Arquitectura de Software en Wallet de Código Abierto para Privacy Coin en Dispositivos Móviles

El caso de estudio de Zcash

Lic. Francisco Gindre

Director:
Dr. Matías Urbieta

Tesis presentada para obtener el grado de
Magister en Ingeniería de Software

Facultad de Informática
Universidad Nacional de La Plata
15 de Septiembre de 2021

Abstract

Desde su surgimiento en 2009, el uso de criptomonedas ha estado en constante crecimiento en términos de cuota de mercado y adopción. Este "boom" está a la vista de todos, y al igual que la gran mayoría de las transacciones del mundo de las finanzas descentralizadas, es de completo estado público. Pese al uso de criptografía de avanzada, la privacidad en el "mundo cripto" es relativamente baja, con excepciones: Las Privacy Coins o Monedas con Mejoramiento del Anonimato (Anonymity Enhancement Coins o AEC). Este trabajo toma la idea de la privacidad como Derecho Humano y se centra en dilucidar los requerimientos para el desarrollo de billeteras electrónicas móviles para AECs, analizando las criptomonedas Monero y Zcash, tomando como caso de estudio esta última. Tiene como aporte una lista de requerimientos funcionales y no funcionales y proponiendo una arquitectura de referencia para cumplir con los mismos de forma abstracta, e identifica áreas a profundizar en materia de revisión sistemática de la literatura, privacidad y seguridad.

Keywords— Zcash, privacy coin, móviles, cliente liviano, arquitectura, criptomonedas

Since their appearance in 2009, the use of cryptocurrencies has been growing constantly in terms of market cap and adoption. This boom is publicly visible as well as the grand majority of the decentralized finance transactions. Despite the use of advanced cryptography, privacy in the "crypto world" is relitvely low, with certain exceptions: Privacy Coins (or Anonymity Enhanced Coins AEC). This work takes the idea of Privacy as a Human Right and focuses on elicitating the requirements for developing mobile wallets for AECs, analyzing the cryptocurrencies Monero and primarily Zcash, taking the latter as study case. Its contributions are: a list of functional and non-functional requirements to develop a privacy coin light client, a reference Architecture that addresses these requirements in an abstract manner and finally a list of future work related to the fields of Systematic Literature Review, Privacy and Security.

Keywords— Zcash, privacy coin, mobile, light client, architecture, cryptocurrency

Agradecimientos

Este trabajo comenzó a escribirse cuando Bitcoin tenia apenas unos diez mil bloques a mediados de 2009, cuando la Dr. Daniela López De Luise me aceptó como colaborador en su laboratorio de Inteligencia Artificial sin más requisitos que la pasión por la investigación. Ningún párrafo de este trabajo podría haberse materializado si no fuese por sus enseñanzas primero como docente, luego como mentora y colega a lo largo de numerosos proyectos de investigación, papers, congresos y seminarios. Si acaso los lectores de estos textos esbozaran algún elogio sobre ellos, probablemente se deban al tiempo que Daniela dedicó a mi formación y a quien participo en este aporte.

En esta misma línea, al Dr. Gustavo Rossi, por sus consejos abiertos, honestos y guía precisa que me encomendase al Dr. Matías Urbieta, quien también es un participe esencial en la realización de este trabajo, por las tantas videollamadas y sus puntillosas devoluciones en una labor a cuya complejidad se le suma el contexto de pandemia.

En este trabajo sobre arquitectura de software, se condensan innumerables cantidades de conocimiento colectivo y avances logrados por muchos profesionales de la ingeniería de software, Criptografía, Privacidad y sobre todo, la cultura Cypherpunk y el Código Abierto. Siempre en estas secciones se trae a colación el hecho de que ser investigador significa en gran medida "estar parado sobre los hombros de gigantes", en este caso tengo la dicha de poder conocer y trabajar personalmente con algunos de ellos como Zooko y Nathan Wilcox, Jack 'Str4d' Grigg, Daira Hopwood, Sean Bowe, Kevin Gorham, Taylor Hornby, Kris Nuttycomb, Larry Ruane, Joseph Van Geffen, David Campbell, Ying Tong Lai, Bradley Miller, Linda Liu y todo el resto del equipo de Electric Coin Company.

A Aditya Kulkarni de ZecWallet. A los científicos y desarrolladores de la Zcash Foundation y la comunidad de Zcash.

A mis colegas y amigos Joaquín González y Federico Elgarte, que insistentemente repetían su mantra "metete en cripto" mientras pasaban nuestros días como desarrolladores iOS de productos de consumo masivo.

Por último a mi pareja, a mi familia y a mis amigas y amigos. Aunque probablemente no entienden mucho de este asunto, me apoyan incondicionalmente en todos mis proyectos. Cada minuto invertido aquí es tiempo que no he podido compartir con ellas y ellos.

Índice de figuras

Índice de cuadros

Índice de código fuente

Índice general

Capítulo 1

Introducción

1.1. Motivación

La idea de una plataforma de "efectivo electrónico" que provea anonimato en las transacciones y a su vez permita auditorías fue publicada en 1999 en el paper de Tomas Sander and Amnon Ta–Shma: "Auditable, Anonymous Electronic Cash" [1]. Allí se teoriza un sistema de efectivo electrónico que aún proveyendo anonimato, pueda garantizar un nivel de verificabilidad e integridad a todos los actores involucrados. Los autores establecen la importancia de que un sistema de estas características, tiene que tener mecanismos para poder detectar y evitar usos maliciosos o ataques a la moneda[1]. Establecen una serie de requisitos indispensables para un sistema de efectivo electrónico:

- *Imposibilidad de falsificación*[2]: Debe ser imposible la falsificación de pagos, aún sea el caso que varios actores actúen en connivencia para alterar la base monetaria y acuñar moneda de forma ilegítima.
- *Auditabilidad*: debe existir un registro público que permita verificar las operaciones y los actores involucrados en ellas.
- *No rigidez*: El sistema puede aceptar pagos en distintos tipos de activos acorde a un protocolo de depósitos, pero también, debe poder invalidad dichos depósitos o activos.
- *Anonimato incondicional:* Plantean que si bien los usuarios deben identificarse ante autoridad monetaria, la actividad de estos debe poder ser identificable. Las operaciones deben ser indistinguibles unas de otras de forma tal que estas generen un anonimato estadístico mediante el cual se reduzcan las posibilidades de que terceros puedan descubrir la identidad de quienes operan en la red.

Proponen que estos principios podrían ser alcanzados (en teoría) gracias a la utilización de pruebas de cero-conocimiento (Zero Knowledge proofs), hashing y árboles de Merkle. Estar herramientas, son hoy las bases de las criptomonedas con privacidad en asiento contable (privacy coin) actuales.

[1] Los Autores definen estos ataques a actividades tales como el robo de fondos o lavado de activos.

[2] Traducción: Unforgeability

Desde el paper publicado bajo el pseudónimo Satoshi Nakamoto [2] que se considera como el origen de Bitcoin, se registró la creación de un gran número de criptomonedas y el despliegue de sus respectivas redes descentralizadas de cómputo y minado de transacciones. Pasaron varios años desde ese entonces para que se levantara la cuestión de la privacidad en el debate público de la escena de las criptofinanzas. La publicación del paper Zerocoin [3], que brindaba la posibilidad de hacer anónimo el origen de una transacción en Bitcoin y su extensión Zerocash [4] (Zcash), que propone un protocolo de criptomonedas basado en la privacidad de las transacciones entre pares para todas sus partes, fue un punto de inflexión en esta industria. A diferencia de Bitcoin o Ethereum [5], este tipo de criptomoneda ofrece un grado de privacidad en las transacciones. La cadena de bloques propuesta en Zerocash, sigue siendo pública y se encuentra almacenada en su totalidad en los nodos que componen la red al igual que en Bitcoin. También soporta transacciones transparentes similares y compatibles con el protocolo Bitcoin. Sin embargo, ofrece la posibilidad de transacciones blindadas (shielded) que resguardan los detalles de las mismas. Ello no impide que nuevos nodos puedan validar la cadena de bloques, ya que se asegura la integridad de toda la cadena mediante la utilización de pruebas de "cero-conocimiento" que permiten demostrar la autenticidad e integridad de información a terceros sin que sea necesario mostrar parte alguna de la información en cuestión.

Las ventajas de las criptomonedas que proveen privacidad y anonimato en las transacciones son claras. El paper "CryptoNote v2.0" de Monero Labs [6] enumera las condiciones que definen una "privacy coin":

- No trazabilidad: para cada transacción entrante, todos los remitentes son igualmente probables
- No asociatividad: para dos transacciones salientes, es imposible determinar que fueron enviadas al mismo destinatario

No obstante, la promesa de privacidad y anonimato no se tradujo instantáneamente en mayores volúmenes de adopción de usuarios o márgenes de mercado. La presencia de privacy coins en los sitios de cambio (Exchanges) de criptomonedas son en cierto grado menores que aquellas monedas que no ofrecen privacidad alguna. Este factor se incrementa cuando se concentra la mirada en plataformas móviles como iOS o Android.

En una blockchain transparente como la implementada en Bitcoin, una transacción entrante puede identificarse mediante la derivación de las propias direcciones y la comparación lineal de dichos valores [7] en los campos presentes en las transacciones. Mientras que en criptomonedas como Zcash o Monero, dicha información no esta completa o siquiera presente, y se debe intentar descifrar cada transacción, siendo el éxito del descifrado lo que determina si una transacción tiene a ese actor como destinatario o no. Esto conlleva a que las aplicaciones cliente de este tipo de blockchain requieren una mayor cantidad de poder de cómputo y almacenamiento solo para el acto pasivo de recibir una transacción detectada por un nodo que actúa a su vez como "Servidor" de clientes móviles (wallets). Además, para el caso de generar una transacción hacia uno o más destinatarios, se requieren más recursos que los necesarios para crear su contraparte no privada. Las criptomonedas con mejoras en el anonimato[3] (privacy coins) requieren de mecanismos tales como las "pruebas de cero conocimiento" (Zero Knowledge proofs o ZK-proofs) para reservar la información que es de público conocimiento en las cadenas de bloques públicas (como Bitcoin o Ethereum). Las ZK-Proofs son un avance primordial en el ámbito de la privacidad. Proveen mecanismos

[3]Traducción del inglés Anonymity Enhancing Cryptocurrencies (AEC)

para demostrar inequívoca y unilateralmente la existencia y la potestad de piezas de información sin necesidad alguna de revelar su contenido. Al verificar información con una prueba de cero-conocimiento, el verificador, no toma conocimiento alguno sobre los datos que está verificando.

El tamaño de la blockchain es una limitante para su inclusión en pequeños dispositivos. Para emitir o recibir una transacción es necesario estar sincronizado con la cadena de bloques y reaccionar a sus cambios. Para el caso de las cadenas públicas, los dispositivos con pocos recursos (comparativamente hablando con un servidor o computadora personal) utilizan clientes livianos (light clients) que se sirven de un nodo (full node) que tiene completo conocimiento de la blockchain y que oficia de mediador entre el procesamiento y almacenamiento pesado que requiere la blockchain y el dispositivo mediante una API cliente (ya sea REST, RPC, web-socket, etcétera). Allí, ese intermediario no representa problema alguno, ya que la blockchain ya es en sí de estado público y toda la información vertida esta disponible para todos los involucrados.

¿Qué pasa cuando nos interesa resguardar la privacidad de los actores de una blockchain? La utilización de clientes livianos y nodos intermediarios descrita anteriormente se vuelve un problema que compromete la privacidad de los actores involucrados.

Este trabajo propone un caso de estudio sobre Zcash, una criptomoneda basada en protocolo Zerocash. Puntualmente en una arquitectura que permite realizar y recibir transacciones desde dispositivos móviles Android y iOS sin la necesidad de disponer de una copia total de la blockchain y sin comprometer la privacidad de los actores mediante actores intermediarios.

1.2. Objetivo

Pese a que aún no se registra una adopción masiva de las criptomonedas, la base de usuarios crece diariamente. El advenimiento de este nuevo paradigma económico, requerirá la creación de muchas aplicaciones que lleven las finanzas descentralizadas a las manos de los millones de usuarios que hoy operan en las finanzas tradicionales.

Si bien existen numerosas aplicaciones para operar con ellas desde dispositivos móviles, son pocas las que soportan y priorizan la privacidad de los usuarios mediante las utilización de criptomonedas con privacidad (privacy coin o AEC) como Zcash o Monero[4].

El objetivo de este trabajo es indagar sobre esta situación y como aporte, delinear una lista de requerimientos y una arquitectura de referencia para billeteras electrónicas que les permita operar en dispositivos móviles.

Para ello, en esta tesis se desarrollará una revisión sistemática de la literatura (SLR, por sus siglas en inglés) con el objetivo de relevar el estado del arte en materia de billeteras para dispositivos móviles y *privacy coins* y de esta forma identificar oportunidades de contribución.

Este estudio apunta a relevar literatura alrededor de las siguientes interrogantes:

[4]En este trabajo se utilizan "Privacy Coin" y AEC (Anonymity Enhancing Cryptocurrency) de forma equivalente. Este último es el término legal que distintos organismos regulatorios de los Estados Unidos han tomado para definir a las privacy coins. En Contraposición, distintos actores del mundo cypherpunk han definido al resto de las criptomonedas como Surveillance-Enabling Cryptocurrencies haciendo referencia a que el estado público del asiento contable y grafo de transacciones de las mismas permiten y habilitan la vigilancia de los Estados y las corporaciones sobre las personas. Irónicamente la sigla resultante, SEC, es homónima a la del organismo que regula el intercambio de activos financieros en Estados Unidos, la *Security Exchange Commission* (SEC)

- ¿Qué diferencias de requerimientos funcionales y no funcionales existen entre wallets que soportan Privacy Coins en Desktop, Mobile y Web?
- ¿Qué experiencias hay del cómputo de transacciones del lado del cliente?
- ¿Qué esquemas de respaldo de cadena de bloques se pueden utilizar?

Se proponen estos criterios preliminares de selección de material a revisar:

- Se refiere una o mas monedas listadas en el compendio de privacy coins [8]
- El proyecto se encuentra desplegado en producción (es decir, en el mercado)
- Utiliza blockchain
- la privacidad no es obtenida mediante técnicas de «Anonimización» y/o manipulación de transacciones públicas como por ejemplo *CoinJoining* o *Snowballing*.

Se relevarán historias de usuario y requerimientos de billeteras móviles de código abierto para las criptomonedas Zcash y Monero. En base a los hallazgos de la elicitación de requerimientos, se llevará a cabo una comparación de las historias de usuario comunes y particulares del caso. Esta comparativa se utilizará como material de la arquitectura de referencia que permita condensar integrar estas cadenas de bloques en billeteras digitales (wallets) en dispositivos móviles (iOS o Android) de forma estructurada, considerando las historias de usuario y requerimientos relevados.

Además de la descripción de arquitectura mencionada, uno de los aportes importantes de este trabajo es el resultado de la elicitación se condensará en una lista de requerimientos e historias de usuarios que debe tener una billetera electrónica para criptomonedas con anonimato mejorado (AEC) o *privacy coins*

La eventual implementación de la propuesta permitirá construir una aplicación donde:

- se utilice la custodia soberana de las claves privadas del usuario de forma segura;
- los usuarios puedan operar criptomonedas que provean mecanismo de salvaguarda de la privacidad en las operaciones;
- las transacciones se computen y almacenen en el propio dispositivo;
- no se requiera correr localmente un nodo con la totalidad de la cadena de bloques para tales fines.

Se analizarán distintos proyectos de código abierto para las criptomonedas Zcash y Monero a fin de elicitar requerimientos, mientras que la definición de la propuesta de arquitectura se centrará en Zcash.

Queda fuera del alcance de este trabajo la discusión sobre los patrones de diseño y/o arquitecturas inherentes a la capa de presentación de los dispositivos móviles (Model View Controller, Model View Presenter, Model View ViewModel, View Interactor Presenter Entity Router, etcétera) [9].

1.3. Limitaciones/exclusiones

Durante la realización del proyecto de este trabajo de tesis se encontraron diversos temas y problemáticas en estrecha relación con el tema principal a abordar. A fin de acotar el alcance del trabajo se han los siguientes aspectos dejado fuera del mismo:

- Seguridad informática relacionados con ataques a servidores de clientes livianos (light clients).

- Criptografía de Curvas Elípticas.
- privacidad en la comunicación de datos entre aplicaciones cliente y servidores.
- Capa de presentación, experiencia de usuario y diseño de interfaces gráficas.

1.4. Estructura del trabajo

La forma de este trabajo surge del proceso de redacción. A medida que se redactaban los distintos capítulos y secciones, la lectura se encontraba interrumpida por la introducción de conceptos, términos y jerga propios del dominio de las criptomonedas. A raíz de ello, surgió la necesidad de extraer todo este conocimiento entramado en las cuestiones técnicas del trabajo en su propio capítulo. De allí, el motivo de la redacción del trabajo intenta cumplir con la premisa de poder alcanzar a quienes se encontrasen ante la tarea de desarrollar una billetera y no tuvieran conocimiento alguno sobre criptomonedas ni *privacy coins*.

El presente capítulo 1 presenta los objetivos, su motivación y el alcance del trabajo. El capítulo 2 se encuentra dedicado enteramente a introducir al lector en el ámbito de las criptomonedas desde su funcionamiento y propósito, comenzando por Bitcoin, donde se explican conceptos de identidad dentro de la red, pseudo-anonimato, recepción de fondos, custodia y envío de los mismos. Luego se aborda el tema de la privacidad como concepto, como requerimiento y como derecho y se explican las similitudes y diferencias entre los conceptos vertidos para Bitcoin en las criptomonedas Zcash y Monero, así como los tipos de billeteras disponibles.

El capítulo 3 describe el proceso y las conclusiones de la revisión sistemática de la literatura existente sobre el tema en cuestión.

El capítulo 4 versa sobre el relevamiento de requerimientos realizado sobre distintas billeteras que utilizan Zcash o Monero. De aquí surge una lista de requerimientos funcionales y no funcionales, que se describen en forma de Temas, Épicas, e historias de usuario de acuerdo a la metodología Scrum [10]. Finalmente se analizan la estructura y funcionalidades de los kits de desarrollo de Zcash y Monero.

El capítulo 5 describe la arquitectura de referencia propuesta a partir de los requerimientos relevados, primero desde una estructura general y luego describiendo los componentes principales siguiendo una estructura uniforme para cada uno de ellos. Adicionalmente, se presentan casos de estudio donde algunos de estos componentes se implementan en proyectos que se encuentran desplegados y productivos.

El capítulo 6 contiene las conclusiones del trabajo y las áreas propuestas como posible trabajo a futuro.

Capítulo 2

Introducción a las Criptomonedas

Para empezar a comprender como funciona una billetera de una criptomoneda, es conveniente comenzar por lo conocido, yendo al caso de una Aplicación de Billetera Virtual que opera sobre el sistema monetario de una nación, arbitrado por un banco central. Cuando el usuario ingresa por primera vez, se presenta al usuario con un "Login" donde este debe identificarse como un usuario registrado o bien, registrarse. Para registrarse, debe realizar una serie de pasos para identificarse unívocamente que suelen ser provistos y dictados por la autoridad monetaria. Allí un usuario debe ingresar su nombre completo, identificaciones nacionales y tributarias y hasta su fotografía sosteniendo dicha documentación. Estos pasos son conocidos en la jerga financiera como *Know Your Customer* (KYC), o "conozca a su cliente". Realizados estos pasos y pasadas las verificaciones, se asigna un "nombre de usuario" que se vincula a la persona física. A partir de allí todas las operaciones que realice el "usuario" del sistema, serán asociadas a esa persona física. Todo esto es posible porque hay una autoridad central que delega estos controles a los distintos actores del sistema, pero que retiene autoridad y custodia total sobre los datos, fondos y operaciones realizadas y a realizarse hasta que el usuario en cuestión salga de las fronteras del sistema financiero.

Las criptomonedas son sistemas descentralizados entre pares (peer-to-peer). Donde que quienes operan pueden hacerlo a título propio, sin necesidad alguna de pasar por un proceso de KYC, dado que no hay una autoridad monetaria central, sino que esta es reemplazada por un algoritmo de consenso y un protocolo de intercambio, donde todos los integrantes de la red son idempotentes en cuanto a funcionalidad y autoridad[1]. Cada uno de los nodos que componen la red de pares a su vez contiene la información de todas las operaciones realizadas. La confiabilidad de una criptomoneda no se basa en el poder de una autoridad central, sino en el de una red prolífica de nodos idempotentes (pares) que proveen sustentabilidad, redundancia y robustez a todo el ecosistema.

Esta topología es la que da origen a la frase "Sea su propio banco"[2].

[1] esto aplica para el caso de los participantes "Full Node". Es posible (y esperable) que, participantes del tipo cliente liviano no sean idempotentes respecto de los pares que corren un nodo

[2] En esta frase, se da por implícito que la criptomoneda como herramienta de consenso y

2.1. Blockchain, a 10.000 metros de altura

Detengámonos un momento en la última frase: "Sea su propio banco". ¿Qué implicaría para Alicia, una inquieta contadora, ser su propia banca? Alicia tiene que poder (1) identificarse unívocamente como una actora dentro del sistema financiero. Una vez identificada, (2) debe poder recibir valores (tokens o unidades de la moneda) a su nombre. Recibidos esos valores, Alicia debe poder (3) resguardar sus valores indefinidamente hasta que ella decida cuando operar con ellos. Y por último, (4) Alicia tiene que ser capaz de gastar esos valores sin que estos puedan ser puestos en duda de su autenticidad. Esto requiere que esos valores sean fungibles (que se consumen en su uso y no puedan gastarse dos veces) e intercambiables sin distinción unos por otros a igual valor. Es necesario que nadie más que Alicia pueda decidir gastar esos valores. Existe además una alternativa al punto 2. Alicia también puede ser la "Casa de La Moneda", cualquiera puede sumarse a la red como una "Minera" y trabajar para validar pruebas de trabajo, generar nuevos bloques y emitir nuevas fracciones de la criptomoneda.[3]

Las criptomonedas son medios de intercambio y de reserva de valor en base a la confianza todos los actores de la red agregan al sistema. Cada persona que corra un nodo dentro de una criptomoneda, agrega valor de diferentes formas. Si solo está computando las transacciones, está invirtiendo recursos en verificar bloques y transacciones a lo largo de la cadena, dispersando las transacciones verificada y descartando las que no cumplen con las reglas de consenso. Por otra parte quienes están minando los bloques, agregan valor generando esas nuevas unidades de criptomoneda, creando los bloques y ejecutando las transacciones que previamente han validado otros nodos pares o ellos mismos. Ya sea computando transacciones o minando, todos contribuyen a la confiabilidad de la red y al valor inherente de la misma.

2.1.1. (1) - Yo, soy quien que soy: Identidad, pseudo anonimato, anonimato y privacidad

En el sistema financiero centralizado, alguien con la potestad adecuada dirá: Ella es Alicia, identificación número "XYZ123". Y en base a la confianza delegada al sistema central, todos los que operan allí, tomarán esa designación como algo indisputable. En un sistema descentralizado, como Bitcoin o Zcash, no son el centro las personas físicas o jurídicas, sino los hechos que se impactan sobre la cadena de bloques. Solo basta que Alicia genere sus claves públicas y privadas desde una entropía para comenzar a operar. No existe una entidad central que identifique unívocamente actores y direcciones. En términos meramente operativos, la identidad, no es un problema central dentro del mundo criptomonedas, sino lo que validan los distintos protocolos de consenso que conforman una blockchain, son los bloques y las transacciones que estas conforman. Operar en el *pseudonanonimato* es la normalidad. No son los protocolos quienes establecen y crean métodos para unir los puntos de los grafos que conforman las operaciones dentro de las distintas cadenas de bloques, sino los marcos regulatorios que miran a las criptomonedas desde la óptica ávida de control propia de un sistema monetario centralizado. Este criterio no es uniforme, así como muchos Estados las ignoran completamente, otros prohíben su uso y utilizan el análisis sistemático de las cadenas de bloques («chainalysis») para poder identificar a los ciudadanos que allí operan y "ajustarlos a derecho".

---

confianza seria el Banco (de)central, y el usuario, un banco de individuos y empresas.

[3] Ver "Mining Bitcoin" https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch02.asciidoc

El «chainalysis» permite condensar toda la información pública de una blockchain y poder realizar análisis de datos sobre ellos para obtener información sobre sus usos y sobre todo sobre los movimientos que realizan los usuarios. Por ejemplo, en una cadena de bloques de transacciones públicas como Bitcoin o Ethereum, es posible utilizar estos sistemas para determinar todo el grafo de operaciones de una dirección dada, tanto en balance como en movimientos. Lo que equivaldría a que un desconocido pudiera acceder libremente al historial bancario de cualquier persona solo con el numero de cuenta y permitiéndole determinar cuánto dinero tiene en su haber y cómo operó históricamente. Toda esta información pública es utilizada en conjunto con otros metadatos, como direcciones IP, cuentas conocidas compartidas por personas de interés para romper el velo del pseudoanonimato, y ponerle una identidad a esas direcciones y remover todo vestigio de privacidad del usuario bajo observación y posiblemente de las personas con el que este hubiera operado. En definitiva el «chainalysis» es una método de análisis de datos que permitiría saber quién es quién en un mar de direcciones pseudoanónimas gracias a la información que publican los propios usuarios al operar en ciertas cadenas de bloques.

Este hecho fue sintetizado por el ingenio popular Cypherpunk que definió a Bitcoin como “Twitter para tu cuenta bancaria”.

Son las Privacy Coins las que ponen el foco en la identidad de los usuarios. No para identificarlo, sino por el contrario, para anonimizarlo. Ponen de manifiesto la privacidad como un derecho humano. Los protocolos de consenso deben incorporar la tecnología criptográfica necesaria para garantizar el anonimato para garantizar un “Sistema de Efectivo entre pares Electrónico” tal cual titulaba el paper de Satoshi Nakamoto[4].

2.1.2. (2) - Recibir fondos

Alicia no tiene que hacer nada en especial para poder recibir fondos más que informar una dirección suya para que sea incluida en una transacción. En el momento que ella deriva sus claves públicas y privadas, ya puede crear direcciones de wallets que sólo ella podrá reclamar como propias. Roberto, es un fanático de las criptomonedas, que tiene algunos servidores para hacer minería de distintas criptomonedas. Conoció a Alicia en un grupo de Telegram[5] de intercambio de criptomonedas buscando un profesional de la contaduría que aceptara “cripto” como pago de honorarios y desde entonces es cliente de sus servicios contables. Cuando llega el primero de mes, Roberto crea una transacción con la cantidad de *satoshis* de Bitcoin para el pago de los honorarios de Alicia[6]. Esta transacción se conforma con fondos (inputs) provenientes de algún bloque ha minado con sus equipos, y dos salidas (outputs), una conteniendo el monto de los honorarios mensuales de Alicia y otra el monto del cambio a retornar a la dirección de Roberto. La diferencia entre los inputs y los outputs será tomada como la “comisión” a pagar al minero que incluya esta transacción en un bloque de la cadena. Los inputs contienen la firma de Roberto indicando que él ha utilizado esos fondos en esta transacción, que se convertirán en outputs rumbo a la billetera de Alicia, y el cambio rumbo a su propia billetera. Como una blockchain es un gran asiento contable, es importante destacar que existen Outputs gastados y no gastados, llamados TXOs

[4]”Bitcoin: A Peer-to-Peer Electronic Cash System”

[5]Telegram es un servicio de mensajería instantánea muy utilizado por la comunidad de criptomonedas

[6]1 satoshi es la unidad más pequeña de Bitcoin que puede enviarse a través de la red, y equivale a 0,00000001 Bitcoin (BTC)

Inputs	Valor	Outputs	Valor	Destinatario
input 1	0,01	output 1	0,045	Alicia
input 2	0,04	output 2	0,004	Roberto (cambio)
total	0,05	total	0,049	
comisión para minero	0,001			

Cuadro 2.1: Ejemplo de una transacción desde el punto de vista "contable" como asiento con entradas y salidas.

y UTXOs respectivamente. Para generar una transacción un usuario debe buscar un UTXO en su haber y marcarlo como input. Un UTXO es como un "billete" tradicional. Su denominación tiene el valor que se le dio en la transacción que lo originó. Por ello lo habitual es que todas las transacciones tengan este "cambio" que se menciona antes.

La tabla 2.1 muestra el ejemplo de una transacción. En la cual Roberto le envía 0,045 de alguna criptomoneda a Alicia[7]. Se supone que en la billetera de Roberto hay dos fracciones de criptomoneda sin gastar, que suman 0,05. Roberto tiene que enviar el monto requerido por Alicia y también dejar un resto para el eventual minero. Por ello la transacción tiene como destinatarios a Alicia, con el monto requerido y a Roberto, el originante, con el cambio resultante. Todo lo que reste de la diferencia entre los *Inputs* y los *Outputs* queda como comisión para quien mine un bloque e incluya esta transacción en él.

2.1.3. (3) - Resguardo del valor

Las criptomonedas son sistemas de "efectivo electrónico". Conservar el valor adquirido es algo conocido por quienes hayan tenido una alcancía, media (calcetín), cofre para ahorrar. Quien tenga acceso físico a ese medio de almacenamiento de moneda, controla los fondos. Quien tenga acceso a nuestros UTXOs controla nuestros valores. Por eso cuando se habla de Wallets, hay una división de aguas en cuanto a quién controla el acceso a las claves privadas, que no significa necesariamente una toma de posición, sino dos formas diferentes de abordar el mismo problema. En este aspecto, existen dos tipos de Wallets: Custodian y Non Custodian.

Las primeras, introducen la figura del Custodio. En este caso tampoco se ha inventado nada nuevo. Los Custodios llevan siglos operando en los sistemas monetarios del mundo. Un ejemplo de ellos, son los servicios de cajas de seguridad. Cuando Alicia se recibió de contadora, Ada, su madre, cumplió una promesa que la llenaba de orgullo: darle a Alicia el capital para abrir su Estudio Contable. Es así que Ada sacó su llave de un lugar recóndito de la casa, tomó su identificación nacional y se dirigió al subsuelo del banco a buscar esos ahorros que empezó a juntar el día que Alicia rindió su primer final. El banco actúa como custodio de todos los valores que se encuentran en esa caja de seguridad bajo dos llaves, una en su poder y otra en poder de Ada. Para acceder a sus valores, Ada tiene que identificarse ante el custodio, mostrando su identidad y utilizando su llave para desbloquear la cerradura que protege sus preciados ahorros.

[7]Se omiten la moneda y el valor para concentrarse en el concepto.

Entre el Custodio y el propietario de los valores resguardados hay una relación comercial, no sobre los valores, sino sobre la seguridad y el control restringido del acceso a ellos. Nada le impide a Ada, arquitecta de profesión, tener una caja fuerte de seguridad en su casa. Ella, por decisión propia, elige delegar todas los riesgos inherentes al resguardo de valor al banco, quien corre con los gastos que implican la infraestructura, la seguridad, los controles, y demás etcéteras que hacen que unas meras cajas con llave puedan llamarse "Cajas de Seguridad".

En criptomonedas, existen servicios de custodia de las claves privadas, su modelo de negocio es como el del banco que provee la caja de seguridad de Ada, la madre de Alicia. Estos servicios son utilizados por Exchanges [8] y plataformas de intercambio, y también por usuarios. Aunque en el caso de estos últimos, lo hacen probablemente por medio de las Wallets que les proveen los Exchanges. Utilizar una billetera tipo Custodian, es básicamente seguir los pasos de Ada, donde en lugar de un sitio físico (el banco) en el cual autenticarse en persona, se utilizará una aplicación cliente mediante un usuario, contraseña y mecanismo de autenticación de segundo factor (2FA) como lo haría en un servicio de Home Banking tradicional.

La segunda variante es tomar el camino de la custodia soberana de las claves privadas (poner la caja fuerte en casa). Para ello existen las Wallets *Non Custodian*. Este tipo de billeteras contienen las claves privadas del usuario, siendo este último el responsable de su resguardo. Al iniciar una billetera de Non-Custodian, el usuario debe inicializarla con una entropía desde la cual se derivaran las claves privadas y públicas. De las segundas se obtendrán las direcciones mediante las cuales se recibirán las transacciones.

Este proceso concentra una gran cantidad de fricción y dificultad para los usuarios. En primer lugar, los seres humanos, como sujetos racionales y subjetivos, no somos buenos seleccionando datos aleatorios. Los datos pseudo-aleatorios van a estar influenciados por los propios sesgos personales y culturales del usuario. En segundo lugar, es necesario recordar esos datos aleatorios que darán origen a la billetera. Y en tercer lugar, está la cuestión del resguardo y el acceso a ellos para disponer de los valores. El olvido o la pérdida de este dato aleatorio significa la potencial pérdida de los fondos asociados a esa clave privada.

Para el primer punto, existen numerosas aplicaciones, desde sitios web, aplicaciones de escritorio, utilitarios de terminal y hardware especializado (como las Cold Wallets Ledger o Trezor) que se encargan en generar los bytes de entropía necesarios para derivar una clave privada de forma automática sin intervención humana. En segundo lugar, esos bytes siguen siendo un problema para memorizarlos, transcribirlos para luego resguardarlos y para escribirlos a la hora de restaurar una wallet. Cualquier error tipográfico se traducirá en la pérdida del acceso a esos fondos. Para ello, es que se utilizan las frases semilla. Una serie de palabras escogidas de un diccionario de 2048 vocablos únicos que facilitan la lecto-escritura humana y a su vez permiten reproducir esos bytes de la entropía generada para la derivación de claves de una billetera. La figura 4.4 en la sección 4.3.1 muestra un ejemplo de frase semilla en inglés que se utilizará a lo largo del trabajo.

2.1.4. (4) - Enviar fondos

Aunque algunas tienen un devenir que tiende a la reserva de valor, las criptomonedas fueron concebidas como medios de intercambio mediante el cual las personas

[8] entidades que proveen servicios de intercambio de criptomonedas y monedas tradicionales

puedan comerciar sin terceros oficiando de intermediarios. Por ello, aún cuando Roberto como minero de varios tipos de criptomonedas tiene cuentas *Custodian* en casas de cambio para su conveniencia, no necesita ninguna de ellas para enviarle los honorarios al estudio contable de Alicia. Con el simple hecho de que se comuniquen entre ambos y acuerden un monto a intercambiar y una dirección de destino para una criptomoneda dada, no se requiere nada más que realizar la transferencia utilizando una wallet que haga efectivo el pago enviando una transacción a la red.

Tal como se lo explica en el capítulo "Transacciones" de *Mastering Bitcoin* [11] la parte clave de una transacción son los outputs. Para realizar el pago la wallet de Roberto tiene que buscar los outputs que no hayan sido ya gastados para llegar a acumular el monto que le requiere Alicia y cubrir la comisión del minero[9].

Las transacciones entre pares o peer-to-peer (p2p) pueden ser comunicaciones informales entre ambos, o bien utilizar un URI que detalle las direcciones a las cuales se efectuarán. El BIP-0021 [12] detalla la convención que rige estos URIs de pagos tal como lo muestra la figura 2.1.4 donde Alicia utiliza una aplicación de e-commerce que le permite solicitar un pago mediante URIs de pago de Bitcoin. Este tipo de recursos permiten pulir la fricción que hay en términos de usabilidad y experiencia de usuario y hacer la gestión de pagos más asequible y tolerante a errores.

Código fuente 2.1: Una URI representando el pedido de un pago de 0.0001 BTC a una dirección con un mensaje a mostrar al usuario "Roberto Minero"

```
bitcoin:175tWpb8K1S7NmH4Zx6rewF9WQrcZv245W?
amount=0.0001\&label=Roberto\%20Minero
\&message=Honorarios\%20estudio\%20contable
```

Cuando una transacción es enviada a través de la red y resulta incluida en un bloque, se refiere a ella como "Confirmada" con 1 (una) confirmación. Cada bloque subsiguiente sumará una confirmación a cada transacción minada. En un esquema de *Proof-of-Work* la cadena de bloques es inmutable hacia atrás, pero en su extremo final varias versiones compiten por establecer quien ha procesado la cadena que reúne la mayor cantidad de pruebas de trabajo. [10]

¿Cuántas confirmaciones ser requieren para considerar que una transacción está (valga la redundancia) confirmada? No existe una respuesta unánime a esta pregunta. Diferentes wallets y exchanges utilizan un número distinto que va desde una a más de veinte. Es probable que Alicia considere como tal el número de confirmaciones definidas por su wallet preferida.

2.2. Privacy Coins: Criptomonedas públicas con privacidad en transacciones

Cuando se habla de criptomonedas, se refiere al uso de criptografía asimétrica en la creación de un sistema de "efectivo electrónico". Sin embargo, existe la creencia de que las criptomonedas son de alguna una garantía de privacidad en sí mismas.

[9] Por el carácter de descentralizado de las criptomonedas, es muy probable que no sea el propio Roberto quien incluya el bloque con su transacción en la cadena de bloques. Los algoritmos de consenso están diseñados para distribuir la carga de la red en diferentes actores con el objetivo de mantener la seguridad de la red de pares y evitar que uno o más actores puedan tomar control sobre los demás.

[10] Este tema se profundiza en la sección 4.3.9

La realidad es que observando cualquier "Explorador de bloques"[11] es posible ver muchísima información al respecto de las transacciones, la dirección de los fondos, el monto de las mismas. De hecho, es posible calcular todos los UTXOs (los "billetes" no gastados) de una dirección dada y así dar con el balance de dicha dirección en un momento dado. Una vez que se atan los cabos sobre la identidad detrás de una dirección de criptomonedas como Bitcoin o Ethereum[12], es cuestión de poner a correr las herramientas adecuadas para tener "el debe y el haber" de esa personas y todo lo que ello conlleva.

2.3. Origen y motivación, la privacidad como derecho

> *Nadie será objeto de injerencias arbitrarias en su vida privada, su familia, su domicilio o su correspondencia, ni de ataques a su honra o a su reputación. Toda persona tiene derecho a la protección de la ley contra tales injerencias o ataques.* - Declaración Universal de los Derechos Humanos [13]

Tal como lo establece al Declaración Universal de los Derechos Humanos, la privacidad es parte esencial de la vida. Dibujar la frontera entre el ámbito público y el privado es una decisión personal e íntima de cada individuo. La definición citada al comienzo de esta sección pone en pie de igualdad a una injerencia sobre la privacidad de alguien y la palabra ataque. Entendiendo que lo privado es aquello que no se quiere mostrar abiertamente al público, el acceso por parte de terceros a información que una persona ha decidido reservar para sí, conlleva de plano una ruptura al consentimiento de ese individuo y por añadidura a su libertad. En esos términos se puede equiparar el hecho de no tener un opción de ejercer la privacidad en monedas como Bitcoin a la pérdida de libertad.

Estas cuestiones no son ajenas al debate entorno a Bitcoin. Prueba de esto son los dichos del propio Satoshi Nakamoto en los foros «bitcointalk.org» en agosto de 2010. A quien se atribuye la idea fuerza de la criptomoneda Zcash. El primer recuadro sobre la figura 2.1 dice "*Este es un tópico muy interesante -el de la privacidad- . Si se encontrase una solución, una implementación mucho mejor, más fácil, más conveniente de Bitcoin sería posible*". El dilema que enfrenta Bitcoin, es que necesita que la información esté disponible a todos de la misma forma y así evitar "doble gastos" (falsificación de bitcoins). Si parte de la información de la cadena de bloques esta resguardada de los demás, cómo sería posible argumentar que en algún lugar de la cadena de bloques alguien ya se ha gastado antes esta porción de bitcoin[13] que está intentado gastar actualmente. El segundo recuadro "Es difícil pensar cómo aplicar pruebas de cero conocimiento en este caso".

Pasaron varios años hasta que las pruebas de Cero Conocimiento llegaran, no a Bitcoin, sino a un fork, que pasó a llamarse Zcash. Zcash no es la única criptomoneda que tiene transacciones "privadas". Monero, es otra de ellas. Su funcionamiento es muy

[11] Un explorador de bloques o Block Explorer, en Inglés, es un sitio web que permite a sus visitantes realizar consultas sobre los bloques de una o más blockchains.

[12] Criptomoneda principal que corre sobre la red Ethereum

[13] Se utiliza el nombre de la criptomoneda en minúscula para referirse a una porción de fondos de la misma, mientras que el nombre de la moneda en mayúscula denota que se refiere al proyecto.

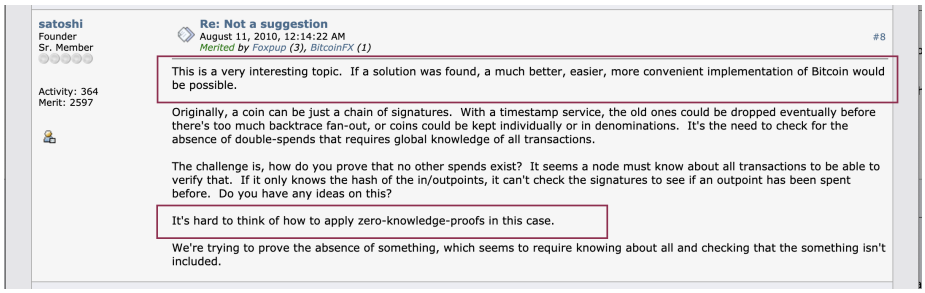


Figura 2.1: Bitcoin Talk: el dilema de Satoshi Nakamoto y la privacidad de la blockchain.

distinto al de Zcash, comenzando por el hecho de que utiliza el protocolo CryptoNote [6] o RingCT que consiste a grandes rasgos en agrupar las transacciones en grupos e intercambiar las claves con las que se firman con el objetivo de anonimizar a quien las envía.[14] Algo similar ocurre para el caso de la criptomoneda Dash y su método 'PrivateSend', donde las transacciones pueden ser privadas o públicas a elección del remitente. El *PrivateSend* solo va por una red de *mixers* que mezclan los outputs e inputs de esa transacción con las de otras transacciones repetidas veces, con el objetivo de dificultar el rastreo y seguimiento del tráfico de la red y de la blockchain[15].

2.4. Zcash, a 10.000 metros de altura

En las siguientes secciones se hará un repaso de los puntos referidos a blockchain en general, pero esta vez explicando las particularidades de Zcash.

Zcash es un fork Bitcoin del código fuente de Bitcoin, más precisamente hablando, la versión Bitcoin Core 0.14. Todas las funcionalidades de Bitcoin, se utilizaron como base para una modalidad conocida como 'Zcash Transparente'. Por más que funcionalmente esta parte casi equivalente a dicha versión de Bitcoin, no hubo un "chain fork", por lo cual Zcash y Bitcoin, a pesar de compartir parte del código de consenso y de los nodos, tienen blockchains diferentes.

La otra parte de Zcash es aquella conocida como Blindada (shielded), que es la que hace uso de las pruebas de conocimiento-cero llamadas ZK-SNARKs. Las transacciones blindadas, hacen que el remitente de una transacción, pueda probar que esos zatoshis [16] no han sido gastados previamente.

Las transacciones dentro de Zcash tienen inputs y outputs transparentes pero también pueden tenerlos blindados. Las partes transparentes de una transacción pueden verse de la misma manera que en Bitcoin, están allí a la vista de todos, mientras

[14] Es pertinente mencionar que existen indicios de que compañías de rastreo de blockchains como «chainalysis» han roto el protocolo CryptoNote según sus propias aseveraciones y que pueden deshacer esa mezcla de firmas y rastrear quien ha enviado y cada una de las transacciones de un grupo.

[15] En el transcurso de la elaboración de este trabajo, los creadores de Dash han realizado ciertas actualizaciones que podrían indicar que esta criptomoneda y su comunidad desean abandonar el espacio de las AEC. Se ha decidido mantener esta mención en referencia al estado de esa criptomoneda al momento de la confección de la propuesta.

[16] Zatoshi es el equivalente de Zcash a la unidad Satoshi de Bitcoin

que los inputs y outputs blindados solo pueden ser vistos por quienes tengan las claves criptográficas adecuadas para desencriptarlos. Esta diferencia puede aparentar ser simple pero tienen muchas implicaciones en el funcionamiento de la criptomoneda, en especial desde el punto de vista de los usuarios.

2.4.1. Identidad: anonimato y pseudoanonimato

En Zcash, existen 4 tipos de transacciones. Las primeras son llamadas "t2t", transparente a transparente y son prácticamente equivalentes las de Bitcoin. Las segundas son aquellas consideradas "blindadas" (Shielded) o "z2z". Estas transacciones contienen inputs y outputs encriptados cuya verificación global se realiza mediante las pruebas de cero conocimiento. Solo quienes tengan las claves adecuadas podrán desencriptar la información que hay en esos componentes (dirección de destino, dirección de origen, valor y y el memo encriptado). Los otros dos tipos de transacciones corresponden a si los inputs o outputs son blindados y sus contrapartes transparentes. Las transacciones denominadas "z2t" contienen inputs blindados y outputs transparentes y se las denominan como "desblindar" (deshielding); mientras que la inversa "t2z" contienen inputs transparentes y outputs blindados y tiene como propósito blindar (shielding) fondos transparentes.

En Zcash, Alicia tiene dos identidades: una anónima y otra pseudoanónima. Estas son sus direcciones blindadas y transparentes respectivamente. De esta forma ella puede elegir cómo sus transacciones se verán a los ojos del público. En cualquiera de los casos, la comisión de los mineros es información pública en la blockchain. Las transacciones blindadas son fácilmente distinguibles de las transparentes en los exploradores de blockchain. En la sección que estaría dedicada a enumerar las entradas y salidas de la transacción, no habrá información alguna mas que el número de éstas, sin ningún otro detalle. A diferencia de las transacciones transparente que son equivalentes a las de Bitcoin al momento del fork, las blindadas no son distinguibles unas de otras. Por lo tanto el análisis de metadatos por parte de los exploradores de cadenas son hasta el momento infructuosos. No obstante, la privacidad no es absoluta pues, la frontera entre lo blindado y lo transparente ofrece un punto de referencia dichos análisis. Tal como la privacidad en la vida misma, puede ser puesta en jaque por las propias acciones de quien busca (infructuosamente) mantenerla. Así como en materia de seguridad informática suele decirse que "la única computadora segura es aquella que se encuentra apagada, dentro de un bloque de concreto, en una habitación sellada con plomo y rodeada con guardias armados y aún así tengo mis dudas"[17], los fondos realmente privados son aquellos cuyo origen es blindado y jamás se mueven de lugar ni se opera con ellos de forma alguna. De todas formas, a sabiendas de que cada operación tiene la potencialidad de revelar un poco de aquello que deseamos conservar en el ámbito privado, las privacy coins proveen una mejora substancial en la soberanía de la información de quienes operan con ellas. Este tema excede el alcance del presente trabajo, pero se considera importante su mención aunque sea para delinearlo como una linea de trabajo importante en materia de privacidad de la información.

