Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA (SG)

Grado en Ingeniería Informática de Servicios y Aplicaciones

Estructura y tecnologías de la criptomoneda Bitcoin

Alumno: Ignacio Aranguren Cabezón

Tutor: Fernando Díaz Gómez

Índice de contenidos

Propiedades

Ecuaciones

Ilustraciones

Ejemplos

Tablas

1. Introducción

El presente trabajo de fin de grado tiene como objetivo fundamental dotar al lector de una idea completa de cuáles son las tecnologías involucradas en la criptomoneda *Bitcoin*. Se proporcionará una visión exhaustiva de todos los elementos que interoperan en la red, proveyendo de explicaciones, diagramas e ilustraciones de los conceptos a desarrollar.

Este trabajo no se centrará en las aplicaciones que estas tecnologías puedan tener más allá del ámbito puramente técnico. Los aspectos monetarios, financieros o económicos no se tratarán más allá de lo estrictamente necesario, para proveer de contexto durante la explicación.

Por tanto, el público al que se dirige el trabajo, son aquellas personas con una formación en ingeniería informática o similar. Se discutirán conceptos relacionados con los campos de estructuras de datos, criptografía, redes de computadores y sistemas de compiladores. Durante la presentación de los contenidos, el lector podrá relacionar los conceptos de las áreas anteriormente descritas, para, finalmente, obtener un entendimiento completo de la interconexión de éstas y de sus interdependencias. La visión que se obtendrá al final de la explicación será total; el lector tendrá las herramientas necesarias para adentrarse más profundamente y con rotunda confianza en las sutilezas de cada una de las tecnologías desarrolladas.

En los primeros apartados se explicará el proceso de desarrollo por el cual se ha elaborado la presente documentación. Se explicarán las etapas por las que el proceso de desarrollo ha evolucionado. También, se proporcionará, mediante una serie de diagramas, cuál ha sido la planificación y cómo se ha desarrollado a lo largo del periodo de elaboración global del proyecto.

La explicación del contenido sigue una estructura de menos a más, explicando los elementos más fundamentales primero, para después introducir los elementos más complejos de *Bitcoin.* De esta forma, el lector construirá una base sólida antes de adentrarse, por completo, en las tecnologías inherentes y originales de *Bitcoin*.

Por último, se presentará un apartado de conclusiones, donde se expresará la opinión del autor hacia el tema de desarrollo. En dichas conclusiones se tratará de establecer cuáles son las ventajas y desventajas más significativas, y cuáles son los retos que *Bitcoin* tendrá que afrontar en el futuro.

2. Metodología de trabajo

Este apartado recoge los procedimientos por los cuales se ha realizado el trabajo de fin de grado. La elección de una metodología fija desde el principio del proyecto facilitará la obtención de resultados de calidad y rigurosos.

Se presentarán varios apartados, en los cuales se tratarán aspectos relevantes para la consecución del anterior objetivo.

2.1. Ciclo de vida

En este apartado se presenta la metodología usada en el presente trabajo. Se utilizará una metodología evolutiva basada en un ciclo de vida iterativo, dado que la manera más efectiva sobre la cual se podrá construir una explicación completa y detallada de cada módulo de *Bitcoin*, es, explicando exhaustivamente cada uno de ellos, proveyendo en cada iteración más profundidad a las explicaciones.

Cada iteración consta de tres etapas fundamentales:

- **Investigación:**
  Recopilación de los aspectos sobre los que se desarrollará en cada iteración. Dependiendo de la iteración, se recopilará bibliografía o se investigará sobre una fuente ya derivada.

  El proceso se realizará de una forma incremental, empezando por la investigación de los elementos más básicos de las tecnologías que están involucradas en *Bitcoin*, para después construir sobre estas bases y poder abordar de una manera efectiva todos los conceptos de estas tecnologías.

- **Elaboración:** Proceso de desarrollo y explicación de contenidos.

  El proceso de desarrollo esta formado por varios módulos. Cada módulo representa un tema sobre el cual se desarrollará. Una vez realizado el proceso de investigación en un módulo, se procederá a la elaboración de su contenido. Se dará paso al inicio de la elaboración del siguiente módulo cuando se considere que la explicación y los elementos introducidos son adecuados para proseguir al siguiente módulo.

- **Revisión:** Comprobación de los dos anteriores apartados.

  Cada vez que se realice un borrador en una iteración se realizará un proceso de revisión de los contenidos del mismo. El proceso de revisión anterior se complementará con la revisión periódica, por parte del tutor, del trabajo, en su estado actual.

El proceso de desarrollo consta de **tres** iteraciones:

- **Fase 1:** Estudio de viabilidad
- **Fase 2:** Análisis y estructura de contenidos.
- **Fase 3:** Elaboración de contenidos técnicos.

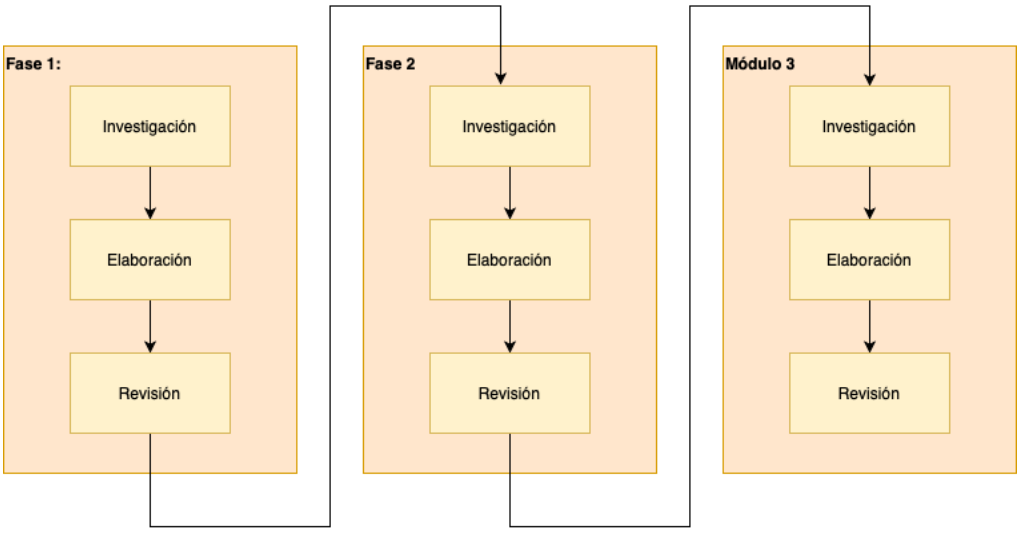


Ilustración 2-1 Ciclo de vida

2.2. Planificación

La planificación del proyecto consta de tres fases fundamentales (cada una representan una iteración (Ilustración 2-1)):

*Estudio de viabilidad: **(1 diciembre 2020 – 20 de diciembre 2020)***

Dedicación parcial: 2 horas diarias en días laborables. 3 horas diarias fines de semana.

Se realizará un estudio de la viabilidad del proyecto. Para este propósito se leerá por primera vez la guía: *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction.*

Análisis de estructura y contenidos: ***(1 de mayo 2021 – 20 de mayo 2021)***

Dedicación parcial: 4 horas diarias.

Para ello se recopilará información y bibliografía para determinar cuáles son los aspectos más relevantes sobre los que se tratará en el trabajo.

Para este propósito se estudiará la guía: *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction.* Esta fuente proporcionará una visión de alto nivel para el desarrollo de conceptos más complejos. Un aspecto muy relevante de esta fuente es la propuesta bibliográfica que tiene, a partir de la cual se puede profundizar en una amplia variedad de temas.

Después de realizar este proceso se deberá tener una visión clara, de alto nivel, de los temas sobre los que se van a desarrollar.

Elaboración de contenidos técnicos: ***(20 de mayo 2021 – 18 de junio 2021)***

Dedicación total: 6 horas diarias.

Para la elaboración de contenidos técnicos se hará uso de la guía técnica de referencia: *Mastering Bitcoin: Unlocking Digital Crypto-Currencies.*

Mediante esta fuente, se obtendrá gran detalle sobre los conceptos técnicos a desarrollar, los cuales se procesarán y adaptarán para ser incluidos en el trabajo. Además, se proporcionará un ejemplo aplicado mediante la instalación de un cliente *Bitcoin*.

Diagrama de Gant

Fase 1	DICIEMBRE		
	Semana 1	Semana 2	Semana 3
Fase 1: Introducción.			
Fase 1: Criptografía y estructura de datos.			
Fase 1: Transacciones y red.			

Fase 2	MAYO			
	Semana 1	Semana 2	Semana 3	Semana 4
Fase 2: Introducción y contexto.				

Fase 2: Criptografía de curvas elípticas, funciones *hash* y estructuras de datos.				
Fase 2: Transacciones: Script, scripts de bloqueo y desbloqueo, y tipos.				
Fase 2: Red y minería				

Fase 3	JUNIO			
	Semana 1	Semana 2	Semana 3	Semana 4
Fase 3: Criptografía de curvas elípticas, funciones *hash* y estructuras de datos.				
Fase 3: Transacciones: Script, scripts de bloqueo y desbloqueo, y tipos.				
Fase 3: Red y minería.				
Fase 3: Cliente Bitcoin core y revisión.				

	Fase 1	Fase 2	Fase 3	TOTAL
Numero total horas	46	80	174	300

2.3. Estructura del contenido

El contenido del trabajo se divide en módulos. Un módulo representa una entidad fundamental, dentro del contexto de la tecnología *Bitcoin*. Los módulos que se tratarán en este trabajo son los que siguen:

1) **Etapa de contexto histórico de *Bitcoin*:** En este apartado se dotará de la motivación por la cual una criptomoneda con el esquema de *Bitcoin* es necesaria, y cómo su tecnología se ha desarrollado a lo largo de la historia.

2) **Etapa criptográfica:** En este apartado se establecerá los conceptos criptográficos sobre los cuales se apoya *Bitcoin*. El trabajo de investigación se realizará ampliando conceptos obtenidos a lo largo del grado, en asignaturas relacionadas, complementándose con los obtenidos a partir de los contenidos bibliográficos.

3) **Etapa claves criptográficas:** Se establecerán los procesos mediante los cuales se dan soporte a determinadas funcionalidades de *Bitcoin*. Estos procesos, envuelven la derivación de claves criptográficas para propósitos esenciales para el funcionamiento de la criptomoneda.

4) **Etapa de estructuras de datos:** Se realizará un estudio completo de todas las estructuras de datos fundamentales de *Bitcoin,* relacionándolas con conceptos ya introducidos en asignaturas relacionadas del grado.

5) **Etapa de transacciones:** Se realizará un estudio exhaustivo de la forma en la que los *Bitcoins* cambian de propietario. Se establecerá cuáles son las transacciones más utilizadas actualmente y cómo se hacen efectivas.

6) **Etapa de red:** En esta etapa se relacionarán conceptos ya adquiridos en el grado en materia de redes de computadores, y se expandirán, proveyendo de una visión novedosa de interconexión entre distintas partes en una red.

7) **Etapa de minería:** Quizás esta es la etapa que recoge el aspecto más relevante de *Bitcoin*; el elemento más innovador que implementa esta criptomoneda; un instrumento de ingeniería social, monetaria, y tecnológica sin precedentes.

8) **Etapa de instalación de cliente Bitcoin:** En esta etapa se proveerá al lector de una serie de herramientas prácticas.

2.4. Herramientas utilizadas

El presente trabajo de fin de grado se ha realizado apoyándose fundamentalmente en las herramientas:

- **Microsoft Word:** Toda la documentación, contenidos… se han elaborado mediante el uso del editor de texto Microsoft Word 2016.

- **DrawIO:** Todos los diagramas son originales y se han realizado mediante el software de edición en línea de diagramas *DrawIO*.

- **Fuentes principales:** Los contenidos y organización del trabajo se han estructurado y basado principalmente en dos fuentes:

    - Contenidos técnicos y estructura:

      Antonopoulos, Andreas M. (April 2014). Mastering Bitcoin: Unlocking Digital Crypto-Currencies. O'Reilly Media.

    - Estructura:

Arvind Narayanan. Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press.

3. Dominio de aplicación de Bitcoin

Durante las dos últimas décadas el termino criptomoneda o moneda digital ha pasado del contexto puramente académico-técnico al popular, dando inicio a una de las revoluciones digitales con más arraigo y de más profundo impacto desde la propia creación de la *World Wide Web*. Términos como *blockchain* (cadena de bloques en su traducción literal al castellano), descentralización, o minería de criptomonedas, han pasado a formar parte del lenguaje coloquial en conversaciones fuera del ámbito de la informática o la economía.

En este trabajo de fin de grado se abordan las tecnologías fundamentales de la criptomoneda más conocida hoy en día: el *Bitcoin.* ¿Cómo funciona el consenso implícito a mas bajo nivel?, ¿cómo se incentiva a los participantes de la red? y, ¿qué es el *blockchain?* son solo algunas de las preguntas sobre las cuales se desarrollará el presente trabajo de fin de grado.

Como parte introductoria y para empezar a dar un poco de contexto al lector acerca del tema que concierne, empezaremos respondiendo a la pregunta: ¿Qué es *Bitcoin*?

La criptomoneda *Bitcoin* nace en 2008 como obra de un individuo anónimo conocido popularmente como Satoshi Nakamoto, en un intento de crear una moneda virtual descentralizada inherente y nativa al propio internet. En la representación más simplificada de la criptomoneda *Bitcoin*, se podría abstraer a la siguiente terna: red, minero y protocolo.

Ilustración 3-1 Elementos generales Bitcoin

Las tecnologías que implementa la red no son excesivamente complejas (realmente casi ningún elemento en *Bitcoin* lo es), se trata simplemente de una red P2P (*Peer to Peer)* donde cada nodo actúa a su vez de cliente, enviando una serie de transacciones a los demás nodos, y de servidor, escuchando por transacciones de los demás nodos de la red.

Los nodos *Bitcoin*, por tanto, son iguales los unos a los otros; no hay diferencia alguna en cuanto a capacidades ni funcionalidades más fundamentales. No existen nodos autoritarios que regulen de ningún modo el funcionamiento de la red, y es justamente aquí donde radica el fundamento básico de esta tecnología: se trata de una moneda descentralizada; sin ningún regulador central. ¿Pero qué quiere decir esto exactamente?

Pensemos de la siguiente forma: Las monedas tradicionales (*fiat*) actuales, como el USD o EUR, están reguladas por una autoridad conocida como banco central. Esta autoridad da soporte a la moneda y la regula de diversas maneras. Puede emitir más masa monetaria según lo crea conveniente, o por el contrario contraerla, retirando de circulación cierta porción de la

masa monetaria total. Luego, esta entidad tiene el poder sobre la red; ella es la autoridad y sus participantes están sujetos a sus reglas y condiciones.

La principal diferencia de *Bitcoin* y las *fiat* es que la primera no tiene ninguna autoridad central que la regule, sino que es justamente la red en sí misma, sus participantes y protocolo, quienes se encargan de autorregularla y de garantizar su correcto funcionamiento.

Como toda moneda, ya sea tradicional o virtual, tiene que haber una forma de generar valor, alguna forma por la cual una persona esté interesada en formar parte de la red y de contribuir a ella.

Tradicionalmente las divisas han estado respaldadas por lo que se conoce como el patrón oro. Esto no es más que expedir porciones de un bien determinado, como pueden ser billetes o monedas, las cuales representan un subyacente de alta demanda y de estricta escasez. Esta dicotomía es la que hace que una moneda tenga valor: demanda y escasez.

Pues bien, *Bitcoin* no está respaldado ni por el patrón oro, por diamantes, o por cualquier otro subyacente material con las anteriores características. En cambio, *Bitcoin* realiza algo muy inteligente, confiando en la criptografía mediante el uso de las funciones hash y de sus propiedades. La capacidad de cómputo es el subyacente de esta moneda virtual, y es la dificultad de los problemas a resolver por los nodos lo que generan esta escasez tan necesaria.

3.1. Primera aproximación a la minería

La demanda se genera por medio del proceso conocido como minería. Este proceso es una carrera en la que varios actores compiten por generar bloques que se introducen en el *blockchain*.

Como hemos dicho anteriormente, una moneda necesita generar escasez y demanda. La minería se apoya en las funciones *hash* y en sus propiedades criptográficas para generarlas. Ahora bien, ¿cómo consigue una función *hash* controlar el flujo de creación de *Bitcoins*?

Para responder a la pregunta, necesitamos entender qué hace una función *hash*. A continuación, se provee de una explicación de muy alto nivel sobre qué es una función *hash* y por qué su uso es tan adecuado en *Bitcoin*.

Todo se reduce a los espacios de salida y llegada de estas funciones:

El espacio de salida de las funciones *hash*, es el conjunto de todas las combinaciones de bits posibles y de longitudes arbitrarias. Este espacio es infinito. Sim embargo, el espacio de llegada es finito, pero lo suficientemente grande para que dos mismos valores en este espacio sean altamente improbables de hallar. El *hash-puzzle* o *proof of work* consiste en encontrar para un cierto valor del espacio de salida, un valor correspondiente en el espacio de llegada que sea menor que un *target* u objetivo.

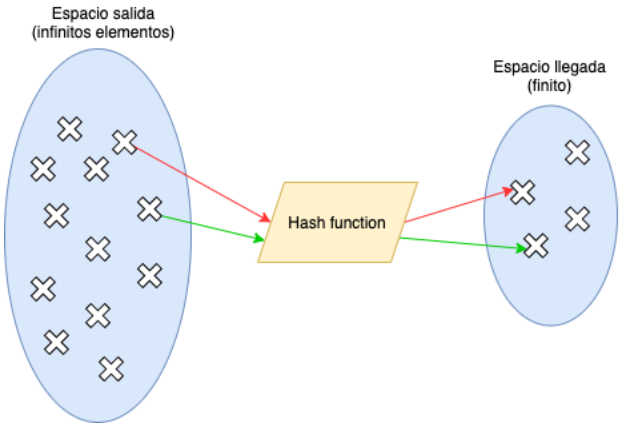


Ilustración 3-2 Funciones hash

En la Ilustración 3-2, se puede observar la representación gráfica de la derivación de dos elementos del espacio de salida, y sus correspondientes valores en el espacio de llegada. Se muestran dos resultados de aplicar la función de *hash* a dos elementos del espacio de salida. En uno se indica que, para ese elemento, su correspondiente valor, después de aplicar la función de *hash*, no recae sobre el *target* indicado (flecha en rojo) en el espacio de llegada. Todo lo contrario, para el elemento con flecha verde: en este caso después de aplicar la función de *hash* a este elemento del espacio de salida, el resultante elemento en el espacio de llegada, sí recae sobre un *target* indicado. Básicamente, que un valor recaiga sobre el *target*, quiere decir que el *hash* calculado está por debajo del valor numérico que representa el target.

Por tanto, si se extrapola este ejemplo a la realidad, se tendrá un valor que cumple el *hash-puzzle* o *proof of work*, lo que supone la posibilidad de proponer el siguiente bloque en el *blockchain*, y lo que es mejor, ganar varios *Bitcoin* o, lo que es lo mismo, una buena suma de EUR o USD.

Por tanto, la dificultad de resolver una serie de cálculos y obtener un valor que cumpla los requisitos anteriormente presentados son los que generan la escasez en *Bitcoin.* La demanda se crea según más personas estén dispuestas a participar directa o indirectamente en la red. Además, la dificultad es dinámica, ya que varía a lo largo del tiempo según las tecnologías hardware mejoran y, por tanto, el poder de computación. También, el tiempo medio en el que los bloques deben ser minados es de 10 minutos. Cabe destacar que no hay nada especial en el número de 10 minutos, podría haber sido 6 y la red probablemente hubiera funcionado correctamente.

Como primera aproximación, en este apartado se han presentado unas explicaciones al más alto nivel del proceso de minería de bloques. En los siguientes capítulos se explicará cómo se utilizan dichas funciones, cómo logran generar este valor tan crítico para la viabilidad de toda moneda, ya sea virtual o tradicional, cómo se ajusta la dificultad para que un bloque sea minado en 10 minutos, y qué se recoge en un bloque.

3.2. Minero

De una forma coloquial diremos que probablemente el lector ya sabe "por donde van los tiros" y puede empezar a vislumbrar una idea de quiénes son los encargados de generar este valor "virtual". Si el lector es mínimamente perspicaz estará en lo cierto y sabrá a quién nos referimos: los mineros.

Los mineros son usuarios de la red que operan un nodo completo. Estos usuarios, a través del uso de su poder de cómputo, resuelven los *hash-puzzles* y crean *Bitcoin.* También, son los encargados de incluir en el *blockchain* y validar las transacciones que reciben de los demás nodos.

Los incentivos de participar en la minería de *Bitcoin* son más que evidentes: actualmente el cambio de *Bitcoin* por USD oscila entre los 30.000 $ - 50.000$. Eso sí, hay que destacar, que también la dificultad de minar un bloque es muy grande y requiere de una gran potencia de cómputo como se ha explicado anteriormente. Por tanto, los mineros actuales (aquellos que de verdad tienen alguna probabilidad de minar un bloque) son pocos y poseen unos equipos e inversión no apta para todo el mundo.

3.3. Primera aproximación al protocolo *Bitcoin*

Por otro lado, el protocolo *Bitcoin* es una serie de estándares que deben operar los nodos de la red para que éstos sean aptos para participar en el consenso implícito, proponer y recibir transacciones, y minar bloques. Hay diferentes versiones de protocolos con amplia variedad de características y funcionalidades, pero en todas ellas los fundamentos son los mismos indicados anteriormente. Esto quiere decir que, aunque haya diferentes versiones de protocolos que operan en diferentes nodos, las funcionalidades de *Bitcoin* básicas son las mismas.

Por tanto, el protocolo aúna los elementos básicos para el funcionamiento de la red, e implementaciones adicionales que se han ido proponiendo a medida que el ciclo de vida de la moneda ha ido avanzando. Todo ello con el fin proveer mejoras, las cuales solventan problemas previamente no abordados o innecesarios hasta entonces. Este tipo de mejoras se suelen realizar mediante las Propuestas de Mejora de *Bitcoin* o *Bitcoin Improvement Proposal* (BIP) en inglés.

El protocolo *Bitcoin* se apoya ampliamente en la criptografía: criptografía asimétrica y simétrica, funciones de *hash*, firmas digitales, etc. Estos términos serán tratados en los siguientes capítulos en detalle, y a medida que se avance con la explicación, el lector tendrá en su mano los conocimientos básicos para poder abordar los conceptos más complejos del funcionamiento de *Bitcoin* de forma exitosa.

