puzzle-friendliness* de las funciones de hash 4. Si la función de hash tiene esta propiedad entonces no habrá atajos para resolver el desafió y se tendrá que intentar con diferentes valores de *nonce* hasta tener suerte y cumplir la inecuación 4.2.

El primer nodo en resolver el problema envía el nuevo bloque y el valor del hash que cumple con la dificultad del problema y espera que su bloque sea agregado de forma definitiva en la *blockchain*. En caso de que eso suceda, el nodo obtendrá la recompensa prometida (una cierta cantidad de *bitcoins*). Utilizar incentivos para persuadir a una gran cantidad de participantes a que ejecuten un algoritmo de consenso de forma honesta es necesario para garantizar que las propiedades deseadas de cualquier algoritmo de consenso sean cumplidas 4.6.

## 4.7. Árboles de Merkle

El concepto de árbol de Merkle, fue introducido por primera vez por Ralph Markle en 1978 [85]. Un árbol de Merkle es un árbol binario que resume y verifica la integridad de un conjunto grande de elementos. Como cualquier árbol binario, cada

7 Spam: hacen referencia a los mensajes no solicitados, no deseados o con remitente no conocido (o incluso correo anónimo o de falso remitente), habitualmente de tipo publicitario.

8 ataque de denegación de servicio (**D**enial **o**f **S**ervice): es un ataque a un sistema de computadoras que causa que un servicio o recurso sea inaccesible a los usuarios legítimos de forma temporal o permanente.

nodo de un árbol de Merkle tiene a lo sumo dos nodos hijos. *Blockchain* usan árboles Merkle para resumir todas las transacciones de un bloque y verificar si una transacción específica se encuentra en un bloque o no, utilizando un algoritmo de búsqueda eficiente con complejidad de tiempo $O(log_2(n))$ siendo $n$ la cantidad de transacciones en el bloque [86].

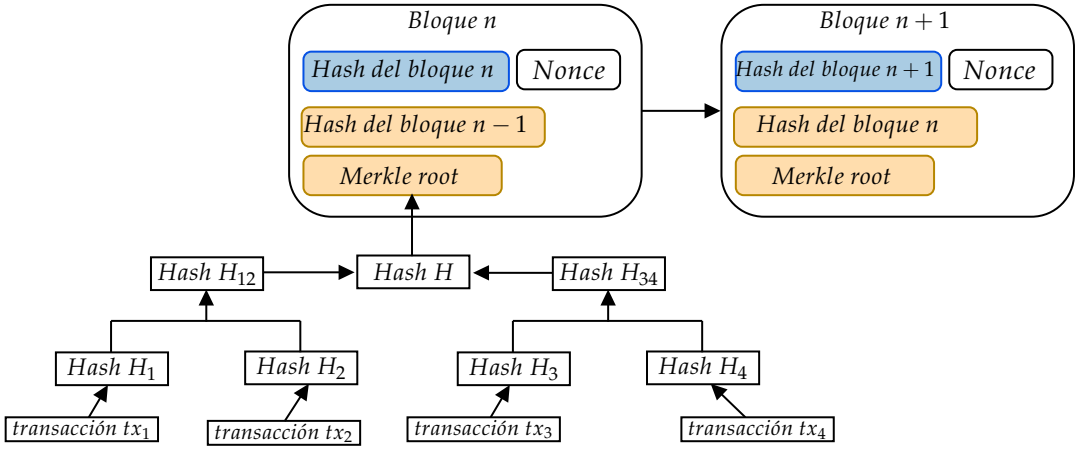


FIGURA 4.7: Árboles de Merkle

En los encabezados de los bloques de Bitcoin y *blockchain* se tiene un resumen de todas las transacciones que existen en el bloque usando una estructura de árbol de Merkle [87] como se ilustra en la figura 4.7. Utilizando el mismo mecanismo de hash descrito en la sección 3.2, en cada bloque construirá un árbol de Merkle partiendo desde "abajo" y yendo hacía "arriba" ejecutando la función de hash de forma recursiva en pares de nodos. Este proceso continúa hasta que solo haya un hash restante, al que se le llama raíz de Merkle (Merkle root). Por lo tanto, los hash de las transacciones se almacenan acumulativamente en cada nodo secundario, y se filtran hasta la raíz de Merkle, formando un camino de Merkle (Merkle path) [88].

La verificación de pagos simplificados (Simplifed Payment Verification (SPV)), introducida en el artículo de Satoshi Nakamoto, es un notable caso de uso que implementa la técnica de árboles de Merkle [89]. Los nodos en SPV no tienen que descargar todas las transacciones de un bloque para determinar que determinada transacción está incluida en ese bloque o no, en lugar de eso el camino de Merkle como estrategia.

## 4.8. Servicio de timestamp

*Blockchain* define un servicio *timestamp* (marca de tiempo) como un método para registrar el tiempo que los nodos reciben y acuerda a cada transacción [90]. El servicio de *timestamp* toma el hash de cada bloque (lo que sería el hash de todas las transacciones en un bloque como se explica en las sección 3.2) y lo estampa con el tiempo como se ilustra en la figura 4.8. Este *timestamp* contiene la marca de tiempo del bloque anterior y demuestra la existencia de datos en ese momento.

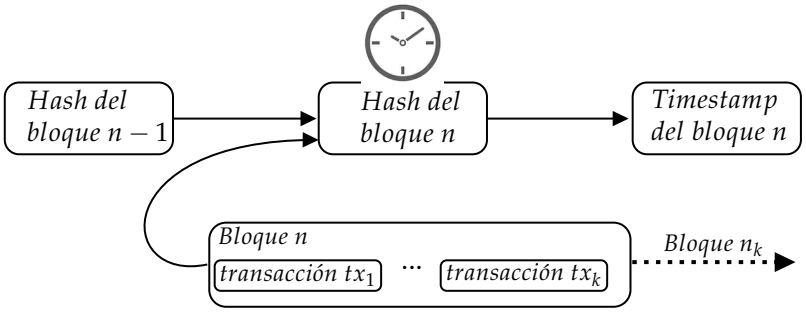


FIGURA 4.8: Servicio timestamp

## 4.9. Smart contract

Los *smart contracts* fueron creados por Szabo [91, 92] en el final de la década de los noventa, pero su potencial fue explotado realmente con la creación de *Bitcoin*. Szabo describió a los *smart contracts* de la siguiente manera:

> "Un *smart contract* es un protocolo transaccional electrónico que ejecuta los términos de un contrato."

Esta idea de *smart contracts* tuvo su primer implementación limitada en *Bitcoin*, donde se hacía uso de un lenguaje de *script* con operaciones limitadas que permitía transferir *bitcoins* entre usuarios por medio de una red de peers, sin la necesidad de tener confianza entre ellos ni de tener confianza en terceros. En la comunidad existen diversas definiciones formales sobre lo que es un *smart contract* pero en este proyecto se toma la definición dada en [93]. La cual dice que un *smart contract* es:

> *Un smart contract es un programa informático seguro e imparable que representa un acuerdo que se puede ejecutar y hacer cumplir automáticamente.*

A partir de la definición se puede decir que los *smart contracts* son:

- Programas informáticos.
- Que su formulación es de mutuo acuerdo entre los participantes.
- Que se ejecutan de forma automática en caso de que ocurra determinado evento.
- Todos los términos definidos en el mismo se ejecutan como se espera, aún en presencia de adversarios, por lo que se puede decir que son seguros.

Una característica que no se nombra en la definición es la necesidad de que todos los *smart contracts* sean deterministas, es decir, dada una entrada siempre se obtenga como resultado el mismo valor. Esto es realmente importante ya que si el valor obtenido difiere aunque sea sutilmente entre dos nodos esto dispararía un problema a la hora de generar consenso. Esto podría llevar a que el algoritmo de consenso utilizado en la *blockchian* falle.

# Capítulo 5

# HyperLedger Fabric

En este Capítulo se presenta Hyperledger Fabric, el cual es una blockchain de configuración *permissioned*, y que se toma como punto de referencia para el análisis de privacidad en las estrategias que se estudian en los capítulos 6 y 7. Dichas dos estrategias presentadas ofrecen nuevas alternativas con mejores propiedades de privacidad que se las que se encuentran por defecto en Hyperledger Fabric.

## 5.1. Introducción

Hyperledger Fabric (o simplemente Fabric) es un *framework* de tecnología DTL de código abierto y de nivel empresarial, diseñada para utilizarse en contextos empresariales, que ofrece algunas características particulares con respecto a otras plataformas DTL o *blockchain* populares.

Un punto clave de diferenciación de Hyperledger con otras plataformas de tecnología DTL es que se estableció bajo la *Linux Fundation* [13], que tiene una larga y exitosa historia en fomentar proyectos de código abierto bajo una gobernanza abierta haciendo crecer comunidades sólidas y sostenibles. Hyperledger está gobernado por un comité directivo técnico extenso y diverso, y el proyecto Hyperledger Fabric por un conjunto diverso de contribuyentes de múltiples organizaciones.

Fabric tiene una arquitectura modular y altamente configurable, que permite la innovación, la versatilidad y la optimización para una amplia gama de casos de uso de la industria. Algunos de los casos de uso pertenecen a los rubros de las finanzas, el manejo de los seguros, la atención médica, los recursos humanos, cadenas de suministros e incluso la entrega de música digital.

Fabric es una de las primeras plataformas DTL que admite los *smart contract* (contratos inteligentes) creados en lenguajes de programación de propósito general como Java [94], Go [95] y Node.js [96] (a los que llama *Chaincode*), en lugar de lenguajes DSL [1]. Esto significa que la mayoría de las empresas ya tienen el conjunto de habilidades necesarias para desarrollar contratos inteligentes, y no se necesita capacitación adicional para aprender un nuevo lenguaje de programación.

La plataforma Fabric es *permissioned*, lo que significa que, a diferencia de una red pública, los participantes se conocen entre sí (en lugar de ser anónimos y, por lo tanto, poseen cierta confianza entre ellos).Esto último significa que si bien los participantes pueden no confiar completamente el uno en el otro (es decir, por ejemplo,

1 DSL (Domain Specific language): lenguaje de programación o especificación dedicado a resolver un problema en particular.

ser competidores en el mismo rubro), una red puede funcionar bajo un modelo de gobernanza que se basa en la confianza que existe entre los participantes, como un acuerdo legal o marco para el manejo de disputas.

Otra de las diferencias más importantes de la plataforma es su soporte de tipo *plug-and-play* para los algoritmos de consenso, que permiten que la plataforma se personalice de manera efectiva para adaptarse a casos de uso particulares. Por ejemplo, cuando se implementa dentro de una sola empresa, o es operado por una autoridad confiable, un consenso totalmente tolerante a fallas bizantinas (BTF) puede considerarse innecesario y una carga excesiva en el rendimiento. En situaciones como esa, un protocolo de consenso tolerante a fallas (CFT) podría ser más que adecuado, mientras que, en un caso de uso descentralizado y en donde existan conflictos de intereses entre las partes, se podría requerir un protocolo de consenso tolerante a fallas bizantinas (BFT).

La combinación de estas características de diseño hacen que Fabric sea una de las plataformas de mejor rendimiento disponible en la actualidad tanto en términos de procesamiento de transacciones, como también de latencia en la confirmación de transacciones. Además permite algún grado de privacidad y confidencialidad en las transacciones y los *smart contracts* [97, 98].

## 5.2. Arquitectura de Fabric

Fabric es una *permissioned blockchain* con una arquitectura modular. La cual utiliza la tecnología de contenedores para alojar *chaincodes* que contienen la lógica de la aplicación. Antes de describir cada componente en detalle, veamos un flujo de transacciones de alto nivel y los componentes básicos involucrados en la figura 5.1 [98, 99].

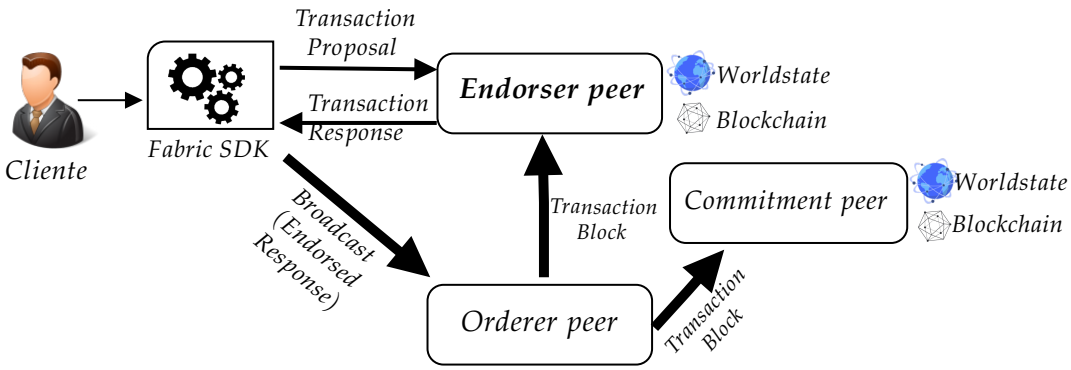


FIGURA 5.1: Arquitectura de Hyperledger Fabric

Fabric está compuesta por los siguientes componentes: *peers*, *Channels*, *Ordering service*, *Fabric CA*, *Ledger*, *Chaincodes*, MSP. A su vez los usuarios podrán ser clientes de alguna aplicación propiedad de una organización o clientes de los administradores de la red *blockchain*. A continuación, se detallan los componentes de Fabric:

- **Peer:** Un *peer* recibe actualizaciones sobre el estado en forma de bloques del ordering service, y mantiene el estado y la ledger.
  Además los *peer* pueden tomar cierto rol especial, que es el de *endorsing peer*, o

*endorser*. La función del *endorsing peer*, ocurre con respecto a una *chaincode* particular y consiste en aprobar una transacción antes de que se confirme. Todas los *chaincode* deben especificar una política de endorsement, que será referida a su conjunto de *endorsing peers*. Esta política define las condiciones necesarias y suficientes para validar las transacciones de endorsement.

- **Ordering service:** El *ordering service* es el componente dedicado a llevar adelante el consenso. Los *orderer peer* son los peers que forman el *ordering service*, es decir, un tejido de comunicación que proporciona garantías de entrega. El *ordering service* se puede implementar de diferentes maneras: desde un servicio centralizado hasta protocolos distribuidos que se dirigen a diferentes modelos de fallas de red.
  El *Ordering service* proporciona un canal de comunicación compartido a clientes y peers, ofreciendo un servicio de difusión para mensajes que contienen transacciones. Los clientes se conectan al canal y pueden transmitir mensajes en el canal que luego se envían a todos los peers. El canal admite la entrega atómica de todos los mensajes, es decir, la comunicación de mensajes con entrega en orden. Los mensajes comunicados son las transacciones candidatas para su inclusión en la *blockchain*.

- **Channel (canal)**: Un *channel* en Fabric es una "subred" privada de comunicación entre dos o más miembros específicos de la red, con el fin de realizar transacciones visibles únicamente por los miembros del canal. Un canal está definido por miembros (organizaciones), la ledger, las aplicaciones de *chaincode* y los peers de *ordering*. Cada transacción en la red se ejecuta en un canal, donde cada participante debe ser autenticado y autorizado para realizar transacciones en ese canal. Cada participante que se une a un canal, tiene su propia identidad dada por un proveedor de servicios de membresía (MSP), que autentica a cada participante con sus pares y servicios de canal.

- **Fabric CA:** Fabric CA es una autoridad certificadora (CA) basada en una infraestructura de clave pública (PKI) de Fabric, que proporciona características como: registro de identidades, manejo de *LDAPs* [2], emisión de certificados (ECerts), renovación y revocación de certificados.

- **Ledger:** La *ledger* (que es el registro inmutable de transacciones) consta de dos partes diferentes, un *world state* (estado general) y una *blockchain* los cuales registran los sucesos de la red.

  **World state:** Es una base de datos que contiene el estado actual de la ledger. Estos estados se expresan como pares clave-valor. Tiene el hecho de ser un objeto comercial. Se puede crear, actualizar y eliminar en ellos. Cuando una aplicación envía la transacción o cuando se trata de confirmar la validez de una transacción, primero es confirmada en el *world state* y luego se actualiza en la ledger.

  **Blockchain:** Es un registro de transacciones que mantiene todos los cambios que han resultado en el *world state* actual. Su estructura de datos es diferente ya que una vez escrita no se puede eliminar ni modificar. Es un registro histórico de hechos sobre cómo llegaron los objetos al estado actual. Está estructurado

2 LDAP (Lightweight Directory Access Protocol): protocolo de capa de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar información en un entorno de red.

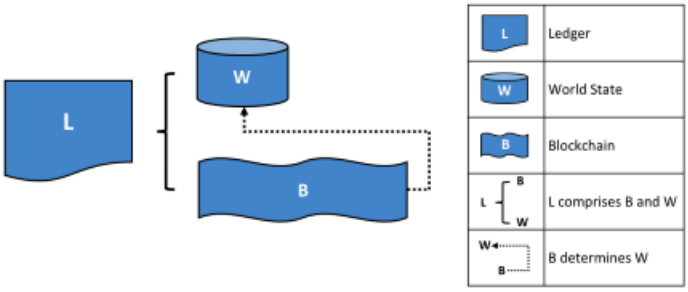


FIGURA 5.2: La *ledger* por formada por el Worldstate y por la Blockchain.

como un registro secuencial de bloques interconectados, donde cada bloque contiene una secuencia de transacciones. Además, cada transacción representa una consulta o actualización del *world state*.
Cada encabezado de bloque incluye un hash de transacciones de bloques, así como una copia de un hash del encabezado del bloque anterior. *Blockchain* siempre se implementa como un archivo, y no como el *world state* que se implementa mediante una base de datos. El primer bloque en la *blockchain* es el bloque génesis. Aunque este es un punto de partida de la *ledger* este no contiene ningún detalle de ninguna transacción, sin embargo, contiene datos sobre la configuración que describen el estado inicial de la red.

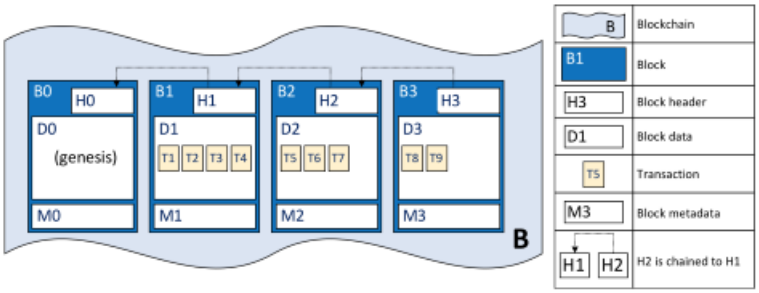


FIGURA 5.3: Estructura de la blockchain en Fabric.

Por otro lado, tendremos que cada bloque esta formado por los siguientes elementos:

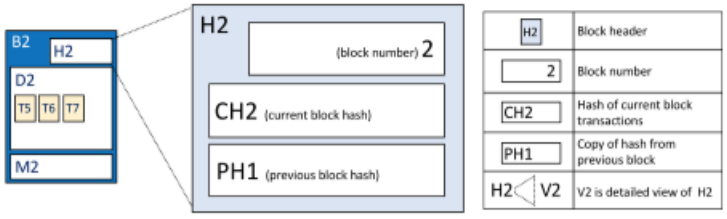


FIGURA 5.4: Estructura interna de los bloques en Hyperledger.

- **Cabecera (*header*)**: Básicamente consiste en tres campos,
  1. Número de bloque: un número entero que comienza desde 0, el bloque de génesis, y se va incrementando en 1 por cada nuevo bloque agregado.
  2. Hash de bloque actual: Hash de todas las transacciones contenidas en el bloque.
  3. Hash de bloque anterior: es el valor del hash del bloque anterior.
- **Datos**: Consiste en una lista de transacciones organizadas en orden.
- **Metadatos**: Contiene el momento en que se escribió el bloque, el certificado, la clave pública y la firma del escritor del bloque. Un bloque confirmado también agrega un indicador válido/no válido para cada transacción. Esta información no está incluida en el hash del bloque.
- Además tendremos las transacciones que tienen la siguiente estructura:

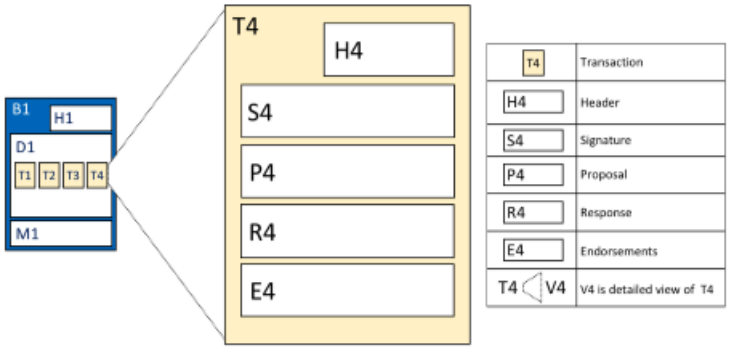


FIGURA 5.5: Estructura de las transacciones en Hyperledger.

  - Encabezado: contiene metadatos esenciales sobre la transacción. Por ejemplo: nombre de la *chaincode* relevante y su versión.
  - Firma: contiene la firma digital creada por la aplicación cliente. Se utiliza para verificar si los detalles de la transacción no han sido alterados.
  - Proposal: codifica los parámetros de entrada suministrados por la aplicación al *smart contract*. Cuando se ejecuta el *chaincode*, esta propuesta proporciona un conjunto de parámetros de entrada que, en combinación con el *world state* actual, determina el nuevo *world state*.
  - Response: Captura los valores anteriores y posteriores del *world state* como conjunto de lectura/escritura (RW).
  - Endorsements: es la lista de response de transacciones firmadas, deben ser la cantidad suficiente para satisfacer la política de *endorsement* establecida. Para cada una de las propuesta de transacción, tendremos múltiples responses las cuales tienen que ser consistentes (debe de ser posible verificar que el que construyó esa response es quien dice ser y que es un miembro autorizado para simular la propuesta de transacción) y suficientes (tienen que cumplir la cantidad mínima establecida por la política de endorsement).

- **Smart contract:** Desde la perspectiva de un desarrollador de aplicaciones, un *smart contract*, junto con la ledger, forman el corazón de un sistema *blockchain*

en Fabric. Mientras que la *ledger* contiene hechos sobre el estado actual e histórico de un conjunto de objetos comerciales, un *smart contract* define la lógica ejecutable que genera nuevos hechos que se agregan a la ledger.

- **Chaincode:** Los *chaincode*, es el nombre que toman la implementación de los *smart contract* en Fabric. Un *chaincode* es un programa informático, escrito en Go, Node.js o Java que implementa una interfaz. Cada *chaincode* se ejecuta en un contenedor Docker [3] aislado del proceso de verificación. Un *chaincode* inicializa y gestiona el estado de la *ledger* a través de las creación de nuevas transacciones.
  Se puede invocar un *chaincode* para actualizar o consultar a la *ledger* mediante una nueva transacción. Con los permiso apropiado, un *chaincode* puede invocar a otro *chaincode*, ya sea del mismo canal o de diferentes canales, para acceder a su estado.

