MISTIC: Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones

TRABAJO DE FINAL DE MÁSTER

David Llop Vila

Análisis de Zcash

Universitat Oberta de Catalunya

Supervisor: Jordi Herrera Joancomartí

Área: Seguridad en redes y aplicaciones distribuidas

Junio 2018

A Jordi Herrera Joancomartí, por haber propuesto un tema tan interesante, por su acompañamiento y sus revisiones.

A Samer Hassan y al resto de compañeros de P2P Models, por su confianza, su apoyo y su enorme paciencia y comprensión.

A todos/as vosotros/as, ¡21 millones de gracias!

Título: Análisis de Zcash

Autor: David Llop Vila, Universitat Oberta de Catalunya

Supervisor: Jordi Herrera Joancomartí, Universitat Autònoma de Barcelona

Resumen: Zcash es una criptomoneda que usa pruebas de conocimiento nulo para proporcionar mayor privacidad a sus usuarios al no publicar las transacciones de manera transparente en la blockchain. En este análisis estudiamos el problema de la privacidad en las criptomonedas, y qué solución proporciona Zcash. Analizamos sus predecesores, las herramientas que hay disponibles a día de hoy, su estructuras de datos y su funcionamiento a nivel de transacciones y minado. Además dedicamos un capítulo a la introducción de los zk-SNARKs, las novedosas pruebas de conocimiento nulo mucho más eficientes en tamaño y tiempo de verificación, y estudiamos el carácter de su comunidad y cómo esta se plantea su futuro.

Palabras clave: zcash, pruebas de conocimiento nulo, blockchain

Title: Zcash Analysis

Author: David Llop Vila, Universitat Oberta de Catalunya

Supervisor: Jordi Herrera Joancomartí, Universitat Autònoma de Barcelona

Abstract: Zcash is a cryptocurrency that uses zero-knowledge proofs to provide increased privacy to its users by not publishing transactions transparently in the blockchain. In this analysis we study the problem of privacy in cryptocurrencies, and the solution provided by Zcash. We analyze Zcash's predecessors, the tools that are available today, its data structures and mechanics at the level of transactions and mining. We also dedicate a chapter to the introduction of the zk-SNARKs, the new zero-knowledge proofs that are much more efficient in size and verification time, and we study the character of its community and how it plans its future.

Keywords: zcash, zero-knowledge proofs, blockchain

Índice general

1. Introducción a Zcash

1.1. ¿Qué es Zcash?

Zcash es una criptomoneda que usa pruebas de conocimiento nulo para proporcionar privacidad mejorada a sus usuarios en comparación con otras criptomonedas como Bitcoin. Zcash es una implementación del protocolo Zerocash, que a su vez es el sucesor del protocolo Zerocoin.

Zerocoin fue desarrollado por cuatro criptógrafos en la Johns Hopkins University, y se presentó en una conferencia en 2013 [11]. Algunos de ellos se asociaron más tarde con otros criptógrafos que habían llevado a cabo investigaciones sobre pruebas de conocimiento nulo. Juntos, incorporaron una prueba más eficiente y nuevas características de privacidad a Zerocoin, lo que resultó en Zerocash, y presentaron su trabajo en otra conferencia en 2014 [20]. En 2015 fundaron Zerocoin Electric Coin Company (también conocida como Zcash Company) para dirigir el desarrollo de las mejoras del protocolo y la implementación de referencia, llamada Zcash. En 2016 es cuando lanzaron Zcash, primero como *Vista previa técnica* el 20 de enero[1], y más tarde el mainnet cobró vida el 28 de octubre[2], cuando acuñan el bloque génesis. Finalmente, en 2017, fundaron la Zcash Foundation[3].

Como cualquier otra criptomoneda descentralizada, los pagos de Zcash se publican en una blockchain pública, pero los usuarios pueden usar una característica de privacidad opcional para ocultar al emisor, al destinatario y al monto que se tramita.

Tener esta característica de privacidad opcional hace que en Zcash haya dos tipos de transacciones: transacciones "transparentes", que son similares a las transacciones de bitcoin, y transacciones "blindadas", que usan un nuevo tipo de prueba de conocimiento nulo llamados argumentos no interactivos y concisos de conocimiento (zk-SNARKs) [3] para proteger la confidencialidad de los datos.

Desafortunadamente, solo el 4 % de las monedas de Zcash en uso hoy en día están en direcciones blindadas (diciembre de 2017 [15]), lo que significa que la característica principal de Zcash, la que la diferencia de Bitcoin, apenas se usa. Esto puede deberse a que, como vemos en el Capítulo 2, muchos monederos y casas de cambio que integran Zcash solo admiten direcciones transparentes, que son más fáciles de implementar. En el momento en el que escribo estas líneas, sólo la implementación de referencia (zcashd) y la casa de cambio de ShapeShift.io admiten direcciones blindadas [4].

Al igual que Bitcoin, Zcash tiene un suministro total fijo de 21 millones de unidades. En cambio, la dificultad está ajustada para que se genere un bloque cada 2.5 minutos (a diferencia de los 10 minutos de Bitcoin), y sus bloques son de 2MB (en lugar de 1MB de bitcoin).

El algoritmo de minado que utiliza es Equihash, un algoritmo de prueba de trabajo ideado por Dmitry Khovratovich y Alex Biryukov. Utiliza la RAM como cuello de botella para generar pruebas y se creía que eso lo haría resistente a la

[1] https://blog.z.cash/helloworld/
[2] https://blog.z.cash/zcash-begins/
[3] https://z.cash.foundation/blog/hello-world/
[4] https://www.zcashcommunity.com/wallets/

creación de circuitos especializados para su minado (las ASIC), que han centralizado el minado de bitcoin en muy pocas manos. Veremos más sobre ese algoritmo en el Capítulo 6.

Pero antes de comprender mejor cómo Zcash logra el anonimato, veamos por qué la privacidad es importante en las criptomonedas y por qué es tan dificil anonimizar las transacciones en Bitcoin.

1.2. La importancia de la privacidad

En el primer post del blog de Zcash, el CEO de Zcash Company, Zooko Wilcox, explica por qué la privacidad es tan importante en las criptomonedas.

Ante todo, dice que **la privacidad es un derecho humano**. Las personas tienen el derecho de elegir cuáles de sus movimientos, palabras y acciones serán compartidas o publicadas. Eso es necesario para valores humanos básicos como la dignidad, la intimidad y la moralidad.

La privacidad también es necesaria para las empresas. En los sistemas bancarios tradicionales, el banco actúa como auditor y solo sabe cuánto dinero tiene la gente en sus cuentas. De esta manera, se preserva la privacidad. Pero en un sistema descentralizado abierto, la figura de este auditor centralizado desaparece, y todos los usuarios en el sistema deberían poder auditar la contabilidad, lo que genera problemas a las empresas que quieren usar las tecnologías blockchain en su vida diaria.

La privacidad también es la única forma de garantizar la **fungibilidad**. En economía, la fungibilidad es la propiedad de un bien o una mercancía cuyas unidades individuales son esencialmente intercambiables. Por el contrario, si es fácil seguir el rastro de una moneda, algunas de las monedas pueden perder o ganar valor, porque pueden marcarse como deseables o indeseables. Si esto sucede, es difícil para un comercio a gran escala aceptar monedas provenientes de fuera de un pequeño grupo de confianza compartida. La fungibilidad es una propiedad importante, y Zcash la ha convertido en su lema: "Todas las monedas se crean iguales".

La privacidad también es un valor social, porque ayuda a las sociedades a ser más pacíficas y más prósperas. Una sólida tradición de privacidad es una característica común en las sociedades ricas y pacíficas, y la falta de privacidad a menudo se encuentra en las sociedades con dificultades y en crisis.

Zooko también afirma que sabe que las monedas que protegen la privacidad como Zcash se usarán para realizar actividades ilegales. De la misma manera, afirma que los delincuentes también usan el dinero en efectivo, el sistema bancario actual e Internet. Técnicamente, no es posible evitar que lo hagan, por lo que ese no es su objetivo. Por el contrario, el software que desarrollan puede servir a muchas buenas personas.

1.3. Anonimato en Bitcoin

1.3.1. El presunto anonimato en Bitcoin

Se ha afirmado muchas veces que Bitcoin es una criptomoneda anónima, y aún en el momento de escribir esto, podemos leer en la página de donaciones de Wikileaks la siguiente declaración[5]: "Bitcoin es una moneda digital segura y anónima. Los bitcoins no pueden ser rastreados fácilmente, y es una alternativa más segura y rápida a otros métodos de donación".

Pero, ¿Bitcoin es verdaderamente anónimo? Pues por un lado no necesitamos proporcionar nuestra identidad real para crear una dirección bitcoin, mientras que sí tenemos que hacerlo para crear una cuenta bancaria. Pero las direcciones de bitcoins son identidades por sí mismas, por lo que deberíamos considerar Bitcoin como una criptomoneda seudónima, no como una anónima.

Bitcoin no es verdaderamente anónimo porque todas las transacciones son públicas (están en la cadena de bloques), por lo que las identidades (direcciones) seudónimas son enlazables, y de acuerdo con la nomenclatura común [13], la desvinculación (*unkinkability*) es un requisito para el anonimato.

Por lo tanto, en el caso de las criptomonedas, para obtener **anonimato tanto de emisor como de destinatario**, ninguna transacción en particular debe poder enlazarse a ningún emisor o destinatario, y para ningún emisor o destinatario particular, ninguna transacción debe ser enlazable.

Más concretamente, según investigadores de la Universidad de Princeton [12]:

1. Debería ser difícil vincular diferentes direcciones del mismo usuario.

2. Debería ser difícil vincular diferentes transacciones realizadas por el mismo usuario.

3. Debería ser difícil vincular al emisor de un pago con su destinatario.

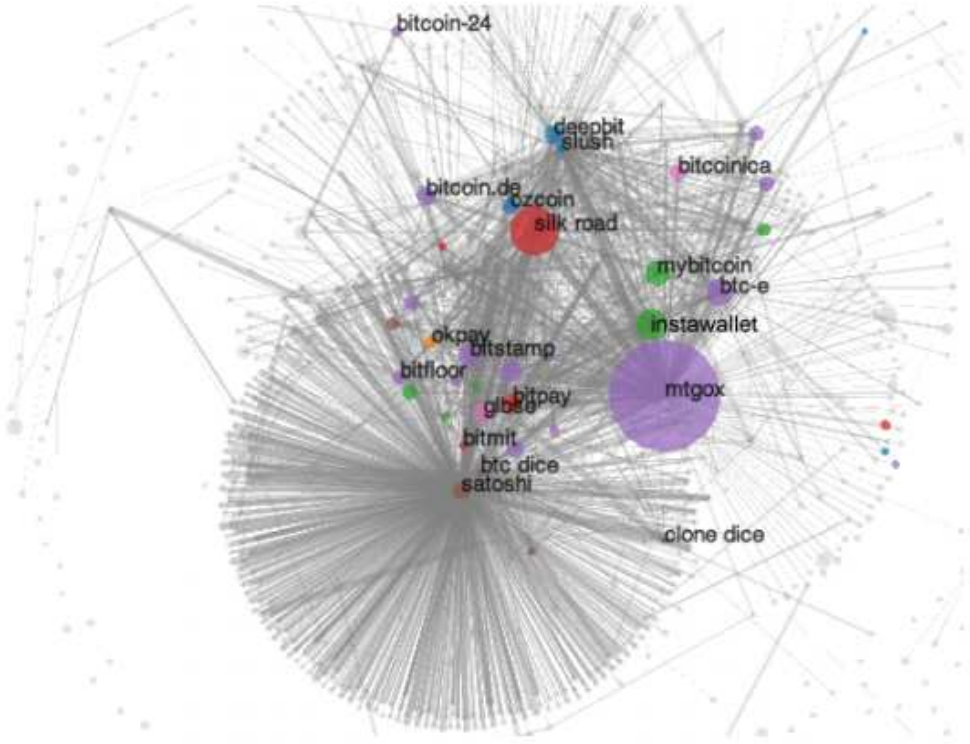
Los dos primeros requisitos se pueden cumplir si los usuarios usan una dirección diferente cada vez que realizan una nueva transacción. Desafortunadamente el tercer requisito es más difícil de cumplir, porque cada transacción registrada en la blockchain indica un conjunto de entradas y un conjunto de salidas, que conectan a los remitentes con los destinatarios.

Y peor aún, si se usa más de una entrada en la misma transacción, podemos suponer que están controladas por la misma entidad. Esta información se puede utilizar para rastrear el dinero y conectar a distintas transacciones bajo el mismo *clúster*, desanonimizando así a los usuarios de bitcoin, como se muestra en [10].

En ese estudio, los investigadores realizaron transacciones con algunos servicios de bitcoin y analizaron el grafo de transacciones al cabo del tiempo, para rastrear a dónde fue su dinero, especialmente se fijaron en cuales eran las entradas controladas conjuntamente y las direcciones de cambio. Usando este método, agruparon las direcciones de los principales servicios online de bitcoin (como intercambios, juegos de azar, monederos...) como se muestra en la Figura 1.1.

[5] https://shop.wikileaks.org/donate

Figura 1.1: Red de usuarios de Bitcoin en 2013. El área del clúster representa el valor de entrada externo (bitcoins), y los bordes representan las transacciones (al menos 200 transacciones para aparecer en el gráfico) [10].



1.3.2. Mezclado de Bitcoin

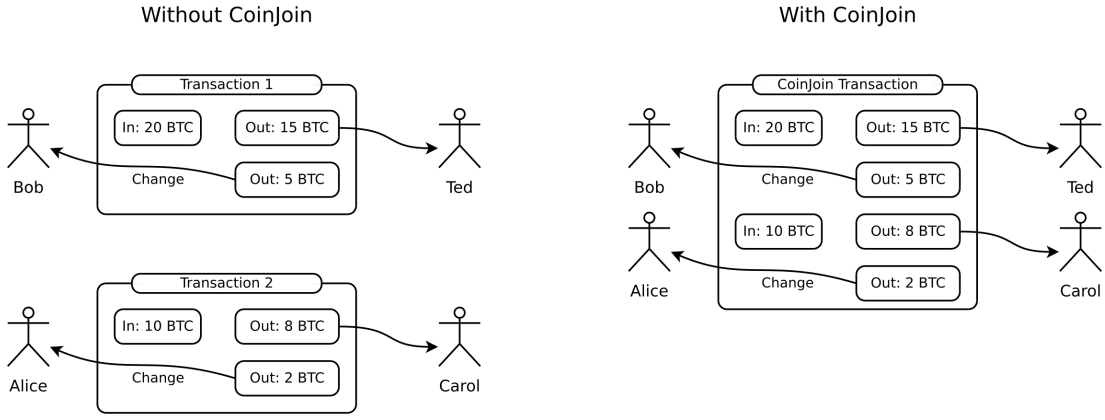
Hay varios mecanismos para hacer el análisis del grafo de transacciones menos efectivo. Uno de ellos es el mezclado, que consiste en usar un intermediario para realizar un pago. Los usuarios le envían a un mezclador una cantidad de bitcoins, y le indican una dirección a la que enviar una cantidad equivalente (los bitcoins de otro usuario). De esta manera la relación entre emisor y receptor se difumina entre todos los emisores y receptores que usan el mezclador.

El problema con los servicios de mezclado es que son servicios centralizados, en los que hay que depositar cierta confianza. El mezclador podría quedarse con el dinero que se le ha enviado y nunca enviar su contraparte a la dirección indicada. También hay que confiar en que no guarda el registro de qué transacción corresponde a qué, ya que eso podría desanonimizar a sus usuarios.

Para evitar esos problemas, Gregory Maxwell, desarrollador de Bitcoin Core, propuso un método para mezclar varias transacciones en una, sin necesidad de una tercera parte de confianza, llamada CoinJoin. De esta manera, se hacen transacciones en las que las entradas están controladas por más de una entidad, y las salidas también suelen ser a entidades diferentes. De esta manera se difumina quién paga a quién en el grafo de transacciones.

Más tarde, investigadores de la Universidad de Saarland desarrollaron sobre el concepto de CoinJoin y propusieron CoinSuffle [18], un protocolo que incrementaba la privacidad haciendo que, en grupos de más de dos, ninguno de los participantes pudiera conocer cuales eran las entradas y salidas originales de los otros.

Figura 1.2: Si combinamos las dos transacciones (Bob a Ted y Alice a Carol), obtenemos una transacción en la que el resultado es el mismo pero desde fuera ya no podemos saber si ha sido Bob quien ha pagado a Ted o si ha sido Alice. Fuente: Wikipedia.