Por tanto, *Bitcoin* es la combinación de múltiples tecnologías, apoyadas las unas en las otras, y que proporcionan instrumentos para hacer de esta moneda virtual una criptomoneda viable. En este trabajo de fin de grado, se explican cuáles son estas tecnologías, cómo funcionan y que papel juegan en la gran red de *Bitcoin*. El objetivo fundamental es dotar al lector de una base sólida del funcionamiento de la red, y de proveerle de las herramientas para adentrarse en el mundo de *Bitcoin*.

4. Antecedentes históricos

El camino de *Bitcoin* puede parecer que haya sido inmediato y que no haya habido proyectos anteriores de criptomonedas o medios de pagos diferente. En realidad, *Bitcoin* es la heredera de una serie de anteriores criptomonedas y diferentes formas de pago que allanaron el camino en sentido comercial y tecnológico. En esta sección describiremos algunas de ellas para ver el contexto en el que se halló Satoshi Nakamoto durante el desarrollo de *Bitcoin*.

A lo largo de la historia, el intercambio de bienes entre diferentes partes ha sido primordial y necesario, dando paso a tres esquemas fundamentales en los cuales se puede intercambiar bienes tangibles o intangibles: sistemas basados en intercambio, crédito, o monetarios.

2.1. Diferentes sistemas

Ninguno de los anteriores sistemas es claramente mejor que el otro, pero sí es cierto que, en el caso del sistema basado en trueque, una persona que participa en una transacción asume el riesgo de quedarse con un bien estancado:

Supóngase por ejemplo que María quiere intercambiar una naranja por un pomelo con Juan. Ambas partes están de acuerdo en el trueque de ambos vienes. El trueque se realiza y ahora Juan tiene una naranja y María tiene un pomelo. Ahora se supone lo siguiente: ¿Qué pasaría si María era la única persona en el mundo que le gustan los pomelos y Juan no quiere volver a recuperar el pomelo? María tiene un bien del que nunca se va a poder deshacer y esencialmente ha asumido una perdida debido a que la naranja sí es intercambiable.

También en este mismo esquema, la dificultad de tasar el precio de un determinado bien, y de intercambiarlo por uno de coste similar, es una tarea ardua y muchas veces difícil de llevar a cabo. Hay muchos ejemplos de este tipo de sistemas en el mundo de la informática, como pueden ser las herramientas de intercambio de archivos: *Utorrent*, *BitSharing*, *Emule*...

En cuanto a los sistemas de crédito, una desventaja de participar en dicho sistema es la posibilidad de no poder nunca liquidar esa deuda con su emisor:

Supóngase que María necesita desesperadamente una regadera porque sus plantas están muriendo y no hay pronostico de lluvia en los próximos días. Juan, su vecino, tiene una regadera, pero él, por el contrario, necesita rápidamente zapatillas nuevas para participar en la carrera de la cuidad el mes que viene. María habla con Juan y le dice que le dé la regadera y que ella le conseguirá, también, de la misma, forma unas zapatillas. Ambos aceptan la

transacción y ahora María ha asumido una deuda con Juan. Pero ¿qué pasa si llega el mes que viene y María no ha encontrado a nadie que quiera darle unas zapatillas? Juan estará furioso porque no podrá participar en la carrera y María habrá fallado su deuda. Juan podría decirle a toda la gente de su vecindario que María no salda sus deudas y María podría verse en serias dificultades porque nadie estaría dispuesto a participar con ella en una futura transacción.

Estas explicaciones, aunque sean muy básicas, muestran una idea aproximada de cuáles son las desventajas que se pueden encontrar en los sistemas de crédito y trueque. A continuación, se presenta la historia y desarrollo de los sistemas basados en crédito y monetarios.

2.2. Sistemas basados en crédito

Hoy en día el medio de pago más utilizado en internet es, sin duda alguna, los sistemas basados en crédito. Seguramente el lector ha hecho más de una vez una compra en algún *E-commerce* y no ha dudado en entregar los detalles de su tarjeta de crédito al sitio en cuestión.

Puede que actualmente dichos datos se entreguen tranquilamente, dado que el usuario confía en el vendedor; ya sea porque es ampliamente utilizado, tiene alguna garantía gubernamental, o se apoya en tecnologías que garanticen la privacidad del usuario. Sim embargo, hace 30 años cuando la *World Wide Web* estaba todavía en su niñez y los protocolos de cifrado seguro *HTTPS*, *TLS* o *SSL* estaban todavía por implementar o implementándose, los usuarios no estaban tan dispuestos, ni mucho menos, a entregar sus datos bancarios a ningún sitio web.

Esta desconfianza dio paso al desarrollo de herramientas que usaban una estructura de intermediario (similar a la actual de *PayPal*), donde el comprador entregaba los datos de sus tarjetas de crédito al intermediario y, de la misma forma, el vendedor contactaba con el intermediario con los detalles del producto a vender. Finalmente, el intermediario era el encargado de liquidar los balances de ambas partes de la transacción, comprobar los datos de las transacciones, y notificar que la transacción se había realizado de forma exitosa.

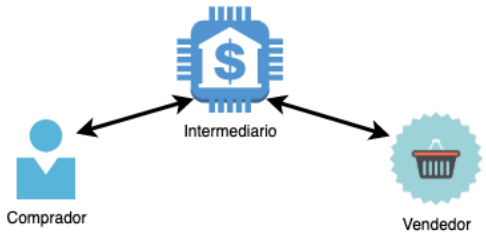


Ilustración 4-1 Esquema sistema basado en crédito

Pero esta forma de pago carecía de anonimidad: el comprador tenía que compartir los datos de la transacción con el vendedor.

Para solventar este problema, a principios de los años noventa, hubo una innovadora propuesta de una forma de pago mediante intermediario que evitaba el envío de los detalles de los datos de la transacción al vendedor: las *SET*.

Las *SET* funcionaban de la siguiente forma: el comprador disponía de una aplicación de escritorio donde se cifraban todos los datos de la transacción. Las claves de descifrado eran exclusivamente conocidas por el intermediario y el propio comprador. El usuario enviaba el mensaje cifrado al intermediario, el que a su vez redirigía estos datos cifrados al vendedor. El vendedor aceptaba ciegamente los datos cifrados del comprador, cifraba su parte de la transacción y la enviaba al intermediario. Una vez el intermediario contaba con ambos criptogramas (los del comprador y vendedor) los descifraba, y si los datos de las transacciones de ambas partes coincidían (recordemos que el intermediario era el único participante que, además del propio creador del criptograma, conocía las claves de descifrado) la transacción se daba por valida y el pago se efectuaba.

Uno de los grandes problemas de este método de pago es que todos los participantes debían contar con un certificado propio, una tarea muy tediosa de llevar a cabo, dado que un comprador corriente no tiene por qué saber que es un certificado, ni para qué se utiliza. Este hecho supuso el gran inconveniente de las *SET* y la razón por la cual no acabaron asentándose.

Como se verá más adelante, *Bitcoin* evita el problema del uso de certificados mediante las identidades anónimas que no son mas que *hashes* de claves públicas.

2.3. Sistemas monetarios

Una de las grandes diferencias entre los sistemas de intercambio o deuda y los sistemas monetarios es el anonimato. En los dos primeros, el intermediario siempre va a tener algún indicio que pueda identificar a los participantes en una transacción, pero en cambio, en los sistemas monetarios, el anonimato es total. El dinero (en su sentido material) es completamente anónimo; un Euro en condiciones normales no proporciona detalle alguno sobre los participantes en una transacción.

Bitcoin logra el anonimato hasta un cierto punto. Es cierto que las direcciones *Bitcoin* son anónimas, pero mediante el uso de sofisticados algoritmos, es posible determinar patrones que sugieran que unas determinadas transacciones pertenecen a una persona en concreto. Estos algoritmos no darán una certeza completa de la identidad de un usuario de la red, pero sí que darán con un alto porcentaje de seguridad que existen un conjunto de transacciones que sugieren que se están comportando como Juan.

El primer intento serio en proponer una moneda virtual llego de la mano del criptógrafo David Chaum en 1983. Pero primero pensemos en cómo funciona una divisa tradicional:

Un billete no es más que en un trozo de papel que una determinada entidad regulatoria afirma que posee un cierto valor. La principal característica que debe tener una determinada moneda es que esta sea infalsificable, en caso contrario su uso sería absurdo, dado que todo el mundo sería capaz de crear su dinero propio. Chaum, de la mano de la criptografía, aborda este problema de forma satisfactoria, proporcionando a su moneda un identificador único infranqueable; indetectable para los demás participantes. La propuesta fue un gran paso para el desarrollo de las criptomonedas actuales, pero insuficiente dado que confiaba en una autoridad central para su regulación y emisión.

En 1988, otra vez Chaum y de la mano de otros dos criptógrafos, Amos Fiat y Moni Naor, abordaron el problema de la falsificación de las monedas virtuales de forma novedosa. La pregunta que se hicieron fue la siguiente: ¿Y si en vez de tratar de evitar la falsificación, tratamos de detectarla? Esta idea fue el fundamento de uno de los grandes problemas que *Bitcoin* solventa: prevenir el *double-spending.*

El *double-spending* sucede cuando una persona gasta una misma moneda más de una vez. Una estrategia eficiente de detección o prevención de este fraude es primordial en el desarrollo de una criptomoneda. Si una moneda no proporciona soluciones a este problema, su éxito será inalcanzable y fracasará.

Chaum, Fiat y Naor propusieron lo siguiente: pongamos que una persona es propietaria de una moneda en la cual cifra su identidad. Ella es la única que puede reconocer dicha identidad cifrada, ya que posee la clave de descifrado. La moneda al ser gastada requiere que el propietario descifre una porción aleatoria del criptograma. ¿Qué pasa si el propietario decide gastar la moneda dos veces? La segunda vez sucederá exactamente lo mismo: el propietario tendrá que descifrar otra porción, la cual es altamente improbable que sea la misma anteriormente descifrada. La ingeniosa idea de este proceso recae en que cada vez que el receptor recibe la moneda, éste tiene que ir a la autoridad central a liquidar el pago. Y es precisamente la autoridad central la que, al juntar las porciones del criptograma, sabrá si se ha usado la misma moneda en diferentes transacciones. Esto sucede debido a que una porción del criptograma no es suficiente para comprometer la identidad del propietario, pero con dos porciones del criptograma, la cosa cambia: habrá una alta probabilidad por la cual la identidad del propietario pueda verse comprometida, ya que dos porciones del criptograma son suficientes para obtener datos de la identidad del propietario. Por tanto, si la autoridad central puede obtener la información del propietario de una moneda, será acusada de haber realizado un *double-spending*.

Otra diferencia de este modelo frente a *Bitcoin* era que, a pesar de que los propietarios de las monedas, aquellos que la gastaban, sí eran anónimos, los receptores no, dado que éstos debían volver a la autoridad central para liquidar la moneda y para que tuviera constancia del pago.

Además de lo anterior, esta moneda no era fraccionable. No había forma alguna de separar una moneda de 20$ en una de 15$ y 5$, sino que había que volver a la entidad reguladora para efectuar este cambio

Mediante la implementación del anterior esquema, la primera compañía que trato de solventar el problema de los pagos online surgió: *DigiCash*. *DigiCash* no se consolidó debido a la baja aceptación que tuvo tanto en la parte del comprador, como en la parte del vendedor; pero a pesar de su fracaso, sentó las bases de muchos conceptos que fueron adoptados por futuras criptomonedas.

2.4. Merkle trees

En 1991, Tatsuaki Okamoto y Kazuo Ohta propusieron un esquema mediante el cual posibilitaron la subdivisión de una misma moneda mediante el uso de *merkle-trees*. Como se verá más adelante, el *merkle-trees* es una tecnología fundamental en la red *Bitcoin* usada para la validación de transacciones en un mismo bloque.

Otra compañía que contribuyó notablemente a otra parte diferente en el desarrollo de una de las tecnologías más fundamentales de *Bitcoin* fue *HashCash*.

HashCash estaba orientada a limitar la capacidad de los *spamers* para generar correos electrónicos basura. Aunque la motivación de *HashCash* era diferente a la de *Bitcoin*, su uso de las funciones *hash* y de sus propiedades, sentaron las bases del modo en el cual *Bitcoin* genera la escasez que da valor a su moneda. Toda aquella persona que quisiese enviar un correo debía realizar una serie de operaciones de *hash*, asumibles para cualquier ordenador común, pero que, para un *spamer,* el cual envía cientos de miles de correos en un periodo de tiempo reducido, dicha tarea se volvía inasumible.

Recordemos que los bloques de *Bitcoin,* aquellos que se introducen el famoso *blockchain*, también realizan operaciones de *hashing* con el fin de resolver *hash-puzzles* para proponer el siguiente bloque. La naturaleza inherente y compleja de los *hash-puzzles,* hace que los bloques sean muy costosos de implementar y que requieran un tiempo medio de cálculo de 10 minutos. Estos dos hechos, como sabemos, hacen que el *Bitcoin* sea por naturaleza escaso y que generen cierto valor. También como se ha indicado anteriormente los incentivos para participar en la red deben de ser atractivos, si no nadie estaría dispuesto a poner en riesgo su capital y capacidad de cómputo para dicho fin.

Otra tecnología fundamental sobre la que Bitcoin se asienta es el *blockchain*. Ésta tampoco se trata de una tecnología novedosa. Una de las primeras implementaciones del *blockchain* fue para la validación de fechas de documentos mediante el esquema *Haber y Stornetta* propuesto en 1991.

En este esquema se hace uso de *timestamps*, los cuales son simplemente marcas de tiempo incrementales que representan el momento en el cual los archivos fueros creados. Un bloque en este esquema de *blockchain*, es un fichero acompañado de un *timestamp*, una firma digital que firma el documento y el *timestamp,* y un *hash* de un puntero que apunta al bloque anterior.

Lo más interesante de este modelo es que cada elemento posterior valida la integridad de todos los anteriores, dado que modificar tan solo un bit de un elemento anterior dará como resultado un puntero *hash* diferente debido a las características de prevención de colisiones de las funcione de *hashing*, las cuales se explicarán más adelante.

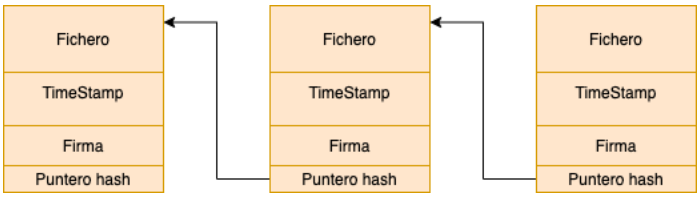


Ilustración 4-2 Timestamp blockchain

En este capitulo se han discutido algunas de las tecnologías y las correspondientes empresas que las implementaron y que han llevado a *Bitcoin* a convertirse en lo que actualmente es. El proceso fue una combinación de esfuerzo e inteligencia, que han hecho que actualmente el futuro de las monedas digitales sea alentador y pujante. Grandes entidades financieras como JP Morgan, Santander, BBVA… están dotando de gran apoyo monetario a *Bitcoin*, lo que

supone que *Bitcoin* haya pasado meramente de las publicaciones académicas y los proyectos experimentales, a alternativas reales a las divisas tradicionales.

En los siguientes capítulos se tratarán en profundidad con un enfoque técnico, dotando de diagramas e implementaciones de cómo funcionan los conceptos vagamente explicados hasta ahora. Al final del presente trabajo de fin de grado el lector tendrá una idea completa y técnica de cómo funciona *Bitcoin.*

3. Elementos criptográficos

En este apartado se discutirán en profundidad los elementos criptográficos usados en *Bitcoin*, los cuales son fundamentales y básicos para poder abordar el funcionamiento real de la red.

La criptografía implementada en *Bitcoin* se basa en criptografía de clave pública, la cual es un tipo de criptografía asimétrica.

3.1. Criptografía de clave pública

Los criptosistemas de clave pública tienen dos elementos fundamentales: clave privada y clave pública.

La clave pública, como su nombre indica, es conocida por todo el mundo. Esta clave, por tanto, se comparte con la red en su totalidad. Una persona que quiera cifrar un mensaje en texto plano lo hará con la clave pública de alguien en la red, de forma que esta última persona sea la única con la capacidad de descifrar el criptograma mediante su clave privada.

Toda clave pública ha de estar dotada de su clave privada asociada, de lo contrario, no habrá forma alguna de descifrar el mensaje.

Por tanto, la clave publica sirve para cifrar y la clave privada para descifrar.

$$Clave\ pública \rightarrow f(m) = criptograma$$
$$Clave\ privada \rightarrow f^{-1}(criptograma) = m$$

Ecuación 3-1

Si el lector no tiene una formación previa en criptografía puede que esté confuso dado que podría sugerir que, para descifrar el mensaje, simplemente hay que calcular la inversa de f. Pero lo que no sabe es que el cálculo de m a partir del $criptograma$, es un problema que requiere una capacidad de computo inabordable con medios "convencionales" de computación. Estos problemas son los llamados NP difícil (NP hard en inglés).

3.1.1. Problemas NP difícil

La función indicada en Ecuación 3-1, es *one-way* o de sentido único. Esto quiere decir que es fácil de computar en un sentido, pero en altamente compleja de computar en su sentido inverso.

Un ejemplo de problema NP difícil es la factorización de números primos:

Dados dos números primos p y q, supongamos de 100 cifras,

$$N = p * q$$

Ecuación 3-2

El cálculo de N (producto de dos primos) es fácilmente computable, pero la inversa de la función (factorizar N) es altamente compleja de computar. Este problema es en el que se basa el algoritmo de cifrado asimétrico RSA.

Existe también, otro problema NP difícil: el del logaritmo discreto. Este problema tiene múltiples aplicaciones entre ellas la criptografía basada en curvas elípticas.

Dados a, n y m, si trata de buscar x tal que,

$$a^x \equiv m \ (mod \ n)$$

Ecuación 3-3

En este caso, la función *one-way* es la exponenciación modular $a^x \ mod \ n$. La función inversa, por el contrario, sería el logaritmo discreto de m en base a, módulo n, la cual es imposible de computar mediante una estrategia eficiente.

Por tanto, la idea a resaltar es que en un cifrado asimétrico, la función f es fácilmente computable en una dirección, pero infinitamente costosa en la otra.

Todo lo anterior permite que una persona pueda cifrar un mensaje en texto plano con su clave pública, conocida por todo el mundo, y la cual se puede compartir en la red. Pero solamente ella podrá descifrar un criptograma con su clave privada (siempre y cuando la almacene en un lugar seguro). Este hecho es el que hace que la criptografía de clave pública sea tan ampliamente utilizada hoy en día; básicamente la seguridad informática actual está construida sobre esta tecnología.

3.2. Curvas elípticas

Las curvas elípticas proporcionan un método para el cálculo de la clave pública, a partir de una clave privada previamente calculada.

Una curva elíptica es una curva con la siguiente gráfica en el plano $\mathbb{R}^2$:

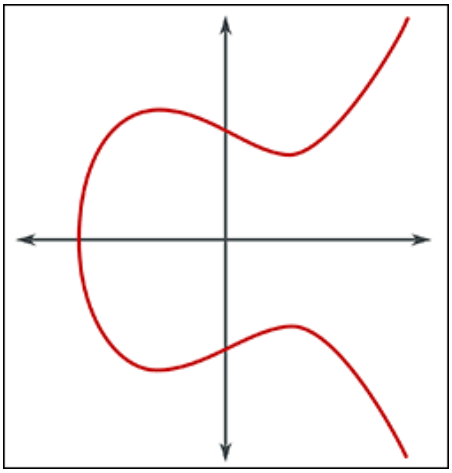


Ilustración 3-1 Una curva elíptica

En concreto la curva que usa el algoritmo de obtención de clave pública de *Bitcoin* es la curva *secp256k1*, definida por la siguiente ecuación:

$$y^2 = (x^3 + 7), \qquad x, y \in \mathbb{F}_p \ ,$$

Ecuación 3-4

donde $\mathbb{F}_p$ es:

$$y^2 \ mod(p) = (x^3 + 7) \ mod(p), \quad p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

Ecuación 3-5

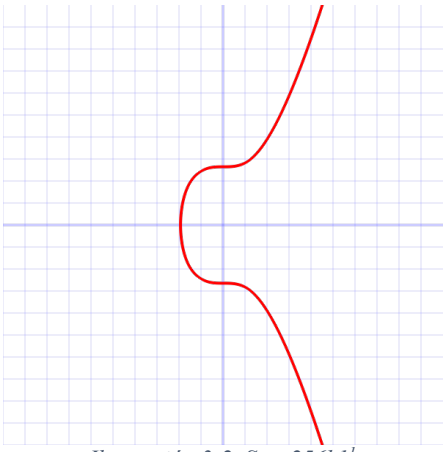


Ilustración 3-2. Secp256k1[1]

Ahora se presentan cuáles son las propiedades de la aritmética sobre puntos en una curva elíptica en un cuerpo finito $\mathbb{F}_p$. Estas propiedades, como se verá más adelante, serán la base de diseño de las herramientas necesarias para el cálculo de la clave pública, y se relacionan a continuación.

[1] Imagen obtenida de https://en.bitcoin.it/wiki/File:Secp256k1.png

Propiedad 3.1

El elemento neutro para la suma en las curvas elípticas se llama “punto al infinito” y realiza la misma función que el 0 en una suma en $\mathbb{R}$.

Propiedad 3.2

La suma de dos elementos en una curva elíptica corresponde a un tercer elemento también en la misma curva elíptica.

$$P_3 = P_1 + P_2$$

Ecuación 3-6

La representación geométrica de lo anterior es crear una línea que una los dos puntos P_1 y P_2. La intersección de la línea con la curva elíptica será el punto P'_3. Una vez calculado este punto, obtenemos la reflexión del punto P'_3 sobre el eje OX y obtenemos el punto P_3.

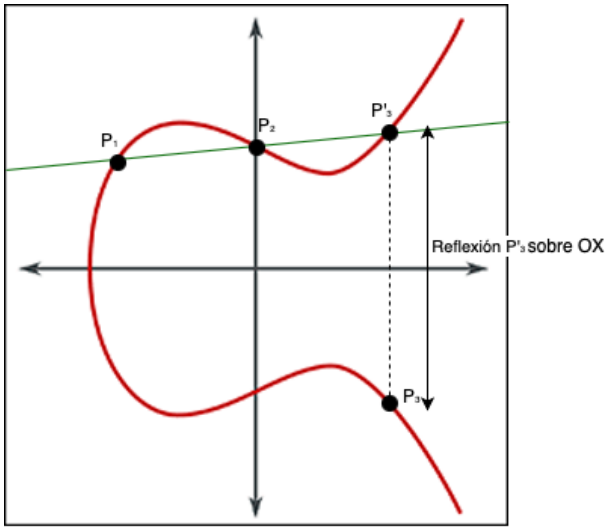


Ilustración 3-3 Suma dos puntos en una curva elíptica

Propiedad 3.3

La suma de un punto, consigo mismo, se obtiene calculando la reflexión respecto el eje OX del punto resultante de calcular la intersección de la recta tangente a la curva elíptica con la curva en un punto $-2P$.

