Elogios a 'Mastering Bitcoin'

> "Cuando hablo de bitcoin al público en general, a veces me preguntan '¿pero cómo funciona realmente?' Ahora tengo una gran respuesta para esa pregunta, porque cualquiera que lea *Mastering Bitcoin* tendrá un profundo conocimiento de cómo funciona y estará preparado para escribir la siguiente generación de increíbles aplicaciones para criptodivisas."
>
> — Gavin Andresen, Director Científico de la Fundación Bitcoin

> "Bitcoin y las tecnologías blockchain se están convirtiendo en los pilares fundamentales de construcción para el Internet de próxima generación. Los mejores y más brillantes de Silicon Valley ya están trabajando en ello. El libro de Andreas le ayudará a unirse a la revolución del software en el mundo de las finanzas."
>
> — Naval Ravikant, Co-fundador AngelList

> "*Mastering Bitcoin* es la mejor referencia técnica disponible hoy sobre bitcoin. Y visto en retrospectiva, bitcoin es probablemente la tecnología más importante de esta década. Como tal, este libro es absolutamente imprescindible para cualquier desarrollador, especialmente aquellos interesados en la creación de aplicaciones con el protocolo bitcoin. Muy recomendable."
>
> — Balaji S. Srinivasan (@balajis), Socio General, Andreessen Horowitz

> "La invención de la Bitcoin Blockchain representa una plataforma completamente nueva para trabajar, una que habilitará un ecosistema tan amplio y diverso como el propio Internet. Como uno de los líderes de opinión por excelencia, Andreas Antonopoulos es la elección perfecta para escribir este libro."
>
> — Roger Ver, Emprendedor de Bitcoin e Inversor

Indice

Prólogo

Escribiendo el libro de Bitcoin

La primera vez que tropecé con bitcoin fue a mediados del año 2011. Mi reacción inmediata fue más o menos "¡Pfft! ¡Dinero para friquis!" y lo ignoré durante unos seis meses, sin alcanzar a comprender su importancia. Me consuela haber comprobado que esta reacción se produce en muchas de las personas más inteligentes que conozco. Cuando me encontré por segunda vez con una referencia a bitcoin en una discusión de una lista de correo, decidí leer el libro blanco escrito por Satoshi Nakamoto. Ahí encontré la fuente de referencia que me sirvió para entenderlo todo. Aún recuerdo el momento en que terminé de leer aquellas nueve páginas. Fue cuando me di cuenta de que bitcoin no es simplemente una moneda digital, sino una red de confianza que se extiende mucho más allá del restringido ámbito de las monedas. La comprensión de que "esto no es dinero, esto es una red de confianza descentralizada" me inició en un un viaje de cuatro meses en el que devoré cada trozo de información sobre bitcoin que pude encontrar. Llegué a estar obsesionado y fascinado, dedicando 12 o más horas cada día pegado a la pantalla, leyendo, escribiendo, programando y aprendiendo todo lo que podía. Volví a la realidad habiendo adelgazado más de 10 kilos por la falta de consistencia en las comidas, decidido a dedicarme a trabajar para bitcoin.

Pasados dos años y después de crear varias empresas de productos y servicios de bitcoin decidí que había llegado la hora de escribir mi primer libro. Mi interés por Bitcoin me consumía el pensamiento y mi creatividad fue frenética; era la tecnología más excitante desde la aparición de Internet. Era hora de compartir mi pasión sobre esta impresionante tecnología con todo el mundo.

Audiencia Prevista

Este libro está dirigido principalmente hacia programadores. Si sabes programar, este libro te enseñará cómo funcionan las monedas criptográficas, cómo usarlas, y cómo desarrollar software que trabaje con ellas. Los primeros capítulos sirven también como una introducción a bitcoin para no programadores que quieran entender el funcionamiento interno de bitcoin y las monedas criptográficas en profundidad.

Convenciones Usadas en este Libro

Este libro usa las siguientes convenciones tipográficas:

Cursiva

Indica términos nuevos, URLs, direcciones de email, nombres de archivo y extensiones de archivo.

Ancho constante

Usado para listados de programas, así como dentro de párrafos para referirse a elementos de un programa como nombres de variables o funciones, bases de datos, tipos de datos, variables de entorno, sentencias y palabras clave.

`Ancho constante con negrita`
: Muestra comandos u otro texto que debe ser tecleado literalmente por el usuario.

Ancho constante con cursiva
: Muestra texto que debe ser reemplazado por valores provistos por el usuario o valores determinados por contexto.

> **TIP** Este icono significa un consejo, sugerencia o nota general.

> **WARNING** Este icono indica una advertencia o cuidado.

Ejemplos de Código

Los ejemplos están desarrollados en Python, C++, y usando la línea de comandos de un sistema operativo tipo Unix como Linux o Mac OS X. Todos los fragmentos de código están disponibles en repositorio GitHub en el subdirectorio *code* del repo principal. Haga un fork al código del libro, pruebe los programas de ejemplo, o envíe correcciones a través de GitHub.

Todos los fragmentos de código pueden ejecutarse en la mayoría de los sistemas operativos con una instalación mínima de compiladores e intérpretes para los lenguajes correspondientes. Cuando sea necesario, proporcionamos instrucciones de instalación básicas y ejemplos paso a paso.

Algunos de los fragmentos de código y sus salidas se han reformateado en la impresión. En todos estos casos, las líneas se han dividido por un carácter de barra invertida (\), seguido de un carácter de nueva línea. Cuando se quiera transcribir los ejemplos, elimine esos dos caracteres y una las líneas de nuevo para ver los resultados idénticos a como se muestran en el ejemplo.

Siempre que sea posible, los fragmentos de código utilizan valores y cálculos reales, por lo que se puede construir cada uno de los ejemplos y obtener los mismos resultados en cualquier código que escriba para calcular los mismos valores. Por ejemplo, las claves privadas, las claves públicas correspondientes y las direcciones son reales. Las transacciones de la muestra, los bloques y las referencias a la cadena de bloques se han introducido en la cadena de bloques real de bitcoin y forman parte del libro contable, por lo que puede verificarse en cualquier sistema bitcoin.

Agradecimientos

Este libro representa el esfuerzo y las contribuciones de muchas personas. Agradezco toda la ayuda que he recibido de mis amigos, colegas e incluso extraños quienes se han unido a mi en este esfuerzo de escribir el libro técnico definitivo sobre monedas criptográficas y bitcoin.

Resulta imposible distinguir entre la tecnología de bitcoin y la comunidad de bitcoin, siendo este libro un producto de esa comunidad tanto como lo es un libro sobre la tecnología. De principio a fin he recibido de la comunidad de bitcoin en su totalidad el entusiasmo, el ánimo, la recompensa y apoyo que me han permitido la realización de este libro. Mas que ninguna otra cosa, este libro me ha incluido

en un maravillosa comunidad durante dos años y no puedo daros suficientes gracias por aceptarme como miembro. Hay demasiadas personas como para mencionarlas a todas por nombre - la gente que he conocido durante conferencias, eventos, quedadas, cenando pizza y otras pequeñas tertulias, así como aquellos que se han comunicado conmigo a través de Twitter, en reddit, bitcointalk.org y en GitHub y han dado algo que ha contribuido a este libro. Cada idea, analogía, pregunta, respuesta y explicación que encontrarás en este libro ha sido en algún momento inspirado, contrastado o mejorado por medio de mis interacciones con la comunidad. Gracias a todos por vuestro apoyo; sin vosotros este libro existiría. Estaré agradecido para siempre.

El camino hacia la autoría tuvo sus comienzos, por supuesto, mucho antes de este primer libro. Mi idioma materno es griego y es en el que se me impartió escuela, por lo que durante mi primer año de universidad tuve que atender un curso para mejorar mi inglés. Debo dar las gracias a mi profesora de escritura Diana Kordas quién me ayudó a establecer confianza en mis habilidades. Más tarde desarrollé la capacidad de escribir material técnico acerca de centros de procesamiento de datos para la revista *Network World*. Agradecimientos a John Dix y John Gallant, me dieron mi primer trabajo, escribiendo una columna para *Network World* y a mi editor Michael Cooney y a mi colega Johna Till Johnson que revisaron y editaron mi trabajo para su publicación. El escribir 500 palabras a la semana durante cuatro años me procuró la experiencia suficiente como para plantearme hacerme autor. Gracias a Jean de Vera por animarme a ello y por insistir en creer que yo tendría un libro que escribir.

Gracias también a aquellos que me apoyaron cuando propuse este libro a la editorial O´Reilly, dando referencias y revisando la propuesta. En concreto a John Gallant, Gregory Ness, Richard Stiennon, Joel Snyder, Adam B. Levine, Sandra Gittlen, John Dix, Johna Till Johnson, Roger Ver y Jon Matonis. En especial agradecimientos a Richard Kagan y Tymon Mattoszko quienes revisaron las primeras versiones de la propuesta y a Matthew Owain Taylor quién la editó.

Gracias a Cricket Liu, autor del título de O'Reilly *DNS y BIND*, quien me presentó a O'Reilly. Gracias también a Michael Loukides y Allyson MacDonald de O'Reilly, quienes trabajaron durante meses para ayudar a hacer posible este libro. Allyson fue especialmente paciente cuando no se cumplían los plazos y los entregables se retrasaban a medida que la vida se interponía en nuestra planificación.

Los primeros borradores de los primeros capítulos fueron los más difíciles, ya que Bitcoin es un tema difícil de desenmarañar. Cada vez que empezaba a tirar de uno de los hilos de la tecnología bitcoin, tenía que echarlo atrás completamente. En repetidas ocasiones me quedé atascado y un poco desanimado mientras luchaba para que el tema fuera fácil de entender y pudiera crearse una narrativa fluida en torno a un tema técnico tan denso. Finalmente, me decidí a contar la historia de bitcoin a través de las historias de las personas que utilizan bitcoin y todo el libro se convirtió en algo mucho más fácil de escribir. Le debo gracias a mi amigo y mentor, Richard Kagan, quien me ayudó a desentrañar la historia y superar los momentos de bloqueo, y a Pamela Morgan, que revisó los primeros borradores de cada capítulo y hacía esas preguntas difíciles que servían para mejorarlos. También, gracias a los desarrolladores del grupo de San Francisco Bitcoin Developers Meetup y a Taariq Lewis, co-fundador del grupo, por ayudar a poner a prueba el material desde el principio.

Durante el desarrollo del libro, dejé los primeros borradores disponibles en GitHub e invité a que el público los comentara. Me llegaron más de un centenar de comentarios, sugerencias, correcciones y aportaciones. Reconozco y agradezco explícitamente todas esas contribuciones en Borrador de Entrega

Temprana (Contribuciones de GitHub). Un agradecimiento especial a Minh T. Nguyen, quien se ofreció como voluntario para gestionar las contribuciones de GitHub y que asimismo ha añadido otras valiosas contribuciones. También a Andrew Naugler, gracias por su diseño infográfico.

Una vez redactado, el libro pasó por varias rondas de revisión técnica. Gracias a Cricket Liu y Lorne Lantz por su minuciosa revisión, comentarios y apoyo.

Varios desarrolladores bitcoin aportaron ejemplos de código, opiniones, comentarios, y ánimo. Gracias a Amir Taaki y Eric Voskuil por sus fragmentos de código y por sus muchos y valiosos comentarios; Vitalik Buterin y Richard Kiss por su ayuda en las matemáticas de curva elíptica y sus contribuciones de código; Gavin Andresen por las correcciones, comentarios y estímulo; Michalis Kargakis por los comentarios, aportes y valoración crítica de btcd; y Robin Inge por la presentación de erratas para la mejora de la segunda impresión.

Mi amor por los libros y la palabra se lo debo a mi madre Teresa quién me crió en una casa donde los libros cubrían las paredes. Mi madre también me compró mi primer ordenador en 1982 aun cuando se confesó como tecnófoba. Mi padre Menelaos, un ingeniero civil que acaba de publicar su primer a los 80 años de edad, me enseñó el pensamiento lógico y analítico y el amor por la ciencia y la ingeniería.

Gracias a todos ustedes por ayudarme a lo largo de esta travesía.

Borrador de Entrega Temprana (Contribuciones de GitHub)

Muchos colaboradores han ofrecido comentarios, correcciones y ampliaciones al borrador de entrega temprana en GitHub. Gracias a todos por sus contribuciones a este libro. A continuación se presenta una lista de colaboradores notables en GitHub, incluyendo su GitHub ID entre paréntesis:

- Minh T. Nguyen, editor de contribuciones en GitHub (enderminh)
- Ed Eykholt (edeykholt)
- Michalis Kargakis (kargakis)
- Erik Wahlström (erikwam)
- Richard Kiss (richardkiss)
- Eric Winchell (winchell)
- Sergej Kotliar (ziggamon)
- Nagaraj Hubli (nagarajhubli)
- ethers
- Alex Waters (alexwaters)
- Mihail Russu (MihailRussu)
- Ish Ot Jr. (ishotjr)
- James Addison (jayaddison)
- Nekomata (nekomata-3)

- Simon de la Rouviere (simondlr)
- Chapman Shoop (belovachap)
- Holger Schinzel (schinzelh)
- effectsToCause (vericoin)
- Stephan Oeste (Emzy)
- Joe Bauers (joebauers)
- Jason Bisterfeldt (jbisterfeldt)
- Ed Leafe (EdLeafe)

Edición Abierta

Atribución

Mastering Bitcoin por Andreas M. Antonopoulos LLC https://bitcoinbook.info

Traducción

Si usted está leyendo este libro en un idioma distinto del inglés, se ha traducido por voluntarios. Las siguientes personas contribuyeron a esta traducción:

0b10n3 (Ovidio Vazquez), Jzenemig (Jose Gimenez), LeoWandersleb (Leo Wandersleb), SmeagolGollum (Smeagol Gollum), Tremz (Sergy R. Nava), Zukaza (Andres J. Gonzalez), alexladerman (Alex Laderman), andreselozadam (Andres Eloy Lozada Molinett), dhersan (Daniel Hernandez), diegopher (diego fernandez), elepolan (Elena Prieto), estebansaa (Esteban Saa), evamorano (Eva Maria Morano Gonzalez), fergc (fer), fjrojasgarcia (Javier Rojas), fjsanchezgil (Francisco Javier Sánchez Gil), francisguarderas (Francisco Guarderas), franckuestein (franckuestein), jabravo, javiromero (Javier Romero), jmiehau (Jorge Mielgo Haurie), jmorocho (Jonathan Morocho), josefelip (Jose Felip), josepimpo (Jose Antonio Rodriguez), jujumax (Juan Garavaglia), kennethfolk (Kenneth Folk), litri5 (carlos martin-moreno), marshalltlong (Marshall Long), minskburrly (Joel Cannon), olivermasiosare (Oliver Masiosare), pilaf, tasmanoide, traductor10, vgpastor (Victor García)

Glosario Rápido

Este glosario rápido contiene muchos de los términos relacionados con bitcoin. Estos términos se usan en todo el libro. Agregue una marca para una rápida referencia.

dirección

Una dirección de Bitcoin se parece a 1DSrfJdB2AnWaFNgSbv3MZC2m74996JafV. Consiste en una cadena de letras y números y se inicia con un "1" (número uno). De la misma forma que solicita que le envien un correo a su dirección de correo, también puede solicitar que le envien bitcoin a su dirección de bitcoin.

bip

Propuestas de Mejora de Bitcoin: Un conjunto de propuestas que los miembros de la comunidad bitcoin han enviado para mejorar bitcoin. Por ejemplo, BIP0021 es una propuesta para mejorar el esquema de Identificador de Recursos Uniforme (URI) de bitcoin.

bitcoin

El nombre de la unidad monetaria (la moneda), la red, y el software.

bloque

Una agrupación de transacciones, marcadas con un sello de tiempo, y una huella digital del bloque anterior. Se obtiene el hash de la cabecera de bloque para producir una prueba de trabajo, validando así las transacciones. Los bloques válidos se añaden a la cadena de bloques principal por consenso de la red.

cadena de bloques

Una lista de bloques validados, cada uno conectado con su precedente hasta el bloque génesis.

confirmaciones

Una vez que la transacción es incluida en un bloque, tiene una confirmación. Tan pronto como -otro - bloque se mina en la misma cadena, la transacción tiene dos confirmaciones, y así sucesivamente. Se considera suficiente prueba de que la transacción no será revocada con seis o más confirmaciones.

dificultad

Una configuración que aplica a toda la red y que controla cuánta capacidad de computación se requiere para producir una prueba de trabajo.

objetivo de dificultad

La dificultad a la que toda la computación de la red encontrará bloques aproximadamente cada 10 minutos.

reajuste de dificultad

Recálculo de la dificultad que aplica a toda la red y que ocurre cada 2.106 bloques en base a la

potencia de hashing de los 2.106 bloques anteriores.

comisiones

El emisor de una transacción a menudo incluye una comisión para que la red procese la transacción solicitada. La mayoría de las transacciones requieren una tasa mínima de 0,5 mBTC.

hash

Una huella digital de alguna entrada binaria.

bloque génesis

El primer bloque de la cadena de bloques, utilizado para inicializar la criptodivisa.

minero

Un nodo de la red que encuentra pruebas de trabajo válidas para los nuevos bloques, mediante la ejecución reiterada de hashes.

red

Una red peer-to-peer (red entre iguales) que propaga las transacciones y bloques a cada nodo bitcoin de la red.

Prueba De Trabajo

Una porción de datos que solo es posible obtener tras haber ejecutado una notable cantidad de cómputo. En bitcoin, los mineros deben encontrar una solución numérica al algoritmo SHA256 que satisfaga el objetivo de dificultad, que aplica a toda la red.

recompensa

Una cantidad incluida en cada nuevo bloque como recompensa de la red al minero que encontró la solución a la prueba de trabajo. Actualmente es de 25BTC por bloque.

clave secreta (alias clave privada)

El número secreto que desbloquea los bitcoins enviados a la dirección correspondiente. Una clave secreta se asemeja a 5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibVPxh.

transacción

En términos simples, una transferencia de bitcoins de una dirección a otra. Concretamente, una transacción es una estructura de datos firmada que expresa una transferencia de valor. Las transacciones se transmiten a través de la red bitcoin, son recogidas por los mineros e incluidas en bloques que se mantienen permanentes en la blockchain.

cartera/monedero

El software que guarda todas tus direcciones bitcoin y claves privadas. Úsala para enviar, recibir y guardar tus bitcoin.

Introducción

¿Qué es Bitcoin?

Bitcoin es un conjunto de conceptos y tecnologías que conforman un ecosistema de dinero digital. El almacenamiento y transmisión de valor entre los participantes de la red bitcoin se consigue mediante la utilización de las unidades monetarias llamadas bitcoins. Los usuarios de bitcoin se comunican entre ellos usando el protocolo bitcoin, principalmente a través de Internet, aunque también se pueden utilizar otras redes de transporte. La pila de protocolos bitcoin, disponible como software open source, puede ejecutarse sobre una amplia variedad de dispositivos, incluyendo laptops y smartphones, lo que hace que la tecnología sea fácilmente accesible.

Los usuarios pueden transferir bitcoins a través de la red para hacer prácticamente cualquier cosa realizable con monedas convencionales, incluyendo comprar y vender bienes, enviar dinero a personas y organizaciones, o extender créditos. Los bitcoins pueden comprarse, venderse e intercambiarse por otras monedas en casas de cambio especializadas. En cierta forma bitcoin es la forma de dinero perfecta para Internet, ya que es rápido, seguro y carente de fronteras.

A diferencia de las monedas tradicionales, los bitcoins son completamente virtuales. No existen monedas físicas y en sentido estricto, ni siquiera existen monedas digitales. Las monedas están implícitas en transacciones que mueven valor de un remitente a un destinatario. Los usuarios de bitcoin poseen claves que les permiten demostrar la propiedad de las transacciones en la red bitcoin, otorgando acceso a gastar su valor transfiriéndolo a un nuevo destinatario. Esas claves están normalmente almacenadas en una cartera digital (en inglés, "wallet") en el computador de cada usuario. La posesión de la clave que libera una transacción es el único prerrequisito para gastar bitcoins, poniendo completo control en las manos de cada usuario.

Bitcoin es un sistema entre pares (peer-to-peer) distribuido. Como tal no existe ningún servidor o punto de control "central". Los bitcoins se crean mediante un proceso llamado "minería," que se basa en una competencia por encontrar soluciones a un problema matemático a la vez que se procesan transacciones bitcoin. Cualquier participante de la red bitcoin (léase, cualquier persona utilizando un dispositivo con la pila de protocolos bitcoin completa) puede operar como minero, utilizando el poder de cómputo de su computador para verificar y registrar transacciones. Cada 10 minutos en promedio alguien consigue validar las transacciones de los últimos 10 minutos y es recompensado con nuevos bitcoins. En esencia, la minería de bitcoins descentraliza la función de emisión de moneda y la autorización de un banco central, y reemplaza la necesidad de un banco central con esta competencia global.

El protocolo bitcoin incluye algoritmos que regulan la función de minería en toda la red. La dificultad de la tarea de procesamiento que los mineros deben ejecutar—para registrar con éxito un bloque de transacciones para la red bitcoin—se ajusta dinámicamente de forma que, en promedio, alguien será exitoso cada 10 minutos sin importar cuántos mineros (y CPUs) hayan trabajado en la tarea en cada momento. Cada cuatro años, el protocolo también reduce a la mitad la tasa a la que se crean nuevos bitcoins, asegurando se seguirán creando bitcoins hasta un valor límite de 21 millones de monedas. El

resultado es que el número de bitcoins en circulación sigue de cerca una curva fácilmente predecible que alcanza los 21 millones en el año 2140. Debido a la decreciente tasa de emisión, bitcoin es deflacionario en el largo plazo. Además bitcoin no puede ser inflado a través de la "impresión" de nuevo dinero por encima de la tasa de emisión esperada.

Tras bambalinas, bitcoin es también el nombre del protocolo, una red y una innovación de computación distribuida. La moneda bitcoin es tan solo la primera aplicación de esta invención. Como desarrollador veo a bitcoin como algo similar a la Internet del dinero, una red para propagar valor y asegurar la propiedad de activos digitales via computación distribuida. Bitcoin es mucho más que lo que inicialmente aparenta.

En este capítulo comenzaremos por explicar algunos de los principales conceptos y términos, obtener el software necesario, y usar bitcoin para transacciones simples. En próximos capítulos empezaremos a desenvolver las capas de tecnología que hacen a bitcoin posible y examinaremos el funcionamiento interno de la red y protocolo bitcoin.

Monedas Digitales Antes de Bitcoin

El surgimiento de dinero digital viable se encuentra estrechamente relacionado a desarrollos en criptografía. Esto no es una sorpresa cuando uno considera los desafíos fundamentales involucrados en utilizar bits para representar valor intercambiable por bienes y servicios. Dos preguntas básicas para cualquiera que acepte dinero digital son:

1. ¿Puedo confiar en que el dinero es auténtico y no una falsificación?
2. ¿Puedo estar seguro de que nadie más aparte de mí puede alegar que este dinero le pertenece? (También conocido como el problema del "doble gasto" o "double-spend".)

Los emisores de dinero en papel luchan constantemente contra el problema de la falsificación utilizando tecnologías de papel y de impresión cada vez más sofisticadas. El dinero físico resuelve el problema del doble gasto fácilmente ya que el mismo billete no puede estar en dos lugares a la vez. Por supuesto, el dinero convencional también se almacena y se transmite digitalmente. En estos casos los problemas de la falsificación y el doble gasto se resuelven autorizando todas las transacciones electrónicas a través de autoridades centrales que tienen una visión global de toda la moneda en circulación. Para el dinero digital, que no puede aprovecharse de tintas esotéricas o bandas holográficas, la criptografía proporciona la base para confiar en la legitimidad del acceso al valor que pueda ejercer un usuario. Específicamente las firmas criptográficas digitales permiten al usuario firmar un activo o una transacción digital demostrando su propiedad sobre ese activo. Con la arquitectura adecuada las firmas digitales también pueden usarse para resolver el problema del doble gasto.

Cuando la criptografía comenzaba a estar más ampliamente disponible y entendida a finales de la década de 1980, muchos investigadores comenzaron a intentar utilizar la criptografía para construir monedas digitales. Estos primeros proyectos de monedas digitales emitían dinero digital, generalmente respaldado por una moneda nacional o un metal precioso como el oro.

Aunque estas primeras monedas digitales funcionaban, eran centralizadas, y como resultado eran fáciles de atacar por gobiernos y hackers. Las primeras monedas digitales utilizaban autoridades centrales para liquidar todas las transacciones en intervalos regulares, tal como lo hace el actual sistema bancario. Desafortunadamente en la mayoría de los casos estas monedas digitales emergentes fueron el objetivo de gobiernos preocupados y en última instancia litigadas hasta desaparecer. Algunas se desplomaron de forma espectacular al quebrar abruptamente sus empresas padre. Para resultar robusta frente a la intervención antagonística, ya fuera de gobiernos legítimos o de elementos criminales, era necesaria una moneda digital descentralizada sin un punto único de ataque. Bitcoin es ese sistema, completamente descentralizado por diseño, y libre de cualquier autoridad central o punto de control que pueda ser atacado o corrompido.

Bitcoin representa la culminación de décadas de investigación en criptografía y sistemas distribuidos e incluye cuatro innovaciones clave reunidas en una combinación única y potente. Bitcoin consiste de:

- Una red entre pares distribuida (el protocolo bitcoin)
- Un libro contable público (la cadena de bloques, o "blockchain")
- Un sistema distribuido, matemático y determinísitico de emisión de moneda (minería distribuida)
- Un sistema descentralizado de verificación de transacciones (script de transacciones)

Historia de Bitcoin

Bitcoin fue inventado en 2008 con la publicación de un paper titulado "Bitcoin: A Peer-to-Peer Electronic Cash System," escrito bajo el apodo de Satoshi Nakamoto. Nakamoto combinó varias invenciones previas tales como b-money y HashCash para crear un sistema de efectivo electrónico completamente descentralizado el cual no depende de una autoridad central para su emisión o la liquidación y validación de transacciones. La innovación clave fue el uso de un sistema de computación distribuida (llamado un algoritmo de "prueba de trabajo") para llevar a cabo una "elección" global cada 10 minutos, permitiéndole a la red descentralizada llegar a un *consenso* acerca del estado de transacciones. Esto resuelve de forma elegante el problema del doble gasto por el cual una misma unidad de moneda puede gastarse dos veces. Hasta entonces el problema del doble gasto era una limitación de las monedas digitales, que se abordaba mediante la verificación de todas las transacciones a través de una autoridad central.

La red bitcoin fue iniciada en 2009, basada en una implementación de referencia publicada por Nakamoto, y que ha sido modificada por muchos otros programadores desde entonces. La computación distribuida que provee seguridad y resistencia a bitcoin ha crecido exponencialmente, y actualmente excede el poder de procesamiento combinado de los supercomputadores más avanzados del mundo. El valor de mercado total de bitcoin se estima entre 5 y 10 mil millones de dólares estadounidenses, dependiendo de la tasa de cambio de bitcoins a dólares. La mayor transacción procesada hasta el momento por la red fue de 150 millones de dólares, transmitidos instáneamente y procesados sin tarifas.

Satoshi Nakamoto se retiró del público en abril de 2011, legando la responsabilidad de desarrollar el código y la red a un grupo creciente de voluntarios. La identidad de la persona o personas detrás de bitcoin es aún desconocida. Sin embargo, ni Satoshi Nakamoto ni nadie más posee control sobre el sistema bitcoin, el cual opera basado en principios matemáticos completamente transparentes. La invención en sí misma es revolucionaria y ya ha derivado en una nueva ciencia en los campos de computación distribuida, economía y econometría.

Una Solución a un Problema de Computación Distribuida

La invención de Satoshi Nakamoto es también una solución a un problema previamente sin solución en computación distribuida, conocido como el "Problema de los Generales Bizantinos." Brevemente, el problema consiste en tratar de llegar a un consenso al respecto de un plan de acción intercambiando información a través de una red poco fiable y potencialmente comprometida. La solución de Satoshi Nakamoto, que utiliza el concepto de prueba de trabajo para alcanzar un consenso sin requerir confianza en una autoridad central, representa un avance en computación distribuida y posee amplias aplicaciones más allá de las monedas. Puede ser utilizada para alcanzar consenso en redes distribuidas para probar la legitimidad de elecciones, loterías, registros de activos, autorizaciones bajo notario digitales, y más.

Usos de Bitcoin, Usuarios y Sus Historias

Bitcoin es una tecnología, pero expresa dinero que es fundamentalmente un lenguaje para intercambiar valor entre personas. Echemos un vistazo a las personas que usan bitcoin y algunos de los casos de uso más comunes de la moneda y el protocolo a través de sus historias. Reutilizaremos estas historias a lo largo del libro para ilustrar los usos del dinero digital en la vida real y cómo son posibles gracias a las varias tecnologías que conforman bitcoin.

Venta de artículos de bajo valor en Norteamérica

Alice vive en la costa del Norte de California. Escuchó acerca de bitcoin a través de sus amigos tecnófilos y quiere comenzar a usarlo. Seguiremos su historia a medida que aprende sobre bitcoin, adquiere algunos, y luego gasta parte de sus bitcoins para comprar una taza de café en el Café de Bob en Palo Alto. Esta historia nos presentará el software, las casas de cambio y las transacciones básicas desde el punto de vista de un consumidor.

Venta de artículos de alto valor en Norteamérica

Carol es la dueña de una galería de arte en San Francisco. Vende pinturas costosas a cambio de bitcoins. Esta historia presentará los riesgos de un ataque de consenso del "51%" para vendedores de artículos de valor elevado.

Tercerización de servicios al extranjero

Bob, el dueño del café en Palo Alto, está construyendo un nuevo sitio web. Ha contratado a un desarrollador, Gopesh, que vive en Bangalore, India. Gopesh ha aceptado ser remunerado en bitcoins. Esta historia examinará el uso de bitcoin para tercerización, contrato de servicios y pagos

internacionales.

Donaciones de caridad

Eugenia es la directora de una organización benéfica para niños en las Filipinas. Recientemente ha descubierto bitcoin y quiere usarlo para alcanzar un grupo completamente nuevo de donantes nacionales y extranjeros para financiar su organización. También está investigando formas de usar bitcoin para distribuir fondos rápidamente a áreas en necesidad. Esta historia mostrará el uso de bitcoin para recaudación de fondos a lo largo de diversas monedas y fronteras y el uso de un libro contable abierto para mejorar la transparencia de organizaciones de caridad.

Importación/exportación

Mohammed es un importador de electrónica en Dubai. Está intentando usar bitcoin para importar electrónica de los EEUU y China hacia los Emiratos Árabes Unidos, acelerando el proceso de pagos para importaciones. Esta historia mostrará cómo bitcoin puede ser usado para pagos business-to-business internacionales relacionados con bienes físicos.

Minando bitcoins

Jing es un ingeniero de sistemas en Shanghái. Ha construido un equipo de "minería" para minar bitcoins, utilizando sus habilidades ingenieriles para suplementar sus ingresos. Esta historia examinará la base "industrial" de bitcoin: el equipo especializado utilizado para asegurar la red bitcoin y emitir nueva moneda.

Cada una de estas historias está basada en personas e industrias reales que actualmente utilizan bitcoin para crear nuevos mercados, nuevas industrias y soluciones innovadoras a problemas de la economía global.

Primeros Pasos

Para unirse a la red bitcoin y comenzar a usar la moneda, todo lo que un usuario debe hacer es descargar una aplicación o usar una aplicación web. Ya que bitcoin es un estándar existen diversas implementaciones del software cliente de bitcoin. Existe también una implementación de referencia, también conocida como el cliente Satoshi, que se administra como un proyecto de código abierto por un equipo de desarrolladores y deriva de la implementación originalmente escrita por Satoshi Nakamoto.

Las tres formas principales de clientes bitcoin son:

Cliente completo

Un cliente completo, o "nodo completo," es un cliente que almacena la totalidad del historial de transacciones bitcoin (cada transacción de cada usuario de todos los tiempos), administra las carteras del usuario y puede iniciar transacciones directamente sobre la red bitcoin. Esto es análogo a un servidor de email autónomo en el sentido en que se ocupa de cada aspecto del protocolo sin requerir de ningún otro servidor o servicio de terceros.

Cliente ligero

Un cliente ligero almacena las carteras del usuario pero depende de servidores de terceros para acceder a las transacciones y la red bitcoin. El cliente ligero no almacena una copia completa de las transacciones y por lo tanto debe confiar en los servidores de terceros para la validación de transacciones. Esto es similar a un cliente de email autónomo que se conecta a un servidor de correo para acceder al buzón, ya que depende de terceros para interacciones con la red.

Cliente web

Los clientes web se acceden a través de un navegador web y almacenan las carteras de los usuarios en servidores de terceros. Esto es análogo al webmail ya que depende enteramente de servidores de terceros.

Bitcoin Móvil

Los clientes móviles para smartphones, como los basados en el sistema Android, pueden operar como clientes completos, clientes ligeros o clientes web. Algunos clientes móviles se sincronizan con una web o cliente de escritorio, proporcionando una cartera multiplataforma a través de múltiples dispositivos pero con una fuente común de fondos.

La elección de cliente bitcoin depende de cuánto control quiera tener el usuario sobre sus fondos. Un cliente completo ofrece el mayor nivel de control e independencia al usuario, pero en contraparte coloca la responsabilidad de realizar backups y mantener la seguridad sobre el usuario. En el otro extremo del espectro de opciones, el cliente web es el más simple de montar y usar, pero su contra es que introduce riesgo ajeno ya que la seguridad y el control están compartidos entre el usuario y el dueño del servicio web. Si una cartera web se pusiera en peligro, tal como ha sucedido con varias, el usuario podría perder potencialmente todos sus fondos. Por el contrario, si el usuario posee un cliente completo sin los backups correspondientes, podría perder sus fondos debido a un desperfecto del computador.

Para los propósitos de este libro haremos una demostración del uso de una variedad de clientes descargables, desde la implementación de referencia (el cliente de Satoshi) hasta las carteras web. Algunos de los ejemplos requerirán el uso del cliente de referencia, el cual, además de ser un cliente completo, también expone APIs para la cartera, la red y los servicios de transacción. Si planeas explorar las interfaces programáticas del sistema bitcoin necesitarás el cliente de referencia.

Comienzo Rápido

Alice, a quien introdujimos en Usos de Bitcoin, Usuarios y Sus Historias, no es una usuaria técnica y ha oído sobre bitcoin muy recientemente a través de un amigo. Ha comenzado su travesía visitando el sitio oficial bitcoin.org, donde ha encontrado una amplia selección de clientes bitcoin. Siguiendo el consejo del sitio bitcoin.org ha elegido el cliente bitcoin ligero Multibit.

Alice sigue un enlace desde el sitio bitcoin.org para descargar e instalar Multibit en su computador de escritorio. Multibit está disponible para Windows, Mac OS y Linux.

WARNING

Una cartera bitcoin debe protegerse mediante una palabra o frase clave. Existen muchos agentes maliciosos intentando romper contraseñas inseguras, así que asegúrate de elegir una que no sea fácil de romper. Utiliza una combinación de caracteres en mayúsculas y minúsculas, números y símbolos. Evita incluir información personal como fechas de cumpleaños o nombres de equipos deportivos. Evita palabras comúnmente encontradas en diccionarios de cualquier idioma. De ser posible utiliza un generador de contraseñas para generar una contraseña completamente aleatoria que sea de al menos 12 caracteres de largo. Recuerda: bitcoin es dinero y puede ser movido instantáneamente a cualquier punto del planeta. Si no se protege debidamente, puede ser fácilmente robado.

Una vez que Alice ha descargado e instalado la aplicación Multibit, lo ejecuta y es recibida por la pantalla de Bienvenida, tal como se muestra en La pantalla de Bienvenida del cliente bitcoin Multibit.

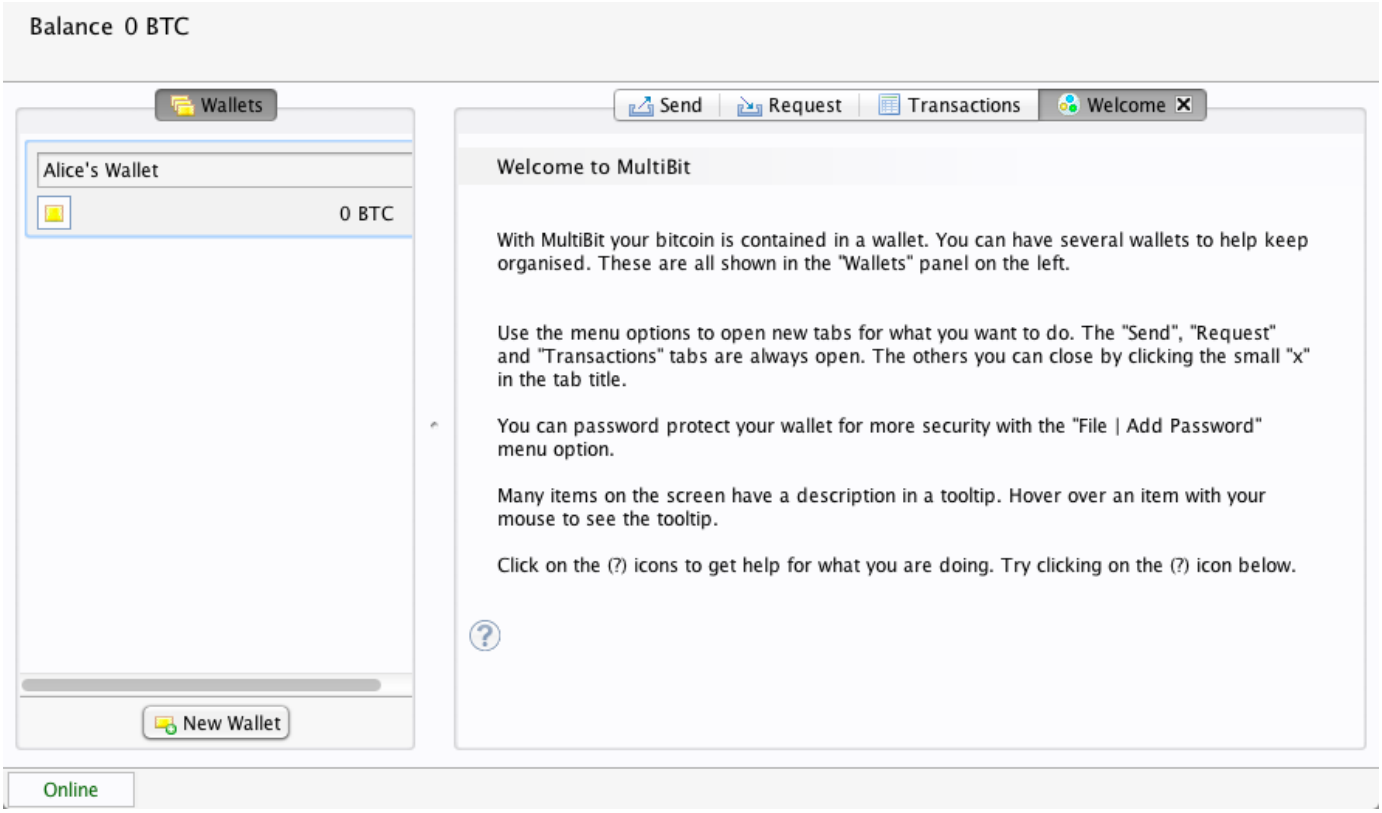


Figure 1. La pantalla de Bienvenida del cliente bitcoin Multibit

Multibit automáticamente crea una cartera y una nueva dirección bitcoin para Alice, que Alice puede ver haciendo clic sobre la pestaña de "Request" ("Solicitar") mostrada en La nueva dirección bitcoin de Alice en la pestaña de Solicitar del cliente Multibit.

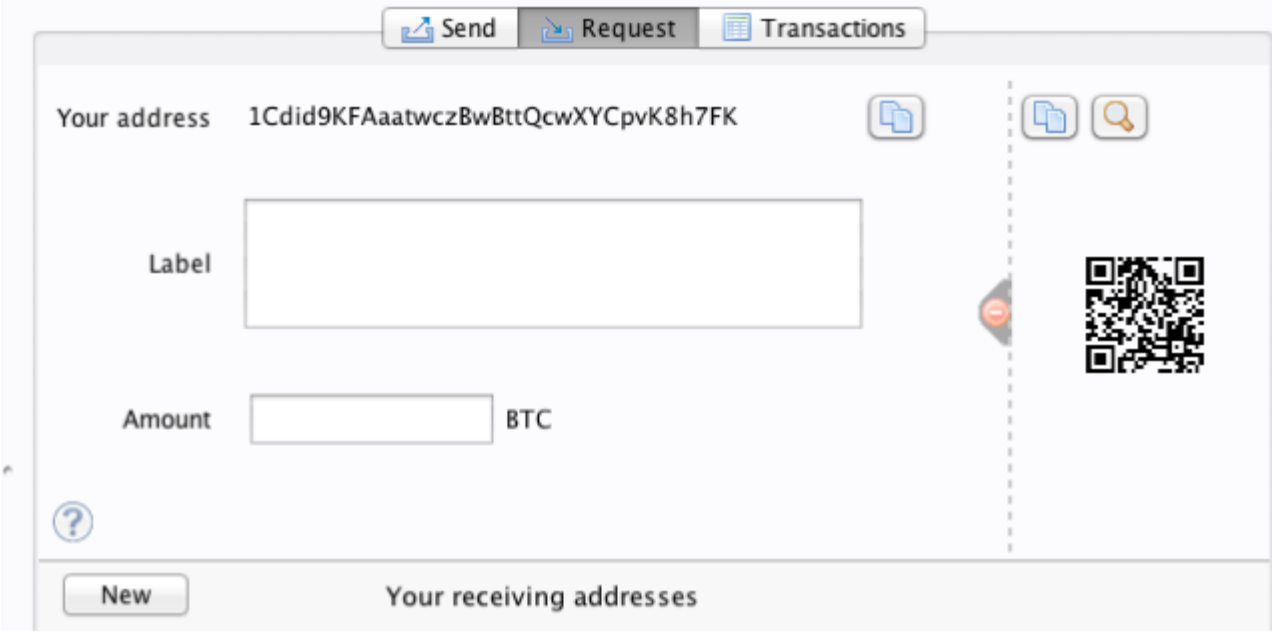


Figure 2. La nueva dirección bitcoin de Alice en la pestaña de Solicitar del cliente Multibit

La parte más importante de esta pantalla es la *dirección bitcoin* de Alice. Al igual que una dirección de email, Alice puede compartir esta dirección y cualquiera puede usarla para enviar dinero directamente a su nueva cartera. En la pantalla aprece como una larga secuencia de letras y números: 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK. Junto a la dirección bitcoin en su cartera hay un código QR, una forma de código de barras que contiene la misma información en un formato que puede ser escaneado por la cámara de un smartphone. El código QR es el cuadro en blanco y negro a la derecha de su ventana. Alice puede copiar la dirección bitcoin o el código QR a su portapapeles haciendo clic en el botón de copiado adyacente a cada uno de ellos. Un clic sobre el código QR en sí mismo lo ampliará para que sea fácilmente escaneable por la cámara de un teléfono.

Alice también puede imprimir el código QR como una forma sencilla de dar su dirección a otros sin ellos tener que teclear una larga cadena de letras y números.

TIP

Las direcciones bitcoin comienzan con el dígito 1 o 3. Al igual que las direcciones de email pueden ser compartidas con otros usuarios de bitcoin quienes pueden usarlas para enviar bitcoin directamente a tu cartera. A diferencia de las direcciones de email puedes crear nuevas direcciones tan frecuentemente como desees, todas las cuales dirigirán fondos a tu cartera. Una cartera es sencillamente una colección de direcciones y las claves que permiten gastar los fondos en ellas. Puedes incrementar tu privacidad usando una dirección distinta para cada transacción. Prácticamente no existe límite al número de direcciones que un usuario puede crear.

Alice ahora se encuentra preparada para empezar a usar su nueva cartera bitcoin.

Obteniendo tus Primeros Bitcoins

No es posible comprar bitcoins en un banco o casa de cambio de monedas extranjeras por el momento. Hacia 2014 es aún muy difícil comprar bitcoins en la mayoría de los países. Existe un número de casas

de cambio especializadas donde uno puede comprar y vender bitcoins por moneda local. Éstas operan como mercados de moneda en la web e incluyen:

Bitstamp

Un mercado europeo de divisas que soporta varias monedas, incluyendo euros (EUR) y dólares estadounidenses (USD) a través de transferencias bancarias.

Coinbase

Una cartera y plataforma bitcoin radicada en EEUU donde comerciantes y consumidores pueden realizar transacciones en bitcoin. Coinbase facilita la compra y venta de bitcoins, permitiendo a sus usuarios conectar sus cuentas corrientes de bancos estadounidenses a través del sistema ACH.

Las casas de cambio de criptomonedas como estas operan como punto de conexión entre monedas nacionales y criptomonedas. Como tales se encuentran sujetas a regulaciones internacionales y usualmente se limitan a un único país o región económica y se especializan en las monedas nacionales de esa área. Tu elección de casa de cambio de monedas será específica a la moneda nacional que uses y limitada a las casas de cambio que operan dentro de la jurisdicción legal de tu país. Al igual que abrir una cuenta bancaria, puede llevar días o hasta semanas el montar cuentas con estos servicios ya que requieren de varias formas de identificación para cumplir con los requisitos de las regulaciones bancarias KYC (know your customer) y AMl (ant-money laundering, o anti lavado de dinero). Una vez que tengas una cuenta en una casa de cambio de bitcoin puedes comprar y vender bitcoins rápidamente tal y como harías con moneda extranjera a través de una agencia de corredores.

Puedes encontrar una lista más completa en bitcoin charts, un sitio que ofrece cotizaciones de precios y otros datos del mercado para varias docenas de casas de cambio de monedas.

Existen otros cuatro métodos de obtención de bitcoins como usuario nuevo:

- Encuentra un amigo que tenga bitcoins y cómprale algunos directamente. Muchos usuarios de bitcoin comienzan de esta forma.
- Utilizar un servicio de clasificados como localbitcoins.com para encontrar un vendedor en tu área a quien comprarle bitcoins con efectivo en una transacción en persona.
- Vender un producto o servicio por bitcoins. Si eres un programador puedes ofrecer tus habilidades de programación.
- Usa un cajero automático bitcoin en tu ciudad. Encuentra un cajero cercano a ti usando el mapa de CoinDesk.

Alice conoció bitcoin gracias a un amigo y por lo tanto tiene una forma simple de obtener sus primeros bitcoins mientras espera a que su cuenta en un mercado de divisas de California sea verificada y activada.

Enviando y Recibiendo Bitcoins

Alice ha creado su cartera bitcoin y ahora está lista para recibir fondos. Su aplicación de cartera generó una clave pública aleatoriamente (descrita en más detalle en [private_keys]) junto con su

correspondiente dirección bitcoin. A este punto su dirección bitcoin no es conocida a la red bitcoin ni está "registrada" en ninguna parte del sistema bitcoin. Su dirección bitcoin es simplemente un número que corresponde a una clave que ella puede usar para acceder a los fondos. No existe ninguna cuenta ni asociación entre direcciones y una cuenta. Hasta el momento en que una dirección es referenciada como la destinataria del valor en una transacción publicada en el libro contable bitcoin (la cadena de bloques), es simplemente parte de un vasto número de posibles direcciones "válidas" en bitcoin. Una vez que ha sido asociada con una transacción se convierte en parte de las direcciones conocidas por la red y Alice puede verificar su saldo en el libro contable público.

Alice se encuentra con su amigo Joe, quien le presentó bitcoin, en un restaurante para intercambiar algunos dólares estadounidenses y poner algunos bitcoins en su cuenta. Ella ha llevado impresas su dirección y el código QR tal como lo muestra su cartera bitcoin. No hay nada sensible, desde una perspectiva de seguridad, sobre la dirección bitcoin. Puede ser publicada en cualquier parte sin arriesgar la seguridad de su cuenta.

Alice quiere convertir tan solo 10 dólares estadounidenses a bitcoin para evitar arriesgar demasiado dinero en esta nueva tecnología. Entrega a Joe un billete de $10 y la impresión de su dirección para que Joe pueda enviar el valor equivalente en bitcoins.

A continuación Joe debe averiguar la tasa de cambio para poder dar la cantidad correcta de bitcoins a Alice. Existen centenares de aplicaciones y sitios que informan de la tasa de mercado actual. Aquí hay algunas de las más populares:

Bitcoin Charts

: Un servicio de listado de datos que muestra la tasa de cambio de bitcoin para diversas casas de cambio alrededor del mundo, denominadas en distintas monedas locales

Bitcoin Average

: Un sitio que proporciona una vista simple del promedio ponderado por volumen de cada moneda

ZeroBlock

: Una aplicación gratuita para Android e iOS que muestra el precio de bitcoin de distintas casas de cambio (ver ZeroBlock, una aplicación de precio de mercado de bitcoin para Android e iOS)

Bitcoin Wisdom

: Otro servicio de listado de datos de mercado

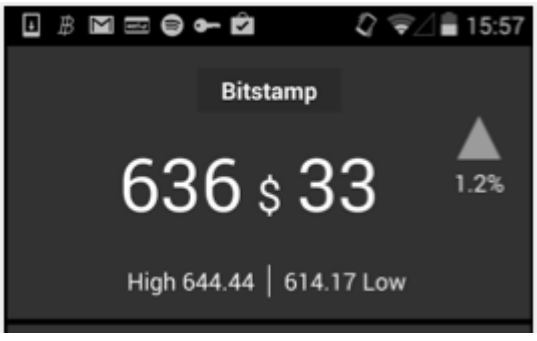


Figure 3. ZeroBlock, una aplicación de precio de mercado de bitcoin para Android e iOS

Usando una de las aplicaciones o sitios web listados, Joe determina que el precio de un bitcoin es aproximadamente 100 dólares norteamericanos. A este cambio él debe dar a Alice 0.10 bitcoin, támbien conocidos como 100 milibits, a cambio de los 10 dólares americanos que ella dio a él.

Una vez que Joe ha establecido un precio justo abre su aplicación de cartera y selecciona "enviar" bitcoins. Por ejemplo, si utiliza la cartera móvil de Blockchain en un teléfono Android vería una pantalla requiriendo dos valores, como se muestra en La pantalla de envío de bitcoins de la cartera móvil Blockchain.

- La dirección bitcoin destinataria para la transacción
- El monto de bitcoins a enviar

En el campo de texto de la dirección bitcoin existe un pequeño icono que se ve como un código QR. Esto permite a Joe escanear el código con la cámara de su teléfono, evitando teclear la dirección bitcoin de Alice (1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK), la cual es larga y difícil de teclear. Joe toca sobre el icono del código QR y activa la cámara de su teléfono, escaneando el código QR de la cartera impresa de Alice que ella ha llevado. La aplicación de cartera móvil llena la dirección bitcoin y Joe puede verificar que se ha escaneado correctamente comparando los dígitos de la dirección con la versión impresa por Alice.

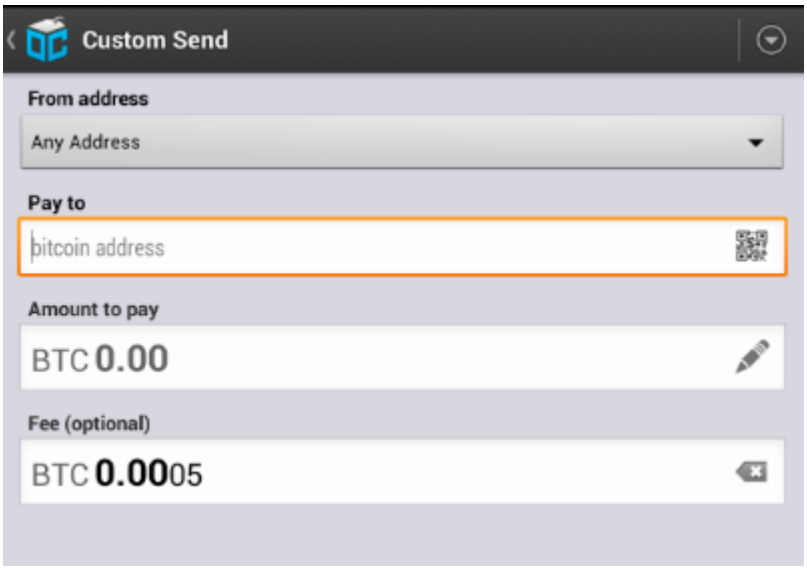


Figure 4. La pantalla de envío de bitcoins de la cartera móvil Blockchain

Joe ingresa el valor bitcoin para la transacción, 0,10 bitcoins. Verifica cuidadosamente que ha ingresado el valor correcto, ya que está a punto de enviar dinero y cualquier error puede resultar costoso. Finalmente pulsa Send para transmitir la transacción. La cartera bitcoin móvil de Joe construye una transacción que asigna 0,10 bitcoins a la dirección provista por Alice, derivando los fondos de la cartera de Joe y firmando la transacción con la firma digital de Joe. Esto notifica a la red bitcoin que Joe ha autorizado transferir valor de una de sus direcciones a la dirección de Alice. Como la dirección se transmite mediante un protocolo entre pares, rápidamente se propaga por toda la red bitcoin. En menos de un segundo, la mayoría de los nodos bien conectados en la red reciben la transacción y ven la dirección de Alice por primera vez.

Si Alice tiene un smartphone o laptop consigo, también será capaz de ver la transacción. El libro contable bitcoin—un archivo en constante crecimiento que registra cada transacción bitcoin que ha ocurrido desde el comienzo—es público, lo cual significa que todo lo que ella debe hacer es buscar su propia dirección y ver si se le han enviado fondos. Hacer eso es muy fácil en el sitio web blockchain.info, simplemente ingresando su dirección en el campo de búsqueda. El sitio le mostrará una página listando todas las transacciones desde y hacia su dirección. Si Alice está observando esa página se actualizará mostrando una nueva transacción transfiriendo 0,10 bitcoins a su saldo poco después de que Joe presione Enviar.

Confirmaciones

Al principio la dirección de Alice mostrará la transacción de Joe como "sin confirmar". Esto significa que la transacción ha sido propagada por la red pero no ha sido incluida aún en el libro contable bitcoin, conocido como la cadena de bloques (blockchain). Para ser incluida, la transacción debe ser "recogida" por un minero e incluida en un bloque de transacciones. Una vez que se cree un nuevo bloque, en aproximadamente 10 minutos, las transacciones dentro del bloque serán aceptadas y "confirmadas" por la red y pueden ser gastadas. La transacción es vista instantáneamente, pero solo es "confiable" por todos cuando ha sido incluida en un bloque minado.

Alice es ahora la orgullosa propietaria de 0,10 bitcoins que puede gastar. En el próximo capítulo echaremos un vistazo a su primera compra con bitcoin y examinaremos las tecnologías de transacción y propagación en mayor detalle.

¿Cómo funciona Bitcoin?

Transacciones, Bloques, Minado, y la Cadena de Bloques

El sistema bitcoin, a diferencia de los sistemas de pago del sistema bancario tradicional, está basado en confianza descentralizada. En vez de confiar en una autoridad central, en bitcoin, la confianza se consigue como una propiedad emergente de las interacciones de diferentes participantes en el sistema bitcoin. En este capítulo, examinaremos bitcoin desde un alto nivel con el seguimiento de una sola transacción a través del sistema bitcoin y veremos cómo se convierte en "de confianza" y aceptada por el mecanismo de consenso distribuido de bitcoin y es finalmente guardada en la cadena de bloques, el libro contable de todas las transacciones.

Cada ejemplo se basa en una transacción hecha en la red bitcoin, simulando las interacciones entre los usarios (Joe, Alice, Bob) enviando fondos desde una cartera a otra. Siguiendo una transacción a través de la red de bitcoin y la cadena de bloques, usaremos un sitio web *explorador de la cadena de bloques* para ver cada paso. Un explorador de la cadena de bloques es una aplicación web que opera como un buscador de bitcoin, que te permite buscar direcciones, transacciones y bloques y ver las relaciones y flujos entre ellas.

Algunos exploradores son:

- Blockchain info
- Bitcoin Block Explorer
- insight
- blockr Block Reader

Cada una de estas tiene una función de búsqueda que puede llevarte a una dirección, hash de transacción o número de bloque y encontrar los datos en la red bitcoin y la cadena de bloques. Con cada ejemplo, daremos una URL que te llevará directamente a la entrada correspondiente, para que puedas estudiarla en detalle.

Visión General de Bitcoin

En el diagrama de visión general mostrado en Visión general de Bitcoin, vemos que el sistema bitcoin consiste en usuarios con carteras que contienen claves, transacciones que se propagan a través de la red y mineros que producen (a través de competición competitiva) el consenso en la cadena de bloques, que es libro de contabilidad de todas las transacciones. En este capítulo, haremos el seguimiento de una sola transacción a través de la red y examinaremos las interacciones entre cada parte del sistema bitcoin, a un alto nivel. Los siguientes capítulos profundizarán en la tecnología detrás de las carteras, el minado y los sistemas mercantes.

Figure 1. Visión general de Bitcoin

Comprando una Taza de Café

Alice, presentada en el anterior capítulo, es una nueva usuaria que quiere adquirir su primer bitcoin. La transacción creada por Joe transfirió a la cartera de Alice 0.10 BTC. Ahora Alice hará su primera adquisición, comprando una taza de café en la cafetería de Bob en Palo Alto, California. La cafetería de Bob empezó hace poco a aceptar pagos en bitcoin, añadiendo la opción a su sistema de punto de ventas. Los precios en la cafetería de Bob están listados tanto en dólares como en bitcoin. Alice pide su taza de café y Bob introduce la transacción en la máquina registradora. El punto de ventas convierte el precio total de $ a bitcoins a la tasa de cambio del momento en el mercado y enseña los precios en ambas monedas, así como un código QR que contiene una *solicitud de pago* por su transacción (ver Código QR de solicitud de pago (Pista: ¡Prueba escaneando esto!)):

```
Total:
$1,50 USD
0,015 BTC
```

Figure 2. Código QR de solicitud de pago (Pista: ¡Prueba escaneando esto!)

El código de solicitud de pago da la siguiente URL, definida en BIP0021:

```
bitcoin:1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA?
cantidad=0.015&
etiqueta=Bob%27s%20Cafe&
mensaje=Purchase%20at%20Bob%27s%20Cafe

Composición de la URL

Una dirección bitcoin: "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
La cantidad a pagar: "0.015"
Una etiqueta para la dirección del receptor: "Cafetería de Bob"
Una descripción del pago: "Compra en Cafetería de Bob"
```

TIP A diferencia del código QR que solo contiene una dirección de destino, una petición de pago es una URL codificada en QR que contiene una dirección destino, la cantidad a pagar y una descripción genérica como "Cafetería de Bob". Esto permite a la aplicación de cartera bitcoin rellenar la información usada para enviar el pago mientras muestra al usuario una descripción legible. Puede escanear el código QR con una aplicación de cartera bitcoin para ver lo que Alice vería.

Bob dice, "Son 1,50$, o 15 milibits."

Alice usa su móvil para escanear el código de barras en la pantalla. Su móvil muestra un pago de 0,0150 BTC a Cafetería de Bob y selecciona Enviar para autorizar el pago. En unos pocos segundos (más o menos el mismo tiempo que la autorización de una tarjeta de crédito), Bob debería ver la transacción en el registro, completando la transacción.

En la siguiente sección examinaremos esta transacción en más detalle, veremos cómo la construyó la cartera de Alice, cómo fue propagada a través de la red, cómo se verificó y, finalmente, cómo Bob puede gastar esa cantidad en siguientes transacciones

NOTE

La red bitcoin puede transferir en valores de fracciones, por ejemplo, desde mili-bitcoins (1/1000 parte de bitcoin) hasta 1/100.000.000 parte de bitcoin, lo que se conoce como satoshi. A lo largo de este libro usaremos el término "bitcoin" al referirnos a la cantidad de moneda bitcoin, desde la más pequeña unidad (1 satoshi) al número total (21.000.000) de todos los bitcoins que serán minados.

Transacciones Bitcoin

En términos simples, una transacción dice a la red que el propietario de un número de bitcoins ha autorizado la transferencia de algunos de esos bitcoins a otro propietario. El nuevo propietario puede ahora gastar esos bitcoins creando otra transacción que autorice la transferencia a otro propietario, y así sucesivamente, en una cadena de propiedad.

Las transacciones son como líneas en un libro contable de doble entrada. Simplificando, cada transacción contiene una o más "entradas", que son débitos contra una cuenta bitcoin. En el otro lado de la transacción, hay una o más "salidas", que son créditos añadidos a una cuenta bitcoin. Las entradas y salidas (débitos y créditos) no suman necesariamente la misma cantidad. En su lugar, las salidas suman un poco menos que las entradas y la diferencia representa una comisión de transacción implícita, que es un pequeño pago recogido por el minero que incluye la transacción en el libro contable. Una transacción bitcoin se muestra como una entrada del libro contable en Transacciones como contabilidad de doble entrada.

La transacción también contiene la prueba de propiedad para cada cantidad de bitcoin (entradas) desde las que se transfiere valor, en forma de una firma digital del propietario, que cualquiera puede validar independientemente. En términos de bitcoin, "gastar" es firmar una transacción que transfiera valor desde una transacción previa hacia un nuevo propietario identificado por una dirección bitcoin.

TIP

Las *transacciones* mueven valor desde las *entradas de transacción* a las *salidas de transacción*. Una entrada es desde donde viene el valor de la moneda, normalmente la salida de una transacción previa. Una salida de transacción asigna el valor a un nuevo propietario al asociarlo a una clave. La clave de destino es en realidad un *script de bloqueo*. El script de bloqueo es una secuencia de comandos que requiere de una firma digital u otra forma de validacion (*script de desbloqueo*) para poder acceder a los fondos en futuras transacciones. Las salidas de una transacción se podrán usar como entradas en una nueva transacción posterior, creando una cadena de propiedad mientras el valor se mueve desde una dirección a otra dirección de forma sucesiva (ver Una cadena de transacciones, donde la salida de una transacción es la entrada de la siguiente transacción).

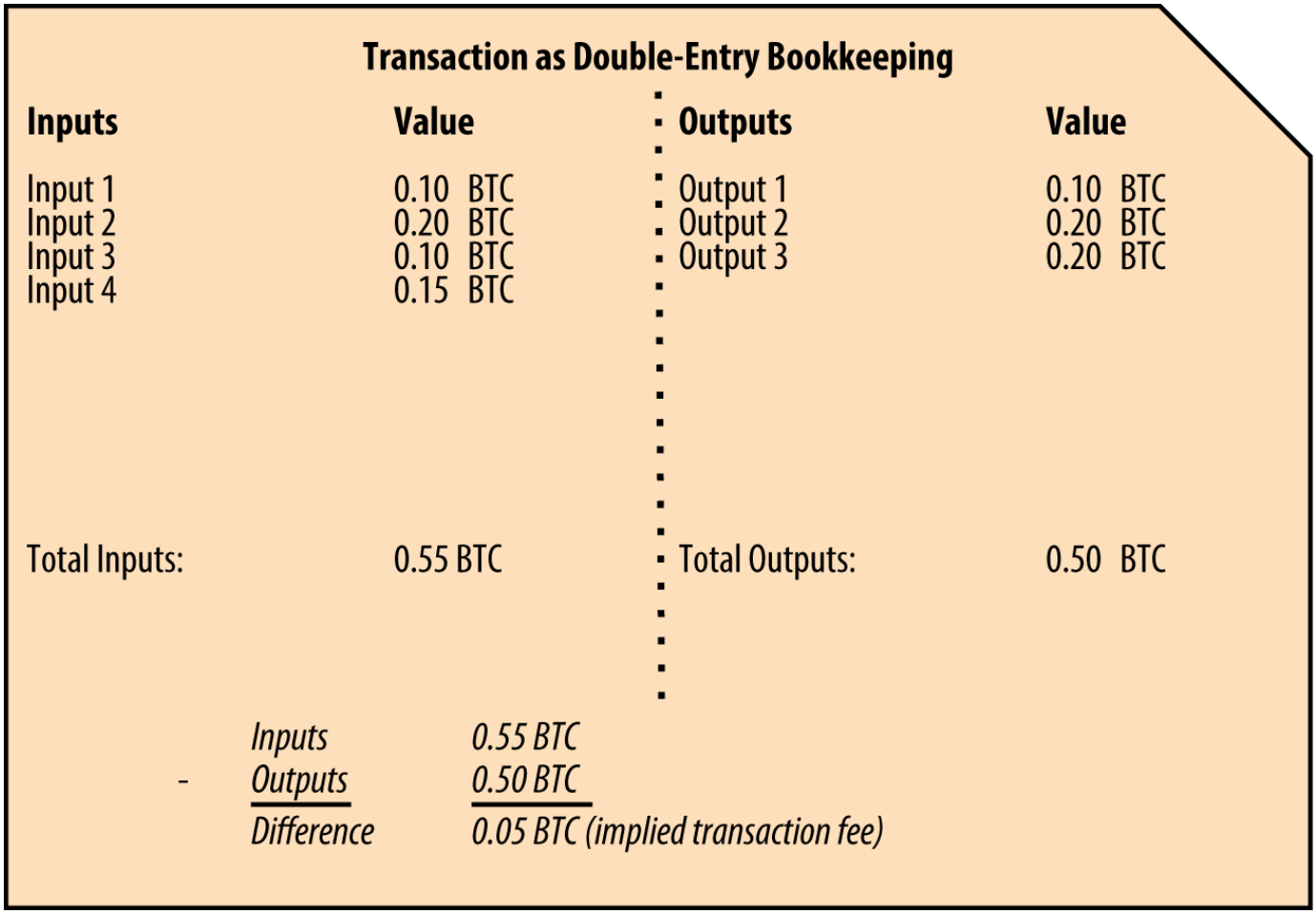


Figure 3. Transacciones como contabilidad de doble entrada

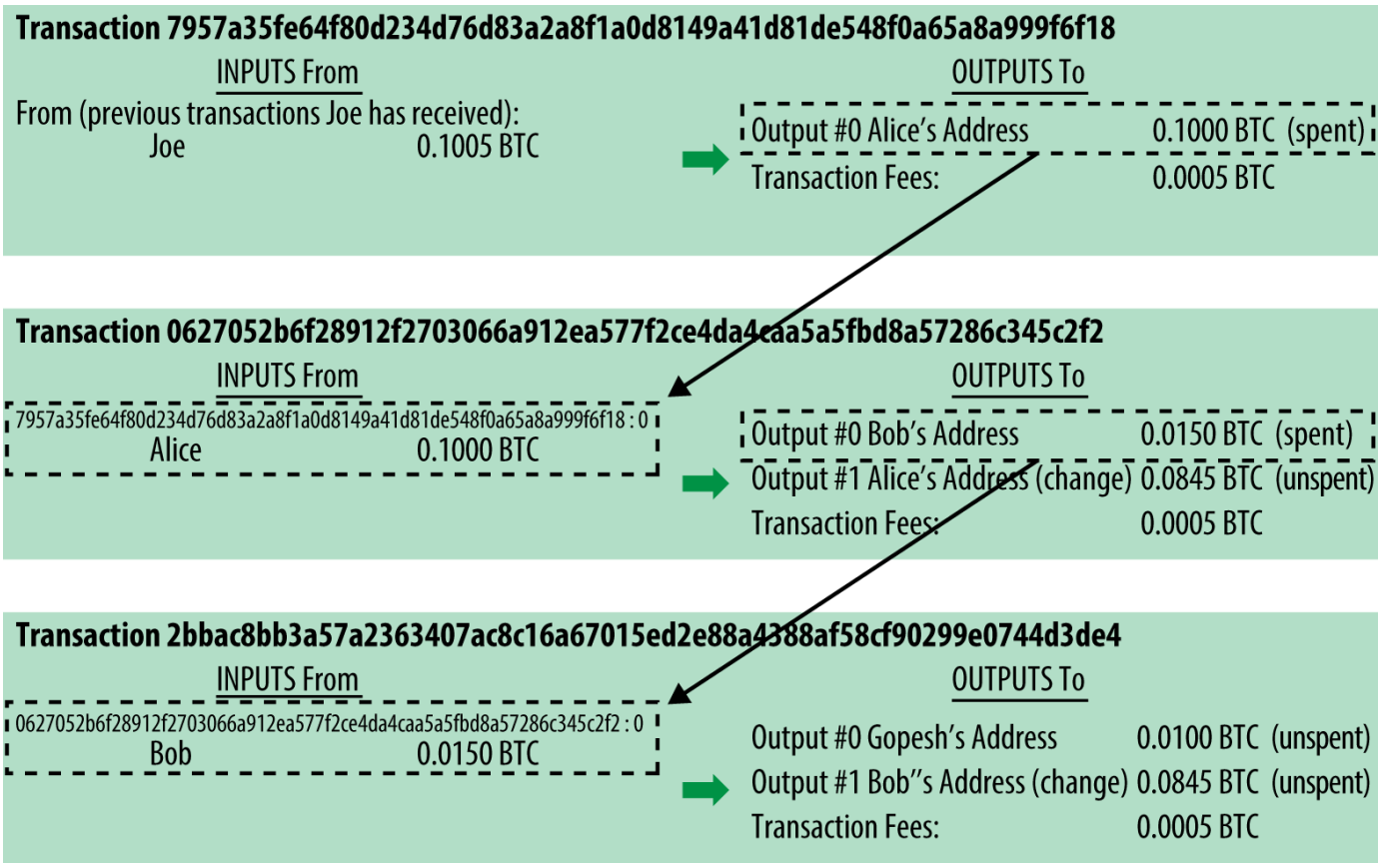


Figure 4. Una cadena de transacciones, donde la salida de una transacción es la entrada de la siguiente transacción

El pago de Alice a la Cafetería de Bob usa una transacción previa como su entrada. En el anterior capítulo Alice recibió algún bitcoin de su amigo Joe a cambio de efectivo. La transacción tiene un número de bitcoins bloqueados (obstruidos) que solo la clave de Alice puede desbloquear. Su nueva transacción a la Cafetería de Bob hace referencia a la transacción previa como entrada y crea nuevas salidas para pagar la taza de café y recibir el cambio. Las transacciones forman una cadena, donde las entradas de la última transacción corresponden a salidas de transacciones previas. La clave de Alice proporciona la firma que desbloquea esas salidas de la transacción anterior, demostrando a la red bitcoin que ella es la propietaria del dinero. Ella adjunta el pago del café a la dirección de Bob, y por tanto "obstruye" esa salida bajo el requisito de que Bob produzca una firma para gastar esa cantidad. Esto representa una transferencia de valor entre Alice y Bob. Esta cadena de transacciones, desde Joe a Alice a Bob, está ilustrada en Una cadena de transacciones, donde la salida de una transacción es la entrada de la siguiente transacción.

Formas Comunes de Transacción

La forma más común de transacción es un pago simple de una dirección a otra, que a menudo incluye algo de "cambio" devuelto al propietario original. Este tipo de transacción tiene una entrada y dos salidas y se puede ver en Transacción más común.

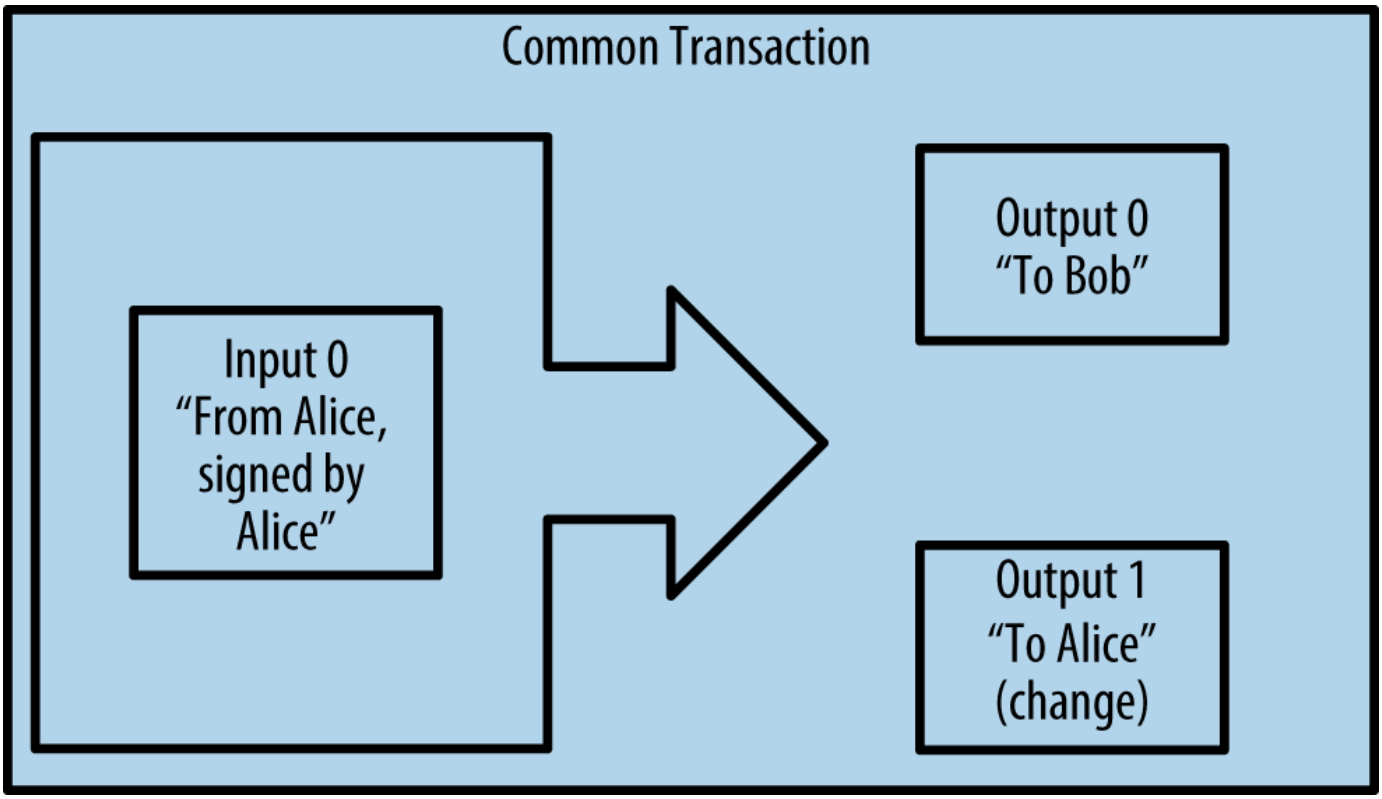


Figure 5. Transacción más común

Otra forma común de transacción es una que agrega muchas entradas en una sola salida (ver Transacciones de agregación de fondos). Esto representa el equivalente en el mundo real a intercambiar un montón de monedas y billetes en un único billete más grande. Las transacciones como esas a veces se generan por las aplicaciones de monedero para limpiar muchas cantidades pequeñas recibidas como cambio por pagos.

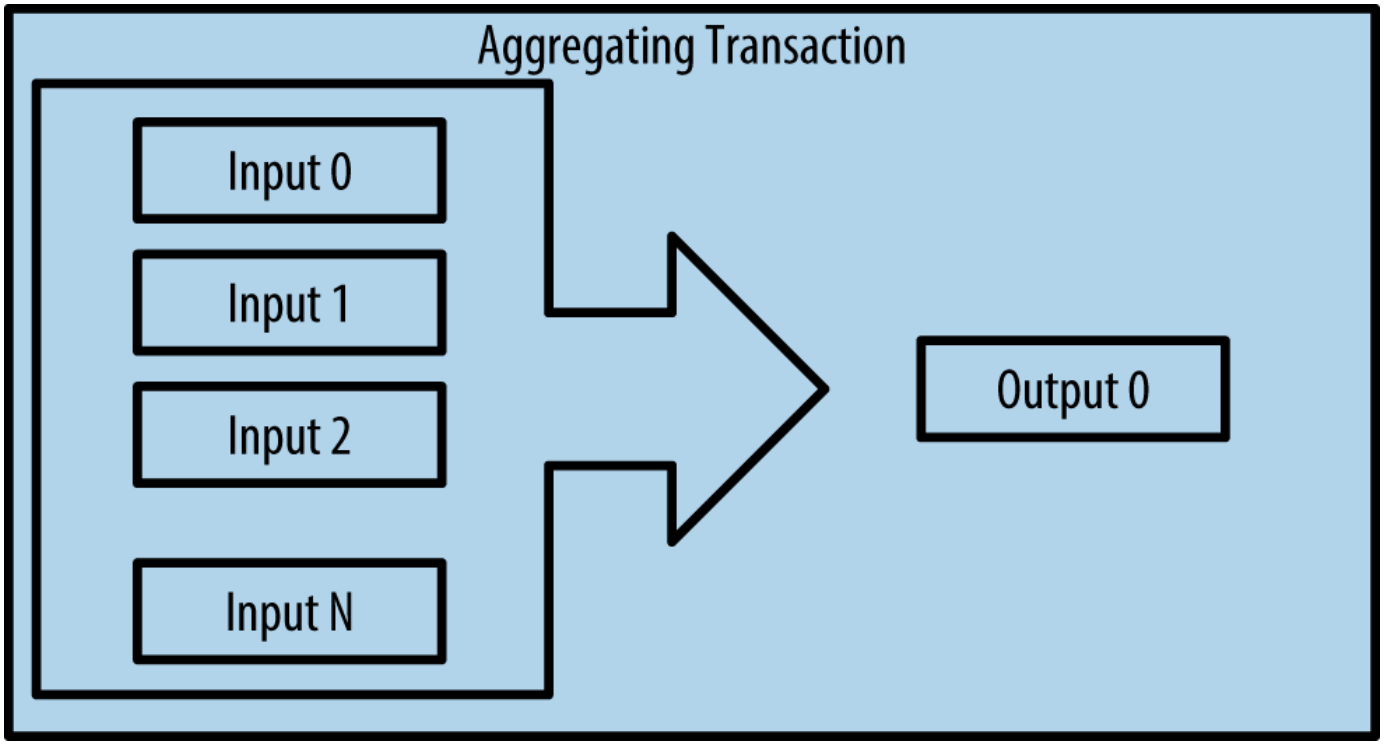


Figure 6. Transacciones de agregación de fondos

Finalmente, otra forma de transacción que se ve a menudo en el libro contable de bitcoin es una transacción que distribuye una entrada a múltiples salidas representando múltiples receptores (ver Distribución de fondos de transacción). Este tipo de transacción es a veces usada por las entidades comerciales para distribuir fondos, como cuando se procesan salarios a múltiples empleados.

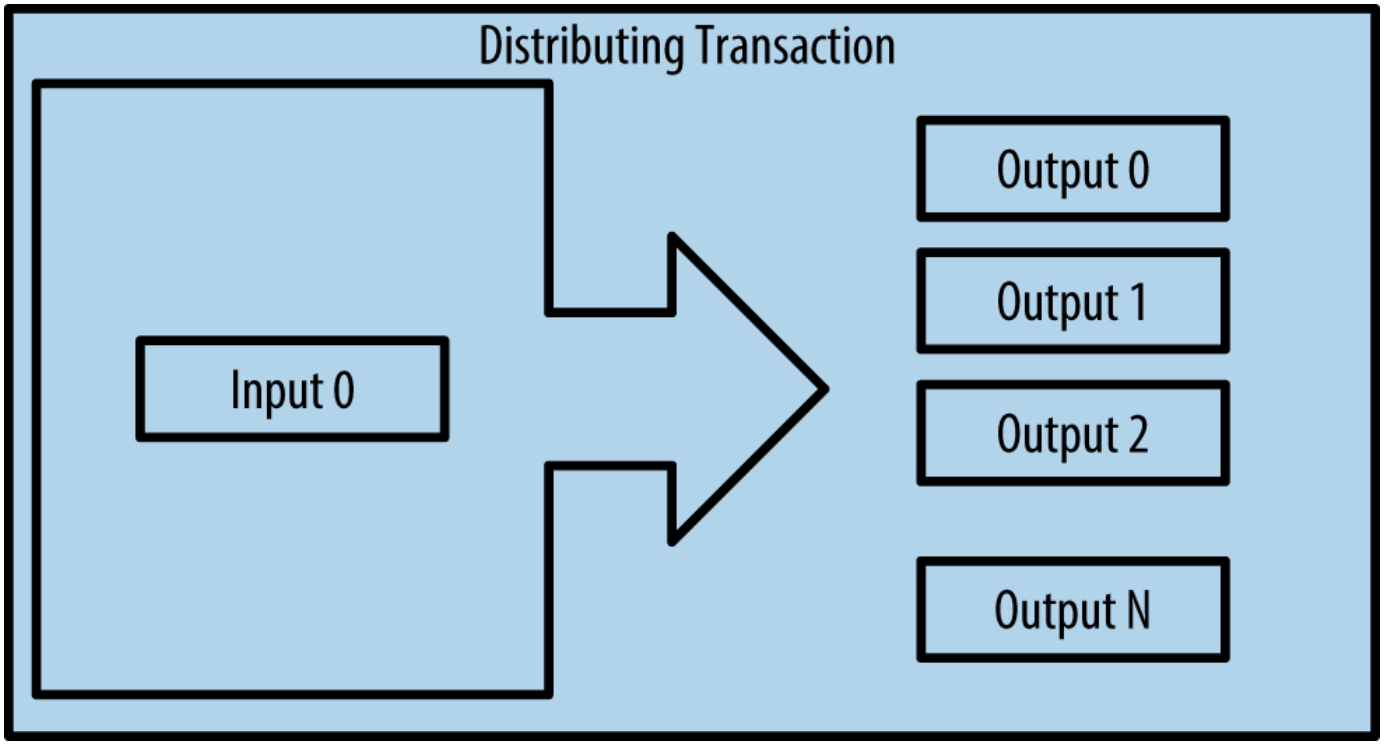


Figure 7. Distribución de fondos de transacción

Construyendo una Transacción

La aplicación de monedero de Alice contiene toda la lógica necesaria para seleccionar entradas y salidas al construir una transacción que cumpla los requisitos de Alice. Alice solo necesita especificar un destino y una cantidad, y el resto sucede en el monedero sin que ella tenga que ver los detalles. Es importante destacar que un monedero puede construir transacciones aun cuando esté completamente sin conexión. De la misma forma que se puede escribir un cheque en casa y luego enviarlo al banco en un sobre, la transacción no necesita ser construida y firmada mientras se está conectado a la red bitcoin. Solo tiene que ser enviada a la red para ser ejecutada.

Consiguiendo las Entradas Correctas

La aplicación de monedero de Alice tendrá primero que encontrar entradas que puedan pagar la cantidad que ella quiere enviar a Bob. La mayoría de aplicaciones de monedero mantienen una pequeña base de datos de "salidas de transacción no gastadas" que están bloqueadas (obstruidas) con las propias claves del monedero. Por tanto, el monedero de Alice puede contener una copia de la salida de la transacción de Joe, que fue creada a cambio de efectivo (ver [getting_first_bitcoin]). Un monedero que funcione como un cliente completo en realidad contiene una copia de las salidas no gastadas de todas las transacciones en la cadena de bloques. Esto permite al monedero construir nuevas entradas de transacción, así como verificar rápidamente que las transacciones entrantes tienen las entradas correctas. Sin embargo, debido a que un cliente completamente indexado requiere mucho espacio en el disco, la mayoría de usuarios utilizan clientes "ligeros" que solo llevan el control de las salidas no gastadas del propio usuario.

Si el monedero no mantiene una copia de las salidas no gastadas, puede consultar a la red bitcoin para que le proporcione esa información, usando una variedad de APIs disponibles a través de diferentes proveedores o solicitándoselo a un nodo completamente indexado mediante el API JSON RPC de bitcoin. Observa todas las salidas no gastadas de la dirección bitcoin de Alice. muestra una petición API RESTful construida a partir de un comando HTTP GET a una URL específica. Esta URL devolverá todas las salidas de transacción no gastadas de una dirección, proporcionando a cualquier aplicación la información que necesita para construir las entradas de transacción para ser gastadas. Usamos el cliente HTTP simple de línea de comandos *cURL* para obtener la respuesta.

Example 1. Observa todas las salidas no gastadas de la dirección bitcoin de Alice.

```
$ curl https://blockchain.info/unspent?active=1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK
```

Example 2. Respuesta a la búsqueda.

```
{

    "unspent_outputs":[

        {
            "tx_hash":"186f9f998a5...2836dd734d2804fe65fa35779",
            "tx_index":104810202,
            "tx_output_n": 0,
            "script":"76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
            "value": 10000000,
            "value_hex": "00989680",
            "confirmations":0
        }

    ]
}
```

La respuesta en Respuesta a la búsqueda. muestra una salida no gastada (una que aún no ha sido recuperada) bajo la propiedad de la dirección de Alice 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK. La respuesta incluye la referencia a la transacción en la que esta salida no gastada está contenida (el pago de Joe) y su valor en satoshis, a 10 millones, equivalente a 0.10 bitcoin. Con esta información, el monedero de Alice puede construir una transacción para transferir esa cantidad a la dirección del nuevo propietario.

TIP Ver transacción de Joe a Alice.

Como se puede ver, el monedero de Alice contiene suficientes bitcoins en una sola salida no gastada para pagar la taza de café. Si no fuera el caso, el monedero de Alice tendría que haber rebuscado en otras salidas no gastadas más pequeñas, como eligiendo monedas del bolsillo hasta que se encuentren las suficientes para pagar el café. En ambos casos, puede ser necesario que devuelvan el cambio, lo que veremos en la siguiente sección, donde el monedero crea las salidas de la transacción (pagos).

Creando las Salidas

Una salida de transacción se crea en la forma de un script que crea una obstrucción en el valor y solo puede ser recuperado por la introducción de una solución al script. Simplificando, la salida de transacción de Alice contendrá un script que dice algo así como, "Esta salida es pagable a quien pueda presentar una firma desde la clave correspondiente a la dirección pública de Bob". Debido a que solo Bob tiene el monedero con las claves que corresponden a esa dirección, solo el monedero de Bob puede presentar esa firma para recuperar esa salida. Alice por tanto habrá "obstruido" el valor de la salida con una solicitud de la firma de Bob.

Esta transacción también incluirá una segunda salida, porque los fondos de Alice están en la forma de una salida de 0.10 BTC, demasiado dinero para los 0.015 de la taza de café. Alice necesitará 0.085 BTC como cambio. El cambio de Alice se crea *por el monedero de Alice* en exactamente la misma transacción que el pago a Bob. En definitiva, el monedero de Alice divide sus fondos en dos pagos: uno a Bob, y otro de vuelta a ella misma. Entonces podrá usar la salida de cambio en la siguiente transacción, y por tanto gastarla más tarde.

Finalmente, para que la transacción sea procesada por la red de manera oportuna, el monedero de Alice añadirá una pequeña comisión. Esta no está explícita en la transacción; se saca de la diferencia entre entradas y salidas. Si en vez de llevarse 0.085 de cambio, Alice crea solo 0.0845 como segunda salida, habrá perdido 0.0005 BTC (medio milibitcoin). La entrada de 0.10 BTC no estará completamente gastada con las dos salidas, porque sumarán menos que 0.10. La diferencia resultante es la *comisión de transacción* que es recogida por el minero como tasa por incluir la transacción en un bloque e introducirla en la cadena de bloques.

La transacción resultante puede verse usando un explorador web de la cadena de bloques, tal como se muestra en Transacción de Alice a la Cafetería de Bob.

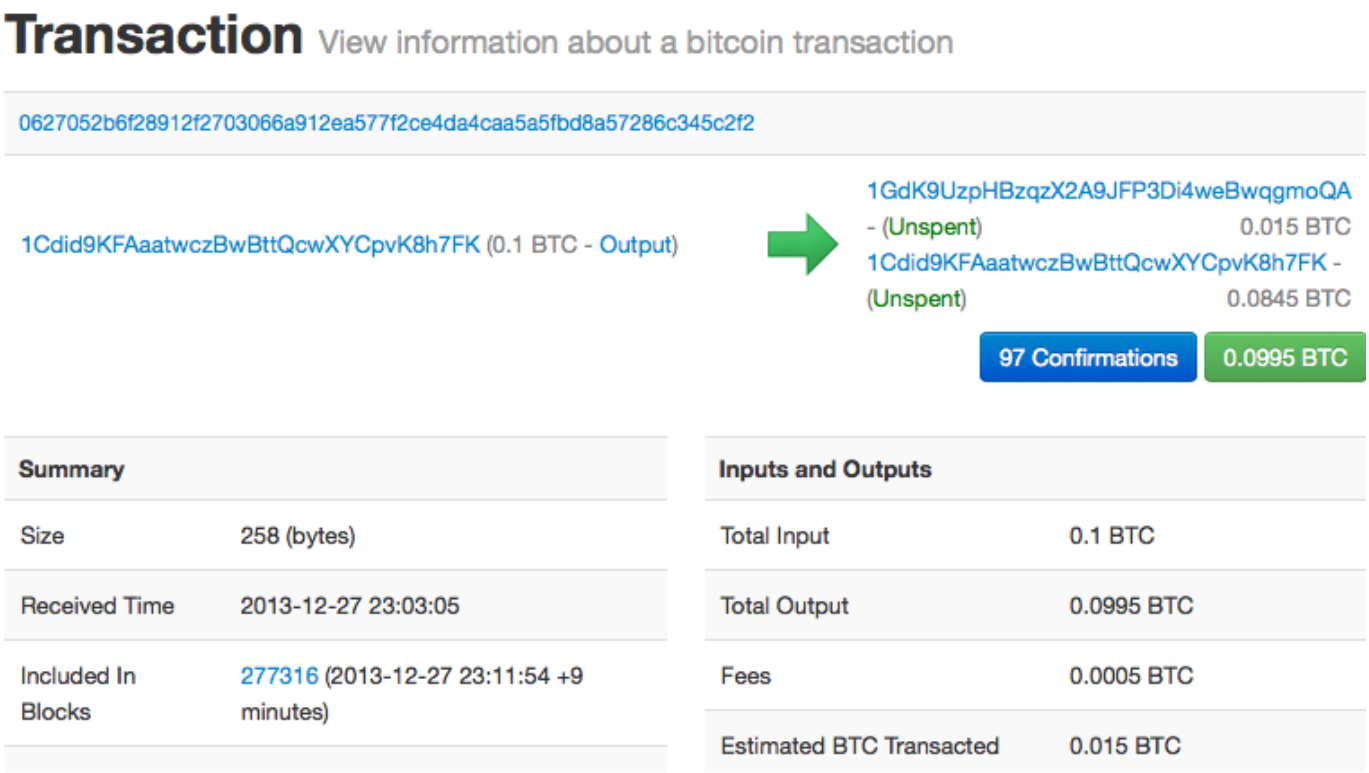


Figure 8. Transacción de Alice a la Cafetería de Bob

TIP Ver transacción de Alice a la Cafetería de Bob.

Añadiendo la Transacción al Libro Contable

La transacción creada por el monedero de Alice es de 258 bytes y contiene todo lo necesario para confirmar la propiedad de los fondos y asignar nuevos propietarios. Ahora, la transacción debe ser

transmitida a la red bitcoin donde será parte del libro contable distribuido (la cadena de bloques). En la siguiente sección veremos cómo una transacción se convierte en parte de un nuevo bloque y cómo se "mina" el bloque. Finalmente, veremos cómo el nuevo bloque, una vez añadido a la cadena de bloques, es cada vez más confiable por la red cuantos más bloques se añadan.

Transmitiendo la Transacción

Debido a que la transacción contiene toda la información necesaria para ser procesada, no importa cómo o desde dónde es transmitida a la red bitcoin. La red bitcoin es una red de igual a igual (P2P), en la que cada cliente de bitcoin participa conectándose a muchos otros clientes bitcoin. El propósito de la red bitcoin es propagar las transacciones y bloques a todos los participantes.