Una de las grandes funcionalidades que diferencia a Zcash del resto de las AECs es la posibilidad de control sobre qué información se expone y a quiénes. No solo en cuanto a la posibilidad de operar tanto en modalidad blindada como transparente,

[17] La frase original, "The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards - and even then I have my doubts." se atribuye a Gene Spafford

sino que la modalidad blindada se encuentra estratificada de forma tal que, de precisarlo, una persona puede generar claves de visibilidad (Viewing Keys) que le permiten desencriptar parte (o la totalidad) de una o más transacciones [14] [15]. De esta forma Alicia puede utilizar una *viewing key* derivada de la frase semilla de Roberto para hacer su trabajo de contabilidad sin que éste ponga en riesgo sus fondos, puesto que dichas claves no tienen autoridad de gasto alguna.

2.4.2. Recibir Fondos

Llega el fin de mes y Alicia le manda su factura a Roberto, quien le comenta que este último mes ha sumado una nueva moneda a su cartera: Zcash. Tuvo la dicha de poder minar algunos bloques y utilizando la modalidad de "coinbase blindado" (shielded coinbase) [16], la recompensa al minero (miner's reward) le fue transferida a su dirección blindada directamente. Alicia entonces le envía un email (ver 2.4.2) comentándole que ella tiene una wallet Zcash y le pasa su dirección blindada Sapling[18] que comienza con la letra 'z'.

> Roberto, esta es mi dirección blindada de Zcash
>
> zs1t2scx025jsy04mqyc4x0fsyspxe86gf3t6gyfhh9qd
>
> zq2a789sc2eccslflawf2kpuvxcqfjsef
>
> te pido que en el memo incluyas tu dirección de origen y el mensaje "Honorarios de Noviembre. Roberto".

Las transacciones blindadas de Zcash tiene un campo llamado 'memo' en cada uno de sus outputs, también llamados "encrypted memo fields". Ellos permiten adjuntar 512 bytes encriptados a cada output de una transferencia. En principio habilitan un campo para la comunicación entre remitente y destinatarios de una transacción, aunque es posible utilizar ese campo para incluir metadatos en las transacciones para permitir cierta "programabilidad" mediante mensajes estructurados que distintas aplicaciones clientes pueden leer sobre la blockchain utilizándola como fuente de datos como si fuera un back-end. [19]

Cuando Alicia recibe la transacción de Roberto, a priori, no podrá ver quién le envió la transacción, pues esa parte de la transacción está encriptada con una clave privada que no le pertenece y que sus propias claves no pueden desencriptar. Por ello es que le hace ese pedido especial al remitente de incluir su dirección en el memo para poder identificar la transacción rápidamente. Este mecanismo tiene una falla, pues es susceptible a que un tercero se presente como el remitente original con el objetivo de confundir al receptor. Por ello, la forma adecuada de incluir una dirección de respuesta en un memo, debería incluir la firma de la dirección de forma tal que solo Roberto puede producir una firma que verifique la autoría del mensaje en base a su dirección Sapling y nadie más[20]. No obstante, es posible lograr esto mismo de otra manera utilizando

[18] En el protocolo Zcash, el término Sapling refiere tanto al nombre que toma el Network Upgrade mediante el cual se despliega la implementación en la red, como la criptografía de cero-conocimiento que reemplaza a la versión inicial conocida como Sprout

[19] Existen varios proyectos en este sentido, como Zec Pages una BBS (bulletin board service) que se monta sobre la blockchain de Zcash, o Zbay, una herramienta de comunicación grupal y marketplace.

[20] Estas firmas son llamadas Sapling Address Signatures. `https://zips.z.cash/zip-0304`. Permiten utilizar una prueba que no puede ser falsificada por terceros. Los receptores de una dirección conocida y una firma, pueden tener una prueba fehaciente de que es el dueño de esa dirección quien firma ese mensaje que incluye la dirección a modo informativo.

una funcionalidad llamada "Divulgación de pago" [17] [18] (Payment Disclosure). Le permite al remitente de transacción blindada (z2z), generar una divulgación selectiva de la información contenida en esta, para que un tercero que no posea las claves para verla en su totalidad, pueda hacerlo.[21] Cuando la red de pares genera un nuevo bloque y el pago de Roberto es minado, Alicia puede tomar la transacción que se encuentra en la blockchain y utilizar los datos que recibió de Roberto como divulgación de pago para verificar que esa transacción es efectivamente suya. En Zcash, el balance de una cuenta se calcula en base a los inputs y outputs de las transacciones. El balance que corresponde a las operaciones realizadas en la parte transparente del protocolo, es llamado "Transparent Zcash" y se calcula en base a la sumatoria de UTXOs de igual manera que en Bitcoin, mientras que para las transacciones blindadas, el balance de "Shielded Zcash", se calcula en base a los "notes" que no se han gastado. A diferencia que los UTXOs, que son transparentes a todos los usuarios, estos "notes" (la traducción sería "billetes", pero la obviaremos porque puede resultar confuso conceptualmente) se obtienen de desencriptar las transacciones con una Incoming Viewing Key (ivk). Para conocer cuanto valor resguarda una clave privada (haciendo usa de la ivk derivada de ella), se requiere sincronizar o escanear todos los bloques en busca de aquellos que las claves en poder del usuario pueden desencriptar.

2.4.3. Resguardo de valor

En este punto no hay mayores cambios conceptuales respecto de otras criptomonedas Proof-of-Work como Bitcoin, pero sí es necesario resaltar que para el caso de Zcash, los usuarios tienen dos tipos de balances, uno transparente y otro blindado. La diferencia entre con el punto de la sección anterior, yace principalmente en la disponibilidad (de mercado) de servicios Custodian que soporten direcciones blindadas de Zcash, mientras que las transparentes por su compatibilidad (técnica) con Bitcoin las hace mas sencillas de implementar. Al tiempo de redacción de este trabajo son no más de dos los Exchanges que soportan extracciones con direcciones blindadas de Zcash. En el modelo Non-Custodian, también se observa este fenómeno. Pese a que Zcash es una de las criptomonedas más populares, existen solo cinco wallets non-custodian (Zec Wallet Lite, Zec Wallet CLI, ECC Wallet, NightHawk y Unstoppable) entre las cuales se incluye la wallet de referencia de este proyecto; y se encuentra en desarrollo un firmware que permita a la wallet Cold-Storage Ledger soportar direcciones Blindadas de Zcash.

2.4.4. Enviar Fondos

Al enviar fondos con Zcash, existen ciertas variantes que surgen del origen de los fondos utilizados y el tipo de dirección del destinatario. Como se menciona en la sección 2.4.1, es posible enviar fondos entre direcciones blindadas o desde y hacia transparentes y blindadas. Una wallet de Zcash que soporte todas las funcionalidades previstas en el protocolo, deberá poder derivar direcciones blindadas y transparentes desde las claves privadas del usuario y por añadidura (y estándar *de facto* de la industria) presentar códigos QR de las mismas.

[21] La definición de esta funcionalidad está habilitada por el protocolo, pero no se encuentra desarrollada para transacciones Sapling (si de forma experimental en el protocolo Sprout). La definición de esta funcionalidad se está en revisión y su futura publicación podrá hallarse en `https://zips.z.cash/zip-0311`.

En cuanto a los URIs de pagos, Zcash tiene su propio ZIP[22] referido a ellas [19]. Es similar al citado sobre Bitcoin, pero incorpora las particularidades de Zcash, al poder combinar direcciones transparentes y blindadas y para estas últimas también adjuntar memos encriptados.

2.5. Monero, a 10.000 metros de altura

Monero una criptomoneda con privacidad creada en el 2014, en base al protocolo entonces conocido como "RandomX", luego difundido como "CryptoNote". Surge también desde el objetivo de proveer privacidad en la cadena de bloques, mediante un mecanismo de ofuscación de las transacciones, sus montos, remitentes y destinatarios. A su vez el protocolo contempla un mecanismo de ocultamiento del grafo transaccional direccionado que se genera con la operatoria de la criptomoneda.

2.5.1. Identidad: anonimato y pseudoanonimato

El libro "Mastering Monero" [20] caracteriza a la criptomoneda como "privacidad por defecto", donde se hace énfasis en la privacidad del "libro contable" y del enmascaramiento del grafo de transacciones. Una identidad dentro de monero se establece, al igual que en otras criptomonedas, a través de una frase semilla o bytes iniciales y genera cuatro claves distintas: una clave pública de visualización (public viewing key) utilizada para verificar la validez de las direcciones, una clave privada de visualización (private view key) utilizada para visualizar el balance, las comisiones de los mineros y el monto de las transacciones, una clave pública de gasto (public spending key) utilizada para verificar transacciones y una clave privada de gasto (private spend key) utilizada para firmar transacciones y enviar Monero. La dirección es la representación de las claves públicas.

La identidad de los usuarios de Monero se encuentra oculta detrás de varios mecanismos que se describirán en las siguientes secciones.

2.5.2. Recibir Fondos

Para recibir Monero se requiere un dirección, que es la representación textual de una clave pública. A diferencia de Zcash no existe una diferencia entre transacciones blindadas o transparentes. En Monero todas las direcciones tiene las mismas capacidades. Solo se las distingue entre principal y secundaria. La diferencia entre una y otra solo es el nivel de derivación criptográfica de las mismas donde la dirección principal se distingue por comenzar con el carácter '4' mientras las secundarias lo hacen con el carácter '8'. Se considera que la dirección principal es estática, única para una determinada frase semilla, mientras que las secundarias son derivadas de la primera. Su rol es posibilitar a los usuarios proporcionar distintas direcciones a diferentes actores. Todos los fondos recibidos serán asociados a la dirección principal.

[22] ZIP refiere a *Zcash Improvement process*. Los *Zips* son documentos propuestos por personas o instituciones interesadas en proponer un cambio o una mejora en el protocolo Zcash o en su funcionalidad general. Pueden consultarse en https://zips.z.cash

2.5.3. Resguardo de valor

A diferencia de Bitcoin, Zcash o Ethereum, la generación de los bytes que derivan las claves de un usuarios, no es a través del estándar de Bitcoin BIP-39 [21] sino mediante uno propio. Es posible que distintas wallet de Monero no posean frases semillas compatibles entre sí. La identidad depende de los bytes generados por esa convención. Fuera de esta particularidad, aplican los mismos preceptos que para las demás monedas.

2.5.4. Enviar Fondos

Para garantizar la privacidad de las transacciones, Monero dispone de una combinación de mecanismos. El primero es la generación de direcciones de un solo uso. Cuando Alicia le requiera sus honorarios a Roberto, su wallet generará una dirección nueva que solo ella puede asociar a su dirección principal. Esa clave pública será uno de las entradas que la wallet de Roberto utilizará para usar una dirección sigilosa (stealth address), desde la cual se hará el envío de los fondos que Alicia le requiere desde su dirección de un solo uso. De esta forma se pretende lograr un grado mayor de anonimato de las partes involucradas en una transacción ya que las direcciones no se podrían asociar a las cuentas originantes[23]. El resto de los datos de las transacciones fueron de carácter público hasta que se incorporó una técnica llamada Transacciones Confidenciales en Anillo (RingCT) donde el emisor de la transacción encripta el monto comprometido en ella. Para poder verificar dichos montos terceras partes utilizan un árbol de *Pedersen Commitments*. Cuando Roberto envía la transacción a Alicia, los nodos implementan una tercera metodología de enmascaramiento del grafo transaccional, llamada Firmas en anillo (Ring Signatures). El protocolo Monero obliga a los nodos a enviar las transacciones junto con otras N transacciones señuelo, que simulan ser transacciones legítimas al receptor de la transacción original, integrada por outputs ya utilizados anteriormente en la red y firmada con otras claves[24]. Esta técnica procura ocultar la transacción original en un "cardúmen" de otras transacciones aparentemente válidas para un observador externo.

2.6. Tipos de Wallets

2.6.1. Wallet Full Node

Como lo indica su nombre, se trata de una wallet que contiene un nodo corriendo localmente en el mismo equipo y lo utiliza para interactuar con la blockchain y otros nodos de la red. Su arquitectura es monolítica, donde en un mismo equipo corre el nodo en modo servidor, y un cliente se conecta a él localmente.

Las wallets full node contienen una copia completa de la blockchain. Para poder utilizar este tipo de wallet es preciso, descargar y verificar cada bloque de la cadena desde el génesis hasta último minado. Por ello una buena opción para quienes no puedan o necesiten tal nivel de constatación y almacenamiento, protocolos como Bitcoin o Zcash contemplan clientes livianos.

[23] Contrario a lo que reclaman sus creadores, distintos trabajos demuestran que es posible desanonimizar el grafo de transacciones de Monero

[24] Existen varias críticas a este modelo de ocultamiento del grafo transaccional, la más mencionada son las transacciones *zero-mixins*, que refiere a aquellas cuyo anillo es nulo (N=0) o de un N lo suficientemente pequeño que un observador pasivo puede descifrar.

En el caso de Bitcoin o Zcash, el código fuente del nodo produce un ejecutable con el *daemon* del nodo (bitcoind o zcashd) y el cliente de terminal (bitcoin-cli o zcash-cli). Las wallets full node son bastante populares entre quienes han adoptado la tecnología blockchain de manera temprana o la utilizan en PCs de escritorio que tienen recursos de sobra para correr el sistema operativo y las exigentes especificaciones de los nodos. En cierta forma es el precio que se paga para obtener una configuración más segura de conectarse a la red desde y mediante un nodo que el usuario controla completamente.

No es necesario ser un experto para correr este tipo de wallets. De hecho, las wallets más populares de criptomonedas privadas como Zcash (ver figura 2.2) o Monero tienen su variante full node como versiones iniciales y luego han ido incorporando versiones "lite".

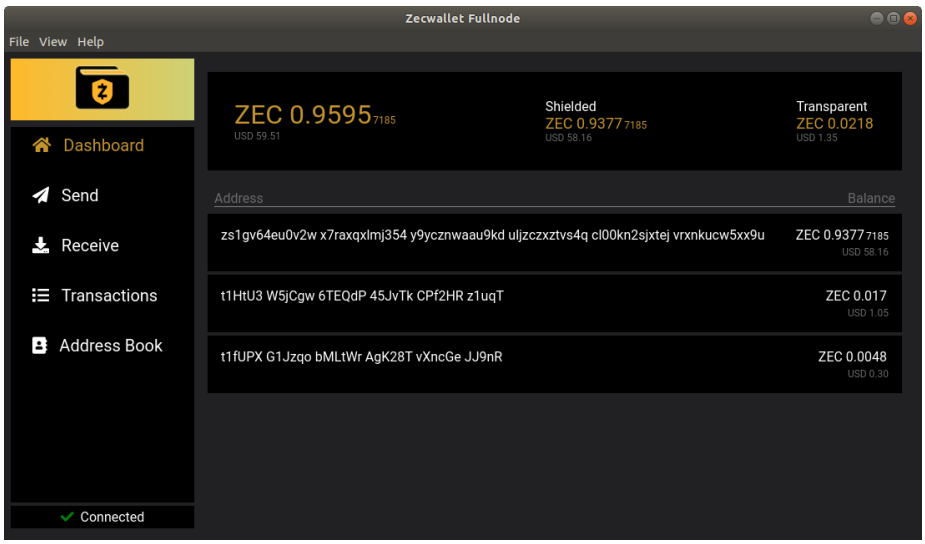


Figura 2.2: ZecWallet Full Node. Billetera de Zcash

2.6.2. Clientes Livianos

Este tipo de clientes surgen de la necesidad de poder operar en una blockchain sin tener que contar con una copia completa de la misma y correr un nodo completo. Lo primero que viene a la mente en estos casos son las aplicaciones móviles, pero además conforme el uso de una blockchain crece, no solo se prolonga en cantidad de bloques, sino que cada uno de ellos contiene más transacciones, volviendo menos conveniente correr clientes full node en computadoras personales.

Además, reducir la cantidad de información por bloque procesado, los clientes livianos pueden no contener la totalidad de ellos. Si una persona quiere crear una nueva clave privada para utilizar una criptomoneda, sabe a priori que no habrá transacciones que le pertenezcan a esa clave antes del momento de su creación. Si se sincronizara desde el bloque génesis hasta la punta de esa blockchain no se encontraría transacción alguna. Esto lo garantiza el amplio espacio de la combinatoria de claves privadas posibles, donde una colisión de la función de hashing para una clave generada desde

bytes aleatorios es considerada "imposible". Por tanto, carece de sentido desde la óptica de un cliente liviano, descargar la porción de blockchain previa a la existencia de dichas claves.

De allí surge la idea de que la claves privadas tienen una "fecha de nacimiento" expresada en la altura vigente al momento de su creación para una cadena de bloques en particular. Para poder utilizar estas fechas de nacimiento, es necesario obtener un punto de referencia sobre el cual un cliente liviano pueda retomar la lectura y verificación de la porción de la cadena de bloques de su interés. Estos puntos de referencias se llaman "checkpoints" y son básicamente una fotografía de la verificación del Árbol de Merkle de la cadena de bloques. Un checkpoint es una pequeña estructura de datos que contiene la altura que representa, su timestamp unix, el hash del bloque anterior y hash del Merkle Tree. Esos cuatro datos permiten que un cliente liviano pueda retomar la verificación del la cadena de bloques desdes cualquier punto dado, lo cual permite ahorrar ancho de banda y tiempo de procesamiento a la hora de sincronizar una clave privada con la cadena de bloques.

Los clientes livianos, deben utilizar un nodo completo como back end ven la blockchain a través suyo. En su whitepaper, Satoshi Nakamoto presenta el modelo de verificación de pagos simple (SPV por sus siglas en inglés) donde propone un tipo de cliente que descargue solo los encabezados de los bloques para poder verificar que un pago recibido pertenece a la cadena de bloques. Zcash propone un modelo alternativo llamado *Light Client Protocol for Payment Detection* [22] (conocido como *lightwalletd*, por su implementación) en el cual los clientes livianos se conectan a un servidor que les provee una versión compacta de la cadena de bloques de Zcash, que este obtiene y procesa a medida que se generan y publican nuevos bloques. Lo cual permite un ahorro de un 90 % en la cantidad de datos descargados, sin relegar la posibilidad de desencriptar transacciones que pertenezcan a una o más claves determinadas. La arquitectura que se describirá en este trabajo considera la utilización de este protocolo. La sección 4.3.9 cubre en mayor detalle el proceso de sincronización de un cliente liviano mediante la utilización de un servidor Lightwalletd.

Capítulo 3

Revisión Sistemática de la Literatura

3.1. Alcance y diseño de la Revisión Sistemática de la literatura

A los efectos de establecer puntos de mejoras y áreas a profundizar se diseño un estudio de Revisión Sistemática de la Literatura (SLR) con el fin de relevar el estado del arte que es objeto del presente trabajo. Existen varias metodologías para conducir este tipo de investigaciones. La revisión realizada se basa en el trabajo [23], que trata las especificidades que presenta el campo de la ingeniería de software para las SLR. Adicionalmente se utilizó una herramienta llamada "*Scolr*" para capturar todo el proceso y generar los reportes con los resultados que se detalla en la sección "Metodología". Para esta revisión se utilizaron los siguientes repositorios de literatura:

- Scopus
- ACM Library
- IEEE Explore

Se dejaron fuera de esta revisión:

- Repositorios de Software Libre y/o de Código Abierto
- White Papers y documentación en sitios web de los propios proyectos a analizar

3.1.1. Preguntas a responder

Este estudio apunta a relevar literatura alrededor de las siguientes interrogantes:

- ¿Qué diferencias de requerimientos funcionales y no funcionales existen entre wallets que soportan Privacy Coins en Desktop, Mobile y Web?
- ¿Qué experiencias hay del cómputo de transacciones del lado del cliente?
- ¿Qué esquemas de respaldo de cadena de bloques se pueden utilizar?

No se establecieron dimensiones generales a analizar, sino conocer en líneas generales los aportes de los trabajos que se hayan encontrado en la búsqueda de acuerdo a los criterios seleccionados.

3.2. Metodología

3.2.1. Procedimiento

La revisión sistemática de la literatura fue realizada utilizando *Scolr* [1], cuyo flujo de trabajo se describe a continuación

Planificación Los revisores definen las preguntas que la revisión intentará responder, junto con los criterios de inclusión y exclusión de la bibliografía. Esta información se encuentra disponible a lo largo del proceso para consulta. En base a estas preguntas se definen las consultas que se realizarán en las bases de datos que se definan, junto con las palabras clave a utilizar y los operadores lógicos.

Carga Se seleccionan las bases de datos de literatura que se consideren necesarias para consulta y se realizan las búsquedas en términos de lo definido en la etapa anterior. Los resultados son descargados en un formato adecuado (bibtex o csv), que deben contener título, autor, año y abstracto. *Scolr* detecta duplicados incluyendo estas ocurrencias solo una vez en los resultados a utilizar en etapas posteriores.

Clasificación Todos los participantes en la revisión deben expresar para cada artículo, si. debe ser incluido o no en base a los criterios de exclusión e inclusión definidos mediante la lectura de los abstracts. Al final de esta fase todos los artículos están en verde (incluidos), rojos (todos acuerdan en excluirlos) o amarillos (hay desacuerdos). Antes de seguir con la siguiente fase todos los desacuerdos deben ser resueltos.

Revisión Solo los artículos en verde alcanzan este punto, donde comienza el núcleo del trabajo de revisión. Para ellos se obtiene el texto completo de todas las fuentes. *Scolr* no ofrece funcionalidad alguna por tanto deben obtenerse por otros medios pertinentes. En este punto ya se tienen una idea del universo de artículos a revisar y de las palabras clave recolectadas de la literatura seleccionada, por lo tanto pueden definirse dimensiones en las cuales se revisarán los artículos. La herramienta provee una dimensión inicial llamada "aspectos generales", pudiéndose agregar otras durante la revisión de los artículos. Cuando se agrega una dimensión esta queda disponible para todos los demás. Los autores de esta herramienta recomiendan que los artículos sean leídos al menos por dos revisores.

Report *Scolr* permite realizar un reporte automatizado de los contenidos vertidos en la herramienta. Este capítulo está basado en dicho reporte, y ha sido ajustado en base a los resultados y conclusiones del trabajo de Revisión Sistemática de la Literatura tal como se planteó en los objetivos del mismo.

3.2.2. Criterios de clasificación

Inicialmente se propusieron los criterios preliminares de selección del material a revisar:

- Se refiere una o más monedas listadas en el compendio de privacy coins [8]
- El proyecto se encuentra desplegado en producción (es decir, en el mercado)
- Utiliza blockchain

[1] http://scolr.cientopolis.org

- la 'privacidad' no es obtenida mediante técnicas de «Anonimización» y/o manipulación de transacciones públicas como CoinJoining, Snowballing

Sin embargo cabe destacar que luego de pasar por el conjunto de los resultados por el tamiz de los criterios de clasificación, se concluyó que resultaban muy excluyentes, puesto que eliminaban más del 95 % de los resultados.

En base a eso se realizaron otras búsquedas en las mismas fuentes con dos criterios: ampliar el espectro mediante el reemplazo de los AND por OR y aumentar la especificidad utilizando solo los términos PRIVACY, CRYPTOCURRENCY, ZCASH, MONERO. Siendo que los resultados obtenidos o bien contenían a la totalidad de los artículos iniciales o bien eran un subconjunto muy menor de ellos, se concluyó que se mantendrían las búsquedas planteadas inicialmente y que en su lugar se modificarían los criterios de selección. Que se detallan a continuación:

- refiere una o más monedas listadas en el compendio de privacy coins [8]
- refiere a métodos de pago con dispositivos móviles
- aborda el asunto de la privacidad en las transacciones realizadas por medios electrónicos
- discute la utilización de clientes livianos en tecnologías móviles

De esta forma se amplía el carácter exploratorio de la revisión, que permitió finalmente arribar a conclusiones respecto de la metodología empleada en realización de este tipo de relevamientos sistemáticos en el ámbito del estudio aspectos ligados a criptomonedas que se detallan en las conclusiones3.5.

3.3. Análisis

3.3.1. Resultado de las búsquedas

La Tabla 3.1 provee un panorama de las búsquedas realizadas. La primera columna indica la base de datos bibliográfica utilizada, "Artículos" indica el número de artículos obtenidos para cada fuente y la consulta en cuestión. Las consultas fueron realizadas en orden de aparición donde solo se incluyen los artículos no repetidos. El proceso de búsqueda arrojó un total de 193 artículos.

3.3.2. Aclaraciones generales sobre el proceso de clasificación

Durante el proceso de clasificación se evaluaron todos los artículos hallados contra los criterios de exclusión e inclusión. Cada articulo fue señalado para ser incluido o excluido. Del total de 193, un total de 18 fueron seleccionados para ser incluidos en el reporte para la fase de revisión.

La figura 3.3.2 muestra los artículos incluidos por año de publicación.

3.3.3. Aclaraciones sobre la etapa de revisión

Cada artículo que pasó a esta etapa fue leído en su totalidad y resumido bajo la dimensión "apreciaciones generales". Resultó ser el caso de algunos artículos que presentaban indicios para ser incluidos en su abstracto, pero que en una lectura completa

Fuente	Texto de búsqueda	Artículos
Scopus	(mobile OR desktop OR Web) AND privacy AND (cryptocurrency OR coin) AND (requirements OR architecture OR Computing OR processing OR transaction) or wallet	27
IEEExplore	(mobile OR desktop OR Web) AND privacy AND (cryptocurrency OR coin) AND (requirements OR architecture OR Computing OR processing OR transaction) or wallet	1
ACM library	(mobile OR desktop OR Web) AND privacy AND (cryptocurrency OR coin) AND (requirements OR architecture OR Computing OR processing OR transaction) AND wallet	165

Cuadro 3.1: Consultas realizadas por repositorio y resultados arrojados

Figura 3.1: Distribución de los artículos incluidos por año de publicación

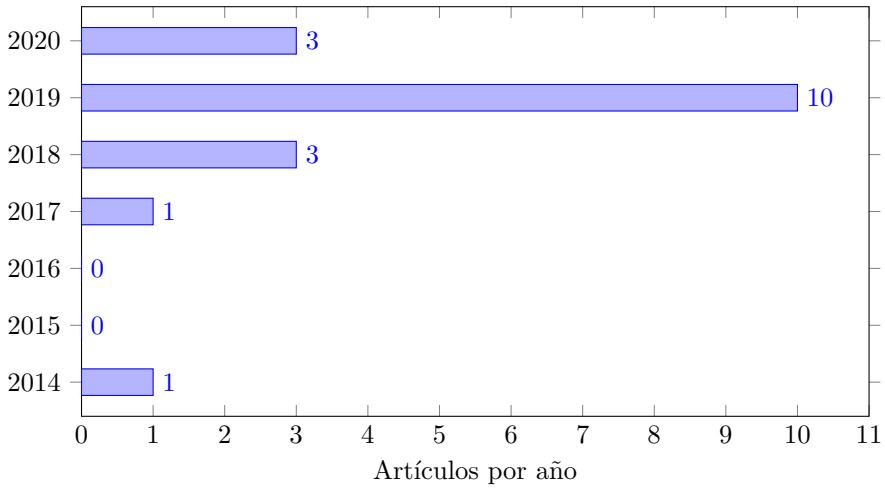
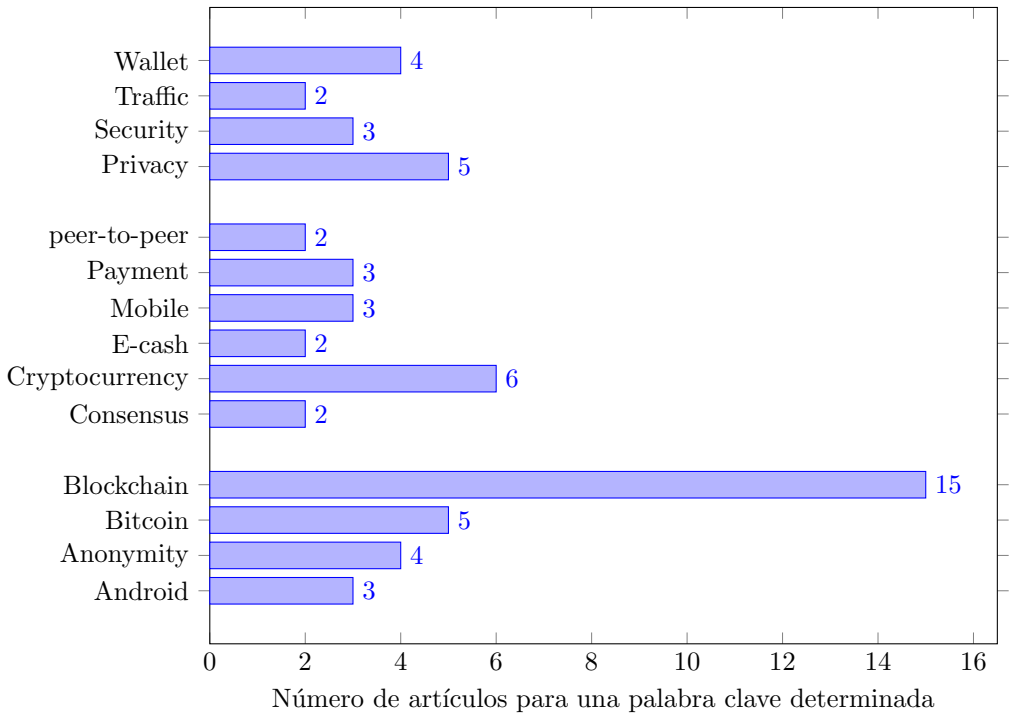


Figura 3.2: Palabras clave con N>1 en artículos revisados



fallaban en cumplir los criterios de inclusión o cumplían los criterios de exclusión y por tanto fueron excluidos de esta fase.

En la figura 3.3.3 se detallan todas las palabras claves encontradas en estos artículos con más de una ocurrencia. Para los casos en los que las palabras clave estaban compuestas, por ejemplo "Privacy and Anonymity" se computa una ocurrencia a "Privacy" y una "Anonymity". De la lista sin depurar se aprecia una distribución muy dispersa de las palabras clave.

3.4. Revisión

Estas son todas las notas recogidas en la revisión agrupada por dimensión y luego artículo.

3.4.1. Comentarios Generales

A Pattern Collection for Blockchain-Based Applications

Los autores abordan la problemática de la falta de enfoques formales en el desarrollo de software en base a tecnologías que emplean cadenas de bloques. Realiza

definiciones básicas sobre la materia. Define como propiedades fundamentales de una cadena de bloques la inmutabilidad y el no-repudio de la información firmada criptográficamente y almacenada en la cadena; la integridad de los datos; la transparencia y la equidad de derechos sobre los mismo; y la confianza que se desprende de la propia interacción de los numerosos actores que la componen y operan en ella. Define como grandes limitaciones la privacidad de los datos y la escalabilidad. A modo de conclusión, el trabajo [24] establece cuatro categorías para los patrones que sus realizadores han relevado: Patrones para la interacción con el mundo exterior, Patrones de Manejo de Datos, Patrones de Seguridad y Patrones estructurales de Contratos (Smart Contracts). Dentro de estas categorías enumeran 15 patrones arquitecturales para aplicaciones basadas en blockchain utilizando el formato descrito en el paper "A Pattern Language for Pattern Writing. Pattern languages of program design", donde cada uno contiene su resumen, contexto, problema a resolver, etc.

A Survey on Privacy Concerns in Blockchain Applications and Current Blockchain Solutions to Preserve Data Privacy

El paper [25] repasa conceptos básicos de la blockchain como Transparencia, carencia de un punto único de falla, inmutabilidad, descentralización, Contratos Inteligentes. Realizaron búsquedas sobre los términos 'privacidad' (privacy), 'blockchain', 'smart contracts' (contratos inteligentes) entre los años 2015 y 2019 en los sitios IEEE y ScienceDirect. Como criterio de aceptación seleccionaron aquellos artículos que analizaban arquitecturas basadas en blockchain o algoritmos que preservaran la privacidad. Rechazaron incluir todos aquellos artículos en cuyo título y abstract no se pudieran deducir el cumplimiento de los criterios de inclusión. Repasan superficialmente algunos puntos delicados en cuanto a la privacidad en diversas aplicaciones de blockchain como aplicaciones financieras, Sanitarias e Internet de las cosas, haciendo hincapié en el registro (ledger) público de las transacciones como el mayor obstáculo para la privacidad. Relevan trabajos sobre el uso de cadenas de bloques privadas para fines particulares como preservar la privacidad de la información en dispositivos móviles, hogares inteligentes y contratos inteligentes. Esta SLR comparte criterios y temporalidad con la del presente trabajo pero se aprecia poco abarcativa y relevante, a punto tal que al enunciar el problema del ledger público no hacen mención alguna a la blockchains con algún grado de privacidad como son Zcash, Monero o Dash, el uso extensivo de pruebas de cero-conocimiento, Ring Signatures o Mixnets.

A survey on anonymity and privacy in bitcoin-like digital cash systems

Khalilov y Levi realizan en [26] un estado del arte sobre la trazabilidad de Bitcoin y propone un estudio sobre el anonimato y la privacidad en Bitcoin y otras criptomonedas afines. Recorren aspectos básicos de la utilización de Bitcoin, explicando conceptos como blockchain, transacciones, doble gasto, direcciones. Incluye una taxonomía sobre los tipos de estudios de análisis de anonimato y privacidad en Bitcoin, clasificándolos por sus resultados y su metodología. Demuestran un estudio exhaustivo de la literatura disponible. Aportan comparativas de las metodologías de desanonimización existentes en base a contextos y resultados. A su vez, realizan un extenso análisis sobre el anonimato y la privacidad de los protocolos CryptoNote, Zerocash y Zerocoin. Comparan estudios publicados sobre la utilización de pruebas de cero conocimiento para anonimizar información dentro de la blockchain y comparan los resultados y finalidades de cada uno. También se enfocan en relevar los métodos de almacenamiento off-chain,

su rendimiento. Concluyen en aportar 4 criterios para diseñar una mejora sobre el anonimato y la privacidad de Bitcoin o sistemas similares.

Bolt: Anonymous Payment Channels for Decentralized Currencies

Ian Miers y Matthew Green fueron parte del equipo que creó Zcash. En este trabajo [27] presentan una tecnología de canales de pago para criptomonedas llamado BOLT (Blind Off-chain Lightweight Transactions). El cual combina firmas ciegas (blind signaturas) y pruebas de cero-conocimiento para establecer un canal de pago entre 2 parte de forma tal que aún cuando se opere con contrapartes maliciosas, ninguna de las dos pueda aprender nada sobre el comportamiento del otro fuera de lo relacionado con el canal de pago que comparten. Se establece que estos canales puedan ser unidireccionales como bidireccionales y a su vez involucrar a terceras partes. Los autores especifican se requiere que la criptomoneda que implemente estos canales, debe contar con la capacidad de arbitrar depósitos (escrows). Se define que un canal involucra dos ó tres participantes "desconfiados", donde una de las partes enviará uno o más pagos a otra contraparte pudiendo incluir una tercera parte como intermediaria. Una de las partes sera la depositante y la otra la receptora de los fondos. La primera debe realizar un deposito de los fondos que según el protocolo establecido, generarán tokens que consistirán en las transferencias de valor a la parte receptora. La parte pagadora deberá realizar una prueba de cero-conocimiento que permita a las demás verificar la autenticidad de los fondos dispuestos para el canal de pagos abierto. Estos canales de pago permitirían el intercambio de valor por fuera de la cadena de bloques de una forma más ágil sin resignar confiabilidad ni agregar riesgos de ataques de doble gasto y se proponen como una forma de escalar la capacidad de las cadenas de bloques. Los autores señalan que para el caso de Bitcoin y sus derivados, se requiere una extensión del lenguaje de scripting.

CryptoAR Wallet: A Blockchain Cryptocurrency Wallet Application That Uses Augmented Reality for On-Chain User Data Display

El trabajo versa sobre una wallet llamada CryptoAR [28]. Explora el uso de Realidad Aumentada (AR) en wallets de dispositivos móviles con el objetivo de reducir los puntos álgidos encontrados por diversos usuarios. Utilizaron una wallet de referencia sin AR, y con ella entrevistaron cinco usuarios con poca experiencia en criptomonedas que sí habían reportado haber operado en blockchains anteriormente, con un rango etario de entre 20 y 29 años. A todos ellos le realizaron 3 preguntas sobre los problemas encontraban en la wallet. Ese estudio delineó ciertos aspectos que los usuarios identificaban como problemáticos como el uso de terminología que les era ajena, interfaces demasiado abstractas y falta de instrucciones de uso y comunicación del progreso de las operaciones que les dieran certezas de lo que debían realizar para transaccionar y para saber que la acción realizada había funcionado o estaba en progreso. Los autores crearon un prototipo que incorporaba AR en dichos puntos de fricción con el objetivo de saber si su inclusión generaría una mejora en la experiencia de usuario. En el paper declaran que esto se verifica pero no hay más detalles al respecto.

Mind Your Wallet's Privacy: Identifying Bitcoin Wallet Apps and User's Actions through Network Traffic Analysis

Los autores realizan un estudio sobre la privacidad de las 10 wallets más populares del App Store de Apple y Play Store de Google [29]. Para ellos crean un modelo de machine learning sobre el tráfico que las aplicaciones generan en las distintas pantallas y acciones principales como iniciar, recibir Bitcoin o altcoins, historial de transacciones y enviar Bitcoin o altcoins. Entrenaron un modelo de Support Vector Machines y otro de Random Forest con los paquetes de red interceptados por un observador pasivo del tráfico de red de un Access Point WiFi. Presentan resultados muy buenos de clasificación del tráfico en todas las dimensiones, sistema operativo, aplicación y acción realizada con exactitudes mayores al 90 %, mostrando que wallets como Coinbase, Luno, Xamo, Mycelium, Bitcoin Wallet son susceptibles a este tipo de ataques a la privacidad de los usuarios.

Moneywork: Practices of Use and Social Interaction around Digital and Analog Money

Sus autores Mark Perry y Jennifer Ferreira realizan un estudio del 'Moneywork': el ciclo de vida que captura el trabajo que los usuarios ponen a las practicas que implican el manejo del dinero alrededor de pagos a terceros y como los sistemas existentes le dan forma a esas interacciones, desde su inicio hasta su conclusión [30]. Proveen un marco de trabajo donde describen el trabajo que las personas hacen para llevar a cabo una transacción, desde lo social y lo económico y discuten como esos patrones de comportamiento se encuadran en un ecosistema más amplio donde actividades digitales y analógicas juegan un rol. Se concentran en el caso de la Libra de Bristol (Bristol Pound, ahora conocido como Bristol Pay), la moneda local más grande del Reino Unido, creada como un medio de intercambio en esa localidad como forma de incentivar el comercio de intra-zona. Estudian el método de pago entre pares utilizando la plataforma Text2Pay, donde identifican un flujo de actividades y decisiones entorno a una transacción. Identifican sus etapas: Pre-transacción, en-transacción y pos-transacción. Donde cada una de ellas tiene su flujo específico de acciones y actores definidos. Destacan que en sistemas monetarios locales quedan expuestos más notoriamente los roles sociales del dinero como medio de intercambio, donde los involucrados se ven influenciados por el nivel de beneficio o costo que les significa la utilización de un sistema de pagos que favorece la localidad frente al sistema nacional. El objetivo principal que persigue la investigación es entender cómo funciona la mecánica transaccional del uso del dinero, analizar formas de generar dinámicas que fomenten el intercambio en medios digitales. Uno de los hallazgos que destacan, es que encuentran imposible de disociar los sistemas digitales del físico para el caso de las monedas corrientes de uso legal que tienen su contrapartida digital.

On The Unforkability of Monero

Este trabajo [31] analiza los distintos chain-forks de Monero y su impacto en tanto en la cadena de bloques como en la no-trazabilidad de sus transacciones. Relevan la cantidad de inputs que son trazables dentro de los forks MoneroV y Monero6, dimensionando el problema de la reutilización de claves privadas y su impacto en la escalabilidad de esta criptomoneda. Proponen una estrategia para mitigar el problema de la reutilización de claves a través de los distintos chain-forks. En la solución

propuesta, sugieren la utilización de un identificador de cadena en las transacciones (chain_id) y de la inclusión de un parámetro llamado fork_point para indicar el punto en el cual un fork a ocurrido en la cadena con el propósito de evitar 'replay attacks'. Proponen unificar el manejo de las claves con las cuales se firman las transacciones en el modelo de anillo (ring signature) dado que la existencia de distintas cadenas con claves mellizas, y distintos parámetros del protocolo (cantidad de claves del anillo, intervalos de bloque, etc) permiten la desanonimización de las transacciones. Por último plantean la necesidad de incluir Filtros de Bloom escalables (scalable bloom filters) para la proveer un mejor mecanismo de selección de las claves existentes permitiendo mayor anonimato para cuando las mismas claves gastan las mismas monedas en distintos forks.

On the Privacy Provisions of Bloom Filters in Lightweight Bitcoin Clients

Trata el tema de la privacidad de los filtros Bloom utilizados por los clientes Simple Payment Verification (SPV) de Bitcoin [32]. Estos filtros se utilizan con el objetivo de mejorar el conjunto de anonimato de los clientes livianos al crear estos filtros que realizan consultas al nodo de Bitcoin que resultan en información adicional a la que se requiere específicamente con el objetivo engañar o enmascarar el requerimiento original de ese usuario. El trabajo propone que las ventajas enunciadas para los filtros de Bloom no son tales, sino que por el contrario su uso podría ir además en detrimento de la privacidad de sus usuarios, ya que servirían de vehículo para revelar más información de la que en realidad se enmascara o anonimiza. Muestran como un solo filtro de Bloom puede delatar todas las direcciones de un cliente SPV con un numero menor a 20 direcciones, que adversarios pueden analizar estos filtros y asociarlos a sus originantes con gran exactitud y que recolectados al menos un par de éstos, el adversario puede conseguir un numero considerable de direcciones pertenecientes a un solo cliente SPV. Proponen medidas para contrarrestar estos efectos que, según los autores requieren mínimas modificaciones a los clientes SPV. En su análisis consideran que los clientes SPV se conectan a cualquier nodo de la red p2p de Bitcoin y que estos últimos desconocen la dirección IP de los clientes. En cuanto al 'atacante' consideran que conocen los parámetros de creación de los filtros de Bloom que los clientes generan y disponen de una copia de toda la blockchain (que es pública) y que además pueden recolectar metadatos de los clientes SPV que se conectan a un nodo dado. Establecen una métrica de privacidad de los filtros de Bloom en base a la probabilidad de que un adversario adivine alguna dirección que no sea ya conocida por el adversario. Mientras más alta es dicha probabilidad, menos efectivo es el filtro en cuestión. En el trabajo experimental realizado detectan que aquellos clientes que recalculan sus filtros de Bloom periódicamente filtran más información y elevan la probabilidad de que el adversario adivinen las direcciones pertinentes a ese cliente y que además la efectividad de esos disminuye considerablemente cuando el cliente SPV dispone de menos de 20 direcciones derivadas de las claves privadas, y que para el caso de que se detecten múltiples filtros de un mismo cliente la privacidad es aún menor, pudiendo inferir además si este ha reiniciado estos filtros adquiriendo más información sobre el cliente utilizado. Observan que para mitigar estos efectos, los clientes deben maximizar el numero de inputs del filtro, reducir la generación de los mismos por cliente y almacenar los ya generados para evitar volver a computarlos con los mismos inputs (direcciones) y diferentes parámetros, no ingresar tanto la clave pública como el hash siendo que con este último es suficiente para obtener todas las transacciones relevantes. La solución

propuesta contempla estas observaciones e involucra solamente cambiar la forma en que los clientes SPV administran y utilizan estos filtros de manera tal que se minimice la información que se filtra a posible adversarios.

Pitfalls of Open Architecture: How Friends Can Exploit Your Cryptocurrency Wallet

Los autores de [33] exploran vulnerabilidades de wallets que utilizan Remote Procedure Calls (RPC) en computadoras personales con múltiples usuarios. Encuentran que existen formas de atacar una wallet desde una misma computadora utilizando otro usuario autenticado legítimamente en ese equipo. Crean 2 tipos de ataques verificados en wallets de Ethereum, Bitcoin, Dash y Monero. Los ataques descriptos son: *Server impersonation*, donde el usuario malicioso logra pasar como el proceso al que la wallet le envía las llamadas RPC; *Client Impersonation*, donde el usuario malicioso logra presentarse como la wallet a la cual el servidor responde llamadas RPC. Los autores definen mecanismos de defensa que suponen adecuados para esta vulnerabilidad y establecen una política de divulgación responsable a las partes afectadas.

Privacy and security analysis of cryptocurrency mobile applications

Esta publicación [34] apunta a realizar un análisis de diferentes wallets de criptomonedas entorno al top 10 de vulnerabilidades de OWASP[2], utilizando como base tests que se realizan en aplicaciones bancarias y bursátiles. Buscan responder las siguientes preguntas: ¿Son las aplicaciones de criptomonedas más prominentes para Android vulnerables a amenazas de seguridad y privacidad comunes? ¿Son estas aplicaciones menos seguras que otras aplicaciones de finanzas convencionales?. Analizan aplicaciones del Play Store de Google (Android) y las dividen en cuatro categorías dependiendo de la cantidad de descargas. De cada categoría seleccionan alrededor de 16 aplicaciones, que analizarán con una herramienta (DroidSafe) y luego mediante un proceso manual. Los resultados concluyen que no hay diferencias substanciales entre la seguridad provista por los dos grupos de aplicaciones en estudio.

Private and Secure Mixing in Credit Networks

Los autores de este trabajo [35] desarrollan la problemática de la ausencia de mecanismos de privacidad en las redes de crédito *peer-to-peer*. A diferencia del campo de las criptomonedas, no ha habido desarrollos en esta materia en el sector de créditos de las finanzas descentralizadas. Las redes de mixing se utilizan extensamente para proveer cierta privacidad en las transacciones de una blockchain, existen varias como Tumbler, Mixcoin o CoinJoin, las cuales tienen todas sus ventajas y desventajas pero no se aplican a redes de crédito. Los autores mencionan PathShuffle como una herramienta de mixing para la red de crédito p2p Ripple, aunque destacan que varias de sus desventajas a la hora de anonimizar una transacción, como el bajo numero del mixers promedio, es así que proponen un mecanismo de mixing que permite a un usuario realizar múltiples transacciones simultáneas preservando su anonimato mediante el uso de ring signatures y zero knowledge proofs. La solución propuesta por los autores presenta ciertas ventajas por sobre las existentes, donde permiten que haya una disponibilidad mayor de intermediarios para el mixing de la transacción, no requieren un tiempo de espera para unirse a la red ni exigen a los usuarios que transmitan valores

[2] `https://owasp.org/`

aleatorios en sus transacciones, aumentando la tasa de éxito en las mismas. El sistema propuesto esta compuesto por una serie de intermediarios que integran un anillo en el cual firman las transacciones en un esquema de ring signatures. Cuando un usuario A le envía fondos a B, comparten un commitment fuera de la red. Este commitment es el que usará B para reclamar los fondos a los intermediarios del anillo mediante una prueba de cero conocimiento. Realizaron una evaluación de los tiempos de respuesta, el costo de despliegue y el tiempo de procesamiento de una transacción pero no tienen un estudio comparativo con las soluciones existentes que relevan en la introducción.