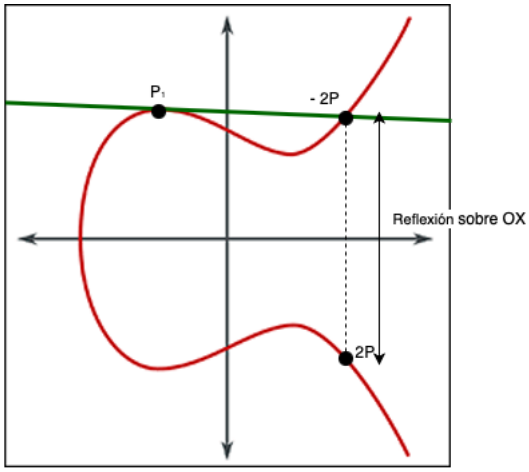


Ilustración 3-4 Suma mismo punto en una curva elíptica

Propiedad 3.4

La suma de elementos en la curva elíptica es asociativa.

Dados A, B y C, tres puntos de la curva elíptica, se cumple que:

$$(A + B) + C = A + (B + C)$$

Ecuación 3-7

Propiedad 3.5

Una vez defina la suma de elementos en la curva elíptica y su asociatividad, se puede también definir la multiplicación de un escalar k por un punto P sobre una curva elíptica:

$$k * P = P + \cdots + P \ (k \ veces)$$

Ecuación 3-8

Realmente esta operación no es más que sumar k veces un punto P. Por tanto, se aplicará k veces, lo explicado en la propiedad en Propiedad 3.3.

Ahora que ya se han introducido las operaciones de suma de dos puntos de la curva elíptica y producto de un punto en la curva por un escalar, se está en condiciones para definir el cálculo de una clave pública a partir de una clave privada k y un generador G (ver 4.2).

3.3. Funciones *hash*

En los anteriores apartados se ha usado frecuentemente términos relacionados con funciones de *hashing*, habiendo proporcionado al lector, en el apartado 3.1, una idea aproximada de qué es una función de *hash*. Aun así y con el fin de ahondar en dicha explicación, en este apartado

se presentan las características fundamentales de estas funciones matemáticas y se explica por qué su uso es tan adecuado en *Bitcoin.*

3.3.1. Características generales de una función de hash:

Propiedad 3.6

Una función de *hash* toma valores como entrada de longitud cualquiera y retorna valores de longitud fija.

Propiedad 3.7

Una función de *hash* puede producir dos salidas iguales para distintos elementos del espacio de entrada. Esto se denomina colisión.

Propiedad 3.8

Se trata de una función de única dirección o *one-way.*

Propiedad 3.9

La complejidad algorítmica del cómputo del valor de salida para una entrada debe ser $O(n)$.

Las propiedades 3.3.1 son generales para todas las funciones de *hash*, y con ellas se podría construir una estructura de datos del tipo de tablas *hash*. Pero para satisfacer los propósitos de *Bitcoin,* se necesitan las siguientes propiedades adicionales para hacer de una función *hash* una función criptográficamente segura:

3.3.2. Características de una función de hash criptográficamente segura:

Propiedad 3.10

En la Propiedad 3.7 se ha presentado el concepto de colisión, pero para que una función de *hash* sea criptográficamente segura, dicha colisión ha de ser imposible de localizar.

Una función *hash* siempre va a tener colisiones, es imposible que una función de *hash* no produzca una colisión por un simple hecho: el espacio de entrada es infinito y el de llegada, finito; por tanto, siempre habrá:

$$x, y \in Espacio\ de\ entrada, \qquad tales\ que\ H(x) = H(y)$$

Ataque de cumpleaños

El proceso de ataque de cumpleaños (*bithday-attack* o *square-root-attack* en inglés*)* consiste en los siguiente:

En una función de hashing ideal con un resultado de n bits, encontrar una colisión requiere $2^{n/2}$ operaciones.

Supóngase la siguiente situación:

Se tiene un espacio de cadenas de bits de longitud variable y una función de *hash* que convierte esas cadenas de bits a unas de tamaño 256 bits. Ahora compute el hash de $2^{256} + 1$ valores diferentes del espacio de entrada, habrá al menos dos *outputs* para los cuales el valor de salida sea el mismo. Lo anterior contempla el peor de los casos. En realidad, es altamente probable (99,8 %) [2]que se encuentre una colisión computando $2^{130} + 1$ valores.

La siguiente propiedad es la más inherente a *Bitcoin* y quizás la más importante, dado que es sobre la que se fundamenta el proceso de minado. Dice lo siguiente:

Propiedad 3.11

Se dice que una función es *puzzle-friendly* cuando para todo posible valor de salida de n bits, si k es elegido de una fuente de azar o de mínima entropía, no será posible encontrar x tal que $H(k \,||\, x) = y$ en un tiempo significativamente menor que 2^n.

Las funciones de *hash* en las que se apoya *Bitcoin* son aquellas de las cuales se sospecha que puede haber colisiones, pero aun no se ha hallado ningún método eficiente para localizarlas. Las funciones de *hash* en las que los protocolos de seguridad informática confían, son aquellas en las que ha habido personas que han intentado localizar colisiones concienzudamente y, aun así, han fracasado.

3.4. SHA-256

El algoritmo de hashing SHA-256 es uno de los más utilizados hoy en día en seguridad informática, y es el algoritmo de *hashing* principal sobre el que se apoya *Bitcoin*. En este apartado se verá cómo éste se implementa, explicando a alto nivel cada uno de sus pasos y procedimientos.

3.6.1 SHA-256: Pre-procesamiento

La función *SHA-256* recibe como entrada, cadenas de bits de longitud variable dentro del rango longitud $0 - 2^{64}$.

El paso del pre-procesamiento consiste en añadir a la cadena de bits, los bits adicionales necesarios para obtener una cadena de bits que tenga una longitud múltiplo de 512.

Este paso consiste en añadir un 1 seguido de tantos 0s sean necesarios para completar la cadena de bits y obtener un múltiplo válido. También, se añade una representación en 64 bits de la longitud del mensaje, la cual se divide en dos partes iguales (x, y) de 32 bits, y se escriben en orden inverso (y, x).

[2] Fuente: "Bitcoin and cryptocurrencies technologies: Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller and Steven Goldfeder".

3.6.2. SHA-256: Inicialización

Se crea lo que se conoce como el vector de inicialización, que no es más que la inicialización de 8 variables de encadenamiento. Estas variables son, por así decirlo, la fuente de alimentación del algoritmo y las que se usan para lanzar el bucle principal.

3.6.3. SHA-256: Bucle principal

El bucle se lanza y se ejecuta para tantos bloques de 512 bits como tenga el mensaje. Estos bloques se denominan como *Words,* y en cada iteración éstos son sometidos a una serie de operaciones. El resultado de cada bloque o *Word* se toma como valor inicial para las variables inicialización del siguiente bloque.

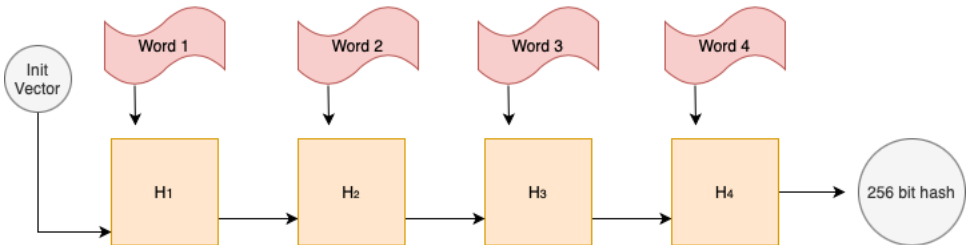


Ilustración 3-5 SHA-256

3.5. RIPEMD160

La función RIPEMD160 nace como mejora de las funciones de hashing MD4, MD5 y RIPEMD. Se espera que esta función sea segura para los próximos 10 años o más [3]. Esta función está basada en la construcción de Merkle–Damgård , y produce un *output* de 160 bits.

No se profundizará en la implementación de esta función para el presente trabajo. Si el lector desea obtener más información acerca de cómo implementar esta función, puede consultar el siguiente enlace: https://en.bitcoin.it/wiki/RIPEMD160

3.6. Firmas digitales

Las firmas digitales sirven para declarar que un cierto mensaje pertenece al emisor que lo firma, y, además, como modo de identificación del firmante. De este modo, una vez el mensaje es firmado, la parte firmante se compromete a lo que indique el contenido. Las firmas digitales solo pueden ser emitidas por el firmante, nadie más puede firmar de la misma manera que otra persona; esto es fundamental. También, las firmas deben de ser imposibles de exportar de un documento a otro, del mismo modo, que no se puede recortar una firma tradicional de un

[3] Fecha de publicación de la fuente 2012: “Bosselaers, Antoon. The hash function RIPEMD-160"

documento y pegarlo en otro. Además, una firma digital debe ser fácilmente calculable y verificable, pero altamente compleja de recrear a partir de la mera observación de dicha clave.

Con el fin de explicar los pasos en un proceso de firma digital se definen los siguientes conceptos:

1) Rúbrica: La rúbrica de un mensaje es otro mensaje que se obtiene cuando una persona cifra dicho mensaje con su clave privada.
2) Firma: La firma es una rúbrica cifrada por la clave pública del otro participante en la transacción.

María (M) quiere firmar un mensaje m y retransmitirlo a Juan (J). Con el fin de aligerar la explicación definiremos las siguientes primitivas:

- $m = mensaje\ en\ texto\ plano.$
- $f_x^{-1} = función\ de\ descifrado\ del\ actor\ x\ mediante\ el\ uso\ de\ su\ clave\ privada.$
- $f_x = funcion\ de\ cifrado\ del\ actor\ x\ mediante\ su\ clave\ pública.$
- $r = rubrica\ del\ mensaje\ m.$
- $s = rúbrica\ cifrada\ con\ una\ clave\ pública.$

El proceso de creación de firma digital consta fundamentalmente de los siguientes pasos:

1) María crea la rúbrica con su clave privada del mensaje m.

   $r = f_M^{-1}(m)$

2) María cifra r con la clave pública de Juan para retransmitirla de forma segura.

   $s = f_J(r)$

3) María retransmite el criptograma s y lo envía a Juan. Juan descifra el criptograma con su clave privada.

   $r = f_J^{-1}(s)$

4) Juan comprueba la firma de María aplicando la clave pública de María a r

   $f_M(r) = f_M\left(f_M^{-1}(r)\right) = m$

5) Juan comprueba que el m obtenido concuerda con el retransmitido por María

Este es el proceso fundamental por el cual se valida la firma de digital de una persona determinada. Como podemos ver, el proceso es sencillo, lo cual satisface la condición de que la firma sea fácilmente computable y verificable.

Todo esto, en *Bitcoin,* sirve para lo siguiente: cuando en una transacción se indica que una dirección en concreto puede reclamar unos fondos, los fondos en cuestión serán reclamables

si, y solo si, algún usuario de la red proporciona una firma valida. Por tanto, la firma digital sirve para desbloquear una transacción bloqueada previamente por el creador de la transacción.

3.7. Elliptic Curve Digital Signature Algorithm (ECDSA)

En el apartado 3.6 se ha visto cómo funciona el proceso de creación y validación de una firma digital. Ahora, se explicará cómo Bitcoin usa el algoritmo de cifrado *ECDSA* (que no es más que un *DSA,* pero usando curvas elípticas) para la creación de firmas digitales. La explicación se hará, dotando al lector de una serie de explicaciones matemáticas de alto nivel sobre el funcionamiento del algoritmo en cuestión.

Supongamos de nuevo que María (M) quiere enviar un mensaje firmado al Juan (J).

Al igual que en 3.6, definimos las primitivas a usar en el proceso:

- $m = mensaje\ en\ texto\ plano.$
- $f_x^{-1} = función\ de\ descifrado\ del\ actor\ x\ mediante\ el\ uso\ de\ su\ clave\ privada.$
- $f_x = funcion\ de\ cifrado\ del\ actor\ x\ mediante\ su\ clave\ pública.$
- $r = rubrica\ del\ mensaje\ m.$
- $k_x^+ = clave\ pública\ de\ x.$
- $k_x^- = clave\ privada\ de\ x.$
- $n = 256$
- $H(m) = hash\ de\ m.$
- $G = punto\ generador\ en\ la\ curva\ elíptica$

El proceso de creación de validación de la firma constará de los siguientes pasos:

1) Generación de la firma digital:

    a. Obtención de claves privadas y públicas: este proceso es el mismo que el que se explicará en los apartados 4.1 y 4.2.

    $$Clave\ publica = (x, y)$$

    b. Calculo $r = f_M^{-1}(H(m) + k_M^- * x)\ (mod\ n)$ si $r = 0$, se reinicia el cálculo y volvemos a computar (a).
    c. La firma digital es el par (x, r)

2) Verificación de la firma:

    a. Juan comprueba que $1 < x, r < n$, y calcula $H(m)$ y $w = r^{-1}\ (mod\ n)$.
    b. Cálculo $u_1 = H(m) \cdot w\ (mod\ n)$ y $u_2 = x \cdot w\ (mod\ n)$
    c. Cálculo del punto resultante en la curva elíptica $u_1 G + u_2 k_B^+ = (x', y')$, tomando como $resultado = x'\ (mod\ n)$

d. Comprueba si resultado es igual a la coordenada x del punto de la clave pública de Juan.

4. Direcciones Bitcoin

Tradicionalmente, cuando se ejecutaba una transacción mediante la emisión de un cheque, había un beneficiario y un emisor. Este beneficiario podía ser una persona física, institución, empresa, etc. Este hecho hacía de los cheques una forma de pago muy flexible. Pues bien, la idea de las direcciones *Bitcoin* es similar.

El capitulo 3 provee de una serie de técnicas teórico-prácticas que permiten obtener las claves necesarias siguiendo un esquema de cifrado asimétrico de clave pública mediante curva elíptica, y sienta la base de una serie de propiedades fundamentales acerca de las funciones de *hash* usadas en Bitcoin y sus tipos.

En este capitulo se explicarán qué son las direcciones *Bitcoin* comunes, cómo se generan, cómo se codificación, y cuáles son los formatos de representación de claves públicas y privadas. Se finalizará con una presentación de las direcciones avanzadas que proporciona el protocolo. Estas direcciones *Bitcoin* comunes sirven fundamentalmente para dotar de mejoras al protocolo, y con la finalidad de abordar nuevas funcionalidades, no previstas hasta entonces.

En *Bitcoin*, la clave pública no se usa solamente con propósitos criptográficos. Además, el doble *hash* de una la clave pública representa la identidad de un individuo que opera un nodo en la red. Cuando algún nodo realiza una transacción, básicamente, lo que se esta diciendo es que alguien con la dirección *x* (doble *hash* de una clave pública) puede reclamar los *Bitcoins* que se indiquen. Claro está, tiene que haber alguna forma de validar que efectivamente esa dirección *Bitcoin* es suya. Aquí es donde entra en juego la otra aplicación de la criptografía de clave pública en *Bitcoin*: la firma digital.

La clave privada es una clave de 256 bits, la cual es y debe ser conocida exclusivamente por la persona que posee la clave pública correspondiente. Si se da el caso en el que un individuo pierde la clave privada o ésta es comprometida por algún fallo de seguridad, los fondos *Bitcoin* se perderán, ya que como se verá más adelante, no podrá proporcionar una firma digital válida.

El algoritmo de generación clave pública y privada de *Bitcoin* está basado en dos partes fundamentales: curvas elípticas y una fuente de mínima entropía o azar seguro

4.1. Generación clave privada

La generación de una clave privada en *Bitcoin* requiere encontrar un número al azar entre 1 y $n - 1$ donde $n = 1{,}158 * 10^{77} \approx 2^{256}$. La función de generación de estos números en *Bitcoin* se realiza mediante el generador de números aleatorios del sistema, inicializado por una fuente o semilla humana de azar como puede ser el tecleo del ordenador o el movimiento del ratón.

En el siguiente ejemplo se muestra la implementación del código [4]en *Python* del cálculo de una clave privada:

```
import pybitcointools as bitcoin

# Algoritmo de generación de clave privada
valid_priv_key = False
while not valid_priv_key:
      # Llamada a la función de aleatoriedad
      priv_key = bitcoin.random_key()
      # Decodificamos la clave
      decoded_priv_key = bitcoin.decode_privkey(priv_key, 'hex')
      # Comprobamos si el valor cae sobre el rango
      valid_priv_key = 0 < decoded_priv_key < bitcoin.N
```

Código 4-1

Fundamentalmente, lo que realiza el algoritmo de generación de clave privada Bitcoin es:

1) Recibir como entrada una cadena de bits (obtenida de una fuente de mínima entropía) de tamaño mayor que 256.
2) Aplicar la función de *hashing SHA-256* para reducir esa longitud variable (superior a 256 bits) a una fija de 256 bits.
3) Comprobar que el resultado obtenido sea menor que $n - 1$, siendo $n = 1{,}158 * 10^{77} \approx 2^{256}$.
4) El resultado se expresa en codificación hexadecimal.

4.2. Generación clave pública mediante curvas elípticas

Las claves públicas de Bitcoin se obtienen a partir de las claves privadas, y a través del uso de las curvas elípticas. Para ello se hace uso de la siguiente ecuación:

$$K = k * G$$

Ecuación 4-1

Donde K es la clave pública resultante, G es una constante que pertenece a un punto de la curva elíptica en el plano, y k es la clave privada computada. Cabe recordar que estamos tratando con un tipo cifrado asimétrico, por tanto, la función es irreversible, o lo que es lo mismo, a partir de un K y G conocido, no existe forma de calcular k. La única forma conocida para que un atacante pueda obtener una clave privada a partir de G y K es mediante la prueba de fuerza bruta (Ecuación 3-3. Resolución del problema del algoritmo discreto), lo cual consiste en simplemente ir probando valores y ver si éstos cumplen la igualdad.

G es una constante común definida en el protocolo de *Bitcoin* la cual es la misma para todos los usuarios de la red. El valor de G en hexadecimal sin comprimir es el siguiente

[4] Código basado en "Generación de clave privada en Python. Mastering Bitcoin: Andreas Antonapoulus".

$$G = 0479BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8$$

Supongamos el siguiente ejemplo:

A partir de la clave privada k:

$$k = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD$$

Calculamos K, siguiendo la idea de la Propiedad 3.5.

$$K = k * G$$

La clave pública se define mediante la proyección ortogonal del punto resultante K en el eje OX y en el eje OY:

$$K = (x, y)$$

$$x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A$$
$$y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB$$

Ejemplo 4-1 Coordenadas de una clave pública

Como se sabe, K es simplemente es el resultado de la multiplicación de un escalar por un punto G en la curva elíptica, o lo que es lo mismo, sumar k veces dicho punto (ver Propiedad 3.5).

Lo que realiza la operación es lo siguiente:

Sabemos que la suma de un punto más el mismo punto equivale a trazar la recta tangente a la curva que pasa por el punto G. La reflexión del punto resultante de la intersección de la recta con la curva en otro punto que no sea el propio punto tangencial es el resultado (ver Ilustración 3-4 Suma mismo punto en una curva elíptica).

Pues bien, realizamos este proceso k veces, para finalmente obtener K, el cual es una tupla (x, y) donde cada coordenada representa la proyección ortogonal de dicho punto en el eje de abscisas y ordenadas.

4.3. Generación dirección *Bitcoin*

En el apartado 4.2, se ha explicado cómo se generaba una clave pública mediante las propiedades de las curvas elípticas. Una dirección *Bitcoin* no es más que la aplicación de dos funciones de *hashing* a dichas claves públicas, concretamente la función *SHA-256* primero, y, luego, la *RIPEMD-160*.

$$Dirección\ bitcoin = RIPEMD160\big(SHA256(clave\ pública)\big),\ \ clave\ pública\ = K = (x, y)$$

Ecuación 4-2

Por tanto, el resultado final de aplicar dicha función será un doble *hash* de una clave pública de 160 longitud bits.

De modo que las transacciones *Bitcoin*, en vez de usar el nombre del beneficiario (volviendo a la analogía del cheque), usan los dobles *hashes* de una clave pública o direcciones.

Por tanto, una transacción *Bitcoin* especifica que alguien con la dirección *Bitcoin x* puede reclamar esos fondos si, y solo si, como se verá más adelante, tiene una firma electrónica válida que esté asociada al par clave privada-clave pública.

4.4. Codificación Base58Check

En informática los sistemas de codificación se usan para representar información en diferentes formatos. Por ejemplo, en binario se usa solamente el `1` y el `0`, y en hexadecimal los caracteres numéricos del 0-9 y los alfabéticos de A-F.

También existe otra codificación ampliamente utilizada para trasmitir cadenas de bits en texto plano: *Base64*. Esta codificación utiliza las 52 letras mayúsculas y minúsculas, los caracteres numéricos del 0-9, y dos caracteres especiales `"/"` y `"+"`. Todo ello sumando 64 caracteres.

Alfabeto *Base64*:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

La codificación *Base58* no es más que un subconjunto de *Base64*, exclusivamente diseñado para *Bitcoin* con el fin de mejorar la legibilidad humana. Esta codificación elimina caracteres ambiguos, que al ser leídos pueden dar lugar a confusiones simplemente por el hecho de ser visualmente semejantes a otros. Estos caracteres son el número 0 y la letra `"O"`, la letra `"l"` (L minúscula) y la letra `I` (i mayúscula), y los dos caracteres especiales: "/" y "+".

Alfabeto *Base 58*:

ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz123456789

Esta última codificación se usa en *Bitcoin* para transcribir información relevante que puede estar sujeta a un procesamiento humano.

4.4.1. Proceso de derivación *Base58Check*

La codificación *Base58* proporciona un modo para la representación de información sin ambigüedades en *Bitcoin*. Además, el formato *Base58Check* proporciona una forma de comprobar la integridad del mensaje mediante el uso de un *checksum*, y de un identificador de versión el cual identifica el tipo de información codificada.