Cómo se propaga

El monedero de Alice puede enviar la nueva transacción a cualquiera de los otros clientes bitcoin conectados a cualquier conexión de internet: por cable, WiFi o móvil. Su monedero bitcoin no tiene por qué estar conectado al de Bob directamente y no tiene por qué usar la conexión a internet ofrecida por la cafetería, aunque ambas opciones son posibles. Cualquier nodo de la red bitcoin (otro cliente) que reciba una transacción válida que no haya visto anteriormente, la reenviará inmediatamente a los otros nodos a los que esté conectado. Por tanto, la transacción se propaga rápidamente a través de la red P2P, alcanzando un elevado porcentaje de los nodos en cuestión de segundos.

El Punto de Vista de Bob

Si el monedero de Bob está directamente conectado al monedero de Alice, el monedero de Bob podría ser el primer nodo en recibir la transacción. Sin embargo, aun si el monedero de Alice envía la transacción a través de otros nodos, alcanzará el monedero de Bob en unos pocos segundos. El monedero de Bob inmediatamente identificará la transacción de Alice como un pago entrante porque contiene salidas redimibles por las claves de Bob. El monedero de Bob puede también verificar independientemente que la transacción está bien formada, que usa entradas no gastadas previamente y que contiene una comisión de transacción suficiente para ser incluida en el siguiente bloque. En este punto Bob puede asumir, con bajo riesgo, que la transacción será incluida en un bloque y confirmada en poco tiempo.

TIP	Una concepción errónea común sobre las transacciones con bitcoin es que deben ser "confirmadas" esperando 10 minutos a un nuevo bloque, o hasta 60 minutos para completar seis confirmaciones. Aunque las confirmaciones aseguran que la transacción ha sido aceptada por la red en su totalidad, este retraso es innecesario para ítems de bajo valor como una taza de café. Un vendedor puede aceptar una transacción de bajo valor sin confirmaciones, tal como ya lo hacen hoy normalmente.

Minería de Bitcoin

La transacción se ha propagado en la red bitcoin. No es parte del libro contable compartido (la *cadena*

de bloques) hasta que se verifica y se incluye en un bloque por un proceso llamado *minería*. Ver [ch8] para una explicación detallada.

El sistema de confianza de bitcoin se basa en la computación. Las transacciones son empaquetadas en *bloques*, que requieren una enorme capacidad de computación para ser válidos, pero solo una pequeña cantidad de computación para ser validados. El proceso de minado sirve dos propósitos en bitcoin:

- La minería crea nuevos bitcoins en cada bloque, casi como un banco central imprimiendo nuevo dinero. La cantidad de bitcoin creado por bloque es fijo y disminuye con el tiempo.
- La minería crea confianza asegurando que las transacciones solo se confirman si se ha dedicado suficiente poder computacional al bloque que lo contiene. Más bloques significa más computación, lo que significa más confianza.

Una buena manera de describir la minería es como un juego competitivo de sudoku que se reinicia cada vez que alguien encuentra la solución y cuya dificultad automáticamente se ajusta para que lleve aproximadamente 10 minutos encontrar una solución. Imagina un sudoku gigante, de muchos miles de filas y columnas. Si te lo muestro completado puedes verificarlo rápidamente. Sin embargo, si el puzzle tiene unas pocas casillas completadas y el resto está vacío, ¡lleva mucho trabajo resolverlo! La dificultad del sudoku puede ajustarse cambiando su tamaño (más o menos filas y columnas), pero puede seguir siendo verificado fácilmente aunque sea enorme. El puzzle usado en bitcoin está basado en hashes criptográficos y tienen similares características: es asimétricamente dificil de resolver pero fácil de verificar, y su dificultad se puede ajustar.

En [user-stories], presentamos a Jing, un estudiante de ingeniería de computadoras en Shanghai. Jing participa en la red bitcoin como minero. Cada 10 minutos o así, Jing se une a miles de otros mineros en una carrera global para encontrar la solución a un bloque de transacciones. Encontrar esa solución, también llamada Prueba de Trabajo o PoW (Proof of Work en inglés) requiere cuatrillones de operaciones de hasheo por segundo a través de toda la red bitcoin. El algoritmo de la prueba de trabajo implica hacer hash repetidamente de las cabeceras del bloque y un número aleatorio con el algoritmo criptográfico SHA256 hasta que una solución encaje con un determinado patrón. El primer minero que encuentra esa solución gana la ronda de competición y publica ese bloque en la cadena de bloques.

Jing empezó a minar en 2010 usando una computadora de escritorio muy rápida para encontrar la correspondiente prueba de trabajo de nuevos bloques. Al incorporarse más mineros a la red bitcoin, la dificultad del problema fue creciendo rápidamente. Pronto, Jing y otros mineros actualizaron a hardware más específico, como unidades procesadores de gráficos especializados de gama alta (GPUs), tarjetas como las que se usan en ordenadores utilizados para videojuegos en ordenadores de escritorio o consolas. En el momento en que esto se escribe, la dificultad es tan alta que es rentable solamente minar con circuitos integrados de aplicación específica (ASIC), esencialmente miles de algoritmos de minería impresos en hardware, funcionando en paralelo en un único chip de silicio. Jing también se unió a una agrupación de minería (pool en inglés), que como una asociación lotera permite a diversos participantes compartir sus esfuerzos y las recompensas. Jing ahora hace funcionar dos máquinas ASIC conectadas mediante USB para minar bitcoin 24 horas al día. Paga sus costes de electricidad vendiendo los bitcoins que produce de la minería, generando algunos ingresos de los beneficios. Su computadora ejecuta una copia de bitcoind, el cliente bitcoin de referencia, como apoyo a su software

especializado en minería.

Minando Transacciones en Bloques

Una transacción transmitida a través de la red no es verificada hasta que forma parte del libro contable distribuido global, la cadena de bloques. Cada 10 minutos de media, los mineros generan un nuevo bloque que contiene todas las transacciones desde el último bloque. Las nuevas transacciones fluyen constantemente en la red desde los monederos de usuarios y otras aplicaciones. Cuando son vistas por los nodos de la red, se añaden a un conjunto temporal de transacciones no verificadas que es mantenido por cada nodo. Una vez los mineros crean un nuevo bloque, añaden las transacciones no verificadas desde esta agrupación a un nuevo bloque y luego intentan resolver un problema muy complejo (también conocido como prueba de trabajo) para probar la validez del nuevo bloque. El proceso de minado se explica en detalle en [mining].

Las transacciones se añaden al nuevo bloque, priorizadas por las de mayor comisión y algunos otros criterios. Cada minero empieza el proceso de minar un nuevo bloque de transacciones tan pronto como recibe el bloque anterior desde la red, sabiendo que que ha perdido la anterior ronda de competición. Inmediatamente crea un nuevo bloque, y lo rellena con transacciones y la huella digital del anterior bloque, y empieza calculando la prueba de trabajo del nuevo bloque. Cada minero incluye una transacción especial en su bloque, una que paga a su propia dirección bitcoin una recompensa de bitcoins recién creados (actualmente 25 BTC por bloque). Si encuentra una solución que haga ese bloque válido, "gana" esa recompensa porque su bloque exitoso es añadido a la cadena de bloques y la transacción de recompensa se convierte en gastable. Jing, que participa en la agrupación de minado, ha configurado su software para crear nuevos bloques que asignen la recompensa a la dirección de la agrupación. Desde ahí, una parte de la recompensa se distribuye a Jing y otros mineros en proporción a la cantidad de trabajo con que hayan contribuido en la última ronda.

La transacción de Alice fue recogida por la red e incluida en la agrupación de transacciones no verificadas. Debido a que tenía suficientes comisiones, fue incluida en un nuevo bloque generado por la agrupación minera de Jing. Aproximadamente cinco minutos después de que la transacción fuera transmitida por el monedero de Alice, el minero ASIC de Jing encontró la solución para el bloque y lo publicó como bloque #277316 en la red bitcoin, conteniendo otras 419 transacciones más. El minero ASIC de Jing publicó el nuevo bloque en la red bitcoin, donde otros mineros lo validaron y empezaron de nuevo la carrera por generar el siguiente bloque.

Puede ver el bloque que incluye la transacción de Alice

Unos minutos más tarde, un nuevo bloque, #277317, es minado por otro minero. Debido a que este nuevo bloque está basado en el bloque previo (#277316) que contiene la transacción de Alice, añade aún más computación sobre ese bloque, y por tanto fortalece la confianza en esas transacciones. El bloque que contiene la transacción de Alice se cuenta como una "confirmación" de esa transacción. Cada bloque minado sobre el que contiene la transacción es una confirmación adicional. Cuando los bloques se apilan unos encima de otros, se vuelve exponencialmente más difícil deshacer la transacción, con lo que se hace más y más confiable por la red.

En el diagrama en Transacción de Alice incluida en el bloque #277316 podemos ver el bloque #277316,

que contiene la transacción de Alice. Bajo ella hay 277.316 bloques (incluyendo el bloque #0), unido uno a otro en una cadena de bloques (blockchain) todo el camino atrás hasta el bloque #0, conocido como *bloque génesis*. Con el tiempo, a medida que la "altura" en los bloques aumenta, también lo hace la dificultad de cómputo para cada bloque y la cadena en su conjunto. Los bloques minados después del que contiene la transacción de Alice actúan como mayor garantía, a medida que acumulan más computación en una cadena más y más larga. Por convención, cualquier bloque con más de seis confirmaciones se considera irrevocable, porque se requiere una inmensa cantidad de cálculo computacional para invalidar y recalcular seis bloques. Examinaremos ese proceso de minado y la manera en que genera confianza en más detalle en [ch8].

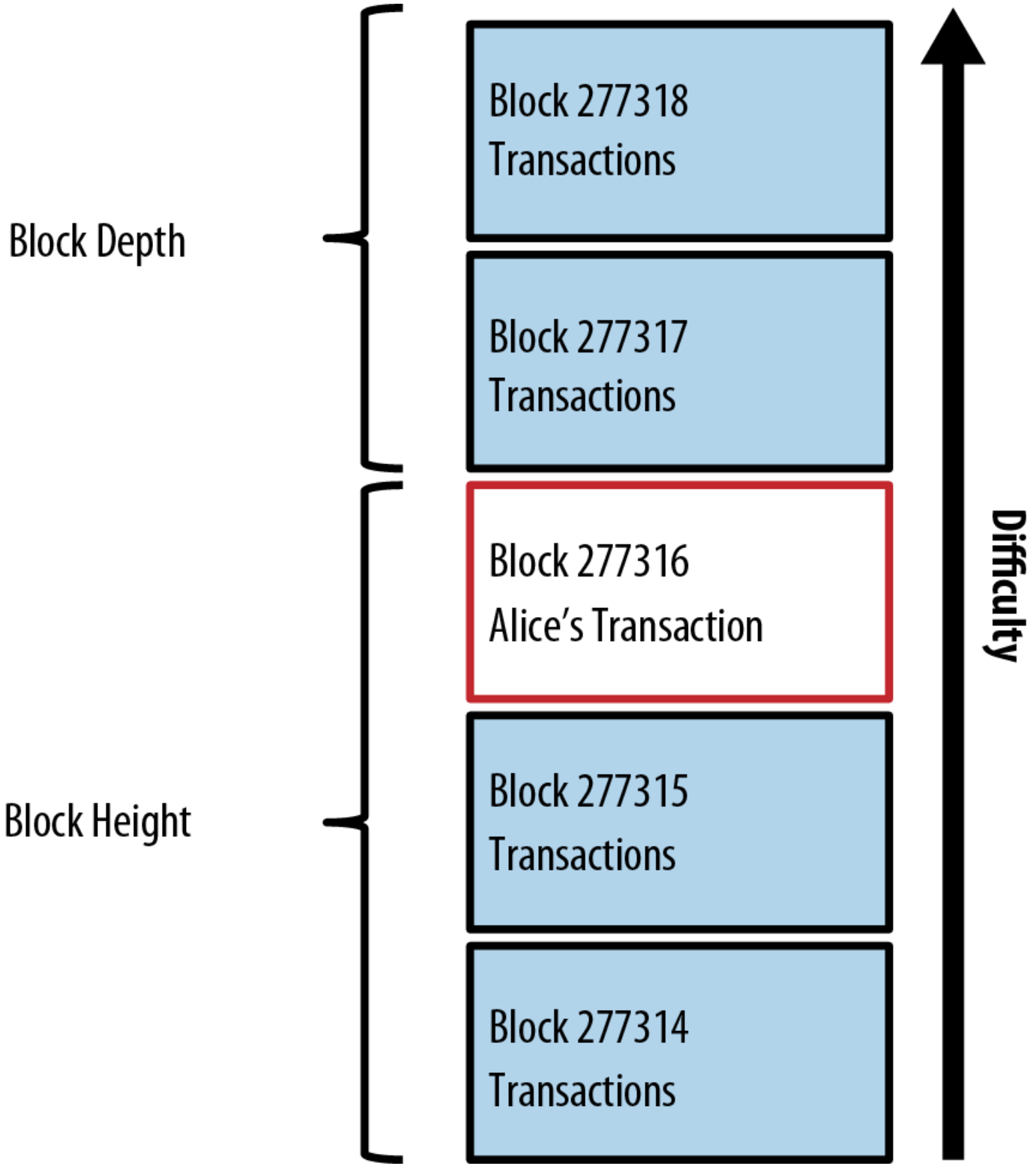


Figure 9. Transacción de Alice incluida en el bloque #277316

Gastando la Transacción

Ahora que la transacción de Alice ha sido incluida en la cadena de bloques como parte de un bloque, es parte del libro contable distribuido y es visible para todas las aplicaciones bitcoin. Cada cliente de bitcoin puede verificar independientemente la transacción como válida y gastable. Los clientes con el

índice completo pueden rastrear el origen de los fondos desde el momento en que fueron generados en un bloque, de transacción en transacción, hasta que alcanzan la dirección de Bob. Los clientes ligeros pueden hacer lo que se llama una verificación de pago simplificada (ver [spv_nodes]) confirmando que la transacción está en la cadena de bloques y que tiene unos cuantos bloques minados después de ella, y por tanto asegurando que la red la acepta como válida.

Bob puede ahora gastar la salida de esta y otras transacciones, creando su propia transacción que haga referencia a esas salidas como sus entradas y asignándolas a un nuevo propietario. Por ejemplo, Bob puede pagar a los proveedores transfiriendo valor desde la taza de café de Alice a estos nuevos propietarios. Seguramente, el software bitcoin de Bob agregará muchos pequeños pagos en un pago más grande, tal vez concentrando las ganancias en bitcoin de todo un día en una sola transacción. Esto movería varios pagos a una única dirección, que se usaría como la cuenta general de compras del comercio. Para un diagrama de una transacción agregadora, ver Transacciones de agregación de fondos.

Una vez que Bob gasta los pagos recibidos desde Alice y otros clientes, extiende la cadena de transacciones, que a su vez son añadidas al libro contable global llamado blockchain o cadena de bloques, que todos pueden ver y confiar. Asumamos que Bob paga a su diseñador web Gopesh en Bangalore por una nueva página web. Ahora la cadena de transacciones se verá como La transacción de Alice como parte de una cadena de transacciones desde Joe a Gopesh.

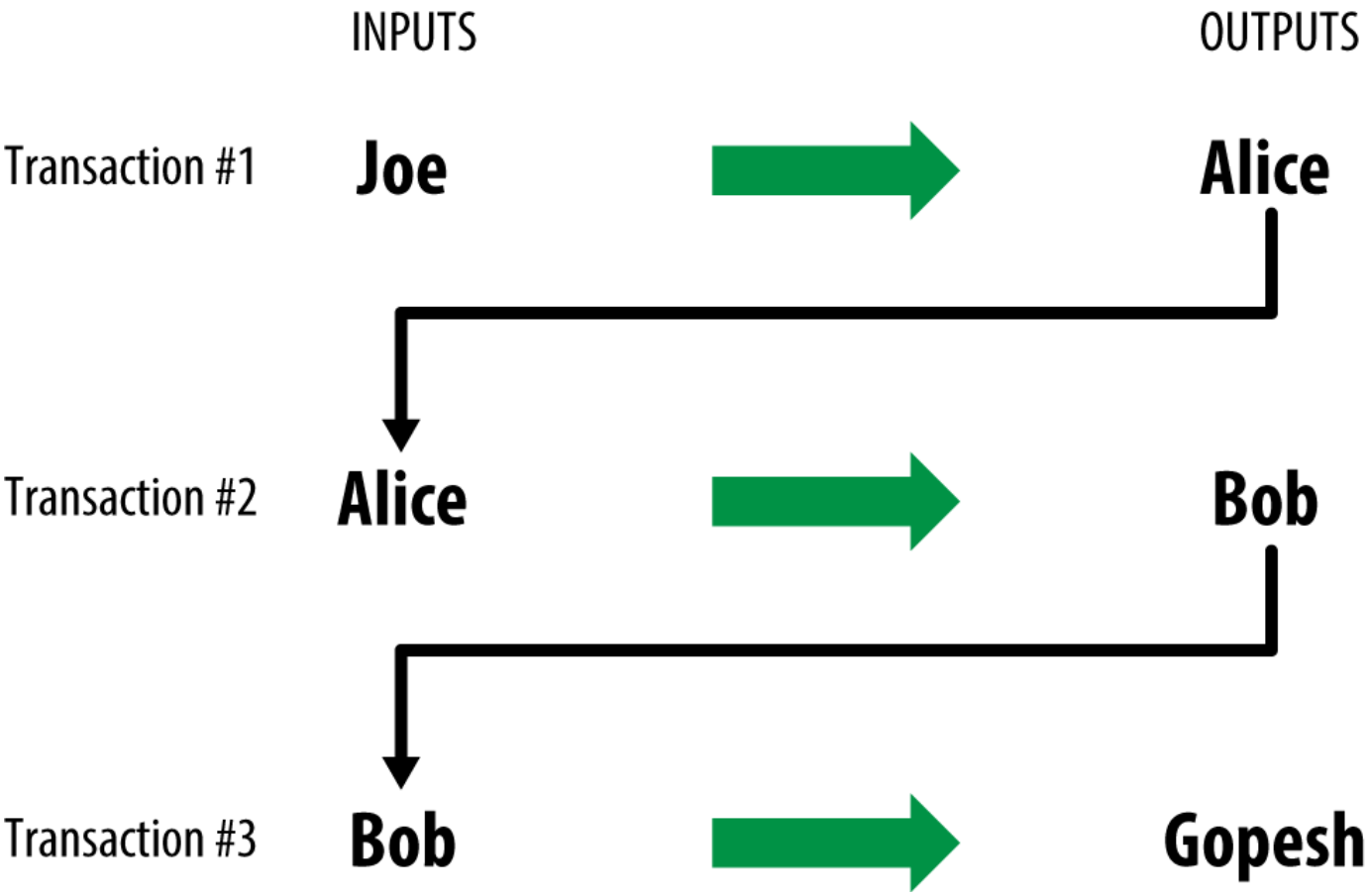


Figure 10. La transacción de Alice como parte de una cadena de transacciones desde Joe a Gopesh

El Cliente Bitcoin

El Núcleo de Bitcoin: La Implementación de Referencia

Puede descargar el cliente de referencia *Bitcoin Core*, también conocido como "cliente Satoshi" desde bitcoin.org. Este cliente implementa todos los aspectos del sistema bitcoin, incluyendo carteras, un motor de verificación de transacciones con una copia completa del histórico de transacciones (blockchain, la cadena de bloques), y un nodo completo de la red peer-to-peer bitcoin.

En la página Elija Su Cartera Bitcoin, seleccione Bitcoin Core para descargar el cliente de referencia. Dependiendo de su sistema operativo, descargará un instalador ejecutable. Para Windows, esto es o bien un archivo ZIP o un archivo ejecutable .exe. Para Mac OS es una imagen de disco .dmg. Las versiones de Linux incluyen un paquete PPA para Ubuntu o un archivo tar.gz. La página bitcoin.org muestra una lista de clientes bitcoin recomendados en el desplegable [bitcoin-choose-client].

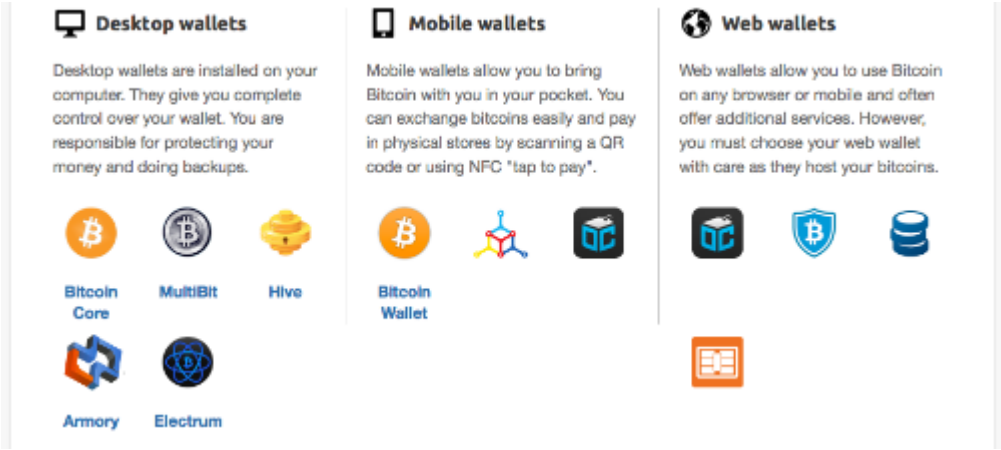


Figure 1. Eligiendo un cliente bitcoin en bitcoin.org

Ejecutando Bitcoin Core por Primera Vez

Si usted descarga un paquete instalable, como un archivo .exe, .dmg o PPA, puede instalarlo de la misma forma que cualquier aplicación de tu sistema operativo. En Windows, ejecute el archivo .exe y siga las instrucciones paso a paso. Para Mac OS, ejecute el archivo .dmg y arrastre el icono de Bitcoin-QT en su carpeta de *Aplicaciones*. Para Ubuntu, haga doble clic en el fichero PPA en el Explorador de archivos y abrirá el gestor de paquetes para instalar el paquete. Una vez que se haya completado la instalación debe tener una nueva aplicación llamada Bitcoin-Qt en su lista de aplicaciones. Haga doble clic en el icono para iniciar el cliente de Bitcoin.

La primera vez que ejecute Bitcoin Core se iniciará la descarga de la cadena de bloques, un proceso que podría tardar varios días (ver << bitcoin-qt-firstload >>). Deje que se ejecute en segundo plano hasta que aparezca "sincronizada" y no muestre más "fuera de sincronización" al lado del saldo.

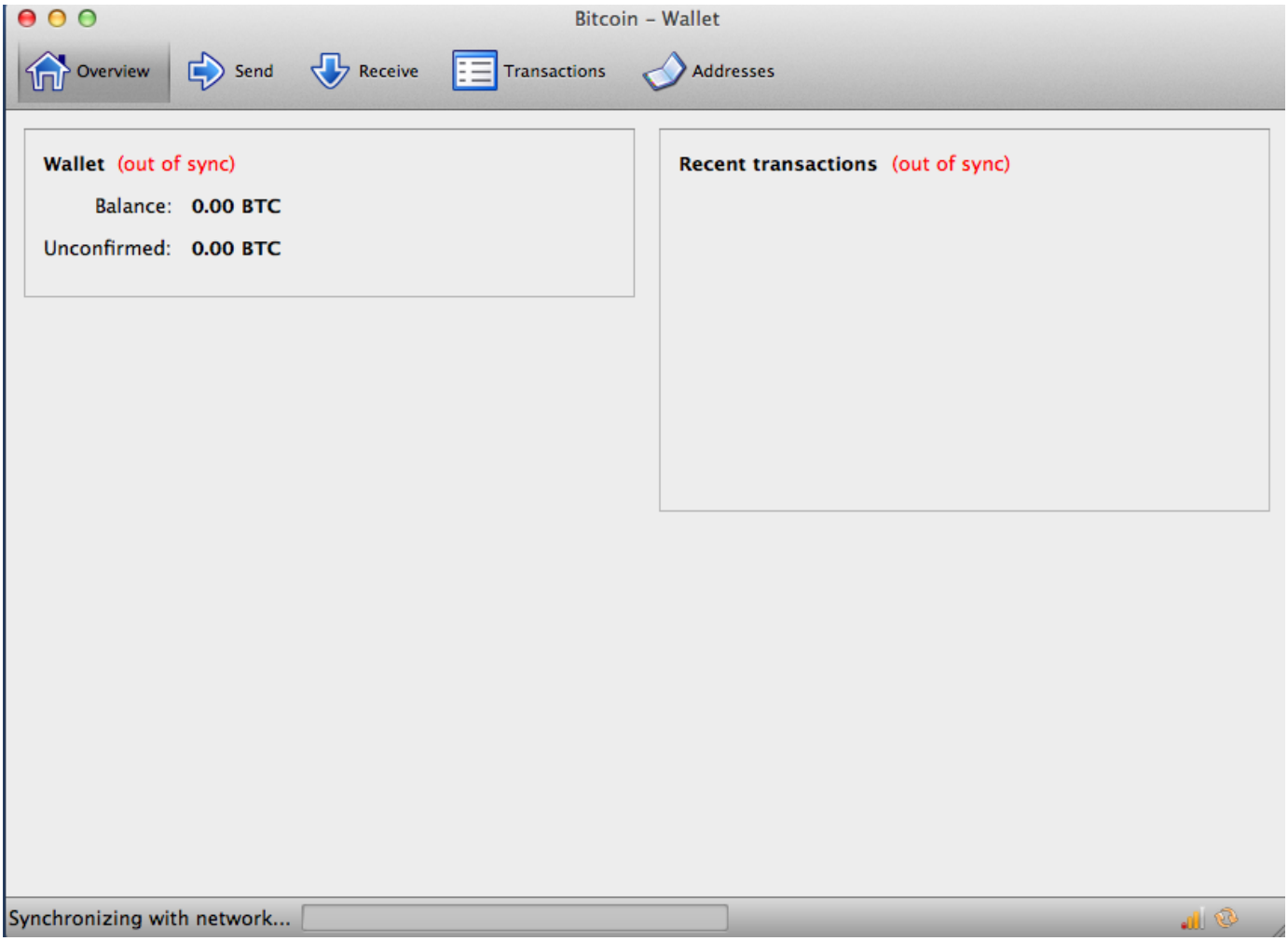


Figure 2. Bitcoin Core durante la inicialización de la cadena de bloques

TIP

Bitcoin Core mantiene una copia completa del histórico de transacciones de la red bitcoin (cadena de bloques), con cada transacción que haya ocurrido en la red Bitcoin desde su creación en 2009. Este conjunto de datos es de varios gigabytes de tamaño (aproximadamente 16 GB a finales de 2013) y se descarga gradualmente durante varios días. El cliente no será capaz de procesar transacciones o actualizar los saldos de cuenta hasta que se descargue el conjunto de datos completo perteneciente a la cadena de bloques. Durante ese tiempo, el cliente mostrará "fuera de sincronización" al lado de los saldos de las cuentas y mostrará "Sincronizando" en el pie de página. Asegúrese de que tiene suficiente espacio en disco, ancho de banda, y tiempo para completar la sincronización inicial.

Compilando Bitcoin Core desde el Código Fuente.

Para los desarrolladores, también existe la opción de descargar el código fuente completo como un archivo ZIP o clonando el repositorio fuente mediante GitHub. En la página bitcoin de GitHub, seleccione Descargar ZIP de la barra lateral. También puede utilizar la línea de comandos git para crear una copia local del código fuente en su sistema. En el siguiente ejemplo, clonamos el código fuente desde una línea de comandos Unix, Linux o Mac OS:

```
$ git clone https://github.com/bitcoin/bitcoin.git
Clonando en 'bitcoin'...
remote: Contando objetos: 31864, hecho.
remote: Comprimiendo objetos: 100% (12007/12007), hecho.
remote: Total 31864 (delta 24480), reutilizados 26530 (delta 19621)
Recibiendo objetos: 100% (31864/31864), 18.47 MiB | 119 KiB/s, hecho.
Resolviendo deltas: 100% (24480/24480), hecho.
$
```

TIP Las instrucciones y salida resultante puede variar de una versión a otra. Siga la documentación que viene con el código, incluso si se diferencia de las instrucciones que usted ve aquí, y no se sorprenda si la salida que se muestra en la pantalla es ligeramente diferente de la de los ejemplos aquí.

Cuando la operación de clonación git ha completado, tendrá una copia local completa del repositorio de código fuente en el directorio *bitcoin*. Cambie a este directorio tecleando cd bitcoin en el símbolo de sistema:

```
$ cd bitcoin
```

Por defecto, la copia local se sincronizará con el código más reciente, que podría ser una versión inestable o beta de bitcoin. Antes de compilar el código, seleccione una versión específica comprobando las *etiquetas* de versión. Esto sincronizará la copia local con un snapshot específico del repositorio de código identificado por una etiqueta de palabra clave. Las etiquetas se utilizan por los desarrolladores para marcar versiones específicas del código por número de versión. En primer lugar, para encontrar las etiquetas disponibles, utilizamos el comando git tag:

```
$ git tag
v0.1.5
v0.1.6test1
v0.2.0
v0.2.10
v0.2.11
v0.2.12

[... muchas más etiquetas ...]

v0.8.4rc2
v0.8.5
v0.8.6
v0.8.6rc1
v0.9.0rc1
```

La lista de etiquetas muestra todas las versiones publicadas de bitcoin. Por convención, los *candidatos a versión*, que están destinados a pruebas, tienen el sufijo "rc" (del inglés, "release candidate", "candidatos a versión"). Las versiones estables que se pueden ejecutar en producción no tienen sufijo. De la lista anterior, seleccione la de mayor número, que en el momento de escribir este libro es la v0.9.0rc1. Para sincronizar el código local con esta versión, utilice el comando git checkout:

```
$ git checkout v0.9.0rc1
Nota: comprobando 'v0.9.0rc1'.

HEAD está ahora en 15ec451... Combinando la petición de extracción #3605
$
```

El código fuente incluye la documentación, que se puede encontrar en un conjunto de archivos. Revise la documentación principal ubicada en *README.md* en el directorio bitcoin escribiendo more README.md en el símbolo de sistema y use la barra espaciadora para pasar a la página siguiente. En este capítulo, vamos a construir el cliente bitcoin desde línea de comandos, también conocido como bitcoind en Linux. Revise las instrucciones para compilar el cliente bitcoind desde línea de comandos en su plataforma escribiendo more doc/build-unix.md. Se pueden encontrar instrucciones alternativas para Mac OS X y Windows en el directorio *doc*, como *build-osx.md* o *build-msw.md*, respectivamente.

Revise cuidadosamente los prerrequisitos para hacer la construcción, que están en la primera parte de la documentación. Esas son las bibliotecas que deben estar presentes en su sistema antes de que pueda comenzar a compilar bitcoin. Si estos requisitos previos no se hicieran, la construcción dará un error. Si esto sucede porque no se ha hecho un requisito previo, puede instalarlo y luego reanudar el proceso de construcción desde donde lo dejó. Suponiendo que se instalaron los requisitos previos, puede iniciar el "build" mediante la generación de un conjunto de scripts de construcción mediante el script *autogen.sh*.

TIP El proceso de construcción de Bitcoin Core fue modificado para utilizar el sistema autogeneración / configuración / make a partir de la versión 0.9. Las versiones más antiguas utilizan un sencillo Makefile y funcionan de forma ligeramente diferente al siguiente ejemplo. Siga las instrucciones para la versión que desea compilar. El sistema autogeneración / configuración / make introducido en la versión 0.9 es probable que sea el sistema de construcción utilizado para todas las futuras versiones del código y se muestra en los siguientes ejemplos.

```
$ ./autogen.sh
configure.ac:12: instalando `src/build-aux/config.guess'
configure.ac:12: instalando `src/build-aux/config.sub'
configure.ac:37: instalando `src/build-aux/install-sh'
configure.ac:37: instalando `src/build-aux/missing'
src/Makefile.am: instalando `src/build-aux/depcomp'
$
```

El script *autogen.sh* crea un conjunto de scripts de configuración automática que recogen información de su sistema para descubrir los ajustes correctos y asegurarse de que tiene todas las librerías necesarias para compilar el código. El más importante de ellos es el configure script que ofrece diferentes opciones para personalizar el proceso de construcción. Escriba ./configure --help para ver las distintas opciones:

```
$ ./configure --help

`configure 'configura Bitcoin Core 0.9.0 para adaptarse a muchos tipos de sistemas.

Uso: ./configure [OPTION]... [VAR=VALUE]...

Para asignar variables de entorno (por ejemplo, CC, CFLAGS ...), especifíquelas como
VAR=VALUE. Vea a continuación las descripciones de algunas de las variables útiles.

Los valores por defecto para las opciones están entre corchetes

Configuración:
  -h --help muestra esta ayuda y sale
      --help=short muestra las opciones específicas de este paquete
      --help=recursive muestra la ayuda corta de todos los paquetes incluidos
  -V, --version muestra la información de la versión y sale

[... muchas más opciones y variables se muestran a continuación ...]

Funcionalidades opcionales:
  --disable-option-checking ignora lo no reconocido --enable/--with opciones
  --disable-CARACTERISTICA no incluye CARACTERISTICA (es lo mismo que --enable
-FEATURE=no)
  --enable-CARACTERISTICA[=ARGUMENTO] incluye CARACTERISTICA [ARGUMENTO=yes]

[... más opciones ...]

Use esas variables para invalidar las elecciones hechas por `configure' o para ayudarlo
a encontrar librerías y programas de nombres o rutas no estándar

Reportar errores a <info@bitcoin.org>.

$
```

El script configure le permite activar o desactivar ciertas características de bitcoind a través del uso de las opciones --enable-CARACTERISTICA y --disable-CARACTERISTICA , donde CARACTERISTICA es el nombre de la función, escrita como se muestra en la ayuda . En este capítulo, vamos a construir el cliente bitcoind con todas las características predeterminadas. No vamos a estar utilizando las opciones de configuración, pero debe revisarlos para entender las características opcionales que son parte del cliente. A continuación, ejecute el script configure para detectar automáticamente todas las

librerías necesarias y crear un script de construcción personalizado para su sistema:

```
$ ./configure
comprobando tipo de sistema de construcción... x86_64-unknown-linux-gnu
comprobando tipo de sistema de host... x86_64-unknown-linux-gnu
comprobando compatibilidad con BSD... /usr/bin/install -c
comprobando si el entorno de construcción es adecuado... yes
comprobando que mkdir -p es thread-safe... /bin/mkdir -p
comprobando si existiera gawk... no
comprobando si existiera mawk... mawk
comprobando si make configura $(MAKE)... yes

[... muchas más comprobaciones de caracerísticas son probadas ...]

configure: creando ./config.status
config.status: creando Makefile
config.status: creando src/Makefile
config.status:  creando src/test/Makefile
config.status:  creando src/qt/Makefile
config.status:  creando src/qt/test/Makefile
config.status:  creando share/setup.nsi
config.status:  creando share/qt/Info.plist
config.status:  creando qa/pull-tester/run-bitcoind-for-test.sh
config.status:  creando qa/pull-tester/build-tests.sh
config.status: creating src/bitcoin-config.h
config.status: executing depfiles commands
$
```

Si todo sale bien, el comando configure creará scripts de compilación personalizados que nos permitirán compilar bitcoind. Si hay bibliotecas faltantes o errores, el comando configure terminará con error en vez de crear los scripts de compilación. Si ocurre un error, es muy probable que se deba a una biblioteca faltante o incompatible. Relea la documentación de compilación y asegúrese de instalar los requisitos faltantes. Luego ejecute configure nuevamente y vea si eso corrige el error. Luego compilará el código fuente, un proceso que puede tardar hasta una hora en terminar. Durante el proceso de compilación verá mensajes de salida cada pocos segundos o minutos, o un error si algo sale mal. El proceso de compilación puede completarse en cualquier momento si es interrumpido. Teclee make para comenzar a compilar:

```
$ make
Making all in src
make[1]: Entering directory `/home/ubuntu/bitcoin/src'
make  all-recursive
make[2]: Entering directory `/home/ubuntu/bitcoin/src'
Making all in .
make[3]: Entering directory `/home/ubuntu/bitcoin/src'
  CXX    addrman.o
  CXX    alert.o
  CXX    rpcserver.o
  CXX    bloom.o
  CXX    chainparams.o

[... siguen muchos otros mensajes de compilación ...]

  CXX    test_bitcoin-wallet_tests.o
  CXX    test_bitcoin-rpc_wallet_tests.o
  CXXLD  test_bitcoin
make[4]: Leaving directory `/home/ubuntu/bitcoin/src/test'
make[3]: Leaving directory `/home/ubuntu/bitcoin/src/test'
make[2]: Leaving directory `/home/ubuntu/bitcoin/src'
make[1]: Leaving directory `/home/ubuntu/bitcoin/src'
make[1]: Entering directory `/home/ubuntu/bitcoin'
make[1]: Nothing to be done for `all-am'.
make[1]: Leaving directory `/home/ubuntu/bitcoin'
$
```

Si todo sale bien bitcoind se encuentra ahora compilado. El paso final es instalar el ejecutable de bitcoind en la ruta del sistema usando el comando make:

```
$ sudo make install
Making install in src
Making install in .
 /bin/mkdir -p '/usr/local/bin'
  /usr/bin/install -c bitcoind bitcoin-cli '/usr/local/bin'
Making install in test
make  install-am
 /bin/mkdir -p '/usr/local/bin'
  /usr/bin/install -c test_bitcoin '/usr/local/bin'
$
```

Puede confirmar que bitcoin se encuentra correctamente instalado preguntándole al sistema por la ruta a los dos ejecutables, de esta forma:

```
$ which bitcoind
/usr/local/bin/bitcoind

$ which bitcoin-cli
/usr/local/bin/bitcoin-cli
```

La instalación por defecto de bitcoind lo coloca en */usr/local/bin*. Cuando ejecute bitcoind por primera vez le recordará que debe crear un archivo de configuración con una contraseña fuerte para la interafaz JSON-RPC. Ejecute bitcoind tecleando bitcoind en la terminal:

```
$ bitcoind
Error: Para usar la opción "-server" debe establecer un valor rpcpassword en el archivo
de configuración:
/home/ubuntu/.bitcoin/bitcoin.conf
Se recomienda utilizar la siguiente contraseña aleatoria:
rpcuser=bitcoinrpc
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
(no es necesario recordar esta contraseña)
El nombre de usuario y la contraseña DEBEN NO ser iguales.
Si el archivo no existe, créelo con permisos de archivo de solo lectura.
Se recomienda también establecer alertnotify para recibir notificaciones de problemas.
Por ejemplo: alertnotify=echo %s | mail -s "Bitcoin Alert" admin@foo.com
```

Edite el archivo de configuración en su editor preferido y establezca los parámetros, reemplazando la contraseña con una contraseña fuerte tal como lo recomienda bitcoind. *No* use el password que se muestra ahí. Cree un archivo dentro del directorio *.bitcoin* llamado *.bitcoin/bitcoin.conf* e ingrese un usuario y password:

```
rpcuser=bitcoinrpc
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
```

Mientras edite este archivo de configuración puede que quiera establecer algunas otras opciones, tales como txindex (ver [txindex]). Para una lista completa de las opciones disponibles teclee bitcoind --help.

Ahora ejecute el cliente Bitcoin Core. La primera vez que lo ejecute reconstruirá la cadena de bloques descargando todos los bloques. Este es un archivo de varios gigabytes y tardará en promedio dos días en ser descargado por completo. Puede acortar los tiempos de inicialización de la cadena de bloques descargando una copia parcial de la cadena de bloques usando un cliente BitTorrent de SourceForge.

Ejecute bitcoind en segundo plano con la opción -daemon:

```
$ bitcoind -daemon

Bitcoin versión v0.9.0rc1-beta (2014-01-31 09:30:15 +0100)
Utilizando la versión de OpenSSL 1.0.1c 10 May 2012
Directorio de datos predeterminado /home/bitcoin/.bitcoin
Utilizando directorio de datos /bitcoin/
Usando como máximo 4 conexiones (1024 descriptores de archivos disponibles)
init message: Verificando monedero...
dbenv.open LogDir=/bitcoin/database ErrorFile=/bitcoin/db.log
Bound to [::]:8333
Bound to 0.0.0.0:8333
init message: Cargando el índice de bloques..
Abriendo LevelDB en /bitcoin/blocks/index
LevelDB abierto satisfactoriamente
Abriendo LevelDB en /bitcoin/chainstate
LevelDB abierto satisfactoriamente

[... más mensajes de inicialización ...]
```

Utilizando la API JSON-RPC de Bitcoin Core desde la Línea de Comandos

El cliente Bitcoin Core implementa una interfaz JSON-RPC que puede ser accedida también usando la herramienta de línea de comando bitcoin-cli. La línea de comandos nos permite experimentar interactivamente con las capacidades disponibles también programáticamente a través de la API. Para comenzar invoque el comando help para ver la lista de comandos disponibles del RPC de bitcoin.

```
$ bitcoin-cli help
addmultisigaddress nrequired ["clave",...] ( "cuenta" )
addnode "nodo" "add|remove|onetry"
backupwallet "destino"
createmultisig nrequired ["clave", ...]
createrawtransaction [{"txid":"id","vout":n},...] {"dirección":monto,...}
decoderawtransaction "cadenahex"
decodescript "hex"
dumpprivkey "direcciónbitcoin"
dumpwallet "nombredearchivo"
getaccount "direcciónbitcoin"
getaccountaddress "cuenta"
getaddednodeinfo dns ( "nodo" )
getaddressesbyaccount "cuenta"
getbalance ( "cuenta" minconf )
getbestblockhash
getblock "hash" ( verbose )
```

```
getblockchaininfo
getblockcount
getblockhash index
getblocktemplate ( "objetodepeticiónjson" )
getconnectioncount
getdifficulty
getgenerate
gethashespersec
getinfo
getmininginfo
getnettotals
getnetworkhashps ( altura de bloques )
getnetworkinfo
getnewaddress ( "cuenta" )
getpeerinfo
getrawchangeaddress
getrawmempool ( verbose )
getrawtransaction "txid" ( verbose )
getreceivedbyaccount "cuenta" ( minconf )
getreceivedbyaddress "direcciónbitcoi" ( minconf )
gettransaction "txid"
gettxout "txid" n ( includemempool )
gettxoutsetinfo
getunconfirmedbalance
getwalletinfo
getwork ( "datos" )
help ( "comando" )
importprivkey "claveprivadabitcoin" ( "etiqueta" rescan )
importwallet "nombredearchivo"
keypoolrefill ( tamañonuevo )
listaccounts ( minconf )
listaddressgroupings
listlockunspent
listreceivedbyaccount ( minconf includeempty )
listreceivedbyaddress ( minconf includeempty )
listsinceblock ( "hashdebloque" target-confirmations )
listtransactions ( "cuenta" conteo desde )
listunspent ( minconf maxconf ["dirección",...] )
lockunspent unlock [{"txid":"txid","vout":n},...]
move "desdecuenta" "haciacuenta" amount ( minconf "comentario" )
ping
sendfrom "desdecuenta" "haciadirecciónbitcoin" monto ( minconf "comentario" "comentario-
a" )
sendmany "desdecuenta" {"dirección":monto,...} ( minconf "comentario" )
sendrawtransaction "cadenahex" ( allowhighfees )
sendtoaddress "dirección bitcoin" cantidad ( "comentario" "comentario-para" )
setaccount "direcciónbitcoin" "cuenta"
setgenerate generate ( genproclimit )
```

```
settxfee amount
signmessage "direcciónbitcoin" "mensaje"
signrawtransaction "cadenahexadecimal" (
[{"txid":"id","vout":n,"scriptPubKey":"hex","redeemScript":"hex"},...]
["claveprivada1",...] tipodehashdefirma )
stop
submitblock "datoshexadecimales" ( "objetodeparámetrosjson" )
validateaddress "direcciónbitcoin"
verifychain ( niveldechequeo númerodebloques )
verifymessage "direcciónbitcoin" "firma" "mensaje"
walletlock
walletpassphrase "frasesecreta" timeout
walletpassphrasechange "oldpassphrase" "newpassphrase"
```

Obteniendo Información del Estado del Cliente Bitcoin Core

Comandos: getinfo

El comando RPC de bitcoin getinfo muestra información básica sobre el estado del nodo de la red bitcoin, la cartera, y la base de datos de la cadena de bloques. Use bitcoin-cli para ejecutarlo:

```
$ bitcoin-cli getinfo
```

```
{
    "version" : 90000,
    "protocolversion" : 70002,
    "walletversion" : 60000,
    "balance" : 0.00000000,
    "blocks" : 286216,
    "timeoffset" : -72,
    "connections" : 4,
    "proxy" : "",
    "difficulty" : 2621404453.06461525,
    "testnet" : false,
    "keypoololdest" : 1374553827,
    "keypoolsize" : 101,
    "paytxfee" : 0.00000000,
    "errors" : ""
}
```

Los datos se devuelven en JavaScript Object Notation (JSON), un formato que puede ser fácilmente "consumido" por todos los lenguajes de programación, es también bastante legible por humanos. Entre estos datos vemos el número de versión del cliente bitcoin (90000), protocolo (70002) y monedero (60000). Vemos el balance actual contenido en el monedero, que es cero. Vemos la altura del bloque

actual, mostrándonos cuántos bloques son conocidos por el cliente (286216). Vemos también varias estadísticas sobre la red bitcoin y las configuraciones relacionadas con el cliente. Exploraremos estas configuraciones en más detalle en el resto del capítulo.

TIP	Llevará bastante tiempo, seguramente más de un día, al cliente bitcoind llegar a la altura actual de la cadena de bloques cuando descargue bloques de otros clientes bitcoin. Puede ver el progreso utilizando getinfo para ver el número de bloques conocidos.

Encriptación y Configuración de la Cartera

Comandos: encryptwallet, walletpassphrase

Antes de proceder a crear claves y otros comandos, debe encriptar la cartera con una contraseña. Para este ejemplo, se usará el comando encryptwallet con la contraseña "foo". Por supuesto, ¡cambie "foo" por otra contraseña más fuerte y compleja!

```
$ bitcoin-cli encryptwallet foo
wallet encrypted; Bitcoin server stopping, restart to run with encrypted wallet. The
keypool has been flushed, you need to make a new backup.
$
```

Puede verificar que la cartera ha sido encriptada ejecutando getinfo de nuevo. Esta vez verá una nueva entrada llamada unlocked_until. Es un contador que muestra cuánto tiempo estará en la memoria la contraseña de desencriptado, manteniendo la cartera desbloqueada. Al principio este tiempo será de cero, lo que significa que el monedero está bloqueado.

```
$ bitcoin-cli getinfo
```

```
{
    "version" : 90000,

#[... other information...]

    "unlocked_until" : 0,
    "errors" : ""
}
$
```

Para desbloquear el monedero, lance el comando walletpassphrase, que tiene dos parámetros—la contraseña y un número de segundos hasta que el monedero vuelva a ser bloqueado automáticamente (una cuenta atrás):

```
$ bitcoin-cli walletpassphrase foo 360
$
```

Puede confirmar que el monedero está desbloqueado y ver el tiempo que queda ejecutando getinfo de nuevo.

```
$ bitcoin-cli getinfo
```

```
{
    "version" : 90000,

#[... other information ...]

    "unlocked_until" : 1392580909,
    "errors" : ""
}
```

Copia de Seguridad del Monedero, Volcado de Texto Plano y Restauración.

Comandos: backupwallet, importwallet, dumpwallet

A continuación, practicaremos creando un archivo de copia de seguridad y restauraremos la cartera desde la copia de seguridad. Utilice el comando backupwallet para hacer la copia, proporcionando el nombre de archivo como parámetro. Aquí hacemos la copia de seguridad al archivo *wallet.backup*:

```
$ bitcoin-cli backupwallet wallet.backup
$
```

Ahora, para restaurar la copia de seguridad, usamos el comando importwallet. Si su cartera está bloqueada, necesitará desbloquearla primero (ver walletpassphrase en la anterior sección) para importar la copia de seguridad:

```
$ bitcoin-cli importwallet wallet.backup
$
```

El comando dumpwallet puede ser usado para volcar el archivo en un archivo de texto legible por humanos:

```
$ bitcoin-cli dumpwallet wallet.txt
$ more wallet.txt
# Wallet dump created by Bitcoin v0.9.0rc1-beta (2014-01-31 09:30:15 +0100)
# * Created on 2014-02- 8dT20:34:55Z
# * Best block at time of backup was 286234
(000000000000000f74f0bc9d3c186267bc45c7b91c49a0386538ac24c0d3a44),
#   mined on 2014-02- 8dT20:24:01Z

KzTg2wn6Z8s7ai5NA9MVX4vstHRsqP26QKJCzLg4JvFrp6mMaGB9 2013-07- 4dT04:30:27Z change=1 #
addr=16pJ6XkwSQv5ma5FSXMRPaXEYrENCEg47F
Kz3dVz7R6mUpXzdZy4gJEVZxXJwA15f198eVui4CUivXotzLBDKY 2013-07- 4dT04:30:27Z change=1 #
addr=17oJds8kaN8LP8kuAkWTco6ZM7BGXFC3gk
[... many more keys ...]

$
```

Direcciones de Cartera y Recepción de Transacciones

Comandos: getnewaddress, getreceivedbyaddress, listtransactions, getaddressesbyaccount, getbalance

El cliente bitcoin de referencia mantiene una agrupación de direcciones, el tamaño del cual se muestra en keypoolsize cuando usa el comando getinfo. Estas direcciones se generan automáticamente y pueden luego usarse como direcciones públicas de recepción o direcciones de cambio. Para conseguir una de estas direcciones, use el comando getnewaddress:

```
$ bitcoin-cli getnewaddress
1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL
```

Ahora podemos usar esta dirección para enviar una pequeña cantidad de bitcoin a nuestra cartera bitcoind desde una cartera externa (asumiendo que usted tiene algunos bitcoins guardados en una casa de cambio, cartera web u otra cartera bitcoind en otro ordenador). Para este ejemplo, enviaremos 50 milibits (0.050 bitcoin) a la dirección anterior.

Ahora consultaremos al cliente bitcoind por la cantidad recibida por esta dirección, y especificaremos cuántas confirmaciones se requieren antes de que la cantidad se sume al saldo. Para este ejemplo, especificaremos cero confirmaciones. Unos segundos más tarde de enviar los bitcoins desde la otra cartera, lo veremos reflejado en la cartera. Usaremos getreceivedbyaddress con la dirección y el número de confirmaciones configurado a cero (0):

```
$ bitcoin-cli getreceivedbyaddress 1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL 0
0.05000000
```

Si omitimos el cero del final de este comando, veremos solo las cantidades que tienen al menos

minconf confirmaciones, donde minconf es la configuración para el mínimo número de confirmaciones antes de que una transacción sea listada en el saldo. La configuración minconf se especifica en el archivo de configuración de bitcoind. Debido a que esta transacción fue enviada en los últimos segundos, aún no tiene confirmaciones y por tanto veremos listado un saldo de cero.

```
$ bitcoin-cli getreceivedbyaddress 1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL
0.00000000
```

Las transacciones recibidas por el monedero completo pueden verse usando el comando listtransactions:

```
$ bitcoin-cli listtransactions
```

```
[
    {
        "account" : "",
        "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
        "category" : "receive",
        "amount" : 0.05000000,
        "confirmations" : 0,
        "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
        "time" : 1392660908,
        "timereceived" : 1392660908
    }
]
```

Podemos listar todas las direcciones del monedero usando el comando getaddressesbyaccount:

```
$ bitcoin-cli getaddressesbyaccount ""
```

```
[
    "1LQoTPYy1TyERbNV4zZbhEmgyfAipC6eqL",
    "17vrg8uwMQUibkvS2ECRX4zpcVJ78iFaZS",
    "1FvRHWhHBBZA8cGRRsGiAeqEzUmjJkJQWR",
    "1NVJK3JsL41BF1KyxrUyJW5XHjunjfp2jz",
    "14MZqqzCxjc99M5ipsQSRfieT7qPZcM7Df",
    "1BhrGvtKFjTAhGdPGbrEwP3xvFjkJBuFCa",
    "15nem8CX91XtQE8B1Hdv97jE8X44H3DQMT",
    "1Q3q6taTsUiv3mMemEuQQJ9sGLEGaSjo81",
    "1HoSiTg8sb16oE6SrmazQEwcGEv8obv9ns",
    "13fE8BGhBvnoy68yZKuWJ2hheYKovSDjqM",
    "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
    "1KHUmVfCJteJ21LmRXHSpPoe23rXKifAb2",
    "1LqJZz1D9yHxG4cLkdujnqG5jNNGmPeAMD"
]
```

Finalmente, el comando getbalance mostrará el saldo total del monedero, añadiendo todas las transacciones confirmadas con al menos las confirmaciones de minconf:

```
$ bitcoin-cli getbalance
0.05000000
```

TIP	Si la transacción aún no se ha confirmado, el saldo devuelto por getbalance será cero. La opción de configuración "minconf" determina el número mínimo de confirmaciones que es necesario para que una transacción aparezca en el saldo.

Explorando y Decodificando Transacciones

Comandos: gettransaction, getrawtransaction, decoderawtransaction

Ahora exploraremos la transacción entrante que fue listada previamente usando el comando gettransaction. Podemos obtener una transacción por su hash de transacción, mostrado previamente en txid, con el comando gettransaction:

```
{
    "amount" : 0.05000000,
    "confirmations" : 0,
    "txid" : "9ca8f969bd3ef5ec2a8685660fdbf7a8bd365524c2e1fc66c309acbae2c14ae3",
    "time" : 1392660908,
    "timereceived" : 1392660908,
    "details" : [
        {
            "account" : "",
            "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
            "category" : "receive",
            "amount" : 0.05000000
        }
    ]
}
```

TIP

Las ID de transacción no son oficiales hasta que una transacción se confirma. No tener hash de transacción no quiere decir que la transacción no haya sido procesada. Esto se conoce como "maleabilidad de transacciones", porque los hashes de transacción pueden ser modificados anteriormente a la confirmación del bloque. Después de la confirmación, el txid es inmutable y oficial.

La forma de transacción mostrada con el comando gettransaction es la forma simplificada. Para conseguir la transacción completa y descodificarla, usaremos dos comandos: getrawtransaction y decoderawtransaction. Previamente, getrawtransaction lleva el *hash de transacción (txid)* como un parámetro y devuelve la transacción completa como una cadena hexadecimal en bruto, exactamente como existe en la red bitcoin:

Para decodificar esta cadena hexadecimal, usaremos decoderawtransaction. Copiar y pegar el hexadecimal como primer parámetro de decoderawtransaction para obtener el contenido completo interpretado como una estructura de datos JSON (por razones de formato la cadena hexadecimal se acorta en el siguiente ejemplo):

La decodificación de la transacción muestra todos los componentes de la transacción, incluyendo las entradas y salidas de la transacción. En este caso vemos que la transacción que acredita nuestra nueva dirección con 50 milibits usa una entrada (input) y ha generado dos salidas (outputs). La entrada a esta transacción fue la salida de una previa transacción confirmada (mostrada como el vin txid que empieza con d3c7). Las dos salidas corresponden a 50 milibits de crédito y la salida con cambio para el emisor.

Podemos explorar posteriormente la cadena de bloques examinando la transacción previa referida por su txid en esta transacción usando los mismos comandos (e.g., gettransaction). Saltando de transacción en transacción podemos seguir una cadena de transacciones hacia atrás a medida que las monedas se transmiten de la dirección de un propietario a la dirección a otro propietario.

Una vez que la transacción que hemos recibido ha sido confirmada incluyéndose en un bloque, el comando gettransaction devolverá información adicional, mostrando el *hash de bloque (identificador)* en el que se ha incluido la transacción:

Aquí, vemos la nueva información en las entradas blockhash (el hash del bloque en que ha sido incluida la transacción), y blockindex con el valor 18 (indicando que nuestra transacción fue la transacción en la posición 18 de ese bloque).

1. Índice de base de datos de transaccion y opción txindex

> Por defecto, Bitcoin Core construye una base de datos que contiene *solamente* las transacciones relacionadas con la cartera del usuario. Si desea acceder a *cualquier* transacción con comandos como gettransaction, necesita configurar Bitcoin Core para que construya un índice de transacciones completo, lo cual se consigue con la opción txindex . Establezca txindex=1 en el fichero de configuración de Bitcoin Core (normalmente se encuentra en *.bitcoin/bitcoin.conf* del directorio home). Una vez que haya modificado este parámetro, es necesario que reinicie bitcoind y espere a que se reconstruya el índice.

Explorando Bloques

Comandos: getblock, getblockhash

Ahora que sabemos en qué bloque fue incluida nuestra transacción, podemos consultar dicho bloque. Usamos el comando getblock con el hash del bloque como parámetro:

El bloque contiene 367 transacciones y, como puede observar, la transacción listada en la posición 18 (9ca8f9...) es la txid de la que acreedita 50 millibits a nuestra dirección. La altura (height) nos dice que este es el bloque número 286384 en la cadena de bloques.

También podemos recuperar datos de un bloque por su altura usando el comando getblockhash, el cual toma la altura del bloque como parámetro y devuelve el hash de bloque para ese bloque:

Aquí recuperamos el hash de bloque del "bloque génesis", el primer bloque minado por Satoshi Nakamoto, con altura cero. Recuperar este bloque muestra:

Los comandos getblock, getblockhash y gettransaction pueden usarse para explorar la base de datos de la cadena de bloques programáticamente:

Creando, Firmando y Enviando Transacciones Basadas en <phrase role="keep-together">Salidas Sin Gastar</phrase>

Comandos: listunspent, gettxout, createrawtransaction, decoderawtransaction, signrawtransaction, sendrawtransaction

Las transacciones en bitcoin se basan en el concepto de "salidas," las cuales son el resultado de

transacciones previas, para crear una cadena de transacciones que transfiere propiedad de dirección en dirección. Nuestra cartera ahora ha recibido una transacción que asigna una salida a nuestra dirección. Una vez confirmada podemos gastar esa salida.

Primero utilizamos el comando listunspent para mostrar todas las salidas *confirmadas* sin gastar en nuestra cartera:

```
$ bitcoin-cli listunspent
```

Vemos que la transacción 9ca8f9... creó una salida (con índice de vout 0) asignada a la dirección 1hvzSo... por el monto de 50 milibits, la cual a este punto ha recibido siete confirmaciones. Las transacciones utilizan salidas creadas previamente como sus entradas refiriéndose a ellas por el txid e índice de vout previos. Ahora crearemos una transacción que gastará el vout 0-ésimo de la transacción 9ca8f9... como su entrada y la asignará a una nueva salida que envíe el valor a una nueva dirección.

Primero observemos esta salida específica en mayor detalle. Usamos gettxout para obtener los detalles de esta salida sin gastar. Las salidas de transacciones son siempre referenciadas por txid y vout, y éstos son los parámetros que pasamos a gettxout:

Lo que vemos aquí es la salida que asignó 50 milibits a nuestra dirección 1hvz.... Para gastar esta salida debemos crear una nueva transacción. Primero creemos una dirección a la cual enviaremos el dinero:

```
$ bitcoin-cli getnewaddress
1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb
```

Enviaremos 25 milibits a la nueva dirección 1LnfTn... que acabamos de crear en nuestra cartera. En nuestra nueva transacción gastaremos una salida de 50 milibits y enviaremos 25 milibits a esta nueva dirección. Ya que tenemos que gastar la salida de la transacción previa *entera*, también tendremos que generar cambio. Generaremos el cambio de regreso a la dirección 1hvz..., enviando el cambio de vuelta a la dirección de la cual sale el valor originalmente. Finalmente también tendremos que pagar una pequeña comisión por esta transacción. Para pagar la comisión debemos reducir la salida del cambio en 0.5 milibits y devolver 24,5 milibits de cambio. La diferencia entre la suma de las nuevas salidas (25 mBTC + 24,5 mBTC = 49,5 mBTC) y la entrada (50 mBTC) será recolectada por los mineros como la comisión de transacción.

Usamos createarawtransaction para crear esta transacción. Como parámetros para createrawtransaction proporcionamos la entrada de la transacción (la salida sin gastar de 50 milibits de nuestra transacción confirmada) y las dos salidas de la transacción (dinero enviado a la nueva dirección y cambio enviado de vuelta a la dirección previa):

El comando createrawtransaction produce una cadena hexadecimal en crudo que codifica los detalles de transacción que hemos provisto. Confirmemos que todo esté correcto decodificando esta cadena en crudo usando el comando decoderawtransaction:

¡Eso se ve correcto! Nuestra nueva transacción "consume" la salida sin gastar de nuestra transacción confirmada y luego la gasta en dos salidas, una por 25 milibits a nuestra nueva dirección y una por 24,5 milibits como cambio de regreso a la dirección original. La diferencia de 0,5 milibits representa la comisión de transacción y será acreditada al minero que encuentre el bloque que incluye nuestra transacción.

Como puede haber notado, la transacción contiene un scriptSig vacío ya que no la hemos firmado aun. Sin una firma esta transacción carece de significado; no hemos aún probado que la dirección de la cual proviene la salida sin gastar nos pertenece. Al firmar removemos el candado sobre la salida y probamos que somos dueños de esta salida y podemos gastarla. Usamos el comando signrawtransaction para firmar la transacción. El comando toma la cadena hexadecimal de la transacción en crudo como parámetro:

TIP	Una cartera encriptada debe ser abierta antes de que la transacción sea firmada ya que firmar requiere de acceso a las claves secretas en la cartera.

El comando signrawtransasction devuelve otra transacción en crudo codificada en hexadecimal. La decodificamos para ver qué ha cambiado con decoderawtransaction:

Ahora las entradas usadas en la transacción contienen un scriptSig, el cual es una firma digital probando la pertenencia de la dirección 1hvz... y removiendo el cerrojo sobre la salida para que pueda ser gastada. La firma hace a esta transacción verificable por cualquier nodo en la red bitcoin.

Ahora es tiempo de enviar la transacción recientemente creada a la red. Hacemos esto con el comando sendrawtransaction, el cual toma la cadena hexadecimal en crudo producida por signrawtransaction. Esta es la misma cadena que acabamos de decodificar:

El comando sendrawtransaction devuelve un *hash de transacción (txid)* y envía la transacción a la red. Ahora podemos consultar ese ID de transacción con gettransaction:

```
{
    "amount" : 0.00000000,
    "fee" : -0.00050000,
    "confirmations" : 0,
    "txid" : "ae74538baa914f3799081ba78429d5d84f36a0127438e9f721dff584ac17b346",
    "time" : 1392666702,
    "timereceived" : 1392666702,
    "details" : [
        {
            "account" : "",
            "address" : "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb",
            "category" : "send",
            "amount" : -0.02500000,
            "fee" : -0.00050000
        },
        {
            "account" : "",
            "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
            "category" : "send",
            "amount" : -0.02450000,
            "fee" : -0.00050000
        },
        {
            "account" : "",
            "address" : "1LnfTndy3qzXGN19Jwscj1T8LR3MVe3JDb",
            "category" : "receive",
            "amount" : 0.02500000
        },
        {
            "account" : "",
            "address" : "1hvzSofGwT8cjb8JU7nBsCSfEVQX5u9CL",
            "category" : "receive",
            "amount" : 0.02450000
        }
    ]
}
```

Al igual que antes, podemos examinar esto en mayor detalle usando los comandos getrawtransaction y decoderawtransaction. Estos comandos devolverán la misma cadena hexadecimal que hemos producido y decodificado previamente a enviarla a la red.

Clientes Alternativos, Bibliotecas y Kits de Herramientas

Más allá del cliente de referencia (bitcoind) se pueden utlizar otros clientes y bibliotecas para

interactuar con la red bitcoin y sus estructuras de datos. Estos están implementados en una variedad de lenguajes de programación, ofreciendo a los programadores interfaces nativas en su propio lenguaje.

Implementaciones alternativas incluyen:

libbitcoin
: Kit de Herramientas de Desarrollo Multiplataforma Bitcoin en C++

bitcoin explorer
: Herramienta Bitcoin de Línea de Comando

bitcoin server
: Nodo Completo y Servidor de Consultas Bitcoin

bitcoinj
: Una biblioteca de nodo completo en Java

btcd
: Un cliente bitcoin de nodo completo en lenguaje Go

Bits of Proof (BOP)
: Una implementación de bitcoin en Java de nivel empresarial

picocoin
: Una implementación de una biblioteca de cliente ligero para bitcoin en C

pybitcointools
: Una biblioteca bitcoin en Python

pycoin
: Otra biblioteca bitcoin en Python

Existen otras muchas bibliotecas en multitud de lenguajes de programación y se siguen creando todo el tiempo.

Libbitcoin y Bitcoin Explorer

La biblioteca libbitcoin es un kit de herramientas de desarrollo multiplataforma C++ que soporta el nodo completo libbitcoin-server y la herramienta de línea de comandos Bitcoin Explorer (bx).

Los comandos bx ofrecen muchas de las mismas capacidades que los comandos del cliente bitcoind ilustrados en este capítulo. Los comandos bx también ofrecen algunas herramientas de administración y manipulación de claves que bitcoind no posee, incluyendo claves deterministas tipo-2 y codificación de claves mnemónicas, así como direcciones sigilosas, pagos y soporte de consultas.

Instalando Bitcoin Explorer

Para usar Bitcoin Explorer, simplemente descargue el ejecutable firmado para su sistema operativo. Existen versiones para mainnet y testnet para Linux, OS X y Windows.

Teclee bx sin parámetros para mostrar la lista de todos los comandos disponibles (ver [appdx_bx]).

Bitcoin Explorer también provee un instalador para compilar a partir del código fuente en Linux y OS X, y también proyectos Visual Studio para Windows. Los códigos fuente también pueden ser compilados manualmente por medio de Autotools. Estos también instalan la biblioteca libbitcoin de la cual dependen.

> **TIP** Bitcoin Explorer ofrece muchos comandos útiles para codificar y decodificar direcciones y convertirlas entre formatos y representaciones. Úselos para explorar varios formatos tales como Base16 (hexadecimal), Base58, Base58Check, Base64, etc.

Instalando Libbitcoin

La biblioteca libbitcoin provee un instalador para compilar a partir del código fuente en Linux y OS X, y también proyectos Visual Studio para Windows. Los códigos fuente también pueden ser compilados manualmente por medio de Autotools.

> **TIP** El instalador de Bitcoin Explorer instala bx y también la biblioteca libbitcoin, así que si ha compilado bx a partir del código fuente puede saltar este paso.

pycoin

La biblioteca Python *pycoin*, originalmente escrita y mantenida por Richard Kiss, es una biblioteca escrita en Python que soporta manipulación de claves y transacciones bitcoin, soportando inclusive el lenguaje de scripting lo suficiente como para lidiar apropiadamente con transacciones no estándar.

La biblioteca pycoin soporta tanto Python 2 (2.7.x) como Python 3 (para versiones mayores a 3.3) y viene con algunas utilidades de línea de comandos prácticas, ku y tx. Para instalar pycoin 0.42 bajo Python 3 en un entorno virtual (venv), utilice lo siguiente:

```
$ python3 -m venv /tmp/pycoin
$ . /tmp/pycoin/bin/activate
$ pip install pycoin==0.42
Downloading/unpacking pycoin==0.42
  Downloading pycoin-0.42.tar.gz (66kB): 66kB downloaded
  Running setup.py (path:/tmp/pycoin/build/pycoin/setup.py) egg_info for package
pycoin

Installing collected packages: pycoin
  Running setup.py install for pycoin

    Installing tx script to /tmp/pycoin/bin
    Installing cache_tx script to /tmp/pycoin/bin
    Installing bu script to /tmp/pycoin/bin
    Installing fetch_unspent script to /tmp/pycoin/bin
    Installing block script to /tmp/pycoin/bin
    Installing spend script to /tmp/pycoin/bin
    Installing ku script to /tmp/pycoin/bin
    Installing genwallet script to /tmp/pycoin/bin
Successfully installed pycoin
Cleaning up...
$
```

Aquí hay un script Python de ejemplo para traer y gastar algunos bitcoins usando la biblioteca pycoin:

```
#!/usr/bin/env python

from pycoin.key import Key

from pycoin.key.validate import is_address_valid, is_wif_valid
from pycoin.services import spendables_for_address
from pycoin.tx.tx_utils import create_signed_tx

def get_address(which):
    while 1:
        print("enter the %s address=> " % which, end='')
        address = input()
        is_valid = is_address_valid(address)
        if is_valid:
            return address
        print("invalid address, please try again")

src_address = get_address("source")
spendables = spendables_for_address(src_address)
print(spendables)

while 1:
    print("enter the WIF for %s=> " % src_address, end='')
    wif = input()
    is_valid = is_wif_valid(wif)
    if is_valid:
        break
    print("invalid wif, please try again")

key = Key.from_text(wif)
if src_address not in (key.address(use_uncompressed=False),
key.address(use_uncompressed=True)):
    print("** WIF doesn't correspond to %s" % src_address)
print("The secret exponent is %d" % key.secret_exponent())

dst_address = get_address("destination")

tx = create_signed_tx(spendables, payables=[dst_address], wifs=[wif])

print("here is the signed output transaction")
print(tx.as_hex())
```

Para ejemplos usando las utilidades de línea de comandos ku y tx, ver [appdxbitcoinimpproposals].

btcd

btcd es una implementación de nodo completo bitcoin escrita en Go. Actualmente descarga, valida y sirve la cadena de bloques usando las mismas reglas (incluyendo errores) para aceptación de bloques que la implementación de referencia, bitcoind. También transmite correctamente bloques recientemente minados, mantiene una reserva de transacciones y transmite transacciones individuales que no han sido aún ingresadas en un bloque. Se asegura de que todas las transacciones individuales admitidas en la reserva sigan las reglas requeridas y también incluye la vasta mayoría de los chequeos más estrictos de filtrado de transacciones basados en requerimientos de mineros (transacciones "estándar").

Una diferencia clave entre btcd y bitcoind es que btcd no incluye la funcionalidad de cartera, y ésta fue una decisión de diseño muy intencional. Esto significa que no puedes crear o recibir pagos directamente con btcd. Esa funcionalidad está provista por los proyectos btcwallet y btcgui, ambos cuales se encuentran bajo activo desarrollo. Otras diferencias notorias entre btcd y bitcoind incluyen el soporte de btcd para solicitudes HTTP POST (como bitcoind) y el método preferido de Websockets, y el hecho de que las conexiones RPC de btcd habilitan TLS por defecto.

Instalando btcd

Para instalar btcd en Windows, descargue y ejecute el msi disponible en GitHub, o ejecute el siguiente comando en Linux, asumiendo que ya tiene instalado el lenguaje Go:

```
$ go get github.com/conformal/btcd/...
```

Para actualizar btcd a la versión más reciente simplemente ejecute:

```
$ go get -u -v github.com/conformal/btcd/...
```

Controlando btcd

btcd posee un número de opciones de configuración, las cuales puede ver ejecutando:

```
$ btcd --help
```

btcd viene pre-empaquetado con algunas utilidades tales como btcctl, el cual es un programa de línea de comandos que puede ser usado tanto para controlar como para consultar btcd a través de RPC. btcd no habilita su servidor RPC por defecto; debe configurar como mínimo un nombre de usuario y contraseña RPC en los archivos de configuración siguientes:

- *btcd.conf*:

```
[Application Options]
rpcuser=myuser
rpcpass=SomeDecentp4ssw0rd
```

- *btcctl.conf*:

```
[Application Options]
rpcuser=myuser
rpcpass=SomeDecentp4ssw0rd
```

O si prefiere sobreescribir los archivos de configuración desde la línea de comandos:

```
$ btcd -u myuser -P SomeDecentp4ssw0rd
$ btcctl -u myuser -P SomeDecentp4ssw0rd
```

Para una lista de las opciones disponibles, ejecute lo siguiente:

```
$ btcctl --help
```

Claves, Direcciones, Carteras

Introducción

La propiedad de bitcoins se establece a través de *claves digitales*, *direcciones bitcoin*, y *firmas digitales*. Las claves digitales no son almacenadas realmente en la red, sino que son creadas y almacenadas por usuarios en un archivo o simple base de datos llamada *cartera* (wallet). Las claves digitales en la cartera de un usuario son completamente independientes del protocolo bitcoin y pueden ser generadas y administradas por el software de cartera del usuario sin referencia alguna a la cadena de bloques o acceso a Internet. Las claves habilitan muchas de las propiedades interesantes de bitcoin, incluyendo la confianza descentralizada y el control, comprobación de propiedad, y el modelo de seguridad de pruebas criptográficas.

Cada transacción bitcoin requiere una firma válida para ser incluida en la cadena de bloques, la cual puede generarse con claves digitales válidas; por lo tanto, quien posea una copia de dichas claves tendrá control de los bitcoins en esa cuenta. Las claves vienen en pares, consistiendo de una clave privada (secreta) y una clave pública. Imagine la clave pública como si fuera el número de una cuenta bancaria y la clave privada el PIN secreto o la firma en un cheque que proporciona control sobre la cuenta. Estas claves digitales son rara vez vistas por los usuarios de bitcoin. Normalmente se almacenan dentro del archivo cartera y son administradas por el software de cartera bitcoin.

En la parte del pago de una transacción bitcoin, la clave pública del destinatario es representada por su huella digital, llamada una *dirección bitcoin*, la cual es usada de igual forma que el nombre del beneficiario en un cheque (i.e., "Páguese a la orden de..."). En la mayoría de los casos una dirección bitcoin se genera a partir de y corresponde a una clave pública. Sin embargo, no todas las direcciones bitcoin representan una clave pública; también pueden representar otros beneficiarios tales como scripts, como veremos más tarde en este capítulo. De esta forma las direcciones bitcoin abstraen al destinatario de los fondos, flexibilizando el destino de las transacciones, de forma similar a los cheques en papel: un único instrumento de pago que puede ser usado para pagar a cuentas de personas, compañías, pagar facturas o pagar por efectivo. La dirección bitcoin es la única representación de las claves que los usuarios ven rutinariamente ya que esta es la parte que necesitan compartir con el mundo.

En este capítulo presentaremos carteras, las cuales contienen claves criptográficas. Echaremos un vistazo a cómo las claves son generadas, almacenadas y administradas. Analizaremos los varios formatos de codificación utilizados para representar claves privadas y públicas, direcciones y direcciones script. Finalmente veremos usos especiales de claves: para firmar mensajes, probar propiedad y crear direcciones de vanidad (vanity addresses) y carteras de papel (paper wallets).

Criptografía de Clave Pública y Criptomonedas

La criptografía de clave pública fue inventada en la década de 1970 y es la base matemática de la seguridad informática.

Desde la invención de la criptografía de clave pública, se han descubierto varias funciones matemáticas adecuadas, tales como exponenciación de números primos y multiplicación de curvas elípticas. Estas funciones matemáticas son prácticamente irreversibles, lo cual significa que son fáciles de calcular en una dirección e inviables de calcular en la dirección opuesta. Basada en estas funciones matemáticas, la criptografía permite la creación de secretos digitales y firmas digitales infalsificables. Bitcoin utiliza la multiplicación de curvas elípticas como base para su criptografía de clave pública.

En bitcoin utilizamos la criptografía de clave pública para crear un par de claves que controla el acceso a los bitcoins. El par de claves consiste en una clave privada y—derivada de esta última—una clave pública única. La clave pública se usa para recibir bitcoins, y la clave privada se usa para firmar transacciones y gastar dichos bitcoins.

Existe una relación matemática entre las claves pública y privada que permiten que la clave privada sea utilizada para generar firmas en mensajes. Estas firmas pueden ser validadas contra la clave pública sin necesidad de revelar la clave privada.

Cuando los bitcoins son gastados el dueño actual de los bitcoins presenta su clave pública y firma (diferente cada vez, pero creada a partir de la misma clave privada) en una transacción para gastar esos bitcoins. A través de la presentación de la clave pública y firma, todos los participantes en la red bitcoin pueden verificar y aceptar la transacción como válida, confirmando que la persona que transfiere los bitcoins los posee al momento de la transferencia.

TIP

En la mayoría de las implementaciones de carteras las claves privadas y públicas son almacenadas juntas como *pares de claves* por conveniencia. Sin embargo la clave pública puede calcularse a partir de la clave privada, por lo que almacenar únicamente la clave privada también es posible.

Claves Privadas y Públicas

Una cartera bitcoin contiene una colección de claves, cada una compuesta de una clave privada y una pública. La clave privada (k) es un número, generalmente elegido aleatoriamente. A partir de la clave privada utilizamos multiplicación de curva elíptica, una función criptográfica de sentido único, para generar la clave pública (K). A partir de la clave pública (K) utilizamos una función de hash criptográfica de sentido único para generar la dirección bitcoin (A). En esta sección comenzaremos por generar una clave privada, echar una mirada a la matemática de curva elíptica usada para covertirla en una clave pública, y finalmente generar una dirección bitcoin a partir de la clave pública. La relación entre clave privada, clave pública y dirección bitcoin se ilustra en Clave privada, clave pública y dirección bitcoin.

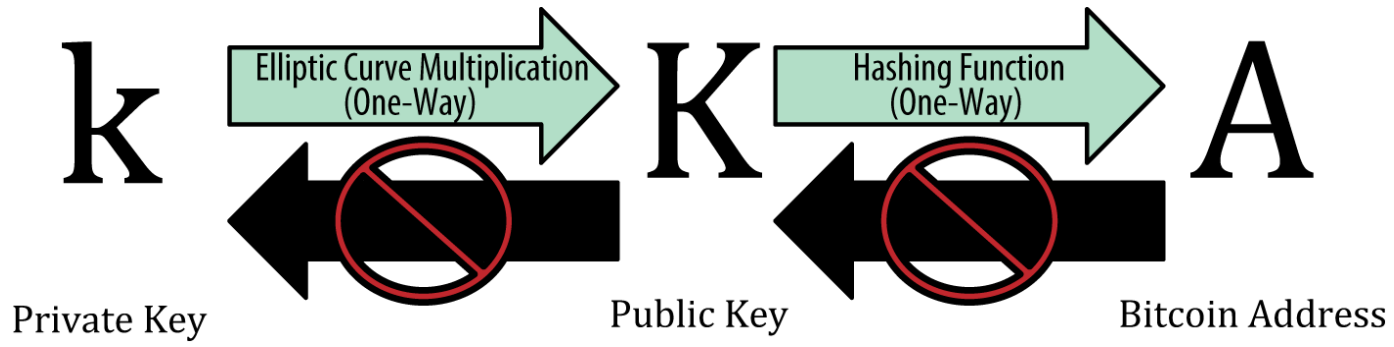


Figure 1. Clave privada, clave pública y dirección bitcoin

Claves Privadas

Una clave privada es simplemente un número escogido al azar. La propiedad y control de una clave privada es la raíz del control del usuario sobre los fondos asociados con la dirección bitcoin correspondiente. La clave privada se usa para crear las firmas requeridas para gastar bitcoins demostrando la pertenencia de los fondos usados en una transacción. La clave privada debe permanecer en secreto en todo momento, ya que revelarla a terceros es el equivalente a darles el control sobre los bitcoins asegurados por dicha clave. También deben hacerse copias de respaldo de las claves privadas para protegerlas de pérdidas accidentales, ya que si se pierde no puede ser recuperada y los fondos asegurados por ella se perderán para siempre.

TIP La clave privada bitcoin es simplemente un número. Puedes elegir tu clave privada aleatoriamente usando simplemente una moneda, papel y lápiz: arroja la moneda 256 veces y tendrás los dígitos binarios de una clave privada aleatoria que puedes usar en una cartera bitcoin. La clave pública puede luego ser generada a partir de la clave privada.

Generando una clave privada a partir de un número aleatorio

El primer y más importante paso en la generación de claves privadas es encontrar una fuente de entropía o azar segura. Crear una clave bitcoin es esencialmente lo mismo que "elegir un número entre 1 y 2^{256}." El método exacto utilizado para elegir tal número no importa siempre y cuando no sea predecible ni repetible. El software bitcoin utiliza los generadores de números aleatorios del sistema para generar 256 bits de entropía (azar). Usualmente el generador de números aleatorios del sistema operativo es inicializado por una fuente humana de azar, lo cual es la razón por la que puede que te solicite que muevas tu mouse durante algunos segundos. Para los verdaderamente paranoicos nada vence a un dado, papel y lápiz.

Más precisamente, la clave privada puede ser cualquier número entre 1 y n - 1, donde n es una constante (n = 1,158 * 10^{77}, poco menos que 2^{256}) definida como el orden de la curva elíptica usada en bitcoin (ver Criptografía de Curva Elíptica Explicada). Para crear tal clave elegimos un número de 256 bits y verificamos que sea menor a n - 1. En términos de programación, esto es generalmente se logra alimentando una cadena más grande de bits aleatorios, recolectada a partir de una fuente de aleatoriedad criptográficamente segura, en un algoritmo de hash SHA256 que convenientemente producirá un número de 256 bits. Si el resultado es menor a n - 1 hemos obtenido una clave privada apropiada. De lo contrario, simplemente lo intentamos nuevamente con otro número aleatorio.

TIP

No crees tu propio código para generar un número aleatorio ni uses un generador de números aleatorios "simple" ofrecido por tu lenguaje de programación. Utiliza una fuente de números pseudo-aleatorios criptográficamente segura (FNPACS) con una semilla a partir de una fuente de entropía suficiente. Estudia la documentación de la biblioteca de generación de números aleatorios que elijas para estar seguro de que sea criptográficamente segura. La correcta implementación de una FNPACS es crítica para la seguridad de las claves.

La siguiente es una clave privada (k) generada aleatoriamente, mostrada en formato hexadecimal (256 dígitos binarios representados en 64 dígitos hexadecimales, cada uno con 4 bits):

```
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
```

TIP

El tamaño del espacio de claves privadas bitcoin, 2^{256} es un número inimaginablemente grande. Es aproximadamente 10^{77} en decimal. Se estima que el universo visible contiene 10^{80} átomos.

Para generar una clave nueva con el cliente Bitcoin Core (ver [ch03_bitcoin_client]) usa el comando getnewaddress. Por razones de seguridad muestra solamente la clave pública, no la clave privada. Para pedir a bitcoind que exhiba la clave privada usa el comando dumpprivkey. El comando dumpprivkey muestra la clave privada en formato codificado en Base58 con checksum, *Formato de Importación de Cartera* (Wallet Import Fromat, o WIF), el cual examinaremos en mayor detalle en Formatos de claves privadas. Aquí hay un ejemplo de generación y exhibición de una clave privada usando estos dos comandos:

```
$ bitcoind getnewaddress
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
$ bitcoind dumpprivkey 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

El comando dumpprivkey abre una cartera y extrae la clave privada que fue generada por el comando getnewaddress. No es posible para bitcoind conocer la clave privada a partir de la clave pública a menos que estén ambas almacenadas en la cartera.

TIP

El comando dumpprivkey no está generando una clave privada a partir de una clave pública, ya que esto es imposible. El comando simplemente revela la clave privada ya conocida por la cartera, la cual ha sido generada por el comando getnewaddress.

También puedes usar la herramienta de línea de comando Bitcoin Explorer (ver [libbitcoin]) para generar y mostrar claves privadas con los comandos seed, ec-new y ec-to-wif:

```
$ bx seed | bx ec-new | bx ec-to-wif
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

Claves Públicas

La clave pública es generada a partir de la clave privada usando multiplicación de curva elíptica, la cual es irreversible: \(K = k * G\) donde *k* es la clave privada, *G* es un punto constante llamado el *punto generador* y *K* es la clave pública resultante. La operación inversa, conocida como "encontrando el logaritmo discreto"—calcular *k* a partir de *K*—es tan difícil como probar todos los valores de k, es decir, una búsqueda por fuerza bruta. Antes de demostrar cómo generar una clave pública a partir de una clave privada, echemos una mirada a la criptografía de curva elíptica en más detalle.

Criptografía de Curva Elíptica Explicada

La criptografía de curva elíptica es un tipo de criptografía asimétrica o de clave pública basada en el problema del logaritmo discreto tal como se expresa por suma y multiplicación sobre puntos de una curva elíptica.

Una curva elíptica es un ejemplo de una curva elíptica, similar a las usadas por bitcoin.

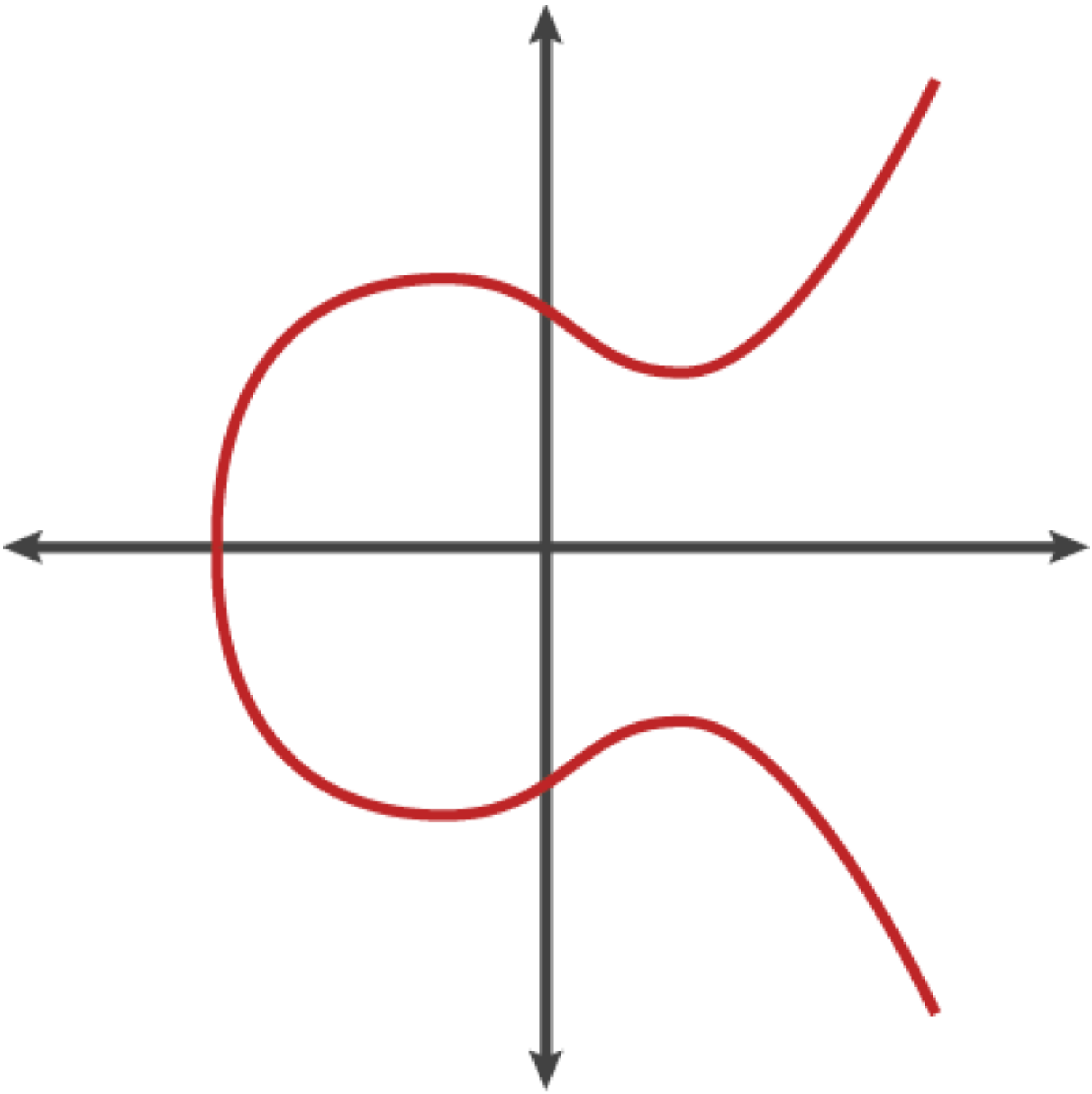


Figure 2. Una curva elíptica

Bitcoin usa una curva elíptica específica y un conjunto de constantes matemáticas definidas en un estándar llamado secp256k1, establecido por el Instituto Nacional de Estándares y Tecnología (National Institute of Standards and Technology, o NIST). La curva secp256k1 es definida por la siguiente función, la cual produce una curva elíptica:

ó

El *mod p* (módulo del número primo p) indica que la curva se encuentra sobre un cuerpo finito de orden *p*, también escrito como \(\mathbb{F}_p\), donde p = $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$, un número primo muy grande.

Ya que la curva se encuentra definida sobre un cuerpo finito de orden primo en vez de sobre los números reales, se ve como un patrón de puntos dispersos en dos dimensiones, lo cual lo vuelve difícil de visualizar. Sin embargo, la matemática es idéntica a la de la curva elíptica sobre los números reales. Como ejemplo, Criptografía de curva elíptica: visualizando una curva elíptica sobre F(p), con p=17 muestra la misma curva elíptica sobre un cuerpo finito mucho menor de orden primo 17, mostrando un patrón de puntos sobre una cuadrícula. La curva elíptica de bitcoin secp256k1 puede pensarse como un patrón mucho más complejo de puntos sobre una cuadrícula inconmensurablemente grande.

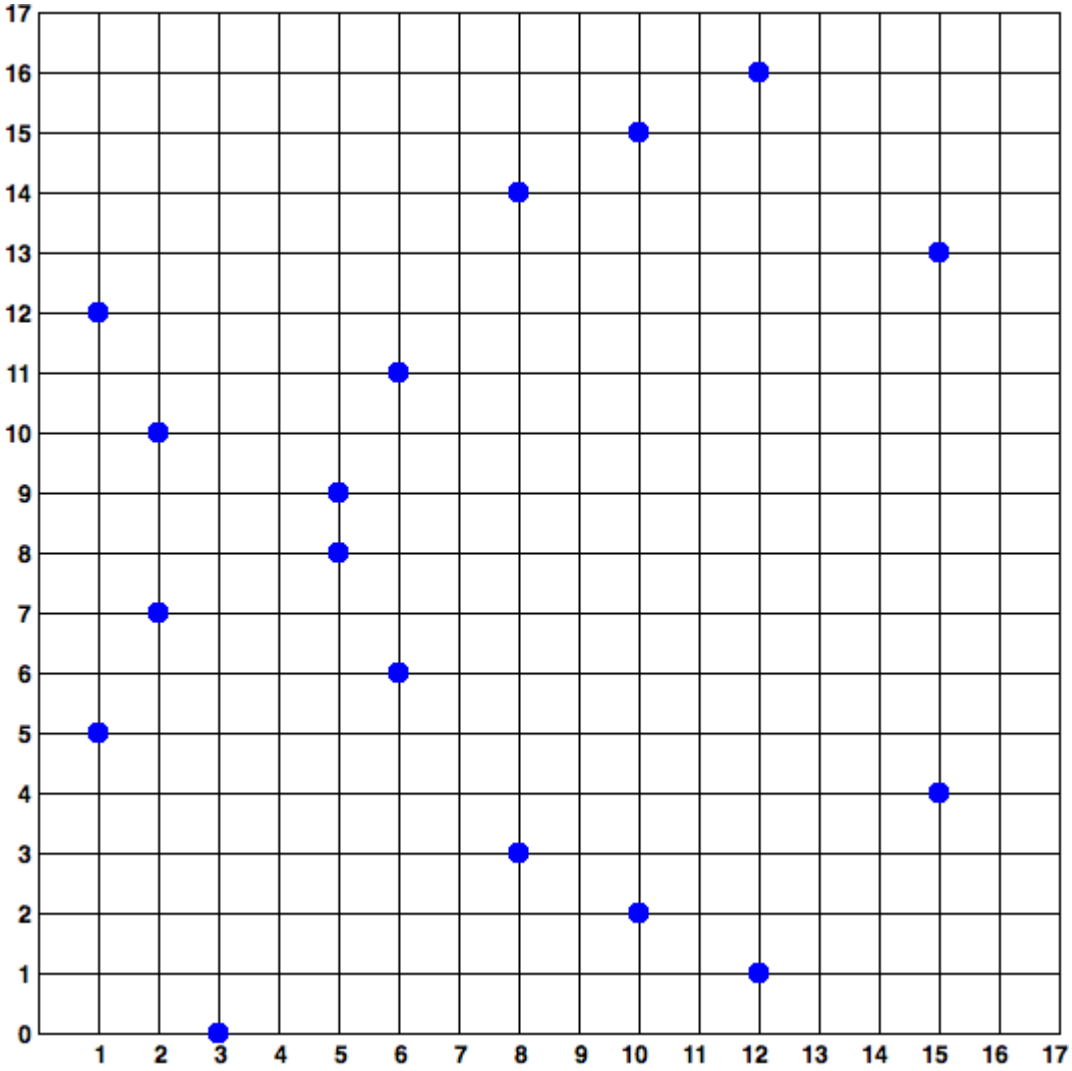


Figure 3. Criptografía de curva elíptica: visualizando una curva elíptica sobre F(p), con p=17

Así que, por ejemplo, el siguiente es un punto P con coordenadas (x,y) que es un punto en la curva secp256k1. Puedes verificar esto tú mismo usando Python:

```
P = (55066263022277343669578718895168534326250603453777594175500187360389116729240,
32670510020758816978083085130507043184471273380659243275938904335757337482424)
```

```
Python 3.4.0 (default, Mar 30 2014, 19:23:13)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> p =
115792089237316195423570985008687907853269984665640564039457584007908834671663
>>> x = 55066263022277343669578718895168534326250603453777594175500187360389116729240
>>> y = 32670510020758816978083085130507043184471273380659243275938904335757337482424
>>> (x ** 3 + 7 - y**2) % p
0
```

En matemática de curva elíptica existe un punto llamado el "punto al infinito," el cual corresponde al rol de 0 en la suma. En computadores a veces es representado como x = y = 0 (lo cual no satisface la ecuación de la curva elíptica, pero es un caso aislado simple que puede ser chequeado).