### 5.2.1. MSP (Membership Service Providers)

El *Membership Service Providers* (proveedor de servicios de membresía) mantiene las identidades de todos los participantes del sistema (clientes, peers, organizaciones, etc) y es el responsable de emitir credenciales que se utilizan para la autenticación y autorización de los peers. Dado que Fabric es de configuración *permissioned*, todas las interacciones entre los peers se producen a través de mensajes autenticados (generalmente mediante firmas digitales).

El MSP es un componente en cada participante de la plataforma, donde puede autenticar transacciones, verificar la integridad de las transacciones, firmar y validar endorsements y autenticar otras operaciones de *blockchain*. Las herramientas para la gestión de claves y el registro de peers también son parte del MSP.

El MSP es una abstracción de la cual pueden existir diferentes instancias. Por defecto en Fabric el MSP utiliza métodos estándar de infraestructura de clave pública (PKI) para el manejo de identificadores, que está basada en firmas digitales. Además MSP proporciona una CA independiente con Fabric, llamada Fabric CA.

3 Docker: Docker es un proyecto de código abierto que brinda automatización para desplegar aplicaciones dentro de uno o varios contenedores de software, brindando de esta forma una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos [100].

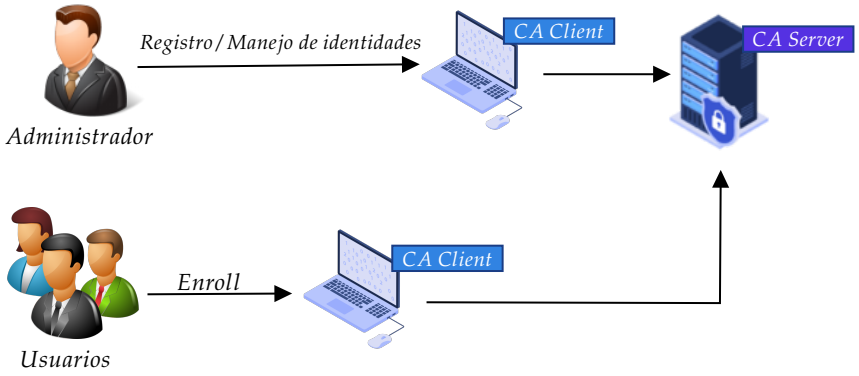


FIGURA 5.7: Operaciones sobre Fabric-CA

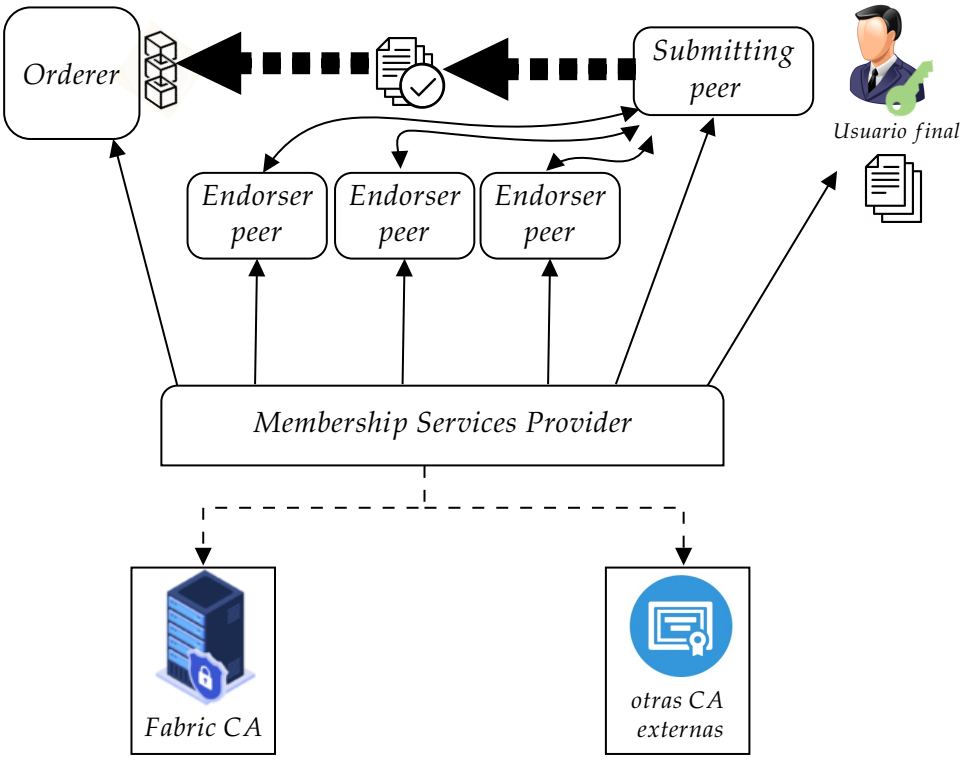


FIGURA 5.6: Interacciones del MSP en Hyperledger Fabric

La configuración de MSP debe asegurarse de que todos los participantes de la red, especialmente todos los peers, reconozcan las mismas identidades y autenticaciones válidas

## 5.3. Manejo de identidades

Para el manejo de identidades Fabric utiliza una infraestructura de clave pública (PKI). Cada componente (*peer, orderer*) y usuario en Fabric se identifica mediante un certificado. Los certificados se gestionan a través de Autoridades Certificadoras (CA). Fabric cuenta con una implementación propia de Autoridad Certificadora llamada *Fabric CA*.

Existen dos implementaciones para las Fabric CA:

- *fabric-ca-server:* Administra las identidades en forma persistente, por ejemplo, una base de datos, o LDAP.
- *fabric-ca-client:* Es una utilidad para gestionar identidades en el *fabric-ca-server*.

La emisión de certificados de *Fabric CA* que se ilustra en la figura 5.7 implica dos acciones:

- *Registro*: El administrador crea una entrada en el servidor *Fabric CA* al asignar una nueva identidad. Esto genera un ID de inscripción y una clave, que el administrador envía al usuario por un medio seguro.
- *Inscripción*: el usuario utiliza el *fabric-ca-client* para ejecutar el comando de inscripción en el *fabric-ca-server* para obtener un certificado de inscripción.

## 5.4. Transaction flow

El *transaction flow* es el proceso que define todas las etapas por las que deberá pasar una transacción desde que es creada por un *client peer* hasta que es persistida en la *ledger* de todos los peers. El *transaction flow* se puede dividir en varias etapas según la perspectiva en que se mire el mismo.

Se puede seguir el enfoque que se realiza en [101], donde se mira el *transaction flow* desde el punto de vista de la ejecución de fases independientes, las cuales se puede dividir en tres grandes etapas, las cuales son [102]:

- ***Fase de ejecución:*** donde se realiza la simulación de la transacción.
- ***Fase de ordenamiento:*** donde se agrupan las transacciones que han cumplido con las políticas de *enderosement* y se ejecuta el algoritmo de consenso.
- ***Fase de validación:*** donde se validan todas las transacciones que están contenidas dentro de un bloque recibido pero sin tener que volver a ejecutarlas.

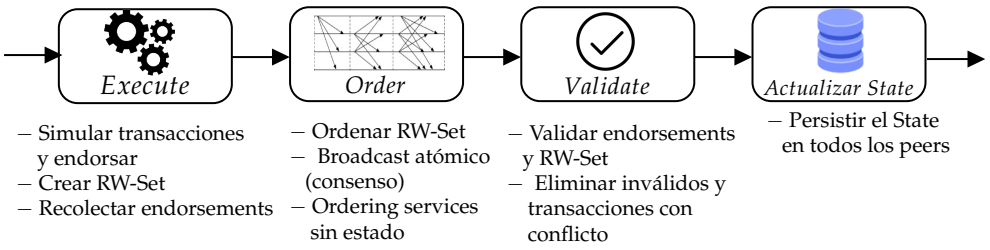


FIGURA 5.8: Arquitectura Execute-Order-Validate

Por otro lado, se puede seguir el enfoque que se realiza en [103], donde se mira el *transaction flow* desde el punto de vista de las interacciones entre los peers de la red, las cuales se pueden dividir en siguientes cinco etapas:

- ***Initiating Transactions:*** donde el *client peer* genera un nuevo mensaje que incluye su firma y una operación a ser ejecutada en la red.
- ***Transaction Proposal:*** donde un *submitting peer* recibe el mensaje generado por el *client peer*. Este nodo valida la firma del cliente y en caso de que la misma sea valida, propone una nueva transaccion en la red.

- ***Transaction Endorsement:*** donde cada *endorser peer* que reciba la propuesta de transacción validará la firma del *client peer*, en caso de que sea válida, el *endorser peer* simula las operaciones que contiene la transacción usando su versión local de la *ledger*. Finalmente, envía los resultados de la simulación en un nuevo mensaje al *submitting peer* que le envió la propuesta de transacción.
- ***Broadcasting to Consensus:*** donde el *submitting peer* recolecta las respuestas de los *endorser peer* hasta tener la cantidad necesaria de respuestas válidas para cumplir con la política de endorsement. En caso de alcanzar la cantidad necesaria de respuestas válidas, crea una *transaction* que contiene las operación que se simuló en la red, con qué parámetros de entrada se ejecutó la misma, el conjunto de *endorser peer* que simularon correctamente la transacción y metadatos. Esta *transaction* es enviada al *ordering service*.
- ***Ordering and packaging:*** donde el *ordering service* recibe las transacciones de los diferentes peers del mismo canal y las agrupa en bloques, los cuales enviará a todos los peers de la red.
- ***Validation and commit:*** donde todos los *peers* que pertenecen a un canal reciben los nuevos bloques por parte del *ordering service*, luego de recibir un nuevo bloque los peers pasan a validar todas las transacciones que contiene sin ejecutarlas nuevamente.

Tenemos que las primeras tres etapas forman la fase de ejecución, la cuarta y quinta etapa se corresponde con la fase de ordenar y la quinta etapa se corresponde con la etapa de validación.

### 5.4.1. Decisiones de diseño

A continuación detallaremos algunas de las decisiones de diseños llevadas a cabo en cada una de las fases de la arquitectura Execute-Order-Validate (que se puede visualizar en la figura 5.8) de Fabric.

#### 5.4.1.1. Execution phase

Una de las decisiones de diseño que se toman en esta fase es establecer que la simulación por parte de los *endorsing peers* se haga de forma independiente y sin realizar ninguna sincronización con otros *endorsing peers*. Esto lleva a que dos *endorsing peer* pueden simular la misma propuesta de transacción con diferentes estados de la *ledger* y obtener un resultado diferente. Esto es un problema cuando se utiliza la política de endorising estándar, ya que la misma requiere que para que una propuesta de transacción cumpla con la política todos los resultados de las simulaciones de los *endorsing peer* deben de obtener el mismo resultado. En caso de que no se cumpla con esta política la transacción será descartada.
Los argumentos para tomar esta decisión de diseño por un lado son mantener la arquitectura simple y por otro lado por que este comportamiento es adecuado para aplicaciones típicas que utilizan *blockchain*.

Otra decisión de diseño que se adopta en esta fase del *transaction flow* es la de ejecutar las transacciones antes de la fase de *ordering*. Esta decisión permite la ejecución de chaincodes no deterministas en la red.

#### 5.4.1.2. Ordering phase

La decisión de diseño más importante que se toma en esta etapa es que el *ordering service* no mantenga ningún estado de la cadena de bloques, ni que simule ni valide transacciones. Esto hace posible una separación absoluta de la etapa de consenso de las etapas de ejecución y de la de validación.

#### 5.4.1.3. Validation phase

La *ledger* de Fabric contiene absolutamente todas las transacciones, incluidas aquellas que son inválidas (contrario a lo que sería Bitcoin o Ethereum, en donde la *blockchain* solo contiene transacciones válidas). Esta característica es necesaria ya que, existen casos de uso en donde es necesario realizar una trazabilidad de transacciones inválidas (por ejemplo para realizar auditorías). Por lo que, podría ser posible que existan clientes que intenten realizar un ataque de denegación de servicio (DoS) inundando la red con transacciones inválidas. Un posible enfoque para solucionar esto, sería tener una lista negra con clientes que no cumplan con determinadas políticas, impidiéndoles seguir enviando propuestas de transacciones.

## 5.5. Privacidad en Hyperledger Fabric

El esquema por defecto del MSP de Fabric utiliza certificados *X.509* [4], los cuales utiliza para validar identidades, o revocarlas conforme al estándar que define. Aprovechando que este esquema es eficiente, flexible y escalable, las organizaciones pueden tener CA organizadas de formas jerárquicas, y esto se traduce a MSP organizadas de formas jerárquicas.

Cada transacción tienen dos campos específicos para realizar validaciones, el campo *Creator*, el cual contiene la identidad del usuario que invoca la transacción, y el campo *Signature*, que contiene la firma del usuario creador de la transacción sobre la misma, la cual termina autorizando la transacción. El usuario que firma la transacción no tiene por que coincidir con el valor *Creator*. Como cada transacción lleva la identidad de sus orígenes, un certificado y su firma, la implementación del estándar X.509 termina comprometiendo el anonimato y también la privacidad de los clientes.

La solución a este problema que propone Fabric es utilizar una estrategia de mezclar identidades a la que llama *identity mixing (Idemix)* [104], El protocolo MSP basado en Idemix permite a los clientes firmar transacciones de forma anónima. Básicamente la estrategia es que en lugar de usar un certificado X.509, un idemix MSP emite una credencial especial que contiene un conjunto de atributos. Entonces para firmar una transacción, el titular de una identidad Idemix genera una prueba de conocimiento cero no interactivo (NIZK) 3.6.3 de que recibió una credencial de Idemix que certifica esos atributos.

4 X.509: es un estándar UIT-T para infraestructuras de claves públicas (PKI). Que especifica (entre otras cosas) formatos estándar para certificados de claves públicas y un algoritmo de validación de la ruta de certificación.

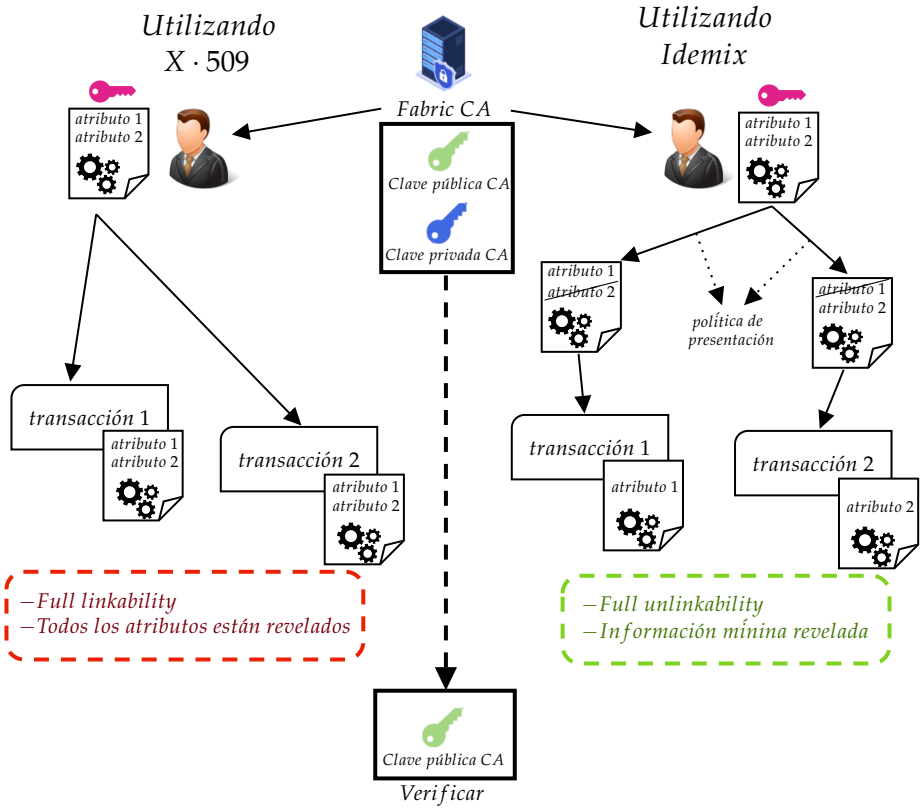


FIGURA 5.9: Esquema X.509 frente al esquema Idemix en Hyperledger Fabric

Más específicamente, si Alicia es miembro de una organización *Org* cuyos miembros están autorizados a realizar ciertas transacciones, Alicia demuestra que posee una credencial Idemix de su MSP que certifica que es miembro de *Org*.
Aunque esta estrategia no cubre todo el problema, debido a que tiene dos inconvenientes, el primero de ellos es que no es posible hoy en día revocar credenciales Idemix, por lo que conlleva a un gran problema en el manejo de las mismas, y por otro lado, dado que a no se desvincula al cliente del MSP, y por lo tanto de su *Org*, esto no es suficiente para tener privacidad absoluta. Se puede seguir investigando esto en el artículo de Angel De Caro *et al.* "Anonymous Transactions with Revocation and Auditing in Hyperledger Fabric"[105].

## 5.6. AleaChain en Hyperledger Fabric

En esta Sección discutiremos una posible instanciación de *AleaChain* bajo el modelo que posee Hyperledger Fabric. Cuando se habla de instanciación, se habla de exponer bajo las restricciones y propiedades del modelo en el cual se instancia, a ejemplo de ver cómo quedaría reflejado el caso de uso bajo dicho esquema. Esto es, expresar, por ejemplo, qué acciones realizaría un actor en el sistema, o cómo se realizaría una determinada acción bajo el esquema en el cual se instancia, a efectos de poder discutir qué ventajas se obtienen, qué desventajas, y también poder comprender qué restricciones brinda dicho modelo para el caso de uso.

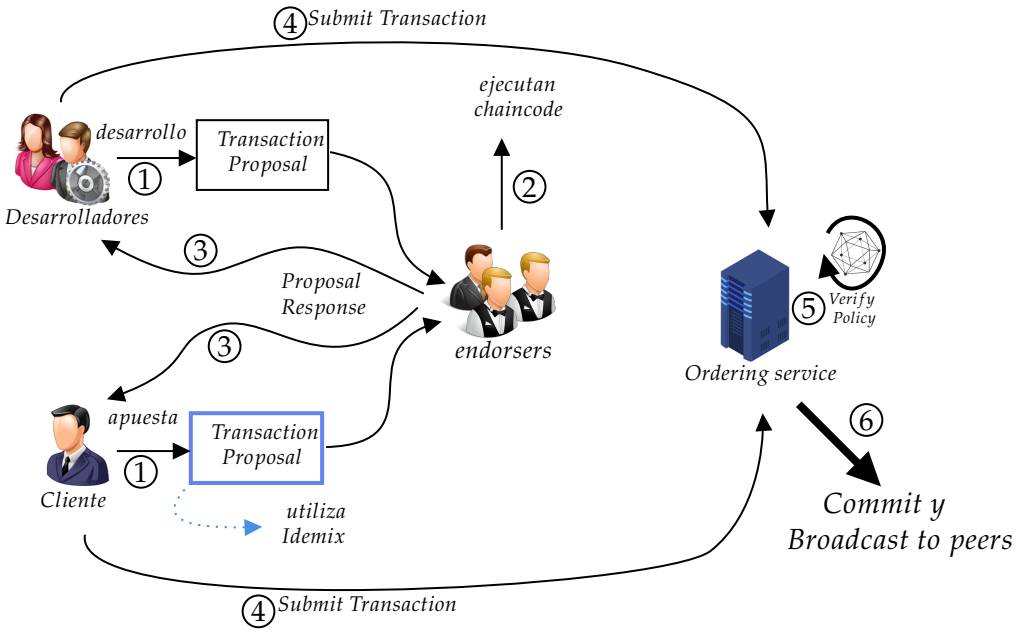


FIGURA 5.10: Transaction Flow de Hyplerledger Fabric instanciando AleaChain

Se puede decir que Fabric al tratarse de una *permissioned blockchain* y de implementar de forma nativa el uso de *smart contracts*, adhiere casi en su totalidad a resolver los requisitos que se definen para AleaChain.

Tendremos que a la hora de realizar una apuesta, los jugadores deberán simplemente abrir un canal entre ellos, ponerse de acuerdo en el tipo de juego a utilizar, y la cantidad a apostar, entablando un *smart contract*. Para la generación de números aleatorios existirán usuarios específicos que cumplirán el rol de oráculos, es decir, se les solicitara que entreguen un número aleatorio. Luego de obtenido el número aleatorio, el *smart contract* definirá quien es el ganador y le enviará lo que corresponda por haber ganado la apuesta.

Al tener un *smart contract* definido, simplemente al finalizar su ejecución los jugadores cobrarán sus respectivos montos de la salida de la apuesta, con validación de la red.

Para administrar tokens en Fabric, se cuenta con FabToken [106], el cual es un sistema de gestión de tokens que le permite emitir, transferir e intercambiar tokens utilizando Fabric. Los tokens se almacenan en la *ledger* del canal y pueden ser propiedad de cualquier miembro del canal. FabToken utiliza los servicios de membresía de Fabric para autenticar la identidad de los propietarios de tokens y administrar sus claves públicas y privadas.

Lo último que resta comentar es sobre el anonimato a la hora de las apuestas, para lo cual se puede utilizar Idemix. Idemix es un mecanismo propio de Hyperledger, que permite a los usuarios firmar transacciones sin vincular completamente su identidad, utilizando una estrategia de mixing. Este mecanismo es el que se describe en la Sección 5.5.

En la figura 5.10 se visualiza el típico *transaction flow* de Fabric 5.4. En dicha figura se muestran los dos tipos de ingresos de transacciones: por un lado las apuestas, y por el otro los cambios en las aplicaciones de apuestas.

Para el caso de las apuestas, se utiliza el servicio de Idemix al comenzar la fase llamada *Transaction Proposal* (primera fase del *transaction flow*). De esta forma se garantiza el anonimato en las apuestas. Posteriormente se adjudica el rol de *endorsers* a los nodos pertenecientes al casino (coloquialmente llamados curier) y potencialmente a cualquier participante externo que esté dispuesto a validar transacciones a cambio de alguna recompensa monetaria. Todos estos nodos validarán las apuestas o los cambios empleados por los desarrolladores sobre los juegos, y luego se continuará con el transaction flow habitual.

Hay que tener en cuenta que se deja de lado el asunto del algoritmo de consenso. Esto aprovechando que Fabric permite escoger el algoritmo de consenso. Además, también se cuenta con una ventaja importante: dado que los nodos endorsers de la red, serán nodos propios de los casinos y por ende nodos de confianza, podría no contar como crítico un ataque de parte de ellos. Lo que sí puede ser un problema, es el hecho de que los nodos fallen, para este caso, los algoritmos de consenso propios de Fabric son *CFT* y gracias a esto prevén este tipo de problemas. Se puede discutir, además, qué ocurriría si los nodos endorsers son ”tomados” en un ataque cibernético. En este caso, se necesitaría tomar al 51 % de los nodos activos para controlar todas las transacciones, y esto sí se convertiría en un problema. Para lo cual, se podría poner un algoritmo del estilo *Proof-of-Work* e incluir a los clientes en el proceso de *endorsement*, e incentivándolos con fichas de casino, por ejemplo.