Security and privacy of mobile wallet users in Bitcoin, Dash, Monero, and Zcash

Biryukov y Tikhomirov analizan en [36] la privacidad de billeteras de Bitcoin, Dash, Monero y Zcash que encuentran promocionadas en los sitios oficiales de las criptomonedas. Se enfocan en la cantidad de información que los desarrolladores pueden recolectar de sus usuarios y establecen los siguientes criterios mínimos de privacidad y seguridad de wallets: (1) no requiere registro; (2) El código fuente se encuentra abierto y al público; (3) las claves privadas son generadas localmente; (4) las transacciones son realizadas a través de una red p2p y no mediante servidores "confiables". Al momento de la publicación (2019) encuentran aplicaciones para Bitcoin, Dash y Monero pero no de Zcash. Tampoco hallan billeteras Multi-monedas que cumplan estos 4 puntos. [3] Analizan los permisos que cada aplicación requiere a los usuarios (son todas aplicaciones Android) donde encuentran la solicitud de algunos permisos que rompen el pseudoanonimato de los usuarios como por ejemplo, la utilización de servicios centralizados de exchanges o el registro de direcciones IP totales o parciales. Utilizan herramientas de análisis estático para indagar en distintas propiedades de las aplicaciones seleccionadas como el flujo de datos desde y hacia las aplicaciones, donde detectan posibilidad de filtraciones a almacenamiento externo a las aplicaciones dentro del dispositivo, ataques de *cross-site scripting* para aquellas aplicaciones que utilizan WebViews e incluso conexiones inseguras. Realizan una matriz comparativa con estos aspectos para todas las wallets analizadas. En cuanto a privacidad de uso de la red, realizan un estudio donde crear un bucle dentro de la red p2p para realizar una técnica llamada *p2p gossip*, mediante la cual pueden obtener información para agrupar transacciones pertinentes a un grupo de billeteras de interés. Para medir el grado de desanonización de cada wallet estudiada utilizan una técnica llamada "grado de anonimato", con la cual se puede comparar el grado de información obtenida por un atacante en contraste con el completo anonimato. Se detecta una apreciación poco exacta respecto de las propiedades de anonimato de lo que el paper considera como privacy coins [4], donde se analizan wallets que dan soporte a transacciones Transparentes de Zcash, las cuales no tienen diferencia alguna respecto de Bitcoin Core 0.14. No se encuentra un análisis profundo respecto de Monero o Zcash blindado (Sapling o Sprout). Este último posiblemente atribuible a que no existían billeteras móviles para Android que soportasen transacciones Blindadas de Zcash.

[3] Al momento de la redacción de este relevamiento, se encuentran billeteras de Zcash que cumplen los primeros 3 puntos, y parcialmente con el cuarto ya que en pie de página indican que consideran que aquellas wallets que permitan al usuario seleccionar distintos servidores "confiables" o uno propio, lo cumplirían

[4] Dash ha salido de la narrativa de ser una privacy coin

Security of Cryptocurrency Using Hardware Wallet and QR Code

Este trabajo [37] discute aspectos de seguridad relacionadas al manejo de las claves privadas de una wallet en tres categorías: Software, Hardware y Papel. Para ellos conducen una revisión sistemática de la literatura en cuanto a trabajos previos en esta material y un recorrido introductorio sobre los vectores de ataque más comunes (double gasto / ataque del 51 %, fuerza bruta y Ataque Finney). Los autores proponen un esquema de firmado y validación de transacciones mediante dos dispositivos Android. Uno que actúa de 'Hot Wallet' conectado a la red p2p de Bitcoin provisto con claves públicas y otro que contiene las claves privadas desconectado de internet oficiando de 'Cold Wallet'.

Toward Scalable Blockchains with Transaction Aggregation

El paper [38] plantea la agregación de transacciones para contribuir a la escalabilidad de las cadenas de bloques. El trabajo aporta un diseño de cadena de bloques doblemente enlazada donde los bloques no solo tienen el vinculo con su predecesor, sino que además el de su sucesor. Los mineros pueden agregar las transacciones ya existentes en otros bloques, aumentando la capacidad de la red en términos de transacciones por segundo. El trabajo esta pensado para redes Proof-of-Stake en las cuales hay muchas transacciones de un conjunto de usuarios, como ocurre en IoT-Blockchain. Caso contrario no presentan mejoras.

Understanding Ethereum via Graph Analysis

El artículo [39] explora el funcionamiento del ecosistema Ethereum mediante el análisis de grafos de las operaciones realizadas. Construyen tres grafos: 1) flujo de dinero (MFG, money flow graph); 2) creación de contratos (CCG, contract creation graph); 3) invocación de contratos (CIG). Para ello recolectaron toda la información de la blockchain hasta ese momento y reprodujeron las transacciones en una EVM local modificada a tales efectos, dado que las transacciones de Ethereum representan ejecuciones de contratos inteligentes consumadas, pero los autores requieren de información adicional derivada de los flujos de ejecución de los contratos para su análisis. mediante el análisis de los grafos creados no solo logran capturar la creación anormal de contratos por parte de diversas cuentas de usuario, lo cual contribuye a un mejor entendimiento del fenómeno. También logran incrementar la trazabilidad de cuentas relacionadas a ataques informáticos mediante el análisis cruzado de los grafos CCG y CIG, pudiendo también detectar otros fenómenos como ataques de denegación de servicio.

You Don't Need a Ledger: Lightweight Decentralized Consensus Between Mobile Web Clients

Kristof, Lagaisse, Bert y Joosen plantean en [40] el uso de un middleware diseñado para proveer una funcionalidad equivalente a una blockchain sin el uso de un ledger y la infraestructura que conlleva, destinado a pequeños comercios que busquen transaccionar descentralizadamente sin la necesidad de involucrar una tercera parte. Dicho middleware (1) tolera nodos maliciosos y fallos (crashes); (2) garantiza el consenso de las acciones tomadas por las partes; (3) propaga el estado de la red en base a un protocolo de replicación de actualizaciones y votos a través de una red p2p y (4) esta

diseñado para entornos de bajos recursos con un solo componente de backend utilizado para el descubrimiento de pares. Los casos de uso planteados para este software son un programa de Lealtad comercial (eLoyalty) para clientes de pequeños comercios donde se reemplaza la corporación que provee el servicio de puntos y descuentos y recompensas a clientes por este middleware y los distintos nodos (comercios) de la red; y una red de préstamos (eLoans) donde se colateralizan las facturas a cobrar de los comercios de la red como garantía de los prestamos a otorgar, utilizando el middleware como una forma de evitar el doble gasto o falsificaciones. El sistema propuesto es que la red conformada por los pares conforme un sistema de votación constante donde las acciones propuestas por los nodos son sometidas a decisión por el consenso de las dos terceras partes de la red más un voto. las acciones se cotejan contra los datos locales de los clientes livianos y se aprueban o deniegan en base al estado que contienen éstos, que es constantemente replicado a través de la red p2p. El protocolo fue puesto a prueba mediante una serie de servidores virtuales, midiendo el rendimiento de la sincronización de los nodos a medida que se agregan más pares a la red observándose que pasados los 50 nodos comienza a crecer linealmente el tiempo de confirmación y sincronización de la red p2p alcanzando tiempos similares al tiempo por bloque de redes algunas PoW (30 segundos por confirmación). El beneficio de no contar con un ledger común resulta en que los recursos necesarios para ejecutar el consenso de la red son bajos, a expensas de la capacidad de auditar la información. Las mejoras en el *throughput* del sistema se plantean como trabajo a futuro .

3.5. Conclusiones

3.5.1. Respuestas a las preguntas planteadas:

¿Qué diferencias de requerimientos funcionales y no funcionales existen entre wallets que soportan Privacy Coins en Desktop, Mobile y Web?
Si bien no se encontraron trabajos que refieran específicamente a este tema, se encuentran algunos puntos importantes a considerar. El trabajo [40] hace referencia a la necesidad de poder contar con rápida sincronización y la reducción de la dependencia del consenso de red respecto del ledger de la blockchain, mientras que [37] presenta un trabajo donde permite utilizar un segundo dispositivo como "cold wallet" delegando así el poder de gasto en un artefacto que no esté conectado a la red. Existen cuestiones referentes a las falta de privacidad del enfoque SPV de Bitcoin o toda tecnología que implique la utilización de filtros de Bloom [32]. El estudio [30] porta ciertas visiones sobre el "Moneywork" de las monedas digitales, su conclusión es interesante en términos de usabilidad donde observa que no hay disociación entre el uso de una moneda física de curso legal y su versión digital. El articulo sobre BOLT [27] refiere a la necesidad de poder establecer canales seguros en una capa superior a la cadena de bloques para poder acelerar los tiempos que lleva poder realizar una transacción entre pares de forma anónima y segura para ambas partes. El modelo de clientes livianos a su vez tiene distintos requerimientos no funcionales en base a las deficiencias de seguridad y privacidad de red que presenta la dependencia en servidores intermediarios por no disponer de una copia completa de la cadena de bloques de la que sí disponen las billeteras full node.

¿Qué experiencias hay del cómputo de transacciones del lado del cliente?
El único trabajo relevado que hace referencia explícita a esta cuestión es [37]. Adicio-

nalmente puede concluirse en base a lo expuesto en el trabajo [40] y [27] la necesidad de poder computar y realizar transacciones entre pares de forma rápida y descentralizada.

¿Qué esquemas de respaldo de cadena de bloques se pueden utilizar? no se encontraron detalles sobre el soporte utilizado en los trabajos que refieren a clientes livianos específicamente. El trabajo [40] plantea la eliminación del ledger público y la utilización de un modelo de asiento contable efímero.

3.5.2. Conclusiones generales de la SLR

Durante la realización de este capítulo se encontraron varios aspectos que no refieren tanto al estado del arte de la materia en cuestión en si mismo sino que revelan el una la situación general del espacio en cuanto a los medios que quienes trabajan en él eligen para difundir su obra y avances. A continuación se detallan las conclusiones del caso:

- Los artículos sobre la cuestión no se encuentran en los repositorios seleccionados pese que éstos son los que la bibliografía respecto de la metodología aconseja.
- Existe abundante cantidad de "pre-prints" y artículos no publicados en congresos formales que tienen mucha relevancia en la práctica y se encuentran en otros lugares como IACR o incluso en repositorios de código abierto o sitios propios de los autores u organizaciones involucradas.
- Se encuentra gran cantidad de literatura gris que resulta relevante para la materia en cuestión mientras que la cantidad de literatura encontrada siguiendo la literatura SLR es poco relevante y sujeta a ser excluida en base a los criterios planteados.
- Se encontró que a los efectos de poder contar con un espectro más amplio de la literatura a censar y evaluar, los revisores debieron proponer nuevos criterios a los originalmente planteados ya que se revelaban demasiado específicos y excluían a casi la totalidad de los resultados.
- Los resultados, comentarios y reflexiones aquí vertidos son producto de la reevaluación de los criterios originales que se mantuvieron por su valor descriptivo de la situación actual.
- No obstante la revisión resultante refleja el estado de la materia en tanto refiere y considere a las metodologías de revisión que son actualmente consideradas como buenas prácticas en materia de SLR.
- A los números que confrontan la cantidad de inclusiones y exclusiones que resultan de esta revisión sistemática de la literatura se concluye que es necesario adaptar la metodología vigente para el caso de la investigación actual de forma tal que pueda incluirse toda la literatura gris o no publicada en canales tradicionales como los utilizados en la búsqueda (IEEE, SCOPUS, ACM, etc.) de manera ordenada, criteriosa, objetiva, reproducible y transparente tal como lo demanda la metodología de investigación.

3.6. Trabajo a futuro

Como hipótesis principal para un trabajo a futuro se divisa que el carácter descentralizado, anti-institucional y de muy rápido avance técnico y tecnológico contribuyen a la no proliferación de la literatura pertinente en canales de difusión del ámbito

académico formal. Por tanto es necesario desarrollar un marco teórico-práctico que contemple esta situación. Una SLR, para ser considerada exhaustiva, debería incluir toda la literatura disponible en su área de investigación contemplando las distintas fuentes disponibles y proveyendo herramientas metodológicas para sistematizar la inclusión de cada tipo de fuente, logrando mantener el carácter formal y sistemático de una SLR, sin que esto signifique dejar de considerar fuentes informativas solo por el hecho de no estar incluidas en un ámbito formal y/o académico.

Capítulo 4

Relevamiento de Requerimientos de Wallets para Zcash y Monero

Para poder establecer una arquitectura de referencia es necesario conocer el estado del arte de las wallets existentes. Para ello se hizo un análisis de las distintas billeteras electrónicas de las AECs Zcash y Monero, realizando una comparación en términos de requerimientos funcionales, no funcionales y arquitectura.

Se relevaron las aplicaciones, el código fuente y la documentación de los repositorios aquí descritos.

- ZecWallet Lite [41]
- Cake Wallet [42]
- Monerujo [43]
- ECC Wallet [44][45]
- Unstoppable Wallet [46][47]
- SDK de Zcash (Android y iOS) [48][49]

En base a ello, se confeccionó una tabla comparativa donde se indica la presencia o no de cada una de las historias de usuario en las aplicaciones relevadas (ver anexo 7.1). En caso de que la historia de usuario no se encuentra desarrollada en una o más aplicaciones, se verificó si se encuentra pendiente en las 'issues' del repositorio de código abierto del proyecto en cuestión.

Para cada uno de los proyectos se resumirá:

- Descripción general del proyecto, cantidad de contribuyentes, licencia Open Source
- Los lenguajes de programación, frameworks y el tipo de stack tecnológico que utiliza, los sistemas operativos que soporta
- Si se verifica o no la utilización de una arquitectura específica (MVC, MVVM, MVP, VIPER, Clean Code, SOLID, etc), o una variante similar de las mismas.

- Si implementa o utiliza kits de desarrollo para operar con el protocolo de la criptomoneda (Zcash o Monero)
- Si existe funcionalidad soportada por medio de "interoperabilidad" para varias plataformas y/o sistemas operativos, describir el enfoque utilizado, lenguaje de programación (ejemplo: C, C++, Rust, etc.) y las interfaces que estos publican.

4.1. Relevamiento de Proyectos

4.1.1. ZecWallet Lite

La billetera ZecWallet Lite es un cliente liviano para Zcash, "z-address first". Lo cual significa que prioriza la utilización de direcciones *Sapling*. En su repositorio de github resalta las siguientes características:

- Enviar y recibir transacciones blindadas.
- Soporta direcciones transparente y sus transacciones.
- Soporte para memos encriptados.
- Encriptación local de claves privadas.

El stack de programación esta compuesto por un front-end *React.js* encargado de la interfaz gráfica de su formato escritorio y un *back-end* compuesto por una interfaz tipo cliente de terminal llamado *Zecwallet-lite-Cli* [50]. Esta es la encargada de interactuar con el servidor de clientes livianos *lightwalletd* y proveer toda la funcionalidad propia de la billetera como generar las claves del usuario, mostrar las direcciones, sincronizar la blockchain, enviar transacciones, etc.

La aplicación de línea de comandos *ZecWallet-Cli* esta implementada en Rust, y cuenta con dos modos, uno interactivo y otro no-interactivo. El segundo permite que aplicaciones como ZecWallet-Lite lo utilicen de "servidor local" a quien envían comandos y este responde en formato JSON. La utilización de este esquema permite que la capa Rust sea utilizada para diversas plataformas y sistemas operativos.

ZecWallet y ZecWallet-lite fueron las primeras billeteras de Zcash con interfaz gráfica. Se realizó en colaboración con la Fundación Zcash[1] mediante un otorgamiento de fondos para su financiación. Es la billetera con más requerimientos funcionales y no funcionales del protocolo Sapling implementados lo cual se refleja en la matriz de requerimientos 7.1.

4.1.2. ZecWallet Lite Mobile

La versión de cliente liviano de escritorio tiene una variante *mobile* llamada *ZecWallet Mobile* [51]. Que utiliza un front-end *React Native* para desplegar la aplicación en Android y iOS. La aplicación utiliza una interfaz de funciones foráneas (FFI) con Rust cuyo objetivo es utilizar la libreria Zec Wallet Lite CLI, como back-end. El componente CLI actúa de *"servidor"* para la aplicación *React Native*, donde esta recibe un nombre de comando y los argumentos en forma de diccionario y retorna un String en formato JSON con la respuesta, lo cual resulta conveniente siendo que para *React* este formato es un *First-class citizen*[2].

[1] http://www.zf.org/

[2] Refiere a "ciudadano de primera". Un termino figurativo que indica que una funcionalidad es clave para un framework o un lenguaje de programación determinado y que la misma es

4.1.3. Cake Wallet

Cake Wallet es una billetera cuya principal moneda es Monero, pero también ofrece la posibilidad de operar con Bitcoin. Esta desarrollada en una tecnología llamada Flutter, cuya finalidad es proveer un desarrollo similar al *Nativo* mediante la utilización de un *runtime* propio y la *renderización* de la interfaz gráfica mediante una vista propia. Las aplicaciones Flutter son desarrolladas en el lenguaje de programación Dart[3] el cual utiliza compilación «Just-in-Time» y «Ahead-of-Time» para reducir tiempos de compilación y despliegue durante el desarrollo.

La finalidad de Flutter es lograr maximizar la cantidad de código fuente compartido entre distintas plataformas: Android, Escritorio (Windows, Linux, MacOS), iOS y Web[4]. Para el caso de Cake Wallet esto se verifica para la lógica de la interfaz de usuario y parte de la lógica de negocio de la wallet, mientras que ciertas funcionalidades núcleo de criptografía y la interacción con la blockchain propiamente dicha se encuentra delegada a clases nativas en Java, Swift, Objective-C y C++ mediante un mecanismo de plug-ins (ver figura 4.1)

A su vez durante el relevamiento de repositorios se halló un fork de una versión nativa para iOS de Cake Wallet, donde utilizan UIKit y Swift para desarrollar la aplicación utilizando una variante de la arquitectura MVC propuesta por el framework de Apple, donde se verifica también la existencia de una interfaz FFI hacia C++ para interactuar con la blockchain.

La tabla en el Anexo 7.1 ilustra las historias de usuario verificadas en la versión Flutter.

4.1.4. Monerujo

Esta billetera permite utilizar Monero en dispositivos Android. Se encuentra desarrollada en Java, utilizando Android Fragments para la interfaz de usuario. Monerujo es una aplicación multi-billetera, de una una sola criptomoneda. Permite utilizar varias billeteras de Monero en la misma aplicación. A su vez utiliza un servicio de terceros llamado *sideshift.io* para pagar a direcciones de Bitcoin[5]. La aplicación ofrece a su vez la posibilidad de utilizar nodos alternativos para realizar transacciones y sincronizar la cadena de bloques.

La interfaz gráfica que expresa la lógica de negocio propia de los casos de uso de una billetera se encuentran codificados en Java, mientras que la interacción con la blockchain y la generación de transacciones se encuentra delegada a una librería C++ mediante una interfaz FFI utilizando Java Native Interface (ver figura 4.3).

---

traducida, interpretada y utilizada como si fuera una primitiva, aún cuando en términos reales, esta no lo fuera. En el caso de tecnologías como React, el formato JSON puede ser utilizado prácticamente como una extensión del lenguaje Javascript, pudiendo los programadores referir a esta construcción sin necesidad de reparar en serializaciones o deserializaciones.

[3] https://dart.dev/overview#platform

[4] Según la documentación oficial de la plataforma.

[5] Sideshift es un sitio de intercambio de criptomonedas centralizado «permissionless» en el cual se puede intercambiar un activo por otro ingresando una dirección en la moneda de destino, para el cual el sitio genera una dirección de recepción a la cual enviar los fondos a un tipo de cambio dado que informa la plataforma.

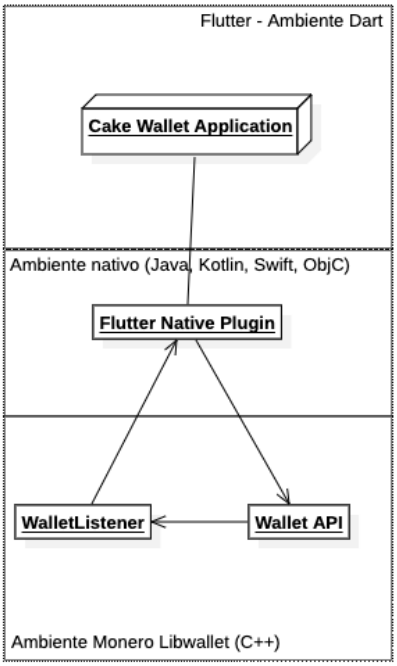


Figura 4.1: Diagrama simplificado de la implementación para Flutter Cake Wallet para iOS y Android

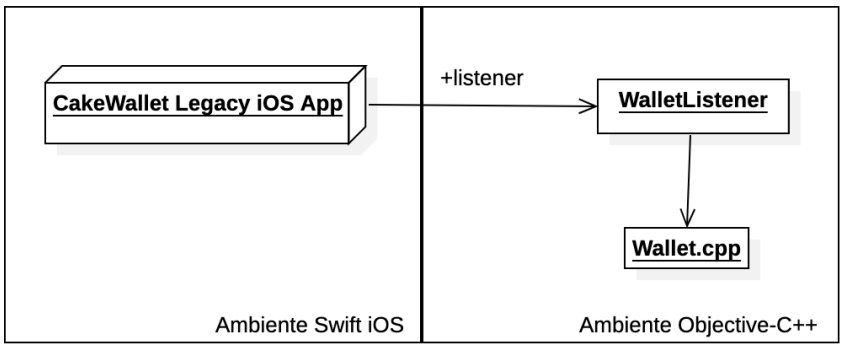


Figura 4.2: Diagrama simplificado de la implementación nativa y deprecada de Cake Wallet

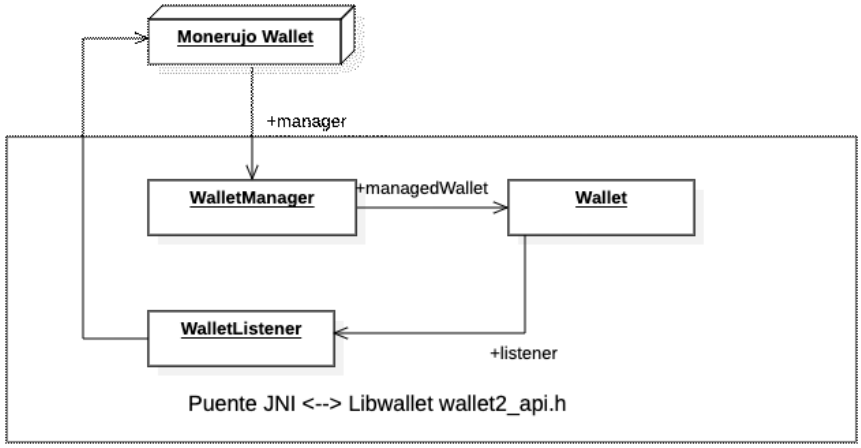


Figura 4.3: Diagrama simplificado de la implementación de Monerujo Wallet

4.1.5. ECC Wallet

La billetera ECC Wallet es una aplicación de referencia cuyo propósito es realizar *Dogfooding*[6] del protocolo de clientes livianos de Zcash. Es realizada por desarrolladores Open Source y desarrolladores de Electric Coin Company, una de las empresas detrás de la creación de Zcash.

El stack de programación está compuesto por las interfaces de usuario nativas de Android y iOS, y sus lenguajes insignia, Kotlin y Swift respectivamente. En cuanto a la interfaz de usuario la aplicación de Android [44] utiliza Android Fragments y *Coroutines* mientras que la versión de iOS [45] utiliza el modelo reactivo de *SwiftUI* y *Combine Framework*.

Ambas aplicaciones utilizan el kit de desarrollo de Zcash [49][48] para operar con el protocolo de clientes livianos de Zcash [22]. Los mismos están desarrollados en Swift y Kotlin, y su estructura se discute en la sección 4.5. Utilizan la librería *Librustzcash* en Rust como módulo de interoperabilidad de forma similar a como ZecWallet lo hace con su CLI.

Los requerimientos y las historias de usuario encontradas sus repositorios fueron utilizados como requerimientos base para la matriz de comparaciones de requerimientos. Este proyecto es uno de los casos de estudios descritos en la sección 5.5.

4.1.6. Unstoppable Wallet

Unstoppable es una wallet multi-moneda que permite a los usuarios operar con distintas criptomonedas, como Bitcoin, Bitcoin Cash, Ethereum (y distintas variantes de ERC-20) y Zcash. Esta última es la única moneda AEC que esta billetera opera. Además cuenta con una sección de trading y educación sobre criptomonedas y su uso.

[6]Refiere al uso de un producto por parte de los propios realizadores con la intención de someterlo a prueba antes de ser lanzado al público.

Unstoppable implementa sus versiones de Android y iOS por separado en sus respectivas plataformas, siendo la primera implementada en Java y RxJava, un framework de programación funcional reactiva para esta plataforma. La interfaz utiliza Android Fragments y la arquitectura que presenta el del tipo *Clean-Code*. En el caso de iOS se observa el mismo esquema donde se utiliza RxSwift.

El soporte multi-moneda esta soportado por abstracciones sobre las representaciones comunes de los requerimientos que todas las criptomonedas comparten y utilizando extensivamente el patrón *Adapter*[7] para manejar las particularidades de la implementación de cada una. El equipo de desarrollo de esta wallet ha desarrollado varios SDK para distintas criptomonedas, los cuales utiliza en esta wallet, logrando así una buena modularización del código fuente y de la aplicación. En la sección 5.5 se describe más extensivamente la arquitectura que refiere a la implementación de Zcash y las particularidades que presenta a nivel de arquitectura y de implementación.

[7] ver libro Design Patterns [52]

4.2. Temas

Del análisis de distintas wallet *non custodian* [8] se identificaron varios temas que refieren al uso de una wallet de cualquier criptomoneda.

La definición de requerimiento empleada es la definida en estándar IEEE 610 [53]:

> (1) Una condición o capacidad necesaria para un usuario con el fin de solucionar un problema dado o cumplir un objetivo
>
> (2) Una condición o capacidad que debe tener o cumplir un sistema o componente del mismo para satisfacer un contrato, estándar, especificación o cualquier otro documento impuesto formalmente
>
> (3) Una representación documental de una condición o capacidad definidas en (1) o (2)

Se definen los requerimientos funcionales como historias de usuario tal cual las define la metodología Ágil Scrum [10].

Los requerimientos no funcionales forman parte de los criterios de aceptación de las historias de usuario. Su ubicación dentro de su propio apartado no responde a una cuestión metodológica sino a una elección de redacción y formato.

4.2.1. Gestión de claves de usuario

La potestad de los fondos de una wallet corresponde a quien posea el control de las claves privadas de las cuales se derivan las direcciones en las que se van a recibir y enviar fondos.

En Bitcoin (y por ende en Zcash), las claves privadas se derivan desde una entropía de 256 bits [54]. A grandes rasgos, la tenencia de una wallet significa custodiar los bits generados desde los cuales se derivan de forma determinística en claves públicas y privadas, direcciones transparentes y blindadas (ver 2.4.1). Aquel usuario que no posea dichos bytes previamente, deberá poder generarlos y tomar conocimiento de ellos para respaldarlos. Mientras que aquellos que ya los posean, necesitarán un mecanismo para ingresar esos bits y derivar sus claves. A este último caso se lo llama restauración (restore)[9]. Cabe señalar que a fin de no requerir a todos a los usuarios que sincronicen la totalidad de la blockchain desde el bloque de génesis el último de la cadena, puede asociar una "fecha de nacimiento" o "bloque de nacimiento" de una wallet bajo la premisa de que una wallet creada en un momento dado, no registra eventos de su pertinencia en bloques previos a su creación. Para la presente temática se identifican las siguientes épicas:

- Nueva Wallet
- Respaldo de claves
- Restaurar una wallet

4.2.2. Operaciones

Se ha definido bajo este tema todo aquello que abarca la utilización de la billetera para Enviar y Recibir transacciones. Para poder operar, es necesario que la wallet se

[8] Wallets que no realizan la custodia de las claves privadas, delegándola al usuario/propietario.

[9] Por lo general los usuarios no manejan los bits en su forma binaria, sino por medio de otros formatos más amigables al ser humano como: frases semilla (seed phrases) o códigos QR

encuentre actualizada respecto de los últimos eventos de la blockchain. No es posible enviar una transacción a través de la red si el cliente no se encuentra al día. También en el acto de sincronizar, se concreta también el hecho de recibir transacciones. Se identifican las siguientes épicas:

- Recibir fondos
- Enviar fondos
- Sincronizar

4.2.3. Estado de wallet

Dada una clave privada asociada a una wallet, se entiende como estado de dicha wallet al balance consolidado producto de transacciones enviadas y recibidas a y desde las direcciones derivadas de dicha clave privada registradas desde el "bloque de nacimiento" hasta una altura de bloque determinada. Se excluye el acto de sincronizar de este tema, por estar más ligado al tema anterior, pero también sería conceptualmente válido si se considerara el caso contrario e incluir la sincronización a este tema. Se identifican las siguientes épicas:

- visualizar balance
- historial de transacciones

4.3. Épicas e Historias de usuario

Es esta sección se profundizarán las "Épicas" agrupadas por "Temas" de acuerdo a la metodología Scrum. De su estudio irán surgiendo componentes de la arquitectura propuesta.

Las historias de usuario están redactadas en términos los actores involucrados en el ecosistema no solo de quienes usan directamente la aplicación final.

Actores Involucrados

Usuario: es quien utiliza la wallet. Se presume que posee las claves privadas asociadas a la wallet que utiliza.

Ingeniero de Seguridad: parte del equipo de desarrollo de una wallet. Quien vela por la seguridad y privacidad de la aplicación y del ecosistema en el que está inserta.

La constitución de una billetera non-custodian de criptomonedas, puede reducirse al hecho de generar las claves públicas y privadas que la constituyen e identifican unívocamente. El proceso en términos criptográficos es un tema de investigación en sí mismo y que excede el alcance de este trabajo. Por tanto nos enfocaremos en el mecanismo que se ha vuelto el estándar de hecho para la generación, resguardo y restauración de Wallets de criptomonedas titulado "Código Mnemónico para la generación de claves" [21] [10] que surge como una propuesta de mejora para Bitcoin (BIP). Cabe resaltar que en Monero se indica la existencia de una convención distinta [20] en términos de la cantidad de palabras utilizadas y en la derivación. No obstante, en cuanto al manejo de las mismas, el concepto en sí mismo no cambia. Se describe a continuación el estándar más utilizado, dejando al lector la posibilidad de profundizar las particularidades de cada criptomoneda que se aparte del mencionado estándar.

4.3.1. Generación de Claves con el estándar BIP-39

Para generar las claves privadas de una Wallet, es necesario contar con una *semilla* binaria. La naturaleza binaria y/o hexadecimal de esta semilla presenta un desafío de usabilidad para las personas que, aún si estas fueran proficientes en este tipo de numeración, la posibilidad de cometer errores es al guardarlas o utilizarlas es mayor que en otro tipo de formatos. Esta propuesta surge de la necesidad de dar respuesta a dicha dificultad. Cabe comentar que en términos de uso y adopción de billeteras electrónicas sin custodia, las frases semilla en formato mnemónico, sigue significando un punto de fricción para los usuarios, y pese a que su parece establecido, sigue siendo sujeto de investigación en materia de experiencia de usuario.

Se define un código o frase Mnemónica a un mecanismo para transpolar la entropía generada por una computadora a un formato posible de ser leído y transcrito por un humano. Las frases de recuperación son utilizadas ampliamente y es posible que lector haya interactuado con ellas aún desconociendo al detalle su utilización en el ámbito de las criptomonedas. En la figura 4.4 se presenta un ejemplo de una frase mnemónica generada aleatoriamente y de acuerdo BIP-39. Es posible derivar todo tipo de direcciones para las distintas criptomonedas. Se desaconseja a los lectores curiosos hacer uso de las frases semilla utilizadas de ejemplo en este trabajo.

[10] título original en inglés: "Mnemonic code for generating deterministic keys"

vague idea lunch fun ancient stable siege purpose drift humble pen middle office monitor people leader path refuse pool devote fever cattle aim ocean

Figura 4.4: Frase semilla mnemónica de 24 palabras

b2ca5069c90a10c82de2701ff93ea34c87d0e4dccafb
40b7bc57e7d5e48304dc646c47655473cced91a5260d
aad1d6e69784c30ae1e39402b74dc11a2414aab8

Figura 4.5: Bytes semillas en base la frase anterior

La semilla binaria resultante de la frase mnemónica se observa en la figura 4.5

El mismo se obtiene en base a un diccionario de 2048 palabras preestablecido en este caso en idioma inglés con las siguientes características:

1. al escribir las primeras cuatro letras se puede identificar unívocamente la palabra en cuestión
2. se evitan palabras similares que puedan introducir errores al ingresarlas o memorizarlas
3. el listado de palabras debe estar ordenado para una búsqueda más eficiente
4. sus caracteres deben estar codificados en UTF-8

Es importante resaltar que es posible comprobar la integridad de una frase mnemónica solo contando con el diccionario utilizado para crearla. El *checksum* de la frase es parte de la misma. Lo cual garantiza que los errores ya sean forzados o involuntarios puedan validarse, pudiendo así evitar pérdida de fondos por el ingreso o respaldo de una frase errónea y/o inconsistente.

Es fundamental contar con una interfaz que enmascare la implementación de esta funcionalidad.

Los detalles de implementación pueden observarse en la librería *MnemonicSwift* [55]. Esta interfaz sera referida nuevamente y explicada en la sección 5.4.2. Los errores estipulados refieren a un *checksum* inválido, una frase generada desde un vocabulario no perteneciente o un numero incorrecto de palabras. La arquitectura de referencia utiliza esta abstracción para manejar las frases mnemónicas, cuyo diagrama se ilustra en la figura 4.6.

4.3.2. E-01 - Crear una nueva wallet

User Story:

Como Usuario quisiera crear una nueva wallet para poder utilizar Zcash

Crear una nueva wallet, es la acción inicial para cualquier usuario nuevo en una billetera. El procedimiento consiste en generar la entropía y los bytes "semilla" que darán origen a las diferentes claves del usuario. Desde su punto de vista, no implica más que accionar un control en la interaz gráfica. Se observan algunos requerimientos no funcionales involucrados: poder generar los bytes semilla una frase acorde a BIP-39 (ver R-12 y R-13 de 7.4), derivar una Sapling Spending Key de los bytes generados a partir de la semilla BIP-39 acorde al protocolo Zcash [56] y al ZIP-32 [57] (R-15)[11].

[11] ver detalle de los requerimientos en el Anexo.

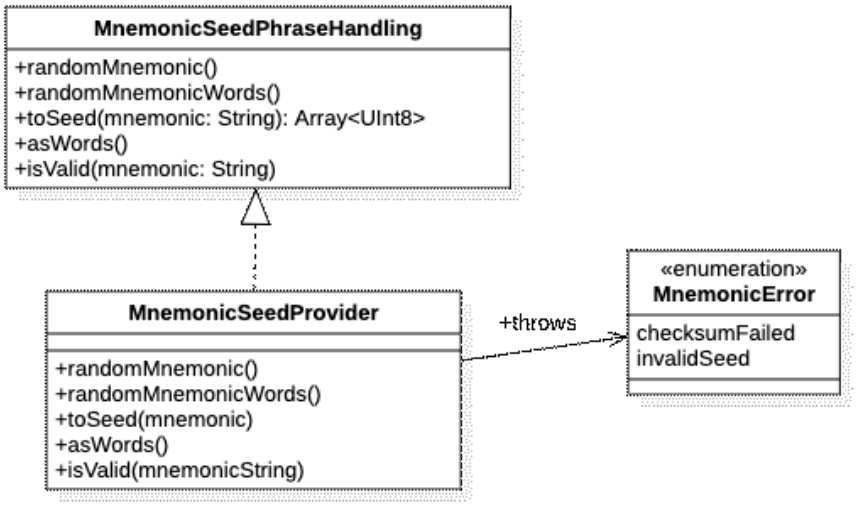


Figura 4.6: Diagrama de la interfaz para el manejo de frases mnemónicas

La creación de una nueva wallet trae aparejada la creación de una *Sapling Spending Key*. El protocolo Zcash especifica distintos tipos de claves con diferentes capacidades y propósitos. La clave fundamental y generadora de las demás es la llamada *Spending Key*, que como su nombre indica permite gastar los fondos de una determinada wallet. Puede entenderse que poseer este tipo de clave es casi equivalente a conocer una frase semilla, a excepción de que no es posible conocer la frase semilla desde la Spending Key. La figura 4.7 se extrae del protocolo Zcash [56] y aunque es críptica en su naturaleza, ilustra cómo se originan las distintas claves a partir de una Spending Key. Lo primero que se debe notar de la figura es el flujo unidireccional (de carácter asimétrico) de las derivaciones. Desde la base hacia arriba de la figura se producen claves con distintos propósitos y usos del protocolo, donde no existe un camino a la inversa. Lo segundo es la estratificación que generan estas transformaciones. Podemos entender esos estratos como roles y permisos dentro de una jerarquía. En la base de la figura podemos observar la Spending Key (sk) que es la clave que permite la "autoridad de gasto" (spending authority) de los fondos asociados a una dirección. De la derivación de la *sk* se obtienen: una clave de gasto extendida conformada por una clave que permite ver pagos salientes (outgoing viewing key u ovk), una clave que autoriza gastos (Spend authorizing key o ask) y una clave privada que anuladora (nullifier private key o nsk). En adelante se observan una derivación que concierne a las pruebas de cero conocimiento y los tres estratos superiores, aunque el protocolo Zcash considera que no deberían exponerse a los usuarios, contienen partes fundamentales que hacen a la experiencia de usuario. Una Full Viewing Key es una clave de visualización completa de las transacciones blindadas asociadas a un usuario ya sean entrantes o salientes. La *ivk* (Incoming Viewing Key) permite desencriptar aquellas transacciones que un usuario recibe, y de ella, junto con el diversificador, se producen las distintas direcciones blindadas (shielded payment address) en las cuales podrá recibir fondos.

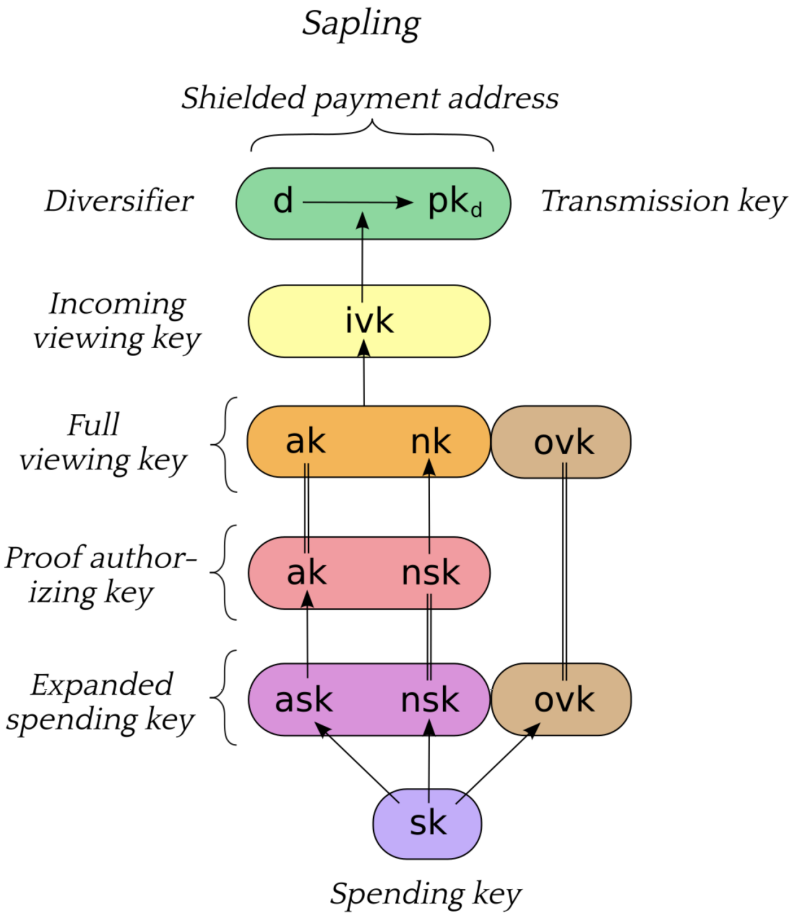


Figura 4.7: Jerarquía de derivación de Claves Sapling en Zcash

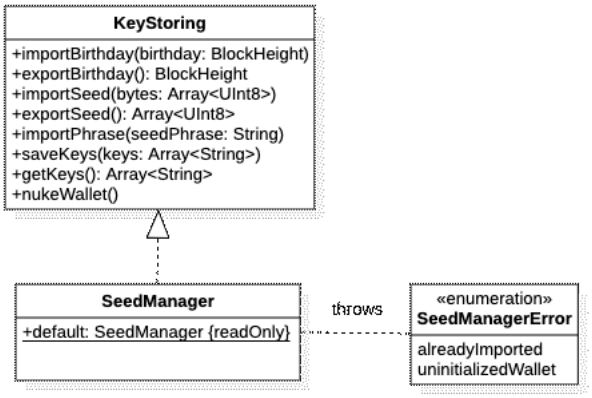


Figura 4.8: Interfaz e implementación de Seed Manager

De forma similar Monero utiliza frases semilla para generar y recuperar claves. De ella se derivan una clave pública y una clave privada. De la clave pública se obtienen las direcciones asociadas con la wallet representada por la frase semilla, mientras que la clave privada es la que tiene autoridad de gasto.

La frase semilla y las claves derivadas deben ser almacenadas de forma segura en el dispositivo (R-20). De este requerimiento surge el componente de administración de claves que debe implementar el almacenamiento seguro (KeyChain en iOS o Secure Storage en Android) tal como la describe la figura 4.8.

Este caso se comienza a delinear una frontera clara entre lo relacionado al desarrollo propio de una aplicación móvil, como la necesidad de almacenar un dato de forma segura de acuerdo a la autenticación de usuario manejada por el sistema operativo en cuestión, y las complejidades adicionales que corresponden al dominio de las criptomonedas.

En la figura 4.9 se introduce como dependencia un kit de desarrollo de Zcash en el que se exponen las funcionalidades de interés para clientes livianos y se abstraen las complejidades propias del dominio de una criptomoneda de la aplicación. Las historias de usuario que comprenden el desarrollo y la funcionalidad del SDK se encuentran en el anexo 7.3.

Si bien los detalles que hacen a la sincronización se verán en la sección 4.3.9, es necesario introducir dos componentes importantes del kit de desarrollo de Zcash. Ellos ofician como fachada en esta "frontera" que se ha comenzado a delimitar entre el dominio de una aplicación cliente y el dominio de la privacy coin.

Para el caso de iniciar una nueva wallet, se identifica un primer componente de esta fachada: El "inicializador". El rol de esta clase es compartimentar la complejidad de poner en marcha todas las partes en movimiento que se requieren para operan con la cadena de bloques. Para este escenario, derivar una *Spending Key*, es una tarea compleja, cuya implementación requiere pleno conocimiento del protocolo y de todos sus fundamentos. Una de las principales decisiones rectoras de esta arquitectura es buscar centralizar todo lo referido a criptografía en una implementación que pueda utilizarse desde múltiples plataformas. Es el caso del proyecto *Librustzcash* [58], implementa

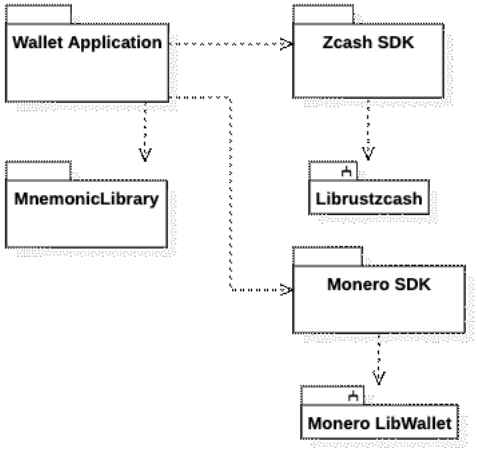


Figura 4.9: Vista general - Dependencias clave: Kit de desarrollo de Zcash, Monero y una librería de frases Mnemónicas

funcionalidades núcleo del protocolo Zcash en Rust, siendo posible utilizar esta dependencia tanto desde iOS como Android. La figura 4.10 muestra el diagrama de la clase *Initializer*. A través suyo se centralizan los accesos a librustzcash mediante la interfaz interna *ZcashRustBackendWelding*. Esta última es implementada por componentes nativos del lenguaje de programación de la plataforma correspondiente: Kotlin + JNI en Android y Swift en iOS. Contar con una interfaz permite la inyección de clases de testeo (Mocks) que emulen funcionalidades de librustzcash para tales fines.