Existe también un operador + llamado "suma," el cual posee ciertas propiedades similares a la suma tradicional de números reales que aprenden los niños en la escuela. Dados dos puntos P_1 y P_2 sobre una curva elíptica, existe un tercer punto $P_3 = P_1 + P_2$, también sobre la curva.

Geométricamente, este tercer punto P_3 es calculado dibujando una línea entre P_1 y P_2. Esta línea intersecará la curva elíptica en exactamente un punto adicional. Llamemos a este punto $P_3' = (x, y)$. Luego reflejamos el eje x para obtener $P_3 = (x, –y)$.

Existen un par de casos especiales que explican la necesidad del "punto al infinito".

Si P_1 y P_2 son el mismo punto, la línea "entre" P_1 y P_2 debe extenderse para ser la tangente sobre la curva en el punto P_1. Esta tangente intersecará la curva en exactamente un nuevo punto. Puedes usar técnicas de cálculo para determinar la pendiente de la línea tangencial. Estas técnicas curiosamente funcionan a pesar de estar restringiendo nuestro interés a puntos sobre la curva con coordenadas de dos enteros.

En algunos casos (esto es, si P_1 y P_2 tienen el mismo valor en x pero distinto valor en y) la línea tangente será exactamente vertical, en cuyo caso P3 = "punto al infinito."

Si P_1 es el "punto al infinito," entonces la suma $P_1 + P_2 = P_2$. Similarmente, si P_2 es el punto al infinito, entonces $P_1 + P_2 = P_1$. Esto muestra que el punto al infinito juega el rol de 0.

Resulta que + es asociativo, lo cual significa que (A + B) + C = A + (B + C). Eso significa que podemos escribir A + B + C sin paréntesis y sin ninguna ambigüedad.

Ahora que hemos definido la suma podemos definir la multiplicación en la forma estándar en que extiende a la suma. Para un punto P sobre la curva elíptica, si k es un número entero, entonces kP = P + P + P + … + P (k veces). Nótese que k es a veces llamada una "exponente" en este caso, lo cual puede causar confusión.

Generando una Clave Pública

Comenzando con una clave privada en la forma de un número *k* generado aleatoriamente, lo multiplicamos por un punto predeterminado de la curva llamado *punto generador G* para producir otro punto en algún otro punto de la curva, el cual será la clave pública *K* correspondiente. El punto generador es especificado como parte del estándar secp256k1 y es siempre el mismo para todas las claves en bitcoin:

donde k es la clave privada, G es el punto generador, y K es la clave pública resultante, un punto sobre la curva. Ya que el punto generador es siempre el mismo para todos los usuarios de bitcoin, una clave privada k multiplicada por G siempre dará como resultado la misma clave pública K. La relación entre k y K es fija, pero solo puede ser calculada en una dirección, de k a K. Esa es la razón por la que una dirección bitcoin (derivada de K) puede ser compartida con cualquiera sin revelar la clave privada (k) del usuario.

> **TIP** Una clave privada puede ser convertida a una clave pública, pero una clave pública no puede ser convertida en una clave privada ya que la matemática solo funciona en un sentido.

Para implementar la multiplicación de curva elíptica tomamos la clave privada k generada previamente y la multiplicamos por el punto generador G para encontrar la clave pública K:

```
K = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD * G
```

La Clave Pública K se define como el punto K = (x,y):

```
K = (x, y)

donde,

x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Para visualizar la multiplicación de un punto y un entero usaremos la más sencilla curva elíptica sobre los números reales—recuerda, la matemática es la misma. Nuestro objetivo es encontrar el múltiplo kG del punto generador G. Eso es lo mismo que sumar G a sí mismo k veces consecutivas. En curvas elípticas sumar un punto a sí mismo es el equivalente a dibujar una línea tangente sobre el punto y hallar dónde interseca la curva nuevamente y luego reflejar ese punto sobre el eje x.

Criptografía de curva elíptica: Visualizando la multiplicacion de un punto G por un entero k sobre una curva elíptica muestra el proceso de derivar G, 2G, 4G, como una operación geométrica sobre la curva.

TIP La mayoría de las implementaciones bitcoin usan la biblioteca criptográfica OpenSSL para realizar los cálculos de curva elíptica. Por ejemplo, para derivar la clave pública se usa la función EC_POINT_mul().

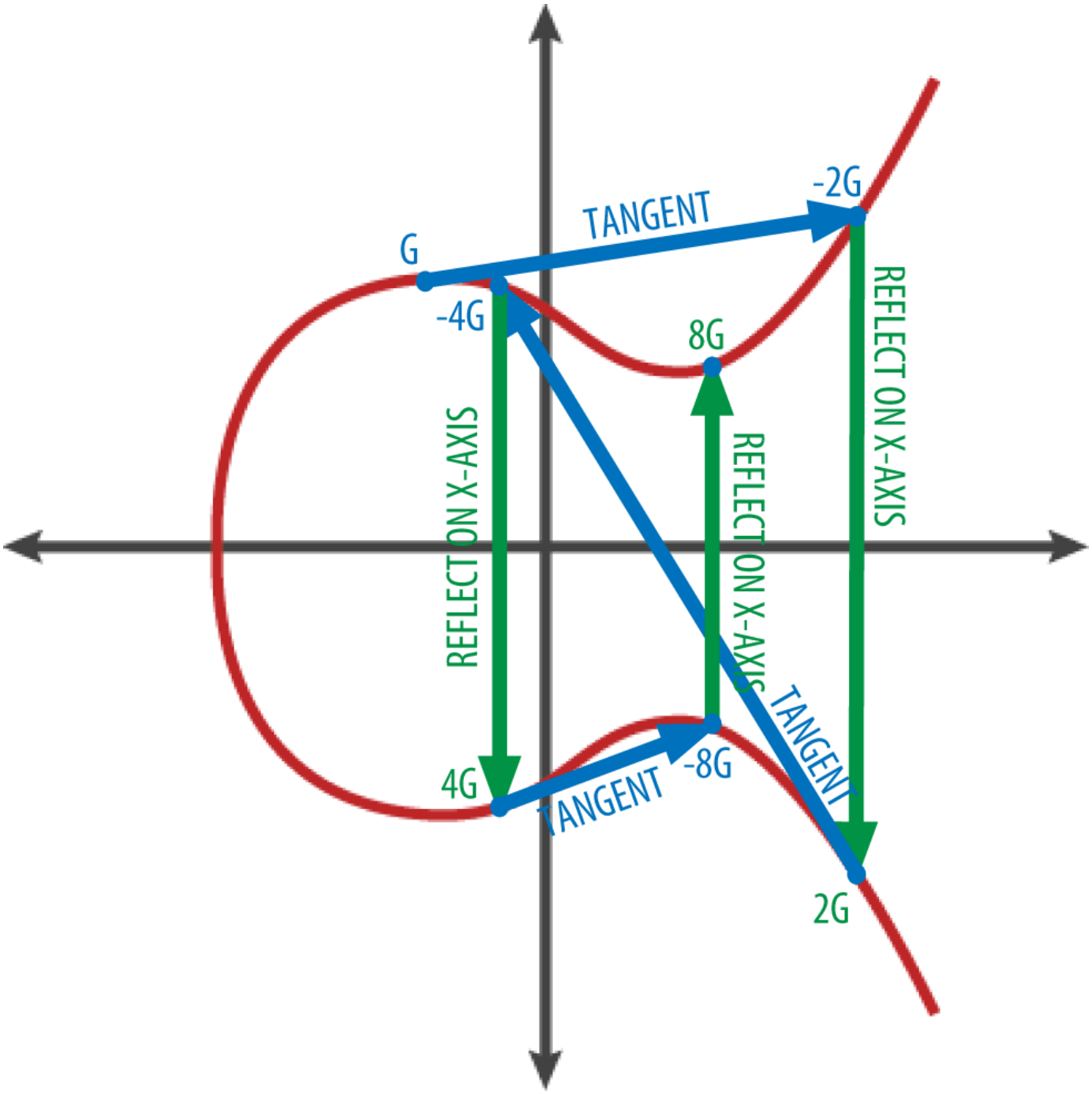


Figure 4. Criptografía de curva elíptica: Visualizando la multiplicacion de un punto G por un entero k sobre una curva elíptica

Direcciones Bitcoin

Una dirección bitcoin es una cadena de dígitos y caracteres que puede ser compartida con cualquiera que desee enviarte dinero. Las direcciones producidas a partir de una clave pública consisten de una

cadena de números y letras, comenzando por el dígito "1". Aquí hay un ejemplo de una dirección bitcoin:

```
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

La dirección bitcoin es lo que más frecuentemente aparece como el "destinatario" de los fondos. Si comparásemos una transacción bitcoin a un cheque en papel, la dirección bitcoin sería el beneficiario, es decir, lo que escribimos en la línea luego de "Páguese a la orden de." En un cheque en papel el beneficiario puede a veces ser el nombre del titular de una cuenta bancaria, pero puede incluir también corporaciones, instituciones, o incluso efectivo. El hecho de que los cheques en papel no requieran de especificar una cuenta, sino que en cambio usan un nombre abstracto como destinatario de los fondos, los convierte en instrumentos de pago muy flexibles. Las transacciones bitcoin usan una abstracción similar para ser muy flexibles: la dirección bitcoin. Una dirección bitcoin puede representar al propietario del par de clave privada y pública, o puede representar otra cosa, como un script de pago, como veremos en [p2sh]. Por ahora examinemos el caso simple: una dirección bitcoin que representa y es derivada de una clave pública.

La dirección bitcoin se obtiene de la clave pública a través del uso de hashing criptográfico de sentido único. Un "algoritmo de hashing", o simplemente un "algoritmo de hash" es una función de sentido único que produce una huella o "hash" a partir de una entrada de tamaño arbitrario. Las funciones de hash criptográfico se usan extensivamente en bitcoin: en las direcciones bitcoin, en las direcciones de script, y en algoritmo de prueba de trabajo de minado. Los algoritmos usados para crear direcciones bitcoin a partir de claves públicas son el Secure Hash Algorithm (SHA) y el RACE Integrity Primitives Evaluation Message Digest (RIPEMD), específicamente SHA256 y RIPEMD160.

A partir de la clave pública K computamos el hash SHA256 y luego computamos el hash RIPEMD160 del resultado, produciendo un número de 160 bits (20 bytes):

donde K es la clave pública y A es la dirección bitcoin resultante.

TIP Una dirección bitcoin *no es* lo mismo que una clave pública. Las direcciones bitcoin se obtienen de una clave pública a través de una función de sentido único.

Las direcciones bitcoin son casi siempre presentadas a los usuarios en una codificación llamada "Base58Check" (ver Codificación Base58 y Base58Check), la cual usa 58 caracteres (un sistema numérico de base 58) y un checksum para ayudar a la legibilidad humana, evitar ambigüedad y proteger de errores en la transcripción y entrada de direcciones. Base58Check también se usa en muchas otras formas en bitcoin, siempre que haya una necesidad de que un usuario lea y transcriba un número correctamente, tal como una dirección bitcoin, una clave privada, una clave encriptada, o un hash de script. En la siguiente sección examinaremos las mecánicas de la codificación y decodificación de Base58Check y las representaciones resultantes. Clave pública a dirección bitcoin: conversión de una clave pública en una dirección bitcoin ilustra la conversión de una clave pública a una dirección bitcoin.

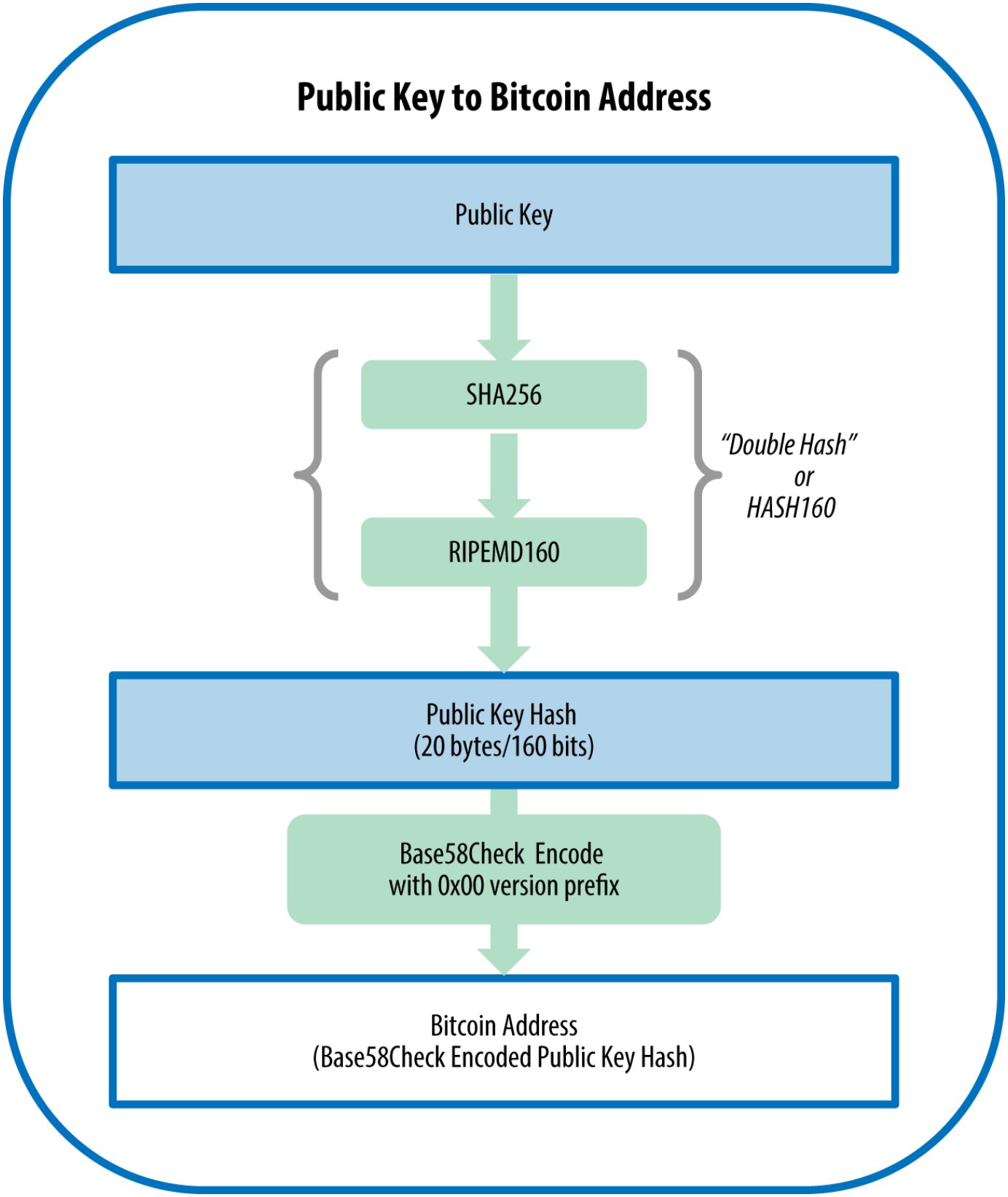


Figure 5. Clave pública a dirección bitcoin: conversión de una clave pública en una dirección bitcoin

Codificación Base58 y Base58Check

Para representar números largos en forma compacta, usando menos símbolos, muchos sistemas informáticos utilizan representaciones alfanuméricas mezcladas con una base mayor a 10. Por

ejemplo, mientras el sistema decimal tradicional utiliza 10 numerales, de 0 a 9, el sistema hexadecimal usa 16, con letras de la A a la F como los seis símbolos adicionales. Un número representado en formato hexadecimal es más breve que su equivalente representación decimal. Aun más compacta, la representación Base-64 usa 26 letras minúsculas, 26 letras mayúsculas, 10 numerales y dos carácteres más como "+" y "/" para transmitir datos binarios sobre medios de texto plano, tal como email. Base-64 es más comúnmente usado para añadir adjuntos binarios en emails. Base58 es un formato de codificación binaria basada en texto desarrollada para su uso en bitcoin y utilizada en muchas otras criptomonedas. Ofrece un balance entre representación compacta, legibilidad y detección y prevención de errores. Base58 es un subconjunto de Base64, usando las letras mayúsculas y minúsculas y números, pero omitiendo algunos carácteres que a menudo son confundidos por otros y pueden verse idénticos cuando se representan con ciertas fuentes. Específicamente, Base58 es Base64 sin el 0 (número cero), O (letra o mayúscula), l (letra L minúscula), I (letra i mayúscula) y los símbolos "\+" y "/". O, puesto de forma más sencilla, es el conjunto de letras mayúsculas y minúsculas y números sin los cuatro (0, O, l, I) que acabamos de mencionar.

Example 1. alfabeto Base58 de bitcoin

```
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
```

Para añadir seguridad extra contra errores tipográficos o de transcripción, Base58Check es un formato de codificación de Base58, usado frecuentemente en bitcoin, el cual posee un código de chequeo de errores. El checksum consiste de cuatro bytes adicionales añadidos al final de los datos siendo codificados. El checksum es derivado del hash de los datos codificados y puede por ende ser usado para detectar y prevenir errores de transcripción o tipeo. Cuando el código de decodificación es presentado con un código Base58Check calculará el checksum de los datos y lo comparará con el checksum incluido en el código. Si no son idénticos significa que ha habido un error y el código Base58Check es inválido. Por ejemplo, esto previene que una dirección bitcoin tipeada erróneamente sea aceptada por el software de cartera como un destinatario válido, un error que de lo contrario resultaría en la pérdida de fondos.

Para convertir datos (un número) al formato Base58Check primero agregamos un prefijo a los datos, llamado el "byte de versión," el cual sirve para identificar fácilmente el tipo de datos siendo codificados. Por ejemplo, en el caso de una dirección bitcoin el prefijo es cero (0x00 en hexadecimal), mientras que el prefijo usado cuando se codifica una clave privada es 128 (0x80 en hexadecimal). Una lista de prefijos de versión comunes puede verse en Prefijos de versión Base58Check y ejemplos de resultados codificados.

A continuación computamos el checksum "doble SHA", lo que significa que aplicamos el algoritmo de hash SHA256 dos veces sobre resultado previo (prefijo y datos):

```
checksum = SHA256(SHA256(prefijo+datos))
```

Del hash de 32 bytes resultante (hash de un hash) tomamos solo los primeros cuatro bytes. Estos cuatro bytes sirven como el código de chequeo de error, o checksum. El checksum es concatenado (o anexado) al final.

El resultado está compuesto de tres elementos: un prefijo, los datos y un checksum. Este resultado es codificado usando el alfabeto Base58 descrito anteriormente. Codificación Base58Check: un formato Base58, con versión y checksum para codificar datos bitcoin sin ambigüedades ilustra el proceso de codificación Base58Check.

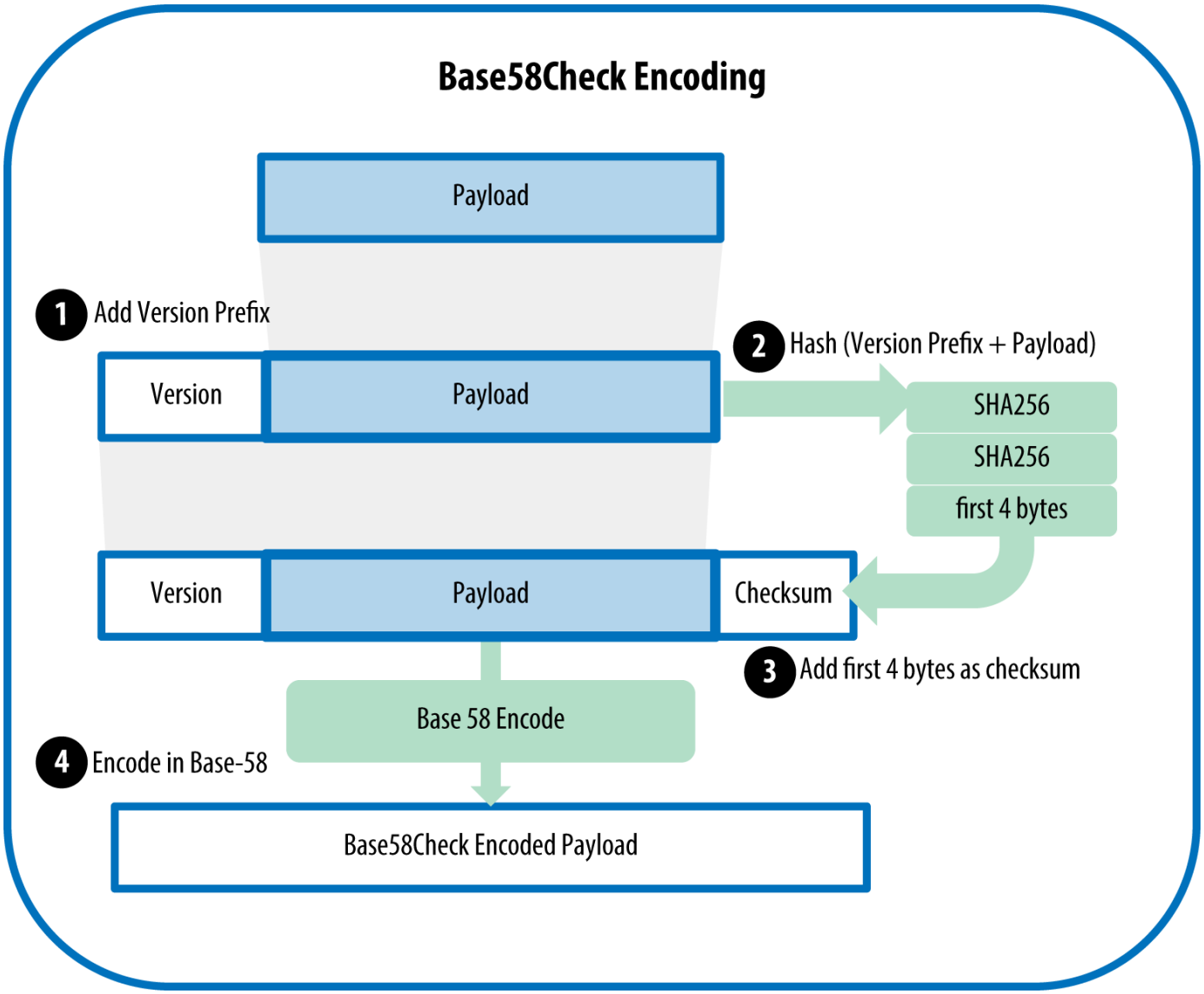


Figure 6. Codificación Base58Check: un formato Base58, con versión y checksum para codificar datos bitcoin sin ambigüedades

En bitcoin la mayoría de los datos presentados al usuario son codificados usando Base58Check para hacerlos compactos, fáciles de leer, y facilitar la detección de errores. El prefijo de versión en la codificación Base58Check se usa para crear formatos distinguibles fácilmente, los cuales al ser codificados en Base58 contienen caracteres específicos al principio de la carga codificada en Base58Check. Estos caracteres facilitan a humanos la identificación del tipo de los datos codificados y cómo usarlos. Esto es lo que diferencia, por ejemplo, a las direcciones bitcoin codificadas en

Base58Check comenzadas en 1 del formato WIF de claves privada codifacada en Base58Check comenzadas en 5. Algunos ejemplos de prefijos de versión y sus caracteres Base58 resultantes se muestran en Prefijos de versión Base58Check y ejemplos de resultados codificados.

Table 1. Prefijos de versión Base58Check y ejemplos de resultados codificados

Tipo	Prefijo de versión (hexadecimal)	Prefijo del resultado Base58
Dirección Bitcoin	0x00	1
Dirección Pago-a-Hash-de-Script	0x05	3
Dirección de Testnet Bitcoin	0x6F	m o n
WIF de Clave Privada	0x80	5, K o L
Clave Privada con Encriptación BIP38	0x0142	6P
Clave Pública Extendida BIP32	0x0488B21E	xpub

Veamos el proceso completo de creación de una dirección bitcoin, partiendo de una clave privada, a una clave pública (un punto en la clave elíptica) a una dirección doblemente hasheada y finalmente la codificación Base58Check. El código C++ Creando una dirección bitcoin codificada en Base58Check a partir de una clave privada muestra el proceso completo paso a paso, desde clave privada hasta dirección bitcoin codificada en Base58Check. El código de ejemplo usa la biblioteca libbitcoin presentada en [alt_libraries] para algunas funciones útiles.

Example 2. Creando una dirección bitcoin codificada en Base58Check a partir de una clave privada

```
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Private secret key.
    bc::ec_secret secret;
    bool success = bc::decode_base16(secret,
        "038109007313a5807b2eccc082c8c3fbb988a973cacf1a7df9ce725c31b14776");
    assert(success);
    // Get public key.
    bc::ec_point public_key = bc::secret_to_public_key(secret);
    std::cout << "Public key: " << bc::encode_hex(public_key) << std::endl;

    // Create Bitcoin address.
    // Normally you can use:
    //   bc::payment_address payaddr;
    //   bc::set_public_key(payaddr, public_key);
    //   const std::string address = payaddr.encoded();

    // Compute hash of public key for P2PKH address.
    const bc::short_hash hash = bc::bitcoin_short_hash(public_key);

    bc::data_chunk unencoded_address;
    // Reserve 25 bytes
    //   [ version:1  ]
    //   [ hash:20    ]
    //   [ checksum:4 ]
    unencoded_address.reserve(25);
    // Version byte, 0 is normal BTC address (P2PKH).
    unencoded_address.push_back(0);
    // Hash data
    bc::extend_data(unencoded_address, hash);
    // Checksum is computed by hashing data, and adding 4 bytes from hash.
    bc::append_checksum(unencoded_address);
    // Finally we must encode the result in Bitcoin's base58 encoding
    assert(unencoded_address.size() == 25);
    const std::string address = bc::encode_base58(unencoded_address);

    std::cout << "Address: " << address << std::endl;
    return 0;
}
```

El código usa una clave privada predefinida de forma que produzca la misma dirección bitcoin cada

vez que es ejecutado, como se muestra en Compilando y ejecutando el código addr.

Example 3. Compilando y ejecutando el código addr

```
# Compilando el código addr.cpp
$ g++ -o addr addr.cpp $(pkg-config --cflags --libs libbitcoin)
# Correr el ejecutable addr
$ ./addr
Public key: 0202a406624211f2abbdc68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa
Address: 1PRTTaJesdNovgne6Ehcdu1fpEdX7913CK
```

Formatos de Claves

Tanto las claves privadas como públicas pueden ser representadas en un número de formatos distintos. Todas estas representaciones codifican el mismo número a pesar de verse diferentes. Estos formatos se usan principalmente para facilitar el trabajo a las personas al leer y transcribir claves sin introducir errores.

Formatos de claves privadas

La clave privada puede ser representada en un número de formatos distintos, todos los cuales corresponden al mismo número de 256 bits. Representaciones de claves privadas (formatos de codificación) muestra tres formatos comunes usados para representar claves privadas.

Table 2. Representaciones de claves privadas (formatos de codificación)

Tipo	Prefijo	Descripción
Hexadecimal	Ninguno	64 dígitos hexadecimales
WIF	5	Codificación Base58Check: Base58 con un prefijo de versión de 128 y checksum de 32 bits
WIF comprimido	K o L	Como el caso anterior, con sufijo 0x01 añadido antes de codificar

La tabla Ejemplo: Misma clave, formatos distintos muestra la clave privada generada en estos tres formatos.

Table 3. Ejemplo: Misma clave, formatos distintos

Formato	Clave Privada
Hexadecimal	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd

WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
WIF comprimido	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Todas estas representaciones son formas distintas de mostrar el mismo número, la misma clave privada. Se ven distintas, pero cualquiera de estos formatos puede ser convertido fácilmente a cualquier otro formato.

Usamos el comando wif-to-ec de Bitcoin Explorer (ver [libbitcoin]) para mostrar que ambas claves WIF representan la misma clave privada:

```
$ bx wif-to-ec 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd

$ bx wif-to-ec KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
```

Decodificar a partir de Base58Check

Los comandos de Bitcoin Explorer (ver [libbitcoin]) facilitan el escribir shell scripts y "tubos" de línea de comando que manipulan claves bitcoin, direcciones y transacciones. Puedes usar Bitcoin Explorer para decodificar el formato Base58Check en la línea de comandos.

Usamos el comando base58check-decode para decodificar la clave sin comprimir:

```
$ bx base58check-decode 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
wrapper
{
    checksum 4286807748
    payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
    version 128
}
```

El resultado contiene la clave como su carga (payload), el prefijo de versión 128 del Formato de Importación de Cartera (WIF) y un checksum.

Nótese que la "carga" de la clave comprimida es anexada con el sufijo 01, dando la señal de que la clave pública derivada debe ser comprimida.

```
$ bx base58check-decode KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
wrapper
{
    checksum 2339607926
    payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01
    version 128
}
```

Codificar de hexadecimal a Base58Check

Para codificar a Base58Check (lo opuesto al comando anterior) usamos el comando base59check-encode de Bitcoin Explorer (ver [libbitcoin]) y proveemos la clave privada hexadecimal, seguida por el prefijo de versión 128 del Formato de Importación de Cartera (WIF):

```
bx base58check-encode 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
--version 128
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

Codificar de hexadecimal (clave comprimida) a Base58Check

Para codificar a Base58Check como una clave privada "comprimida" (ver Claves privadas comprimidas) anexamos el sufijo 01 a la clave hexadecimal y luego codificamos como anteriormente:

```
$ bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01 --version 128
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

El formato resultante WIF comprimido comienza con una "K". Esto denota que la clave privada dentro tiene un sufijo de "01" y será usada para producir claves públicas comprimidas únicamente (ver Claves públicas comprimidas).

Formatos de claves públicas

Las claves públicas también son presentadas en distintas formas, principalmente como claves públicas *comprimidas* o *descomprimidas*.

Como vimos previamente, la clave pública es un punto sobre la curva elíptica el cual consiste de un par de coordenadas (x,y). Usualmente es representada con el prefijo 04 seguido por dos números de 256 bits, uno para la coordenada *x* del punto, y otro para la coordenada *y*. El prefijo 04 es usado para distinguir claves públicas descomprimidas a partir de claves públicas comprimidas que comienzan con un 02 o un 03.

Aquí hay una clave pública generada por la clave privada que creamos previamente, mostrada como las coordenadas x e y.

```
x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Aquí está la misma clave pública mostrada como un número de 520 bits (130 dígitos hexadecimales) con el prefijo 04 seguido por las coordenadas x y luego y, como 04 x y:

```
K = 04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A<?pdf-
cr?>07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Claves públicas comprimidas

<?dbhtml orphans="4"?>Las claves públicas comprimidas fueron introducidas a bitcoin para reducir el tamaño de las transacciones y conservar espacio en disco en los nodos que almacenan la base de datos de la cadena de bloques bitcoin. La mayoría de las transacciones incluye la clave pública requerida para validar las credenciales del propietario y gastar los bitcoins. Cada clave pública requiere 520 bits (prefijo \+ x \+ y), que al ser multiplicados por varios cientos de transacciones por bloque, o decenas de miles de transacciones por día, suman una cantidad significativa de datos a la cadena de bloques.

Como vimos en la sección Claves Públicas, una clave pública es un punto (x,y) sobre una curva elíptica. Ya que la curva expresa una función matemática, un punto sobre la curva representa una solución a una ecuación, y por ende, si conocemos la coordenada *x* podemos calcular la coordenada *y* resolviendo la ecuación $y^2 \bmod p = (x^3 + 7) \bmod p$. Esto nos permite almacenar solamente la coordenada *x* del punto de clave pública, omitiendo la coordenada *y* y reduciendo el tamaño de la clave y el espacio requerido para almacenarla en 256 bits. ¡Una reducción en tamaño de casi el 50% por transacción representa muchos datos ahorrados con el transcurrir del tiempo!

Mientras que las claves públicas descomprimidas llevan el prefijo 04, las claves públicas comprimidas empiezan con el prefijo 02 o 03. Veamos por qué hay dos prefijos posibles: ya que el lado izquierdo de la ecuación es y^2 la solución para y es una raíz cuadrada, la cual puede tener un valor positivo o negativo. Visualmente esto significa que la coordenada *y* resultante puede encontrarse por encima o por debajo del eje x. En el gráfico de la curva elíptica Una curva elíptica se puede observar que la curva es simétrica, lo cual significa que es reflejada como un espejo por el eje x. Entonces, a pesar de poder omitir la coordenada *y* debemos almacenar el *signo* de y (positivo o negativo), o, en otras palabras, debemos recordar si estaba por encima o por debajo del eje x, ya que cada una de esas opciones representa un distinto punto y distinta clave pública. Cuando calculamos la curva elíptica en aritmética binaria sobre el cuerpo finitio de orden primo p, la coordenada *y* es o bien par o impar, lo cual corresponde al signo positivo o negativo explicado anteriormente. Por ende, para distinguir entre los dos posibles valores de y almacenamos una clave pública comprimida con el prefijo 02 si y es par, y 03 si es impar, permitiendo al software deducir correctamente la coordenada *y* a partir de la coordenada *x* y descomprimir la clave pública obteniendo las coordenadas completas del punto. La compresión de clave pública es ilustrada en Compresión de clave pública.

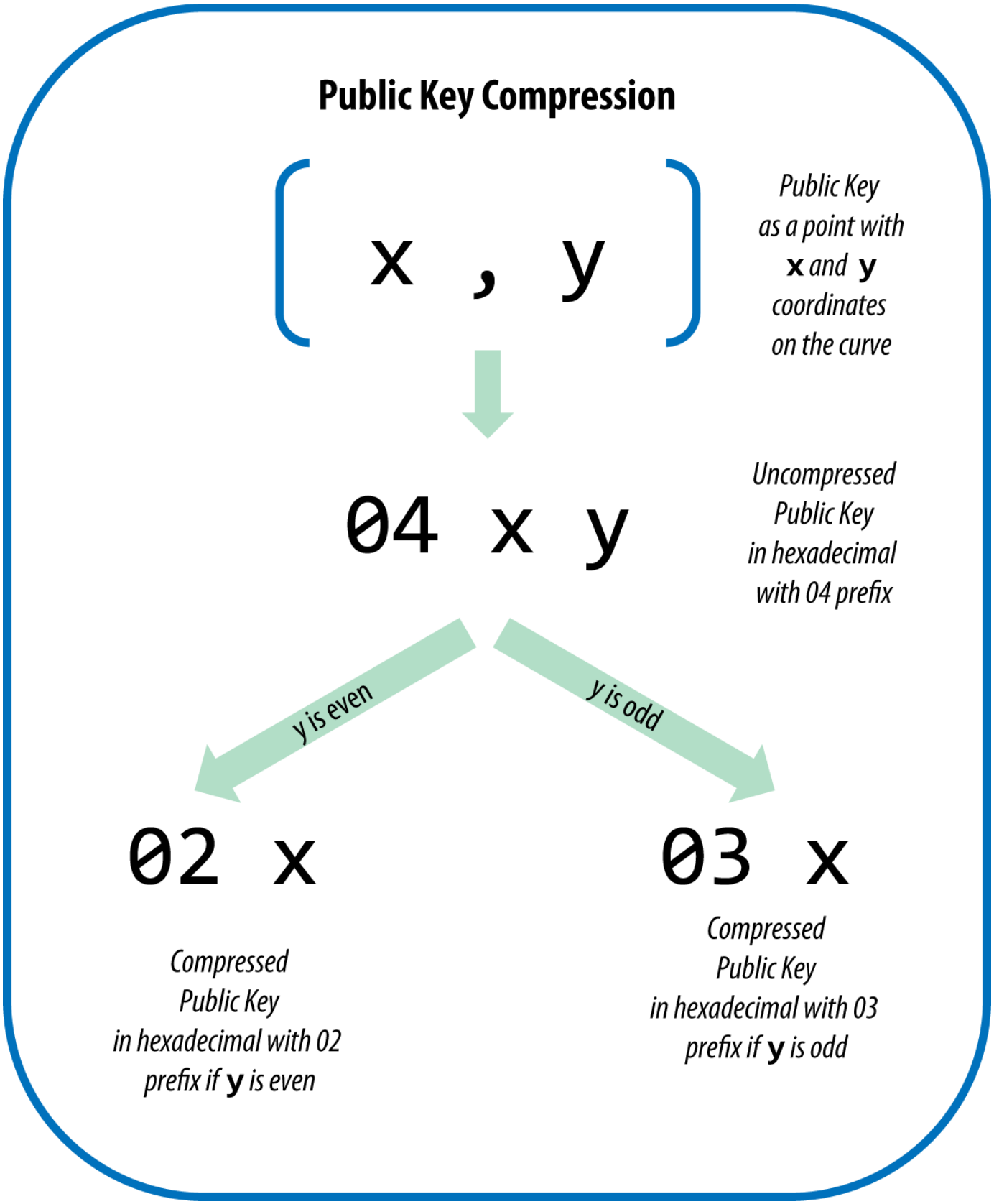


Figure 7. Compresión de clave pública

Aquí está la misma clave pública generada anteriormente, mostrada como una clave pública comprimida almacenada en 264 bits (66 dígitos hexadecimales) con el prefijo 03 indicando que la coordenada *y* es impar:

```
K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
```

Esta clave pública comprimida corresponde a la misma clave privada, lo que significa que es generada a partir de la misma clave privada. Sin embargo se ve distinta de la clave pública descomprimida. Más importante aun, si convertimos esta clave pública comprimida a una dirección bitcoin usando la función de hash doble (RIPEMD160(SHA256(K))) producirá una dirección bitcoin *diferente*. Esto puede resultar confuso, ya que significa que una misma clave privada puede producir una clave pública expresada en dos formatos distintos (comprimida y descomprimida) que producen dos direcciónes bitcoin diferentes. Sin embargo, la clave privada es idéntica para ambas direcciones bitcoin.

Las claves públicas comprimidas se están convirtiendo gradualmente en la opción por defecto en todos los clientes bitcoin, lo cual está teniendo un impacto significativo sobre la reducción del tamaño de transacciones y por ende la cadena de bloques. Sin embargo, no todos los clientes soportan claves públicas comprimidas aún. Los clientes más recientes que soportan claves públicas comprimidas tiene que tener en cuenta transacciones de clientes más antiguos que no soportan claves públicas comprimidas. Esto es especialmente importante cuando una aplicación de cartera importa claves privadas de otra aplicación de cartera bitcoin, ya que la nueva cartera necesita escanear la cadena de bloques para encontrar transacciones correspondientes a estas claves importadas. ¿Qué direcciones bitcoin debe buscar la cartera bitcoin? ¿Las direcciones bitcoin producidas por claves públicas descomprimidas, o las direcciones bitcoin producidas por claves públicas comprimidas? Ambas son direcciones bitcoin válidas, y la clave privada puede firmar por ellas, ¡pero son direcciones distintas!

Para resolver este problema el Formato de Importación de Cartera (WIF) usado al exportar claves privadas de una cartera es implementado en forma diferente en carteras más recientes, indicando de esta forma que dichas claves privadas han sido usadas para producir claves públicas *comprimidas* y por ende direcciones bitcoin *comprimidas*. Esto permite a la cartera a la que se importa distinguir entre claves privadas originadas en carteras recientes o antiguas y buscar transacciones en la cadena de bloques que posean direcciones correspondientes a las claves públicas comprimidas o descomprimidas, respectivamente. Veamos cómo esto funciona en mayor detalle en la siguiente sección.

Claves privadas comprimidas

Irónicamente el término "clave privada comprimida" es engañoso, ya que al exportar una clave privada como WIF comprimida resulta de hecho siendo un byte *más larga* que una clave privada "descomprimida". Esto es debido a que añade el sufijo 01, el cual significa que proviene de una cartera más reciente y no debe ser usada para producir claves públicas comprimidas. Las claves privadas no son comprimidas y no pueden ser comprimidas. El término "clave privada comprimida" en realidad significa "clave privada a partir de la cual debe derivarse una clave pública comprimida," mientras que "clave privada descomprimida" realmente significa "clave privada a partir de la cual debe derivarse una clave pública descomprimida." Para evitar mayor confusión debes referirte a los formatos de exportación como "WIF comprimido" o "WIF" en vez de referirte a las claves privadas como "comprimidas."

Recuerda, estos formatos *no son* usados de manera intercambiable. En una cartera más reciente que implementa claves públicas comprimidas las claves privadas serán exhibidas únicamente como WIF comprimido (con prefijo K o L). Si la cartera es una implementación más antigua y no usa claves públicas comprimidas, las claves privadas solo serán exhibidas como WIF (con prefijo 5). El objetivo aquí es comunicar a la cartera que importará estas claves privadas si debe buscar en la cadena de bloques por direcciones y claves públicas comprimidas o descomprimidas.

Si una cartera bitcoin es capaz de implementar claves públicas comprimidas las usará en todas las transacciones. Las claves privadas en la cartera serán usadas para derivar los puntos de clave pública sobre la curva, los cuales serán comprimidos. Las claves públicas comprimidas serán usadas para producir direcciones bitcoin y esas serán usadas en transacciones. Al exportar claves privadas desde una nueva cartera que implementa claves públicas comprimidas, el Formato de Importación de Cartera es modificado, con el añadido del sufijo de un byte 01 a la clave privada. La clave privada resultante codificada en Base58Check se llama "WIF Comrpimido" y comienza con la letra K o L en vez de con un "5" como es el caso de claves codificadas en WIF (descomprimido) de carteras más antiguas.

Ejemplo: Misma clave, formatos distintos muestra la misma clave codificada en formatos WIF y WIF comprimido.

Table 4. Ejemplo: Misma clave, formatos distintos

Formato	Clave Privada
Hex	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Hexadecimal comprimido	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD_01_
WIF comprimido	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

TIP

¡"Claves privadas comprimidas" es un nombre poco apropiado! No son comprimidas; en cambio, el formato WIF comprimido significa que deben ser usadas para derivar claves públicas comprimidas y sus direcciones bitcoin correspondientes. Irónicamente, una clave privada codificada con WIF comprimido es un byte más largo ya que tiene el sufijo 01 añadido para distinguirlo de una "descomprimida".

Implementando Claves y Direcciones en Python

La biblioteca bitcoin en Python más completa es pybitcointools por Vitalik Buterin. En Generación y formato de clave y dirección con la biblioteca pybitcointools, usamos la biblioteca (importada como "bitcoin") para generar y mostrar claves y direcciones en varios formatos.

Example 4. Generación y formato de clave y dirección con la biblioteca pybitcointools

```
import bitcoin

# Generate a random private key
valid_private_key = False
while not valid_private_key:
    private_key = bitcoin.random_key()
    decoded_private_key = bitcoin.decode_privkey(private_key, 'hex')
    valid_private_key =  0 < decoded_private_key < bitcoin.N

print "Private Key (hex) is: ", private_key
print "Private Key (decimal) is: ", decoded_private_key

# Convert private key to WIF format
wif_encoded_private_key = bitcoin.encode_privkey(decoded_private_key, 'wif')
print "Private Key (WIF) is: ", wif_encoded_private_key

# Add suffix "01" to indicate a compressed private key
compressed_private_key = private_key + '01'
print "Private Key Compressed (hex) is: ", compressed_private_key

# Generate a WIF format from the compressed private key (WIF-compressed)
wif_compressed_private_key = bitcoin.encode_privkey(
    bitcoin.decode_privkey(compressed_private_key, 'hex'), 'wif')
print "Private Key (WIF-Compressed) is: ", wif_compressed_private_key

# Multiply the EC generator point G with the private key to get a public key point
public_key = bitcoin.fast_multiply(bitcoin.G, decoded_private_key)
print "Public Key (x,y) coordinates is:", public_key

# Encode as hex, prefix 04
hex_encoded_public_key = bitcoin.encode_pubkey(public_key,'hex')
print "Public Key (hex) is:", hex_encoded_public_key

# Compress public key, adjust prefix depending on whether y is even or odd
(public_key_x, public_key_y) = public_key
if (public_key_y % 2) == 0:
    compressed_prefix = '02'
else:
    compressed_prefix = '03'
hex_compressed_public_key = compressed_prefix + bitcoin.encode(public_key_x, 16)
print "Compressed Public Key (hex) is:", hex_compressed_public_key

# Generate bitcoin address from public key
print "Bitcoin Address (b58check) is:", bitcoin.pubkey_to_address(public_key)

# Generate compressed bitcoin address from compressed public key
```

```
print "Compressed Bitcoin Address (b58check) is:", \
    bitcoin.pubkey_to_address(hex_compressed_public_key)
```

Ejecutando key-to-address-ecc-example.py muestra la salida de ejecutar este código.

Example 5. Ejecutando key-to-address-ecc-example.py

Un script mostrando la matemática de curva elíptica usada para claves bitcoin es otro ejemplo, usando la biblioteca Python ECDSA para la matemática de curva elíptica y sin usar ninguna biblioteca bitcoin especializada.

Example 6. Un script mostrando la matemática de curva elíptica usada para claves bitcoin

```
import ecdsa
import os
from ecdsa.util import string_to_number, number_to_string

# secp256k1, http://www.oid-info.com/get/1.3.132.0.10
_p = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2FL
_r = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141L
_b = 0x0000000000000000000000000000000000000000000000000000000000000007L
_a = 0x0000000000000000000000000000000000000000000000000000000000000000L
_Gx = 0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798L
_Gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8L
curve_secp256k1 = ecdsa.ellipticcurve.CurveFp(_p, _a, _b)
generator_secp256k1 = ecdsa.ellipticcurve.Point(curve_secp256k1, _Gx, _Gy, _r)
oid_secp256k1 = (1, 3, 132, 0, 10)
SECP256k1 = ecdsa.curves.Curve("SECP256k1", curve_secp256k1, generator_secp256k1,
oid_secp256k1)
ec_order = _r

curve = curve_secp256k1
generator = generator_secp256k1

def random_secret():
    convert_to_int = lambda array: int("".join(array).encode("hex"), 16)

    # Collect 256 bits of random data from the OS's cryptographically secure random
generator
    byte_array = os.urandom(32)

    return convert_to_int(byte_array)

def get_point_pubkey(point):
    if point.y() & 1:
```

```
        key = '03' + '%064x' % point.x()
    else:
        key = '02' + '%064x' % point.x()
    return key.decode('hex')

def get_point_pubkey_uncompressed(point):
    key = '04' + \
          '%064x' % point.x() + \
          '%064x' % point.y()
    return key.decode('hex')


# Generate a new private key.
secret = random_secret()
print "Secret: ", secret

# Get the public key point.
point = secret * generator
print "EC point:", point

print "BTC public key:", get_point_pubkey(point).encode("hex")

# Given the point (x, y) we can create the object using:
point1 = ecdsa.ellipticcurve.Point(curve, point.x(), point.y(), ec_order)
assert point1 == point
```

Instalando la biblioteca Python ECDSA y ejecutando el script ec_math.py muestra la salida producida al ejecutar este script.

El ejemplo previo usa os.urandom, el cual refleja un generador de números aleatorios criptográficamente seguro (CSRNG) provisto por el sistema operativo. En el caso de un sistema tipo UNIX, como Linux, obtendrá sus números de /dev/urandom; y en el caso de Windows llamará a CryptGenRandom(). Si no se encuentra una fuente de azar confiable, se lanzará un error NotImplementedError . Mientras que el generador de números aleatorios usado aquí es para propósitos demostrativos, *no es* apropiado para generar claves bitcoin de calidad de producción ya que no fue implementado con seguridad suficiente.

Example 7. Instalando la biblioteca Python ECDSA y ejecutando el script ec_math.py

```
$ # Instalar el administrador de paquetes Python PIP
$ sudo apt-get install python-pip
$ # Instalar la biblioteca Python ECDSA
$ sudo pip install ecdsa
$ # Ejecutar el script
$ python ec-math.py
Secret:
38090835015954358862481132628887443905906204995912378278060168703580660294000
EC point:
(70048853531867179489857750497606966272382583471322935454624595540007269312627,
105262206478686743191060800263479589329920209527285803935736021686045542353380)
BTC public key: 029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a1522873
```

Carteras

Las carteras son contenedores de claves privadas, usualmente implementadas como archivos estructurados o simples bases de datos. Otro método para hacer claves es la *generación determinista de claves*. En ella derivas cada nueva clave privada usando una función de hash de sentido único de una clave privada previa, vinculándolas en una secuencia. Siempre y cuando puedas recrear esa secuencia tan solo necesitas de la primera clave (conocida como clave *semilla* o *maestra*) para generarlas todas. En esta sección examinaremos las distintas maneras de generar claves y las estructuras de cartera que se construyen a su alrededor.

TIP

Las carteras bitcoin contienen claves, no monedas. Cada usuario posee una cartera conteniendo claves. Las carteras son en esencia llaveros conteniendo pares de claves privadas/públicas (ver Claves Privadas y Públicas). Los usuarios firman transacciones con las claves, demostrando de esa forma que son dueños de las salidas de transacción (sus monedas). Las monedas son almacenadas en la cadena de bloques en forma de salidas de transacción (a menudo notadas como vout o txout).

Carteras No Deterministas (Aleatorias)

En los primeros clientes bitcoin, las carteras eran simplemente colecciones de claves privadas generadas aleatoriamente. Este tipo de cartera se conoce como *cartera no determinista de Tipo-0*. Por ejemplo, el cliente Bitcoin Core genera previamente 100 claves privadas aleatorias cuando se inicia, y genera más claves cuando son necesarias, usando cada clave solamente una vez. Este tipo de cartera se conoce como "Simplemente un Montón De Claves," o JBOK, y están siendo reemplazadas por carteras deterministas porque son engorrosas de manejar, respaldar e importar. La desventaja de las carteras aleatorias es que si generas muchas has de realizar copia de todo, lo que supone que la cartera ha de ser respaldada de forma frecuente. Cada clave ha de respaldarse, o los fondos que controla se pierden irrevocablemente si la cartera pasa a ser inaccesible. Esto entra en conflicto directamente con el

principio de evitar la reutilización de direcciones, usando cada dirección de bicoin durante solo una transacción… La reutilización de direcciones reduce la privacidad mediante la asociación de múltiples transacciones y direcciones con los demás. Una cartera de Tipo-0 no determinista es una mala elección de cartera, sobre todo si se quiere evitar la reutilización de direcciones porque eso significa gestionar muchas claves, lo que crea la necesidad de copias de seguridad frecuentes. Aunque el cliente Bitcoin Core incluye una cartera Tipo-0, el uso de esta cartera es desaconsejado por los desarrolladores de Bitcoin Core. << >> Type0_wallet muestra una billetera no determinista, que contiene un conjunto disperso de claves aleatorias.

Carteras Deterministas (A Partir de Semilla)

("semilla","carteras con"Las carteras deterministas o "con semilla" son carteras que contienen claves privadas que surgen a partir de una semilla común, mediante el uso de una función hash unidireccional. La semilla es un número generado aleatoriamente que se combina con otros datos, como un número de índice o "código de cadena" (ver < <hd_wallets> >) para derivar las claves privadas. En una cartera determinista, la semilla es suficiente para recuperar todas las claves derivadas, y por lo tanto una única copia de seguridad en el momento de la creación es suficiente. La semilla también es suficiente para una exportación de cartera o de importación, lo que permite una fácil migración de todas las claves de los usuarios entre diferentes implementaciones de cartera.

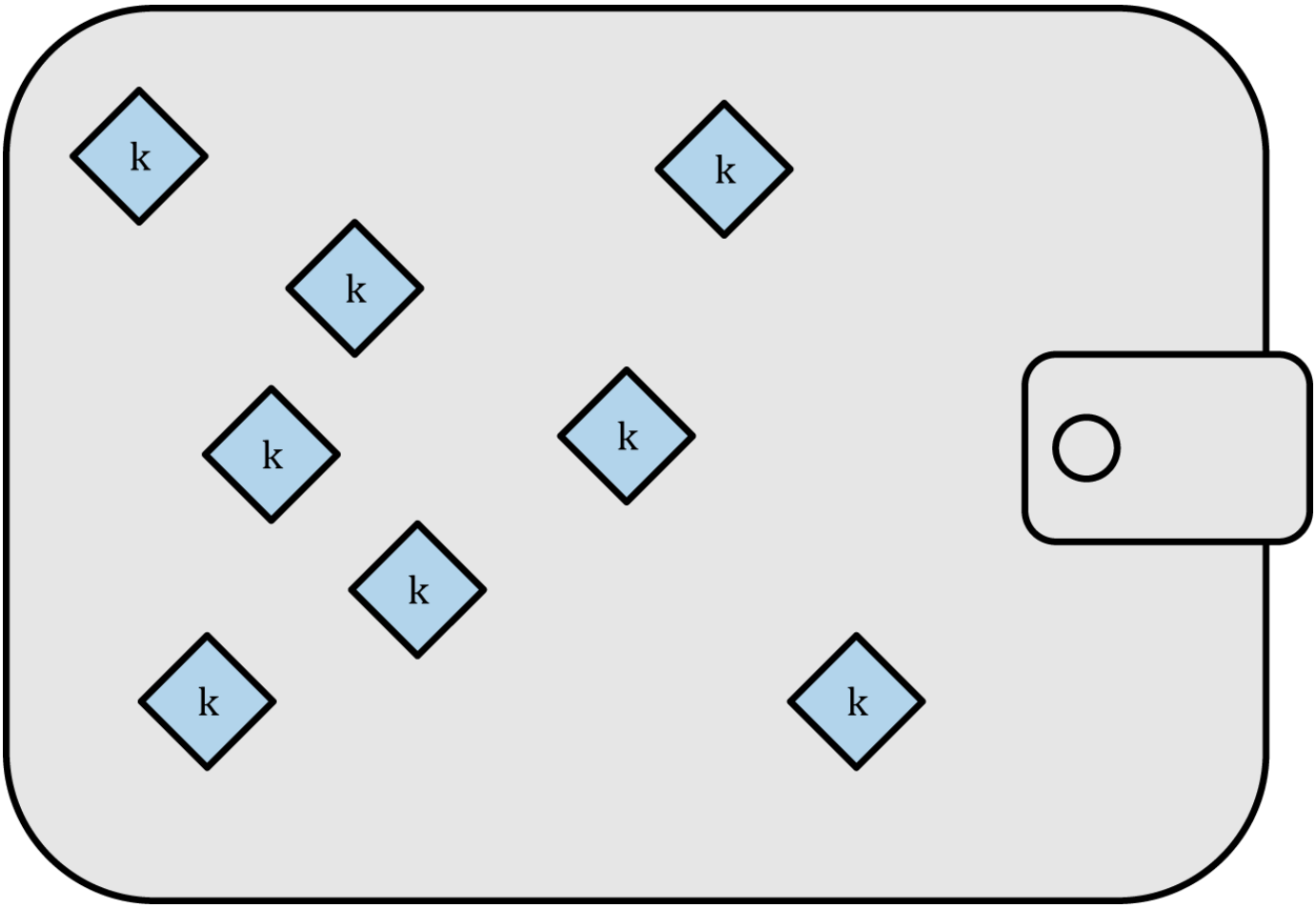


Figure 8. Cartera no determinista (aleatoria) Tipo-0: una colección de claves generadas aleatoriamente

Palabras Código Mnemónicas

Los códigos mnemotécnicos son palabras en inglés que representan (codifican) un número aleatorio utilizado como semilla para obtener una cartera determinista. La secuencia de palabras es suficiente para volver a crear la semilla y desde allí volver a crear la cartera y todas las claves derivadas. Una aplicación de cartera que implementa carteras deterministas con código nemotécnico mostrará al usuario una secuencia de 12 a 24 palabras al crear la cartera por primera vez. Esa secuencia de palabras es la copia de seguridad de la carpeta y se puede utilizar para recuperar y volver a crear todas las claves de la misma o de cualquier aplicación de cartera compatible. Las palabras de código mnemotécnico hace que sea más fácil para los usuarios realizar copias de seguridad de las carteras, ya que son fáciles de leer y transcribir correctamente, en comparación con una secuencia aleatoria de números.

Los códigos mnemotécnicos se definen en la Propuesta de Mejoras de Bitcoin 39 (ver [bip0039]), que actualmente se encuentra en estado de borrador. Tenga en cuenta que BIP0039 es una propuesta de borrador y no un estándar. En concreto, hay un estándar diferente, con un conjunto diferente de palabras, utilizado por la cartera Electrum y que precede a BIP0039. BIP0039 es utilizado por la cartera Trezor y algunas otras carteras, pero es incompatible con la aplicación de Electrum.

BIP0039 define la creación de un código y semilla mnemónicos de la siguiente manera:

1. Crear una secuencia aleatoria (entropía) de 128 a 256 bits.
2. Crear un checksum de la secuencia aleatoria tomando los primeros pocos bits de su hash SHA256.
3. Anexar el checksum al final de la secuencia aleatoria.
4. Dividir la secuencia en secciones de 11 bits, usándolas para indexar un diccionario de 2048 palabras predefinidas.
5. Producir 12 a 24 palabras representando el código mnemónico.

Códigos mnemónicos: entropía y longitud de palabra muestra la relación entre el tamaño de datos de entropía y la longitud de los códigos mnemónicos en palabras.

Table 5. Códigos mnemónicos: entropía y longitud de palabra

Entropía (bits)	Checksum (bits)	Entropía+checksum	Longitud de palabra
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

El código mnemónico representa de 128 a 256 bits, los cuales son usados para derivar una semilla más larga (512 bits) a través del uso de la función de estiramiento de clave PBKDF2. La semilla resultante es

usada para crear una cartera determinista y todas sus claves derivadas.

Las tablas <xref linkend="table_4-6" xrefstyle="select: labelnumber"/> y <xref linkend="table_4-7" xrefstyle="select: labelnumber"/> muestran algunos ejemplos de códigos mnemónicos y las semillas que producen.

Table 6. Código mnemónico de entropía de 128 bits y su semilla resultante

Entropía de entrada (128 bits)	0c1e24e5917779d297e14d45f14e1a1a
Mnemónico (12 palabras)	army van defense carry jealous true garbage claim echo media make crunch
Semilla (512 bits)	3338a6d2ee71c7f28eb5b882159634cd46a898463e9d2d0980f8e80dfbba5b0fa0291e5fb888a599b44b93187be6ee3ab5fd3ead7dd646341b2cdb8d08d13bf7

Table 7. Código mnemónico de entropía de 256 bits y su semilla resultante

Entropía de entrada (256 bits)	2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c
Mnemónico (24 palabras)	cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige
Semilla (512 bits)	3972e432e99040f75ebe13a660110c3e29d131a2c808c7ee5f1631d0a977fcf473bee22fce540af281bf7cdeade0dd2c1c795bd02f1e4049e205a0158906c343

Carteras Deterministas Jerárquicas (BIP0032/BIP0044)

Se desarrollaron carteras deterministas para que fuese fácil de obtener muchas claves de una sola "semilla". La forma más avanzada de carteras deterministas es la *cartera determinista jerárquica* o *cartera HD* definida por el estándar BIP0032. Las carteras deterministas jerárquicas contienen claves derivadas en una estructura de árbol, de tal manera que de una clave padre puede derivarse una secuencia de claves hijas, de cada una de las cuales puede derivarse una secuencia de claves nietos, y así sucesivamente, a una profundidad infinita. Esta estructura de árbol se ilustra en Cartera determinista jerárquica de Tipo-2: un árbol de claves generadas a partir de una única semilla.

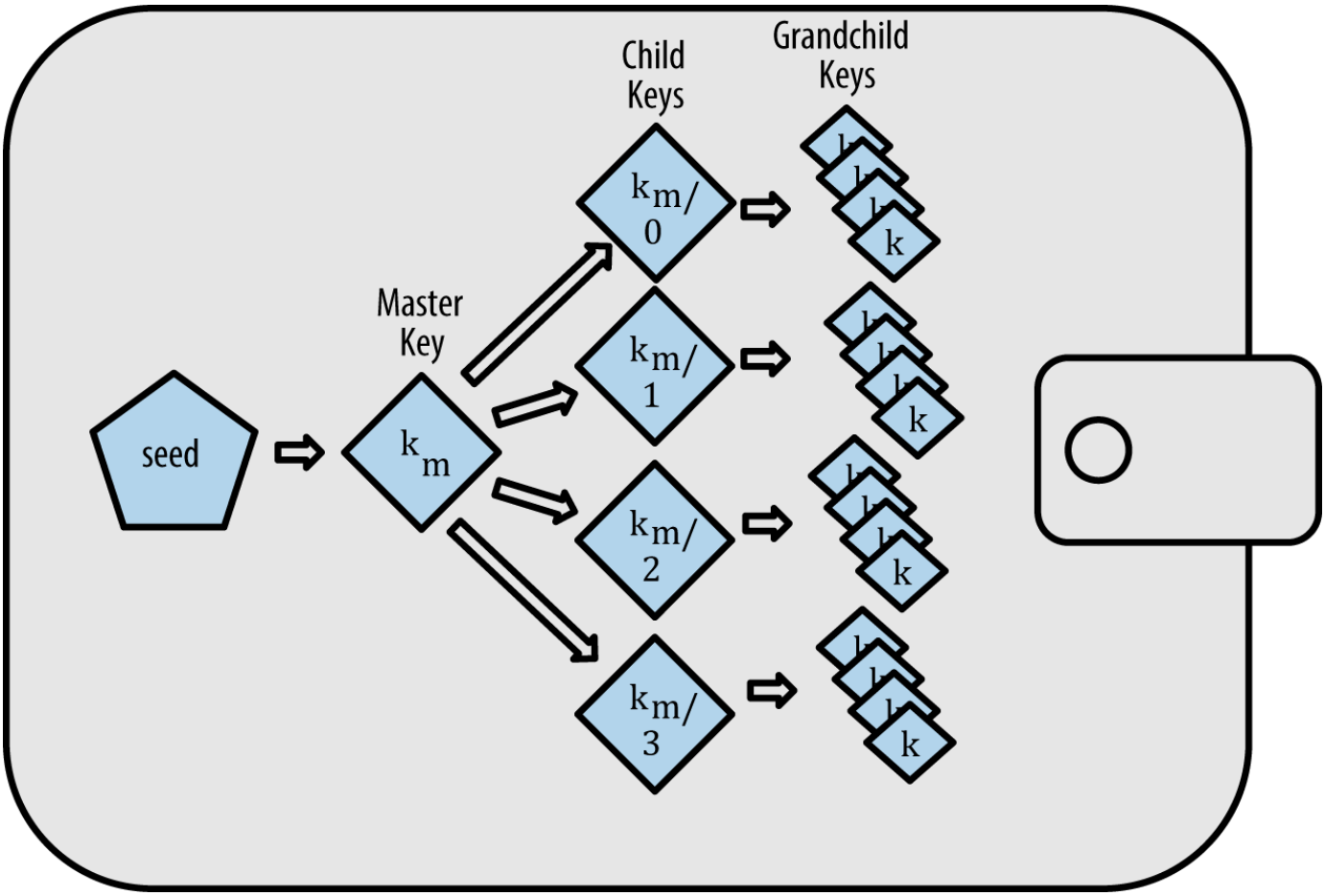


Figure 9. Cartera determinista jerárquica de Tipo-2: un árbol de claves generadas a partir de una única semilla

TIP	Si estás implementando una cartera bitcoin debería ser construida como una cartera HD siguiendo los estándares BIP0032 y BIP0044.

Las carteras HD ofrecen dos grandes ventajas sobre las claves aleatorias (no deterministas). En primer lugar, la estructura de árbol se puede utilizar para expresar un significado organizativo adicional, tal como cuando se utiliza una rama específica de subclaves para recibir los pagos entrantes y una rama diferente se utiliza para recibir el cambio de los pagos salientes. Las ramas de claves también se pueden utilizar en un entorno corporativo, la asignación de diferentes ramas a los departamentos, filiales, funciones específicas o categorías de contabilidad.

La segunda ventaja de las carteras HD es que los usuarios pueden crear secuencias de claves públicas sin tener acceso a las claves privadas correspondientes. Esto permite a las carteras HD ser usadas en servidores inseguros o en capacidad de receptión de fondos únicamente, generando una clave pública distinta para cada transacción. Las claves públicas no necesitan ser pre-cargadas ni derivadas por adelantado, y aun así el servidor no tiene las claves privadas que permiten gastar los fondos.

Creación de una cartera HD a partir de una semilla

Las carteras HD se crean a partir de una sola *semilla raíz*, que es un número aleatorio de 128, 256, o 512 bits. Todo lo demás en la cartera HD se deriva de forma determinista de esta semilla raíz, lo que

hace que sea posible volver a crear toda la cartera HD a partir esa semilla en cualquier cartera HD compatible. Esto hace que sea fácil de hacer copias de seguridad, restaurar, exportar e importar carteras HD que contienen miles o incluso millones de claves mediante la simple transferencia de una única semilla raíz. La forma más común de representar la semilla raíz es mediante una *secuencia de palabras mnemónicas*, como se describe en la sección anterior Palabras Código Mnemónicas, para que sea más fácil para las personas transcribirla y almacenarla.

El proceso de creación de claves maestras y código de cadena maestro para una cartera HD se muestra en Creando claves y códigos de cadena maestros a partir de una semilla raíz.

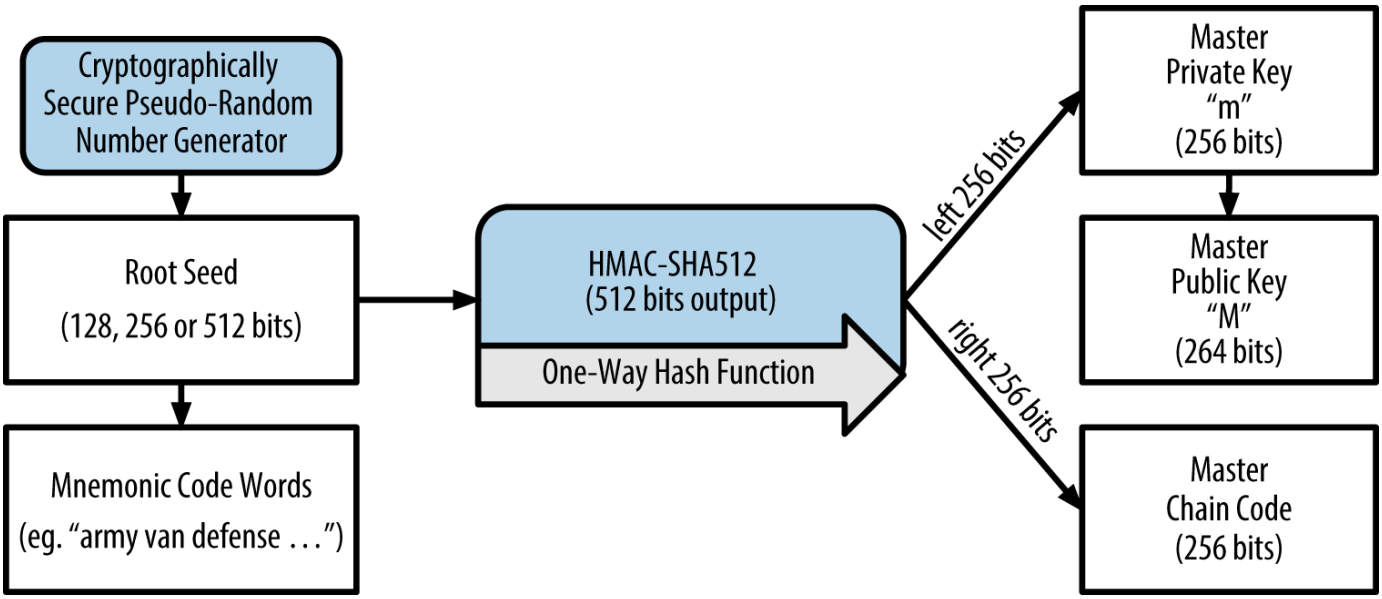


Figure 10. Creando claves y códigos de cadena maestros a partir de una semilla raíz

La semilla raíz es introducida en el algoritmo HMAC-SHA512 y el hash resultante se utiliza para crear una *clave privada maestra* (m) y un *código de cadena maestra*. La clave privada maestra (m) genera luego una clave pública maestra correspondiente (M), utilizando el proceso de multiplicación de curva elíptica normal m * G que vimos anteriormente en este capítulo. El código de cadena se utiliza para introducir entropía en la función que crea claves hijas a partir de claves padres, como veremos en la siguiente sección.

Derivación de la clave pública hija

Las carteras deterministas jerárquicas utilizan una función *de derivación de clave hija* (CKD) para derivar claves hijas de claves padres.

Las funciones de derivación de claves hijas se basan en una función de hash de sentido único que combina:

- Una clave privada o clave pública padre (clave descomprimida ECDSA)
- Una semilla llamada código de cadena (256 bits)
- Un número índice (32 bits)

El código de cadena se utiliza para introducir datos aparentemente aleatorios para el proceso, de modo que el índice no sea suficiente para derivar otras claves secundarias. Por lo tanto, tener una clave hija no permitirá encontrar sus hermanos, a menos que usted también tenga el código de cadena. La semilla del código de la cadena inicial (en la raíz del árbol) se obtiene a partir de datos aleatorios, mientras que los códigos de cadena posteriores se derivan de los códigos de cadena de cada padre.

Estos tres elementos son combinados y hasheados para generar claves hijas, como se ve a continuación.

La clave pública del padre, el código de cadena, y el número de índice se combinan y se hace hash con el algoritmo HMAC-SHA512 para producir un hash de 512 bits. El hash resultante se divide en dos mitades. La mitad derecha de 256 bits de la salida de hash se convirten en el código de cadena para el hijo. Los 256 bits de la mitad izquierda del hash y el número de índice se agregan a la clave privada de los padres para producir la clave privada del hijo. En Extendiendo una clave privada padre para crear una clave privada hijo., vemos esto ilustrado con el conjunto de índices a 0 para producir el hijo del padre de orden 0 (primero del índice).

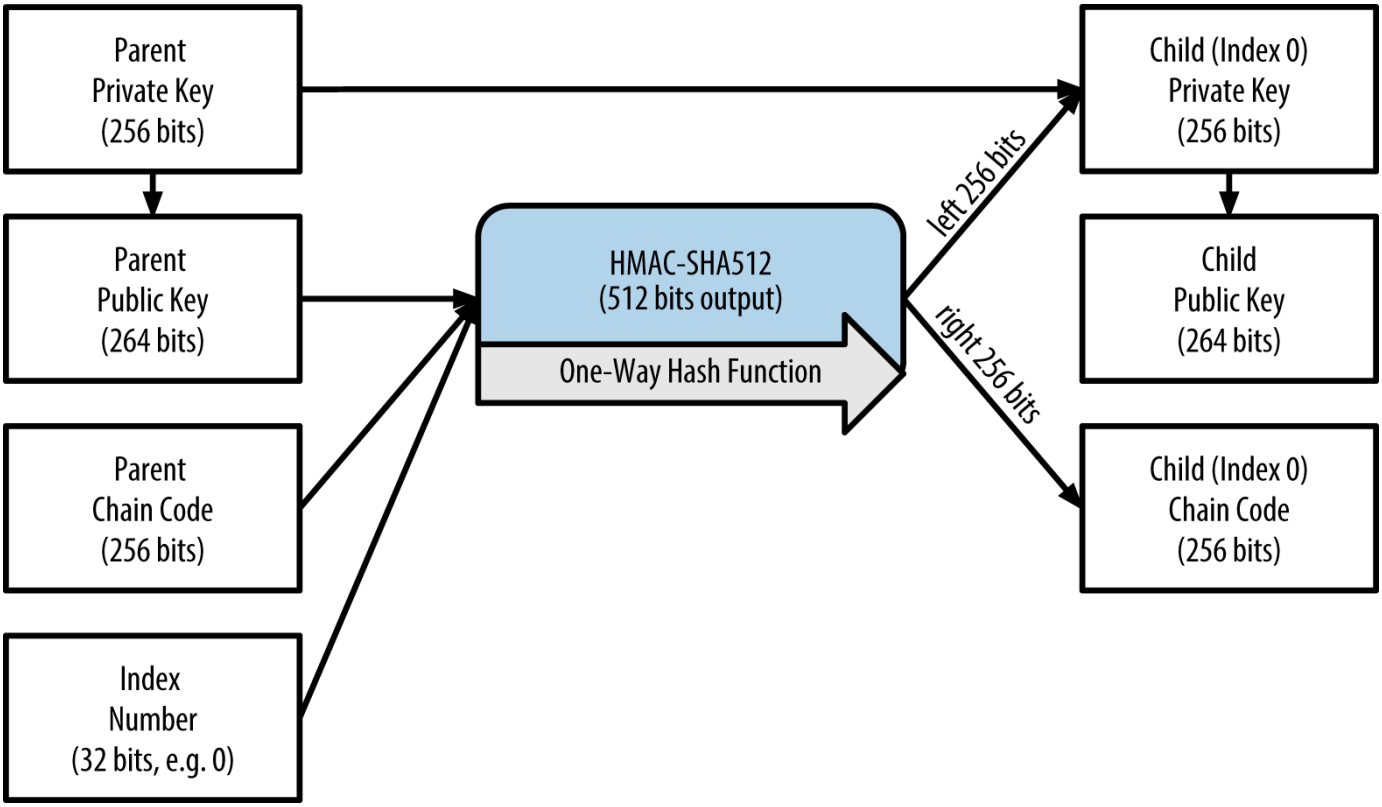


Figure 11. Extendiendo una clave privada padre para crear una clave privada hijo.

Cambiar el indice nos permite ampliar el padre y crear los otros hijos en la secuencia, por ejemplo, Hijo 0, Hijo 1, Hijo 2, etc. Cada clave padre puede tener 2 mil millones de claves hijas.

Repitiendo el proceso en un nivel inferior del árbol, cada hijo puede a su vez convertirse en un padre y crear sus propios hijos, en un número infinito de generaciones.

Usando claves hijas derivadas

Las claves privadas hijas son indistinguibles de las claves no deterministas (aleatorias). Debido a que la función de derivación es una función de un solo sentido, la clave hija no puede ser usada para encontrar la clave principal. La clave hija tampoco se puede utilizar para encontrar ningún hermano. Si usted tiene el hijo n-ésimo, usted no puede encontrar sus hermanos, como el hijo n-1 o el hijo n+1, o cualquier otro hijo que forme parte de la secuencia. Sólo la clave principal y el código de cadena pueden derivar todos los hijos. Sin el código de cadena del hijo, tampoco se puede usar la clave hija para derivar ninguno de los nietos. Es necesario tanto la clave privada del hijo como el código de cadena del hijo para iniciar una nueva rama y derivar nietos.

Entonces, ¿para qué se puede utilizar la clave privada hijo por sí sola? Se puede utilizar para crear una clave pública y una dirección bitcoin. Después, se puede utilizar para firmar transacciones para gastar lo que se haya pagado a esa dirección.

TIP

Una clave privada del hijo, la clave pública correspondiente, y la dirección bitcoin son indistinguibles de las claves y las direcciones creadas aleatoriamente. El hecho de que son parte de una secuencia no es visible, fuera de la función de la cartera HD que los creó. Una vez creadas, funcionan exactamente como claves "normales".

Claves extendidas

Como vimos anteriormente, la función de derivación de claves se puede utilizar para crear los hijos en cualquier nivel del árbol, sobre la base de las tres entradas: una clave, un código de cadena, y el índice del hijo deseado. Los dos ingredientes esenciales son la clave y el código de cadena, que cuando se combinan, forman lo que se llama una *clave extendida*. El término "clave extendida" también podría pensarse como "clave extensible" porque dicha clave se puede utilizar para crear los hijos.

Las claves extendidas se almacenan y se representan simplemente como la concatenación de la clave de 256 bits y el código de cadena de 256 bits en una secuencia de 512 bits. Hay dos tipos de claves extendidas. Una clave privada extendida es la combinación de una clave privada y el código de cadena, y se puede utilizar para derivar las claves privadas hijas (y a partir de ellas, las claves públicas hijas). Una clave pública extendida es una clave pública y el código de cadena, que puede utilizarse para crear las claves públicas hijas, como se describe en Generando una Clave Pública.

Piense en una clave extendida como el origen de una rama en la estructura de árbol de la cartera HD. Con el origen de la rama, puede derivar el resto de la rama. La clave privada extendida puede crear una rama completa, mientras que la clave pública extendida sólo puede crear una rama de claves públicas.

TIP

Una clave extendida consiste en una clave pública o privada y en un código de cadena. Una clave extendida puede crear hijos, generando su propia rama en la estructura de árbol. Compartir una clave extendida da acceso a toda la rama.

Las claves extendidas se codifican utilizando Base58Check, para facilitar la exportación e importación de diferentes carteras compatibles con BIP0032. La codificación Base58Check para las claves

extendidas utiliza un número de versión especial que se traduce en el prefijo "xprv" y "xpub" cuando se codifican en caracteres de Base58, para que sean fácilmente reconocibles. Dado que la clave extendida puede ser de 512 ó 513 bits, es también mucho más larga que otras cadenas codificadas en Base58Check que hemos visto anteriormente.

Aquí hay un ejemplo de una clave privada extendida, codificada en Base58Check:

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMK
Uga5biW6Hx4tws2six3b9c
```

Aquí está la clave privada extendida correspondiente, también codificada en Base58Check:

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJunSDMst
weyLXhRgPxdp14sk9tJPW9
```

Derivación de clave pública hija

Como se mencionó anteriormente, una característica muy útil de las carteras deterministas jerárquicas es la capacidad para derivar claves hijas públicas de las claves públicas de los padres, *sin* tener las claves privadas. Esto nos da dos maneras para obtener una clave pública hija: ya sea desde la clave privada hija, o directamente de la clave pública padre.

Una clave pública extendida puede usarse, por tanto, para derivar todas las claves *públicas* (y solamente las claves públicas) en esa rama de la estructura de la cartera HD.

Este método simplificado se puede utilizar para crear despliegues muy seguros en servidores que solo requieren de claves públicas, mediante una copia de una clave pública extendida, sin claves privadas de ningún tipo. Ese tipo de despliegue puede producir un número infinito de claves públicas y direcciones bitcoin, pero no se puede gastar el dinero enviado a esas direcciones. Mientras tanto, en otro servidor, más seguro, la clave privada extendida puede derivar todas las claves privadas correspondientes para firmar transacciones y gastar el dinero.

Una aplicación común de esta solución es instalar una clave pública extendida en un servidor web que sirve una aplicación de comercio electrónico. El servidor web puede utilizar la función de derivación de clave pública para crear una nueva dirección bitcoin en cada transacción (por ejemplo, para un carrito de la compra del cliente). El servidor web no tendrá ninguna clave privada, que serían vulnerables al robo. Sin carteras HD, la única manera de hacer esto seria generar miles de direcciones de Bitcoin en un servidor seguro por separado y luego cargarlas previamente en el servidor de comercio electrónico. Este enfoque es engorroso y requiere un mantenimiento constante para garantizar que el servidor de comercio electrónico no "agote" las claves.

Otra aplicación común de esta solución es el almacenamiento en frío o en carteras hardware. En este escenario, la clave privada extendida se puede almacenar en una cartera de papel o en un dispositivo de hardware (tal como una cartera hardware Trezor), mientras que la clave pública extendida puede

mantenerse en línea. El usuario puede crear direcciones de "recepción" a voluntad, mientras que las claves privadas se almacenan de forma segura en un lugar sin conexión. Para gastar los fondos, el usuario puede utilizar la clave privada extendida creando una firma en un cliente bitcoin sin conexión a la red, o firmar las transacciones en una cartera hardware (por ejemplo, Trezor). Extendiendo una clave pública padre para crear una clave pública hija ilustra el mecanismo para extender una clave pública padre para derivar claves públicas hijas.

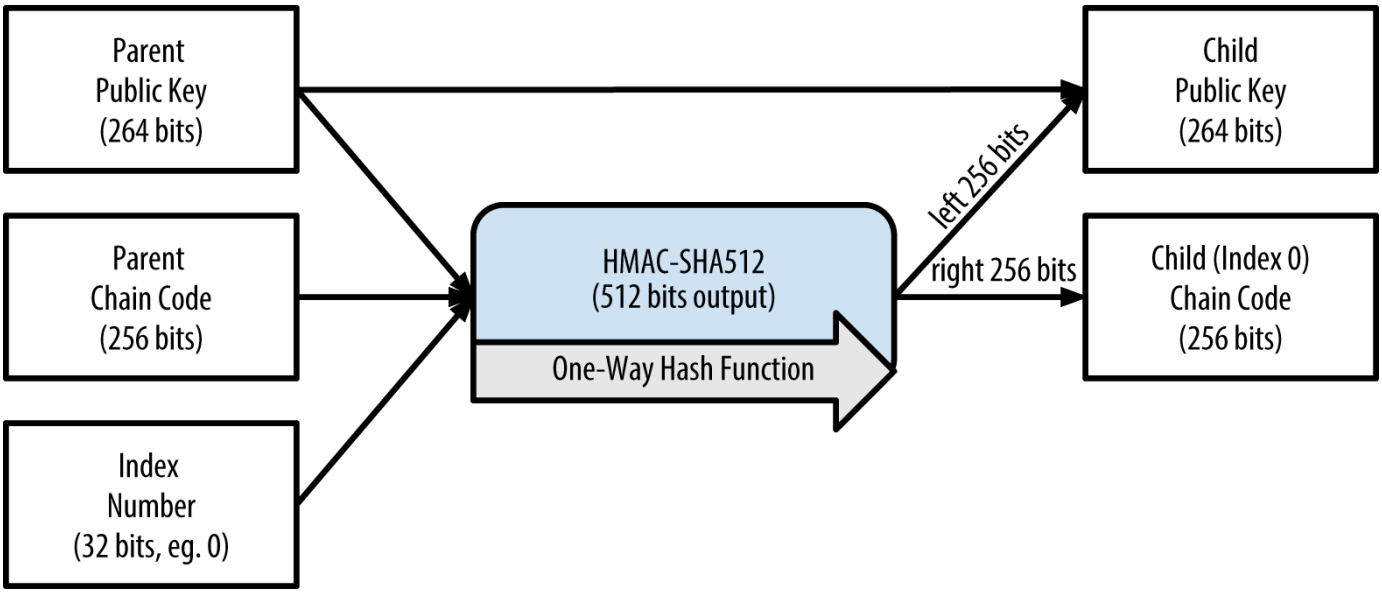


Figure 12. Extendiendo una clave pública padre para crear una clave pública hija

Derivación reforzada de claves hijas

La capacidad de derivar una rama de claves públicas de una clave pública extendida es muy útil, pero viene con un riesgo potencial. El acceso a una clave pública extendida no da acceso a las claves privadas hijas. Sin embargo, debido a que la clave pública extendida contiene el código de cadena, si se conoce una clave privada hija, o de alguna manera se filtró, se puede utilizar el código de cadena para derivar todas las otras claves privadas hijas. Una única clave privada hija filtrada, junto con un código de cadena padre, revela todas las claves privadas de todos los hijos. Peor aún, la clave privada hija junto con un código de cadena de los padres se puede utilizar para deducir la clave privada padre.

Para contrarrestar este riesgo, las carteras HD utilizan una función de derivación alternativa llamada *derivación reforzada*, que "rompe" la relación entre la clave pública padre y el código de cadena hijo. La función de derivación reforzada utiliza la clave privada padre para derivar el código de cadena hijo, en lugar de la clave pública padre. Esto crea un "cortafuegos" en la secuencia padre/hijo, con un código de cadena que no puede ser utilizado para comprometer un clave privada padre o hermana. La función de derivación reforzada parece casi idéntica a la derivación normal de la clave privada hija, a excepción de que la clave privada padre se utiliza como entrada a la función hash, en lugar de la clave pública padre, como se muestra en el diagrama en Derivación reforzada de una clave hija; omite la clave pública padre.

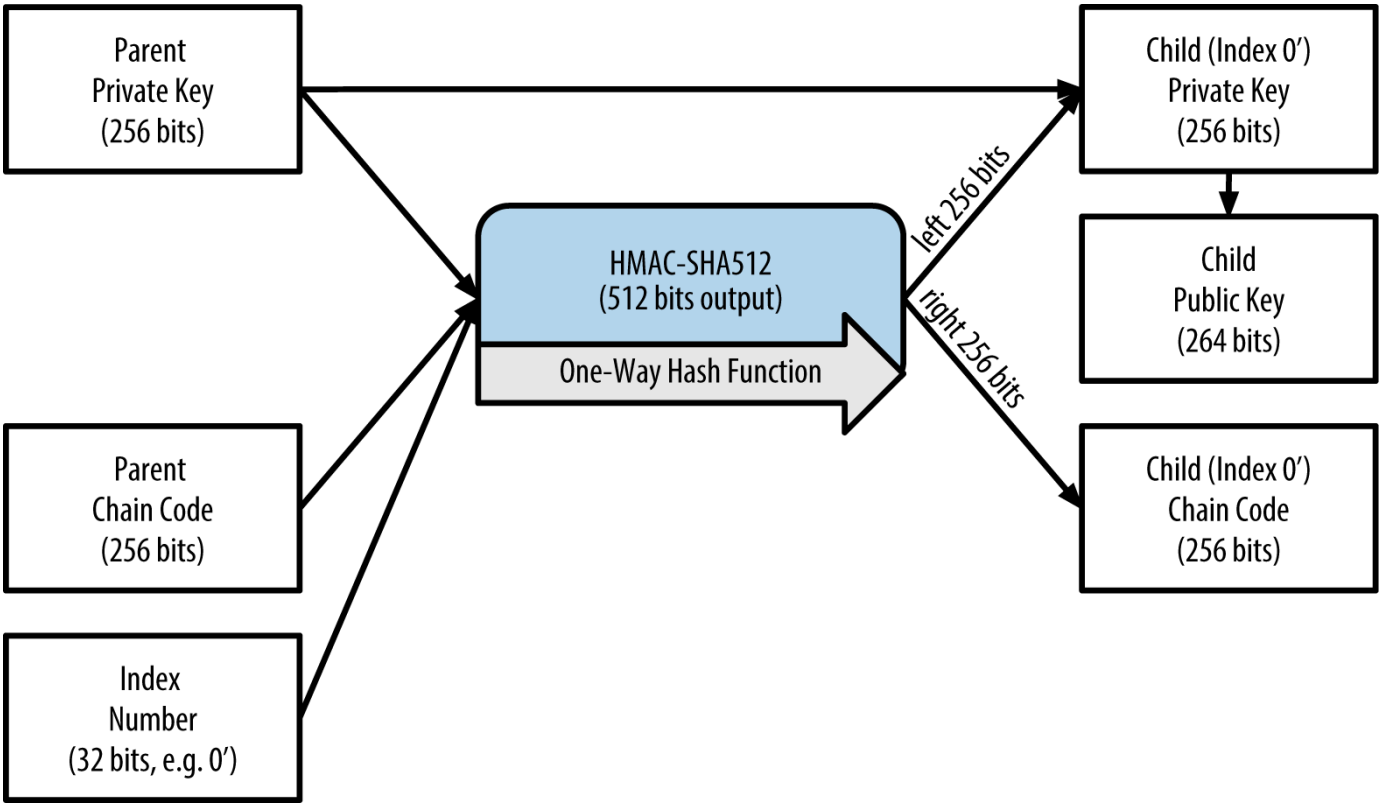


Figure 13. Derivación reforzada de una clave hija; omite la clave pública padre

Cuando se utiliza la función de derivación privada reforzada, la clave privada hija resultante y el código de cadena son completamente diferentes de lo que resultaría de la función normal de derivación. La "rama" resultante de las claves puede utilizarse para producir las claves públicas extendidas que no son vulnerables, debido a que el código de cadena que contienen no puede ser explotado para revelar ninguna clave privada. Por lo tanto, la derivación reforzada se utiliza para crear un "espacio" en el árbol por encima del nivel donde se utilizan las claves públicas extendidas.

En términos simples, si usted desea utilizar la conveniencia de una clave pública extendida para derivar ramas de claves públicas, sin exponerse al riesgo de que se difunda un código de cadena, debe derivarlo de un padre reforzado, en lugar de un padre normal. Como práctica recomendada, los hijos de nivel-1 de las claves maestras siempre deberian obtenerse a través de la derivación reforzada, para evitar el compromiso de las claves maestras.

Números índice para derivación normal y reforzada

El número de índice que se utiliza en la función de derivación es un entero de 32 bits. Para distinguir fácilmente entre claves derivadas a través de la función normal de derivación frente a claves derivadas a través de la derivación reforzada, este número de índice se divide en dos rangos. Los números de índice entre 0 y 2^{31}-1 (0x0 a 0x7FFFFFFF), se usan *solo* para la derivación normal. Números de índice entre 2^{31} y 2^{32}-1 (0x80000000 a 0xFFFFFFFF), se usan *solo* para la derivación reforzada. Por lo tanto, si el número de índice es menor que 2^{31}, eso significa que el hijo es normal, mientras que si el número de índice es igual o superior a 2^{31}, el hijo es reforzado.

Para que el número de índice sea más fácil de leer y de mostrar en pantalla, el número de índice para

los hijos reforzados se presenta empezando de cero, pero con un símbolo prima. Por tanto, la primera clave hija normal se muestra como 0, mientras que el primer hijo reforzado (índice 0x80000000) se muestra como <markup>0'</markup>. Continuando la secuencia, la segunda clave reforzada tendría índice 0x80000001 y se mostraría como 1 ', y así sucesivamente. Cuando vea un índice en una cartera HD i', significa 2³¹+i.

Identificador de clave de cartera HD (ruta)

Las claves en una cartera HD se identifican mediante un convenio de descripción de "ruta", con cada nivel del árbol separado por el carácter barra (/) (ver Ejemplos de rutas de cartera HD). Las claves privadas derivadas de la clave privada maestra empiezan con "m". Las claves públicas derivadas de la clave pública maestra empiezan con "M". Por lo tanto, la primera clave privada hija de la clave privada maestra es m/0. La primera clave pública hija es M/0. El segundo nieto del primer hijo es m/0/1, y así sucesivamente.