CoinSuffle++ [19] es propuesto por los mismos investigadores más tarde, y supone un incremento en la eficiencia, pasando de un número lineal de rondas respecto al número de usuarios, a un número constante, gracias a que está basado en el problema de la cena de los criptógrafos, que estudia cómo hacer una computación segura multipartita (*secure multiparty computation*) de la función booleana OR.

Tanto CoinJoin, como CoinSuffle y CoinSuffle++ desvinculan los emisores de los receptores de varias transacciones haciendo una transacción conjunta. Por lo tanto, para alguien que esté observando la transacción desde fuera, no puede saber quién ha pagado a quién, sólo que alguna o algunas de las entradas corresponde con alguna o algunas de las salidas. Decimos entonces que el conjunto de anonimato de alguien usando una de estas técnicas es el de todas las monedas de esa ronda de mezclado. Por el contrario, veremos que en Zcash el conjunto de anonimato son todas las monedas que hay en el sistema, haciendo que la privacidad sea mucho mayor.

Para incrementar la privacidad en el mezclado de Bitcoin, se recomienda mezclar varias veces. Añadir un nuevo participante en una mezcla sólo incrementa el conjunto de mezclado linealmente, mientras que si mezclas varias veces con otros participantes, se puede conseguir un conjunto de anonimato que crece exponencialmente a cada mezcla. Por contrapartida, hay que pagar comisiones por cada nueva transacción y puede salir caro conseguir ese nivel de anonimato.

Por otro lado, el mezclado de Bitcoin sólo es eficaz si las monedas mezcladas tienen la misma denominación (el mismo valor). En caso contrario, si mezclamos una transacción que envía 5 milibitcoins de una dirección a otra, con otra transacción que envía 2 milibitcoins de una transacción a otra, no estaríamos anonimizándolas, ya que sería trivial conectar las entradas con las salidas de esa transacción mezclada.

Para solucionar ese problema podemos o usar denominaciones estándar (por ejemlo 10 mBTC, 5 mBTC, 2 mBTC, 1 mBTC...) en las que para pagar por algo que vale 12 mBTC, lo haríamos en dos transacciones, una de 10 mBTC y otra de 2 mBTC. Otra solución propuesta es combinar CoinSuffle con las Transacciones

Confidenciales propuestas por Blockstream (requiere de *soft fork*), para ocultar el valor de las entradas y las salidas en las transacciones que son mezcladas. Ese es el enfoque propuesto en ValueSuffle [17].

1.4. Zerocoin

Zerocoin [11] es un protocolo que extiende una criptomoneda para proporcionarle garantías fuertes de anonimato. Utiliza pruebas de conocimiento nulo para **desvincular a las monedas de su historia pasada**, y así prevenir el análisis del grafo de transacciones.

Las **pruebas de conocimiento nulo** son métodos por los cuales una entidad (la que prueba) puede demostrar a otra entidad (la que verifica) que sabe un valor x sin desvelar ninguna otra información aparte del hecho que sabe el valor x. Las pruebas de conocimiento nulo, incluidas las zk-SNARKs, son tratadas con más profundidad en el Capítulo 5.

Gracias a las pruebas de conocimiento nulo, se puede probar que se conoce x tal que $h(x)$ pertenece al conjunto S sin desvelar nada acerca de x ni qué elemento del conjunto es igual a $h(x)$.

Entendemos $h(x)$ como una función hash criptográfica [16]. En las funciones hash, la entrada puede ser de cualquier tamaño, produce una salida de tamaño fijo y se puede computar eficientemente. Además, para ser criptográficamente segura, tiene que ser resistente preimágenes, a preimágenes-segundas y a colisiones.

1.4.1. Funcionamiento de Zerocoin

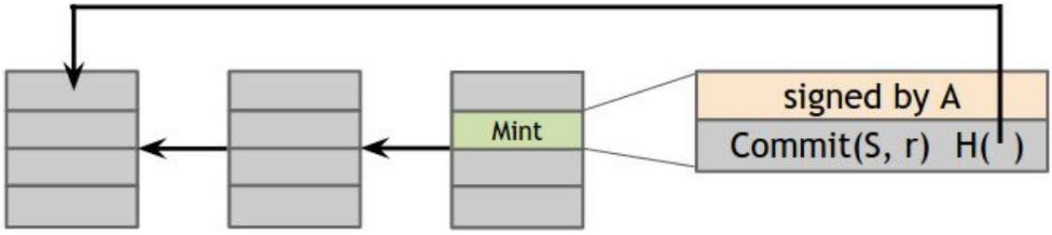
Zerocoin es un protocolo que se usa sobre una criptomoneda como Bitcoin, a la que llamamos Basecoin. Se pueden convertir basecoins a zerocoins y luego volver a convertirlos a basecoins otra vez. Cuando se hace eso, se rompe la traza entre el basecoin original y el nuevo. En este sistema, Basecoin es la criptomoneda que se usa para realizar transacciones y Zerocoin provee del mecanismo para cambiar los basecoins de los usuarios por otros nuevos, *quemando* los antiguos.

Para crear un zerocoin se añade en la blockchain una transacción que acuña ese zerocoin, y para poder hacerlo hay que *quemar* un basecoin. Acuñar un zerocoin se hace en tres pasos:

1. Generar un número de serie S y un número secreto aleatorio r.
2. Calcular $c := h(S||r)$, al que llamaremos compromiso (*commitment*).
3. Publicar el compromiso en la blockchain, junto con la referencia al basecoin que se está quemando y una firma válida de que el propietario de ese basecoin quiere quemarlo a cambio de publicar ese compromiso (ver Figura 1.3). Tanto el número de serie como el número aleatorio quedan ocultos por el momento.

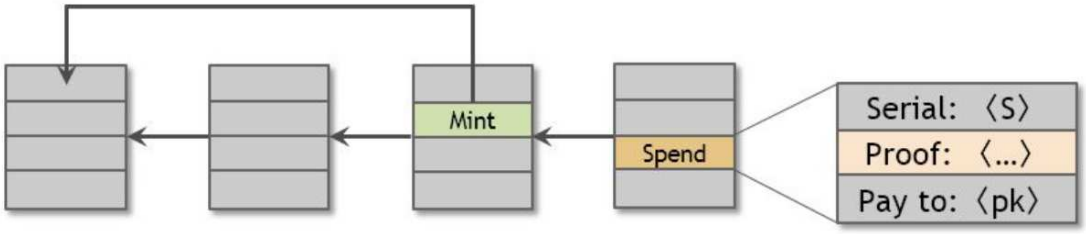
Para gastar un zerocoin y obtener un basecoin, el usuario necesita probar que ha acuñado previamente un zerocoin. Podría probarlo revelando S y r, pero eso enlazaría el basecoin antiguo con el nuevo. En lugar de eso, el usuario demuestra, usando una prueba de conocimiento nulo no interactiva, que conoce S y r tal que

Figura 1.3: Transacción de acuñado de un zerocoin. Se registra el compromiso, el puntero a un output de basecoin anterior, y se firma. Imagen original: [12].



$c := h(S||r)$ pertenece al conjunto de compromisos $c_1, c_2, ..., c_n$. Además de esa prueba, el usuario también publica en una transacción a la blockchain el número de serie S, que sólo es válido si no ha sido usado en una transacción anterior, para evitar gastos dobles (ver Figura 1.4). Con eso puede transferir un nuevo basecoin a la dirección deseada.

Figura 1.4: Transacción de gasto de un zerocoin. Se registra el número de serie, la prueba de conocimiento nulo, y la dirección a la que enviar el basecoin. Imagen original: [12]



Los zerocoins tienen denominaciones estándares (1 zerocoin, 0.1 zerocoins...), tal y como ya hemos visto cuando hablamos de mezclado. No se puede acuñar una moneda por valor de 0.111 zerocoins porque eso implicaría que el conjunto de anonimizado sería demasiado pequeño. Esa moneda sólo sería indistinguible de las otras ese mismo valor. Por eso, zerocoin se usa sólo para borrar la historia de monedas *enteras*, y basecoin para enviar dinero de una dirección a otra, separar, y juntar monedas. Veremos que en Zerocash eso ya no es necesario, ya que la denominación se oculta.

Por otro lado, el tamaño de las pruebas de conocimiento nulo propuestas en el protocolo Zerocoin crece logarítmicamente respecto al número de compromisos que hay en la blockchain, en lugar de linealmente, como cabría esperear. Eso es gracias a que los compromisos se ordenan dentro de un árbol de Merkle, cuya raíz es la que se usa como hash de la prueba. Zerocash reduce aún más el tamaño de la prueba, gracias al uso de zk-SNARKs, cuyo tamaño es constante.

2. Herramientas de Zcash

2.1. zcashd

`zcashd` es la implementación de referencia de Zcash. Oficialmente sólo está soportado en sistemas Linux de 64 bits, pero existen ports para Windows (winzec[1]) y MacOS (zcash-apple[2]). Zcash dispone de una guía de usuario que explica cómo instalar `zcashd` y cómo empezar a hacer los primeros pagos[3].

Se trata de un fork de la implementación de referencia de Bitcoin (`bitcoind`), al que se le han añadido nuevas funciones para soportar direcciones blindadas por el protocolo Zerocash, además de cambiarle algunos parámetros (nomenclaturas, algoritmo de minado, tamaño y tiempo de bloque...).

Las operaciones que han añadido a Bitcoin tienen el prefijo *z_*. Por ejemplo, `getbalance` sólo te muestra el saldo que hay en las direcciones transparentes (tiene exactamente la misma interfaz que en Bitcoin), mientras que `z_getbalance` sirve para consultar el saldo de cualquier dirección, ya sea transparente o blindada. Nótese que para consultar el saldo total se usa `z_gettotalbalance`.

Para interactuar tanto con `bitcoind` como con `zcashd`, podemos usar una herramienta de línea de comandos (`bitcoin-cli` y `zcash-cli`, respectivamente), o usar su interfaz JSON-RPC.

La herramienta de línea de comandos viene integrada con `zcashd`, y se conecta por RPC a un nodo, que por defecto es el nodo ejecutándose en la máquina local (`127.0.0.1:8232`) aunque, por supuesto, se puede especificar otro nodo. Con esta herramienta podemos ejecutar todas las operaciones con `zcash-cli [comando]`, como por ejemplo:

```
$ zcash-cli getinfo
```

Que devuelve un JSON con la misma información que devolvería un nodo Bitcoin: número de versión, protocolo y monedero, balance (sólo de las direcciones transparentes), número de conexiones...

También está disponible una interfaz JSON-RPC, que es la que usan la mayoría de herramientas que interactúan con `zcashd`. Existen librerías para interactuar con JSON-RPC en muchos lenguajes de programación. Veamos cómo sería la llamada al método anterior con la herramienta `curl`:

```
$ curl --user username --data-binary '{"jsonrpc": "1.0", \
"id":"curltest", "method": "getinfo" }' \
-H 'content-type: text/plain;' http://127.0.0.1:8232/
```

Las operaciones que empiezan por `z_`, y por tanto son extensiones que Zcash hace a Bitcoin, son:

- `z_getbalance` y `z_gettotalbalance`: permiten consultar el saldo de una dirección (ya sea transparente o blindada), o de todo el monedero.

[1] https://github.com/radix42/winzec
[2] https://github.com/kozyilmaz/zcash-apple
[3] https://github.com/zcash/zcash/wiki/1.0-User-Guide

- `z_getnewaddress`, `z_listaddresses` y `z_validateaddress`: permiten generar una nueva dirección blindada, mostrar las direcciones blindadas del monedero, y información sobre una dirección blindada (como sus claves de pago y transmisión).

- `z_exportkey`, `z_importkey`, `z_exportwallet`, `z_importwallet`, y las recientemente añadidas `z_exportviewingkey` y `z_importviewingkey`: permiten importar y exportar claves específicas o todas las del monedero, además de permitir importar y exportar sólo las claves de visionado, que son las que permiten monitorizar los pagos que recibe o realiza una dirección, pero no gastar sus fondos.

- `z_getoperationresult`, `z_getoperationstatus` y `z_listoperationids`: permiten ver el estado de las operaciones que han sido lanzadas asíncronamente (algunas pueden tardar varios minutos, y por eso es necesario poder realizar este seguimiento).

- `z_listreceivedbyaddress`, `z_sendmany`, `z_mergetoaddress` (aún experimental) y `z_shieldcoinbase`: permiten ver las transacciones que ha recibido una dirección, enviar dinero de una dirección a varias, de varias direcciones a una, y cobrar la recompensa de un bloque minado.

- `z_getpaymentdisclosure` y `z_validatepaymentdisclosure`: son operaciones experimentales para procesar pruebas de que un pago ha sido realizado a una dirección blindada, pudiendo ser verificada esa prueba por una tercera parte. Corresponde al ZIP 303 sobre Payment Disclosure (ZIP es el acrónimo de Zcash Improvement Proposals, más información en el Capítulo 7).

Se puede consultar más información sobre cada una de las operaciones consultado `zcash-cli help [operación]` o visitando la referencia de la API de pagos de Zcash[4]. En la documentación oficial faltaban algunas de las operaciones, así que a raíz de la escritura de este capítulo he mandado un pull request para que sean añadidas[5].

Crearemos direcciones nuevas usando `getnewaddress` y `z_getnewaddress`, para crear direcciones transparentes y blindadas, respectivamente, y podemos usar `z_sendmany` para enviar dinero, tanto si es desde/hacia una dirección transparente como si es desde/hacia una dirección blindada. En el caso de que no se consuma todo el dinero de un pago previo al enviarlo a una nueva dirección, `z_sendmany` creará una dirección de cambio para recoger el dinero sobrante.

```
$ zcash-cli z_sendmany "$DIRECCION_EMISOR"
'[{"address": "$DIRECCION_RECEPTOR",
"amount": $CANTIDAD}]'
```

El comando va a devolver un identificador de operación y va a procesar la operación asíncronamente. Si en el pago hay involucradas direcciones blindadas, éste va a ser más costoso por el coste de generar las pruebas de conocimiento nulo, y las operaciones `z_getoperationstatus` y `z_getoperationresult` nos van a servir para consultar el estado de la operación. Hasta que la operación no haya

[4]https://github.com/zcash/zcash/blob/master/doc/payment-api.md
[5]https://github.com/zcash/zcash/pull/3264

terminado no se conoce el identificador de la transacción, ya que este es el hash de los datos de esta, y esos datos no están disponibles hasta entonces.

2.2. Herramientas con interfaz gráfica

Existen diferentes interfaces gráficas para usar Zcash sin necesidad de interacturar directamente con la línea de comandos. La más antigua de ellas es *zcash-swing-wallet-ui*[6], obsoleta desde enero de 2018, y muy similar a la interfaz de *bitcoin-qt*, pero con la estética Swing de Java (Figura 2.1).