4.3.3. E-02 - Respaldo de Frase Semilla

User Story:

> *Como Usuario quisiera poder ver mi frase semilla y mi bloque inicial para poder resguardar mi semilla*

La segunda acción que (idealmente) cualquier usuario debería realizar luego de crear una nueva wallet, respaldar sus bytes semilla. El libro "Mastering Bitcoin" [54] cubre en su capítulo cuarto las distintas formas de formas de almacenamiento "en frío" [12] de las claves generadas para una determinada wallet. Analizando las diferentes alternativas, desde billeteras de papel, ya sean manuscritas o generadas por alguna herramienta para ser impresas 4.11, hasta las más sofisticadas conocidas como "hardware wallet" como las comercializadas bajo las marcas Ledger® o Trezor®, todas tienen en común la necesidad de contar con los bytes semilla. Por tanto para la arquitectura propuesta se ha generalizado este requerimiento en el componente del manejador de semillas descrito en la sección 4.3.1. Además de la frase semilla, es conveniente informar el *"bloque de nacimiento"* o *"bloque inicial"* de la wallet en cuestión. Conocer la

[12] pese a que se asocia este termino a las formas más rudimentarias y analógicas de almacenar una wallet, se refiere a modos de almacenamiento lejos de cualquier tipo de conexión a la red

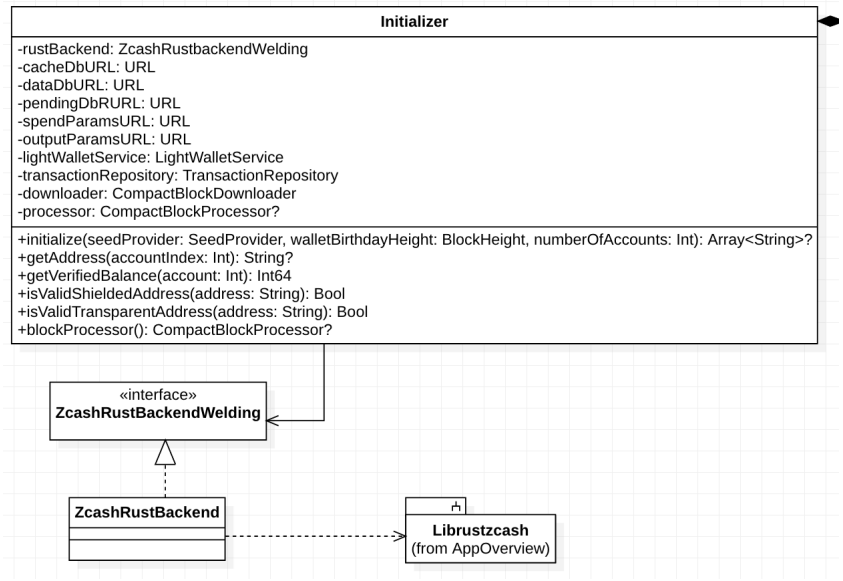


Figura 4.10: Initializer - encapsular la complejidad de requerimientos derivados del protocolo Zcash

altura de la blockchain a partir de la cual existe una billetera es importante a la hora de restaurar una billetera en un cliente.

4.3.4. E-03 - Restaurar wallet desde frase semilla

User Story:

> *Como Usuario poder restaurar mi billetera desde una frase y bloque inicial para poder recuperar mi cuenta completamente y hacer uso de mis fondos*

En las cadenas de bloques de "public ledger" (asiento contable público), todas las transacciones y direcciones son de carácter público. Solo basta contar con acceso a un *full node* con toda la blockchain para poder consultar el saldo de una dirección. Para el caso de Zcash es posible conocer el balance de una dirección transparente y acceder a sus UTXOs[13] de igual manera que en Bitcoin. Esto no es posible en la parte blindada del protocolo, donde es necesario contar con las *Viewing Keys* correspondientes a esos *Inputs* y *Outputs* para poder visualizarlos mediante un intento de desencripción (Trial Decrypt).

Bloque Inicial o de Nacimiento

Cuando se crea una wallet, es correcto asumir que no hay transacciones involucradas en bloques previos a su creación, por ello se existe el concepto del "bloque de nacimiento" o "bloque inicial" (wallet birthday) a partir del cual se considera que

[13] Unspent Transaction Outputs, refiere al valor de las transacciones que no han sido gastados. ver Mastering Bitcoin capítulo 6 [11]

Figura 4.11: Paper Wallet generada por la herramienta bitaddress.org

esa frase semilla está activa en la cadena de bloques. Un bloque inicial puede ser establecido en el momento que el usuario genera una nueva frase semilla aleatoria, o bien cuando se registra la primera transacción que involucra una de las direcciones derivadas desde la semilla.

La diferencia entre tomar una altura para el bloque de nacimiento al momento de la generación de la semilla o hacerlo cuando se registra una transacción, radica en la exactitud del bloque de nacimiento. Nada impide que en usuario cree una billetera y no la use durante un año. Un bloque de nacimiento registrado a la fecha de creación, hará que cuando el usuario requiera restaurar esta billetera desde foja cero, el bloque de nacimiento se encuentre un año atrasado respecto del primer uso real de la billetera, y este deba descargar un año de bloques y analizarlos cuando hay una gran certeza de que no hay en esos bloques ninguna transacción que involucre a esta billetera.

Checkpoints: Puntos de partida para sincronización

La cadena de bloques es una entidad divisible en N bloques pero válida solo en su integridad. Los detalles de la sincronización se cubrirán en la sección 4.3.9. A grandes rasgos, para sincronizar un cliente con la blockchain para una semilla dada, tiene que tomarse un punto de partida, como por ejemplo, el bloque génesis y desde allí empezar a validar la cadena en toda su extensión. Como los bloques anteriores forman parte de la validación de un bloque X dado ente el bloque 0 (Génesis) y el último, es posible establecer puntos de partida para validar solo a partir de ese bloque en adelante ignorando todo lo ocurrido anteriormente en la cadena de bloques. Estos *Checkpoints* permiten que sincronicemos menos bloques de la cadena, conservando la posibilidad de seguir calculando los *hashes* de bloques pasados pero posteriores a ese punto. Contar con este recurso es vital para las billeteras que corran sobre clientes livianos donde el almacenamiento y el tiempo procesamiento son bienes escasos.

Código fuente 4.1: Estructura de un bloque en forma mensaje de RPC de *Lightwalletd*

```
message TreeState {
  string network = 1;// mainnet or testnet
  uint64 height = 2;
  bytes hash = 3// block id
  uint32 time = 4// Unix epoch when the block was mined
  bytes tree = 5;// commitment tree state
}
```

El listado anterior corresponde al Mensaje RPC expuesto por el servidor Lightwalletd. Cabe aclarar que no se recomienda que los clientes consuman los checkpoints directamente desde un único servidor, pues comprometido este, podrían utilizarse checkpoints apócrifos para hacer ataques de denegación de servicio a los clientes que los consuman. Idealmente estos deben proveerse mediante un medio confiable en términos de consenso y de inmutabilidad, como la propia red de nodos pares o desde un clúster dedicado a este fin. Las implicaciones de esa decisión, es materia de infraestructura y topología del ecosistema de la criptomoneda en cuestión y excede el alcance del presente trabajo. En cuanto a requerimientos no funcionales, se ven involucrados los ítem R-12 y R-13 del anexo 7.4. En el caso de Monero, las aplicaciones clientes solo especifican la altura mediante su representación numérica, al nodo al cual se conectan dado que el modelo que utiliza asume plena confianza en él.

4.3.5. E-04 - Recibir fondos

User Story:

> *Como **Usuario** quisiera ver mi dirección como QR para que otro usuario pueda escanearla y enviarme fondos*
>
> *Como **Usuario** quisiera poder copiar mi dirección al porta papeles para poder compartirla por otros medios en forma textual*

Uno de los puntos de fricción en el uso de criptomonedas, es la complejidad de las direcciones. Resultan ilegibles en términos prácticos, y a su vez su ingreso erróneo puede resultar en la pérdida de fondos. En el caso que un usuario ingresara una dirección válida semánticamente pero que no es exactamente la correspondiente al destinatario deseado, no es posible advertirlo. Esto resulta en una transacción que probablemente irá a "hacia la nada", es decir a una dirección que no pertenece a nadie, y aún si las probabilidades se alineasen, estos fondos acabarían en la cuenta de un tercero difícil de localizar y contactar, y en el caso de Zcash, casi imposible de contactar. Para colmo, una vez enviadas a través de la red, las transacciones no pueden ser revertidas.

¿Cómo minimizar los riesgos en una acción tan básica como enviar o recibir fondos adecuadamente? La respuesta yace en proveer medios a las partes para que puedan prescindir del ingreso manual de estos datos. Es ahí donde la representación de información mediante códigos QR viene en auxilio de esta problemática. A su vez, existe la posibilidad de no requerir más que la representación textual a ser compartida por medios electrónicos, para lo cual las plataformas móviles como iOS o Android cuentan con menús estándar en sus sistemas operativos para compartir un ítem de texto hacia otras aplicaciones (mensajería instantánea, correo electrónico, etcétera). La figura 4.12 ilustra la representación de la dirección blindada derivada desde la semilla de la figura 4.4 tanto en forma de código QR como en una representación textual que, por motivos

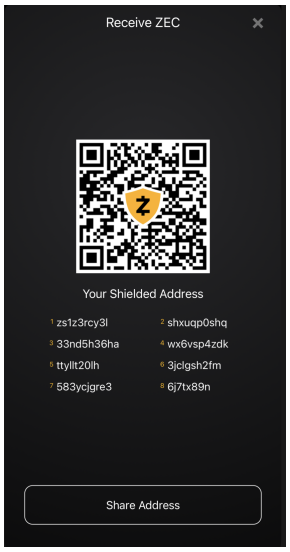


Figura 4.12: Pantalla de visualización de dirección blindada de Zcash. ECC Wallet

visuales y estéticos, ha sido descompuesta en ocho tramos numerados. Al compartir la dirección se lo hará en su forma completa sin subdivisiones.

En la práctica, es muy improbable que una dirección se ingrese manualmente. No obstante, sí se utiliza su forma textual alfanumérica para identificar que se ha escaneado o ingresado el destinatario correcto. De hecho las billeteras de *ledger* público suelen generar direcciones diferentes para cada destinatario con el afán de dificultar la asociación de una dirección a un destinatario específico y preservar el pseudo-anonimato de los usuarios. Estas derivaciones se conocen como Wallets de jerarquia determinística (HD Wallets).

Monero implementa un esquema similar con este objetivo al disponer de una dirección "maestra" que aglutina todos los pagos recibidos y direcciones "secundarias" las cuales se derivan de la primera. Sin importar la cantidad de derivaciones todos los pagos hacia direcciones secundarias serán detectables a nombre de la dirección maestra que el libro *Mastering Monero* [20] recomienda no compartir salvo para casos de uso puntuales.

En definitiva las direcciones tienen una contradicción fundamental. A los efectos de los protocolos son esenciales para poder enviar y recibir transacciones, pero desde la perspectiva humana, son un concepto difícil de asimilar y más aún de comprobar, verificar o siquiera recordar. Por esto último se identifican dos requerimientos base. El Primero, y más superficial, es poder generar un código QR desde un String (R-36). Esta funcionalidad es provista por las respectivas plataformas de los dispositivos móviles. Si así no fuera, debiera implementarse una interfaz que abstraiga dicha funcionalidad. Se considera innecesario para el caso de estudio. El segundo requerimiento identificado es poder derivar una dirección desde una *Spending Key* o clave privada (R-17).

4.3.6. E-05 - Balance

User Stories:

> *Como **Usuario** quisiera poder visualizar los fondos pendientes para confirmar que me han enviado fondos*
>
> *Como **Usuario** quisiera poder visualizar el cambio para poder diferenciarlo de mis fondos disponibles*
>
> *Como **Usuario** ver una pantalla especial si no tengo fondos para no realizar acciones innecesarias*
>
> *Como **Usuario** quisiera poder visualizar mis fondos disponibles para poder enviarlos*

Tal como lo es en Bitcoin, el balance de una wallet es la Zcash es un la suma ingresos y egresos. Los fondos de una privacy coin se distinguen de los de otras criptomonedas por no tener plena revelación de las operaciones de la blockchain. Es importante resaltar que en Zcash, existen dos tipos de Balances: transparente y blindado. Este trabajo se ocupará del último, pues el primero no difiere en nada con Bitcoin y se considera un "problema resuelto" en cuanto a sus objetivos. Para poder ver el detalle de las transacciones es necesario tener las claves adecuadas e ir desencriptando bloque a bloque las transacciones para poder dar con aquellas que corresponden a la wallet en cuestión. En el caso de Zcash, existen varios estratos en los tipos de claves que pueden generarse a partir de una *Spending Key*. Para el caso de los clientes livianos, el balance es el resultado final de sincronizar desde un checkpoint dado hasta el bloque compacto más reciente con un una *Incoming Viewing Key*. Por ello, las operaciones de balance se incluyen en el componente *Synchronizer*. Existen 2 tipos de Balance: el total, y el verificado. La diferencia entre estos yace en la cantidad de confirmaciones que tienen las distintas transacciones que componen un balance. Una transacción que ha sido minada en el bloque *U - 1* siendo U el último bloque, tendrá una confirmación de 1 bloque, y una que lo ha sido en el bloque *U - 5* tendrá 5 confirmaciones. Si bien no existe un consenso para la cantidad de confirmaciones que determinan que ciertos fondos han sido verificados, en Zcash esta consideración se ubica en diez bloques, pero existen casos donde este valor es mucho mayor, o inclusive mucho menor. Sitios de cambio como Coinbase[14] dan por asentada una transacción pasadas las veinticuatro confirmaciones, mientras que clientes de mensajería y *marketplace* como Zbay [15] utilizan cero (leer el mempool[16]) a una confirmación. La diferencia entre fondos totales y verificados es una forma de determinar el "Cambio" pendiente luego de realizar una transacción (ver épica "enviar fondos"). Los clientes livianos que operan con Monero, realizan esta operación de una manera distinta. Donde en primer lugar el cliente le envía al nodo mediante el cual se conecta a la red de pares una copia de una clave de visualización (similar a una IVK de Zcash) mediante la cual éste podrá detectar pagos destinados a dicha clave y comunicarlos al cliente en cuestión. No hay diferencia a los efectos finales de dar a conocer el balance al usuario, pero sí las hay en cuanto a la carga de procesamiento que los clientes livianos delegan a estos nodos. También hay una diferencia clave en el modelo de seguridad y privacidad que conlleva la decisión de delegar claves a un tercero. Este tipo de análisis costo/beneficio donde se compensan

[14] https://www.coinbase.com/

[15] https://www.zbay.app/

[16] el mempool es un sector en memoria donde los nodos Zcash mantienen registro de la transacciones que han sido transmitidas a ellos o sus pares pero no han sido incluidas en un bloque

procesamiento y recursos contra menores capacidades en cuanto a privacidad es un análisis recurrente en este área, no implica que una decisión sea mejor que otra, sino que responden a modelos de usabilidad, seguridad y privacidad diferentes.

4.3.7. E-06 - Enviar Fondos

> *Como **Usuario** quisiera escanear un código QR de otra wallet para poder enviar fondos a ella*
>
> *Como **Usuario** quisiera pegar una dirección desde el portapapeles para poder enviar fondos a ella*
>
> *Como **Usuario** quisiera ver si la dirección ingresada es correcta para poder validar que me proporcionaron un dato válido para el envío de fondos*
>
> *Como **Usuario** quisiera poder ingresar un monto para enviar esos fondos a otra billetera*
>
> *Como **Usuario** para poder ingresar un Memo para la transacción para adjuntar un mensaje encriptado a mi transacción*
>
> *Como **Usuario** quisiera ver el progreso de mi transacción mientras se crea para saber que la aplicación esta funcionando*
>
> *Como **Usuario** quisiera recibir una confirmación de envío de la transacción para poder confirmar que se ha enviado sin errores*
>
> *Como **Usuario** quisiera recibir un link (URI) con información para un pago para poder realizar una transacción a un tercero con toda la información requerida para esta, sin la necesidad de ingresar los datos manualmente yo mismo*

Enviar fondos condensa una gran parte de la experiencia de usuario en una wallet non-custodian y junto con el manejo de la frase semilla aglutinan gran cantidad de fricción en esta materia. A su vez depende de varias otras historias de usuario, requerimientos funcionales y no funcionales.

Para enviar una transacción, se requiere estar sincronizado con la cadena de bloques, ya que para confeccionarla es necesario contar con todos *Notes* y *Nullifiers* que conforman el monto a enviar, y el balance disponible. Una transacción se crea a partir de una o más direcciones (en Zcash pueden ser T o Z), y en el caso de direcciones Sapling de Zcash es posible adjuntar uno o más *memos* de hasta 512 bytes[17].

Las direcciones de las criptomonedas son legibles, pero como se discute en la sección 4.3.5, por su longitud y composición alfabética poco prácticas para su utilización. En términos generales puede afirmarse que son escasos los escenarios en los cuales un usuario ingresa la dirección de un destinatario de forma manual. Por lo cual estas se copian y pegan en el portapapeles, se escanean por medio de códigos QR o una dirección URI, de forma tal que la acción efectuada por el usuario es la constatación visual de que la dirección en cuestión se asemeja a la del destinatario y a una confirmación que indique a la aplicación cliente que este es el parámetro a incluir.

Código fuente 4.2: Estructura de una dirección URI acorde al ZIP-321: Una URI representando el pedido de un pago de 1 ZEC a una dirección blindada con un

[17] un memo es un campo opcional dentro de un output de una transacción, conformado por 512 caracteres bytes, que en se interpretaran como caracteres UTF-8. Estos memos están encriptados y solo pueden ser descifrados por los destinatarios o quienes posean las *viewing Keys* correspondientes

memo codificado en base 64 y un mensaje para ser mostrado visualmente al usuario

```
zcash:ztestsapling10yy2ex5dcqkclhc7z7yrnj
q2z6feyjad56ptwlfgmy77dmaqqrl9gyhprdx59q
gmsnyfska2kez?amount=1&
memo=VGhpcyBpcyBhIHNpbXBsZSBtZW1vLg
&message=Than\%20you\%20for\%20your\%20purchase
```

Para enviar una transacción el cliente debe poder generar la Prueba de Cero Conocimiento mediante la cual el resto de los pares de la red pueden verificar que los Notes y Nullifiers utilizados en este "Gasto" (Spend) son válidos y corresponden a remitente. Realizar estas ZK-Proofs es costoso en términos de recursos computacionales. Se requieren aproximadamente 10 segundos para confeccionar una transacción[18]. Por ello es necesario que esta operación sea asincrónica y que no sea interrumpida por el sistema operativo. Localmente las transacciones se realizan en base a los datos disponibles en el cliente. Su validez a nivel global se refrenda en los nodos de la red. La recepción de una transacción por parte de uno de los nodos de la red, implica que ésta es válida en términos del protocolo de consenso y que será incluida en el "mempool", un repositorio en memoria de transacciones que han sido recibidas por uno o más nodos. Cada nodo tiene su propio mempool. Este se nutre de transacciones que han sido enviadas a él o que otros nodos pares han dispersado a través de la red. Las transacciones aguardan allí hasta que un nodo minero descifra el hash correspondiente para generar un nuevo bloque y las incluyen o no en éste. Además del monto propio de la transacción, se incluye una comisión para los mineros. El monto de la comisión es público a toda la red, aún si esta es una transacción que se realiza entre direcciones blindadas. Si bien este monto no está predeterminado y puede contener cualquier monto de zatoshi válido[19], es recomendable para una privacy coin que este valor sea uniforme, ya que si determinados clientes difieren en este valor de carácter público, se abre una ventana para rastrear e desanonimizar a las partes involucradas gracias a que ese monto revela un aspecto de la transacción.

Es imposible asegurar que una transacción será incluida en un bloque fehacientemente. En cuando red de nodos está congestionada con una alta demanda de transacciones a verificar e incluir en un bloque, es posible que aquellas que tengan comisiones bajas, queden relegadas, nunca sean minadas. Si esto ocurriese el remitente de una transacción tendría bloqueados los UTXOs utilizados en esa transacción hasta tanto se descongestione la red y un nodo la elija para conformar un bloque que se ha minado. Por lo tanto, es necesario establecer un mecanismo para poner un limite de tiempo a la validez de una transacción. Para esto, al crearlas, los remitentes utilizan la altura del bloque actual [20] y establecen una fecha de expiración expresada en la *altura actual* + una cantidad de determinada de bloques que están dispuesto a esperar antes de considerar que la transacción ha expirado y debe ser descartada del mempool de acuerdo al protocolo de consenso. Esto significa que si llegado el bloque de expiración, la transacción no ha sido incluida en un bloque, debe ser eliminada de la red. Esto posibilita que los fondos comprometidos en ella puedan volver a ser utilizados por

[18]Se realizaron pruebas en Dispositivos Físicos: iPhone 7, iPhone 11, Android Pixel 5 y sus respectivos simuladores

[19]para evitar ataques de denegación de servicio, el protocolo permite que haya una transacción con una comisión de cero zatoshi por bloque

[20]La cantidad de bloques que tiene una cadena define la altura de la misma. El numero de orden de un bloque se conoce como 'blockheight'.

el remitente. No existe una "orden" por parte del creador de una transacción para "cancelar" la operación, sino que al crearla, mediante este mecanismo, todos los nodos saben cuando es el momento de descartar una transacción que ha expirado.

De modo inverso una transacción que ha sido minada, es decir, seleccionada para conformar un bloque, será leída en la blockchain al sincronizar por parte de los destinatarios y su remitente. Para los primeros se transformará en "fondos pendientes" y para este último en una transacción a confirmar.

¿Por qué se habla de fondos pendientes de confirmación cuando la transacción fue incluida en un bloque? ¿no es acaso la blockchain inmutable?

La respuesta a estas preguntas yace en la naturaleza del consenso Proof-of-Work. Si bien existe una sola blockchain para una criptomoneda dada, esa cadena de bloques puede tener varias puntas diferentes en su extremos final. Los bloques y las transacciones no se dispersan por la red de nodos pares de forma uniforme. Entonces, así como las transacciones llegan a los distintos nodos en distintos momentos, los bloques creados por mineros se dispersan por la red de la misma forma. Y por ello se habla de que al extremo final de la cadena se forman hay cadenas hermanas, que son producidas por distintos mineros que van ensamblándolas a medida que van obteniendo los nuevos bloques que se propagan a través de la red. Llega un punto en que, mediante el protocolo de consenso, el nodo que tenga la cadena hermana que acumule más trabajo (más bloques y más transacciones) es la **Best Chain**. Y todos los nodos de la red tienen que reorganizarse para volver hacia atrás unos bloques y verificar el nuevo extremo de la cadena.

Es por ello que una transacción puede formar parte de un bloque en una cadena hermana y no de la «*Best Chain*». Como se detalla en la sección 4.3.9, de ocurrir una reorganización de la cadena, es posible que esta transacción no sea minada, o que lo esté en otro bloque. Por tanto la confirmación de una transacción no es binaria, sino que se establece a partir de que se han minado una cantidad adicional de bloques. El número establecido como "prudente" para esa cantidad de confirmaciones varía dependiendo del protocolo de cada criptomoneda y del criterio y las necesidades de cada aplicación cliente.

4.3.8. E-07 - Historial de Transacciones

> *Como **Usuario** quisiera ver las transacciones pendientes de confirmación para conocer los detalles de las transacciones que he enviado o recibido recientemente*
>
> *Como **Usuario** quisiera ver las transacciones fallidas en mi historial para poder conocer el historial de envíos recientes con exactitud*
>
> *Como **Usuario** quisiera ver las transacciones recibidas históricamente para poder conocer el historial de transacciones recientes con exactitud*
>
> *Como **Usuario** quisiera ver el monto, altura, memo y fecha de las transacciones enviadas y recibidas para poder conocer el historial de transacciones recientes con exactitud*
>
> *Como **Usuario** quisiera ver el ID de la transacción realizada para poder verlo en un explorador de bloques de la blockchain*

El viejo y conocido "Extracto" o "Resumen de cuenta" no es ajeno al mundo de las criptomonedas. De hecho, si uno se sitúa en un bloque dado y mira hacia sus predecesores en la cadena de bloques hasta el bloque génesis, verá en definitiva un

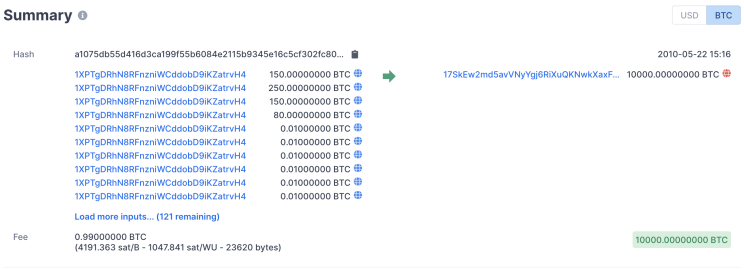


Figura 4.13: Bitcoin Pizza. La transacción de criptomoneda más famosa

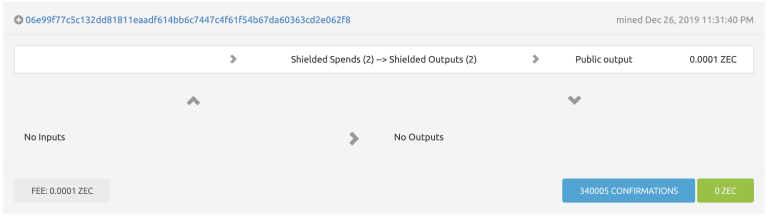


Figura 4.14: Primera transacción z2z, desde un dispositivo iOS. Indistinguible de cualquier otra. Realizada por Francisco Gindre

"extracto" de todas las "cuentas" habidas hasta el momento de la creación de ese bloque desde donde se mira hacia el origen.

La principal diferencia entre una *privacy coin* y una criptomoneda de libro contable (ledger) público, yace en el nivel de detalle de la información que es de carácter público frente a aquello que es de carácter privado y requiere de claves privadas especiales para ser visualizada. En la figura 4.13 se ve una captura de pantalla de un explorador de bloques de Bitcoin, donde se visualiza la transacción conocida como "Bitcoin Pizza"[59], en la que Laszlo Hanyecz pagó 10.000 BTC por dos pizzas a otro usuario de bitcointalk.com en mayo de 2010. Fuera de lo ilustre y mundano del contexto, se puede observar que hay varios "inputs" y un solo "output" hacia una dirección de BTC.

En el caso de Zcash, la parte de la blockchain referida a transacciones entre direcciones transparentes, se asemeja al nivel de visibilidad pública que podría encontrarse en Bitcoin 0.14, versión de la cual se realizó el Fork. No así para las transacciones blindadas, donde existen tantas vistas posibles como combinaciones de Viewing Keys. En la figura 4.14 se ve una captura de pantalla de la primera transacción generada desde un dispositivo iOS. Es preciso dar cuenta que esta última afirmación es imposible de constatar[21]. Sólo podemos observar que es una transacción Sapling de Zcash entre dos direcciones blindadas, con una comisión para el minero de 1000 zatoshis[22]

[21] Quizás un aspecto "novedoso" del presente trabajo es contener una cita bibliográfica que sea imposible de constatar sin que ésta no haga mella en la rigurosidad del mismo

[22] La comisión al momento de escribir este capítulo era de 10.000 zatoshis. El ZIP-313 modificó este monto a 1000 zatoshis. `https://zips.z.cash/zip-0313`

(0,00001 ZEC) un input y dos outputs. Los detalles de estos no podrán visualizarse sin desencriptar esta transacción con las correspondientes claves del emisor y el destinatario.

A diferencia de las Transacciones Transparentes, que pueden consultarse en una API remota directamente en un nodo, para poder listar le historial de transacciones blindadas de un usuario es necesario sincronizar con la blockchain y desencriptar cada bloque compacto en busca de transacciones que le pertenezcan con las claves de visualización (Viewing Keys) en poder del usuario. Esto sea hace mediante un ciclo de "prueba y error", donde fallar en desencriptar una transacción se considera meramente como el hecho de que esa transacción no es pertinente a las claves que están cargadas en una instancia del sincronizador (ver 4.3.9).

4.3.9. E-08 - Sincronizar

> *Como **Usuario** estar al día con la blockchain para poder conocer mi balance actualizado al ultimo bloque minado y disponer de mis fondos*
>
> *Como **Usuario** quisiera conocer si la wallet está sincronizándose con un porcentaje para poder saber que la aplicación no está colgada y que está trabajando*
>
> *Como **Ing. seguridad** requiero que la wallet esté siempre actualizada a la «best chain» para que no puedan hacerse ataques de «side-chain»*

Se llama sincronizar al proceso que a refiere descargar los bloques que van desde el bloque génesis o un *checkpoint* hasta el último bloque minado. En un protocolo tipo *Proof-of-Work* la mayor parte de la cadena es compartida por todos los nodos de la red. Como se menciona anteriormente, es común y necesario que haya varios nodos computando y minando bloques. La propagación de estos bloques a través de la red no es uniforme. Es posible que se generen cadenas alternativas que solo difieren en un puñado de bloques al extremo de la misma con la cadena válida o incluso contienen los mismos bloques pero en distinto orden. En el capítulo 10 del libro "Mastering Bitcoin" [7] se ilustra completamente el proceso.

Mediante el algoritmo de consenso, llega un punto en el tiempo donde se determina que hay una cadena que acumula más pruebas de trabajo que las demás y se convierte en la cadena válida. Todos los nodos que hayan estado sincronizando otras ramas de la cadena válida, deberán reorganizarse, descartar los bloques inválidos y ponerse al día con el tramo de cadena que les falta para estar sincronizados en la cadena válida. Las reorganizaciones son frecuentes dentro del consenso y una de las principales razones por las cuales existe el concepto de "confirmaciones" cubierto anteriormente. Los clientes livianos no son ajenos a estos cambios en el extremo de la cadena de bloques. Como se detalla en el capítulo 2, los clientes livianos reciben bloques compactos derivados de un *full node*. En el momento que ocurre una reorganización en el nodo, este ocurre en cascada en el servidor de bloques compactos (LightWalletd en el caso de Zcash) y lo mismo debe ocurrir en las wallets. De no ser así exista los clientes livianos quedan desfasados respecto de la mejor cadena y dejan de coincidir con los valores de consenso del resto de la red. Esto genera que no se reciban fondos y sobre todo que no puedan enviarse, dado que los *hashes* sobre los cuales se anclarán los *Notes* que se utilizarán en posibles transacciones no serán válidas y serán rechazadas por los nodos de la red, quedando inutilizado todos los clientes livianos que sufrieran este desfasaje producto de una no-reorganización.

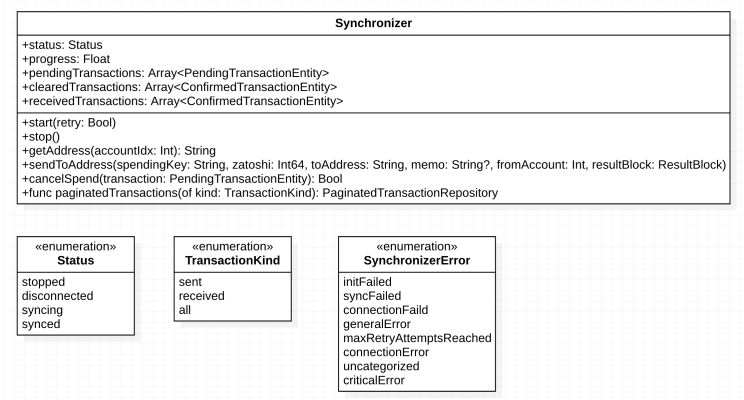


Figura 4.15: Synchronizer: el corazón de un cliente liviano

Sincronizar es en esencia la tarea fundamental de toda aplicación cliente de una blockchain. Los detalles de cómo se realiza, dependen del tipo de protocolo de consenso que tenga cada criptomoneda. No obstante es posible abstraer los bloques fundamentales y exponerlos en una interfaz. La figura 4.15 ilustra el diagrama UML del componente Synchronizer, que expone todos los atributos y comportamientos necesarios para que un cliente liviano se mantenga sincronizado la blockchain. Este componente sirve como façade de toda la lógica del dominio correspondiente al protocolo de la criptomoneda y expone el estado en el cual el cliente liviano se encuentra respecto de la cadena de bloques tanto a nivel conectividad, indicar el progreso en el caso de estar sincronizando, y las transacciones enviadas (pendientes o no) y recibidas. La implementación de este componente depende mucho de la plataforma y de la arquitectura de aplicación elegida por los desarrolladores. En primer lugar porque es necesario contar con el conocimiento del ciclo de vida de la aplicación en el marco del sistema operativo en el que se está ejecutando, para poder conocer si la aplicación está corriendo activamente o bien si ha sido suspendida por el usuario y no está siendo utilizada activamente, así como la disponibilidad de conexión a Internet y el manejo de las interrupciones del sistema. En segundo lugar tanto en Android como en iOS conviven distintos tipos de paradigmas de programación que influyen substancialmente en el desarrollo de una aplicación. Esta interfaz puede traducirse linealmente en el caso de utilizar programación imperativa tanto en Java, Kotlin, Objective-C o Swift. No es así cuando se emplea programación reactiva como RxJava, Kotlin CoRoutines y Flows para el caso de Android y RxSwift o Combine para iOS. El componente en sí seguirá existiendo como entidad, pero su interfaz y comportamientos deberán adecuarse a los previstos este paradigma de programación.

4.4. Resumen de Historias de usuario de billeteras electrónicas de Privacy Coins

En el capítulo 2 se introduce el concepto general de las criptomonedas y en particular de las AEC Monero y Zcash. Ya en ese capítulo, sin ahondar en especificidades se van delineando trazas gruesas de casos de uso, funcionalidades, temáticas. Quienes sean más adeptos a las metodologías ágiles podrán reconocer cómo el escenario donde Alicia recibe sus honorarios del minero Roberto describe varias historias de usuario.

El trabajo de elicitación solo comienza allí. En la sección 4.1 se relevaron distintos proyectos de wallets de código abierto para corroborar si estos supuestos que surgen de la propia explicación teórica del uso de privacy coins o también se verifican en las propias billeteras en forma de funcionalidad y de qué forma.

La tabla de historias de usuario agrega las historias de usuario en Temáticas (themes, Épicas (epics). Las historias de usuario están relatadas en forma de tarjetas o fichas de papel de acuerdo a la metodología Scrum siguiendo el formato *COMO* - actor - *QUIERO* - acción a realizar *PARA* - Objetivo a lograr.

Theme	epic	Epic ID	Story ID	COMO	QUIERO	PARA
Gestión de claves públicas y privadas	nueva wallet	E-01	1	Usuario	Crear una nueva frase semilla	utilizar mi wallet
	respaldo de seed frase	E-02	2	Ing. seguridad	un recordatorio a los usuarios que resguarden su frase	que los usuarios resguarden efectivamente su frase semilla
			3	usuario	poder ver mi frase semilla y mi bloque inicial	poder resguardar mi semilla
			4	usuario	poder copiar mi frase semilla al portapapeles	resguardarla más rápidamente
	restaurar wallet desde frase	E-03	5	usuario	poder restaurar mi billetera desde una frase y bloque inicial	poder recuperar mi cuenta completamente y hacer uso de mis fondos
operaciones	recibir fondos	E-04	6	usuario	poder visualizar mi dirección en un QR	recibir fondos mediante el escaneo de ese código desde otra wallet
			7	usuario	poder visualizar mi dirección como texto y copiarla al portapapeles	poder compartirla por medios electrónicos a otras personas
estado de la wallet	balance	E-05	8	usuario	poder visualizar los fondos pendientes	confirmar que me han enviado fondos
			9	usuario	poder visualizar el cambio	para poder diferenciarlo de mis fondos disponibles
			10	usuario	ver una pantalla especial si no tengo fondos	no realizar acciones innecesarias
			11	usuario	poder visualizar mis fondos disponibles	poder enviarlos
operaciones	enviar fondos	E-06	12	usuario	escanear un código QR de otra wallet	poder enviar fondos a ella
			13	usuario	pegar una dirección desde el portapapeles	enviar fondos a ella
			14	usuario	ver si la dirección ingresada es correcta	validar que me proporcionaron un dato correcto para el envío de fondos
			15	usuario	poder ingresar un monto	enviar esos fondos a otra billetera
			16	usuario	recibir una confirmación de envío de la transacción	poder confirmar que se ha enviado sin errores
			17	usuario	recibir un link (URI) con información para un pago	“poder realizar una transacción a un tercero con toda la información requerida para esta,sin la necesidad de ingresar los datos yo mismo”
			18	usuario	ver los errores que hayan ocurrido durante el envío de la transacción	poder confirmar que no se ha enviado
estado de la wallet	historial de transacciones	E-07	19	usuario	ver las transacciones pendientes de confirmación	conocer los detalles de las transacciones que he enviado o recibido recientemente
			20	usuario	ver las transacciones fallidas en mi historial	poder conocer el historial de envíos recientes con exactitud
			21	usuario	ver las transacciones recibidas históricamente	poder conocer el historial de transacciones recientes con exactitud

			22	usuario	ver el monto y fecha de las transacciones enviadas y recibidas	poder conocer el historial de transacciones recientes con exactitud
			23	usuario	ver el ID de la transacción realizada	para poder verlo en un explorador de bloques de la blockchain
operaciones	sincronizar	E-08	24	usuario	estar al día con la blockchain	poder conocer mi balance actualizado al ultimo bloque minado y disponer de mis fondos
			25	usuario	conocer si la wallet se esta sincronizando con un porcentaje	para saber que la aplicación no esta colgada y que esta trabajando
			26	Ing. seguridad	requiero que la wallet esté siempre actualizada a la 'best chain'	para que no puedan hacerse ataques de 'side-chain'

Cuadro 4.1: Resumen de historias de usuario surgidas del relevamiento de requerimientos

4.5. Kit de desarrollo de Zcash

La funcionalidad núcleo de una wallet es la interacción con el ecosistema de cada criptomoneda que soporta. Muchas funcionalidades son comunes en todas las criptomonedas en su esencia, cambiando solamente en las particularidades de cada protocolo. Es posible y usual que en una Wallet "Multi-coin" [23] donde se soportan distintas criptomonedas, se utilice la misma frase semilla para un mismo usuario, donde lo que cambia es la derivación de la entropía en las claves privadas y públicas pertinentes. Lo mismo aplica para funcionalidades como mostrar las direcciones del usuario etcétera. Las wallets no son otra cosa que aplicativos cliente de abstracciones generadas sobre la complejidad de cada criptomoneda.

Es así que uno de los componentes más importantes de la arquitectura propuesta es un SDK y las interfaces que conectan al ecosistema del protocolo Zcash con los clientes livianos.

Hay varios factores que contribuyen a la necesidad de generar un kit de desarrollo que haga las veces de "Caja Negra". En primer lugar están las buenas prácticas del arte de la programación donde poder generar interfaces que expresen comportamientos lo suficientemente abstractos para favorecer su reutilización, es algo positivo para cualquier pieza de software. En segundo lugar, la complejidad inherente del campo de las criptomonedas pone en relieve una estructura donde existe un núcleo donde se implementa el protocolo y luego otros desarrolladores que parten de aquella base para crear aplicaciones y herramientas con distintos niveles de abstracción y funcionalidades. En el anexo puede encontrarse la tabla 7.2 que resume una colección de historias de usuario que se condensan en el SDK de Zcash.

El SDK tiene una arquitectura de capas, que proveen distintos niveles de agregación y complejidad, lo cual da lugar a los desarrolladores que lo utilizan a elegir si desean una integración sencilla de unas pocas líneas de código o bien más a medida de sus necesidades puntuales. En la figura 4.16 se ilustra la estructura general de estos kits de desarrollo.

4.5.1. Capa superficial: Inicializar y Sincronizar

Initializer: punto de partida e instanciación de componentes del SDK

Condensa la complejidad de crear todos los componentes necesarios para utilizar Zcash en un cliente liviano. El rol de un objeto inicializador es requerir al desarrollador el mínimo indispensable de parámetros requeridos para instanciar todos los componentes necesarios para el uso del SDK, escondiendo los detalles de implementación de las interfaces publicas que están disponibles para este.

Es así que se entiende a este *Initializer* como un objeto que se utiliza como parámetro construir el grafo de objetos que componen el SDK, como el *SDKSynchronizer* que se describe a continuación.

[23] refiere a aquellas aplicaciones que soportan varias criptomonedas y protocolos

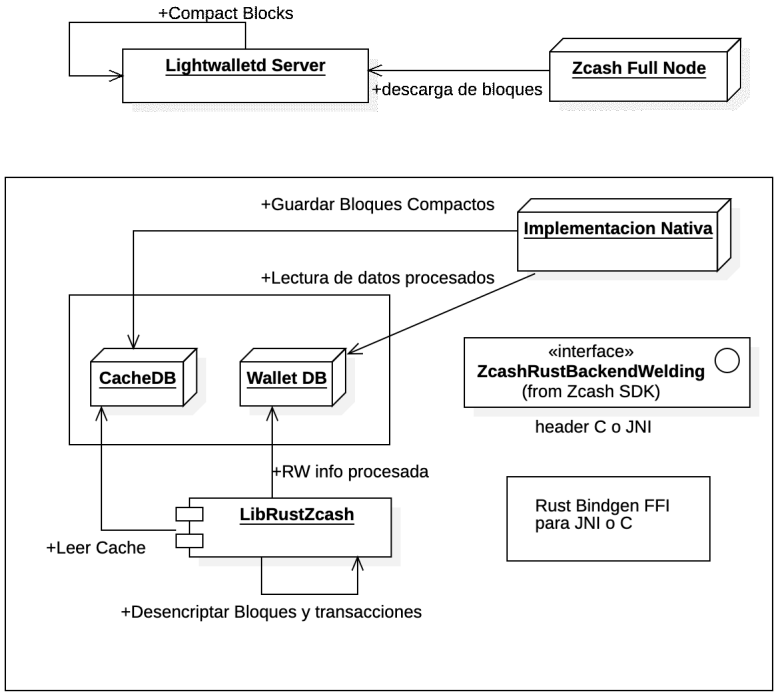


Figura 4.16: Diagrama de la estructura del SDK de Zcash para Android y iOS

Figura 4.17: Diagrama de clases de *Initializer* y *Synchronizer*

Synchronizer: la fachada simplificada del SDK

Se considera a la interfaz "Synchronizer" como la capa más exterior y de uso general del SDK. Esta provee una funcionalidad clara y concisa: Sincronizar la cadena de bloques e informar sobre ciertos eventos de interés.

La implementación de esta interfaz llamada "SDKSynchronizer" provee todas las historias de usuario descritas en las épicas de la sección 4.3 a excepción de la derivación de frases semillas BIP-39 a bytes, y la derivación de claves a direcciones.

Este componente apunta a aquellos desarrolladores que no requieren más que proveer las claves del usuario e informar transacciones, balance y progreso de la sincronización y el estado de la misma.

Al observar la interfaz, pueden identificarse numerosas funcionalidades. Puede interpretarse a un Sincronizador como la façade del kit de desarrollo del sistema. Todos los métodos y variables allí publicados condensan casi la totalidad de todo lo que es posible realizar con el los componentes que integran esta herramienta.

A su vez la interfaz muestra la intención y la funcionalidad claramente. Al leer los primeros métodos ya se puede deducir que un sincronizador, puede iniciarse (start), detenerse (stop), que tiene un estado (status), y que dentro de esos estados tiene un indicador de progreso. Que además pueden obtenerse las direcciones que se están sincronizando y enviar fondos a una determinada dirección o intentar cancelar un envío realizado.

Al momento de enviar una transacción, es necesario estar "al día" con la cadena de bloques, de lo contrario la transacción generada estará expirada antes de siquiera salir del dispositivo que la genera. Se define la expiración de una transacción en 20 bloques a partir de la altura de la cadena desde al cual fue generada. Esto quiere decir que si al momento de generar la transacción el cliente estaba sincronizado al bloque 1000, la transacción creada expirará en el bloque 1020. Aquellas transacciones que no hayan sido incluidas en un bloque cuando se alcanza su altura de expiración, son eliminadas de todos los nodos de la red, de acuerdo con lo explicitado en el protocolo de consenso.

El sincronizador también se encarga de exponer las transacciones existentes para la claves provistas. Existen dos tipos de transacciones: Confirmadas y Pendientes. Según el protocolo definido en ZIP-307, los clientes livianos no tienen un acceso al *mempool* de los nodos a los cuales están conectados mediante el servidor *lightwalletd*. Por lo tanto esta diferenciación es distinta a la que se encuentra en otro tipo de wallets. Al no tener acceso a las transacciones que están en la memoria de los nodos, el procesador de bloques solo puede conocer las transacciones que han sido encontradas en los bloques (ConfirmedTransaction). En este ámbito, las transacciones Pendientes (PendingTransaction) hacen referencia a aquellas transacciones que han sido enviadas desde el dispositivo en cuestión y no han sido leídas dentro de un bloque procesado.

En definitiva, el componente *SDKSynchronizer* que implementa la interfaz *Synchronizer*, es una clase cliente del *CompactBlockProcessor*. Este es la pieza clave de todo el kit de desarrollo, siendo que es el encargado de realizar la sincronización y el procesamiento de la cadena de bloques compactos que se consume desde el servidor que implementa el protocolo de clientes livianos [22] descripto en la sección "Protocolo de Clientes livianos para detección de pagos de Zcash" 7.5 de las *incoming viewing keys* derivadas de las frase semilla del usuario.

4.5.2. Capa configurable: Procesamiento de bloques

En este nivel de abstracción se pueden observar los componentes (y sus interfaces) que proveen todas las funcionalidades que se requieren para sincronizar la cadena de bloques y utilizar la cadena de bloques. De modo tal que es posible que ante un requerimiento puntual, se pueda reemplazar la implementación concreta de una interfaz por otra. Por ejemplo, utilizar un protocolo distinto para conectarse al servidor de bloques compactos o definir un modelo de almacenamiento diferente a SQLite3. La intención de definir un nivel de procesamiento de bloques que utiliza interfaces, es justamente dejar abierta la posibilidad de modificar estos aspectos concretos que surgen de elecciones propias de los desarrolladores del proyecto y que puede no resultar convenientes o viables para otros.

CompactBlockProcessor: el núcleo de la arquitectura.

El procesador de bloques compactos tiene como tarea principal estar al día con la cadena de bloques. Periódicamente (o a pedido expreso del desarrollador) debe calcular la diferencia de altura entre el último bloque descargado y el extremo actual de la cadena y realizar la actualización correspondiente[24].

Para esto se establece una constante que determina cuántos bloques tendrá cada tanda de bloques a descargar y procesar (La clase ZcashSDK contiene un número por defecto pero éste es configurable en el propio procesador). El Proceso de sincronización es sencillo y cuenta con 4 etapas: descarga, validación, escaneo y "mejoramiento" (enhancement) que se repiten en ciclos hasta llegar a la última altura publicada.

En la etapa de descarga, el procesador requiere al servidor un rango de bloques de $1 <$ rango $<$ lote_máximo. Estos se guardan como bytes en una base de datos SQLite llamada «"CacheDB"», en la tabla «compactblocks» que los indexa por altura de bloque. Esto responde a la necesidad de establecer un método de intercambio de datos accesible desde *Librustzcash*.

Una vez descargado el rango de bloques de interés, se debe proceder a la validación de los mismos. Esta consiste en un llamado a la función *validate_combined_chain()* dentro de *librustzcash* quien verifica que en la base de datos indicada como cache existe la tabla *compactblocks* y que que las alturas de los bloques allí almacenados sean continuas y que además los hashes de los bloques sean consistentes unos con otros (recordemos que un bloque contiene el hash del bloque que lo precede). Esto permite detectar errores en la descarga y reorganizaciones de la cadena de bloques compactos.