Los "antepasados" de una clave se leen de derecha a izquierda, hasta llegar a la clave maestra de la que deriva. Por ejemplo, el identificador m/x/y/z describe la clave que es el hijo z-ésimo de la clave m/x/y, que a su vez es el hijo y-ésimo de la clave m/x, que es el hijo x-ésimo de m.

Table 8. Ejemplos de rutas de cartera HD

Ruta HD	Clave descrita
m/0	La primera (0) clave privada hija de la clave privada maestra (m)
m/0/0	La primera clave privada nieta del primer hijo (m/0)
m/0'/0	El primer nieto normal del primer hijo *reforzado* (m/0')
m/1/0	La clave privada del primer nieto del segundo hijo (m/1)
M/23/17/0/0	La clave pública del primer tataranieto del primer bisnieto del nieto 18 del hijo 24

Navegando por la estructura de árbol de la cartera HD

La estructura de árbol de la cartera HD ofrece una gran flexibilidad. Cada clave extendida padre puede tener 4 mil millones de hijos: 2 mil millones de hijos normales y 2 mil millones de hijos reforzados. Cada uno de estos hijos puede tener otros 4 mil millones de hijos, y así sucesivamente. El árbol puede ser tan profundo como se desee, con un número infinito de generaciones. Con toda esta flexibilidad, sin embargo, se hace muy difícil de navegar por este árbol infinito. Es especialmente difícil para transferir carteras HD entre implementaciones, debido a que las posibilidades de organización interna en ramas principales y secundarias son infinitas.

Dos propuestas de mejora Bitcoin (BIPs) ofrecen una solución a esta complejidad, mediante la creación

de algunas de las normas propuestas para la estructura de los árboles de cartera HD. BIP0043 propone el uso del primer índice hijo reforzado como un identificador especial que significa el "propósito" de la estructura de árbol. Basado en BIP0043, una cartera HD debería utilizar solo una rama del árbol de nivel-1, con el número de índice identificando la estructura y el espacio de nombres del resto del árbol mediante la definición de su propósito. Por ejemplo, una cartera HD que utilice una única rama m/i'/ intenta significar un propósito específico y ese propósito es identificado por el número de índice "i".

Ampliando esa especificación, BIP0044 propone una estructura multicuenta cuyo "objetivo" es el número 44' bajo BIP0043. Todas las carteras HD que cumplen con la estructura BIP0044 se identifican por el hecho de que sólo utilizan una rama del árbol: m/44'/.

BIP0044 especifica que la estructura se basa en cinco niveles predefinidos del árbol:

+ m / propósito' / tipo_moneda '/ cuenta' / cambio / indice_dirección+

El primer nivel "propósito" está siempre ajustado a 44'. El segundo nivel "tipo_moneda" especifica el tipo de moneda criptomoneda, permitiendo carteras HD multidivisa donde cada moneda tiene su propio sub-árbol bajo el segundo nivel. Hay tres monedas definidas por ahora: Bitcoin es m/44'/0', Bitcoin Testnet es <markup>m/44'/1'</markup>; y Litecoin es <markup>m/44'/2'</markup>.

El tercer nivel del árbol es "cuenta", que permite a los usuarios que subdividan sus carteras en subcuentas lógicas separadas, para la contabilidad o para propósitos organizativos. Por ejemplo, una billetera HD puede contener dos "cuentas" bitcoin: pass: [<markup>m/44'/0'/0'</markup>] y <markup>m/44'/0'/1'</markup>. Cada cuenta es la raíz de su propio subárbol.

En el cuarto nivel, "cambio", una cartera HD tiene dos subárboles, uno para la creación de direcciones que reciben y otro para la creación de direcciones de cambio. Tenga en cuenta que mientras que los niveles anteriores utilizaron derivación reforzada, este nivel utiliza derivación normal. Esto se hace para permitir que este nivel del árbol pueda exportar las claves públicas extendidas para el uso en un entorno no seguro. Las direcciones utilizables se derivan de la cartera HD como hijos del cuarto nivel, haciendo que el quinto nivel del árbol sea el "indice_dirección". Por ejemplo, la tercera dirección de recepción para los pagos bitcoin en la cuenta principal sería M/44'/0'/0'/0/2. La Ejemplos de estructuras de carteras HD BIP0044 muestra algunos ejemplos más.

Table 9. Ejemplos de estructuras de carteras HD BIP0044

Ruta HD	Clave descrita
M/44'/0'/0'/0/2	La tercera clave pública receptora para la cuenta bitcoin primaria
M/44'/0'/3'/1/14	La decimoquinta clave pública de la dirección de cambio para la cuarta cuenta bitcoin
m/44'/2'/0'/0/1	La segunda clave privada en la cuenta principal litecoin, para las transacciones de firma

Experimentando con carteras HD usando Bitcoin Explorer

("Bitcoin Explorer","carteras HD y"Mediante la herramienta de línea de comandos Bitcoin Explorer introducida en el capítulo [ch03_bitcoin_client], se puede experimentar con la generación y ampliación de claves deterministas de BIP0032, así como su visualización en diferentes formatos:

```
$ bx seed | bx hd-new > m # crear una nueva clave privada maestra a partir de una
semilla y almacenarla en el archivo "m"
$ cat m # mostrar la clave privada extendida maestra
xprv9s21ZrQH143K38iQ9Y5p6qoB8C75TE71NfpyQPdfGvzghDt39DHPFpovvtWZaRgY5uPwV7RpEgHs7cvdg
fiSjLjjbuGKGcjRyU7RGGSS8Xa
$ cat m | bx hd-public # generar la clave pública extendida M/0
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJunS
DMstweyLXhRgPxdp14sk9tJPW9
$ cat m | bx hd-private # generar la clave privada extendida m/0
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6G
oNMKUga5biW6Hx4tws2six3b9c
$ cat m | bx hd-private | bx hd-to-wif # mostrar la clave privada de m/0 como un WIF
L1pbvV86crAGoDzqmgY85xURkz3c435Z9nirMt52UbnGjYMzKBUN
$ cat m | bx hd-public | bx hd-to-address # mostrar la dirección bitcoin de M/0
1CHCnCjgMNb6digimckNQ6TBVcTWBAmPHK
$ cat m | bx hd-private | bx hd-private --index 12 --hard | bx hd-private --index 4 #
generar m/0/12'/4
xprv9yL8ndfdPVeDWJenF18oiHguRUj8jHmVrqqD97YQHeTcR3LCeh53q5PXPkLsy2kRaqgwoS6YZBLatRZRy
UeAkRPe1kLR1P6Mn7jUrXFquUt
```

Claves y Direcciones Avanzadas

En las siguientes secciones veremos formas avanzadas de claves y direcciones, tales como claves privadas encriptadas, direcciones de scripts y multifirma, direcciones de vanidad y carteras de papel.

Claves Privadas Encriptadas (BIP0038)

Las claves privadas deben mantenerse en secreto. La necesidad de *confidencialidad* de las claves privadas es una perogrullada que es difícil de lograr en la práctica, porque entra en conflicto con el igualmente importante objetivo de *disponibilidad*. Mantener en privado la clave privada es mucho más difícil cuando se necesita almacenar copias de seguridad de la clave privada para evitar perderla. Una clave privada almacenada en una cartera que se cifre con una contraseña puede ser segura, pero se deben hacer copias de respaldo. En ocasiones, los usuarios necesitan mover las claves de una cartera a otra —para actualizar o reemplazar el software de la cartera, por ejemplo. Las copias de seguridad de claves privadas también podrían guardarse en papel (ver Carteras de Papel) o en medios de almacenamiento externo, como una unidad flash USB. Pero ¿y si la propia copia de seguridad se pierde o es robada? Estos conflictos en los objetivos de seguridad llevaron a la introducción de un

estándar compatible y práctico para el cifrado de claves privadas que puede ser entendido por muchas carteras y clientes bitcoin diferentes, estandarizada en la Propuesta de Mejora Bitcoin 38 o BIP0038 (ver [bip0038]).

BIP0038 propone una norma común para el cifrado de claves privadas con una contraseña larga y codificado con Base58Check para que puedan almacenarse de forma segura en cualquier medio utilizado para la copia de seguridad, transportarse de forma segura entre carteras, o guardarse en situaciones donde la clave pueda estar expuesta. El estándar para el cifrado utiliza el Advanced Encryption Standard (AES), un estándar establecido por el Instituto Nacional de Estándares y Tecnología (NIST) y se utiliza ampliamente en las implementaciones de cifrado de datos comerciales y aplicaciones militares.

Un esquema de encriptación BIP0038 toma como entrada una clave privada bitcoin, generalmente codificada en el formato de importación Wallet (WIF), como una cadena Base58Check con un prefijo de "5". Además, el esquema de encriptación BIP0038 toma una frase de paso —contraseña larga— generalmente compuesta de varias palabras o una cadena compleja de caracteres alfanuméricos. El resultado del esquema de encriptación BIP0038 es una clave privada encriptada con codificación Base58Check que comienza con el prefijo 6P. Si ve una clave que comienza con 6P, significa que está encriptada y requiere una contraseña para convertir (descifrar) de nuevo en una clave privada con formato WIF (prefijo 5) para que se pueda utilizar en cualquier cartera. Muchas aplicaciones de cartera ahora reconocen las claves privadas cifradas-BIP0038 y se solicitará al usuario una contraseña para descifrar e importar la clave. Las aplicaciones de terceros, como el increíblemente útil Bit Address (Pestaña Wallet Details), se puede utilizar para descifrar claves BIP0038.

El caso de uso más común para claves encriptadas en BIP0038 es para carteras de papel que se pueden utilizar como copia de seguridad de las claves privadas en un pedazo de papel. Siempre y cuando el usuario seleccione una frase fuerte como contraseña, una billetera de papel con claves privadas encriptada de BIP0038 es increíblemente segura y una gran manera de crear el almacenamiento bitcoin fuera de línea (también conocido como "almacenamiento en frío").

Pruebe las claves encriptadas en Ejemplo de una clave privada encriptada BIP0038 usando bitaddress.org para ver cómo puede obtener la clave desencriptada ingresando la frase secreta.

Table 10. Ejemplo de una clave privada encriptada BIP0038

Clave Privda (WIF)	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Frase secreta	MyTestPassphrase
Clave Encriptada (BIP0038)	6PRTHL6mWa48xSopbU1cKrVjpKbBZxcLRRCdctLJ3z5yxE87MobKoXdTsJ

Direcciones de Pago-a-Hash-de-Script (P2SH) y Multi-Firma

Como sabemos, tradicionalmente las direcciones de Bitcoin empiezan con el número "1" y se derivan de la clave pública, que se deriva de la clave privada. Aunque cualquier persona puede enviar a una

dirección bitcoin "1", los bitcoin solo pueden gastarse mediante la presentación de la correspondiente firma de clave privada y hash de clave pública.

Las direcciones Bitcoin que empiezan con el número "3" son direcciones pago-a-script-hash (P2SH, pay-to-script-hash), a veces erróneamente llamadas multi-firma o direcciones multi-sig. Designan al beneficiario de una transacción bitcoin como el hash de un script, en lugar del propietario de una clave pública. La función se introdujo en enero de 2012 como Propuesta de Mejora Bitcoin 16 o BIP0016 (ver [bip0016]) y está siendo ampliamente adoptado, ya que proporciona la oportunidad de agregar funcionalidad a la dirección en sí misma. A diferencia de las transacciones que "envían" fondos para las direcciones tradicionales "1" de bitcoin, también conocidas como pago-a-clave-pública-hash (P2PKH, pay-to-public-key-hash), los fondos enviados a las direcciones de "3" requieren algo más que la presentación de un hash de clave pública, y una clave privada de firma como prueba de propiedad. Los requisitos se designan en el momento en que se crea la dirección, dentro del script, y todas las entradas a esta dirección serán bloqueadas con los mismos requisitos.

Una dirección hash pay-to-script se crea a partir de un script de transacción, que define quién puede gastar una salida de transacción (para más detalles, consulte [p2sh]). La codificación de una dirección hash de pay-to-script implica usar la misma función doble-hash que se utilizó durante la creación de una dirección bitcoin, solo que ahora se aplica al script en lugar de a la clave pública:

```
hash de script = RIPEMD160(SHA256(script))
```

El "hash del script" resultante está codificado con Base58Check con un prefijo de versión de valor 5, lo que resulta en una dirección codificada que comienza con un 3. Un ejemplo de una dirección P2SH es 3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM, que se puede derivar mediante los comandos del Explorador de Bitcoin ("comando script-encode (bx)") script-encode, sha256, ("Bitcoin Explorer", "comando ripemd160") ripemd160 y ("comando base58check-encode (bx)") base58check-encode (ver [libbitcoin]) como sigue:

```
$ echo dup hash160 [ 89abcdefabbaabbaabbaabbaabbaabbaabbaabba ] equalverify checksig >
script
$ bx script-encode < script | bx sha256 | bx ripemd160 | bx base58check-encode --version
5
3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM
```

TIP P2SH no es necesariamente lo mismo que una transacción multi-firma estándar. Una dirección P2SH representa *la mayor parte de las veces* un script multi-firma, pero también sería posible representar un script que codifique otros tipos de transacciones.

Direcciones multi-firma y P2SH

Actualmente, la aplicación más común de la función P2SH es el script de dirección multifirma. Como su nombre indica, el script subyacente requiere más de una firma para demostrar la propiedad y por lo

tanto poder gastar los fondos. La multifirma de bitcoin se distingue porque está diseñada para requerir M firmas (también conocido como el "umbral") de un total de N claves, conocido como un multifirma M-de-N, donde M es igual o inferior a N. Por ejemplo, Bob, el dueño de la cafetería del [ch01_intro_what_is_bitcoin] podría utilizar una dirección multifirma que requiera de 1-de-2 firmas, una de las claves de su propiedad y la otra clave perteneciente a su cónyuge, garantizando que cualquiera de los dos pueda firmar para gastar una salida de transacción que se encuentre bloqueada en esta dirección. Esto sería similar a una "cuenta conjunta" tal como se aplica en la banca tradicional, donde cualquiera de los cónyuges puede transferir con una sola firma. O Gopesh, el diseñador web pagado por Bob para crear un sitio web, podría tener una dirección multifirma 2-de-3 para su negocio que garantice que no se pueden gastar los fondos a menos que dos de los socios firmen la transacción.

Exploraremos cómo crear transacciones que gastan fondos de direcciones P2SH (y multifirma) en [transactions].

Direcciones de Vanidad

Las direcciones de vanidad son direcciones bitcoin válidas que contienen mensajes legibles. Por ejemplo, 1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33 es una dirección válida que contiene las letras que forman la palabra "Love" con las primeras cuatro letras en Base-58. Las direcciones de vanidad requieren generar y comprobar miles de millones de claves privadas candidatas, hasta que de una se derive una dirección bitcoin con el patrón deseado. Aunque hay algunas optimizaciones en el algoritmo de generación de la vanidad, el proceso implica esencialmente que escoge una clave privada al azar, derivando la clave pública, derivando la dirección bitcoin, y comprobando si coincide con el patrón de vanidad deseado, repitiendo miles de millones de veces hasta encontrar una coincidencia.

Una vez que se encuentra una dirección de vanidad que coincida con el patrón deseado, la clave privada de la que se deriva puede ser utilizada por el propietario para gastar bitcoins exactamente de la misma manera que cualquier otra dirección. Las direcciones de vanidad no son menos o más seguras que cualquier otra dirección. Dependen de la misma criptografía de curva elíptica (ECC) y Secure Hash Algorithm (SHA) de cualquier otra dirección. No es más fácil encontrar la clave privada de una dirección que comienza con un patrón de vanidad de lo que sería cualquier otra dirección.

En [ch01_intro_what_is_bitcoin], presentamos a Eugenia, directora de caridad para niños que funciona en las Filipinas. Digamos que Eugenia está organizando una unidad de recaudación de fondos en bitcoin y que quiere utilizar una dirección bitcoin de vanidad para dar a conocer la recaudación de fondos. Eugenia creará una dirección de vanidad que comience por "1Kids" con ese propósito. Vamos a ver cómo se crea esta dirección de vanidad y lo que significa para la seguridad de la obra benéfica de Eugenia.

Generando direcciones de vanidad

Es importante tener en cuenta que una dirección bitcoin es simplemente un número representado por símbolos en el alfabeto Base58. La búsqueda de un patrón como "1Kids" puede entenderse como la búsqueda de una dirección en el rango de 1Kids111111111111111111111111111111 a 1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzzz. Hay aproximadamente 58^{29} (aproximadamente 1,4 * 10^{51}) direcciones en ese rango, todas ellas comenzando por "1Kids". La El rango de direcciones de vanidad

comenzando con "1Kids" muestra el rango de direcciones que tienen el prefijo 1Kids.

Table 11. El rango de direcciones de vanidad comenzando con "1Kids"

Desde	1Kids111111111111111111111111111
	1Kids111111111111111111111111112
	1Kids111111111111111111111111113
	...
Hasta	1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzz

Veamos el patrón "1Kids" como un número y la frecuencia con la que podríamos encontrar este patrón en una dirección bitcoin (ver La frecuencia del patrón de vanidad (1KidsCharity) y el tiempo de búsqueda promedio en una PC de escritorio). De promedio, un ordenador de sobremesa sin ningún hardware especializado puede buscar aproximadamente 100.000 claves por segundo.

Table 12. La frecuencia del patrón de vanidad (1KidsCharity) y el tiempo de búsqueda promedio en una PC de escritorio

Longitud	Patrón	Frecuencia	Tiempo de búsqueda promedio
1	1K	1 en 58 claves	< 1 milisegundos
2	1Ki	1 en 3.364	50 milisegundos
3	1Kid	1 en 195.000	< 2 segundos
4	1Kids	1 en 11 millones	1 minuto
5	1KidsC	1 en 656 millones	1 hora
6	1KidsCh	1 en 38 mil millones	2 días
7	1KidsCha	1 en 2.2 billones	3–4 meses
8	1KidsChar	1 en 128 billones	13–18 años
9	1KidsChari	1 en 7 mil billones	800 años
10	1KidsCharit	1 en 400 mil billones	46.000 años
11	1KidsCharity	1 en 23 trillones	2,5 millones de años

Como se puede ver, Eugenia no podrá crear la dirección de vanidad "1KidsCharity" en el corto plazo, incluso si ella tuviese acceso a varios miles de ordenadores. Cada caracter adicional aumenta la dificultad en un factor de 58. Los patrones con más de siete caracteres se encuentran generalmente con hardware especializado, tales como escritorios hechos a medida con múltiples unidades de procesamiento de gráficos (GPU). Estos son a menudo reutilizados por "plataformas" mineras de bitcoin que ya no son rentables para la minería bitcoin, pero que se pueden utilizar para encontrar direcciones de vanidad. Las búsquedas de vanidad sobre sistemas de GPU son varios órdenes de

magnitud más rápidas que las de una CPU de propósito general.

Otra manera de encontrar una dirección de vanidad es la de subcontratar el trabajo a un grupo de mineros de vanidad, como en el pool Vanity pool. Un pool es un servicio que permite ganar bitcoin a las personas con hardware de GPU buscando direcciones de vanidad para los demás. Por un pequeño pago (0,01 bitcoin o aproximadamente $5 en el momento de escribir esto), Eugenia puede externalizar la búsqueda de una dirección con un patrón de vanidad de siete caracteres y obtener resultados en un par de horas en lugar de tener que realizar una búsqueda de CPU durante meses.

Generar una dirección de vanidad es un ejercicio de fuerza bruta: probar una clave aleatoria, compruebar la dirección resultante para ver si coincide con el patrón deseado, repetir hasta que tenga éxito. Minero de direcciones de vanidad muestra un ejemplo de un "minero de vanidad", un programa diseñado para encontrar direcciones de vanidad, escrito en C++. El ejemplo utiliza la biblioteca libbitcoin, que ya se presentó en [alt_libraries].

Example 8. Minero de direcciones de vanidad

```
#include <bitcoin/bitcoin.hpp>

// The string we are searching for
const std::string search = "1kid";

// Generate a random secret key. A random 32 bytes.
bc::ec_secret random_secret(std::default_random_engine& engine);
// Extract the Bitcoin address from an EC secret.
std::string bitcoin_address(const bc::ec_secret& secret);
// Case insensitive comparison with the search string.
bool match_found(const std::string& address);

int main()
{
    // random_device on Linux uses "/dev/urandom"
    // CAUTION: Depending on implementation this RNG may not be secure enough!
    // Do not use vanity keys generated by this example in production
    std::random_device random;
    std::default_random_engine engine(random());

    // Loop continuously...
    while (true)
    {
        // Generate a random secret.
        bc::ec_secret secret = random_secret(engine);
        // Get the address.
        std::string address = bitcoin_address(secret);
        // Does it match our search string? (1kid)
        if (match_found(address))
```

```
        {
            // Success!
            std::cout << "Found vanity address! " << address << std::endl;
            std::cout << "Secret: " << bc::encode_hex(secret) << std::endl;
            return 0;
        }
    }
    // Should never reach here!
    return 0;
}

bc::ec_secret random_secret(std::default_random_engine& engine)
{
    // Create new secret...
    bc::ec_secret secret;
    // Iterate through every byte setting a random value...
    for (uint8_t& byte: secret)
        byte = engine() % std::numeric_limits<uint8_t>::max();
    // Return result.
    return secret;
}

std::string bitcoin_address(const bc::ec_secret& secret)
{
    // Convert secret to pubkey...
    bc::ec_point pubkey = bc::secret_to_public_key(secret);
    // Finally create address.
    bc::payment_address payaddr;
    bc::set_public_key(payaddr, pubkey);
    // Return encoded form.
    return payaddr.encoded();
}

bool match_found(const std::string& address)
{
    auto addr_it = address.begin();
    // Loop through the search string comparing it to the lower case
    // character of the supplied address.
    for (auto it = search.begin(); it != search.end(); ++it, ++addr_it)
        if (*it != std::tolower(*addr_it))
            return false;
    // Reached end of search string, so address matches.
    return true;
}
```

El ejemplo anterior usa std::random_device. Dependiendo de la implementación puede reflejar un generador de números aleatorios criptográficamente seguro (CSRNG) proporcionado por el sistema operativo subyacente. En el caso de los sistemas operativos basados en UNIX, como Linux, se nutre de /dev/urandom. El generador de números aleatorios utilizado aquí es para fines de demostración y *no* es apropiado para generar claves Bitcoin de calidad de producción, ya que no está implementado con la suficiente seguridad.

El código de ejemplo debe ser compilado usando un compilador de C y enlazado con la biblioteca libbitcoin (que debe ser instalado por primera vez en ese sistema). Para ejecutar el ejemplo, lance el ejecutable vanity-miner++ sin parámetros (ver Compilando y ejecutando el ejemplo de vanity-miner) e intentará encontrar una dirección de vanidad que empiece por "1kid".

Example 9. Compilando y ejecutando el ejemplo de vanity-miner

```
$ # Compilar el código con g++
$ g++ -o vanity-miner vanity-miner.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Ejecutar el ejemplo
$ ./vanity-miner
Found vanity address! 1KiDzkG4MxmovZryZRj8tK81oQRhbZ46YT
Secret: 57cc268a05f83a23ac9d930bc8565bac4e277055f4794cbd1a39e5e71c038f3f
$ # Ejecutarlo otra vez para obtener un resultado distinto
$ ./vanity-miner
Found vanity address! 1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUFn
Secret: 7f65bbbbe6d8caae74a0c6a0d2d7b5c6663d71b60337299a1a2cf34c04b2a623
# Usar "time" para ver cuánto tarda en encontrar un resultado
$ time ./vanity-miner
Found vanity address! 1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM
Secret: 2a802e7a53d8aa237cd059377b616d2bfcfa4b0140bc85fa008f2d3d4b225349

real    0m8.868s
user    0m8.828s
sys 0m0.035s
```

El código de ejemplo tardará unos segundos en encontrar una coincidencia para el patrón de tres caracteres "kid", como podemos ver cuando usamos el comando time de Unix para medir el tiempo de ejecución. Cambie el patrón de búsqueda en el apartado search del código fuente y ¡vea cuánto tiempo más se tarda entre un patrón de cuatro caracteres y otro de cinco!

Seguridad de direcciones de vanidad

Las direcciones de vanidad pueden utilizarse para mejorar *y* para vencer a las medidas de seguridad; son realmente un arma de doble filo. Cuando se utiliza para mejorar la seguridad, una dirección distintiva hace que sea más difícil para los adversarios sustituirla por su propia dirección y engañar así

a los clientes para que les paguen a ellos en lugar de a usted. Desafortunadamente, las direcciones de vanidad también hacen posible que cualquier persona pueda crear una dirección que *se asemeje* a cualquier dirección aleatoria, o incluso a otra dirección de vanidad, engañando de esta manera a sus clientes.

Eugenia podría publicitar una dirección generada aleatoriamente (por ejemplo, 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy) a la cual la gente podría enviar sus donaciones. O podría generar una dirección de vanidad que comience con 1Kids para hacerla más distintiva.

En ambos casos, uno de los riesgos del uso de una dirección fija única (en lugar de una dirección dinámica separada por donante) es que un ladrón podría ser capaz de infiltrarse en su sitio web y reemplazarla con su propia dirección, desviando así las donaciones a sí mismo. Si ha anunciado su dirección de donación en diferentes lugares, los usuarios pueden inspeccionar visualmente la dirección antes de hacer un pago para asegurarse de que es la misma que vieron en su sitio web, en su correo electrónico y en su propaganda. En el caso de una dirección aleatoria como 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy, el usuario medio querrá quizá inspeccionar los primeros caracteres "1J7mdg" y estar convencido de que la dirección coinde. Mediante el uso de un generador de direcciones de vanidad, una persona con la intención de robar podría sustituir la dirección original con otra de aspecto similar, generada rápidamente mediante la coincidencia en sus primeros caracteres, como se muestra en Generando direcciones de vanidad para coincidir con una dirección aleatoria.

Table 13. Generando direcciones de vanidad para coincidir con una dirección aleatoria

Dirección Aleatoria Original	1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
Vanidad (coincidencia de 4 carácteres)	1J7md1QqU4LpctBetHS2ZoyLV5d6dShhEy
Vanidad (coincidencia de 5 carácteres)	1J7mdgYqyNd4ya3UEcq31Q7sqRMXw2XZ6n
Vanidad (coincidencia de 6 carácteres)	1J7mdg5WxGENmwyJP9xuGhG5KRzu99BBCX

Entonces, ¿una dirección de vanidad aumenta la seguridad? Si Eugenia genera la dirección de vanidad 1Kids33q44erFfpeXrmDSz7zEqG2FesZEN, los usuarios tenderán a mirar la palabra del patrón de vanidad y algunos caracteres más allá, por ejemplo, notando la parte "1Kids33" de la dirección. Eso obligaría a un atacante a generar una dirección de vanidad de al menos seis caracteres (dos más), gastando un esfuerzo que es 3364 veces (58 × 58) más alto que el que Eugenia gastó para su vanidad de cuatro caracteres. En esencia, el esfuerzo que Eugenia gastó (o pagó a un pool de vanidad) "empuja" al atacante a tener que producir un patrón de vanidad de mayor longitud. Si Eugenia paga a un pool para generar una dirección de vanidad de 8 caracteres, el atacante podría verse empujado a buscar una de 10 caracteres, que es inviable en un ordenador personal y caro incluso con un equipo personalizado para minería de vanidad o con una pool de vanidad. Lo que es asequible para Eugenia se convierte en inaccesible para el atacante, especialmente si la recompensa potencial de fraude no es lo suficientemente alta para cubrir el costo de la generación de direcciones vanidad.

Carteras de Papel

Las carteras de papel son claves privadas de bitcoin impresas en papel. A menudo, la cartera de papel también incluye la dirección bitcoin correspondiente por conveniencia, pero esto no es necesario, ya que puede ser derivada de la clave privada. Las carteras de papel son una forma muy efectiva para crear copias de seguridad o almacenamiento sin conexión bitcoin, también conocido como "almacenamiento en frío." Como un mecanismo de copia de seguridad, una billetera de papel puede proporcionar seguridad contra la pérdida de la clave debido a un percance informático como un fallo de disco duro, robo o eliminación accidental. El mecanismo de "almacenamiento en frío" es mucho más seguro contra hackers, keyloggers y otras amenazas informáticas en línea, si las claves de la cartera de papel se generan fuera de línea y no son almacenadas en ningún sistema informático, .

Las carteras de papel vienen en muchas formas, tamaños y diseños, pero a un nivel muy básico son simplemente una clave y una dirección impresas en papel. La forma más simple de una cartera de papel—una impresión de la dirección bitcoin y clave privada. muestra la forma más sencilla de una cartera de papel.

Table 14. La forma más simple de una cartera de papel—una impresión de la dirección bitcoin y clave privada.

Dirección Pública	Clave Privada (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

Las carteras de papel se pueden generar fácilmente utilizando una herramienta web JavaScript en *bitaddress.org*. Esta página contiene todo el código necesario para generar claves y carteras de papel, incluso completamente desconectado de Internet. Para usarlo, guarde la página HTML en la unidad local o en una unidad flash USB externa. Desconéctese de Internet y abra el archivo en un navegador. Aún mejor, arranque el ordenador utilizando un sistema operativo original, como por ejemplo, un CD-ROM de arranque del sistema operativo Linux. Cualquier clave generada con esta herramienta sin conexión se puede imprimir en una impresora local mediante un cable USB (no inalámbrica), creando así carteras de papel cuyas claves sólo existen en el papel y nunca han sido almacenados en ningún sistema en línea. Para implementar una solución sencilla pero muy eficaz de "almacenamiento en frío", ponga esas carteras de papel en una caja fuerte a prueba de fuego y "envíe" bitcoins a su dirección bitcoin. Un ejemplo de una cartera de papel simple de bitaddress.org Muestra una cartera de papel generada desde el sitio bitaddress.org.

Figure 14. Un ejemplo de una cartera de papel simple de bitaddress.org

La desventaja del sistema de cartera papel simple es que las claves impresas son vulnerables al robo. Un ladrón que es capaz de tener acceso al papel puede robar o fotografiar las claves y tomar el control de los bitcoins bloqueados con dichas claves. Un sistema de almacenamiento de la cartera de papel más sofisticado se utiliza en BIP0038 mediante el uso de claves privadas encriptadas. Las claves impresas en la cartera de papel están protegidas por una contraseña que el propietario ha memorizado. Sin la contraseña, las claves cifradas son inútiles. Sin embargo, todavía son superiores a una cartera con una frase de contraseña-protegida porque las claves nunca han estado en línea y deben ser recuperadas físicamente de un almacenamiento seguro o protegido físicamente. Un ejemplo de una cartera de papel encriptada de bitaddress.org. La frase secreta es "test." muestra una cartera de papel con una clave privada encriptada (BIP0038) creada en el sitio bitaddress.org.

Figure 15. Un ejemplo de una cartera de papel encriptada de bitaddress.org. La frase secreta es "test."

Aunque se pueden depositar fondos varias veces en una cartera de papel, se deben retirar todos los fondos de una sola vez, gastándolo todo. Esto se debe a que en el proceso de desbloqueo de los fondos y de gasto, algunas carteras podrían generar una dirección de cambio si se gasta menos de la totalidad del importe. Además, si el equipo que se utiliza para firmar la transacción se ve comprometido, corre el riesgo de exponer la clave privada. Al gastar la totalidad del saldo de una cartera de papel solo una vez, se reduce el riesgo de compromiso de la clave. Si necesita solo una pequeña cantidad, envíe los fondos restantes a una nueva cartera de papel en la misma transacción.

Las carteras de papel vienen en muchos diseños y tamaños, con muchas características diferentes. Algunas están destinadas a ser dadas como regalos y tienen temas estacionales, como la Navidad y temas de Año Nuevo. Otras están diseñadas para el almacenamiento en una bóveda bancaria o caja de seguridad con la clave privada oculta de alguna manera, ya sea con pegatinas-rasca opacas o plegados y sellados con una lámina adhesiva a prueba de manipulaciones. Las imágenes de <xref linkend="paper_wallet_bpw" xrefstyle="select: labelnumber"/> a pass: [<xref linkend="paper_wallet_spw" xrefstyle="select: labelnumber"/>] muestran varios ejemplos de carteras de papel con características de seguridad y de copia de respaldo.

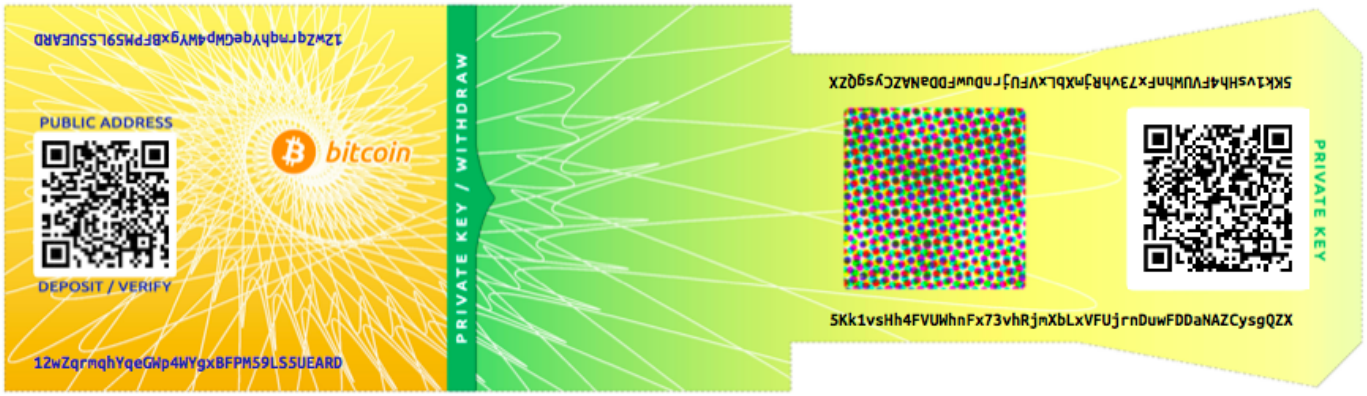


Figure 16. Un ejemplo de una cartera de papel de bitcoinpaperwallet.com con la clave privada en una solapa plegable.

Figure 17. La cartera de papel de bitcoinpaperwallet.com con la clave privada oculta.

Otros diseños cuentan con copias adicionales de la clave y de la dirección, en forma de fichas

separables similares a talones de boletos, lo que le permite almacenar múltiples copias para protegerse contra incendios, inundaciones u otros desastres naturales.

Figure 18. Un ejemplo de una cartera de papel con copias adicionales de las claves en un "talón" de respaldo.

Transacciones

Introducción

Las transacciones son la parte más importante del sistema bitcoin. Todo lo demás en bitcoin fue diseñado para asegurar que las transacciones puedan ser creadas, propagadas por la red, validadas y finalmente añadidas al libro contable global (la cadena de bloques). Las transacciones son estructuras de datos que codifican la transferencia de valor entre los participantes en el sistema bitcoin. Cada transacción es una entrada pública en la cadena de bloques de bitcoin, el libro contable global de contabilidad por partida doble.

En este capítulo examinaremos las varias formas de transacciones, qué contienen, cómo crearlas, cómo se verifican y cómo se vuelven parte del registro permanente de todas las transacciones.

Ciclo de Vida de una Transacción

El ciclo de vida de una transacción comienza con la creación de la transascción, también conocido como *generación*. La transacción es luego firmada con una o más firmas indicando la autorización a gastar los fondos referenciados por la transacción. La transacción es luego transmitida sobre la red bitcoin, donde cada nodo de la red (participante) valida y propaga la transacción hasta que alcanza a (casi) todos los nodos en la red. Finalmente la transacción es verificada por un nodo minero e incluida en un bloque de transacciones que es registrado en la cadena de bloques.

Una vez registrada en la cadena de bloques y confirmada por suficientes bloques subsecuentes (confirmaciones), la transacción pasa a ser una parte permanente del libro contable y es aceptada como válida por todos los participantes. Los fondos asignados a un nuevo dueño por la transacción pueden luego ser gastados en una nueva transacción, extendiendo la cadena de propiedad y comenzando el ciclo de vida de una transacción nuevamente.

Creando Transacciones

En algunas formas ayuda pensar en una transacción como si fuera un cheque de papel. Al igual que un cheque, una transacción es un instrumento que expresa la intención de transferir dinero y no es visible en el sistema financiero hasta que es enviado para ser liquidado. Tal como con un cheque, el originador de una transacción no necesita ser quien firme la transacción.

Las transacciones pueden ser creadas online u offline por cualquiera, incluso si la persona creando la transacción no es un firmante autorizado de la cuenta. Por ejemplo, un empleado a cargo de cuentas a pagar puede poseer cheques pagables para ser firmados por el presidente ejecutivo. De forma similar, un empleado a cargo de cuentas a pagar puede crear transacciones bitcoin y luego enviarlas al presidente ejecutivo para aplicar su firma digital y así hacerlas válidas. Así como un cheque referencia una cuenta específica como fuente de los fondos, una transacción bitcoin referencia una transacción previa específica como su fuente, en vez de una cuenta.

Una vez que una transacción ha sido creada, es firmada por el dueño (o dueños) de los fondos fuente. Si está propiamente formada y firmada, la transacción es ahora válida y contiene toda la información necesaria para ejecutar la transferencia de fondos. Finalmente, la transacción válida debe alcanzar la red bitcoin para que pueda ser propagada hasta alcanzar un minero para su inclusión en el libro contable público (la cadena de bloques).

Transmitiendo Transacciones a la Red Bitcoin

Primero, una transacción debe ser enviada a la red bitcoin para poder ser propagada e incluida en la cadena de bloques. En esencia, una transacción bitcoin contiene entre 300 y 400 bytes de datos y debe alcanzar alguno las decenas de miles de nodos bitcoin. Los remitentes no necesitan confiar en los nodos que utilizan para transmitir la transacción siempre y cuando utilicen más de uno para asegurar su propagación. Los nodos no necesitan confiar en el remitente para establecer la "identidad" del remitente. Como la transacción se encuentra firmada no contiene información confidencial, claves privadas o credenciales, puede ser transmitida públicamente usando cualquier red de transporte que resulte conveniente. A diferencia de transacciones de tarjetas de crédito, por ejemplo, las cuales contienen información sensible y solo pueden ser transmitidas sobre redes encriptadas, una transacción bitcoin puede ser enviada sobre cualquier red. Siempre y cuando la transacción pueda alcanzar un nodo de la red bitcoin que la propague, no importa cómo es transportada al primer nodo.

Las transacciones bitcoin pueden, por lo tanto, ser transmitidas a la red bitcoin a través de redes inseguras tales como WiFi, Bluetooth, NFC, Chirp, códigos de barras, o copiando y pegando de un formulario web. En casos extremos, una transacción bitcoin puede ser transmitida a través de paquetes vía radio, transmisión satelital, onda corta usando transmisión de ráfagas, espectro ensanchado o salto de frecuencia para evadir detección o interferencia. Una transacción bitcoin puede incluso ser codificada como smileys (emoticonos) y publicada en un foro público o enviada como un mensaje de texto de Skype. Bitcoin ha convertido al dinero en una estructura de datos, haciendo prácticamente imposible el evitar que cualquiera ejecute una transacción bitcoin.

Propagando Transacciones sobre la Red Bitcoin

Una vez que una transacción bitcoin es enviada a un nodo conectado a la red bitcoin, la transacción será validada por dicho nodo. Si es válida el nodo la propagará a otros nodos a los que se encuentra conectado, y un mensaje de éxito será devuelto sincrónicamente al originador. Si la transacción es inválida, el nodo la rechazará y devolverá un mensaje de rechazo sincrónicamente al originador.

La red bitcoin es una red entre pares (peer-to-peer), lo cual significa que cada nodo bitcoin se encuentra conectado a unos pocos otros nodos bitcoin que descubre durante su inicialización a través del protocolo entre pares. La totalidad de la red forma una malla parcialmente conectada sin una topología rígida ni estructura, haciendo de cada nodo un par equitativo. Los mensajes, incluyendo transacciones y bloques, son propagados de cada nodo a todos los pares a los que se encuentra conectado, un proceso conocido como "inundación" (flooding). Una nueva transacción validada inyectada en cualquier nodo de la red será enviada a todos sus nodos conectados a él (vecinos), cada uno de los cuales enviará la transacción a todos sus vecinos, y así sucesivamente. De esta forma, en apenas unos pocos segundos una transacción válida se propagará en una onda en expansión

exponencial a través de la red hasta que todos los nodos de la red la hayan recibido.

La red bitcoin fue diseñada para propagar transacciones y bloques a todos los nodos de manera eficiente y resistente a ataques. Para prevenir el spamming, ataques por denegación de servicio u otros ataques molestos al sistema bitcoin, cada nodo valida cada transacción independientemente antes de continuar con su propagación. Una transacción malformada no se propagará más allá de un nodo. Las reglas por las que las transacciones son validadas se encuentran explicadas en mayor detalle en [tx_verification].

Estructura de una Transacción

Una transacción es una *estructura de datos* que codifica una transferencia de valor de una fuente de fondos, llamada *entrada* (input), a un destinatario, llamado una *salida* (output). Las entradas y salidas de una transacción no se encuentran relacionadas a cuentas ni identidades. En cambio debes pensar en ellas como montos de bitcoin—trozos de bitcoin—asegurados con un secreto específico que solo su dueño, o persona que conoce el secreto, puede liberar. Una transacción contiene un número de campos, como se detalla en La estructura de una transacción.

Table 1. La estructura de una transacción

Tamaño	Campo	Descripción
4 bytes	Versión	Especifica qué reglas sigue esta transacción
1–9 bytes (VarInt)	Contador de Entradas	Cuántas entradas son incluidas
Variable	Entradas	Una o más entradas de la transacción
1–9 bytes (VarInt)	Contador de Salidas	Cuántas salidas son incluidas
Variable	Salidas	Una o más salidas de la transacción
4 bytes	Locktime	Un sello de tiempo (timestamp) Unix o número de bloque

Tiempo de Bloqueo de una Transacción

El tiempo de bloqueo (Locktime), también conocido como nLockTime por el nombre de variable utilizado para en el cliente de referencia, define el tiempo más cercano en que una transacción será válida y puede ser transmitida a la red e incluida en la cadena de bloques. En la mayoría de las transacciones su valor se establece en cero para indicar propagación y ejecución inmediatos. Si el tiempo de bloqueo no es cero y por debajo de 500 millones, se interpreta como una altura de bloque, lo cual significa que la transacción no es válida y no es transmitida ni incluida en la cadena de bloques antes de alcanzar la altura de bloque especificada. Si se encuentra por encima de los 500 millones es interpretada como un sello de tiempo Unix Epoch (segundos transcurridos desde el 1ro de enero de 1970) y la transacción no se considera válida antes del tiempo especificado. Las transacciones con tiempo de bloqueo referenciando un tiempo o bloque futuros deben ser conservadas por el sistema originario y transmitidas a la red bitcoin únicamente luego de volverse válidas. El uso del tiempo de bloqueo es equivalente a posfechar un cheque en papel.

Entradas y Salidas de una Transacción

La pieza fundamental de una transacción bitcoin es una *salida de transacción no gastada* (unspent transaction output), o UTXO. Las UTXO son trozos indivisibles de moneda bitcoin atados a un propietario específico, registrados en la cadena de bloques y reconocido como unidades de moneda por toda la red. La red bitcoin monitorea todas las UTXO, actualmente estimadas en millones. Cuando un usuario recibe bitcoins, ese monto es registrado en la cadena de bloques como una UTXO. Por lo tanto, los bitcoins de un usuario pueden estar dispersados como UTXOs entre cientos de transacciones y cientos de bloques. De hecho, no existe tal cosa como un saldo almacenado de una dirección bitcoin o una cuenta; tan solo hay UTXOs dispersados, asignados a propietarios específicos. El concepto de saldo de bitcoins de un usuario es una construcción creada por la aplicación de cartera. La cartera calcula el saldo del usuario escaneando la cadena de bloques y sumando todos los UTXOs pertenecientes a ese usuario.

TIP No existe cuentas o saldos en bitcoin; solo *salidas de transacciones sin gastar* (UTXO) dispersados en la cadena de bloques.

Una UTXO puede tener un valor arbitrario denominado como un múltiplo de satoshis. Tal como los dólares pueden ser divididos hasta dos cifras decimales en centavos, los bitcoins pueden ser divididos hasta ocho cifras decimales en satoshis. Aunque una UTXO puede ser de cualquier valor arbitrario, una vez creada es indivisible tal como una moneda que no puede ser partida a la mitad. Si una UTXO es mayor que el valor deseado de la transacción, aún debe ser consumida por completo y debe generarse cambio en la transacción. En otras palabras, si tienes una UTXO de 20 bitcoins y quieres pagar 1 bitcoin, tu transacción debe consumir la UTXO de 20 bitcoins entera y producir dos salidas: una pagando 1 bitcoin al destinatario deseado y otra pagando 19 bitcoins de cambio de regreso a tu cartera. Como resultado la mayoría de las transacciones bitcoins generarán cambio.

Imagina una consumidora comprando una bebida de $1,50, abriendo su cartera para buscar una

combinación de monedas y billetes que cubran el costo de $1,50. La consumidora elegirá el cambio exacto de estar disponible (un billete de un dólar y dos monedas de 25 centavos), o una combinación de denominaciones menores (seis monedas de 25 centavos), o, de ser necesario, una unidad mayor como un billete de cinco dólares. Si paga al vendedor con un valor mayor, digamos $5, ella esperaría recibir $3,50 de cambio, los cuales regresarán a su cartera y los tendrá disponibles para futuras transacciones.

De forma similar, una transacción bitcoin debe ser creada a partir de las UTXOs del usuario en cualquier combinación de denominaciones que el usuario tenga disponible. Los usuarios no pueden partir una UTXO a la mitad de la misma forma que un billete de un dólar no puede ser cortado a la mitad y aún ser usado como moneda. La aplicación de cartera del usuario usualmente seleccionará de entre las UTXOs del usuario varias unidades para componer un monto mayor o igual al de la transacción deseada.

Al igual que en la vida real, una aplicación bitcoin puede usar varias estrategias para satisfacer el monto de la compra: combinar varias unidades más pequeñas, encontrar el cambio exacto, o usar una única unidad mayor al valor de la transacción y generar cambio. Todo este complejo montaje de UTXOs es calculado por la cartera del usuario automáticamente y es invisible al usuario. Solo es relevante si estás construyendo transacciones en crudo a partir de UTXOs programáticamente.

Las UTXOs consumidas por una transacción se llaman entradas de transacción (inputs), y las UTXOs creadas por la transacción se llaman salidas de transacción (outputs). De esta forma, trozos de valor en bitcoin son movidos de un dueño al siguiente en una cadena de transacciones consumiendo y generando UTXOs. Las transacciones consumen UTXOs al liberarlas con la firma del propietario corriente y crean UTXOs al asignarlas a la dirección bitcoin del nuevo propietario.

La excepción en la cadena de salidas y entradas es un tipo especial de transacción llamada transacción *coinbase*, la cual es la primera transacción en cada bloque. Esta transacción es colocada allí por el minero "ganador" y crea nuevos bitcoins asignados a dicho minero como recompensa por el minado. Así es como la masa monetaria de bitcoin es creada durante el proceso de minado, como veremos en [ch8].

TIP ¿Qué estuvo primero? ¿Entradas o salidas, el huevo o la gallina? Hablando en sentido estricto, las salidas están primero porque las transacciones coinbase, las cuales generan nuevos bitcoins, no poseen entradas y generan salidas de la nada.

Salidas de Transacción

Toda transacción bitcoin crea salidas, las cuales son registradas en el libro de transacciones bitcoin. Casi todas estas salidas, con una excepción (ver Salida de Datos (OP_RETURN)) crean trozos de bitcoin gastables llamados *salidas de transacción sin gastar* (unspent transaction outputs) o UTXO, las cuales son reconocidas por toda la red y están disponibles para que el propietario las gaste en transacciones futuras. Enviar bitcoins a alguien significa crear una salida de transacción sin gastar (UTXO) registrada con su dirección y disponible para ser gastadas.

Las UTXOs son monitoreadas por todos los nodos completos de bitcoin como un set de datos conocido

como la *set UTXO* o *reserva UTXO*, mantenido en una base de datos. Las nuevas transacciones consumen (gastan) una o más de estas salidas del set UTXO.

Las salidas de una transacción consisten en dos partes:

- Una cantidad de bitcoins denominada en *satoshis*, la unidad más pequeña de bitcoin
- Un *script de bloqueo* (locking script), también conocido como una "obstrucción" que "bloquea" este monto especificando las condiciones que deben ser cumplidas para gastar esta salida

El lenguaje de scripting de transacciones, usado por el script de bloqueo mencionado anteriormente, es analizado en detalle en Scripts de Transacción y Lenguaje de Script. La estrucutra de una salida de transacción muestra la estructura de una salida de transacción.

Table 2. La estrucutra de una salida de transacción

Tamaño	Campo	Descripción
8 bytes	Cantidad	Valor de bitcoins en satoshis (10^{-8} bitcoins)
1-9 bytes (VarInt)	Tamaño del Script de Bloqueo	Longitud en bytes del Script de Bloqueo, a seguir
Variable	Script de Bloqueo	Un script definiendo las condiciones necesarias para gastar la salida

En Un script que llama a la API de blockchain.info para encontrar la UTXO relacionada a una dirección usamos la API de blockchain.info para encontrar las salidas sin gastar (UTXO) de una dirección específica.

Example 1. Un script que llama a la API de blockchain.info para encontrar la UTXO relacionada a una dirección

```
# get unspent outputs from blockchain API

import json
import requests

# example address
address = '1Dorian4RoXcnBv9hnQ4Y2C1an6NJ4UrjX'

# The API URL is https://blockchain.info/unspent?active=<address>
# It returns a JSON object with a list "unspent_outputs", containing UTXO, like this:
#{  "unspent_outputs":[
#   {
#     "tx_hash":"ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167",
#     "tx_index":51919767,
#     "tx_output_n": 1,
#     "script":"76a9148c7e252f8d64b0b6e313985915110fcfefcf4a2d88ac",
#     "value": 8000000,
#     "value_hex": "7a1200",
#     "confirmations":28691
#   },
# ...
#]}

resp = requests.get('https://blockchain.info/unspent?active=%s' % address)
utxo_set = json.loads(resp.text)["unspent_outputs"]

for utxo in utxo_set:
    print "%s:%d - %ld Satoshis" % (utxo['tx_hash'], utxo['tx_output_n'],
utxo['value'])
```

Al ejecutar el script vemos una lista de IDs de transacciones, un carácter de dos puntos, el número de índice de la salida de transacción sin gastar (UTXO) específica, y el valor de esa UTXO en satoshis. El script de bloqueo no es mostrado en la salida en Ejecutando el script get-utxo.py.

Example 2. Ejecutando el script get-utxo.py

```
$ python get-utxo.py
ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167:1 - 8000000 Satoshis
6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf:0 - 16050000
Satoshis
74d788804e2aae10891d72753d1520da1206e6f4f20481cc1555b7f2cb44aca0:0 - 5000000 Satoshis
b2affea89ff82557c60d635a2a3137b8f88f12ecec85082f7d0a1f82ee203ac4:0 - 10000000
Satoshis
...
```

Condiciones de gasto (obstrucciones)

Las salidas de transacción asocian un monto específico (en satoshis) con una *obstrucción* específica o script de bloqueo que define la condición que debe ser cumplida para gastar ese monto. En la mayoría de los casos el script de bloqueo asignará la salida a una dirección bitcoin específica, de esa forma transfiriendo la propiedad de ese monto a su nuevo dueño. Cuando Alice paga al Café de Bob por su taza de café, su transacción crea una salida de 0,015 bitcoins *obstruida* o asignada a la dirección bitcoin del café. Esa salida de 0,015 bitcoins fue registrada en la cadena de bloques y se convirtió en parte del grupo de Salidas de Transacción Sin Gastar (UTXO), lo que significa que se mostrará en la cartera de Bob como parte de su saldo disponible. Cuando Bob decide gastar ese monto, su transacción liberará la obstrucción, destrabando la salida al proveer un script de desbloqueo que contenga la firma proveniente de la clave privada de Bob.

Entradas de Transacción

En términos simples, las entradas de transacción son punteros a UTXOs. Apuntan a una UTXO específica referenciando el hash de transacción y número de secuencia donde la UTXO se encuentra registrada en la cadena de bloques. Para gastar una UTXO, una entrada de transacción también incluye scripts de desbloqueo que satisfacen las condiciones de gasto establecidas por la UTXO. El script de desbloqueo es generalmente una firma la cual prueba la pertenencia de la dirección bitcoin que se encuentra en el script de bloqueo.

Cuando los usuarios hacen un pago sus carteras construyen una transacción seleccionando de sus UTXOs disponibles. Por ejemplo, para realizar un pago de 0,015 bitcoins, la aplicación de cartera puede seleccionar una UTXO de 0,01 y otra de 0,005, usando ambas para sumar el monto de pago deseado.

En Un script para calcular cuántos bitcoins serán emitidos en total vemos el uso de un algoritmo "codicioso" para seleccionar de entre los UTXOs disponibles para llegar al monto de un pago específico. En este ejemplo las UTXOs disponibles son provistas como una cadena de constantes, pero en la realidad las UTXOs disponibles serían pedidas con una llamada a RPC a Bitcoin Core o una API de terceros como se muestra en Un script que llama a la API de blockchain.info para encontrar la UTXO relacionada a una dirección.

Example 3. Un script para calcular cuántos bitcoins serán emitidos en total

```
# Selects outputs from a UTXO list using a greedy algorithm.

from sys import argv

class OutputInfo:

    def __init__(self, tx_hash, tx_index, value):
        self.tx_hash = tx_hash
        self.tx_index = tx_index
        self.value = value

    def __repr__(self):
        return "<%s:%s with %s Satoshis>" % (self.tx_hash, self.tx_index,
                                             self.value)

# Select optimal outputs for a send from unspent outputs list.
# Returns output list and remaining change to be sent to
# a change address.
def select_outputs_greedy(unspent, min_value):
    # Fail if empty.
    if not unspent:
        return None
    # Partition into 2 lists.
    lessers = [utxo for utxo in unspent if utxo.value < min_value]
    greaters = [utxo for utxo in unspent if utxo.value >= min_value]
    key_func = lambda utxo: utxo.value
    if greaters:
        # Not-empty. Find the smallest greater.
        min_greater = min(greaters)
        change = min_greater.value - min_value
        return [min_greater], change
    # Not found in greaters. Try several lessers instead.
    # Rearrange them from biggest to smallest. We want to use the least
    # amount of inputs as possible.
    lessers.sort(key=key_func, reverse=True)
    result = []
    accum = 0
    for utxo in lessers:
        result.append(utxo)
        accum += utxo.value
        if accum >= min_value:
            change = accum - min_value
            return result, "Change: %d Satoshis" % change
    # No results found.
    return None, 0
```

```
def main():
    unspent = [

OutputInfo("ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167", 1,
8000000),

OutputInfo("6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf", 0,
16050000),

OutputInfo("b2affea89ff82557c60d635a2a3137b8f88f12ecec85082f7d0a1f82ee203ac4", 0,
10000000),

OutputInfo("7dbc497969c7475e45d952c4a872e213fb15d45e5cd3473c386a71a1b0c136a1", 0,
25000000),

OutputInfo("55ea01bd7e9afd3d3ab9790199e777d62a0709cf0725e80a7350fdb22d7b8ec6", 17,
5470541),

OutputInfo("12b6a7934c1df821945ee9ee3b3326d07ca7a65fd6416ea44ce8c3db0c078c64", 0,
10000000),

OutputInfo("7f42eda67921ee92eae5f79bd37c68c9cb859b899ce70dba68c48338857b7818", 0,
16100000),
    ]

    if len(argv) > 1:
        target = long(argv[1])
    else:
        target = 55000000

    print "For transaction amount %d Satoshis (%f bitcoin) use: " % (target,
target/10.0**8)
    print select_outputs_greedy(unspent, target)

if __name__ == "__main__":
    main()
```

Si ejecutamos el script *select-utxo.py* sin un parámetro intentará construir un juego de UTXOs (y cambio) para un pago de 55.000.000 satoshis (0,55 bitcoins). Si proveemos un pago objetivo como parámetro el script seleccionará UTXOs para alcanzar el monto de ese pago objetivo. En Ejecutando el script select-utxo.py ejecutamos el script intentando hacer un pago de 0,5 bitcoins o 50.000.000 satoshis.

Example 4. Ejecutando el script select-utxo.py

```
$ python select-utxo.py 50000000
Para un monto de transacción de 50000000 Satoshis (0,500000 bitcoins) usa:
([<7dbc497969c7475e45d952c4a872e213fb15d45e5cd3473c386a71a1b0c136a1:0 with 25000000
Satoshis>, <7f42eda67921ee92eae5f79bd37c68c9cb859b899ce70dba68c48338857b7818:0 with
16100000 Satoshis>,
<6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf:0 with 16050000
Satoshis>], 'Change: 7150000 Satoshis')
```

Una vez que las UTXOs son seleccionadas, la cartera produce scripts de desbloqueo conteniendo firmas para cada una de las UTXO, de esa forma haciéndolas gastables al satisfacer las condiciones del script de bloqueo. La cartera añade estas referencias a UTXOs y scripts de desbloqueo como entradas de la transacción. La estructura de la entrada de una transacción muestra la estructura de una entrada de transacción.

Table 3. La estructura de la entrada de una transacción

Tamaño	Campo	Descripción
32 bytes	Hash de Transacción	Puntero a la transacción que contiene la UTXO a ser gastada
4 bytes	Índice de Salida	El número de índice de la UTXO a ser gastada; comenzando por 0
1-9 bytes (VarInt)	Tamaño del Script de Desbloqueo	Longitud del Script de Desbloqueo en bytes, a seguir
Variable	Script de Desbloqueo	Un script que cumple con las condiciones del script de bloqueo de UTXOs
4 bytes	Número de Secuencia	Funcionalidad de reemplazo de transacción actualmente deshabilitada, establecer en 0xFFFFFFFF

El número de secuencia se usa para sobreescribir una transacción previamente a la expiración del tiempo de bloqueo de la transacción, lo cual es una funcionalidad actualmente deshabilitada en bitcoin. La mayoría de las transacciones establecen este valor al máximo valor entero (0xFFFFFFFF) y es ignorado por la red bitcoin. Si la transacción posee un tiempo de bloqueo distinto de cero al menos una de sus entradas debe tener un número de secuencia por debajo de 0xFFFFFFFF para habilitad el tiempo de bloqueo.

Comisiones de Transacción

La mayoría de las transacciones incluyen tarifas de transacción, las cuales compensan a los mineros bitcoin por asegurar la red. La minería y las tarifas y recompensas recolectadas por los mineros son analizadas en mayor detalle en [ch8]. Esta sección examina cómo las tarifas de transacción son incluidas en una transacción típica. La mayoría de las carteras calculan e incluyen tarifas de transacción automáticamente. Sin embargo, si estás construyendo transacciones programáticamente o usando una interfaz de línea de comando, debes tomar en cuenta e incluir estas tarifas manualmente.

Las tarifas de transacción sirven de incentivo para incluir (minar) una transacción en el siguiente bloque y también como desincentivo contra el "spam" de transacciones o cualquier tipo de abuso del sistema, al requerir un pequeño costo en cada transacción. Las tarifas de transacción son recolectadas por el minero que mina el bloque que registra la transacción en la cadena de bloques.

Las tarifas de transacción son calculadas basadas en el tamaño de la transacción en kilobytes, no el valor de la transacción en bitcoins. En general las tarifas de transacción son establecidas basadas en fuerzas del mercado en la red bitcoin. Los mineros priorizan transacciones basados en distintos criterios, incluyendo tarifas y pueden hasta procesar transacciones sin tarifa bajo ciertas circunstancias. Las tarifas de transacción afectan la prioridad de procesado, lo cual significa que una transacción con tarifa suficiente será muy probablemente incluida en el próximo bloque en ser minado, mientras que una transacción con una tarifa pequeña o sin tarifa puede ser demorada, procesada cuando sea posible luego de algunos bloques, o jamás procesada. Las tarifas de transacción no son obligatorias y las transacciones sin tarifa pueden resultar finalmente procesadas; sin embargo, incluir tarifas en transacciones incentiva al procesado prioritario.

Con el tiempo la forma en que las tarifas de transacción son calculadas y el efecto que tienen sobre la priorización ha ido evolucionando. Al principio las tarifas de transacción eran fijas y constante en toda la red. Gradualmente la estructura de tarifas ha sido relajada de forma que pueda ser influenciada por fuerzas del mercado, basadas en la capacidad de la red y el volumen de transacciones. La mínima tarifa de transacción actual está fijada en 0,0001 bitcoin, o una décima parte de un milibitcoin por kilobyte, recientemente reducida de un milibitcoin. La mayoría de las transacciones pesan menos de un kilobyte; sin embargo, aquellas con múltiples entradas o salidas pueden ser mayores. En próximas versiones del protocolo bitcoin se espera que las aplicaciones de cartera usen análisis estadístico para calcular la tarifa más adecuada para cada transacción basadas en promedios de tarifas de transacciones recientes.

El algoritmo actual utilizado por mineros para priorizar transacciones para inclusión en un bloque basados en sus tarifas es examinado en detalle en [ch8].

Añadiendo Comisiones a Transacciones

La estructura de datos de transacciones no posee un campo para tarifas. En cambio, las tarifas estan implícitas como la diferencia entre la suma de las entradas y la suma de las salidas. Cualquier monto que sobre luego de que las salidas han sido descontadas de todas las entradas será la tarifa recolectada por los mineros.

Las tarifas de transacción son implícitas como el excedente de entradas menos salidas:

```
Tarifas = Suma(Entradas) - Suma(Salidas)
```

Esto es un elemento un tanto confuso de las transacciones y un punto importante a entender, ya que si estás construyendo tus propias transacciones debes asegurarte de no incluir una tarifa muy grande por descuido al gastar las entradas de menos. Esto significa que debes tener en cuenta todas las entradas, de ser necesario creando cambio, ¡o terminarás dándole a los mineros una propina muy grande!

Por ejemplo, si consumes una UTXO de 20 bitcoins para hacer un pago de 1 bitcoin, debes incluir una salida de cambio de 19 bitcoins de regreso a tu cartera. De lo contrario, los 19 bitcoins "sobrantes" serán contados con la tarifa de transacción y serán recolectados por el minero que mine tu transacción en un bloque. Aunque recibirás procesado prioritario y harás muy feliz a un minero, esto probablemente no sea lo que planeabas hacer.

WARNING	Si te olvidas de añadir una salida de cambio en una transacción construida manualmente terminarás pagando el cambio como tarifa de transacción. "¡Quédate el cambio!" puede no haber sido tu intención.

Veamos cómo funciona esto en la práctica usando de ejemplo la compra de café de Alice nuevamente. Alice quiere gastar 0,015 bitcoins para pagar por un café. Para asegurar que esta transacción sea procesada rápidamente ella querrá incluir una tarifa de transacción, digamos que 0,001. Esto significará que el costo total de la transacción será de 0,016 bitcoins o más y, de ser necesario, creará cambio. Digamos que su cartera tiene una UTXO de 0,2 bitcoins disponible. Por lo tanto necesitará consumir esta UTXO, crear una salida para el Café de Bob por 0,015, y una segunda salida con 0,184 bitcoins de cambio de regreso a su propia cartera, dejando 0,001 bitcoins sin distribuir, lo cual será la tarifa implícita para la transacción.

Ahora veamos un caso diferente. Eugenia, nuestra directora de la beneficencia para niños en las Filipinas ha completado una recaudación de fondos para adquirir libros para los niños. Ella ha recibido varios miles de pequeñas donaciones de personas alrededor del mundo, las cuales suman 50 bitcoins, por lo que su cartera está llena de pagos muy pequeños (UTXOs). Ahora ella quiere comprar cientos de libros escolares a una editorial local, pagando en bitcoins.

Ya que la aplicación de cartera de Eugenia intenta construir una única transacción de pago mayor, debe sacar de la reserva de UTXOs disponibles, la cual está compuesta de múltiples montos más pequeños. Esto significa que la transacción resultante usará de fuente más de cien UTXOs de pequeño valor como entradas y solo una salida, pagando a la editorial de libros. Una transacción con tantas entradas será más grande que un kilobyte, quizá resulte de 2 o 3 kilobytes en tamaño. Por lo tanto requerirá una tarifa de transacción mayor a la mínima tarifa de la red de 0,0001 bitcoins.

La aplicación de cartera de Eugenia calculará la tarifa adecuada midiendo el tamaño de la transacción y multiplicándolo por la tarifa por kilobyte. Muchas carteras pagan tarifas más altas de lo necesario para transacciones muy grandes para asegurarse de que la transacción sea procesada rápidamente. La

tarifa elevada no es porque Eugenia esté gastando más dinero, sino porque su transacción es más compleja y grande en tamaño—la tarifa es independiente del valor en bitcoins de la transacción.

Encadenamiento de Transacciones y Transacciones Huérfanas

Como hemos visto, las transacciones forman una cadena en la cual una transacción gasta las salidas de la transacción previa (conocida como madre) y crea salidas para una transacción subsecuente (conocida como hija). A veces una cadena entera de transacciones dependientes unas de otras—digamos una transacción madre, hija y nieta—son creadas al mismo tiempo para cumplir con un flujo de trabajo transaccional complejo que requiere que transacciones hijas válidas sean firmadas antes de que la transacción madre sea firmada. Por ejemplo, esta es una técnica usada en transacciones CoinJoin donde varios participantes unen transacciones para proteger su privacidad.

Cuando una cadena de transacciones es transmitida a través de la red, no siempre llegan en el mismo orden. A veces la transacción hija puede llegar antes que la madre. En ese caso, los nodos que ven la transacción hija primero pueden ver que se refiere a una transacción madre aun desconocida. En vez de rechazar a la hija, la colocan en una reserva temporaria para esperar el arribo de su transacción madre y propagarla a todos los demás nodos. La reserva de transacciones sin madres es conocida como la *reserva de transacciones huérfanas* (orphan transaction pool). Una vez que la transacción madre arriba, cualquier huérfana que referencie la UTXO creada por la madre será liberada de la reserva, revalidada recursivamente, y luego la cadena de transacciones entera puede ser incluida en la reserva de transacciones, lista para ser minada en un bloque. Las cadenas de transacciones pueden ser arbitrariamente largas, con cualquier número de generaciones transmitidas simultáneamente. El mecanismo de conservar huérfanas en la reserva de huérfanas asegura que transacciones que serían válidas de otra forma no sean rechazadas simplemente porque su madre ha sido demorada y que finalmente la cadena a la que pertenecen sea reconstruida en el orden correcto, independientemente del orden de llegada.

Existe un límite al número de transacciones huérfanas almacenadas en memoria para prevenir ataques de denegación de servicio (denial of service) contra los nodos bitcoin. El límite está definido como MAX_ORPHAN_TRANSACTIONS en el código fuente del cliente de referencia bitcoin. Si el número de transacciones huérfanas en la reserva excede MAX_ORPHAN_TRANSACTIONS, uno o más transacciones huérfanas seleccionadas aleatoriamente serán removidas de la reserva hasta que el tamaño de la reserva regrese a los límites permitidos.

Scripts de Transacción y Lenguaje de Script

Los clientes bitcoin validan transacciones ejecutando un script escrito en un lenguaje de scripting similar a Forth. Tanto el script de bloqueo (obstrucción) colocado sobre una UTXO como el script de desbloqueo que generalmente contiene una firma son escritos en este lenguaje de scripting. Cuando una transacción es validada, el script de desbloqueo en cada entrada es ejecutado junto con su correspondiente script de bloqueo para verificar que satisfaga la condición de gasto.

Hoy en día la mayoría de las transacciones procesadas a través de la red bitcoin tienen la forma de "Alice paga a Bob" y se basan en el mismo script llamado script de Pago-a-Hash-de-Clave-Pública (Pay-to-Public-Key-Hash script). Sin embargo, el uso de scripts para bloquear outputs y desbloquear inputs significa que mediante el uso del lenguaje de programación las transacciones pueden contener un número infinito de condiciones. Las transacciones bitcoin no se limitan a la forma y patrón de "Alica paga a Bob.

Esto es tan solo la punta del iceberg de posibilidades que pueden ser expresadas con este lenguaje de scripting. En esta sección haremos una demostración de los componentes del lenguaje de scripting de transacciones bitcoin y mostraremos cómo puede ser utilizado para expresar condiciones complejas para gastar y cómo esas condiciones pueden ser satisfechas por scripts de desbloqueo (unlocking scripts).

TIP	La validación de transacciones bitcoin no se basa en un patron estático, sino que es alcanzada a través de la ejecución de un lenguaje de scripting. Este lenguaje permite una variedad casi infinita de condiciones a ser expresadas. Así es cómo bitcoin adquiere el poder de "dinero programable".

Construcción de Scripts (Bloqueo + Desbloqueo)

El motor de validación de transacciones de bitcoin depende de dos tipos de scripts para validar transacciones: un script de bloqueo (locking script) y un script de desbloqueo (unlocking script).

Un script de bloqueo (locking script) es una obstrucción colocada sobre una salida, el cual especifica las condiciones que deben cumplirse para gastar dicha salida en el futuro. Históricamente a los scripts de bloqueo se los llamaba un *scriptPubKey*, ya que usualmente contenían una clave pública o dirección bitcoin. En este libro nos referiremos a ellos como "script de bloqueo" para reconocer el mucho mayor espectro de posibilidades de esta teconología de scripting. En la mayoría de las aplicaciones bitcoin a lo que nos referimos como script de bloqueo aparecerá en el código fuente como scriptPubKey.

Un script de desbloqueo (unlocking script) es un script que "resuelve," o satisface, las condiciones establecidas por una salida y un script de bloqueo y permite que la salida sea gastada. Los scripts de desbloqueo son parte de cada entrada de transacción, y la mayoría de las veces contienen una firma digital producida por la cartera del usuario a partir de su clave privada. Históricamente el script de desbloqueo era llamado *scriptSig*, ya que usualmente contenía uan firma digital. En la mayoría de las aplicaciones bitcoin el código fuente se refiere al script de desbloqueo como scriptsig. En este libro nos referiremos a ellos como "script de desbloqueo" para reconocer el espectro mucho más amplio de requerimientos de scripts de bloqueo, ya que no todos los scripts de desbloqueo requieren firmas.

Todo cliente bitcoin debe validar transacciones ejecutando los scripts de bloqueo y desbloqueo en simultáneo. Para cada entrada de la transacción el software traerá primero la UTXO referenciada por la entrada. Esa UTXO contiene un script de bloqueo definiendo las condiciones requeridas para enviarla. El software de validación luego tomará el script de desbloqueo contenido en la entrada que está intentando gastar esta UTXO y ejecutará ambos scripts.

En el cliente bitcoin original, los scripts de bloqueo y desbloqueo eran concatenados y ejecutados en

secuencia. Por razones de seguridad esto fue cambiado en 2010, debido a una vulnerabilidad que permitía que un script de desbloqueo malformado enviara datos a la pila y corrompiera el script de bloqueo. En la implementación actual los scripts son ejecutados en forma separada y la pila es transferida entre ejecuciones, como se describe a continuación.

Primero, el script de desbloqueo es ejecutado utilizando el motor de ejecución de pila. Si el script de desbloqueo es ejecutado sin errores (por ejemplo, no posee operadores sobrantes "colgando"), la pila principal (no la pila alternativa) es copiada y el script de bloqueo es ejecutado. Si el resultado de ejecutar el script de bloqueo con los datos de la pila copiados del script de desbloqueo es "VERDADERO", el script de desbloqueo ha sido exitoso en resolver las condiciones impuestas por el script de bloqueo y, por tanto, la entrada es una autorización válida para gastar la UTXO. Si cualquier resultado que no sea "VERDADERO" permanece luego de la ejecución del script combinado, la entrada es inválida porque ha fallado en satisfacer las condiciones de gastado colocadas sobre la UTXO. Nótese que la UTXO es registrada permanentemente en la cadena de bloques, y por ello es invariable y no se ve afectada por intentos fallidos de gastarla por referencia en una nueva transacción. Únicamente una nueva transacción que satisface las condiciones de la UTXO correctamente resulta en la UTXO siendo marcada como "gastada" y removida de la reserva de UTXOs disponibles (sin gastar).

Combinando scriptSig y scriptPubKey para evaluar un script de transacción es un ejemplo de los scripts de desbloqueo y bloqueo para el tipo más común de transacción bitcoin (un pago a un hash de clave pública), mostrando el script combinado que resulta de la concatenación de los scripts de desbloqueo y bloqueo previo a la validación por script.

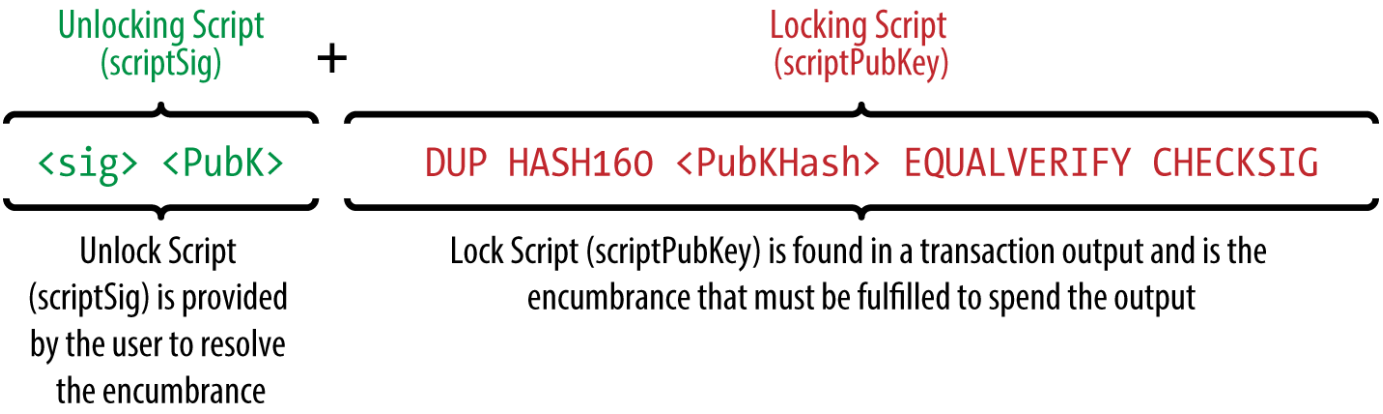


Figure 1. Combinando scriptSig y scriptPubKey para evaluar un script de transacción

Lenguaje de Scripting

El lenguaje de scripts de transacción bitcoin, llamado *Script*, es un lenguaje de ejecución basada en pila con notación polaca inversa similar a Forth. Si eso no tiene sentido para ti, probablemente sea que no has estudiado lenguajes de programación de la década de 1960. Script es un lenguaje muy simple diseñado para ser limitado en alcance y ejecutable en un rango amplio de hardware, quizá hasta tan simple como un dispositivo embebido, tal como una calculadora de mano. Requiere procesamiento mínimo y no puede hacer muchas de las cosas sofisticadas que los lenguajes modernos sí pueden. En el caso del dinero programable, esto es una medida intencional de seguridad.

El lenguaje de scripting de bitcoin es llamado un lenguaje de ejecución basada en pila porque utiliza

una estructura de datos llamada una *pila* (stack). Una pila es una estructura de datos muy simple, la cual puede ser visualizada como una pila de cartas. Una pila permite realizar dos operaciones: empujar y sacar. Empujar añade un elemento al tope de la pila. Sacar remueve el elemento en el tope de la pila.

El lenguaje de scripting ejecuta el script procesando cada ítem de izquierda a derecha. Los números (constantes de datos) son empujados a la pila. Los operadores empujan o sacan uno o más parámetros de la pila, actúan sobre ellos, y pueden empujar un resultado a la pila. Por ejemplo, OP_ADD sacará dos elementos de la pila, los sumará, y luego empujará la suma resultante a la pila.

Los operadores condicionales evalúan una condición, produciendo un resultado booleano de VERDADERO o FALSO. Por ejemplo, OP_EQUAL saca dos elementos de la pila y empuja VERDADERO (VERDADERO es representado por el número 1) si son iguales y FALSO (representado por cero) si no son iguales. Los scripts de transacción bitcoin usualmente contienen un operador condicional, de forma que puedan producir el valor VERDADERO que significa que la transacción es válida.

En El script de validación de Bitcoin haciendo matemática simple, el script 2 3 OP_ADD 5 OP_EQUAL muestra el operador de adición aritmética OP_ADD, el cual suma dos números y coloca el resultado en la pila, seguido por el operador condicional OP_EQUAL, el cual verifica que el resultado de la suma sea igual a 5. Para ser concisos, el prefijo OP_ es omitido en el ejemplo paso-a-paso.

Lo que sigue es un script levemente más complejo, el cual calcula 2 + 7 - 3 + 1. Nótese que cuando el script contiene varios operadores en hilera, la pila permite que los resultados de un operador sean utilizados por el siguiente operador:

```
2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL
```

Intenta validar el script previo tú mismo usando papel y lápiz. Cuando la ejecución del script acaba, deberías terminar con el valor VERDADERO en la pila.

Aunque la mayoría de los scripts de bloqueo se refieren a una dirección bitcoin o clave pública, y por lo tanto requiriendo prueba de pertenencia para gastar los fondos, el script no necesita ser tan complicado. Una combinación de scripts de bloqueo y desbloqueo que resulta en VERDADERO es válido. La aritmética simple que usamos como ejemplo del lenguaje de scripting es también un script de bloqueo válido que puede ser usado para bloquear una salida de transacción.

Usar parte del script de ejemplo aritmético como el script de bloqueo:

```
3 OP_ADD 5 OP_EQUAL
```

lo cual puede ser satisfecho por una transacción que contenga una entrada con el script de desbloqueo:

```
2
```

El software de validación combina los scripts de bloqueo y desbloqueo y el script resultante es:

```
2 3 OP_ADD 5 OP_EQUAL
```

Como vimos en el ejemplo paso-a-paso en El script de validación de Bitcoin haciendo matemática simple, cuando el script es ejecutado, el resultado es OP_TRUE, haciendo a la transacción válida. No solo es esto un script de bloqueo de salida de transacción válido, sino que el UTXO resultante puede ser gastado por cualquiera con la habilidad aritmética para saber que el número 2 satisface el script.

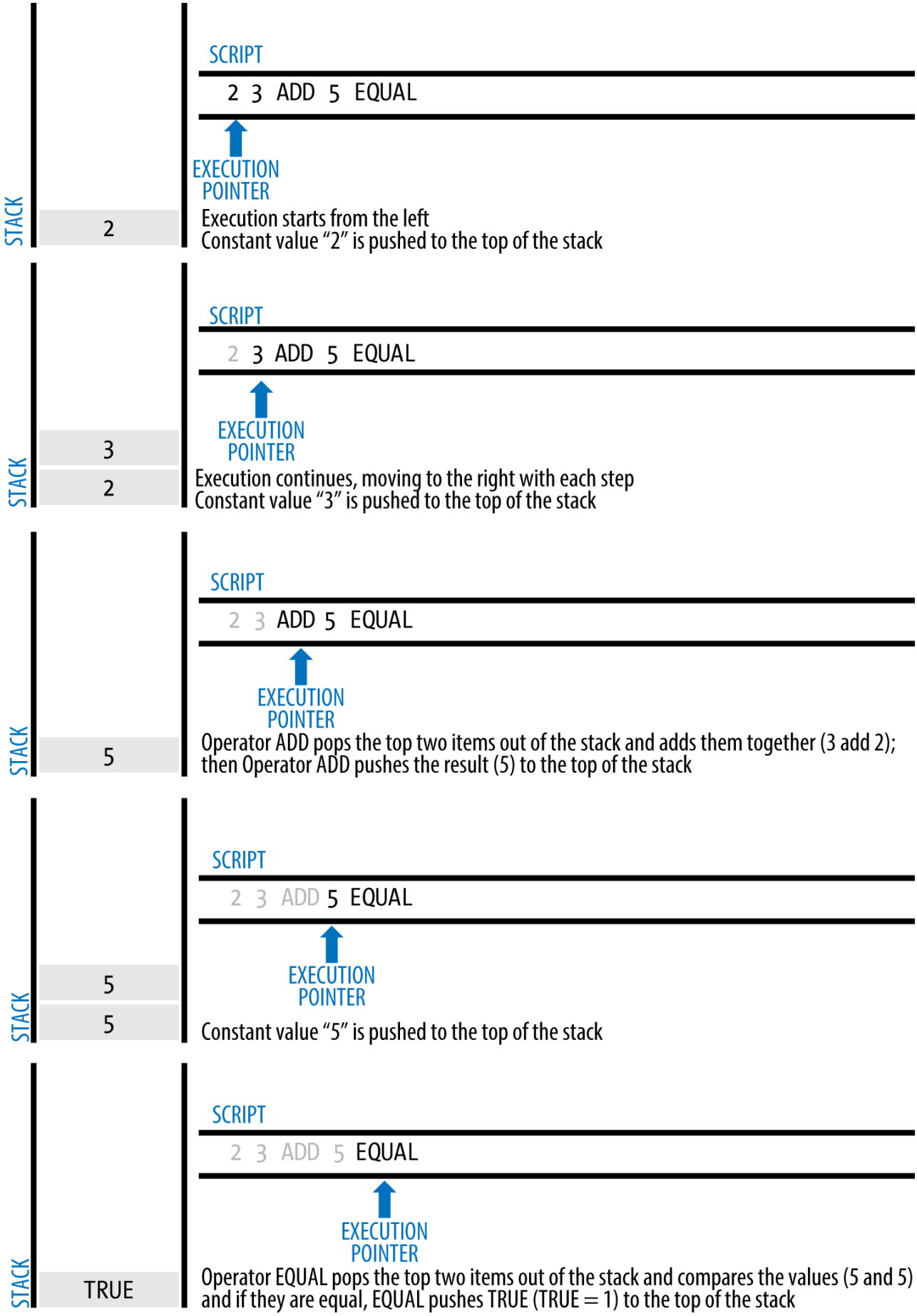

SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
2
Execution starts from the left
Constant value "2" is pushed to the top of the stack
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
3
2
Execution continues, moving to the right with each step
Constant value "3" is pushed to the top of the stack
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
5
Operator ADD pops the top two items out of the stack and adds them together (3 add 2);
then Operator ADD pushes the result (5) to the top of the stack
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
5
5
Constant value "5" is pushed to the top of the stack
SCRIPT
2 3 ADD 5 EQUAL
EXECUTION POINTER
STACK
TRUE
Operator EQUAL pops the top two items out of the stack and compares the values (5 and 5)
and if they are equal, EQUAL pushes TRUE (TRUE = 1) to the top of the stack

Figure 2. El script de validación de Bitcoin haciendo matemática simple

TIP

Las transacciones son válidas si el resultado en el tope de la pila es VERDADERO (notado como {0x01}), cualquier valor distinto de cero o si la pila se encuentra vacía luego de la ejecución del script. Las transacciones son inválidas si el valor en el tope de la pila es FALSO (un valor vacío de longitud cero, notado como {}), o si la ejecución del script es detenida explícitamente por un operador, tal como OP_VERIFY, OP_RETURN, o un condicional terminante como OP_ENDIF. Ver [tx_script_opts] para más detalles.

Incompletitud Turing

El lenguaje de script de transacciones bitcoin contiene muchos operadores, pero se encuentra deliberadamente limitado en una forma importante—no tiene la capacidad de realizar bucles ni controles de flujo complejos más allá de los controles de flujo condicionales. Esto asegura que el lenguaje no es *Turing Completo*, lo cual significa que los scripts tienen complejidad limitada y tiempos de ejecución predecibles. Script no es un lenguaje de propósito general. Estas limitaciones aseguran que el lenguaje no pueda ser usado para crear un bucle infinito u otras formas de "bombas lógicas" que pudieran ser embebidas en una transacción de forma que causara un ataque de denegación de servicio contra la red bitcoin. Recuerda, cada transacción es validada por cada nodo completo en la red bitcoin. Un lenguaje limitado previene que el mecanismo de validación de transacciones sea usado como una vulnerabilidad.

Verificación Sin Estado

El lenguaje de script de transacciones bitcoin es carente de estado en el sentido en que no existe un estado previo a la ejecución del script, o un estado guardado luego de la ejecución del script. Por lo tanto, toda la información necesaria para ejecutar el script se encuentra contenida en el mismo script. Un script se ejecutará predeciblemente de la misma forma en cualquier sistema. Si tu sistema verifica un script, puedes estar seguro que cualquier otro sistema en la red bitcoin también verificará el script, lo cual significa que una transacción es válida para todos y todos saben esto. Esta predictibilidad de resultados es un beneficio esencial del sistema bitcoin.

Transacciones Estándar

En los años iniciales del desarrollo de bitcoin, los desarrolladores introdujeron algunas limitaciones en los tipos de scripts que podían ser procesados por el cliente de referencia. Estas limitaciones se encuentran codificadas en una función llamada isStandard() (es estándar), la cual define cinco tipos de transacciones "estándar". Estas limitaciones son temporales y pueden encontrarse removidas para cuando leas esto. Hasta entonces, los cinco tipos de scripts de transacciones son los únicos aceptados por el cliente de referencia y la mayoría de los mineros que ejecutan el cliente de referencia. Aunque es posible crear transacciones no estándar que contengan un script que no es uno de los tipos estándar, debes encontrar un minero que no aplique estas limitaciones para minar esa transacción en un bloque.

Compruebe el código fuente del cliente Bitcoin Core (la implementación de referencia) para ver qué está permitido actualmente como script de transacción válido.

Los cinco tipos estándar de scripts de transacción son pago-a-hash-de-clave-pública (pay-to-public-key-hash, o P2PKH), clave-pública (public-key), multi-firma (multi-signature, limitado a 15 claves), pago-a-hash-de-script (pay-to-script-hash, o P2SH), y salida de datos (OP_RETURN), los cuales se describen en más detalle en las secciones siguientes.

Pago-a-Hash-de-Clave-Pública (P2PKH)

La vasta mayoría de las transacciones procesadas en la red bitcoin son transascciones P2PKH. Estas contienen un script de bloqueo que solicita a la salida un hash de clave pública, más comúnmente conocido como una dirección bitcoin. Las transacciones que pagan a una dirección bitcoin contienen scripts P2PKH. Una salida bloqueada por un script P2PKH puede ser desbloqueada (gastada) presentando una clave pública y una firma digital creada por la clave privada correspondiente.

Por ejemplo, veamos el pago de Alice al Café de Bob nuevamente. Alice hizo un pago de 0,015 bitcoins a la dirección bitcoin del café. Esa salida de transacción tendría un script de bloqueo del tipo:

```
OP_DUP OP_HASH160 <Hash de Clave Pública del Café> OP_EQUAL OP_CHECKSIG
```

El Hash de Clave Pública del Café es equivalente a la dirección bitcoin del café, sin la codificación Base58Check. La mayoría de las aplicaciones mostrarían el *hash de clave pública* en codificación hexadecimal y no el familiar formato Base58Check de la dirección bitcoin comenzado en "1".

El script de bloqueo anterior puede ser satisfecho con un script de desbloqueo de la forma:

```
<Firma del Café> <Clave Pública del Café>
```

Los dos scripts juntos formarían el siguiente script de validación combinado:

```
<Firma del Café> <Clave Pública del Café> OP_DUP OP_HASH160
<Hash de Clave Pública del Café> OP_EQUAL OP_CHECKSIG
```

Cuando es ejecutado, este script combinado será evaluado a VERDADERO si, y solo si, el script de desbloqueo cumple las condiciones establecidas por el script de bloqueo. En otras palabras, el resultado será VERDADERO si el script de desbloqueo contiene una firma válida proveniente de la clave privada del café que corresponde al hash de clave pública establecido como obstrucción.

Las figuras <xref linkend="P2PubKHash1" xrefstyle="select: labelnumber"/> y <xref linkend="P2PubKHash2" xrefstyle="select: labelnumber"/> muestran (en dos partes) una ejecución paso a paso del script combinado, el cual demostrará que es una transacción válida.

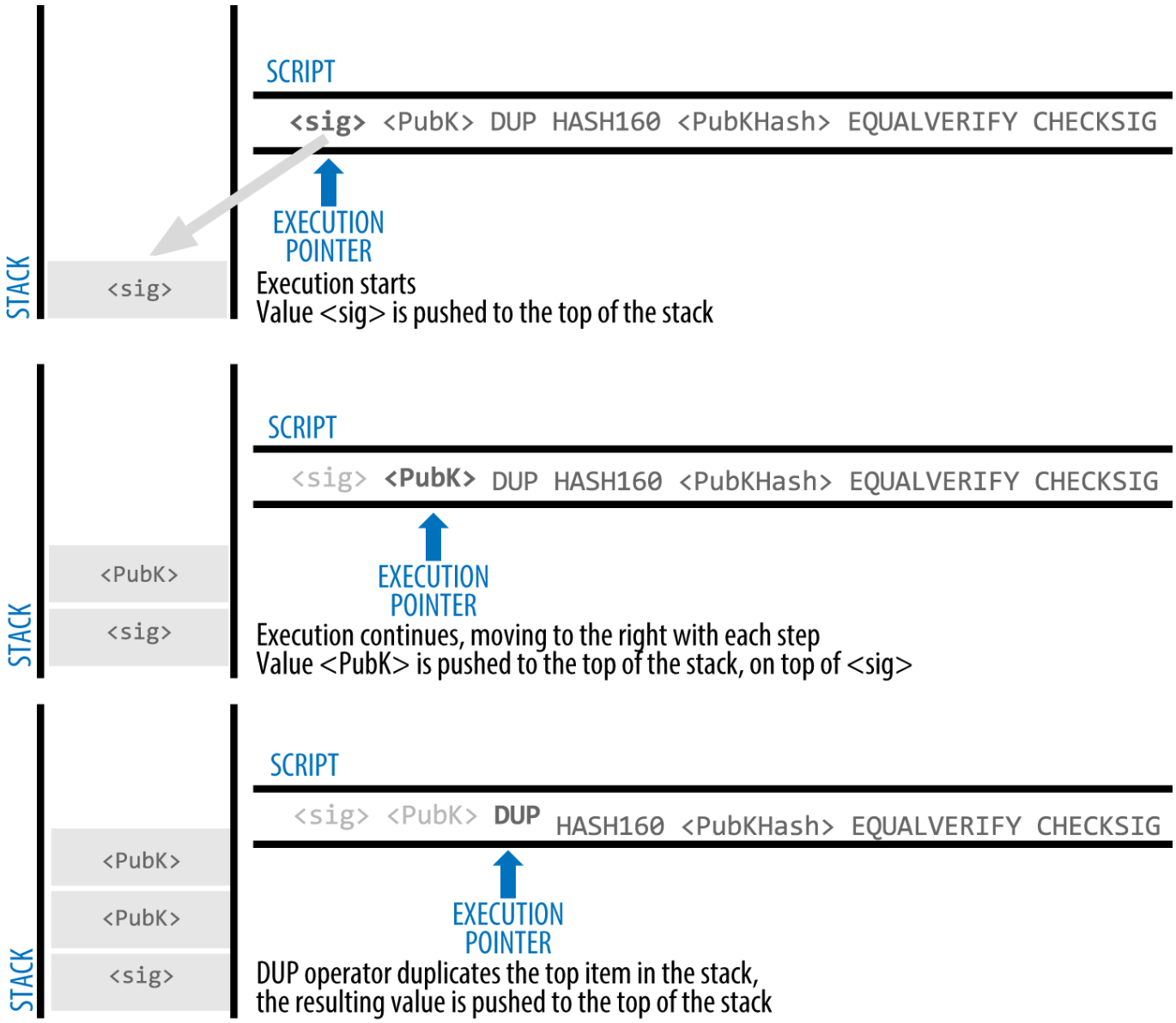


Figure 3. Evaluando un script para una transacción P2PKH (Parte 1 de 2)

Pago-a-Clave-Pública

Pago-a-clave-pública (pay-to-public-key) es una forma más simple de un pago bitcoin que pago-a-hash-de-clave-pública. Con esta forma de script la clave pública misma es almacenada en el script de bloqueo, en vez de un hash de clave pública como se vio en P2PKH anteriormente, lo cual resulta mucho más corto. El pago-a-hash-de-clave-pública fue inventado por Satoshi para hacer las direcciones de bitcoin más cortas, para facilidad de uso. Hoy en día el pago-a-clave-pública se ve principalmente en transacciones coinbase, generadas por software de minería más antiguo que no ha sido actualizado para utilizar P2PKH.