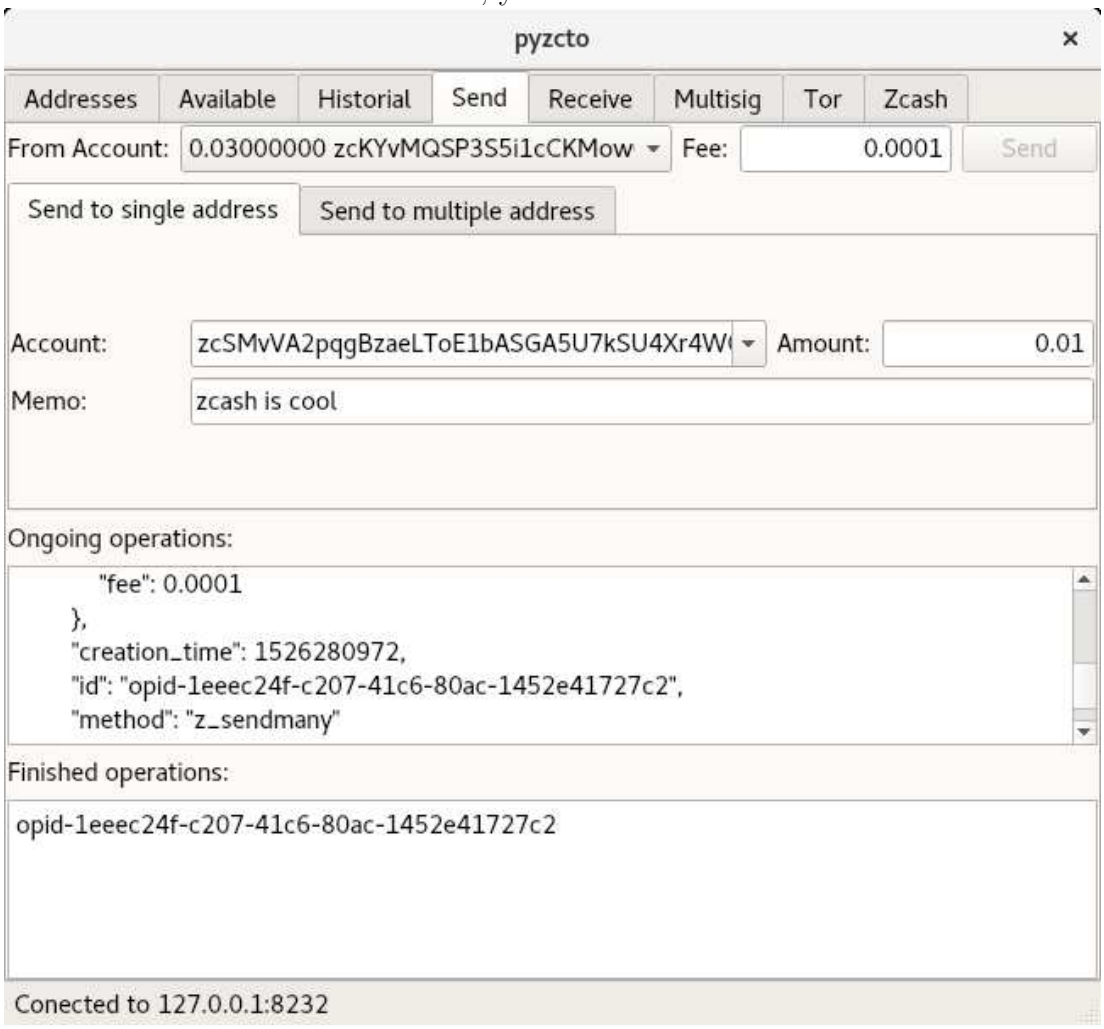
Figura 2.1: Interfaz de Swing Wallet UI. Véase que la interfaz resulta bastante familiar para los usuarios de Bitcoin. Como diferencia principal podemos observar que las transacciones transparentes y blindadas están claramente diferenciadas, así como los saldos transparente y privado.

Por otro lado tenemos la interfaz de Pyzcto[7], desarrollada con Python y Qt5 (Figura 2.2). La interfaz permite conectarse a Tor para poder acceder a la interfaz JSON-RPC desde otros dispositivos a través de un servicio oculto Onion. Como prueba de concepto han desarrollado también una aplicación para Android[8] que se conecta al servicio usando un código QR en el que está codificado el usuario y la contraseña del RPC y la URL. De esta manera se puede controlar el nodo desde el móvil a través de internet.

De nuevo, a raíz de la elaboración de este proyecto, nos hemos dado cuenta de un bug en la recolección de los datos del fichero zcash.conf que hace Pyzco, y lo hemos solucionado[9].

[6]https://github.com/vaklinov/zcash-swing-wallet-ui
[7]https://github.com/miguelmarco/pyzcto
[8]https://github.com/miguelmarco/ZcashPannel
[9]https://github.com/miguelmarco/pyzcto/pull/2

Figura 2.2: Vemos que la interfaz de pyzcto ofrece más funcionalidades que Swing Wallet, como enviar a múltiples direcciones en una sola transacción, crear y sacar fondos de una dirección multifirma, y servir la API JSON-RPC a través de Tor.



2.3. Otras herramientas

A día de hoy no se encuentran disponibles implementaciones alternativas a `zcashd`, al contrario de lo que pasa en otras criptomonedas, como es el caso de Bitcoin y Ethereum. En el caso de Bitcoin, Bitcoin Core es la implementación de referencia, y tenemos otras implementaciones totales o parciales del protocolo, como libbitcoin, BitcoinJ y bitcoinjs-lib. En el caso de Ethereum, geth es la implementación mantenida por la Ethereum Foundation, y además tenemos Parity, que es otra implementación total del protocolo hecha por un equipo y lenguaje diferentes.

Eso implica que para usar Zcash con la funcionalidad de las direcciones blindadas no haya más remedio que usar *zcashd*. Eso hace que muchas herramientas que integran Zcash sólo soporten direcciones transparentes, pues son muy similares a Bitcoin y evitan la necesidad de ejecutar un nodo completo de Zcash, que sólo está disponible oficialmente para Linux de 64 bits.

Además, el cálculo de las pruebas de conocimiento nulo que se realiza al enviar y recibir en direcciones blindadas requiere de muchos ciclos de cómputo y

memoria RAM. Está previsto que esos requisitos sean reducidos drásticamente en el *hard fork* previsto para la versión 2.0 de Zcash (llamada Sapling). Se espera que entonces muchos más servicios integren las transacciones privadas, como es el caso de OpenBazaar, que ya se ha posicionado a favor[10]. Al reducir los requisitos también se podrían usar transacciones privadas en dispositivos móviles.

Entre los monederos que ahora mismo soportan direcciones transparentes de Zcash encontramos Guarda Wallet, Exodus, Jaxx, Bitpie, Coinomi, Carbon Wallet, HolyTransaction, Cryptonator, Freewallet, Coinbr y Waterhole, así como también los monederos de hardware Ledger y Trezor.

Guarda Wallet, que ha sido destacado recientemente en el blog de zcash[11], recibió 30.000 dólares por parte de la Zcash Foundation para el desarrollo de una librería de verificación simplificada de pagos (SPV) para Zcash[12], y de esa manera poder hacer monederos ligeros, que consultan y realizan transacciones sin necesidad de descargar la blockchain entera, sólo las cabeceras de los bloques.

En cuanto a casas de cambio que soportan Zcash podemos listar 34, de las cuales sólo una, Shapeshift.io, soporta direcciones blindadas.

Disponemos de dos exploradores de bloques de Zcash: zcashnetwork.info y explorer.zcha.in. El primero recibió una beca de 7.000 dólares por parte de la Zcash Foundation (ver Capítulo 7), pero ninguno de ellos tiene relación con la Zcash Company. Zcashnetwork es mantenido por David Mencer y usa el software Insight, originalmente desarrollado por Bitpay para Bitcoin, y que Jack Grigg adaptó para Zcash. El segundo es mantenido por Bitfly y usa software privativo.

Analizando el código de Jack Grigg, que puede encontrarse en los repositorios de la herramienta `bitcore-node-zcash`, la librería `bitcore-lib-zcash`, sus dependencias `bitcore-message-zcash` y `bitcore-build-zcash` y las interfaces `insight-ui-zcash` y `insight-api-zcash`, vemos que no son necesarias grandes modificaciones para transformar un explorador de bloques de Bitcoin a Zcash si el explorador está basado en bitcoind. Eso se debe a que zcashd ofrece una API muy similar a bitcoind[13], y por lo tanto en gran parte el código sólo hace falta cambiar las denominaciones (de bitcoin a zcash, de BTC a ZEC...) y algunos parámetros como el tiempo y tamaño de bloque o el puerto RPC. La mayor carga de desarrollo ha ido destinada a diseccionar y mostrar el campo JoinSplit de las transacciones de Zcash[14] y a implementar las modificaciones a la cabecera del bloque[15]. En el siguiente capítulo analizaremos las transacciones y las cabeceras de bloques y veremos para qué sirven esos campos. Como el software lleva sin actualizarse desde 2016, no soporta campos añadidos recientemente y debería adaptarse a la nueva actualización de la red prevista para junio de 2018, llamada Overwinter.

[10] https://www.reddit.com/r/OpenBazaar/comments/7v7aek/support_for_shielded_zcash_addresses/

[11] https://blog.z.cash/zcash-community-spotlight-guarda-wallet/

[12] https://github.com/guardaco/zcash-SPV

[13] Aunque es verdad que Jack Grigg debe añadir soporte zmq a zcashd para poder proveer a bitcore-node de datos.

[14] https://github.com/str4d/bitcore-lib-zcash/commit/80f5b45034e5174afedd722c6f730850d407f03b

[15] https://github.com/str4d/bitcore-lib-zcash/commit/8740d9e964e106aa14132cbeaf6d2e03f04e86ba

3. Anatomía de Zcash

En este capítulo analizamos los componentes que forman la estructura de datos de la blockchain de Zcash: sus transacciones y sus bloques.

3.1. Análisis de las transacciones

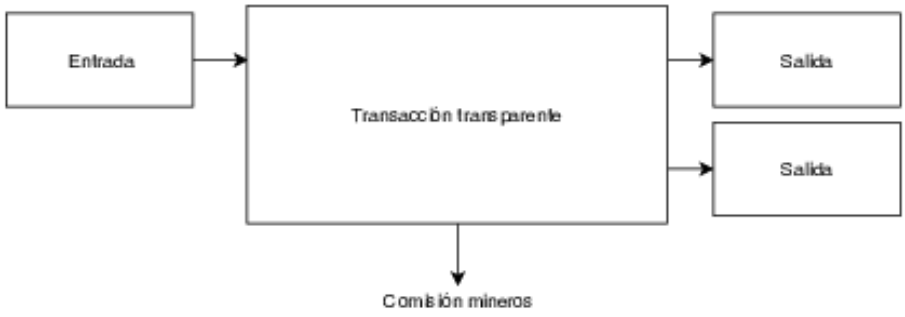
Hay cuatro tipos de transacciones que pueden darse entre una dirección transparente y una blindada en Zcash:

- Públicas: de dirección transparente a dirección transparente
- Blindantes: de dirección transparente a dirección blindada
- Desblindantes: de dirección blindada a dirección transparente
- Privadas: de dirección blindada a dirección blindada

Vamos a analizar cada una de ellas, y para hacerlo usaremos la operación `getrawtransaction` de zcash-cli, que devuelve una cadena codificada en hexadecimal o su representación en JSON. Dicho JSON contiene campos en hexadecimal, como los identificadores de transacción, las firmas y los campos cifrados. Para facilitar la lectura, sólo mostramos los 12 primeros dígitos seguidos de puntos suspensivos, pero nótese que algunos campos pueden llegar a contener más de 600 bytes.

3.1.1. Transacción pública

Figura 3.1: Representación de una transacción transparente, con una entrada, dos salidas y la comisión para los mineros.



Las transacciones públicas son muy similares a Bitcoin. La transacción recoge como entradas una serie de salidas de transacciones anteriores que aún no han sido gastadas (a las que llamamos UTXOs, *unspent transaction outputs*), y las gasta para generar nuevas salidas que pueden recoger otras transacciones en el futuro, también como entradas.

Para que una transacción sea válida, sus entradas deben hacer referencia a salidas válidas de transacciones anteriores, que no hayan sido usadas por otra transacción anterior (para evitar gastos dobles), la suma del valor de las salidas no puede superar al de las entradas (no podemos crear valor en una transacción), y debe ir acompañado de las firmas de los propietarios de todas las UTXOs consumidas.

Hemos transferido 0.008 ZEC de una dirección transparente a otra. Como la UTXO que consumimos tiene el valor de 0.0096 ZEC, y depositamos una fee de 0.0001 ZEC, sobran 0.0015 ZEC que nos vuelven a una dirección nueva (dirección de cambio). El tamaño de la transacción resultante es 226 bytes y su identificador:

97b24ba5cdf8aed31fb61c65ef4b16974a8ac17cecc09d7e49390ca8e55fefcd

Puede consultarse online usando los exploradores que hemos señalado en el Capítulo 2. Este es el JSON que obtenemos si decodificamos la transacción en hexadecimal:


```
{
  "txid": "97b24ba5...",
  "overwintered": false,
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "fae08b99...",
      "vout": 0,
      "scriptSig": {
        "asm": "30450221... 035f0723...",
        "hex": "48304502..."
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.00800000,
      "valueZat": 800000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 b7da0506... OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a914b7...",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "t1adiqi7CknCS8Enfj1TWpxZmxQjGNAQqbS"
        ]
      }
    },
    {
      "value": 0.00150000,
      "valueZat": 150000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 1ef6d210... OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9141e...",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "t1LhKxV1tjPwfPJM5ym8jmgK2vxvios3Ffa"
        ]
      }
    }
  ],
  "vjoinsplit": [
  ]
}
```

Esta no es la representación literal de la transacción codificada en hexadecimal. Algunos datos están repetidos (son el mismo dato representado de dos maneras distintas como en *value* y *valueZat* y en *hex* y *asm*) y otros están codificados dentro de otros campos (*overwintered* está codificado dentro del campo *version*). Esta es la representación que nos devuelve zcashd, y nos es útil para analizar la transacción.

Como podemos observar, la transacción tiene campos de metadatos (txid, overwintered, version y locktime), una lista de entradas (vin), una lista de salidas (vout) i una lista vacía de JoinSplits (vjoinsplit).

Entre los metadatos encontramos:

- *txid*: el hash de la transacción que sirve de identificador.
- *version*: versión de la transacción. Si es 1, la transacción es como en Bitcoin; 2, la transacción que soporta JoinSplits; 3, la transacción soporta campos nuevos de Overwinter; y si es 4, la transacción está codificada siguiendo el formato de Sapling, que cambia considerablemente.
- *overwintered*: Es un flag que llevan las transacciones a partir de la actualización Overwinter. Como veremos en el Capítulo 8, la actualización de Overwinter hace más fácil el despliegue de actualizaciones posteriores, y por eso es importante indicar que la transacción se tiene que recoger por los mineros siguiendo las reglas de Overwinter.
- *locktime*: Tiempo o número de bloques hasta el cual no se debe incluir una transacción en un bloque.

En la lista de entradas encontramos listado el UTXOs que consume la transacción. El UTXO se identifica por el hash de la transacción que lo produjo y un índice (vout), ya que puede haber varias salidas en una transacción. Además, para que el input sea válido se tiene que proveer una firma (scriptSig) conforme el propietario del UTXO está conforme con el gasto. En este caso estamos gastando un UTXO que si consultamos en la transacción anterior es de 0.0096 ZEC. El campo sequence es heredado de Bitcoin. Sirve para cancelar el locktime de una transacción anterior, pero fue desactivado en Bitcoin para evitar ataques de denegación de servicio. Se mantiene en Zcash para preservar la compatibilidad hacia atrás.

En la lista de salidas encontramos las salidas transparentes que produce la transacción. En este caso son dos, una por valor de 0.008 ZEC y otra que tal como hemos dicho antes, es el cambio. Ambos usan el script estándar para transacciones del tipo *Pay to PubKey Hash* (P2PKH). Los scripts de Zcash son iguales a los de Bitcoin (excepto por un OPCODE que ha sido eliminado[1]), y por lo tanto las transacciones transparentes ofrecen la misma versatilidad: se puede pagar a multigs y otros scripts.

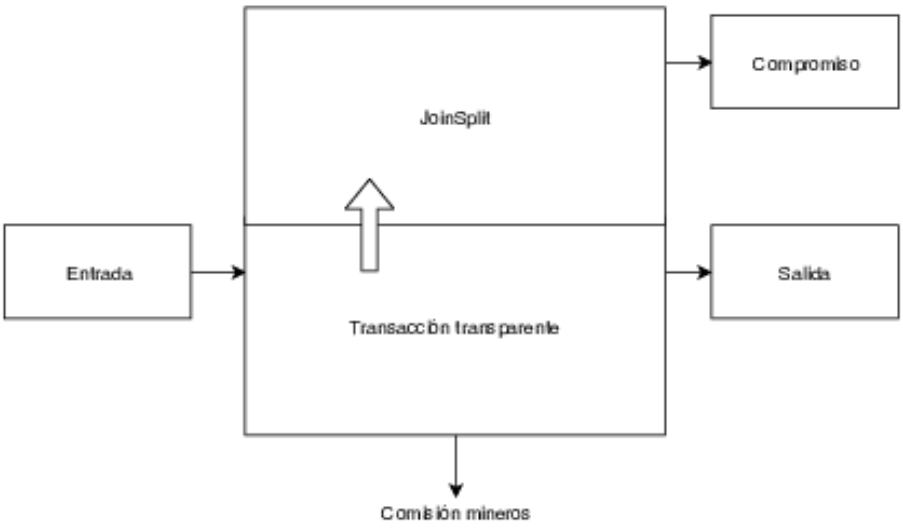
También vemos que el campo vjoinsplit está vacío. Este campo sirve para mostrar las transferencias de valor blindadas, y está vacío porque las transacciones con versión 1 no pueden tenerlo, ya que Bitcoin no soporta esa funcionalidad.

Vemos que si restamos las salidas a las entradas, estamos dejando 0.0001 ZEC de comisión al minero que incluya la transacción en un bloque.

[1]El opcode OP_CODESEPARATOR que ya no afecta más al cálculo del hashes de firmas.