Una vez que el mensaje (dirección, clave privada, etc.) se deriva, se añade el byte de la versión. Este byte puede tomar los siguientes valores dependiendo del tipo del *payload* [5] a codificar:

Tipo	**Prefijo hexadecimal**	**Prefijo base58**
Dirección Bitcoin	0x00	1
Dirección pago HashScript	0x05	3
Dirección a Tesnet Bitcoin	0x6F	m, n
WIF de clave privada	0x80	5, K, L
Clave privada con encriptación BIP38	0x0142	6P
Clave pública extendida BIP32	0x0488B21E	xpub

Tabla 4-1 Prefijos de versión[6]

Cabe destacar que los prefijos que empiezan con el carácter `"6",` indican que las claves privadas en el payload necesitarán alguna acción adicional para que sean utilizables. En el caso de la `6P`, el carácter `"P"` nos está indicando que dicha clave privada está cifrada y necesitará de un clave de descifrado para visualizar la clave privada subyacente en *Base58* (ver 4.6.1).

Después de añadir el byte o bytes de la versión, se obtiene el siguiente datagrama:

Ilustración 4-1. Formato del payload base 58 y prefijo

Con el fin de garantizar la integridad de los datos a transmitir, se puede computar un *checksum*, el cual es la aplicación de dos veces la función de *hashing SHA-256* sobre el *payload.* Una vez computado dicho *hash*, se seleccionan solo los 4 primeros bytes y se incluyen al final del *payload* para obtener:

Ilustración 4-2 Formato del payload base 58, prefijo y checksum

El datagrama resultante se codifica en *Base58.*

Una vez finalizado este proceso, se tendrá un *payload* representado en *Base58Check.*

[5] De aquí en adelante, nos referiremos a un mensaje o información que vaya a ser codificado como *payload. Este mensaje puede ser una clave privada, clave pública, dirección…*

[6] Fuente: Tabla 1. Prefijos de versión Base58Check y ejemplos de resultados codificados "Mastering Bitcoin: Andreas Antonopoulus".

4.5. Resumen proceso generación de una dirección Bitcoin

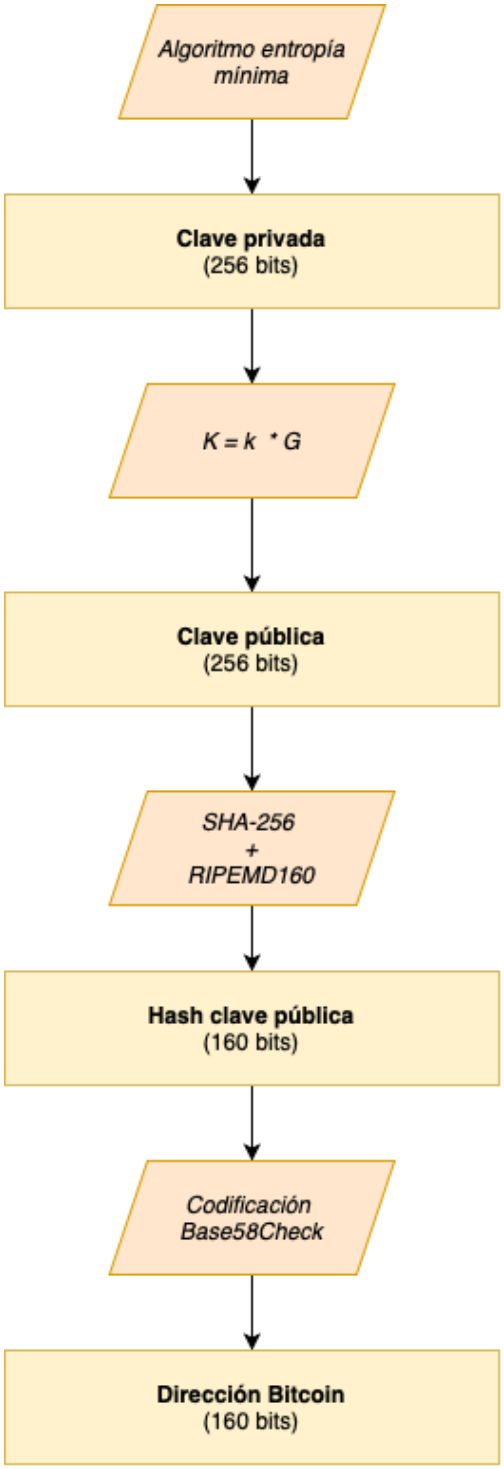


Ilustración 4-3 Proceso generación dirección Bitcoin

4.6. Formatos de claves

Como se vio en el apartado 4.4, existen diferentes maneras de codificar la información en el protocolo de *Bitcoin.* Todas ellas con el objetivo de facilitar la lectura y transcripción de los caracteres que forman una clave o mensaje, los cuales necesitan un procesamiento visual por algún actor determinado.

4.6.1. Clave privada

Como se ha visto en el apartado 4.1, la clave privada es una cadena de 256 bits. Estos bits se pueden expresar de distintas maneras mediante el uso de diferentes codificaciones, pero todas ellas seguirán representando la misma información, solo que en diferente formato.

Los formatos en los que se pueden representar las claves privadas son los siguientes:

Formato	Prefijo	Descripción
Hexadecimal	-	
WIF *(Wallet Import Format)*	5	Codificación base58Check con prefijo 128
WIF comprimido (Compressed Wallet Import Format)	K, L	Igual que en WIF pero añadiendo un sufijo de 01 al payload a codificar.

Tabla 4-2 Formatos de representación de clave privada

Cabe destacar que una clave privada representada mediante el formato WIF (*Wallet Import Format*) comprimida, no está ocupando menos espacio que una WIF. Esto simplemente sirve para indicar el tipo de clave pública que ha de derivarse a partir de dicha clave privada. Esta puntualización será tratada con detalle a continuación.

4.6.2. Formatos clave pública

Recordemos qué representaba una clave pública en *Bitcoin*: Una clave pública en *Bitcoin* hace referencia a las coordenadas (x, y) de un punto en una curva elíptica, el cual se obtiene al derivar dicha clave a partir de una clave privada generada aleatoriamente. Por tanto, el *payload* en hexadecimal a codificar de una clave pública llevará la siguiente información: el prefijo `04` en el caso de ser una clave pública descomprimida, y `03` o `02` en el caso de ser una comprimida.

> $Clave\ pública\ descomprimida\ (hexadecimal) =$
> `04`4f355bdcb7cc0af728ef3cceb9615d90684bb5b2ca5f859ab0f0b704075871a
> a385b6b1b8ead809ca67454d

> $Clave\ pública\ comprimida\ (hexadecimal) =$
> `03`4f355bdcb7cc0af728ef3cceb9615d90684bb5b2ca5f859ab0f0b704075871aa

> $Clave\ pública\ comprimida\ (hexadecimal) =$
> `02`4f355bdcb7cc0af728ef3cceb9615d90684bb5b2ca5f859ab0f0b704075871aa

Ejemplo 4-2 Formatos de clave pública

Puede que al lector le haya llamado la atención el hecho de que una clave pública comprimida pueda tomar dos prefijos diferentes. ¿Por qué es esto?

Como se hacía referencia anteriormente, una clave pública no es más que la representación de las coordenadas *(x, y)* de un punto en una curva elíptica. Por tanto, usando los pares de coordenadas obtenidas en Ejemplo 4-1, el *payload* de representación de una clave pública descomprimida tendrá el formato:

$$K = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A\\07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB$$

Ejemplo 4-3 Clave pública descomprimida

Simplemente lo que se ha hecho es juntar los valores de las coordenadas *(x, y)* en una misma cadena de caracteres.

Ahora bien, lo anterior puede inspirar la siguiente pregunta: ¿la coordenada en el eje 0X del punto de la clave pública en la curva elíptica, no basta para derivar la coordenada en el eje OY? Efectivamente, esto es cierto: sustituyendo en la ecuación implícita de la curva se podría obtener el valor de *y*, ahorrando escribir el valor de *y* en la clave.

Así pues, se puede optar por almacenar una sola de las coordenadas, y calcular la otra coordenada a partir de la ecuación implícita de la curva, lo que supone un ahorro de almacenamiento de casi el 50%. Esta es una ventaja tremendamente beneficiosa, si se tiene en cuenta, que los usuarios de la red que operan un nodo completo tienen que almacenar la totalidad del *blockchain,* el cual, como se explicará más adelante, almacena una copia completa de todos los bloques desde el bloque génesis. Estos bloques recogen todas las transacciones que un nodo, que propone el siguiente bloque, recoge de otros nodos de la red, y así, en cada bloque del *blockchain*. A su vez, la mayoría de las transacciones *coinbase* (aquellas que crean fondos *Bitcoin*) son del tipo P2PK (pago a clave pública), las cuales almacenan la clave pública a la que se quiere pagar.

Este hecho hace que el coste de almacenar dichos datos sea muy alto y, además, según *Bitcoin* avance en su ciclo de vida, irá incrementando, es decir, no está acotado.

Pero la cosa no acaba aquí. Cabe destacar que, al despejar la variable y de la Ecuación 3-4, se obtiene una raíz cuadrada. Un hecho que supone trabajar con dos soluciones válidas para un punto determinado en la curva elíptica: una por debajo del eje OX y la otra por encima. Cabe destacar que las dos son igualmente correctas; sin distinciones.

Ambas soluciones están a la misma distancia mínima del eje OX. Esto es debido a que una curva elíptica es una función simétrica respecto al eje de abscisas. Por tanto, no basta con almacenar simplemente la coordenada X, sino que también es necesario almacenar el signo: positivo o negativo. Y es aquí donde entran a escena los prefijos a los que nos referíamos anteriormente.

Los prefijos `02` (signo positivo) y `03` (signo negativo) de una clave pública nos indican que, a partir de esas claves, hay que derivar una clave pública comprimida, la cual representa el punto de la clave pública y su signo.

Aun así, incluir ese prefijo diferenciador de signo supone un coste mínimo si lo comparamos respecto a la alternativa de almacenar las dos coordenadas x e y del punto K.

Ejemplo clave pública comprimida en formato hexadecimal:

$$K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A$$

Ejemplo 4-4 Clave pública comprimida

En el anterior resultado podemos ver que se ha hecho uso del prefijo `03` seguido de la coordenada x del punto K (clave pública).

Ahora bien, no todos los nodos de Bitcoin operan las mismas versiones del protocolo. Esto es debido a que, al ser una red descentralizada, no existe forma efectiva, sin que no suponga crear un *hard-fork* (ver 7.6) de obligar a los nodos implementar una versión determinada. Esto hace que *Bitcoin* tenga que ser flexible y permita a sus usuarios realizar transacciones operando versiones distintas.

Las versiones más recientes implementan las claves públicas comprimidas, en las cuales simplemente hay que especificar el sufijo `02` (coordenada y positiva) o `03` (coordenada y negativa, el *payload* (clave pública con solo la coordenada x), y *checksum*. Por el contrario, las más viejas operan con la representación de clave pública descomprimida: sufijo `04`, el payload (ambas coordenadas), y el checksum.

Lo anterior genera, a su vez, otro problema, ya que para una clave privada puede haber dos claves publicas válidas: aquellas comprimidas y aquellas sin descomprimir y las dos igualmente correctas.

Cuando una aplicación de cartera, la cual solo almacena la clave privada, examine el *blockchain* para detectar cuáles son las transacciones que puede reclamar su usuario, tendrá que saber por cuales buscar: por las de formato comprimido o formato descomprimido. Si se acuerda de lo mencionando en 4.6.1, el formato de WIF o WIF comprimido indica qué tipo de clave pública habrá que derivar a partir la clave privada. De esta forma, la aplicación de cartera podrá buscar (**de forma efectiva**) el tipo de clave pública correcto en la cadena de bloques y se habrá reducido el espacio de almacenamiento de esta clave casi un 50%.

4.7. Direcciones Bitcoin avanzadas

Los apartados 4.1, 4.2, 4.3 y 4.4, han tratado en profundidad el concepto de direcciones *Bitcoin,* respondiendo a las preguntas de qué eran, cómo se derivaban y de qué formas se podían representar. Este tipo de direcciones fueron las primeras en ser utilizadas en las primeras

releases del protocolo *Bitcoin.* Pero a medida que el protocolo fue desarrollándose, nuevas necesidades aparecieron, dando paso al desarrollo de los tipos que se explicarán en 4.7.1 y 4.7.2.

4.7.1. Claves privadas encriptadas BIP0038

Recordemos que aquél que tenga una clave privada, podrá reclamar los fondos correspondientes a transacciones sin gastar en la cadena de bloques, cuando presente una firma válida y una dirección que concuerde. Básicamente, tener una clave privada, significa ser el propietario de fondos *Bitcoin*.

Debido al uso que se hace de las direcciones *Bitcoin* se puede dotar de un cifrado a las claves privadas que se almacenan en una cartera *Bitcoin.* Esto es así debido a que una clave privada, al ser expuesta en un entorno potencialmente hostil, puede verse comprometida o, directamente, puede que se pierda.

Supóngase el caso de la exportación de claves de una cartera a otra. En el momento de la retransmisión de la clave, su confidencialidad puede verse perdida debido a ataques externos o deficiencias del canal. También, a la hora de realizar copias de seguridad en dispositivos de almacenamiento electrónicos o en papel, el riesgo de que la clave se vea comprometida aumenta.

Además, estas direcciones proveen de un método menos laborioso de implementar una capa de seguridad externa que el que proporcionan las metodologías de doble factor, tal y como se explican a continuación:

Las metodologías de doble factor son aquellas que requieren un emisor externo que recibe una clave pública por parte de un cliente. El emisor responde al cliente con el envío de la clave pública con algún dato adicional (*token*). Esto habitualmente se materializa en una moneda física, la cual es vinculada a la clave pública del cliente, y que deberá ser gastada mediante una herramienta externa.

El hecho de utilizar una herramienta externa para gastar nuestros fondos y confiar en una entidad desconocida, hace que el uso del doble factor se vuelva menos atractivo respecto al cifrado de claves privadas. Ya que, este último, no requiere actor externo alguno en el cual haya que confiar ningún dato personal. Mediante el uso de las claves privadas cifradas, el propietario de la clave privada deberá recordar solamente una palabra clave o secreto, mucho más corto que una clave privada en formato WIF, y aplicarla para el cifrado/descifrado.

Por tanto, las claves privadas cifradas nacen de la necesidad de proveer de una capa de seguridad adicional a una clave privada, la cual sea simple de implementar y no requiera participantes externos. Su implementación se llevo a cabo en la propuesta de mejora *Bitcoin* 38 (BIP0038)[7]. Dicha propuesta implementaba el cifrado simétrico *Advanced Encryption Standard* (AES). Sin entrar en detalles, un cifrado simétrico, al contrario que uno asimétrico, usa la misma clave para descifrar y cifrar. La clave siempre tiene que ser guardada por el usuario en un lugar seguro.

[7] Propuesta de mejora BIP00038: https://github.com/bitcoin/bips/blob/master/bip-0038.mediawiki

Como se sabe, todo elemento de *Bitcoin* que requiera un procesamiento visual, se codificará en *Base58Check,* eliminando así la posibilidad de errores de transcripción o procesamiento debido a la aparición de caracteres ambiguos, y comprobando la integridad del *payload*. Como se indicaba en Tabla 4-1, una clave privada cifrada tendrá el identificador "`6P`" como prefijo. De esta forma, se sabrá que se está tratando con una clave privada que requiere la aplicación de un descifrado simétrico mediante la aplicación de una palabra clave.

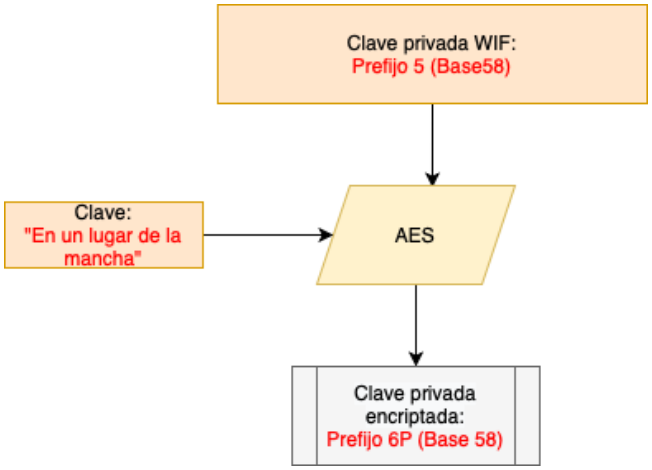


Ilustración 4-4 Derivación clave privada cifrada.

En resumen, las claves privadas cifradas proporcionan un método simple para implementar una capa de seguridad adicional sobre nuestra clave privada. De esta forma, una cartera cifrada se podría enviar por algún medio de transmisión, sin preocuparse de aspectos de seguridad. Simplemente habría que preocuparse de entregar el secreto o clave que toma como entrada el algoritmo de cifrado simétrico AES. Esta clave puede ser intercambiada de la misma forma que se intercambia en el protocolo HTTPS, por teléfono o en persona.

4.7.2. Direcciones Pay to Script Hash (P2SH) BIP0016 BIP0013

Las direcciones *Bitcoin P2SH* son una manera muy inteligente que usa el protocolo para solventar problemas de incompatibilidades entre actores en una transacción. Lo que hace esta funcionalidad es liberar, al emisor de la transacción, de la responsabilidad de proporcionar las condiciones que han de ser cumplidas y otorgársela al receptor de los fondos. Este tipo de direcciones fueron sugeridas en la Propuesta de Mejora *Bitcoin* 16 o BIP0016[8], y en la Propuesta de Mejora Bitcoin 13 o BIP0013[9].

Supóngase que un nodo *Bitcoin* opera una versión la cual da soporte al uso del método MULTISIG.

[8] https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki

[9] https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki

Función MULTISIG *(multi-signature):* Básicamente lo que especifica este método es que, en vez de necesitar la firma de un solo actor, este tipo de direcciones necesitarán la firma de *n* actores de un total de m, con $n \leq m$ para desbloquear una transacción. Se tratará en detalle el uso de este tipo de funciones en el apartado 6.4.3.

Supóngase el caso en el que María quiere realizar una transacción con Juan. Juan implementa un nuevo protocolo *Bitcoin* que tiene funcionalidades que María no conoce, como por ejemplo MULTISIG. María, por tanto, dirá: "*No sé qué es una función MULTISIG"*; y no realizará la transacción. Pues bien, en vez de rechazar la transacción, este tipo de incompatibilidades que se pueden dar entre ambas partes de la transacción, se solventan mediante el uso de las direcciones P2SH.

En este tipo de direcciones, la dirección a la que se envían los fondos *Bitcoin* no es una dirección *Bitcoin* común, sino que es el *doble-hash* de un *script*[10]*:*

$$Dirección\ P2SH = RIPEMD160(SHA256(script\ file))$$

Ecuación 4-3

De esta forma, se aísla por completo las diferentes implementaciones entre Juan y María. María simplemente especificará que aquel con la dirección P2SH x, podrá reclamar los fondos. Pero con la sutileza que será el propietario del *script* el responsable de realizar el desbloqueo de la transacción y de especificar el *script* en cuestión para reclamar las monedas. Además, Juan será el que se encargue de cumplir las condiciones que especifique en su *script* de desbloqueo, por ejemplo: si Juan ha usado la función MULTISIG, deberá proveer *n* de *m* firmas validas para poder desbloquear la transacción.

Como se sabe, una dirección *Bitcoin* normal usa el prefijo `"1"` en codificación *Base58Check*. Por tanto, siempre que se vea una cadena de caracteres que empiece con el carácter `"1"` sabremos que nos estamos refiriendo a una dirección *Bitcoin* normal. En cambio, las direcciones P2SH utilizan el prefijo `"3"` en *Base58Check*, indicando que dicha dirección tendrá que proporcionar un *script*, cuya firma concuerde con el *hash* indicado por María.

$$Dirección\ P2SH = \mathbf{3}N5i3Vs9UMyjYbBCFNQqU3ybSuDepX7oT3$$

4.7.3. Direcciones de vanidad

Las direcciones de vanidad no implementan ninguna funcionalidad adicional sobre el protocolo de *Bitcoin*. Simplemente son direcciones en las que se puede observar un patrón legible, el cual tiene un significado semántico para una persona.

$$Dirección\ de\ vanidad = 1BoatSLRHtKNngkdXEeobR76b53LETtpyT$$

[10] Un *script Bitcoin*, no es más que un conjunto de operaciones *Script* (ver 6.3).

Como se observa aparece el patrón legible "*Boat*" (bote en inglés).

El proceso de derivación de este tipo de claves es el mismo que el de una dirección normal. Solamente que se deberán computar combinaciones diferentes de claves privadas, aplicar el proceso de derivación de dirección *Bitcoin*, y comprobar si se obtiene el patrón deseado.

Este tipo de direcciones no supone ninguna desventaja, son igualmente de seguras que una dirección *Bitcoin* normal y se computan de la misma forma. La única diferencia que tiene este tipo respecto a las normales es que a más longitud de caracteres del patrón que se quiera obtener, más claves privadas se tendrán que computar para después derivar una dirección, la cual, al ser codificada a *Base58*, nos proporcione el patrón deseado.

Existen dos formas de generar estas claves: 1) Usando la capacidad de computo local disponible. 2) Contratando los servicios de empresas dedicadas, las cuales usan su capacidad de computo para obtener las direcciones con los patrones indicados por el cliente.

Como se ha mencionado anteriormente, la dificultad del cálculo de direcciones de vanidad aumenta según se requiera un patrón más o menos extenso.

Patrón	Dificultad	Tiempo medio
1B	22	< 1s
1Bi	1,330	< 1s
1Bit	77,178	< 1s
1Bitc	4,476,342 (4.48E+6)	< 10s
1Bitco	259,627,881 (2.6E+8)	3 min
1Bitcoi	15,058,417,127 (1.506E+10)	3 h
1Bitcoin	8.7339E+11	1 semana
1BitcoinE	5.0657E+13	1 año
1BitcoinEa	2.9381E+15	60 años
1BitcoinEat	1.7041E+17	3,500 años

1BitcoinEate	9.8837E+18	200,000 años
1BitcoinEater	5.7325E+20	11,700,000 años

Tabla 4-3[11] Complejidad de generación de direccione de vanidad

[11] Fuente: https://en.bitcoin.it/wiki/Vanitygen

5. Estructuras de datos

Al igual que en los registros tradicionales, donde se guardaba un histórico de todas las transacciones llevadas a cabo por los clientes en un determinado banco, en *Bitcoin,* se hace uso de una estructura de datos que permite recoger el histórico de las transacciones de la red. Esta estructura permite saber, a una aplicación de cartera, cuáles son los fondos que se pueden reclamar, validar las transacciones anteriores, y proporcionar una base de datos que permita guardar las nuevas transacciones que propone un minero, al crear un bloque.