Un script de bloqueo de pago-a-clave-pública se ve así:

```
<Clave Pública A> OP_CHECKSIG
```

El script de desbloqueo correspondiente que debe presentarse para desbloquear este tipo de salida es una simple firma, como esta:

```
<Firma de Clave Pública A>
```

El script combinado, el cual es validado por el software de validación de transacción, es:

```
<Firma de Clave Pública A> <Clave Pública A> OP_CHECKSIG
```

El script es una simple invocación del operador CHECKSIG, el cual valida que la firma pertenezca a la clave correcta y devuelve VERDADERO en la pila.

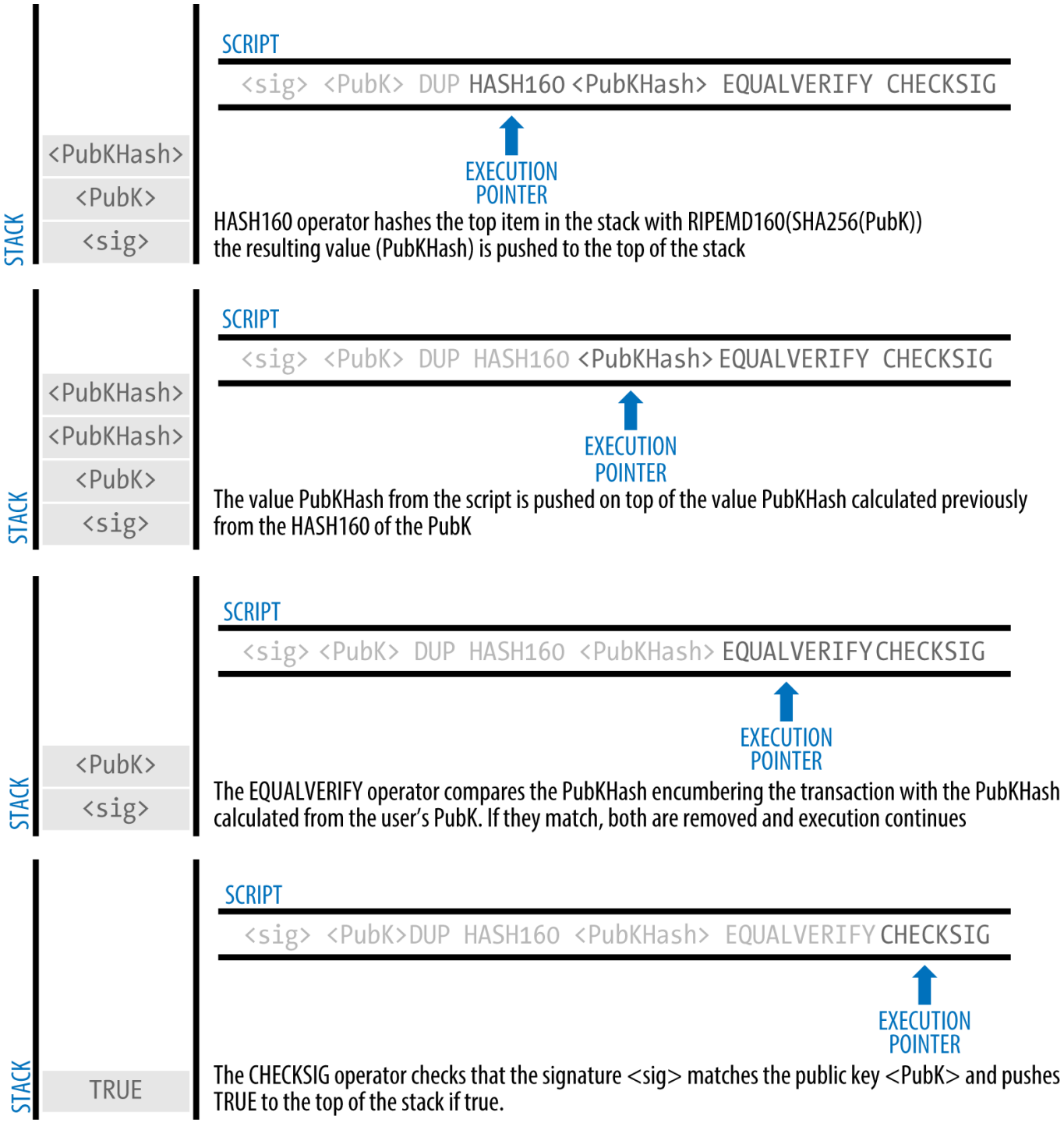


Figure 4. Evaluando un script para una transacción P2PKH (Parte 2 de 2)

Multi-Firma

Los scripts multi-firma establecen una condición donde N claves públicas son registradas en el script y al menos M de esas deben proveer firmas para liberar la obstrucción. Esto se conoce también como un esquema M-de-N, donde N es el número total de claves y M es el umbral de firmas requeridas para la validación. Por ejemplo, una multi-firma 2-de-3 es una donde tres claves públicas son listadas como potenciales firmantes y al menos dos de ellas deben ser usadas para crear firmas para una transacción válida para gastar los fondos. En este momento los scripts multi-firma se encuentran limitados a como

máximo a 15 claves públicas listadas, lo cual significa que es posible hacer cualquier combinación multi-firma desde 1-de-1 hasta 15-de-15. La limitación de 15 claves listadas puede haber cambiado a fecha en que este libro sea publicado, así que examine la función isStandard() para ver qué es aceptado actualmente por la red.

La forma general de un script de bloqueo estableciendo una condición multi-firma M-de-N es:

```
M <Clave Pública 1> <Clave Pública 2> ... <Clave Pública N> N OP_CHECKMULTISIG
```

donde N es el número total de claves públicas listadas y M es el umbral de firmas requeridas para gastar la salida.

Un script de bloqueo estableciendo una condición multifirma 2-de-3 se ve así:

```
2 <Clave Pública A> <Clave Pública B> <Clave Pública C> 3 OP_CHECKMULTISIG
```

El script de bloqueo anterior puede ser satisfecho con un script de desbloqueo conteniendo pares de firmas y claves públicas:

```
OP_0 <Firma B> <Firma C>
```

o cualquier combinación de dos firmas a partir de las claves privadas correspondientes a las tres claves públicas listadas.

> El prefijo OP_0 es requerido por un error en la implementación original de CHECKMULTISIG por el cual se saca un item de más de la pila. Es ignorado por CHECKMULTISIG y es sencillamente un marcador de posición.

Los dos scripts juntos formarían el script de validación combinado:

```
OP_0 <Firma B> <Firma C> 2 <Clave Pública A> <Clave Pública B> <Clave Pública C> 3
OP_CHECKMULTISIG
```

Cuando es ejecutado, el script combinado evaluará a VERDADERO si, y solo si, el script de desbloqueo cumple las condiciones establecidas por el script de bloqueo. En este caso, la condición es si el script de desbloqueo contiene una firma válida proveniente de las dos claves privadas que corresponden a dos de las tres claves públicas establecidas como obstrucciones.

Salida de Datos (OP_RETURN)

El libro contable bitcoin distribuido y con sellado de tiempo, la cadena de bloques, tiene muchos

potenciales usos más allá de pagos. Varios desarrolladores han intentado usar el lenguaje de scripting de transacciones para tomar ventaja de la seguridad y resistencia del sistema para aplicaciones como servicios de notaría digital, contratos inteligentes y certificado de acciones. Los primeros intentos de usar el lenguaje de script de bitcoin para estos propósitos involucraron crear salidas de transacciones que registraran datos en la cadena de bloques; por ejemplo, para registrar una huella digital de un archivo de forma que cualquiera pudiera establecer prueba de existencia de ese archivo en una fecha específica por referencia a dicha transacción.

El uso de la cadena de bloques de bitcoin para almacenar información sin relación a pagos bitcoin es un tema controvertido. Muchos desarrolladores lo consideran un abuso y prefieren desalentarlo. Otros lo ven como una demostración de las poderosas posibilidades de la tecnología de la cadena de bloques y prefieren alentar su experimentación. Quienes objetan a la inclusión de datos no relacionados a pagos argumentan que causa "hinchazón de la cadena de bloques", colocando una carga sobre quienes corren nodos bitcoin completos al tener que almacenar datos que la cadena de bloques no fue pensada para albergar. Adicionalmente, esas transacciones crean UTXOs que no pueden ser gastados, utilizando la dirección bitcoin de destino como un campo libre de 20 bytes. Ya que la dirección es utilizada para datos no corresponde a una clave privada y la UTXO resultante no puede ser gastada *jamás*; es un falso pago. Estas transacciones que no pueden ser gastadas no pueden ser removidas de la colección de UTXOs y provocan que el tamaño de la base de datos de UTXOs crezca o "se hinche" para siempre.

En la versión 0.9 del cliente Bitcoin Core se alcanzó un mutuo acuerdo con la introducción del operador OP_RETURN. OP_RETURN permite a los desarrolladores añadir 80 bytes de datos no relacionados a pagos a la salida de una transacción. Sin embargo, a diferencia del uso de UTXOs falsas, el operador OP_RETURN crea una salida *demostrablemente ingastable* explícitamente, la cual no necesita ser almacenada en la colección de UTXOs. Salidas del tipo OP_RETURN son registradas en la cadena de bloques, por lo que consumen espacio en disco y contribuyen al incremento de tamaño de la cadena de bloques, pero no son almacenadas en la colección de UTXOs y por ende no hinchan la reserva de memoria de UTXOs ni incomodan a los nodos completos con el costo de memoria RAM adicional.

Los scripts de OP_RETURN se ven así:

```
OP_RETURN <datos>
```

La porción de datos se limita a 80 bytes y frecuentemente representa un hash, tal como la salida del algoritmo SHA256 (32 bytes). Muchas aplicaciones colocan un prefijo en frente de los datos para ayudar a identificar la aplicación. Por ejemplo, el servicio de autorización bajo notario digital Proof of Existence usa el prefijo de 8 bytes "DOCPROOF," el cual es ASCII codificado como 44f4350524f4f46 en hexadecimal.

Tenga en cuenta que no existe un "script de desbloqueo" que corresponda a un OP_RETURN que pudiera ser usado para "gastar" una salida OP_RETURN. El propósito de OP_RETURN es que no se pueda gastar el dinero bloqueado en esa salida y por lo tanto no requiere ser conservado en la colección UTXO como potencialmente gastable—OP_RETURN es *demostrablemente ingastable*. OP_RETURN es generalmente una salida con un monto de cero bitcoins, ya que cualquier monto en

bitcoins asignado a tal salida sería efectivamente perdido para siempre. Si un OP_RETURN es encontrado por el software de validación de scripts, resultaría en la detención inmediata de la ejecución del script de validación y marca de la transascción como inválida. Por este motivo, si accidentalmente se referencia una salida OP_RETURN como entrada de una transacción, esa transacción será inválida.

Una transacción estándar (una que cumple con los chequeos de isStandard()) puede tener tan solo una salida OP_RETURN. Sin embargo, una salida OP_RETURN única puede ser combinada en una transacción con varias salidas de otros tipos.

Dos nuevas opciones de línea de comandos han sido añadidas en Bitcoin Core en su versión 0.10. La opción datacarrier controla la transmisión y minado de transacciones OP_RETURN, con el valor por defecto de "1" para permitirlas. La opción datacarriersize toma un argumento numérico especificando el tamaño máximo en bytes de los datos en OP_RETURN, con 40 bytes por defecto.

> OP_RETURN fue propuesto inicialmente con un límite de 80 bytes, pero el límite fue reducido a 40 bytes cuando la funcionalidad fue liberada. En febrero de 2015, en la versión 0.10 de Bitcoin Core, el límite fue elevado nuevamente a 80 bytes. Los nodos pueden optar por no transmitir o minar OP_RETURNs, o simplemente transmitir y minar OP_RETURNs que contengan menos de 80 bytes de datos.

Pago-a-Hash-de-Script (P2SH)

El pago-a-hash-de-script (P2SH) fue introducido en 2012 como un poderoso nuevo tipo de transacción que generalmente simplifica el uso de scripts de transacciones complejos. Para explicar la necesidad de P2SH, veamos un ejemplo práctico.

En [ch01_intro_what_is_bitcoin] presentamos a Mohammed, un importador de productos electrónicos en Dubai. La compañia de Mohammed usa la funcionalidad multi-firma de bitcoin de forma exclusiva para sus cuentas corporativas. Los scripts multi-firma son uno de los usos más comunes de las capacidades avanzadas de scripting de bitcoin y son una funcionalidad muy potente. La compañía de Mohammed usa un script multi-firma para todos los pagos de clientes, conocidos en contabilidad como "cuentas por cobrar." Con el esquema multi-firma, cualquier pago hecho por clientes es bloqueado de forma que requieran al menos dos firmas para ser liberados, una de Mohammed y otra de sus socios o un abogado con una clave de backup. Un esquema multi-firma como ese ofrece a la gerencia corporativa controles y la protege de robo, malversación o pérdida.

El script resultante es bastante largo y se ve así:

```
2 <Clave Pública de Mohammed> <Clave Pública de Socio1> <Clave Pública de Socio2> <Clave
Pública de Socio3> <Clave Pública de Abogado> 5 OP_CHECKMULTISIG
```

Aunque los scripts multi-firma son una funcionalidad potente, son algo engorrosos de usar. Dado el script anterior, Mohammed tendría que comunicar este script a cada comprador antes de realizarse el

pago. Cada cliente tendría que usar un software de cartera bitcoin especial con la habilidad de crear scripts de transacción personalizados, y cada cliente tendría que entender cómo crear una transacción utilizando scripts personalizados. Es más, la transacción resultante tendría que ser alrededor de cinco veces mayor que una transacción de pago común y corriente, ya que este script contiene claves públicas muy largas. La carga de esa transacción extra larga recaería sobre el cliente en forma de tarifas. Por último, un script de transacción grande como este terminaría en la colección de UTXOs en la RAM de cada nodo completo hasta ser gastado. Todos estos problemas hacen el uso de scripts de salida complejos difícil en la práctica.

Pago-a-hash-de-script (pay-to-script-hash, o P2SH) fue desarrollado para resolver estas dificultades prácticas y hacer el uso de scripts complejos tan fácil como un pago a una dirección bitcoin. Con pagos P2SH los scripts de bloqueo complejos son reemplazados con su huella digital, un hash criptográfico. Cuando una transacción que intenta gastar una UTXO es luego presentada, debe contener el script que concuerda con el hash, además del script de desbloqueo. En términos simples, P2SH significa "pagar a un script que concuerde con este hash, un script que será presentado más tarde cuando esta salida sea gastada."

En las transacciones P2SH, el script de bloqueo que es reemplazado por un hash es referenciado como el *script de liquidación* (redeem script), ya que es presentado al sistema al momento de liquidación en vez de como un script de bloqueo. Script complejo sin P2SH muestra el script sin P2SH y Script complejo como P2SH muestra el mismo script codificado con P2SH.

Table 4. Script complejo sin P2SH

Script de Bloqueo	2 ClavePública1 ClavePública2 ClavePública3 ClavePública4 ClavePública5 5 OP_CHECKMULTISIG
Script de Desbloqueo	Firma1 Firma2

Table 5. Script complejo como P2SH

Script de Liquidación	2 ClavePública1 ClavePública2 ClavePública3 ClavePública4 ClavePública5 5 OP_CHECKMULTISIG
Script de Bloqueo	OP_HASH160 <hash de 20 bytes del script de liquidación> OP_EQUAL
Script de Desbloqueo	Firma1 Firma2 script de liquidación

Como puede ver de las tablas, con P2SH el script complejo que detalla las condiciones para gastar la salida (script de liquidación) no es presentado en el script de bloqueo. En cambio, solo su hash se encuentra en el script de bloqueo y el script de liquidación en sí es presentado luego, como parte del script de desbloqueo cuando la salida es gastada. Esto mueve la carga tarifaria y la complejidad del remitente al destinatario (gastador) de la transacción.

Observemos la compañia de Mohammed, el complejo script multi-firma, y los scripts P2SH resultantes.

Primero, el script multi-firma que usa la compañia de Mohammed para todos sus pagos de clientes entrantes:

```
2 <Clave Pública de Mohammed> <Clave Pública de Socio1> <Clave Pública de Socio2> <Clave Pública de Socio3> <Clave Pública de Abogado> 5 OP_CHECKMULTISIG
```

Si los marcadores de posición son reemplazados por claves públicas reales (mostradas aquí como números de 520 bits comenzados en 04) se puede observar que el script se vuelve muy largo:

```
2
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7EEC6984
D83F1F50C900A24DD47F569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308
EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D78D0E34224858008E8B49047E632
48B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA7D4420D7E1A9162F0279CFC1
0F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD7149B255382ED7F78E946580657EE6FDA162A187543A9
D85BAAA93A4AB3A8F044DADA618D087227440645ABE8A35DA8C5B73997AD343BE5C2AFD94A5043752580AFA1E
CED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF518C022DD618DA774D207D1
37AAB59E0B000EB7ED238F4D800 5 OP_CHECKMULTISIG
```

La totalidad de este script puede ser en cambio reemplazado por un hash criptográfico de 20 bytes, aplicando primero el algoritmo de hashing SHA256 y luego el algoritmo RIPEMD160 sobre el resultado. El hash de 20 bytes del script anterior es:

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

Una transacción P2SH bloquea la salida a este hash en vez del script más largo, usando el script de bloqueo:

```
OP_HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e OP_EQUAL
```

el cual, como se puede ver, es mucho más breve. En vez de "pagar a este script multi-firma de 5 claves," la transacción P2SH equivalente es "pagar al script con este hash." Un cliente realizando un pago a la compañía de Mohammed solo necesita incluir este script de bloqueo mucho más corto en su pago. Cuando Mohammed desea gastar esta UTXO, debe presentar el script de liquidación original (cuyo hash bloqueó la UTXO) y las firmas necesarias para desbloquearla, así:

```
<Firma1> <Firma2> <2 CP1 CP2 CP3 CP4 CP5 5 OP_CHECKMULTISIG>
```

Ambos scripts son combinados en dos etapas. Primero, el script de liquidación es chequeado contra el script de bloqueo para asegurar que el hash concuerda:

```
<2 CP1 CP2 CP3 CP4 CP5 5 OP_CHECKMULTISIG> OP_HASH160 <hash de script de liquidación>
OP_EQUAL
```

Si el hash del scritp de liquidación concuerda, el script de desbloqueo es ejecutado por su cuenta para desbloquear el script de liquidación:

```
<Firma1> <Firma2> 2 CP1 CP2 CP3 CP4 CP5 5 OP_CHECKMULTISIG
```

Direcciones pago-a-hash-de-script

Otra parte importante de la funcionalidad de P2SH es la habilidad de codificar un hash de un script en una dirección, tal como se define en BIP0013. Las direcciones P2SH son codificaciones Base58Check del hash de 20 bytes de un script, tal como las direcciones bitcoin son codificaciones Base58Check del hash de una clave pública de 20 bytes. Las direcciones P2SH usan el prefijo de versión "5", que resulta en direcciones codificadas en Base58Check comenzadas en "3". Por ejemplo, el script complejo de Mohammed, hasheado y codificado en Base58Check como una dirección P2SH se convierte en 39RF6JqABiHdYHkfChV6USGMe6Nsr66Gzw. Ahora Mohammed puede distribuir esta "dirección" a sus clientes y ellos pueden usar prácticamente cualquier cartera bitcoin para hacer un pago sencillo, como si fuera una dirección bitcoin. El prefijo 3 da un indicio de que este es un tipo especial de dirección, uno que corresponde a un script en vez de a una clave pública, pero funciona exactamente de la misma manera que un pago a una dirección bitcoin salvando esa diferencia.

Las direcciones P2SH esconden toda la complejidad de forma que la persona realizando el pago no vea el script.

Beneficios del pago-a-hash-de-script

La funcionalidad de pago-a-hash-de-script ofrece los siguientes beneficios comparado al uso directo de scripts complejos en bloqueo de salidas:

- Scripts complejos son reemplazados por huellas más cortas en la salida de transacción, reduciendo la transacción.
- Los scripts pueden ser codificados como una dirección, de forma que el remitente y la cartera del remitente no necesitan ingeniería compleja para implementar P2SH.
- P2SH desplaza la carga de construir el script al destinatario, no el remitente.
- P2SH mueve la carga en almacenamiento de datos para scripts largos de la salida (la cual se encuentra en la colección de UTXOs) a la entrada (almacenada en la cadena de bloques).
- P2SH mueve la carga en almacenamiento de datos para el script largo del tiempo presente (pago) a un tiempo futuro (cuando es gastado).
- P2SH mueve los costos de tarifas de transacción de un script largo del remitente al destinatario, quien debe incluir el largo script de liquidación para gastarlo.

Script de liquidación y validación isStandard

Antes de la versión 0.9.2 del cliente Bitcoin Core, el pago-a-hash-de-script se encontraba limitado a tipos de scripts de transacciones bitcoin estándar, validados por la función isStandard(). Eso significa que el script de liquidación presentado en la transacción de gasto podía ser tan solo uno de los tipos estándar: P2PK, P2PKH, o multi-firma, excluyendo OP_RETURN y P2SH mismo.

Hacia la versión 0.9.2 del cliente Bitcoin Core, las transacciones P2SH pueden contener cualquier script válido, haciendo al estándar P2SH mucho más flexible y permitiendo experimentación con muchos tipos de transacciones novedosos y complejos.

Nótese que no es posible poner un P2SH dentro de un script de liquidación P2SH ya que la especificación P2SH no es recursiva. Tampoco es posible utilizar OP_RETURN dentro de un script de liquidación ya que OP_RETURN no puede ser liquidado por definición.

Nótese que como el script de liquidación no se presenta a la red hasta el momento de gastar una salida P2SH, si usted bloquea una salida con el hash de una transacción inválida será procesado de todos modos. Sin embargo, no podrá gastarlo ya que la transacción de gasto, la cual incluye el script de liquidación, no será aceptado ya que es un script inválido. Esto crea un riesgo, ya que es posible bloquear bitcoins en un P2SH que no puede ser gastado. La red aceptará la obstrucción P2SH aun si corresponde a un script de liquidación inválido, ya que el hash del script no da ninguna indicación de qué script representa.

WARNING	Los scripts de bloqueo P2SH contienen el hash de un script de liquidación, el cual no da pistas acerca del contenido del script de liquidación mismo. La transacción P2SH será considerada válida y aceptada aun si el script de liquidación no lo es. Es posible bloquear bitcoins accidentalmente de forma que no puedan ser gastados luego.

La Red Bitcoin

Arquitectura de Red Entre Pares (P2P)

Bitcoin se estructura como una arquitectura de red P2P sobre Internet. El término de igual a igual, o P2P, significa que las computadoras que participan en la red son iguales entre sí, que no hay nodos "especiales", y que todos los nodos comparten la carga de proveer servicios a la red. Los nodos de la red se interconectan en una malla de redes con una topología "plana". No hay servidor, no hay servicio centralizado, ni jerarquía dentro de la red. Los nodos en una red P2P proporcionan servicios y consumen servicios al mismo tiempo, con la reciprocidad como incentivo para participar. Las redes P2P son inherentemente resistentes, decentralizadas y abiertas. El ejemplo más destacado de una arquitectura de red P2P fue la Internet temprana, donde los nodos de la red IP eran iguales. La estructura del Internet actual es más jerárquica, pero el Protocolo de Internet mantiene la esencia de topología plana. Más allá de bitcoin, la más grande y exitosa aplicación de tecnologías P2P, está la compartición de archivos con Napster como pionera y la red BitTorrent como la más reciente evolución de la arquitectura.

La arquitectura de red entre pares (P2P) de bitcoin es mucho más que una elección topológica. Bitcoin es un sistema de dinero en efectivo entre pares por diseño, y la arquitectura de red es tanto una reflexión y un fundamento de esa característica base. La descentralización del control es un principio de diseño base y eso solo puede ser alcanzado y mantenido por una red de consenso entre pares plana y descentralizada.

El término "red bitcoin" se refiere a la colección de nodos que ejecutan el protocolo p2p bitcoin. Además del protocolo P2P bitcoin, hay otros protocolos tales como Stratum, que se utilizan para la minería y carteras ligeras o móviles. Estos protocolos adicionales son proporcionados por servidores de enrutamiento de puerta de enlace que acceden a la red Bitcoin utilizando el protocolo P2P bitcoin, y que luego se extienden por esa red de nodos que ejecutan otros protocolos. Por ejemplo, los servidores Stratum conectan los nodos de minería Stratum través del protocolo Stratum a la red bitcoin principal y hacen de puente entre el protocolo Stratum y el protocolo P2P bitcoin. Utilizamos el término "red bitcoin extendida" para referirnos a la red global que incluye el protocolo p2p bitcoin, los protocolos de pool de minería, el protocolo de Stratum, y cualesquiera otros protocolos relacionados que conectan los componentes del sistema de Bitcoin.

Tipos de Nodos y Roles

Aunque los nodos en la red P2P bitcoin son iguales, puede que asuman roles distintos dependiendo de la funcionalidad que soporten. Un nodo bitcoin es una colección de funciones: enrutamiento, la base de datos de la cadena de bloques (en inglés, "blockchain"), minado y servicios de cartera. Una nodo completo con todas estas funciones se detalla en Un nodo de la red bitcoin con todas sus cuatro funciones: cartera, minero, base de datos de cadena de bloques completa, y enrutamiento de red.

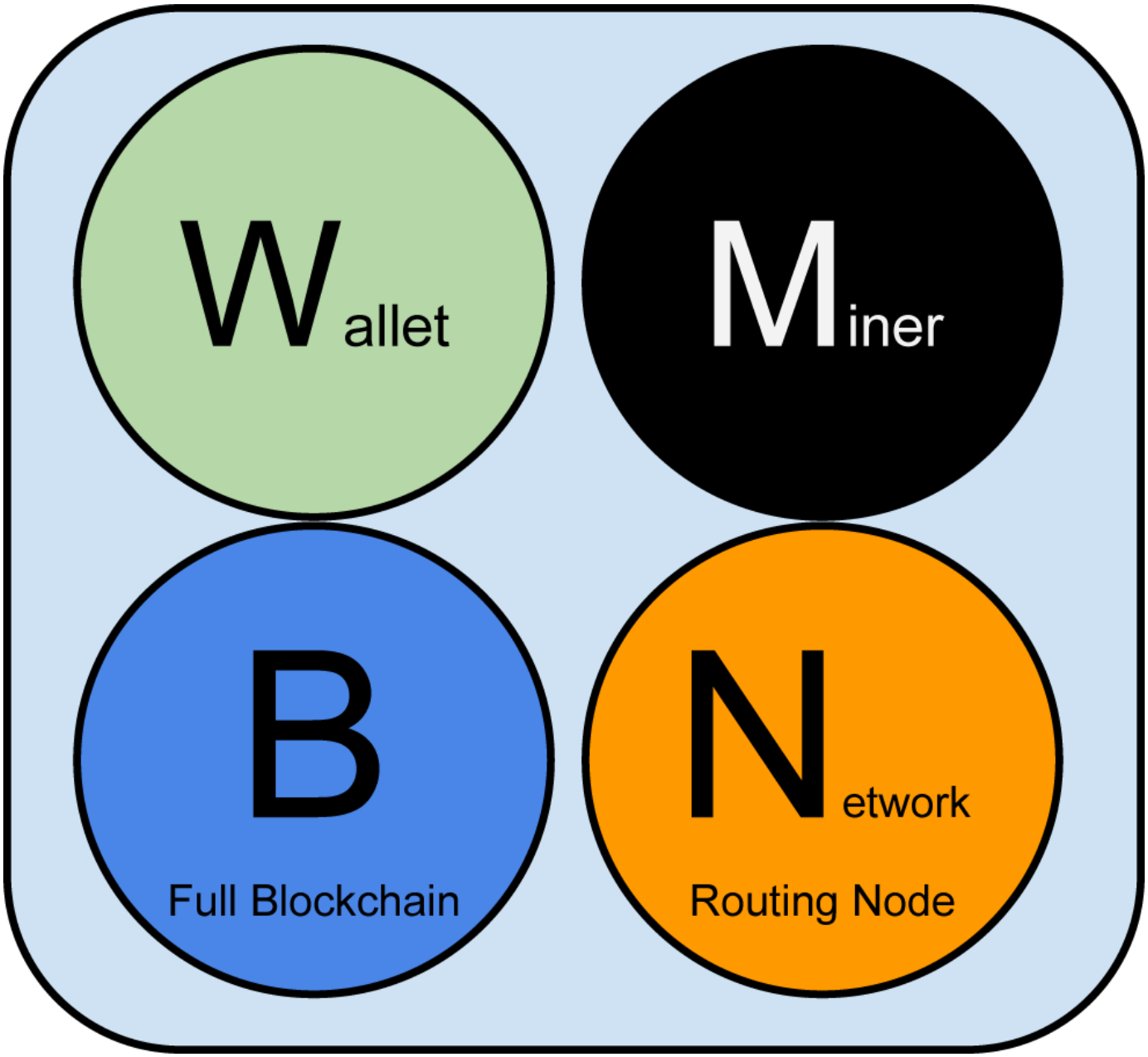


Figure 1. Un nodo de la red bitcoin con todas sus cuatro funciones: cartera, minero, base de datos de cadena de bloques completa, y enrutamiento de red

Todos los nodos incluyen la función de enrutamiento para participar en la red y pueden incluir otras funcionalidades. Todos los nodos validan y propagan las transacciones y bloques, y descubren y mantienen conexiones con sus compañeros. En el ejemplo de nodo completo Un nodo de la red bitcoin con todas sus cuatro funciones: cartera, minero, base de datos de cadena de bloques completa, y enrutamiento de red, la función de enrutamiento se indica mediante un círculo de color naranja llamado "Network Routing Node".

Algunos nodos, denominados nodos completos, también mantienen una completa y actualizada copia de la cadena de bloques. Los nodos completos pueden verificar cualquier transacción de forma autónoma, concluyente y sin referencia externa. Algunos nodos mantienen solo un subconjunto de la cadena de bloques y verifican las transacciones utilizando un método llamado nodos de *verificación de pago simplificado* o SPV. Estos nodos son conocidos como SPV o nodos de peso ligero. En la figura de

ejemplo, la función de base de datos de la cadena de bloques de un nodo completo se indica mediante un círculo azul llamado "Full Blockchain." En la La red bitcoin extendida muestra varios tipos de nodos, puertas de enlace y protocolos, los nodos SPV se dibujan sin el círculo azul, mostrando que no tienen una copia completa de la cadena de bloques.

Los nodos de Minería compiten para crear nuevos bloques ejecutando hardware especializado para resolver el algoritmo de prueba de trabajo. Algunos nodos de minería son también nodos completos, manteniendo una copia completa del blockchain, mientras que otros son nodos ligeros que participan en el pool de la minería y en función de un servidor de grupo para mantener un nodo completo. La función de la minería se muestra en el nodo completo como un círculo negro llamado "Miner".

Carteras de usuario podrían formar parte de un nodo completo, como suele ser el caso con los clientes Bitcoin escritorio. Cada vez más, muchas carteras de los usuarios, en especial los que se ejecuta en dispositivos con recursos limitados, tales como teléfonos inteligentes, son nodos SPV. La función de la cartera se muestra en la Un nodo de la red bitcoin con todas sus cuatro funciones: cartera, minero, base de datos de cadena de bloques completa, y enrutamiento de red como un círculo verde llamado "Monedero".

Además de los principales tipos de nodos en el protocolo P2P bitcoin, hay servidores y nodos que ejecutan otros protocolos, como los protocolos de pool de minera especializados y protocolos de cliente de acceso ligeros.

Diferentes tipos de nodos sobre la red bitcoin extendida muestra los tipos de nodos más comunes en la red bitcoin extendida.

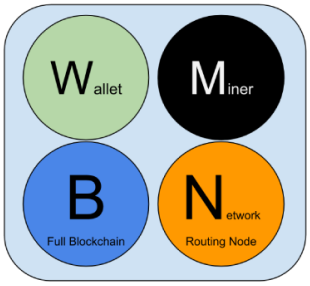
La Red Bitcoin Extendida

La red bitcoin principal, ejecuta el protocolo P2P bitcoin, consta de entre 7.000 y 10.000 nodos de escucha que ejecutan diferentes versiones del bitcoin cliente de referencia (Bitcoin Core) y unos pocos cientos de nodos que ejecutan varias otras implementaciones del protocolo P2P bitcoin, tales como ("biblioteca libbitcoin") BitcoinJ, Libbitcoin, y btcd. Un pequeño porcentaje de los nodos de la red P2P bitcoin también están minando los nodos, que compiten en el proceso de minería, la validación de las transacciones, y la creación de nuevos bloques. Varios interfaz de grandes empresas con la red bitcoin ejecutando clientes de nodo completo basado en el cliente Bitcoin Core, con copias completas de la blockchain y un nodo de la red, pero sin la minería o las funciones de monedero. Estos nodos actúan como routers de borde de la red, permitiendo que varios otros servicios (casas de cambio, carteras, exploradores bloque, procesamiento de pagos de comerciantes) que se construirán en la parte superior.

La red bitcoin extendida incluye la red que ejecuta el protocolo P2P bitcoin, descrito anteriormente, así como nodos que ejecutan protocolos especializados. Adjuntos a la red P2P bitcoin principal hay una serie de servidores de pool y pasarelas de protocolo que conectan los nodos que ejecutan otros protocolos. Estos otros nodos de protocolo son en su mayoría los nodos de minería del pool (ver [ch8]) y los clientes de carteras ligeros, que no llevan una copia completa de la cadena de bloques.

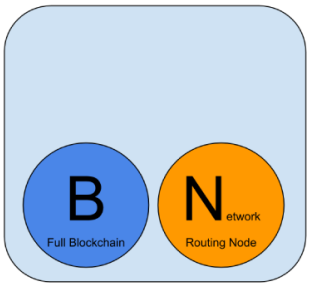
La red bitcoin extendida muestra varios tipos de nodos, puertas de enlace y protocolos muestra la red

bitcoin extendida con los distintos tipos de nodos, servidores de puerta de enlace, los routers de borde, y los clientes de cartera y los distintos protocolos que utilizan para conectarse entre sí.



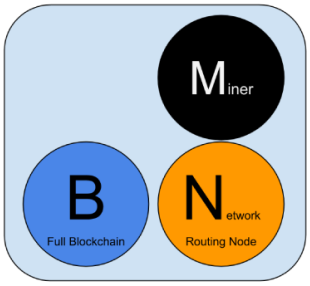
Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.



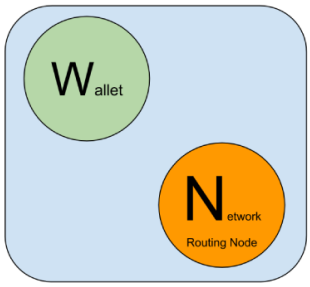
Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.



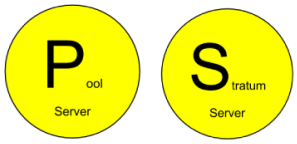
Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.



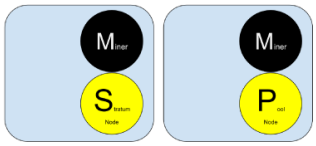
Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.



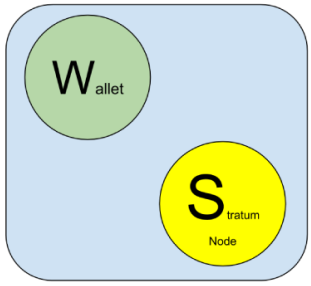
Pool Protocol Servers

Gateway routers connecting the bitcoin P2P network to nodes running other protocols such as pool mining nodes or Stratum nodes.



Mining Nodes

Contain a mining function, without a blockchain, with the Stratum protocol node (S) or other pool (P) mining protocol node.



Lightweight (SPV) Stratum wallet

Contains a Wallet and a Network node on the Stratum protocol, without a blockchain.

Figure 2. Diferentes tipos de nodos sobre la red bitcoin extendida

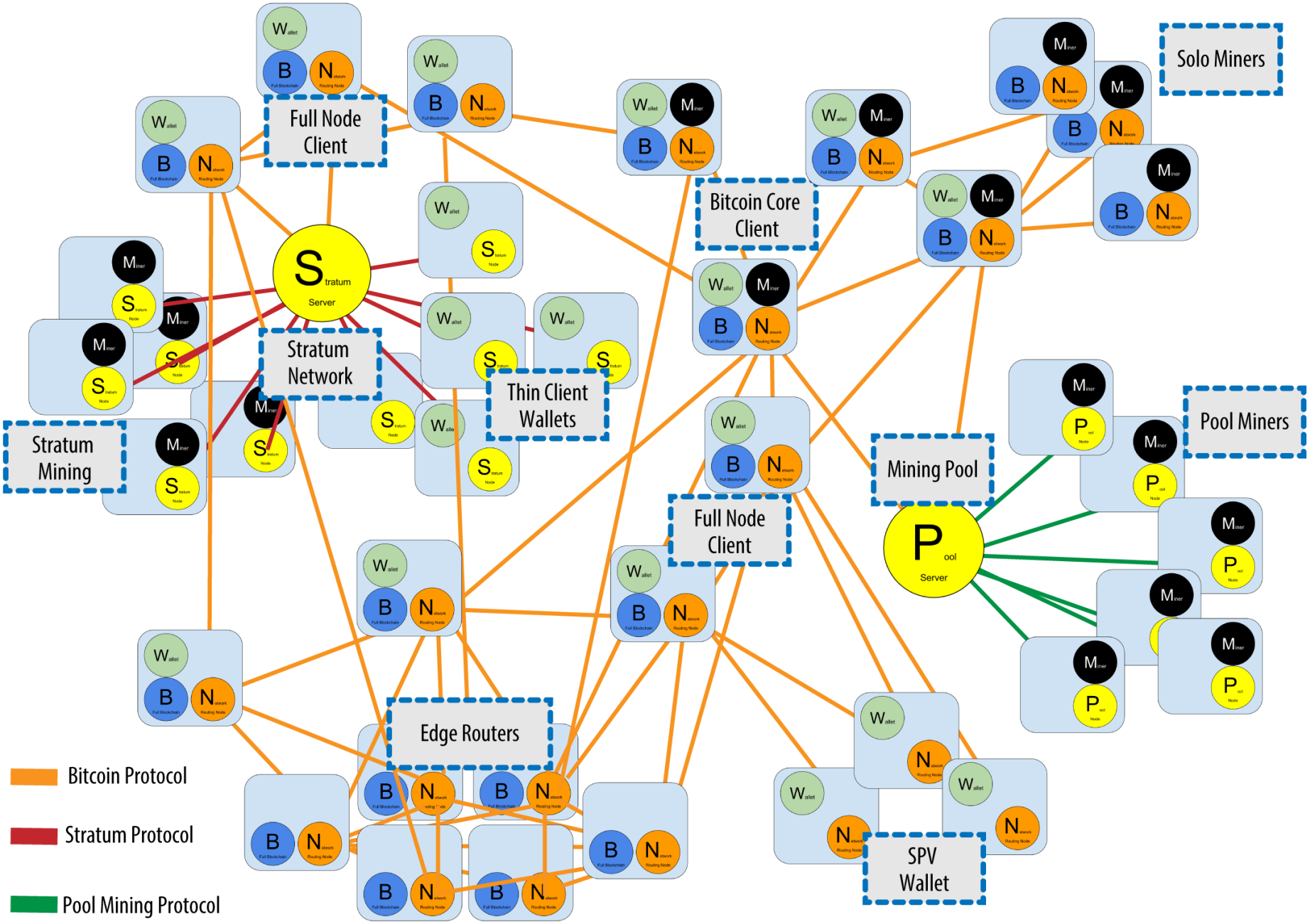

Full Node Client
Solo Miners
Bitcoin Core Client
Stratum Server
Stratum Network
Thin Client Wallets
Stratum Mining
Mining Pool
Pool Server
Pool Miners
Full Node Client
Edge Routers
SPV Wallet
Bitcoin Protocol
Stratum Protocol
Pool Mining Protocol

Figure 3. La red bitcoin extendida muestra varios tipos de nodos, puertas de enlace y protocolos

Descubrimiento de Red

Cuando un nodo nuevo arranca, debe descubrir otros nodos en la red bitcoin para poder participar. Para iniciar este proceso, un nuevo nodo debe descubrir al menos un nodo existente en la red y conectarse a él. La ubicación geográfica de los otros nodos es irrelevante; la topología de la red bitcoin no está definida geográficamente. Por lo tanto, cualquier nodo bitcoin existente puede ser seleccionado al azar.

Para conectarse a un compañero conocido, los nodos establecen una conexión TCP, por lo general en el puerto 8333 (puerto conocido generalmente como el utilizado por bitcoin), o un puerto alternativo si se proporciona. Al establecer una conexión, el nodo se iniciará un "apretón de manos" (ver <<network_handshake> >) mediante la transmisión de un mensaje de versión, que contiene información básica de identificación, incluyendo:

PROTOCOL_VERSION

Una constante que define la versión del protocolo P2P bitcoin que el cliente "habla" (por ejemplo, 70002)

nLocalServices

Una lista de los servicios locales soportados por el nodo, actualmente solo NODE_NETWORK

nTime

La fecha y hora actuales

AddrYou

La dirección IP del nodo remoto como se ve desde este nodo

AddrMe

La dirección IP del nodo local, tal como se ve desde el otro nodo local

Subver

Una sub-versión que muestra el tipo de software que se ejecuta en este nodo (por ejemplo, "/Satoshi:0.9.2.1/")+

BestHeight

La altura de bloque de la cadena de bloques de este nodo

(Ver GitHub para un ejemplo de la versión del mensaje de red).

El nodo de pares responde con verack para reconocer y establecer una conexión, y envía opcionalmente su propio mensaje de versión si quiere corresponder a la conexión y conectar de nuevo como un igual.

¿Cómo funciona un nuevo nodo para encontrar compañeros? El primer método consiste en hacer una consulta DNS utilizando una serie de "semillas de DNS", que son los servidores DNS que proporcionan una lista de direcciones IP de nodos Bitcoin. Algunas de esas semillas DNS proporcionan una lista estática de direcciones IP de los nodos Bitcoin estables que están a la escucha. Algunas de las semillas de DNS son implementaciones personalizadas de BIND (Berkeley Internet Name Daemon) que devuelven un subconjunto aleatorio de una lista de direcciones de nodo bitcoin recogidos por un rastreador o un nodo bitcoin de larga duración. El cliente Bitcoin Core contiene los nombres de cinco semillas DNS diferentes. La diversidad de la propiedad y la diversidad de la implementación de las diferentes semillas DNS ofrece un alto nivel o fiabilidad del proceso de arranque inicial. En el cliente Bitcoin Core, la preferencia de utilizar las semillas de DNS se controla con la opción -dnsseed (ajustado a 1 por defecto, para usar la semilla DNS).

Alternativamente, un nodo nuevo en el proceso de arranque que no sabe nada de la red debe tener la dirección IP de al menos un nodo bitcoin, después de lo cual se puede establecer conexiones a través de nuevas presentaciones. El argumento de línea de comandos -seednode se puede utilizar para conectarse a un nodo solo para presentaciones, usándolo como una semilla. Después de utilizar el nodo de semilla inicial para hacer las presentaciones, el cliente se desconecta de ella y utiliza los compañeros recién descubiertos.

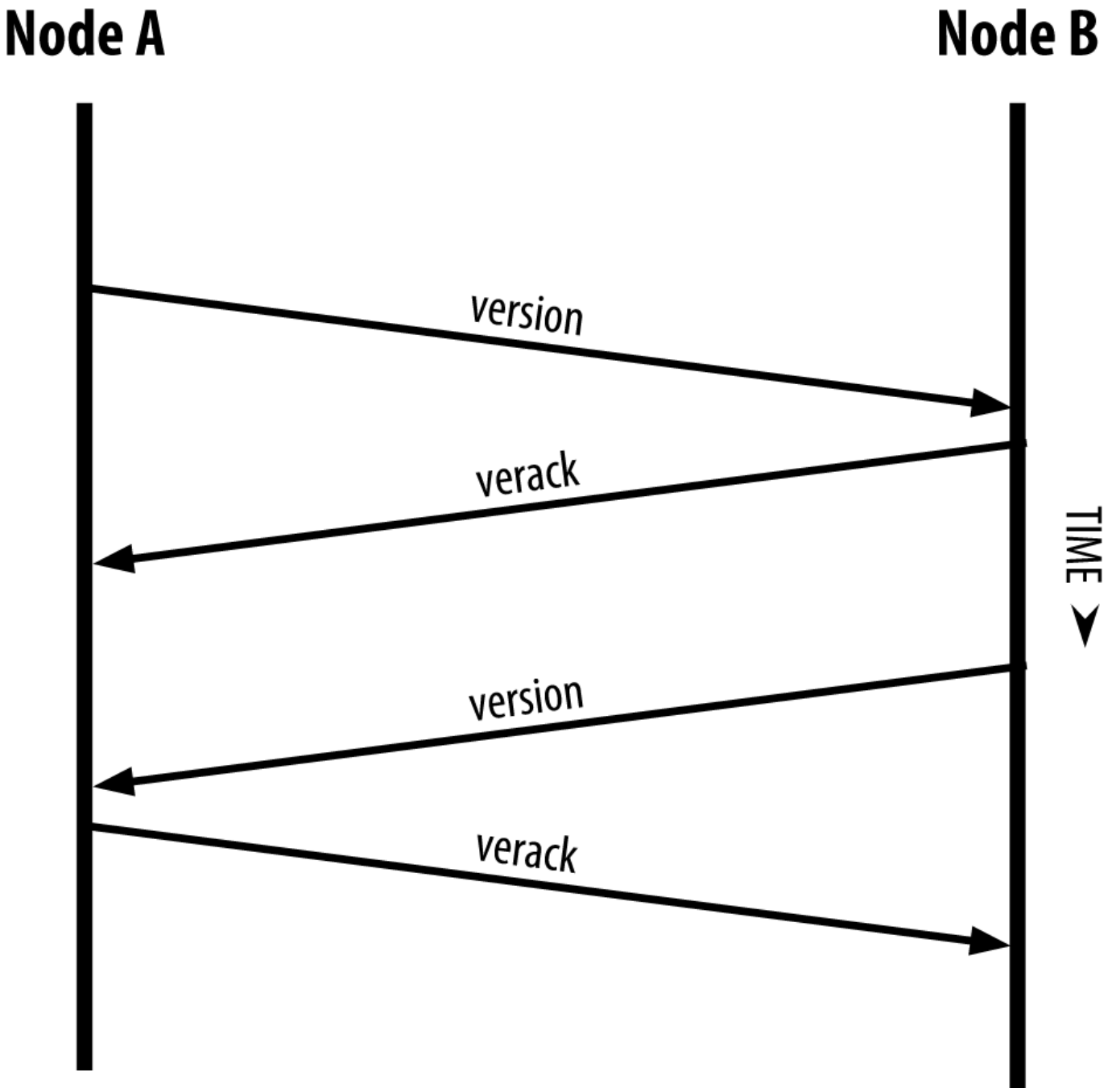


Figure 4. El apretón de manos inicial entre pares

Una vez que se han establecido una o más conexiones, el nuevo nodo enviará un mensaje addr que contiene su propia dirección IP para sus vecinos. Los vecinos, a su vez, remitirán el mensaje addr a sus vecinos, lo que garantiza que el nodo recién conectado se convierte en bien conocido y mejor conectado. Además, el nodo recién conectado puede enviar getaddr a los vecinos, pidiéndoles que le devuelvan una lista de direcciones IP de otros compañeros. De esa manera, un nodo puede encontrar compañeros para conectarse y anunciar su existencia en la red para que otros nodos puedan encontrarlo. Descubrimiento y propagación de la dirección muestra el protocolo de descubrimiento de direcciones.

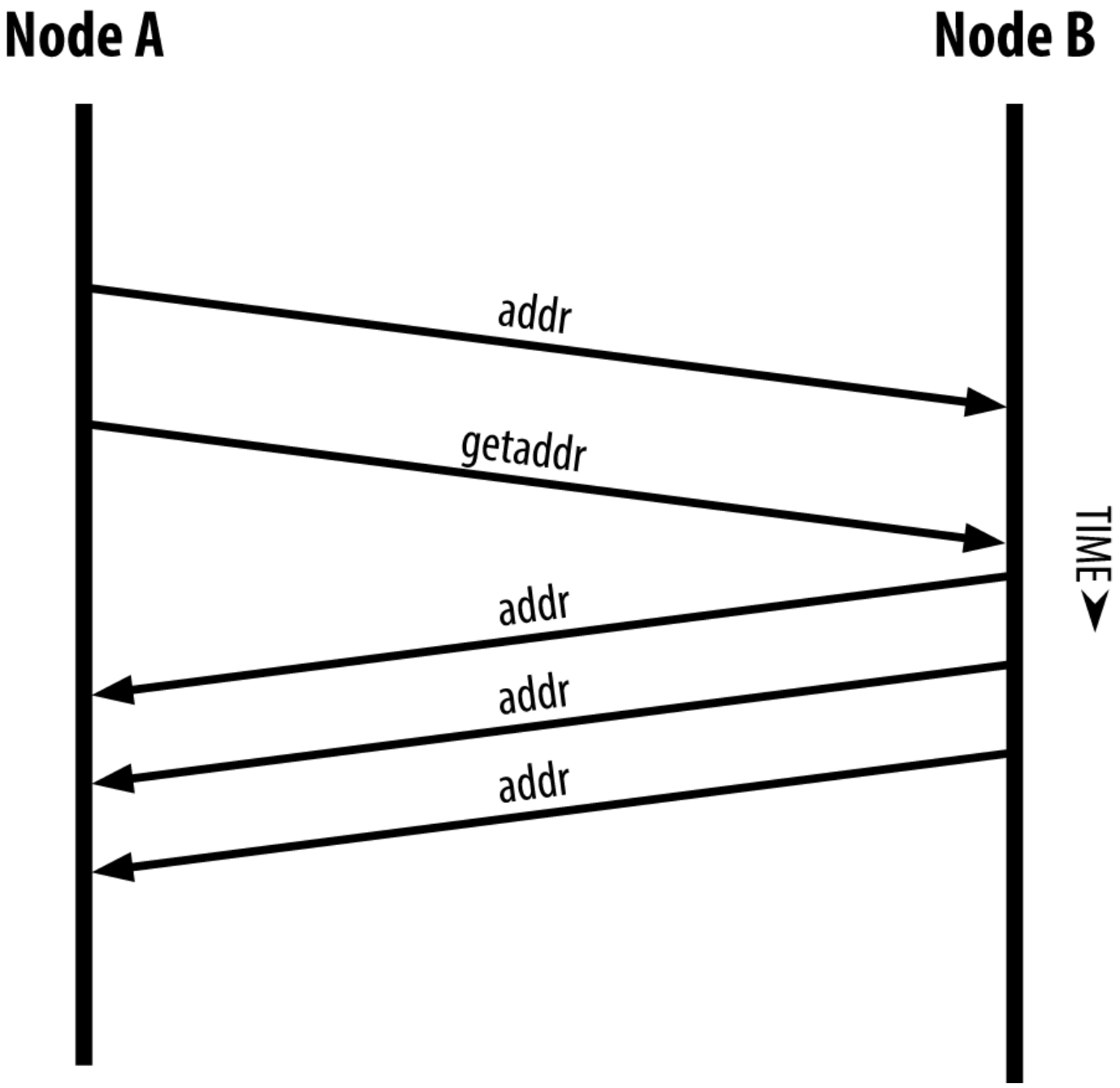


Figure 5. Descubrimiento y propagación de la dirección

Un nodo debe conectarse a unos cuantos compañeros diferentes a fin de establecer diversos caminos en la red Bitcoin. Los caminos no son fiables—nodos que van y vienen—, por lo que el nodo debe seguir descubriendo nuevos nodos a medida que pierde conexiones antiguas, así como ayudar a otros nodos que inician su proceso de arranque. Solo se necesita una conexión para arrancar, ya que el primer nodo puede ofrecer presentaciones a sus nodos pares y los pares puede ofrecer nuevas presentaciones. También es innecesario y derrochador de recursos de red conectarse a más de un puñado de nodos. Después de iniciarse, un nodo se acordará de sus conexiones exitosas entre pares más recientes, por lo que si se reinicia puede restablecer rápidamente las conexiones con su antigua red de pares. Si ninguno de los pares anteriores responder a su solicitud de conexión, el nodo puede utilizar los nodos de semillas para realizar el arranque de nuevo.

En un nodo que ejecuta el cliente Bitcoin Core, puede listar las conexiones entre pares con el comando getpeerinfo:

```
$ bitcoin-cli getpeerinfo
```


```
[
    {
        "addr" : "85.213.199.39:8333",
        "services" : "00000001",
        "lastsend" : 1405634126,
        "lastrecv" : 1405634127,
        "bytessent" : 23487651,
        "bytesrecv" : 138679099,
        "conntime" : 1405021768,
        "pingtime" : 0.00000000,
        "version" : 70002,
        "subver" : "/Satoshi:0.9.2.1/",
        "inbound" : false,
        "startingheight" : 310131,
        "banscore" : 0,
        "syncnode" : true
    },
    {
        "addr" : "58.23.244.20:8333",
        "services" : "00000001",
        "lastsend" : 1405634127,
        "lastrecv" : 1405634124,
        "bytessent" : 4460918,
        "bytesrecv" : 8903575,
        "conntime" : 1405559628,
        "pingtime" : 0.00000000,
        "version" : 70001,
        "subver" : "/Satoshi:0.8.6/",
        "inbound" : false,
        "startingheight" : 311074,
        "banscore" : 0,
        "syncnode" : false
    }
]
```


Para ignorar la gestión automática de pares y especificar una lista de direcciones IP, los usuarios pueden proporcionar la opción -connect=<IPAddress> y especificar una o más direcciones IP. Si se utiliza esta opción, el nodo sólo se conectará a las direcciones IP seleccionados, en lugar de descubrir y mantener las conexiones entre pares automáticamente.

Si no hay tráfico en una conexión, los nodos se envían periódicamente un mensaje para mantener la conexión. Si un nodo no se ha comunicado en una conexión por más de 90 minutos, se supone que está desconectado y se buscará un nuevo par. De este modo, la red se ajusta dinámicamente a los nodos transitorios y problemas de la red, y puede crecer y encogerse orgánicamente según sea necesario, sin ningún tipo de control central.

Nodos Completos

Los nodos completos son nodos que mantienen una cadena de bloques completa con todas las transacciones. Más exactamente, probablemente deberían ser llamados "nodos completos de la cadena de bloques." En los primeros años del bitcoin, todos los nodos eran nodos completos y actualmente el cliente Bitcoin Core es un nodo completo de la cadena de bloques. En los últimos dos años, sin embargo, las nuevas formas de clientes Bitcoin que se han introducido no mantienen una cadena de bloques completa, sino que corren como clientes ligeros. Examinaremos estos últimos con más detalle en la siguiente sección.

Los nodos completos de la cadena de bloques mantienen una copia completa y actualizada de la cadena de bloques de bitcoin con todas las transacciones, que construyen y verifican de forma independiente, empezando por el primer bloque (bloque génesis) y construyendo hacia arriba hasta el último bloque conocido en la red. Un nodo completo de la cadena de bloques puede verificar cualquier transacción de forma independiente y concluyente sin recurrir o depender de ningún otro nodo o fuente de información. El nodo completo de la cadena de bloques depende de la red para recibir actualizaciones sobre nuevos bloques de transacciones, que posteriormente verifica e incorpora en su copia local de la cadena de bloques.

La ejecución de un nodo completo de la cadena de bloques le da la pura experiencia bitcoin: la verificación independiente de todas las transacciones sin la necesidad de depender o confiar en ningún otro sistema. Es fácil saber si se está ejecutando un nodo completo, ya que requiere más de 20 gigabytes de almacenamiento persistente (espacio en disco) para almacenar la cadena de bloques completa. Si usted necesita una gran cantidad de espacio en disco y se tarda dos o tres días para sincronizar con la red, está ejecutando un nodo completo. Ese es el precio de la completa independencia y libertad de la autoridad central.

Hay algunas implementaciones alternativas en los clientes bitcoin completos de la cadena de bloques, construidas utilizando diferentes lenguajes de programación y arquitecturas de software. Sin embargo, la aplicación más común es el cliente de referencia Bitcoin Core, también conocido como el cliente Satoshi. Más del 90% de los nodos en la red bitcoin ejecutan varias versiones de Bitcoin Core. Se identifica como "Satoshi" en la cadena de sub-versión enviada en el mensaje versión y se muestra mediante el comando getpeerinfo como vimos anteriormente; por ejemplo, /Satoshi:0.8.6/.

Intercambiando "Inventario"

La primera cosa que un nodo completo hará una vez que se conecta a los compañeros es tratar de construir una cadena de bloques completa. Si es un nodo nuevo y no tiene cadena de bloques en absoluto, entonces solo conoce un bloque, el bloque génesis, que está integrado de forma estática en el

software del cliente. Comenzando con el bloque #0 (el bloque génesis), el nuevo nodo tendrá que descargar cientos de miles de bloques para sincronizar con la red y volver a establecer la cadena de bloques completa.

El proceso de sincronización de la cadena de bloques comienza con el mensaje version, porque contiene la BestHeight, que es la altura actual de la cadena de bloques de un nodo (número de bloques). Un nodo verá los mensajes version de sus compañeros para saber cuántos bloques tiene cada uno, y ser capaz de comparar con el número de bloques que tiene en su propia cadena de bloques. Los nodos intercambiarán el mensaje getblocks que contiene el hash (huella digital) del bloque de la parte superior de su cadena de bloques local. Uno de los compañeros será capaz de identificar el hash recibido como perteneciente a un bloque que no está en la cima, sino que pertenece a un bloque más antiguo, deduciendo de esta manera que su propia cadena de bloques local es más larga que la de su compañero.

El nodo que tiene la cadena de bloques más larga, tiene mayor cantidad de bloques y puede identificar qué bloques necesita el otro nodo para "ponerse al día". Identificará los primeros 500 bloques a compartir y transmitirá sus valores hash utilizando un mensaje inv + (de inventario). El nodo al que le falten estos bloques podrá luego recuperarlos mediante la emisión de una serie de mensajes +getdata, solicitando los datos del bloque completo e identificando los bloques solicitados mediante los hashes del mensaje inv.

Supongamos, por ejemplo, que un nodo solo tiene el bloque de génesis. A continuación, recibirá un mensaje inv de sus pares que contiene los hashes de los próximos 500 bloques en la cadena. Comenzará solicitando bloques de todos sus pares conectados, repartiendo la carga y asegurando que no abrume sus peticiones a ningún par. El nodo mantiene un registro de cuántos bloques están "en tránsito" por cada conexión de pares, es decir, aquellos bloques que ha solicitado pero que aún no ha recibido, comprobando que no exceden un límite (MAX_BLOCKS_IN_TRANSIT_PER_PEER). De esta manera, si necesita una gran cantidad de bloques, solo solicitará otros nuevos a medida que se completan las solicitudes anteriores, permitiendo a los compañeros controlar el ritmo de las actualizaciones y no sobrecargar la red. A medida que se recibe cada bloque, se va agregando a la cadena de bloques, tal como veremos en [blockchain]. A medida que la cadena de bloques local se va construyendo gradualmente, se solicitan y se reciben más bloques, y el proceso continúa hasta que el nodo se pone al día con el resto de la red.

Este proceso de comparar la cadena de bloques local con los compañeros y la recuperación de todos los bloques que faltan sucede cada vez que un nodo se desconecta por cualquier período de tiempo. Ya sea un nodo que ha estado desconectado durante unos minutos y faltan pocos bloques, o un mes y faltan unos pocos miles de bloques, se inicia mediante el envío de getblocks, recibe un inv de respuesta, y comienza la descarga de los bloques que faltan. Nodo sincronizando la cadena de bloques pidiendo bloques a un par muestra el protocolo de inventario y la propagación de bloque.

Nodos de Verificación de Pago Simplificada (SPV)

No todos los nodos tienen la capacidad de almacenar la cadena de bloques completa. Muchos clientes bitcoin están diseñados para funcionar en dispositivos con restricciones de espacio y de energía, tales

como teléfonos inteligentes, tabletas o sistemas embebidos. Para tales dispositivos, se utiliza un método de *verificacion de pago simplificada* (SPV) que permite operar sin almacenar la cadena de bloques completa. Este tipo de clientes se llaman clientes SPV o clientes ligeros. Con el aumento en la adopción de bitcoin, el nodo SPV se está convirtiendo en la forma más común de nodo bitcoin, especialmente para carteras Bitcoin.

Los nodos SPV descargan solo las cabeceras de bloque y no descargan las transacciones incluidas en cada bloque. La cadena resultante de bloques, sin transacciones, es 1000 veces menor que la cadena de bloques completa. Los nodos SPV no pueden construir una imagen completa de todos los UTXOs que están disponibles para el gasto, ya que no saben acerca de todas las transacciones en la red. Los nodos SPV verifican las transacciones utilizando una metodología ligeramente diferente que depende de los pares para proporcionar vistas parciales de las partes relevantes de la cadena de bloques bajo demanda.

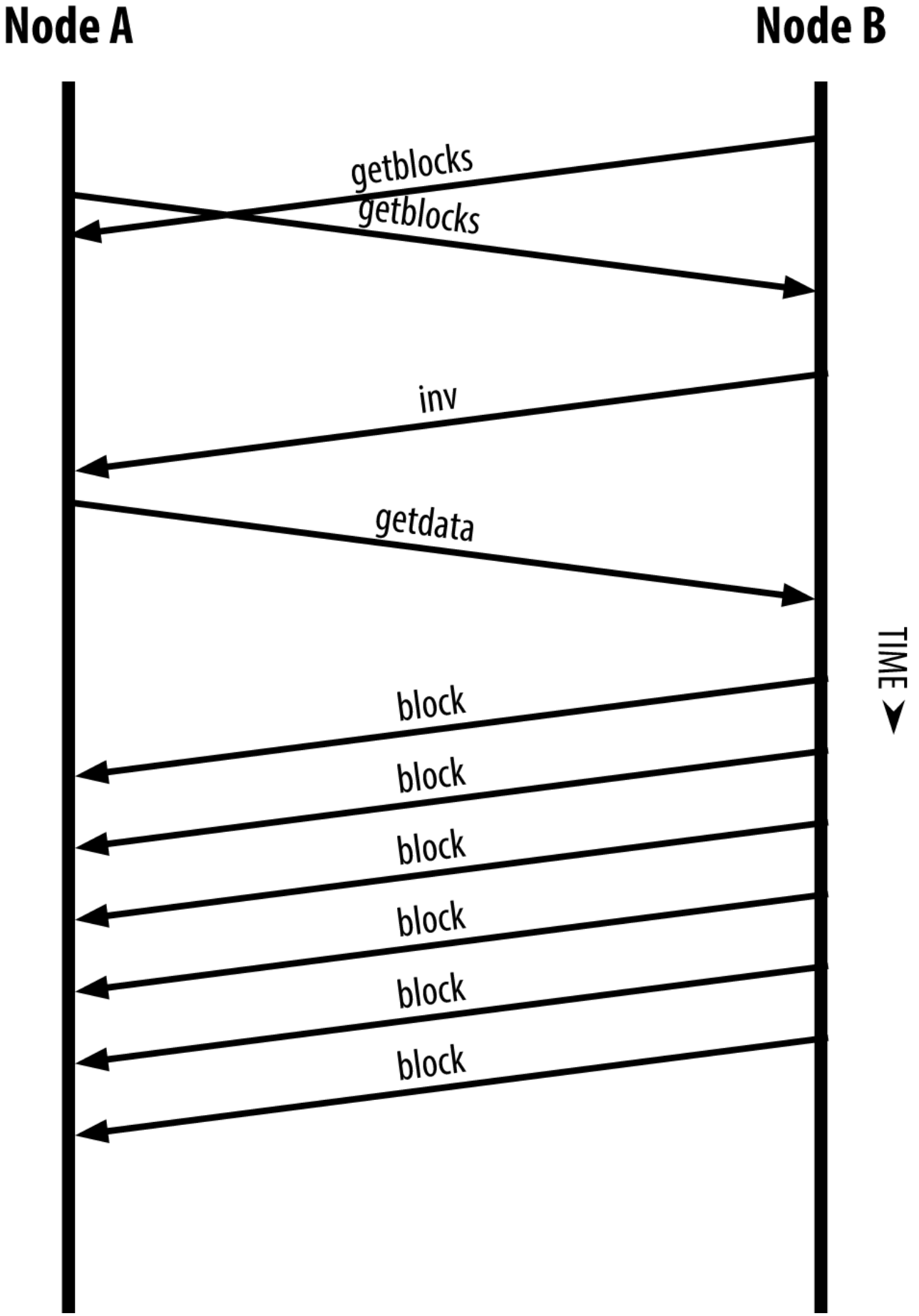

Node A
Node B
getblocks
getblocks
inv
getdata
block
block
block
block
block
block
TIME

Figure 6. Nodo sincronizando la cadena de bloques pidiendo bloques a un par

Como analogía, un nodo completo es como un turista en una ciudad extraña, equipado con un mapa detallado de cada calle y de cada dirección. En comparación, un nodo SPV es como un turista en una ciudad extraña preguntando a extraños al azar indicaciones giro a giro conociendo solo una avenida principal. Aunque ambos turistas pueden verificar la existencia de una calle al visitarla, el turista sin un mapa no sabe lo que hay más allá de las calles laterales y no sabe qué otras calles existen. Situado frente a 23 Church Street, el turista sin un mapa no puede saber si hay una docena de otras direcciones "23 Church Street" en la ciudad y si esta es la correcta. La mejor opción del turista sin mapas es preguntar a bastante gente y esperar que algunos de ellos no le estén tratando de robar.

La verificación de pago simplificada verifica las transacciones en función de su *profundidad* en la cadena de bloques, en vez de en su *altura*. Mientras que un nodo completo de la cadena de bloques construirá una cadena completamente verificada de miles de bloques y transacciones que alcanza (atrás en el tiempo) toda la cadena de bloques hasta el bloque génesis, un nodo SPV verificará la cadena de todos los bloques (pero no todas las transacciones) y vinculará esa cadena a la transacción de interés.

Por ejemplo, al examinar una transacción en el bloque 300.000, un nodo completo enlaza todos los 300.000 bloques desde el bloque génesis y crea una base de datos completa de UTXO, estableciendo la validez de la transacción mediante la comprobación de que el UTXO se encuentra sin gastar. Un nodo SPV no puede validar si el UTXO está sin gastar. En su lugar, el nodo SPV establecerá un vínculo entre la transacción y el bloque que lo contiene, usando una *ruta merkle*(ver [merkle_trees]). A continuación, el nodo SPV espera hasta que ve los seis bloques de 300.001 a 300.006, apilados encima del bloque que contiene la transacción y lo verifica mediante el establecimiento de su profundidad bajo los bloques 300.006 a 300.001. El hecho de que otros nodos de la red acepten el bloque 300.000 y luego hayan hecho el trabajo necesario para producir seis bloques más en la parte superior del mismo es la prueba, de forma indirecta, de que la operación no fue un doble gasto.

No se puede convencer a un nodo SPV de que existe una transacción en un bloque cuando la transacción en realidad no existe. El nodo SPV establece la existencia de una transacción en un bloque solicitando una prueba de ruta merkle y validando la prueba de trabajo en la cadena de bloques. Sin embargo, la existencia de una transacción puede estar "oculta" para un nodo SPV. Un nodo SPV puede definitivamente probar que existe una transacción, pero no puede verificar que una transacción, como un doble gasto de la misma UTXO, no exista, ya que no tiene un registro de todas las transacciones. Esta vulnerabilidad se puede utilizar en un ataque de denegación de servicio o en un ataque de doble gasto contra nodos SPV. Para defenderse de esto, un nodo SPV necesita conectarse al azar a varios nodos, y así aumentar la probabilidad de que esté en contacto con al menos un nodo honesto. Esta necesidad de conectarse de forma aleatoria tiene como consecuencia que los nodos SPV también sean vulnerables a los ataques de particionamiento de la red o a ataques Sybil, donde están conectados a nodos falsos o a redes falsas y no tienen acceso a nodos honestos o a la red bitcoin real.

A efectos prácticos, los nodos SPV bien conectados son suficientemente seguros, manteniendo el equilibrio adecuado entre las necesidades de recursos, practicidad y seguridad. Sin embargo, para una seguridad infalible lo mejor es ejecutar un nodo completo de cadena de bloques.

TIP

Un nodo completo de cadena de bloques verifica una transacción mediante la comprobación de toda la cadena de miles de bloques por debajo de ella con el fin de garantizar que el UTXO no esté gastado, mientras que un nodo SPV comprueba la profundidad del bloque, que estará cubierto solo por un puñado de bloques por encima de ella.

Para obtener las cabeceras de bloque, los nodos SPV utilizan un mensaje getheaders en lugar de getblocks. El compañero que responda, enviará hasta 2.000 cabeceras de bloques utilizando un único mensaje headers. El proceso es similar al utilizado por un nodo completo para recuperar bloques completos. Los nodos SPV también establecen un filtro en la conexión con sus compañeros, filtrando el flujo de bloques y futuras transacciones enviados por los compañeros. Cualquier transacción de interés se recupera mediante una petición getdata. El compañero genera como respuesta un mensaje tx que contiene las transacciones. Nodo SPV sincronizando las cabeceras de bloque muestra la sincronización de las cabeceras de bloque.

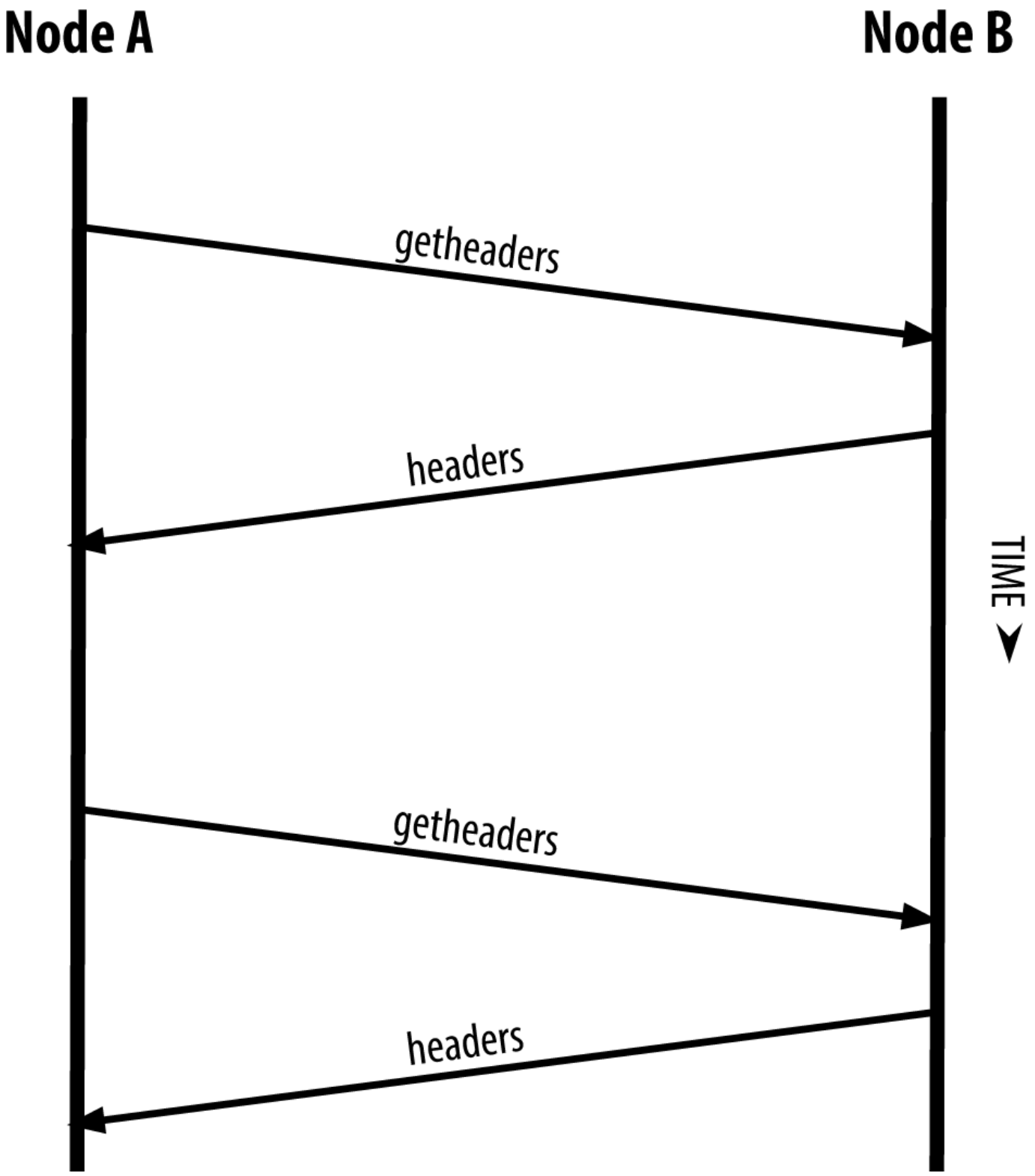


Figure 7. Nodo SPV sincronizando las cabeceras de bloque

El hecho de que los nodos SPV necesiten recuperar transacciones específicas para verificarlas de forma selectiva, hace que se genere un riesgo para la privacidad. A diferencia de los nodos completos de cadena de bloques, que recogen todas las transacciones dentro de cada bloque, las peticiones de datos específicos por parte de los nodos SPV pueden revelar inadvertidamente las direcciones de su cartera. Por ejemplo, un tercero podría monitorear la red y llevar un registro de todas las transacciones

solicitadas por una cartera en un nodo SPV, utilizando las asociaciones de direcciones bitcoin con el usuario de esa cartera y destruyendo la privacidad del usuario.

Poco después de la introducción de los SPV/nodos ligeros, los desarrolladores de bitcoin añadieron una característica llamada *filtros bloom* para abordar los riesgos de privacidad de los nodos SPV. Los filtros bloom permiten a los nodos SPV recibir un subconjunto de transacciones sin revelar con precisión en qué direcciones están interesados, a través de un mecanismo de filtrado que utiliza probabilidades en lugar de patrones fijos.

Filtros Bloom

Un filtro bloom es un filtro de búsqueda probabilística, una manera de describir un patrón deseado sin especificarlo exactamente. Los filtros bloom ofrecen una forma eficiente de expresar un patrón de búsqueda al mismo tiempo que se protege la privacidad. Se utilizan por los nodos SPV para pedir a sus compañeros las transacciones que coincidan con un patrón específico, sin revelar exactamente qué direcciones están buscando.

En nuestra analogía anterior, un turista sin mapas está pidiendo direcciones a una dirección específica, "23 Church St." Si preguntara a extraños por las direcciones a esta calle, estaría inadvertidamente revelando su destino. Un filtro de bloom es como preguntar: "¿Hay calles en este barrio cuyos nombres terminen en R-C-H?" Una pregunta como esa revela un poco menos sobre el destino deseado que pedir directamente "23 Church St." Usando esta técnica, un turista puede especificar la dirección deseada con mayor detalle, como "que termine en R-C-H", o con menor detalle, como "que termine en H." Mediante la variación de la precisión de la búsqueda, el turista revela más o menos información, a expensas de obtener resultados más o menos específicos. Si el turista preguntara con un patrón menos específico, obtendría muchas más direcciones posibles y una mejor privacidad, pero muchos de los resultados serían irrelevantes. Si preguntara con un patrón muy específico, obtendría menos resultados, pero perdería privacidad.

Los filtros bloom consiguen cumplir esta función al permitir que un nodo SPV especifique un patrón de búsqueda para las transacciones que puede ajustarse hacia precisión o hacia privacidad. Un filtro bloom más específico producirá resultados precisos, pero a expensas de revelar las direcciones que se utilizan en la cartera del usuario. Un filtro de bloom menos específico producirá más datos sobre más transacciones, muchos irrelevantes para el nodo, pero permitirá que el nodo pueda mantener mejor la privacidad.

Un nodo SPV inicializará un filtro bloom como "vacío", es decir, no contendrá ningún patrón. Después, el nodo SPV hará una lista de todas las direcciones en su cartera y creará un patrón de búsqueda que coincida con la salida de transacción que corresponda a cada dirección. Por lo general, el patrón de búsqueda es un script pago-a-clave-pública-hash (P2PKH, pay-to-public-key-hash en inglés), que es el script de bloqueo que debería estar presente en cualquier transacción que pague a la clave-pública-hash (dirección). Si el nodo SPV está rastreando el saldo de una dirección P2SH, entonces el patrón de búsqueda será un pago-a-script-hash (pay-to-script-hash en inglés). Después, el nodo SPV añadirá cada uno de los patrones de búsqueda al filtro bloom, de manera que el filtro bloom pueda reconocer el patrón de búsqueda en el caso de que esté presente en alguna transacción. Finalmente, el filtro bloom

se envía al compañero, que lo utiliza para encontrar transacciones que coincidan para ser transmitidas al nodo SPV.

Los filtros bloom se implementan como un vector (en inglés, "array") de tamaño variable de N dígitos binarios (un campo de un bit) y un número variable M de funciones hash. Las funciones hash están diseñadas para producir siempre una salida que está comprendida entre 1 y N, que corresponde al vector de dígitos binarios. Las funciones hash se generan de manera determinista, de modo que cualquier nodo que ejecute un filtro bloom siempre utilizará las mismas funciones hash y obtendrá los mismos resultados para una entrada específica. El filtro bloom puede ajustarse eligiendo diferentes longitudes (N) y un número diferente (M) de funciones de hash, variando así el nivel de precisión y por lo tanto la privacidad.

En Un ejemplo de filtro bloom simple, con un campo de 16 bits y tres funciones hash, usamos un pequeñ vector de 16 bits y un conjunto de tres funciones hash para demostrar cómo funcionan los filtros de Bloom.

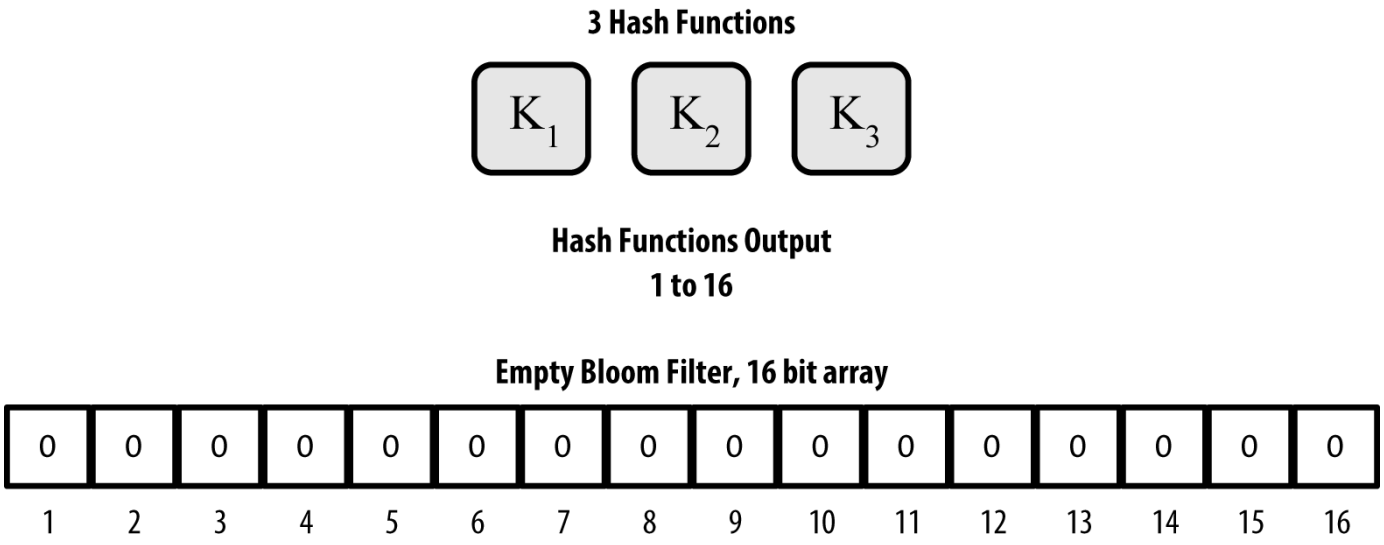


Figure 8. Un ejemplo de filtro bloom simple, con un campo de 16 bits y tres funciones hash

El filtro bloom se inicializa para que el vector de bits sea todo ceros. Para agregar un patrón al filtro bloom, se hace hash del patrón, una vez para cada función hash. La aplicación de la primera función de hash a la entrada da como resultado un número entre 1 y N. Se localiza el bit correspondiente en el vector (indexado de 1 a N) y se pone a 1, quedando registrada así la salida de la función hash. Entonces, se ejecuta la siguiente función hash para establecer otro bit, y así sucesivamente. Una vez de que se han aplicado todas las funciones de hash M, el patrón de búsqueda queda "registrado" en el filtro bloom como M bits que han cambiado de 0 a 1.

Añadiendo un patrón "A" a nuestro filtro bloom simple es un ejemplo de la adición de un patrón "A" para el filtro bloom sencillo mostrado en Un ejemplo de filtro bloom simple, con un campo de 16 bits y tres funciones hash.

Añadir un segundo patrón es tan simple como repetir este proceso. Se hace hash del patrón mediante la ejecución cada una de las funciones hash, y el resultado se registra mediante el establecimiento de

los bits a 1. Tenga en cuenta que a medida que un filtro bloom se llena con más patrones, algún resultado de la función de hash podría coincidir con uno que ya está marcado a 1, en cuyo caso no se cambia el bit. En esencia, la aparición de más patrones que se registren como bits superpuestos es la señal de que el filtro bloom comienza a saturarse con más bits establecidos en 1, haciendo que la precisión del filtro disminuya. Por ello, el filtro es una estructura de datos probabilística que se vuelve menos precisa a medida que se agregan más patrones. La precisión depende del número de los patrones agregados en relación con el tamaño del vector de bits (N) y el número de funciones hash (M). Un vector de bits más grande y con más funciones hash puede registrar más patrones con mayor precisión. Un vector de bits más pequeño o con menos funciones hash registrará menos patrones y producirá menos precisión.

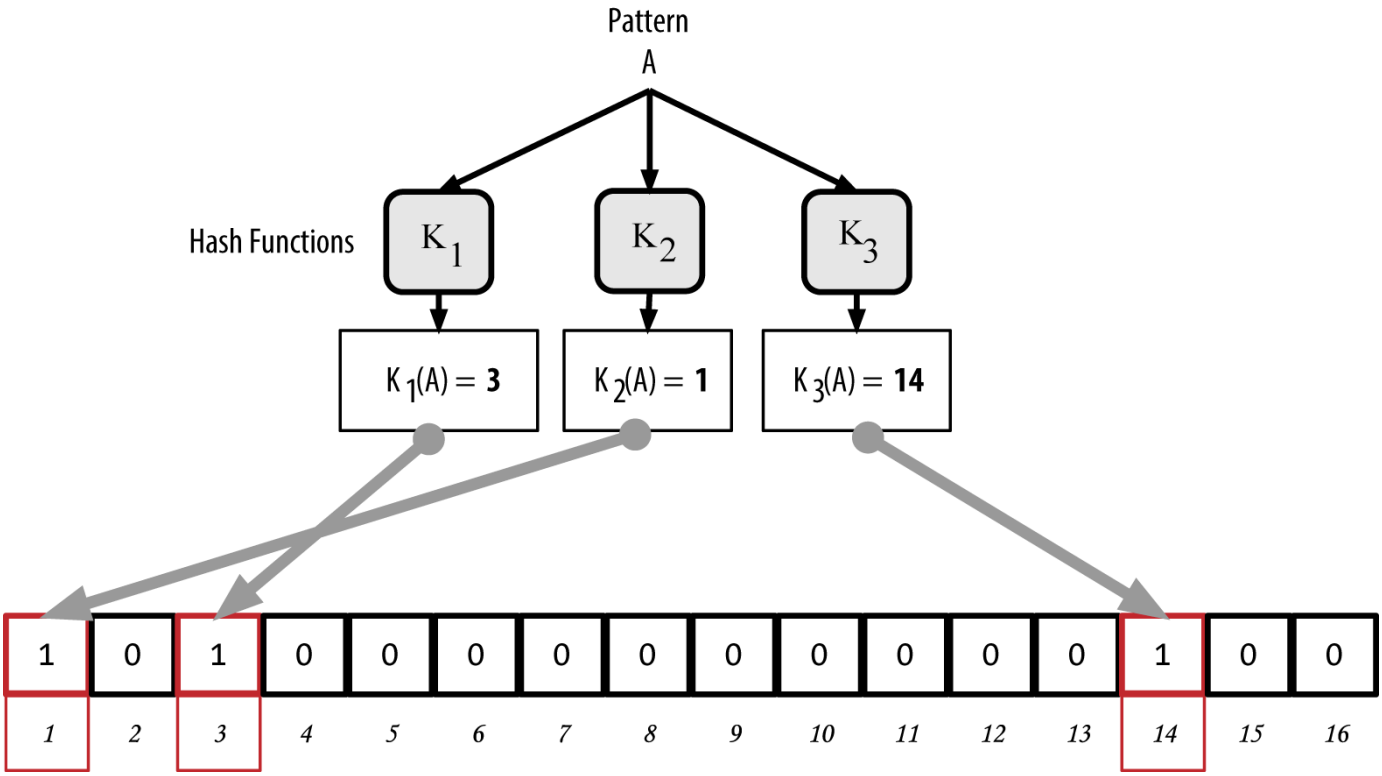


Figure 9. Añadiendo un patrón "A" a nuestro filtro bloom simple

Añadiendo un segundo patrón "B" a nuestro filtro bloom simple es un ejemplo que añade un segundo patrón "B" al filtro bloom simple.

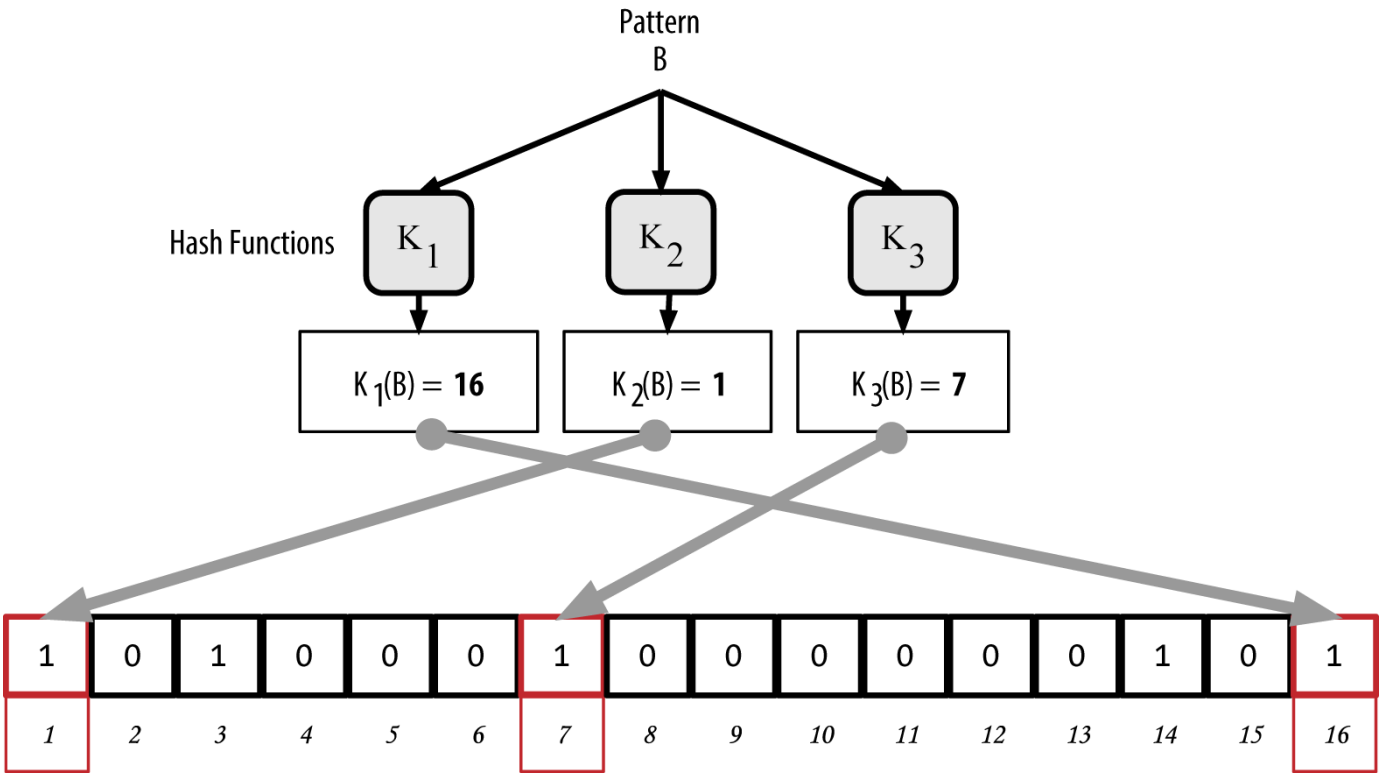


Figure 10. Añadiendo un segundo patrón "B" a nuestro filtro bloom simple

Para probar si un patrón es parte de un filtro bloom, se hace hash del patrón con cada una de las funciones hash, y el patrón de bits resultante se chequea contra el vector de bits. Si todos los bits indexados por las funciones hash se establecen en 1, entonces el patrón está *probablemente* registrado en el filtro de Bloom. Debido a que los bits se pueden establecer debido a la superposición originada por múltiples patrones, la respuesta no es irrefutable, sino que es probabilística. En términos simples, un resultado positivo de filtro bloom es un "Tal vez, Sí."

Probando la existencia del patrón "X" en el filtro bloom. El resultado es una coincidencia positiva probabilística, es decir, "Tal vez." es un ejemplo de pruebas de la existencia del patrón "X" en el filtro bloom simple. Los bits correspondientes se establecen en 1, por lo que el patrón es probablemente una coincidencia.