3.1.2. Transacción blindante

Figura 3.2: Representación de una transacción blindante, donde hay una entrada, una salida (dirección de cambio), un v_{pub}^{old} positivo, y un compromiso que puede ser gastado en el futuro.



Ahora veamos qué pasa si enviamos dinero de una dirección transparente a una blindada. Hemos consumido el UTXO de 0.008 de la transacción anterior para enviarle 0.005 a una dirección blindada y recibir el cambio en una dirección nuestra. Además, al ser una transacción blindante, se puede escribir un texto cifrado en el memo de hasta 512 bits, y que va a recibir la persona que reciba el dinero, y sólo ella podrá leer. Nosotros hemos escrito el siguiente texto:

"The only way to keep a secret is to never have one. -Julian Assange"

La transacción puede consultarse en la blockchain, ocupa 2090 bytes y tiene el siguiente identificador:

3dd5ee2c5e8066e18e3100cf33eb7c26e1b1e9ede3b844bcbfd45bab603e9e20

Si sacamos el JSON con `decoderawtransaction` obtenemos:


```
{
  "txid": "3dd5ee2c...",
  "overwintered": false,
  "version": 2,
  "locktime": 0,
  "vin": [
    {
      "txid": "97b24ba5...",
      "vout": 0,
      "scriptSig": {
        "asm": "30440220... 02eb5951...",
        "hex": "47304402..."
      },
      "sequence": 4294967295
    }
  ],
```

```
  "vout": [
    {
      "value": 0.00290000,
      "valueZat": 290000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 1ef6d210... OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9141e...",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "t1LhKxV1tjPwfPJM5ym8jmgK2vxvios3Ffa"
        ]
      }
    }
  ],
  "vjoinsplit": [
    {
      "vpub_old": 0.00500000,
      "vpub_new": 0.00000000,
      "anchor": "3ea411eb...",
      "nullifiers": [
        "fade0fd6...",
        "70e5537e..."
      ],
      "commitments": [
        "7c288624...",
        "66e1e808..."
      ],
      "onetimePubKey": "26f1081e...",
      "randomSeed": "2b00fe25...",
      "macs": [
        "a6172df9...",
        "b008bc12..."
      ],
      "proof": "03094516...",
      "ciphertexts": [
        "47d53a42...",
        "49c2cd36..."
      ]
    }
  ]
}
```


Como podemos ver, en la lista de entradas tenemos el id de la transacción anterior y el índice `vout` es 0 porque estamos consumiendo la primera salida de esa transacción. En las salidas sólo vemos nuestra dirección de cambio, porque el traspaso de valor a una dirección blindada se hace dentro del JoinSplit.

En esta ocasión el campo `vjoinsplit` sí que está lleno, y aunque no se observa en el JSON que mostramos porque los hexadecimales están cortados, es el que ocupa más tamaño (1802 bytes por JoinSplit).

Un JoinSplit tiene como campos:

- v_{pub}^{old} y v_{pub}^{new} indican cuanto valor transparente recoge o deja disponible el JoinSplit.
- La *anchor* es la raíz del árbol Merkle de compromisos sobre el que se construye el JoinSplit.
- Los *nullifiers* son equivalentes a los números de serie de las monedas que se indican en el Capítulo 1, en Zcash se denominan nullifiers porque una vez en la blockchain, anulan el valor de esa moneda.
- Los *commitments* son los compromisos de las nuevas monedas (notas en nomenclatura de Zcash) pueden gastarse en el futuro al mostrar su nullifier.

- La *onetimePubKey* es una clave pública que se usa sólo una vez y que ha servido para cifrar los *ciphertexts*.
- La *randomSeed* es un número aleatorio de 256 bits generado para cada JoinSplit.
- Los *macs* son los códigos de autenticación de mensaje que vinculan las claves privadas con la transacción como mecanismo anti-*malleability*.
- La *proof* es la prueba de conocimiento nulo tipo zk-SNARK que demuestra que los nullifiers son correctos para compromisos dentro del árbol Merkle.
- Los *ciphertexts* son textos cifrados para el receptor, donde se incluye el *memo* cifrado.

Entraremos en más detalle en cómo se relacionan estos campos en el Capítulo 4. De momento nos centraremos en entender v_{pub}^{old}, v_{pub}^{new}, los nullifiers y los compromisos.

Vemos que en este caso el v_{pub}^{old} es de 0.005 ZEC y el v_{pub}^{new} es de 0 ZEC, porque hay 0.005 ZEC que pasan de ser públicos a estar blindados.

En esta transacción, los nullifiers no están anulando ningún compromiso previo, uno de los dos compromisos contiene 0.005 ZEC, y su ciphertext correspondiente contiene la frase de Julian Assange que hemos escrito en el memo, pero esto no puede saberlo alguien que esté observando la transacción sin saber nada del contexto, sólo si está en posesión de la clave privada de visionado.

Nótese también que la diferencia entre el valor de las entradas (0.008 ZEC) y el de las salidas (0.0029 ZEC), menos el valor de v_{pub}^{old} (0.005 ZEC) es de 0.0001, que es la comisión que puede cobrar el minero.

3.1.3. Transacción desblindante

A continuación transferimos dinero desde una dirección blindada a una dirección transparente. Consumimos un compromiso que tiene 0.004 ZEC para transferir 0.003 a una dirección transparente, y el resto vuelve a la dirección privada. En este caso no hemos escrito nada en el memo, pero este sigue estando, ahora relleno de zeros. El JoinSplit ocupa una medida fija, 1802 bytes, por lo que añadir datos o no en ese campo no afecta al tamaño de la transacción, ni tampoco al coste de sus comisiones. La transacción ocupa 1943 bytes y este es el su id:

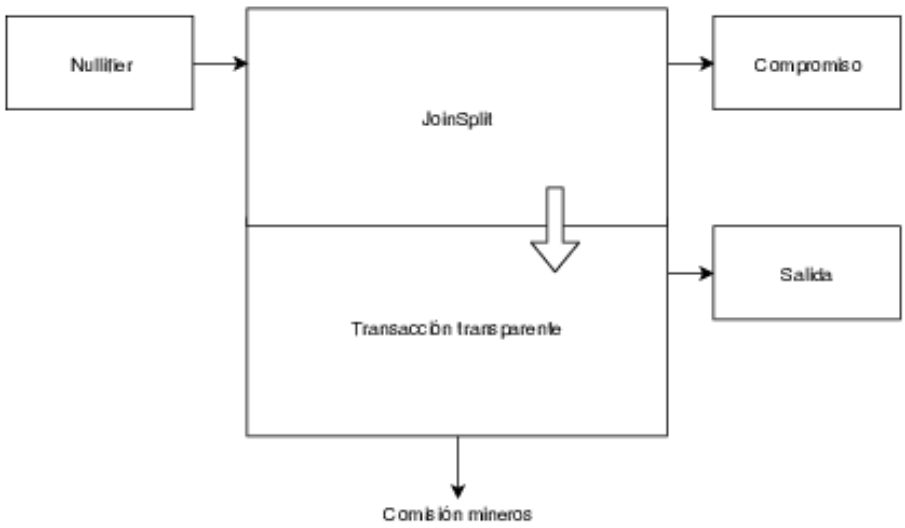
3fe0cad35fbc354cf8b064c7183a4478ae7531989457890b2149899bfdd7234c

Y este su JSON:


```
{
  "txid": "3fe0cad3...",
  "overwintered": false,
  "version": 2,
  "locktime": 0,
  "vin": [
  ],
  "vout": [
    {
      "value": 0.00300000,
      "valueZat": 300000,
      "n": 0,
```

```
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 1ef6d210... OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9141e...",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "t1LhKxV1tjPwfPJM5ym8jmgK2vxvios3Ffa"
        ]
      }
    }
  ],
  "vjoinsplit": [
    {
      "vpub_old": 0.00000000,
      "vpub_new": 0.00310000,
      "anchor": "98a972ea...",
      "nullifiers": [
        "656dd5ba...",
        "8a8a504c..."
      ],
      "commitments": [
        "6ee2a9af...",
        "923af123..."
      ],
      "onetimePubKey": "40fa9a63...",
      "randomSeed": "ef7c424c...",
      "macs": [
        "03f2202c...",
        "6518b75e..."
      ],
      "proof": "022e1728...",
      "ciphertexts": [
        "765b9c25...",
        "cb9d7a19..."
      ]
    }
  ]
}
```


Figura 3.3: Transacción desblindante, donde no hay entradas pero si hay salidas transparentes, porque hay un nullifier que invalida un compromiso previo. Un compromiso nuevo es generado con el valor del cambio.



La información que todo el mundo puede ver es que no hay entradas y sólo hay una salida de 0.003 ZEC a la dirección t1LhKxV1tjPwfPJM5ym8jmgK2vxvios3Ffa.

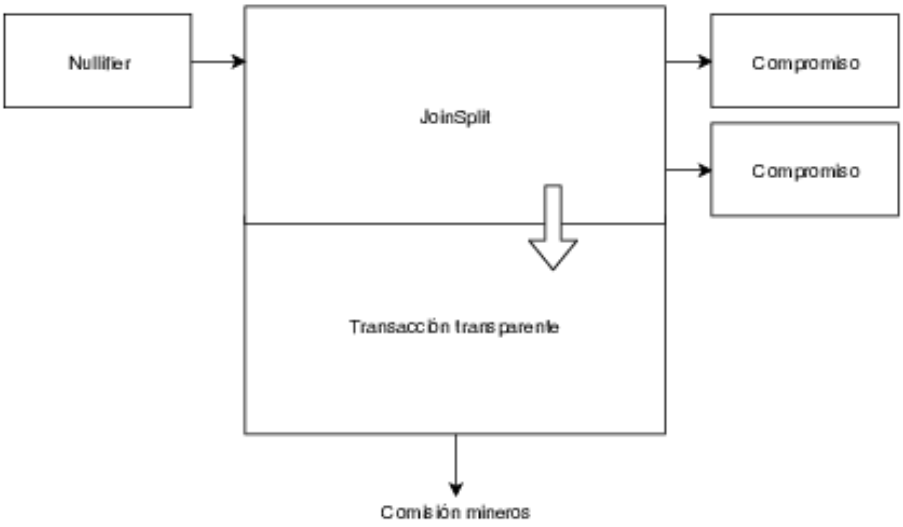
El v_{pub}^{new} de 0.0031 ZEC es el que permite que eso sea posible (si no, estaríamos creando dinero de la nada), y deja una comisión de 0.0001 al minero.

Nosotros, que hemos generado la transacción y tenemos la clave privada de visionado, podemos saber que uno de los dos nullifiers consume el compromiso de 0.004 ZEC que usamos como entrada. También podemos saber que uno de los dos compromisos tiene un valor de 0.0009 ZEC y está asignado a la misma dirección blindada con la que hemos hecho la transferencia (lo vemos con la operación `z_listunspent`).

Vemos entonces, que las transacciones originadas con una dirección transparente reciben el cambio en una dirección transparente nueva, mientras que las originadas con una dirección blindada, reciben el cambio en si mismas sin que eso represente un problema de privacidad, como si pasa en Bitcoin.

3.1.4. Transación privada

Figura 3.4: Transacción sin entradas ni salidas públicas. Un nullifier es publicado y por tanto un compromiso previo deja de estar disponible. Dos compromisos nuevos se crean, uno para realizar el pago y otro para recoger el cambio.



Por último vemos qué pasa al enviar dinero de una dirección blindada a otra. Consumimos un compromiso de 0.005 para enviar 0.004 ZEC a otra dirección. Nos corresponden 0.0009 ZEC de cambio, porque pagamos 0.0001 ZEC de comisión. Hemos escrito en el memo una frase larga para ilustrar la cantidad de texto que cabe en 512 bits:

> "The world is not sliding, but galloping into a new transnational dystopia. This development has not been properly recognized outside of national security circles. It has been hidden by secrecy, complexity and scale. The internet, our greatest tool of emancipation, has been transformed into the most dangerous facilitator of totalitarianism we have ever seen. The internet is a threat to human civilization. [...]

Cryptography is the ultimate form of non-violent action. — Julian Assange, Cypherpunks book."

La transacción tiene un tamaño de 1909 bytes y su identificador es:

c00b09d709e1a1a73ef399631aa1016eaa327dadd24b18f5b5c430e353e988f8


```
{
  "txid": "c00b09d7...",
  "overwintered": false,
  "version": 2,
  "locktime": 0,
  "vin": [
  ],
  "vout": [
  ],
  "vjoinsplit": [
    {
      "vpub_old": 0.00000000,
      "vpub_new": 0.00010000,
      "anchor": "024d6376...",
      "nullifiers": [
        "8f7d4154...",
        "7340ddd5..."
      ],
      "commitments": [
        "2b377116...",
        "b3e581a3..."
      ],
      "onetimePubKey": "7cbc3a67...",
      "randomSeed": "dcc4176f...",
      "macs": [
        "041c6727...",
        "fdbaa024..."
      ],
      "proof": "0312a4ce...",
      "ciphertexts": [
        "47469c7e...",
        "c71affbf..."
      ]
    }
  ]
}
```


Como podemos ver, tanto las entradas como las salidas están vacías, y sólo hay un JoinSplit que hace transparente 0.0001 ZEC. Como no es consumido por ninguna salida, es la comisión que recibe el minero. Los mineros siempre reciben sus ganancias (recompensa de bloque más comisiones) en direcciones transparentes.

Sabemos que uno de los nullifiers consume el compromiso de 0.005 ZEC, uno de los compromisos generados contiene 0.004 ZEC que puede ser consumido por la clave privada de la dirección blindada al que lo hemos enviado, y el otro compromiso contiene 0.0009 ZEC, que es el cambio y podemos gastar en otra ocasión.

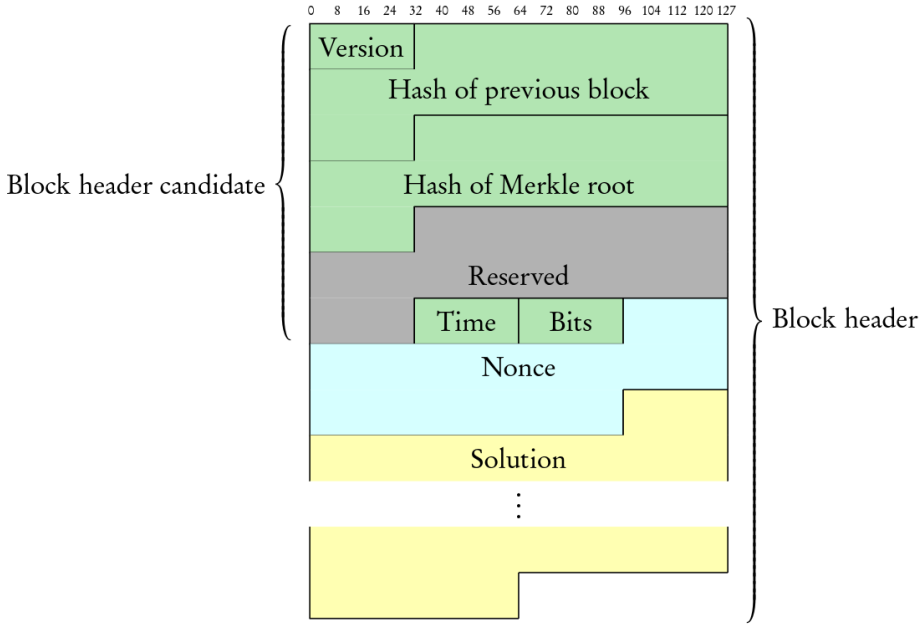
Obsérvese que usemos uno o dos nullifiers, uno o dos compromisos, o uno o dos textos cifrados, en todas las transacciones siempre aparecen dos de cada. Eso es un desperdicio de espacio en la blockchain, pero hace mucho más resistente a Zcash respecto a los ataques de análisis de grafo de transacciones.

Mientras que una transacción transparente (sin JoinSplits) suele ocupar unos 200 bytes como en Bitcoin, una transacción que usa o genera fondos blindados suele ocupar 2 kilobytes si usa un JoinSplit. Y se necesitan más JoinSplits si son necesarios más de dos nullifiers o se requiere generar más de dos compromisos. Si todas las transacciones de Zcash fueran blindadas, el crecimineto de tamaño

de la blockchain sería diez veces más elevado que en Bitcoin. Es por eso que, como vemos en el siguiente apartado, los bloques en Zcash son más grandes y se producen con más rapidez.

3.2. Análisis de los bloques

Figura 3.5: Cabecera de un bloque de Zcash. Imagen original de David A. Dalrymple. https://github.com/davidad/libeqh/blob/master/doc.pdf.