# Capítulo 6

# BlindCons

En este Capítulo se presenta la primera variante estudiada durante esta tesis que busca aumentar el nivel de privacidad en blockchains *permissioned*. Esta variante fue presentada por Koscina *et al.* en el artículo "BlindCons: A Consensus Algorithm for Privacy Preserving Private Blockchains" [103]. El elemento central de este artículo es la descripción de un algoritmo de consenso que utiliza un algoritmo de firma ciega en sustitución de los algoritmos de firma digital habituales.

## 6.1. Contextualización

Las redes blockchains de configuración *permissioned* son cada vez más usadas por las organizaciones. Esto ocurre, principalmente porque permite a las organizaciones no tener que cargar con las limitantes que tienen las *blockchain* de tipo *permissionless*, o públicas, en términos de consumo de energía, eficiencia y principalmente en el control de los usuarios que hacen uso del sistema.
Sin embargo, este tipo de configuración, de alguna manera hace que se pierda la esencia de *blockchain*, particularmente su característica fundamental de no ser controlada por un autoridad central que gobierne las funcionalidades que tiene la *blockchain*, en donde se incluye, por supuesto, el registro de usuarios.

## 6.2. Problemática

Bajo una configuración *permissioned* de *blockchain*, las transacciones realizadas por los usuarios serán visibles, para los demás usuarios que se encuentren registrados en la plataforma. Esto implica que cualquier usuario de la plataforma puede observar que es lo que otros usuarios guardan en la *blockchain*, y en algunos casos saber quienes son estos usuarios. Dependiendo el caso de uso en que se este usando esta configuración, esto puede ser una solución correcta o una que no lo es y que pone en peligro la privacidad de los usuarios.

Por ejemplo, si se toma, el caso de uso del artículo "DABSTERS: a Privacy Preserving e-Voting Protocol for Permissioned Blockchain"[107], en dónde se presenta un sistema de votación electrónico el cual está basado en BlindCons. En este caso de uso se define que los votos, deben de cumplir determinadas propiedades, las cuales son: cada voto debe de ser emitido por un votante registrado y este deberá de ser único por cada votante. Cada voto emitido, debe poder ser públicamente verificable luego de haberse publicado los resultados pero de cada uno de ellos no se debe de poder obtener ninguna información sobre quien es el votante. Estas propiedades no pueden ser obtenidas en una *blockchain* del tipo *permissioned* puesto que si no existe

forma de identificar a los votantes, tampoco existe forma de saber si estos han votado una única vez y si se mantiene la identificación de los votantes en la *blockchain* no se garantiza que el voto sea secreto. En este caso, la integración de un algoritmo de firma ciega al proceso de votación garantiza que nadie podría votar más de una vez ya que la entidad certificadora lleva un registro interno de quienes han votado. En caso de que alguien quiera realizarlo dos veces, la entidad certificadora puede fácilmente anular el voto, mientras que al sustituir la firma digital del votante por una firma de la entidad certificadora en la transacción, se garantiza que el voto es secreto puesto que se ha eliminado la vinculación entre el votante y el voto. Se podría extrapolar el ejemplo, y su problemática a la hora de cubrir los requisitos que debe tener el sistema, y tomar también el ejemplo de AleaChain expuesta en el Capítulo 2, en donde también se busca lograr la privacidad de los usuarios en un sistema de casino electrónico.

BlindCons [103] es un artículo en donde se presenta un protocolo para redes *blockchain* con configuración *permissioned* por el cual se obtiene privacidad de las transacciones registradas en la *blockchain*. Esto elimina la posibilidad de realizar un ataque pasivo en la *blockchain* como el descrito en 1.3.1.1. El protocolo toma como base a Fabric, especialmente en su algoritmo de consenso, pero podría decirse que es aplicable a cualquier otro algoritmo de consenso que se rija mediante una autoridad certificadora. En el artículo se presentan los cambios necesarios para integrar una primitiva de firma ciega en sustitución del algoritmo de firma digital que usa de forma predeterminada Fabric. Este cambio de algoritmo permitirá a los nodos de la red, firmar transacciones de forma tal que su identidad no quede expuesta.

Continuando lo expuesto en la sección anterior sobre el artículo DABSTERS, se puede observar en dicho artículo un análisis de la posibilidad de utilizar *blockchains permissioned* para sistemas de votos electrónicos, en dicho artículo, se evalúa, algunos sistemas de votación electrónica basados en *blockchain* de configuración *permissioned*. Luego de este análisis, se propone un protocolo llamado DABSTERS para garantizar la privacidad de los votantes, además de la integridad de los votos y la verificabilidad de los resultados de las votaciones. Luego proponen un sistema de voto electrónico completamente descentralizado basado en una *blockchain permissioned*. El protocolo, denominado DABSTERS, utiliza un algoritmo de consenso basado en el implementado en Fabric que sustituye el esquema de firma digital por un esquema de firma ciega, en particular, se utiliza el algoritmo de Okamoto-Schnorr [56] para preservar la privacidad de los votantes, basado en la propuesta de BlindCons.

## 6.3. Primitivas criptográficas

Además de los esquemas de infraestructura de clave pública, el principal mecanismo que se emplea en la propuesta BlindCons, es la sustitución de la primitiva de firmas digitales que se usa habitualmente en Fabric por una primitiva de firma ciega. Este cambio permite ocultar la identidad de los usuarios, al momento de generar nuevas transacciones. El mecanismo de firma ciega empleado es el de Okamoto-Schnorr, el cual es un esquema adecuado para arquitecturas blockchains *permissioned* debido a que el proceso de firma ciega, puede ser ejecutado por la misma autoridad certificadora.
Si bien, implica además, una sobrecarga respecto a dicha autoridad, debido a que

para ejecutar un proceso de firma, es necesario interactuar con la entidad, esta sobrecarga puede ser aminorada empleando una infraestructura de clave pública con varias entidades certificadoras.

## 6.4. Propuesta BlindCons

La propuesta, emplea diferentes cambios que impactan en las cinco etapas del *transaction flow* de Fabric descritas en 5.4. A continuación, se detalla cuales son las modificaciones que se realizan en cada una de las etapas.

### 6.4.1. Cambios en la etapa de Initiating Transaction

Para explicar los cambios se definen las operaciones $BlindSign(M,(\beta,\sigma,\delta),y)$ que es la función para crear una firma ciega y $Verify-BlindSign(M,(\rho,\delta,\epsilon),y)$ que es la función que verifica una firma ciega. Los valores mencionados en estas funciones corresponde a:

- $M$: mensaje a ser firmado.
- $(\beta,\sigma,\delta)$: valores secretos y aleatoriamente obtenidos ($\xrightarrow{\boxdot} \mathbb{Z}_q$) por el cliente.
- $(\rho,\delta,\epsilon)$: resultado de la firma ciega.
- $y$: clave pública de la **CA**.

El resultado obtenido de la función $BlinSign$ (que resulta de una interacción de un cliente con la entidad certificadora que lo registró) corresponde a la firma ciega $(\rho,\delta,\epsilon)$, y por otro lado la función $Verify-BlindSign$ retornará únicamente los valores $valid$ o $invalid$ que correspondan en caso de que la firma sea válida o inválida, respectivamente.

Al momento de iniciar una transacción el primer cambio reflejado que se ve, es que al momento de iniciar esta fase, se reemplaza el $client_{id}$ ($c_{bc}$) por un valor aleatorio, y que no atribuya ninguna identidad válida ($crand_{bc}$), y se procede a realizar la firma ciega (se omite la firma del esquema original, definida como $\sigma_{bc}$), lo cual implica ejecutar la operación $BlindSign$ (cuyo valor $M$ será la tupla $(crand_{bc}, o_{bc}, Payload, retryFlag)$ que produce la salida $(\rho,\delta,\epsilon)$), esta terna sera adjuntada en la transacción para la posterior verificación de la firma ciega.

El cambio sería el siguiente:
$< SUBMIT, c_{bc}, o_{bc}, Payload, \sigma_{bc}, retryFlag >$
$\xrightarrow[cambio]{} < SUBMIT, crand_{bc}, o_{bc}, Payload, retryFlag, (\rho,\delta,\epsilon) >$

Nota:

- $c_{bc}$: es el ID del cliente ($client_{id}$).
- $o_{bc}$: refiere a la operación que se desea ejecutar dentro del sistema distribuido.
- $Payload$: es la carga útil de la transacción enviada.
- $\sigma_{bc}$ es la firma del cliente.
- $retryFlag$: es una variable booleana que indica si el mensaje debe ser reenviado en caso de que la transacción falle.

### 6.4.2. Cambios en el Transaction Proposal

Este etapa inicia, validando la firma ciega de la transacción con la operación $Verify - BlindSign(M, (\rho, \delta, \epsilon), y)$ donde $M$ es el mensaje firmado, $(\rho, \delta, \epsilon)$ es la firma ciega realizada por el cliente, y el valor $y$ es la clave pública de la **CA**. En caso de que la función para verificar la firma retorne el valor *invalid* entonces se rechaza la transacción; en otro caso el nodo ejecutará la operación $o_{bc}$ sobre el $Payload$, y finalmente obtendrá los valores de $verDep$ y el $stateUpdate$. Por último el *submitting peer* genera el mensaje PROPOSAL para enviarlo a los *endorsers*.

El cambio sería el siguiente:
Sea el resultado de el *Transaction Proposal* $< PROPOSAL, m_c, trans_{prop} >$, donde:
$m_c = (c_{bc}, o_{bc}, Payload, \sigma_c)$
$\xrightarrow[cambio]{} m_c = (crand_{bc}, o_{bc}, Payload, (\rho, \delta, \epsilon))$
$trans_{prop} = (sp_{bc}, o_{bc}, ContentBlob, stateUpdate, verDep)$
$\xrightarrow[cambio]{}$ No sufre cambios.

Nota:

- $sp_{cb}$: es el ID del *submitting peer*.
- $crand_{cb}$: es el valor aleatorio que sustituye al id del cliente ($client_{id}$) que originó la transacción.
- $o_{bc}$ que contiene las operaciones a ejecutarse.
- $Payload$: que contiene datos para la ejecución de la transacción.
- $stateUpdate$: son las actualizaciones de estados que aplica la transacción.
- $verDep$: es la versión de dependencia, que contiene las variables involucradas en la transacción y su versión actual en el state y su respectiva operación.
- **$(\rho, \sigma, \epsilon)$:** está tripla representa los valores de firma ciega de Okamoto-Schnorr.

### 6.4.3. Cambios en el Transaction Endorsement

Este etapa comienza con el proceso de validación de la firma ciega utilizando la función $Verify - BlindSign$ con el mensaje $M = (crand||o_{bc}||Payload||retryFlag)$, la firma $(\rho, \delta, \epsilon)$, y la clave pública de la **CA** $y$. En caso de que la operación retorne $valid$, se continúa validando el $verDep$ y también el estado $stateUpdate$. En caso de que la $verDep$ corresponda a la última versión que el *endorser* poseé localmente, y el estado $stateUpdate$ es el mismo que tiene en el estado local, y además las políticas de seguridad $securityPolicies$ autorice a la operación $o$, entonces se endorsa el mensaje, sino se rechaza. Para endosar el mensaje, el *endorsement* crea una nueva firma ciega usando la función $BlindSign$ sobre un nuevo mensaje M que incluye el identificador de la transacción y un indicador de que la misma es válida. Luego de crear esta firma, el *endorsement* envía el mensaje al *submitting peer*.

El cambio sería el siguiente:
$< TRANSACTION - VALID, t_{id}, \sigma_{ep} >$
$\xrightarrow[cambio]{} < TRANSACTION - VALID, t_{id}, (\rho_{ep}, \delta_{ep}, \epsilon_{ep}) >$

Nota:

- $t_{id}$: es el ID del *Payload* de la transacción ($m.Payload.t_{id}$).
- $(\rho_{ep}, \delta_{ep}, \epsilon_{ep})$: es el resultado de la firma ciega del *endorsement* sobre el mensaje $M =< TRANSACTION - VALID||t_{id} >$.

#### 6.4.3.1. Discusión acerca de la firma ciega realizada por el endorser peer

En esta sección se trata de discutir si es necesaria que la firma del *endorser* sea una firma ciega, dado que, al momento de recibir la $trans_{prop}$ el *endorser* conocerá quién es el remitente de dicha transacción, por lo que para este esquema es requisito que el *submitting peer* confíe en que el *endorser* no revele esta información, pero a los efectos de la realizar la firma ciega el *endorsement* (es decir al momento de endosar y firmar la respuesta), sería únicamente útil si existiera alguna manera de vincular al *submitting peer* mediante los *endorsement* que han validado la transacción, esto es, visualizando el *blob* envíado por el *submitting peer* al *orderer service*, y de esa forma obtener información respecto a los *endorsement* que participaron en la fase de *transaction endorsement* de dicha transacción. En todo caso, se podría considerar que no sería útil la firma ciega de parte de los *endorsement*, ya que a efectos de la validación se enlentece las distintas fases del *transaction flow*, al requerir que cada *endorsement* interactúe con la **CA** previo a firmar el *endorser* de una transacción, sumado al hecho de que su no vinculación con la fase de *endorserment* de la transacción no aporta nada.

Un ejemplo en donde sería útil dicha firma, es si por ejemplo, un cliente utiliza regularmente un determinado conjunto de *endorsers*, ya sea por comodidad o por confianza que tiene, por lo que si esos *endorsers* firman con la firma habitual, sería válido verificar las transacciones realizadas en la ledger y ver qué *endorsers* han participado de la fase de *transaction endorsement* en las transacciones, y de esta forma poder obtener un vínculo, entre una transacción firmada ciegamente y el cliente, a pesar de no estar vinculada la identidad del cliente en la transacción.

### 6.4.4. Cambios en la etapa de Broadcasting to Concensus

En esta etapa, el *submitting peer* $sp_{bc}$ recibe las respuesta de los *endorsing peers*. Estas respuestas son recolectadas dentro de un arreglo llamado *endorsement*. Una vez que el *submitting peer* ha recolectado una cantidad suficiente de respuestas válidas en el arreglo *endorsement* (al menos 50 % + 1), envía la transacción a los *orderers* para que se incluyan en el siguiente bloque, y para esto utiliza la función *broadcast*. Esta etapa no tiene ninguna modificación con respecto a la misma etapa del *transaction flow* de Fabric.

### 6.4.5. Cambios en la etapa de Commitment

Por último, en la etapa de *Commitment*, se valida que la versión de dependencia *varDep* no ha cambiado durante el proceso de validación, y el arreglo *endorsement* es válido de acuerdo a las políticas de seguridad para la operación $o_{bc}$, para luego añadir la transacción en un nuevo bloque. El nuevo bloque *NewBlock* es un arreglo donde se guardan las nuevas transacciones para ser resueltas, y una vez que dicho bloque alcanza su tamaño máximo (*BlockLengthMax*, que es configurado en la red), el bloque es separado para ser commiteado por los nodos utilizando la función *Commit2Network*. En caso de que el proceso de validación falle, la transacción es rechazada, aunque de todas formas se registra en la *blockchain*, y el valor de *retryFlag* es colocado en *True*. En esta etapa tampoco habría cambios respecto al flujo habitual.

### 6.4.6. Integración en Fabric

BlindCons es instanciado en Fabric, ya que esta plataforma cumple con los requisitos previos, como el tener una configuración *permissionless blockchain*, entre otras. En el artículo no se deja del todo claro algunos puntos respecto a una posible integración, pero en este proyecto se procedió de todas formas con una implementación de la firma de Okamoto-Schnorr, y posteriormente se experimentó con una integración en Fabric de este algoritmo. El objetivo de esta integración era realizar una implementación completa de Blindcons. En el Apéndice A se detalla el trabajo realizado y una guía para realizar una implementación completa de Blindcons en Fabric.

## 6.5. Conclusiones

En el artículo, se prueban algunas propiedades de *Blindness* y *Unforgeability* basado en la hipótesis inicial que se tiene en Fabric, y también en las propiedades de Okamoto-Schnorr.
Es importante aclarar que el eje principal de BlindCons es el algoritmo de firma ciega, pero el protocolo fue diseñado para implementar algún esquema de firma ciega utilizando los componentes que tiene cualquier arquitectura de *permissioned blockchain*. Por lo tanto, el algoritmo realiza el proceso de cegado de la firma utilizando la **CA** que ya existe en el esquema *permissioned*, por lo que es eficiente para ser implementado en estos tipos de arquitecturas *blockchain*.
Blindcons a pesar de resolver el anonimato a nivel de transacción no ataca el anonimato a nivel de red. En otras palabras, BlindCons es vulnerable aún para el usuario, en el sentido de que todavía se puede hacer un seguimiento a través de la red (por ejemplo mediante la dirección de IP), haciendo que a nivel de red se rompa la propiedad de *unlinkability*.

## 6.6. AleaChain en BlindCons

En esta Sección se presenta el ejemplo de AleaChain utilizando la estrategia de BlindCons instanciada en Fabric, tomando en cuenta lo visto en la Sección 6.4. Al igual que en la Sección 5.6, al hablar de instancia, se habla de instancia a modo ejemplo, y no de una implementación bajo dicho esquema.

BlindCons bajo un esquema de Fabric, tendrá cambios determinantes, pero que en cierta forma serán análogos al ejemplo en Fabric. Es decir, se conservarán los actores y sus roles, como por ejemplo, que tanto los clientes como los desarrolladores serán los únicos *submiting peer*, y que los nodos de los casinos a los que se les llama *croupiers* (que bien son personas o nodos autómatas que dirigen los casinos) serán *endorsers peers*. El cambio se produce a la hora de ejecutar el *transaction flow*, debido a la aparición de un nuevo actor a la hora de firmar las transacciones: la autoridad certificadora.

Ocurre que, ahora cada envío de transacción, ya sea del *submmiting peer* a la hora de enviar el *transaction proposal*, o en la parte en dónde los *endorser peers* envían el *proposal response*, se necesitará de una interacción con la autoridad certificadora, para ejecutar el esquema de firma ciega de Okamoto-Schnorr [56]. Y así obtener una firma ciega. Es válido aclarar en este punto, que a la hora de enviarse el *proposal response* o el *proposal response*, o incluso en la *submit transaction*, el nodo que recibe dicha

información sabe quién la envía y evidentemente podrá inferir más allá de la firma ciega, de quién es lo que se envía debido a esto. Esta problemática no es atacada por esta solución, sino que lo que busca esta solución es evitar ataques pasivos por medio de la lectura de los datos persistidos. Lo que se desea en el fondo, es que esta información no quede en el registro final que tendrá el proceso completo, y es por esto que es útil el uso de BlindCons. También puede ocurrir que los usuarios revelen qué *endorsers* están firmando ciegamente el *submit transaction*, se puede afirmar que dicha información no es relevante, y si se desea, en la Sección 6.4.3.1 se discute sobre la relevancia de esto.

Visto lo anterior, se ve en la Figura 6.1 cómo quedaría definido el *transaction flow*, como idea central, se toma el esquema visto en la Sección 5.6, pero se agregan los pasos previos interactivos que ocurren con la entidad certificadora.

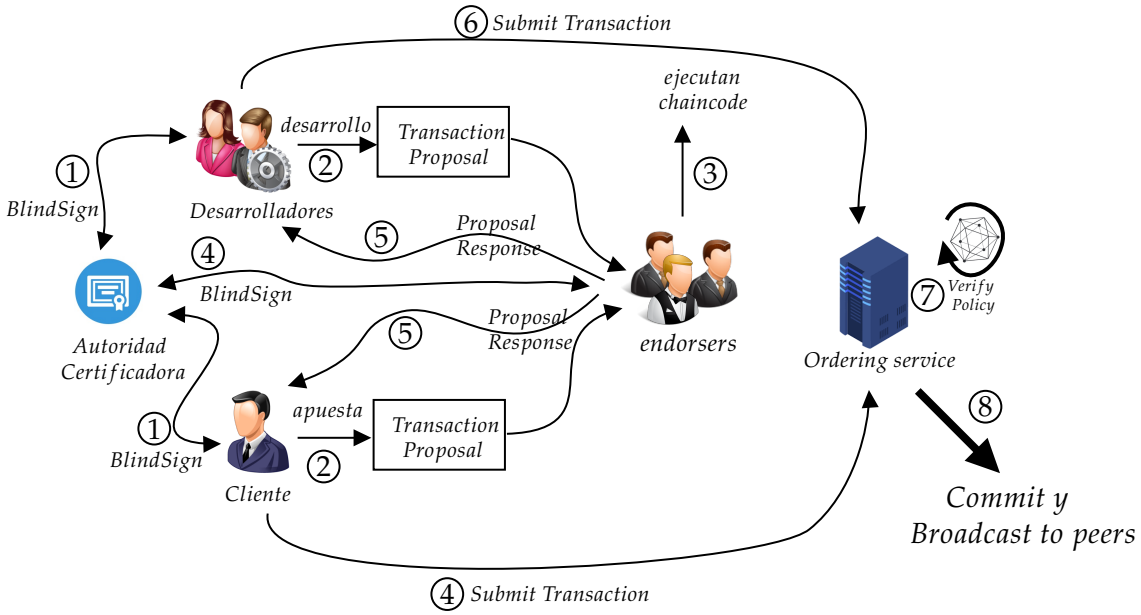


FIGURA 6.1: Transaction Flow de Hyplerledger Fabric bajo la estrategia de Blindcons, instanciando AleaChain

# Capítulo 7

# Privacy-preserving auditable token payments

En este Capítulo se presenta una variante que busca aumentar el nivel de privacidad en blockchains *permissioned*. La misma fue introducida por Androulaki *et al.* en el artículo "Privacy-preserving auditable token payments" [108]. El elemento central de este artículo es el manejo de tokens.

## 7.1. Problemática

En palabras coloquiales, un token, posee un propietario, y se definen dos operaciones para su manejo, una llamada *issue* (la cual permite generar nuevos tokens y agregarlos a la *ledger* asignándoles un propietario) y la otra *transfer* la cual permite a el poseedor de un token enviárselo a otro usuario en la plataforma. A partir de estas operaciones, se desean dos cosas, por un lado que a la hora de crear nuevos tokens, se desconozca a quién pertenece el token creado, y que a la hora de transferir un token, ocultar el propietario origen y el propietario destino de dicho token.

Además, se desea que todo el sistema pueda ser auditable (que puede llegar a ser útil, en el caso de que, por ejemplo un token sea una moneda, y se desee evitar que existan blanqueos de dinero o evitar la existencia de fraudes). Esta auditabilidad implica qué, cada usuario tendrá asignado un auditor, el cual ve toda la información de la transacciones relacionadas con ese usuario en particular sin el consentimiento explícito de este último.