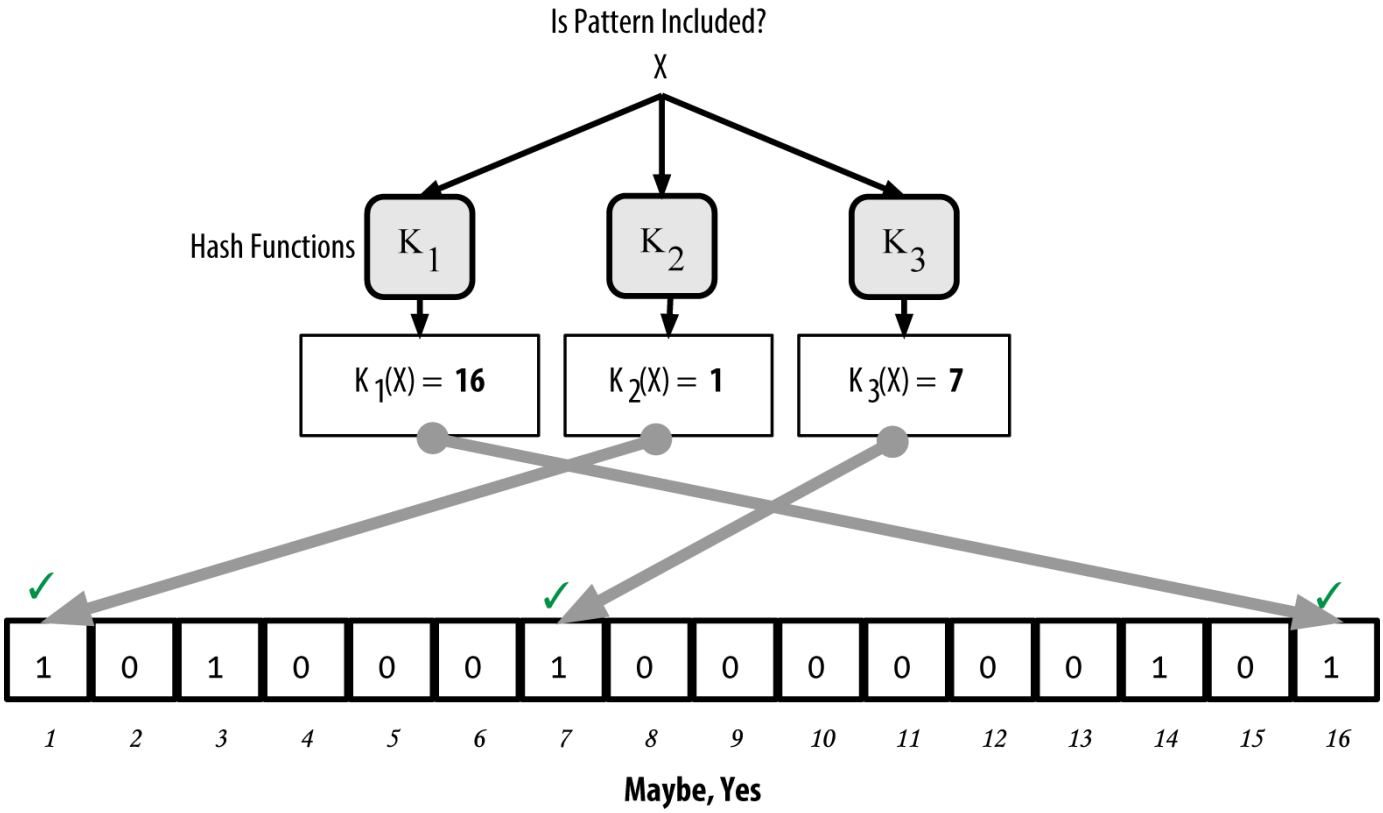


Figure 11. Probando la existencia del patrón "X" en el filtro bloom. El resultado es una coincidencia positiva probabilística, es decir, "Tal vez."

Por el contrario, si un patrón se prueba contra el filtro bloom y uno cualquiera de los bits se establece en 0, queda demostrado que el patrón no se registró en el filtro bloom. Un resultado negativo no es una probabilidad, es una certeza. En términos simples, un resultado negativo en un filtro bloom es un "¡Definitivamente No!"

Testeando la existencia del patrón "Y" en el filtro bloom. El resultado es una coincidencia negativa definitiva, que significa "¡Definitivamente No!" es un ejemplo para probar la existencia del patrón "Y" en el filtro bloom simple. Uno de los bits correspondientes se establece en 0, por lo que el patrón es definitivamente una no coincidencia.

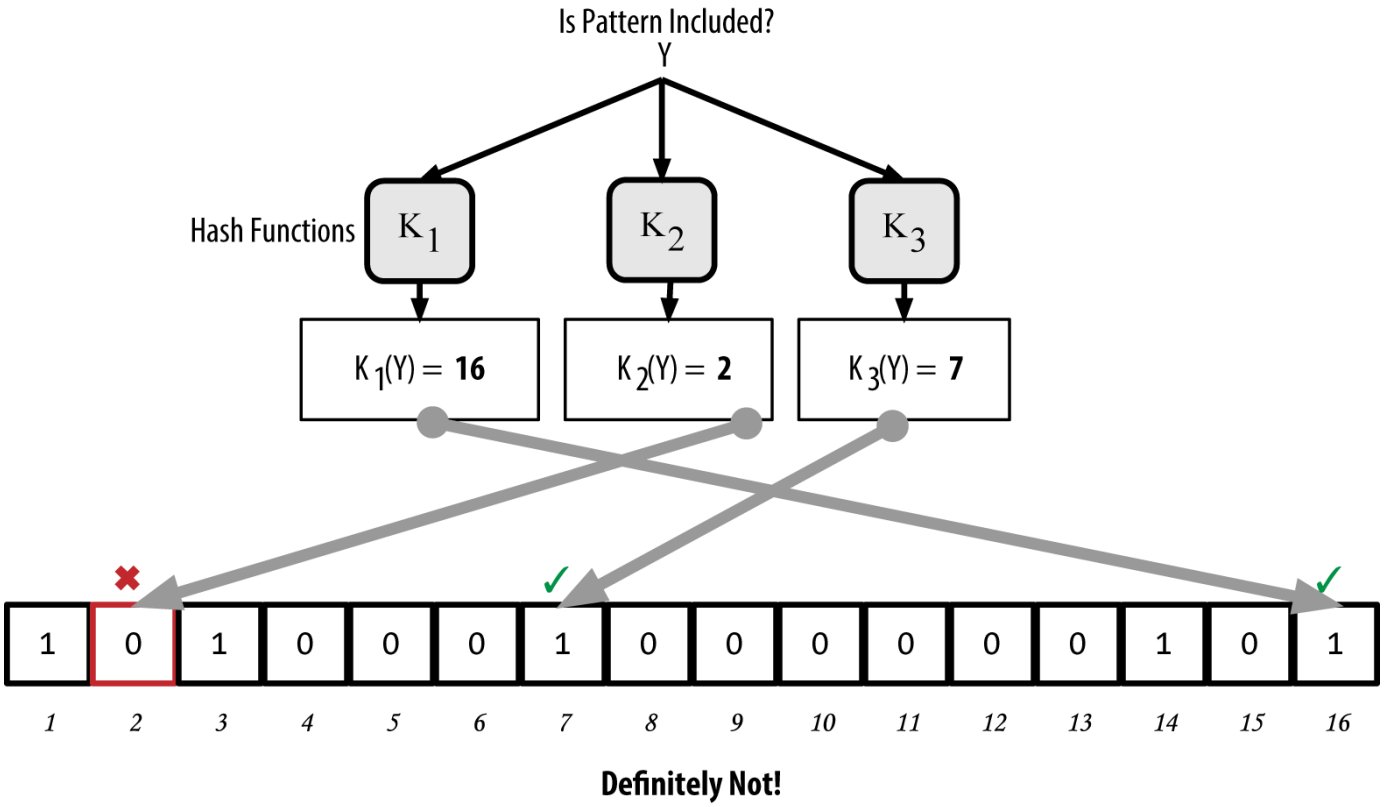


Figure 12. Testeando la existencia del patrón "Y" en el filtro bloom. El resultado es una coincidencia negativa definitiva, que significa "¡Definitivamente No!"

La aplicación de filtros bloom en Bitcoin se describe en la Propuesta de Mejora Bitcoin 37 (BIP0037). Consulte [appdxbitcoinimpproposals] o visite GitHub.

Filtros Bloom y Actualizaciones de Inventario

Los filtros bloom se utilizan para filtrar las transacciones (y bloques que las contienen) que un nodo SPV recibe de sus compañeros. Los nodos SPV crearán un filtro que haga coincidir solo las direcciones mantenidas en la cartera del nodo SPV. El nodo SPV enviará un mensaje filterload al compañero, que contiene el filtro bloom a usar en la conexión. Después de haber establecido un filtro, el compañero probará cada una de las salidas de transacción contra el filtro bloom. Solo las operaciones que coincidan con el filtro se envían al nodo.

En respuesta a un mensaje getdata desde el nodo, los compañeros enviarán un mensaje merkleblock que contiene solo las cabeceras de bloques para los bloques que coinciden con el filtro y una ruta merkle (ver [merkle_trees]) para cada transacción correspondiente. El compañero entonces también enviará mensajes tx que contienen las transacciones coincidentes por el filtro.

El nodo que crea el filtro bloom puede añadir patrones al filtro de forma interactiva mediante el envío de un mensaje filteradd. Para borrar el filtro bloom, el nodo puede enviar un mensaje filterclear. Debido a que no es posible eliminar un patrón de un filtro bloom, un nodo tiene que borrar y volver a enviar un nuevo filtro bloom si ya no se desea un patrón.

Reservas de Transacciones

Casi todos los nodos en la red bitcoin mantienen una lista temporal de las transacciones no confirmadas llamada *memory pool*, *mempool*, o *transaction pool*. Los nodos utilizan esta reserva (en inglés, "pool") para mantener un registro de las transacciones que son conocidas por la red, pero que aún no están incluidas en la cadena de bloques. Por ejemplo, un nodo que tiene la cartera de un usuario utilizará el pool de transacciones para rastrear los pagos entrantes a la cartera del usuario que se han recibido en la red, pero aún no han se confirmado.

Cuando se reciben y se verifican las transacciones, se añaden al pool de transacciones y se retransmiten a los nodos vecinos para que se propaguen en la red.

Algunas implementaciones de nodo también mantienen un pool separado de transacciones huérfanas. Si las entradas de una transacción se refieren a una transacción que aún no se conoce, tal como un padre que falta, la transacción huérfana será almacenada temporalmente en el pool huérfano hasta que llegue la transacción padre.

Cuando se añade una transacción al pool de transacciones, se comprueba el pool huérfano para cualquier huérfano que haga referencia a las salidas de esta transacción (sus hijos). Se validan los huérfanos que coincidan. Si es válido, se retira del pool huérfano y se añade al pool de transacciones, completando la cadena que comenzó con la transacción padre. A la luz de la transacción que se acaba de agregar, que ya no es huérfana, el proceso se repite recursivamente en busca de descendientes, hasta que no se encuentren más descendientes. A través de este proceso, la llegada de una transacción padre desencadena una reconstrucción en cascada de toda una cadena de transacciones interdependientes por volver a unir a los huérfanos con sus padres hasta el final de la cadena.

Tanto el pool de transacciones como el pool huérfano (en el caso de que esté implementado) se almacenan en la memoria local y no se guardan en almacenamiento persistente; más bien, se llenan dinámicamente de los mensajes entrantes de la red. Cuando se inicia un nodo, los dos pools están vacíos y son progresivamente ocupados con nuevas transacciones recibidas en la red.

Algunas implementaciones del cliente bitcoin también mantienen una base de datos UTXO o pool UTXO, que es el conjunto de todas las salidas no gastadas en la cadena de bloques. Aunque el nombre "pool UTXO" suena similar al pool de transacciones, representa un conjunto diferente de datos. A diferencia de los pools de transacciones y huérfanos, el pool UTXO no se inicializa vacío sino que contiene millones de salidas de transacción no gastadas, incluyendo algunas que datan de 2009. El pool UTXO puede guardarse en la memoria local o como una tabla de base de datos indexada en almacenamiento persistente.

El pool de transacciones y el pool huérfano representan la perspectiva local de un solo nodo y pueden variar significativamente de un nodo a otro, dependiendo de cuando se inicia o reinicia el nodo. Sin embargo, el pool UTXO representa el consenso emergente de la red y por lo tanto va a variar poco entre los nodos. Además, los pools de transacciones y huérfanos solo contienen transacciones no confirmadas, mientras que el pool UTXO solo contiene salidas confirmadas.

Mensajes de Alerta

Los mensajes de alerta son una funcionalidad que se usa en raras ocasiones, pero que sin embargo aplica a la mayoría de los nodos. Los mensajes de alerta son el "sistema de transmisión de emergencia" de bitcoin, un medio por el cual los desarrolladores del núcleo de bitcoin pueden enviar un mensaje de texto de emergencia a todos los nodos bitcoin. Esta funcionalidad permite que el equipo de desarrollo del núcleo pueda notificar a todos los usuarios de bitcoin de un grave problema en la red bitcoin, como un error crítico que requiera la intervención del usuario. El sistema de alerta solo se ha utilizado un puñado de veces, sobre todo a principios de 2013, cuando un error crítico de base de datos causó un fork multibloque en la cadena de bloques de bitcoin.

Los mensajes de alerta se propagan por el mensaje alert. El mensaje de alerta contiene varios campos, incluyendo:

ID

Un identificador de alerta de modo que las alertas duplicadas puede ser detectadas

Expiration

Un tiempo después del cual expira la alerta

RelayUntil

Un tiempo después del cual la alerta no debe ser transmitida

MinVer, MaxVer

El rango de versiones del protocolo bitcoin a los que se aplica esta alerta

subVer

La versión del software de cliente a la que se aplica esta alerta

Prioridad

Un nivel de prioridad de alerta, actualmente no se utiliza

Las alertas son criptográficamente firmadas por una clave pública. La clave privada correspondiente está en manos de unos pocos miembros selectos del equipo de desarrollo del núcleo. La firma digital asegura que no se propagarán alertas falsas en la red.

Cada nodo que reciba este mensaje de alerta debe verificarlo, comprobar el vencimiento y propagarlo a todos sus compañeros, garantizando así la propagación rápida en toda la red. Además de la propagación de la alerta, los nodos pueden implementar una función de interfaz de usuario para presentar la alerta al usuario.

En el cliente Bitcoin Core, la alerta está configurada con la opción de línea de comandos -alertnotify +, que especifica un comando a ejecutar cuando se recibe una alerta. El mensaje de alerta se pasa como un parámetro al comando +alertnotify. Por lo general, el comando alertnotify se establece para generar un mensaje de correo electrónico al administrador del nodo, que contiene el mensaje de alerta. La

alerta también se muestra como un cuadro de diálogo emergente en la interfaz gráfica de usuario (bitcoin-Qt) si se está ejecutando.

Otras implementaciones del protocolo bitcoin pueden manejar la alerta de diferentes maneras. Muchos sistemas de minería bitcoin con hardware embebido pueden no aplicar la función de mensajes de alerta porque no tienen interfaz de usuario. Se recomienda encarecidamente que los mineros que ejecutan estos sistemas de minería se suscriban a las alertas a través de un operador del pool de minería o ejecutando un nodo ligero solo para fines de alerta.

La Cadena de Bloques

Introducción

La estructura de datos de la cadena de bloques (en inglés, "blockchain") es una lista ordenada, enlazada hacia atrás en el tiempo, de bloques de transacciones. La cadena de bloques se puede almacenar como un archivo plano, o en una base de datos simple. El cliente Bitcoin Core almacena los metadatos de la cadena de bloques usando la base de datos LevelDB de Google. Los bloques están enlazados "hacia atrás en el tiempo", cada uno referenciando al bloque anterior de la cadena. La cadena de bloques a menudo se visualiza como una pila vertical, con los bloques en capas uno encima de otro, sirviendo el primer bloque como la base de la pila. La visualización de bloques apilados unos encima de otros resulta en el uso de términos como "altura" para referirse a la distancia desde el primer bloque, y "arriba" o "punta" para referirse al bloque añadido más recientemente.

Cada bloque dentro de la cadena de bloques se identifica mediante un hash, generado utilizando el algoritmo criptográfico hash SHA256 en la cabecera del bloque. Cada bloque también hace referencia a un bloque anterior, conocido como el bloque *padre*, a través del campo "hash de bloque anterior" en la cabecera del bloque. En otras palabras, cada bloque contiene el hash de su padre dentro de su propia cabecera. La secuencia de los hashes que unen cada bloque a su padre crea una cadena que se remonta hasta el final del primer bloque jamás creado, conocido como el bloque génesis.

Aunque un bloque solo tiene un padre, puede tener temporalmente varios hijos. Cada uno de los hijos tiene una referencia al mismo bloque, al que consideran su padre, y cada uno de los hijos contiene también el mismo hash (de padre) en el campo "hash de bloque anterior". Los hijos múltiples surgen durante una bifurcación (en inglés, "fork") de la cadena de bloques, una situación temporal que se produce cuando se descubren diferentes bloques casi simultáneamente por diferentes mineros (ver [forks]). Con el tiempo, un bloque hijo se convierte en parte de la cadena de bloques y la "bifurcación" se resuelve. A pesar de que un bloque puede tener más de un hijo, cada bloque solo puede tener un padre. Esto se debe a que un bloque tiene un solo campo "hash de bloque anterior" que hace referencia a su único padre.

El campo "hash de bloque anterior" está dentro de la cabecera del bloque y por lo tanto afecta al hash del bloque actual. La identidad propia del hijo cambia si la identidad de los padres cambia. Cuando el padre se modifica de alguna manera, los cambios de hash de los padres tambien cambian. Cuando el hash del padre cambia requiere un cambio en el puntero "hash de bloque anterior" del hijo. Esto a su vez hace que el hash del hijo cambie, lo que requiere un cambio en el puntero del nieto, que a su vez cambia el nieto, y así sucesivamente. Este efecto cascada asegura que, una vez que un bloque tiene muchas generaciones siguientes, no puede ser cambiado sin forzar un nuevo cálculo de todos los bloques siguientes. Debido a que un nuevo cálculo requeriría una computación enorme, la existencia de una larga cadena de bloques hace que la historia profunda de la cadena de bloques sea inmutable, que es una característica clave de la seguridad de bitcoin.

Una forma de pensar en la cadena de bloques es como capas de una formación geológica o como muestras del núcleo glaciar. Las capas superficiales pueden cambiar con las estaciones, o incluso ser

destruidas antes de que tengan tiempo para asentarse. Pero una vez que profundizas unas pocas pulgadas, las capas geológicas se vuelven más y más estables. Para cuando nos fijamos en unos pocos cientos de pies abajo, ya estaríamos buscando en una instantánea del pasado que ha permanecido inalterado durante millones de años. En la cadena de bloques, los bloques más recientes pueden ser revisados si hay un nuevo cálculo de la cadena debido a una bifurcación. Los seis bloques más altos son como unas cuantas pulgadas de tierra vegetal. Pero una vez que se mete más profundamente en la cadena de bloques, más allá de seis bloques, es cada vez menos probable que los bloques cambien. Después de retroceder 100 bloques, hay tanta estabilidad que la transacción coinbase —la transacción que contiene los bitcoins recién minados— ya puede ser gastada. Unos pocos miles de bloques hacia atrás (un mes) y la cadena de bloques es historia inmutable para todos los propósitos prácticos. Aunque el protocolo siempre permite que una cadena sea deshecha por una cadena más larga, y a pesar de que siempre existe la posibilidad de que cualquier bloque sea revertido, la probabilidad de un evento de ese tipo disminuye a medida que pasa el tiempo hasta que se convierte en infinitesimal.

Estructura de un Bloque

Un bloque es una estructura de datos contenedor que agrupa las transacciones para su inclusión en el libro de contabilidad público, la cadena de bloques. El bloque se compone de una cabecera, que contiene metadatos, seguido por una larga lista de operaciones que componen la mayor parte de su tamaño. La cabecera del bloque es de 80 bytes, mientras que la transacción promedio es de al menos 250 bytes y el bloque promedio contiene más de 500 transacciones. Un bloque completo, con todas las transacciones, por lo tanto, es 1000 veces más grande que la cabecera del bloque. La estructura de un bloque describe la estructura de un bloque.

Table 1. La estructura de un bloque

Tamaño	Campo	Descripción
4 bytes	Tamaño de Bloque	El tamaño del bloque en bytes después de este campo
80 bytes	Cabecera de Bloque	Varios campos componen la cabecera del bloque
1-9 bytes (VarInt)	Contador de Transacción	Cuántas transacciones hay a continuación
Variable	Transacciones	Las transacciones registradas en este bloque

Cabecera de Bloque

La cabecera del bloque se compone de tres conjuntos de metadatos de bloque. En primer lugar, hay una referencia a un hash del bloque anterior, que conecta este bloque al bloque anterior en la cadena de bloques. El segundo conjunto de metadatos, concretamente la *dificultad*, el *sello de tiempo* y el *nonce*, están relacionados con la competencia en la minería, como se detalla en [ch8]. La tercera pieza de metadatos es la raíz del árbol merkle, una estructura de datos utilizada para resumir de manera

eficiente todas las transacciones en el bloque. La estructura de la cabecera de bloque describe la estructura de una cabecera de bloque.

Table 2. La estructura de la cabecera de bloque

Tamaño	Campo	Descripción
4 bytes	Versión	Un número de versión para seguir las actualizaciones de software y protocolo
32 bytes	Hash del Bloque Anterior	Una referencia al hash del bloque anterior (padre) en la cadena
32 bytes	Raíz Merkle	Un hash de la raíz del árbol de merkle de las transacciones de este bloque
4 bytes	Sello de Tiempo	El tiempo de creación aproximada de este bloque (segundos desde Unix Epoch)
4 bytes	Objetivo de Dificultad	El objetivo de dificultad del algoritmo de prueba de trabajo para este bloque
4 bytes	Nonce	Un contador usado para el algoritmo de prueba de trabajo

El nonce, objetivo de dificultad y sello de tiempo son usados en el proceso de minado y serán analizados en mayor detalle en [ch8].

Identificadores de Bloque: Hash de Cabecera de Bloque y Altura de Bloque

El identificador primario de un bloque es su hash criptográfico, una huella digital, que se obtiene al hacer hash de la cabecera de bloque dos veces a través del algoritmo SHA256. El hash de 32 bytes resultante se llama hash de bloque pero es más preciso llamarlo hash de cabecera de bloque, <phrase role="keep-together">porque para calcularlo solo se utiliza la cabecera del bloque. Por ejemplo,</phrase> 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f es el hash de bloque del primer bloque bitcoin jamás creado. El hash de bloque identifica un bloque de forma única e inequívoca y se puede derivar de forma independiente por cualquier nodo simplemente haciendo hash de la cabecera del bloque.

Tenga en cuenta que el hash de bloque no está realmente incluido dentro de la estructura de datos del bloque, ni cuando el bloque es transmitido en la red, ni cuando se guarda en el almacenamiento persistente de un nodo como parte de la cadena de bloques. En cambio, cada nodo calcula el hash de

bloque cuando recibe el bloque de la red. El hash de bloque podría ser almacenado en una tabla separada de la base de datos como parte de los metadatos del bloque, para facilitar la indexación y hacer más rápida la recuperación de los bloques desde el disco.

Una segunda manera de identificar un bloque es por su posición en la cadena de bloques, denominada la <phrase role="keep-together"><emphasis>altura del bloque</emphasis>. El primer bloque jamás creado está a la altura del bloque 0 (cero) y es el</phrase> <phrase role="keep-together"> mismo bloque que se ha referenciado anteriormente con el siguiente hash de bloque</phrase> 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f. Así, un bloque se puede identificar de dos maneras: haciendo referencia al hash de bloque o haciendo referencia a la altura del bloque. Cada bloque posterior que se añade "encima" de ese primer bloque está en una posición "superior" en la cadena de bloques, como cajas apiladas una encima de la otra. El 1 de enero de 2014 la altura del bloque era 278000 aproximadamente, lo que significa que había 278000 bloques apilados en la parte superior del primer bloque creado en enero de 2009.

A diferencia del hash de bloque, la altura del bloque no es un identificador único. Aunque cada bloque siempre tendrá una altura de bloque específica e invariante, lo contrario no es cierto—la altura del bloque no siempre identifica a un solo bloque. Dos o más bloques que compiten por la misma posición en la cadena de bloques podrían tener la misma altura del bloque. Este escenario se discute en detalle en la sección [forks]. Además, la altura del bloque no forma parte de la estructura de datos del bloque; no se almacena dentro del bloque. Cada nodo identifica dinámicamente la posición de un bloque (altura) en la cadena de bloques cuando se recibe desde la red bitcoin. La altura del bloque también podría almacenarse como metadatos en una tabla indexada de base de datos para recuperarlo más rápidamente.

TIP

El *hash de bloque* de un bloque siempre identifica un bloque de forma única. Un bloque también tiene siempre una *altura del bloque* específica. Sin embargo, no siempre una altura del bloque concreta identifica a un único bloque. Más bien, dos o más bloques pueden competir por una misma posición en la cadena de bloques.

El Bloque Génesis

El primer bloque en la cadena de bloques se llama el bloque génesis y fue creado en 2009. Es el ancestro común de todos los bloques en la cadena de bloques, lo que significa que si comienza en cualquier bloque y sigue la cadena hacia atrás en el tiempo, finalmente llegará al bloque génesis.

Cada nodo siempre comienza con una cadena de bloques de al menos un bloque ya que el bloque génesis está codificado de forma estática en el software del cliente bitcoin, de forma que no pueda ser alterado. Cada nodo siempre "sabe" el hash y estructura del bloque génesis, la fecha fija en que fue creado, e incluso la única transacción contenida en él. Por lo tanto, cada nodo tiene el punto de partida para la cadena de bloques, una "raíz" segura desde la que construir una cadena de bloques de confianza.

Vea el bloque génesis codificado estáticamente dentro del cliente Bitcoin Core, en chainparams.cpp.

El siguiente identificador de hash pertenece al bloque génesis:

```
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Puede buscar ese hash de bloque en cualquier sitio web de explorador de bloques, como blockchain.info, y le llevará a una página que describe el contenido de este bloque, con una dirección URL que contiene ese hash:

https://blockchain.info/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

https://blockexplorer.com/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

Usando el cliente de referencia Bitcoin Core en la línea de comandos:

```
$ bitcoind getblock 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

```
{
    "hash" : "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
    "confirmations" : 308321,
    "size" : 285,
    "height" : 0,
    "version" : 1,
    "merkleroot" : "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
    "tx" : [
        "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
    ],
    "time" : 1231006505,
    "nonce" : 2083236893,
    "bits" : "1d00ffff",
    "difficulty" : 1.00000000,
    "nextblockhash" : "00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"
}
```

El bloque génesis contiene un mensaje oculto en su interior. La entrada de transacción coinbase contiene el texto "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks." (traducido al español: "The Times 03/Ene/2009 Canciller preparado para segundo rescate a los bancos."). Este mensaje tenía la intención de ofrecer la prueba de la fecha más antigua en la que este bloque fue creado, haciendo referencia al titular del periódico británico *The Times*. También sirve como un recordatorio irónico de la importancia de un sistema monetario independiente, precisamente cuando el lanzamiento de bitcoin coincide en el tiempo con una crisis monetaria mundial sin precedentes. El mensaje se registró en el primer bloque por Satoshi Nakamoto, creador de bitcoin.

Enlazando Bloques en la Cadena de Bloques

Los nodos completos de bitcoin mantienen una copia local de la cadena de bloques, comenzando en el bloque génesis. La copia local de la cadena de bloques se actualiza constantemente a medida que se encuentran y se utilizan nuevos bloques para extender la cadena. A medida que un nodo recibe bloques entrantes desde la red, validará estos bloques y luego los enlazará a la cadena de bloques existente. Para establecer un enlace, un nodo examinará la cabecera del bloque entrante buscando el "hash del bloque anterior."

Supongamos, por ejemplo, que un nodo tiene 277.314 bloques en la copia local de la cadena de bloques. El último bloque que el nodo conoce es el bloque 277.314, con un hash de cabecera de bloque de 00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249.

El nodo bitcoin recibe después un nuevo bloque de la red, que se analiza de la siguiente manera:

```
{
    "size" : 43560,
    "version" : 2,
    "previousblockhash" :
        "00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
    "merkleroot" :
        "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
    "time" : 1388185038,
    "difficulty" : 1180923195.25802612,
    "nonce" : 4215469401,
    "tx" : [
        "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",

 #[... muchas otras transacciones omitidas ...]

        "05cfd38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
    ]
}
```

Interpretando este nuevo bloque, el nodo encuentra el campo previousblockhash, que contiene el hash de su bloque padre. Es un hash que el nodo ya conocía, y que corresponde al último bloque en la cadena, a la altura de 277.314. Por lo tanto, este nuevo bloque es un hijo del último bloque de la cadena y extiende la cadena de bloques existente. El nodo añade este nuevo bloque al final de la cadena, añadiendo a la cadena de bloques una nueva altura de 277.315. Bloques enlazados en una cadena, por referencia al hash de la cabecera del bloque anterior muestra la cadena de tres bloques, enlazados por referencias en el campo previousblockhash.

Árboles Merkle

Cada bloque en la cadena de bloques bitcoin contiene un resumen de todas las transacciones en el bloque, utilizando un *árbol merkle*.

Un *árbol merkle*, también conocido como un árbol hash binario, es una estructura de datos que se usa para resumir y verificar de manera eficiente la integridad de grandes conjuntos de datos. Los árboles merkle son árboles binarios que contienen hashes criptográficos. El término "árbol" se usa en informática para describir una estructura de datos de ramificación, pero estos árboles por lo general aparecen al revés, con la "raíz" en la parte superior y las "hojas" en la parte inferior de un diagrama, como se verá en los ejemplos que siguen.

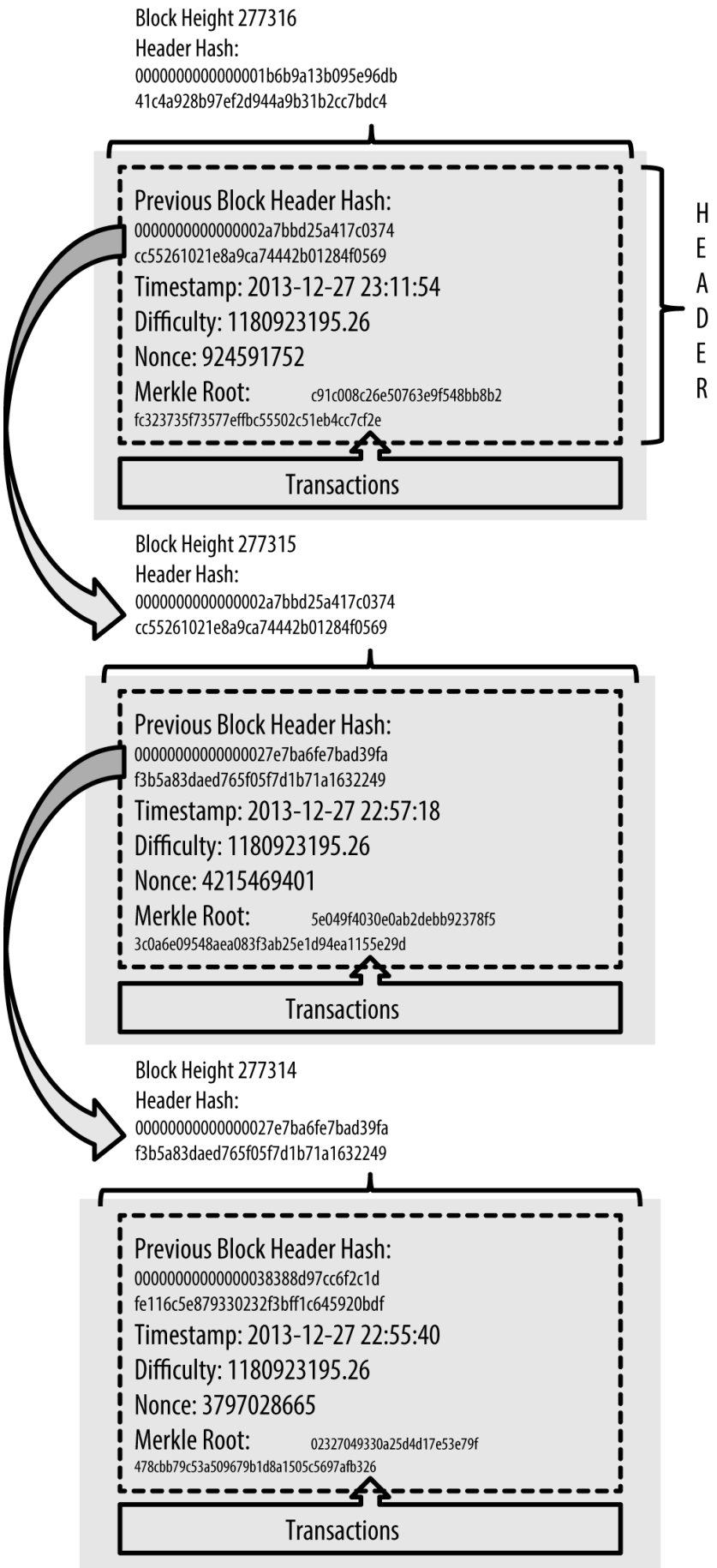

Block Height 277316
Header Hash:
0000000000000001b6b9a13b095e96db
41c4a928b97ef2d944a9b31b2cc7bdc4
Previous Block Header Hash:
0000000000000002a7bbd25a417c0374
cc55261021e8a9ca74442b01284f0569
Timestamp: 2013-12-27 23:11:54
Difficulty: 1180923195.26
Nonce: 924591752
Merkle Root: c91c008c26e50763e9f548bb8b2
fc323735f73577effbc55502c51eb4cc7cf2e
Transactions
HEADER
Block Height 277315
Header Hash:
0000000000000002a7bbd25a417c0374
cc55261021e8a9ca74442b01284f0569
Previous Block Header Hash:
00000000000000027e7ba6fe7bad39fa
f3b5a83daed765f05f7d1b71a1632249
Timestamp: 2013-12-27 22:57:18
Difficulty: 1180923195.26
Nonce: 4215469401
Merkle Root: 5e049f4030e0ab2debb92378f5
3c0a6e09548aea083f3ab25e1d94ea1155e29d
Transactions
Block Height 277314
Header Hash:
00000000000000027e7ba6fe7bad39fa
f3b5a83daed765f05f7d1b71a1632249
Previous Block Header Hash:
00000000000000038388d97cc6f2c1d
fe116c5e879330232f3bff1c645920bdf
Timestamp: 2013-12-27 22:55:40
Difficulty: 1180923195.26
Nonce: 3797028665
Merkle Root: 02327049330a25d4d17e53e79f
478cbb79c53a509679b1d8a1505c5697afb326
Transactions

Figure 1. Bloques enlazados en una cadena, por referencia al hash de la cabecera del bloque anterior

Los árboles merkle se usan en bitcoin para resumir todas las transacciones en un bloque, produciendo una huella digital completa de todo el conjunto de transacciones, proporcionando un proceso muy eficiente para verificar si una transacción está incluida en un bloque. Un árbol merkle se construye mediante la ejecución de una función de hash en pares de nodos de forma recursiva hasta que solo queda un único hash, al que se le llama *raíz* o *raíz merkle*. El algoritmo de hash criptográfico utilizado en los árboles merkle de bitcoin es SHA256 aplicado dos veces, también conocido como doble-SHA256.

Cuando se toman N elementos de datos, se hace hash de cada uno de ellos y se resumen en un árbol merkle, se puede comprobar si cualquier elemento de datos está incluido en el árbol con un máximo de 2*log~2~(N) cálculos, convirtiéndolo en una estructura de datos muy eficiente.

El árbol merkle se construye de abajo hacia arriba. En el siguiente ejemplo, comenzamos con cuatro transacciones, A, B, C y D, que forman la *hojas* del árbol Merkle, como se muestra en <<simple_merkle> >. Las transacciones no se almacenan en el árbol de Merkle; más bien, se hace hash de sus datos y el hash resultante se almacena en cada nodo hoja como H_A, H_B, H_C, y H_D:

```
H~A~ = SHA256(SHA256(Transacción A))
```

Después, los pares consecutivos de nodos hoja se resumen en un nodo padre, concatenando los dos hashes y haciendo hash de ese dato concatenado. Por ejemplo, para construir el nodo padre H_{AB}, los dos valores hash de 32 bytes de los hijos se concatenan para crear una cadena de 64 bytes. Se hace entonces un doble-hash de esa cadena para producir el hash del nodo padre:

```
H~AB~ = SHA256(SHA256(H~A~ + H~B~))
```

El proceso continúa hasta que solo hay un nodo en la parte superior, el nodo conocido como la raíz Merkle. Ese hash de 32 bytes se almacena en la cabecera del bloque y resume todos los datos de las cuatro transacciones.

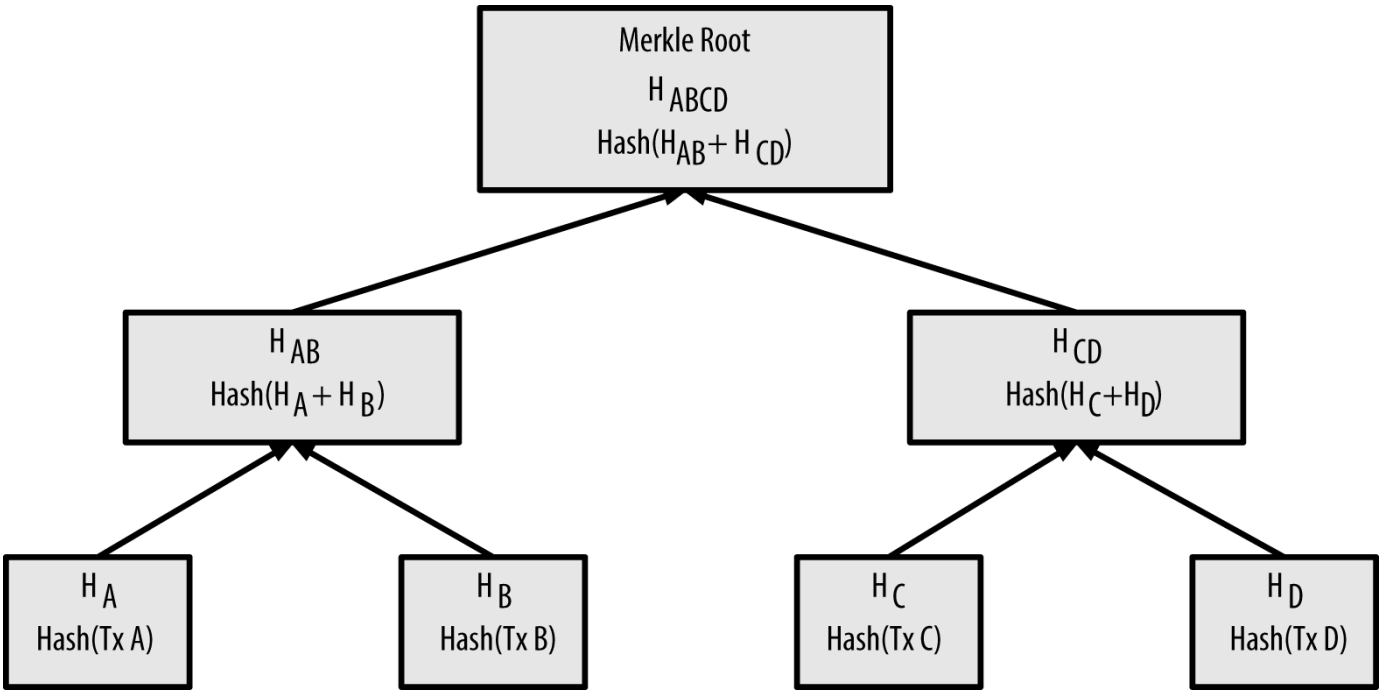


Figure 2. Calculando los nodos en un árbol merkle

Debido a que el árbol merkle es un árbol binario, se necesita un número par de nodos hoja. Si hay un número impar de transacciones para resumir, el último hash de transacción se duplicará para crear un número par de nodos hoja, también conocido como *árbol equilibrado*. Esto se muestra en <<merkle_tree_odd> >, donde se duplica la transacción C.

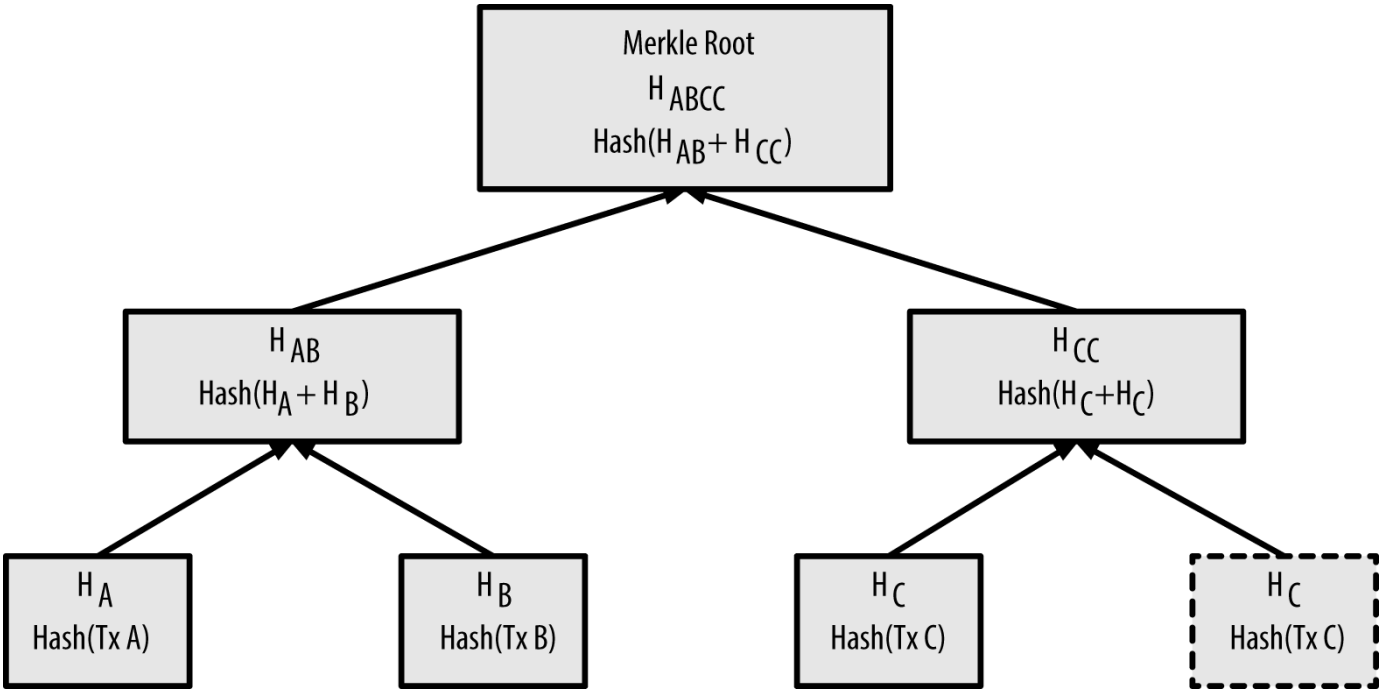


Figure 3. Duplicando un elemento de datos para alcanzar un número par de elementos de datos

El mismo método para la construcción de un árbol de cuatro transacciones se puede generalizar para construir árboles de cualquier tamaño. En bitcoin es común tener de varios cientos a más de mil transacciones en un solo bloque, que se resumen de la misma forma, produciendo solo 32 bytes de

datos en una única raíz merkle. En Un árbol merkle resumiendo muchos elementos de datos, verá un árbol construido a partir de 16 transacciones. Tenga en cuenta que, aunque la raíz se ve más grande que los nodos hoja en el diagrama, tiene exactamente el mismo tamaño, solo 32 bytes. Independientemente de si existe una transacción o cien mil transacciones en el bloque, la raíz merkle siempre los resume en 32 bytes.

Para demostrar que una transacción específica está incluida en un bloque, un nodo solo necesita producir log~2~(N) hashes de 32 bytes, elaborando una *ruta de autenticación* o *ruta merkle* que conecte la transacción específica a la raíz del árbol. Esto es especialmente importante a medida que el número de transacciones aumenta, porque el logaritmo en base-2 del número de transacciones aumenta mucho más lentamente. Esto permite que los nodos bitcoin produzcan eficientemente rutas de 10 ó 12 hashes (320-384 bytes), que pueden proporcionar la prueba de la existencia de una sola transacción entre más de mil transacciones en un bloque de un megabyte de tamaño.

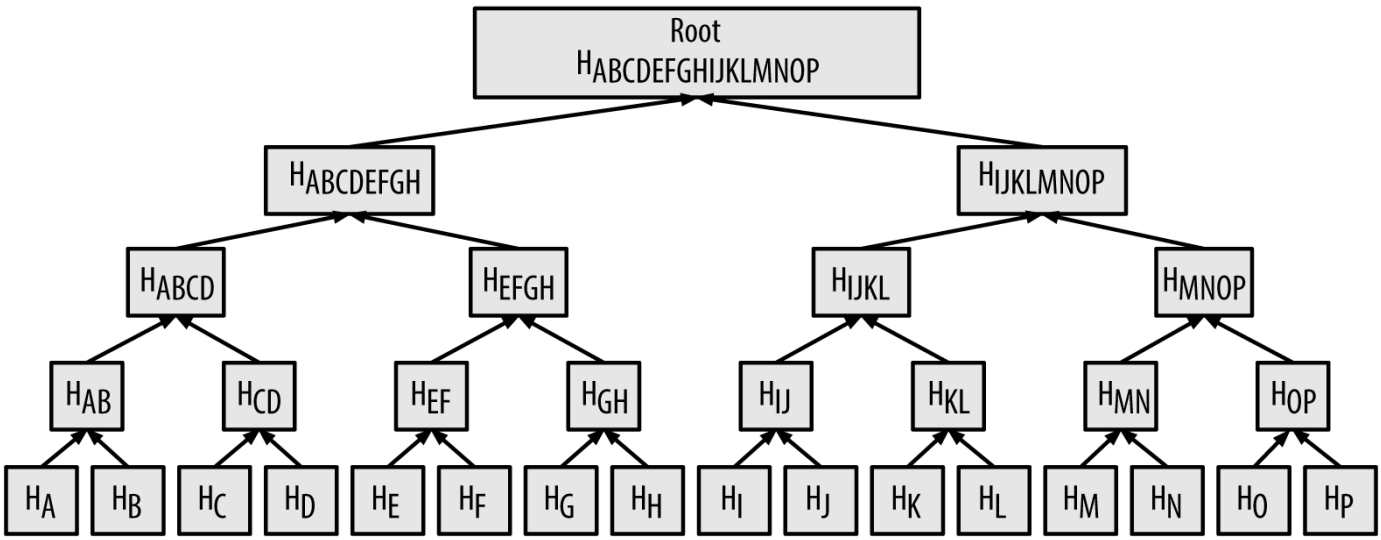


Figure 4. Un árbol merkle resumiendo muchos elementos de datos

En Una ruta merkle utilizada para probar la inclusión de un elemento de datos, un nodo puede demostrar que una transacción K está incluida en el bloque mediante la producción de una ruta merkle que ocupa solo cuatro hashes de 32-bytes de largo (128 bytes en total). La ruta consta de los cuatro valores hash (señalados en azul en Una ruta merkle utilizada para probar la inclusión de un elemento de datos) H_L, H_{IJ}, H_{MNOP} and $H_{ABCDEFGH}$. Con esos cuatro hashes suministrados a modo de ruta de autenticación, cualquier nodo puede demostrar que H_K (marcado en verde en el diagrama) está incluido en la raíz merkle mediante el cálculo de cuatro hashes adicionales por pares H_{KL}, H_{IJKL}, $H_{IJKLMNOP}$, y la raíz del árbol merkle (descrito en una línea de puntos en el diagrama).

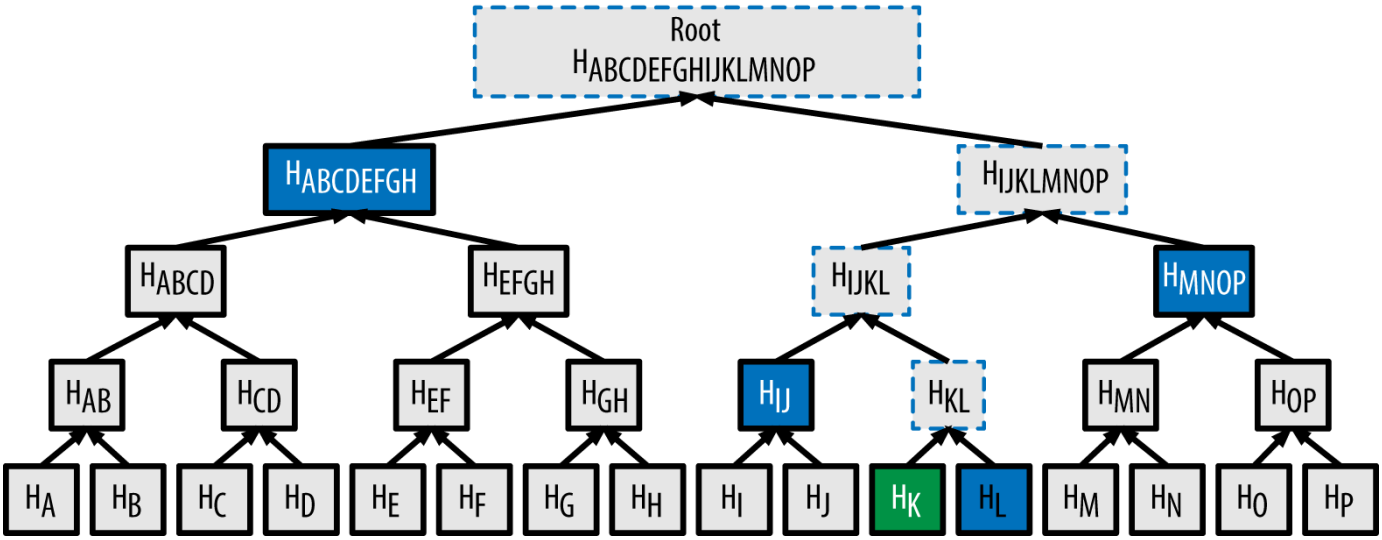


Figure 5. Una ruta merkle utilizada para probar la inclusión de un elemento de datos

El código en Construyendo un árbol merkle demuestra el proceso de crear un árbol merkle desde el hash del nodo hoja hasta la raíz, utilizando la biblioteca libbitcoin para algunas funciones auxiliares.

Example 1. Construyendo un árbol merkle

```
#include <bitcoin/bitcoin.hpp>

bc::hash_digest create_merkle(bc::hash_list& merkle)
{
    // Stop if hash list is empty.
    if (merkle.empty())
        return bc::null_hash;
    else if (merkle.size() == 1)
        return merkle[0];

    // While there is more than 1 hash in the list, keep looping...
    while (merkle.size() > 1)
    {
        // If number of hashes is odd, duplicate last hash in the list.
        if (merkle.size() % 2 != 0)
            merkle.push_back(merkle.back());
        // List size is now even.
        assert(merkle.size() % 2 == 0);

        // New hash list.
        bc::hash_list new_merkle;
        // Loop through hashes 2 at a time.
        for (auto it = merkle.begin(); it != merkle.end(); it += 2)
        {
            // Join both current hashes together (concatenate).
            bc::data_chunk concat_data(bc::hash_size * 2);
```

```
            auto concat = bc::make_serializer(concat_data.begin());
            concat.write_hash(*it);
            concat.write_hash(*(it + 1));
            assert(concat.iterator() == concat_data.end());
            // Hash both of the hashes.
            bc::hash_digest new_root = bc::bitcoin_hash(concat_data);
            // Add this to the new list.
            new_merkle.push_back(new_root);
        }
        // This is the new list.
        merkle = new_merkle;

        // DEBUG output -------------------------------------
        std::cout << "Current merkle hash list:" << std::endl;
        for (const auto& hash: merkle)
            std::cout << "  " << bc::encode_hex(hash) << std::endl;
        std::cout << std::endl;
        // --------------------------------------------------
    }
    // Finally we end up with a single item.
    return merkle[0];
}

int main()
{
    // Replace these hashes with ones from a block to reproduce the same merkle root.
    bc::hash_list tx_hashes{{

bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000000"),

bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000011"),

bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000022"),
    }};
    const bc::hash_digest merkle_root = create_merkle(tx_hashes);
    std::cout << "Result: " << bc::encode_hex(merkle_root) << std::endl;
    return 0;
}
```

Compilando y ejecutando el código de ejemplo merkle muestra el resultado de compilar y ejecutar el código merkle.

Example 2. Compilando y ejecutando el código de ejemplo merkle

```
$ # Compilar el código merkle.cpp
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Ejecutar el ejecutable merkle
$ ./merkle
Lista actual de hash merkle:
  32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27006
  30861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7540c4

Lista actual de hash merkle:
  d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

Result: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3
```

La eficiencia de los árboles merkle se hace evidente a medida que aumenta la escala. Eficiencia de un árbol merkle muestra la cantidad de datos que necesitan intercambiarse como una ruta merkle para demostrar que una transacción está incluida en un bloque.

Table 3. Eficiencia de un árbol merkle

Número de transacciones	Tamaño aprox. del bloque	tamaño de ruta (hashes)	Tamaño de ruta (bytes)
16 transacciones	4 kilobytes	4 hashes	128 bytes
512 transacciones	128 kilobytes	9 hashes	288 bytes
2048 transacciones	512 kilobytes	11 hashes	352 bytes
65.535 transacciones	16 megabytes	16 hashes	512 bytes

Como se puede ver en la tabla, mientras que el tamaño de bloque aumenta rápidamente, de 4 KB con 16 transacciones a un tamaño de bloque de 16 MB para incluir a 65.535 transacciones, la ruta merkle requerida para demostrar la inclusión de una transacción aumenta mucho más lentamente, de 128 bytes a solo 512 bytes. Con árboles merkle, un nodo puede descargar solo las cabeceras de bloque (80 bytes por bloque) y aún así ser capaz de identificar la inclusión de una transacción en un bloque mediante la recuperación de una ruta merkle pequeña de un nodo completo, sin almacenar o transmitir la gran mayoría de la cadena de bloques, que puede ser de varios gigabytes de tamaño. Los nodos que no mantienen una cadena de bloques completa, llamados de verificación de pago simplificada (nodos SPV), usan rutas merkle para verificar las transacciones sin necesidad de descargar bloques completos.

Árboles Merkle y Verificación de Pago Simplificada (SPV)

Los árboles merkle son ampliamente utilizados por los nodos SPV. Los nodos SPV no tienen todas las transacciones y no descargan bloques completos, solo las cabeceras de bloque. Con el fin de verificar que una transacción está incluida en un bloque sin tener que descargar todas las transacciones del bloque, utilizan una ruta de autenticación, o ruta merkle.

Consideremos, por ejemplo, un nodo SPV que esté interesado en los pagos entrantes a una dirección incluida en su cartera. El nodo SPV establecerá un filtro bloom en sus conexiones con sus compañeros para limitar las transacciones recibidas a solo aquellas que contengan direcciones de interés. Cuando un compañero vea una transacción que coincida con el filtro de bloom, enviará ese bloque usando un mensaje merkleblock. El mensaje merkleblock contiene la cabecera del bloque, así como una ruta merkle que vincula la transacción de interés con la raíz merkle en el bloque. El nodo SPV puede utilizar esta ruta merkle para conectar la transacción con el bloque y verificar que la transacción está incluida en el bloque. El nodo SPV también utiliza la cabecera del bloque para vincular el bloque con el resto de la cadena de bloques. La combinación de estos dos enlaces, entre la transacción y bloque, y entre el bloque y la cadena de bloques, prueba que la transacción está registrada en la cadena de bloques. Con todo, el nodo SPV habrá recibido menos de un kilobyte de datos para la cabecera del bloque y la ruta merkle, una cantidad de datos que es más de mil veces menor que un bloque completo (aproximadamente 1 megabyte actualmente).

Minería y Consenso

Introducción

La minería es el proceso por el cual se añaden nuevos bitcoin a la oferta de dinero. La minería también sirve para asegurar el sistema bitcoin contra transacciones fraudulentas o transacciones que gastan la misma cantidad de bitcoin más de una vez, conocido como un doble gasto. Los mineros proporcionan la potencia de procesamiento de la red bitcoin a cambio de la oportunidad de ser recompensados en bitcoin.

Los mineros validan nuevas transacciones y las graban en el libro contable global. Un nuevo bloque, que contiene las transacciones que tuvieron lugar desde el último bloque, se "extrae" cada 10 minutos de promedio, añadiendo esas transacciones a la cadena de bloques. Las transacciones que pasan a formar parte de un bloque y se agregan a la cadena de bloques se consideran "confirmadas", y permite a los nuevos propietarios de bitcoin gastar los bitcoin que recibieron en esas transacciones.

Los mineros reciben dos tipos de recompensas por la minería: nuevas monedas creadas con cada bloque nuevo, y las comisiones de transacción de todas las transacciones incluidas en el bloque (también llamadas tasas o tarifas de transacción). Para ganar este premio, los mineros compiten para resolver un problema matemático difícil basado en un algoritmo de hash criptográfico. La solución al problema, llamada la prueba de trabajo, se incluye en el nuevo bloque y actúa como prueba de que el minero gasta esfuerzo de computación significativa. La competencia para resolver el algoritmo de prueba de trabajo para ganar la recompensa y el derecho de registrar las transacciones en la cadena de bloques es la base del modelo de seguridad de bitcoin.

El proceso de generación de nueva moneda se llama minería porque la recompensa está diseñada para simular rendimientos decrecientes, al igual que la minería de metales preciosos. La oferta monetaria de bitcoin se crea a través de la minería, de forma similar a como un banco central crea dinero nuevo imprimiendo billetes. La cantidad de nuevo bitcoin que un minero puede añadir a un bloque disminuye aproximadamente cada cuatro años (o precisamente cada 210.000 bloques). Se comenzó con 50 bitcoin por bloque en enero de 2009 y se redujo a la mitad, a 25 bitcoin por bloque, en noviembre de 2012. Se reducirá a la mitad otra vez, a 12,5 bitcoin por bloque, en algún momento de 2016. Sobre la base de esta fórmula, las recompensas de la minería en bitcoin disminuyen exponencialmente hasta aproximadamente el año 2140, cuando se habrán emitido todos los bitcoin (20,99999998 millones). Después de 2140, no se emitirán nuevos bitcoins.

Los mineros de Bitcoin también ganan comisiones por transacciones. Cada transacción puede incluir una comisión de transacción, en la forma de un superávit de bitcoin entre entradas y salidas de la transacción. El minero de bitcoin ganador es el que se queda "con el cambio" en las transacciones incluidas en el bloque ganador. Hoy en día, las comisiones representan 0.5% o menos de los ingresos de un minero bitcoin, la gran mayoría procedentes de los bitcoins de nuevo cuño. Sin embargo, como la recompensa disminuye con el tiempo y el número de transacciones por bloque aumenta, una mayor proporción de los ingresos de la minería bitcoin provendrá de las comisiones. Después de 2140, todos los ingresos de los mineros bitcoin serán en forma de comisiones por transacción.

La palabra "minería" es algo engañosa. Al evocar la extracción de metales preciosos, se centra la atención en la recompensa para la minería de los nuevos bitcoins en cada bloque. Aunque la minería está incentivada por esta recompensa, el propósito principal de la minería no es la recompensa o la generación de nuevas monedas. Si ve la minería solo como el proceso mediante el cual se crean las monedas, está confundiendo los medios (incentivos) con el objetivo del proceso. La minería es el proceso principal de la cámara de compensación descentralizada, por el cual las transacciones son validadas y compensadas. La minería asegura el sistema bitcoin y permite la aparición de un consenso en toda la red sin una autoridad central.

La minería es la invención que hace especial a bitcoin, un mecanismo de seguridad descentralizado que es la base del dinero digital peer-to-peer. La recompensa de las monedas recién acuñadas y las comisiones de transacción es un plan de incentivos que alinea las acciones de los mineros con la seguridad de la red, mientras al mismo tiempo aplica la oferta monetaria.

En este capítulo, vamos a examinar primero la minería como un mecanismo de oferta monetaria y luego veremos la función más importante de la minería: el mecanismo de consenso emergente descentralizado en el que se basa la seguridad de bitcoin.

Economía Bitcoin y Creación de Moneda

Los bitcoins son "acuñados" durante la creación de cada bloque a un ritmo fijo y decreciente. Cada bloque, generado en promedio cada 10 minutos, contiene bitcoins totalmente nuevos, creados de la nada. Cada 210.000 bloques, o aproximadamente cada cuatro años, la velocidad de emisión de moneda se reduce en un 50%. Durante los primeros cuatro años de funcionamiento de la red, cada bloque contenía 50 nuevos bitcoins.

En noviembre de 2012, la nueva velocidad de emisión de bitcoin se redujo a 25 bitcoins por bloque y se reducirá de nuevo a 12.5 bitcoins en el bloque 420.000, que se extraerá en algún momento de 2016. La velocidad de creación de nuevas monedas disminuirá exponencialmente a través de más de 64 "reducciones a la mitad" hasta el bloque 13.230.000 (que se extraerá aproximadamente en el año 2137), cuando alcance la unidad monetaria mínima de 1 satoshi. Finalmente, después de 13,44 millones de bloques, en aproximadamente 2140, se habrán emitido casi 2.099.999.997.690.000 satoshis, o casi 21 millones de bitcoins. A partir de entonces, los bloques no contendrán nuevos bitcoins, y los mineros serán recompensados únicamente a través de las comisiones de transacción.La oferta de moneda bitcoin a lo largo del tiempo se basa en una velocidad de emisión geométricamente decreciente muestra el total de bitcoins en circulación a lo largo del tiempo, a medida que la emisión de moneda disminuye.

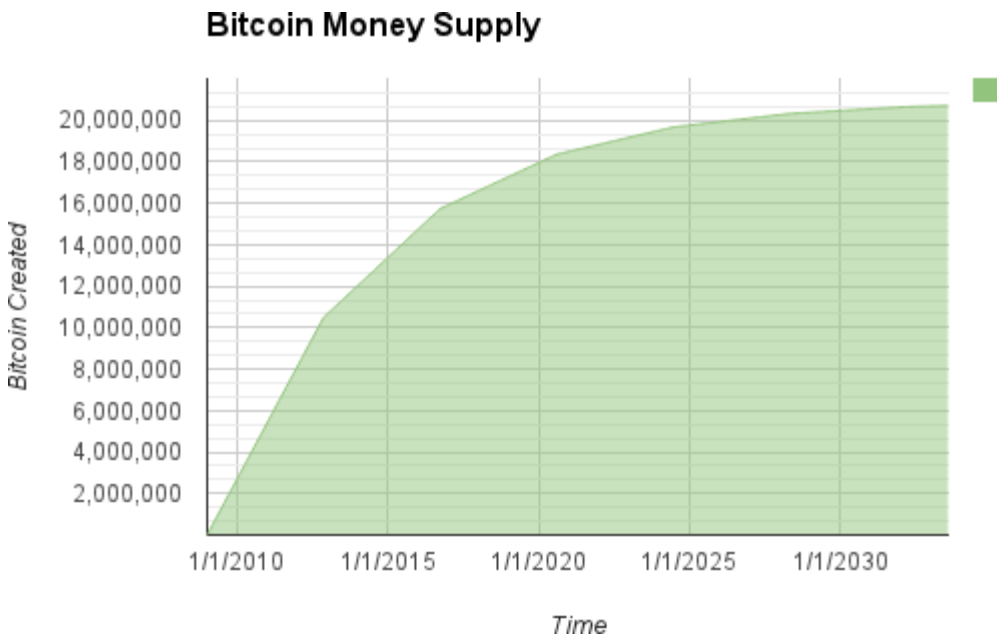


Figure 1. La oferta de moneda bitcoin a lo largo del tiempo se basa en una velocidad de emisión geométricamente decreciente

> El número máximo de monedas minadas es el *limite superior* de posibles recompensas mineras para bitcoin. En la práctica, un minero puede explotar intencionadamente un bloque para tomar menos de la recompensa completa. Dichos bloques ya han sido extraídos y más pueden ser extraído en el futuro, lo que resulta en una emisión total más baja de la moneda.

En el código de ejemplo Un script para calcular cuántos bitcoins serán emitidos en total calculamos el número total de bitcoins que serán emitidos.

Example 1. Un script para calcular cuántos bitcoins serán emitidos en total

```
# Original block reward for miners was 50 BTC
start_block_reward = 50
# 210000 is around every 4 years with a 10 minute block interval
reward_interval = 210000

def max_money():
    # 50 BTC = 50 0000 0000 Satoshis
    current_reward = 50 * 10**8
    total = 0
    while current_reward > 0:
        total += reward_interval * current_reward
        current_reward /= 2
    return total

print "Total BTC to ever be created:", max_money(), "Satoshis"
```

Ejecutando el script max_money.py muestra la salida producida al ejecutar el script.

Example 2. Ejecutando el script max_money.py

```
$ python max_money.py
BTC total que nunca se creará: 2099999997690000 Satoshis
```

La emisión finita y decreciente crea una oferta monetaria que resiste a la inflación. A diferencia de las monedas fiduciarias, las cuales pueden ser impresas hasta el infinito por bancos centrales, bitcoin no puede ser inflado con impresión.

Dinero Deflacionario

La más importante y debatida consecuencia de una emisión monetaria fija y decreciente es que la moneda tenderá a ser inherentemente *deflacionaria*. La deflación es el fenómeno de apreciación de valor debido a la discordancia entre oferta y demanda que hace que el valor (y el tipo de cambio) de una moneda suban. Siendo lo opuesto a la inflación, la deflación significa que el dinero adquiere mayor poder de compra con el tiempo.

Muchos economistas sostienen que la economía deflacionaria es un desastre y debe ser evitada a toda costa. Eso se debe a que en un período de deflación acelerada la gente tiende a acaparar dinero en vez de gastarlo, esperando que los precios caerán. Tal fenómeno se desató durante la "Década Perdida" de Japón, durante la cual un colapso completo de la demanda empujó a la moneda hacia una espiral deflacionaria.

Los expertos de bitcoin sostienen que la deflación no es mala por sí misma. En cambio, la deflación es asociada con el colapso de la demanda ya que ese es el único tipo de ejemplo de deflación que tenemos para estudiar. En una moneda fiduciaria con la posibilidad de impresión ilimitada es muy difícil entrar en una espiral deflacionaria a menos que ocurra un colapso completo en la demanda sumada a un rechazo a imprimir moneda. La deflación en bitcoin no es causada por un colapso en la demanda, sino por una oferta predecible y restringida.

En la práctica, se ha hecho evidente que el instinto de acaparamiento causado por una moneda deflacionaria puede superarse mediante descuentos de los proveedores, hasta que el descuento venza al instinto de acaparamiento del comprador. Debido a que el vendedor también está motivado para acaparar, el descuento se convierte en el precio de equilibrio en el que se comparan los dos instintos de acaparamiento. Con descuentos del 30% sobre el precio de bitcoin, la mayoría de los minoristas de bitcoin no están experimentando dificultades para superar el instinto de acaparamiento y la generación de ingresos. Queda por ver si el aspecto deflacionario de la moneda es realmente un problema cuando no es impulsado por una rápida retracción económica.

Concenso Descentralizado

En el capítulo anterior vimos la cadena de bloques, el libro contable global (lista) de todas las transacciones, que todo el mundo en la red bitcoin acepta como el registro de autoridad de la propiedad.

Pero, ¿cómo puede todo el mundo en la red estar de acuerdo sobre una única "verdad" universal, sobre quién es dueño de qué, sin tener que confiar en nadie? Todos los sistemas de pago tradicionales están basados en un modelo de confianza en el que una autoridad central proporciona un servicio de cámara de compensación, básicamente, verificando y compensando todas las transacciones. Bitcoin no tiene autoridad central, pero de alguna manera cada nodo completo tiene una copia completa de un libro de contabilidad público en el que se puede confiar como registro de autoridad. La cadena de bloques no está creada por una autoridad central, pero se monta de forma independiente por cada

nodo de la red. De alguna manera, cada nodo de la red, que actúa sobre la información transmitida a través de conexiones de red inseguras, puede llegar a la misma conclusión y montar una copia del mismo libro de contabilidad público que los demás. En este capítulo se examina el proceso por el cual la red bitcoin logra un consenso global sin autoridad central.

La invención principal de Satoshi Nakamoto es el mecanismo descentralizado para un *consenso emergente*. Emergente, porque el consenso no se logra de forma explícita, no hay elección o momento fijo cuando se produce el consenso. En cambio, el consenso es un artefacto emergente de la interacción asíncrona de miles de nodos independientes, todos siguiendo reglas simples. Todas las propiedades de bitcoin se derivan de esta invención, incluyendo moneda, transacciones, pagos, y el modelo de seguridad que no depende de una autoridad central o de la confianza, .

El consenso descentralizado de bitcoin emerge de la interacción de cuatro procesos que ocurren de forma independiente en los nodos de la red:

- La verificación independiente de cada transacción, por cada nodo completo, basado en una amplia lista de criterios
- La incorporación independiente de esas transacciones en nuevos bloques por nodos de minería, junto con la computación demostrada a través de un algoritmo de prueba de trabajo
- La verificación independiente de los nuevos bloques por cada nodo y el montaje en una cadena
- La selección independiente, por cada nodo, de la cadena con la mayor computación demostrada a través de prueba de trabajo

En las próximas secciones examinaremos estos procesos y cómo interactúan para crear la propiedad emergente de consenso de toda la red que permite a cualquier nodo bitcoin montar su propia copia de autoridad, confiable, pública, del libro contable global.

Verificación Independiente de Transacciones

En [transactions], vimos cómo el software de cartera crea transacciones mediante la recopilación de UTXO, proporcionando los scripts de desbloqueo apropiados, y construyendo después nuevas salidas asignadas a un nuevo propietario. La transacción resultante se envía entonces a los nodos vecinos en la red bitcoin de manera que se pueda propagar a través de toda la red bitcoin.

Sin embargo, antes de remitir las transacciones a sus vecinos, cada nodo bitcoin que recibe una transacción primero verifica la transacción. Esto garantiza que sólo las transacciones válidas se propaguen a través de la red, mientras que las transacciones no válidas se descartan en el primer nodo que las encuentra.

Cada nodo verifica cada transacción a través de una larga lista de criterios:

- La sintaxis de la transacción y su estructura de datos deben ser correctos.
- Ni las listas de entradas ni de salidas estén vacías.
- El tamaño de la transacción en bytes es inferior a MAX_BLOCK_SIZE.

- Cada valor de salida, así como el total, deben estar dentro del rango permitido de valores (menos de 21m monedas, más de 0).
- Ninguna de las entradas tiene de hash=0, N=-1 (transacciones coinbase no deben ser transmitidas).
- nLockTime es menor o igual a INT_MAX.
- El tamaño de la transacción en bytes es mayor o igual a 100.
- El número de operaciones de firma contenidas en la transacción es menor que el límite de operación de firma.
- El script de desbloqueo (scriptSig) solo puede empujar números en la pila, y el script de bloqueo (scriptPubkey) debe respetar los formatos isStandard (esto rechaza transacciones "no estándar").
- Una transacción coincidente en el pool, o en un bloque en la rama principal, debe existir.
- Para cada entrada, si existe la salida de referencia en cualquier otra transacción del pool, la transacción debe ser rechazada.
- Para cada entrada, busque en la rama principal y en el pool de transacciones para encontrar la transacción de salida de referencia. Si la transacción de salida no se encuentra para cualquier entrada, esta será una transacción huérfana. Añadir al pool de transacciones huérfanas, si una transacción coincidente no está ya en el pool.
- Para cada entrada, si la transacción de salida de referencia es una salida coinbase, debe tener por lo menos COINBASE_MATURITY (100) confirmaciones.
- Para cada entrada, debe existir la salida de referencia y ya no puede ser gastada.
- Usando las transacciones de salida de referencia para obtener los valores de entrada, compruebe que cada valor de entrada, así como la suma, están en el rango permitido de valores (menos de 21m monedas, más de 0).
- Rechazar si la suma de los valores de entrada es inferior a la suma de los valores de salida.
- Rechazar si la tarifa de transacción sería demasiado baja para entrar en un bloque vacío.
- Los scripts de desbloqueo para cada entrada se deben validar contra los scripts de bloqueo de salida correspondientes.

Estas condiciones pueden verse en detalle en las funciones AcceptToMemoryPool, CheckTransaction, y CheckInputs en el cliente de referencia bitcoin. Tenga en cuenta que las condiciones cambian con el tiempo, para hacer frente a los nuevos tipos de ataques de denegación de servicio o, a veces se relajan las normas a fin de incluir más tipos de transacciones.

Al verificar de forma independiente cada transacción, en el momento en que se recibe y antes de propagarlo, cada nodo construye un conjunto de transacciones válidas (pero sin confirmar) conocido como el *pool de transacciones*, *pool de memoria* o *mempool*.

Nodos de Minería

Algunos de los nodos de la red Bitcoin son nodos especializados llamados *mineros*. En

[ch01_intro_what_is_bitcoin] presentamos a Jing, un estudiante de ingeniería informática en Shanghai, China, que es un minero bitcoin. Jing gana bitcoin ejecutando una "plataforma minera", que es un sistema informático de hardware especializado diseñado para minar bitcoins. El hardware minero especializado de Jing se conecta a un servidor que ejecuta un nodo bitcoin completo. A diferencia de Jing, algunos mineros minan sin un nodo completo, como veremos en Pools de Minería. Como cualquier otro nodo completo, el nodo de Jing recibe y propaga transacciones sin confirmar en la red bitcoin. El nodo de Jing, sin embargo, también agrega estas transacciones en nuevos bloques.

El nodo de Jing está a la escucha de nuevos bloques, propagados en la red bitcoin, al igual que hacen todos los nodos. Sin embargo, la llegada de un nuevo bloque tiene un significado especial para un nodo de minería. La competencia entre los mineros termina efectivamente con la propagación de un nuevo bloque que actúa como un anuncio del ganador. Para los mineros, recibir un nuevo bloque significa que otra persona ganó la competición y que ellos perdieron. Sin embargo, el final de una ronda de la competición marca también el comienzo de la siguiente ronda. El nuevo bloque no es sólo una bandera a cuadros, que marca el final de la carrera; también es el pistoletazo de salida en la carrera por el siguiente bloque.

Agregando Transacciones en los Bloques

Después de validar transacciones, un nodo bitcoin las añadirá al *pool de memoria*, o *pool de transacciones*, donde las transacciones quedan esperando hasta que puedan ser incluidas (minadas) en un bloque. El nodo de Jing recoge, valida, y retransmite nuevas transacciones al igual que cualquier otro nodo. Sin embargo, a diferencia de otros nodos, el nodo de Jing agregará estas transacciones en un *bloque candidato*.

Sigamos los bloques que se crearon durante el tiempo en que Alice compró una taza de café de Bob Cafe (ver [cup_of_coffee]). La transacción de Alice se incluyó en el bloque 277.316. Con el fin de demostrar los conceptos de este capítulo, vamos a suponer que el bloque fue minado por el sistema de minería de Jing y seguiremos la transacción de Alice, hasta que pasa a formar parte de este nuevo bloque.

El nodo de minería de Jing mantiene una copia local de la cadena de bloques, la lista de todos los bloques creados desde el comienzo del sistema bitcoin en 2009. Para cuando Alice compra la taza de café, el nodo de Jing ha montado una cadena hasta el bloque 277.314. El nodo de Jing está a la escucha de transacciones, tratando de explotar un nuevo bloque, y también a la escucha de bloques descubiertos por otros nodos. Mientras el nodo de Jing está minando, recibe el bloque 277.315 a través de la red Bitcoin. La llegada de este bloque significa el final de la competición para el bloque 277.315 y el comienzo de la competición para crear el bloque 277.316 .

Durante los 10 minutos anteriores, mientras que el nodo de Jing estaba buscando una solución para el bloque 277.315, estaba al mismo tiempo recogiendo las transacciones en preparación para el siguiente bloque. Para entonces habrá recogido unos pocos cientos de transacciones en el pool de memoria. Al recibir el bloque 277.315 y validarlo, el nodo de Jing también comprobará todas las transacciones en el pool de memoria y retirará las que se hayan incluido en el bloque 277.315. Las transacciones que aún permanezcan en el pool de memoria seguirán sin confirmar y estarán esperando a ser registradas en

un nuevo bloque.

El nodo de Jing construye de inmediato un nuevo bloque vacío, un candidato para el bloque 277.316. Este bloque se denomina bloque candidato porque aún no es un bloque válido, ya que no contiene una prueba de trabajo válida. El bloque se vuelve válido sólo si el minero tiene éxito en la búsqueda de una solución para el algoritmo de prueba de trabajo.

Edad de Transacción, Comisiones, y Prioridad

Para construir el bloque candidato, el nodo bitcoin de Jing selecciona las transacciones del pool de memoria mediante la aplicación de una métrica de prioridad a cada transacción , agregando en primer lugar las transacciones de mayor prioridad. Las transacciones se priorizan en base a la "edad" de la UTXO que se gasta en las entradas, lo que permite que se dé preferencia a las entradas más antiguas y de alto valor frente a las entradas más nuevas y de menor valor. Las transacciones más prioritarias se pueden enviar sin comisiones, si hay suficiente espacio en el bloque.

La prioridad de una transacción se calcula como la suma del valor y la edad de las entradas dividido por el tamaño total de la transacción:

```
Prioridad = Sum (Valor de la entrada * Edad de la entrada) / Tamaño de Transacción
```

En esta ecuación, el valor de una entrada se mide en la unidad base, satoshis (1/100millonésima de un bitcoin). La edad de un UTXO es el número de bloques que han transcurrido desde que la UTXO se registró en la cadena de bloques, tomando como medida a cuántos bloques de "profundidad" está en la cadena de bloques. El tamaño de la transacción se mide en bytes.

Para que una transacción se considere de "alta prioridad", su prioridad debe ser superior a 57,600.000, lo que corresponde a un bitcoin (100 millones de satoshis), con una edad de un día (144 bloques), en una transacción de 250 bytes de tamaño en total:

```
Alta Prioridad > 100,000.000 satoshis * 144 bloques / 250 bytes = 57,600.000
```

Los primeros 50 kilobytes de espacio de transacción en un bloque se reservan para las transacciones de alta prioridad. El nodo de Jing llenará los primeros 50 kilobytes, dando prioridad a las transacciones de mayor prioridad en primer lugar, independientemente de la comisión. Esto permite que las transacciones de alta prioridad puedan ser procesadas, incluso si llevan cero comisiones.

El nodo de minería de Jing entonces llena el resto del bloque hasta el tamaño de bloque máximo (MAX_BLOCK_SIZE en el código), con transacciones que llevan al menos la comisión mínima, dando prioridad a las que tienen la comisión más alta por kilobyte de transacción.

Si hay algún espacio restante en el bloque, el nodo de minería de Jing podría elegir llenarlo con las transacciones sin comisiones. Algunos mineros pueden esforzarse por añadir transacciones sin comisiones. Otros mineros pueden optar por ignorar las transacciones sin comisiones.

Cualquier transacción que quede en el pool de memoria, después de haberse llenado el bloque, permanecerá en el pool para su inclusión en el siguiente bloque. A medida que las transacciones permanecen en el pool de memoria, sus entradas se hacen cada vez más antiguas, ya que la UTXO gastada va obteniendo más profundidad en la cadena de bloques con los nuevos bloques adicionales que van añadiéndose en la parte superior. Debido a que la prioridad de una transacción depende de la edad de sus entradas, las transacciones que quedan en el pool envejecerán y por lo tanto aumentarán de prioridad. Con el tiempo, una transacción sin comisiones podría alcanzar la prioridad suficiente para ser incluida en el bloque de forma gratuita.

Las transacciones bitcoin no tienen fecha de caducidad. Una transacción que es válida ahora será válida para siempre. Sin embargo, si una transacción se propaga a través de la red una sola vez, persistirá solo mientras se mantenga en el pool de memoria de algún nodo de minería. Cuando se reinicia un nodo de minería, su pool de memoria se vacía, porque se trata de almacenamiento no persistente. Aunque una transacción válida se haya propagado a través de la red, si no se ejecuta, puede que eventualmente deje de residir en el pool de memoria de cualquier minero. El software de cartera debería retransmitir este tipo de transacciones o reconstruirlas con comisiones más altas si no se ejecutan con éxito dentro de un plazo razonable de tiempo.

Cuando el nodo de Jing agrega todas las transacciones del pool de memoria, el nuevo bloque candidato tiene 418 operaciones con un total en comisiones de transacción de 0.09094928 bitcoin. Puede ver este bloque en la cadena de bloques utilizando la interfaz de línea de comandos del cliente Bitcoin Core, como se muestra en Bloque 277.316.

```
$ bitcoin-cli getblockhash
277316000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4

$ bitcoin-cli getblock
000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

Example 3. Bloque 277.316

```
{
    "hash" : "000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
    "confirmations" : 35561,
    "size" : 218629,
    "height" : 277316,
    "version" : 2,
    "merkleroot": "c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e",
    "tx" : [
        "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
        "B268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877c503fbe",

        ... 417 transacciones más ...

       ],
    "time" : 1388185914,
    "nonce" : 924591752,
    "bits" : "1903a30c",
    "difficulty" : 1180923195.25802612,
    "chainwork" : "0000000000000000000000000000000000000000000000934695e92aaf53afa1a",
    "previousblockhash" :
"000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569",
    "nextblockhash" :
"000000000000000010236c269dd6ed714dd5db39d36b33959079d78dfd431ba7"
}
```

La Transacción Generación

La primera transacción añadida al bloque es una transacción especial, llamada *transacción generación* o *transacción coinbase*. Esta transacción es construida por el nodo de Jing y es su recompensa por el esfuerzo de la minería. El nodo de Jing crea la transacción generación como un pago a su propia cartera: "Pague a la dirección de Jing 25,09094928 bitcoin." La cantidad total de recompensa que Jing recibe por haber minado un bloque es la suma de la recompensa coinbase (25 nuevos bitcoins) y las comisiones de transacción (0,09094928) de todas las transacciones incluidas en el bloque como se muestra en Transacción Generation:

```
$ bitcoin-cli getrawtransaction
d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f 1
```

Example 4. Transacción Generation

```
{
    "hex" :
"010000000100000000000000000000000000000000000000000000000000000000ffffffff0f
03443b0403858402062f503253482fffffffff0110c08d9500000000232102aa970c592640d19de03ff6f
329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac00000000",
    "txid" : "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "coinbase" : "03443b0403858402062f503253482f",
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 25.09094928,
            "n" : 0,
            "scriptPubKey" : {
                "asm" :
"02aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21OP_CHECKSIG",
                "hex" :
"2102aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac",
                "reqSigs" : 1,
                "type" : "pubkey",
                "addresses" : [
                    "1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N"
                ]
            }
        }
    ],
    "blockhash" : "000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
    "confirmations" : 35566,
    "time" : 1388185914,
    "blocktime" : 1388185914
}
```

A diferencia de las transacciones normales, la transacción generación no consume (gasta) UTXO como entradas. En su lugar, solo tiene una entrada, llamada el *coinbase*, que crea bitcoin de la nada. La transacción generación tiene una salida, que paga a la propia dirección bitcoin del minero. La salida de la transacción generación envía el valor de 25,09094928 bitcoins a la dirección bitcoin del minero, en este caso 1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N.

Recompensa de Coinbase y Comisiones

Para construir la transacción generación, el nodo de Jing primero calcula el importe total de las comisiones por transacciones mediante la suma de todas las entradas y salidas de las 418 transacciones que se han añadido al bloque. Las comisiones se calculan como:

```
Total Comisiones = Suma(Entradas) - Suma(Salidas)
```

En el bloque 277.316 el total de comisiones de transacción es de 0,09094928 bitcoins.

A continuación, el nodo de Jing calcula la recompensa correcta para el nuevo bloque. La recompensa se calcula en base a la altura del bloque, comenzando por los 50 bitcoins por bloque y reduciendo a la mitad cada 210.000 bloques. Debido a que este bloque está a la altura de 277.316, la recompensa correcta es 25 bitcoins.

El cálculo se puede ver en la función GetBlockValue en el cliente Bitcoin Core, como se muestra en Calculando la recompensa de bloque — función GetBlockValue, cliente Bitcoin Core, main.cpp, línea 1305.

Example 5. Calculando la recompensa de bloque — función GetBlockValue, cliente Bitcoin Core, main.cpp, línea 1305

```
int64_t GetBlockValue(int nHeight, int64_t nFees)
{
    int64_t nSubsidy = 50 * COIN;
    int halvings = nHeight / Params().SubsidyHalvingInterval();

    // Forzar la recompensa de bloque a cero cuando desplazamiento a la derecha no
está definido.
    if (halvings >= 64)
        return nFees;

    // La recompensa se reduce a la mitad cada 210.000 bloques que se producirán
aproximadamente cada 4 años.
    nSubsidy >>= halvings;

    return nSubsidy + nFees;
}
```

La recompensa inicial se calcula en satoshis multiplicando 50 por la constante COIN (100,000.000 satoshis). Esto establece la recompensa inicial (nSubsidy) a 5 mil millones de satoshis.

A continuación, la función calcula el número de halvings que se han producido ("halving" es un término en inglés que significa "reducción a la mitad"). Para ello, la función divide la altura del bloque

actual por el intervalo de "halving" (SubsidyHalvingInterval). En el caso del bloque 277.316, con un intervalo de "halving" cada 210.000 bloques, el resultado es 1 "halving".

El número máximo de "halvings" permitido es 64, por lo que el código impone una recompensa cero (retorna solo las comisiones) si se superan los 64 "halvings".

A continuación, la función utiliza el operador de desplazamiento-derecha-binario para dividir la recompensa (nSubsidy) entre dos para cada ronda de "halving". En el caso del bloque 277.316, se haría un único desplazamiento binario hacia la derecha de la recompensa de 5 mil millones de satoshis (un "halving") dejando como resultado 2,5 mil millones de satoshis o 25 bitcoins. Se utiliza el operador desplazamiento-binario-derecha porque es más eficiente para dividir entre dos que la división de números enteros o de punto flotante.

Finalmente, se añade la recompensa coinbase (nSubsidy) a las comisiones de transacción (nFees), y se devuelve la suma.

Estructura de la Transacción Generación

Con estos cálculos, el nodo de Jing construye después la transacción generación, pagándose a sí mismo 25,09094928 bitcoin.

Como se puede ver en Transacción Generation, la transacción generación tiene un formato especial. En lugar de una entrada de transacción que especifica una UTXO anterior para ser gastado, tiene una entrada de "coinbase". Examinamos las entradas de transacción en [tx_in_structure]. Vamos a comparar una entrada de transacción normal con una entrada de transacción generación. La estructura de una entrada de transacción "normal" muestra la estructura de una transacción normal, mientras que La Estructura de una entrada de transacción generación muestra la estructura de entrada de la transacción generación.

Table 1. La estructura de una entrada de transacción "normal"

Tamaño	Campo	Descripción
32 bytes	Hash de Transacción	Puntero a la transacción que contiene la UTXO a ser gastada
4 bytes	Índice de Salida	El número de índice de la UTXO que se pasó, primera es 0
1-9 bytes (VarInt)	Tamaño del Script de Desbloqueo	Longitud del Script de Desbloqueo en bytes, a seguir
Variable	Script de Desbloqueo	Un script que cumple con las condiciones del script de bloqueo de UTXOs

4 bytes	Número de Secuencia	Funcionalidad de reemplazo de transacción actualmente deshabilitada, establecer en 0xFFFFFFFF

Table 2. La Estructura de una entrada de transacción generación

Tamaño	Campo	Descripción
32 bytes	Hash Transacción	Todos los bits son cero: No es una referencia de hash transacción
4 bytes	Índice de salida	Todos los bits son requeridos: 0xFFFFFFFF
1-9 bytes (VarInt)	Tamaño de datos Coinbase	Longitud de los datos coinbase, de 2 a 100 bytes
Variable	Coinbase datos	datos arbitrarios utilizados para nonce extra y etiquetas mineras en v2 bloques, debe comenzar con la altura del bloque
4 bytes	Número de secuencia	Ajuste a 0xFFFFFFFF

En una transacción de generación, los dos primeros campos se establecen en valores que no representan una referencia UTXO. En lugar de un "Hash Transacción", el primer campo se rellena con 32 bytes todos a cero. El "Índice de Salida" se rellena con 4 bytes todos a 0xFF (255 decimal). El "Script de Desbloqueo" se sustituye por los datos coinbase, un campo de datos arbitrario utilizado por los mineros.

Datos Coinbase

Las transacciones generación no tienen un campo para script de desbloqueo(scriptSig). En cambio, este campo se sustituye por los datos coinbase, que deben ser de entre 2 y 100 bytes. A excepción de los primeros bytes, el resto de los datos coinbase pueden ser utilizados por los mineros en cualquier forma que deseen; se trata de datos arbitrarios.

En el bloque de génesis, por ejemplo, Satoshi Nakamoto añadió el texto "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks" (traducido al español: "The Times 03/Ene/2009 Canciller preparado para segundo rescate a los bancos") en los datos coinbase, usándolo como una prueba de la fecha y para transmitir un mensaje. Actualmente, los mineros utilizan los datos coinbase para incluir valores nonce adicionales y cadenas que identifican el pool de minería, como veremos en las siguientes secciones.

Los primeros bytes del coinbase solían ser arbitrarios, pero ya no es así. Según la Propuesta de Mejora Bitcoin 34 (BIP0034), los bloques de versión-2 (bloques cuyo campo de versión se establece en 2) deben contener el índice de altura del bloque como un script de operación "push" en el principio del campo

coinbase.

En el bloque 277.316 vemos que el coinbase (ver Transacción Generation), que está en el "Script de Desbloqueo" o en el campo scriptSig de la entrada de transacción, contiene el valor hexadecimal 03443b0403858402062f503253482f. Vamos a decodificar este valor.

El primer byte, 03, indica al motor de ejecución de scripts que empuje los siguientes tres bytes en la pila de script (ver [tx_script_ops_table_pushdata]). Los siguientes tres bytes, 0x443b04, son la altura del bloque codificado en formato little-endian (hacia atrás, byte menos significativo primero). Invertiendo el orden de los bytes, el resultado sería 0x043b44, que es 277.316 en decimal.

Las siguientes dígitos hexadecimales (03858402062) se utilizan para codificar un *nonce* extra (ver La Solución de Nonce Extra), o valor aleatorio, que se utiliza para encontrar una solución adecuada a la prueba de trabajo.

La parte final de los datos coinbase (2f503253482f) es la cadena codificada en ASCII /P2SH/, que indica que el nodo de minería que extrae este bloque apoya la mejora pago-a-hash-de-script (P2SH) definida en BIP0016. La introducción de la capacidad P2SH requirió un "voto" por los mineros para respaldar ya fuese BIP0016 o BIP0017. Aquellos que respaldaron la implementación BIP0016 incluyeron /P2SH/ en sus datos coinbase. Aquellos que respaldaron la implementación BIP0017 de P2SH incluyeron la cadena p2sh/CHV en sus datos coinbase. El BIP0016 fue elegido como el ganador, y muchos mineros continuaron incluyendo la cadena /P2SH/ en su coinbase para indicar el apoyo a esta funcionalidad.

Extraer los datos coinbase del bloque génesis utiliza la biblioteca libbitcoin introducida en [alt_libraries] para extraer los datos coinbase del bloque génesis que muestran el mensaje de Satoshi. Tenga en cuenta que la biblioteca libbitcoin contiene una copia estática del bloque génesis, por lo que el código de ejemplo puede recuperar el bloque génesis directamente desde la biblioteca.

Example 6. Extraer los datos coinbase del bloque génesis

```
/*
  Display the genesis block message by Satoshi.
*/
#include <iostream>
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Create genesis block.
    bc::block_type block = bc::genesis_block();
    // Genesis block contains a single coinbase transaction.
    assert(block.transactions.size() == 1);
    // Get first transaction in block (coinbase).
    const bc::transaction_type& coinbase_tx = block.transactions[0];
    // Coinbase tx has a single input.
    assert(coinbase_tx.inputs.size() == 1);
    const bc::transaction_input_type& coinbase_input = coinbase_tx.inputs[0];
    // Convert the input script to its raw format.
    const bc::data_chunk& raw_message = save_script(coinbase_input.script);
    // Convert this to an std::string.
    std::string message;
    message.resize(raw_message.size());
    std::copy(raw_message.begin(), raw_message.end(), message.begin());
    // Display the genesis block message.
    std::cout << message << std::endl;
    return 0;
}
```

Compilamos el código con el compilador GNU C++ y lanzamos el ejecutable resultante, como se muestra en Compilando y ejecutando el código de ejemplo satoshi-words.

Example 7. Compilando y ejecutando el código de ejemplo satoshi-words

```
$ # Compilando el código
$  g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Ejecutar el ejecutable
$ ./satoshi-words
^D    <GS>^A^DEThe Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

Construyendo la Cabecera de Bloque

Para construir la cabecera de bloque, el nodo de minería tiene que llenar seis campos, como se indica en La estructura de la cabecera de bloque.