La principal motivación de la utilización del *blockchain* es la validación de transacciones que se lleva a cabo cuando un bloque se mina. En el proceso de minado se recogen una serie de transacciones en el bloque y se validan. Cada bloque valida al anterior. El *blockchain*, por tanto, es una estructura de datos compartimentada en bloques, los cuales verifican transacciones y validan todos los bloques anteriores.

De una forma más formal, se dirá que el *blockchain* es la estructura de datos fundamental de la red, y está formada por bloques que recogen metadatos del propio bloque, y por transacciones que, un determinado *pool* de transacciones, ha ido acumulando según eran promocionadas por los distintos nodos de la red.

5.1. Blockchain

Técnicamente se dice que el *blockchain* es una lista enlazada, en la cual los nodos padre son referenciados por siguiente nodo o el nodo hijo. El *blockchain* sigue la misma estructura, pero simplemente con un matiz: los punteros a los bloques anteriores son doblemente *hasheados* por una función SHA-256.

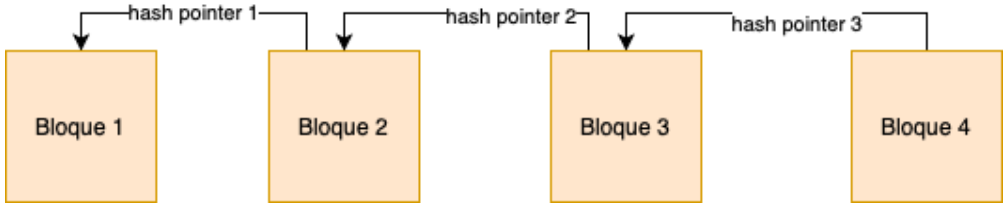


Ilustración 5-1 Estructura de datos blockchain

Los bloques tienen el siguiente formato:

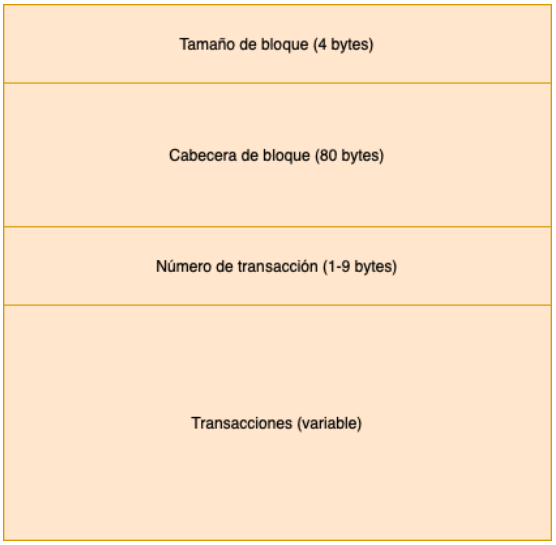


Ilustración 5-2 Estructura de un bloque

- **Tamaño de bloque:** Tamaño del bloque a partir de este campo.
- **Cabecera de bloque:** Metadatos de bloque.
- **Numero de transacciones:** Número de transacciones que se incluyen el *merkle-tree*.
- **Transacciones:** Transacciones que recoge el bloque.

Como se ha mencionado anteriormente, los punteros que sirven para referenciar a los bloques anteriores o nodos padre, son doblemente *hasheados* mediante la función *hash* SHA-256. Cabe destacar que, el valor de entrada que toma esta función, es solamente la cabecera de un bloque en concreto.

$$Hashpointer = SHA256(SHA256(cabecera))$$

Ecuación 5-1

El formato de la cabecera de un bloque en el *blockchain* es el siguiente:

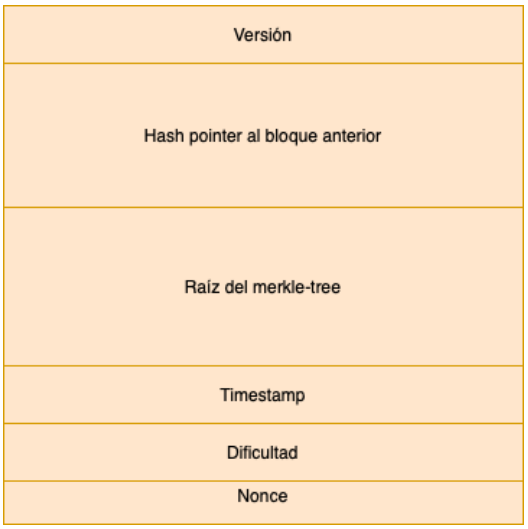


Ilustración 5-3 Cabecera del bloque

- **Versión:** Número de versión del protocolo *Bitcoin* que implementa el minero que propone el bloque.
- ***Hash-pointer* al bloque anterior:** Resultado del aplicar dos veces la función de SHA-256 a la cabecera del nodo padre.
- **Raíz del *merkle-tree***: Raíz del *merkle-tree* usado para la validación de todas las transacciones incluidas en el bloque.
- ***Timestamp*:** El instante de tiempo en segundos en el que se ha creado el bloque de acuerdo con el formato *Unix Epoch*.
- **Dificultad:** Dificultad del *hash-puzzle* o *proof of work* resuelto.
- ***Nonce*:** Numero expresado en 4 bytes que se usa como una de las entradas de la función *hash* en el proceso del reto minero.

Un aspecto por recordar y que tiene que quedar claro, es que la función de *hash* no toma como *input* todos los datos del bloque, sino que solamente "*hashea*" las cabeceras. Por tanto, los *hash-pointers* siguen la siguiente estructura:

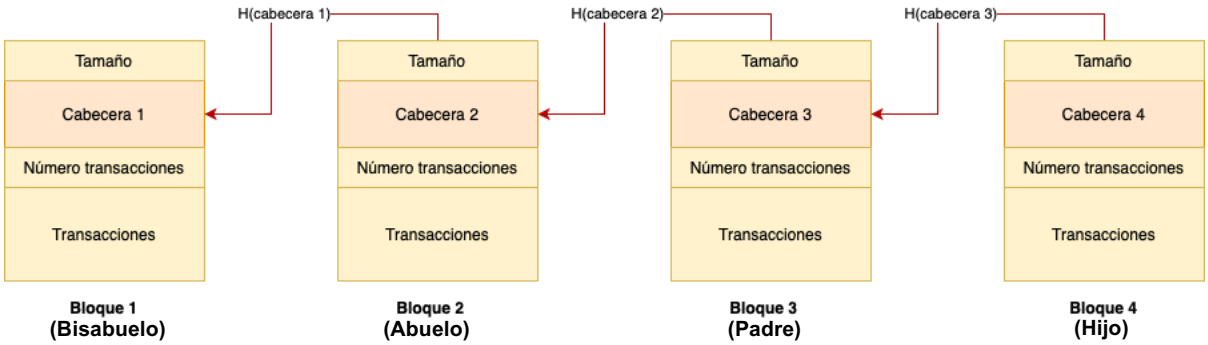


Ilustración 5-4 Blockchain extendido

Mediante la Ilustración 5-3 y Ilustración 5-4 Blockchain extendido, se aprecia que, por tanto, un *hash-pointer* determinado, no solo valida el nodo padre, sino el abuelo, bisabuelo…

Por ello el uso de las funciones *hash* es tan esencial en esta estructura de datos. Recuerde que una función *hash* criptográficamente segura, garantiza que los *outputs* de esta función no sean iguales, o, lo que es lo mismo, que no habrá dos *hash-pointers* iguales. Este principio es la razón por la cual el *blockchain* funciona como una libreta infranqueable, donde todos los elementos tienen identificadores únicos y se validan los unos a los otros. No hay forma que dos distintos bloques con diferentes datos empaquetados en sus cabeceras originen un mismo *hash-pointer*. Si se cambia tan solo un bit de el nodo abuelo, los *hash-pointers* del padre y el hijo no coincidirán con el previamente calculado e introducido en la cabecera del hijo.

Uno podría pensar que, simplemente, para romper la cadena e introducir un elemento entre dos bloques ya incluidos, se tendría que recalcular todos los *hash-pointers* de la cadena: desde el propio elemento "corrupto" o "intruso", hasta el último bloque, cambiando así, los valores de

los campos de la cabecera con los nuevos computados. Pero como se verá en el capitulo de minería (ver Minería), *Bitcoin* aborda este problema de una forma muy inteligente mediante las reglas de consenso implícito, las cuales generan que, a la hora de recalcular los *hash-pointers,* llevar a cabo un solo recalculo de un bloque en concreto, puede que sea factible, pero a medida que se quiera recalcular más bloques, más difícil se volverá tal proceso. Se puede pensar que la dificultad de recalcular los *hash-pointers* desde un bloque determinado hasta el final de la cadena, funciona de la misma forma que cavar un hoyo: Las primeras capas de tierra ser fáciles de remover, pero a medida que se profundice más, la tierra será más compacta y difícil de cavar.

Se ha mencionado que un *blockchain* es una estructura de datos basada en una lista enlazada, pero habitualmente se suele representar mediante la estructura de datos de pila. Como el lector sabrá, una pila es un tipo de estructura de datos en la cual el primer elemento en ser introducido es el último en salir, y el último introducido el primero. De esta forma el *blockchain* se puede representar de esta misma manera: los últimos bloques son los primeros en la pila, y los primeros los últimos. Mediante esta representación se obtiene la definición de altura de un bloque:

Se dice que un bloque N tiene una altura $N - 1$, siendo $N - 1$ el número total de elementos anteriores que preceden el bloque N.

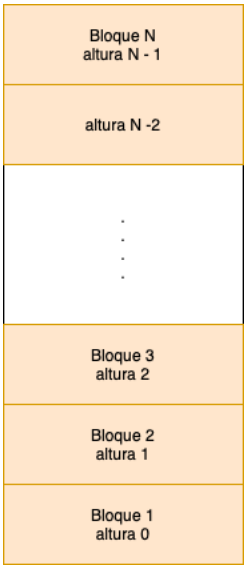


Ilustración 5-5 Altura de un bloque

Esto permite definir dos formas de identificar un mismo bloque en el *blockchain:* 1) *Hash* de la cabecera de bloque. 2) Altura del bloque.

Cabe destacar que, mientras que el *hash* de la cabecera es un identificador único, debido a las propiedades de la función *hash*, la altura de cabecera no lo es. Esto es debido a que puede haber dos mismas versiones del *blockchain,* lo que se denomina como *fork* (bifurcación en inglés) (ver 7.6). Estos dos mineros querrán introducir un nuevo bloque (diferente el uno al otro) a la misma altura, pero con diferentes valores en sus cabeceras, lo cual *hasheará* a diferentes *hashes*

de cabecera de bloque. Esta es la razón por la cual se dice que un *hash* de cabecera es un identificador único.

El *blockchain* se puede almacenar en un archivo o en una base de datos indexada, de forma que se pueda acceder a los elementos de una manera optimizada, dependiendo de la estrategia empleada por el sistema gestor de la base de datos en cuestión. Por ejemplo, el cliente *Bitcoin Core* almacena los metadatos de la cadena en una *LevelBD*[12].

5.2. Merkle-tree

Los *merkle-trees* son la estructura de datos usada para la validación de transacciones que recoge un determinado bloque. La principal ventaja, del uso de esta estructura de datos, es la eficiencia que supone validar la pertenencia de una transacción a un determinado bloque.

Un *merkle-tree* es un tipo de árbol binario. Como el lector sabrá, un árbol binario se construye de la siguiente forma:

Existe un nodo raíz para el cual se derivan hasta dos nodos hijos, de los cuales se pueden derivar hasta otros dos más, y así, sucesivamente. Cada nodo, por tanto, almacena dos referencias o punteros a sus dos nodos hijo. Los nodos hojas son aquellos que tienen sus dos referencias a sus dos nodos hijos establecidas a *null*.

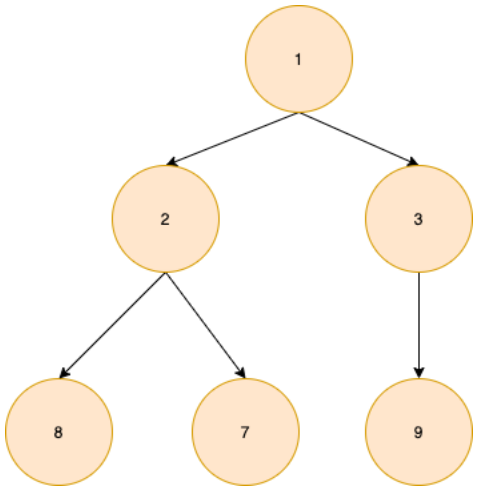


Ilustración 5-6 Un árbol binario

En los arboles de *merkle*, en vez de guardar referencias a sus dos nodos hijos, se guardan los *dobles-hash* SHA-256 de la concatenación de los *hashes* de sus dos nodos hijos.

Los nodos hoja, en este tipo de árbol, son un *doble-hash* SHA-256 que toma, como valor de entrada, los bytes de una transacción.

A continuación, se aborda la construcción de los *merkle-trees*, explicando los dos casos diferentes que existen para llevar a cabo su construcción, según la paridad los nodos hojas.

[12] "Mastering Bitcoin: Andreas Antonopoulus". Página 177,

1) Construcción de *merkle-tree* con elementos pares.
2) Construcción de *merkle-tree* con elementos impares.

5.2.1. Merkle-tree con número par de nodos hojas

Se sabe que, para inicializar el algoritmo de construcción, el doble SHA-256 necesita tomar como *input* los *bytes* de una transacción.

Por ejemplo, para las transacciones A, B, C y D, se computarían los siguientes *hashes*:

$$H(A) = SHA256(SHA256(transacción\ A))$$
$$H(B) = SHA256(SHA256(transacción\ B))$$
$$H(C) = SHA256(SHA256(transacción\ C))$$
$$H(D) = SHA256(SHA256(transacción\ D))$$

Estos valores de inicialización son los nodos hoja del árbol.
Luego, todos estos *hashes* de inicialización se almacenan en una lista enlazada.

En el siguiente paso la lista enlazada se agrupa por pares y se da paso al calculo del nodo padre:

Calculo del nodo padre, por ejemplo, de A y B. Se realiza lo siguiente:

$$H(AB) = SHA256(SHA256(H(A) + H(B)))$$

Lo mismo para CD:

$$H(CD) = SHA256(SHA256(H(C) + H(D)))$$

Una vez obtenidos los nodos padre, se seguiría calculando nodos padre de la misma manera, hasta llegar a la raíz del árbol o *merkle-root:*

En nuestro caso no hace falta calcular más nodos padre, sino que en el siguiente paso se obtiene directamente la raíz del árbol:

$$H(ABCD) = SHA256(SHA256(H(AB) + H(CD)))$$

La estructura del árbol sería la siguiente para el anterior proceso:

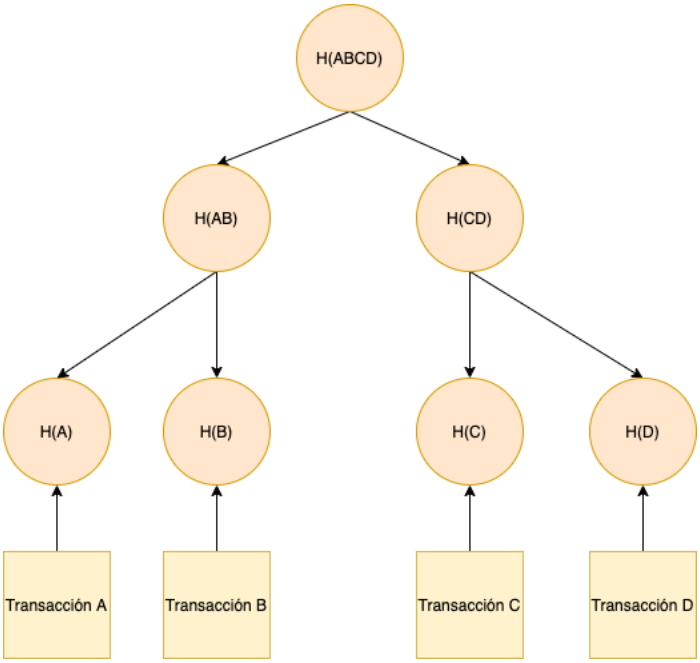


Ilustración 5-7 Merkle-tree básico

5.2.2. Merkle-tree con número impar de nodos hojas

Suponiendo ahora un número impar de transacciones: A, B, D, C y E:

Se inicializan los nodos hoja de la misma forma que en el caso anterior (ver 5.2.1):

$$H(A) = SHA256(SHA256(transacción\ A))$$
$$H(B) = SHA256(SHA256(transacción\ B))$$
$$H(C) = SHA256(SHA256(transacción\ C))$$
$$H(D) = SHA256(SHA256(transacción\ D))$$
$$H(E) = SHA256(SHA256(transacción\ E))$$

Y obtenemos los pares AB y BC. Nótese que para el elemento E no existe pareja. En este caso, simplemente lo que se realiza es duplicar dicho elemento y obtener el par EE.

Después de aplicar lo anterior, se obtienen los dobles pares `(AB, CD)` y `(EE, _).` Nótese que se vuelve a la misma situación: no existe pareja para EE. Por tanto, se aplica la misma estrategia, y se duplica el nodo EE para obtener el par `(EE, EE).`

Se computan los nodos padre ABCD y EEEE:

$$H(ABCD) = SHA256(SHA256(H(AB) + H(CD)))$$
$$H(EEEE) = SHA256(SHA256(H(EE) + H(EE)))$$

Y trivialmente se computa el nodo raíz:

$$H(ABCDEEEE) = SHA256(SHA256(H(ABCD) + H(EEEE)))$$

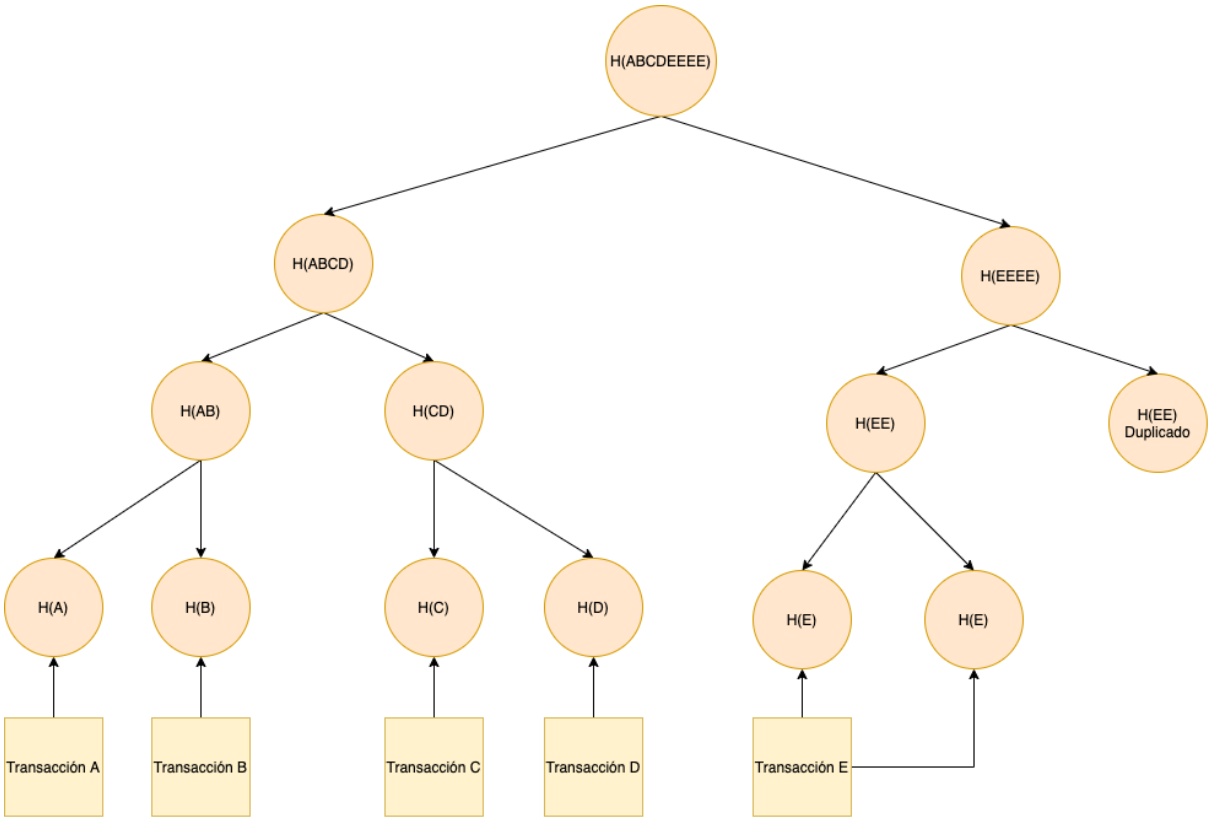


Ilustración 5-8 Merkle-tree de elementos impares

5.2.3. Validación de pertenencia (*proof of membership*)

Como se ha mencionado anteriormente, el *merkle-tree* es una estructura de datos empleada en *Bitcoin* por su eficiencia. Esto es así debido a la reducción de procesamiento que supone validar que una transacción pertenece al bloque. Habitualmente se usa el termino de *proof of membership* para referirse a este proceso de validación.