Manejar las reorganizaciones es parte de las tareas delegadas al CompactBlock-Processor. Resolver una reorganización consiste en retroceder una cantidad de bloques en la sincronización local de la cadena y volver a intentar volver a la última altura publicada. Aunque es más que infrecuente, es hipotéticamente posible encontrar reorganizaciones de varios bloques, siendo 100 (cien) el límite establecido por el protocolo en el ZIP-307 [22]. Si bien en condiciones óptimas la descarga de 100 bloques de información es irrelevante, las reorganizaciones se resuelven progresivamente, primero

[24] Es necesario considerar que los clientes livianos siempre visualizan la cadena de acuerdo al servidor de bloques compactos al que se encuentra conectado. Es posible que ese nodo no esté sincronizando la «Best-Chain»

regresando 10 bloques hacia atrás y luego incrementalmente hasta llegar al máximo de bloques permitido. Debe considerarse que deshacer un tramo de bloques ya sincronizados no solo requiere borrar los bloques compactos descargados y reemplazarlos por unos solicitados al momento de detectar la reorganización, sino que además, se deben eliminar todos los datos derivados de la sincronización, para restaurar la wallet al estado que tenia al momento de la altura a la cual se desea "rebobinar" (rewind).

Los datos derivados a los que hace alusión el párrafo anterior son generados en la etapa denominada escaneo, que consiste en la desencripción por prueba y error de las transacciones compactas contenidas en los bloques descargados mediante la utilización de las claves de visualización provistas por el usuario. Esta operación también la realiza *librustzcash* por medio de un llamado a la función *scan_blocks* de la interfaz FFI. Esta incluye las URLs de las bases de dato «CacheDB» y «DataDB», donde se almacenan los bloques compactos y donde se almacenarán los datos desencriptados de los mismos respectivamente. La base de datos denominada DataDB, contendrá las transacciones extraídas, los *received_notes* y los *sent_notes* que visibles a las claves proporcionadas por el usuario. Con ellos puede determinarse el balance de una 'cuenta'[25]. Es importante destacar que dentro de la *compact blockchain* solo se encuentran los inputs y outputs blindados. Todo lo que refiere a operaciones transparentes, pueden consultarse directamente sin necesidad de hacer el proceso de "trial-decryption" que se describió anteriormente. Por otra parte los *encrypted-memo* de las transacciones no se encuentran codificados dentro de las transacciones compactas. Cada memo tiene 512 bytes, contenga este o no dato significativo alguno. Lo mismo ocurre con los datos necesarios para establecer la diferencia entre transacciones salientes y entrantes.

Para ello existe el paso denominado "mejoramiento" o «enhancement». Una vez verificado y escaneado un rango de bloques determinado, es posible determinar si este contiene transacciones de interés para las claves de visualización presentes. Por lo tanto la interfaz de GRPC de lightwalletd cuenta con un servicio que retorna una transacción completa (no compacta) en base a los bytes que conforman un «txid». Estas transacciones son las mismas que se encuentran en los nodos que conforman el ecosistema Zcash. Al desencriptar una de estas transacciones se obtienen todos los componentes especificados en el protocolo Zcash, pudiendo así completar la información que no esta contenida en la cadena de bloques compacta (Requerimiento R-22 anexo 7.4).

Esto responde a que la mayoría de los bloques no tendrá transacciones de interés para una clave dada, y por lo tanto es conveniente excluir la mayor cantidad de información posible de la versión de la blockchain que los servidores de clientes livianos exponen a estos. La reducción en términos de ancho de banda ronda el 90 %.

Existe un problema con este esquema de "mejoramiento" de la información mediante la descarga de la transacción completa. Al requerir una transacción puntual al servidor, el usuario revela su interés por la misma. Si bien esto no reviste un problema de seguridad, sí lo es en cuanto a privacidad se refiere. Como ya se ha mencionado este trabajo no se enfoca en este tipo de cuestiones. Aún así se identifica este problema como una línea de investigación posible a futuro en el campo de la recuperación privada de información o PIR (Private Information Retrieval) por sus siglas en inglés. Este

[25] recordemos que el concepto de cuenta no existe en el protocolo, solo es una abstracción representada a fin de mejorar la usabilidad y entendimiento para los usuarios.

asunto también trae a colación algo mencionado casualmente al inicio de esta sección, donde se establece que el procesador de bloques se inicia por requisito del usuario o periódicamente. Toda interacción de la wallet con un nodo o un servidor presenta algún tipo de filtración de información para un observador pasivo del tráfico de red. En términos de privacidad, lo óptimo sería que todos los clientes livianos sincronizaran su información a un ritmo similar, de modo tal que sea más difícil desanonimizar a un usuario por medio de la observación del comportamiento de su aplicación en cuanto a tráfico de red.

Así como el proceso de descarga, validación, escaneo y mejoramiento de bloques ser repite cíclicamente desde la altura almacenada localmente hasta la última altura publicada por el nodo al cual la wallet tiene acceso. Existe un escenario primigenio que solo ocurre al momento de restaurar una frase semilla por primera vez. En este caso no habrá una última altura de bloque descargada. Para ello el procesador de bloques debe ser inicializado con *bloque de nacimiento* tal como se menciona en la sección 2.6.1. Este puede ser basado en un checkpoint que el usuario haya destacado o bien la constante «SAPLING_ACTIVATION». Lo ideal es que el usuario siempre pueda restaurar su wallet por completo descargando la menor cantidad de bloques posible, pero que no disponga de una altura de referencia para hacerlo no debe ser un impedimento. Si bien el usuario que desconozca su bloque de nacimiento sera "penalizado" al bajar más cantidad de datos a su dispositivo, una vez conocida la primera transacción que pertenece a esas claves, será posible proporcionar el bloque de nacimiento exacto.

El procesador de bloques es un objeto complejo, que esta compuesto por varias clases que le proveen distintos comportamientos.

El objeto de configuración *CompactBlockProcessor.Configuration* provee los distintos parámetros que ser requieren para instanciar todos estos componentes y a su vez determinar ciertos comportamientos como el tamaño de cada rango de bloques a descargar o el número máximo de bloques a retroceder en caso de reorganización de la cadena de bloques.

La interfaz *CompactBlockDownloading* define el comportamiento esperado para un componente dedicado a descargar bloques compactos y guardarlos localmente en el cliente para su posterior uso. En caso de tener que volver hacia atrás en procesamiento de bloques la clase que implemente esta interfaz deberá poder hacer el correspondiente rollback del cache.

LightWalletService: conexión al protocolo de clientes livianos

Si bien la implementación especifica de la comunicación entre nodos y servicio de clientes livianos y los propios clientes se materializa mediante gRPC[26] la interfaz para comunicarse a un servidor desde clientes livianos, no se encuentra expresado a través de ellos. Sino que presenta abstracciones de acuerdo a los datos que se utilizan en el kit de desarrollo. Esta interfaz expresa los requerimientos no funcionales que presentan en el ZIP-307, los cuales son necesarios para sincronizar la cadena de bloques compactos y enviar transacciones. La figura 4.18 muestra la interfaz abreviada de este

[26]gRPC (gRPC Remote Procedure Calls) es un sistema de llamada a procedimiento remoto (RPC) de código abierto desarrollado inicialmente en Google

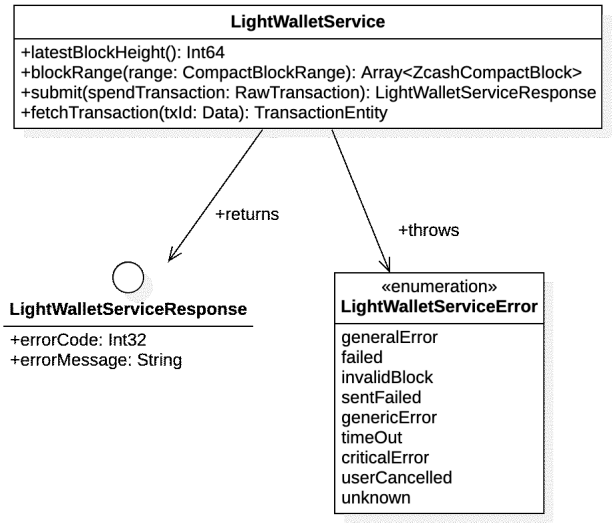


Figura 4.18: Interfaz abreviada de LightWalletService con sus métodos síncronos.

servicio conteniendo solo sus métodos sincrónicos[27]. A simple vista puede observarse que tiene métodos sincrónicos y asíncronos. Esto responde a la necesidad de contar con comunicaciones que se inicien síncronamente desde hilos de ejecución secundarios o bien, de forma asíncrona desde el hilo de ejecución primario.

TransactionRepository: Acceso a Transacciones detectadas

Otro componente del procesador es el repositorio de transacciones o *Transaction-Repository* que provee el acceso a los datos derivados del procesamiento de la cadena de bloques en forma de transacciones. La implementación del correspondiente *data-access-object* (DAO) tendrá la función de obtener los datos desde la base de datos (BlockDB) a en la cual escribe *librustzcash* cuando escanea bloques.

4.5.3. Capa Núcleo: interconexión FFI con LibRustZcash

motivación

Las wallets se están expandiendo a muchas plataformas y tipos de lenguajes de programación, no obstante esto conlleva a la situación de requerir algoritmos que no solo no están implementados en el lenguaje de programación que se está utilizando, sino que su implementación es un proyecto si mismo.

¿Cuántos desarrolladores estarían interesados en formar parte de una comunidad para desarrollar un circuito que genere una prueba de Cero-Conocimiento en Swift,

[27]La interfaz GRPC de Lightwalletd tiene métodos sincrónicos y asincrónicos

Kotlin, Objective-C o Java?

Zcash presenta fenómeno interesante a este tipo de situaciones. Donde la comunidad de desarrolladores se centra en desarrollar en un lenguaje como Rust, que permite la compilación del código para distintas arquitecturas y la generación de interfaces de programación foráneas (FFI) para distintos lenguajes de uso más masivo como C, C++, Java, Swift, Javascript, etc.

Esto hace posible que no sea necesario concentrar esfuerzos en desarrollar implementaciones especificas para las distintas arquitecturas de los clientes livianos, sino que todo el trabajo central que corresponde al dominio de aplicación del protocolo Zcash se encuentre programado en Rust y reutilizado por medio de FFI. De esta forma cuando se dan cambios en el protocolo de Zcash estos se implementan y verifican en una sola base de código común.

Esto último es muy importante por el hecho de que en sistemas descentralizados como las criptomonedas, todos aquellos nodos o clientes que se aparten del protocolo de consenso tienen la potencialidad de generar un «*chain fork*»: una versión paralela e irreconciliable de parte del ecosistema respecto de la implementación del protocolo de consenso. Los efectos no deseados de un evento así son complejos, vale remarcar que incluyen la potencial pérdida de fondos para aquellos usuarios que queden el lado equivocado de la cadena.

A los efectos de integrarse de forma segura y sustentable al ecosistema se eligió ir por la alternativa de utilizar una interfaz FFI hacia varios módulos de Rust (Crates) agrupados en el proyecto "LibRustZcash" que proveerá todas las funcionalidades que son parte del núcleo del ecosistema Zcash. [28]

Estructura

Para poder utilizar cualquier módulo de Rust en una aplicación móvil, es necesario crear una interfaz de funciones foráneas (FFI) utilizando una herramienta llamada CBINDGEN, que genera una interfaz en base a los parámetros de las funciones que están señaladas en el código fuente de Rust como funciones de FFI.

Cabe aclarar que no es el código nativo del cliente liviano quien hace referencia a los módulos Rust en cuestión (en este caso LibRustZcash) sino que se requiere crear un pequeño módulo (también en este lenguaje) que implemente las funciones que serán expuestas en la FFI específicamente. Este módulo intermedio es quien declarará como dependencias los módulos que se desean invocar ulteriormente desde el cliente liviano, haciendo de mediador entre ambos ambientes.

Esta no es una tarea trivial ya que es en este código intermedio donde se tienen que entre otras cosas, conciliar el manejo de memoria de un lenguaje y otro (Garbage collector de JVM para Android o ARC en el caso de iOS).

Las funciones de una FFI tienen ciertas limitaciones, por ejemplo, los tipos de datos que pueden retornar y recibir. Por ello el diseño del SDK de Zcash contempla el uso de tipos de datos primitivos y vectores de bytes para el caso de rutas de archivo y cadenas de caracteres, delegando la persistencia y lectura de datos compuestos a dos

[28] Estos se encuentran detallados en el Anexo de Requerimientos funcionales y no funcionales, comprenden los requerimientos R14 a R22 y R23 a R34.

bases de datos SQLite3: un cache de para bloques compactos («CacheDB») y una base de datos donde se guardarán los datos desencriptados de ese cache («DataDB») [29].

El SDK tiene que arbitrar el uso de estas bases de datos de forma tal de garantizar que no hayan lecto-escrituras concurrentes (SQLite3 no las soporta). Para minimizar este escenario, el cliente liviano, encargado de descargar los bloques compactos al caché escribirá dicha base de datos para que estos sean leídos por el módulo en Rust, quien escribirá los datos desencriptados utilizando las claves de visualización provistas en la base de datos «BlockDB» para que sean leídas por el cliente liviano por medio de las interfaces data access objects del SDK.

Interfaz FFI y Código

Pese que en los diagramas se muestra como algo directo, existen varios niveles de indirección para los llamados que se hacen de una aplicación cliente del SDK hacia un módulo de LibRustZcash.

En primer lugar existe una interfaz nativa, que conecta el lenguaje nativo del cliente liviano con el lenguaje puente que genera la herramienta cbindgen para el módulo rust intermediario. Para el caso de iOS, se implementa la interfaz *RustWelding* que convierte los tipos de datos de Swift a los requeridos por el Header en C de la intefase FFI generada. Algo equivalente se da en el caso de Kotlin y la interfaz Java Native Interface generado por *cbindgen*. Una vez invocada la interfaz generada, esta ejecuta la función foránea perteneciente al codigo Rust alojado en el repositorio del SDK de Zcash, que convertirá esos datos primitivos en Java o C en estructuras de datos de Rust y llamará las funciones pertinentes en LibRustZcash. La figura 4.19 muestra la interfaz que expone esta interconexión.

Si bien la reutilización de código entre plataformas es alto, existe esa capa intermedia de código Rust que media lidia con JNI y C que debe mantenerse para cada cliente liviano. Si en un futuro se implementara un cliente liviano en Web Assembly, también existiría una capa intermedia a tales efectos en el SDK en cuestión.

Lo ideal en este caso lograr que esta capa intermedia sea lo más fina posible conteniendo solo la lógica necesaria para la conversion JNI a Rust o C a Rust.

Intercambio de datos mediante SQLite

Una de las limitaciones de utilizar una base de datos local como medio de intercambio de aquella información derivada de las transacciones detectadas a la hora de desencriptar bloques compactos con las claves proporcionadas por el usuario es la duplicación del código que implementa el este DAO. Lo ideal sería lograr una API código nativo-Rust que tenga la capacidad de delegar unilateralmente la potestad de esta información a uno de los dos "lados". Si estos datos se manejaran únicamente del lado del cliente, se obtiene como resultante un componente FFI sin estado interno (stateless), más ligero y adaptable. Como contrapartida, la implementación del lado nativo de la solución (para el caso, Swift, Java o Kotlin) tendrá el agregado de cargar

[29] Parte del trabajo a futuro en materia de utilización de Rust como lenguaje multi-plataforma para aplicaciones móviles, es el mejoramiento de este aspecto que permita interfaces de programación más ricas y expresivas entre la parte nativa y la multi-plataforma.

Figura 4.19: Interfaz Rustwelding, que implementa la utilización la interfaz FFI con Rust

responsabilidad sobre el manejo y de estos datos, compartiendo parte de la lógica de dominio de aplicación del protocolo Zcash pero pudiendo tener mayor flexibilidad al escoger el modo en el cual se implementará el almacenamiento, gestión y provisión de los datos que se requieran al momento de realizar operaciones desde el modulo implementado en Rust.

A la inversa, si el módulo Rust condensara todas las responsabilidades sobre la información derivada (creación, almacenamiento y provisión), esto haría la interfaz FFI más compleja, teniendo que proveer un manera de comunicar tipos de datos compuestos en lugar de los primitivos se requieren para informar el resultado de operaciones cuyos resultados se impactan en una base de datos. A su vez acompañaría un incremento en el tamaño y complejidad propia del módulo SQLite de LibRustZcash, quien manejaría tanto el estado como los datos como la provisión directa de los mismos. Convirtiendo la implementación de un *TransactionManager* en un mero puente hacia la FFI, sumado a las posibles adaptaciones que se requieran a nivel estructuras de datos, ciclo de vida de las entidades creadas y manejo de memoria.

El mayor impedimento para lograr cualquiera de estas dos implementaciones descritas anteriormente está en la dificultad de la interfaz FFI generada por *cbindgen* en base el código intermediario escrito en Rust. Es por ello que frente a esa limitación, surge la necesidad e utilizar la base de datos SQLite como una forma de intercambio de datos en un repositorio probado, estructurado y disponible en gran cantidad de plataformas.

Diagrama de clases

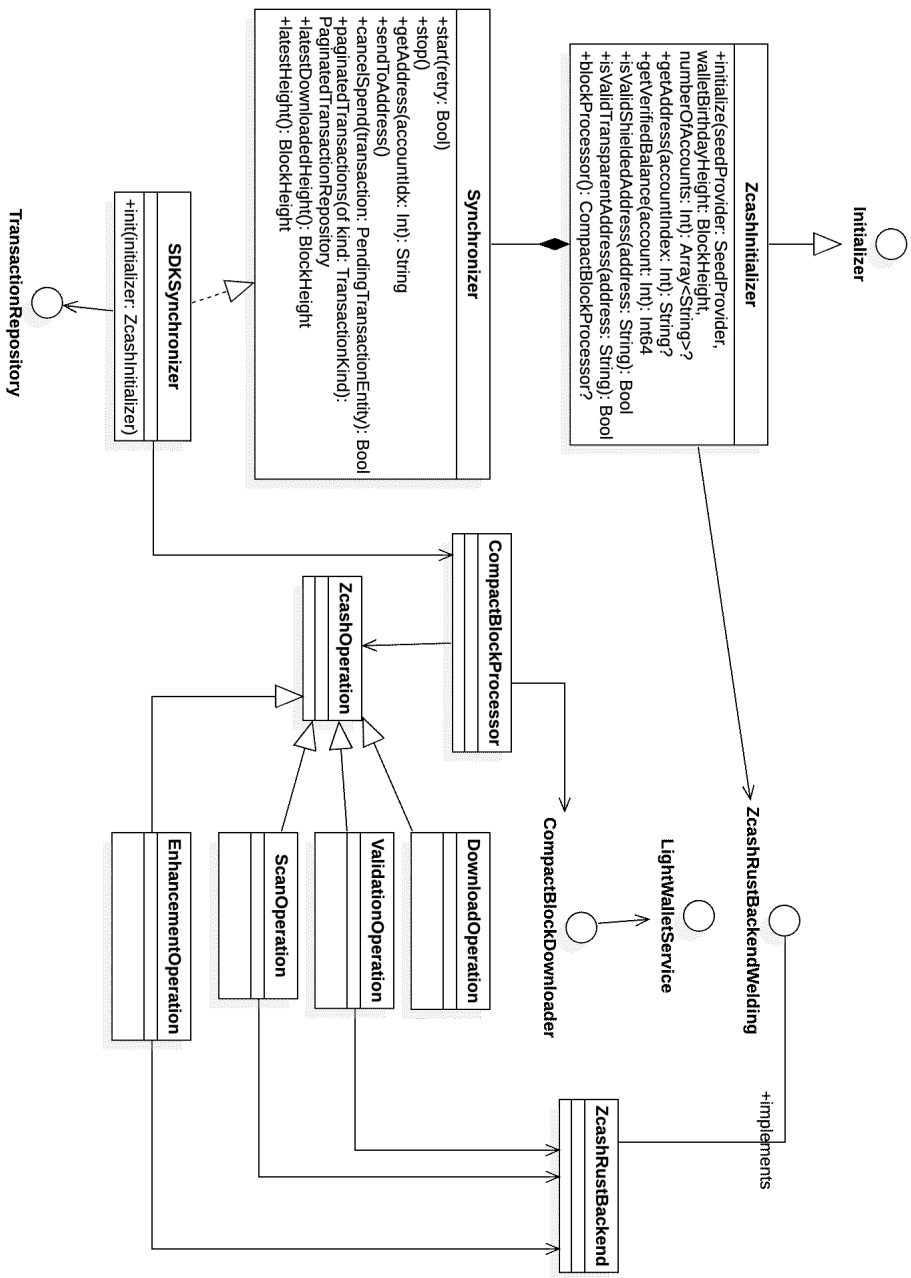


Figura 4.20: Diagrama de clases simplificado del SDK de Zcash para iOS

4.6. kit de desarrollo de Monero

Monero proporciona un kit de desarrollo en varios lenguajes, siendo el más completo implementado en C++ llamado "Monero-cpp"[30]. Este SDK contiene funcionalidades tanto para clientes livianos y aplicaciones de escritorio que expongan funcionalidades de "Full-Node".

A su vez, el repositorio de código del nodo full node [60], contiene una sección denominada "libwallet" donde se exponen clases y estructuras de datos orientadas a los requerimientos de las billeteras Monero. Aplicaciones como Monerujo [43] o Cake Wallet [42] utilizan un "Wrapper" mediante las interfaces FFI de los lenguajes de programación que utilizan (Java y Dart respectivamente) con la cual encapsulan la funcionalidad desarrollada en C++ de *libwallet* expuesta en el encabezado "wallet2_api.h"

Surge como interrogante si este tipo de implementación es generalizada para las aplicaciones cliente de Monero. Por lo cuál, se realizó una búsqueda del término "wallet2_api.h" en todo el repositorio de software de código abierto Github. La búsqueda retornó mas de 5.300 resultados en varios centenares de proyectos y forks.

Las herramientas de desarrollo de Monero presentan una estructura mas monolítica que las expuestas por Zcash. Esto no presenta un inconveniente en sí para el diseño propuesto, sino que requiere un enfoque dedicado a adaptar las interfaces públicas de estas herramientas al modelo "Inicializar y Sincronizar" propuesto en este trabajo.

4.6.1. Capa superficial: Inicializar y Sincronizar

Una cuestión a observar es la diferencia entre la inicialización concreta que requieren las diversas criptomonedas, lo cual depende de la implementación de las herramientas que implementan el protocolo. Para el caso de Monero, tanto la API full node como la de wallet liviana incluyen diferentes métodos para crear o restaurar una wallet. Estos se resumen en:

- Creación mediante frase semilla de 25 palabras
- Creación mediante claves ya derivadas
- Restauración mediante contraseña local

Estas particularidades pueden aislarse en el objeto de inicialización (Initializer), permitiendo que todas las cuestiones que hacen a las generalidades que se observan en las historia de usuario detalladas en en las sección 4.4 puedan expresarse directamente en la interfaz del sincronizador (Synchronizer)

4.6.2. Capa Núcleo: interconexión FFI con "wallet2_api.h"

Motivación

La proliferación de "wrappers" en cada una de las wallets para los distintos lenguajes de programación (Java, Swift, Kotlin, Dart, etc.) en miles de proyectos da

[30] de acuerdo con la documentación presentada por los desarrolladores de Monero

indicios de una gran adopción de esta interfaz pública en los proyectos que implementan wallets Monero. Tal como ocurre en el caso de Zcash, implementar un protocolo de consenso enteramente dedicado a una plataforma o lenguaje de programación requiere de grandes recursos. Por ello resulta conveniente la implementación de estas interfaces intermedias que permiten la implementación en múltiples plataformas mediante interfaces FFI bien definidas.

Estructura

La interfaz pública de "libwallet"[31] condensa todos los requerimientos que se detallan en la sección de épicas e historias de usuario en la sección 4.3. Esto no significa que se esto deba trasladarse la interfaz pública de un kit de desarrollo para un lenguaje determinado. Sino que puede estructurarse el código de forma tal que se respete una estructura común utilizando el patrón de diseño Adapter.

Tomando como referencia los repositorios de Cake Wallet [42] y Monerujo [43], se hace evidente que se requieren módulos FFI que mantengan el estado de los objetos inicializados por API en C++ y expongan los métodos, estructuras de datos y callbacks a la capa superior en el lenguaje nativo de la plataforma. En el caso de Cake Wallet esto se hace a nivel Plugin de la plataforma Flutter[32] donde cada plataforma (Android o iOS) debe implementar el plugin en su lenguaje nativo (Java o Kotlin para Android o Objective-C o Swift para iOS). La figura4.1 ilustra la implementación en esta versión multi-plataforma, comparativamente, mientras que la integración de 'libwallet' en Monerujo se hace a nivel interfaz JNI o de disponer una versión para iOS se haría con una clase que encapsule los llamados a la FFI de C++. Este esquema se encuentra presente en la implementación original de Cake Wallet para iOS, de la cual se halló una copia en un repositorio de Github que data del año 2018 [61], donde se implementa una interfaz Objective-C++ que contiene las referencias hace la API C++ de Monero, adapta los tipos de datos y los callbacks, que se ilustra en la figura 4.2.

Intercambio de datos

A diferencia del modelo planteado en Zcash. Las APIs que exponen las herramientas de Monero manejan los datos unilateralmente y el intercambio de datos se produce a través de dichas interfaces, a excepción de algunos elementos puntuales como las transacciones generadas por la propia wallet, donde la firma de una transacción puede estar disociada de su creación, permitiendo que dicha firma se realice en otro dispositivo (una hardware wallet o una cold wallet).

Diagrama general propuesto

En la figura 4.21 se aporta un diagrama que ilustra la estructura de un SDK para Monero que implementa la estructura de Inicializado + Sincronizador que utiliza el

[31] el encabezado "wallet2_api"

[32] Flutter es una plataforma dedicada a realizar aplicaciones multi-plataforma para iOS, Android y de escritorio donde emplean un lenguaje de programación común, llamado Dart y un runtime propio que se corre encima de cada plataforma buscando abstraer las particularidades de cada una.

SDK de Zcash

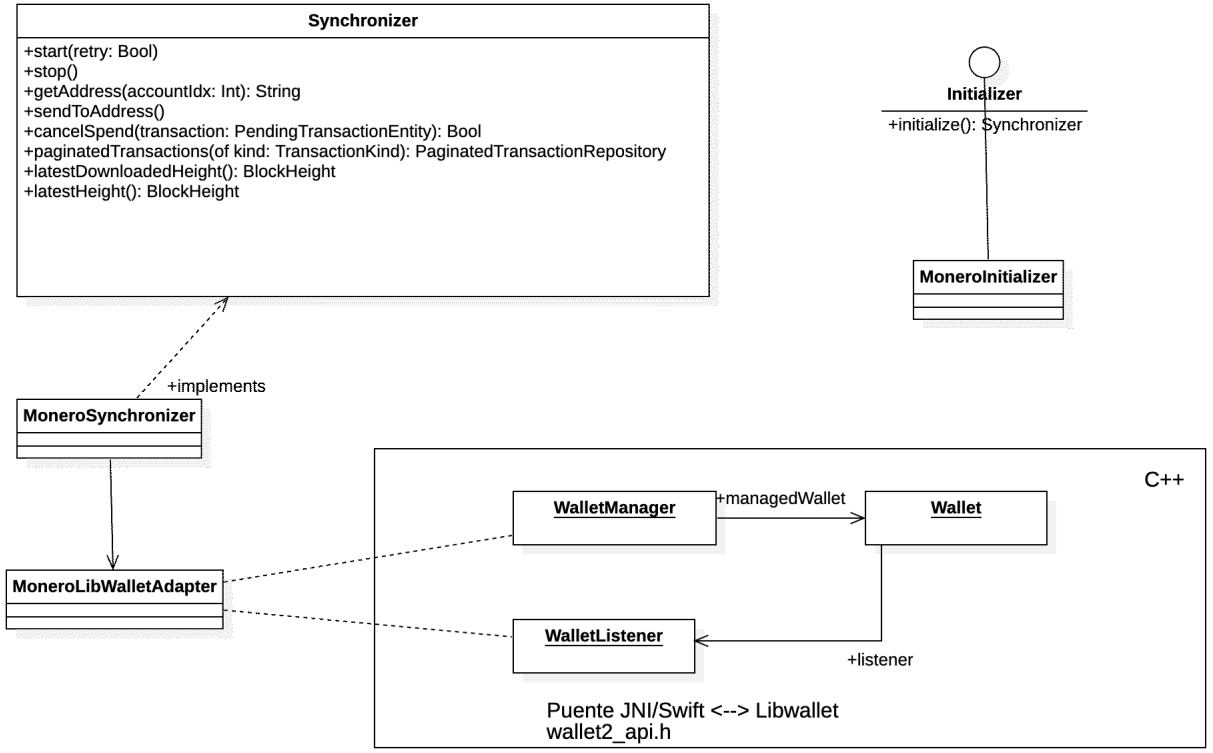


Figura 4.21: Diagrama simplificado del SDK propuestos para Monero

Capítulo 5

Arquitectura de Referencia

5.1. Introducción

En el capítulo 2 se describieron a grandes rasgos los conceptos que introducen al uso básico de criptomonedas Proof-of-Work utilizando el caso de Bitcoin como base conceptual e introduciendo las similitudes y diferencias que existen a nivel de usuario para una criptomoneda con privacidad en transacciones como Zcash.

Este capítulo describe la arquitectura de referencia que surge del trabajo de análisis y relevamiento de requerimientos de los kit de desarrollo de Zcash y Monero para clientes livianos Android y iOS, y las distintas wallets que los utilizan que fue cubierto en el capítulo 4. Allí se realizó un relato enumerando Temas, Épicas, Historias de usuario haciendo referencias a requerimientos funcionales y no funcionales, resaltando entidades y componentes importantes para la propuesta que se realizará a continuación.

El diseño a abordar es el de un cliente liviano que implementa funcionalidades esperables de una wallet. Lo integran distintos componentes que satisfacen los requerimientos ya mencionados.

5.2. ¿Arquitectura o Framework?

Esta interrogante habita al autor desde los orígenes del trabajo como probablemente a sus lectores. A lo largo de los capítulos anteriores se ha presentado la problemática del dominio de aplicación, sus requerimientos y desgranado los componentes que cubren esos requerimientos de mayor a menor. Si bien el título de la propuesta para este trabajo indica como objetivo hablar de una arquitectura no es el espíritu de su autor evadirse a las preguntas incómodas. Ante la necesidad de encuadrar todo este trabajo en categorías conocidas, surge esta cuestión. ¿Está el lector ante el planteo y descripción de una Arquitectura de Aplicación, o de un Framework? ¿Es posible acaso encuadrarlo en alguna de las dos categorías?

Para ello es necesario recurrir a las definiciones fundamentales de estos conceptos y definir los términos mediante los que se intentará catalogar el trabajo.

El libro “Patrones de Diseño: Elementos de software orientado a objetos reutilizable” [52] realiza en sus páginas iniciales una definición taxativa de lo que hace que una pieza de software sea considerada un framework:[1]

> Un framework es un conjunto de clases que cooperan entre sí construyendo un diseño reutilizable para una clase específica de Software [..]. Un framework se personaliza a una aplicación específica mediante la creación de subclases específicas de clases abstractas del mismo. El framework dicta la arquitectura de tu aplicación. Definirá la estructura general, y su partición en clases y objetos, las responsabilidades claves, cómo estas clases y objetos colaboran y el hilo de control. Un framework predefine estos parámetros de diseño para que el desarrollador de la aplicación pueda concentrarse en lo específico de su aplicación. El framework captura las decisiones de diseño que son comunes a su dominio de aplicación, enfatizando la reutilización del diseño por sobre la reutilización del código, empero un framework usualmente incluye usualmente subclases que pueden ponerse a trabajar inmediatamente. Tal reutilización implica una inversión del control. Cuando se utiliza una librería, se escribe el cuerpo principal de la aplicación y se llama al código a reutilizar. Cuando se utiliza un framework, reutilizas el cuerpo principal de la aplicación y escribes el código que él -framework- *utilizará*. Aún así tendrás que escribir operaciones con nombres y convenciones particulares, pero ello reduce las decisiones que deberás tomar.[..] Como las aplicaciones son tan dependientes del framework para su diseño, son particularmente sensibles a los cambios en las interfaces del mismo. A medida que el framework evoluciona las aplicaciones deberán evolucionar con él

La definición de arquitectura de software, por si sola podría ser un trabajo en sí mismo y de hecho lo ha sido siendo que diversos autores han tratado el tema y dado definiciones, alcances e implicaciones del término. En un elogio de la brevedad, se recurre al texto ubicado en la Sección 3.5 del libro “Code Complete” de Steve McConnell en su segunda edición. Si bien esta bibliografía se define como un libro sobre construcción de software y no sobre Arquitectura de Software, condensa en su interior literatura fundamental al respecto de la cual se desprende la siguiente definición:

> La arquitectura de software es la parte de alto nivel del diseño de software, el marco que sostiene las partes más detalladas del diseño [..] Una arquitectura bien pensada provee la estructura necesaria para mantener la integridad conceptual de un sistema...

McConnnell al final de cada sección ofrece a los lectores una «checklist» de elementos a tener en cuenta y cuestiones esenciales donde para el caso el objetivo final de este cuestionario que McConnell presenta a sus lectores tiene como objetivo evaluar si una propuesta dada ha reparado en las cuestiones clave para ser presentada ante un equipo de programación. Puede que todas las preguntas que ese autor arroja no apliquen a todos las propuestas de arquitecturas de software pero sin lugar a dudas,

[1] El autor no dispone de la edición en Español de este libro, por tanto, ha realizado a su mejor saber y entender una traducción del texto en su edición en Inglés.

al justificar por qué no han de aplicarse en un caso puntual sirve como ejercicio de reflexión. En la que respecta a calidad general de una propuesta, también tiene una lista de interrogantes a responder.

5.2.1. Lista de tópicos generales a cubrir por una propuesta de arquitectura de software

¿Es clara la organización general del programa, incluyendo información general adecuada y su justificación?

A lo largo del trabajo se han desarrollado el contexto y la justificación del problema a abordar, y en las secciones siguientes se hará referencia a la organización general en un sentido mas concreto.

¿Están bien definidos los elementos clave incluyendo sus áreas de incumbencia y sus interfaces con otros elementos clave?

Es posible también tildar esta casilla de la lista haciendo referencia a las secciones 4.3 y 5.3.

¿Están todas las funciones listadas en los requerimientos cubiertas en forma sensata no por pocos o por muchos elementos clave?

la sección 4.3 combina requerimientos e historias de usuario haciendo referencia a los componentes clave en base a ellos.

¿Se encuentran las clases más criticas descriptas y justificadas?

Si, el capítulo 4 y las secciones siguientes cumplen con este objetivo.

¿Se encuentra definido el diseño de los datos?

En particular el diseño de los datos se encuentra delegado a la definición que se haga en el protocolo de cada criptomoneda. Uno de los puntos clave del modelo propuesto es la puesta en común de funcionalidades y requisitos no funcionales posiblemente dispares en lo específico pero similares en lo general.

¿Se encuentra especificada la organización y el contenido de la base de datos?

En este caso puede considerarse que la base de datos principal de una wallet es en definitiva la blockchain y por tal, se encuentran definidas por su protocolo de consenso.

¿Se encuentran todos las reglas de negocio clave identificadas y su impacto en el sistema definido?

En particular se describen las reglas de negocio en forma de requisitos e historias de usuario en las secciones 2 y 4, a su vez cada criptomoneda

tiene estas definiciones formales en sus protocolos a los cuales hace referencia este trabajo.

¿Existe una descripción de la estrategia para la interfaz de usuario? ¿Se encuentra la interfaz de usuario modularizada de tal forma que sus cambios no afectarán otras partes del programa?

Las interfaces de usuario se encuentran fuera del alcance de este trabajo, sin embargo en los casos de estudio tomados (ver sección 5.5) se observa cómo este tipo de esquema favorece la utilización de distintos paradigmas de capas de presentación.

¿Existe una estrategia de I/O descripta y justificada?

No es el objetivo del trabajo profundizar sobre esta cuestión en lo puntual, sí para el caso de abordar la necesidad de una estrategia de datos que se adecue a los recursos de los clientes livianos, que se encuentra cubierta por el anexo 7.5.

¿Se encuentran descritos los estimativos y la estrategia para el uso de recursos escasos y su administración como hilos de ejecución, conexiones a bases de datos, ancho de banda, etc?

Este punto no se encuentra entre los objetivos del trabajo.

¿Existe una descripción de los requerimientos de seguridad de la arquitectura?

En la definición del alcance del trabajo se refiere puntualmente a la exclusión de este área de la problemática, describiendo antes un panorama general de los desafíos en la materia para este tipo de clientes y dejando un apartado de la sección de trabajo a futuro 6.2.5.

¿Define la arquitectura los presupuestos logísticos y temporales para cada clase, subsistema o funcionalidad?

La ejecución de una posible implementación no esta en discusión en el trabajo.

¿Describe la arquitectura como se logrará escalabilidad?

Aspectos en esa materia se describen en la sección de trabajo a futuro.

¿Se refiere la arquitectura a la interoperabilidad?

La propuesta es en su esencia un trabajo abocado a la interoperabilidad. No dependen de él ni sistemas operativos ni implementaciones particulares de nodos para las AEC que se discuten en él.

¿Se encuentra definida una estrategia de localización / internacionalización?

Este asunto no está abordado en el trabajo por considerar la interfaz de usuario como algo independiente y fuera del alcance de la propuesta.

¿Hay una estrategia coherente para el manejo de errores?

En la sección 5.3.6 se trata esta cuestión considerando que los errores en este tipo de ecosistemas tienen múltiples orígenes y causales que deben ser condensados de una manera asequible y comprensible para los desarrolladores de las aplicaciones cliente asumiendo que ordenando dicha complejidad se logrará trasladar esa misma noción hacia el usuario final.

¿Se requiere una estrategia de tolerancia a fallos, y en caso positivo, existe?

Por su naturaleza, la tolerancia a fallos viene dada por la naturaleza misma de las redes de pares que conforman las criptomonedas. A nivel del cliente los datos cruciales se delegan a subsistemas de los sistemas operativos destinados a tales fines. La prioridad está puesta en evitar la pérdida de fondos mediante la preservación de las frases semilla o claves privadas derivadas de estas.

¿Se ha establecido la factibilidad técnica de cada parte del sistema?

Los casos de estudio abordan ciertas implementaciones similares a la propuesta en la sección 5.5.

¿Se ha definido un tratamiento de la sobre-ingeniería?

La propuesta no pone demasiado énfasis en las implementaciones como para marcar convenciones respecto de este asunto.

¿Son necesarias y se encuentran incluidas las decisiones de "comprar vs. construir"?

Este punto no aplica para el caso del presente trabajo pues su origen es la construcción misma.

¿Describe la arquitectura como el código reutilizado se conformará y contribuirá a otros objetivos de la arquitectura?

La propuesta es en definitiva una puesta en común de requerimientos funcionales y no funcionales que hace énfasis en la resutilización de código estratégico y núcleo de cada proyecto. En la secciones 4.5 y 4.6 ilustra como desde sus particularidades los kits de desarrollo de Zcash y Monero respectivamente constribuirían a la propuesta general.

¿Acaso el diseño de la arquitectura contempla cambios futuros probables?

La propuesta está confeccionada desde la base de que su dominio de aplicación se encuentra en constante evolución, donde no hay estructuras osificadas ni cuestiones "intocables". Cualquier criptomoneda podría en definitiva cambiar la raíz de su funcionamiento (como por ejemplo pasar de Proof-of-work a Proof-of-stake) y la afectación a la propuesta debería ser desde el punto de vista de un desarrollador de wallets, poco significativa.

5.2.2. Cuestionario de Calidad general de una arquitectura

¿Considera la arquitectura propuesta todos los requerimientos?

El autor considera que se ha realizado un trabajo de relevamiento exhaustivo para el alcance del trabajo.

¿Se encuentra alguna de parte de la arquitectura sobre-diseñada o infra-diseñada? ¿se encuentran definidos explícitamente los requerimientos para dichas partes?

El trabajo hace referencia en la definición de su alcance a las partes en las que profundizará y las que excluirá justificando cada caso.

¿Encuentra que la arquitectura definida se sostiene conceptualmente como un todo?

Sin pretender realizar el trabajo de los eventuales evaluadores del mismo, el autor considera que la propuesta tiene consistencia conceptual y argumental.

¿Es el diseño de alto nivel independiente de la máquina y el lenguaje en el cual se implementará?

El motivo del trabajo es presentar una propuesta para un tipo de cliente en base a sus capacidades técnicas (cliente liviano) independientemente de plataforma, lenguaje o sistema operativo.

¿Se encuentran definidas las motivaciones para las principales decisiones tomadas?

En la sección 1 se describen los motivos principales del trabajo y luego a lo largo del trabajo se argumentan las decisiones tomadas en base a los requerimientos encontrados y los objetivos del trabajo.

¿Se encuentra usted, como programador dedicado al desarrollo del sistema, conforme con la arquitectura a implementar?

Como se detalla en la sección de conclusiones finales (ver 6.1), este trabajo busca ser aquel que su autor hubiera deseado encontrar luego de que le

encomendaran el trabajo de crear una Wallet para una «Anonimity Enhancing Cryptocurrency».

En respuesta a la pregunta de esta sección, podría afirmarse que la propuesta tiene muchos puntos en concordancia con la «checklist» que propone McConnell. Sin embargo, conceptualmente también podría afirmarse de igual manera sobre la definición que el «Gang-of-Four» propone para delimitar las características de un framework. En conclusión, el origen de la propuesta es delinear los requerimientos de una arquitectura para billeteras electrónicas para AECs, aunque ello no quita que de ellos pueda derivar la creación de un Framework que provea el marco técnico para la adopción de distintas criptomonedas del estilo.

5.3. Arquitectura de Referencia: Estructura general

En el capítulo 3 se describe la revisión sistemática de la literatura en relación en lo pertinente a clientes livianos de criptomonedas con privacidad. El capítulo 4 muestra un relevamiento de las principales wallet que soportan *privacy coins*, recolectando los requerimientos e historias de usuario generales y particulares de cada caso. Las conclusiones sobre las cuestiones en común se condensan en las dos secciones anteriores en forma de Temas (sección 4.2), Épicas (sección 4.3) e historias de usuario. En las secciones 4.5 y 4.6 recorren las particularidades de los kits de desarrollo de Zcash y Monero en particular, donde se ilustran sus implementaciones y para el caso de Monero se propone una estructura general para un kit de desarrollo en línea con la propuesta de este trabajo.

A lo largo del trabajo se han presentado varios tipos de billeteras y sus implementaciones en distintos Sistemas Operativos, mediante diferentes tipos de frameworks de presentación como los nativos de los sistemas Android y iOS o híbridos como Flutter o React Native. A su vez en el capítulo 5.5 se describen distintos casos de estudio donde las aplicaciones se implementan con distintas arquitecturas de aplicación como Model View Controller, Model View Presenter o Clean Code Architecture. La presentación de la arquitectura de referencia no refiere a ningún sistema operativo, plataforma o arquitectura de capa de presentación en particular, sino que apela a figuras abstractas que estos exponen a los desarrolladores, utilizándolas como referencia del punto de partida a ella.

En la figura 5.1 se diagrama la arquitectura de referencia propuesta de forma general. A la izquierda se detallan los servicios del sistema operativo que se utilizan directamente [2]. En el centro se ubica el objeto delegado de la aplicación concreta que el sistema operativo expone a los desarrolladores. Alrededor de él se encuentran los distintos componentes que se han ido identificando a lo largo del trabajo y la relación entre ellos y la aplicación cliente. En las secciones siguientes se describirá en detalle el diagrama elemento por elemento. A modo de consulta se desarrolla también la sección 5.4 donde se describe puntualmente cada uno de ellos de forma individual incluyendo su intención, motivación, aplicabilidad, estructura, otras entidades participantes, colaboraciones, consecuencias de su implementación, código de ejemplo, usos y temas relacionados, evocando un patrón conocido a los lectores del libro “Patrones de Diseño” [52].

[2] se excluyen los servicios de red dado que se da por entendido que el acceso a la blockchain es siempre mediante éstos y resulta una aclaración innecesaria para el diagrama

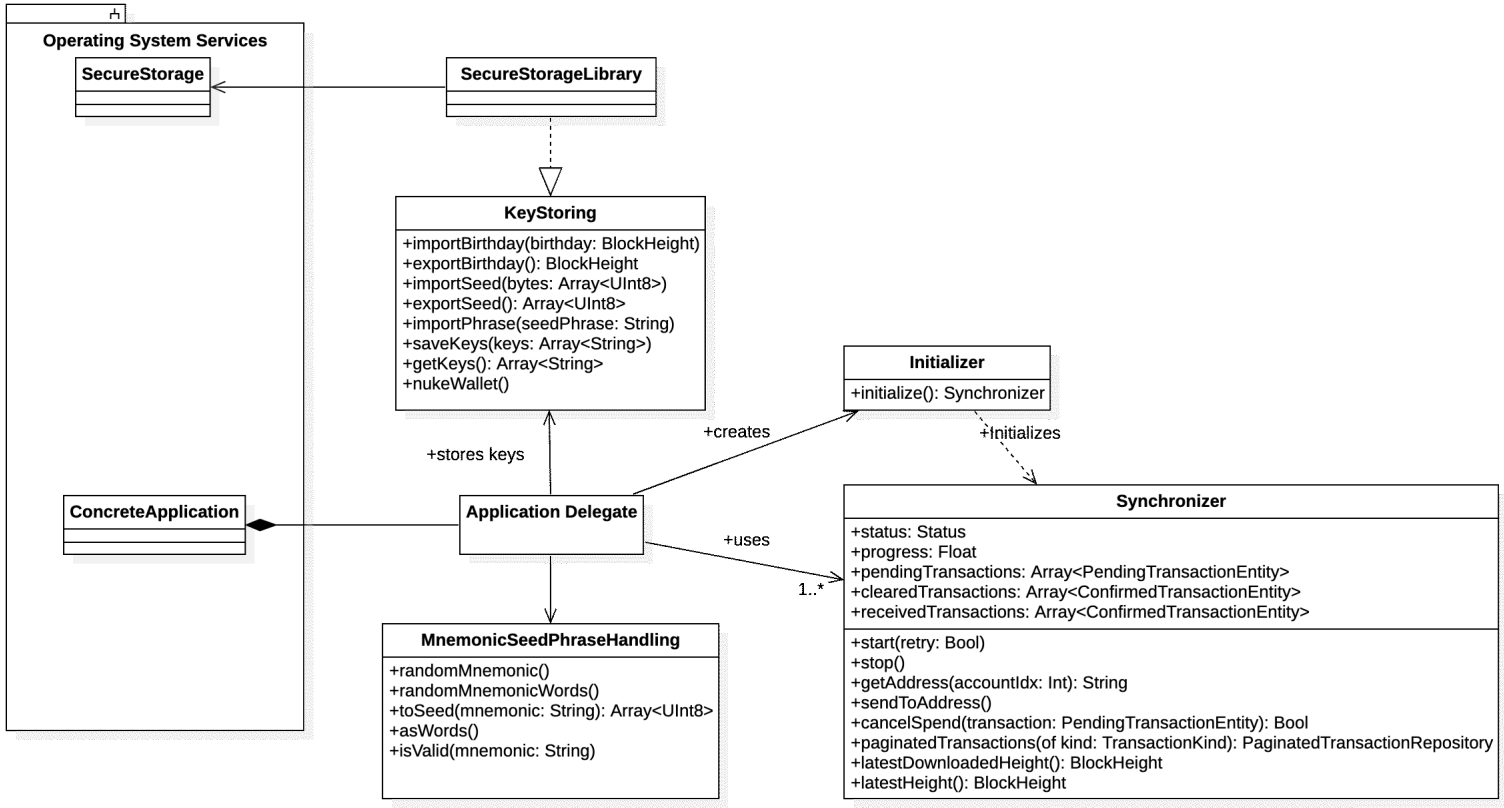


Figura 5.1: Diagrama general de la Arquitectura de referencia propuesta

5.3.1. Punto de ingreso: delegación del Sistema Operativo a la aplicación

Comienza por el centro, donde se puede observar el "Application Delegate" que hace referencia a aquel objeto o punto en el código fuente que un framework de aplicaciones móviles expone a los desarrolladores como el punto de inicio. También refiere al objeto en el cual el desarrollador de la aplicación utiliza como referencia primaria de la aplicación concreta y el administrador de procesos del sistema operativo. Este objeto es la representación visible de la aplicación a los ojos del sistema operativo, y en él se delega todo lo que concierne al ciclo de vida de la aplicación para que el desarrollador, de ser necesario, programe el comportamiento de la aplicación para los eventos de sistema y las fases del ciclo de vida de las aplicaciones corren en él. Se asume que el objeto "Application Delegate" no puede existir sin la aplicación concreta y que al dejar de existir, todos los recursos a los que este refiere, dejan de estar en memoria y ser utilizados en el ambiente de ejecución del sistema operativo. También se asume que todas las referencias que este objeto tiene, serán retenidas y mantenidas en memoria por el tiempo en el que el objeto también lo esté.