## 7.2. Desafío y limitantes

Intuitivamente, lo que se desea, es que las soluciones planteadas, incluyan los requisitos como el de ser escalable, el de tener rapidez a la hora de ingresar transacciones, tener facilidad de uso, entre otros. Pero este artículo, gira particularmente, entre dos requisitos principales: la privacidad de los usuarios, y la posibilidad de ser auditable, a pesar de que esto implique dejar de lado alguno de los demás requisitos antes nombrados. En el artículo, se comenta algunos de los trabajos relacionados, y las desventajas de unos y de otros. Como que por ejemplo Zerocoin, logra anonimizar los tokens, pero no ofrece la posibilidad de auditar. O también, se nombra el caso de Zerocash, que ofrece anonimity y unlinkability, pero que requiere pasar por fase previa de configuración de parámetros (*trusted setup*), donde se debe de confiar en los participantes de dicha sesión, y posteriormente confiar en aquellos que generaron las claves del sistema es esa sesión, y en qué efectivamente se destruyeron

cuando terminaron. En particular, la estrategia también se basa en el modelo *Unspent Transaction Output* (UTXO) de Bitcoin.

## 7.3. Primitivas criptográficas

La estrategia planteada en el artículo utiliza una batería de primitivas criptográficos bastante amplia. A continuación, se expone la tabla 7.1 con los más relevantes y su uso aplicado:

| **Primitiva** | **Utilizado para** |
|---|---|
| Verifiable random functions (VRF) [64] y Serial numbers [11] | En particular Dodis-Yampolskiy [65], utilizadas para la creación de serial numbers para los nuevos tokens, que se utilizan para evitar el double spending. |
| Cifrado ElGamal [109] | Manejo de credenciales. |
| Firmas de Groth [110] | Vincular la clave pública del usuario, con la clave pública del auditor asignado. |
| Commitment de Pedersen [62] | Cada token se representa en un commitment, que contiene el identificador del propietario, el tipo de token, y la cantidad. |
| Firma Pointcheval-Sanders [111] | Implementada para certificar los tokens. |
| Pruebas de Groth-Sahai [112] | Son pruebas Non-interactive Zero Knowledge (NIZK), se utilizan para probar la correcta información de determinado commitment, sin revelarlo. |
| Firmas threshold [113] | Para la validación de la transferencia de tokens. |
| Zero Knowledge Proof [58] | Varios usos: Groth-Sahai, y para probar que un token pertenece a la *ledger*, y que el usuario está registrado. |

CUADRO 7.1: Primitivas utilizadas en la estrategia de Privacy-preserving token

## 7.4. Fundamentos de sistemas de tokenización

Los sistemas de gestión de tokens para redes de tipo *permissioned* cumplen con las siguientes propiedades:

- *Privacidad*: Las transacciones escritas en la blockchain ocultan tanto los valores que se transfieren, así como la relación entre emisor y receptor. La transacción no revela información sobre los tokens gastados en esta transacción, solamente revela el hecho de que son válidos y que no fueron utilizados antes.
- *Autorización*: Los usuarios autorizan transacciones a través de credenciales; es decir, la autorización para gastar un token está vinculada a la identidad del usuario en lugar de un pseudónimo (o dirección). La autorización utiliza credenciales anónimas (son esquemas de credenciales, que permiten al usuario autenticarse mientras muestra solamente los atributos que son necesarios en ese entorno determinado [105]) y preserva la privacidad.
- *Auditabilidad*: A cada usuario se le asigna un auditor que puede ver la información de la transacción relacionada con ese usuario en particular.

Cumplir estos requisitos, es importante para implementar cualquier tipo de sistema de tokens, que proteja la privacidad de los usuarios, y al mismo tiempo cumpla con posibles regulaciones.

### 7.4.1. Sistemas de tokens permissioned

En un sistema de tokens *permissioned* como es Fabric con el *framework* de FabToken, o Quorum, los usuarios poseen credenciales a largo plazo que refleja sus atributos y sus roles. Los tokens son introducidos por usuarios especiales llamados *issuers* a través de transacciones llamadas *issue* (o de emisión). También existen políticas de emisión las cuales definen qué *issuers* están autorizados para crear qué tokens y en qué condiciones.
De manera similar a las transacciones *issuers*, la transferencia también se puede validar según las políticas: la más simple de las cuales es que una transferencia solo puede realizarse entre usuarios registrados. Una propiedad fundamental de sistemas *permissioned* es que las transacciones se firman con credenciales a largo plazo. Como derivado de esto, las transacciones se pueden rastrear hasta su origen, haciendo cumplir así los requisitos de auditoría y rendición de cuentas.

### 7.4.2. Prueba de pertenencia basada en firmas

Se utilizan firmas para implementar pruebas de pertenencia de conocimiento cero (ZK membership proof). En términos generales, se considera un conjunto $S$ que consta de elementos que están firmados con una clave secreta $sk$ asociada con $S$. De ello se deduce que probar el conocimiento de algún elemento $e$ en $S$ mediante ZK es equivalente a *(i)* calcular un *commitment* de e, *(ii)* probar el conocimiento de una firma, calculada con el valor del *commitment* del paso $i$; el concepto de *commitment* implica realizar un bloqueo y ocultar cierta información, que más adelante se piensa utilizar (es decir se mostrará), pero que por el momento funcionará como garantía.
Se utiliza este mecanismo para dos propósitos: *(i)* para demostrar que un usuario está en el conjunto de usuarios registrados, *(ii)* para mostrar que un token está en el conjunto de tokens válidos, y sin utilizar tokens que ya fueron utilizados en la *ledger*.

### 7.4.3. Auditabilidad basada en cifrado

Para permitir la auditabilidad en este sistema, se cifra determinada información de las transferencias realizadas. Lo que se cifra en cada transacción son los datos del remitente (en particular la clave pública), destinatarios (en caso de ser una transacción tipo *transfer*), el tipo de los tokens y valores correspondientes, todo ello, bajo las claves públicas de los auditores del remitente y de los destinatarios (si los hubiera). Las pruebas *Zero-knowledge* en conjunto con el cifrado ElGamal permiten abordar este desafío de manera relativamente eficiente.

## 7.5. Modelo de arquitectura

El primer componente de la solución son los encoding token. Cada token es representado con un *commitment* (en el esquema se utiliza el *commitment* de Pedersen 3.6.2.1), que contiene la identidad del propietario del token, el tipo de token y su cantidad. El ciclo de vida del token está gobernado por dos tipos de transacciónes: *issue* y *transfer*. En una transacción *issue* se crea un token, se le asigna un tipo y también un valor, y luego se lo asigna a un usuario *issuer* (es decir, el creador de la transacción). Para que una transacción *issue* sea válida, debe ser presentada por un usuario *issuer* autorizado a crearla. Una vez que se crea un token, se cambia de propietario a través de la transacción de *transfer*. Dado que se trabaja dentro del marco UTXO, la transacción de *transfer* consiste en un conjunto de tokens de entrada que se consumirá, y

un conjunto de tokens de salida que se creará, y que se validará según las siguientes cinco reglas:

(1) El autor de la transacción es el propietario legítimo de los tokens de entrada.

(2) Los propietarios de los tokens de salida están registrados.

(3) Se conservan el tipo y el valor de los tokens.

(4) Los tokens de entrada pueden rastrearse hasta transacciones válidas en la *ledger*.

(5) Los tokens de entrada no se consumieron antes (para evitar el double spending).

Este conjunto de reglas, son de gran importancia para este esquema. A continuación se explica el porqué estos cinco puntos son determinantes a la hora de obtener una validación dentro de este esquema:

**(1)** El propietario del token debe poder demostrar que efectivamente el token a consumir es de su propiedad, si evitásemos esta validación se podrían dar casos de usuarios utilizando tokens de otros usuarios. El esquema representa los tokens como *commitments* de la forma $(issue, v, cm, \psi_0, s)$ (en caso de ser una transacción *issue*) o de la siguiente forma $(transfer, v, cm, \psi_1, \psi_2)$ (para las transacciones de tipo *transfer*) al momento de almacenarlas en la *ledger*. Ambos tipos de transacciones tienen una estructura similar, el primer valor referirá de qué tipo de transacción se trata, si de transferencias de token (*transfer*) o de creación de tokens (*issue*), luego se tendrá por un lado el valor $v$ (que contiene el tipo de token y su cantidad), $cm$ que es el propio *commitment*, los otros dos valores que nos restan en cada forma se explican a continuación:

- Issue: se tiene que en el caso de las transacciones *issue* el valor $\psi_0$ representa una prueba de conocimiento cero no interactiva (NIZK, visto en la sección 3.6.3) cuya semántica es de la siguiente forma $\psi_0 \leftarrow PK\{(r_{cm}) : open(crs, cm, r_{cm}, (v, P)) = true\}$ que comprobará que el *commitment* contiene la información esperada.

  Si el usuario satisface correctamente dicha información, entonces el *verifier* (que para este esquema y en este caso será el *certifier*) de la prueba se convencerá en todos los casos (propiedad *completeness*). Y en caso de que no la satisface entonces la probabilidad de que el *verifier* se termine convenciendo será lo suficientemente chica (propiedad *soundness*)).

  Es decir que por medio de *completeness* se garantiza que ningún usuario que es el legítimo propietario de un token se le niegue el derecho a transferirlo. Y por medio de *soundness* el *verifier* no será estafado al aceptar un token de un usuario.

  Por último para el caso de las transacciones *issue* el valor de $s$ es la firma sobre la tupla $(v, cm, \psi_0)$ que es lo que brinda que el mensaje ni la prueba han sido alterados y a su vez auténtica la información, brindando la propiedad de no repudio.
  Por lo que para la creación de un token ocurre lo siguiente: un usuario *issue* $I$, genera un nuevo *commitment* $cm \leftarrow (crs, (v, I); r_{cm})$ (siendo $crs$

el *common reference string*, $v$ el tipo y cantidad del token a crear, y $r_{cm}$ un string aleatorio secreto que sera utilizado para revelar el contenido del *commitment* como se describe en la sección 3.6.2), luego se generará la prueba $\psi_0$ como se describió anteriormente, y por último el usuario $I$ creará el valor $s$ que corresponde a la firma de $I$ sobre el mensaje $(v, cm, \psi_0)$, por último en la *ledger* se registrará la transacción $(issue, v, cm, \psi_0, s)$.

- Transfer: para la transferencia de un token (es decir un *commitment* $cm$) desde un participante $P$ (propietario del token) a otro participante $R$, la idea será transferir la propiedad de el *commitment* $cm$, para lo cual $P$ deberá generar un nuevo *commitment* $cm' \leftarrow (crs, (v, R); r'_{cm})$, generando a su vez una prueba NIZK $\psi_1$ que prueba que dicha información es correcta (incluyendo así también que $R$ es un receptor válido), y otra prueba NIZK en $\psi_2$ en donde se prueba que el usuario $P$ es el auténtico propietario de $cm$.

Se tiene que en el caso de una transacción *issue*, basta con probar la realización de la prueba NIZK $\psi_0$ que retornará *true* si el usuario $I$ es el propietario del *commitment*.
Para el caso de las transacciones del tipo *transfer*, se deberá crear un nuevo *commitment* $\psi_1$, que tendrá de propietario al usuario receptor del token, y a su vez se deberá probar que el usuario emisor de la transferencia es el propietario del token que corresponde al *commitment* $\psi_2$.

**(2)** Al momento de transferir un token, el nuevo propietario del token deberá ser un usuario válido, este hecho puede determinar en un blanqueo de monedas. Tomemos un ejemplo del mundo real, donde se tiene que un token representa algo que se encuentra en el mundo real, por lo que si se destina esa cantidad a un usuario inválido, no se puede reconstruir, es decir, se habrá perdido la posibilidad de poder realizar acciones con dicho token. Pues, un usuario inválido no podrá realizar ninguna acción con dicho token. Otro hecho que no es menor, es la imposibilidad de auditar, ya que cada usuario tiene asignado un auditor, para lo cual dicho auditor puede realizar seguimientos sobre el usuario, pero si se hace una transacción hacía un usuario inválido el cual no contará con un auditor, no se podrá construir el flujo "hacía atrás" de dicho token, debido a que no cuenta con un auditor autorizado a realizar el seguimiento. El mecanismo que verifica esto es una prueba NIZK, que se crea en la transacción *transfer* $(crs, (v, R), \psi_1, \psi_2)$ en donde $\psi_1$ es una prueba que verifica que el usuario $R$ está registrado y por tanto es válido.

**(3)** Una transacción de transferencia no podría destinar más (ni menos tampoco) tokens que los que entran. Esto es importante ya que es una de las problemáticas que atacan las blockchains. Esto generaría tokens "de la nada", haciendo así perder valor al token (que recordemos, representan unidades de la vida real, para lo cual la generación sin control de dichos tokens harían que se tuvieran inconsistencias graves). El motivo para que no se cambie el tipo de token, es exactamente el mismo. Incluso se puede afirmar que al tratarse de un sistema basado en transacciones UTXO, esto no debe ocurrir ya que viola dicha propiedad. El mecanismo que verifica esto es una prueba NIZK, que se crea en la transacción *transfer* $(crs, (v, R), \psi_1, \psi_2)$ en donde $\psi_2$ prueba que el usuario creador de la transferencia cuenta efectivamente con un *commitment* de valor $v$.

**(4)** Esta propiedad es muy requerida por las blockchains, agrega transparencia a las transacciones. Debido al esquema de *commitments* para el manejo de tokens, y a cómo esto se manejan, es decir, un token es creado a partir de una transacción del tipo *issue*, de allí puede solamente utilizarse en transacciones *transfer*, cambiando simplemente de propietario, se tendrá un registro de cada una de estas transferencias. Confiando que las transacciones en la *ledger* son inmutables, el hecho de probar que al momento de transferir un token (un *commitment*), basta probar que dicho *commitment* está en la *ledger*, y que se le atribuye un valor y un tipo, también correctos, y que hasta entonces no se ha gastado; este problema es resuelto mediante la cerificación. Recordemos que cada participante tiene asignado un *certifier* $C$, que otorga un vaucher que valida al usuario a utilizar su token esto se demuestra utilizando VRF y computando el serial number del token a la hora de utilizarlo.

**(5)** Aquí, lo que se quiere evitar es el *double spending* (doble gasto). Para ello se emplea el uso de pruebas estándar NIZK (en particular las pruebas de Groth-Sahai). Más precisamente, aprovecha la configuración tipo *permissioned* para usar pruebas de pertenencia (*membership proof*) basadas en la firma ZK para determinar que un usuario está registrado y que un token pertenece a la *ledger* de una manera que preserve la privacidad.

A la hora de emplear transferencias, los usuarios se contactan con una solicitud de certificación para garantizar la validez de los tokens que poseen.

Una solicitud de certificación contiene un token (es decir, el *commitment*) y al recibir dicha solicitud, el *certifier* verifica si el token está incluido en una transacción válida en la *ledger*. Si es así, el *certifier* firma ciegamente el token y la firma resultante se puede utilizar posteriormente para demostrar que el token es legítimo. Para evitar el doble gasto, se aprovecha el uso de los números de serie para identificar a los tokens cuando se consumen (como en Zerocash).

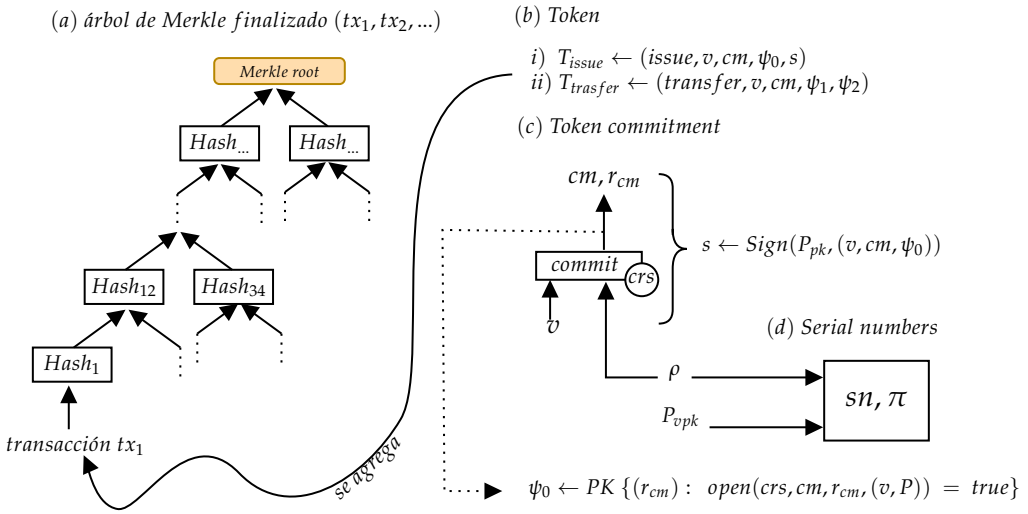


FIGURA 7.1: Ilustración del uso de commitmens del esquema de Privacy-perserving auditable token payments

Es importante que estos números de serie satisfagan las siguientes propiedades de seguridad: *(i)* Resistencia a colisiones: dos tokens resultan en dos números de

serial diferentes; *(ii)* Determinismo: el mismo token produce siempre el mismo número de serie; *(iii) Unforgeability*: solo el propietario del token puede producir un número de serial válido. Además se utilizan VRF (en particular se utiliza el esquema de Dodis-Yampolskiy) para generar los números de serie que son una función de la clave secreta del propietario del token junto con una aleatoriedad vinculada al token en el momento de su creación.

Para permitir la auditabilidad, se cifra la información en las transacciones de *transfer* (que contiene la información del remitente, los destinatarios, el tipo y los valores) bajo las claves públicas de los auditores del remitente y del receptor. Para acomodar casos de uso del mundo real, esta solución no asume un solo auditor para todos los usuarios. Esto significa que el esquema de cifrado no solo debe ser semánticamente seguro sino también de clave privada, por ejemplo, utilizando cifrado ElGamal.

### 7.5.1. Esquema de arquitectura

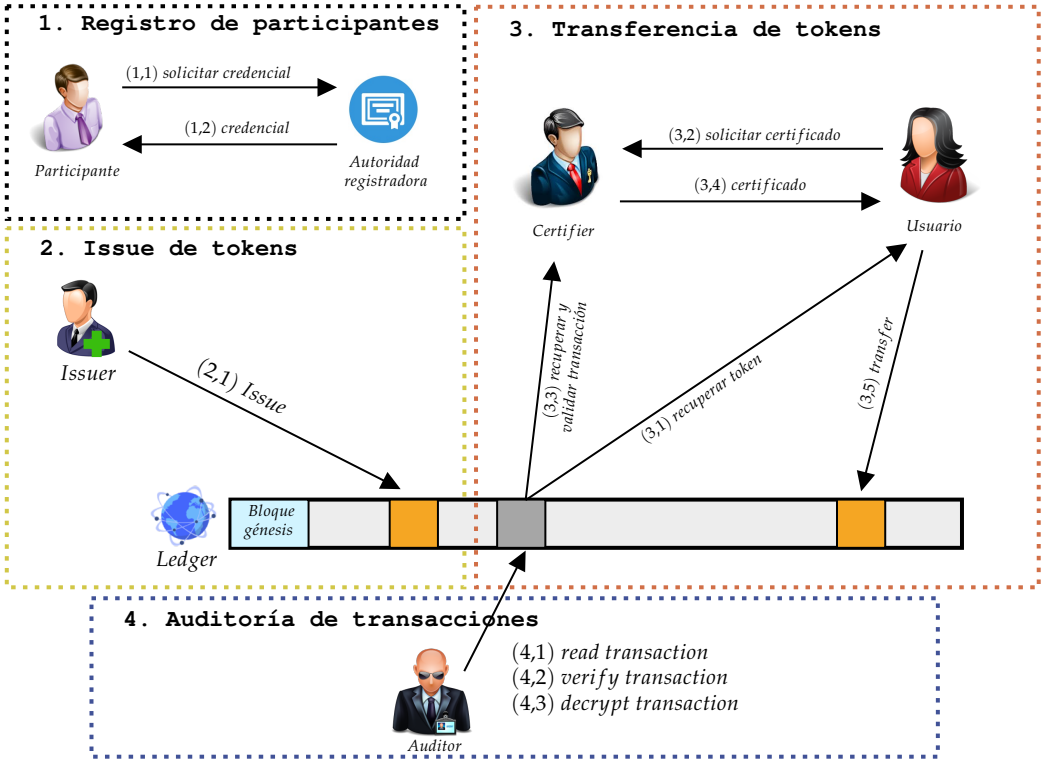


FIGURA 7.2: Esquema de interacciones de sistemas de tokens privacy-perserving

La figura 7.2 muestra las interacciones que ocurren entre los participantes dentro del sistema. Los usuarios, *issuers* , auditors, y *certifiers* obtienen credenciales interactuando con una autoridad registradora. Tras una decisión externa de crear nuevos tokens, uno o más *issuers* enviarán transacciones *issue* a la *ledger*, y la *ledger* automáticamente agrega las transacciones. Para transferir un token, un usuario contacta con un *certifiers* solicitando un certificado que hace referencia al token a firmar y a la transacción que lo creó. Si se trata de una transacción *issue*, el *certifiers* verifica si el autor de la transacción está autorizado. Si se trata de una transacción de transferencia, el *certifier* verifica si la prueba ZK es válida. Una vez que el propietario de un token recibe el certificado correspondiente, puede transferirlo a usuarios registrados.

Finalmente, los auditores asignados a un usuario pueden auditar las transacciones de ese usuario obteniendo acceso a la *ledger*.

### 7.5.2. Integración en Hyperledger Fabric

A continuación, se explican los posibles pasos para la integración con Hyperledger Fabric, la cuál se extrae del artículo de este Capítulo.

Como primer paso, se requiere que cada *issuer*, usuarios y auditor operen como clientes de Fabric. Estos clientes son utilizados para generar transacciones *issue* o transacciones transfers, enviar solicitudes de certificación de tokens, y leer de la *ledger*.

Siguiendo esta línea, se utilizan las operaciones criptográficas necesarias para generar transacciones de tokens a un chaincode prover con el objetivo de aliviar la carga en el cliente. Esta configuración supone que cada cliente posee un par en el que confía para el cálculo de las pruebas zero knowledge y los serial numbers.
Se afirma que esta es una suposición razonable, debido a que Fabric se enfoca en aplicaciones empresariales. También se utilizaría el protocolo de comunicación ya existente entre los clientes y los *peers* a implementar, al cual denotan como *certifier chaincode* .

Este chaincode, se ejecuta únicamente por un conjunto de *peers* previamente seleccionados en el momento de la configuración para certificar conjuntamente token válidos. Cada uno de estos *peers* está dotado de una parte de la clave de firma utilizada para certificar, y siempre que se invoca, proporciona su parte al código de la chaincode prover.