Al igual que Bitcoin, las transacciones de Zcash son recogidas en bloques. Todos los bloques incluyen metadatos en sus cabezeras. Podemos consultar la cabecera de un bloque con la operación `getblockheader`, que nos devuelve un JSON como el siguiente:

```
{
  "hash": "01731559ebff...",
  "confirmations": 23,
  "height": 324454,
  "version": 4,
  "merkleroot": "72926b0fd04a...",
  "time": 1526502026,
  "nonce": "000000000000...",
  "solution": "f38e82ab629c...",
  "bits": "1c0f33ad",
  "difficulty": 8829017.308570275,
  "chainwork": "000000000000...",
  "previousblockhash": "02c3b1d76698...",
  "nextblockhash": "039cd84041b4..."
}
```

De nuevo, el JSON que nos devuelve Zcash no se corresponde con lo que está realmente codificado dentro de la cabecera. Sería absurdo guardar en la blockchain el número de confirmaciones que tiene un bloque, o cuál es el hash del siguiente bloque. Lo que está haciendo zcashd al mostrar el JSON es combinar la información de la cabecera del bloque con datos que el demonio tiene sobre éste.

Los datos que se guardan en la cabecera, tal y como aparecen en la especificación del protocolo, son: la versión del bloque, el hash del bloque previo, el hash de

la raíz del árbol Merkle que contiene las transacciones, un espacio que se reserva por si en el futuro hace falta codificar otro hash, el tiempo en el que el minero empezó a minar el bloque, el campo nBits donde se especifica la dificultad de minado, la *nonce* que sirve al minero para poder probar números distintos hasta encontrar un bloque válido, y la solución al problema Equihash. En la Figura 3.5 podemos ver representado un bloque con sus campos y los bits que ocupa cada uno.

Los campos que son nuevos respecto a Bitcoin son: el espacio reservado para un hash por si hace falta en el futuro y la solución al Equihash. Eso es debido a que además de la nonce, para verificar una solución Equihash se necesita una lista de bits, y ahí reside la necesidad de ese campo extra. Veremos más sobre la solución de Equihash en el Capítulo 6.

Además, se aplican algunas reglas distintas a los bloques de Zcash, siendo la más importante que el tamaño máximo de bloque sea de 2 MB en lugar de 1 MB.

Como es habitual en Bitcoin, la primera transacción de cada bloque será la que pague al minero una recompensa por su trabajo, y es llamada coinbase. En el caso de Zcash, además de recompensar al minero (en 10 ZEC a 2018), la coinbase recompensa también a los fundadores (en 2,5 ZEC). Para recoger esa recompensa, tanto mineros como fundadores deben transferir esos fondos a direcciones blindadas, usando la operación diseñada para ello `z_shieldcoinbase`.

Los fundadores sólo son recompensados de esa forma los primeros cuatro años (840000 bloques minados). Las direcciones a las que se asignan los fondos se escogen de una lista especificada en el protocolo y codificada en la implementación de referencia. Volvemos a hablar de la recompensa a los fundadores en el Capítulo 7.

4. Privacidad en Zcash

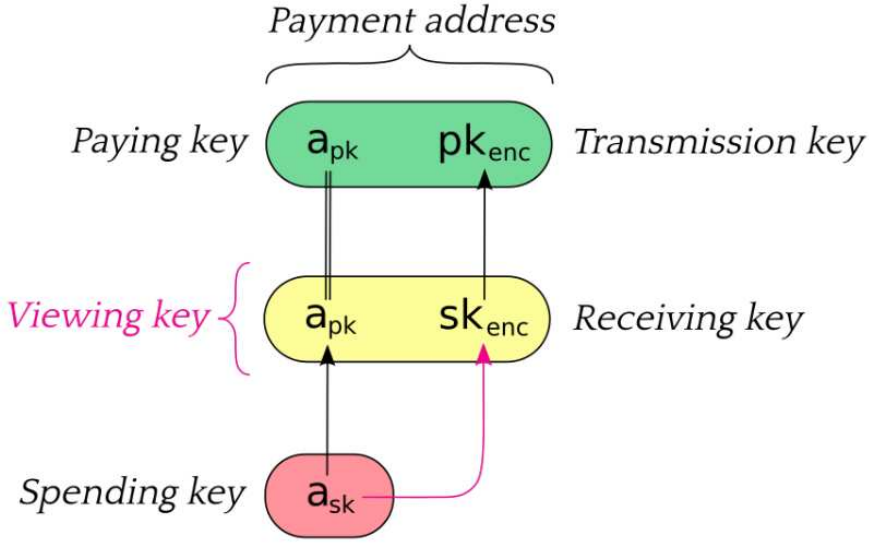
4.1. Direcciones blindadas

Las direcciones blindadas de Zcash son aproximadamente tres veces más largas que las transparentes, ya que en lugar de especificar el hash de la clave pública o script de 160 bits, especifican dos clave públicas de 256 bits, una de pago a_{pk} y otra de transmisión pk_{enc}. La primera corresponde a la clave privada a_{sk} que sirve para mover fondos de una dirección a otra, y la segunda corresponde a la clave privada sk_{enc} que sirve para descifrar los mensajes recibidos por esa dirección blindada, y de esa forma obtener también información necesaria para recibir los pagos.

La clave privada de transmisión sk_{enc} se puede usar conjuntamente con la clave pública de pago a_{pk} para observar las transacciones blindadas que ha recibido una dirección, sin tener la capacidad de gastarlos. Eso es útil cuando un servicio web debe gestionar los pagos de sus usuarios (compras, suscripciones...), pero se desea que esos fondos estén protegidos en un *cold storage* (tener las claves privadas guardadas offline). El servicio web tendría esa clave de visionado (a_{pk}, sk_{enc}) con la que podría gestionar las compras que hacen sus usuarios sin el inconveniente de poner en riesgo los fondos si es hackeado.

Sólo hace falta la clave de pago privada a_{sk} para derivar la clave de pago pública a_{pk} y la clave privada de transmisión sk_{enc}, de la que se puede derivar también la clave pública de transmisión pk_{enc}, tal como vemos en la Figura 4.1[1].

Figura 4.1: A partir de la clave de pago podemos derivar la clave de visionado, y de ésta, la dirección blindada. Imagen original del protocolo de Zcash.



Entonces, si decodificamos en base58[2] la siguiente dirección blindada:

[1]En este trabajo estamos simplificando la nomenclatura. En Zcash llaman a a_{sk} *spending key*; a a_{pk}, *paying key*; a sk_{enc}, *receiving key*; y a pk_{enc}, *transmission key*.

[2]Podemos usar http://lenschulwitz.com/base58

`zcH9dehq4w2WbLUBDHjjzzLD8q9kM5yWCNdjr6oMZqYYUisnMpb9WJBuy8j5RLxpEs`
`EMGFxqCkYjicRCPGZWQjDJWGzyT6e`

Observamos que los dos primeros bytes `0x169A` corresponden a la versión de la dirección y a su red (en este caso es la red de producción, ya que para testnet sería `0x16B6` y la dirección empezaría por `zt`). Los siguientes 32 bytes (0x44..A2) corresponden a la clave pública de pago, y los siguientes 32 (0x11..16) a la clave pública de transmisión. Los últimos 4 bytes (0x8AE8985F) no están recogidos en la especificación, y corresponden al *checksum* que se calcula con el hash SHA-256 doble de los 66 bytes anteriores y cogiendo los cuatro primeros bytes. El mismo mecanismo de *checksum* se aplica a las direcciones transparentes. Esta práctica se hereda de Bitcoin.

4.2. Notas

Una nota es la representación de un valor v que puede ser gastado por quien posee la clave privada de pago a_{sk}. Se trata de una tupla (a_{pk}, v, ρ, r) con estos parámetros:

- a_{pk}: clave pública a la que se destinan los fondos, su par puede retirarlos.
- v: denominación en zatoshis (10^{-8} ZEC) de la nota.
- ρ: equivalente al número de serie en Zerocoin, es el valor del que se deriva el *nullifier*.
- r: número aleatorio que sirve para generar el compromiso de la nota. Sin saber este número, el compromiso no revela ninguna información, mientras que con este número y el valor v se puede comprobar que el contenido de la nota no ha sido modificado.

En el capítulo anterior vimos que los JoinSplits registran en la blockchain una serie de valores, que para gastar una nota se tiene que publicar su nullifier para evitar que se vuelva a gastar en el futuro y que para generar una nota se tiene que publicar su compromiso para que pueda ser gastable. Recordemos también que los valores $v_{pub}^{old}, v_{pub}^{new}$ sirven para blindar y desblindar valor transparente y que en un JoinSplit se consumen dos notas y se generan dos notas nuevas, pudiendo, algunas de ellas, estar vacías.

En este capítulo definiremos cómo se generan los compromisos y *nullifiers*, y qué papel tienen el resto de parámetros en un JoinSplit. El compromiso de una nota es el hash SHA-256 de sus cuatro valores (a_{pk}, v, ρ, r), mientras que el *nullifier* se deriva con una función pseudoaleatoria (SHA256Compress) con ρ y la clave privada de pago a_{sk} como parámetros.

La prueba zk-SNARK demuestra que:

- El balance de entrada y salida del JoinSplit es correcto.
- Por cada nota de entrada no vacía, hay un compromiso anterior que revela que esa nota existe.

- El que genera la prueba sabe la clave privada que permite gastar las notas de entrada.
- Los *nullifiers* y los compromisos están calculados correctamente.
- La clave privada de pago a_{sk} firma la transacción entera, para evitar que un tercero modifique partes de la transacción (protección anti-*malleability*).
- Las notas de salida se han generado de manera que sus *nullifiers* no colisionen.

Fuera del zk-SNARK se comprueba también que los *nullifiers* que se usan como entradas al JoinSplit no hayan sido ya revelados.

4.3. Pagos blindados

En este apartado describiremos con más detalle los campos del JoinSplit que vimos en la Sección 3.1.

Supongamos que Alice posee una nota $(a_{pk}^{alice}, v^{old}, \rho^{old}, r^{old})$ y quiere transferirle cierta cantidad de dinero a Bob a través de una transacción blindada, devolviendo el resto a su dirección blindada de cambio. A grandes rasgos, estos son los pasos que sigue:

1. Genera un par de claves asimétricas para firmar los JoinSplits de la transacción, que usará como mecanismo anti-*malleability* y un número aleatorio randomSeed. Codifica la clave pública y el randomSeed dentro de la descripción JoinSplit.
2. Genera el *nullifier* de la nota que gasta con una función pseudoaleatoria y los parámetros ρ^{old} y la clave privada a_{sk}^{alice}. Genera también el *nullifier* de una nota vacía, ya que el JoinSplit necesita dos notas de entrada. Codifica los *nullifiers* en el campo *nullifiers* del JoinSplit.
3. Genera las dos notas nuevas $(a_{pk}^{bob}, v_1^{new}, \rho_1^{new}, r_1^{new})$ y $(a_{pk}^{alice}, v_2^{new}, \rho_2^{new}, r_2^{new})$ con los siguientes parámetros:
   - Las claves públicas de Bob y Alice a_{pk}^{bob} y a_{pk}^{alice}, respectivamente.
   - Los valores que se envían a Bob y que recibe Alice como cambio, v_1^{new} y v_2^{new}, respectivamente.
   - Los ρ_1^{new} y ρ_2^{new} que se generan a partir de una función pseudoaleatoria cogiendo como parámetros la randomSeed, los *nullifiers* y la clave pública del JoinSplit que hemos generado en el primer paso.
   - Un par de números aleatorios r_1^{new} y r_2^{new}.
4. Cifra los valores $(v^{new}, \rho^{new}, r^{new})$ de cada nota tal como se explica en el Sección 4.4 y los codifica en el campo *ciphertexts*. También guarda la clave efímera usada en el campo *ephemeralKey*.
5. Genera los compromisos de las dos notas nuevas con un hash de cada una de ellas y los codifica en el campo *commitments*.

6. Genera la prueba zk-SNARK que demuestra que las notas gastadas y las generadas tienen el mismo valor y que se cumplen el resto de reglas del protocolo, sin desvelar ninguna otra información de la que ya se está demostrando. Lo codifica dentro del campo *zkproof*. También codifica en el campo *anchor* la raíz del árbol de compromisos sobre el que se realiza el JoinSplit.

7. Aleatoriza el orden de las notas de entrada y salida para minimizar el filtrado de información.

8. Genera el código de autenticación de mensaje que enlaza la clave privada de gasto de Alice a_{sk}^{alice} con la clave pública del JoinSplit (generada en el primer paso) para demostrar que el JoinSplit está autorizado por Alice. Genera otro código para la nota vacía.

9. Firma toda la transacción usando la clave privada generada en el primer paso, para evitar que esa lista de JoinSplit sea usada en otra transacción.

10. Finalmente envía la transacción a la red, para que sea eventualmente incluida en un bloque.

Al contrario que en Bitcoin, no se sabe qué notas de la blockchain han sido ya gastadas y cuales no. En lugar de un conjunto de UTXOs, para el valor blindado de Zcash se mantiene un árbol Merkle con todos los compromisos que se han hecho por los usuarios y un conjunto de *nullifiers* que invalidan algunos de ellos. Sin la clave privada de pago a_{sk} de esos compromisos, no se puede saber si han sido gastados o no.

4.4. Distribución de secretos

No se pueden obtener los valores (a_{pk}, v, ρ, r) de la nota a partir de un compromiso, ni siquiera poseyendo la clave privada de pago a_{sk} a la que la nota de ese compromiso paga. Es por eso que se tienen que hacer llegar los valores (v, ρ, r) al receptor del dinero, para que pueda cobrarlo.

Se podría hacer usando cualquier sistema de mensajería que proteja la confidencialidad del mensaje, pero en Zcash se ha preferido cifrar esos datos y registrarlos en la blockchain para que se puedan recuperar los fondos que ha recibido una dirección blindada exclusivamente con la clave privada a_{sk}, sin tener que guardar un registro de cobros *out-of-band*. Esos (v, ρ, r), más el *memo* cifrado, son los campos que se guardan en el campo *ciphertexts* de los JoinSplit.

Para cifrar los datos se crean un par de claves efímeras asimétricas y se computa un secreto compartido con la clave pública de transmisión pk_{enc}, utilizando un esquema de acuerdo de claves basado en curvas elípticas. De ese secreto compartido se deriva una clave utilizando otros parámetros conocidos por ambas partes con una función hash, y finalmente se cifra el texto plano con un algoritmo de cifrado simétrico de una sola vez[3].

[3]Aunque dicho algoritmo, chacha20-poly1305, sí permitiría cifrar múltiples textos con la misma clave, el protocolo Zcash usa 0 como nonce, y por eso se puede usar una sola vez.

Con la clave privada de visionado (a_{pk}, sk_{enc}) se puede escanear la blockchain en busca de notas que se puedan descifrar. Esas son transacciones blindadas dirigidas a a_{pk} y de las que podemos desvelar su valor v, los otros dos parámetros ρ y r, y el *memo*.

Nótese que para saber si una nota ha sido gastada, hace falta tener conocimiento de la clave privada de pago a_{sk}, ya que esta es necesaria para derivar el *nullifier* y comprobar si este está o no está ya gastado. Eso significa que sólo con la clave de visionado no se puede saber el saldo de una dirección blindada.

5. zk-SNARKs

5.1. Pruebas de conocimiento nulo

Informalmente, una prueba de conocimiento nulo es un procedimiento por el cual alguien (*el que prueba*) puede convencer a otra persona (*el que verifica*) de que un cierto hecho es cierto sin darle a esa persona ninguna información extra que lo que se está probando.

Por ejemplo, el que prueba puede demostrar que conoce la raíz cuadrada de un número y módulo $N = pq$ sin desvelarla, es decir, conoce x tal que:

$$x^2 \equiv y \pmod{N}$$

Encontrar una raíz cuadrada de un número módulo un número compuesto es equivalente a factorizar ese número compuesto, y si p y q son números primos grandes, es computacionalmente costoso encontrar esa raíz cuadrada. Criptosistemas como el de Rabin o el de Goldwasser-Micali están basados en esa dificultad.