5.3.2. Acceso a las claves del usuario

Este objeto tiene la referencia a la interfaz "KeyStoring" la cual tiene como objetivo principal almacenar las claves del usuario de forma encriptada utilizando un almacenamiento seguro. La propuesta hace referencia al provisto por el sistema operativo, pero podría ser el caso que la aplicación utilice uno propio, con sus propias claves privadas y métodos de recuperación. Sea cual fuere el caso la interfaz propuesta requiere que la implementación concreta pueda guardar y almacenar de forma encriptada el bloque de almacenamiento, la frase semilla y las claves, para su consecuente recuperación cuando la aplicación lo requiera (por ejemplo para firmar una transacción). A su vez debe poder destruir esta información a pedido del usuario.

5.3.3. Manejo de frase semilla

En relación a las claves, el punto de inicio de la arquitectura propuesta tiene una referencia a la interfaz «MnemonicSeedPhraseHandling», la cual tiene como objetivo la creación, validación y recuperación de frases semillas y los bytes que estas representan. Es importante resaltar que se referencia a la interfaz porque a pesar de que el estándar *de facto* de manejo de este tipo de claves es el utilizado en Bitcoin (BIP-39 [21]) hay criptomonedas como Monero que utilizan un esquema que es representado en forma de frase Mnemónica, la representación de los bytes que derivan de ella y la cantidad de palabras y diccionario pueden ser diferentes a BIP-39. El objetivo es ilustrar que se requiere obtener los bytes desde una frase semilla, dejando de lado las cuestiones particulares de la implementación

Cabe destacar que las frases semillas son sólo una convención sobre la recuperación de los bytes que generan las claves privadas de una wallet. Actualmente existen otros métodos de recuperación estudio, como la "recuperación social" de las mismas, cuya premisa es definir "guardianes" de una wallet. Ellos son quienes tienen una custodia colateral de las claves de usuario, de forma tal que de ser necesario el dueño de la

billetera puede ser asistido por sus guardianes para recuperar una billetera. Argent, una billetera DeFi para Ethereum es la primera en implementar este tipo de recuperación y llevarla a producción. Adicionalmente permite que los usuarios definan ciertas reglas de uso para que en caso de detectar cierto tipo de transacciones "inusuales", la billetera informe a sus *guardianes* de la operación.

Tal como lo expresa Vitalik Buterin, creador de Ethereum en el artículo "Por qué es necesaria la amplia adopción de billeteras con recuperación social" [62], las frases semillas son insuficientes a la hora de preservar las claves de usuario, más allá de ser mas convenientes que manejar bytes hexadecimales, son un punto único de falla, difícil de administrar y que no resisten el paso del tiempo. En la sección 6.2.4 se define una línea de trabajo a futuro respecto de este tema.

5.3.4. Inicialización

Surge del relevamiento que las wallets mantienen una cantidad considerable de estado de forma local. Esta información está respaldada por medios de almacenamiento que van desde archivos a bases de datos locales. A su vez cada criptomoneda tiene una o más conexiones a la red de pares que la conforman con sus respectivos protocolos que, pueden coincidir o no con otros que incluya la aplicación en cuestión en caso de ser multi-moneda. Para el caso de plataformas que utilizan JNI como interfaz nativa con otro lenguaje como C o C++ (es el de las wallets relevadas tanto para Monero como Zcash), las clases tienen que ser cargadas en tiempo de ejecución bajo demanda.

Todas estas responsabilidades y roles son llevadas a cabo por distintas clases y componentes que conforman un grafo de objetos cuya complejidad es considerable y debe ser administrada. Por ello la propuesta que aquí se expone cuenta con un actor llamado *Inicializador*, a quien se delega la responsabilidad de instanciar los distintos objetos que se requieran en la implementación interna. Su fin es encapsular las complejidades inherentes de los detalles de implementación de cada kit de desarrollo. Esto le proporciona al desarrollador de la aplicación cliente una interfaz simplificada, que expresa sus requerimientos en forma de parámetros en su constructor o en un método de inicialización. Además allí puede contenerse una serie de errores característicos del manejo de recursos del dispositivo o el sistema operativo como lo son archivos, colas de ejecución, creación de hilos, clases y librerías dinámicas. Lo que permite ofrecer una API de errores más concisa que se abstraiga de los detalles de implementación.

Esto último es muy importante. En el ámbito de las criptomonedas es común que lo único que se mantenga inmutable sea la cadena de bloques ya creada y todo lo demás a su alrededor se encuentre en constante ebullición y cambio. Estas tecnologías son muy jóvenes y dinámicas. Se encuentran sujetas a cambios radicales en cada actualización y por ello es deseable encapsular todas estas peculiaridades y mantener cierta estabilidad de cara a los desarrolladores de aplicaciones cliente. Sobre todo en las wallets multi-moneda, donde conviven muchas criptomonedas y la conveniencia de juega un factor importante a la hora de evaluar la incorporación de cada una de ellas al producto. Este punto surge del análisis del caso de estudio "Unstoppable Wallet" que se describe en el capítulo 5.5.

El objeto *Initializer* tiene como función adicional administrar el grafo de dependencias de los objetos que conforman el kit de desarrollo de una criptomoneda. Se

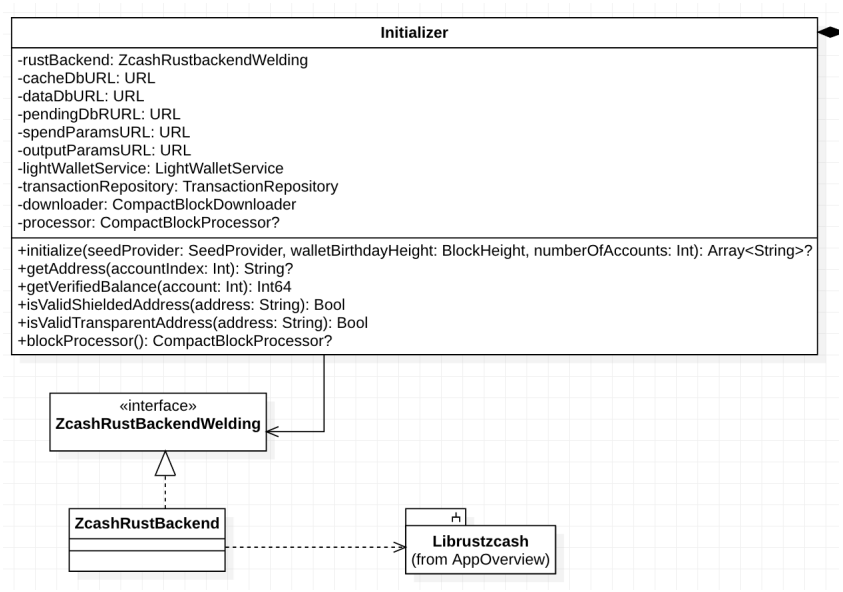


Figura 5.2: Initializer - encapsular la complejidad de requerimientos derivados del protocolo Zcash

puede observar que hayan clases públicas dentro de estos objetos que incluyan constructores que tengan al *Inicializador* como parámetro de entrada. El Sincronizador o el procesador de bloques de Zcash (CompactBlockProcessor) y los respectivos repositorios (TransactionRepository, etc), tienen constructores y métodos factoría[3] que reciben una instancia de *Initializer* por parámetro.

5.3.5. Sincronización con la cadena de bloques

En el capítulo 2 se describe el funcionamiento de criptomonedas de asiento contable (ledger) público o privado. Se puede observar que el concepto al que llamamos billetera o wallet, es una abstracción por sobre la representación real de la información de una cadena de bloques. En bruto, una wallet es un punto de vista de la blockchain, desde un bloque de nacimiento hasta el bloque se que ubica en el extremo más reciente de la cadena. El punto de vista lo definen las claves públicas con las que el usuario desea hacer el recorrido, con los cuales encontrará los elementos que le son propios. Para proponer nuevos elementos (transacciones con inputs y outputs) debe hacer uso de sus claves privadas.

Elementos como el balance y las transacciones, son subproductos de un proceso denominado sincronización que consiste en tomar claves públicas y recorrer la cadena de bloques recolectando (desencriptando) los elementos de interés para las mismas. Disponer de estos elementos es necesario a su vez para garantizar una mayor probabilidad

[3] Ver Factory Methods [52]

de éxito al enviar una transacción a través de la red de pares, pues las transacciones se anclan en los subproductos de la sincronización y conforme el estado de uno de ellos no corresponda al observable desde el último bloque generado en la cadena, la transacción no cumplirá con alguna regla del consenso, y será rechazada[4].

Por esta razón se han condensado las funcionalidades núcleo de una wallet en la Interfaz Synchronizer. Una de las críticas a este modelo podría ser que la clase que implemente dicha interfaz, sería un antipatrón llamado "God Object" (objeto Dios). Una entidad que condensa toda la funcionalidad de un dominio de aplicación en tanto lógica de dominio como estado de la aplicación.

Al volver a recorrer los párrafos iniciales de esta sección se vislumbra un dilema muy propio del dominio de aplicación de las billeteras de criptomonedas: No existe una wallet sin un Sincronizador.

Esta acción es constitutiva pues sin este acto de reconocer los elementos propios dentro de la cadena de bloques, una wallet es solo un par de claves criptográficas.

Es así que la propuesta impone una *interfaz* de sincronización, con atributos y métodos específicos que cumplen con los requerimientos relevados en la sección 4.3.9 que corresponde a la épica de sincronización, pues eso es todo lo importante desde el punto de vista de una billetera para clientes livianos.

Quizás sea el caso que detrás de esta interfaz haya una gran cantidad de interfaces, clases, módulos, submódulos que complementariamente provean todos los requerimientos, o simplemente haya como se relata en la sección sobre el kit de desarrollo de Monero 4.6, un punto de entrada a un componente monolítico detrás de una interfaz FFI.

En la figura 5.3 se ilustra la implementación de esta interfaz en el kit de desarrollo de Zcash. Detrás de ella colaboran muchas entidades, librerías, componentes con el fin ulterior de proporcionar el punto de vista que convierten a un par de claves en una Wallet.

5.3.6. Consideraciones para una estrategia de manejo de errores

En este aspecto cabe mencionar que como aplicación cliente, una wallet non-custodial no escapa a las generalidades que aplican para la parte cliente de una arquitectura Cliente-Servidor donde la contraparte expone errores propios de la llamada *lógica de negocio* (que en este caso seria el protocolo de consenso de la blockchain en cuestión) y errores técnicos (desconexiones, fallas de hardware en alguna parte de la cadena de infraestructura, etc).

Sin embargo, al ejercer la custodia de las claves y sostener un estado local de la blockchain, es posible que se generen errores propios del cliente que interrumpen el normal funcionamiento de la aplicación. De mayor a menor gravedad se propone esta taxonomía de Errores:

- Pérdida de fondos
- Inconsistencia local con inutilización permanente

[4] asumiendo que el nodo con el cual se comunica la billetera no es malicioso y cumple con las reglas de consenso del protocolo

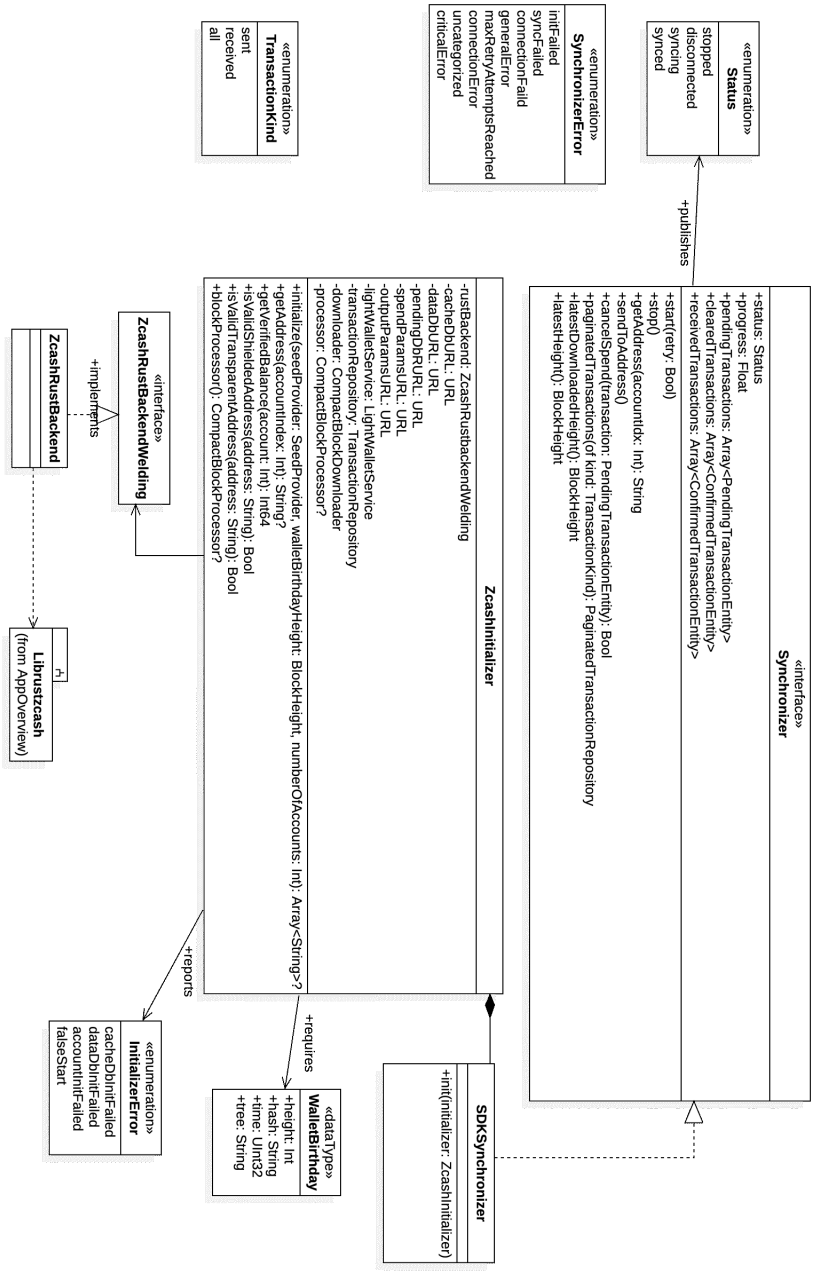


Figura 5.3: Synchronizer - No existe una wallet sin un Sincronizador. Esta acción es constitutiva pues sin este acto de reconocer lo propio dentro de la cadena de bloques, una wallet es solo un par de claves criptográficas.

- Inconsistencia local con inutilización temporal
- Errores técnicos locales recuperables
- Errores técnicos en servidor

Errores de Pérdida de Fondos

Son aquellos que tienen la potencialidad de representar un riesgo de pérdida de los fondos o las claves que dan acceso a ellos. Cabe recordar que el aspecto de no custodia impone grandes responsabilidades tanto sobre los clientes como sobre los usuarios. Se asume que nadie más que estos últimos disponen de las claves privadas que comprueban la potestad de determinados fondos. Dentro de este tipo de errores se encuentran todos aquellos que tengan dentro de sus efectos bloquear el uso de los fondos, derivarlos hacia destinatarios indebidos por error u omisión (y/o mala intención), o bien imposibilitar la recuperación de las claves privadas para su posterior resguardo.

Una regla básica para determinar si se está ante un error de este tipo es si se confirma que el propietario de los fondos se encuentra imposibilitado de gastarlos luego de la ocurrencia del error. Puede hacerse la distinción entre *irrecuperable* y *recuperable*, donde en el primero el control sobre los fondos se pierde indefectiblemente sin importar si el usuario dispone de una copia de las claves privadas a resguardo; y para el segundo tipo, donde la potestad puede recuperarse para aquellos usuarios que disponen de una copia de seguridad de sus claves privadas pero resultaría en una pérdida de fondos en caso de no disponer de ellas y/o no poder acceder a las mismas por cualquier motivo.

Sin importar su origen o causa, si se comprueba lo expuesto anteriormente. Se está en presencia de este tipo de error en alguna de sus dos variantes de severidad.

Errores Inconsistencia local con inutilización permanente

Este tipo de fallas responden a cualquier error que derive en que la aplicación cliente pierda la consistencia de su estado local respecto de la cadena de bloques de forma tal que, aún disponiendo de conectividad, claves apropiadas y sin haber presencia de pérdida de fondos, no se pueda revertir la situación y se requiera de la reinstalación del software previa recuperación de claves y re-sincronización con la cadena de bloques desde el bloque de nacimiento respectivo.

Inconsistencia local con inutilización temporal

Estas fallas refieren a cualquier error que derive en que la aplicación no pueda ser utilizada, ya sea por pérdida de consistencia local, caída fuera del consenso por falta de actualización y/o vigencia pero que puede recuperarse mediante la acción del usuario simple del usuario sin requerir la reinstalación del software y el reingreso de las claves privadas. Por ejemplo, si por alguna razón cliente registrara una transacción como enviada cuando no lo estuvo por un error en el servidor o en la conexión con el mismo, y retrocediendo sobre la sincronización y re-sincronizando hasta el último bloque se restableciera la consistencia de los datos.

Errores técnicos locales recuperables

Refieren a cualquier error que, sin estar afectada la capacidad de mantener consenso o disponer de los fondos, la operación de la wallet se ve interrumpida porque elementos del entorno interfieren con requerimientos no funcionales de alguna de sus partes. Por ejemplo, la falta de almacenamiento para el estado local de la wallet debido a que el usuario ha agotado este recurso y puede solucionarse mediante la recuperación de espacio disponible. Otro caso común podría ser la anulación de los permisos de la aplicación para con un recurso del dispositivo móvil que puede ser restablecido, como el acceso al tiempo de ejecución en segundo plano o a la conectividad de red, cuota de almacenamiento, etc.

Errores técnicos en servidor

Refiere a todo tipo de errores del servidor, ya sean de conectividad o de consenso con la red de pares, que no implican riesgos ni caen en los criterios de errores de pérdida de fondos, pudiendo recuperarse de los mismos mediante la utilización de otro servidor temporal o permanentemente.

5.4. Resumen de los componentes principales

Esta sección condensa los conceptos vertidos anteriormente y su objetivo es asistir al lector que acude al trabajo a modo de consulta

5.4.1. KeyStoring: Almacenamiento de claves

Intención

Proveer una interfaz que describa los comportamientos que ser requieren para almacenar claves privadas de forma segura.

Motivación

Dependiendo del dominio de aplicación este tipo de interfaces pueden tener distinta jerarquía. Para el caso de una billetera *non-custodian* el resguardo del valor es una función núcleo puesto que una wallet es un concepto de visualización sobre el valor que resguarda un conjunto de claves en una cadena de bloques específica. Sin claves no hay wallet posible y por ende lo que pudiera ser una clase utilitaria en otro tipo de aplicaciones, aquí es un componente neurálgico. A su vez una interfaz simplificada en comportamientos específicos permite abstraerse de las funcionalidades especificas de los componentes de almacenamiento seguro y centrarse en los requerimientos que deben satisfacerse. Por otra parte componentes como KeyChain de Apple o SecureStorage de Android disponen de modalidades de almacenamiento que disponibilizan los valores almacenados en base a niveles de autenticación del usuario y el estado del dispositivo (suspendido, activo, bloqueado, desbloqueado, etc.) por lo cual esta interfaz permite quitar esa complejidad y delegarla a la clase que la implementa.

Aplicabilidad

Los principales sistemas operativos actuales disponen de componentes nativos para el almacenamiento seguro de datos sensibles como contraseñas y certificados. No obstante, esto no significa que sus interfaces sean convenientes para exponer en aplicaciones clientes sin más. Además, estas cambian con el paso del tiempo impactando a la base de código de aplicaciones cliente. Al disponer de un contrato sobre el comportamiento requerido, estos cambios pueden quitarse del medio y administrarse en la implementación pudiendo ser ésta incluso una librería de terceras partes.

Estructura

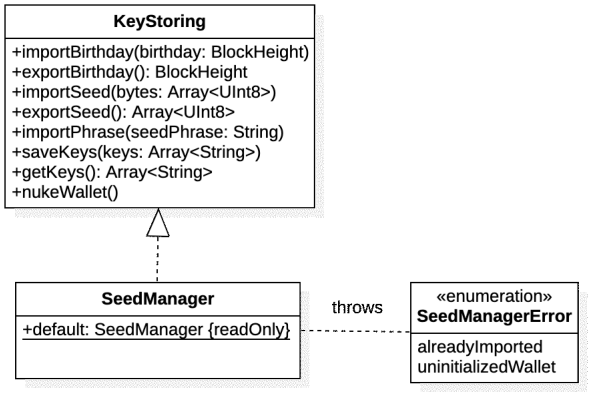


Figura 5.4: KeyStoring: una interfaz para almacenar claves y otros datos sensibles.

Participantes

SeedManager: Representa la implementación concreta de la interfaz. Ésta es quien se comunica con (y adapta) la interfaz del módulo de almacenamiento seguro a lo requerido por KeyStoring.

SeedManagerError: Administra y transforma los errores del módulo de almacenamiento seguro concreto a aquellos que tienen un significado puntual en el dominio de aplicación.

Colaboraciones

Esta interfaz se utiliza en conjunto con *MnemonicSeedHandling* 5.4.2.

Consecuencias

Al ser una adaptación de un componente concreto se corre el riesgo de incurrir en un anti-patrón, en los casos donde se esté envolviendo una interfaz que ya es una adaptación de una API de bajo nivel o inclusive una FFI cuyo destino se encuentra en un ambiente de otro lenguaje. Las cadenas de adaptadores, además de agregar niveles de indirección, pueden agregar complejidad en lugar de restarla en el caso de que un miembro de la cadena cambie y esto desate un efecto no deseado en el comportamiento de la clase concreta final.

Implementación

Uno de los aspectos a tener en cuenta a la hora de almacenar claves privadas para una wallet en particular es el tiempo de vida de estas dentro del sistema. ¿Qué sucede con ellas cuando el usuario elimina la aplicación? ¿son las claves eliminadas automáticamente por el sistema o permanecen en él hasta que el usuario las elimina manualmente? ¿están incluidas en las copias de respaldo del sistema? ¿qué sucede al reinstalar la aplicación en caso de que haya claves pre-existentes?

Otra cuestión es qué información guardar. Dependiendo de las capacidades del sistema y los tipos de clave, la derivación de una frase semilla puede ser costosa en términos computacionales y un problema para el rendimiento de la aplicación. El desarrollador debe evaluar si solo le basta con guardar la frase semilla haciendo la derivación bajo demanda al momento que se requiera una clave o si, por el contrario, estas deben derivarse única y oportunamente y luego guardarse en el almacenamiento seguro para su uso posterior. Esta última opción tiene un riesgo: es muy sencillo en términos de errores humanos introducir una linea de código que disimuladamente pueda ocasionar que la aplicación guarde las derivaciones de una frase semilla distinta. El almacenamiento de subproductos de la clave principal, agrega lógica para mantener un estado que puede potencialmente constituirse en un error de pérdida de fondos temporal o permanente.

Código de ejemplo

A continuación se expone el ejemplo de la interfaz Keystoring como "protocolo" en el lenguaje Swift (versión 5). El término Importar (import) se refiere a almacenar el valor en el almacenamiento del dispositivo. Mientras que exportar (export) recupera ese dato (sin eliminarlo).

Para el caso en el que el usuario quisiera destruir estas credenciales el método *nukeWallet* debe eliminar de forma permanente las claves almacenadas.

Código fuente 5.1: Interfaz *KeyStoring*

```
protocol KeyStoring {
    func importBirthday(_ height: BlockHeight) throws
    func exportBirthday() throws -> BlockHeight
    func importPhrase(bip39 phrase: String) throws
    func exportPhrase() throws -> String
    func nukeWallet()
}
```

En el listado a continuación se ve cómo la clase *SeedManager* implementa el protocolo *KeyStoring*.

A modo de conveniencia tiene una instancia *Singleton* para conservar una sola instancia de la misma. Luego se puede observar que el almacenamiento sobre el que esta implementado es de tipo Clave-Valor, siendo este detalle irrelevante para la interfaz propuesta.

Es conveniente que todos los métodos que retornen claves públicas o privadas lo hagan de forma tal que de no poder recuperarse no sean *anulables*[5] de forma tal que la falla sea notable para el desarrollador que esta implementando la aplicación cliente.

Código fuente 5.2: Implementación de interfaz *KeyStoring* en ECC Wallet

```
final class SeedManager: KeyStoring {

    enum SeedManagerError: Error {
        case alreadyImported
        case uninitializedWallet
    }

    static var `default`: SeedManager = SeedManager()
    private static let zECCWalletKeys = "zECCWalletKeys"
    private static let zECCWalletSeedKey = "zEECWalletSeedKey"
    private static let zECCWalletBirthday = "zECCWalletBirthday"
    private static let zECCWalletPhrase = "zECCWalletPhrase"

    private let keychain = KeychainSwift()

    func importBirthday(_ height: BlockHeight) throws {
        guard keychain.get(Self.zECCWalletBirthday) == nil else {
            throw SeedManagerError.alreadyImported
        }
        keychain.set(String(height), forKey: Self.zECCWalletBirthday)
    }

    func exportBirthday() throws -> BlockHeight {
        guard let birthday = keychain.get(Self.zECCWalletBirthday),
            let value = BlockHeight(birthday) else {
                throw SeedManagerError.uninitializedWallet
        }
        return value
    }

    func importPhrase(bip39 phrase: String) throws {
        guard keychain.get(Self.zECCWalletPhrase) == nil else {
            throw SeedManagerError.alreadyImported }
        keychain.set(phrase, forKey: Self.zECCWalletPhrase)
    }
```

[5]Del inglés nullable, es decir, que pueden retornar un valor nulo

```
func exportPhrase() throws -> String {
    guard let seed = keychain.get(Self.zECCWalletPhrase) else {
        throw SeedManagerError.uninitializedWallet }
    return seed
}

/**
 Use carefully: Deletes the seed phrase from the keychain
 */
func nukePhrase() {
    keychain.delete(Self.zECCWalletPhrase)
}
/**
    Use carefully: Deletes the keys from the keychain
 */
func nukeKeys() {
    keychain.delete(Self.zECCWalletKeys)
}

/**
   Use carefully: Deletes the seed from the keychain.
 */
func nukeSeed() {
    keychain.delete(Self.zECCWalletSeedKey)
}

/**
 Use carefully: deletes the wallet birthday from the keychain
 */

func nukeBirthday() {
    keychain.delete(Self.zECCWalletBirthday)
}

/**
There's no fate but what we make for ourselves - Sarah Connor
*/
func nukeWallet() {
    nukeKeys()
    nukeSeed()
    nukePhrase()
    nukeBirthday()

    // Fix: retrocompatibility with old wallets,
    // previous to IVK Synchronizer updates
    for key in keychain.allKeys {
        keychain.delete(key)
    }
}
```

Usos

Las wallets analizadas en los Casos de Estudio 5.5 hacen uso de interfaces que permiten abstraer el almacenamiento seguro de las claves del usuario y la lógica de la aplicación. Por otro lado la destrucción de las mismas es parte de la funcionalidad disponible en las billeteras Unstoppable y ECC Wallet.

Temas relacionados

Ver Adapter, Proxy y Chain of Responsability en [52]. Esta interfaz tiene estrecha relación con el componente (5.4.2) que se detalla a continuación.

5.4.2. MnemonicSeedHandling: manejo de frases Mnemónicas

Intención

Permite abstraer la recuperación y la creación de frases semilla de la lógica de negocio de la billetera.

Motivación

Existen distintos estándares de generación y recuperación de bytes en base a frases semilla. Si bien la proliferación se redujo drásticamente cuando se publicó el método establecido en BIP-39 [21] al día de hoy distintos coexisten distintos tipos de frases semilla. Es común que los desarrolladores de software wallets o hardware wallets (como Trezor® o Ledger®) indiquen explícitamente qué tipo de semilla soportan o directamente (como el caso de Ledger) tengan una advertencia para sus usuarios donde indican sólo dar soporte a frases semilla generadas con su billetera pese a que en la práctica, el utilizado sea BIP-39. Los requerimientos son bien claros: la interfaz debe permitir:

- Generar una frase mnemónica aleatoria como String
- Generar una frase mnemónica aleatoria como vector de Strings
- Generar una serie de bytes a partir de una frase mnemónica como String
- Generar una serie de bytes a partir de una frase mnemónica como Vector de Strings
- Validar un String como frase mnemónica

Aplicabilidad

Al momento de realizar este trabajo las frases semillas son el mecanismo más utilizado para generar y recuperar bytes aleatorios para generar claves de usuario. Esta interfaz permite condensar estos requerimientos, ocultando la complejidad de la generación de los bytes y delegando sus particularidades a la implementación pudiendo ser esta propia o una librería desarrollada por terceros.

Estructura

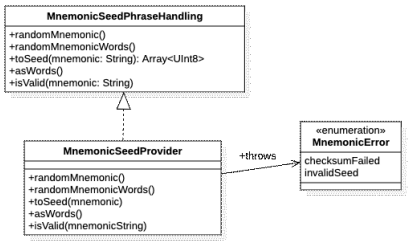


Figura 5.5: MnemonicPhraseHandling: una interfaz para manejar frases Mnemónicas.

Participantes

MnemonicPhraseHandling: la interfaz propuesta que condensa los requerimientos enumerados anteriormente.

MnemonicSeedProvider: es la implementación del la interfaz. Puede implementarla directamente o delegar esta implementación a una librería en cuyo caso actuaria como *adaptador* entre la interfaz propuesta y la existente en la librería utilizada.

MnemonicError: representa los errores posibles para estos requerimientos. Estos pueden nuclearse en dos errores primarios. Uno es *checksumFailed*, que refiere a la comprobación del checksum resultante de convertir la frase provista a bytes y verificar que esta frase, cuyas palabras corresponden al diccionario utilizado, sea íntegra en base al estándar utilizado. El otro es *InvalidSeed* refiere a que la frase propuesta es invalida en términos del diccionario propuesto.

Colaboraciones

Esta interfaz se utiliza en conjunto con la interfaz *KeyStoring* detallada en 5.4.1.

Consecuencias

El desarrollador debe tener en cuenta si la implementación concreta de la interfaz está adaptando una implementación que adapta otra y evitar la cadena de adaptaciones. Por otra parte, soportar múltiples implementaciones de generación de bytes puede otorgar un sentido de uniformidad que no es tal, complicando el manejo de las frases semillas y requiriendo a los desarrolladores tener conocimiento de cada implementación en particular y tener cuidado de no confundir una por otra en tiempo de ejecución.

Implementación

La librería *MnemonicSwift* [55] implementa BIP-39 utilizando esta interfaz.

Código de ejemplo

A continuación se detalla la interfaz en Swift.

Código fuente 5.3: Interfaz *MnemonicSeedPhraseHandling* en lenguaje Swift

```
enum MnemonicError: Error {
    case invalidSeed
    case checksumFailed
}

protocol MnemonicSeedPhraseHandling {
    /**
     frase Mnemonica de 24 palabras
     */
    func randomMnemonic() throws -> String
    /**
    frase Mnemonica de 24 palabras como vector de
    Strings
    */
    func randomMnemonicWords() throws -> [String]

    /**
     generar una semilla deterministica
     desde una frase
     */
    func toSeed(mnemonic: String) throws -> [UInt8]

    /**
     obtener las palabras desde esta frase
    */
    func asWords(mnemonic: String) throws -> [String]

    /**
     valida si la frase provista se justa a BIP-39
     y tiene coherencia interna
     */
    func isValid(mnemonic: String) throws
}
```

ECC Wallet implementa una delegación de la interfaz a la libreria *MnemonicSwift*

Código fuente 5.4: Implementación de *MnemonicSeedPhraseHandling* en ECC Wallet

```
import MnemonicSwift
class MnemonicSeedProvider: MnemonicSeedPhraseHandling {

    static let `default` = MnemonicSeedProvider()

    private init(){}
```

```
    func randomMnemonic() throws -> String {
        try Mnemonic.generateMnemonic(strength: 256)
    }

    func randomMnemonicWords() throws -> [String] {
        try randomMnemonic().components(separatedBy: " ")
    }

    func toSeed(mnemonic: String) throws -> [UInt8] {
        let data = try Mnemonic.deterministicSeedBytes(from: mnemonic)
        return [UInt8](data)
    }

    func asWords(mnemonic: String) throws -> [String] {
        mnemonic.components(separatedBy: " ")
    }

    func isValid(mnemonic: String) throws {
        try Mnemonic.validate(mnemonic: mnemonic)
    }
}
```

Usos

La proliferación del uso de frases semilla puede dar la impresión de la existencia de un uso generalizado y estandarizado de las mismas, cuando en realidad hay una técnica rectora (la frase semilla) pero luego los distintos actores pueden tomar su propio estándar o utilizar BIP-39 [21] indistintamente. En la sección 4.6 puede verse que para el caso de Monero utilizan frases de 25 palabras, lo cual genera un conjunto de bytes distintos a los utilizados por otras billeteras que utilizan 12 o 24 palabras como especifica BIP-39. Disponer de una forma de abstraer estas particularidades es una forma de evitar errores en la generación de claves privadas y la posible pérdida de fondos. Las Wallets ECC y Unstoppable (ver Casos de Estudio) utilizan interfaces de este estilo.

Temas relacionados

Ver patrones Adapter y Proxy del libro Design Patterns [52].

5.4.3. Initializer: Inicialización

Intención

Proveer un mecanismo para encapsular y abstraer la complejidad de inicializar los componentes requeridos para poder sincronizar una cadena de bloques y controlar el grafo de objetos y entidades derivados de éstos

Motivación

Bases de datos, URLs de hosts y nodos, parámetros para la creación de pruebas de cero-conocimiento o una interfaz FFI, son algunos elementos que se encuentran entre los requerimientos no funcionales de un cliente liviano de una AEC. A su vez, todos estos elementos contienen clases o entidades que los utilizan o administran. La existencia de un inicializador permite concentrar toda esta complejidad en una clase cuya intención está descrita en su firma. Por otra parte permite ser un punto de inyección de dependencias para testeo unitario. Un inicializador tiene como objetivo además contribuir a la economía de recursos mediante el control de los sub-componentes que requieren otros elementos como el Sincronizador. En el caso del SDK de Zcash, hace las veces de objeto constructor y nuclea ciertas funciones de conveniencia de la FFI.

Aplicabilidad

Se pude utilizar una clase de inicialización en el caso que se la utilización de una determinada blockchain requiera múltiples sub-componentes que interactúan entre sí o de forma independiente pero que cubren distintos requerimientos funcionales y no funcionales de esa blockchain. En la sección Usos se describen ciertas implementaciones en distintos proyectos.

Estructura

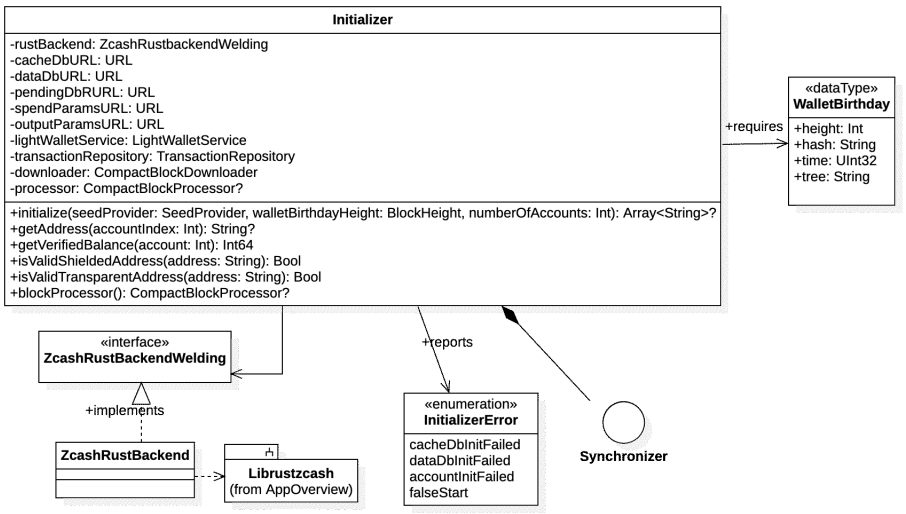


Figura 5.6: Initializer: encapsular la complejidad de la creación de un conjunto de componentes en una misma clase.

Participantes

Recursos del sistema: el inicializador recibe referencias a los recursos del sistema que se requieren para poder sincronizar una cadena de bloques.

Recursos de la aplicación cliente: referencias a los recursos de la aplicación que se requieren para poder sincronizar una cadena de bloques, como por ejemplo archivos de bases de datos, parámetros, referencias a clases de FFI.

Recursos provistos por el usuario: puede requerir claves o datos provistos por o relacionados al usuario o sus claves.

Objetos inicializados: Distintas clases que utilicen recursos nucleados en el inicializador pueden utilizarlo como parámetro o como constructor.

Colaboraciones

El inicializador colabora con distintos actores, en el caso de Zcash, permite inicializar las bases de datos que utiliza el componente LibrustZcash mediante la interfaz foránea en Rust, actúa como objeto constructor de distintos repositorio y puntualmente como parámetro tanto para el *SDKSynchronizer* (ver sección 4.5.1) y el *CompactBlockProcessor* (ver sección 4.5.2).

Consecuencias

La existencia de un inicializador evidencia la complejidad de conectar una aplicación cliente a un protocolo descentralizado de una AEC. Uno de los efectos no deseados es el potencial de "Objeto Dios" que el mismo tiene. Al acumular funcionalidad diversa (aunque con objetivos similares) se vuelve una situación recurrente en el proceso de desarrollo el debate de incluir métodos *de creación* [6] en él solo por el hecho de que no se arriba a un acuerdo dentro de un equipo de desarrollo sobre su ubicación y resulta conveniente.

Implementación

En la figura 5.7 se muestra en un diagrama de secuencia (abreviado) la inicialización del SDK de Zcash que se transcriben en el código fuente de ejemplo de los listados 5.4.3 y 5.4.3. Puede apreciarse la cantidad de clases que intervienen para poner en funcionamiento los distintos elementos que se necesitan para utilizar la cadena de bloques. A continuación se describe cada carril del diagrama:

Wallet Application: Código fuente de la billetera que utiliza el SDK de ZcashLightClientKit.

Initializer: Instancia de la clase. Su propósito es poner en marcha los distintos sub-componentes internos del SDK y actúa como objeto constructor de componentes

[6] En el libro Design Patterns se refiere a estos métodos como *Creational Methods*.

públicos como *CompactBlockProcesssor*.

CompactBlockStorage: Implementación de la interfaz *CompactBlockDAO*, se encarga de persistir en disco bloques compactos.

LightWalletService: Interfaz que maneja la conexión con el servidor de clientes livianos (Lightwalletd).

TransactionRepositoryBuilder: es una clase factoría que se encarga de construir instancias concretas de una interfaz llamada *TransactionRepository* cuyo objetivo es actuar como repositorio de transacciones.

CompactBlockDownloader: Una clase que descarga bloques compactos mediante la interfaz *LightWalletService* y utiliza *CompactBlockStorage* como persistencia.

CompactBlockProcessorBuider: es una clase factoría que se encarga de construir instancias de procesador de bloques *CompactBlockProcessor*.

ZcashRustBackendWelding: Interfaz con la FFI en lenguaje C del componente *Librustzcash*, que provee diversas funcionalidades pertinentes al protocolo Zcash.

En el primer carril de la figura 5.7 se aprecia que la iniciación del *Initializer* se realiza en dos pasos: la construcción del objeto *Initializer* y la puesta en marcha de sus componentes bajo la función *initialize()*.

En la construcción de *Initializer* se instancian otros componentes y luego por último el constructor principal retorna la instancia de *Initializer*. En la posterior llamada al método *initialize()*, el cliente (la wallet) provee los bytes semilla, un bloque de nacimiento asociado a estas claves y el número de cuentas a derivar. Este método deriva las claves, crea las tablas donde se guardarán los bloques desencriptados, las transacciones y las cuentas derivadas. El código relevante al diagrama de secuencia se encuentra en sub-sección siguiente.

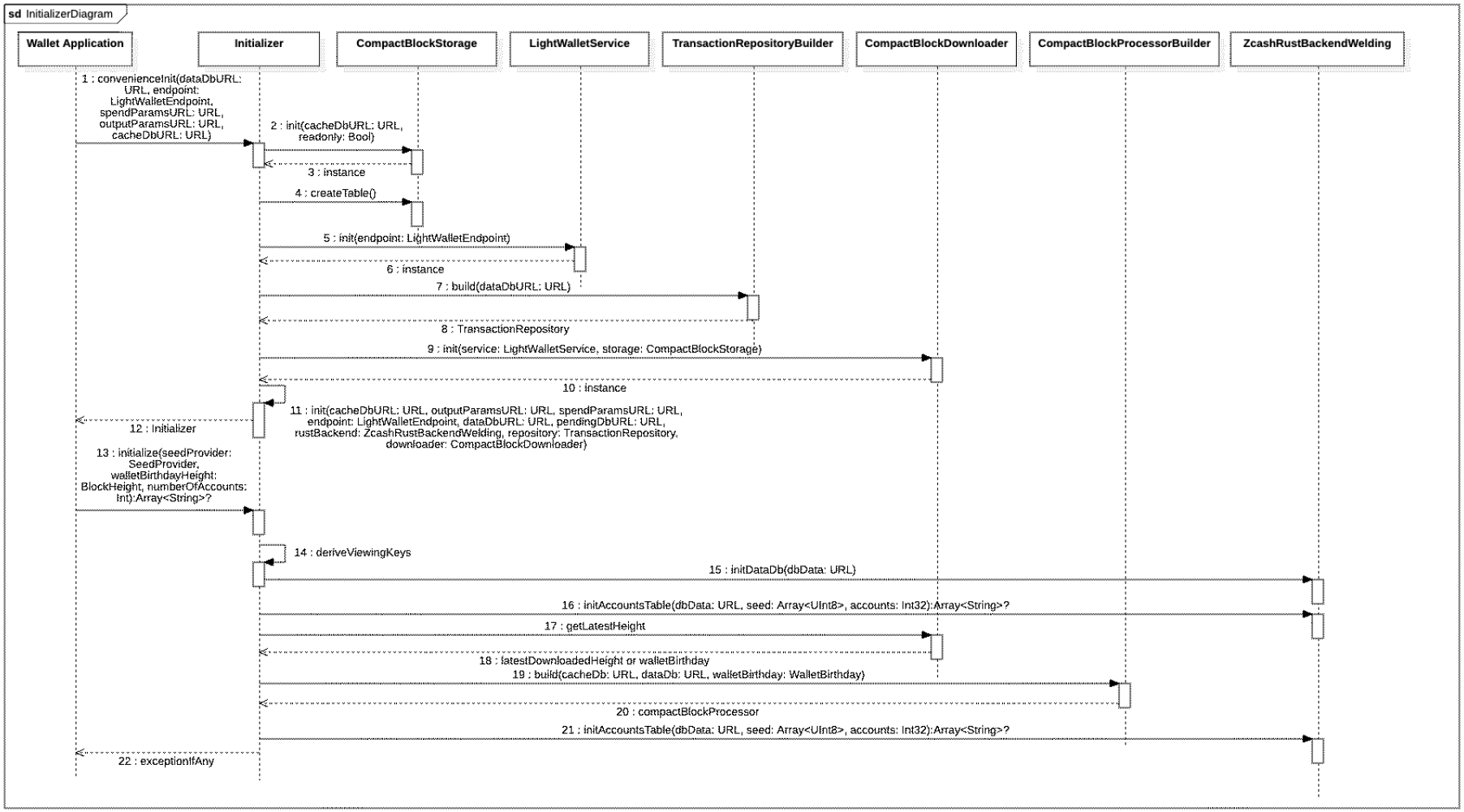


Figura 5.7: Diagrama de secuencia de la inicialización del SDK de Zcash desde una wallet que utiliza esa dependencia para interactuar con la blockchain.

Código de ejemplo

Interfaz pública del Inicializador en ZcashLightClientKit 10.2[7]. Agrupa los elementos que se requieren para inicializar distintos componentes que hacen posible la sincronización de una wallet a la blockchain de Zcash.

Código fuente 5.5: Declaración de la clase *Initializer* en ZcashLightClientKit 0.10.2

```
/**
 Wrapper for all the Rust backend functionality that
 does not involve processing blocks. This class initializes
 the Rust backend and the supporting data required to exercise
 those abilities.
 The CompactBlockProcessor handles all the remaining Rust backend
 functionality, related to processing blocks.
 */
public class Initializer {

    private(set) var rustBackend: ZcashRustBackendWelding.Type
    private(set) var alias: String
    private(set) var endpoint: LightWalletEndpoint

    private var lowerBoundHeight: BlockHeight
    private(set) var cacheDbURL: URL
    private(set) var dataDbURL: URL
    private(set) var pendingDbURL: URL
    private(set) var spendParamsURL: URL
    private(set) var outputParamsURL: URL
    private(set) var lightWalletService: LightWalletService
    private(set) var transactionRepository: TransactionRepository
    private(set) var downloader: CompactBlockDownloader
    private(set) var processor: CompactBlockProcessor?

    ....
}
```

También provee constructores de conveniencia[8] que enmascaran puntos de inyección de dependencias.