Finalmente, se aprovecha la infraestructura del MSP de Fabric, para otorgar identidades a largo plazo a los *issuers* y a los usuarios. En particular, se integraría el *identity mixer* (Idemix) MSP de Fabric, con esta solución para permitir la autenticación del usuario preservando la privacidad. Para asignar auditores a los usuarios, se utilizará un *channel out-off-band* para vincular las identidades idemix de los usuarios con las claves públicas cifradas de los auditores.

En una implementación real, esto podría ser ejecutado por un servicio de gestión de identidades externo, y preferiblemente distribuido. Se debe tener en cuenta que este protocolo utiliza la *ledger* como un servicio de *timestamping*, sin ninguna funcionalidad de validación, estos se transfieren indirectamente a *certifiers* y auditores. Esto podría ser soportado por Fabric directamente estableciendo cualquier política de endorsement que contemple la chaincode prover.

Se observa que se planea extender este prototipo para permitir que la *ledger* también valide las transacciones de tokens. Más concretamente, se pretende explotar el hecho de que Fabric admite la validación de transacciones pluggables, que en este caso, dicha validación consistiría en verificar las *zero knowledge proof*.

### 7.5.3. Cuestiones generales

Luego de haber transcurrido por varias secciones, en donde se introducen varios términos, y primitivas criptográficas complejas, a cualquiera le podría venir a la

mente varias preguntas. Preguntas del estilo más práctico, como por ejemplo: ¿qué tan fácil sería auditar? ¿Cómo un usuario corriente podría saber la cantidad de tokens que posee? ¿Qué pasa si un cliente pierde las credenciales?.

Estas preguntas no son abarcadas por el artículo, en el cual se basa éste Capítulo. Este estrategia ataca una problemática (en este caso la de otorgar privacidad en una red blockchain con configuración *permissioned*) y trata de probar que con determinada maquinaria es posible lograr esto. Pero además de eso, no se involucra en cuestiones del ámbito práctico, y se puede suponer a qué dejan libre esta parte para quién ponga en práctica el contenido del paper. A continuación, se verá, un esquema para el uso punto a punto, de la red blockchain.

La idea es partir de la implementación del sistema, bajo el esquema de Fabric, tal como se plantea en la Sección 7.5.2, para de esta forma tener todas las ventajas que se tienen con Fabric (la cual ya cuenta con una SDK para el manejo de consultas en la blockchain).

Una vez implementada la blockchain bajo esta estrategia, se tendrá una arquitectura cliente-servidor, en donde en la parte del servidor se cuenta con el entorno *Node.js* [96] (un lenguaje de programación de backend basado en Javascript) el cual interactúa con la SDK de Fabric para atender las diversas consultas, como se puede ver en la Figura 7.3.


FIGURA 7.3: Estrategia de Privacy-perserving token payments step-by-step

Como interfaz de usuario web (Web UI), se puede utilizar el *framework* de Angular, en donde el usuario podrá ver su registro de acciones y la posibilidad de interactuar con la blockchain. Angular se comunica con la aplicación servidor, para realizar dichas acciones.
A continuación, se verá la descripción del flujo de la Figura 7.3:

1– El usuario interactúa con la interfaz de usuario web Angular, chequear su estado de cuenta y realizar determinadas acciones (en principio si se trata de un usuario común, solamente podrá transferir tokens a otras credenciales o consultar su saldo)

2– La UI realiza llamadas a la aplicación Node.js que corre en el lado del servidor.

3– La aplicación Node.js del lado del servidor, realiza llamadas a la SKD de Fabric.

4– La SDK de Fabric interactúa enviando transacciones a la blockchain de Fabric (recordemos que las consultas también son transacciones).

Con esto, se puede afirmar que un entorno así, puede ser posible. Aún queda la pregunta de, ¿qué sucede si un usuario pierde las credenciales? y la respuesta en

crudo (es decir, sin contar con ningún sistema adicional), sería la misma que ocurre en Bitcoin: el usuario quedaría sin la posibilidad de volver a recuperar su cuenta.

En cuanto a las llamadas de auditoría, no es fácil dilucidar el peso que tienen las mismas leyendo solamente el artículo, sino que requiere de un estudio práctico sobre el tiempo de respuesta que se obtiene. De todas formas en el paper se comenta que utilizando cifrado ElGamal y mediante pruebas *Zero-Knowledge* se puede logra hacerlo de forma eficiente.

A continuación en la Figura 7.4 (ejemplo basado en una posible instanciación en Fabric, tal como se plantea en la Sección 7.5.2), para expresar mejor el último punto se expone un paso a paso, en donde dado un usuario $UserX$, su correspondiente auditor $A_x$ asignado, y una ventana de tiempo $X = Time_x - Time_y$, se muestran los pasos que ocurren hasta llegar a obtener toda la información de las transacciones del usuario $UserX$ dentro de la ventana de tiempo $X$.

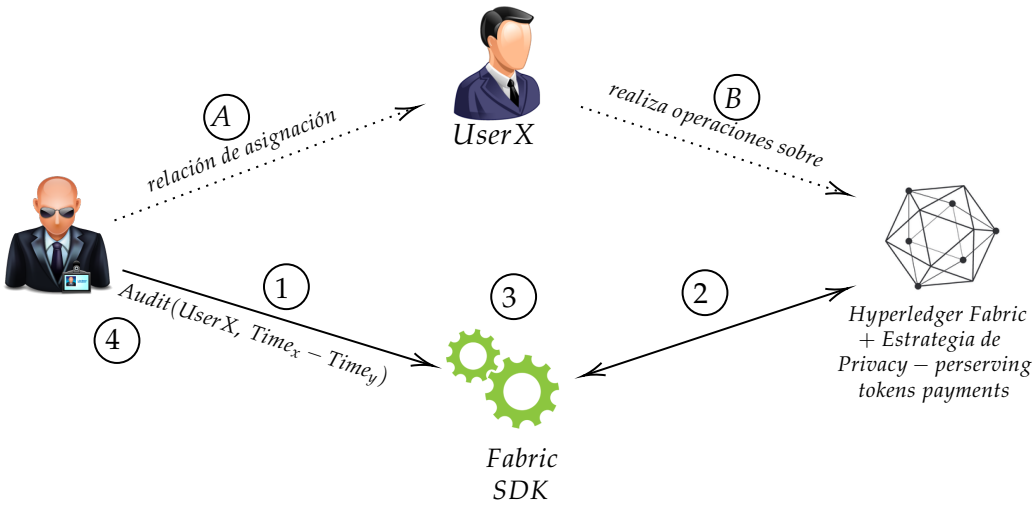


FIGURA 7.4: Step-by-step pedido de auditoria en Privacy-perserving token payments

Suponiendo que antes de ejecutar toda acción, el usuario Auditor ($A_x$) ha iniciado sesión en el sistema de forma satisfactoria, se detalla cada uno de los pasos a continuación:

$A-$ Expresa que el usuario auditor $A_x$ está vinculado al usuario $UserX$. Recordemos que el auditor asignado a un usuario es capaz de ver toda la información de las transacciones relacionadas al usuario asignado y sin el consentimiento explícito de este último.

$B-$ Expresa que el usuario $UserX$, es un usuario que realiza modificaciones al estado general de la blockchain.

Paso 1$-$ El usuario auditor ejecuta la operación $Audit(UserX,\ Time_x - Time_y)$ de la SDK de Fabric, (dicha operación no es nativa de Fabric, pero puede ser implementada utilizando funciones propias de la SDK).

Paso 2$-$ Este paso implica obtener todos las transacciones, ocurridas cronológicamente entre los tiempos $Time_x$ y $Time_y$. Debido a que existe un canal propio (out-of-band) entre el $A_x$ y $UserX$, pues es fácil obtener del estado general de la

blockchain todas las transacciones relacionadas al usuario $A_x$, y filtrar aquellas que no se encuentran en dicho intervalo de tiempo.

Paso 3− La SDK debe leer las transacciones, y luego debe verificar el estado de las transacciones (recordemos que pueden existir transacciones que se encuentran en la blockchain pero que a su vez son inválidas, y simplemente quedan como registro), y en caso de ser válidas, deberá descifrarlas debido a que se encuentran cifradas con la clave pública de $A_x$.

Paso 4− La SDK retornará al usuario $A_x$ las transacciones válidas, que cumplen con el criterio de búsqueda solicitado.

## 7.6. AleaChain en Privacy-preserving auditable token payments

En la estrategia presentada en el Capítulo 7, en la cual se puede observar que la arquitectura propuesta, posee un conjunto de roles de usuarios, para los cuales habría que volcar a la realidad de AleaChain.

Al igual que en el planteo de las dos estrategias anteriores, se contará con usuarios del casino o nodos del casino, que representan el papel de curiers en los casinos físicos. Pero además, se tiene un nuevo usuario, que juega un rol más administrativo, debido a que es creador de los token del sistema (que representarán las fichas) y que representan a los *issuer* del sistema, es decir, los únicos con la posibilidad de generar nuevos tokens mediante transacciones *issue*.

El rol del auditor, podrá ser ocupado por un usuario propio del casino, o de autoridades terceras que necesiten ver el cumplimiento de las diferentes medidas legales, y de forma tal de facilitar la explicación, se cuenta con un único usuario auditor (asignado a todos los usuarios).

De esta forma, estará cubierta la representación del sistema a nivel de fichas, quedando representado en la figura 7.5. Además, en la figura 7.1, se puede visualizar la representación de los roles en el sistema.

En detalle, solo el funcionario administrativo de Aleachain generará tokens en el sistema (detallado en la figura 7.5 como el paso 1) y luego se transferirán a funcionarios de Aleachain permaneciendo en la *ledger*, el usuario comprará tokens (se puede observar que en el paso 2, el usuario demuestra la posesión ante un funcionario de AleaChain), los cuales serán transferidos con la certificación de un funcionario de AleaChain (pasos 3 y 4), quedando así la transferencia de los tokens del cliente representados en el paso 5 (recordemos que los tokens, no se transfieren de forma literal, sino que se destruyen y se genera la misma cantidad pero con otro propietario), y todo esto podrá ser auditado por las autoridades.

Cabe aclarar que la definición de funcionario no necesariamente implica que será tarea de un usuario humano, sino que puede referirse a un nodo, propiedad de AleaChain y por tanto controlado por la misma, el cual cumple ese rol, ya sea automatizando transferencias, o generando nuevos tokens dentro del sistema. Incluso el rol del auditor puede llegar a ser un nodo autómata que controle determinadas

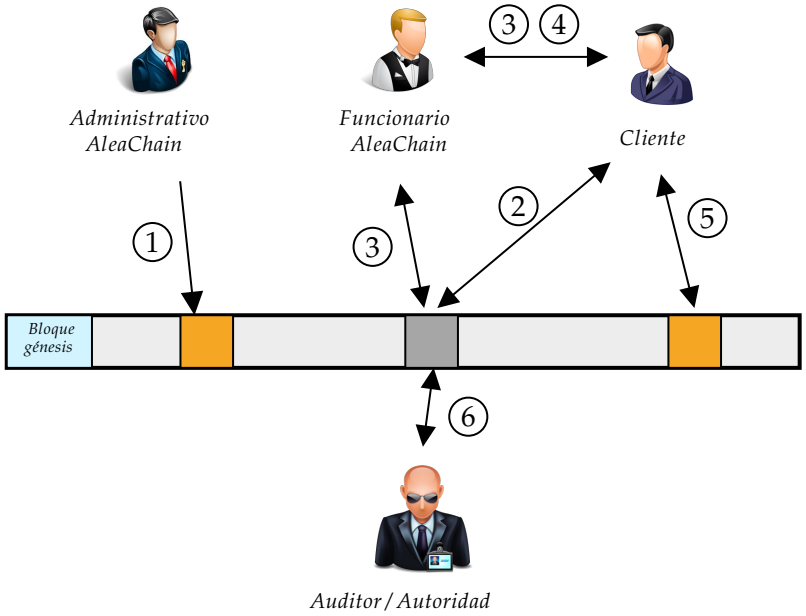


FIGURA 7.5: Representación del manejo de los tokens, en Aleachain

acciones del sistema.

Planteado entonces, el manejo de los tokens bajo esta estrategia, aún queda un problema: la estrategia de privacy-preserving tokens no maneja el término de smart contracts por sí mismo, pues se limita únicamente al manejo de tokens. Aquí para resolver este problema, podría hacerse otro planteo, tomando en cuenta lo que se expone en el artículo respecto a una posible implementación de la estrategia en Fabric (la cual se puede ver en 7.5.2).

Para implementar esta estrategia y utilizarla bajo el esquema de Fabric, se mapean los roles de la estrategia como clientes de Fabric.

Para esto se realiza, una "capa extra" a la arquitectura, la cuál registra como *smart contracts* (chaincodes en términos de Fabric) los pasos 1, 2 y 5 que se pueden ver en la figura 7.6. Es decir, las operaciones *transfer* e *issue* se contemplan como chaincodes.

Luego, mediante una *chaincodeissue* especial, llamada *chaincode prover*, se modelan los pasos 3 y 4.

Y continuando, restaría incluir la plataforma de apuestas, es decir los pasos 6 y 7. Estos pasos contemplarán el escenario de un cliente apostando en la plataforma determinada cantidad de fichas (mediante un *smart contract* correspondiente al juego de azar escogido) con el que participa con algún nodo perteneciente a AleaChain. De esta forma, el dinero apostado sufre un bloqueo en la cuenta del usuario y "pasa" a manos del *smart contract*. Luego de ejecutado el resultado del juego de azar, corresponderá, mantener esas fichas en dicho nodo, transferir más fichas al usuario, o retirarle las fichas al usuario, de forma tal que quede saldada la cuenta en términos de fichas respecto a lo jugado (ya sea perdido o ganado). Está operaciones serán de tipo *transfer* y/o *issue*.

Por último, el paso 8, queda contemplado, el cual es la asignación de auditores a los usuarios, mediante un canal de Fabric, que funciona out-of-band respecto a las

acciones realizadas. Esta estrategia, tiene la desventaja de que la *ledger* no validará ninguna de las acciones, sino que la *ledger* se utiliza como un servicio de timestamping, pero, que deja que estas validaciones caigan directamente en los nodos propios de AleaChain.

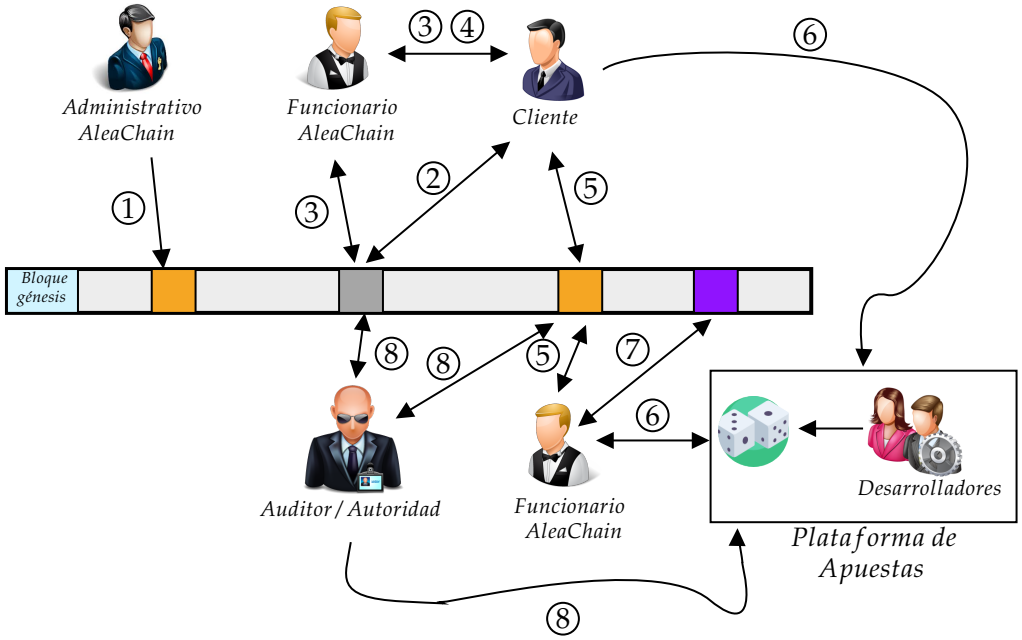


FIGURA 7.6: Representación del sistema AleaChain con plataforma de apuestas

# Capítulo 8

# Análisis y conclusiones

En este Capítulo se presentan las conclusiones obtenidas en este proyecto. Estas conclusiones están formadas por tres elementos independientes. Por un lado, en la sección 8.1 se presentan las conclusiones generales del trabajo. Por otro lado, en la Sección 8.2 se presenta una comparativa de las tres instanciaciones de *Aleachain* en las variantes descriptas a lo largo del proyecto. Por último, en la sección 8.4 se presentan posibles líneas de trabajo futuro.

## 8.1. Conclusiones generales

En este proyecto se presentó un análisis de diferentes alternativas que hacían uso de *permissioned blockchains* como elemento central y de diversas primitivas criptográficas para lograr en mayor o menor medida cierto grado de privacidad de los datos escritos en la *blockchain*.

En los capítulos 3 y 4 se presentaron todos los fundamentos teóricos que son necesarios para comprender los estrategias presentadas en los capítulos 5, 6 y 7. Específicamente el capítulo 3 se han presentado los conceptos criptograficos utilizados, mientras que en el capítulo 4 se han presentado todos los elementos que forman una *blockchain*.

En el Capítulo 5 se presentó de manera detallada el *framework Fabric*. La elección del framework utilizado en este trabajo estuvo condicionado a que las estrategias presentadas en los capítulos 6 y 7 lo utilizaban como parte fundamental de sus soluciones. El proyecto se vió guiado por estas dos estrategias, las cuales resuelven un mismo problema: proveer anonimidad en un sistema *blockchain* de configuración *permissioned*. Tomando como problema una realidad planteada en el Capítulo 2.

Como conclusión principal, se puede afirmar que las dos estrategias planteadas, superan ampliamente a la estrategia de Idemix que posee Fabric, esto en cuanto a los requisitos planteados durante el Capítulo 2. Pero Idemix con sus fallas, y defectos, hoy en día es una herramienta ya integrada a Fabric, y por tanto más cercano a producción, a diferencia de la estrategia de BlindCons o la de *Privacy-perserving* de tokens.

Si se continúa analizando las estrategias haciendo foco en las implementaciones de cada una de ellas, se tiene que la estrategia de *privacy-perserving* de tokens tiene una batería de primitivas mucho más amplia que la que tiene BlindCons, y por tanto, resulta más difícil su puesta en funcionamiento. Y si bien, este último punto, puede tratar de minimizarse mediante lo que propone el propio artículo de la estrategia

de tokens, es decir, utilizando Fabric como base, y de allí mapear las interacciones y roles, esto último generara un *overhead* mayor en comparación a BlindCons. En cambio, BlindCons solo aplica un cambio, que es en la manera de firmar, aunque dicho cambio implica una interacción adicional con una CA. De todas formas es de esperar que el cambio en BlindCons fuera considerablemente menor a la implementación de *Privacy-perserving* de tokens.

También es útil comentar, que debido a la falta de documentación, no es sencillo llevar a cabo las modificaciones propuestas por ambas estrategias, ya que requieren cambios a nivel del código fuente de Fabric.

## 8.2. Comparativas de estrategias

Los ítems comparativos se toman los que se consideran clave para medir las estrategias vistas son los siguientes:

- Transparencia: Se toma el ítem de transparencia en el sentido de qué tan sencillo es realizar una trazabilidad en las transacciones (no a nivel de monedas).
- Seguridad: Se hace énfasis en los modelos de seguridad criptográficos en los que se hacen las pruebas de seguridad cada estrategia. Como punto a tener en cuenta, diremos que el modelo random oracle al ser utilizado en *blockchain* (por ejemplo en el manejo de direcciones de hash), utilizarlo en otro punto, no implica degradar el sistema.
- Eficiencia: Hace referencia a qué tanto puede degradar el sistema una posible implementación de la estrategia en Fabric, en base a la cantidad de iteraciones, u operaciones adicionales que se deben de realizar.
- Privacidad: Hace referencia a qué tan vinculado queda el usuario creador de una determinada transacción a dicha transacción.

### 8.2.1. Comparativa en transparencia

| Estrategia | Diferencias/Características |
|---|---|
| Fabric/Idemix | Fabric permite configurar el acceso de forma tal que todas las transacciones de la *ledger* sean visibles por todos los participantes. Pero de forma nativa no puede ocultar a sus autores. Con Idemix esto es posible y de esta forma se oculta el autor, no así el contenido. |
| Fabric/BlindCons | Utilizando la estrategia implementada en Fabric, vale lo mismo que con la estrategia de Idemix, pero en vez de mezclar atributos de identidades, se utiliza la firma ciega de Okamoto-Schnorr, en la firma de la transacción. |
| Fabric/Tokens | A nivel de configuración, solo los auditores poseerán los permisos necesarios para ver transacciones, por ejemplo, cada auditor tendrá el permiso de lectura para todos los elementos de la *blockchain* relacionados con el usuario al que audita. |

### 8.2.2. Comparativa en seguridad

| Estrategia | Diferencias/Características |
|---|---|
| Fabric/Idemix | Idemix un esquema de firma basado en *pairings* que fue propuesto en [104, 114]. En la implementación actual, se utiliza *zero knowledge proof* para probar el valor de una firma, utilizando *bilinear maps*, y curvas elípticas, para el desarrollo de las pruebas de seguridad se utiliza *generic group model*. |
| Fabric/BlindCons | La seguridad se basa en usar firmas ciegas en las transacciones en vez de firmas convencionales. El algoritmo utilizado cuenta con las propiedades de *blindness* y *unforgeability*, pero en particular, las pruebas de seguridad de este algoritmo se realizan haciendo uso de *Random Oracle Model* (lo cual no es un problema en *blockchain*), y además el encontrar un algoritmo eficiente capaz de romper alguna de estas dos propiedades equivale a encontrar un algoritmo eficiente que resuelve el problema del logaritmo discreto, el cual se presume *difícil*. |
| Fabric/Tokens | En cuanto a seguridad la estrategia de *Privacy-perserving* sobre el manejo de Tokens, posee una batería de primitivas criptográficas, que van desde algoritmos basados en el logaritmo discreto o en *pairing settings*, y hasta algoritmos *structure-preserving* como lo son Dodis-Yampolskiy VRF, cifrado ElGamal, firmas de Groth, commitment de Pedersen, y firmas Pointcheval-Sanders. En principio se podría apelar a que son estructuras que logran seguridad bajo el supuesto estándar, y que si bien no existe una implementación que las pruebe en conjunto, se tiene elementos como el uso de *serial numbers* implementado en Zerocash, que sí tienen una implementación ampliamente utilizada, lo que puede ser considerado un respaldo adicional. |

### 8.2.3. Comparativa en eficiencia

| Estrategia | Diferencias/Características |
|---|---|
| Fabric/Idemix | Si bien Idemix utiliza el estándar X.509, que permite verificación de claves de forma rápida, ocurre que en Idemix es necesario utilizar un certificado nuevo X.509 por cada vez que se realiza una transacción, lo que genera una administración de claves compleja y una sobrecarga de comunicación y almacenamiento. |
| Fabric/BlindCons | Existe un tiempo de carga extra, en la iteración que tiene un peer y la CA a la hora de enviar el *transaction proposal*, esta iteración es bloqueante, es decir, que hasta no obtener respuesta el usuario no podrá proceder con la transacción. Esto no solo suma un tiempo de espera extra, sino que vuelve menos escalable el sistema, debido a que la CA tendrá que atender más pedidos de lo normal. Teniendo que optar por incluir más CA y por ende un manejo más complejo de los certificados. |
| Fabric/Tokens | La implementación en Fabric, implica agregar una nueva capa lógica al sistema, es decir, la utilización de los roles de la estrategia, bajo el esquema original de Fabric, agrega un overhead extra al sistema. De todas formas, los esquemas utilizados permiten utilizar de forma relativamente eficiente las pruebas de Groth-Sahai. |

### 8.2.4. Comparativa en privacidad

| Estrategia | Diferencias/Características |
|---|---|
| Fabric/Idemix | Idemix proporciona *anonimity* (envío de transacciones sin tener que revelar su identidad) y *unlinkeability* (envió de múltiples transacciones sin revelar que todas las transacciones provienen de la misma fuente). Como punto en contra, se tiene que Idemix no puede aplicarse a nodos endorsement, además, hay casos en los que es importante que ni siquiera la CA encargada de emitir los certificados pueda vincular todas las transacciones al usuario, e Idemix depende de las CA para la emisión de certificados. |
| Fabric/BlindCons | Aplica lo mismo que con la estrategia de Fabric/Idemix. |
| Fabric/Tokens | La implementación, contempla la privacidad a nivel de tokens, o el uso de la moneda de apuesta tanto *anonimity*, y *unlinkeability*, pero no tiene una estrategia para los *smart contracts*, por lo que se requerirá utilizar otro artificio a la hora de otorgar privacidad a los *smart contracts*. |

* Transversal entre las tres soluciones, y respecto a la privacidad: no existe anonimidad a nivel de capa de red al enviar las transacciones.