Tomando como ejemplo de *merkle-tree*, el presentado en Ilustración 5-8 Merkle-tree de elementos impares, observe qué haría falta para comprobar que una transacción pertenece a dicho árbol:

Supongamos que se quiere validar que la transacción C pertenezca al bloque. ¿Cómo se comprobaría?
Para determinar que, efectivamente, una transacción está incluida en el bloque, se necesita una ruta en el árbol, llamada *merkle-branch.* El *merkle-branch* es la ruta directa y sus nodos hermanos directos, la cual proporciona el **mínimo** de los elementos necesarios que entrarían en juego para la verificación de un elemento en el árbol. El *merkle-branch* para la validación de la transacción C sería el siguiente:

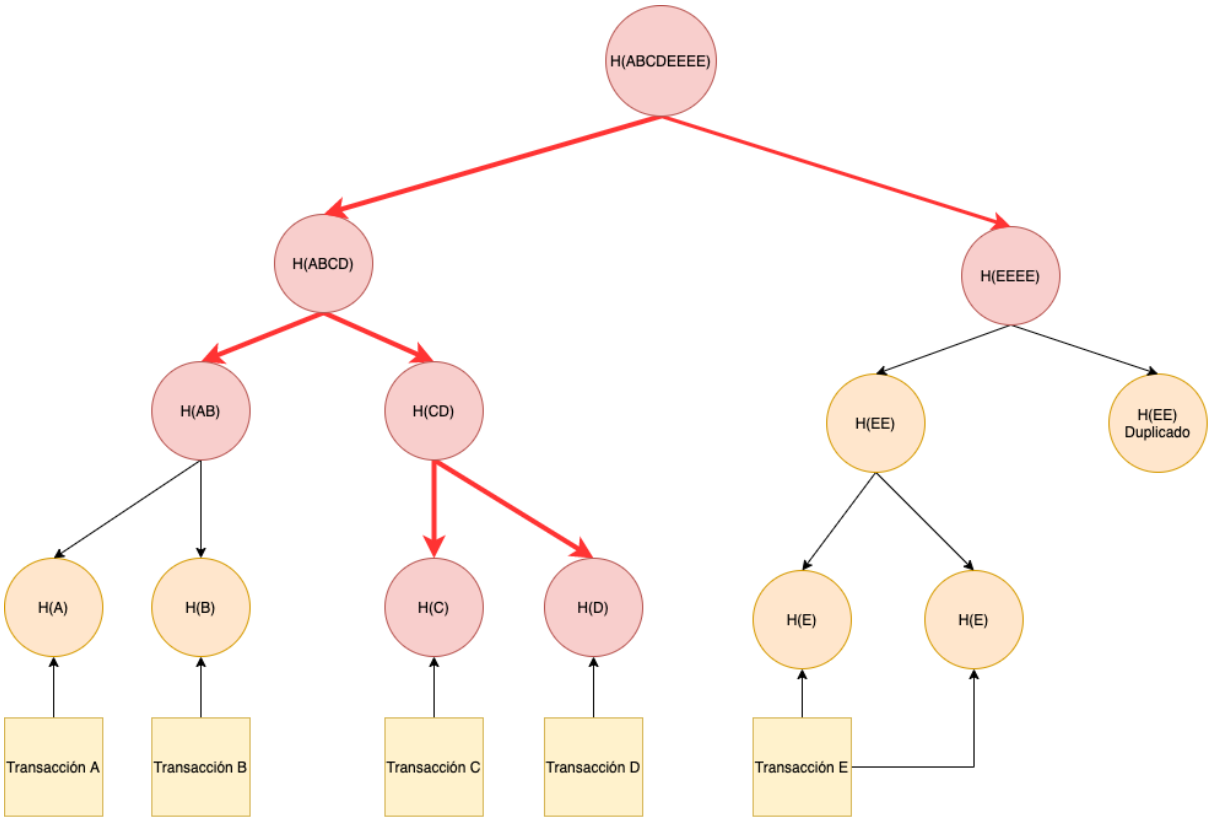


Ilustración 5-9 Validación de pertenencia de una transacción

Merkle-branch para la validación de la transacción C:

$$\{H(C), H(D), H(CD), H(AB), H(ABCD), H(EEEE)\ H(ABCDEEEEE)\}$$

Cabe destacar que el único valor que nos interesa de la anterior lista es $H(ABCDEEEEE)$, ya que sirve, como se sabe (dado las propiedades de una función de *hash* criptográfica), de huella digital de todos los nodos interiores y sus hojas.

Por tanto, el nodo que quisiera validar la transacción C tendría que computar la ruta por su cuenta, tomando como único valor de entrada la transacción C, y usando los valores del *merkle-branch*:

Valor de entrada:

$$H(transacción\ C)$$

Computar los *hashes* apoyándose en los nodos,

$$\{H(D), H(CD), H(AB), H(ABCD), H(EEEE)\ H(ABCDEEEEE)\}$$

y obtener finalmente la raíz del *merkle-tree.*

$$H'(ABCDEEEEE)$$

Si esta raíz concuerda con $H(ABCDEEEEE)$ (recuerde que este valor se almacena en la cabecera de bloque), las huellas digitales del árbol serán las mismas y se habrá probado la pertenencia o *proof of membership* de la transacción.

5.2.4. Eficiencia merkle-tree

Nótese que la eficiencia algorítmica de esta operación de validación es del orden de $log(n)$ siendo n el número de nodos en el árbol. Es, por tanto, muy eficiente, si se tiene en cuenta, que las transacciones que maneja un *merkle-tree* en un bloque son del orden de miles.

Número transacciones	Tamaño de bloque	Numero de hashes en el merkle-branch	Tamaño del merkle-branch
16	4 kB	4	128 B
512	128 kB	9	288 B
2.048	512 kB	11	352 B
6.5535	16 MB	16	512 B

Tabla 5-1 Tamaño de merkle-branch

Cabe destacar que, no todos los nodos de la red tienen que computar un *merkle-tree* completo. En tal caso, la eficiencia de la validación se vería incrementada, dado que se tendría que computar el árbol entero, para después validarlo.

Como se verá más adelante existen varios tipos de nodos, entre ellos los *light-weight-nodes (SPV)* y *full-nodes.* Los full-nodes sí tendrían que calcular el *merkle-tree* al completo. Pero la buena noticia es, que la mayoría de los nodos en la red son del tipo *SPV*, y estos nodos solamente almacenan el valor de la raíz del *merkle-tree.* De esta forma el nodo *SPV* contactaría a un *full-node* y le pediría el *merkle-branch* para una determinada transacción, aprovechándose, de esta forma, de la eficiencia logarítmica (ver **Error! Reference source not found.**).

6. Transacciones

Las transacciones son, quizás, el elemento más importante de todo *Bitcoin*. Todos los demás elementos: minería, criptografía, estructuras, red, etc... están creados para proveer a las transacciones de una infraestructura eficiente y robusta.

Una transacción *Bitcoin* es una estructura de datos con el siguiente formato:

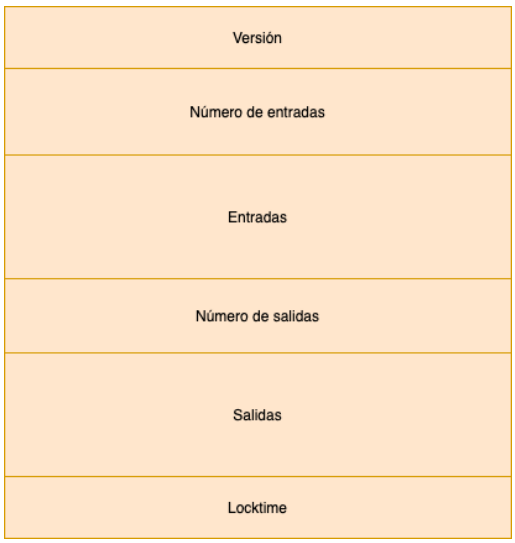


Ilustración 6-1 Estructura de una transacción.

- **Versión:** versión del nodo que crea la transacción.
- **Número de entradas:** número de salidas de otras transacciones que se toman como entradas en la nueva transacción.
- **Entradas:** salidas de transacciones previas que toma como valores de entrada la transacción.
- **Número de salidas:** número total de las salidas.
- **Salidas:** salidas de los fondos *Bitcoin* mediante las cuales se transfieren fondos a otras direcciones.
- ***Locktime*:** Un sello de tiempo en formato *timestamp* (*Unix Epoch),* o una altura de bloque. Este tipo de parámetro se usa para especificar el momento en el que una transacción debe ser validada y transmitida. Hay dos modalidades:

    - Si su valor es menor que 500 millones y mayor que 0, se interpretará como una altura de bloque, hasta la cual la transacción no podrá ser validada ni retransmitida.

- Si su valor es mayor que 500 millones, se interpreta como un *timestamp*. Este sello de tiempo indica el momento a partir el cual la transacción podrá ser retransmitida y procesada.
- Si su valor es 0 (en la mayoría de las transacciones lo es) indica retransmisión inmediata

El parámetro *Locktime* es muy útil para transacciones del tipo de pago de nóminas.

6.1. Unspent transaction outputs (UTXO)

Una UTXO es una transacción que es reclamable por un nodo determinado. Las UTXO son, por así decirlo, las monedas que conforman el "saldo de una determinada cuenta de *Bitcoin*". Cabe destacar que, en *Bitcoin,* no existe el concepto de "saldo de cuenta", sino que son las aplicaciones de cartera las encargadas de crear esta ilusión de balance de una cuenta tradicional. En realidad, los fondos asociados a una dirección *Bitcoin,* no son más que el total de las transacciones UTXO que son reclamables por una determinada dirección a lo largo del *blockchain*.

El registro total de todos los UTXO en el *blockchain* o, simplemente, reserva UTXO, es conservado dinámicamente por los nodos completos o *full-nodes* en una base de datos. Estos nodos llevan el registro de todas las transacciones que son reclamables por direcciones.

Cuando una aplicación de cartera requiera obtener la cantidad total de BTC (unidad de representación monetaria de Bitcoin) asociados a una determinada cuenta, contacta mediante una API (API `blockchain.info`) con los nodos completos, preguntando por todas las UTXO reclamables por la dirección. Éstos le contestarán con un *dataset* conteniendo todas UTXO reclamables para ese cliente o dirección.

Un UTXO es una cantidad determinada de monedas *Bitcoin*, concretamente un múltiplo determinado de *Satoshi*. Un *Satoshi* en *Bitcoin* equivale 10^{-8} BTC. Esta unidad es la base de la moneda; no hay valor menor que un *Satoshi*, de la misma forma que no existe un valor menor que 1 céntimo de euro. Esta analogía sirve también para ilustrar la indivisibilidad de los UTXO: Un UTXO por valor de 50 BTC es indivisible, al igual que un billete de 5€ no se puede partir por la mitad y derivar un valor de 2,5 € y 2,5€.

Luego, las UTXO sirven para nutrir las futuras transacciones a incluir en la cadena de bloques. Las entradas de una transacción hacen referencia a un UTXO. Estas entradas deben de ser validadas para ser reclamables, aportando, como se verá más adelante, un *script*, y si el proceso de validación se satisface, se podrán utilizar para crear nuevas UTXO o salidas de una transacción. Este proceso de retroalimentación hace que una transacción sea un elemento, el cual sirve para consumir entradas, validándolas como se verá a continuación, y para generar salidas o nuevos UTXO.

Uno de los aspectos relevantes de las entradas y salidas, son las combinaciones que se pueden hacer a la hora de realizar un pago con múltiplos de *Satoshi*, con el fin de generar nuevas salidas

con un valor determinado. Supóngase que se requiere pagar 5 BTC. Hay tres escenarios posibles para reunir estos fondos:

- Se dispone de una dirección con un UTXO exacto de 5 BTC.

  De esta manera las entradas de la transacción serán 5 BTC y la salida 5 BTC.

- Se dispone de una fracción de 5 BTC, por ejemplo, dos UTXO de 2 BTC y 3 BTC.

  Por tanto, la transacción tomará como entrada estas dos UTXO y se generará una única salida de 5 BTC. Esencialmente, lo que se hace es fusionar dos UTXO de valor menor a uno con la suma de los fondos de las dos UTXO.

- Se dispone de un UTXO con un valor reclamable de 10 BTC.

  Se creará una transacción con una única entrada que tome como valor los 10 BTC y se crearán dos salidas: una con 5 BTC pagando a la dirección deseada, y la otra de 5 BTC pagando a la dirección que el emisor de la transacción (esta es la forma por la cual se obtiene cambio en *Bitcoin)*.

Existe un tipo de transacción especial, la cual es llamada *coinbase,* que no toma ninguna UTXO como entrada. Esto es así, debido a que se están creando, precisamente, *Bitcoins* en esa transacción. Esta transacción es la que un minero que ha resuelto el *hash-puzzle* o *proof of work*, crea cuando éste mina un nuevo bloque.

6.1.1. Formato de la salida de una transacción

Tamaño	**Campo**
8 B	Cantidad
1-9 B	Tamaño del *script* de bloqueo
Variable	*Script* de bloqueo

Tabla 6-1 Formato de una salida de una transacción

- **Cantidad:** La cantidad es el valor en Satoshis.
- **Tamaño:** Tamaño del *script* de bloqueo.
- ***Script* de bloqueo:** *script* con una serie de órdenes, el cual tiene que ser validado o desbloqueado por un *script* de desbloqueo. El concepto de *scripts* de bloqueo y desbloqueo se tratará en profundidad en los siguientes puntos.

6.1.2. Formato de la entrada de una transacción

Tamaño	**Campo**
32 B	*Hash* de transacción.
4 B	Índice de la salida.
1-9 B	Tamaño de *script* de desbloqueo.

Variable	*Script* de desbloqueo
4 B	Número de secuencia

Tabla 6-2 Formato de una entrada de una transacción

- ***Hash* de la transacción:** *hash* de la UTXO a la cual hace referencia y que será gastada.
- **Índice de la salida:** índice de la transacción UTXO en el bloque que la contiene. Este índice empieza por 0. La trnasacción 0 es normalmente una de tipo *coinbase*.
- **Tamaño:** tamaño del s*cript* de desbloqueo.
- ***Script* de desbloqueo:** *script* que contiene todas las órdenes que, al ser juntadas con las del *script* de bloqueo, liberan la UTXO y hace que sus fondos puedan ser transferidos a otra dirección como una nueva UTXO.
- **Número de secuencia:** se usa para sobrescribir una transacción previamente a la expiración del tiempo de desbloqueo. Actualmente esta funcionalidad está deshabilitada.

6.2. Scripts de bloqueo y desbloqueo

En este apartado se describe el mecanismo de validación de una transacción y mediante el cual se transfieren UTXO a nuevas salidas de una transacción, está fundamento en el uso de los *scripts* de bloqueo y desbloqueo.

Como se especificaba en Tabla 6-1, las salidas tienen un *script* de bloqueo el cual especifica las condiciones mediante las que una determinada dirección podrá reclamar sus fondos y usarlos para incluirlos en nuevas entradas de una futura transacción. Las UTXO tienen, por tanto, una serie de salidas bloqueadas a la espera de que se les dote de su correspondiente *script* de desbloqueo. Este *script* de desbloqueo recoge una serie de instrucciones, las cuales, al ser ejecutadas junto a las de el *script* de bloqueo, liberarán la salida bloqueada, si se dan una serie de condiciones. Este *script* de desbloqueo se incluye en las entradas de las transacciones.

Tradicionalmente estos *scripts* de bloqueo y desbloqueo se han llamado por los nombres de *ScriptPubKey* y *ScriptSig* respectivamente. Este tipo de *script* son del tipo: 1) Pago a la clave pública *x* (*ScriptPubKey*), y 2) esta clave pública *x* podrá reclamar sus fondos dotando a al *ScriptPubKey* de un *ScriptSig,* el cual contiene una firma generada por su clave privada y una clave pública.

Esta visión de llamar a los *scripts* de bloqueo y desbloqueo, *ScriptSig* y *ScriptPubKey*, es limitada, ya que estos *scripts* son solo para un tipo en concreto de transacciones. Como se verá más adelante, existe una amplia variedad de tipos de transacciones que involucran diferentes *scripts* de bloqueo y desbloqueo, aparte de los *ScriptPubKey* y *ScriptSig*.

6.2.1. Ejecución de *Scripts* de bloqueo y desbloqueo

Las instrucciones recogidas en los *scripts* de bloqueo y desbloqueo se ejecutan de la siguiente manera:

Supónganse los *scripts* de bloqueo y desbloqueo especificados en Ilustración 6-2. Las ordenes de los *scripts* se ejecutan secuencialmente en la rutina principal de ejecución: primero las del *script* de desbloqueo y, después, las del de bloqueo.

Representando gráficamente lo anterior:

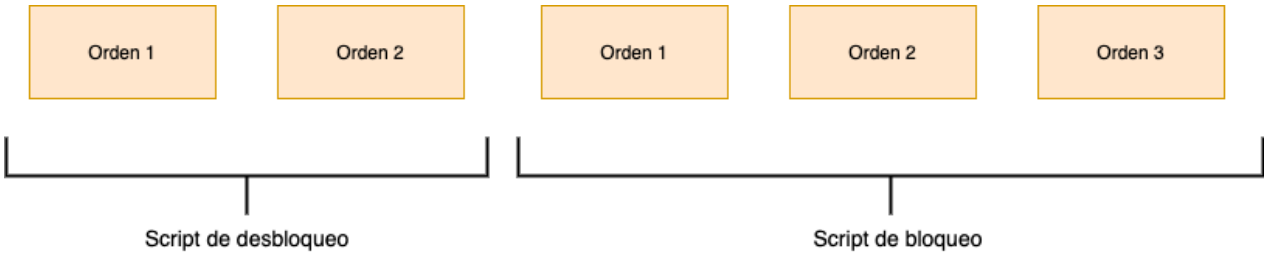


Ilustración 6-2 Secuencia de ejecución de Scripts

$$Rutina\ de\ ejecución\ =\ \{Orden\ 1^{*}, Orden\ 2^{*}, Orden\ 1, Orden\ 2, Orden\ 3\}$$

Nota: El (*) indica que son ordenes pertenecientes al script de desbloqueo.

La rutina de ejecución principal cuenta con una pila auxiliar en la que se incluyen o extraen datos. La inclusión o extracción de estos datos en la pila sigue el patrón FIFO (*First In, First Out*). Esta pila sirve como una especie de *buffer* para la rutina de ejecución principal. Los elementos se apilan cuando el puntero de ejecución recoge la referencia de la orden (entre la secuencia principal de ordenes), y, después, las apila o extrae según indique la orden (ver Ilustración 6-2).

Otro aspecto relevante es que no existe una pila común para las ordenes del *script* de bloqueo y para las de desbloqueo; son dos pilas diferentes. Cuando el puntero de ejecución recorre todas las órdenes del *script* de desbloqueo, hace una copia del estado de la pila del *script* de desbloqueo e inicializa la pila del *script* de bloqueo con la copia.

La ejecución se realiza de la siguiente forma:

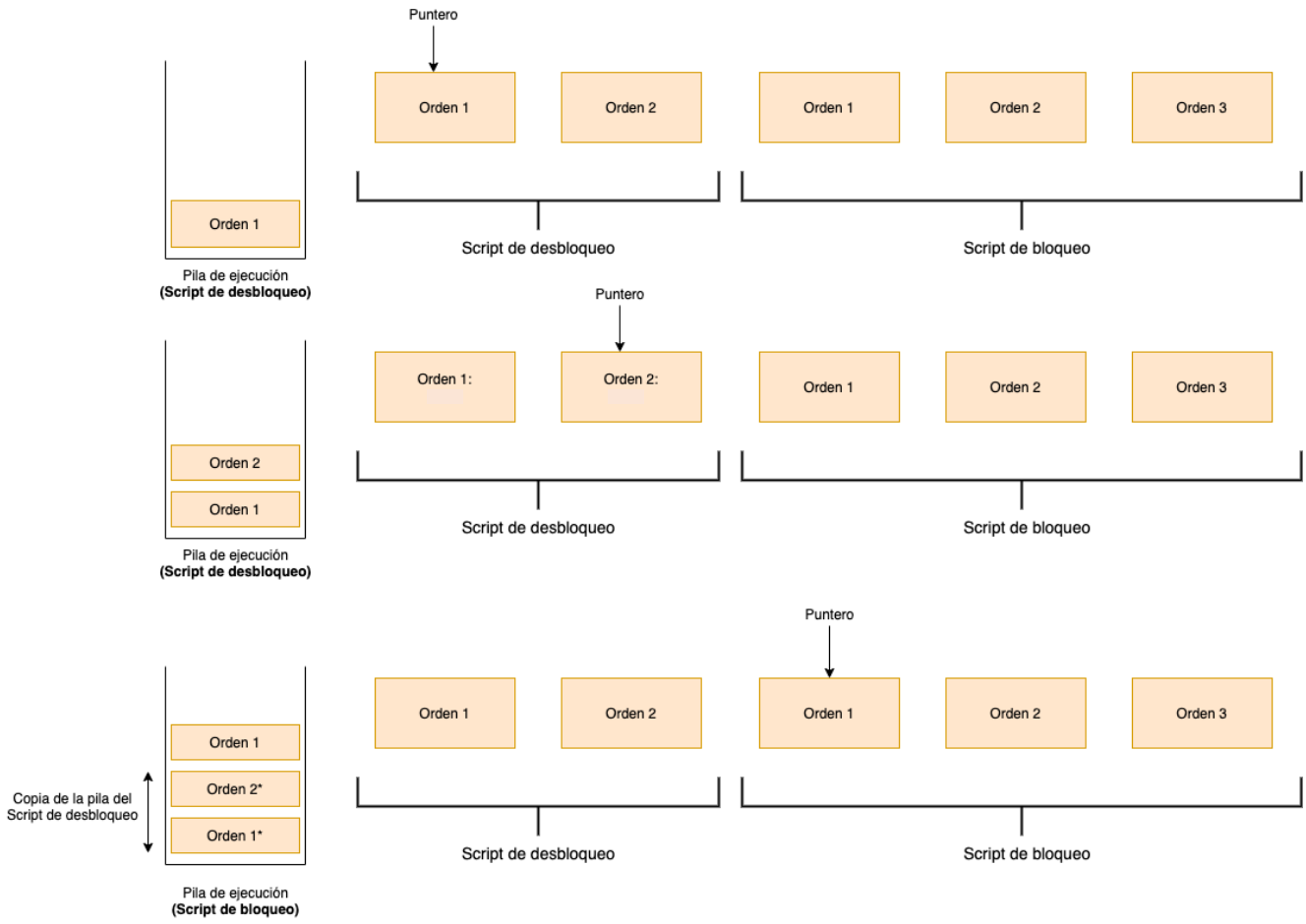


Ilustración 6-3 Secuencia de ejecución de ordenes

Nota: El (*) indica que son ordenes pertenecientes a la pila del script de desbloqueo que han sido copiadas.

El puntero se inicializa sobre la primera orden de ejecución del *script* de desbloqueo. Este puntero recoge la referencia a la orden en la memoria principal, y la introduce en la pila. El proceso se repite, moviendo el puntero secuencialmente a lo largo de todas las ordenes del *script* de desbloqueo hasta recorrer todas las ordenes incluidas en dicho *script*.

Una vez todas las ordenes del *script* de desbloqueo se han ejecutado, se copia el estado de la pila de ejecución del *script* de desbloqueo, y se inicializa la pila del *script* de bloqueo con los datos de la copia.

El proceso de ejecución es el mismo para la pila del *script* de bloqueo: el puntero va recogiendo las ordenes de la secuencia y va introduciéndolas en la pila.

En el anterior ejemplo se ha supuesto que todas las ordenes apilan datos en la pila, pero esto no tiene por qué ser así. Como se verá a continuación, existen ordenes que, simplemente, consumen datos de la pila, y otras que además de consumir datos, introducen derivados de los datos de entrada en la pila.

6.3. Lenguaje Script

Este apartado presenta la clave del proceso de bloqueo-desbloqueo. El lenguaje *Script* es un lenguaje muy similar a *Forth* (otro lenguaje basado en pilas) y orientado a limitar las acciones que se puedan realizar en los *scripts* de bloqueo y desbloqueo.