Table 3. La estructura de la cabecera de bloque

Tamaño	Campo	Descripción
4 bytes	Versión	Un número de versión para seguir las actualizaciones de software y protocolo
32 bytes	Hash del Bloque Anterior	Una referencia al hash del bloque anterior (padre) en la cadena
32 bytes	Raíz Merkle	Un hash de la raíz del árbol merkle de las transacciones de este bloque
4 bytes	Hora	El tiempo de creación aproximada de este bloque (segundos desde Unix Epoch)
4 bytes	Objetivo de Dificultad	El objetivo de dificultad del algoritmo de prueba de trabajo para este bloque
4 bytes	Nonce	Un contador usado para el algoritmo de prueba de trabajo

En el momento que el bloque 277.316 fue minado, el número de versión que describe la estructura de bloques es la versión 2, que está codificada en formato little-endian en 4 bytes como 0x02000000.

A continuación, el nodo de minería tiene que añadir el "Hash del Bloque Anterior." Ese es el hash de la cabecera de bloque del bloque 277.315, el bloque anterior recibido de la red, que el nodo de Jing ha aceptado y elegido como el padre del bloque candidato 277.316. El hash de la cabecera de bloque para el bloque 277.315 es:

```
0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

El siguiente paso es resumir todas las transacciones con un árbol merkle, a fin de añadir la raíz merkle a la cabecera de bloque. La transacción generación aparece como la primera transacción en el bloque. Después de ella, se añaden otras 418 transacciones más, para un total de 419 operaciones en el bloque. Como vimos en [merkle_trees], debe haber un número par de nodos "hoja" en el árbol, por lo que la última transacción se duplica, creando 420 nodos, cada uno con el hash de una transacción. Los hashes de transacción se combinan de dos en dos, creando cada nivel del árbol, hasta que todas las

transacciones se resumen en un solo nodo en la "raíz" del árbol. La raíz del árbol merkle resume todas las transacciones en un solo valor de 32 bytes, que se puede ver marcado como "merkle root" en Bloque 277.316, y aquí:

```
c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e
```

El nodo de minería añadirá después un sello de tiempo de 4 bytes, codificado como una fecha "Unix Epoch", que indica el número de segundos transcurridos desde la medianoche UTC/GMT del 1 de enero de 1970. La hora 1388185914 es igual a viernes, 27 de diciembre 2013, 23:11:54 UTC/GMT.

Después, el nodo rellena el objetivo de dificultad, que define la dificultad de la prueba de trabajo que se requiere para hacer que este sea un bloque válido. La dificultad se almacena en el bloque como una métrica "bits de dificultad", que es una codificación exponente-mantisa del objetivo. La codificación tiene un exponente de 1 byte, seguido por una mantisa de 3-bytes (coeficiente). En el bloque 277.316, por ejemplo, el valor de los bits de dificultad es 0x1903a30c. La primera parte 0x19 es un exponente hexadecimal, mientras que la siguiente parte, 0x03a30c, es el coeficiente. El concepto de un objetivo de dificultad se explica en Objetivo de Dificultad y Recálculo de Dificultad. y la representación de los "bits de dificultad" se explica en Representación de la Dificultad.

El campo final es el nonce, que se inicializa a cero.

Con todos los otros campos llenos, la cabecera de bloque ya está completa y el proceso de la minería puede comenzar. El objetivo es ahora encontrar un valor para el nonce que resulte en un hash de cabecera de bloque que sea menor que el objetivo de dificultad. El nodo de minería tendrá que probar miles de millones o billones de valores nonce antes de encontrar un nonce que satisfaga el requisito.

Minando el Bloque

Ahora que el nodo de Jing ha construido un bloque candidato, es hora de que la plataforma minera de hardware de Jing "mine" el bloque, para encontrar una solución al algoritmo de prueba de trabajo que haga que el bloque sea válido. A lo largo de este libro hemos estudiado las funciones hash criptográficas, tal como se usan en varios aspectos del sistema bitcoin. La función hash SHA256 es la función que se utiliza en el proceso de minería de bitcoin.

En los términos más simples, la minería es el proceso de hacer hash de la cabecera del bloque de manera repetitiva, cambiando un parámetro, hasta que el hash resultante coincida con un objetivo específico. El resultado de la función hash no puede determinarse de antemano, ni se puede crear un patrón que vaya a producir un valor hash específico. Esta característica de las funciones de hash significa que la única manera de producir un hash resultante que coincida con un objetivo específico es intentarlo una y otra vez, modificando aleatoriamente la entrada hasta que el hash resultante que se desea aparezca por casualidad.

Algoritmo de Prueba De Trabajo

Un algoritmo de hash toma una entrada de datos de longitud arbitraria y produce un resultado determinista de longitud fija, una huella digital de la entrada. Para cualquier entrada específica, el hash resultante será siempre el mismo y puede ser calculado y verificado fácilmente por cualquier persona aplicando el mismo algoritmo de hash. La característica clave de un algoritmo de hash criptográfico es que es prácticamente imposible encontrar dos entradas diferentes que produzcan la misma huella digital. Como corolario, también es prácticamente imposible seleccionar una entrada de tal forma que produzca una huella digital deseada. La única manera es intentar con entradas de manera aleatoria.

Con SHA256, la salida es siempre de 256 bits de longitud, independientemente del tamaño de la entrada. En [sha256_example1], vamos a utilizar el intérprete de Python para calcular el hash SHA256 de la frase: "I am Satoshi Nakamoto."

Ejemplo .SHA256

```
$ python
```

```
Python 2.7.1
>>> import hashlib
>>> print hashlib.sha256("I am Satoshi Nakamoto").hexdigest()
5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e
```

[sha256_example1] muestra el resultado de calcular el hash de "I am Satoshi Nakamoto" 5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e. Este número de 256 bits es el *hash* o *digest* de la frase y depende de todos y cada uno de los elementos de frase. Si se añade una sola letra, signo de puntuacion, o cualquier otro carácter, producirá un hash diferente.

Por tanto, si cambiamos la frase, deberíamos ver hashes completamente diferentes. Vamos a comprobarlo añadiendo un número al final de nuestra frase, usando el script simple de Python en SHA256 Un script para generar muchos hashes iterando un nonce.

Example 8. SHA256 Un script para generar muchos hashes iterando un nonce

```
# example of iterating a nonce in a hashing algorithm's input

import hashlib

text = "I am Satoshi Nakamoto"

# iterate nonce from 0 to 19
for nonce in xrange(20):

    # add the nonce to the end of the text
    input = text + str(nonce)

    # calculate the SHA-256 hash of the input (text+nonce)
    hash = hashlib.sha256(input).hexdigest()

    # show the input and hash result
    print input, '=>',  hash
```

La ejecución de este script producirá los hashes de varias frases. Se ha añadido un número al final de las frases para hacerlas diferentes entre sí. Incrementando el número, podemos obtener diferentes hashes, como se muestra en Salida SHA256 de un script para generar muchos hashes iterando un nonce.

Example 9. Salida SHA256 de un script para generar muchos hashes iterando un nonce

```
$ python hash_example.py
```

```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

Cada frase produce un hash resultante completamente diferente. Parecen completamente al azar, pero si replica los valores exactos de este ejemplo en cualquier computadora con Python, obtendrá exactamente los mismos hashes.

El número que se ha utilizado como variable en este escenario se llama *nonce*. El nonce se utiliza para variar la salida de una función criptográfica, en este caso para variar la huella digital SHA256 de la frase.

Como ejemplo de este algoritmo, vamos a establecer un objetivo arbitrario: encontrar una frase que produzca un hash hexadecimal que comience con un cero. Afortunadamente, ¡esto no es difícil! Salida SHA256 de un script para generar muchos hashes iterando un nonce muestra que la frase "I am Satoshi Nakamoto13" produce el hash 0ebc56d59a34f5082aaef3d66b37a661696c2b618e62432727216ba9531041a5, que se ajusta a nuestros criterios. Se necesitaron 13 intentos para encontrarlo. En términos de probabilidades, si la salida de la función hash se distribuye uniformemente deberíamos encontrar un resultado con un 0 como prefijo hexadecimal una vez de cada 16 hashes (uno de cada 16 dígitos hexadecimales 0 a F). En términos numéricos, eso significaría encontrar un valor hash que sea menor que 0x1000000000000000000000000000000000000000000000000000000000000000. A este umbral, lo

llamamos *objetivo*, y la intención es encontrar un hash que sea numéricamente *menor que el objetivo*. Si disminuimos el objetivo, la tarea de encontrar un hash que sea menor que el objetivo se vuelve más y más difícil.

Por hacer una analogía, imagine un juego donde los jugadores tiran un par de dados en repetidas ocasiones, tratando sacar un valor por debajo de un objetivo especificado. En la primera ronda, el objetivo es 12. Si obtiene cualquier valor distinto de un doble seis, usted gana. En la siguiente ronda el objetivo es 11. Los jugadores deben obtener un 10 o menos para ganar, de nuevo una tarea fácil. Digamos que un par de rondas más tarde, el objetivo se ha reducido a 5. Ahora, más de la mitad de los lanzamientos de dados sumarían más de 5 y por lo tanto serían jugadas perdedoras. Cuanto menor sea el objetivo, se necesitarán exponencialmente más lanzamientos de dados para ganar. Finalmente, cuando el objetivo sea 2 (el mínimo posible), sólo un lanzamiento de cada 36, o el 2% de ellos, producirá un resultado ganador.

En Salida SHA256 de un script para generar muchos hashes iterando un nonce, el "nonce" ganador es 13 y este resultado puede ser confirmardo independientemente por cualquier persona. Cualquiera puede añadir el número 13 como sufijo a la frase "I am Satoshi Nakamoto" y calcular el hash, verificando que es menor que el objetivo. El resultado exitoso es también una prueba de trabajo, porque prueba que hicimos el trabajo para encontrar el nonce. Si bien solo se necesita un cálculo de hash para verificar, nos tomó 13 cálculos de hash para encontrar un nonce que funcionara. Si tuviéramos un objetivo menor (mayor dificultad) se necesitarían muchos más cálculos de hash para encontrar un nonce adecuado, pero solo un cálculo de hash para que cualquiera pueda verificarlo. Por otra parte, al conocer el objetivo, cualquiera puede estimar la dificultad mediante el uso de estadísticas y por lo tanto saber la cantidad de trabajo que se necesitó para encontrar ese nonce.

La prueba de trabajo de Bitcoin es muy similar al desafío que se muestra en Salida SHA256 de un script para generar muchos hashes iterando un nonce. El minero construye un bloque candidato lleno de transacciones. A continuación, el minero calcula el hash de la cabecera de este bloque y ve si es más pequeño que el *objetivo* actual. Si el hash no es menor que el objetivo, el minero modificará el nonce (por lo general incrementando solo por uno) y lo intentará de nuevo. Para la dificultad actual en la red bitcoin, los mineros tienen que intentar trillones de veces antes de encontrar un nonce que produzca un hash de cabecera de bloque lo suficientemente bajo.

Un algoritmo muy simplificado de prueba de trabajo se implementa en Python en Implementación de prueba de trabajo simplificada.

Example 10. Implementación de prueba de trabajo simplificada

```
#!/usr/bin/env python
# example of proof-of-work algorithm

import hashlib
import time

max_nonce = 2 ** 32 # 4 billion
```

```
def proof_of_work(header, difficulty_bits):

    # calculate the difficulty target
    target = 2 ** (256-difficulty_bits)

    for nonce in xrange(max_nonce):
        hash_result = hashlib.sha256(str(header)+str(nonce)).hexdigest()

        # check if this is a valid result, below the target
        if long(hash_result, 16) < target:
            print "Success with nonce %d" % nonce
            print "Hash is %s" % hash_result
            return (hash_result,nonce)

    print "Failed after %d (max_nonce) tries" % nonce
    return nonce


if __name__ == '__main__':

    nonce = 0
    hash_result = ''

    # difficulty from 0 to 31 bits
    for difficulty_bits in xrange(32):

        difficulty = 2 ** difficulty_bits
        print "Difficulty: %ld (%d bits)" % (difficulty, difficulty_bits)

        print "Starting search..."

        # checkpoint the current time
        start_time = time.time()

        # make a new block which includes the hash from the previous block
        # we fake a block of transactions - just a string
        new_block = 'test block with transactions' + hash_result

        # find a valid nonce for the new block
        (hash_result, nonce) = proof_of_work(new_block, difficulty_bits)

        # checkpoint how long it took to find a result
        end_time = time.time()

        elapsed_time = end_time - start_time
        print "Elapsed Time: %.4f seconds" % elapsed_time

        if elapsed_time > 0:
```

```
            # estimate the hashes per second
            hash_power = float(long(nonce)/elapsed_time)
            print "Hashing Power: %ld hashes per second" % hash_power
```

Mediante la ejecución de este código, puede establecer la dificultad que desee (en bits, cuántos de los primeros bits deben ser cero) y ver cuánto tiempo se necesita para encontrar una solución en su equipo. En Ejecutando el ejemplo de prueba de trabajo para diversas dificultades, se puede ver cómo funciona en un ordenador portátil normal.

Example 11. Ejecutando el ejemplo de prueba de trabajo para diversas dificultades

```
$ python proof-of-work-example.py*
```

```
Dificultad: 1 (0 bits)

[...]

Dificultad: 8 (3 bits)
Empezando la búsqueda ...
Encontrado con nonce 9
Hash es 1c1c105e65b47142f028a8f93ddf3dabb9260491bc64474738133ce5256cb3c1
Tiempo transcurrido: 0,0004 segundos
Poder de Hashing: 25065 hashes por segundo
Dificultad: 16 (4 bits)
Empezando la búsqueda ...
Encontrado con nonce 25
Hash es 0f7becfd3bcd1a82e06663c97176add89e7cae0268de46f94e7e11bc3863e148
Tiempo transcurrido: 0,0005 segundos
Poder de Hashing: 52507 hashes por segundo
Dificultad: 32 (5 bits)
Empezando la búsqueda ...
Encontrado con nonce 36
Hash es 029ae6e5004302a120630adcbb808452346ab1cf0b94c5189ba8bac1d47e7903
Tiempo transcurrido: 0,0006 segundos
Poder de Hashing: 58164 hashes por segundo

[...]

Dificultad: 4194304 (22 bits)
Empezando la búsqueda ...
Encontrado con nonce 1759164
Hash es 0000008bb8f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cefc3
Tiempo transcurrido: 13,3201 segundos
```

```
Poder de Hashing: 132068 hashes por segundo
Dificultad: 8388608 (23 bits)
Empezando la búsqueda ...
Encontrado con nonce 14214729
Hash es 000001408cf12dbd20fcba6372a223e098d58786c6ff93488a9f74f5df4df0a3
Tiempo transcurrido: 110.1507 seconds
Poder de Hashing: 129048 hashes por segundo
Dificultad: 16777216 (24 bits)
Empezando la búsqueda ...
Encontrado con nonce 24586379
Hash es 0000002c3d6b370fccd699708d1b7cb4a94388595171366b944d68b2acce8b95
Tiempo transcurrido: 195.2991 seconds
Poder de Hashing: 125890 hashes por segundo

[...]

Dificultad: 67108864 (26 bits)
Empezando la búsqueda ...
Encontrado con nonce 84561291
Hash es 0000001f0ea21e676b6dde5ad429b9d131a9f2b000802ab2f169cbca22b1e21a
Tiempo transcurrido: 665.0949 seconds
Poder de Hashing: 127141 hashes por segundo
```

Como se puede ver, el aumento de la dificultad por 1 bit provoca un aumento exponencial en el tiempo que se tarda en encontrar una solución. Si piensa en todo el espacio de números de 256 bits, cada vez que añade un bit más a cero, se reduce el espacio de búsqueda a la mitad. En Ejecutando el ejemplo de prueba de trabajo para diversas dificultades, se necesitan 84 millones de intentos de hash para encontrar un nonce que produzca un hash con sus primeros 26 bits a cero. Incluso a una velocidad de más de 120.000 hashes por segundo, aún se requerirían 10 minutos en un portátil de consumo para encontrar esta solución.

En el momento de escribir esto, la red está tratando de encontrar un bloque cuyo hash de cabecera sea inferior a 000000000000004c296e6376db3a241271f43fd3f5de7ba18986e517a243baa7. Como se puede ver, hay un montón de ceros al comienzo de ese hash, lo que significa que el rango aceptable de valores hash es mucho menor, por lo tanto, es más difícil encontrar un hash válido. Tomará en promedio más de 150 billones de cálculos de hash por segundo a la red para descubrir el siguiente bloque. Esto puede parecer una tarea imposible, pero afortunadamente la red está tramitando 100 petahashes por segundo (PH/seg) de potencia de procesamiento, lo que le hará capaz de encontrar, en promedio, un bloque cada 10 minutos.

Representación de la Dificultad

En Bloque 277.316, vimos que el bloque contiene el objetivo de dificultad, en una notación denominada "bits de dificultad" o simplemente "bits", que en el bloque 277.316 tiene el valor de 0x1903a30c. Esta notación expresa el objetivo de dificultad en un formato de coeficiente/exponente, con los dos

primeros dígitos hexadecimales para el exponente y los siguientes seis dígitos hexadecimales como el coeficiente. En este bloque, por lo tanto, el exponente es 0x19 y el coeficiente es 0x03a30c.

La fórmula para calcular el objetivo de dificultad a partir de esta representación es:

```
objetivo = coeficiente * 2^(8 * (exponente – 3))
```

Usando esa fórmula, y el valor de los bits de dificultad 0x1903a30c, obtenemos:

```
objetivo = 0x03a30c * 2^(0x08 * (0x19 - 0x03))^

=> objetivo= 0x03a30c * 2^(0x08 * 0x16)^

=> objetivo = 0x03a30c * 2^0xB0^
```

que en decimal es:

```
=> objetivo = 238.348 * 2^176^

=> objetivo =
22.829.202.948.393.929.850.749.706.076.701.368.331.072.452.018.388.575.715.328
```

cambiando de nuevo a hexadecimal:

```
=> objetivo = 0x00000000000000003A30C0000000000000000000000000000000000000000000
```

Esto significa que un bloque válido para la altura 277.316 es uno que tenga un hash de cabecera de bloque que sea menor que el objetivo. En binario, ese número tendría al menos los primeros 60 bits puestos a cero. Con este nivel de dificultad, un solo minero procesando un billón de hashes por segundo (1 tera-hash por segundo o 1 TH/seg) solo podría encontrar una solución una vez cada 8496 bloques o una vez cada 59 días, en promedio.

Objetivo de Dificultad y Recálculo de Dificultad.

Como hemos visto, el objetivo determina la dificultad y por lo tanto afecta a cuánto tiempo se tarda en encontrar una solución al algoritmo de prueba de trabajo. Esto lleva a la pregunta obvia: ¿Por qué es la dificultad ajustable, qué se ajusta, y cómo?

Los bloques de Bitcoin se generan cada 10 minutos, de promedio. Este es el latido del bitcoin y sostiene la frecuencia de emisión de la moneda y la velocidad de la liquidación de las transacciones. Tiene que permanecer constante no solo en el corto plazo, sino a lo largo de muchas décadas. Durante este tiempo, se espera que la potencia de los ordenadores seguirá aumentando a un ritmo rápido. Además,

el número de participantes en la minería y los ordenadores que utilizan están también en constante cambio. Para mantener el tiempo de generación de bloque en 10 minutos, la dificultad de la minería debe ajustarse para tener en cuenta estos cambios. De hecho, la dificultad es un parámetro dinámico que se ajusta periódicamente para cumplir con un objetivo de bloque de 10 minutos. En términos simples, el objetivo de dificultad se ajusta a la potencia de minería que hace que el intervalo entre bloques sea de 10 minutos.

Entonces, ¿Cómo se hace un ajuste de este tipo en una red completamente descentralizada? El recálculo de la dificultad se produce en cada nodo completo de forma automática e independiente. Cada 2016 bloques, todos los nodos recalculan el objetivo de dificultad de la prueba de trabajo. La ecuación para recalcular el objetivo de dificultad mide el tiempo que se tardó en encontrar los últimos 2016 bloques y lo compara con el tiempo esperado de 20160 minutos (dos semanas sobre la base de un bloque de tiempo deseado de 10 minutos). Se calcula el cociente entre el intervalo de tiempo real y el intervalo de tiempo deseado y se hace el ajuste correspondiente (arriba o abajo) a la dificultad. En términos simples: Si la red está encontrando bloques más rápido que cada 10 minutos, la dificultad aumenta. Si la extracción de bloques es más lenta de lo esperado, la dificultad disminuye.

La ecuación se puede resumir como:

```
Nueva Dificultad = Dificultad Antigua * (Tiempo Real de 2016 Últimos Bloques / 20160
minutos)
```

Recálculo de la dificultad de prueba de trabajo—GetNextWorkRequired () en pow.cpp, línea 43 muestra el código utilizado en el cliente Bitcoin Core.

Example 12. Recálculo de la dificultad de prueba de trabajo—GetNextWorkRequired () en pow.cpp, línea 43

```
// Retroceder lo que queremos que sean 14 días en bloques
const CBlockIndex* pindexFirst = pindexLast;
for (int i = 0; pindexFirst && i < Params().Interval()-1; i++)
    pindexFirst = pindexFirst->pprev;
assert(pindexFirst);

// Etapa de ajuste de límite
int64_t nActualTimespan = pindexLast->GetBlockTime() - pindexFirst->GetBlockTime();
LogPrintf("  nActualTimespan = %d  before bounds\n", nActualTimespan);
if (nActualTimespan < Params().TargetTimespan()/4)
    nActualTimespan = Params().TargetTimespan()/4;
if (nActualTimespan > Params().TargetTimespan()*4)
    nActualTimespan = Params().TargetTimespan()*4;

// Recálculo
uint256 bnNew;
uint256 bnOld;
bnNew.SetCompact(pindexLast->nBits);
bnOld = bnNew;
bnNew *= nActualTimespan;
bnNew /= Params().TargetTimespan();

if (bnNew > Params().ProofOfWorkLimit())
    bnNew = Params().ProofOfWorkLimit();
```

Aunque la calibración de la dificultad sucede cada 2016 bloques, debido a un error de desvío-por-uno en el cliente original Bitcoin Core, el tiempo total se obtiene de los 2015 bloques anteriores (no 2016 como debería ser), lo que resulta en un sesgo en el recálculo hacia una mayor dificultad en 0,05%.

Los parámetros Interval (2016 bloques) y TargetTimespan (dos semanas como 1,209.600 segundos) se definen en *chainparams.cpp*.

Para evitar la volatilidad extrema en la dificultad, el ajuste de recálculo debe ser inferior a un factor de cuatro (4) por ciclo. Si el ajuste de dificultad requerida es mayor que un factor de cuatro, se ajustará por el máximo y no más. Cualquier otro ajuste se llevará a cabo en el siguiente período de recálculo porque el desequilibrio persistirá a través de los siguientes 2016 bloques. Por lo tanto, las grandes discrepancias entre la potencia de hash y la dificultad podrían tardar varios ciclos de 2016 bloques en equilibrarse.

TIP	La dificultad de encontrar un bloque bitcoin es aproximadamente '10 minutos de procesamiento' para toda la red, tomando como base el tiempo que se tardó en encontrar los 2016 bloques anteriores, ajustado cada 2016 bloques.

Tenga en cuenta que el objetivo de dificultad es independiente del número de transacciones o del valor de las transacciones. Esto significa que la cantidad de potencia de hash y por tanto la electricidad gastada para asegurar bitcoin también es completamente independiente del número de transacciones. Bitcoin puede tramitar más transacciones, lograr una adopción más amplia, y permanecer seguro sin aumentar la potencia de hash desde el nivel actual. El aumento de la potencia de hash es una representación de las fuerzas del mercado a medida que nuevos mineros entran en el mercado para competir por la recompensa. Mientras exista una potencia de hash suficiente bajo el control de mineros honestos en la búsqueda de la recompensa, se prevendrán los ataques de "toma de control" y, por lo tanto, será suficiente para asegurar bitcoin.

El objetivo de dificultad está estrechamente relacionado con el coste de la electricidad y el tipo de cambio de bitcoin con la moneda utilizada para pagar la electricidad. Los sistemas de minería de alto rendimiento han alcanzado prácticamente la máxima eficiencia que le permite la generación actual de fabricación de silicio, convirtiendo la electricidad en la mayor potencia de hash posible. El principal factor que influye en el mercado de la minería es el precio de un kilovatio-hora en bitcoin, ya que determina la rentabilidad de la minería y por lo tanto los incentivos para entrar o salir del mercado minero.

Éxito en el Minado de un Bloque

Como vimos anteriormente, el nodo de Jing ha construido un bloque candidato y lo ha preparado para la minería. Jing tiene varias plataformas para minería hardware con circuitos integrados de aplicación específica (en inglés, "ASIC"), donde cientos de miles de circuitos integrados ejecutan en paralelo el algoritmo SHA256 a velocidades increíbles. Estas máquinas especializadas están conectadas a su nodo de minería a través de USB. A continuación, el nodo de minería que se ejecuta en el escritorio de Jing transmite la cabecera del bloque a su hardware de minería, que comienza a probar billones de nonces por segundo.

Casi 11 minutos después de comenzar a minar el bloque 277.316, una de las máquinas de minería hardware encuentra una solución y lo envía de vuelta al nodo de minería. Cuando se inserta en la cabecera del bloque, el nonce 4215469401 produce un hash de bloque:

```
0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

que es menor que el objetivo:

```
00000000000000003A30C0000000000000000000000000000000000000000000
```

Inmediatamente, el nodo de minería de Jing transmite el bloque a todos sus compañeros. Ellos reciben,

validan, y luego propagan el nuevo bloque. A medida que el bloque se va extendiendo a través de la red, cada nodo lo añade a su propia copia de la cadena de bloques, extendiéndola a una nueva altura de 277.316 bloques. A medida que los nodos de minería reciben y validan el bloque, abandonan sus esfuerzos para encontrar un bloque a la misma altura y comienzan de inmediato a calcular el siguiente bloque de la cadena.

En la siguiente sección, vamos a ver el proceso que cada nodo utiliza para validar un bloque y seleccionar la cadena más larga, creando el consenso que forma la cadena de bloques descentralizada.

Validación de un Nuevo Bloque

El tercer paso en el consenso del mecanismo de bitcoin es la validación independiente de cada nuevo bloque por cada nodo en la red. A medida que el bloque recién resuelto se propaga a través de la red, cada nodo lleva a cabo una serie de pruebas de validación antes de propagarlo a sus compañeros. Esto asegura que solo los bloques válidos se propagan en la red. La validación independiente también asegura que los mineros que actúan con honestidad consigan que sus bloques sean incorporados a la cadena de bloques, lo que les hace ganar la recompensa. Aquellos mineros que actúen deshonestamente verán que sus bloques son rechazados y no solo pierden la recompensa, sino que también pierden el esfuerzo realizado para encontrar una solución de prueba de trabajo, incurriendo así en el coste de la electricidad sin compensación.

Cuando un nodo recibe un nuevo bloque, validará el bloque verificando contra una larga lista de criterios que deberán cumplirse; de lo contrario, el bloque se rechaza. Estos criterios se pueden ver en el cliente Bitcoin Core en las funciones CheckBlock y CheckBlockHeader e incluyen:

- La estructura de datos de bloque es sintácticamente válida
- El hash de cabecera del bloque es menor que el objetivo de dificultad (hace cumplir la prueba de trabajo)
- El sello de tiempo del bloque es menos de dos horas en el futuro (lo que permite errores de tiempo)
- El tamaño del bloque está dentro de límites aceptables
- La primera transacción (y solo la primera) es una transacción generación coinbase
- Todas las transacciones dentro del bloque son válidas utilizando la lista de verificación de transacciones discutida en Verificación Independiente de Transacciones

La validación independiente de cada nuevo bloque por cada nodo de la red asegura que los mineros no puedan hacer trampas. En secciones anteriores hemos visto cómo los mineros llegan a escribir una transacción que les premia con los nuevos bitcoins creados dentro del bloque y reclamar las comisiones de transacción. ¿Por qué los mineros no escriben una transacción dirigida a ellos mismos por valor de miles de bitcoin en lugar de la recompensa correcta? Porque todos los nodos validan los bloques respetando las mismas reglas. Una transacción coinbase inválida haría todo el bloque no válido, lo que resultaría en que el bloque sea rechazado y, por tanto, la transacción no se convertiría en parte del libro contable. Los mineros tienen que construir un bloque perfecto, basado en las normas comunes que todos los nodos siguen, y minarlo con una solución correcta de la prueba de trabajo. Para

ello, gastan mucha electricidad en la minería, y si hacen trampas, perderían toda la electricidad y el esfuerzo. Esta es la razón por la que la validación independiente es un componente clave del consenso descentralizado.

Montaje y Selección de Cadenas de Bloques

El último paso en el mecanismo de consenso descentralizado de bitcoin es el montaje de bloques en cadenas y la selección de la cadena con la mayor prueba de trabajo. Una vez que un nodo ha validado un nuevo bloque, entonces intentará montar una cadena conectando el bloque a la cadena de bloques existente.

Los nodos mantienen tres conjuntos de bloques: los relacionados con la cadena de bloques principal, los que forman las ramas de la cadena de bloques principal (cadenas secundarias) y, por último, los bloques que no tienen un padre conocido en las cadenas conocidas (huérfanos). Los bloques no válidos son rechazados en cuanto falla uno cualquiera de los criterios de validación y, por tanto, no están incluidos en ninguna cadena.

La "cadena principal" en cualquier momento es la cadena con bloques que tiene asociada la mayor cantidad de dificultad. Bajo la mayoría de circunstancias, esto coincide con la cadena que tiene el mayor número de bloques, excepto cuando haya dos cadenas de igual longitud y una de ellas tenga mayor prueba de trabajo que la otra. La cadena principal también tendrá ramas con bloques que son "hermanos" de bloques en la cadena principal. Estos bloques son válidos, pero no forman parte de la cadena principal. Se mantienen para futuras referencias, en caso de que una de esas cadenas se extienda y supere en dificultad a la cadena principal. En la siguiente sección (Bifurcaciones de la Cadena de Bloques), vamos a ver cómo se producen cadenas secundarias cuando se minan bloques casi simultáneamente a la misma altura.

Cuando se recibe un nuevo bloque, un nodo intentará ensamblarlo en la cadena de bloques existente. El nodo leerá de ese bloque el campo "hash de bloque anterior", que es la referencia al padre del nuevo bloque. A continuación, el nodo intentará encontrar ese padre en la cadena de bloques existente. La mayoría de las veces, el padre será la "punta" de la cadena principal, lo que significaría que este nuevo bloque extiende la cadena principal. Por ejemplo, el nuevo bloque 277.316 tiene una referencia al hash de su bloque padre 277.315. La mayoría de los nodos que reciban 277.316 ya tendrán al bloque 277.315 como la punta de su cadena principal y por lo tanto enlazarán el nuevo bloque y extenderán esa cadena.

A veces, como veremos en Bifurcaciones de la Cadena de Bloques, el nuevo bloque extiende una cadena que no es la cadena principal. En ese caso, el nodo extenderá la cadena secundaria al conectar el nuevo bloque y luego comparará la dificultad de la cadena secundaria con la de la cadena principal. Si la cadena secundaria tiene más dificultad acumulada que la cadena principal, el nodo *reconverge* en la cadena secundaria, lo que significa que se seleccionaría la cadena secundaria como su nueva cadena principal, haciendo que la cadena principal se convierta en una cadena secundaria. Si el nodo es un minero, a partir de entonces, construirá el próximo bloque como ampliación de este nueva y más larga cadena.

Si se recibe un bloque válido y no se encuentra ningún padre en las cadenas existentes, ese bloque se considera un "huérfano". Los bloques huérfanos se guardan en el pool de bloques huérfanos donde permanecerán hasta que se reciba a su padre. Una vez que se recibe el padre y se enlaza a las cadenas existentes, el huérfano se puede sacar del pool de huérfanos y enlazarlo al padre, haciéndolo parte de una cadena. Los bloques huérfanos suelen ocurrir cuando dos bloques que fueron minados en poco tiempo el uno del otro se reciben en orden inverso (hijo antes del padre).

Al seleccionar la cadena de mayor dificultad, todos los nodos finalmente logran el consenso de toda la red. Las discrepancias temporales entre las cadenas se resuelven con el tiempo a medida que se añade más prueba de trabajo, extendiendo una de las posibles cadenas. Los nodos de minería "votan" con su poder de minería para elegir qué cadena se extiende por la minería del siguiente bloque. Cuando mina un nuevo bloque y se extienda la cadena, ese nuevo bloque representa su voto.

En la siguiente sección vamos a ver cómo las discrepancias entre las cadenas que compiten entre sí (bifurcación) se resuelven mediante la selección independiente de la cadena de dificultad más larga.

Bifurcaciones de la Cadena de Bloques

Debido a que la cadena de bloques es una estructura de datos descentralizada, sus diferentes copias no siempre son coherentes. Los bloques podrían llegar a diferentes nodos en diferentes momentos, haciendo que los nodos tengan diferentes perspectivas de la cadena de bloques. Para resolver esto, cada nodo siempre selecciona e intenta extender la cadena de bloques que represente la mayor prueba de trabajo, también conocida como la cadena más larga o la cadena de mayor dificultad acumulada. Sumando la dificultad registrada en cada bloque en una cadena, un nodo puede calcular la cantidad total de prueba de trabajo que se ha gastado para crear esa cadena. Mientras todos los nodos seleccionen la cadena de dificultad acumulada más larga, la red bitcoin mundial finalmente converge a un estado coherente. Las bifurcaciones producen inconsistencias temporales entre versiones de la cadena de bloques, que finalmente se resuleven por reconvergencia a medida que se añaden más bloques a una de las bifurcaciones.

En los próximos diagramas, seguimos el progreso de un evento "bifurcación" en toda la red. El diagrama es una representación simplificada de bitcoin como una red global. En realidad, la topología de la red bitcoin no está organizada geográficamente. Más bien, se forma una red de malla de nodos interconectados, que podrían situarse muy lejos unos de otros geográficamente. La representación de una topología geográfica es una simplificación utilizada para ilustrar una bifurcación. En la red bitcoin real, la "distancia" entre nodos se mide en "saltos" de un nodo a otro, no mediante su ubicación física. Con fines ilustrativos, los diferentes bloques se muestran con diferentes colores, extendiéndose por la red y coloreando las conexiones que atraviesan.

En el primer diagrama (Visualización de un evento bifurcación de la cadena de bloques—antes de la bifurcación.), la red tiene una perspectiva unificada de la cadena de bloques, con el bloque azul como la punta de la cadena principal.

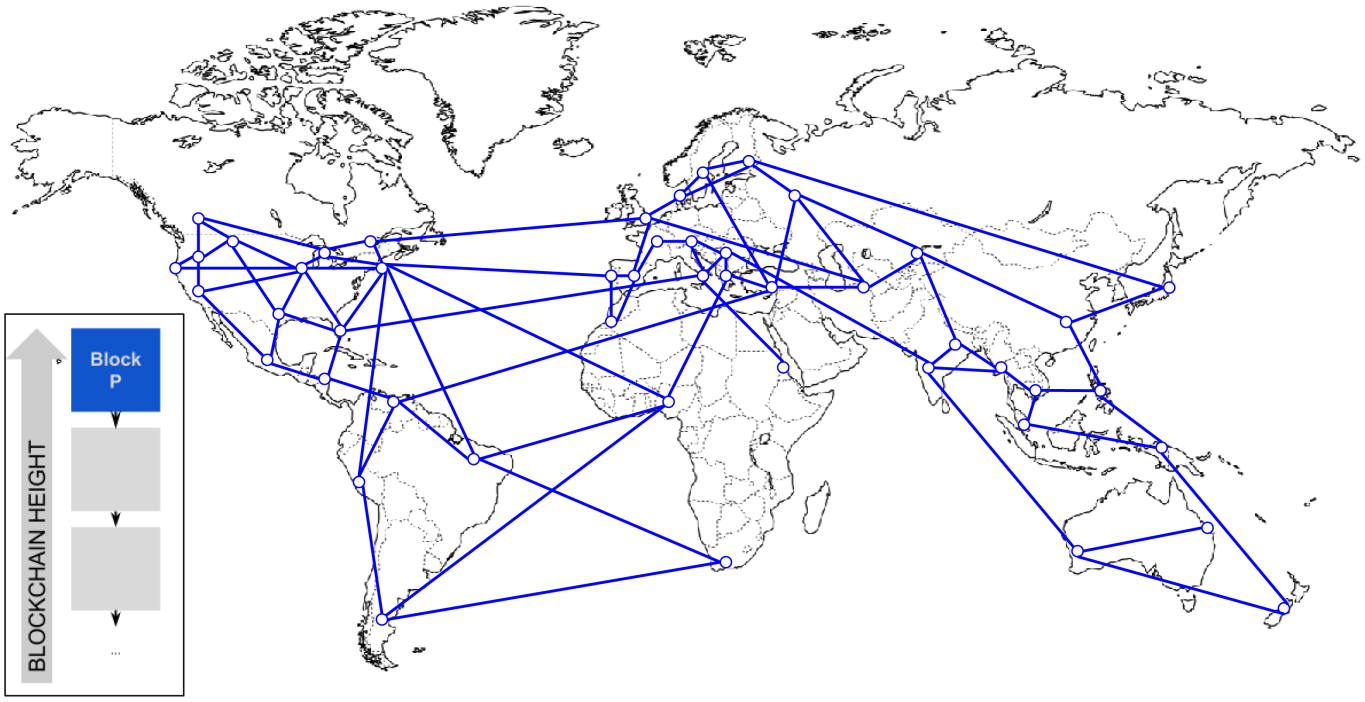


Figure 2. Visualización de un evento bifurcación de la cadena de bloques—antes de la bifurcación.

Una "bifurcación" se produce siempre que hay dos bloques candidatos que compiten para formar la cadena de bloques más larga. Esto ocurre en condiciones normales cuando dos mineros resuelven el algoritmo de prueba de trabajo dentro de un corto período de tiempo el uno del otro. Como los dos mineros descubren una solución para sus respectivos bloques candidatos, de inmediato emiten su propio bloque "vencedor" a sus vecinos inmediatos que comienzan a propagar el bloque a toda la red. Cada nodo incorporará en su cadena de bloques el bloque válido que haya recibido, extendiendo la cadena de bloques en un bloque. Si ese nodo más tarde ve otro bloque candidato extendiendo el mismo padre, conectará el segundo candidato en una cadena secundaria. Como resultado, algunos nodos "verán" un bloque candidato primero, mientras que otros nodos verán el otro bloque candidato, y así surgirán dos versiones rivales de la cadena de bloques.

En Visualización de un evento bifurcación en la cadena de bloques: se encuentran dos bloques simultáneamente, vemos a dos mineros que extraen dos bloques diferentes casi al mismo tiempo. Ambos bloques son hijos del bloque azul, creados con la intención de extender la cadena mediante la construcción en la parte superior del bloque azul. Para ayudarnos a rastrearlo, uno se visualiza como un bloque rojo procedente de Canadá, y el otro está marcado como un bloque verde procedente de Australia.

Supongamos, por ejemplo, que un minero en Canadá encuentra una solución de prueba de trabajo para un bloque "rojo" que extiende la cadena de bloques sobre la parte superior del bloque padre "azul." Casi al mismo tiempo, un minero australiano que también estaba extendiendo a partir del bloque "azul," encuentra una solución para el bloque "verde," su bloque candidato. Ahora, hay dos bloques posibles, que llamamos "rojo," originario de Canadá, y uno que llamamos "verde" originario de Australia. Ambos bloques son válidos, ambos bloques contienen una solución válida para la prueba de trabajo, y ambos bloques extienden el mismo padre. Ambos bloques probablemente contengan prácticamente las mismas transacciones, y quizás solamente algunas diferencias en el orden de las

transacciones.

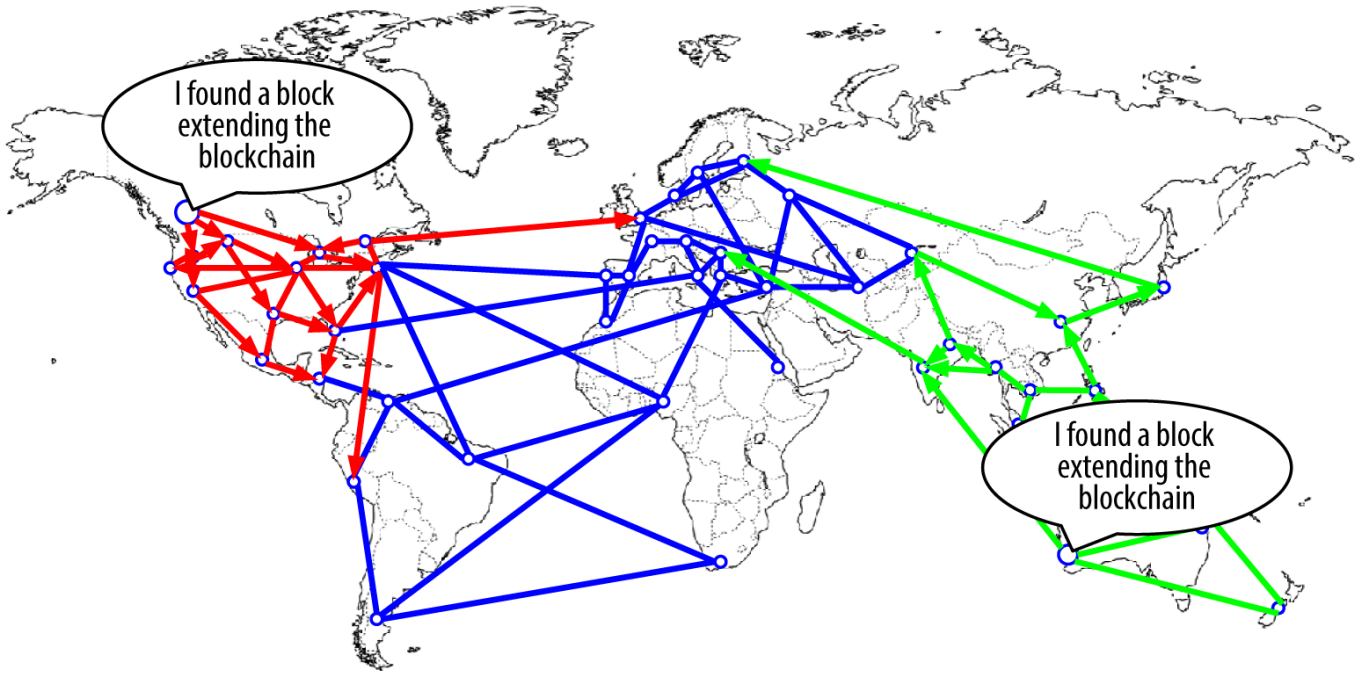


Figure 3. Visualización de un evento bifurcación en la cadena de bloques: se encuentran dos bloques simultáneamente

A medida que los dos bloques se propagan, algunos nodos reciben el bloque "rojo" primero y algunos reciben el bloque "verde" en primer lugar. Como se muestra en Visualización de un evento bifurcación en la cadena de bloques: se propagan dos bloques, seccionando la red, la red se secciona en dos perspectivas diferentes de la cadena de bloques, una con el bloque rojo en lo más alto, y la otra con el bloque verde.

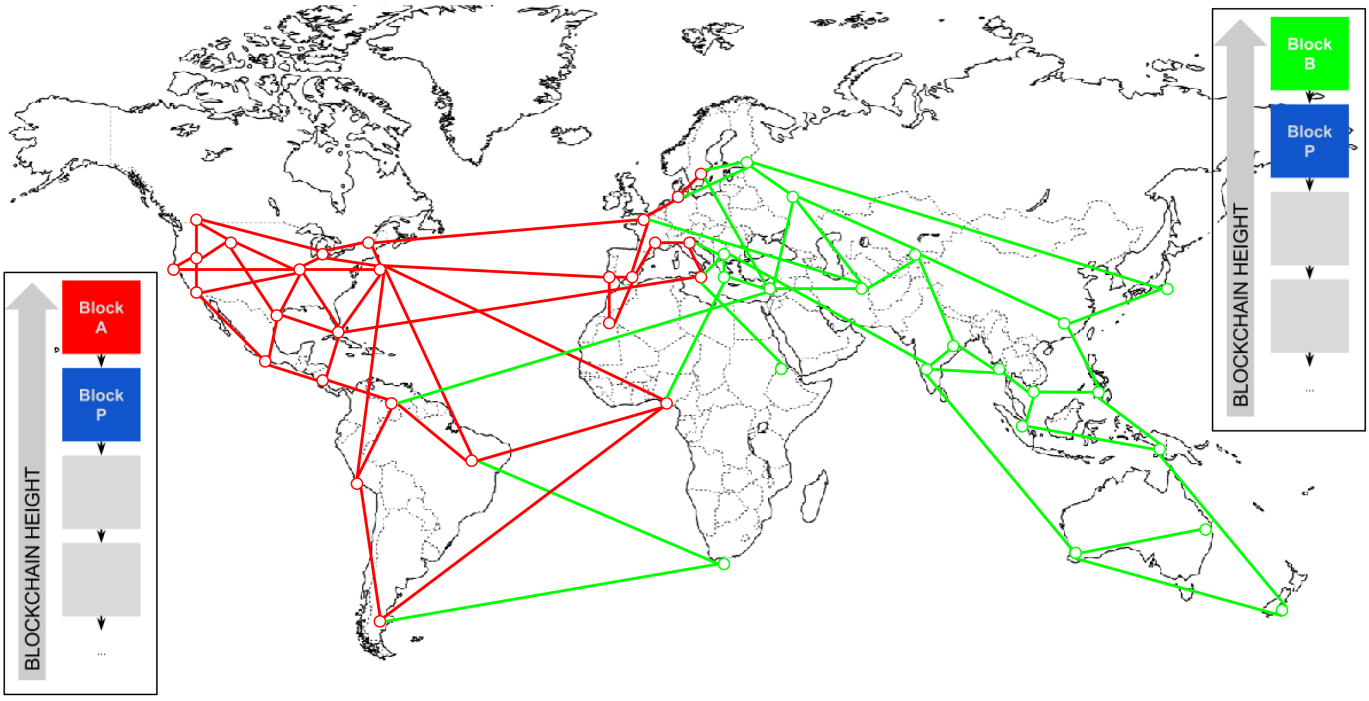


Figure 4. Visualización de un evento bifurcación en la cadena de bloques: se propagan dos bloques,

seccionando la red

A partir de ese momento, los nodos de la red bitcoin más cercanos (topológicamente, no geográficamente) al nodo canadiense oirán acerca del bloque "rojo" primero y crearán una cadena de bloques con la mayor dificultad acumulada con el bloque "rojo" como último bloque de la cadena (por ejemplo, azul-rojo), ignorando el bloque candidato "verde" que llega un poco más tarde. Mientras tanto, los nodos más cerca del nodo de Australia tomarán a ese bloque como el ganador y extenderán la cadena de bloques con el "verde" como el último bloque (por ejemplo, azul-verde), haciendo caso omiso del "rojo" cuando llega a los pocos segundos. Los mineros que hayan visto el "rojo" en primer lugar, inmediatamente construirán bloques candidatos que referencien al "rojo" como padre y empezarán a tratar de resolver la prueba de trabajo para estos bloques candidatos. Por contra, los mineros que aceptaron "verde" empezarán a construir por encima del"verde", extendiendo esa cadena.

Las bifurcaciones casi siempre se resuelven en el tiempo que se tarda en encontrar un bloque. Mientras parte de la potencia de hash de la red se dedica a la construcción con el "rojo" como padre, otra parte de la potencia de hash se dedica a la construcción por encima del "verde". Incluso si la potencia de hash se dividiera casi en partes iguales, es probable que un grupo de mineros encuentre una solución y la propague antes de que el otro grupo de mineros haya encontrado alguna solución. Digamos, por ejemplo, que los mineros que construyen en la parte superior del "verde" encuentran un nuevo bloque "rosa" que extiende la cadena (por ejemplo, azul, verde y rosa). Propagan inmediatamente este nuevo bloque y toda la red lo ve como una solución válida, como se muestra en Visualización de un evento bifurcación de la cadena de bloques: un nuevo bloque extiende una bifurcación.

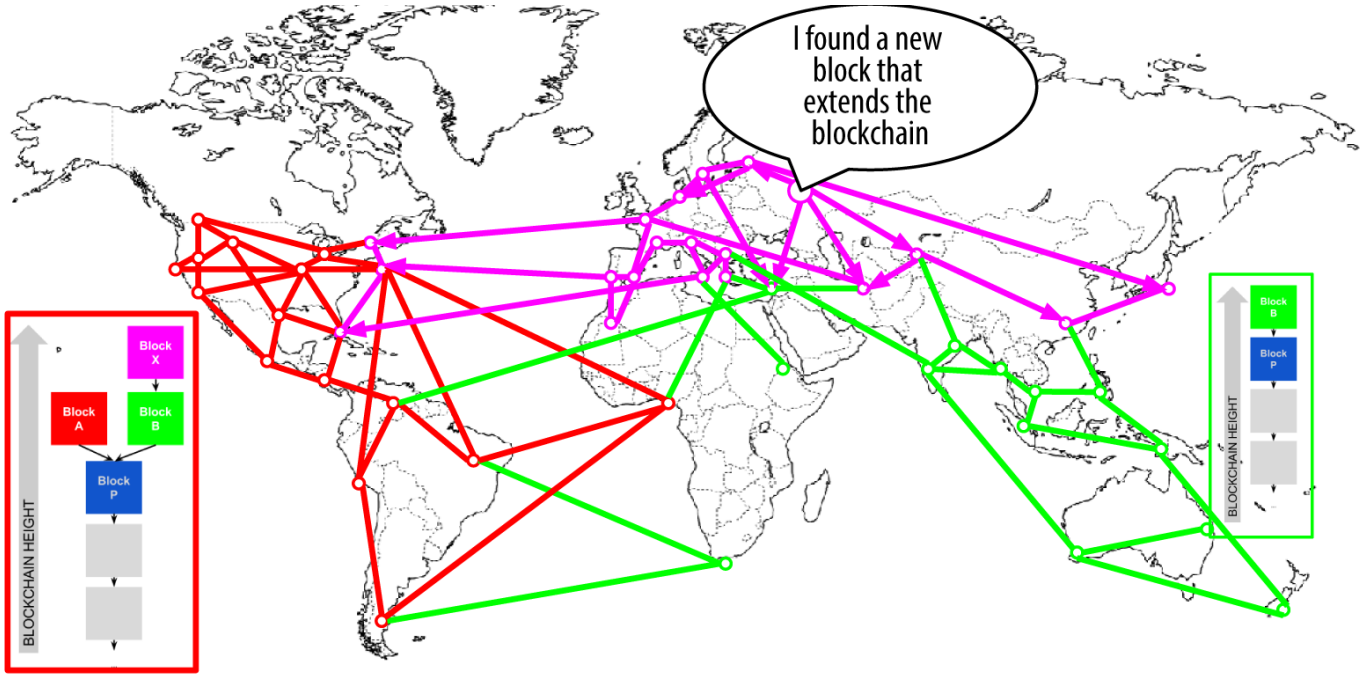


Figure 5. Visualización de un evento bifurcación de la cadena de bloques: un nuevo bloque extiende una bifurcación

Todos los nodos que habían elegido "verde" como el ganador en la ronda anterior simplemente extenderán la cadena un bloque más. Los nodos que eligieron "rojo" como el ganador, sin embargo,

ahora verán dos cadenas: azul-verde-rosa y azul-rojo. La cadena azul-verde-rosa es ahora más larga (mayor dificultad acumulada) que la cadena azul-rojo. Como resultado, esos nodos establecen la cadena azul-verde-rosa como la cadena principal y convierten la cadena azul-rojo en una cadena secundaria, como se muestra en Visualización de un evento bifurcación de la cadena de bloques: la red reconverge en una nueva cadena más larga. Esto es una reconvergencia de cadena, porque esos nodos se ven obligados a revisar su visión de la cadena de bloques para incorporar la nueva evidencia de una cadena más larga. Cualquier minero que trabaje en la extensión de la cadena azul-rojo ahora detendrá ese trabajo porque su bloque candidato es un "huérfano", ya que su padre, "rojo," ya no está en la cadena más larga. Las transacciones dentro de "rojo" se ponen en cola de nuevo para su procesamiento en el bloque siguiente, ya que el bloque ya no está en la cadena principal. La red entera re-converge en una sola cadena de bloques azul-verde-rosa, con "rosa" como el último bloque de la cadena. Todos los mineros comienzan a trabajar inmediatamente en bloques candidatos que hacen referencia a "rosa" como su padre para extender la cadena azul-verde-rosa.

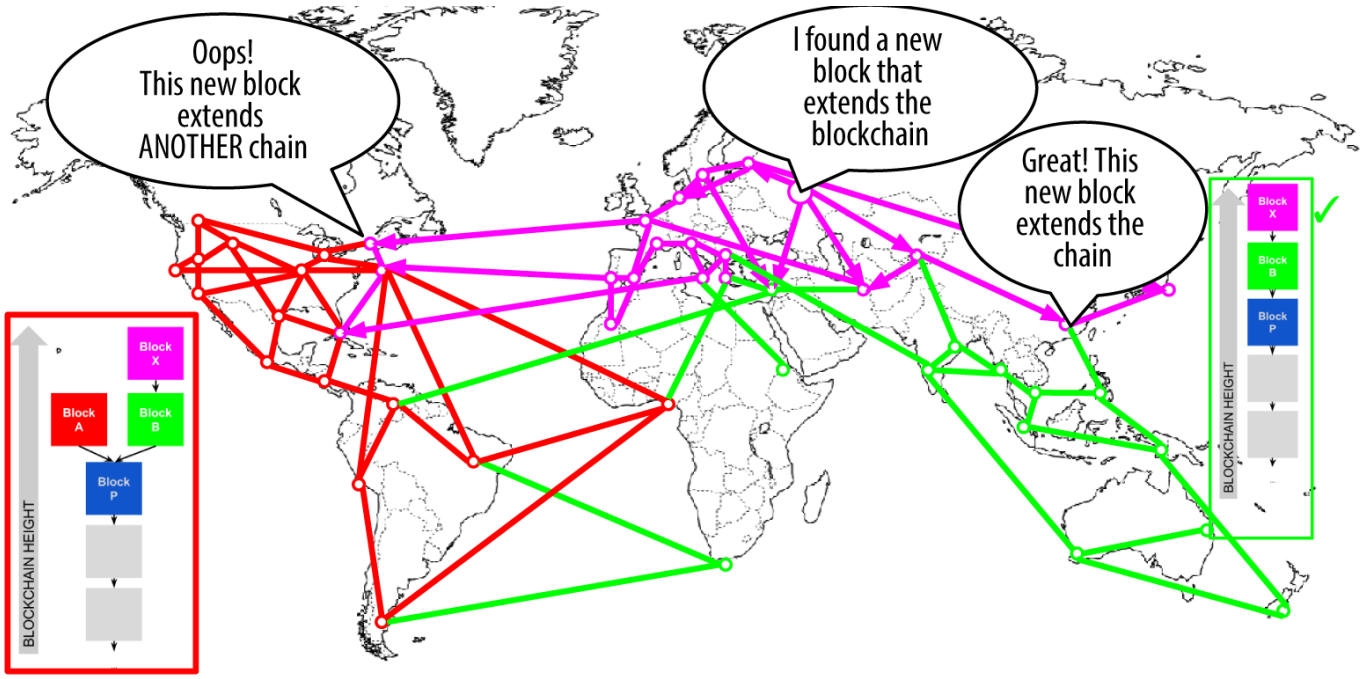


Figure 6. Visualización de un evento bifurcación de la cadena de bloques: la red reconverge en una nueva cadena más larga

Es teóricamente posible que una bifurcación se extienda a dos bloques, si los mineros encuentran casi al mismo tiempo dos bloques en "lados" opuestos de una bifurcación anterior. Sin embargo, la posibilidad de que eso ocurra es muy baja. Mientras que una bifurcación de un bloque puede ocurrir todas las semanas, una bifurcación de dos bloques es muy rara.

El intervalo de bloque de bitcoin de 10 minutos es un compromiso de diseño entre tiempos rápidos de confirmación (liquidación de las transacciones) y la probabilidad de una bifurcación. Un tiempo más rápido de bloque haría las transacciones claramente más rápidas, pero daría lugar a bifurcaciones más frecuentes de la cadena de bloques, mientras que un tiempo de bloque más lento disminuiría el número de bifurcaciones pero harían la liquidación más lenta.

Minería y la Carrera de Hashing

La minería bitcoin es una industria extremadamente competitiva. La potencia de hash se ha incrementado exponencialmente cada año de existencia de bitcoin. Algunos años el crecimiento ha reflejado un cambio completo de la tecnología, como en 2010 y 2011, cuando muchos mineros pasaron de utilizar la minería CPU a utilizar minería GPU y FPGA. En 2013 la introducción de la minería ASIC supuso otro paso de gigante en el poder minería, mediante la colocación de la función SHA256 directamente en chips de silicio especializados para el propósito de la minería. El primero de estos chips podía ofrecer más potencia de minería en una sola caja que toda la red bitcoin en 2010.

La siguiente lista muestra la potencia total de hash de la red bitcoin durante los primeros cinco años de operación:

2009

0.5 MH / seg-8 MH / seg (16#x00D7; crecimiento)

2010

8 MH / seg-116 GH / s (14500× crecimiento)

2011

16 GH/sec–9 TH/sec (562× crecimiento)

2012

9 TH/sec–23 TH/sec (2.5× crecimiento)

2013

23 TH/sec–10 PH/sec (450× crecimiento)

2014

10 PH/sec–150 PH/sec in August (15× crecimiento)

En el gráfico Potencia de hash total, gigahashes por segundo, durante dos años, vemos el aumento de potencia de hash de la red bitcoin en los últimos dos años. Como puede ver, la competencia entre los mineros y el crecimiento de bitcoin se ha traducido en un aumento exponencial de la potencia de hash (total de hashes por segundo a través de la red).

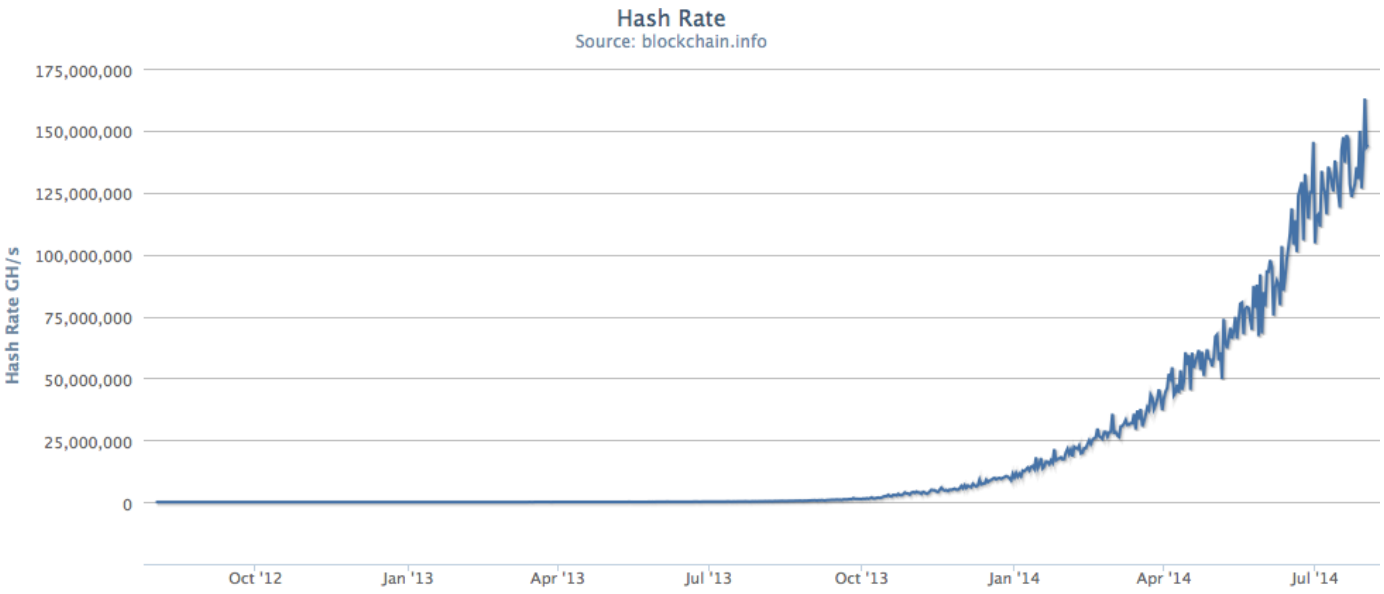


Figure 7. Potencia de hash total, gigahashes por segundo, durante dos años

La potencia de hash aplicada a la minería de bitcoin ha subido muchísimo, por lo que la dificultad ha aumentado proporcionalmente. La métrica de dificultad mostrada en [bitcoin_difficulty] se mide como una relación entre la dificultad actual y la dificultad mínima (la dificultad del primer bloque).

La métrica de dificultad para la minería de bitcoin, durante dos años image::images/msbt_0808.png["BitcoinDifficulty"]

En los últimos dos años, los chips de minería ASIC se han vuelto cada vez más densos, acercándose al límite de la fabricación de silicio con un tamaño de elemento (resolución) de 22 nanómetros (nm). Actualmente, los fabricantes de ASIC tienen el objetivo de superar a los fabricantes de chips de CPU de propósito general, con el diseño de chips con un tamaño de elemento de 16 nm, debido a que la rentabilidad de la minería está impulsando esta industria aún más rápido que la computación general. Ya no quedan más saltos de gigante en la minería bitcoin, porque la industria ha llegado a la vanguardia de la Ley de Moore, que establece que la densidad de computación se duplicará aproximadamente cada 18 meses. Sin embargo, la potencia de minería de la red sigue avanzando a un ritmo exponencial a medida que crece la carrera por chips de mayor densidad y por centros de datos también de mayor densidad, donde se pueden desplegar miles de estos chips. Ya no se trata de cuánto se puede minar con un chip, sino de cuántos chips se pueden incluir en un edificio, mientras se mantiener un entorno apropiado para disipar el calor y proporcionar la energía adecuada.

La Solución de Nonce Extra

Desde 2012, la minería de bitcoin ha evolucionado para resolver una limitación fundamental en la estructura de la cabecera de bloque. En los primeros días de bitcoin, un minero podía encontrar un bloque iterando a través del nonce hasta que el hash resultante fuera inferior al objetivo. A medida que la dificultad aumentaba, los mineros a menudo daban la vuelta por todos los 4 mil millones de valores del nonce sin encontrar un bloque. Sin embargo, esto se resolvió fácilmente mediante la modificación del sello de tiempo del bloque para tener en cuenta el tiempo que había transcurrido. Como el sello de tiempo es parte de la cabecera, el cambio permitiría a los mineros iterar a través de

los valores del nonce de nuevo y obtener resultados diferentes. Sin embargo, una vez que el hardware de minería superó los 4 GHash/seg, este enfoque se hizo cada vez más difícil ya que los valores del nonce se agotaban en menos de un segundo. A medida que los equipos de minería ASIC comenzaban a superar la velocidad de THash/seg, el software de minería necesitaba un espacio mayor en los valores nonce para encontrar bloques válidos. El sello de tiempo podría extenderse un poco, pero si se movía demasiado lejos en el tiempo, el bloque se convertiría en inválido. Se necesitaba una nueva fuente de "cambio" en la cabecera del bloque. La solución fue usar la transacción coinbase como una fuente de valores nonce adicionales. Debido a que el script coinbase puede almacenar entre 2 y 100 bytes de datos, los mineros comenzaron a utilizar ese espacio como espacio nonce extra, lo que les permitía explorar una gama mucho más amplia de valores de cabecera de bloque para encontrar bloques válidos. La transacción coinbase está incluida en el árbol merkle, lo que significa que cualquier cambio en la secuencia de comandos coinbase provoca que la raíz merkle cambie. Ocho bytes de nonce extra, además de los 4 bytes de nonce "estándar" permiten a los mineros explorar un total de 2^{96} (8 seguido de 28 ceros) posibilidades *por segundo* sin tener que modificar la fecha y hora. Si, en el futuro, los mineros agotaran todas estas posibilidades, entonces podrían modificar el sello de tiempo. También hay más espacio en el script coinbase para la futura expansión del espacio nonce extra.

Pools de Minería

En este entorno altamente competitivo, los mineros individuales que trabajan solos (también conocidos como mineros en solitario) no tienen ninguna oportunidad. La probabilidad de que encuentren un bloque para compensar sus costos de electricidad y hardware es tan baja que pasa a ser una apuesta, como jugar a la lotería. Incluso el sistema de minería de consumo ASIC más rápido no puede mantenerse al día con los sistemas comerciales que acumulan decenas de miles de estos chips en almacenes gigantes cerca de centrales hidroeléctricas. Los mineros ahora colaboran para formar pools de minerías, poniendo en común su potencia de hash y compartiendo la recompensa entre miles de participantes. Al participar en un pool, los mineros reciben una parte más pequeña de la recompensa total, pero por lo general se ven recompensados todos los días, lo que reduce la incertidumbre.

Veamos un ejemplo específico. Supongamos que un minero ha comprado hardware de minería con una potencia de hash total de 6000 gigahashes por segundo (GH/s), o 6 TH/s. En agosto de 2014 este equipo costaba aproximadamente $10.000. El hardware consume 3 kilovatios (kW) de electricidad cuando se ejecuta, 72 kilovatios-hora al día, a un costo de $7 o $8 por día en promedio. Con la dificultad bitcoin actual, el minero podrá minar en solitario un bloque cada 155 días aproximadamente, esto es, cada 5 meses. Si el minero encuentra un solo bloque en ese plazo, el pago de 25 bitcoins, a aproximadamente $600 por bitcoin, se traducirá en un solo pago de $15.000, que cubrirá todo el costo del hardware y de la electricidad consumida en ese período de tiempo, dejando un beneficio neto de aproximadamente $3.000. Sin embargo, la posibilidad de encontrar un bloque en un período de cinco meses depende de la suerte del minero. Podría darse el caso de que encontrara dos bloques en cinco meses, y obtener un gran beneficio. O podría no encontrar un bloque durante 10 meses y sufrir una pérdida financiera. Peor aún, con el crecimiento actual de la potencia de hashing, es probable que la dificultad del algoritmo de prueba de trabajo de bitcoin suba significativamente durante ese período, lo que significa que el minero tendría, como máximo, seis meses para amortizar el hardware antes de que quede obsoleto y deba ser reemplazado por hardware de minería más potente. Si este minero

participa en un pool de minería, en lugar de esperar a que una vez en cinco meses obtenga una ganancia inesperada de $15.000, ahora será capaz de ganar aproximadamente de $500 a $750 por semana. Los pagos regulares de un pool de minería le ayudarán a amortizar el costo del hardware y de la electricidad a lo largo del tiempo sin tomar un riesgo enorme. El hardware terminará siendo obsoleto al cabo de seis a nueve meses y el riesgo sigue siendo alto, pero el ingreso es por lo menos regular y fiable a lo largo de ese período.

Las pools de minería coordinan muchos cientos o miles de mineros, mediante protocolos especializados para pools de minería. Los mineros individuales configuran sus equipos de minería para conectarse a un servidor de pool, después de haber creado una cuenta con el pool. Su hardware de minería permanece conectado al servidor del pool mientras están minando, sincronizando así sus esfuerzos con los otros mineros. Por lo tanto, los mineros de pool comparten el esfuerzo para extraer un bloque y luego comparten las recompensas.

Los bloques exitosos pagan la recompensa a una dirección bitcoin del pool, en lugar de a los mineros individuales. El servidor del pool hará periódicamente pagos a las direcciones bitcoin de los mineros, cuando su parte de la recompensa haya llegado a un cierto umbral. Normalmente, el servidor de pool cobra una tarifa de porcentaje de los beneficios por la prestación del servicio de minería en pool.

Los mineros que participan en un pool dividen el trabajo de búsqueda de una solución para un bloque candidato, ganando "cuotas" (en inglés, "shares") por su contribución de minería. El pool de minería fija un objetivo de dificultad menor para ganar una cuota, por lo general más de 1000 veces más fácil que la dificultad de la red bitcoin. Cuando alguien en el pool mina con éxito un bloque, el pool gana la recompensa y luego la comparte con todos los mineros en proporción al número de cuotas con que contribuyeron a este esfuerzo.

Los pools están abiertos a cualquier minero, grande o pequeño, profesional o aficionado. Por lo tanto, un pool tendrá algunos participantes con una sola máquina pequeña minera, y otros con un garaje lleno de hardware de minería de gama alta. Algunos minarán con unas pocas decenas de kilovatios de electricidad, mientras que otros lo harán en un centro de datos consumiendo un megavatio de potencia. ¿Cómo mide un pool de minería las contribuciones individuales, con el fin de distribuir equitativamente los beneficios, sin la posibilidad de hacer trampa? La respuesta es utilizar el algoritmo de prueba de trabajo de bitcoin para medir la contribución de cada minero del pool, pero fijado a una dificultad menor para que incluso los mineros más pequeños del pool ganen cuotas con la suficiente frecuencia como para que les valga la pena contribuir al pool. Al establecer una dificultad menor para ganar cuotas, el pool mide la cantidad de trabajo realizado por cada minero. Cada vez que un minero del pool encuentra un hash de cabecera de bloque que es menor que la dificultad del pool, demuestra que ha hecho el trabajo de hashing para encontrar ese resultado. Más importante aún, el trabajo para encontrar cuotas contribuye, de manera estadísticamente medible, al esfuerzo global para encontrar un hash más bajo que el objetivo de la red bitcoin. Con el tiempo, los miles de mineros que intenten encontrar hashes de bajo valor van a encontrar uno lo suficientemente bajo como para satisfacer el objetivo de la red bitcoin.

Volvamos a la analogía de un juego de dados. Si los jugadores de dados están tirando los dados con el objetivo de lanzar menos de cuatro (la dificultad general de la red), un pool fijaría un objetivo más fácil, contando cuántas veces los jugadores del pool lograron tirar menos de ocho. Cuando los

jugadores del pool tiran menos de ocho (el objetivo de cuota del pool), ganan cuotas, pero no ganan el juego, ya que no alcanzan el objetivo del juego (menos de cuatro). Los jugadores del pool lograrán el objetivo del pool, que es más fácil, con mucha más frecuencia, ganando cuotas muy regularmente, incluso cuando no logran el objetivo más difícil de ganar el juego. De vez en cuando, uno de los jugadores del pool lanzará un tiro de dados combinado de menos de cuatro y en ese caso, el pool gana. Entonces, las ganancias se pueden distribuir a los jugadores del pool sobre la base de las cuotas que habían conseguido. A pesar de que el objetivo de ocho-o-menos no significaba ganar, era una forma razonable de medir el número de lanzamientos de dados de los jugadores, y que de vez en cuando produce un tiro de menos-que-cuatro.

Del mismo modo, un pool de minería establecerá una dificultad de pool que asegure que un minero individual del pool pueda encontrar bastante a menudo hashes de cabecera de bloque que sean menores que la dificultad del pool, ganando cuotas. De vez en cuando, uno de estos intentos producirá un hash de cabecera de bloque que sea menor que el objetivo de la red bitcoin, por lo que será un bloque válido y toda el pool gana.

Pools gestionados

La mayoría de los pools de minería están "gestionados", lo que significa que hay una empresa o individuo que administra un servidor del pool. El propietario del servidor del pool se llama el *operador del pool*, y cobra una tarifa de pool a los mineros según el porcentaje de las ganancias.

El servidor del pool ejecuta un software especializado y un protocolo de minería de pool que coordina las actividades de los mineros del pool. El servidor del pool también está conectado a uno o más nodos bitcoin completos y tiene acceso directo a una copia completa de la base de datos de la cadena de bloques. Esto permite que el servidor del pool pueda validar los bloques y las transacciones en nombre de los mineros del pool, liberándolos de la carga de ejecutar un nodo completo. Para los mineros del pool, esta es una consideración importante, porque un nodo completo requiere un ordenador dedicado con por lo menos 15 a 20 GB de almacenamiento permanente (disco) y al menos 2 GB de memoria (RAM). Además, el software de bitcoin que se ejecuta en el nodo completo necesita ser monitoreado, mantenido y actualizado con frecuencia. Cualquier tiempo de inactividad causado por la falta de mantenimiento o falta de recursos va a menguar la rentabilidad del minero. Para muchos mineros, la capacidad de minar sin ejecutar un nodo completo es otro gran beneficio de unirse a un pool gestionado.

Los mineros del pool se conectan al servidor del pool utilizando un protocolo de minería como Stratum (STM) o GetBlockTemplate (GBT) . Un estándar más antiguo llamado GetWork (GWK) ha quedado fundamentalmente obsoleto desde finales de 2012, ya que no admite fácilmente la minería a velocidades superiores a 4 GH/s. Tanto el protocolo STM como el GBT crean *plantillas de bloque* que contienen una plantilla de una cabecera de bloque candidato. El servidor del pool construye un bloque candidato mediante la agregación de las transacciones, la agregación de una transacción coinbase (con espacio de nonce extra), el cálculo de la raíz merkle, y el encadenamiento con el hash del bloque anterior. La cabecera del bloque candidato se envía entonces a cada uno de los mineros del pool como una plantilla. Después, cada minero del pool mina utilizando la plantilla de bloque, a una dificultad más baja que la dificultad de la red Bitcoin, y envía los resultados exitosos de nuevo al servidor del pool para ganar cuotas.

P2Pool

Los pools gestionados crean la posibilidad de que el operador del pool pueda hacer trampas dirigiendo el esfuerzo del pool para hacer doble gasto de transacciones o invalidar bloques (consulte Ataques de Consenso). Además, los servidores del pool centralizado representan un solo punto de fallo. Si el servidor del pool cae o se ralentiza por un ataque de denegación de servicio, los mineros del pool no pueden minar. En 2011, para resolver estos problemas de la centralización, se propone y se implementa un nuevo método de minería de pool: P2Pool es un pool de minería de igual a igual, sin operador central.

P2Pool descentraliza las funciones del servidor de pool, implementando un sistema similar a una cadena de bloques paralela que se llama *cadena de cuotas* (en inglés, "share chain"). Una cadena de cuotas es un cadena de bloques que funciona a una dificultad más baja que la cadena de bloques de bitcoin. La cadena de cuotas permite que los mineros del pool puedan colaborar en un pool descentralizado, minando cuotas en la cadena de cuotas a una velocidad de un bloque de cuota cada 30 segundos. Cada uno de los bloques en la cadena de cuotas registra una participación proporcional en la recompensa para los mineros del pool que contribuyen con trabajo, arrastrando las cuotas hacia adelante desde el bloque de cuota anterior. Cuando uno de los bloques de cuota alcanza también el objetivo de dificultad de la red bitcoin, se propaga y se incluye en la cadena de bloques de bitcoin, premiando a todos los mineros del pool que contribuyeron con todas las cuotas que precedieron al bloque de cuota ganador. En esencia, en vez de un servidor de pool que lleva el seguimiento de todas las cuotas y recompensas de los mineros del pool, la cadena de cuotas permite que todos los mineros del pool lleven el seguimiento de todas las cuotas utilizando un mecanismo de consenso descentralizado similar al mecanismo de consenso en la cadena de bloques de bitcoin.

La minería P2Pool es más compleja que la minería de pool, ya que requiere que los mineros del pool dispongan de un equipo dedicado con suficiente espacio en disco, memoria y ancho de banda de Internet para soportar un nodo bitcoin completo y el software del nodo P2Pool. Los mineros P2Pool conectan su hardware de minería a su nodo P2Pool local, que simula las funciones de un servidor de pool mediante el envío de las plantillas de bloque al hardware de minería. En P2Pool, los mineros de pool individuales construyen sus propios bloques candidatos, agregando las transacciones de forma similar a como lo hacen los mineros en solitario, pero después minan colaborativamente en la cadena de cuotas. P2Pool es un enfoque híbrido que tiene la ventaja de proporcionar pagos mucho más granulares que la minería en solitario, pero sin delegar demasiado control a un operador de pool como ocurre con los pools gestionados.

Recientemente, la participación en P2Pool ha aumentado significativamente a medida que la concentración de la minería en los pools de minería se ha acercado a los niveles que crean preocupación de un ataque del 51% (ver Ataques de Consenso). El desarrollo del protocolo P2Pool continúa con la expectativa de eliminar la necesidad de ejecutar un nodo completo y por lo tanto hacer que la minería descentralizada sea aún más fácil de usar.

Aunque P2Pool reduce la concentración de potencia por los operadores de pools de minería, existe la posibilidad de que sea vulnerable a los ataques del 51% contra la propia cadena de cuotas. Una adopción más amplia de P2Pool no resuelve el problema de ataque del 51% para bitcoin. Más bien, P2Pool hace a bitcoin más robusto, como parte de un ecosistema de minería diversificado.

Ataques de Consenso

El mecanismo de consenso de bitcoin es, al menos en teoría, vulnerable a los ataques de los mineros (o pools) que tratan de utilizar su potencia de hash para fines deshonestos o destructivos. Como vimos, el mecanismo de consenso depende de que una mayoría de los mineros actúen honestamente por su propio interés. Sin embargo, si un minero o un grupo de mineros pueden alcanzar una participación significativa en la potencia de minería, pueden atacar el mecanismo de consenso perturbando la seguridad y disponibilidad de la red bitcoin.

Es importante tener en cuenta que los ataques de consenso solo pueden afectar el consenso futuro, o a lo sumo, el pasado más reciente (decenas de bloques). El libro contable de bitcoin se vuelve más y más inmutable a medida que pasa el tiempo. Aunque en teoría, una bifurcación puede lograrse a cualquier profundidad, en la práctica, la potencia de cálculo necesaria para forzar una bifurcación muy profunda es inmensa, lo que hace a los bloques antiguos prácticamente inmutables. Los ataques de consenso tampoco afectan a la seguridad de la clave privada ni al algoritmo de firma (ECDSA). Un ataque de consenso no puede robar bitcoins, gastar bitcoins sin firmas, redirigir bitcoins, ni cambiar transacciones pasadas o registros de propiedad. Los ataques de consenso solo pueden afectar a los bloques más recientes y causar interrupciones de denegación de servicio en la creación de bloques futuros.

Un escenario de ataque contra el mecanismo de consenso se llama el "ataque del 51%." En este escenario un grupo de mineros, con el control de la mayoría (51%) de la potencia de hashing total de la red, conspira para atacar bitcoin. Con la capacidad de extraer la mayoría de los bloques, los mineros atacantes pueden causar "bifurcaciones" deliberadas en la cadena de bloques y hacer doble de gasto de transacciones o ejecutar ataques de denegación de servicio contra transacciones o direcciones específicas.En un ataque de bifurcación/doble gasto, el atacante invalida bloques que habían sido confirmados previamente, bifurcando por debajo de ellos y reconvergiendo en una cadena sustituta. Con la potencia suficiente, un atacante puede invalidar seis o más bloques de una sola vez, haciendo que las transacciones que se consideraban inmutables (seis confirmaciones) sean invalidadas. Tenga en cuenta que un doble gasto solo puede hacerse sobre las transacciones propias del atacante, para las que el atacante pueda producir una firma válida. Hacer doble gasto de las transacciones propias resulta rentable si invalidando una transacción el atacante puede obtener un pago irreversible de una casa de cambio o un producto sin tener que pagar por ello.

Examinemos un ejemplo práctico de un ataque del 51%. En el primer capítulo, nos fijamos en una transacción entre Alice y Bob para una taza de café. Bob, el dueño del café, está dispuesto a aceptar el pago de tazas de café sin esperar la confirmación (minado en un bloque), porque el riesgo de un doble gasto en una taza de café es baja en comparación con la comodidad de dar un servicio rápido al cliente. Esto es similar a la práctica de las tiendas de café que aceptan pagos con tarjeta de crédito sin una firma para montos inferiores a $25, porque el riesgo de una devolución de cargo de tarjeta de crédito es baja, mientras que el costo de retrasar la operación para obtener una firma es comparativamente mayor. Por el contrario, la venta de un artículo más caro con bitcoin corre el riesgo de un ataque de doble gasto, donde el comprador emite una transacción paralela que gasta las mismas entradas (UTXO) y cancela el pago al comerciante. Un ataque de doble gasto puede suceder de dos maneras: antes de que se confirme una transacción, o si el atacante se aprovecha de una bifurcación

de la cadena de bloques para deshacer varios bloques. Un ataque del 51% permite a los atacantes hacer doble gasto de sus propias transacciones en la nueva cadena, deshaciendo así la transacción correspondiente en la cadena antigua.

En nuestro ejemplo, el atacante malicioso Mallory va a la galería de Carol y compra una hermosa pintura tríptica que representa a Satoshi Nakamoto como Prometeo. Carol vende pinturas "El Gran Fuego" por $250.000 en bitcoin, a Mallory. En lugar de esperar seis o más confirmaciones sobre la transacción, Carol envuelve y entrega las pinturas a Mallory después de solo una confirmación. Mallory trabaja con un cómplice, Paul, que opera un gran pool de minería, y el cómplice lanza un ataque del 51% tan pronto como la transacción de Mallory se ha incluido en un bloque. Paul dirige el pool de minería para que vuelva a minar a la misma altura de bloque que el bloque que contiene la transacción de Mallory, sustituyendo el pago de Mallory a Carol con una transacción que haga doble gasto de la misma entrada que el pago de Mallory. La operación de doble gasto consume la misma UTXO y devuelve el pago a la cartera de Mallory, en lugar de pagar a Carol, esencialmente permitiendo a Mallory que conserve el bitcoin. Paul dirige entonces el pool de minería para que extraiga un bloque adicional, a fin de que la cadena que contiene la transacción de doble gasto sea más larga que la cadena original (causando una bifurcación por debajo del bloque que contiene la transacción de Mallory). Cuando la bifurcación de la cadena de bloques se resuelve en favor de la nueva cadena (más larga), la transacción de doble gastado reemplaza al pago original a Carol. Carol está perdiendo las tres pinturas y tampoco tiene ningún pago de bitcoin. A lo largo de toda esta actividad, los participantes del pool de minería de Paul podrían permanecer totalmente ignorantes de la tentativa de doble gasto, ya que minan con equipos de minería automatizados y no pueden supervisar cada transacción o bloque.

Para protegerse contra este tipo de ataques, un comerciante de artículos de venta de alto valor debe esperar por lo menos seis confirmaciones antes de dar el producto al comprador. Por otra parte, el comerciante debe utilizar un depósito (en inglés, "escrow") multifirma, esperando también varias confirmaciones después de que se haya provisto de fondos al depósito. Cuantas más confirmaciones transcurren, más difícil se hace invalidar una transacción con un ataque del 51%. Para los artículos de alto valor, el pago por bitcoin seguirá siendo conveniente y eficiente, incluso si el comprador tiene que esperar 24 horas para la entrega, lo que garantizaría 144 confirmaciones.

Además de un ataque de doble gasto, el otro escenario de un ataque de consenso es negar el servicio a participantes bitcoin específicos (direcciones específicas de bitcoin). Un atacante con una mayoría de la potencia de minería puede simplemente ignorar transacciones específicas. Si se incluyen en un bloque extraído por otro minero, el atacante puede bifurcar deliberadamente y volver a minar ese bloque, excluyendo de nuevo las transacciones específicas. Este tipo de ataque puede resultar en una denegación de servicio sostenida contra una dirección específica o un conjunto de direcciones durante el tiempo que el atacante controle la mayoría de la potencia de minería.

A pesar de su nombre, la posibilidad de un ataque del 51% en realidad no requiere el 51% de la potencia de hash. De hecho, este tipo de ataque se puede intentar con un porcentaje menor de la potencia de hash. El umbral del 51% es simplemente el nivel en el que el ataque es casi seguro que tenga éxito. Un ataque de consenso es esencialmente una lucha por el siguiente bloque y el grupo "más fuerte" es el que tiene más probabilidades de ganar. Con menor poder de hash, la probabilidad de éxito se reduce, debido a que otros mineros controlan la generación de algunos bloques con su potencia de

minería "honesta". Una forma de verlo es que cuanta más potencia de hash tenga un atacante, mayor será la longitud de la bifurcación que pueda crear deliberadamente, mayor será el número de bloques del pasado reciente que pueda invalidar, o mayor será el número de bloques en el futuro que pueda controlar. Existen grupos de investigación de seguridad que han utilizado modelos estadísticos para afirmar que es posible llevar a cabo diversos tipos de ataques de consenso con tan solo el 30% de la potencia de hash.

El aumento masivo de la potencia de hash total ha hecho que probablemente bitcoin sea inmune a los ataques de un solo minero. No hay manera posible de que un minero en solitario pueda controlar más de un pequeño porcentaje de la potencia de minería total. Sin embargo, la centralización del control causado por los pools de minería ha introducido el riesgo de ataques con afán de lucro por parte de un operador de pool de minería. El operador del pool en un pool gestionado controla la construcción de bloques candidatos y también controla qué transacciones se incluyen. Esto le da al operador del pool la facultad de excluir transacciones o de introducir transacciones de doble gasto. Si ese abuso de poder se hace de una manera limitada y sutil, un operador de pool posiblemente podría pasar desapercibido y beneficiarse de un ataque de consenso.

Sin embargo, no todos los atacantes estarán motivados por el lucro. Un escenario de posible ataque es donde un atacante tiene la intención de interrumpir la red bitcoin sin la posibilidad de beneficiarse de dichas perturbaciones. Un ataque malicioso con la intención de paralizar bitcoin requeriría enormes inversiones y planificación encubierta, pero posiblemente podría ser lanzado por un atacante bien financiado, probablemente patrocinado por el estado. Alternativamente, un atacante bien financiado podría atacar el consenso de bitcoin acumulando simultáneamente hardware de minería, comprometiendo los operadores de pool y atacando a otros pools con ataques de denegación de servicio. Todos estos escenarios son teóricamente posibles, pero cada vez más impracticables a medida que la potencia de hash total de la red bitcoin sigue creciendo exponencialmente.

Sin lugar a dudas, un ataque de consenso grave erosionaría la confianza en bitcoin en el corto plazo, causando posiblemente una disminución significativa del precio. Sin embargo, la red y el software bitcoin están en constante evolución, por lo que los ataques de consenso se enfrentarían con contramedidas inmediatas por la comunidad bitcoin, haciendo a bitcoin más resistente, más sigiloso, y más robusto que nunca.

Cadenas Alternativas, Monedas, <phrase role="keep-together">y Aplicaciones</phrase>

Bitcoin fue el resultado de 20 años de investigación en sistemas distribuidos y monedas y trajo una nueva tecnología revolucionaria en el espacio: el mecanismo de consenso descentralizado basado en la prueba de trabajo. Esta invención en el corazón del bitcoin, ha iniciado una ola de innovación en las monedas, los servicios financieros, la economía, los sistemas distribuidos, sistemas de votación, gobierno corporativo y contratos.

En este capítulo vamos a examinar los muchos descendientes de la invención del bitcoin y de la cadena de bloques: las cadenas alternativas, monedas y aplicaciones construidas desde la introducción de esta tecnología en el año 2009. En su mayoría, veremos las monedas alternativas, o *alt coins*, que son monedas digitales implementadas utilizando el mismo patrón de diseño que bitcoin, pero con una cadena de bloques y una red completamente independiente.

Por cada moneda alternativa mencionada en este capítulo, habrá otras 50 o más que no se van a mencionar aquí, generando aullidos de indignación por parte de sus creadores y aficionados. El propósito de este capítulo no es evaluar o clasificar monedas alternativas, ni tampoco mencionar las más significativas basadas en alguna evaluación subjetiva. En su lugar, vamos a destacar algunos ejemplos que muestran la amplitud y variedad de los ecosistemas, apuntando la primera de su tipo para cada innovación o diferenciación significativa. Algunos de los ejemplos más interesantes de monedas alternativas son de hecho un completo fracaso desde el punto de vista monetario. Eso tal vez los hace aún más interesantes para el estudio y destaca el hecho de que este capítulo no es para ser utilizado como una guía de inversión.

Con nuevas monedas introducidas cada día, sería imposible no perderse alguna moneda importante, tal vez la que cambie la historia. El ritmo de innovación es lo que hace de este espacio tan emocionante y garantiza que este capítulo será incompleto y fuera de fecha tan pronto como se publique.

Una Taxonomía de Monedas y Cadenas Alternativas

Bitcoin es un proyecto de código abierto, y su código se ha utilizado como base para muchos otros proyectos de software. La forma más común de software generado a partir de código fuente del bitcoin son monedas alternativas descentralizadas, o *alt coins*, que utilizan los mismos bloques de construcción básicos para implementar monedas digitales.

Hay una serie de capas de protocolo implementadas en la parte superior de la cadena de bloques de bitcoin. Estas *meta monedas*, *meta cadenas*, o *aplicaciones de cadena de bloques* utilizan la cadena de bloques como una plataforma de aplicaciones o para ampliar el protocolo bitcoin mediante la adición de capas de protocolo. Los ejemplos incluyen monedas de colores, Mastercoin, NXT y Counterparty.

En la siguiente sección vamos a examinar algunas monedas alternativas notables, como Litecoin, Dogecoin, Freicoin, Primecoin, Peercoin, Darkcoin y Zerocoin. Estas monedas alternativas son notables

por razones históricas o porque son buenos ejemplos de un tipo específico de innovación de moneda alternativa, no porque sean las más valiosas o las "mejores" monedas alternativas.

Además de las monedas alternativas, también hay una serie de implementaciones alternativas de la cadena de bloques que no son realmente "monedas", lo que yo llamo *cadenas alternativas*. Estas cadenas alternativas implementan un algoritmo de consenso y un libro contable distribuido como una plataforma para contratos, registro de nombres, u otras aplicaciones. Las cadenas alternativas utilizan los mismos bloques de construcción básicos y, a veces también usan una moneda o ficha como mecanismo de pago, pero su objetivo principal no es la moneda. Veremos Namecoin y Ethereum como ejemplos de cadenas alternativas.

Por último, hay un número de contendientes de bitcoin que ofrecen moneda digital o redes de pago digitales, pero sin necesidad de utilizar un libro contable descentralizado o mecanismo de consenso basado en prueba de trabajo, tales como Ripple y otros. Estas tecnologías que no utilizan cadena de bloques están fuera del alcance de este libro y no se tratan en este capítulo.

Plataformas Meta Moneda

Las meta monedas y las meta cadenas son capas de software implementadas en la parte superior de bitcoin, ya sea implementando una moneda-dentro-de una-moneda, o una plataforma/protocolo superpuesto dentro del sistema bitcoin. Estas capas de función extienden el núcleo del protocolo bitcoin y añaden características y capacidades mediante la codificación de datos adicionales dentro de las transacciones bitcoin y de las direcciones bitcoin. Las primeras implementaciones de meta monedas utilizaron varios remiendos para añadir metadatos a la cadena de bloques de bitcoin, tales como el uso de direcciones bitcoin para codificar datos o el uso de los campos no utilizados de transacción (por ejemplo, el campo de secuencia de la transacción) para codificar metadatos sobre la capa de protocolo añadida. Desde la introducción del código de operación de script de transacción OP_RETURN, las meta monedas han sido capaces de registrar los metadatos de forma más directa en la cadena de bloques, y la mayoría están migrando a utilizarse en su lugar.

Monedas de Color

Las *monedas de color* es un meta protocolo que superpone información sobre pequeñas cantidades de bitcoin. Una moneda "de color" es una cantidad de bitcoin reutilizada para expresar otro activo. Imagínese, por ejemplo, tomar un billete de $1, y poner un sello en él que diga: "Este es un certificado de 1 acción de Acme Inc." Ahora los $1 sirven para dos propósitos: se trata de un billete monetario y también de un certificado de acciones. Debido a que es más valioso como una acción, usted no querrá utilizarlo para comprar dulces, por lo que efectivamente ya no es útil como moneda. Las monedas de color funcionan de la misma manera mediante la conversión de una cantidad concreta muy pequeña de bitcoin, en un certificado comercial que representa otro activo. El término "color" se refiere a la idea de dar un significado especial a través de la adición de un atributo como un color—es una metáfora, no una asociación real a un color. No hay colores en las monedas de color.

Las monedas de color se gestionan con carteras especializadas que graban e interpretan los metadatos asociados a los bitcoins de color. Usando una cartera de ese tipo, el usuario utiliza una cantidad de

bitcoins, convirtiéndolas de monedas sin color a monedas de color mediante la incorporación de una etiqueta que tiene un significado especial. Por ejemplo, una etiqueta podría representar certificados de acciones, cupones, bienes inmuebles, materias primas, o fichas coleccionables. La asignación e interpretación del significado del "color" asociado con cada moneda específica depende totalmente de los usuarios de las monedas de color. Para colorear las monedas, el usuario define los metadatos asociados, tales como el tipo de emisión, si se puede subdividir en unidades más pequeñas, un símbolo y descripción, y otra información relacionada. Una vez coloreadas, estas monedas pueden comprarse y venderse, subdividirse y agregarse, y recibir pagos de dividendos. Las monedas de color también pueden ser "sin color" mediante la eliminación de la asociación especial y canjeadas por su valor nominal en bitcoin.

Para demostrar el uso de monedas de color, hemos creado un conjunto de 20 monedas de color con el símbolo "MasterBTC" que representan cupones para una copia gratuita de este libro, tal como se muestra en El perfil de metadatos de las monedas de color registrado como un cupón para una copia gratuita de un libro. Cada unidad de MasterBTC, representada por estas monedas de color, ahora se puede vender o dar a cualquier usuario de bitcoin con una cartera de adaptada a monedas de color, que luego los puede transferir a otros o canjearlos con el emisor para obtener una copia gratuita del libro . Este ejemplo de monedas de colores puede verse aquí.

Example 1. El perfil de metadatos de las monedas de color registrado como un cupón para una copia gratuita de un libro