### 8.2.5. Conclusiones sobre la implementación

Como se presumía desde el comienzo del desarrollo de las dos implementaciones, la implementación escrita en Go obtuvo rendimientos superiores a la desarrollada en Python en todos los *benchmarks* ejecutados. Esto puede explicarse fácilmente por el tipo de lenguaje que es cada uno, por un lado tenemos que Go es un lenguaje compilado mientras que Python es un lenguaje interpretado.

A pesar de describir la plataforma de cómputo, el software utilizado, y los resultados obtenidos, queda como trabajo futuro la construcción de un contenedor que contenga todas las bibliotecas requeridas para la ejecución del código fuente. Esto permitiría a cualquier interesado en volver a ejecutar el código bajo unas condiciones similares a las descritas en este proyecto. Debido a que esto último es una de las características más importantes de la investigación científica en todos sus ámbitos.

## 8.3. Conclusiones en base a los objetivos propuestos

En la Sección 1.4, se establecen los objetivos propuestos al comienzo de este proyecto. A continuación, se analiza el grado de cumplimiento de estos objetivos.

Diremos entonces que se cumple el objetivo de abarcar en el conocimiento teórico sobre criptografía y *blockchain*. También se profundizó sobre el conocimiento del anonimato. Se estudió la plataforma Fabric, con algún tinte de implementación en el medio, que dio paso a obtener un conocimiento más allá del teórico, en donde se dedicó tiempo a entender conceptos y estudiar artículos que ayudaron a guiar la continuidad del proyecto. Se estudiaron dos estrategias alternativas para otorgar anonimidad, llegando a comprender el enfoque de ambas estrategias, sus ventajas y desventajas. Además, el caso de estudio empleado, propuesto por los autores, ayudó también a obtener un conocimiento adicional, debido a que al usarlo como motor, fue abriendo el camino para seguir investigando y contestar dudas o cuestiones que

escapaban de lo propuesto inicialmente, como por ejemplo las limitaciones o entender mejor ciertas cosas, dejando así, un aprendizaje no pensado. Uno de los objetivos propuestos pero que no pudo completarse debido a la dificultad fue la implementación BlindCons en Fabric, de todas formas el conocimiento obtenido en base a lo realizado en ese objetivo fue algo satisfactorio, dado a que se experimentó la limitación del sistema y ayudó a entender parte de su funcionamiento.

## 8.4. Trabajo futuro

Existen múltiples caminos para continuar este trabajo, o al menos, de avance tomando como base lo realizado en este trabajo. Una vía, puede ser la de explorar otras estrategias que abarquen el mismo problema que se presenta en este proyecto, es decir utilizar otro mecanismo que otorgue privacidad y anonimato en las transacciones realizadas en una *blockchain permissioned*. Algunos de estos mecanismos podrían ser el uso de otras primitivas criptográficas, como lo son el cifrado homomórfico, Secure Multi-Party Computation, o otro tipos de pruebas de conocimiento cero (como las zk-STARK).

Existe también el camino de implementar alguna de las dos estrategias presentadas en este trabajo, que para el caso de BlindCons se podría esperar a una documentación más detallada por parte de Fabric para así poder llevar a cabo una implementación funcional, además de lo expresado en la Sección A.6.3. Realizar estas implementaciones sería de gran utilidad a la hora de comparar las dos estrategias, ya que se pasaría de una comparación teórica a una comparación práctica. En esta comparación práctica entrarían en juego aspectos como el rendimiento, la latencia a la hora de crear una nueva transacción, la cantidad de usuarios concurrentes en la plataforma, etc.

Otra vía adicional, sería la implementación o estudio de anonimidad a nivel de capa de red, esto podría llegar a solucionarse, por ejemplo, utilizando *onion routing*.

# Apéndice A

# Implementación de BlindCons en Hyperledger Fabric

Este apéndice, describe el trabajo realizado para implementar BlindCons en Hyperledger Fabric. Uno de los objetivos planteados al inicio del proyecto fue el de implementar (aunque sea en parcialmente) alguna de las técnicas tratadas en este proyecto. Por lo que se tomó la idea de cambiar la firma convencional de Hyperledger Fabric por la de firma ciega de Okamoto-Schnorr del artículo de BindCons [103]. Es oportuno aclarar que la versión de Fabric utilizada para la implementación es la release 1.4. [76].

## A.1. Proceso de implementación e implantación

Preliminarmente se implementó el mecanismo de firma ciega de Okamotro-Schnorr en los lenguajes de programación de Python y en el de Golang [115].

La justificación de esto, es que la idea al comienzo fue implementar de forma realista el algoritmo de Okamoto-Schonorr en un lenguaje de programación amigable y familiar de utilizar. Para lo cual se optó por Python, debido a la experiencia con la que se contaba en este lenguaje. Luego de implementada esta primera versión, se desarrolló el algoritmo en el lenguaje de programación Golang, debido a que es el lenguaje que utiliza Fabric en su arquitectura.

En el resto de esta Sección, se explican los cambios que son requeridos a nivel de una transacción en Fabric para ejecutar un cambio de este porte. El tiempo aproximado que tomó, documentar, diseñar e implementar esta implementación, fue de aproximadamente cinco meses.

### A.1.1. Cambios a nivel de transacción

En el Capítulo 6 se revisaron los cambios sufridos a nivel de transacción (sobre el *transaction flow*) al llevar a cabo la idea de BlindCons. En el siguiente diagrama se describe el protocolo usado en Fabric para modificar la ledger al enviarse una transacción desde un cliente, pero teniendo en cuenta la firma ciega que realiza el *submiting peer* a la hora de firmar la transacción.

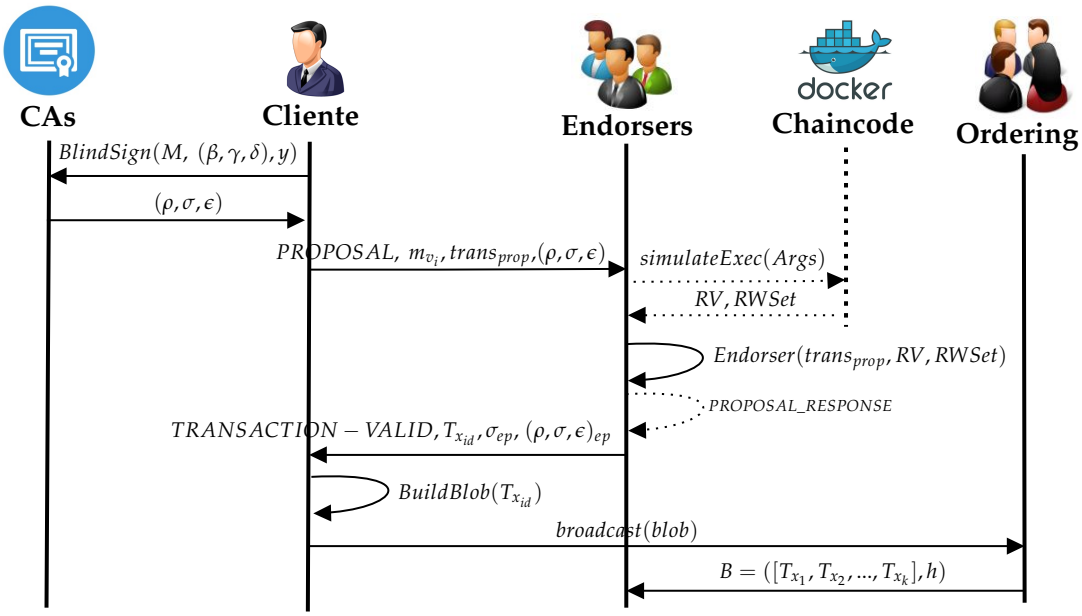


FIGURA A.1: Transaction flow, en Hyperledger Fabic con el uso de Okamoto-Schnorr como firma

En la Figura A.1 se ve el *transaction flow* habitual, pero se agrega un paso previo, en el cual se interactúa con la CA para obtener la firma ciega. A continuación, se explica esta figura a modo repaso de los visto en el Capítulo 6. El flujo envuelve los siguientes pasos:

1. El cliente solicita un *Commitment* a la CA para generar una firma ciega invocando la primitiva *BlindSign*, para ello envía los valores aleatorios escogidos.

2. La CA retorna los valores de firma ciega al cliente.

3. El cliente envía el mensaje $PROPOSAL$ a los *endorsement peers* con su firma ciega ya establecida dentro del cabezal del Proposal.

4. Cada *endorsement peer* que reciba el $PROPOSAL$ validará la firma ciega enviada, en caso de no ser válida la transacción es rechazada por el *endorser peer*. En otro contrario, invoca apropiadamente el *chaincode* (es decir simula la transacción en su ledger local almacenada en un contenedor Docker) utilizando los argumentos dados por el cliente en el $PROPOSAL$ y obteniendo el RWSet (es decir, una propuesta de actualización del *world state*. Obteniendo los valores de $verDep$ y $updateState$, es decir la información del estado, y los valores y versiones de las distintas variables).

5. El *endorsement peer* realiza el *endorser* de la ejecución y sus resultados, creando un *ProposalResponse* y lo envía de vuelta al cliente, que básicamente es retornar el $trans_{prop}$ y el conjunto $RWSet$ validado y firmado ($TRANSACTION - VALID$). Esta firma es la firma ciega propia del *endorser peer*, y acordada de forma previa con la CA (de igual forma que en el paso 1). Aquí se puede dar paso a la discusión, de, si es necesario que los *endosers peer* utilicen una firma ciega, viste en la Sección 6.4.3.1.

6. El cliente ahora recolecta las transacciones ya validadas y firmadas por los *endorsers peer*, construyendo así un arreglo con todas estas respuestas a la que

llamamos *endorsements* luego se construye el *blob*, el cual está compuesto por el $tras_{prop}$, el arreglo de *endorsements* y la firma del cliente.

7. El *ordering service* se encarga de recibir transacciones, validar las firmas, determinar la correctitud de cuales son válidas, para luego empaquetarlas dentro de un bloque y de hacer un *broadcast* del bloque hacia toda la red.

8. Los *peers* reciben el bloque ordenado, lo validan y hacen un *commit* en su *blockchain* con el nuevo bloque.

### A.1.2. Cambios a nivel de bloque

En esta Sección, se verá en forma de pseudocódigo, los cambios necesarios a nivel de bloque en cada uno de los pasos explicados en la Sección anterior:

**Nota**: En color rojo se encuentran los campos que contienen los cambios que aplica BlindCons, respecto al diseño original que posee Fabric.

**Paso 1 y 2**: El rol de la CA a efectos de la interacción lo hará la MSP, que será la interfaz intermediaria del proceso. El cliente utilizará la primitiva *getCommitment*, el cual será atendido por la MSP la cual le solicitará a la CA el valor $a$ correspondiente al esquema de firma de Okamoto-Schnorr descrito en la sección 3.5.3. El cliente luego de obtener dicho valor generará los valores aleatorios requeridos por el esquema, y realizará a su vez las operaciones para generar el valor $e$ que enviará al MSP con la primitiva $getBlindSign(e)$. Una vez realizado esto, el MSP interactúa con la CA para retornar los valores $R$ y $S$ al cliente, para que el cliente pueda finalizar el proceso de firma.

**Paso 2**: La estructura del SignedProposal tendrá la siguiente forma:

```
SignedProposal {
    Proposal Proposal {
        ChannelHeader ChannelHeader {
                ChannelId string
                ChaincodeName string
                ChaincodeVersion string
                TxId string
        }
        SignatureHeader SignatureHeader {
            ClientID []byte
            Nonce []byte
        }
        Args [][]byte
    }
    Signature []byte
}
```

Aquí, los cambios que veremos son, que el valor $ClientID$ ($client_{id}$) será sustituido por el de $crand_{bc}$, es decir una tira numérica aleatoria la cual no vincula dicho valor numérico con ninguna identidad existente en el sistema. Posteriormente el $SignalProposal$ es firmado, y su valor es almacenado en el campo *Signature*, que en este caso se tratará de la concatenación de los valores de la firma ciega, es decir $(\rho||\sigma||\epsilon)$.

**Notas:**

- *ChannelId*: es el nombre del *fabric channel*, que es el objetivo de esta transacción, un canal es una forma de crear más de una instancia lógica de la ledger/world state y un *chaincode namespace* dentro de una sola instancia física de fabric.
- *ChaincodeName* y *ChaincodeVersion*: Son el nombre y versión del *chaincode* invocada durante este proposal.
- *TxId*: Es el identificador de la transacción, que es el hash del SignatureHeader.
- *RandomNumber*: es el valor aleatorio que se ubicará en el campo. Creator, que incluye el certificado del nodo creador de la transacción.
- *Nonce*: es un arreglo de bytes pseudoaleatorio.
- *Args*: es el conjunto de argumentos para la invocación de esta *chaincode*.
- *BlindSignature*: es la firma cegada, luego de la interacción de los pasos 1-4.

**Paso 3**: En este paso, el *endorsement peer* luego de validar la firma del cliente, contacta al contenedor Docker y solicita ejecutar el *chaincode* basado en los argumentos suministrados por el paso 2 (*Args*). El *chaincode* se ejecuta y puede realizar la llamada *GetState* para leer el *world state* actual.

Cualquier llamada para modificar el *world state* (con las operaciones de *PutState* o *DeleteState*) no afecta el *world state* porque se harán en una copia local dentro del contenedor; además, realiza un seguimiento de los cambios que tendrá esta propuesta si se agregará al *world state* dentro del *RWSet*. El *chaincode* devuelve al par el *RWSet* y un valor de retorno.

**Paso 4**: El *endorsement peer* utiliza el *proposal*, su propia clave privada (para firmar) y el resultado de la ejecución del *chaincode* para generar el *ProposalResponse* que dará como retorno al cliente:

```
ProposalResponse {
        ProposalResponsePayload ProposalResponsePayload {
            ProposalHash []byte
            ChaincodeName string
            ChaincodeVersion string
            RWSet []byte
            RV []byte
        }
        Endorsement Endorsement {
            Endorser []byte
            Signature []byte
        }
    }
```

El *endorser* firmará mediante firma ciega el *ProposalResponse*.
**Notas:**

- *ProposalHash*: es el hash del proposal.
- *Endorser*: es la identidad serializada del *endorsement peer*.
- *Signature*: es la firma sobre el (payload del ProposalResponse || Endorser).

**Paso 5**: El cliente usa el *ProposalResponse* para crear la transacción:

```
Transaction {
    Payload Payload {
        ChannelHeader ChannelHeader {
            ChannelId string
            ChaincodeName string
            ChaincodeVersion string
            TxId string
        }
        SignatureHeader SignatureHeader {
            ClientID []byte
            Nonce []byte
        }
        TransactionAction TransactionAction {
            Args [][]byte
            ProposalResponsePayload ProposalResponsePayload {
                ProposalHash []byte
                ChaincodeName string
                ChaincodeVersion string
                RWSet []byte
                RV []byte
            }
            Endorsements []Endorsement {
                Endorser []byte
            }
        }
    }
    Signature []byte
}
```

**Notas**:

- *ChannelHeader* y *SignatureHeader*: son los mismos que en el Proposal.
- *Args*: es el mismo que en la ejecución del *chaincode*.
- *ProposalResponsePayload*: es el mismo que el *ProposalResponse* de este *chaincode*.
- El arreglo *Endorsements* es rellenado con el campo del mismo nombre que viene en el *ProposalResponse*.

**Paso 6**: El *orderer server* usa algoritmos internos para crear una secuencia ordenada de transacciones. Las transacciones en el bloque se etiquetan como válidas o no válidas.

**Paso 7**: Los peers reciben el bloque del orderer:

```
Bloque {
    BlockHeader BlockHeader {
        Numero uint64
        HashPrevio []byte
        DataHash []byte
    }
    OrdererSignature OrdererSignature {
        SignatureHeader SignatureHeader {
            Creator []byte //Es el id del Orderer no del peer
            Nonce []byte
        }
        Signature []byte
    }
    Transactions []Transaction {
        Payload Payload {
            ChannelHeader ChannelHeader {
```

```
                ChannelId string
                ChaincodeName string
                ChaincodeVersion string
                TxId string
            }
            SignatureHeader SignatureHeader {
                      ClientID []byte
                   Nonce []byte
            }
            TransactionAction TransactionAction {
                Args [][]byte
                ProposalResponsePayload ProposalResponsePayload {
                    ProposalHash []byte
                    ChaincodeName string
                    ChaincodeVersion string
                    RWSet []byte
                    RV []byte
                }
                Endorsements []Endorsement {
                    Endorser []byte
                    Signature []byte
                }
            }
        }
        Signature []byte
    }
}
```

**Notas**:

- *Numero*: es el número del bloque.
- *HashPrevio*: es el hash del bloque previo.
- *DataHash*: es el hash de la porción de datos del bloque.
- *OrdererSignature*: contiene la firma sobre el BlockHeader y la OrdererSignature. SignatureHeader.
- *Transactions*: es una arreglo con todas las transacciones del bloque.

Los *endorsement* validan lo siguiente:

- Que no existan dos transacción confirmadas en la ledger con el mismo $T_{id}$.
- El arreglo *Endorsement* cumple con las políticas de endoserment especificadas en los canales de *chaincode*.
- Cada firma en $Endorsement[i]$ son firmas válidas.
- ProposalHash es el hash de la concatenación de: Payload.ChannelHeader, Payload.SignatureHeader y Payload.TransactionAction.Args

### A.1.3. Proceso de integración de la firma ciega en Hyperledger Fabric

Como se explicita en el comienzo de este apéndice, la versión utilizada de Fabric para emplear los cambios fue la v1.4. Descargando dicha versión, desde su proyecto oficial localizado en GitHub, y en particular su rama master, se deberá hacer foco en dos carpetas. La primera de ellas, es el módulo de algoritmos, y si se desea adicionar módulos criptográficos se recomienda incluirlos en dicho módulo ./fabric-master/bccsp, aquí también se recomienda incluir el módulo desarrollado en Golang de A.3 que tiene el mismo nombre y volcarlo en ese mismo directorio. Y por

segundo, tendremos la carpeta que posee las definiciones de los peers, y sus operaciones, en el módulo .fabric-master/vendor/github.com/hyperledger, en la cual se encuentran la mayoría de las estructuras de Fabric, de esta última carpeta, se deja a continuación su estructura de directorios, para facilitar la explicación:

```
fabric-master/vendor/github.com/hyperledger
├── fabric-amcl
│   └── amcl
│       └── FP366BN
├── fabric-chaincode-go
│   ├── pkg
│   │   └── statebased
│   ├── shim
│   │   └── internal
│   └── shimtest
├── fabric-config
│   ├── configtx
│   │   ├── internal
│   │   │   └── policydsl
│   │   ├── membership
│   │   └── orderer
│   └── protolator
│       └── protoext
│           ├── commonext
│           ├── ledger
│           │   └── rwsetext
│           ├── mspext
│           ├── ordererext
│           └── peerext
├── fabric-lib-go
│   └── healthz
└── fabric-protos-go
    ├── common
    ├── discovery
    ├── gossip
    ├── ledger
    │   ├── queryresult
    │   └── rwset
    │       └── kvrwset
    ├── msp
    ├── orderer
    │   └── etcdraft
    ├── peer
    │   └── lifecycle
    └── transientstore
```

FIGURA A.2: Estructura de directorios de Hyperledger Fabric

Se describen los cambios en base a los pasos descritos en la sección anterior.

**Paso 1**: Este paso es posiblemente el que más cambios implique, debido a que cambia completamente el proceso que posee Fabric, ya que, el usuario antes de crear el *Transaction Proposal*, deberá interactuar ancons con la CA en dos ocasiones a raíz de aplicar la firma ciega de Okamoto-Schnorr.
Se puede por tanto, utilizar el MSP, para realizar dicha acción, se deberán crear en el módulo ./fabric-protos-go/msp/msp_config.pb.go una función para solicitar la firma ciega (*getCommitment*), y otra función que dado el *commitment* realizado por el usuario, genere la firma ciega (*getBlindSign(e)*). Para este paso, se deben utilizar las funciones definidas en A.3.