Para que el que prueba pueda demostrar que conoce x sin dar ninguna información de x al que verifica, se puede usar el siguiente protocolo interactivo:

1. El que prueba elige un número aleatorio r módulo N y envía al que verifica $s \equiv r^2 \pmod{N}$.
2. El que verifica elige $\beta \in \{0, 1\}$ y envía β al que prueba.
3. El que prueba calcula y envía al que verifica $z \equiv \begin{cases} r \pmod{N} & \text{si } \beta = 0 \\ xr \pmod{N} & \text{si } \beta = 1 \end{cases}$
4. El que verifica calcula z^2 y comprueba que $z^2 \equiv \begin{cases} s \pmod{N} & \text{si } \beta = 0 \\ ys \pmod{N} & \text{si } \beta = 1 \end{cases}$

Se puede comprobar que eso es cierto porque:

$$z \equiv x^\beta r \pmod{N}$$
$$z^2 \equiv x^{2\beta} r^2 \equiv y^\beta s \pmod{N}$$

Por lo que si es correcto, entonces el que verifica acepta la prueba. Si x no es raíz cuadrada de y hay un 50 % de pasar la prueba, por lo que si se repite el protocolo n veces (de ahí que sea interactivo) y todas las pruebas recibidas son válidas, el que verifica tiene la certeza de que el y es un cuadrado módulo N, con una probabilidad negligible de 2^{-n} de que no lo sea.

De esta manera, el que prueba pude demostrar que conoce x tal que $x^2 \equiv y \pmod{N}$ sin desvelar x a través de una prueba interactiva de conocimiento nulo. Esta es la primera prueba de conocimiento nulo para un problema concreto, presentada en 1985 por Goldwasser et al. [9]. Desde entonces, el campo de las pruebas de conocimiento nulo ha sido muy estudiado en criptografía.

5.1.1. Pruebas de conocimiento nulo no-interactivas

Blum et al. [5] desarrollaron las primeras pruebas de conocimiento nulo no-interactivas en 1988. La interacción entre el que prueba y el que verifica es reemplazada por una cadena de bytes aleatorios compartida entre los dos. El concepto se ha desarrollado en muchos otros sistemas, tal como muestra el *survey* de Wu and Wang sobre pruebas de conocimiento nulo no-interactivas [23].

Las pruebas de conocimiento nulo de Zerocoin y Zerocash deben ser del tipo no-interactivo, ya que tienen que publicarse en la blockchain y poder verificarse públicamente.

Aunque teóricamente se puede realizar una prueba de conocimiento nulo no interactiva sin la necesidad de esa cadena [23], la única manera conocida de producir pruebas lo suficientemente cortas como para poder ser publicadas en una blockchain requiere de esa cadena de referencia común (CRS). Esa cadena, que forma parte de los parámetros públicos del sistema, tiene que ser calculada en una fase de configuración inicial.

El problema es que si alguien tuviera acceso a los números aleatorios secretos usados para generar esa cadena, podría crear pruebas falsas que parecerían verdaderas. Es por eso que ese secreto también es llamado *basura tóxica*[1], ya que es un desecho indeseable de esa fase inicial.

En Zcash, quien tuviera acceso a ese secreto podría falsificar las pruebas de conocimiento nulo de los JoinSplits, permitiendo saltarse las reglas del protocolo, y generar moneda de la nada, además de pasar desapercibido. No podría, en cambio, conectar los compromisos de otras personas con los *nullifiers*, por lo que la privacidad de los usuarios quedaría resguardada aún y en ese fatídico caso.

En la Sección 5.3 veremos como se puede calcular esa cadena de manera segura, a partir de cálculos de múltiples participantes, de manera que si al menos uno de ellos es honesto, queda garantizado que no se puede conocer la basura tóxica del sistema. Antes, veamos con más detalle en qué consisten las pruebas de conocimiento nulo usadas en Zcash, los zk-SNARKS.

5.2. zk-SNARKs

El acrónimo zk-SNARK significa *Zero-Knowledge Succinct Non-Interactive Argument of Knowledge*. Son las pruebas de conocimiento nulo que utiliza Zcash.

Como hemos visto, con una prueba de conocimiento nulo se puede probar que un declaración es cierta, sin revelar ninguna información extra más allá de la validez de la declaración misma, como por ejemplo demostrar la existencia de la pre-imagen de un hash. En cambio, con una "prueba de conocimiento"de conocimiento nulo (zero-knowledge "Proof of Knowledge"), se puede demostrar que se conoce dicha pre-imagen.

Nos referimos a esa demostración como un *argumento* de conocimiento y no como una *prueba* de conocimiento, ya que éste solo es sólido ante un probador acotado polinomialmente. También es *succinct*, porque puede ser verificado en pocos milisegundos y su longitud es corta, de unos cientos de bytes.

Una prueba zk-SNARK demuestra que se han producido una serie de computaciones de un programa y que el resultado es correcto sin desvelar la información

[1] https://blog.z.cash/the-design-of-the-ceremony/

sobre la que se han realizado dichas computaciones. En Zcash, esas computaciones son las reglas del protocolo y la información que no es revelada son las notas, las claves privadas y los demás secretos necesarios para realizar la transacción blindada.

Para conseguir un zk-SNARK, primero hay que transformar la función que queremos verificar en un circuito aritmético, que es la representación matemática del programa. Eso significa que se *compila* el programa en las operaciones aritméticas suma, resta y multiplicación, en lugar de compilarlo en las operaciones lógicas AND, OR y NOT.

A partir del circuito aritmético construimos un sistema de restricciones de rango 1 (también llamado R1CS) para asegurarse que el circuito aritmético se recorre en el sentido correcto. En esta representación el que verifica debe comprobar muchas restricciones, una por casi cada cable del circuito.

En 2012, Gennaro et al. [8] presentaron una manera de simplificar todas esas restricciones en una sola, una restricción entre polinomios en lugar de entre enteros, usando la representación de circuito llamada Quadratic Arithmetic Program (QAP). Esos polinomios pueden llegar a ser muy grandes, pero como es muy improbable que dos polinomios diferentes compartan la gran mayoría de puntos, podemos dar por válida una prueba comprobando que concuerdan en un solo punto seleccionado aleatoriamente.

Si el que prueba supiera con anterioridad qué punto se va a usar para la verificación, podría generar un polinomio inválido que compartiera como mínimo ese punto, y se daría la prueba como válida. Para evitarlo, se usa cifrado homomórfico y pairings de curvas elípticas para evaluar los polinomios de manera ciega, sin saber qué punto aleatorio se está evaluando.

Los parámetros públicos son usados para determinar qué punto va a ser comprobado, pero de manera cifrada, haciendo que ni el que prueba ni el que verifica sepan cuál es. Por eso es tan importante eliminar la basura tóxica, porque con ese secreto se podría desvelar ese punto y falsear cualquier prueba.

Conseguimos que la prueba sea de conocimiento nulo aplicando un desplazamiento aleatorio a los polinomios originales. Al desplazar a ambos lo mismo, se puede seguir comprobando que cualquiera de sus puntos coincide.

Se puede encontrar una descripción más detallada de los zk-SNARKs en la página oficial de Zcash[2], junto con una serie de siete artículos.

El primer artículo explica la criptografía homomórfica. La criptografía homomórfica es aquella que permite operar los textos cifrados, tal que se puede calcular $E(x+y)$ a partir de $E(x)$ y $E(y)$ sin conocer x ni y. Es útil para generar pruebas de conocimiento nulo, ya que se permite operar sin conocer qué números se están operando, de manera ciega.

El segundo artículo explica la evaluación ciega de polinomios. Debido a que la criptografía homomórfica soporta combinaciones lineales, se puede calcular el resultado de un polinomio cifrado substituyendo su variable por un número sin conocerlo. A partir de las distintas potencias cifradas de ese número $E(1), E(s), \ldots, E(s^d)$, se puede calcular el resultado del polinomio cifrado $E(P(s))$ gracias a que P es una combinación lineal de $1, s, \ldots, s^d$.

El tercer artículo explica el test de conocimiento de coeficiente. Se trata de un test en el que se define un α-par como (a, b) tal que $b = \alpha a$ y ni a ni b son 0.

[2]https://z.cash/technology/zksnarks.html

El reto es encontrar un α-par diferente (a', b') que cumpla la misma propiedad. Para hacerlo se puede multiplicar el par por γ tal que $(a', b') = (\gamma a, \gamma b)$.

El cuarto artículo explica la evaluación ciega y verificable de polinomios que se construye extendiendo el test de conocimiento de coeficiente visto en el artículo anterior. La verificabilidad de la evaluación consiste en que se pueda demostrar que el resultado de $E(P(s))$ es correcto. Para ello extiende el test de conocimiento de coeficiente en el que se parte de d α-pares y hay que producir uno nuevo de la forma:

$$(a', b') = (\sum_{i=1}^{d} c_i a_i, \sum_{i=1}^{d} c_i b_i)$$

El quinto artículo explica cómo se transforma una computación en un polinomio. Detalla los pasos de cómo reducir un circuito aritmético a un QAP, obteniendo como resultado los polinomios $L_1, \ldots, L_m, R_1, \ldots, R_m, O_1, \ldots, O_m$ y el polinomio objetivo T, todos de grado d.

El sexto artículo explica el Protocolo Pinocchio. Ese protocolo usa la evaluación ciega y verificable de polinomios, vista en la parte cuatro, para comprobar que dos polinomios extraídos de la computación anterior son iguales. Explica cómo elegir esos polinomios y cómo asegurar el conocimiento nulo a partir de un desplazamiento realizado a los polinomios.

Por último, el séptimo artículo explica los pairings de curvas elípticas, ya que estos permiten la multiplicación homomórfica (sólo en algunos contextos), además del porqué son necesarios para la fase de configuración inicial de parámetros y hacer así pruebas no-interactivas.

Recomendamos la lectura de esos siete artículos si se quiere profundizar más en el funcionamiento de los zk-SNARKs. La librería libsnark[3] implementa los zk-SNARKs en C++. Recomendamos el tutorial de Howard Wu para empezar a usarla[4].

5.3. Generación de los parámetros públicos

La manera más simple de producir la cadena de referencia común para los zk-SNARKs también produce un tipo de *residuo tóxico* criptográfico que puede ser usado para falsificar pruebas de conocimiento nulo.

Para solucionar eso, Ben-Sasson et al. [2] desarrollaron un método por el cual múltiples participantes calculan partes de los parámetros públicos y los juntan más tarde, sin juntar nunca sus contrapartes privadas. Más tarde, Bowe et al. [6] diseñaron una ceremonia para llevar a cabo ese cálculo de una manera segura.

La cadena de referencia común (CRS) para los zk-SNARKs de Zcash tiene la forma: $(g, s\dot{g}, s^2\dot{g}, \ldots, s^d\dot{g})$ tal que g es el generador del grupo $\mathcal{G}$ (donde el logaritmo discreto se considera difícil), y $s \in \mathbb{F}^*$ es un número aleatorio[5].

Esta cadena de referencia es usada para probar que dos polinomios P y Q son iguales en un punto, y es indispensable para hacer la prueba no-interactiva. Siendo $P(X) = a_0 + a_1X + a_2X^2 + \ldots + a_dX^d$, entonces $P(s)g = a_0g + a_1sg + a_2s^2g + \ldots + a_ds^dg$, de los que $g, sg, s^2g, \ldots, s^dg$ se pueden encontrar en la cadena

[3] https://github.com/scipr-lab/libsnark

[4] https://github.com/howardwu/libsnark-tutorial

[5] https://blog.z.cash/generating-zcash-parameters/

de referencia común, y por lo tanto se puede calcular si $P(s) = Q(s)$ calculando si $P(s)g = Q(s)g$.

Para generar una cadena de referencia común entre dos partes (Alice y Bob), se puede seguir el siguiente protocolo (generaremos (sg, s^2g) por simplicidad):

- Alice elige un número $a \in \mathbb{F}^*$ aleatorio y envía $(A, B) = (ag, a^2g)$.
- Bob elige un número $b \in \mathbb{F}^*$ aleatorio y envía $M = (bA, b^2B)$

Por lo que obtienen $M = (abg, (ab)^2g)$. Si al menos una de las partes es honesta y destruye su número aleatorio, no se puede recuperar la parte privada s ya que $s = ab$.

Siendo $\mathcal{G}$ un grupo con un *bilinear pairing* que nos permite $e(ag, bg) = g^{ab}$, para comprobar que ambas partes han usado a^2 y b^2 en el segundo componente, podemos comprobar que $e(g, B) = e(A, A)$, ya que:

$$e(g, B) = e(g, s^2g) = g^{s^2} = e(sg, sg) = e(A, A)$$

5.3.1. La ceremonia de generación de parámetros

Entre el 21 y 23 de octubre de 2016 tuvo lugar la ceremonia de cómputo de los parámetros públicos usados en los zk-SNARKs de la versión 1.0 de Zcash[6][7].

La ceremonia tuvo lugar en seis localizaciones diferentes por seis personas: Andrew Miller (asesor técnico de Zcash), Peter Van Valkenburgh (director de investigación en Coin Center), John Dobbertin (pseudónimo), Zooko Wilcox (CEO de Zcash), Derek Hinch (del NCC Group) y Peter Todd (desarrollador de Bitcoin Core).

La ceremonia se llevó a cabo con tres defensas principales para evitar que un atacante pudiera hacerse con los secretos. Las tres defensas fueron la computación multi-parte descrita en la sección anterior, usar ordenadores no conectados a la red (airgapped) y la recogida de evidencias en discos no reescribibles (DVDs).

Todos los ordenadores usados fueron comprados unos días antes de la ceremonia por las personas que los iban a usar, y fueron destruidos al terminar ésta. Cada participante usó dos ordenadores, uno para realizar el cómputo de los parámetros, y otro para comunicarse con el resto de participantes. Antes de encender el ordenador de cómputo, se pidió a los participantes que les retiraran los dispositivos de Wi-fi y Bluetooth, y que nunca se los conectara a Internet. Se les instaló una distribución Linux preparada específicamente para el evento.

Tanto el código de los programas que se usaron, como las ISOs de los DVDs que recogieron las evidencias, como múltiples vídeos i las declaraciones de sus participantes después de la ceremonia están a disposición del público.

Las dos claves generadas, la que sirve para probar y la que sirve para verificar son las que se descargan en el directorio `.zcash-params` al instalar `zcashd`.

En noviembre de 2017 se realizó otra ceremonia para la generación de nuevos parámetros. Esta vez con un protocolo mejorado [7] que permitió participar a más personas. Se hizo un llamamiento a la comunidad y en total participaron 87 personas. Se espera realizar una nueva ceremonia en verano de 2018 para computar los nuevos parámetros de la versión Sapling de Zcash.

[6] https://blog.z.cash/the-design-of-the-ceremony/

[7] https://z.cash/technology/paramgen.html

6. Minería en Zcash

Una de las particularidades de Zcash es que usa Equihash como algoritmo de prueba de trabajo. En este capítulo veremos en qué consiste el algoritmo y cómo se está minando por los usuarios de la red. Antes, recordemos algunos conceptos de los algoritmos de prueba de trabajo.

6.1. Algoritmos de prueba de trabajo

Los algoritmos de prueba de trabajo deben requerir mucho trabajo computacional para ser ejecutados, y deben ser fácilmente verificables.

Por ejemplo en Hashcash [1], el algoritmo en el que se basa la prueba de trabajo de Bitcoin, cuesta mucho encontrar una solución, porque requiere encontrar una pre-imagen cuyo hash tenga unas propiedades concretas, y la mejor manera es la fuerza bruta. Para verificar que la solución es correcta, en cambio, sólo hace falta computar un solo hash y comprobar si tiene esas propiedades.

En el caso de Bitcoin, un minero calculará múltiples hash SHA-256d[1] a partir de la cabecera de un bloque, aplicando pequeños cambios en ésta (cambiando el campo *nonce*), hasta que consiga un hash menor a un número prefijado.

Esa asimetría entre la creación y la verificación de una solución a un puzzle es muy útil para crear una blockchain como Bitcoin. Gracias a que es difícil crear un nuevo bloque válido (una solución), los nodos pueden competir en crear bloques nuevos (encontrar soluciones) a partir de anteriores que tengan más prueba de trabajo, e ir así descartando las ramas con menos prueba de trabajo, creando consenso en cuál es la historia de las transacciones. Por supuesto, es muy importante que la prueba de trabajo sea rápida de verificar, para que los nodos puedan decidir rápidamente si seguir buscando soluciones en la rama en que se encuentran o pasarse a otra con más prueba de trabajo.

Además, en el diseño de un buen algoritmo de prueba de trabajo, se debe tener en cuenta que éste sea libre de progreso. Libre de progreso (*progress-free*) significa que cada intento por encontrar una solución tiene las mismas probabilidades de éxito que el anterior, es decir, que no se tiene en cuenta el trabajo realizado en el pasado. Por lo tanto, el periodo de tiempo para encontrar una solución está determinado solamente por la velocidad a la que los participantes pueden ejecutar la función de hash (el *hashrate*). Todo nuevo participante tiene las mismas posibilidades de encontrar una solución que otro que ya lleve tiempo, independientemente del momento en que empiece. Esta propiedad es esencial para que no se centralice fuerza de trabajo en unos pocos nodos.