Código fuente 5.6: Constructor de conveniencia de la clase *Initializer*.

```
/**
    Constructs the Initializer
    - Parameters:
       - cacheDbURL: location of the compact blocks cache db
       - dataDbURL: Location of the data db
       - pendingDbURL: location of the pending transactions database
       - endpoint: the endpoint representing  the lightwalletd
```

[7] URL: https://github.com/zcash/ZcashLightClientKit/releases/tag/0.10.2

[8] En Swift existe un concepto llamado *"Convenience Initializers"*. Su propósito es aportar constructor diferente para ciertos casos de uso donde se crea un una instancia de un objeto

```
        instance you want to point to
        - spendParamsURL: location of the spend parameters
        - outputParamsURL: location of the output parameters
     */
    convenience public init (cacheDbURL: URL,
                dataDbURL: URL,
                pendingDbURL: URL,
                endpoint: LightWalletEndpoint,
                spendParamsURL: URL,
                outputParamsURL: URL,
                alias: String = "",
                loggerProxy: Logger? = nil) {

        let storage = CompactBlockStorage(url: cacheDbURL, readonly: false)
        try? storage.createTable()

        let lwdService = LightWalletGRPCService(endpoint: endpoint)

        self.init(rustBackend: ZcashRustBackend.self,
                  lowerBoundHeight: ZcashSDK.SAPLING_ACTIVATION_HEIGHT,
                  cacheDbURL: cacheDbURL,
                  dataDbURL: dataDbURL,
                  pendingDbURL: pendingDbURL,
                  endpoint: endpoint,
                  service: lwdService,
                  repository: TransactionRepositoryBuilder.build(
                    dataDbURL: dataDbURL
                  ),
                  downloader: CompactBlockDownloader(
                    service: lwdService,
                    storage: storage),
                  spendParamsURL: spendParamsURL,
                  outputParamsURL: outputParamsURL,
                  alias: alias,
                  loggerProxy: loggerProxy
        )
    }
```

El "inicializador" provee un punto de ingreso para la generación del ambiente de los componentes que requieren de las claves del usuario, en este caso mediante el ingreso de claves de visualización.

Código fuente 5.7: Método de inicialización de *Initializer*

```
    public func initialize(viewingKeys: [String],
                           walletBirthday: BlockHeight
                           ) throws {
        let derivationTool = DerivationTool()
        for vk in viewingKeys {
            do {
```

```
            guard try derivationTool.isValidExtendedViewingKey(vk) else {
                throw InitializerError.invalidViewingKey(key: vk)
            }
        } catch {
            throw InitializerError.invalidViewingKey(key: vk)
        }
    }

    do {
        try rustBackend.initDataDb(dbData: dataDbURL)
    } catch RustWeldingError.dataDbNotEmpty {
        // this is fine
    } catch {
        throw InitializerError.dataDbInitFailed
    }

    let birthday = WalletBirthday.birthday(with: walletBirthday)

    do {
        try rustBackend.initBlocksTable(dbData: dataDbURL,
                        height: Int32(birthday.height),
                        hash: birthday.hash,
                        time: birthday.time,
                        saplingTree: birthday.tree)
    } catch RustWeldingError.dataDbNotEmpty {
        // this is fine
    } catch {
        throw InitializerError.dataDbInitFailed
    }

    ....
}
```

De esta manera la aplicación ECC Wallet [45] utiliza el inicialización en su propio ciclo de inicialización (pues es una wallet que solo soporta Zcash)

Código fuente 5.8: Método de inicialización de ECC Wallet

```
func initialize() throws {
    let seedPhrase = try SeedManager.default.exportPhrase()
    let seedBytes = try MnemonicSeedProvider.default.toSeed(
                                mnemonic: seedPhrase)

    let initializer = Initializer(
        cacheDbURL: self.cacheDbURL,
        dataDbURL: self.dataDbURL,
        pendingDbURL: self.pendingDbURL,
        endpoint: endpoint,
        spendParamsURL: self.spendParamsURL,
        outputParamsURL: self.outputParamsURL,
        walletBirthday: try SeedManager.default.exportBirthday(),
```

```
            loggerProxy: logger)

        self.synchronizer = try CombineSynchronizer(
                                    initializer: initializer
                                    )

        _ = try self.synchronizer.initializer.initialize(
                    seedBytes: seedBytes,
                    numberOfAccounts: 1)

        self.subscribeToApplicationNotificationsPublishers()

        fixPendingTransactionsIfNeeded()

        try self.synchronizer.start()
    }
```

Usos

La noción de una clase que se encarga de la inicialización se encuentra presente en varios kits de desarrollo y frameworks. En el caso de las criptomonedas, uno de los más prominente podría ser la instancia Web3 [63]. El de SDK de Zcash utiliza este concepto para concentrar toda el comportamiento "creacional" en esta instancia.

Temas relacionados

Ver Patrones "creacionales" (Creational Patterns) en el Libro Design Patterns [52].

5.4.4. Synchronizer: sincronización con cadena de bloques

Intención

Operar en una cadena de bloques se resume en la acción de estar al día con los datos producidos en la misma desde el punto de vista de un conjunto de claves privadas y/o públicas. Un sincronizador implementa los requisitos funcionales y no funcionales para tales fines.

Motivación

Operar en una blockchain requiere tener conocimiento de un protocolo determinados, cómo opera su consenso y su red de pares. Para operar, ya sea para recibir o emitir transacciones, es necesario estar en sincronía con la cadena de bloques. Esta lógica se encuentra descrita en la documentación de las distintas blockchains y condensan allí requerimientos que tienen una base en común que es definida por el tipo de protocolo de consenso que se utilice: Proof-of-Work, Proof-of-Stake, Proof-of-Space, etcétera. La figura del *Synchronizer* aglutina toda la funcionalidad que se requiere para mantenerse al día con los bloques producidos, recibir y enviar transacciones. Es posible que el acto de sincronizar se realice de forma "remota" o dentro del mismo dispositivo. Muchas criptomonedas de libro contable público centralizan la sincronización en un servidor

o nodo maestro y pasan claves publicas para obtener los *UTXO* pertenecientes a las mismas desde el servidor quien tiene la copia de la clave. Esto no es exclusivo de las criptomonedas "públicas", las privacy coins (AECs) como Monero (o sus variantes) sincronizan mediante el envío de claves de visualización a un nodo que les entrega un conjunto de datos actualizados que representa la vista de la blockchain en base a esas claves. De todas formas, toda esa información se encuentra nucleada en una entidad que enmascara esta funcionalidad y la encapsula en una interfaz.

Aplicabilidad

Como se ha descrito en secciones anteriores el rol del sincronizador se encuentra en todas las blockchains de alguna forma. En algunos casos delegado en un servidor como en el caso de billeteras *custodian* donde la potestad de las claves se encuentra delegada (por ejemplo: wallets de exchanges), o un esquema mixto, como Monero donde se terceriza una clave de visualizador a un servidor para que este realice la desencripción por prueba y error de la cadena de bloques ya sincronizada en él devolviendo al cliente las transacciones pertinentes a esas claves, o en el caso de Zcash, estando completamente del lado del cliente. Como se ha expresado anteriormente, sincronizar es una acción definitoria de una wallet, dado que ella no es más que la representación local de una blockchain desde el punto de vista de un conjunto de claves determinadas.

Estructura

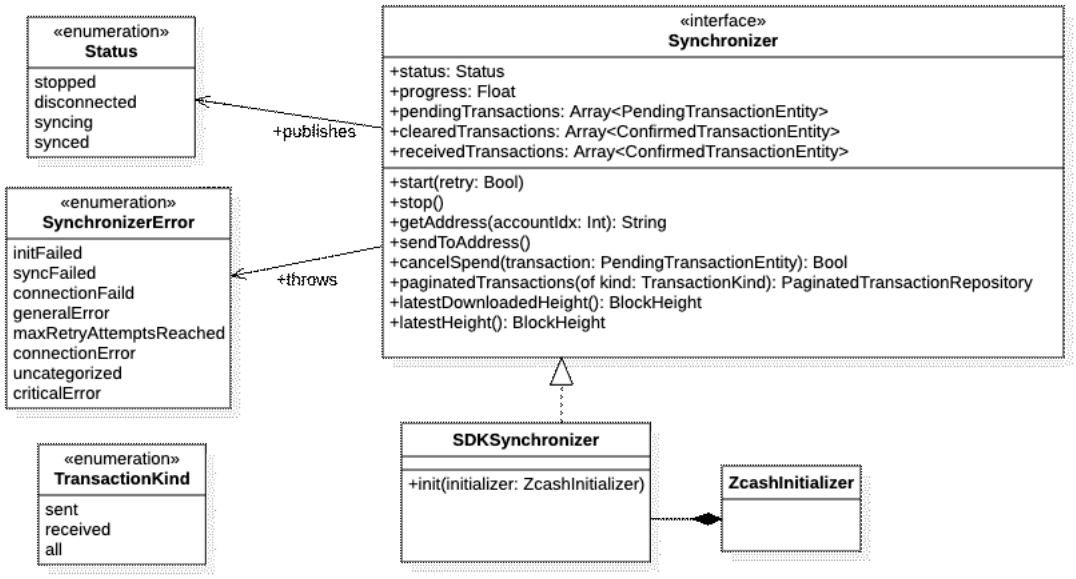


Figura 5.8: Synchronizer: una interfaz que concentra los requerimientos detrás de la sincronización con una cadena de bloques desde el punto de vista de las claves del usuario.

Participantes

Interfaz Synchronizer: expresa en su interfaz pública claramente sus funciones, como *start* y *stop*, etc. Es el punto de entrada hacia una blockchain específica. Puede a su vez contener funcionalidades accesorias que podrían realizarse con librerías como por ejemplo *getAddress*. Emite o publica su estado a través de la enumeración «*Status*» y el progreso de la sincronización mediante «progress». Las transacciones que surgen del proceso de sincronización se exponen mediante las propiedades *pendingTransactions*, *clearedTransactions* y *receivedTransactions*. Las transacciones pendientes (pending) son aquellas que han sido enviadas por el usuario pero no han sido confirmadas aún. Las confirmadas (cleared) son aquellas que han sido minadas y además ha transcurrido una determinada cantidad de bloques luego de que fueron incluidas en la cadena. [9]. Las transacciones recibidas son aquellas que se identifican por contener outputs que

[9]La cantidad de bloques de confirmación varía en wallets y blockchains. ZecWallet utiliza dos confirmaciones mientras que Unstoppable utiliza diez. Un exchange puede, en cambio utilizar veinticuatro o treinta confirmaciones para acciones que interactúen fuera de su dominio como una extracción a una wallet externa, y un número considerablemente menor para confirmar traspasos de fondos internos entre usuarios del propio exchange o cambios de moneda efectuados para el mismo usuario.

hacen referencia a una dirección generada con la claves de la billetera en cuestión.

Instancia concreta, SDKSynchronizer: la implementación de la interfaz. Esta tiene adicionalmente la responsabilidad de conocer el ambiente de ejecución y adaptarse al ciclo del vida del mismo. Puede observarse en las implementaciones de iOS y Android del SDK de Zcash, que las interfaces coinciden en funcionalidad, y difieren en semántica del lenguaje de implementación y sobretodo en diferencias del ambiente de ejecución. Lo mismo puede darse dentro del mismo lenguaje al ejecutarse en otra plataforma. En un ambiente de sistemas operativos de usos múltiples como Windows, Mac OS o GNU/Linux, el ciclo de vida será sustancialmente diferente de aquellos en Android o iOS. La implementación del sincronizador será donde ser expresarán y administrarán fundamentalmente esas diferencias.

Status: Expresa el estado del sincronizador. Éste tiene una gran influencia sobre la interfaz de usuario.

SynchronizerError: Condensa los posibles fallos que de una forma resumida para poder ser atrapados por el código del cliente y mostrados al usuario (o enmascarados) en una forma adecuada según el criterio del desarrollador.

TransactionKind: El tipo de transacciones retornadas por el sincronizador.

Colaboraciones

El sincronizador puede colaborar diversos componentes del sistema, su rol es primordialmente el de condensar los requerimientos para operar dentro de una blockchain de una forma concisa y clara de cara a una aplicación cliente. Para su creación puede colaborar con un *Inicializador*. En el caso de Zcash, la clase *SDKSynchronizer* (ver sección) 4.5.1 enmascara el componente *CompactBlockProcessor* 4.5.2 (ver sección) cuya interfaz es más extensa, granular y de más bajo nivel.

Consecuencias

En contrapartida los beneficios de una interfaz compacta y asertiva, se encuentra el hecho de que para tales fines, muchas decisiones de diseño han sido tomadas por terceros. Este factor es puesto en discusión brevemente en la sección 5.2, donde se cita un apartado sobre la definición de *framework* y librerías en el libro *GoF* [52]. El precio a pagar por la "conveniencia" es generalmente la posibilidad de personalizar el comportamiento de una pieza de software. Esta incidencia no es menor. El dilema "Buy or build" no debe ser tomado a la ligera. El libro "Code Complete" de MCconnell [64] tiene varias secciones al respecto que el lector debiera considerar como referencia en este asunto puntual.

Implementación

La implementación del componente *Synchronizer* esta ligada tanto a la cadena de bloques que sincronizará como a la plataforma en la que se ejecutará. Aquí se deben administrar las particularidades de cada plataforma, el ciclo de vida de la aplicación y cómo este afecta o interfiere con los requerimientos que el sincronizador condensa.

El ambiente de programación también influye considerablemente la implementación del mismo. En el caso de la Wallet ECC-Wallet [45], siendo esta una wallet implementada con el Framework SwiftUI y Combine de Apple, se encuentra signado por lo declarativo del framework de interfaz de usuario y lo reactivo de su flujo de datos. Por ello implementa un *Wrapper* del *SDKSynchronizer* del SDK de Zcash para iOS, llamado *CombineSynchronizer*. Algo similar se describirá en el caso de estudio de la wallet Unstoppable, donde los desarrolladores utilizan el patrón *Adapter* para compatibilizar el carácter *imperativo* del sincronizador al framework *RxSwift* que utilizan en su aplicación.

Código de ejemplo

Interfaz pública generada por el compilador de Swift 5.3 en Xcode para la versión 0.9.3 de ZcashLightClientKit [48]

Código fuente 5.9: Interfaz *Synchronizer* en lenguaje Swift

```
/**
Represents errors thrown by a Synchronizer
 */
public enum SynchronizerError : Error {
    case initFailed(message: String)
    case syncFailed
    case connectionFailed(message: Error)
    case generalError(message: String)
    case maxRetryAttemptsReached(attempts: Int)
    case connectionError(status: Int, message: String)
    case networkTimeout
    case uncategorized(underlyingError: Error)
    case criticalError
    case parameterMissing(underlyingError: Error)
}

/**
Primary interface for interacting with the SDK.
Defines the contract that specific implementations like SdkSynchronizer fulfill.
*/
public protocol Synchronizer {

    /**
    Starts this synchronizer within the given scope.

    Implementations should leverage structured concurrency and
    cancel all jobs when this scope completes.
```

```
*/
func start(retry: Bool) throws

/**
 Stop this synchronizer. Implementations should ensure that
 calling this method cancels all jobs that were created
 by this instance.
 */
func stop() throws

/**
Value representing the Status of this Synchronizer.
As the status changes, a new value will be emitted by KVO
*/
var status: Status { get }

/**
 A flow of progress values, typically corresponding to this
 Synchronizer downloading blocks. Typically, any non-zero
 value below 1.0 indicates that progress indicators can be
 shown and a value of 1.0 signals that progress is complete
 and any progress indicators can be hidden.
 */
var progress: Float { get }

/**
Gets the address for the given account.
- Parameter accountIndex: the optional accountId whose
address is of interest.
By default, the first account is used.
*/
func getAddress(accountIndex: Int) -> String

/**
Sends zatoshi.
- Parameter spendingKey: the key that allows spends to occur.
- Parameter zatoshi: the amount of zatoshi to send.
- Parameter toAddress: the recipient's address.
- Parameter memo: the optional memo to include as part
of the transaction.
- Parameter accountIndex: the optional account id to use.
By default, the first account is used.
*/
func sendToAddress(
        spendingKey: String,
        zatoshi: Int64,
        toAddress: String,
        memo: String?,
        from accountIndex: Int,
        resultBlock: @escaping (
```

```
                _ result: Result<PendingTransactionEntity,
                 Error>) -> Void
    )

/**
   Attempts to cancel a transaction that is about to be sent.
   Typically, cancellation is only an option if the
   transaction has not yet been submitted to the server.
- Parameter transaction: the transaction to cancel.
- Returns: true when the cancellation request was successful.
False when it is too late.
   */
func cancelSpend(
    transaction: PendingTransactionEntity
    ) -> Bool

/**
    all outbound pending transactions that have been sent
    but are awaiting confirmations
 */
var pendingTransactions: [PendingTransactionEntity] { get }

/**
 al the transactions that are on the blockchain
 */
var clearedTransactions: [ConfirmedTransactionEntity] { get }

/**
 All transactions that are related to sending funds
 */
var sentTransactions: [ConfirmedTransactionEntity] { get }

/**
 all transactions related to receiving funds
 */
var receivedTransactions: [ConfirmedTransactionEntity] { get }

/**
    a repository serving transactions in a paginated manner
 - Parameter kind: Transaction Kind expected from this
 PaginatedTransactionRepository
 */
func paginatedTransactions(
    of kind: TransactionKind
    ) -> PaginatedTransactionRepository

/**
 Returns a list of confirmed transactions that preceed the
 given transaction with a limit count.
 - Parameters:
```

```
        - from: the confirmed transaction from which the query
        should start from or nil to retrieve from the most recent
        transaction
        - limit: the maximum amount of items this should return if
        available
      - Returns: an array with the given Transactions or nil

      */
    func allConfirmedTransactions(
        from transaction: ConfirmedTransactionEntity?,
        limit: Int
        ) throws -> [ConfirmedTransactionEntity]?

    /**
        gets the latest downloaded height from the compact block
        cache
      */
    func latestDownloadedHeight() throws -> BlockHeight

    /**
      Gets the latest block height from the provided Lightwallet
      endpoint
      */
    func latestHeight(result: @escaping (Result<BlockHeight, Error>) -> Void)

    /**
      Gets the latest block height from the provided
      Lightwallet endpoint
      Blocking
      */
    func latestHeight() throws -> BlockHeight
}

/**
 The Status of the synchronizer
 */
public enum Status {

    /**
    Indicates that [stop] has been called on this Synchronizer
    and it will no longer be used.
    */
    case stopped

    /**
    Indicates that this Synchronizer is disconnected
    from its lightwalletd server.
    When set, a UI element may want to turn red.
    */
    case disconnected
```

```
    /**
    Indicates that this Synchronizer is not yet synced and
    therefore should not broadcast transactions because it
    does not have the latest data. When set, a UI element
    may want to turn yellow.
    */
    case syncing

    /**
    Indicates that this Synchronizer is fully up to date
    and ready for all wallet functions.
    When set, a UI element may want to turn green.
    */
    case synced
}

/**
 Kind of transactions handled by a Synchronizer
 */
public enum TransactionKind {
    case sent
    case received
    case all
}
```

Inicialización del SDKSynchronizer en la wallet Unstoppable [46] de iOS version 0.21.2, donde se observa la delegación de la derivación de las claves a un componente llamado *DerivationTool* (1) y la construcción del *SDKSynchronizer* utiliza un inicializador (2), luego se suscribe a los eventos del sistema operativo (3) y a los eventos que emite el sincronizador (4).

Código fuente 5.10: Implementación de inicialización de *SDKSynchronizer* en ECC Wallet

```
    let initializer = Initializer(
        cacheDbURL:try! ZcashAdapter.cacheDbURL(
            uniqueId: uniqueId
            ),
        dataDbURL: try! ZcashAdapter.dataDbURL(
            uniqueId: uniqueId
            ),
        pendingDbURL: try! ZcashAdapter.pendingDbURL(
            uniqueId: uniqueId
            ),
        endpoint: LightWalletEndpoint(
            address: endPoint,
            port: 9067
            ),
        spendParamsURL: try! ZcashAdapter.spendParamsURL(
                uniqueId: uniqueId
```

```
            ),
        outputParamsURL: try! ZcashAdapter.outputParamsURL(
            uniqueId: uniqueId
            ),
        loggerProxy: loggingProxy)

    let seedData = [UInt8](seed)
    try initializer.initialize(
        viewingKeys: try DerivationTool.default.deriveViewingKeys(
                            seed: seedData,
                            numberOfAccounts: 1
                        ),
            walletBirthday: BlockHeight(birthday)
        )
    // 1
    keys = try DerivationTool.default.deriveSpendingKeys(
                    seed: seedData,
                    numberOfAccounts: 1
                )

    // 2
    synchronizer = try SDKSynchronizer(initializer: initializer)

    transactionPool = ZcashTransactionPool()

    transactionPool.store(
        confirmedTransactions: synchronizer.clearedTransactions,
        pendingTransactions: synchronizer.pendingTransactions
    )

    balanceState = .syncing(progress: 0, lastBlockDate: nil)
    transactionState = balanceState
    lastBlockHeight = try? synchronizer.latestHeight()

    //3
    NotificationCenter.default.addObserver(
            self,
            selector: #selector(didEnterBackground),
            name: UIApplication.didEnterBackgroundNotification,
            object: nil)

    // 4
    subscribeSynchronizerNotifications()
    subscribeDownloadService()
```

Usos

Es posible apreciar una interfaz similar a la de *Synchronizer* en otros SDK, como BitcoinKit [65], donde la clase BitcoinCore condensa el comportamiento expresado en

la interfaz Synchronizer (y varios métodos más).

Código fuente 5.11: Clase *BitcoinCore* en SDK *BitcoinKit*. Similitudes con *SDKSynchronizer*.

```
extension BitcoinCore {
    public func start()
    internal func stop()
}

extension BitcoinCore {
    public var lastBlockInfo: BlockInfo? { get }
    public var balance: BalanceInfo { get }
    public var syncState: BitcoinCore.KitState { get }
    public func transactions(fromUid: String? = nil,
                             limit: Int? = nil
                         ) -> Single<[TransactionInfo]>
    public func transaction(hash: String) -> TransactionInfo?
    public func send(to address: String,
            value: Int,
            feeRate: Int,
            sortType: TransactionDataSortType,
            pluginData: [UInt8 : IPluginData] = [:]
    ) throws -> FullTransaction

    ....
```

Temas relacionados

El sincronizador es una interfaz que puede ser considerada una Façade[10]. Para su creación se utilizan patrones *creacionales* como Builder o Factory Method [52].

5.5. Casos de Estudio

Zcash fue el primer proyecto que puso un sistema de pruebas de cero-conocimiento con zkSNARKs en producción a escala global. Desde ese entonces fue posible que dos usuarios intercambiaran valor por medio de una criptomoneda con privacidad en la blockchain de forma efectiva. Esas primeras transacciones correspondían a la versión Sprout de Zcash cuyo cómputo demoraba 40 segundos en una computadora de escritorio y eran consideradas imposibles de realizar en dispositivos móviles. En el año 2018, el equipo de ingeniería de Zcash lanza Sapling. Un cambio radical en la forma de realizar las pruebas de cero conocimiento, siendo posible realizarlas en aproximadamente 10 segundos de computo en dispositivos móviles. A partir de ese momento, comenzaron esfuerzos por parte de la comunidad de desarrolladores de Zcash por desarrollar billeteras y clientes livianos capaces de poner esta tecnología literalmente en manos de todos. Fue así que el desarrollo del protocolo para clientes livianos [22] (ver anexo 7.5) por parte de George Tankersley, Matthew Green, Jack Grigg, Daira Hopwood y

[10] Ver patrón en "Design Patterns" [52]

Taylor Hornby, hizo posible contar con una representación de la blockchain blindada de Zcash de sólo un décimo de su tamaño original, de la cual los clientes livianos solo deberían conservar aquellas transacciones relevantes a sus claves.

Esta reducción del procesamiento y el almacenamiento de datos de varios órdenes de magnitud, fue la que dio inicio a varios proyectos que tienen como objetivo llevar las transacciones blindadas de Sapling Zcash a las manos de miles de usuarios de Zcash en sus respectivos dispositivos móviles Android y iOS.

En las siguientes secciones se describirán y pondrán en contexto los proyectos que han sido tomados como casos de estudio para este trabajo.

5.5.1. Wallets de referencia: ECC Wallet y clones

Estas billeteras son parte de una iniciativa de *Dogfooding*[11] por parte Electric Coin Company, una de las organizaciones a cargo del desarrollo de Zcash. El desarrollo de las aplicaciones ECC Wallet para Android y iOS comenzó a fines del año 2019, como un mecanismo para explorar la usabilidad de los frameworks de esa criptomoneda en las respectivas plataformas mobile. Estan construidas de forma "nativa", utilizando SwiftUI y Combine en iOS y *Coroutines* de Android en Kotlin. Ambas wallets utilizan variantes del paradigma de Programación Reactiva. En el caso de Android, el SDK de Zcash se encuentra desarrollado con este mismo paradigma mientras que en su contraparte de iOS, el SDK se encuentra desarrollado utilizando el paradigma imperativo que utilizan la mayoría de las aplicaciones de iOS. Esta elección responde a la diferencia de criterios de Google y Apple para manejar la retro-compatibilidad en sus plataformas. Mientras que Android construye con un grado de compatibilidad hacia versiones anteriores de su propio software, Apple hace exactamente lo contrario: lanzar funcionalidad que depreca interfaces y sólo se encuentra disponible en versiones actuales o en beta hacia adelante. Es por ello que el SDK de iOS soporta versiones de iOS desde 12.0 en adelante, mientras que la aplicación ECC Wallet sólo lo hace desde la versión 13.0 en adelante, la cual introduce el framework de capa de presentación SwiftUI.

La aplicación de iOS presenta un patrón de decoración o envoltorio sobre la clase *SDKSynchronizer* llamada *CombineSynchronizer*, cuyo objetivo es adaptar el funcionamiento de carácter imperativo de la misma al paradigma Reactivo y de *Backpressure* que utiliza Combine[12]

Nighthawk Apps. Forks de ECC Wallet en producción

El código de las aplicaciones ECC Wallet fue liberado en Junio de 2020 en ocasión de un hackatón organizado por la compañía ECC y la organización de Gitcoin, cuya misión es brindar mecanismos de financiamiento y apoyo económico descentralizado a proyectos de Código Abierto mediante Criptomonedas. La compañia NightHawk

[11] Dogfooding refiere al uso de un producto de forma interna en una organización antes de que sea puesto lanzado a la generalidad de sus clientes/usuarios.

[12] Combine es un framework de programación reactiva desarrollado por Apple que fue lanzado junto con SwiftUI, para adoptar este paradigma de programación en su capa de presentación, aunque también es presentado como una herramienta de manejo del flujo de datos en el dominio de aplicación y lógica de negocio de las aplicaciones del ecosistema

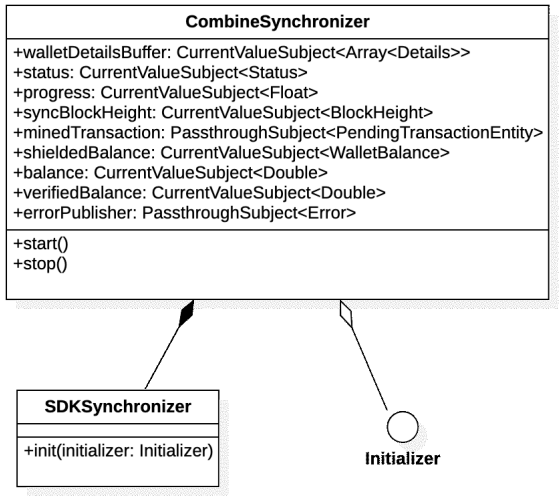


Figura 5.9: CombineSynchronizer: un Decorator del Synchronizer para programación funcional reactiva.

Apps, decide tomar la decisión audaz de realizar *forks* de cada uno de los repositorios y desplegar las aplicaciones ECC Wallet para Android y iOS bajo su "marca blanca" NightHawk Wallet cuyo logotipo insignia hace referencia al icónico caza bombardero antiradares de la fuerza aérea estadounidense.

ZecWallet Lite y CLI

ZecWallet es un desarrollador de wallets para Zcash que trabaja en conjunto con la Zcash Foundation y el Comité de Grandes Concesiones Abiertas (Zcash Open Major Grants Comitee o ZOMG), cuyo objetivo fue crear una billetera con interfaz gráfica para acompañar a los nodos Zcash. Hasta la salida de ZecWallet Full Node, la única forma de utilizar transacciones blindadas con Zcash era utilizar la aplicación de cliente de terminal zcash-cli que incluye el nodo de Zcash (zcashd).

ZecWallet tiene distintos tipos de wallet. Además de la Full Node, existe una versión *lite*, que utiliza el modelo de clientes livianos (ver sección 7.5) mediante la aplicación CLI como se detalla en el capítulo de relevamiento de requerimientos en la sección 4.1.1 y ZecWallet Lite Mobile para dispositivos Android y iOS.

En todas sus versiones *Lite* utiliza el esquema de *front-end* React.js y *back-end* mediante el uso del CLI en Rust. Lo cual permite reutilizar gran cantidad de código fuente para todos los requerimientos que supone operar dentro del protocolo Zcash como cliente liviano.

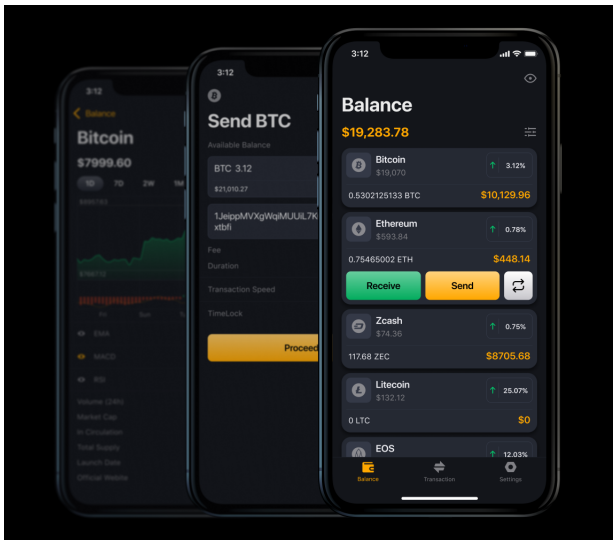


Figura 5.10: Interfaz de Usuario de Unstoppable Wallet

Unstoppable: Wallet Multi Moneda

Desarrollada por Horizontal Systems, Unstoppable [66] es una aplicación que combina una wallet multi-moneda non-custodian con una academia de criptofinanzas de forma nativa para Android [47] y iOS [46]. Horizontal Systems es un equipo de desarrolladores de Europa del Este, con vasta experiencia en la creación de herramientas de desarrollo para criptomonedas, cuentan en su haber con proyectos como "EthereumKit" [67] y "BitcoinKit" [68] que implementan de forma nativa las funcionalidades necesarias para operar en Ethereum y Bitcoin desde aplicaciones móviles.

La arquitectura de las aplicaciones Unstoppable, presenta abstracciones por sobre el dominio de aplicación de las criptomonedas haciendo uso del patrón *Adapter*, para acomodar las particularidades de cada caso en términos de las generalidades de las historias de usuario que caben a todas las criptomonedas (ver historias de usuario base 4.4).

Esta aplicación ofrece a sus usuarios operar con mas de una decena de criptomonedas y ecosistemas cripto como Ethereum, Binance Chain y sus diferentes tokens. A su vez es la primera wallet multi-moneda en adoptar Zcash de forma nativa en dispositivos móviles[13].

Como caso de estudio, se observa la integración del kit de desarrollo de Zcash, utilizando el esquema Inicializador-Sincronizador que el mismo propone mediante un adaptador llamado *ZcashAdapter*. En cuanto a la frase semilla y las claves del usuario, Unstoppable hace uso de su propia librería de frases mnemónicas que cumplen con el estándar BIP-39 mencionado en secciones anteriores, guardando las claves privadas en

[13]en su forma blindada, hasta entonces todas las aplicaciones multi-moneda que ofrecía Zcash como moneda a sus usuarios lo hacían solo soportando direcciones transparentes de la misma forma que soportan Bitcoin

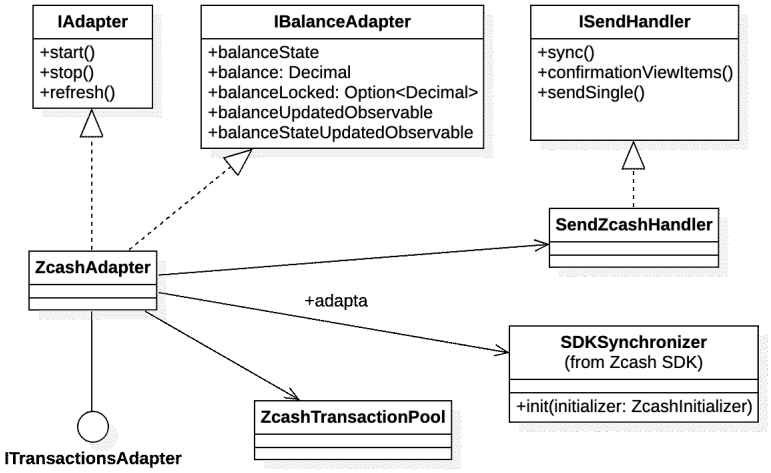


Figura 5.11: Diagrama de clases resumido de la implementación de la wallet Zcash en Unstoppable

el almacenamiento seguro de los respectivos sistemas operativos. La clase ZcashAdapter tiene como finalidad adaptar las distintas propiedades de la clase *Synchronizer* a las abstracciones definidas para la wallet multi-moneda. En este adaptador se observa la linealidad de estas adaptaciones, que responde a la necesidad de consolidar interfaces similares a la arquitectura de la capa de presentación elegida: *RxSwift* y *RxJava*[14] y la arquitectura de aplicación *Clean Code* [69].

[14]RxSwift y RxJava son dos frameworks de desarrollo que implementan el paradigma de programación reactiva, que consiste en utilizar de forma extensiva el esquema Publicador-Suscriptor, *bindings* y *backpressure* para la comunicación entre las vistas y el modelo de aplicación

Capítulo 6

Conclusiones y trabajo a futuro

Este trabajo tiene como objetivo indagar en las cuestiones referentes al desarrollo de una billetera tipo cliente liviano para operar en criptomonedas con mejoramiento del anonimato (AECs en inglés) que permitan a los usuarios hacer custodia soberana de sus claves públicas y privadas.

En un análisis previo se identificaron distintas cuestiones que hacen al desarrollo de clientes livianos para "criptoactivos"[1] y se definieron los objetivos con los temas que serían alcanzados por el trabajo y aquellos que no serían profundizados o dejados fuera en su totalidad.

El objetivo del trabajo quedó establecido en conformar una lista de requerimientos funcionales y no funcionales que un cliente liviano non-custodian para privacy coins debe satisfacer y en base a ellos proponer una arquitectura de referencia que permita desarrollar un cliente para dispositivos móviles. Además se estableció que el alcance de dicha propuesta debía resultar en la construcción de una aplicación donde:

- se utilice la custodia soberana de las claves privadas del usuario de forma segura;
- los usuarios puedan operar criptomonedas que provean mecanismo de salvaguarda de la privacidad en las operaciones;
- las transacciones se computen y almacenen en el propio dispositivo;
- no se requiera correr localmente un nodo con la totalidad de la cadena de bloques para tales fines.

Se acotó el alcance de criptomonedas a analizar a las principales privacy coins: Zcash y Monero.

Por otra parte, dado el amplio espectro de temas que pueden abordarse al respecto de la conjunción de dos dominios de aplicación como blockchain y desarrollo de aplicaciones, se definieron una serie de temas que quedarían fuera del alcance del trabajo:

[1] Término elegido por las autoridades del Banco Central de la República Argentina para referirse a las criptomonedas.

- Seguridad informática relacionados con ataques a servidores de clientes livianos.
- Criptografía de Curvas Elípticas.
- Privacidad en la comunicación de datos entre aplicaciones cliente y servidores.
- Capa de presentación, experiencia de usuario y diseño de interfaces gráficas.
- Frameworks específicos de aplicaciones móviles.
- Discusión sobre implementación de capa de presentación (Model View Controller, Model View Presenter, Model View ViewModel, View Interactor Presenter Entity Router, etcétera).

6.1. Conclusiones

6.1.1. Necesidad de establecer un lenguaje común hacia fuera del dominio de aplicación

Por instantes, a quienes desarrollan su trabajo en el campo de las criptomonedas, les parece que ellas han existido por mucho tiempo. Al momento que se redactan estas líneas solo han transcurrido cinco años desde la creación de Zcash, siete del surgimiento de Monero y doce del lanzamiento de Bitcoin. Desde dentro de este campo en ebullición muchas veces se crea la ilusión de que las criptomonedas están alcance de todo el mundo. Ronda la idea que detrás de los bloques que brotan de los mineros están las transacciones de un mundo de usuarios diverso, descentralizado, donde nadie puede ser censurado, como lo es Internet: un aparato global que nadie puede ya (en teoría) apagar ni detener. Desafiando los limites temporales que imponen husos horarios, miles de desarrolladores van creando a fuerza de commits y pull requests el deseo de un sistema económico diferente que trascienda fronteras, derribe intermediarios y provea a cada ser humano la posibilidad de ser un par dentro de lo que fuera una vez un entramado económico signado por la injusticia y la desigualdad.

Cuando se levanta la mirada fuera del ecosistema "Crypto", en las charlas de café y los encuentros que la pandemia confinó a la virtualidad, el desarrollador dedicado a las criptomonedas puede apreciar que su incursión en este campo, a diferencia de lo aparente de su cotidiano, le ha agregado una complejidad a su presentación ante la pregunta: *"¿Y vos a qué te dedicás?"*

Incluso entre los propios colegas, existe un desconocimiento y descreimiento generalizado sobre las criptomonedas y su funcionamiento, cosa que se acrecienta cuando se agrega la categoría de *Privacy Coins*.

La idea de este trabajo comenzó con la estructura habitual: Marco teórico, Estado del Arte, Desarrollo y Conclusiones. Sin embargo ya en las primeras etapas de la revisión sistemática de la literatura, comenzaron los desafíos de observar que las consultas a las bibliotecas virtuales devolvían listados sin demasiados resultados que cumplieran los criterios ni las expectativas planteadas inicialmente. Posteriormente en el trabajo de descripción de requerimientos, historias de usuario, etc. quienes brindaron su tiempo a revisar esos textos iniciales encontraban que el área les era difícil de abordar. Así fue como surgió el capítulo 2 introductorio a las criptomonedas.

Una conclusión temprana a la que se arriba en los inicios del desarrollo de esta tesis, es que no bastaba solamente con hablar de lo específico de la arquitectura de software, de componentes, diagramas de clases, sino que debía correrse del microclima y el *Hype*[2] tan característico de las tecnologías emergentes y las finanzas y poder responder una pregunta más elemental.

De cumplir con la promesa de adopción masiva, serán muchos los apasionados del software que se verán ante la tarea de tener que diseñar y desarrollar una "billetera electrónica non-custodian para criptomonedas con privacidad en el libro contable", donde lo único que les sea familiar en esa definición de producto, sean las dos primeras palabras.

Es así que quizás un aporte derivado de este trabajo sea la recolección de conceptos, terminología, requerimientos y saberes que hacen al dominio de las criptomonedas y están dispersos a través de muchos proyectos, artículos, bibliografía, comentarios en repositorios de código e inclusive, comentarios en foros y redes sociales, condensándolos en un hilo conductor: El desarrollo de un cliente liviano para dispositivos móviles que pueda proveer la funcionalidad de una wallet para Privacy Coins. En definitiva, este trabajo busca ser aquel que su autor hubiera deseado encontrar luego de que le encomendaran el trabajo de crear una Wallet para una Privacy Coin.

6.1.2. De los desafíos de la Revisión Sistemática de la literatura en el dominio de aplicación

En la propuesta de trabajo se planteó la realización de una revisión de la literatura existente en base a tres interrogantes:

- ¿Qué diferencias de requerimientos funcionales y no funcionales existen entre wallets que soportan Privacy Coins en Desktop, Mobile y Web?
- ¿Qué experiencias hay del cómputo de transacciones del lado del cliente?
- ¿Qué esquemas de respaldo de cadena de bloques se pueden utilizar?

El capítulo 3 describe la metodología utilizada, las búsquedas de bibliografía y el análisis sobre los resultados de las mismas. Durante su realización, conforme se efectuaban las búsquedas siguiendo la metodología en [23] y utilizando la herramienta Scolr[3], se recolectaron numerosos artículos provenientes de los repositorios Scopus, ACM Library e IEEE Explore, que una vez pasados por el tamiz de los criterios de selección quedaban excluidos del estudio. Habiendo ya optimizado los parámetros de búsqueda, se realizó un ajuste en los criterios de exclusión originales:

- Se refiere una o más monedas listadas en el compendio de privacy coins. [8]
- El proyecto se encuentra desplegado en producción (es decir, en el mercado).
- Utiliza blockchain.

[2] Es una expresión utilizada para describir un estado de expectativas y exaltación generado alrededor de un producto u objeto que responde a mecánicas más propias de las especulación y el marketing que de lo fáctico.

[3] http://scolr.cientopolis.org/

- la privacidad no es obtenida mediante técnicas de «Anonimización» y/o manipulación de transacciones públicas como *CoinJoining* o *Snowballing*.

Se los reemplazó por otros menos ambiciosos, que arrojaran un listado de más amplio de artículos a revisar.

- Refiere una o más monedas listadas en el compendio de privacy coins. [8]
- Refiere a métodos de pago con dispositivos móviles
- Aborda el asunto de la privacidad en las transacciones realizadas por medios electrónicos.
- Discute la utilización de clientes livianos en tecnologías móviles.

Una de las conclusiones tempranas de esta etapa del trabajo es la necesidad de ampliar el espectro de fuentes necesarias para una revisión exhaustiva de la literatura (ver sección 3.5.2 donde se exponen las conclusiones completas) donde se incluyan otros repositorios dedicados a "pre-prints", literatura gris o documentación en repositorios de código fuente. Esta problemática se presenta y describe también en las propuestas de trabajo a futuro en las secciones próximas.

En base a las preguntas formuladas y los artículos relevados, se pudieron encontrar los siguientes puntos de interés:

- Es necesario poder contar con rápida sincronización y la reducción de la dependencia del consenso de red respecto del ledger de la blockchain [40].
- Para proveer una mayor seguridad, se debe utilizar un segundo dispositivo como "cold wallet" delegando así el poder de gasto en un artefacto que no esté conectado a la red [37].
- Los filtros de Bloom, no han demostrado ser eficaces a la hora de preservar la privacidad de los clientes livianos que realizan consultas sobre transacciones pertinentes a las claves de los clientes livianos en los esquemas SPV de Bitcoin [32].
- Este último punto puede ser compensado por técnicas que mejoren el anonimato en las comunicaciones entre clientes livianos y la red de pares mediante la utilización de canales seguros como BOLT o zkChannels [27].
- Un estudio sobre el "Moneywork" o uso del dinero concluyó que no hay disociación entre el uso de una moneda física de curso legal y su versión digital, por tanto debe esperarse que los usos de las monedas tradicionales sean extrapolables a sus versiones digitales [30].
- a fin de reducir los tiempos de sincronización y la información expuesta en las cadenas de bloques es posible explorar la eliminación del ledger público y la utilización de un modelo de asiento contable efímero [40].

6.1.3. Relevamiento de requerimientos.

Con el fin de contar con un panorama completo de las funcionalidades esperables para una billetera que opere con privacy coins se planteó un análisis de aplicaciones que estuvieran recomendadas en los sitios oficiales de Zcash y Monero. Se seleccionaron las aplicaciones Zec Wallet Lite (desktop, iOS, Android), ECC Wallet (Android y iOS) y la multi-moneda Unstoppable (android y iOS) por el lado de Zcash; por Monero, Cake Wallet (Android y iOS) y Monerujo (Android).

Dentro del primer análisis del código fuente se observó que en todos los casos había una clara separación entre el dominio de aplicación y todo aquello que concierne al manejo puro de la criptomoneda. Para el caso de Zcash esta separación era explicita mediante un CLI o un SDK, y en el caso de Monero, wrapper sobre un FFI hacia una API de C++. Es por ello que además del trabajo respecto de los requerimientos de las aplicaciones propiamente dichas, se realizó un relevamiento sobre el SDK de Zcash y se propuso una estructura homónima para el caso de Monero, donde particularmente el uso de la interfaz "wallet2_api.h" se encontraba diseminada en miles de proyectos de código abierto sólo en la plataforma GitHub.

Al profundizar en este análisis se pudo concluir en la necesidad de encapsular y desacoplar aquellas interfaces que son vasos comunicantes con la operación de la blockchain a nivel protocolo, de forma tal que interfaces de programación se expresaran entorno a la funcionalidad de las wallets y no respecto de las particularidades de los protocolos.

Muchos requerimientos *funcionales* desde el punto de vista del protocolo como formatos de codificación de datos, algoritmos y pruebas criptográficas, son *no funcionales* cuando se los analiza desde la mirada de un cliente liviano. Por ende es conveniente abstraerlos en su finalidad última y exponerlos mediante una API de manera tal que al variar las implementaciones y particularidades a nivel de protocolo, el impacto de estos cambios sobre las wallets sea de mínimo a nulo.

A modo de ejemplo puede pensarse en los distintos cambios a los formatos de transacciones que han realizado los protocolos de Zcash y Monero. El cambio de la codificación de una dirección, un id, la selección de un algoritmo de hashing, son cuestiones centrales a nivel de protocolo, pero desde el punto de vista de una historia de usuario, son poco relevantes. Esto puede apreciarse visualmente cuando se comparan las tablas de los requerimientos base 4.4, con los requerimientos de una wallet Zcash 7.3 y la tabla que refiere al SDK de Zcash 7.2.

Las secciones 4.5 y 4.6 analizan las particularidades del código que está más cercano al protocolo de Zcash y Monero en las billeteras, y propone una estructura para el caso de Monero que replica la propuesta de Zcash.