Piense en el contexto de desarrollo de *Bitcoin*: un lenguaje destinado a validar salidas de transacciones UTXO, las cuales contienen fondos monetarios, tiene que ser seguro por naturaleza. El lenguaje *Script* consigue esto, limitando el número de acciones que se pueden ejecutar mediante el uso de sus 256 ordenes predefinidas.

Aunque el lenguaje está dotado de elementos de control de flujo condicionales, operadores aritméticos, operaciones de unión de cadenas, operadores numéricos, operadores criptográficos, etc… el lenguaje no es *Turing Completo;* es decir, no existe la posibilidad de crear bucles, ni estructuras de control complejas, de modo que los tiempos de ejecución son predecibles. Esto es una garantía de seguridad adicional, ya que no permite la posibilidad de creación de bucles infinitos, los cuales pueden llevar a denegaciones de servicio en los nodos que los ejecutan.

6.3.1. Ejemplos del lenguaje Script

A continuación, se presentan algunos ejemplos sencillos del tipo de órdenes que proporciona este lenguaje: suma aritmética, condicionales, booleanos, etc. Como se ha dicho anteriormente, las órdenes de *Script* son 256 (muchas de ellas se encuentran actualmente deshabilitadas).

Aunque en los ejemplos, no se tratarán muchos tipos de ordenes *Script*, se dotará al lector de una idea de cómo funciona este lenguaje y las posibilidades que ofrece.

Para empezar, se define una de las operaciones más básicas:

Ejemplo 6-1 Suma de dos números en Script

```
2 4 OP_ADD
```

En realidad, los números 2 y 4 tendrían que estar representados en hexadecimal (`0x02, 0x04`), pero por comodidad para el ejemplo, se tratarán en decimal.

La orden `OP_ADD` saca dos elementos de la pila y los suma. El resultado se introduce en la pila. (ver Ilustración 6-4)

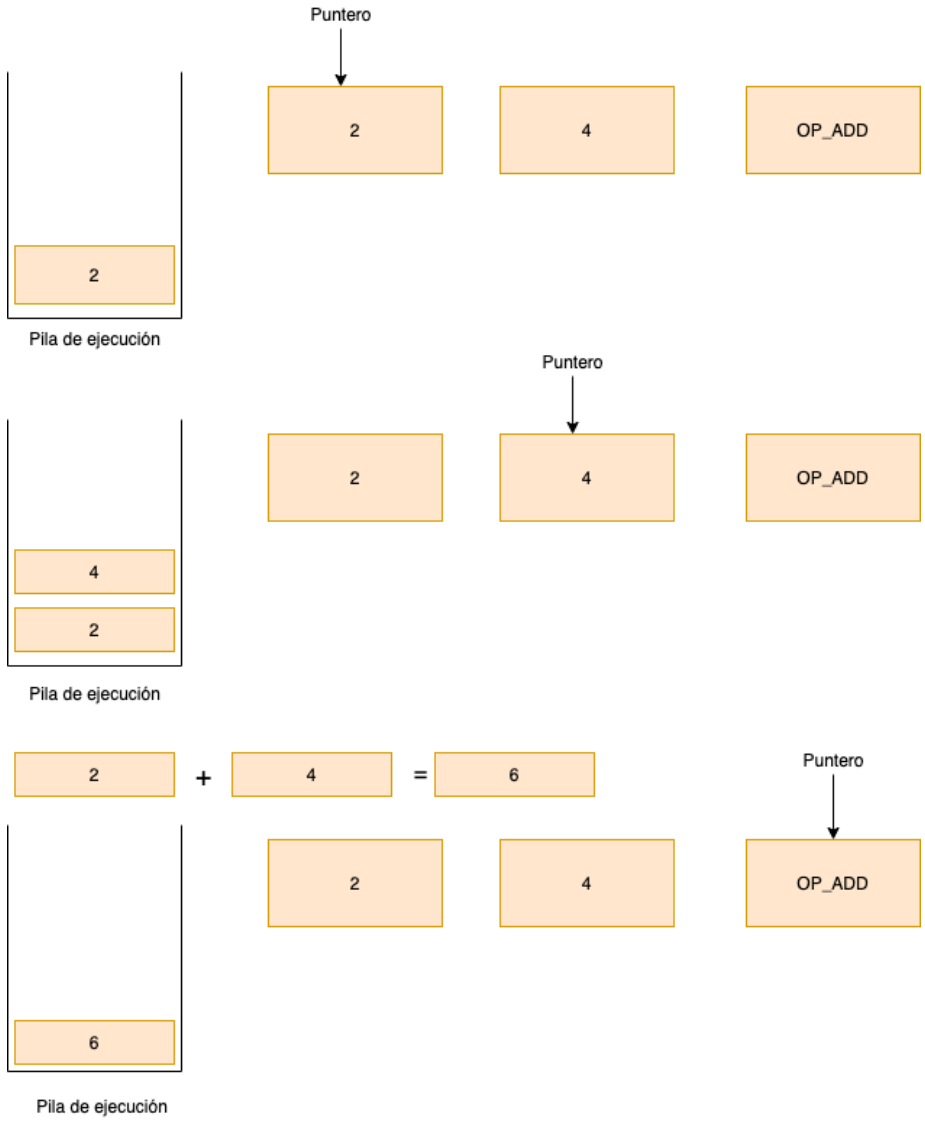


Ilustración 6-4 Ejecución ordenes Script

Ahora, un ejemplo más complejo:

```
2 4 OP_MUL 3 OP_SUB 6 OP_LESSTHAN
```

1) Los elementos 2 y 4 (`0x02, 0x04`) se introducen a la pila.
2) La función `OP_MUL` saca esos dos elementos, los multiplica, e introduce el resultado (`8` o `0x08`) en la pila.
3) Después, se introduce el valor 3 en la pila y se ejecuta la orden `OP_SUB`, la cual saca dos elementos de la pila (8, 3), realiza 8 - 3 = 5, e introduce el resultado (5) en la pila.
4) Se introduce 6 en la pila.
5) Finalmente se ejecuta la orden `OP_LESSTHAN`, la cual saca dos elementos de la pila (5, 6) y comprueba si 5 < 6, lo cual es verdadero.

6) El *script* devuelve verdadero.

6.3.2. Validez-invalidez de una secuencia de ordenes

Una **transacción se considera valida** si el resultado de la ejecución es verdadero, cualquier valor distinto de 0, o una pila vacía.

Las **transacciones son invalidas** si el resultado de la ejecución es falso o si se interrumpe la ejecución mediante una orden indicada explícitamente: `OP_VERIFY`, `OP_RETURN`, o `OP_ENDIF`. Este tipo de órdenes provocan siempre un error al ser ejecutadas.

6.4. Transacciones comunes

A continuación, se presentan los tipos de transacciones más comunes utilizados en *Bitcoin* y los únicos que son aceptados por los nodos que ejecutan el cliente *Bitcoin* de referencia: *Bitcoin Core*. Estos tipos pueden ser descartados en futuras implementaciones del protocolo o puede que se añadan nuevos.

Los tipos estándar de transacciones son los siguientes: *Pay to Public Key Hash* (P2PKH), *Pay to Public Key* (P2PK), *Multi-Signature* (MULTISIG), *Operation-Return* (OP_RETURN), y *Pay to Script Hash* (P2SH).

En este apartado se presentará, detalladamente, cómo funcionan cada uno de los tipos, y cuáles son las órdenes que especifican los *scripts* de bloqueo y desbloqueo de cada uno.

6.4.1. Pay to Public Key Hash (P2PKH)

Este tipo de transacciones son del tipo pago a la dirección *Bitcoin x*. El *script* de bloqueo especifica que aquél con una determinada dirección, podrá reclamar los fondos de la salida de una UTXO, cuando una entrada aporte una firma (emitida mediante la clave privada) y una dirección válida.

Los *scripts* de bloqueo y desbloqueo tienen el siguiente conjunto de ordenes *Script*:

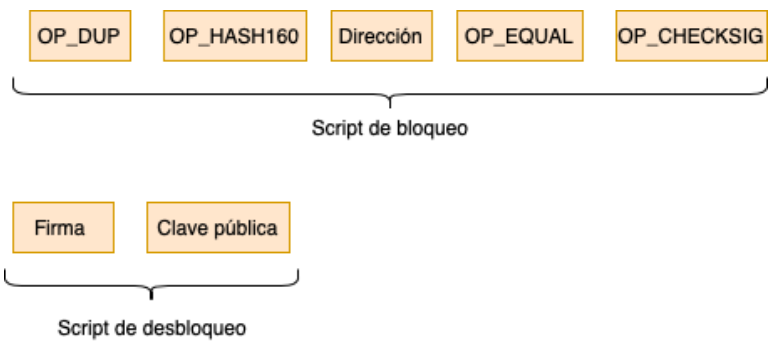


Ilustración 6-5 Ordenes Script P2PKH

Orden Script	Descripción
OP_DUP	Duplica el tope de la pila.
OP_HASH160	Realiza doble *hash* RIPEMD160(SHA256(m)) e introduce el resultado en la pila.
Dirección	Introduce la dirección *Bitcoin* **codificada en hexadecimal**.
OP_EQUAL	Extrae el tope y el elemento anterior al tope de la pila, comprueba si son iguales, e introduce verdadero o falso en función del resultado.
OP_CHECKSIG	Extrae los dos últimos datos de la pila, los cuales han de ser una firma y una clave pública para que la operación se realice con éxito, comprueba si los datos concuerdan con el *hash* de la transacción, y devuelve falso o verdadero a la pila.
Firma	Firma creada por una clave privada en **formato hexadecimal**.
Clave pública	Clave pública en **formato hexadecimal**.

La secuencia de ejecución sería la siguiente:

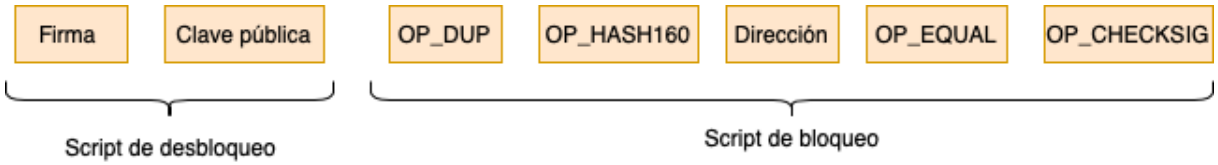


Ilustración 6-6 Secuencia de ejecución P2PKH

1) Inicialización de la pila:

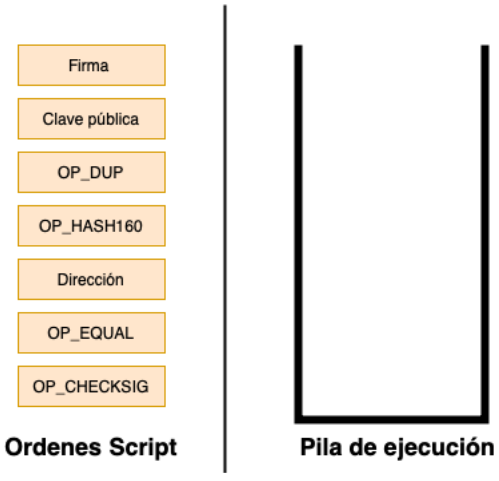


Ilustración 6-7 Ejecución P2PKH (paso 1)

2) Se introduce la firma y clave pública del *script* de desbloqueo:

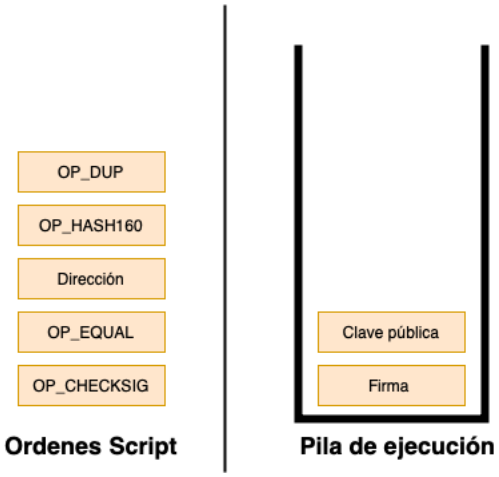


Ilustración 6-8 Ejecución P2PKH 2 (paso 2)

3) Se duplica el tope de la pila (Clave pública):

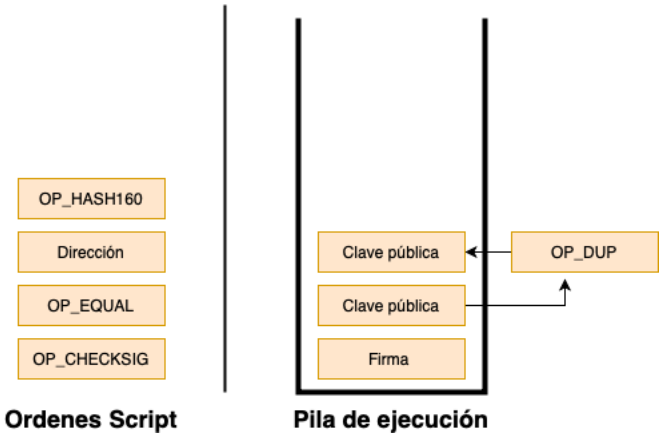


Ilustración 6-9 Ejecución P2PKH 3 (paso 3)

4) Se extrae el tope de la pila (clave pública), y se realiza su doble-hash:

$$RIPEMD160\big(SHA256(Clave\ pública)\big) = H(Clave\ pública) = direccion.$$

El resultado se introduce en la pila.

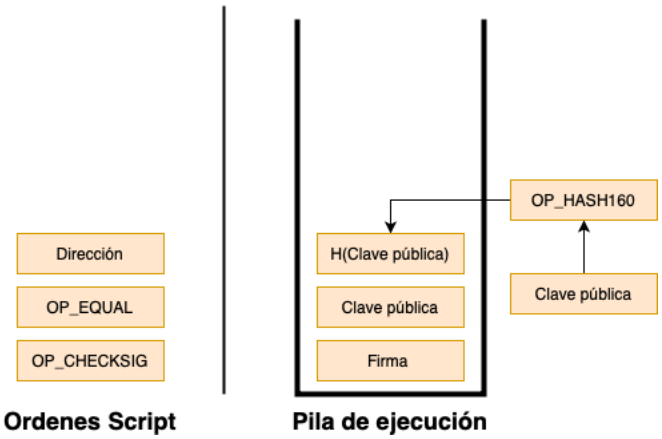


Ilustración 6-10 Ejecución P2PKH (paso 4)

5) Se introduce la dirección (del *script* de bloqueo) a la pila:

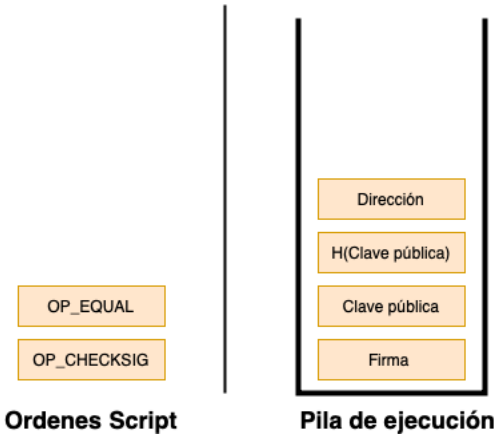


Ilustración 6-11 Ejecución P2PKH (paso 5)

6) Se extraen los datos dirección y $H(Clave\ pública)$. Después, se comprueba que estos dos datos sean iguales.

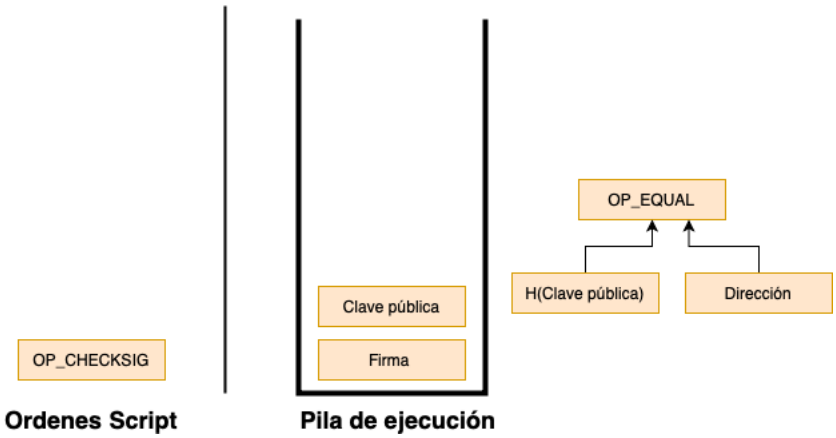


Ilustración 6-12 Ejecución P2PKH (paso 6)

7) Se extraen los datos de clave pública y firma de la pila, los procesa como entradas la función OP_CHECKSIG, y el resultado se introduce en la pila. Para el ejemplo, se supone que la firma es correcta.

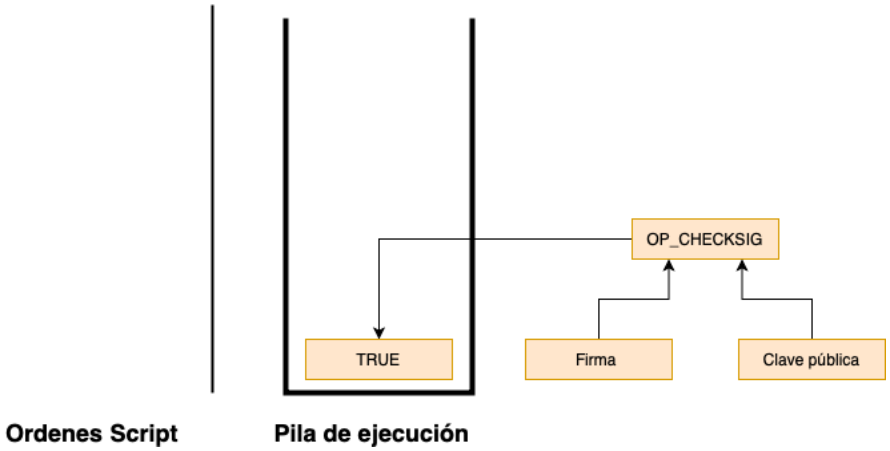


Ilustración 6-13 Ejecución P2PKH (paso 7)

6.4.2. Pay to Public Key (P2PK)

La gran mayoría de transacciones son del tipo P2PKH, pero existen, también, las del tipo P2PK. La única diferencia de este tipo de transacciones con las del tipo P2PKH es que, en el *script* de bloqueo, en vez de especificar una dirección, se especifica la clave pública.

Este tipo de transacciones se utilizaban en las primeras versiones del protocolo. Hoy en día su uso para transacciones comunes es marginal (aunque la mayoría de coinbase son del tipo P2PK), debido a la diferencia en el espacio de almacenamiento que supone almacenar una clave pública de 256 bits, en vez de un *hash* de longitud fija de 160 bits.

Los *scripts* de bloqueo y desbloqueo se ven de la siguiente forma:

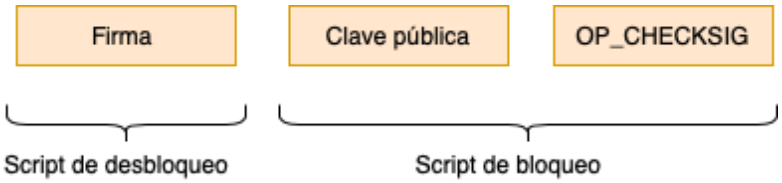


Ilustración 6-14 P2PK scripts bloqueo y desbloqueo

El proceso de ejecución es análogo al explicado en 6.4.1.

6.4.3. Multi-Signature (MULTISIG)

Los *scripts* MULTISIG proveen de una serie de condiciones flexibles para el desbloqueo de una salida de una UTXO. En este tipo de transacciones los *scripts* de bloqueo, en vez de establecer que una única dirección que aporte una firma correcta podrá reclamar los fondos, establece que n de m firmas son requeridas de sus correspondientes n de m claves públicas ($n \leq m$) para el desbloqueo de la salida.

Este tipo de transacción hace uso de la orden *Script* `OP_CHECK_MULTSIG`, la cual verifica que hay al menos *n* firmas válidas, las cuales correspondan a las claves públicas especificadas.

Los *scripts* de bloqueo y desbloqueo tienen el siguiente conjunto de ordenes:

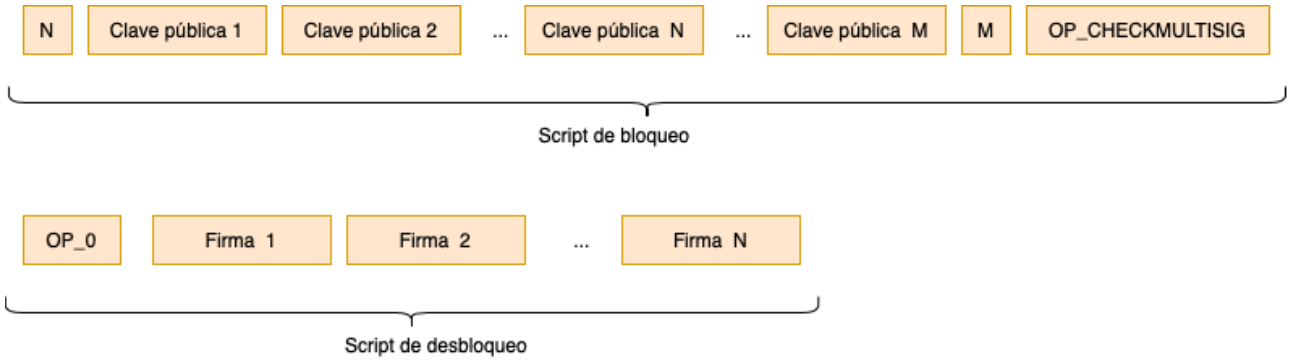


Ilustración 6-15 MULTISIG scripts bloqueo y desbloqueo

La ejecución de las ordenes anteriores es muy simple:

1) Todos los datos (firmas, *n*, *m*, y claves públicas) son apilados en la pila.
2) La orden `OP_CHECK_MULTSIG`, toma todos los datos de la pila como valores de entrada y devuelve VERDADERO o FALSO.

El operador `OP_0` es requerido por un error en la implementación original de la orden `OP_CHECK_MULTSIG`, por el cual se saca un valor de más de la pila. Se ha mantenido debido a que el coste de arreglar dicho error no compensa lo que supondría enmendarlo.

Cabe destacar que cualquier combinación $\binom{m}{n}$, $\binom{m}{n+1}$, ... , $\binom{m}{m}$ es válida para que la operación `OP_CHECK_MULTSIG` devuelva verdadero.