6.1.1. ASICs

El requisito de que las soluciones sean fáciles de verificar ha hecho que la mayoría de algoritmos de prueba de trabajo no requieran de mucha memoria para computarse. Eso ha hecho que sea relativamente barato crear circuitos integrados diseñados específicamente para ellos.

[1] Bitcoin usa la función SHA-256 dos veces, por eso nos referimos a ella como SHA-256d, o "SHA-256 doble".

En 2013 fue el año en el que los circuitos ASIC se desplegaron el la red Bitcoin [21]. Eso supuso un incremento del *hashrate* total de la red, lo que dejó fuera a los mineros que usaban GPU, porque esa práctica dejó de ser rentable. Eso ha centralizado el minado de Bitcoin y otras monedas en los que pueden invertir en esos caros equipos de minería, y este hecho es visto por la comunidad como una consecuencia imprevista e indeseable del algoritmo de prueba de trabajo Hashcash.

Desde entonces se ha abierto un campo de investigación para buscar algoritmos de prueba de trabajo resistentes a las ASICs. Mayoritariamente se ha buscado algoritmos que requieran de un computador de propósito general, o que requieran grandes cantidades de memoria para calcular una solución al puzzle. En esta última categoría se encuentra Equihash.

6.2. Equihash

Equihash [4] es un algoritmo propuesto por Dmitry Khovratovich y Alex Biryukov, del grupo de investigación criptográfica CryptoLUX. Es un algoritmo de prueba de trabajo del tipo *memory-hard*, lo que significa que los mejores métodos resolver el puzzle necesitan grandes cantidades de memoria.

La fortaleza del algoritmo se basa en el problema de cumpleaños generalizado, para el que no se ha encontrado un algoritmo que lo solucione eficientemente sin necesidad de mantener los datos en memoria.

Recordemos que la paradoja del cumpleaños es un modelo probabilístico por el cual sabemos que encontrar una colisión entre dos hashes de una lista es más fácil que encontrar una colisión a un hash concreto.

Podríamos diseñar un algoritmo de prueba de trabajo basado en este problema, en el que el puzzle a resolver sería encontrar i, j tal que $H(S||i) = H(S||j)$ siendo S la cabecera del bloque que estamos minando. El problema es que podemos encontrar esa colisión sin necesidad de guardar grandes cantidades de memoria, usando el algoritmo ρ de Pollard [14].

Es por eso que Equihash no utiliza ese problema, sino su generalización, que consiste en encontrar $x_1, x_2, ..., x_{2^k}$ tales que $H(x_1) \oplus H(x_2) \oplus ... \oplus H(x_{2^k}) = 0$. Este problema también tiene un algoritmo que permite solucionarlo de manera eficaz [22], por lo que Equihash pone una condición extra para evitar armonizaciones (ver siguiente apartado).

Por último, para tener un algoritmo libre de progreso, tal como lo es Hashcash, se requiere que el hash del bloque resultante también sea menor que un valor predeterminado.

6.2.1. Especificación

Sean n y k parámetros predefinidos (para Zcash son $n = 200, k = 9$), y sea d la dificultad, la prueba de trabajo Equihash consiste en, a partir de una semilla I (la cabecera del bloque candidata), encontrar una *nonce* V y una solución $x_1, x_2, ..., x_{2^k}$ que:

- Cumpla con el problema del cumpleaños generalizado:

$$H(I||V||x_1) \oplus H(I||V||x_1) \oplus ... \oplus H(I||V||x_{2^k}) = 0$$

- Se eviten armonizaciones:

  $H(I||V||x_{w2^l+1})\oplus...\oplus H(I||V||x_{w2^l+2^l})$ debe tener $\frac{nl}{k+1}$ ceros a la izquierda

  para toda w, l tal que

  $$(x_{w2^l+1}||x_{w2^l+2}||...||x_{w2^l+2^{l-1}}) < (x_{w2^l+2^{l-1}+1}||x_{w2^l+2^{l-1}+2}||...||x_{w2^l+2^l})$$

- Cumpla con el filtro de dificultad:

  $$H(I||V||x_1||x_2||...||x_{2^k}) \text{ debe tener } d \text{ ceros a la izquierda}$$

Los parámetros n, k y d determinan los requisitos de tiempo y memoria de la siguiente manera:

- Se requieren $2^{\frac{n}{k+1}+k}$ bytes de memoria y $(k+1)2^{\frac{n}{k+1}+d}$ llamadas a la función hash H para ser calculado.
- El tamaño de la solución es de $2^k(\frac{n}{k+1}+1)+160$ bits.
- Sólo se requieren 2^k hashes y XORs para ser verificado.

Por lo que si n es alto y k es bajo, tenemos un algoritmo de prueba de trabajo que requiere de mucha memoria para se calculado y pocos ciclos de cómputo para ser verificado. Además, d no afecta ni al tamaño de la solución ni a su verificación.

En el bloque, además de la nonce V, hay que guardar la lista $x_1, x_2, ..., x_{2^k}$. Es por eso que existe ese campo *solution* en las cabeceras de bloque de Zcash, tal como se puede ver en la Figura 3.5 del capítulo donde analizamos las transacciones y los bloques de Zcash.

6.3. Minado en Zcash

La versión alpha de zcashd venía con una implementación trivial del algoritmo para solucionar Equihash. Estaba tan poco optimizado, que para la primera testnet se usaron unos parámetros menos demandantes de recursos ($n = 96, k = 3$), que después al implementar mineros más eficientes, fueron cambiados los actuales ($n = 200, k = 9$)[2].

Algunas de las optimizaciones que pueden usarse para solucionar un puzzle ya vienen en el propio paper de Equihash [4], como calcular el hash al vuelo o no guardar el índice entero (quinta sección, apartado A). Otras han sido muy investigadas en abierto por la comunidad de Zcash a raíz del concurso *Zcash Open Source Miner Challenge* financiado por la Zcash Company y organizado por Least Authority[3] en octubre de 2016.

En el concurso se pedían implementaciones de un algoritmo solucionador de puzzles Equihash, con una API específica, y concedía 10.000 dólares a la mejor implementación para CPU, otros 10.000 dólares a la mejor para GPU, y otros 10.000 repartidos entre los demás finalistas. Se animaba explícitamente a la participación en abierto y a reutilizar y construir sobre el código de los demás participantes.

Los ganadores y finalistas del concurso fueron:

[2]https://github.com/zcash/zcash/issues/856

[3]https://zcashminers.org/

- **equihash-xenon**: Ganó el primer premio para CPU.
- **silentarmy**: Ganó el primer premio para GPU.
- **Tromp's solvers**: Implementa tanto para CPU como para GPU. Quedó finalista, y ha sido la única herramienta que ha sido integrada más tarde en zcashd. No se usa por defecto, pero puede activarse añadiendo la línea *equihashsolver=tromp* en `zcash.conf`.
- **zceq_solver**: Otra implementación para CPU.
- **davidjaenson's solver**: Implementa tanto para CPU como para GPU.

Entre las optimizaciones que se investigaron se encuentran un correcto uso de multi-hilos, la portabilidad entre diferentes arquitecturas, las optimizaciones en el código (estructuras más eficientes o pequeños trucos para ahorrar bits), y la elección del algoritmo de ordenación que dé el mejor resultado.

La herramienta nheqminer de Nicehash[4] recoge algunos de estos solucionadores y los ofrece desde una sola herramienta. Además, permite conectarnos a una *pool* de minería.

6.3.1. Pools de minería

Las *pools* de minería son servicios privados abiertos al público que permiten coordinar a distintos mineros en la búsqueda de una solución de un mismo bloque. Cada minero recibe una recompensa proporcional a la cantidad de trabajo aportada. Eso hace que el minado sea más previsible y permite la entrada de pequeños mineros.

Uno de los protocolos más usados para coordinar el trabajo entre los mineros es Stratum. Este protocolo proviene de Bitcoin y supone una mejora respecto a protocolos anteriores, ya que anteriormente se consultaba al servidor de la pool con una estrategia tipo pulling HTTP, que hacía inviable su escalado. Stratum usa una estrategia tipo push, a través de TCP.

Si ejecutamos nheqminer y analizamos sus trazas de red con Wireshark, podemos captar los siguientes mensajes:

```
> {"id":1,"method":"mining.subscribe","params":["equihashminer/0.4b",
      null,"eu1-zcash.flypool.org","3333"]}
< {"id":1,"result":["007d525f18", "007d525f18"], "error": null}
> {"id":2,"method":"mining.authorize","params":
      ["t1adiqi7CknCS8Enfj1TWpxZmxQjGNAQqbS.worker1","x"]}
< {"id":2,"result":true, "error": null}
< {"id":null,"method":"mining.set_target",
      "params":["0000aec3..."]}
> {"id":3,"method":"mining.extranonce.subscribe","params":[]}
< {"id":null,"method":"mining.notify","params":["543c8d29..."
      "04000000","6032e159...","43ec87e0...","00000000...",
      "0131045b","0e410c1c",true]}
```

Como podemos observar, el primer mensaje que manda el minero a la *pool* es una petición de suscripción. Los parámetros usados son el nombre de un user-agent, el id de sesión nulo porque queremos empezar una nueva, y el host y

[4]https://github.com/nicehash/nheqminer

el puerto del servidor de la *pool*. El servidor nos devuelve un mensaje con el identificador de la sesión y la porción de *nonce* que quiere que minemos.

El siguiente mensaje que mandamos es de autenticación, con un nombre y una contraseña. En el nombre aparece la dirección transparente que recibe la recompensa. El servidor devuelve un mensaje de aceptación y otro en el que indica la dificultad del bloque que se está minando.

El siguiente mensaje le indica al servidor que puede cambiar la *nonce* que mandó al inicio, porque el cliente soporta esa función.

Por último, el servidor manda notificaciones al cliente con el identificador del trabajo, los campos del bloque que va a minar (la versión -4-, el hash del bloque anterior, el árbol Merkle de transacciones, el bloque reservado -todo ceros-, el tiempo y el número de bits), y un campo booleano CLEAN_JOBS que indica si el cliente debe parar todos los demás trabajos, ya que este los invalida.

Si el minero encuentra una solución parcial, manda un mensaje tipo `mining.submit` con los parámetros: nombre usado en la autenticación, nombre del trabajo, tiempo, segunda parte del *nonce* y solución Equihash.

Al igual que en Bitcoin, las pools de minería en Zcash no están exentas de polémica. A tiempo de escritura de este trabajo, la pool de minería Flypool genera más del 51 % de los nuevos bloques, y eso supone un grave problema para la supuesta descentralización de la red, ya que permite a Flypool realizar ataques del 51 %.

6.3.2. ASICs de Equihash

Otro agente que puede centralizar la red de Zcash es la aparición de ASICs. La empresa Bitmain ha anunciado el lanzamiento de su Antminer Z9 mini para junio de 2018, la primera ASIC especializada en Equihash. La noticia ha generado un gran revuelo en la comunidad, donde se discute la conveniencia de un *hard fork* para cambiar el algoritmo de minado, o al menos sus parámetros n y k.

Desde la Fundación Zcash se va a priorizar el estudio de la resistencia a las ASIC, pero piden prudencia ya que aún no está clara la importancia que van a tener estas en relación a los otros dispositivos dentro de la red[5]. Por otro lado, la Zcash Company también se ha pronunciado al respecto[6], diciendo que su prioridad en la actualidad son las inminentes actualizaciones de la red a Overwinter y Sapling, y proponen que la Fundación o otros individuos de la comunidad coordinen un ZIP para discutir el tema.

[5]https://z.cash.foundation/blog/statement-on-asics/

[6]https://blog.z.cash/zcash-company-statement-on-asics/

7. Consenso en Zcash

7.1. De la Compañía a la Fundación

Zcash parte siendo una start-up fundada por los investigadores que publicaron los papers de Zerocoin [11] y zk-SNARKs [3]. La compañía ha demostrado ser un buen vehículo para monetizar el desarrollo del proyecto, consiguiendo primero un millón de dólares y más tarde otros dos millones en sus dos primeras rondas de inversión.

El problema es que una compañía sirve a unos intereses particulares, y una criptomoneda descentralizada debería estar mantenida por desarrolladores que sirvan a los intereses de todos los usuarios por igual. Es por eso que los fundadores han destinado un 10 % del dinero recaudado con la recompensa de los fundadores (descrita en el siguiente apartado) a la Zcash Foundation, una organización sin ánimo de lucro que pueda coordinar la comunidad con respecto a la gobernanza de la red.

7.1.1. La recompensa de los fundadores

Al igual que en Bitcoin, cada cuatro años se divide por dos el dinero creado por cada bloque.

En el caso de Bitcoin, se crea un bloque cada 10 minutos y la recompensa inicial era de 50 BTC cada bloque. A tiempo de escritura de este trabajo, se ha reducido dos veces la recompensa por bloque, siendo ahora de 12.5 BTC.

En el caso de Zcash, se crea un bloque cada 2.5 minutos y para mantener una masa monetaria equivalente a Bitcoin, la recompensa inicial debe ser cuatro veces menor, de 12.5 ZEC (para que en 10 minutos sea 50 ZEC). La recompensa en el caso de Zcash aún no se ha reducido ninguna vez.

En Zcash hay la particularidad de que durante esos cuatro primeros años antes de la reducción de la recompensa, un 20 % del valor creado por el bloque debe ir a una dirección de los fundadores, especificada en el protocolo. Eso significa que por cada bloque minado, el minero percibe 10 ZEC y los fundadores 2.5 ZEC. Ese 20 % de las monedas distribuidas a los fundadores, un total de 2,1 millones de ZEC, va a representar el 10 % de las monedas totales de Zcash[1].

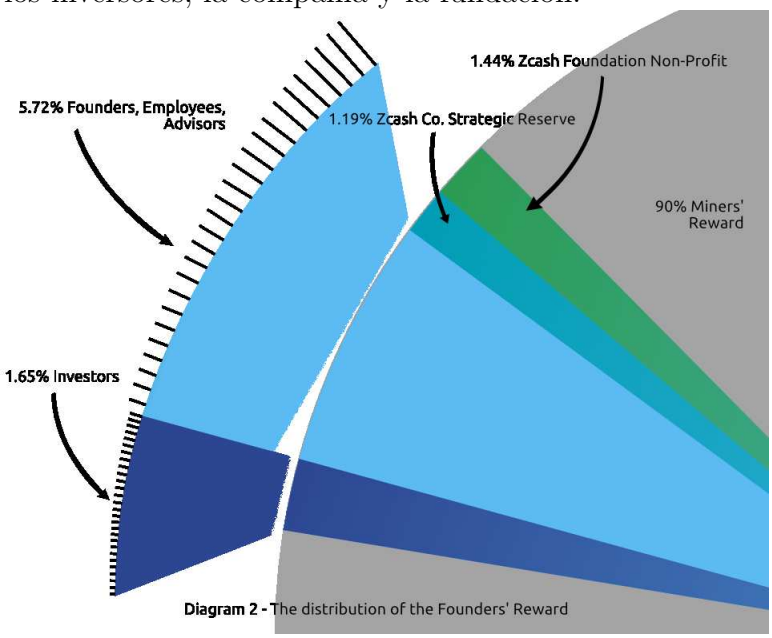
7.1.2. La Zcash Foundation

La Zcash Foundation es una organización sin ánimo de lucro dedicada a la creación de infraestructura de pago y privacidad por Internet para el bien común, que primariamente sirve a los usuarios del protocolo Zcash y su blockchain.

Tienen tres objetivos principales: la promoción de una comunidad sana y diversa, la administración del protocolo y el código fuente de Zcash como si fuera un bien público, y el avance de la ciencia relacionado con la privacidad en las criptomonedas y su divulgación.

[1] https://blog.z.cash/continued-funding-and-transparency/

Figura 7.1: Reparto de la recompensa de los fundadores entre los investigadores fundadores, los inversores, la compañía y la fundación.



Las becas

La Fundación abre cada seis meses un proceso por el cual se proponen proyectos a financiar, y un jurado los evalúa. En el proceso se tienen muy en cuenta las valoraciones de la comunidad.