```
{
  "source_addresses": [
    "3NpZmvSPLmN2cVFw1pY7gxEAVPCVfnWfVD"
  ],
  "contract_url":
"https://www.coinprism.info/asset/3NpZmvSPLmN2cVFw1pY7gxEAVPCVfnWfVD",
  "name_short": "MasterBTC",
  "name": "Free copy of \"Mastering Bitcoin\"",
  "issuer": "Andreas M. Antonopoulos",
  "description": "This token is redeemable for a free copy of the book \"Mastering
Bitcoin\"",
  "description_mime": "text/x-markdown; charset=UTF-8",
  "type": "Other",
  "divisibility": 0,
  "link_to_website": false,
  "icon_url": null,
  "image_url": null,
  "version": "1.0"
}
```

Mastercoin

Mastercoin es una capa de protocolo en la parte superior de bitcoin que soporta una plataforma para diversas aplicaciones que extienden al sistema bitcoin. Mastercoin utiliza la moneda MST como una ficha para realizar transacciones Mastercoin pero no es fundamentalmente una moneda. Más bien, es una plataforma para la construcción de otras cosas, tales como monedas de usuario, fichas de propiedad inteligentes, intercambios descentralizados de activos y contratos. Piense en Mastercoin como un protocolo de capa de aplicación en la parte superior de la capa de transporte de las transacciones financieras de bitcoin, de la misma forma que HTTP se ejecuta sobre TCP.

Mastercoin opera principalmente a través de transacciones enviadas hacia y desde una dirección bitcoin especial llamada la dirección "éxodo" (1EXoDusjGwvnjZUyKkxZ4UHEf77z6A5S4P), al igual que HTTP utiliza un puerto TCP específico (puerto 80) para diferenciar su tráfico del resto del tráfico TCP. El protocolo Mastercoin está haciendo una transición gradual, dejando atrás el uso de la dirección de éxodo especializada y las múltifirmas, y pasando a utilizar el operador bitcoin OP_RETURN para codificar metadatos de transacción.

Counterparty

Counterparty es otra capa de protocolo que se implementa en la parte superior de bitcoin. Counterparty permite monedas de usuario, fichas negociables, instrumentos financieros, los intercambios de activos descentralizados, y otras características. Counterparty se implementa utilizando principalmente el operador OP_RETURN en el lenguaje de scripting de bitcoin para grabar metadatos que mejoren las transacciones bitcoin con significado adicional. Counterparty usa la moneda XCP como ficha para la realización de transacciones de Counterparty.

Monedas Alternativas

La gran mayoría de las monedas alternativas se derivan del código fuente de bitcoin, y también se las conoce como "bifurcaciones" (en inglés, "forks"). Algunas se implementan "desde cero", basándose en el modelo de cadena de bloques pero sin utilizar nada del código fuente de bitcoin. Las monedas alternativas y las cadenas alternativas (en la siguiente sección) son dos implementaciones independientes de la tecnología de la cadena de bloques y ambas utilizan su propia cadena de bloques. La diferencia en los términos es para indicar que las monedas alternativas se utilizan principalmente como moneda, mientras que las cadenas alternativas se utilizan para otros fines, fundamentalmente no como moneda.

Estrictamente hablando, la primera gran bifurcación alternativa del código de bitcoin no fue una moneda alternativa sino la cadena alternativa *Namecoin*, que vamos a discutir en la próxima sección.

Sobre la base de la fecha del anuncio, la primera moneda alternativa como bifurcación de bitcoin apareció en agosto de 2011; se llamaba *IXCoin*. IXCoin modificaba algunos de los parámetros bitcoin, acelerando específicamente la creación de moneda mediante el aumento de la recompensa a 96 monedas por bloque.

En septiembre de 2011, se puso en marcha *Tenebrix*. Tenebrix fue la primera criptomoneda en implementar un algoritmo alternativo de prueba de trabajo, llamado *scrypt*, un algoritmo original diseñado para extensión de contraseñas (resistencia a fuerza bruta). El objetivo declarado de Tenebrix era hacer una moneda que fuera resistente a la minería con las GPU y ASIC, mediante el uso de un algoritmo que fuera intensivo en memoria. Tenebrix no tuvo éxito como moneda, pero fue la base para Litecoin, que ha gozado de gran éxito y ha generado cientos de clones.

Litecoin, además de utilizar scrypt como el algoritmo de prueba de trabajo, también implementó un tiempo de generación de bloque más rápido, dirigido a 2,5 minutos en lugar de los 10 minutos de bitcoin. La moneda resultante se promociona como "la plata tras el oro de bitcoin" y pretende ser una moneda alternativa ligera. Debido al tiempo de confirmación más rápido y el límite total de 84 millones de monedas, muchos partidarios de Litecoin creen que es más adecuado que bitcoin para las transacciones comerciales.

Las monedas alternativas continuaron proliferando en 2011 y 2012, ya sea basadas en bitcoin o en Litecoin. Para 2013, había 20 monedas alternativas compitiendo por posición en el mercado. A finales de 2013, este número había explotado a 200, convirtiéndose 2013 en el "año de las monedas alternativas." El crecimiento de las monedas alternativas continuó en 2014, con la existencia de más de 500 monedas alternativas en el momento de escribir esto. Más de la mitad de las monedas alternativas hoy son clones de Litecoin.

La creación de una moneda alternativa es fácil, por lo que ahora hay más de 500 de ellas. La mayor parte de las monedas alternativas difieren muy poco de bitcoin y no ofrecen nada digno de estudio. Muchas de ellas son de hecho, solo intentos para enriquecer a sus creadores. Entre imitadores y estrategias de pump-and-dump, hay sin embargo, algunas excepciones notables e innovaciones muy importantes. Estas monedas alternativas toman enfoques radicalmente diferentes o añaden innovación significativa al patrón de diseño de bitcoin. Hay tres áreas principales donde estas monedas alternativas se diferencian de bitcoin:

- Distinta política monetaria
- Distinto mecanismo de prueba de trabajo o consenso
- Características específicas, como anonimato fuerte

Para obtener más información, consulte esta cronología gráfica de monedas alternativas y cadenas alternativas.

Evaluando una Moneda Alternativa

Con tantas monedas alternativas por ahí, ¿cómo decidir cuáles son dignas de atención? Algunas monedas alternativas intentan lograr una amplia distribución y su uso como monedas. Otras son laboratorios para experimentar con diferentes características y modelos monetarios. Muchas son simplemente maniobras por parte de sus creadores para hacerse ricos rápidamente. Para evaluar las monedas alternativas, evalúo sus características definitorias y sus métricas de mercado.

Aquí hay algunas preguntas para considerar hasta qué punto una moneda alternativa se diferencia de

bitcoin:

- ¿La moneda alternativa presenta una innovación significativa?
- ¿Es la diferencia suficientemente persuasiva como para atraer usuarios de bitcoin?
- ¿La moneda alternativa apunta a un nicho de mercado o aplicación interesante?
- ¿Puede la moneda alternativa atraer suficientes mineros para estar segura contra ataques de consenso?

Aquí hay algunas de las diferencias financieras clave y métricas de mercado a considerar:

- ¿Cuál es la capitalización de mercado total de la moneda alternativa?
- ¿Cuántos usuarios/carteras estimados tiene la moneda alternativa?
- ¿Cuántos comerciantes aceptan la moneda alternativa?
- ¿Cuántas transacciones diarias (volumen) son ejecutadas en la moneda alternativa?
- ¿Cuánto valor se mueve diariamente?

En este capítulo nos concentraremos principalmente en las características técnicas y potencial de innovación de las monedas alternativas representadas por el primer juego de preguntas.

Parámetros Monetarios Alternativos: Litecoin, Dogecoin, Freicoin

Bitcoin tiene algunos parámetros monetarios que le dan características distintivas de una moneda deflacionaria de emisión fija. Está limitado a 21 millones de unidades mayores de moneda (o 2100 billones de unidades menores), tiene una velocidad de emisión que decrece geométricamente, y un "latido" de 10 minutos por bloque, que controla la velocidad de confirmación de la transacción y la generación de moneda. Muchas monedas alternativas han ajustado los parámetros principales para lograr diferentes políticas monetarias. Entre los cientos de monedas alternativas, algunos de los ejemplos más notables incluyen los siguientes.

Litecoin

Una de las primeras monedas alternativas, lanzada en 2011, Litecoin es la segunda moneda digital más exitosa después bitcoin. Sus innovaciones principales fueron el uso de *scrypt* como el algoritmo de prueba de trabajo (heredado de Tenebrix) y sus parámetros de moneda más rápidos/ligeros.

- Tiempo de generación de bloque: 2,5 minutos
- Moneda total: 84 millones de monedas hacia 2140
- Algoritmo de consenso: Prueba de trabajo Scrypt
- Capitalización de mercado: $160 millones a mediados de 2014

Dogecoin

Dogecoin fue lanzado en diciembre de 2013, como bifurcación de Litecoin. Dogecoin es notable, ya que tiene una política monetaria de emisión rápida y una cima de moneda muy alta, para fomentar el gasto y el depósito. Dogecoin también es notable porque se inició como una broma, pero se hizo muy popular, con una comunidad grande y activa, antes de disminuir rápidamente en 2014.

- Tiempo de generación de bloque: 60 segundos
- Moneda total: 100.000.000.000 (100 mil millones) de Doges hacia 2015
- Algoritmo de consenso: Prueba de trabajo Scrypt
- Capitalización de mercado: $12 millones a mediados de 2014

Freicoin

Freicoin se introdujo en julio de 2012. Se trata de una *moneda con oxidación*, lo que significa que ofrece un tipo de interés negativo para el valor almacenado. Al valor almacenado en Freicoin se añade una comisión del 4,5% TAE, para fomentar el consumo y desalentar el acaparamiento de dinero. Freicoin es notable ya que implementa una política monetaria que es opuesta a la política deflacionaria de Bitcoin. Freicoin no ha tenido éxito como moneda, pero es un ejemplo interesante de la variedad de políticas monetarias que se pueden expresar con las monedas alternativas.

- Generación de bloque: 10 minutos
- Moneda total: 100 millones de monedas hacia 2140
- Algoritmo de consenso: Prueba de trabajo SHA256
- Capitalización de mercado: $130.000 a mediados de 2014

Innovación de Consenso: Peercoin, Myriad, Blackcoin, Vericoin, NXT

Los mecanismos de consenso de Bitcoin se basan en la prueba de trabajo utilizando el algoritmo SHA256. Las primeras monedas alternativas comenzaron a usar scrypt como algoritmo alternativo de prueba de trabajo, como una forma de hacer minería más adaptada a CPU y menos susceptible a la centralización con ASICs. Desde entonces, la innovación en el mecanismo de consenso ha continuado a un ritmo frenético. Varias monedas alternativas adoptaron una variedad de algoritmos como scrypt, scrypt-N, Skein, Groestl, SHA3, X11, Blake, y otros. Algunas monedas alternativas combinan múltiples algoritmos para la prueba de trabajo. En 2013, vimos la invención de una alternativa a la prueba de trabajo, llamada *prueba de participación*, que forma la base de muchas monedas alternativas modernas.

La prueba de la participación es un sistema en el que los propietarios de una moneda pueden poner en "juego" la moneda como una fianza que devenga intereses. Similar a un certificado de depósito (CD), los participantes pueden reservar una parte de sus reservas de moneda, al mismo tiempo que ganan un retorno de inversión en forma de nueva moneda (entregada como pago de intereses) y comisiones por transacción.

Peercoin

Peercoin se introdujo en agosto de 2012 y es la primera moneda alternativa en utilizar un algoritmo híbrido de prueba de trabajo y prueba de participación para emitir nueva moneda.

- Generación de bloque: 10 minutos
- Total de moneda: Sin límite
- Algoritmo de consenso: (Híbrido) prueba de participación con prueba de trabajo inicial Capitalización de mercado: $14 millones a mediados de 2014

Myriad

Myriad se introdujo en febrero de 2014 y es notable, ya que utiliza cinco diferentes algoritmos de prueba de trabajo (SHA256d, Scrypt, Qubit, Skein, o Myriad-Groestl) simultáneamente, con dificultad variable para cada algoritmo en función de la aportación en la minería. La intención es hacer Myriad inmune a la especialización y centralización de los ASIC, así como mucho más resistente a los ataques de consenso, porque tendrían que ser atacados simultáneamente múltiples algoritmos de minería.

- Generación de bloque: 30 segundos promedio (objetivo de 2,5 minutos por algoritmo de minería)
- Moneda total: 2 mil millones para 2024
- Algoritmo de consenso: Prueba de trabajo multi-algorítimica
- Capitalización de mercado: $120.000 a mediados de 2014

Blackcoin

Blackcoin se introdujo en febrero de 2014 y utiliza un algoritmo de consenso de prueba de participación. También es notable por introducir "multiPools", un tipo de pool de minería que puede cambiar entre diferentes monedas alternativas automáticamente, en función de la rentabilidad.

- Generación de bloque: 1 minuto
- Total de moneda: Sin límite
- Algoritmo de consenso: Prueba de participación
- Capitalización de mercado: $3,7 millones a mediados de 2014

VeriCoin

VeriCoin fue lanzado en mayo de 2014. Utiliza un algoritmo de consenso de prueba de participación con una tasa de interés variable que se ajusta dinámicamente basada en las fuerzas del mercado de la oferta y la demanda. También es la primera moneda alternativa que ofrece intercambio automático a bitcoin desde la cartera para el pago en bitcoin.

- Generación de bloque: 1 minuto
- Total de moneda: Sin límite

- Algoritmo de consenso: Prueba de participación
- Capitalización de mercado: $1,1 millones a mediados de 2014

NXT

NXT (pronunciado "Next") es una moneda alternativa "pura" de prueba de participación, ya que no utiliza la minería de prueba de trabajo. NXT es una implementación de una criptomoneda desde cero, no una bifurcación de bitcoin o cualquier otra moneda alternativa. NXT implementa muchas características avanzadas, incluyendo un registro de nombres (similar a Namecoin), un intercambio descentralizado de activos (similar a monedas de color), mensajería integrada descentralizada y segura (similar a Bitmessage), y delegación de participación (delegar la prueba de participación a otros). Los seguidores de NXT la llaman criptomoneda de "próxima generación" o criptomoneda 2.0.

- Generación de bloque: 1 minuto
- Total de moneda: Sin límite
- Algoritmo de consenso: Prueba de participación
- Capitalización de mercado: $30 millones a mediados de 2014

Innovación en Minado de Doble Propósito: Primecoin, Curecoin, Gridcoin

El algoritmo de prueba de trabajo de bitcoin tiene un único propósito: asegurar la red bitcoin. En comparación con la seguridad de un sistema de pago tradicional, el costo de la minería no es muy alto. Sin embargo, ha sido criticado por muchos como "un desperdicio." La siguiente generación de monedas alternativas intenta abordar esta preocupación. Los algoritmos de prueba de trabajo de doble propósito resuelven un problema "útil" específico, mientras producen la prueba de trabajo para asegurar la red. El riesgo de añadir un uso externo a la seguridad de la moneda es que también añade influencia externa a la curva de oferta/demanda.

Primecoin

Primecoin se anunció en julio de 2013. Su algoritmo de prueba de trabajo se basa en la búsqueda de números primos, calculando cadenas de números primos Cunningham y bi-gemelos. Los números primos son útiles en varias disciplinas científicas. La cadena de bloques de Primecoin contiene los números primos descubiertos, produciendo de esta manera un registro público de los descubrimientos científicos de manera simultánea al libro contable público de las transacciones.

- Generación de bloque: 1 minuto
- Total de moneda: Sin límite
- Algoritmo de consenso: Prueba de trabajo con descubrimiento de cadena de números primos
- Capitalización de mercado: $1,3 millones a mediados de 2014

Curecoin

Curecoin fue anunciado en mayo de 2013. Combina un algoritmo SHA256 de prueba de trabajo con la

investigación de plegamiento de proteínas a través del proyecto Folding@Home. El plegamiento de proteínas es una simulación de cómputo intensivo de las interacciones bioquímicas de las proteínas, que se utiliza para descubrir nuevas objetivos de medicamentos para curar enfermedades.

- Generación de bloque: 10 minutos
- Total de moneda: Sin límite
- Algoritmo de consenso: Prueba de trabajo con investigación de plegamiento de proteínas
- Capitalización de mercado: $58.000 a mediados de 2014

Gridcoin

Gridcoin se introdujo en octubre de 2013. Complementa la prueba de trabajo basada en scrypt con subvenciones por participar en la computación abierta en red BOINC. BOINC—Berkeley Open Infrastructure for Network Computing (Infraestructura Abierta de Berkeley para la Computación en Red)— es un protocolo abierto para la computación en red para investigaciones científicas, que permite a los participantes compartir sus ciclos de cómputo desocupados para una amplia gama de cálculos en la investigación académica. Gridcoin utiliza BOINC como una plataforma de computación de propósito general, en lugar de resolver los problemas específicos de la ciencia, como los números primos o el plegamiento de proteínas.

- Generación de bloque: 150 segundos
- Total de moneda: Sin límite
- Algoritmo de Consenso: Prueba de trabajo con subsidio por computación en la red BOINC
- Capitalización de mercado: $122.000 a mediados de 2014

Monedas Alternativas Enfocadas hacia el Anonimato: CryptoNote, Bytecoin, Monero, Zerocash/Zerocoin, Darkcoin

"monedas alternativas","enfocadas hacia el anonimato", id="ix_ch09-asciidoc3", range="startofrange")Bitcoin es a menudo erróneamente caracterizada como una moneda "anónima". De hecho, es relativamente fácil conectar identidades con direcciones bitcoin y, usando análisis big-data, conectar direcciones entre sí para formar una imagen completa de los hábitos de consumo de bitcoin de alguien. Varias monedas alternativas intentan tratar este tema directamente al centrarse en un fuerte anonimato. El primer intento de este tipo probablemente sea *Zerocoin*, un protocolo de meta-moneda para preservar el anonimato en la parte superior del bitcoin, presentado en un artículo en el 2013 IEEE Symposium sobre Seguridad y Privacidad. Zerocoin será implementada como una moneda alternativa completamente separada llamada Zerocash, en desarrollo en el momento de escribir este texto. Un enfoque alternativo al anonimato fue lanzado con *CryptoNote* en un artículo publicado en octubre de 2013. CryptoNote es una tecnología que sirve de base a numerosas bifurcaciones de monedas alternativas que se discutirán a continuación. Además de Zerocash y CryptoNotes, hay otras monedas anónimas independientes, tales como Darkcoin, que utilizan direcciones de sigilo o transacciones de re-mezcla para proporcionar el anonimato.

Zerocoin/Zerocash

Zerocoin es una aproximación teórica al anonimato de moneda digital introducida en 2013 por investigadores de la Johns Hopkins. La moneda alternativa Zerocash es una implementación de Zerocoin aún en desarrollo y pendiente de publicar.

CryptoNote

CryptoNote es una implementación de referencia de moneda alternativa que proporciona la base para el dinero digital anónimo. Fue introducida en octubre de 2013. Está diseñada para ser bifurcada en diferentes implementaciones y tiene incorporado un mecanismo de reseteo periódico que la hace inutilizable como moneda propia. Varias monedas alternativas se han generado a partir de CryptoNote, incluyendo Bytecoin (BCN), Aeon (AEON), Boolberry (BBR), duckNote (DUCK), Fantomcoin (FCN), Monero (XMR), MonetaVerde (MCN) y Quazarcoin (QCN) . CryptoNote también es notable por ser una implementación completamente nueva de criptomoneda, no una bifurcación de bitcoin.

Bytecoin

Bytecoin fue la primera implementación generada a partir de CryptoNote, ofreciendo una moneda anónima viable basada en la tecnología CryptoNote. Bytecoin fue lanzada en julio de 2012. Tenga en cuenta que había una moneda alternativa anterior llamada Bytecoin con el símbolo de moneda BTE, mientras que el Bytecoin derivado de CryptoNote tiene el símbolo de moneda BCN. Bytecoin utiliza el algoritmo de prueba de trabajo Cryptonight, que requiere el acceso a por lo menos 2 MB de RAM por instancia, por lo que es inadecuado para la minería por GPU y ASIC. Bytecoin hereda de CryptoNote las firmas en anillo, las transacciones no vinculables y el anonimato resistente al análisis de la cadena de bloques.

- Generación de bloque: 2 minutos
- Moneda total: 184 mil millones de BCN
- Algoritmo de consenso: Prueba de trabajo Cryptonight
- Capitalización de mercado: $3 millones a mediados de 2014

Monero

Monero es otra implementación de CryptoNote. Tiene una curva de emisión ligeramente más plana que Bytecoin, emitiendo el 80% de la moneda en los primeros cuatro años. Ofrece las mismas características de anonimato heredados de CryptoNote.

- Generación de bloque: 1 minuto
- Moneda total: 18,4 millones de XMR
- Algoritmo de consenso: Prueba de trabajo Cryptonight
- Capitalización de mercado: $5 millones a mediados de 2014

Darkcoin

Darkcoin se puso en marcha en enero de 2014. Darkcoin implementa la moneda anónima utilizando un protocolo de re-mezcla de todas las transacciones denominado DarkSend. Darkcoin también es notable por el uso de 11 rondas de diferentes funciones hash (blake, bmw, groestl, jh, keccak, skein, luffa, cubehash, shavite, simd, echo) para el algoritmo de prueba de trabajo.

- Generación de bloque: 2,5 minutos
- Moneda total: Máximo de 22 millones de DRK
- Algoritmo de consenso: Prueba de trabajo multi-algoritmo multi-ronda
- Capitalización de mercado: $19 millones a mediados de 2014

Cadenas Alternativas No Monetarias

Las cadenas alternativas son implementaciones alternativas del patrón de diseño de la cadena de bloques, que no se utilizan principalmente como moneda. Muchas incluyen una moneda, pero la moneda se utiliza como ficha para la asignación de alguna otra cosa, como un recurso o un contrato. La moneda, en otras palabras, no es el punto principal de la plataforma; es una característica secundaria.

Namecoin

Namecoin fue la primera bifurcación del código bitcoin. Namecoin es un registro clave-valor descentralizado y una plataforma de transferencia que usa una cadena de bloques. Es compatible con un registro de nombre de dominio mundial similar al sistema de registro de nombres de dominio en Internet. Namecoin se utiliza actualmente como una alternativa al servicio de nombres de dominio (DNS) para el dominio de nivel raíz .bit. Namecoin también se puede utilizar para registrar nombres y pares clave-valor en otros espacios de nombres; para almacenar cosas como direcciones de correo electrónico, claves de cifrado, certificados SSL, firmas de archivos, sistemas de votación, certificados de acciones; y una multitud de otras aplicaciones.

El sistema Namecoin incluye la moneda Namecoin (símbolo NMC), que se utiliza para pagar las comisiones de transacción para el registro y la transferencia de nombres. A los precios actuales, la comisión de registrar un nombre es de 0,01 NMC o aproximadamente 1 centavo de dólar. Al igual que en bitcoin, las comisiones son recogidas por los mineros namecoin.

Los parámetros básicos de Namecoin son los mismos que en bitcoin:

- Generación de bloque: 10 minutos
- Moneda total: 21 millones de NMC hacia 2140
- Algoritmo de consenso: Prueba de trabajo SHA256
- Capitalización de mercado: $10 millones a mediados de 2014

Los espacios de nombres de Namecoin no están restringidos, y cualquier persona puede utilizar

cualquier espacio de nombres de cualquier manera. Sin embargo, ciertos espacios de nombres tienen acordado una especificación para que cuando se lean desde la cadena de bloques, el software de nivel de aplicación sepa leer y proceder desde allí. Si está mal formado, entonces cualquier software que haya utilizado para leer desde el espacio de nombres específico lanzará un error. Algunos de los espacios de nombres populares son:

- d/ es el nombre de dominio del espacio de nombres de dominios .bit
- id/ es el espacio de nombres para almacenar personas identificadas tales como direcciones de correo electrónico, claves PGP, etc.
- u/ es una especificación adicional, más estructurada para almacenar identidades (basado en openspecs)

El cliente Namecoin es muy similar a Bitcoin Core, porque se deriva del mismo código fuente. Tras la instalación, el cliente descarga una copia completa de la cadena de bloques Namecoin y luego estará listo para consultar y registrar nombres. Hay tres comandos principales:

name_new

Consulta o pre-registra un nombre

name_firstupdate

Registra un nombre y hace público el registro

name_update

Cambia los detalles o actualiza un registro de nombre

Por ejemplo, para registrar el dominio dominio-bitcoin.bit, utilizamos el comando name_new de la siguiente manera:

```
$ namecoind name_new d/mastering-bitcoin
```

```
[
    "21cbab5b1241c6d1a6ad70a2416b3124eb883ac38e423e5ff591d1968eb6664a",
    "a05555e0fc56c023"
]
```

El comando name_new registra una reclamación en el nombre, mediante la creación de un hash del nombre con una clave aleatoria. Las dos cadenas devueltas por name_new son el hash y la clave aleatoria (a05555e0fc56c023 en el ejemplo anterior) que se puede utilizar para hacer un registro público de nombres. Una vez que la reclamación ha sido registrada en la cadena de bloques de Namecoin se puede convertir en un registro público con el comando name_firstupdate, proporcionando la clave aleatoria:

```
$ namecoind name_firstupdate d/mastering-bitcoin a05555e0fc56c023 "{"map": {"www":
{"ip":"1.2.3.4"}}}}"
b7a2e59c0a26e5e2664948946ebeca1260985c2f616ba579e6bc7f35ec234b01
```

En este ejemplo se asignará el nombre de dominio www.mastering-bitcoin.bit a la dirección IP 1.2.3.4. El hash devuelto es el ID de la transacción que se puede utilizar para realizar un seguimiento de este registro. Usted puede ver qué nombres están registrados ejecutando el comando name_list:

```
$ namecoind name_list
```

```
[
    {
        "name" : "d/mastering-bitcoin",
        "value" : "{map: {www: {ip:1.2.3.4}}}}",
        "address" : "NCccBXrRUahAGrisBA1BLPWQfSrups8Geh",
        "expires_in" : 35929
    }
]
```

Los registros Namecoin necesitan ser actualizados cada 36.000 bloques (aproximadamente de 200 a 250 días). El comando name_update no tiene comisiones y por lo tanto la renovación de dominios en Namecoin es gratuita. Se puede manejar el registro, la renovación automática, y la actualización a través de una interfaz web por parte de proveedores de terceros, por un módico precio. Con un proveedor de terceros puede evitar la necesidad de ejecutar un cliente Namecoin, pero se pierde el control independiente de un registro de nombres descentralizado ofrecido por Namecoin.

Ethereum

Ethereum es una plataforma Turing-completa de procesamiento y ejecución de contratos basado en un libro contable de cadena de bloques. No es un clon de Bitcoin, sino un diseño e implementación completamente independiente. Ethereum tiene una moneda incorporada, llamada *ether*, que se requiere para pagar por la ejecución de contratos. La cadena de bloques de Ethereum registra los *contratos*, que se expresan en un nivel bajo, en un lenguaje Turing-completo, similar a un código de bytes. En esencia, un contrato es un programa que se ejecuta en cada nodo del sistema Ethereum. Los contratos Ethereum pueden almacenar datos, enviar y recibir pagos de ether, almacenar ether, y ejecutar una gama infinita (de ahí Turing-completo) de las acciones computables, actuando como agentes de software autónomos descentralizados.

Ethereum puede implementar sistemas bastante complejos que en otro caso se implementarían como cadenas alternativas. Por ejemplo, el siguiente caso es un contrato de registro de nombres similar a Namecoin escrito en Ethereum (o más exactamente, escrito en un lenguaje de alto nivel que puede ser

compilado a código Ethereum):

```
if !contract.storage[msg.data[0]]: # ¿Ya está tomada la clave?
    # ¡Entonces la tomamos!
    contract.storage[msg.data[0]] = msg.data[1]
    return(1)
else:

    return(0) // De lo contrario no hacer nada
```

Futuro de las Monedas

El futuro de las monedas criptográficas en general es aún más brillante que el futuro de bitcoin. Bitcoin introdujo una nueva forma de organización y consenso descentralizado que ha generado cientos de innovaciones increíbles. Estas invenciones probablemente afectarán a amplios sectores de la economía, desde la ciencia de los sistemas distribuidos hasta las finanzas, economía, monedas, banca central, y la gobernanza corporativa. Muchas actividades humanas que anteriormente requerían instituciones u organizaciones centralizadas para funcionar como puntos de control acreditados o de confianza ahora pueden descentralizarse. La invención de la cadena de bloques (blockchain) y el sistema de consenso reducirá significativamente el costo de organización y coordinación de los sistemas a gran escala, mientras que elimina las oportunidades para la concentración de poder, la corrupción, y la captura del regulador.

Seguridad de Bitcoin

Asegurar bitcoin es un reto porque bitcoin no es una referencia abstracta a un valor, como un saldo en una cuenta bancaria. Bitcoin es muy parecido a dinero en efectivo u oro digital. Probablemente haya escuchado la expresión, "la posesión es nueve décimas partes de la ley." Pues bien, en bitcoin, la posesión es diez décimas partes de la ley. La posesión de las claves para desbloquear el bitcoin es equivalente a la posesión de dinero en efectivo o un trozo de metal precioso. Es posible perderlo, olvidarlo, ser robado o accidentalmente dar una cantidad incorrecta a alguien. En cada uno de estos casos, los usuarios no tienen ningún recurso, al igual que si se le cayera dinero en efectivo en una acera pública.

Sin embargo, bitcoin tiene capacidades que el efectivo, el oro y las cuentas bancarias no tienen. Una cartera bitcoin, que contiene las claves, se puede copiar como cualquier archivo. Se puede almacenar en múltiples copias, incluso escrito en papel como copia impresa de seguridad. No se puede hacer "copia de seguridad" de las cuentas de efectivo, oro, o bancarias. Bitcoin es lo suficientemente diferente de todo lo que ha venido antes como para que debamos pensar en la seguridad de bitcoin también de una forma innovadora.

Principios de Seguridad

El principio básico de bitcoin es la descentralización y tiene importantes implicaciones para la seguridad. Un modelo centralizado, como un banco tradicional o una red de pagos, mantiene a los malos actores fuera del sistema mediante el control de acceso y la investigación de antecedentes. En comparación, un sistema descentralizado como bitcoin traslada la responsabilidad y el control a los usuarios. Dado que la seguridad de la red se basa en la prueba de trabajo, no en el control de acceso, la red puede ser abierta y no se requiere cifrado para el tráfico bitcoin.

En una red de pago tradicional, como un sistema de tarjetas de crédito, el pago es de composición abierta, ya que contiene el identificador privado del usuario (el número de tarjeta de crédito). Después de la carga inicial, cualquier persona con acceso al identificador puede "coger" los fondos y cargarlos del propietario una y otra vez. De este modo, la red de pagos tiene que protegerse con encriptación extremo a extremo y debe asegurarse de que no haya espías o intermediarios que puedan comprometer el tráfico de pagos, en tránsito o cuando se almacena (en reposo). Si un actor malintencionado consigue acceso al sistema, puede comprometer las transacciones en curso *y* los componentes de pago que pueden utilizarse para crear nuevas transacciones. Peor aún, cuando se ven comprometidos los datos de clientes, los clientes están expuestos a robo de identidad y deben tomar medidas para evitar el uso fraudulento de las cuentas comprometidas.

Bitcoin es totalmente diferente. Una transacción bitcoin autoriza solamente un valor específico a un destinatario específico y no puede ser falsificado o modificado. No revela ninguna información privada, como la identidad de las partes, y no puede ser utilizado para autorizar los pagos adicionales. Por lo tanto, una red de pago bitcoin no necesita ser encriptada o protegida contra escuchas. De hecho, usted puede difundir transacciones bitcoin a través de un canal abierto al público, como WiFi o Bluetooth, sin perder seguridad.

El modelo de seguridad descentralizada de bitcoin pone mucho poder en manos de los usuarios. Con ese poder viene la responsabilidad de mantener la confidencialidad de las claves. Para la mayoría de los usuarios eso no es fácil de hacer, sobre todo en los dispositivos informáticos de propósito general como smartphones o portátiles conectados a Internet. Aunque el modelo descentralizado de bitcoin impide el tipo de exposición generalizada de las tarjetas de crédito, muchos usuarios no son capaces de asegurar adecuadamente sus claves y pueden ser hackeados, uno por uno.

Desarrollando Sistemas Bitcoin de Forma Segura

El principio más importante para los desarrolladores de bitcoin es la descentralización. La mayoría de los desarrolladores estarán familiarizados con modelos de seguridad centralizados y podrían estar tentados a aplicar estos modelos a sus aplicaciones bitcoin, con resultados desastrosos.

La seguridad de bitcoin se basa en el control descentralizado de claves y en la validación independiente de las transacciones por los mineros. Si quiere aprovechar la seguridad de Bitcoin, es necesario asegurarse de que permanezca en el modelo de seguridad de Bitcoin. En términos simples: no dejar el control de las claves lejos de los usuarios y no llevar las transacciones fuera de la cadena de bloques.

Por ejemplo, muchas de las primeras casas de cambio de bitcoin concentraban todos los fondos de los usuarios en una sola cartera "caliente" con las claves almacenadas en un solo servidor. Ese diseño quita el control a los usuarios y centraliza el control de las claves en un solo sistema. Muchos de estos sistemas han sido hackeados, con consecuencias desastrosas para sus clientes.

Otro error común es llevar las transacciones "fuera de la cadena de bloques" en un esfuerzo equivocado para reducir las comisiones de transacción o para acelerar el procesamiento de transacciones. Un sistema "fuera de la cadena de bloques" registrará las transacciones en un libro contable interno, centralizado y solo se sincronizará ocasionalmente con la cadena de bloques de bitcoin. Esta práctica, sustituye la seguridad descentralizada de bitcoin con un enfoque cerrado y centralizado. Cuando las transacciones están "fuera de la cadena de bloques", los libros contables centralizados que no estén adecuadamente asegurados pueden ser falsificados, desviando fondos y agotando las reservas de manera desapercibida.

A menos que usted esté dispuesto a invertir fuertemente en la seguridad operacional, con múltiples capas de control de acceso, y en auditorías (como hacen los bancos tradicionales), debería pensarlo cuidadosamente antes de llevar fondos fuera del contexto de seguridad descentralizada de Bitcoin. Incluso si tiene los fondos y la disciplina para implementar un modelo de seguridad robusto, ese diseño tan solo replica el frágil modelo de las redes financieras tradicionales, plagadas por el robo de identidad, la corrupción y la malversación de fondos. Para aprovechar el modelo único de seguridad descentralizada de bitcoin, hay que evitar la tentación de arquitecturas centralizadas que podrían hacerle sentirse cómodo, pero que en última instancia, subvierten la seguridad de Bitcoin.

La Raíz de la Confianza

La arquitectura de seguridad tradicional se basa en un concepto llamado *raiz de confianza*, que es un núcleo confiable que se utiliza como base para la seguridad de todo el sistema o aplicación. La

arquitectura de seguridad se desarrolla alrededor de la raíz de confianza como una serie de círculos concéntricos, como las capas de una cebolla, que extiende la confianza hacia el exterior desde el centro. Cada capa se basa en su capa interna, de mayor confianza, utilizando controles de acceso, firmas digitales, cifrado y otras primitivas de seguridad. A medida que los sistemas de software se hacen más complejos, es más probable que contengan errores, que los hagan vulnerables a comprometer la seguridad. Como resultado, cuanto más complejo es un sistema de software, más difícil es de asegurar. El concepto de raíz de confianza asegura que la mayor parte de la confianza se coloca dentro de la parte menos compleja del sistema, y por tanto menos vulnerable, mientras que el software más complejo está en las capas de alrededor. Esta arquitectura de seguridad se repite a diferentes escalas, estableciendo primero una raíz de confianza en el hardware de un solo sistema, y después extendiendo esa raíz de confianza a través del sistema operativo hasta llegar a los servicios del sistema de nivel superior, y finalmente a través de muchos servidores situados en capas de círculos concéntricos de confianza decreciente.

La arquitectura de seguridad Bitcoin es diferente. En Bitcoin, el sistema de consenso crea un libro contable público confiable que es completamente descentralizado. Una cadena de bloques validada correctamente utiliza el bloque génesis como la raíz de la confianza, construyendo una cadena de confianza hasta el bloque actual. Los sistemas Bitcoin pueden y deben utilizar la cadena de bloques como su raíz de confianza. Al diseñar una aplicación bitcoin compleja que consta de servicios en muchos sistemas diferentes, usted debe examinar cuidadosamente la arquitectura de seguridad con el fin de determinar dónde se va a colocar la confianza. En última instancia, la única cosa que debe ser de confianza expresamente es una cadena de bloques plenamente validada. Si su aplicación de forma explícita o implícita otorga la confianza a cualquier cosa que no sea la cadena de bloques, debería ser una fuente de preocupación porque introduce vulnerabilidad. Un buen método para evaluar la arquitectura de seguridad de la aplicación es considerar cada componente individual y evaluar un escenario hipotético donde el componente esté completamente comprometido y bajo el control de un agente malicioso. Tome cada componente de su aplicación, a su vez, y evalúe los impactos sobre la seguridad general si ese componente se ve comprometido. Si la aplicación ya no es segura cuando esos componentes están en peligro, entonces ha colocado la confianza de manera inapropiada en esos componentes. Una aplicación bitcoin sin vulnerabilidades debe ser vulnerable solamente a un compromiso del mecanismo de consenso bitcoin, lo que significa que su raíz de confianza debe estar anclada en la parte más fuerte de la arquitectura de seguridad bitcoin.

Los numerosos ejemplos de casas de cambio de bitcoin hackeadas sirven para subrayar este punto porque su arquitectura y diseño de seguridad falló, incluso bajo el escrutinio más informal. Estas implementaciones centralizadas habían dedicado la confianza explícitamente en numerosos componentes fuera de la cadena de bloques de bitcoin, como carteras calientes, bases de datos de contabilidad centralizadas, claves de cifrado vulnerables, y estrategias similares.

Mejores Prácticas de Seguridad para el Usuario

Los seres humanos han utilizado los controles de seguridad físicos durante miles de años. En comparación, nuestra experiencia con la seguridad digital tiene menos de 50 años. Los sistemas operativos modernos de propósito general no son muy seguros y no son particularmente adecuados para el almacenamiento de dinero digital. Nuestros equipos están constantemente expuestos a las

amenazas externas a través conexiones a Internet siempre activas. Ejecutan miles de componentes de software de cientos de autores, a menudo con acceso sin restricciones a los archivos del usuario. Una sola pieza de software ilegítimo, entre los muchos miles instalados en el equipo, puede poner en peligro sus claves y archivos, y robar cualquier bitcoin almacenado en aplicaciones de monedero. El nivel de mantenimiento de computadoras requerido para mantener un ordenador libre de virus y troyanos está más allá del nivel de habilidad de la mayoría de los usuarios de computadoras.

A pesar de décadas de investigación y de los avances en la seguridad de la información, los activos digitales siguen siendo lamentablemente vulnerables a un adversario con determinación. Incluso los sistemas más altamente protegidos y restringidos, en las empresas de servicios financieros, agencias de inteligencia y los contratistas de defensa, son vulnerados con frecuencia. Bitcoin crea activos digitales que tienen un valor intrínseco y pueden ser robados y desviados a los nuevos propietarios al instante y de manera irrevocable. Esto crea un incentivo enorme para los piratas informáticos. Hasta ahora, los hackers tuvieron que convertir la información de identidad o los elementos de las cuentas—como tarjetas de crédito y cuentas bancarias—en valor después de haberlos comprometido. A pesar de la dificultad del tráfico y del lavado de la información financiera, hemos visto robos cada vez más intensos. Bitcoin intensifica este problema, ya que no necesita ser traficado o lavado; es valor intrínseco en un activo digital.

Afortunadamente, bitcoin también crea los incentivos para mejorar la seguridad informática. Mientras que antes el riesgo de compromiso del ordenador era vago e indirecto, bitcoin hace que estos riesgos sean claros y evidentes. Guardar bitcoin en un equipo sirve para enfocar la mente del usuario en la necesidad de mejorar la seguridad informática. Como consecuencia directa de la proliferación y la creciente adopción de bitcoin y otras monedas digitales, hemos visto una escalada en las técnicas de hacking y también en las soluciones de seguridad. En términos simples, los hackers tienen ahora un objetivo muy jugoso y los usuarios tienen un claro incentivo para defenderse.

Durante los últimos tres años, como resultado directo de la adopción de bitcoin, hemos visto una enorme innovación en el ámbito de la seguridad de la información en forma de cifrado de hardware, almacenamiento de claves y carteras hardware, tecnología multi-firma y depósito digital en garantía. En las siguientes secciones examinaremos varias mejores prácticas para hacer viable la seguridad del usuario.

Almacenamiento Físico de Bitcoins

Como la mayoría de los usuarios se sienten mucho más cómodos con la seguridad física que con la seguridad de la información, un método muy eficaz para la protección de bitcoins es convertirlos en forma física. Las claves bitcoin no son más que números largos. Esto significa que se pueden almacenar en una forma física, como impresos en papel o grabados en una moneda de metal. Asegurar las claves llega a ser entonces tan simple como asegurar físicamente la copia impresa de las claves de bitcoin. Un juego de claves de bitcoin que se imprime en papel se llama una "cartera de papel", y se pueden utilizar muchas herramientas gratuitas para crearlos. Personalmente mantengo la gran mayoría de mis bitcoins (99% o más) almacenados en carteras de papel, cifradas con BIP0038, con múltiples copias guardadas en cajas fuertes. Mantener bitcoin offline se llama "almacenamiento en frio" y es una de las técnicas de seguridad más eficaces. Un sistema de almacenamiento en frío es uno

donde las claves se generan en un sistema sin conexión (nunca conectado a Internet) y se almacenan también fuera de línea, ya sea en papel o en soporte digital, como un lápiz de memoria USB.

Carteras de Hardware

En el largo plazo, la seguridad de bitcoin tomará cada vez más la forma de carteras de hardware a prueba de manipulaciones. A diferencia de una computadora de escritorio o de un teléfono inteligente, una cartera de hardware bitcoin tiene un solo propósito: almacenar bitcoins de forma segura. Sin la existencia de software de propósito general que pueda comprometerse y con interfaces limitadas, las carteras de hardware pueden ofrecer un nivel de seguridad casi infalible a los usuarios no expertos. Espero ver que las carteras de hardware se conviertan en el método predominante de almacenamiento bitcoin. Para un ejemplo de este tipo de cartera de hardware, consulte el Trezor.

Balance de Riesgo

Aunque la mayoría de los usuarios están justamente preocupados por el robo de bitcoin, existe un riesgo aún mayor. Los archivos de datos se pierden todo el tiempo. Si contienen bitcoin, la pérdida es mucho más dolorosa. En el esfuerzo por asegurar sus carteras bitcoin, los usuarios deben tener mucho cuidado de no ir demasiado lejos y terminar perdiendo el bitcoin. En julio de 2011, un conocido proyecto de concienciación y educación de bitcoin perdió casi 7000 bitcoins. En su esfuerzo por evitar el robo, los propietarios habían implementado una compleja serie de copias de seguridad cifradas. Al final perdieron accidentalmente las claves de cifrado, por lo que las copias de seguridad quedaron sin valor y perdieron una fortuna. Como ocultar dinero enterrándolo en el desierto, si usted asegura su bitcoin demasiado bien, podría ocurrir que no sea capaz de encontrarlo de nuevo.

Diversificación de Riesgo

¿Llevaría todo su patrimonio neto en dinero en efectivo en su cartera? La mayoría de la gente lo consideraría imprudente. Sin embargo, los usuarios de bitcoin a menudo mantienen todos sus bitcoin en una sola cartera. En lugar de ello, los usuarios deben distribuir el riesgo entre múltiples y diversas carteras bitcoin. Los usuarios prudentes mantendrán solo una pequeña fracción, tal vez menos del 5%, de sus bitcoins en una cartera en línea o móvil como "dinero de bolsillo". El resto debe ser dividido entre unos pocos mecanismos de almacenamiento diferentes, tales como una cartera de escritorio y fuera de línea (almacenamiento en frío).

Multi-firma y Gobernanza

Cuando una empresa o un individuo almacena grandes cantidades de bitcoin, debería considerar el uso de una dirección bitcoin multi-firma. La multi-firma proporciona seguridad a los fondos al exigir más de una firma para hacer un pago. Las claves de firma deben almacenarse en diferentes ubicaciones y estar bajo el control de diferentes personas. En un entorno corporativo, por ejemplo, las claves deben generarse de forma independiente y deben mantenerse en manos de varios ejecutivos de la empresa, para garantizar que ninguna persona de manera independiente pueda comprometer los fondos. Las direcciones multi-firma también pueden ofrecer redundancia, donde una sola persona tiene varias claves que se almacenan en ubicaciones diferentes.

Supervivencia

Una consideración importante de seguridad que a menudo se pasa por alto es la disponibilidad, especialmente en el contexto de incapacidad o muerte del titular de la clave. A los usuarios de bitcoin se les dice que deben usar contraseñas complejas y mantener sus claves de manera segura y privada, que no deben compartirlas con nadie. Por desgracia, esa práctica hace que sea casi imposible para la familia del usuario recuperar los fondos si el usuario no está disponible para desbloquearlos. En la mayoría de los casos, de hecho, las familias de los usuarios de bitcoin podrían ignorar completamente la existencia de los fondos de bitcoin.

Si usted tiene un montón de bitcoin, debería considerar compartir los datos de acceso con un familiar o un abogado de confianza. Un plan de supervivencia más complejo se puede configurar con acceso multi-firma y administración patrimonial a través de un abogado especializado como un "ejecutor de activos digitales."

Conclusión

Bitcoin es algo completamente nuevo, sin precedentes, y de compleja tecnología. Con el tiempo vamos a desarrollar mejores herramientas y prácticas que son más fáciles de usar por los no expertos en seguridad. Por ahora, los usuarios de bitcoin pueden utilizar muchos de los consejos discutidos aquí para disfrutar de una experiencia de bitcoin segura y sin problemas.

Appendix A: Comandos del Explorador de Bitcoin (bx)

```
Uso: bx COMANDO [--help]

Info: Los comandos bx son:

descodificar-dirección
incrustar-dirección
codificar-dirección
validar-dirección
descodificar-base16
codificar-base16
descodificar-base58
codificar-base58
descodificar-base58check
codificar-base58check
descodificar-base64
codificar-base64
bitcoin160
bitcoin256
btc-a-satoshi
ec-añadir
ec-añadir-secretos
ec-multiplicar
ec-multiplicar-secretos
ec-nuevo
ec-a-dirección
ec-a-público
ec-a-
fetch-balance
fetch-header
fetch-height
fetch-history
fetch-stealth
fetch-tx
fetch-tx-index
hd-new
hd-private
hd-public
hd-to-address
hd-to-ec
hd-to-public
hd-to-wif
```

```
ayuda
establecer-entrada
firma-entrada
validar-entrada
firma-mensaje
validar-mensaje
descodificar-mnemonic
codificar-mnemonic
ripemd160
satoshi-a-btc
descodificar-script
codificar-script
script-a-dirección
semilla
send-tx
send-tx-node
send-tx-p2p
ajustes
sha160
sha256
sha512
descodificar-stealth
codificar-stealth
pública-stealth
secreta-stealth
compartida-stealth
descodificar-tx
codificar-tx
descodificar-uri
codificar-uri
validar-tx
ver-dirección
wif-to-ec
wif-to-public
descodificar-wrap
codificar-wrap
```

Para más información, ver la página principal del Explorador Bitcoin y la documentación de usuario del Explorador de Bitcoin.

Ejemplos de uso de los comandos bx

Veamos algunos ejemplos de utilización de comandos del Explorador de Bitcoin para experimentar con las claves y las direcciones:

Generar un valor "semilla" aleatorio usando el comando seed, el cual usa el generador de números

aleatorios del sistema operativo. Pasa la semilla al comando ec-new para generar una nueva clave privada. Guardamos el output estándar en el archivo *private_key*:

```
$ bx seed | bx ec-new > private_key
$ cat private_key
73096ed11ab9f1db6135857958ece7d73ea7c30862145bcc4bbc7649075de474
```

Ahora, genera la clave pública a partir de esa clave privada usando el comando ec-to-public. Pasamos el archivo *private_key* a la entrada estándar y guardamos la salida estándar del comando en un nuevo archivo *public_key*:

```
$ bx ec-to-public < clave_privada > public_key
$ cat public_key
02fca46a6006a62dfdd2dbb2149359d0d97a04f430f12a7626dd409256c12be500
```

Podemos reformatear la clave_privada como una dirección usando el comando ec-to-address. Pasamos *public_key* a la entrada estándar:

```
$ bx ec-to-address < public_key
17re1S4Q8ZHyCP8Kw7xQad1Lr6XUzWUnkG
```

Las claves generadas de esta manera produce una cartera de tipo 0 no determinística. Eso significa que cada clave es generada a partir de una semilla independiente. Los comandos de Bitcoin Explorer también pueden generar claves determinísticamente, de acuerdo a BIP0032. En este caso, una clave "maestra" es creada a partir de una semilla y luego extendida determinísticamente para producir un árbol de sub-claves, resultando en una cartera de tipo 2 determinística.

Primero, usamos los comandos seed y hd-new para generar una clave maestra que usaremos como la base para derivar una jerarquía de claves.

```
$ bx seed > seed
$ cat seed
eb68ee9f3df6bd4441a9feadec179ff1

$ bx hd-new < semilla > master
$ cat master
xprv9s21ZrQH143K2BEhMYpNQoUvAgiEjArAVaZaCTgsaGe6LsAnwubeiTcDzd23mAoyizm9cApe51gNfLMkBqkYo
WWMCRwzfuJk8RwF1SVEpAQ
```

Ahora usamos el comando hd-private para generar una clave de "cuenta" endurecida y una secuencia de dos claves privadas dentro de la cuenta.

```
$ bx hd-private --hard < maestra > account
$ cat account
xprv9vkDLt81dTKjwHB8fsVB5QK8cGnzveChzSrtCfvu3aMWvQaThp59ueufuyQ8Qi3qpjk4aKsbmbfxwcgS8PYbg
oR2NWHeLyvg4DhoEE68A1n

$ bx hd-private --index 0 < account
xprv9xHfb6w1vX9xgZyPNXVgAhPxSsEkeRcPHEUV5iJcVEsuUEACvR3NRY3fpGhcnBiDbvG4LgndirDsia1e9F3DW
PkX7Tp1V1u97HKG1FJwUpU

$ bx hd-private --index 1 < account
xprv9xHfb6w1vX9xjc8XbN4GN86jzNAZ6xHEqYxzbLB4fzHFd6VqCLPGRZFsdjsuMVERadbgDbziCRJru9n6tzEWr
ASVpEdrZrFidt1RDfn4yA3
```

A continuación usamos el comando hd-public para generar la correspondiente secuencia de dos claves públicas.

```
$ bx hd-public --index 0 < account
xpub6BH1zcTuktiFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYPFJ3vezHz
5wzaSW4FiGrseNDR4LKqTy

$ bx hd-public --index 1 < account
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWyMaMV1cn7nUPUkgQHPVXJVqsrA8xWbGQD
hohEcDFTEYMvYzwRD7Juf8
```

Las claves públicas también pueden ser derivadas a partir de sus correspondientes claves privadas usando el comando hd-to-public.

```
$ bx hd-private --index 0 < account | bx hd-to-public
xpub6BH1zcTuktiFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYPFJ3vezHz
5wzaSW4FiGrseNDR4LKqTy

$ bx hd-private --index 1 < account | bx hd-to-public
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWyMaMV1cn7nUPUkgQHPVXJVqsrA8xWbGQD
hohEcDFTEYMvYzwRD7Juf8
```

Podemos generar un número prácticamente ilimitado de claves en una cadena determinística, todas derivadas de una misma semilla. Esta técnica es usada en muchas aplicaciones de cartera para generar claves a las que es posible hacer copias de respaldo con un único valor semilla. Esto es más fácil que tener que guardar copias de la cartera con todas sus claves generadas aleatoriamente cada vez que una nueva clave es creada.

La semilla puede ser codificada usando el comando mnemonic-encode.

```
$ bx hd-mnemonic < semilla > words
adore repeat vision worst especially veil inch woman cast recall dwell appreciate
```

La semilla puede luego ser decodificada usando el comando mnemonic-decode.

```
$ bx mnemonic-decode < words
eb68ee9f3df6bd4441a9feadec179ff1
```

La codificación mnemónica puede hacer que una semilla sea más fácil de registrar y hasta recordar.

Appendix A: Propuestas de mejora para Bitcoin

Las propuestas de mejora de Bitcoin (BIP) son documentos de diseño que dan información a la comunidad, o describen una nueva funcionalidad de bitcoin, sus procesos o entorno.

De acuerdo a BIP0001 *Propósito y Pautas de BIP*, existen tres tipos de BIP:

Standard BIP

Describe cualquier cambio que afecte a la mayoría o a todas las implementaciones de Bitcoin, tales como un cambio en el protocolo de red, un cambio en las reglas de validación de bloques o transacciones, o cualquier cambio o adición que afecte a la interoperabilidad de las aplicaciones que usan bitcoin.

Informational BIP

Describe un problema de diseño de bitcoin, o proporciona directrices generales o información a la comunidad bitcoin, pero no propone una nueva característica. Los Informational_ BIP no representan necesariamente un consenso o recomendación de la comunidad bitcoin, por lo que los usuarios y los ejecutores pueden ignorarlos o seguir sus recomendaciones.

Process BIP

Describe un proceso de bitcoin, o propone un cambio a (o un evento en) un proceso. Los *Process* BIP son como BIP estandar, pero aplica a zonas distintas del propio protocolo bitcoin. Pueden proponer una implementación, pero no al código base de Bitcoin; a menudo requieren el consenso de la comunidad; y a diferencia de los informational_BIP, son más que recomendaciones, y los usuarios generalmente no son libres de ignorarlos. Los ejemplos incluyen los procedimientos, directrices, los cambios en el proceso de toma de decisiones, y los cambios en las herramientas o entornos utilizados en el desarrollo de Bitcoin. Cualquier meta-BIP también se considera un Process_ BIP.

Las propuestas de mejora de Bitcoin se registran en un repositorio versionado en GitHub. << table_d-1 >> muestra una instantánea de los BIPs en otoño de 2014. Consulte el repositorio real para estar al día con la información más actualizada de los BIPs activos y de su contenido.

Table 1. Captura de BIPs

BIP#	Enlace	Título	Propietario	Tipo	Estado
1	https://github.com/bitcoin/bips/blob/master/bip-0001.mediawiki	Propósito y Guía de BIP	Amir Taaki	Estándar	Activo

10	https://github.com/bitcoin/bips/blob/master/bip-0010.mediawiki	Distribución de Transacciones Multifirma	Alan Reiner	Informacional	Borrador
11	https://github.com/bitcoin/bips/blob/master/bip-0011.mediawiki	M-of-N Standard Transactions	Gavin Andresen	Estándar	Aceptado
12	https://github.com/bitcoin/bips/blob/master/bip-0012.mediawiki	OP_EVAL	Gavin Andresen	Estándar	Removido
13	https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki	Formato de Direcciones para pay-to-script-hash	Gavin Andresen	Estándar	Final
14	https://github.com/bitcoin/bips/blob/master/bip-0014.mediawiki	Versión de Protocolo y Agente de Usuario	Amir Taaki, Patrick Strateman	Estándar	Aceptado
15	https://github.com/bitcoin/bips/blob/master/bip-0015.mediawiki	Alias	Amir Taaki	Estándar	Retirado
16	https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki	Pay To Script Hash	Gavin Andresen	Estándar	Aceptado

17	https://github.com/bitcoin/bips/blob/master/bip-0017.mediawiki	OP_CHECKHASHVERIFY (CHV)	Luke Dashjr	Retirado	Borrador
18	https://github.com/bitcoin/bips/blob/master/bip-0018.mediawikilink:	hashScriptCheck	Luke Dashjr	Estándar	Borrador
19	https://github.com/bitcoin/bips/blob/master/bip-0019.mediawiki	Transacciones Estándar M-de-N (Low SigOp)	Luke Dashjr	Estándar	Borrador
20	https://github.com/bitcoin/bips/blob/master/bip-0020.mediawiki	Esquema URI	Luke Dashjr	Estándar	Reemplazado
21	https://github.com/bitcoin/bips/blob/master/bip-0021.mediawiki	Esquema URI	Nils Schneider, Matt Corallo	Estándar	Aceptado
22	https://github.com/bitcoin/bips/blob/master/bip-0022.mediawiki	getblocktemplate - Fundamentos	Luke Dashjr	Estándar	Aceptado
23	https://github.com/bitcoin/bips/blob/master/bip-0023.mediawiki	getblocktemplate - Pooled Mining	Luke Dashjr	Estándar	Aceptado

30	https://github.com/bitcoin/bips/blob/master/bip-0030.mediawiki	Transacciones Duplicadas	Pieter Wuille	Estándar	Aceptado
31	https://github.com/bitcoin/bips/blob/master/bip-0031.mediawiki	Mensaje Pong	Mike Hearn	Standard	Estándar
Aceptado	32	https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki	Monederos Jerárquicos Deterministas	Pieter Wuille	Informacional
Aceptado	33	https://github.com/bitcoin/bips/blob/master/bip-0033.mediawiki	Stratized Nodes	Amir Taaki	Estándar
Borrador	34	https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki	Block v2, Altura en coinbase	Gavin Andresen	Estándar
Aceptado	35	https://github.com/bitcoin/bips/blob/master/bip-0035.mediawiki	Mensaje de mempool	Jeff Garzik	Estándar
Estándar	Aceptado	36	https://github.com/bitcoin/bips/blob/master/bip-0036.mediawiki	Servicios Personalizados	Stefan Thomas

Estándar	Borrador	37	https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki	Bloom filtering	Mike Hearn y Matt Corallo
Estándar	Aceptado	38	https://github.com/bitcoin/bips/blob/master/bip-0038.mediawiki	Clave privada protegida por contraseña	Mike Caldwell
Estándar	Borrador	39	https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki	Mnemónico para la generación de claves deterministas claves	Slush
Estándar	Borrador	40		protocolo Stratum en cable	Slush
Estándar	Número BIP asignado	41		protocolo Stratum en minería	Slush
Estándar	Número BIP asignado	42	https://github.com/bitcoin/bips/blob/master/bip-0042.mediawiki	Una oferta monetaria finita para bitcoin	Pieter Wuille
Estándar	Borrador	43	https://github.com/bitcoin/bips/blob/master/bip-0043.mediawiki	Campo adicional para Monederos Deterministas	Slush
Estándar	Borrador	44	https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki	Jerarquía Multi-Cuenta para Monederos Deterministas Monederos	Slush

Estándar	Borrador	50	https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki	Post-Mortem del Fork de Marzo 2013	Gavin Andresen
Informacional	Borrador	60	https://github.com/bitcoin/bips/blob/master/bip-0060.mediawiki	Mensaje "version" de Longitud Fija (Campo Relay-Transactions)	Amir Taaki
Estándar	Borrador	61	https://github.com/bitcoin/bips/blob/master/bip-0061.mediawiki	Mensaje P2P "reject"	Gavin Andresen
Estándar	Borrador	62	https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki	Tratamiento de la maleabilidad	Pieter Wuille
Estándar	Borrador	63		Direcciones Ocultas	Peter Todd
Estándar	Número BIP asignado	64	https://github.com/bitcoin/bips/blob/master/bip-0064.mediawiki	getutxos message	Mike Hearn
Estándar	Borrador	70	https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki	Protocolo de Pagos	Gavin Andresen
Estándar	Borrador	71	https://github.com/bitcoin/bips/blob/master/bip-0071.mediawiki	Tipos MIME del Protocolo de Pagos	Gavin Andresen

Estándar	Borrador	72	https://github.com/bitcoin/bips/blob/master/bip-0072.mediawiki	URIs del Protocolo de Pagos	Gavin Andresen
Estándar	Borrador	73	https://github.com/bitcoin/bips/blob/master/bip-0073.mediawiki	Usar cabecera "Accept" con Petición de Pago URLs	Stephen Pair

Appendix A: pycoin, ku, y tx

La libreria Python pycoin, originalmente escrita y mantenida por Richard Kiss, es una libreria basada en Python que soporta manipulacion de transacciones y llaves bitcoin, incluso soportando suficientemente el lenguaje de scripting para lidiar apropiadamente con transacciones no estandard.

La libreria pycoin soporta ambas Python 2 (2.7x) y Python 3 (despues de la 3.3), y viene con varias utilerias de linea de comandos, ku y tx.

Utilidad de claves (KU)

La utileria de linea de comando ku ("key utility") es una navaja suiza para manipular llaves. Soporta llaves BIP32, WIF y direcciones (bitcoin y altcoins). Enseguida se ven algunos ejemplos.

Crear una clave BIP32 utilizando las fuentes de entropía por defecto de GPG y */dev/random*:

```
$ ku create

input           : create
network         : Bitcoin
wallet key      : xprv9s21ZrQH143K3LU5ctPZTBnb9kTjA5Su9DcWHvXJemiJBsY7VqXUG7hipgdWaU
                    m2nhnzdvxJf5KJo9vjP2nABX65c5sFsWsV8oXcbpehtJi
public version  : xpub661MyMwAqRbcFpYYiuvZpKjKhnJDZYAkWSY76JvvD7FH4fsG3Nqiov2CfxzxY8
                    DGcpfT56AMFeo8M8KPkFMfLUtvwjwb6WPv8rY65L2q8Hz
tree depth      : 0
fingerprint     : 9d9c6092
parent f'print  : 00000000
child index     : 0
chain code      : 80574fb260edaa4905bc86c9a47d30c697c50047ed466c0d4a5167f6821e8f3c
clave privada : sí
secret exponent :
112471538590155650688604752840386134637231974546906847202389294096567806844862
 hex            : f8a8a28b28a916e1043cc0aca52033a18a13cab1638d544006469bc171fddfbe
wif             : L5Z54xi6qJusQT42JHA44mfPVZGjyb4XBRWfxAzUWwRiGx1kV4sP
 uncompressed   : 5KhoEavGNNH4GHKoy2Ptu4KfdNp4r56L5B5un8FP6RZnbsz5Nmb
public pair x   :
764606382405464783648433974782784681018771177678734621270215603682901140160 34
public pair y   :
598078796574697741020401202982722077309212917366332477370774067536768257777 01
 x as hex       : a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
 y as hex       : 843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcd625
y parity        : odd
par de claves como sec :
03a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
 uncompressed   : 04a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
                    843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcd625
hash160         : 9d9c609247174ae323acfc96c852753fe3c8819d
 uncompressed   : 8870d869800c9b91ce1eb460f4c60540f87c15d7
Bitcoin address : 1FNNRQ5fSv1wBi5gyfVBs2rkNheMGt86sp
 uncompressed   : 1DSS5isnH4FsVaLVjeVXewVSpfqktdiQAM
```

Crear una clave BIP32 desde una contraseña:

> La contrasena en este ejemplo es muy facil de adivinar.

```
$ ku P:foo

input           : P:foo
network         : Bitcoin
clave de cartera : xprv9s21ZrQH143K31AgNK5pyVvW23gHnkBq2wh5aEk6g1s496M8ZMjxncCKZKgb5j
                   ZoY5eSJMJ2Vbyvi2hbmQnCuHBujZ2WXGTux1X2k9Krdtq
public version  : xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtS
                   VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
tree depth      : 0
fingerprint     : 5d353a2e
parent f'print  : 00000000
child index     : 0
chain code      : 5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc
clave privada : sí
secret exponent :
65825730547097305716057160437970790220123864299761908948746835886007793998275
 hex            : 91880b0e3017ba586b735fe7d04f1790f3c46b818a2151fb2def5f14dd2fd9c3
wif             : L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
 uncompressed   : 5JvNzA5vXDoKYJdw8SwwLHxUxaWvn9mDea6k1vRPCX7KLUVWa7W
public pair x   :
81821982719381104061777349269130419024493616650993589394553404347774393168191
public pair y   :
58994218069605424278320703250689780154785099509277691723126325051200459038290
 x as hex       : b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
 y as hex       : 826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52
paridad y        : par
par de claves como sec :
02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
 Descomprimidas   : 04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
                    826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52
hash160         : 5d353a2ecdb262477172852d57a3f11de0c19286
 Descomprimidas  : e5bd3a7e6cb62b4c820e51200fb1c148d79e67da
Direccion Bitcoin : 19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
 Descomprimida   : 1MwkRkogzBRMehBntgcq2aJhXCXStJTXHT
```

Obtener información como JSON:

```
$ ku P:foo -P -j
```

```
{
   "y_parity": "even",
   "public_pair_y_hex":
"826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
   "private_key": "no",
   "parent_fingerprint": "00000000",
   "tree_depth": "0",
   "network": "Bitcoin",
   "btc_address_uncompressed": "1MwkRkogzBRMehBntgcq2aJhXCXStJTXHT",
   "key_pair_as_sec_uncompressed":
"04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f826d8b4d3010aea16ff4c1
c1d3ae68541d9a04df54a2c48cc241c2983544de52",
   "public_pair_x_hex":
"b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f",
   "wallet_key":
"xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtSVYFvXz2vPPpbXE1qpjoUFi
dhjFj82pVShWu9curWmb2zy",
   "chain_code": "5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc",
   "child_index": "0",
   "hash160_uncompressed": "e5bd3a7e6cb62b4c820e51200fb1c148d79e67da",
   "btc_address": "19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii",
   "fingerprint": "5d353a2e",
   "hash160": "5d353a2ecdb262477172852d57a3f11de0c19286",
   "input": "P:foo",
   "public_pair_x":
"818219827193811040617773492691304190244936166509935893945534043477743931681 91",
   "public_pair_y":
"589942180696054242783207032506897801547850995092776917231263250512004590382 90",
   "key_pair_as_sec":
"02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f"
}
```


Clave BIP32 pública:

```
$ ku -w -P P:foo
xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtSVYFvXz2vPPpbXE1qpjoUFid
hjFj82pVShWu9curWmb2zy
```

Generar una subclave:

```
$ ku -w -s3/2 P:foo
xprv9wTErTSkjVyJa1v4cUTFMFkWMe5eu8ErbQcs9xajnsUzCBT7ykHAwdrxvG3g3f6BFk7ms5hHBvmbdutNmyg6i
ogWKxx6mefEw4M8EroLgKj
```

Subclave reforzada:

```
$ ku -w -s3/2H P:foo
xprv9wTErTSu5AWGkDeUPmqBcbZWX1xq85ZNX9iQRQW9DXwygFp7iRGJo79dsVctcsCHsnZ3XU3DhsuaGZbDh8iDk
BN45k67UKsJUXM1JfRCdn1
```

WIF

```
$ ku -W P:foo
L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
```

Dirección

```
$ ku -a P:foo
19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
```

Generar un montón de subclaves

```
$ ku P:foo -s 0/0-5 -w
xprv9xWkBDfyBXmZjBG9EiXBpy67KK72fphUp9utJokEBFtjsjiuKUUDF5V3TU8U8cDzytqYnSekc8bYuJS8G3bhX
xKWB89Ggn2dzLcoJsuEdRK
xprv9xWkBDfyBXmZnzKf3bAGifK593gT7WJZPnYAmvc77gUQVej5QHckc5Adtwxa28ACmANi9XhCrRvtFqQcUxt8r
UgFz3souMiDdWxJDZnQxzx
xprv9xWkBDfyBXmZqdXA8y4SWqfBdy71gSW9sjx9JpCiJEiBwSMQyRxan6srXUPBtj3PTxQFkZJAiwoUpmvtrxKZu
4zfsnr3pqyy2vthpkwuoVq
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8EskpzK
L1Y8Gk9aX6QbryA5raK73p
xprv9xWkBDfyBXmZv2q3N66hhZ8DAcEnQDnXML1J62krJAcf7Xb1HJwuW2VMJQrCofY2jtFXdiEY8UsRNJfqK6DAd
yZXoMvtaLHyWQx3FS4A9zw
xprv9xWkBDfyBXmZw4jEYXUHYc9fT25k9irP87n2RqfJ5bqbjKdT84Mm7Wtc2xmzFuKg7iYf7XFHKkSsaYKWKJbR5
4bnyAD9GzjUYbAYTtN4ruo
```

Generar las direcciones correspondientes:

```
$ ku P:foo -s 0/0-5 -a
1MrjE78H1R1rqdFrmkjdHnPUdLCJALbv3x
1AnYyVEcuqeoVzH96zj1eYKwoWfwte2pxu
1GXr1kZfxE1FcK6ZRD5sqqqs5YfvuzA1Lb
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
1Cz2rTLjRM6pMnxPNrRKp9ZSvRtj5dDUML
1WstdwPnU6HEUPme1DQayN9nm6j7nDVEM
```

Generar los WIFs correspondientes:

```
$ ku P:foo -s 0/0-5 -W
L5a4iE5k9gcJKGqX3FWmxzBYQc29PvZ6pgBaePLVqT5YByEnBomx
Kyjgne6GZwPGB6G6kJEhoPbmyjMP7D5d3zRbHVjwcq4iQXD9QqKQ
L4B3ygQxK6zH2NQGxLDee2H9v4Lvwg14cLJW7QwWPzCtKHdWMaQz
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmTDMrqemY8UF
L2oD6vA4TUyqPF8QG4vhUFSgwCyuuvFZ3v8SKHYFDwkbM765Nrfd
KzChTbc3kZFxUSJ3Kt54cxsogeFAD9CCM4zGB22si8nfKcThQn8C
```

Compruebe que funciona eligiendo una cadena BIP32 (la correspondiente a la subclave 0/3):

```
$ ku -W
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8EskpzK
L1Y8Gk9aX6QbryA5raK73p
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmTDMrqemY8UF
$ ku -a
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8EskpzK
L1Y8Gk9aX6QbryA5raK73p
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
```

Sí, parece familiar.

Del exponente secreto:

```
$ ku 1

input           : 1
network         : Bitcoin
secret exponent : 1
 hex            : 1
wif             : KwDiBf89QgGbjEhKnhXJuH7LrciVrZi3qYjgd9M7rFU73sVHnoWn
 uncompressed   : 5HpHagT65TZzG1PH3CSu63k8DbpvD8s5ip4nEB3kEsreAnchuDf
public pair x   :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y   :
32670510020758816978083085130507043184471273380659243275938904335757337482424
 x as hex       : 79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex       : 483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
paridad y        : par
key pair as sec : 0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 uncompressed   : 0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
                    483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160         : 751e76e8199196d454941c45d1b3a323f1433bd6
 uncompressed   : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAMH
 uncompressed   : 1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm
```

Versión Litecoin:

```
$ ku -nL 1

input            : 1
Red        : Litecoin
exponente secreto  : 1
 hex             : 1
wif              : T33ydQRKp4FCW5LCLLUB7deioUMoveiwekdwUwyfRDeGZm76aUjV
 descomprimida    : 6u823ozcyt2rjPH8Z2ErsSXJB5PPQwK7VVTwwN4mxLBFrao69XQ
par publico x    :
55066263022277343669578718895168534326250603453777594175500187360389116729240
par publico y    :
326705100207588169780830851305070431844712733806592432759389043357573374824 24
 x como hex        : 79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y como hex        : 483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
paridad y         : par
par de llaves como sec  :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 descomprimida    : 0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
                      483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160          : 751e76e8199196d454941c45d1b3a323f1433bd6
 descomprimida    : 91b24bf9f5288532960ac687abb035127b1d28a5
Direccion Litecoin : LVuDpNCSSj6pQ7t9Pv6d6sUkLKoqDEVUnJ
 descomprimida    : LYWKqJhtPeGyBAw7WC8R3F7ovxtzAiubdM
```

Dogecoin WIF:

```
$ ku -nD -W 1
QNcdLVw8fHkixm6NNyN6nVwxKek4u7qrioRbQmjxac5TVoTtZuot
```

De par público (en red de pruebas):

```
$ ku -nT
55066263022277343669578718895168534326250603453777594175500187360389116729240,even

entrada                   :
55066263022277343669578718895168534326250603453777594175500187360389116729240
                            89116729240, par
network                 : Bitcoin testnet
public pair x           :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y           :
32670510020758816978083085130507043184471273380659243275938904335757337482424
 x as hex               :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex               :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity                : even
key pair as sec         :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 uncompressed           :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798

483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160                 : 751e76e8199196d454941c45d1b3a323f1433bd6
 uncompressed           : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin testnet address : mrCDrCybB6J1vRfbwM5hemdJz73FwDBC8r
 uncompressed           : mtoKs9V381UAhUia3d7Vb9GNak8Qvmcsme
```

De hash160

```
$ ku 751e76e8199196d454941c45d1b3a323f1433bd6

input           : 751e76e8199196d454941c45d1b3a323f1433bd6
network         : Bitcoin
hash160         : 751e76e8199196d454941c45d1b3a323f1433bd6
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAMH
```

Como dirección de dogecoin:

```
$ ku -nD 751e76e8199196d454941c45d1b3a323f1433bd6

input            : 751e76e8199196d454941c45d1b3a323f1433bd6
network          : Dogecoin
hash160          : 751e76e8199196d454941c45d1b3a323f1433bd6
Dogecoin address : DFpN6QqFfUm3gKNaxN6tNcab1FArL9cZLE
```

Utilidad de transacción (TX)

La utileria de linea de comandos tx desplegara transacciones de manera humanamente comprensible, extraera transacciones base del cache de transacciones pycoin's o de servicios web (actualmente soportados blockchain.info, blockr.io, and biteasy.com), combinara transacciones, agregara o eliminara entradas o salidas y firmara transacciones.

Los siguientes son algunos ejemplos.

Ver la famosa transacción de la "pizza" [PIZZA]:

```
$ tx 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
Advertencia: considera establecer la variable de ambiente
PYCOIN_CACHE_DIR=~/.pycoin_cache  a las transacciones cache extraidas via servicios
web
advertencia: no se encontraron proveedores de servicios para get_tx; considera
extablecer la variable de ambiente
PYCOIN_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCKEXPLORER
uso: tx [-h] [-t VERSION_DE_LA_TRANSACCION] [-l TIEMPO_DE_BLOQUEO] [-n RED] [-a]
          [-i dirección] [-f ruta-a-claves-privadas] [-g GPG_ARGUMENT]
          [--remove-tx-in tx_in_index_to_delete]
          [--remove-tx-out tx_out_index_to_delete] [-F transaction-fee] [-u]
          [-b BITCOIND_URL] [-o ruta-al-archivo-de-salida]
          argument [argumento ...]
tx: error: no se puede encontrar Tx con id
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
```

Huy! No tenemos servicios web configurados. Hagámoslo ahora:

```
$ PYCOIN_CACHE_DIR=~/.pycoin_cache
$ PYCOIN_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCKEXPLORER
$ export PYCOIN_CACHE_DIR PYCOIN_SERVICE_PROVIDERS
```

No se hace automaticamente asi que una herramienta de linea de comando no dejara escapar informacion privada acerca de las transacciones en las que estas interesado a un sitio web de terceros. Si no te importa, tu puedes agregar estas lineas en tu *.profile*.

Intentémoslo de nuevo:

```
$ tx 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
Versión: 1 tx hash 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
159 bytes
TxIn count: 1; TxOut count: 1
Tiempo de bloqueo: 0 (válido en cualquier momento)
Entrada:
  0: (desconocido) desde
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0
Salida:
  0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik recibe 10000000.00000 mBTC
Salida total 10000000.00000 mBTC
incluyendo no gastadas en volcado hexadecimal ya que la transaccion no esta
completamente firmada
010000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a4
93046022100a7f26eda874931999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e
9199238eb73f07c8f209504c84b80f03e30ed8169edd44f80ed17ddf451901ffffffff010010a5d4e8000
0001976a9147ec1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000

** no se puede validar la transaccion debido a transacciones fuente faltantes
```

Aparece la linea final porque para validar las firmas de las transacciones, tu tecnicamente necesitas las transacciones fuente. Asi que agreguemos -a para incrementar las transacciones con informacion fuente:

```
$ tx -a 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
Advertencia: las recomendaciones de calculo y estimacion de las comisiones por
transaccion  pueden ser incorrectas
Advertencia: Si la comision de la transaccion es mas baja que el valor esperado de
0,1 MBTC, la transacción no se podría propagar
Versión: 1 tx hash 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
159 bytes
TxIn count: 1; TxOut count: 1
Tiempo de bloqueo: 0 (válido en cualquier momento)
Entrada:
  0: 17WFx2GQZUmh6Up2NDNCEDk3deYomdNCfk de
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0 10000000.00000
mBTC sig ok
Salida:
  0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik recibe 10000000.00000 mBTC
Entrada total 10000000.00000 mBTC
Salida total 10000000.00000 mBTC
Comisiones totales 0.00000 mBTC

010000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a4
93046022100a7f26eda874931999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e
9199238eb73f07c8f209504c84b80f03e30ed8169edd44f80ed17ddf451901ffffffff010010a5d4e8000
0001976a9147ec1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000

todos los valores de transacción entrantes validados
```

Ahora, veamos una salida no gastada para una direccion especifica (UTXO). En el bloque #1, vemos una transaccion coinbase a 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX. Usemos fetch_unspent para encontrar todas las monedas en esta direccion:

```
$ fetch_unspent 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX
a3a6f902a51a2cbebede144e48a88c05e608c2cce28024041a5b9874013a1e2a/0/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/333000
cea36d008badf5c7866894b191d3239de9582d89b6b452b596f1f1b76347f8cb/31/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
065ef6b1463f552f675622a5d1fd2c08d6324b4402049f68e767a719e2049e8d/86/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
a66dddd42f9f2491d3c336ce5527d45cc5c2163aaed3158f81dc054447f447a2/0/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
ffd901679de65d4398de90cefe68d2c3ef073c41f7e8dbec2fb5cd75fe71dfe7/0/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/100
d658ab87cc053b8dbcfd4aa2717fd23cc3edfe90ec75351fadd6a0f7993b461d/5/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/911
36ebe0ca3237002acb12e1474a3859bde0ac84b419ec4ae373e63363ebef731c/1/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/100000
fd87f9adebb17f4ebb1673da76ff48ad29e64b7afa02fda0f2c14e43d220fe24/0/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/1
dfdf0b375a987f17056e5e919ee6eadd87dad36c09c4016d4a03cea15e5c05e3/1/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/1337
cb2679bfd0a557b2dc0d8a6116822f3fcbe281ca3f3e18d3855aa7ea378fa373/0/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/1337
d6be34ccf6edddc3cf69842dce99fe503bf632ba2c2adb0f95c63f6706ae0c52/1/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/2000000

0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098/0/410496b538e853519c726a2c91e61ec11600ae1390813a627c66fb8be7947be63c52da7589379515d4e0a604f8141781e62294721166bf621e73a82cbf2342c858eeac/5000000000
```

Operadores, Constantes y Símbolos del Lenguaje de Script de Transacción

< <tx_script_ops_table_pushdata> > Muestra operadores para poner los valores en la pila.

Table 1. Empujar valor a la pila

Símbolo	Valor (hexadecimal)	Descripción
OP_0 o OP_FALSE	0x00	Un array vacío es empujado a la pila
1-75	0x01-0x4b	Empuja, los siguientes N bytes a la pila, donde N es de 1 a 75 bytes
OP_PUSHDATA1	0x4c	El siguiente byte del script contiene N, empuja. los siguientes N bytes a la pila
OP_PUSHDATA2	0x4d	Los siguientes 2 byte del script contienen N, empuja. los siguientes N bytes a la pila
OP_PUSHDATA4	0x4e	Los siguientes 4 byte del script contiene N, empuja. los siguientes N bytes a la pila
OP_1NEGATE	0x4f	Empuja, el valor "-1" a la pila
OP_RESERVED	0x50	Parar - Transacción inválida excepto si se encuentra en un OP_IF no ejecutado
OP_1 u OP_TRUE	0x51	Empujar el valor "1" a la pila
OP_2 a OP_16	0x52 a 0x60	Para OP_N, empuja el valor "N" en la pila. Por ejemplo, OP_2 empuja "2"

Control de flujo condicional Muestra los operadores de control de flujo condicional.

Table 2. Control de flujo condicional

Símbolo	Valor (hexadecimal)	Descripción
OP_NOP	0x61	No hacer nada
OP_VER	0x62	Parar - transacción no válida a menos que se encuentre en una cláusula OP_IF no ejecutada

OP_IF	0x63	Ejecutar las siguientes sentencias si el tope de la pila no es 0
OP_NOTIF	0x64	Ejecutar las siguientes sentencias si el tope de la pila es 0
OP_VERIF	0x65	Parar - transacción no válida
OP_VERNOTIF	0x66	Parar - transacción no válida
OP_ELSE	0x67	Ejecutar solo si las sentencias previas no fueron ejecutadas
OP_ENDIF	0x68	Fin del bloque OP_IF, OP_NOTIF u OP_ELSE
OP_VERIFY	0x69	Comprueba la parte superior de la pila, detener y anular la transacción si no es VERDADERO
OP_RETURN	0x6A	Detener y anulará la transacción

[tx_script_opts_table_stack] muestra operadores usados para manipular la pila.

Table 3. Operadores de pila

Símbolo	Valor (hexadecimal)	Descripción
OP_TOALTSTACK	0x6b	Pone elemento superior de la pila y empujar a pila alternativa
OP_FROMALTSTACK	0x6c	Pone elemento superior de la pila alternativa y empuje para apilar
OP_2DROP	0x6d	Pone arriba dos elementos de pila
OP_2DUP	0x6e	Duplica dos elementos de arriba de la pila
OP_3DUP	0x6F	Duplicar los tres elementos de arriba de la pila
OP_2OVER	0x70	Copiar el tercer y cuarto elementos de la pila de la parte superior
OP_2ROT	0x71	Mueva el quinto y sexto elementos de la pila de la parte superior

OP_2SWAP	0x72	Intercambiar los dos primeros pares de elementos de la pila
OP_IFDUP	0x73	Duplicar el elemento superior de la pila si no es 0
OP_DEPTH	0x74	Cuenta los elementos de la pila y empuja el recuento resultante
OP_DROP	0x75	Coge el elemento superior de la pila
OP_DUP	0x76	Duplicar el elemento superior de la pila
OP_NIP	0x77	Coge el segundo elemento de la pila
OP_OVER	0x78	Copie el segundo elemento de la pila y empujarla hacia la parte superior
OP_PICK	0x79	Coge el valor N desde la parte superior, a continuación, copiar el elemento enésimo de la parte superior de la pila
OP_ROLL	0x7a	Coge el valor N desde la parte superior, a continuación, mover el elemento enésimo de la parte superior de la pila
OP_ROT	0x7b	Gira los tres principales elementos de la pila
OP_SWAP	0x7c	Cambia los tres primeros elementos de la pila
OP_TUCK	0x7D	Copiar el elemento superior e insertarlo entre el primer elemento y el segundo.

Operaciones de unión de cadenas muestra los operadores de cadena.

Table 4. Operaciones de unión de cadenas

Símbolo	Valor (hexadecimal)	Descripción
OP_CAT	0x7e	Desactivada (concatena dos principales elementos)
OP_SUBSTR	0x7f	Desactivada (devuelve subcadena)

OP_LEFT	0x80	Desactivada (devuelve subcadena izquierda)
OP_RIGHT	0x81	Desactivada (devuelve subcadena derecha)
OP_SIZE	0x82	Calcular la longitud de cadena del elemento superior y empuje el resultado

Aritmética binaria y condicionales muestra la aritmética binaria y operadores lógicos booleanos.

Table 5. Aritmética binaria y condicionales

Símbolo	Valor (hexadecimal)	Descripción
OP_INVERT	0x83	Deshabilitado (Invertir los bits del ítem superior)
OP_AND	0x84	Deshabilitado (AND booleano de los dos ítems superiores)
OP_OR	0x85	Deshabilitado (OR booleano de los dos ítems superiores)
OP_XOR	0x86	Deshabilitado (booleano XOR de dos elementos principales)
OP_EQUAL	0x87	Empuja VERDADERO (1) si los dos principales elementos son exactamente iguales, empujar FALSO (0) en caso contrario
OP_EQUALVERIFY	0x88	Igual que OP_EQUAL, pero correr OP_VERIFY después de detenerse si no es VERDADERO
OP_RESERVED1	0x89	Parar - transacción no válida a menos que se encontró en una cláusula OP_IF no ejecutada
OP_RESERVED2	0x8A	Parar - transacción no válida a menos que se encontró en una cláusula OP_IF no ejecutada

Operadores numéricos muestra operadores numéricos (aritmética).

Table 6. Operadores numéricos

Símbolo	Valor (hexadecimal)	Descripción
OP_1ADD	0x8b	Sumar 1 al ítem superior

OP_1SUB	0x8c	Restar 1 al ítem superior
OP_2MUL	0x8d	Deshabilitado (multiplicar ítem superior por 2)
OP_2DIV	0x8e	Deshabilitado (dividir ítem superior por 2)
OP_NEGATE	0x8F	Voltear el signo del elemento superior
OP_ABS	0x90	Cambiar el signo del elemento superior a positivo
OP_NOT	0x91	Si el artículo superior es 0 o 1 Boleano lo voltea, de lo contrario devuelve 0
OP_0NOTEQUAL	0x92	Si el artículo superior es 0 devuelve 0, en caso contrario devuelve 1
OP_ADD	0x93	Poner dos artículos en el tope, añadirlos y el empujar el resultado
OP_SUB	0x94	Poner primero dos artículos, restar primero del segundo, empujar resultado
OP_MUL	0x95	Deshabilidato (multiplicar dos primeros artículos)
OP_DIV	0x96	Deshabilitado (dividir segundo punto por el primer artículo)
OP_MOD	0x97	Deshabilitado (resto de dividir segundo artículo por el primer artículo)
OP_LSHIFT	0x98	Deshabilitado (cambiar segundo elemento dejado por el primer número de artículo de bits)
OP_RSHIFT	0x99	Deshabilitado (desplazar hacia la derecha en el segundo elemento el número de bits indicado en el primer elemento)
OP_BOOLAND	0x9a	Operador booleano AND de los dos elementos superiores
OP_BOOLOR	0x9b	Operador booleano OR de los dos elementos superiores

OP_NUMEQUAL	0x9c	Devuelve VERDADERO si los dos elementos superiores son dos números iguales
OP_NUMEQUALVERIFY	0x9d	Igual que NUMEQUAL, después OP_VERIFY para parar si no es VERDADERO
OP_NUMNOTEQUAL	0x9e	Devuelve VERDADERO si los dos elementos superiores no son números iguales
OP_LESSTHAN	0x9f	Devuelve VERDADERO si el segundo elemento es menor que el elemento superior
OP_GREATERTHAN	0xa0	Devuelve VERDADERO si el segundo elemento es mayor que el elemento superior
OP_LESSTHANOREQUAL	0xa1	Devuelve VERDADERO si el segundo elemento es menor o igual que el elemento superior
OP_GREATERTHANOREQUAL	0xa2	Devuelve VERDADERO si el segundo elemento es mayor o igual que el elemento superior
OP_MIN	0xa3	Devuelve el menor de los dos elementos superiores
OP_MAX	0xa4	Devuelve el mayor de los dos elementos superiores
OP_WITHIN	0xa5	Devuelve VERDADERO si el tercer elemento está entre el segundo elemento (o igual) y el primer elemento

Operaciones criptográficas y de hashing muestra operadores de funciones criptográficas.

Table 7. Operaciones criptográficas y de hashing

Símbolo	Valor (hexadecimal)	Descripción
OP_RIPEMD160	0xa6	Devuelve el hash RIPEMD160 del elemento superior
OP_SHA1	0xa7	Devuelve el hash SHA1 del elemento superior
OP_SHA256	0xa8	Devuelve el hash SHA256 del elemento superior

OP_HASH160	0xa9	Devuelve el hash RIPEMD160(SHA256(x)) del elemento superior
OP_HASH256	0xaa	Devuelve el hash SHA256(SHA256(x)) del elemento superior
OP_CODESEPARATOR	0xab	Marca el comienzo de los datos firmados
OP_CHECKSIG	0xac	Extrae una clave pública y una firma y valida la firma con el hash de los datos de la transacción, devuelve VERDADERO si coinciden
OP_CHECKSIGVERIFY	0xad	Igual que CHECKSIG, después OP_VERIFY para parar si no es VERDADERO
OP_CHECKMULTISIG	0xae	Ejecuta CHECKSIG para cada par proporcionado de firma y clave pública. Todos deben coincidir. Un error en la aplicación extrae un valor adicional, prefijar con OP_NOP para solucionarlo.
OP_CHECKMULTISIGVERIFY	0xaf	Igual que CHECKMULTISIG, después OP_VERIFY para parar si no es VERDADERO

No-operadores muestra símbolos que no son operadores

Table 8. No-operadores

Símbolo	Valor (hexadecimal)	Descripción
OP_NOP1-OP_NOP10	0xb0-0xb9	Hacer nada, ignorados

Códigos OP reservados para uso interno por el parser muestra códigos de operador reservados para ser usados por el analizador interno de scripts.

Table 9. Códigos OP reservados para uso interno por el parser

Símbolo	Valor (hexadecimal)	Descripción
OP_SMALLDATA	0xf9	Representa un campo pequeño de datos
OP_SMALLINTEGER	0xfa	Representa un campo de entero pequeño

OP_PUBKEYS	0xfb	Representa campos de clave pública
OP_PUBKEYHASH	0xfd	Representa un campo de hash de clave pública
OP_PUBKEY	0xfe	Representa un campo de clave pública
OP_INVALIDOPCODE	0xff	Representa cualquier código OP que no esté asignado actualmente