Volviendo al relevamiento de requerimientos a nivel de aplicación-billetera, el análisis de las distintas wallets arrojó una lista de requerimientos que se agruparon en forma de temas (sección 4.2), épicas (sección 4.3) e historias de usuario (sección 4.4) en base a la metodología Scrum. En esta última sección se condensan en una tabla las historias redactadas en forma de "tarjeta" en formato *COMO* - actor - *QUIERO* - acción a realizar *PARA* - Objetivo a lograr. El relevamiento también produjo como resultado una tabla comparativa de funcionalidades que puede hallarse en el anexo 7.1. En dicha tabla se reproduce la agrupación de las historias de usuario y se indica su presencia

en cada una de las aplicaciones.

Las comparación de las distintas aplicaciones permitió comprobar la centralidad de ciertos aspectos cruciales que el desarrollo de un cliente liviano tiene que abordar. Se destacan cuestiones como la gestión de las claves privadas del usuario (épicas 01, 02 y 03), el manejo de las operaciones y sus resultados, la recepción (E-04) y envío (E-06) de fondos y el balance (E-05) resultante e historial de transacciones (E-07). Por último, la sincronización (E-08), que es proceso que materializa los datos que permiten realizar todas las funcionalidades anteriores.

Estas épicas son los trazos generales de la arquitectura de referencia propuesta.

6.1.4. Sobre el desarrollo de la arquitectura de referencia

El capítulo 5 comienza con la pregunta respecto de si la propuesta es una arquitectura o un framework. Para ello se recurre a dos bibliografías que son pilares del desarrollo de software: el «Gang-of-Four»[52] y "Code Complete"[64], a fin de encontrar definiciones de ambos conceptos y ensayar una respuesta. La propuesta tiene muchos puntos de contacto con ambas definiciones, y puede concluirse que si bien origen del trabajo es delinear los requerimientos de una arquitectura para billeteras electrónicas para AECs, aunque ello no quita que de ellos pueda derivar la creación de un framework que provea el marco técnico para la adopción de distintas criptomonedas del estilo.

En la sección 5.3 se presenta la arquitectura propuesta en su forma completa y luego se resaltan sus partes. En la sección 5.3.1 se aborda el punto de ingreso y vínculo con la aplicación y el sistema operativo que la aloja, y cómo accederá a los recursos de los cuales depende; luego en punto siguiente (sección 5.3.2) se describe brevemente cómo se proveerá el acceso a las claves de usuario. Consecuentemente, la sección 5.3.3 aborda porción de la propuesta abocada al manejo de frases semilla y su relación con las claves de lo usuarios. Las secciones 5.3.4 y 5.3.5 describen cómo la propuesta complementa los requerimientos que hacen a la inicialización de los recursos necesarios para interactuar con una blockchain y de los componentes que realizarán dicha acción, fundamentalmente la sincronización que se ubica como la parte fundamental de la operatoria de una wallet, concluyendo que:

> *"[..] una wallet es un punto de vista de la blockchain, desde un bloque de nacimiento hasta el bloque se que ubica en el extremo más reciente de la cadena. El punto de vista lo definen las claves públicas con las que el usuario desea hacer el recorrido, con los cuales encontrará los elementos que le son propios. Para proponer nuevos elementos (transacciones con inputs y outputs) debe hacer uso de sus claves privadas."*

En la sección 5.3.6 se discuten varias consideraciones para el manejo de errores en una wallet non-custodian, con una recolección de los posibles escenarios de falla y agrupándolos en una taxonomía de errores posibles en cinco alternativas principales: pérdida de fondos; inconsistencia local con inutilización permanente; con inutilización temporal, errores técnicos locales recuperables y en servidor.

A modo de proveer un área de consulta o resumen, la sección 5.4 recapitula la arquitectura propuesta parte por parte tomando como base el patrón utilizado en

el libro “Patrones de Diseño” [52], discutiendo para cada componente su Intención, Motivación, Aplicabilidad, Estructura, Participantes, Colaboraciones con otras entidades, consecuencias devenidas de su implementación, código que la ejemplifica, usos y temas relacionados. Realizar este apartado tiene como intención principal retomar esta estructura conocida en el ámbito de la Ingeniería de Software como una forma de acercamiento a la novedad de la propuesta desde un lugar conocido.

Por último, la sección 5.5 se vuelven a visitar algunos de los casos de estudio analizados para el relevamiento de requerimientos, esta vez desde una mirada de “alto nivel”, haciendo referencia a cómo distintas implementaciones de clientes livianos de Zcash manejan la complejidad propia de una privacy coin. Independientemente del enfoque utilizado, se observa en líneas generales la existencia de historias de usuario de base, que son implementadas por interfaces que median entre la generalidad de una billetera desde el punto de vista del usuario o de los desarrolladores de la aplicación que estos utilizan y la particularidades propias de las abstracciones que cada criptomoneda propone.

Este trabajo logra poner en relevancia esa dinámica describiéndola en sus distintas formas: Requerimientos, Épicas, Historias de Usuario y por último una propuesta de arquitectura dividida en distintos tipos de componentes que las cumplimentan.

6.2. Trabajo a futuro

6.2.1. Ampliación de las fuentes de la SLR

Durante la realización de la propuesta de este trabajo se ha identificado que es relevante contar con fuentes accesorias a los buscadores indexados tradicionales (Scopus, ACM, IEEExplore, ScienceDirect, etc) puesto que mucha de la información técnica se encuentra en la propia documentación de los proyectos en cuestión. A lo cual se proponen las siguiente fuentes para la SLR:

- Repositorios de Software Libre y/o de Código Abierto.
- White Papers y documentación en sitios web de los propios proyectos a analizar.

Para ello se requiere un estudio dedicado exclusivamente al ámbito de las revisiones sistemáticas de literatura, que incluya una metodología para incluir la "literatura gris" de forma estructurada con criterios de validación y valoración de las fuentes.

6.2.2. Marco de Métricas de Referencia para la evaluación del estado de Proyectos de Criptomonedas:

Busca responder una pregunta esencial al momento evaluar tecnologías blockchain: ¿Cuáles son los proyectos más relevantes en el universo de *privacy coins*, cuáles de ellos deben ser sujeto de estudio para la arquitectura de referencia?.

Al momento de redacción del presente trabajo, se relevaron cerca de cien proyectos de *Privacy Coins* de Código Abierto. En base al sitio de cotizaciones *CryptoSlate* [70] y un listado confeccionado por la comunidad de Zcash [8], se pudo constatar que están activos, sus nodos encuentran desplegados y productivos (se detalla la lista a Junio 2020 en [71]). Se observó que cada uno de ellos refiere a sus artículos de literatura gris (*brochures*, *white papers*, documentación en repositorios, artículos periodísticos) que describen (aunque con fines promocionales) aspectos técnicos no hallados en literatura académica. La ausencia de información curada y no subjetiva sobre los proyectos de criptomonedas dificulta su análisis comparativo en términos de adopción tecnológica.

6.2.3. Evaluación inclusión del protocolo 'FlyClient' en la presente propuesta

El documento ZIP-0221 [72] describe el protocolo *FlyClient* [73] en los nodos de Zcash. El mismo incluye una modificación al encabezado de los bloques mediante la cual se permiten realizar validaciones de pruebas dentro de los clientes ultra-livianos. La utilización de Merkle Mountain Range (MMR) habilita que los clientes livianos puedan validar: un bloque recibido de un full-node; la inclusión de dicho bloque en la cadena; metadatos dentro de cualquier bloque o rango de bloques.

6.2.4. Evaluación de esquemas sociales para la preservación y recuperación de claves privadas

En la sección 5.3.3 se discute brevemente el asunto de la conveniencia (o no) de las frases semillas como método de recuperación y conservación de claves de usuario. Esta

forma de recuperar los bytes originales de un conjunto de claves pertenecientes a una billetera, es un estándar que surge de la propia inconveniencia del manejo de bytes por parte de los seres humanos. Consiste en una transformación de los mismos en una forma más familiar a la escritura occidental, que sin dudas es más conveniente que una secuencia de bytes binarios o hexadecimales, pero que no cubre las tantas aristas de la vida humana, como la pérdida, robo o hurto de las mismas, ni tampoco prueban que sean resistentes al paso del tiempo, tanto del material, como de la tecnología, ni de los propios usuarios, pues si una certeza tienen los humanos en la vida es que en un momento perecerán. Todos los aspectos sociales que hacen al uso de una billetera electrónica escapan a las simples y ampliamente adoptadas frases semilla. Este ámbito debe ser investigado en profundidad para poder asegurar la adopción masiva de las criptomonedas como herramienta de transformación social.

6.2.5. Estrategias de protección de la privacidad del tráfico de red

Uno de los problemas más grandes de los clientes livianos es su dependencia con servidores que brinden soporte para una verificación simple de los pagos que corresponden a una determinada wallet. El documento de modelo de amenazas para el protocolo de clientes livianos [74][4] enumera los tipos de atacantes estipulados de forma jerárquica desde mayor a menor poder de daño, siendo el mayor aquel que ha controlado un servidor *LightWalletD* legítimo y puede servir respuestas directamente a wallets, hasta el menos lesivo, aquel que sólo tiene conocimiento de la dirección de una wallet y puede observar la actividad de red pública. También condensa un larga lista de asunciones sobre lo que cada atacante no pude realizar sobre su objetivo, llamadas "invariantes de seguridad". El modelo describe al monitoreo de actividad de red como el ataque más probable. Este es quien realiza acciones sobre un determinado usuario en base a conocer una dirección a la cual enviar fondos, y mediante la cual puede realizar acciones que conociendo el comportamiento típico de los clientes disponibles para ella, empezar deducir movimientos públicos para romper el anonimato de ese usuario. Resolver este tipo de cuestiones es de vital importancia para garantizar niveles altos de privacidad y anonimato de los usuarios de criptomonedas y en especial de aquellas que se enfocan en este aspecto.

[4] ver anexo 7.5

Capítulo 7

Anexos

7.1. Tabla de Requerimientos por wallet

	epic	epic id	Story ID	ZecWallet	Cake Wallet	Monerujo	ECC Wallet	Unstoppable
Gestión de claves públicas y privadas	nueva wallet	E-01	1.0	✔	✔	✔	✔	✔
	respaldo de frase semilla	E-02	2	✔	✔	✔	✔	✔
			3	✔	✔	✔	✔	✔
			4	✔	✔	✔	✔	✔
	restaurar wallet desde frase	E-03	5	✔	✔	✔	✔	✔
operaciones	recibir fondos	E-04	6	✔	✔	✔	✔	✔
			7	✔	✔	✔	✔	✔
estado de la wallet	balance	E-05	8	✔	✔	✔	✔	✔
			9	✔	✔	✔	✔	✔
			10	✔	✔	✔	✔	X
			11	✔	✔	✔	✔	✔
operaciones	enviar fondos	E-06	12	✔	✔	✔	✔	✔
			13	✔	✔	✔	✔	✔
			14	✔	✔	✔	✔	✔
			15	✔	✔	✔	✔	✔
			16	✔	✔	✔	✔	✔
			17	✔	✔	✔	P	✔
			18	✔	✔	✔	✔	✔
estado de la wallet	historial de transacciones	E-07	19	✔	✔	✔	✔	✔
			20	✔	✔	✔	✔	✔
			21	✔	✔	✔	✔	✔
			22	✔	✔	✔	✔	✔
			23	✔	✔	✔	✔	✔
operaciones	sincronizar	E-08	24	✔	✔	✔	✔	✔
			25	✔	✔	✔	✔	✔
			26	✔	✔	✔	✔	✔

Cuadro 7.1: Matriz comparativa de historias de usuario presentes en cada wallet analizada.

[1]

[1]Referencia: ✔ Disponible, X no disponible, P (pendiente)

7.2. Historias de usuario de kit de desarrollo de Zcash

epic	id	COMO	QUIERO	PARA	Criterios de Aceptación	Req	nota
E-08	1.0	developer	poder saber la altura de la blockchain	poder saber si estoy al día con la blockchain	existe un método para conectarse a Lightwalletd de forma asincrónica y obtener la ultima altura	R-7	[2]
E-01	2.0	developer	poder inicializar una cuenta con una array de bytes	para crear una nueva wallet o restaurar una existentes	puede obtenerse una dirección Zcash determinísticamente desde una array de bytes dado	R-12 R-14 R-15 R-16 R-17 R-20	[3]
E-06	3	developer	enviar una transacción sapling z a sapling z	enviar transacciones a traves de la red	construir una transacción o indicar fallo enviar la transacción por lightclient protocol o indicar fallo enviar la transacción a lightwalletd o indicar fallo confirmar que se incluyo en un bloque (mined) - ¿requiere sincronizar a ultima altura	R-18 R-19 R-25 R-26 R-27	[4]
E-98	4	developer	saber cuando una transacción ha sido minada	para poder notificar al usuario en la interfaz	es posible suscribirse a un evento que informa si una transaccion perteneciente a una de las inbound viewing keys ha sido minada	R-5 R-14 R-18 R-19 R-22 R-25 R-27 R-28	[5]
E-08	5	developer	poder delegar el manejo de reorganizaciones de la cadena de bloques	poder estar al dia con la ultima cadena de bloques valida	detectar una reorganización redescargar los ultimos 100 bloques y re validar	R-14 R-18 R-19 R-25 R-29 R-30	[6]

[2] https://github.com/zcash/ZcashLightClientKit/issues/16
[3] https://github.com/zcash/ZcashLightClientKit/issues/15
[4] https://github.com/zcash/ZcashLightClientKit/issues/18
[5] https://github.com/zcash/ZcashLightClientKit/issues/24
[6] https://github.com/zcash/ZcashLightClientKit/issues/25

E-07	6	developer	quisiera poder acceder a las transacciones almacenadas localmente	para poder mostrarlas al usuario	existe una API para obtener las ultimas N transacciones enviadas, recibidas y pendientes con un offset	R-14 R-18 R-19 R-21 R-22 R-25	[7]
E-08	7	ing de seguridad	debe poder ajustarse el tiempo de actualización de bloques	poder ajustarse a los cambios en el consenso que establezca el protocolo	existe una forma de parametrizar el intervalo de tiempo entre consultas por nuevos bloques	R-33	[8]
	8	developer	poder acceder a logs de eventos y errores	poder observar el comportamiento del SDK y diagnosticar errores	la generación de logs es transparente a su implementación debe poder desactivarse de ser necesario	R-34	
E-07	9	developer	poder acceder al balance total de una viewing key	para poder mostrarlas al usuario	existe un llamado en la api para obtener el balance en Zatoshi.	R-14 R-16 R-18 R-19 R-20 R-21 R-22	
E-05	10	developer	acceder al balance verificado de una viewing key	para poder motrarlo al usuario	existe un llamado en la api para obtener el balance en Zatoshi.	R-14 R-16 R-18 R-19 R-20 R-21 R-22	
E-04	11	developer	acceder a la direccion sapling z-address en base a la spending-key generada con los bytes provistos	para poder mostrarla al usuario para que reciba fondos	existe un llamado en la api para obtener la sapling z-address en forma de string.	R-12 R-13 R-15 R-16 R-17	

[7] https://github.com/zcash/ZcashLightClientKit/issues/26
[8] https://github.com/zcash/ZcashLightClientKit/issues/46

E-08	12	developer	poder desencriptar una transacción completa del full node	para poder acceder acceder a la información que no se encuentra presente en el modelo compacto	existe un paso durante la sincronización que dada la detección de una transacción, requiere la versión completa al servidor para su desencripción y extracción del memo e inputs para poder mostrar el historial de transacciones de manera correcta en todos los dispositivos que sincronicen las claves proporcionadas por el usuario	R-22	

Cuadro 7.2: Historias de Usuario (desarrollador) del Kit de Desarrollo de Zcash

7.3. Listado de Historias de usuario de una wallet Zcash

Theme	Epic	E-id	S-id	COMO	QUIERO	PARA	Criterios de Aceptación	Req.
Gestión de claves	nueva wallet	E-01	1	usuario	crear una nueva wallet	poder utilizar Zcash	Al iniciar la aplicación el usuario ve una pantalla que le informa que debe iniciar una wallet. Hay un botón para crear una wallet nueva con una frase semilla aleatoria	R-12 R-13 R-15 R-16 R-20
	respaldo de frase semilla	E-02	2	Ing. seguridad	un recordatorio a los usuarios que resguarden su frase	que los usuarios resguarden efectivamente su frase semilla	Al crear una nueva wallet se debe recordar al usuario que debe poner la frase semilla a resguardo. La Pantalla no bloquea al usuario de continuar	R-13
			3	usuario	poder ver mi frase semilla y mi bloque inicial	poder resguardar mi semilla	Se muestra al usuario la frase semilla BIP-39 y el bloque inicial permite copiar la frase al portapapeles	R-12 R-13
			4	usuario	poder copiar mi frase semilla al portapapeles	resguardarla más rápidamente	se muestra un botón donde explícitamente se indique la acción de copiar al portapapeles	R-12 R-13
	restaurar wallet desde frase	E-03	5	usuario	poder restaurar mi billetera desde una frase y bloque inicial	poder recuperar mi cuenta completamente y hacer uso de mis fondos	Al iniciar la aplicación el usuario ve una pantalla que le da la opción de restaurar la wallet desde una frase semilla existente. (a) Se presenta una pantalla donde puede ingresar la frase semilla o copiarla del portapapeles; (b) El usuario debe poder ingresar el bloque inicial o bien restaurar desde el bloque inicial de la red; (c) La pantalla valida que la semilla ingresada sea valida acorde a BIP-39 y lo informa al usuario; (d)El usuario no puede iniciar la restauración si la frase y/o el bloque inicial ingresado son inválidos	R-12 R-13

Theme	Epic	E-id	S-id	COMO	QUIERO	PARA	Criterios de Aceptación	Req.
operaciones	recibir fondos	E-04	6	usuario	quiero ver mi Z-Address como QR	que otro usuario pueda escanearla y enviarme fondos	El usuario puede acceder a una pantalla donde: (a) se muestra a su Z-Address como un QR; (b) se anuncia que es su dirección blindada; (c) se indica que esa dirección es para recibir zcash; (d) se muestra la dirección de forma textual.	R-17 R-36
			7	usuario	quiero ver mi T-Address como QR	que otro usuario pueda escanearla y enviarme fondos	El usuario puede acceder a una pantalla donde: (a) se muestra a su T-Address como un QR; (b) se anuncia que es su dirección Transparente; (c) se indica que esa dirección es para recibir ZEC/TAZ; (d) se muestra la dirección de forma textual	R-17 R-36
			8	usuario	quiero copiar mi Z-Address al porta papeles	compartirla por otros medios en forma textual	El usuario puede ingresar a una pantalla a copiar su dirección Z al portapapeles; (a) el usuario puede ver la dirección en su forma textual; (b) existe un botón para copiar el texto fácilmente; (c) hay una confirmación visual que la copia al portapapeles ha sido realizada	R-17
			9	usuario	quiero copiar mi T-Address al porta papeles	compartirla por otros medios en forma textual	(a) El usuario puede ingresar a una pantalla a copiar su dirección T al portapapeles; (b) el usuario puede ver la dirección en su forma textual; (c) existe un botón para copiar el texto fácilmente; (d) hay una confirmación visual que la copia al portapapeles ha sido realizada	R-17
estado de wallet	balance	E-05	10	usuario	poder visualizar los fondos pendientes	confirmar que me han enviado fondos	El usuario puede conocer que ha recibido fondos y que están pendientes de confirmación.	R-18 R-19 R-20 R-21 R-22
			11	usuario	poder visualizar el cambio	para poder diferenciarlo de mis fondos disponibles	el usuario puede conocer que ha enviado fondos y que tiene 'cambio' esperando a ser confirmado	R-18 R-19 R-20 R-21 R-22

Theme	Epic	E-id	S-id	COMO	QUIERO	PARA	Criterios de Aceptación	Req.
			12	usuario	ver una pantalla especial si no tengo fondos	no realizar acciones innecesarias	El usuario debe poder visualizar que no tiene fondos disponibles. Esta acción es no bloqueante para todas aquellas acciones que no impliquen enviar fondos	R-18 R-19 R-20 R-21 R-22
			13	usuario	poder visualizar mis fondos disponibles	poder enviarlos	El usuario puede visualizar sus fondos disponibles en unidades decimales	R-18 R-19 R-20 R-21 R-22
operar	enviar fondos	E-06	14	usuario	escanear un código QR de otra wallet	poder enviar fondos a ella	El usuario puede acceder a un botón que active la cámara del dispositivo, solicite permisos para acceder a ella y le permita escanear un código QR y; (a) la pantalla puede determinar la disponibilidad de la cámara y si ha sido autorizado su uso; (b) si no esta autorizado el uso lo informa, junto con instrucciones de como autorizar nuevamente su uso; (c) al mostrar la captura de video de la cámara se muestra una guía/marco de como enfocar el código QR; (d) el código QR es escaneado y validados sus contenidos informando si el QR escaneado corresponde a una dirección de Zcash válida o no; (e) permite salir al usuario sin escanear una dirección	R-23
			15	usuario	pegar una dirección desde el portapapeles	enviar fondos a ella	Para enviar fondos, el usuario dispone de un campo de texto donde puede escribir y/o pegar una dirección Zcash desde el portapapeles. (a) la dirección se valida al pegarla; (b) se muestra un error no bloqueante en caso de invalidez; (c) se muestra un mensaje de confirmación si la dirección ese correcta	

Theme	Epic	E-id	S-id	COMO	QUIERO	PARA	Criterios de Aceptación	Req.
			16	usuario	ver si la dirección ingresada es correcta	validar que me proporcionaron un dato correcto para el envío de fondos	Para enviar fondos, el usuario dispone de un campo de texto donde puede escribir y/o pegar una dirección Zcash desde el portapapeles. (a) la dirección se valida al pegarla; (b) se muestra un error no bloqueante en caso de invalidez; (c) se muestra un mensaje de confirmación si la dirección ese correcta	R-24
			17	usuario	poder ingresar un monto	enviar esos fondos a otra billetera	Para enviar fondos el usuario dispone de un campo de texto donde puede escribir y/o pegar un monto de ZEC en decimales. (a) Se muestra una teclado numérico acorde; (b) el monto ingresado se valida contra los fondos disponibles y los fondos necesarios para poder enviar una transacción a través de la red; (c) El resultado de la validación se muestra en pantalla al usuario	
			18	usuario	poder ingresar un Memo para la transacción	adjuntar un mensaje encriptado a mi transacción	Al enviar fondos a una Z-Address, el usuario debe tener la opción de adjuntar un texto de 512 caracteres UTF-8 como máximo. (a) El memo puede ser dejado en blanco; (b) no tiene restricciones dentro del set de caracteres UTF-8; (c) no debe permitirse ingresar mas de 512 caracteres UTF-8; (d) debe mostrarse cuantos caracteres tiene en total el mensaje escrito	R-26
			19	usuario	ver el progreso de mi transacción mientras se crea	saber que la aplicación esta funcionando	mientras se crea al transacción se debe mostrar una pantalla de espera al usuario donde se mantenga el estatus de la operación	R-26
			20	usuario	recibir una confirmación de envío de la transacción	poder confirmar que se ha enviado sin errores	luego de enviar la transacción la aplicación debe mostrar un mensaje confirmando que la operación ha sido concretada exitosamente	R-5 R-6

Theme	Epic	E-id	S-id	COMO	QUIERO	PARA	Criterios de Aceptación	Req.
			21	usuario	ver los errores que hayan ocurrido durante el envío de la transacción	poder confirmar que no se ha enviado	en caso de que ocurra un error en el proceso de envío de una transacción, la aplicación debe mostrar el error al usuario mediante un mensaje. El mensaje debe poder ser descartado y permitir al usuario continuar con el uso de la aplicación.	R-5 R-6
			22	usuario	recibir un link (URI) con información para un pago	poder realizar una transacción a un tercero con toda la información requerida para esta,sin la necesidad de ingresar los datos yo mismo	(a) el usuario acciona el link desde otra aplicación, ocasionando que se lance la wallet; (b) la aplicación muestra un mensaje con los detalles del link a ejecutar siendo el usuario que confirma o descarta la acción; (c) En caso favorable se deriva al usuario a la pantalla de envío de transacción con los datos preingresados presentes en el URI lo cuales deben ser suficientes para poder realizar la operación propuesta; (d)En caso negativo el usuario es trasladado a la pantalla inicial	
estado de wallet	historial de transacciones	E-07	23	usuario	ver las transacciones pendientes de confirmación	conocer los detalles de las transacciones que he enviado o recibido recientemente	el usuario debe poder acceder a un listado de transacciones en el cual se muestren sus transacciones pendientes de confirmación. Se considera transacción pendiente a toda transacción que ha sido enviado pero no aún minada, o bien, que ha sido minada pero que cuenta con menos de 10 confirmaciones en la blockchain	R-5 R-6 R-14 R-18 R-19 R-22 R-25
			24	usuario	ver las transacciones fallidas en mi historial	poder conocer el historial de envíos recientes con exactitud	el usuario debe poder acceder a un listado de transacciones en el cual se muestren sus transacciones salientes fallidas y de allí acceder a su detalle. la disponibilidad de estas transacciones, es solo local al dispositivo desde donde se han enviado.	R-25

Theme	Epic	E-id	S-id	COMO	QUIERO	PARA	Criterios de Aceptación	Req.
			25	usuario	ver las transacciones recibidas históricamente	poder conocer el historial de transacciones recientes con exactitud	El usuario debe poder acceder a un listado de transacciones en el cual se muestren sus transacciones recibidas históricamente para la frase semilla de la presente wallet. La información debe reflejar todas las transacciones recibidas desde el bloque inicial correspondiente a esta frase semilla hasta el bloque actual de la blockchain	R-5 R-6 R-14 R-18 R-19 R-22 R-25
			26	usuario	ver el monto, altura, memo y fecha de las transacciones enviadas y recibidas	poder conocer el historial de transacciones recientes con exactitud	el usuario debe poder acceder al detalle de las transacciones correspondientes a su frase semilla.	R-5 R-6 R-14 R-18 R-19 R-22 R-25
			27	usuario	ver el ID de la transacción realizada	para poder verlo en un explorador de bloques de la blockchain	el detalle de la transacción muestra el ID de la transacción correspondiente y permite al usuario copiar la información al portapapeles y también la opción de acceder directamente a un explorador de transacciones en internet mediante el navegador web de la plataforma	R-5 R-6 R-14 R-18 R-19 R-22 R-25
operaciones	sincronizar	E-08	28	usuario	estar al día con la blockchain	poder conocer mi balance actualizado al ultimo bloque minado y disponer de mis fondos	el usuario tiene que estar todo el tiempo al día con la blockchain, sin necesidad de realizar acción adicional que lanzar la aplicación para que esta se actualice.	R-5 R-6

Theme	Epic	E-id	S-id	COMO	QUIERO	PARA	Criterios de Aceptación	Req.
			29	usuario	conocer si la wallet se esta sincronizando con un porcentaje	para saber que la aplicación no esta colgada y que esta trabajando	Se llama sincronizar al acto de recibir los *compact blocks* desde *lightwalletd*. El usuario debe conocer que la aplicación se esta poniendo al día con la blockchain en forma de una barra de progreso. La aplicación debe bloquear toda funcionalidad que no pueda ser accedida correctamente durante esta operación.	R-5 R-6
			30	Ing. seguridad	requiero que la wallet esté siempre actualizada a la 'best chain'	para que no puedan hacerse ataques de 'side-chain'	la aplicación debe estar siempre al día con la mejor cadena candidata de la blockchain. Es decir que de haber reorganizaciones, debe manejarlas de forma transparente, manejando todos los impactos a las transacciones del usuario que esta reorganización pueda causar.	R-2 R-3 R-4 R-5 R-6

Cuadro 7.3: Resumen de historias de usuario surgidas del relevamiento de requerimientos de Zcash

7.4. Listado de Requerimientos relevados para Zcash

#	código	actor	requerimiento
1	R-1	lightwalletd	es un servidor que cumple con los requisitos definidos en el protocolo ZIP-307 [22] para clientes livianos
2	R-2	lightwalletd	provee una interfaz 'protocol buffer' gRPC
3	R-3	lightwalletd	sincroniza la blockchain mediante una nodo Zcash acorde al protocolo y genera los *compact blocks* acorde a ZIP-307 [22]
4	R-4	lightwalletd	maneja cambios en la blockchain y se ajusta a reorganizaciones (reorgs)
5	R-5	wallet	se conecta a la cadena de bloques mediante un servidor light-client protocol
6	R-6	wallet	la wallet se conecta a dicho servidor por una interfaz gRPC [9]
7	R-7	lightwalletd	provee la última altura de la blockchain
8	R-8	lightwalletd	provee transacciones en base a su ID
9	R-9	lightwalletd	provee *compact blocks* en base a su altura (height)
10	R-10	lightwalletd	provee *compact blocks* en base a un rango de alturas
11	R-11	lightwalletd	permite enviar una transacción a la blockchain
12	R-12	wallet	premite generar bytes desde una frase semilla acorde a BIP-39
13	R-13	wallet	premite generar una nueva frase semilla acorde a BIP-39
14	R-14	wallet	contiene una copia local ordenada secuencialemente sin intervalos de los *compact blocks* servidor por lightwalletd
15	R-15	wallet	debe poder derivar una Sapling Spending Key de los bytes generados a partir de la semilla BIP-39 [21] acorde al protocolo Zcash y al ZIP-32 [57]
16	R-16	wallet	debe poder derivar una Sapling Incoming Viewing Key a partir de una Spending Key acorde al protocolo Zcash y al ZIP-32 [57]
17	R-17	wallet	debe poder derivar una T-Address y Z-Address desde uns Sapling Spending Key acorde al protocolo zcash
18	R-18	wallet	debe poder desencriptar un *compact block* y obtener transacciones, notes y nullifiers que den cuenta de créditos y débitos para esa clave privada (spending key)
19	R-19	wallet	debe poder conservar una copia de los derivados de la desencripción del un *compact block*
20	R-20	wallet	debe poder almacenar las claves privadas de forma segura
21	R-21	wallet	debe poder determinar el balance de una determinada Incoming Viewing Key en base a los notes recibidos y enviados
22	R-22	wallet	debe poder desencriptar transacciones del Nodo Full Node de Zcash completando la información de una transacción compacta
23	R-23	wallet	debe poder escanear un código QR
24	R-24	wallet	debe poder determinar si una dirección T-Address o Z-Address es correcta o no
25	R-25	wallet	debe poder desencriptar transacciones compactas en base a una Incoming Viewing Key
26	R-26	wallet	debe poder generar una transacción compacta en base a un ser de Sapling Notes, una Spending Key, un monto en Zatoshi y una dirección Zcash
27	R-27	wallet	debe poder determina que una transacción ha sido minada, es decir incluida en un bloque
28	R-28	wallet	debe poder determinar que una transacción no ha sido minada pasado su tiempo de expiración (20 bloques) y restableces los notes utilizados como no gastados
29	R-29	wallet	debe poder detectar una reorganización en la blockchain mediante el hash previo del bloque compacto
30	R-30	wallet	debe poder retroceder N bloques en la sincronización manteniendo la consistencia de los datos derivados de los bloques compactos almacenados localmente para poder manejar una reorganización de la blockchain
31	R-31	wallet	debe poder obtener los bytes del ID de transacción y mostrarlos en forma reversa en octavos de bits como lo especifica el protocolo de Zcash y mostrarlos hex-encoded
32	R-32	wallet	debe poder utilizar TESTNET o MAINNET de forma transparente para el usuario. los datos de ambos ambientes jamas deben mezclarse

[9] https://grpc.io/

33	R-33	wallet	debe poder establecer un intervalo de tiempo por defecto para consultar a lightwalletd por nuevos bloques acorde al blocktime del consenso del momento
34	R-34	wallet	debe poder loguear información, eventos, errores y mensajes de debug
35	R-35	wallet	debe poder recibir e interpretar URI conformado en base al ZIP-321
36	R-36	wallet	debe poder generar un código QR a partir de un String
37	R-37	wallet	debe poder generar un una ZK-Proof en base a los Sapling notes a ser enviados a 1 o mas Sapling addresses

Cuadro 7.4: Lista de requerimientos de una wallet de transacciones Sapling en Zcash

7.5. Protocolo de Clientes livianos para detección de pagos de Zcash

El ZIP-307 [72] describe el modelo del bloques compactos (*CompactBlock*) cuyo objetivo es permitir que un cliente liviano pueda sincronizar una cadena (compacta) de bloques de forma privada y tomar conocimiento de pagos blindados que sean de su interés. Un bloque compacto solo contiene información necesaria para detectar un pago hacia o desde un dirección Sapling blindada, actualizar la información necesaria para poder generar nuevas pruebas de cero-conocimiento. El protocolo también sugiere la adopción de *Protocol Buffers* como método de comunicación entre cliente y servidor, el cual implementa el servidor Lightwalletd y el prototipo presentado en este trabajo.

7.5.1. Interacción entre un cliente liviano y un servidor SPV (Lightwalletd)

Cuando un cliente liviano se conecta a un servidor lightwalletd. Puede hacerlo desde un checkpoint o desde el primer bloque de la versión del protocolo Sapling (419.200 en mainnet[10] y 280.000 en testnet[11]). Desde allí deberá sincronizar hasta el último bloque conocido. Sapling es el predecesor de Sprout el primer despliegue de Zcash. Son las mejoras que Sapling introduce a la velocidad de verificación y creación de las pruebas de cero-conocimiento las que permitieron que sea posible realizarlas en dispositivos móviles. A modo anecdótico se suele recordar que una prueba en Sprout demoraba unos 40 segundos en hardware especializado cuando su equivalente en Sapling demora solo alrededor de 10 en un celular de gama media del año 2018. [12]

Primeramente de establecer su altura inicial mediante un checkpoint o utilizando las constantes del protocolo conocidas como SAPLING_ACTIVATION, en las cuales se establece una altura inicial para la activación del protocolo *Sapling* dentro de Zcash. A partir de allí el cliente debe ir descargando los bloques compactos, verificándolos y escaneándolos para detectar los pagos perteneciente a las claves de visualización que tiene en su poder. Mediante este proceso, al acercarse al extremo superior de la cadena, podrá encontrarse con alguna reorganización que produce una falla en la validación de los hashes de los bloques compactos contiguos. Estas reorganizaciones son normales, producto de la naturaleza descentralizada de las blockchains. El síntoma que las revela es que el hash del "bloque anterior" en un bloque dado, no coincide con el hash del bloque anterior que el cliente tiene almacenado en su cache. En esos casos los clientes, al igual que lo hacen los *full nodes*, retroceden una cantidad de bloques hacia atrás, vuelven a descargar los bloques necesarios y a validar y escanear la cadena lo cual suele resolver la discrepancia y culminar el proceso de sincronización.

[10] refiere a la red de pares que corre en producción

[11] Refiere a la red de pares que corre en modo de pruebas. La criptomoneda creada en estas redes carece de valor.

[12] todos los network upgrades de Zcash toman sobrenombres derivados de las ciencias botánicas. Hasta el momento de escribir este párrafo los nombres utilizados son: Sprout, Sapling, Blossom, Heartwood y Canopy.

El algoritmo 1 ilustra de forma abstracta este proceso de forma lineal.

Algorithm 1: Como un cliente liviano de Zcash sincroniza la cadena de bloques

Result: Sincronizar hasta ultima altura conocida

$currentHeight \leftarrow loadCheckpointOrSaplingActivation()$;

while $currentHeight < latestHeight()$ **do**
  $storeCompactBlocks(getBlockRange(currentHeight + 1...currentHeight + BATCHSIZE))$;
  if *validateCombinedChain()* **then**
    $currentHeight \leftarrow scanblocks()$;
  else
    $currentHeight \leftarrow handleReorg()$;
  end
end

La especificación completa de las interacciones entre un cliente liviano y un servidor lightwalletd se definen en su totalidad en el ZIP-307 [22]

Índice alfabético

Bibliografía

[1] Tomas Sander and Amnon Ta-Shma. Auditable, anonymous electronic cash. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 555–572, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf.

[3] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, page 397–411, USA, 2013. IEEE Computer Society.

[4] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, page 459–474, USA, 2014. IEEE Computer Society.

[5] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[6] Nicolas Van Saberhagen. Cryptonote v 2.0, 2013.

[7] Andreas M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain.* O'Reilly Media, Inc., 2nd edition, 2017.

[8] List of all privacy coins/projects. Zcash Community Forum, Apr 2019. `https://forum.zcashcommunity.com/t/list-of-all-privacy-coins-projects/33159`.

[9] Chris Eidhof, Matt Gallagher, and Florian Kugler. *App Architecture: iOS Application Design Patterns in Swift.* CreateSpace Independent Publishing Platform, 1st edition, 2018.

[10] Kenneth S. Rubin. *Essential Scrum: A Practical Guide to the Most Popular Agile Process.* Addison-Wesley Professional, 1st edition, 2012.

[11] Andreas M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain*, chapter Chapter 6: Transactions. O'Reilly Media, Inc., 2nd edition, 2017.

[12] Nils Schneider and Matt Corallo. Uri scheme. bitcoin.it, 1 2012. https://en.bitcoin.it/wiki/BIP_0021.

[13] Naciones Unidas. Declaración universal de los derechos humanos. un.org, 11 2020. https://www.un.org/es/universal-declaration-human-rights/.

[14] Daira Hopwood and Jack Grigg. Security properties of sapling viewing keys. ZIP-0310 ZIPS repository, 03 2018. https://zips.z.cash/zip-0310.

[15] Electric Coin Company. Explaining viewing keys. https://electriccoin.co/, 05 2020. https://electriccoin.co/blog/explaining-viewing-keys.

[16] Jack Grigg. Shielded coinbase. ZIP-0213 ZIPS repository, 03 2019. https://zips.z.cash/zip-0213.

[17] Electric Coin Company. Selective disclosure & shielded viewing keys. Electric Coin Company blog, 01 2018. https://electriccoin.co/blog/viewing-keys-selective-disclosure/.

[18] Zcash Developers. Payment disclosure (experimental feature). Github Repository, 11 2017. https://github.com/zcash/zcash/blob/master/doc/payment-disclosure.md.

[19] Kris Nuttycombe and Daira Hopwood. Payment request uris. ZIP-0321 ZIPS repository, 08 2020. https://zips.z.cash/zip-0321.

[20] SerHack. *Mastering Monero: The future of private transactions*. SerHack, 2nd edition, 2018.

[21] Marek Palatinus, Pavol Rusnak, Sean Bowe, and Aaron Voisine. Mnemonic code for generating deterministic keys. Bitcoin Bips, 09 2013. https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki.

[22] Matthew Green Jack Grigg, George Tankersley. Light client protocol for payment detection. Zcash ZIP Repository, 09 2018. https://zips.z.cash/zip-0307.

[23] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571 – 583, 2007. Software Performance.

[24] Xiwei Xu, Cesare Pautasso, Liming Zhu, Qinghua Lu, and Ingo Weber. A pattern collection for blockchain-based applications. In *Proceedings of the 23rd European Conference on Pattern Languages of Programs*, EuroPLoP '18, New York, NY, USA, 2018. Association for Computing Machinery.

[25] Rahim F.A. Baskaran H., Yussof S. A survey on privacy concerns in blockchain applications and current blockchain solutions to preserve data privacy, 2020.

[26] Levi A. Kus Khalilov M.C. A survey on anonymity and privacy in bitcoin-like digital cash systems, 2018.

[27] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 473–489, New York, NY, USA, 2017. Association for Computing Machinery.

[28] You-Ping Chen and Ju-Chun Ko. Cryptoar wallet: A blockchain cryptocurrency wallet application that uses augmented reality for on-chain user data display. In *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '19, New York, NY, USA, 2019. Association for Computing Machinery.

[29] Fabio Aiolli, Mauro Conti, Ankit Gangwal, and Mirko Polato. Mind your wallet's privacy: Identifying bitcoin wallet apps and user's actions through network traffic analysis. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, page 1484–1491, New York, NY, USA, 2019. Association for Computing Machinery.

[30] Mark Perry and Jennifer Ferreira. Moneywork: Practices of use and social interaction around digital and analog money. *ACM Trans. Comput.-Hum. Interact.*, 24(6), jan 2018.

[31] Dimaz Ankaa Wijaya, Joseph K. Liu, Ron Steinfeld, Dongxi Liu, and Jiangshan Yu. On the unforkability of monero. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, Asia CCS '19, page 621–632, New York, NY, USA, 2019. Association for Computing Machinery.

[32] Arthur Gervais, Srdjan Capkun, Ghassan O. Karame, and Damian Gruber. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, page 326–335, New York, NY, USA, 2014. Association for Computing Machinery.

[33] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, and Tuomas Aura. Pitfalls of open architecture: How friends can exploit your cryptocurrency wallet. In *Proceedings of the 12th European Workshop on Systems Security*, EuroSec '19, New York, NY, USA, 2019. Association for Computing Machinery.

[34] Gear A.L. Lero A.R.S., Lero J.B. Privacy and security analysis of cryptocurrency mobile applications, 2019.

[35] Lalitha Muthu Subramanian, Guruprasad Eswaraiah, and Roopa Vishwanathan. Private and secure mixing in credit networks. In *Proceedings of the 2019 International Electronics Communication Conference*, IECC '19, page 52–59, New York, NY, USA, 2019. Association for Computing Machinery.

[36] Tikhomirov S. Biryukov A. Security and privacy of mobile wallet users in bitcoin, dash, monero, and zcash, 2019.

[37] Khan A.G., Zahid A.H., Hussain M., and Riaz U. Security of cryptocurrency using hardware wallet and qr code, 2019.

[38] Sina Rafati Niya, Fabio Maddaloni, Thomas Bocek, and Burkhard Stiller. Toward scalable blockchains with transaction aggregation. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, page 308–315, New York, NY, USA, 2020. Association for Computing Machinery.

[39] Ting Chen, Zihao Li, Yuxiao Zhu, Jiachi Chen, Xiapu Luo, John Chi-Shing Lui, Xiaodong Lin, and Xiaosong Zhang. Understanding ethereum via graph analysis. *ACM Trans. Internet Technol.*, 20(2), apr 2020.

[40] Kristof Jannes, Bert Lagaisse, and Wouter Joosen. You don't need a ledger: Lightweight decentralized consensus between mobile web clients. In *Proceedings of the 3rd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, SERIAL '19, page 3–8, New York, NY, USA, 2019. Association for Computing Machinery.

[41] Aditya Kulkarni. Zec wallet lite. github, 11 2020. `https://github.com/adityapk00/zecwallet-lite`.

[42] Cake Technologies LLC. Cake wallet. github, 11 2020. `https://github.com/cake-tech/cake_wallet`.

[43] Monerujo Team. Monerujo wallet. github, 11 2020. `https://github.com/m2049r/xmrwallet`.

[44] Kevin Gorham. Ecc wallet for android. github, 11 2020. `https://github.com/zcash/zcash-android-wallet`.

[45] Francisco Gindre. Ecc wallet for ios. github, 11 2020. `https://github.com/zcash/zcash-ios-wallet`.

[46] Horizontal Systems. Unstoppable ios wallet. github, 05 2021. `https://github.com/horizontalsystems/unstoppable-wallet-ios`.

[47] Horizontal Systems. Unstoppable android wallet. github, 05 2021. `https://github.com/horizontalsystems/unstoppable-wallet-android`.

[48] Francisco Gindre. Zcashlightclientkit, zcash sdk for ios. github, 12 2019. `https://github.com/zcash/ZcashLightClientKit`.

[49] Kevin Gorham. Zcash sdk for android. github, 7 2019. `https://github.com/zcash/zcash-android-wallet-sdk`.

[50] Aditya Kulkarni. Zec wallet lite cli. github, 11 2020. `https://github.com/adityapk00/zecwallet-light-cli`.

[51] Aditya Kulkarni. Zec wallet lite. github, 11 2020. `https://github.com/zecwalletco/zecwallet-mobile`.

[52] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., USA, 1995.

[53] IEEE. Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.

[54] Andreas M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain*, chapter Chapter 4: Keys, Addresses. O'Reilly Media, Inc., 2nd edition, 2017.

[55] Francisco Gindre. Mnemonicswift: An implementation of bip39 in swift. Github, 2020. https://github.com/zcash-hackworks/MnemonicSwift.

[56] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specifcation. version 2020.1.14 overwinter + sapling + blossom + heartwood + canopy. Zcash ZIP Repository, 2020. https://zips.z.cash/protocol/protocol.pdf.

[57] Pieter Wuille, Marek Palatinus, and Pavol Rusnak. Shielded hierarchical deterministic wallets. ZIPS repository, 05 2018. https://zips.z.cash/zip-0032.

[58] Jack Grigg and Others. Librustzcash - zcash rust crates. Github repository, 11 2019. https://github.com/zcash/librustzcash.

[59] Laszlo Hanyecz. Bitcoin pizza. Bitcoin Blockchain, 05 2010. https://www.blockchain.com/btc/tx/a1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d.

[60] The Monero Project. Monero. github, 03 2014. https://github.com/monero-project/monero.

[61] Cake Technologies LLC. Cake wallet ios. github, 03 2018. https://github.com/fotolockr/CakeWallet/tree/master/CakeWallet/Domain/Monero.

[62] Vitalik Buterin. Why we need wide adoption of social recovery wallets. Website, 01 2021. https://vitalik.ca/general/2021/01/11/recovery.html.

[63] Ethereum. Web3 instance documentation. Website, 05 2017. https://web3js.readthedocs.io/en/v1.3.4/web3.html#web3-instance.

[64] Steve McConnell. *Code Complete, Second Edition*. Microsoft Press, USA, 2004.

[65] Horizontal Systems. Bitcoinkit. github, 05 2021. https://github.com/horizontalsystems/bitcoin-kit-ios.git.

[66] Horizontal Systems. Unstoppable wallet site. website, 05 2021. https://unstoppable.money.

[67] Horizontal Systems. Ethereumkit. github, 05 2021. https://github.com/horizontalsystems/ethereum-kit-ios, https://github.com/horizontalsystems/ethereum-kit-android.

[68] Horizontal Systems. Bitcoinkit. github, 05 2021. https://github.com/horizontalsystems/bitcoin-kit-ios, https://github.com/horizontalsystems/bitcoin-kit-android.

[69] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, USA, 1 edition, 2008.

[70] Privacy coins coins that encrypt their transactions using zero-knowledge proofs or similar private technology, Jul 2020.

[71] Francisco Gindre Zcash Community. List of privacy coins/projects. https://github.com/pacu/privacy-coin-list/, 2020.

[72] Ying Tong Lai, James Prestwich, and Georgios Konstantopoulos. Flyclient - consensus-layer changes. https://zips.z.cash/zip-0221, 2020.

[73] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Superlight clients for cryptocurrencies. Cryptology ePrint Archive, Report 2019/226, 2019. https://eprint.iacr.org/2019/226.

[74] Electric Coin Company. Wallet app threat model. Website, 12 2019. https://zcash.readthedocs.io/en/latest/rtd_pages/wallet_threat_model.html.