A fecha de escritura de este trabajo, sólo se ha concedido una ronda de becas, las concedidas el último trimestre de 2017[2]. Suman un total de 127.000 dólares, y han sido destinadas a:

- Desarrollo: la creación de un cliente SPV por parte de Guarda Wallet (30.000 dólares), un nodo alternativo a zcashd escrito en Rust (15.000 dólares), una herramienta para realizar transacciones atómicas entre las blockchain de Zcash y Bitcoin (15.000 dólares), mejora del soporte multiplataforma (8.000 dólares) y mejora de la interfaz gráfica para escritorio (4.500 dólares).
- Ciencia: financiación de un análisis de la blockchain de Zcash por parte de CryptoLUX, los mismos que crearon Equihash (24.000 dólares).
- Mantenimiento: ayuda con los costes de mantenimiento de "Zcash on Tor" (9.000 dólares) y del explorador de bloques zcashnetwork.info (7.000 dólares).
- Divulgación: dos vídeos cortos sobre Zcash (9.500 dólares) y talleres educativos (5.000 dólares).

El proceso para elegir los siguientes proyectos a financiar ya ha empezado, pero aún es pronto para destacar alguno.

[2]https://z.cash.foundation/blog/grant-awards/

La Zcon

La Fundación Zcash está organizando la primera conferencia de Zcash, la Zcon0, que va a realizarse el 26 de junio de 2018. El evento va a tener una duración de tres días, y se va a realizar en Montreal, Canadá. Las charlas van a tratar sobre el presente y futuro de Zcash, sobre otras temáticas relacionadas con la privacidad y las criptomonedas, o sobre la comunidad y su gobernanza. La conferencia tendrá carácter anual y se retrasmitirá por streaming.

Las elecciones

La junta directiva inicial estaba compuesta de cuatro miembros que se han encargado de montar la infraestructura de la Fundación[3]. Escribieron los documentos fundacionales con la ayuda de algunos miembros de la Zcash Company, han llevado a cabo el proceso de asignación de las primeras becas y están organizando la primera Zcon.

Un año después de su creación, en el momento de escritura de este trabajo, se abre un proceso para elegir a dos nuevos miembros de la junta directiva, que se incorporarían junto a otros tres miembros antiguos.

El proceso de elección consiste en la constitución de un panel amplio de personas que representen a la comunidad, para que voten a los dos nuevos miembros del a junta. El panel (llamado *Community Governance Panel*) va a estar constituido por 200 miembros que tengan presencia pública o que hayan contribuido de alguna manera a la comunidad. Cualquier persona puede pedir estar en ese panel a través de un formulario web, pero deberán ser aceptados por el director ejecutivo y otras personas de la Fundación (se van a publicar todas las peticiones, tanto las aceptadas como las rechazadas, para maximizar la transparencia).

Los candidatos a miembros de la junta, por otro lado, publican sus propias nominaciones en el Github de la Fundación, con un texto de motivación, los objetivos que tienen con la Fundación y su currículum vitae. No es necesario formar parte del *Community Governance Panel* para poder presentarse de candidato. La comunidad podrá realizar sus valoraciones en la caja de comentarios de cada una de las candidaturas.

Van a ser elegidos por un sistema de voto del tipo voto de aprobación de satisfacción por parte del panel a través del software Helios[4], que permite voto secreto y criptográficamente seguro. Los resultados de la elección se presentarán el último día de la Zcon0.

7.2. Actualizaciones del protocolo

Como hemos visto, Zcash es un proyecto de software libre mantenido por una start-up que poco a poco está descentralizando su poder a medida que nuevos actores aparecen en la comunidad, como la Zcash Foundation.

En un post de octubre de 2016[5], Nathan y Zooko Wilcox, CTO y CEO de Zcash, respectivamente, dan a conocer su visión temprana de cómo creen que

[3]https://z.cash.foundation/blog/hello-world/
[4]https://vote.heliosvoting.org/
[5]https://blog.z.cash/consensual-currency/

debe desarrollarse la comunidad.

En dicho post especulan que a medida que Zcash crezca, sus usuarios van a tener diferentes necesidades, y no van a poder satisfacerse todas a gusto de todos. Es por eso que diferentes sub-comunidades van a elegir unas soluciones por encima de otras, creando variantes o forks de Zcash. Hacen una valoración positiva de que eso pueda ocurrir, y piden a la comunidad que aprenda tanto de Bitcoin como de Ethereum para encontrar un balance entre la estabilidad del primero y la agilidad del segundo.

En un post posterior[6], Zooko habla de los tipos de forks que se pueden dar en una comunidad según si generan un fork permanente en la blockchain y/o si generan un cisma en la comunidad. Se pregunta si es posible un fork permanente en la blockchain que no produzca un cisma en el que la comunidad, y enfatiza que le gustaría que los forks producidos en Zcash fueran de este estilo para que los usuarios pudieran beneficiarse de tecnologías diferentes, mientras que son tolerantes y cooperativos los unos con los otros, para el beneficio de todos.

7.2.1. ZIPs

Al igual que Bitcoin con los BIPs (Bitcoin Improvement Propoosals) y Ethereum con los ERCs (Ethereum Request for Comments), Zcash tiene su propio sistema para que la comunidad pueda proponer y discutir cambios en el protocolo, llamados ZIPs (Zcash Improvement Proposals)[7].

En el momento de escribir este trabajo existen seis ZIPs, cinco de ellos (143, 200, 201, 202 y 203) están planeados en desplegarse en la siguiente versión Overwinter, que se lanza en junio de 2018. El otro (el 243) está planeado que se despliegue con Sapling, para setiembre de ese mismo año.

7.2.2. Overwinter

El hard fork de Overwinter despliega un nuevo mecanismo de actualización de la red de Zcash. A través de ZIPs se propone:

- Definir mejor cuando una actualización de la red debe activarse y sus periodos, además de minimizar los riesgos.
- Un mecanismo de desconexión entre nodos que están en diferentes forks.
- un nuevo formato de transacción requerido para el nuevo mecanismo de actualización.
- Una nueva regla de consenso para añadir una fecha de caducidad a las transacciones tras la cual ya no pueden ser minadas, optimizando así la mempool de transacciones.

Overwinter es lo que posibilitará la coexistencia de distintas variantes y forks de Zcash de manera segura para sus usuarios, haciendo realidad la visión de Zooko y Nathan de tener diferentes tecnologías y una comunidad unida.

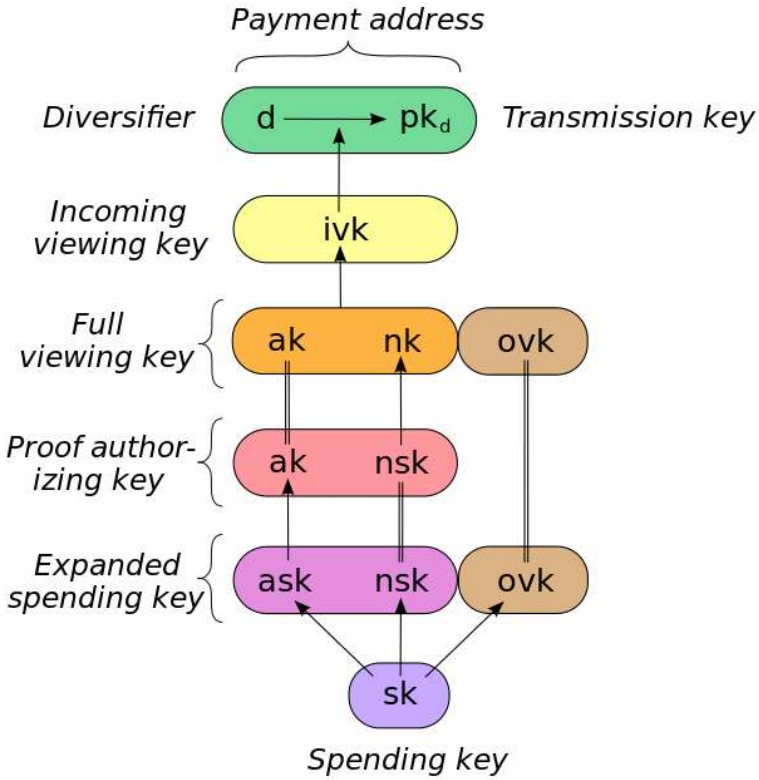
[6] https://blog.z.cash/future-friendly-fork/
[7] https://github.com/zcash/zips

7.2.3. Sapling

La actualización Sapling prevista para setiembre de 2018 conlleva grandes cambios en el protocolo de Zcash con el fin de optimizar la generación de zk-SNARKs. Usa una nueva curva llamada BLS12-381, un nuevo algoritmo de multi-exponenciación y un nuevo sistema que reemplaza a Pinocchio. Han diseñado una curva elíptica llamada Jubjub dentro de la curva BLS12-381, que peremite exponenciar en tiempo récord y reduce los requisitos de tiempo y memoria en un 80 % y un 98 %, respectivamente.

Si analizamos los cambios que comporta en el protocolo[8], vemos grandes cambios. Cambia la codificación de las direcciones blindadas, la definición de nota (ahora (d, pk_d, v, rcm)), se añade un nuevo formato de transacción con campos nuevos, se utiliza el campo reservado que había en los bloques, en lugar de usar JoinSplits, se usan *Spend Transfers* y *Output Transfers*, y muchos más cambios. Además se añaden nuevas claves a las que ya vimos en el Capítulo 4, como se puede observar en la Figura 7.2.

Figura 7.2: Sapling añade claves de visionado de gastos para solventar el problema que había con las claves de visionado anteriores, que no permitían ver si las notas habían sido ya gastadas.



[8]https://github.com/zcash/zips/blob/master/protocol/sapling.pdf

Conclusiones

En este análisis hemos estudiado el problema de la privacidad en las criptomonedas y la solución que ofrece Zcash.

Tal como vemos en el Capítulo 1, en Bitcoin se puede consultar la información completa de cada una de las transacciones realizadas por los usuarios de la red que han sido recogidas en la blockchain. Esto es necesario para verificar de manera descentralizada que cada una de las transacciones se ha añadido a la blockchain siguiendo las reglas consensuadas por la red, y que por lo tanto, que la blockchain sobre la que se está trabajando es válida. Para conseguir esa descentralización, por lo tanto, se pierde en gran medida la privacidad de sus usuarios.

En Bitcoin y en otras criptomonedas se han propuesto sistemas para mezclar las monedas de múltiples transacciones y que sea así más difícil rastrear su origen o a dónde se transfieren (evitando también su trazabilidad a través del grafo de transacciones). Eso es necesario para garantizar su fungibilidad, posibilitando que todas las monedas tengan el mismo valor independientemente de donde provengan, que es una propiedad deseable de todo sistema monetario.

Zcash es la primera implementación de una criptomoneda que usa pruebas de conocimiento nulo para realizar el mezclado de monedas. Eso permite proporcionar mayor privacidad, ya que en lugar de mezclar las monedas de varias transacciones que se están realizando en un determinado momento, como en propuestas anteriores (CoinJoin y demás), Zcash permite mezclar las monedas que se están transfiriendo con el resto de monedas de la red (con todas las que hayan sido transferidas anteriormente de manera blindada).

Para conseguirlo, Zcash construye esa capa de privacidad sobre el protocolo Bitcoin. La implementación de referencia es un fork de Bitcoin Core, al que se añaden nuevas operaciones para gestionar nuevos tipos de direcciones, claves, y transacciones, que permiten ver y/o enviar fondos de manera blindada. Por desgracia, tal como vemos en el Capítulo 2, el ecosistema aún no está muy maduro debido a la dificultad de implementación y los requisitos computacionales que son necesarios para realizar transacciones blindadas.

En el Capítulo 3 hemos estudiado las diferencias entre Bitcoin y Zcash en cuanto a estructura, tanto en transacciones como en cabeceras de bloque. A nivel de transacciones vemos que la principal diferencia es la introducción de una lista de JoinSplits, que a nivel conceptual, son transacciones de valor blindado. Un JoinSplit consume y genera valor, desvinculando el valor de entrada con el de salida, pero demostrando con una prueba de conocimiento nulo que se cumplen las reglas consensuadas en el protocolo. Las pruebas son verificables por cualquier minero o usuario de la red, y por lo tanto todo el mundo puede determinar si una blockchain es válida o no. A nivel de cabecera de bloque vemos que respecto a Bitcoin, además de los hashes al bloque anterior y al árbol Merkle de transacciones, Zcash añade un campo reservado para un hash más. El campo no ha sido usado hasta ahora, pero se pretende usar a partir de la versión 2.0 del protocolo. También, debido a que Zcash usa un algoritmo de prueba de trabajo diferente al de Bitcoin, la cabecera de bloque añade un campo extra necesario para codificar la solución a éste.

En el Capítulo 4 definimos con más claridad qué tipo de claves se usan para

ver y transmitir valor blindado, cómo éste es representado (en forma de notas), cómo es trasferido (usando como ejemplo una transferencia entre Alice y Bob) y finalmente cómo se cifran los datos que no son públicos, para que los receptores de transacciones blindadas puedan ser notificados, preservando al mismo tiempo la confidencialidad de los pagos.

En el Capítulo 5 describimos los tipos de pruebas de conocimiento nulo, y explicamos la necesidad de que sean del tipo no-interactivo para aplicaciones blockchain. Describimos ligeramente los elementos que conforman las pruebas de conocimiento nulo zk-SNARK usadas en Zcash por su reducido tamaño y facilidad de verificación, y describimos la ceremonia realizada para generar los parámetros públicos necesarios para el cómputo de esas pruebas no-interactivas de tamaño reducido.

En el Capítulo 6 también analizamos el algoritmo de prueba de trabajo utilizado en Zcash, Equihash, y el porqué se le consideraba resistente a la creación de circuitos integrados diseñados específicamente para minarlo. Sus altas exigencias de memoria hacían pensar a sus creadores que sería muy complicada la fabricación de ASICs para minar Equihash, pero las primeras en aparecer están anunciadas para junio de 2018, y tanto la Compañía como la Fundación Zcash se han pronunciado al respecto. Además de esto, también analizamos el protocolo Stratum, usado en las pools de minería de multiples monedas y sus adaptaciones para el minado de Zcash.

Finalmente, en el Capítulo 7 vemos qué stakeholders existen dentro de la comunidad. La Compañía Zcash es la que posee el control sobre los repositorios de Github y también es quien especifica el protocolo. Sus fundadores reciben el 10 % de la base monetaria total de Zcash durante los primeros 4 años. La Fundación Zcash es una organización, financiada por parte de los fondos destinados a los fundadores, que pretende descentralizar el poder que tiene la Compañía sobre el resto de la comunidad. Además vemos qué actualizaciones del protocolo están planeadas en Zcash (Overwinter para junio y Sapling para septiembre), y qué mecanismos tiene la comunidad para proponer mejoras al protocolo, los Zcash Improvement Proposals (ZIPs).

Zcash es de las tecnologías más interesantes a estudiar dentro del ámbito de las criptomonedas por su interesante propuesta para preservar la privacidad a la vez que hace cumplir las reglas del protocolo. En este análisis hemos señalado sus características principales y diferenciadoras con respecto a Bitcoin, y esperamos que sirva como una referencia introductoria para su estudio.

Bibliografía

[1] A. Back et al. Hashcash-a denial of service counter-measure. 2002.

[2] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 287–304. IEEE, 2015.

[3] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, pages 781–796, 2014.

[4] A. Biryukov and D. Khovratovich. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. *Ledger*, 2:1–30, 2017.

[5] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 103–112. ACM, 1988.

[6] S. Bowe, A. Gabizon, and M. D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. Technical report.

[7] S. Bowe, A. Gabizon, and I. Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. 2017.

[8] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[9] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.

[10] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 127–140, New York, NY, USA, 2013. ACM.

[11] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[12] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[13] A. Pfitzmann and M. Hansen. Anonymity, unlinkability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology. 2005.

[14] J. M. Pollard. A monte carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975.

[15] J. Quesnelle. On the linkability of zcash transactions. *CoRR*, abs/1712.01210, 2017.

[16] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International workshop on fast software encryption*, pages 371–388. Springer, 2004.

[17] T. Ruffing and P. Moreno-Sanchez. Valueshuffle: Mixing confidential transactions for comprehensive transaction privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 133–154. Springer, 2017.

[18] T. Ruffing, P. Moreno-Sanchez, and A. Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *European Symposium on Research in Computer Security*, pages 345–364. Springer, 2014.

[19] T. Ruffing, P. Moreno-Sanchez, and A. Kate. P2p mixing and unlinkable bitcoin transactions. *IACR Cryptology ePrint Archive*, 2016:824, 2016.

[20] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.

[21] M. B. Taylor. Bitcoin and the age of bespoke silicon. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, page 16. IEEE Press, 2013.

[22] D. Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.

[23] H. Wu and F. Wang. A survey of noninteractive zero knowledge proof system and its applications. *The Scientific World Journal*, 2014, 2014.