**Paso 2**: El cambio para el paso dos, se puede dividir en dos.

1. El primero de los cambios se deberán realizar en el módulo ./fabric-protos-go/common/common.pb.go, en la definición de SignatureHeader, y en el atributo Creator (que deberá tener la identidad serializada otorgada por el MSP del creador del mensaje). Donde se debe modificar el *protobuf* (buffer de protocolo, el cual actúa como XML a la hora de serializar datos, permitiendo leer y escribir durante el flujo de datos) y en su definición colocar alguna función para aleatorizar su valor (o en su defecto dejar prefijado un valor). Este cambio es importante, debido a que sin él, aún firmado ciegamente durante el *proposal transaction*, aún se podrá determinar quién fue el creador de dicha transacción (pues será visible la identidad del creador en la transacción).
2. El segundo cambio, se produce en el módulo ./fabric-protos-go/peer/proposal.pb.go, en donde se deberá cambiar la definición de Signature dentro de la definición de SignedProposal, y cambiarla a un *struct*, que se deberá redefinir. Esta redefinición deberá contemplar los tres campos requeridos para firma ciega (se puede optar por un struct que solo contenga una tira de bytes, que contenga la concatenación de los tres valores de firma).

**Paso 3**: Este paso no requiere modificación adicional en Fabric.

**Paso 4**: Este paso no es necesario, si no se desea cegar las firmas realizadas por los endorsements a la hora de validar transacciones. Pero para realizarlo, se debe repetir lo mismo que se realizó en el segundo cambio del paso 2, pero ahora aplicado para los *endorsement*. El módulo a cambiar será el de ./fabric-protos-go/peer/proposal_response.pb.go, y en la definición de la estructura *Endorsement*, se modifica el atributo Signature, y se cambia a un *struct*, que se deberá redefinir, en donde se pondrán los tres campos requeridos para firma ciega (este *struct* coincide con el definido en el paso 2).

**Paso 5-7**: Estos pasos no requieren modificaciones adiciones en Fabric.

## A.2. Dificultades encontradas

Es válido aclarar, que Fabric, aún sigue en vías de expansión, y tiene muchos campos abiertos en cuanto a su desarrollo. Esto, junto a que, de versión a versión liberada los cambios añadidos son importantes, y que la documentación que se tiene de versiones anteriores queda caduca para versiones más posteriores, hizo dificultosa la tarea de desarrollar el módulo. Incluso en este momento, para 2021, Fabric ya se encuentra en su versión 2.0, y dejará de dar respaldo a la versión 1.4 (en la cual

está basado este proyecto) a partir de abril del 2021.

Como se explica en las secciones anteriores, se implementó de forma idéntica el algoritmo de firma de Okamoto-Schnorr en Python y en Golang. Para luego proceder a integrarlo con Fabric, y se creyó por entonces, que la integración sería una tarea sencilla, pero fue subestimada, debido a que no fue así.

Sin dudas era un gran desafío, el poder lograr esta integración, de todas formas, se probaron diversas implementaciones aunque ninguna de ellas fue de éxito.

El desafío más grande de la implementación, se da en el paso uno, explicado en la sección anterior, al momento de modificar la interacción entre un nodo y la CA de Fabric, debido a que no es sencillo cambiar el paradigma de firma simplemente cambiado de primitivas.

Primero, porque las firmas utilizadas, no solo se utilizan para firmar transacciones, sino que se utilizan también para firmas internas que realiza el sistema (como el manejos de certificados, y registros) lo cual hace que dicho cambio tenga un efecto dominó no querido.

Y segundo, porque un cambio de esta magnitud, en donde, existe más de una interacción con el MSP de forma asincrónica no es el paradigma que posee actualmente Fabric. Y debido a que no es tan simple como establecer funciones o primitivas que simplemente generen las firmas, ya que lograr esa interacción asincrónica no es algo nativo en Fabric (por ejemplo, el usuario quedaría en un proceso bloqueante, esperando el retorno de la variable para realizar el *commitment*, y además, también, cuando envíe el *commitment* nuevamente deberá esperar de forma bloqueante). Para este último punto, no se encontró documentación que facilitara dicha interacción. Hay algunos detalles más, como, por ejemplo, que Fabric verifica que ClientID sea el de un usuario válido, para lo cuál, no se encontró documentación que indique qué módulo realiza tal validez.

Además de la imposibilidad de lograr interacción entre la CA y un nodo de forma asincrónica, surgió un problema al cambiar las primitivas de firmas. Debido a que la primitiva de firmas habitual se utiliza para otras operaciones que se realizan en Fabric Y a esto se le suma también, la escasa documentación sobre los procesos de firma que existen en Fabric. Todas estas dificultades impidieron poder integrar completamente el esquema de firma de BlindCons a Fabric.

## A.3. Implementaciones del algoritmo de firma ciega Okamoto-Schnorr

En esta Sección se describen la estructura de los códigos utilizados para la implementación del algoritmo de firma ciega Okamoto-Schnorr el cual se puede encontrar en el siguiente repositorio de GitLab [115]. El repositorio cuenta con tres directorios, uno llamado ”CodigoPython” que incluye el código de Okamoto-Schnorr desarrollado en Python, así con algunas funciones para realizar pruebas. Luego, hay un segundo directorio llamado ”CodigoGo”, que de igual manera al anterior, incluye el código del algoritmo de firma ciega de Okamoto-Schnorr desarrollado en Go y que contiene algunas funciones de utilidad, en caso de que se quiera probar. Por último

el módulo *./bccsp* que contiene las funciones necesarias para implementar la firma en Fabric, se encuentra en el directorio "CodigoBCCSP".

En las siguientes secciones se explicará con mayor detalle las operaciones de firmar ciegamente un mensaje, de validar una firma ciega y la generación de parámetros necesarios para la ejecución de un algoritmo de firma ciega.

### A.3.1. Generación de parámetros

La generación de parámetros de un sistema es la tarea encargada de generar todos los parámetros públicos que son necesarios para que el sistema funcione. Esta tarea tomará como entrada un número natural conocido como parámetro de seguridad que normalmente se mide en bits. Este parámetro de seguridad definirá el largo máximo de los parámetros públicos a encontrar. La probabilidad del adversario de romper la seguridad se expresa en términos del parámetro de seguridad. Hay que tener en cuenta que existe un costo computacional que pagar a la hora de utilizar parámetros de seguridad de mayor longitud. Por lo que la elección de los parámetros de seguridad va a depender del escenario en que se va a utilizar la primitiva criptográfica. Es habitual pensar en la cantidad de tiempo que se desea garantizar que el adversario no tendrá éxito en romper una primitiva criptográfica. La realidad marca que no es fácil determinar el largo de los parámetros de seguridad para garantizar la seguridad de una primitiva criptográfica.

En nuestras implementaciones, la generación de parámetros toma como entradas dos números naturales los cuales llamamos $N$ y $L$. El valor $N$ es lo que se conoce habitualmente como el parámetro de seguridad, mientras que $L$ determina el orden del subgrupo de $\mathbb{Z}_p^*$. El primer paso del algoritmo es encontrar un número primo de $L$ bits de largo al que llamaremos $q$. A partir de este número $q$ es que buscaremos otro número primo $p$ que cumpla con la siguiente ecuación: $p = (q * e) + 1$ donde e es un número aleatorio de $N - l$ bits. Para finalizar la etapa de generación de parámetros necesitamos encontrar dos elementos $g$ y $h$ de $\mathbb{Z}_p^*$ que sea de orden $q$.

Todos estos parámetros serán públicos y serán usados por todos los usuarios del sistema. En particular, se utilizarán los elementos $g$ y $h$ para construir las claves públicas de todos los usuarios.

### A.3.2. Creación de una firma ciega

Esta función es la encargada de dado un mensaje, generar nueva firma ciega. Para la creación de la firma ciega el usuario tendrá que interactuar con una entidad certificadora. En esta implementación, toda la interacción entre el usuario y la entidad certificadora (CA) se hace de forma secuencial y en el mismo proceso del sistema operativo. Lo primero que buscará hacer esta función es obtener el mensaje a firmar y los parámetros públicos del sistema. Estos valores podrán ser obtenidos por medio de la especificación de un archivo o directamente desde la consola de comandos.

Luego de obtenido el mensaje y los parámetros públicos se comienza con la ejecución del algoritmo de firma ciega. El usuario y la CA deberán de intercambiar una serie de mensajes para poder construir correctamente una firma ciega, los cuales detallamos a continuación:

1. El cliente se comunica con una CA para indicarle que desea crear una nueva firma ciega.
2. La CA responderá al cliente enviándole un valor $a := g^t h^u \pmod p$ donde $t, u \in \mathbb{Z}_q$. Al valor $a$ lo llamaremos *commitment*.
3. El usuario luego de recibir el *commitment* $a$ y calculará los siguientes valores: $\alpha := ag^\beta h^\gamma y^\delta \pmod p$, $\epsilon := \mathcal{H}(m||\alpha)$, $e := \epsilon - \delta \pmod q$ donde $\beta, \gamma, \delta \in \mathbb{Z}_q$. Finaliza este paso enviándole a la CA el valor $e$.
4. La CA recibe el valor $e$ y calculo los siguientes valores: $R := t + er \pmod q$ y $S := u + es \pmod q$. Al finalizar estos cálculos, le envía al usuario los valores $R$ y $S$.
5. El usuario finaliza la creación de la firma ciega calculando los valores: $\rho := R + \beta \pmod q$ y $\sigma := S + \gamma \pmod q$.

La firma ciega final está compuesta por los valores: $(\epsilon, \rho, \sigma)$.

### A.3.3. Validar una firma ciega

Esta función es la encargada de dado un mensaje y una firma ciega de validar si esta firma es correcta. Lo primero que intentará hacer la función es obtener los valores necesarios para validar la firma ciega, ya sea por medio de un archivo donde se encuentran todos estos valores o directamente desde línea de comandos. Será necesario obtener el mensaje firmado al cual llamaremos $m$ y la firma ciega que tendrá la siguiente forma $(\epsilon, \rho, \sigma)$. Luego de obtenidos los valores correctamente pasará a verificar si se cumple la siguiente ecuación $\epsilon = \mathcal{H}(m||\alpha)$ donde $\alpha = g^\rho h^\sigma y^\epsilon \bmod p$

En caso de que los valores que ha obtenido cumplan la igualdad entonces retornará que la firma ciega es correcta, en caso contrario retornará que la firma ciega no es correcta. Por más detalles sobre el procedimiento de validación de una firma ciega dirigirse a la siguiente sección

## A.4. Implementación en Python

La primera implementación del algoritmo de firma ciega Okamoto-Schnorr fue creada haciendo uso del lenguaje de programación Python. Esta decisión de crear esta implementación del algoritmo en un lenguaje que en principio no es compatible con Fabric fue tomada por la experiencia que ya se tenía con este lenguaje de programación. Además, esta primera implementación fue un primer acercamiento al algoritmo de firma ciega el cual permitió un entendimiento más profundo del mismo.

La implementación del algoritmo de firma ciega de Okamoto-Schnorr realizada en python está construida haciendo uso del módulo *argparse* [116]. Este módulo permite agregar una cantidad potencialmente infinita de modificadores a nuestros scripts python. Esta gran cualidad nos permite crear scripts de python muy flexibles. Cuando un script de python que utiliza *argparse* es ejecutado, de forma inmediata se parsean los valores ingresados.

Luego de parseada la entrada del script es que se comienza con la ejecución demanda en los parámetros de entrada. Las posibles operaciones son: generar nuevos

parámetros del sistema, crear una nueva firma ciega para un mensaje dado y verificar una firma ciega para un mensaje dado.

Dentro del repositorio de GitLab donde se han dejado disponibles los códigos utilizados durante este proyecto se ha creado un archivo de ayuda llamado *README.md* donde se explica la utilidad de todos los modificadores de script, se explican las diferentes combinaciones de parámetros que pueden utilizar de forma conjunta y además se deja una lista de ejemplos de uso del script.

## A.5. Implementación en Go

Luego de haber implementado el algoritmo de firma ciega Okamoto-Schnorr en un lenguaje donde se tenia experiencia se decidió realizar una implementación del mismo en el lenguaje de programación que se utiliza de forma nativa en el *framework* Fabric. La implementación de este algoritmo haciendo uso de este lenguaje deja abierta la posibilidad de integrar el algoritmo de firma ciega Okamoto-Schnorr a Fabric.

Esta segunda implementación se caracteriza por ser más rudimentaria que la escrita en Python. Esto se debe a la poca experiencia utilizando el lenguaje Go. Aun así, se ha logrado una versión completa y funcional del algoritmo.

La librería fundamental para la implementación del algoritmo fue la librería nativa de Go llamada *big* [117]. Esta librería permite la creación y operación de números de tamaño arbitrario. Esto fue particularmente útil a la hora de implementar la manipulación de elementos pertenecientes a $\mathbb{Z}_p$.

## A.6. Benchmarks

En esta sección se presentan una serie de pruebas realizadas a las dos implementaciones del algoritmo de firma ciega de Okamoto-Schnorr. Lo que diferencia a las dos implementaciones es el lenguaje de programación con la que se desarrolló cada una. Una de las implementaciones ha sido escrita en el lenguaje de programación *Python* [118], mientras que la otra ha sido escrita en *Go* [95].

### A.6.1. Plataforma computacional

A continuación, se presentan los detalles relevantes de la plataforma computacional utilizada para la realización del conjunto de pruebas. Las pruebas fueron realizadas en el *Centro Nacional de Supercomputación - ClusterUY* [119]. Las características relevantes de la plataforma son:

- Hardware:
  - Procesador: Intel Xeon Gold 6138 2.00GHz.
  - RAM: 128 GB DDR4 2666 MHz.
- Software:
  - Sistema operativo: CentOS versión 7.5.1804.
  - Linux Kernel: 3.10.0-862.11.6.el7.x86_64.

- Lenguajes de programación:
  - Python: 3.8.5.
  - Go: 1.16.3 linux/amd64.

### A.6.2. Descripción de Benchmarks

Para analizar cuál de las dos implementaciones era la mejor se han desarrollado *benchmarks* para las tres operaciones del algoritmo de firma ciega de Okamoto-Schnorr. Las tres operaciones son: generación de parámetros, firma ciega de un mensaje, verificación de una firma ciega. Cada uno de los *benchmarks* ha sido ejecutado cincuenta veces y se reporta la media, la desviación estándar, la mediana, el mejor (menor tiempo de ejecución) y el peor (mayor tiempo de ejecución) del tiempo de ejecución medido en segundos de las cincuenta ejecuciones. También es importante aclarar que aunque en la sección A.6.1 se presente una plataforma computacional con un poder de cómputo importante, para la ejecución de estos *benchmarks* se ha utilizado únicamente un procesador y $4GB$ de memoria RAM.

#### A.6.2.1. Generación de parámetros

Las pruebas realizadas a la función encargada de la generación de parámetros siguen las recomendaciones detalladas por el NIST [1] en el articulo [120]. En dicho artículo se describen cuales son los parámetros recomendados para esquemas de firma digital. En la figura A.1 se pueden ver los resultados obtenidos por la implementación escrita en Python y mientras que en la figura A.2 se pueden ver los resultados obtenidos en la implementación escrita en Python.

| Parámetros | | Resultados | | | | |
|---|---|---|---|---|---|---|
| L (bits) | N (bits) | Media | DE | Mediana | Mejor | Peor |
| 512 | 80 | 0.345 | 0.381 | 0.248 | 0.009 | 2.121 |
| 1024 | 160 | 3.633 | 3.752 | 2.098 | 0.071 | 17.219 |
| 2048 | 224 | 39.031 | 39.170 | 23.528 | 0.544 | 155.316 |
| 2048 | 256 | 36.004 | 39.628 | 23.792 | 1.337 | 183.259 |
| 3072 | 256 | 218.775 | 197.626 | 168.953 | 8.735 | 863.183 |

CUADRO A.1: *Benchmarks* en Python medidos en segundos

| Parámetros | | Resultados | | | | |
|---|---|---|---|---|---|---|
| L (bits) | N (bits) | Media | DE | Mediana | Mejor | Peor |
| 512 | 80 | 0.026 | 0.007 | 0.026 | 0.017 | 0.048 |
| 1024 | 160 | 0.173 | 0.072 | 0.146 | 0.095 | 0.413 |
| 2048 | 224 | 1.586 | 1.162 | 1.357 | 0.567 | 6.736 |
| 2048 | 256 | 1.477 | 0.927 | 1.196 | 0.563 | 4.451 |
| 3072 | 256 | 6.478 | 4.430 | 5.311 | 1.765 | 22.068 |

CUADRO A.2: *Benchmarks* en Go medidos en segundos

1 NIST (National Institute of Standards and Technology): es una agencia de la Administración de Tecnología del Departamento de Comercio de los Estados Unidos. La misión de este instituto es promover la innovación y la competencia industrial en Estados Unidos.

Se puede notar que la implementación escrita en Go tiene tiempos de ejecución menores que la implementación en Python para todas las pruebas. Lo que sí es posible observar que la tasa de crecimiento del tiempo de ejecución en ambas implementaciones son similares, es decir, a medida que se aumenta el largo de los parámetros ninguna de las implementación logra aumentar la brecha de rendimiento. Al ser la generación de parámetros un algoritmo que se ejecuta una única vez, no creemos que sea especialmente necesario utilizar la implementación de Go para la ejecución de la misma.

#### A.6.2.2. Creación de una firma ciega

Las pruebas realizadas a la función encargada de crear una nueva firma ciega se caracterizan por ser firmas sobre mensajes de texto aleatorio de largo variable. En la figura A.3 se pueden ver los resultados obtenidos por la implementación escrita en Python y mientras que en la figura A.4 se pueden ver los resultados obtenidos en la implementación escrita en Python.

| Parámetros | | Resultados | | | | |
|---|---|---|---|---|---|---|
| L (bits) | N (bits) | Media | DE | Mediana | Mejor | Peor |
| 512 | 80 | 5.915 | 0.580 | 5.929 | 2.517 | 6.971 |
| 1024 | 160 | 18.050 | 1.768 | 18.681 | 10.500 | 20.268 |
| 2048 | 224 | 51.532 | 2.978 | 52.655 | 43.021 | 55.166 |
| 2048 | 256 | 55.206 | 2.268 | 56.061 | 48.061 | 58.006 |
| 3072 | 256 | 106.495 | 1.909 | 107.032 | 99.563 | 110.117 |

CUADRO A.3: *Benchmarks* en Python medidos en segundos ($x10^{-3}$)

| Parámetros | | Resultados | | | | |
|---|---|---|---|---|---|---|
| L (bits) | N (bits) | Media | DE | Mediana | Mejor | Peor |
| 512 | 80 | 0.318 | 0.072 | 0.289 | 0.279 | 0.556 |
| 1024 | 160 | 1.195 | 0.072 | 1.177 | 1.160 | 1.555 |
| 2048 | 224 | 4.543 | 0.038 | 4.542 | 4.503 | 4.762 |
| 2048 | 256 | 4.491 | 0.067 | 4.505 | 4.295 | 4.690 |
| 3072 | 256 | 9.506 | 0.036 | 9.500 | 9.449 | 9.645 |

CUADRO A.4: *Benchmarks* en Go medidos en segundos ($x10^{-3}$)

Se puede notar que los resultados obtenidos en los *benchmarks* por la implementación en Go (A.4) son un orden de magnitud menor con respecto a la implementación en Python (A.3). Se debe de considerar que la generación de firmas ciegas es una operación que será ejecutado de forma habitual, por lo tanto, no existe posibilidad ni siquiera de considerar la implementación escrita en Python.

#### A.6.2.3. Verificación de una firma ciega

Las pruebas realizadas a la función encargada de verificar una nueva firma ciega se caracterizan por verificar las firmas obtenidas en las pruebas realizadas a la función de creación de una firma ciega. En la figura A.5 se pueden ver los resultados obtenidos por la implementación escrita en Python y mientras que en la figura A.6 se pueden ver los resultados obtenidos en la implementación escrita en Python.

| Parámetros | | Resultados | | | | |
|---|---|---|---|---|---|---|
| L (bits) | N (bits) | Media | DE | Mediana | Mejor | Peor |
| 512 | 80 | 0.509 | 0.007 | 0.484 | 0.443 | 0.777 |
| 1024 | 160 | 3.354 | 0.145 | 3.342 | 3.145 | 4.162 |
| 2048 | 224 | 15.652 | 0.423 | 15.587 | 15.001 | 16.930 |
| 2048 | 256 | 16.842 | 0.300 | 16.881 | 16.298 | 17.653 |
| 3072 | 256 | 35.492 | 0.630 | 35.443 | 34.320 | 38.326 |

CUADRO A.5: *Benchmarks* en Python medidos en segundos

| Parámetros | | Resultados | | | | |
|---|---|---|---|---|---|---|
| L (bits) | N (bits) | Media | DE | Mediana | Mejor | Peor |
| 512 | 80 | 0.123 | 0.011 | 0.122 | 0.114 | 0.195 |
| 1024 | 160 | 0.498 | 0.014 | 0.496 | 0.478 | 0.561 |
| 2048 | 224 | 1.906 | 0.092 | 1.885 | 1.862 | 2.351 |
| 2048 | 256 | 1.895 | 0.046 | 1.884 | 1.863 | 2.175 |
| 3072 | 256 | 3.970 | 0.046 | 3.956 | 3.914 | 4.160 |

CUADRO A.6: *Benchmarks* en Go medidos en segundos $(x10^{-3})$

Los resultados obtenidos para la operación de verificación de una firma ciega son similares a los de la generación de una firma ciega. La implementación escrita en Go es netamente superior en todos los *benchmarks* ejecutados.

### A.6.3. Trabajo futuro

Como trabajo futuro para las implementaciones realizadas se pueden explorar dos alternativas. La primera implicaría la implementación de las entidades (cliente y entidad certificadora) como dos procesos independientes que ejecutan en la misma máquina física. En este caso, se pueden explorar múltiples herramientas para la comunicación de procesos, por ejemplo, se podría volver a implementar este algoritmo haciendo uso de pasaje de mensajes para las comunicaciones entre cliente y entidad certificadora. Por otro lado, una alternativa aún más flexible que la descripta, implica la implementación de las entidades haciendo uso de sockets de red. Esto permitiría que los procesos puedan ejecutar en diferentes nodos de cómputo, mientras que la comunicación entre ellos se daría por medios de sockets de red. Podríamos decir que la primera de las propuestas tendría un mejor rendimiento al no incurrir en los costos del pasaje de mensajes por medio de la red pero como desventaja tendría que sería una implementación menos flexible que solamente permitiría la ejecución de las entidades dentro de la misma máquina física. Mientras que para la segunda propuesta la ventaja sería su flexibilidad a la hora de iniciar las entidades en diferentes máquinas de la red pero estaría siendo perjudicada por el tiempo en que demoren en llegar los mensajes entre las entidades por medio de la red. Otras posibilidades de trabajo futuro sería la implementación de los algoritmos de firma ciega pero con diferentes funciones de hash criptográficas.

En base a la tarea de *profiling* [2] realizada a las tres operaciones del algoritmo de firma ciega se ha podido identificar que la operación que más tiempo es ejecutada es la multiplicación modular de Montgomery [121]. Esto es habitual en todas las implementaciones de primitivas criptográficas que utilizan elementos pertenecientes a el grupo $\mathbb{Z}_p^*$. Como trabajo futuro se podría implementar el algoritmo de firma ciega haciendo uso de un grupo algebraico más eficiente como son las curvas elípticas o se podría investigar la posibilidad de implementar la multiplicación haciendo uso de múltiples procesadores.

Específicamente hablando de la implementación realizada en Go, se podría cambiar la librería estándar *big* encargada del manejo de los elementos del grupo $\mathbb{Z}_p^*$ por la librería *Gmp* [122]. Esta librería está escrita en el lenguaje C y está altamente optimizada, lo que la llevaría a obtener tiempos de ejecución más cortos.

2 Profiling: es la investigación del comportamiento de un programa informático usando información reunida desde el análisis dinámico del mismo. El objetivo es averiguar el tiempo dedicado a la ejecución de diferentes partes del programa para detectar los puntos problemáticos y las áreas dónde sea posible llevar a cabo una optimización del rendimiento.

# Bibliografía


[1] S. Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System,» *Bitcoin Project*, 2019. dirección: https://bitcoin.org/bitcoin.pdf.

[2] «What is Bitcoin Double Spending?,» 2019. dirección: https://www.bitcoin.com/get-started/what-is-bitcoin-double-spending/.

[3] V. Beal, «Atomic operation.,» 2019. dirección: https://www.webopedia.com/TERM/A/atomic_operation.html.

[4] V. Buterin, «A Next-Generation Smart Contract and Decentralized Application Platform,» 2015.

[5] G. Wood y col., «Ethereum: A secure decentralised generalised transaction ledger,» *Ethereum project yellow paper*, vol. 151, n.º 2014, págs. 1-32, 2014.

[6] S. Noether, «Ring Signature Confidential Transactions for Monero.,» *IACR Cryptology ePrint Archive*, vol. 2015, pág. 1098, 2015.

[7] N. van Saberhagen, «CryptoNote v 2.0,» 2013.

[8] R. L. Rivest, A. Shamir e Y. Tauman, «How to Leak a Secret,» en *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, págs. 552-565, ISBN: 978-3-540-45682-7.

[9] A. Kumar, C. Fischer, S. Tople y P. Saxena, «A Traceability Analysis of Monero's Blockchain,» ago. de 2017, págs. 153-173, ISBN: 978-3-319-66398-2. DOI: 10.1007/978-3-319-66399-9_9.

[10] S. Noether, *Ring Signature Confidential Transactions for Monero*, Cryptology ePrint Archive, Report 2015/1098, https://eprint.iacr.org/2015/1098, 2015.

[11] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer y M. Virza, «Zerocash: Decentralized Anonymous Payments from Bitcoin,» en *2014 IEEE Symposium on Security and Privacy*, 2014, págs. 459-474. DOI: 10.1109/SP.2014.36.

[12] «Sitio oficial de Zerocash,» 2020. dirección: http://zerocash-project.org/.

[13] «Sitio oficial de The Linux Fundation,» 2019. dirección: https://www.linuxfoundation.org/.

[14] A. Tapscott y D. Tapscott, «How blockchain is changing finance,» *Harvard Business Review*, vol. 1, n.º 9, págs. 2-5, 2017.

[15] N. Rifi, E. Rachkidi, N. Agoulmine y N. C. Taher, «Towards using blockchain technology for IoT data access protection,» en *2017 IEEE 17th International Conference on Ubiquitous Wireless Broadband (ICUWB)*, 2017, págs. 1-5.

[16] «Sitio oficial de My Health My Data,» 2020. dirección: http://www.myhealthmydata.eu/.

[17] K. Korpela, J. Hallikas y T. Dahlberg, «Digital supply chain transformation toward blockchain integration,» en *proceedings of the 50th Hawaii international conference on system sciences*, 2017.

[18] S. A. Abeyratne y R. P. Monfared, «Blockchain ready manufacturing supply chain using distributed ledger,» *International Journal of Research in Engineering and Technology*, vol. 5, n.º 9, págs. 1-10, 2016.

[19] A. B. Ayed, «A conceptual secure blockchain-based electronic voting system,» *International Journal of Network Security & Its Applications*, vol. 9, n.º 3, págs. 01-09, 2017.

[20] F. Reid y M. Harrigan, «An analysis of anonymity in the bitcoin system,» en *Security and privacy in social networks*, Springer, 2013, págs. 197-223.

[21] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer y S. Capkun, «Evaluating user privacy in bitcoin,» en *International Conference on Financial Cryptography and Data Security*, Springer, 2013, págs. 34-51.

[22] D. Ron y A. Shamir, «Quantitative analysis of the full bitcoin transaction graph,» en *International Conference on Financial Cryptography and Data Security*, Springer, 2013, págs. 6-24.

[23] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker y S. Savage, «A fistful of bitcoins: characterizing payments among men with no names,» en *Proceedings of the 2013 conference on Internet measurement conference*, 2013, págs. 127-140.

[24] P. Koshy, D. Koshy y P. McDaniel, «An analysis of anonymity in bitcoin using p2p network traffic,» en *International Conference on Financial Cryptography and Data Security*, Springer, 2014, págs. 469-485.

[25] T. Ruffing, P. Moreno-Sanchez y A. Kate, «CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin,» vol. 8713, sep. de 2014. DOI: 10.1007/978-3-319-11212-1_20.

[26] G. Maxwell, «CoinJoin: Bitcoin privacy for the real world,» 2013. dirección: https://bitcointalk.org/index.php?topic=279240.

[27] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll y E. W. Felten, «Mixcoin: Anonymity for bitcoin with accountable mixes,» en *International Conference on Financial Cryptography and Data Security*, Springer, 2014, págs. 486-504.

[28] I. Miers, C. Garman, M. Green y A. D. Rubin, «Zerocoin: Anonymous distributed e-cash from bitcoin,» en *2013 IEEE Symposium on Security and Privacy*, IEEE, 2013, págs. 397-411.

[29] «Problems Online Casino Operators Face and Mistakes They Make,» 2020. dirección: https://slotegrator.pro/analytical_articles/problems-online-casino-operators-face-and-mistakes-they-make/.

[30] «The Evolution of Blockchain Implementation in Casinos and the Gambling Industry,» 2020. dirección: https://hackernoon.com/game-changing-blockchain-casino-technology-whats-the-real-value-26cw32rr.

[31] «The Benefits of Blockchain in the Online Gambling Industry,» 2019. dirección: https://www.thelondoneconomic.com/tech-auto/technology/the-benefits-of-blockchain-in-the-online-gambling-industry/13/05/.

[32] Y. Hu, M. Liyanage, A. Manzoor, K. Thilakarathna y G. Jourjon, «Blockchain-based Smart Contracts - Applications and Challenges,» 2019. dirección: https://arxiv.org/pdf/1810.04699.pdf.

[33] J. Coleman, «State Channels,» 2015. dirección: https://www.jeffcoleman.ca/state-channels/.

[34] J. Longley y O. Hopton, «Funfair technology roadmap and discussion,» 2017. dirección: https://funfair.io/wp-content/uploads/FunFair-Technical-White-Paper.pdf.

[35] A. Chernyaeva, I. Shirobokov y A. Davydov, «Game Channels: State Channels for the Gambling Industry with Built-In PRNG.,» *IACR Cryptology ePrint Archive*, vol. 2019, pág. 362, 2019.

[36] gluk256, «The Signidice Algorithm,» 2017. dirección: https://github.com/gluk256/misc/blob/master/rng4ethereum/signidice.md.

[37] M. Zamani y M. Movahedi, «CryptoRef: Reference on Cryptography,» 2016. dirección: http://mahdiz.com/crypto/basics/.

[38] D. Naccache, «Standard Model,» en *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg y S. Jajodia, eds. Boston, MA: Springer US, 2011, págs. 1253-1253, ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_518. dirección: https://doi.org/10.1007/978-1-4419-5906-5_518.

[39] M. Bellare y P. Rogaway, «Random Oracles are Practical: A Paradigm for Designing Efficient Protocols,» 2021. dirección: http://cseweb.ucsd.edu/~mihir/papers/ro.pdf.

[40] R. Canetti y M. Fischlin, «Universally Composable Commitments,» 2001. dirección: https://eprint.iacr.org/2001/055.pdf.

[41] P. Rogaway e Y. Zhang, «Simplifying Game-Based Definitions,» 2018. dirección: https://eprint.iacr.org/2018/558.pdf.

[42] V. Shoup, «Sequences of Games: A Tool for Taming Complexity in Security Proofs,» 2006. dirección: https://eprint.iacr.org/2004/332.pdf.

[43] M. Bellare y P. Rogaway, *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, 1995.

[44] R. Canetti, O. Goldreich y S. Halevi, «The Random Oracle Methodology, Revisited,» *J. ACM*, vol. 51, n.º 4, 557–594, jul. de 2004, ISSN: 0004-5411. DOI: 10.1145/1008731.1008734. dirección: https://doi.org/10.1145/1008731.1008734.

[45] N. Koblitz y A. Menezes, *Another Look at "Provable Security"*, Cryptology ePrint Archive, Report 2004/152, https://eprint.iacr.org/2004/152, 2004.

[46] ——, *Another Look at "Provable Security". II*, Cryptology ePrint Archive, Report 2006/229, https://eprint.iacr.org/2006/229, 2006.

[47] N. Koblitz y A. J. Menezes, «The Random Oracle Model: A Twenty-Year Retrospective,» *Des. Codes Cryptography*, vol. 77, n.º 2–3, 587–610, dic. de 2015, ISSN: 0925-1022. DOI: 10.1007/s10623-015-0094-2. dirección: https://doi.org/10.1007/s10623-015-0094-2.

[48] «Symmetric vs Asymmetric Encryption – What are differences?,» 2019. dirección: https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences.

[49] D. Pointcheval y J. Stern, «Security Arguments for Digital Signatures and Blind Signatures,» *J. Cryptology 2000*, 2000. dirección: https://www.math.uni-frankfurt.de/~dmst/teaching/WS2013/Vorlesung/Pointcheval,Stern.pdf.

[50] G. Fuchsbauer, A. Plouviez e Y. Seurin, «Blind Schnorr Signatures in the Algebraic Group Model,» 2019. dirección: https://eprint.iacr.org/2019/877.pdf.

[51] ——, «Blind Schnorr Signatures and Signed ElGamal Encryption in the Algebraic Group Model,» 2021. dirección: https://eprint.iacr.org/2019/877.pdf.

[52] D. Chuam, «Blind Signatures for Untraceable Payments,» 1982. dirección: http://www.hit.bme.hu/~buttyan/courses/BMEVIHIM219/2009/Chaum.BlindSigForPayment.1982.PDF.

[53] M. Fischlin, «Round-optimal composable blind signatures in the common reference string model,» 2006. dirección: https://iacr.org/archive/crypto2006/41170058/41170058.pdf.

[54] C. Hanser, «Signatures on Equivalence Classes: A New Tool for Privacy-Enhancing Cryptography,» 2016. dirección: https://diglib.tugraz.at/download.php?id=582ed2865f44f&location=browse.

[55] A. Petzoldt, A. Szepieniec y M. S. Emam, «A Practical Multivariate Blind Signature Scheme,» 2017. dirección: https://eprint.iacr.org/2017/131.pdf.

[56] T. Okamoto, «Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes,» 1992. dirección: https://static.aminer.org/pdf/PDF/000/120/344/provably_secure_and_practical_identification_schemes_and_corresponding_signature_schemes.pdf.

[57] C. Schnorr, «Efficient Signature Generation by Smart Cards,» 1991. dirección: https://d-nb.info/1156214580/34.

[58] S. Goldwasser, S. Micali y C. Rackoff, «The Knowledge Complexity Of Interactive Proof Systems,» 1989. dirección: http://people.csail.mit.edu/silvio/SelectedScientificPapers/ProofSystems/The_Knowledge_Complexity_Of_Interactive_Proof_Systems.pdf.

[59] O. Blazy, X. Bultel, P. Lafourcade y O. P. Kempner, «Generic Plaintext Equality and Inequality Proofs,» 2020. dirección: https://fc21.ifca.ai/papers/79.pdf.

[60] .

[61] I. Damgard, «On Sigma-protocols,» 2010. dirección: https://www.cs.au.dk/~ivan/Sigma.pdf.

[62] T. Pedersen, «Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing,» 1991. dirección: https://sci-hub.tw/https://link.springer.com/chapter/10.1007/3-540-46766-1_9.

[63] M. Fischlin, «Trapdoor Commitment Schemes and Their Applications,» 2001. dirección: https://www.math.uni-frankfurt.de/~dmst/research/phdtheses/mfischlin.dissertation.2001.pdf.

[64] S. Micali, M. Rabiny y S. Vadhan, «Verifiable Random Functions,» 1999. dirección: https://people.csail.mit.edu/silvio/Selected\%20Scientific\%20Papers/Pseudo\%20Randomness/Verifiable_Random_Functions.pdf.

[65] Y. Dodis y A. Yampolskiy, «A Verifiable Random Function with Short Proofs and Keys,» 2004. dirección: https://eprint.iacr.org/2004/310.pdf.

[66] A. Pfitzmann y M. Köhntopp, «Anonymity, Unobservability, Pseudonymity, and Identity Management – A Proposal for Terminology,» 2004. dirección: https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.21.pdf.

[67] R. Zhang, R. Xue y L. Liu, «Security and Privacy on Blockchain,» 2019. dirección: https://dl.acm.org/doi/10.1145/3316481.

[68] S. Haber y W. S. Stornetta, «How to time-stamp a digital document,» en *Conference on the Theory and Application of Cryptography*, Springer, 1990, págs. 437-455.

[69] J. Kurose y K. Ross, «Redes de computadoras: Un enfoque descendente,» *Person 5ta Edición*, 2010.

[70] B. Jürgen y M. Udo, «Central bank money and blockchain: A payments perspective,» *Henry Stewart Publications*, 2016. dirección: https://www.ingentaconnect.com/content/hsp/jpss/2017/00000011/00000002/art00006.

[71] J. Killmeyer, M. White y B. Chew, «Will blockchain transform the public sector?» *Deloitte University Press*, 2017. dirección: https://www2.deloitte.com/content/dam/insights/us/articles/4185_blockchain-public-sector/DUP_will-blockchain-transform-public-sector.pdf.

[72] M. Koscina, O. P. Kempner, N. Laniado, D. Andriopaulou, N. Panourakis, E. Trouva, S. Nifakos y J. Mora, «WP5-Blockchain, Integration & Validation Deliverable D5.1 Blockchain design specifications,» nov. de 2019.

[73] «Hyperledger Fabric oficial web page - Ledger,» 2020. dirección: https://hyperledger-fabric.readthedocs.io/en/release-2.2/ledger/ledger.html.

[74] F. A. Britto y D. Schwartz, «Ripple,» 2021. dirección: https://ripple.com/.

[75] «Libra Association,» 2021. dirección: https://developers.libra.org/docs/assets/papers/the-libra-blockchain.pdf.

[76] «Hyperledger Fabric oficial web page,» 2019. dirección: https://www.hyperledger.org/use/fabric.

[77] «Corda - Open Source Blockchain Platform for Business,» 2019. dirección: https://www.corda.net/.

[78] M. Chakravarty, J. Chapman, K. MacKenzie, O. Melkonian, M. P. Jones y P. Wadler, «The Extended UTXO Model,» 2020. dirección: https://fc20.ifca.ai/wtsc/WTSC2020/WTSC20_paper_25.pdf.

[79] C. Cachin y M. Vukolić, «Blockchain Consensus Protocols in the Wild,» mar. de 2017. dirección: https://drops.dagstuhl.de/opus/volltexte/2017/8016/pdf/LIPIcs-DISC-2017-1.pdf.

[80] C. Hammerschmidt, «Consensus in Blockchain Systems. In Short,» ene. de 2017. dirección: https://medium.com/@chrshmmmr/consensus-in-blockchain-systems-in-short-691fc7d1fefe.

[81] S. Haridi, «Properties: Safety and Liveness,» 2016. dirección: https://courses.edsa-project.eu/pluginfile.php/145/mod_resource/content/1/Lecture_3._Unit_4._Safety_and_Liveness.pdf.

[82] L. Lamport, R. Shostak y M. Pease, «The Byzantine Generals Problem,» *ACM Transactions on Programming Languages and Systems*, págs. 382-401, jul. de 1982. dirección: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/.

[83] M. Castro y B. Liskov, «Practical Byzantine Fault Tolerance,» feb. de 1999. dirección: http://pmg.csail.mit.edu/papers/osdi99.pdf.

[84] C. Dwork y M. Naor, «Pricing via Processing or Combatting Junk Mail,» en *Advances in Cryptology — CRYPTO' 92*, E. F. Brickell, ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, págs. 139-147, ISBN: 978-3-540-48071-6.

[85] R. Merkle, «Method of providing digital signatures,» 1978. dirección: https://patents.google.com/patent/US4309569A/en.

[86] A. Musa, «Data availability proof-friendly state tree transitions,» 2018. dirección: https://ethresear.ch/t/data-availability-proof-friendly-state-tree-transitions/1453.

[87] R. Chan, «Blockchain Data Structure,» feb. de 2018. dirección: https://www.linkedin.com/pulse/blockchain-data-structure-ronald-chan/.

[88] IAB Tech Lab, «Blockchain technology primer,» jul. de 2018. dirección: https://iabtechlab.com/wp-content/uploads/2018/07/Blockchain-Technology-Primer.pdf.

[89] Electrum documentation, «Simple Payment Verification,» 2017. dirección: https://electrum.readthedocs.io/en/latest/spv.html.

[90] A. Narayanan, J. Bonneau, E. Felten, A. Miller y S. Goldfeder, «Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction,» *Princeton University Press Princeton*, 2016. dirección: https://dl.acm.org/citation.cfm?id=2994437.

[91] N. Szabo, «Formalizing and Securing Relationships on Public Networks,» 1997. dirección: https://firstmonday.org/article/view/548/469.

[92] ——, «Secure Property Titles with Owner Authority,» 1998. dirección: https://nakamotoinstitute.org/secure-property-titles/.

[93] I. Bashir, *Mastering Blockchain: Distributed Ledger Technology, Decentralization, and Smart Contracts Explained*, ép. Expert insight. Packt Publishing Limited, 2018, ISBN: 9781788839044. dirección: https://books.google.com.uy/books?id=bsxnswEACAAJ.

[94] «Sitio oficial del lenguaje de programación Java,» 2019. dirección: https://www.java.com/es/.

[95] «Sitio oficial del lenguaje de programación Go,» 2021. dirección: https://golang.org/.

[96] «Sitio oficial de Node.js,» 2019. dirección: https://nodejs.org/.

[97] «A Blockchain Platform for the Enterprise,» *Hyperledger Fabric Documentation*, 2019. dirección: https://hyperledger-fabric.readthedocs.io.

[98] S. Giammusso, «Blockchain for Education Case Study on Hyperledger Fabric,» *Politecnico di Torino*, 2019.

[99] S. Vedpathak, «Hyperledger Fabric — Components and Architecture,» *Clairvoyantsoft*, 2019. dirección: https://blog.clairvoyantsoft.com/hyperledger-fabric-components-and-architecture-b874b36c4af5.

[100] «Sitio oficial de Docker,» 2019. dirección: https://www.docker.com/.

[101] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic y J. Yellick, «Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,» ene. de 2018.

[102] IBM Research Editorial Staff, «Behind the Architecture of Hyperledger Fabric,» feb. de 2018. dirección: https://www.ibm.com/blogs/research/2018/02/architecture-hyperledger-fabric/.

[103] M. Koscina, P. Lafourcade, D. Manset y D. Naccache, «BlindCons: A Consensus Algorithm for Privacy Preserving Private Blockchains,» *LIMOS*, 2018.

[104] «MSP Implementation with Identity Mixer,» *Hyperledger Fabric Documentation*, 2020. dirección: https://hyperledger-fabric.readthedocs.io/en/release-2.2/idemix.html.

[105] A. D. Caro, D. Bogatov, K. Elkhiyaoui y B. Tackmann, «Anonymous Transactions with Revocation and Auditing in Hyperledger Fabric,» 2019. dirección: https://eprint.iacr.org/2019/1097.pdf.

[106] «Using FabToken,» *Usando FabToken*, 2020. dirección: https://fabric-documentations.readthedocs.io/en/latest/token/FabToken.html.

[107] M. Chaieb, M. Koscina, S. Yousfi, P. Lafourcade y R. Robbana, «DABSTERS: a Privacy Preserving e-Voting Protocol for Permissioned Blockchain,» 2020. dirección: https://sancy.iut-clermont.uca.fr/~lafourcade/PAPERS/PDF/CKYLRL19.pdf.

[108] E. Androulaki, J. Camenisch, A. D. Caro, M. Dubovitskaya, K. Elkhiyaoui y B. Tackmann, «Privacy-preserving auditable token payments in a permissioned blockchain system,» 2020.

[109] T. E. Gamal, «A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms,» 1984. dirección: https://link.springer.com/chapter/10.1007\%2F3-540-39568-7_2.

[110] J. Groth, «Efficient Fully Structure-Preserving Signatures for Large Messages,» 2015. dirección: https://eprint.iacr.org/2015/824.pdf.

[111] D. Pointcheval y O. Sanders, «Short Randomizable Signatures,» 2016. dirección: https://eprint.iacr.org/2015/525.pdf.

[112] J. Groth y A. Sahai, «Efficient Non-interactive Proof Systems for Bilinear Groups,» 2016. dirección: https://eprint.iacr.org/2007/155.pdf.

[113] A. Sonnino, M. Al-Bassam, S. Bano, S. Meiklejohn y G. Danezis, «Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers,» 2019. dirección: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_06A-1_Sonnino_paper.pdf.

[114] «Constant-Size Dynamic k-TAA,» 2006. dirección: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.215.2240&rep=rep1&type=pdf.

[115] M. Leal y M. Montiglio, «Provably Secure Blind Signature Schemes - Implementación,» 2020. dirección: https://gitlab.fing.edu.uy/proyecto-de-grado/provably-secure-blind-signature-schemes.

[116] «argparse: Parser for command-line options, arguments and sub-commands,» 2021. dirección: https://docs.python.org/3/library/argparse.html.

[117] «Go oficial web page - Package big,» 2021. dirección: https://golang.org/pkg/math/big/.

[118] «Sitio oficial del lenguaje de programación Python,» 2021. dirección: https://www.python.org/.

[119] S. Nesmachnow y S. Iturriaga, «Cluster-UY: Collaborative Scientific High Performance Computing in Uruguay,» en *Supercomputing*, M. Torres y J. Klapp, eds., Cham: Springer International Publishing, 2019, págs. 188-202, ISBN: 978-3-030-38043-4.

[120] C. F. Kerry, A. Secretary y C. R. Director, *FIPS PUB 186-4 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Digital Signature Standard (DSS)*, 2013.

[121] P. L. Montgomery, «Modular multiplication without trial division,» *Mathematics of computation*, vol. 44, n.º 170, págs. 519-521, 1985.

[122] «The GNU Multiple Precision Arithmetic Library,» 2021. dirección: https://golang.org/pkg/math/big/.