Esto quiere decir que, por ejemplo, si tenemos *n = 2* y *m = 3,* valdrán los siguientes resultados (suponiendo que las firmas sean correctas para las claves públicas especificadas):

Claves publicas especificadas en script de bloqueo:

$$\{Clave\ pública\ 1, clave\ pública\ 2, clave\ pública\ 3\}$$

$n = 2, m = 3$

Serán válidas las siguientes combinaciones de firmas:

- $\binom{m}{n} = \binom{3}{2}$

$$\{ Firma\ 1, Firma\ 2 \}$$

$$\{ Firma\ 2, Firma\ 3 \}$$

- $\binom{m}{n+1} = \binom{3}{3}$

$$\{ Firma\ 1, Firma\ 2, Firma\ 3 \}$$

6.4.4. *Operation-Return* (`OP_RETURN`).

¿Se podría crear de alguna forma una UTXO, cuya salida recoja un *script* de bloqueo que jamás pueda ser gastado por ninguna entrada y en la cual se pueda almacenar algún dato? Si es así, ¿se estarían escribiendo registros inmutables en la cadena de bloques?

Pues bien, las anteriores preguntas se responden afirmativamente mediante el uso de un operador especial *Script*. Este comando se introduce en los *scripts* de bloqueo, y al ejecutarse siempre da error. Esto hace que una UTXO que contenga este comando en el *script* de salida sea ingastable.

Ahora bien, ¿pero y si, además de ser ingastable, se pudiera escribir algún dato en la cadena de bloques, de manera que se pudiera registrar información inmutable en ella, la cual jamás sería referenciada y gastada?

El operador `OP_RETURN` va seguido de un campo de datos, en el cual se puede escribir hasta 80 *bytes*.

De modo, que el `OP_RETURN,` proporciona una forma de registrar información inmutable e ingastable en el *blockchain*. La utilidad de esta herramienta es muy amplia: notarios online, registros de archivos, contratos inteligentes, certificados…

Este tipo de transacciones solo tiene un *script* de bloqueo, el cual tiene el siguiente formato:

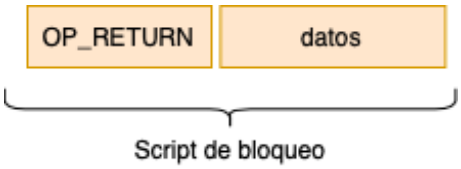


Ilustración 6-16 OP-RETURN Script de bloqueo

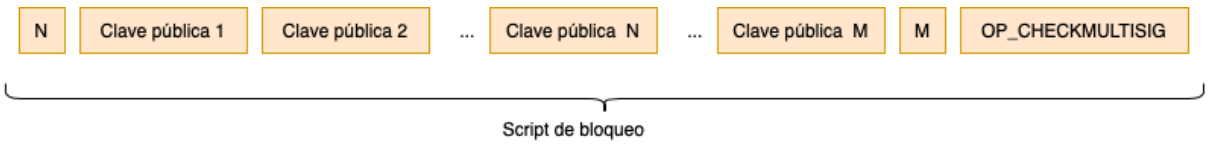
Además, desde el punto de vista del almacenamiento, no supone un coste adicional para los nodos completos que recogen el total de las UTXO, ya que las transacciones que contengan en sus salidas este operador serán ignoradas, y, por tanto, no almacenadas, liberando así al nodo del mantener en su reserva UTXO este tipo de transacciones.

6.4.5. Pay to Script Hash (P2SH)

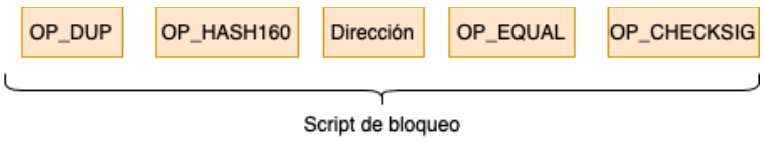
Volviendo al apartado 6.4.3, suponga que su nodo ejecuta una versión del protocolo *Bitcoin,* en la cual no existe la posibilidad de hacer pagos a direcciones que bloqueen la transacción usando MULTISIG, y alguien le reclama efectuar un pago determinado a su dirección. Ahora resulta que esta dirección utiliza una versión, en la cual sí se da soporte a la posibilidad de realizar transacciones MULTISIG y requiere un bloqueo *multi-firma*. ¿Qué haría en este caso?

Además, supóngase un escenario en el cual un minero recoge miles de transacciones, cada una de ellas del tipo MULTSIG y con 14 de 15 firmas necesarias. El *script* de bloqueo tendría un tamaño mucho más grande, en comparación al de transacciones P2PKH o P2PK. Este aumento de tamaño significa que el minero tendrá que incluir transacciones de mayor tamaño en el bloque, lo que supondría un aumento en el coste de comisión para los emisores de las transacciones.

Recuérdese que el formato de un *script* de bloqueo de una transacción de tipo MULTISIG era el siguiente:



Y el de una transacción P2PKH:



La diferencia de tamaño es evidente: en P2PKH con una única dirección, solo se requiere una longitud fija de 160 bits. En cambio, en las transacciones de tipo MULTISIG se requiere de, al menos, una clave pública de 256 bits. Para nuestro ejemplo de 14 de 15 firmas, la relación anterior solo incrementaría en contra del tipo MULTISIG.

Este coste de comisión no lo asumiría la persona que está implementando el *script* de desbloqueo, sino que sería el pagador, quien es totalmente ajeno a esta característica, quien tendría que asumir una comisión mayor, por el simple hecho de que a la dirección a la que está efectuando el pago, implementa algunas funcionalidades complicadas.

El beneficio del uso de las transacciones P2SH es alto e interesante, no solo desde el punto de vista de la comisión, sino también en lo relativo a los beneficios de ocupación de memoria y almacenamiento: hasta que una salida no es gastada, la UTXO estará en la reserva, consumiendo memoria y almacenamiento del nodo completo. Por tanto, cualquier forma de optimizar y minimizar el tamaño de las UTXO interesará a un nodo que las almacena.

Los ejemplos anteriores se han ilustrado por medio de las transacciones MULTISIG, pero no están limitados en cuanto al alcance, en lo que se refiere a variedad de tipos de *scripts* de bloqueo que puedan implementar. De la misma forma que se ha dicho MULTISIG se podría haber tratado con cualquier otro tipo de *script* de bloqueo, el cual necesitase alguna funcionalidad rara o compleja. La necesidad de buscar las mismas soluciones a los problemas que plantearían sería exactamente la misma.

Lo que realizan las transacciones P2SH es una especie de *bypass,* mediante el cual se ignora cuáles son las funcionalidades complejas que un determinado usuario pueda implementar. Para lograrlo, en vez de incluir las ordenes *Script* en el *script* de bloqueo, se incluye simplemente su *hash*. De esta forma, al igual que en las transacciones P2PKH, donde se paga al *doble-hash* de una clave pública, en las P2SH se paga a un *doble-hash* de un *script*.

Como se indicaba en 4.7.2, existe un tipo de direcciones *Bitcoin* especiales: las direcciones P2SH. Estas se procesan de la misma forma que las direcciones a partir de clave pública, pero con la diferencia de que los datos de entrada no son una clave publica, sino órdenes del lenguaje *Script* especificadas en un *script*.

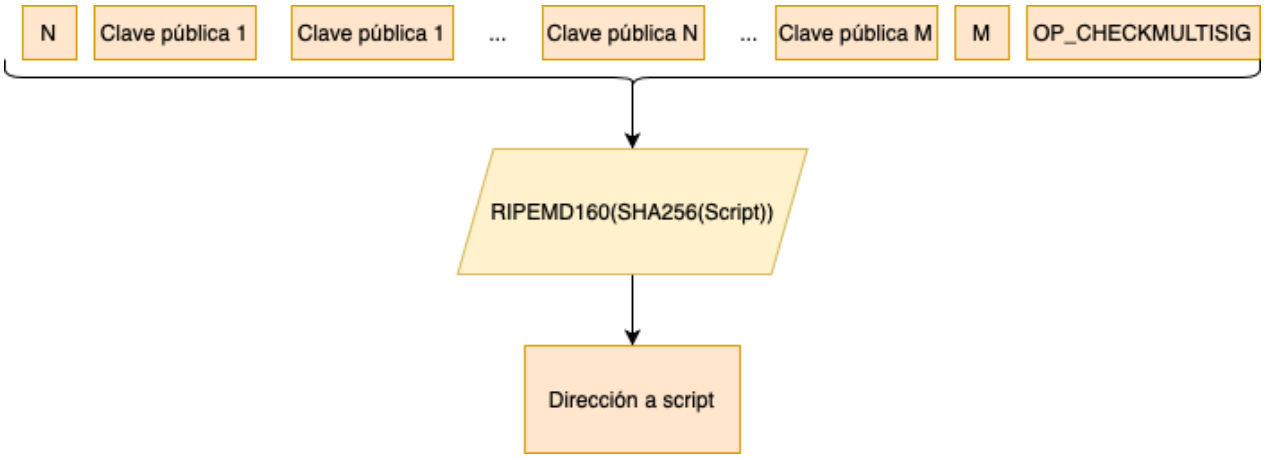


Ilustración 6-17 Dirección P2SH

Mediante el uso de esta funcionalidad, se libera al pagador de comisiones molestas por culpa de que un nodo implemente órdenes inusuales en un *script* de desbloqueo.

La idea fundamental de este proceso es que sea el liquidador de la transacción aquél que proporcione el *script* (el que representa la dirección P2SH) durante el proceso de desbloqueo. Este *script* se llama habitualmente como *redem-script.*

Un *redem-script* MULTISIG para una transacción P2SH tiene el siguiente formato:

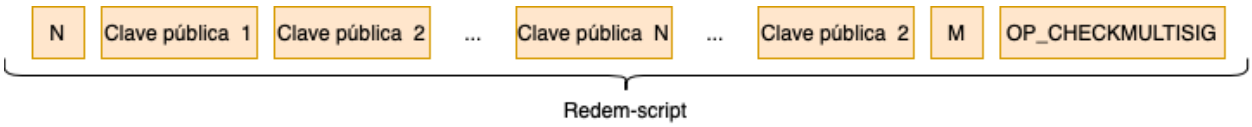


Ilustración 6-18 Redem script MULTISIG P2SH

El formato de un *script* de desbloqueo dependerá del tipo de instrucciones incluidas en el *redem-script.* En nuestro caso será un *script* de desbloqueo en el cual se aporten las firmas necesarias.

Cabe destacar que, aunque se ha usado un *redem-script* que contenía las ordenes para un desbloqueo de tipo MULTISIG, el *redem-script* no tiene por qué ser de este formato. Es cierto que casi todos los *redem-scripts* que se usan en la red son de este tipo, pero en realidad podría estar formado por cualquier conjunto de ordenes *Script.* Esto es muy importante tenerlo en cuenta. Si se entiende esto se habrá entendido cuál es el poder de las transacciones P2SH.

Puede que la idea del funcionamiento de estas transacciones esté un poco difusa. Para aclarar este uso se muestra el siguiente ejemplo:

Supóngase que Juan y María son propietarios de una cuenta común. Juan y María no confían el uno en el otro, y deciden aplicar una regla de multi-firma (2 de 2) para cobrar los pagos que reciba la cuenta. Por tanto, aplican el esquema de transacciones MULTISIG para liberar UTXO a sus direcciones. Además, son conscientes de que, los nodos que quieran enviarles transacciones y que no implementen soporte para MULTISIG, podrían tener problemas para "entender" este tipo de transacciones; por tanto, deciden usar transacciones P2SH.

El *redem-script* que María y Juan usarán es el siguiente:

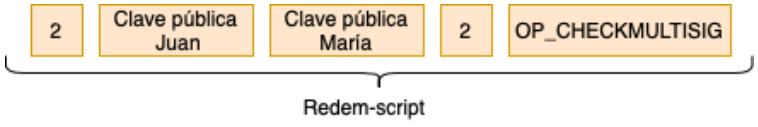


Ilustración 6-19 Redem-script Juan y María

A este *script* se le aplica el *doble-hash* RIMPEND160 y SHA256, y se obtiene una dirección P2SH.

Ahora Paco quiere traspasar una serie de BTC a la cuenta de María y Juan. Paco pregunta por la dirección y se le proporciona la dirección P2SH (ver Ilustración 6-17). Paco elabora la transacción, la cual se incluye en el siguiente bloque. La transacción se marca como una salida de una UTXO bloqueada por el *hash* del *redem-script* indicado en Ilustración 6-18, el cual especifica la dirección P2SH de Juan y María.

La aplicación de cartera de Juan y María pregunta por todas las UTXO reclamables por su dirección, y obtiene como respuesta la transacción de Paco.

Ahora Juan y María quieren liquidar esta UTXO para poder pagar otros gastos. El proceso de liberación de la salida de la UTXO se divide en dos pasos: 1) proporcionar el script de desbloqueo 2) aportar el *redem-script*:

Redem-script:

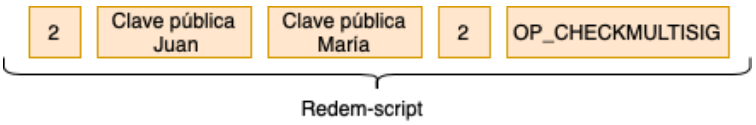


Ilustración 6-20 Redem-script de Juan y María

Script de desbloqueo:

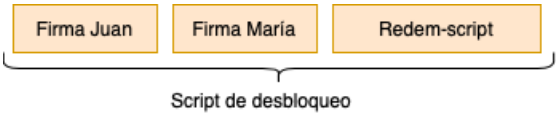


Ilustración 6-21 Script de desbloqueo María y Juan

Cabe destacar que el *script* de desbloqueo se parte para cada paso:

1) Proporcionar el script de desbloqueo

La secuencia de ordenes completa (concatenación del *script* de desbloqueo sin firmas y *script* bloqueo) para la primera parte de validación es:

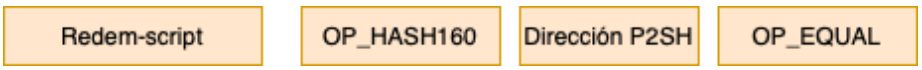


Ilustración 6-22 P2SH primera secuencia de validación

Esta secuencia simplemente comprueba que el *hash* del *redem-script (*Dirección *P2SH)* proporcionado por María y Juan, concuerda con la dirección especificada en el *script* de bloqueo.

Si ambos son iguales se ejecuta la segunda secuencia:

2) Aportar el *redem-script*:

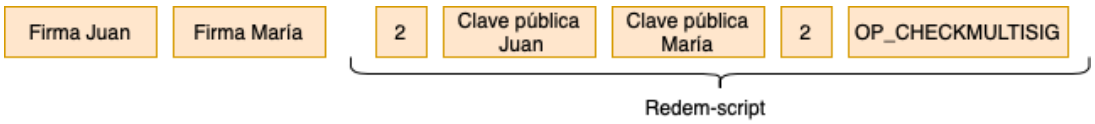


Ilustración 6-23 P2SH segunda secuencia de validación

De la secuencia especificada en Ilustración 6-23 son exclusivamente responsables Juan y María. Es aquí donde se ilustra la ventaja de este tipo de transacciones. En la primera secuencia (Ilustración 6-22) simplemente se comprueba que las direcciones concuerdan (se ha liberado al creador de las UTXO de la responsabilidad de proporcionar las ordenes indicadas en el *redem-script*). Es en la secuencia de la Ilustración 6-23 donde Juan y María tendrán que proporcionar el *redem-script* para poder reclamar los BTC.

6.5. Comisiones de transacción

Las comisiones de transacciones en *Bitcoin* se establecen con el objetivo de recompensar a aquellos nodos mineros que introducen transacciones en el bloque que proponen.

En un primer momento, las transacciones *Bitcoin* estaban fijadas a un valor fijo, pero según la red fue desarrollándose, se estableció un modelo por el cual las comisiones se cobraban dinámicamente en función de su tamaño.

Cabe destacar que la comisión no se calcula en función de la cantidad BTC que las transacciones recogen, sino, como se ha mencionado, en función de su tamaño. Por lo que una transacción de 10 BTC que, por ejemplo, pague a múltiples salidas, tendrá una comisión mayor que una que pague a solamente a una. En concreto la comisión que cobra actualmente (10 de junio de 2020) por kB de transacción es de 0,00018 BTC. Está transacción se calcula diariamente, y fluctúa según las tendencias del mercado (oferta-demanda actual de *Bitcoins*).

Además, las comisiones, aparte de ser calculadas dinámicamente, pueden modificarse a voluntad del creador de la transacción, pudiendo cobrar más, menos, o incluso nada. Como es de esperar, los mineros favorecerán aquellas transacciones dentro de su *pool* de transacciones, las cuales tengan una comisión mayor. Por tanto, establecer una comisión mayor para una transacción determinada, favorecerá su procesado por parte de los mineros.

7. Red Bitcoin

En este capitulo se presenta la estructura por la cual todos los elementos anteriores son capaces de interoperar: comunicación entre nodos, intercambio de transacciones, consulta de UTXO, etc.

La red *Bitcoin* se basa en un protocolo *Bitcoin P2P*. En dicho protocolo, no existen nodos jerárquicos: todos los nodos son "iguales", y se encargan tanto de proveer, como de solicitar servicios los unos a los otros. Esta homogeneidad proporciona una topología de red plana y descentralizada, muy adecuada para una herramienta descentralizada como es *Bitcoin*. La elección de esta arquitectura proporciona la herramienta base sobre la cual se da soporte al consenso implícito, validación de transacciones y mantenimiento del *blockchain*.

Se puede entender la red *Bitcoin* como un subconjunto de una red mayor llamada red *Bitcoin* extendida, la cual da soporte a protocolos especializados y enfocados a minería de bloques (cualquier otro tipo de servicio que requiera una conexión con algún componente *Bitcoin P2P).*

7.1. Tipos de nodos Bitcoin

Aunque se diga que los nodos son iguales, esto no es completamente cierto: todos los nodos comparten las capacidades de enrutamiento o *Network-routing*. Estas funcionalidades posibilitan la interconexión de diferentes *peers* en la red, pero, además de esta capacidad básica, los nodos pueden tener las funcionalidades de *Full-blockchain*, *Wallet* y *Miner*.

Los nodos con funcionalidad de tipos *full-blockchain* recogen una copia completa de la cadena de bloques, desde la última altura hasta el bloque génesis (primer bloque de la cadena).

Los nodos con funcionalidad *Wallet* son aquellos que recogen todas las funcionalidades de una cartera *Bitcoin*. Una cartera realiza todas las acciones de mantenimiento de claves de una dirección, creación de nuevas claves, consulta de UTXO asociadas a una dirección, gestión, creación y cobro de transacciones, etc. En resumen, todas las funcionalidades que haría una cuenta de ahorros tradicional.

Los nodos con funcionalidad *Miner* son los nodos que participan en la carrera de resolver los *hash-puzzles* o algoritmo de *proof of work*. Estos nodos son los encargados de minar nuevos bloques, de validar las transacciones que recogen, de cobrar comisiones, etc.

Mediante la combinación de estás funcionalidades (*Network-routing*, *Full-blockchain*, *Wallet* y *Miner*) se pueden definir los diferentes tipos de nodos que existen en la red *Bitcoin* simple (sin tener en cuenta protocolos adicionales):

Nodos completos o Full node

Se dice que los nodos que implementan estas cuatro funcionalidades son nodos completos o *full-nodes*. Estos nodos podrán validar transacciones, sin ninguna consulta externa, ya que mantienen una base de datos con todas UTXO reclamables.

Estos nodos completos, almacenan una copia completa de todo el *blockchain*, desde el bloque génesis hasta la última altura conocida. Esto les proporciona una gran ventaja frente a otros nodos que no guardan una copia completa de la cadena: pueden construir su propia reserva UTXO escaneando todos los bloques. Ahora bien, como es de esperar esta independencia también supone disponer de un gran espacio de almacenamiento de, al menos 65 GB (a fecha de 16 de junio de 2021).

La implementación de referencia de los nodos completos es la que proporciona el cliente *Bitcoin Core*. Las versiones de *Bitcoin Core* se identifican como *Satoshi* y se indican en el campo de *sub-version* del mensaje *version* (ver Ilustración 7-7 Formato del payload de versión)

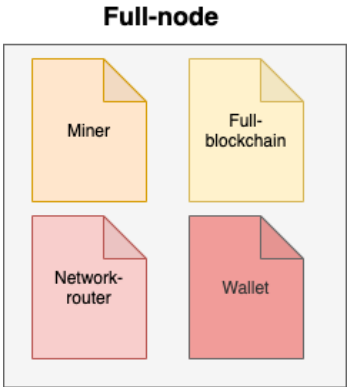


Ilustración 7-1 Full node

Nodos light-weight o SPV

En cambio, los nodos *light-weight* o SPV, son nodos creados exclusivamente para clientes que operan en un dispositivo con capacidades limitadas, como puede ser una *Tablet* o un *Smartphone.*

La diferencia principal de estos nodos frente a los nodos completos, es que almacenan una copia parcial de la cadena de bloques: solo almacenan las cabeceras, lo cual supone almacenar una copia local del *blockchain* 1000 veces menor[13] respecto a lo que supondría una copia completa.

Por tanto, esto requiere de una forma diferente de validación de transacciones en un bloque, ya que no disponen de una base de datos actualizada con todos los UTXO. La forma de validar las transacciones es mediante el valor del campo *raíz-merkle* de la cabecera del bloque y el *merkle-branch* para la transacción que se requiera validar. Este *merkle-branch* se solicita a los nodos completos (ver 5.2.3).

[13] Fuente: “Mastering Bitcoin: Andreas Antonopoulus”. Página 165.

Los nodos SPV o *light-weight* son los más utilizados hoy en día debido a la reducción de recursos que supone su implementación y despliegue en un dispositivo.

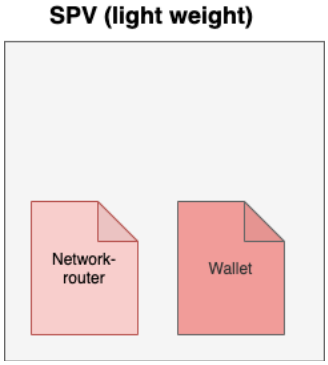


Ilustración 7-2 Nodo SPV

Solo miner

También existe otro tipo de nodo, el cual se encarga de realizar las operaciones de *hashing* necesarias para resolver el *hash-puzzle* o *proof of work:* Nodo *Solo Miner.*

Este nodo es, también, del tipo *Full-blockchain,* ya que de esta forma agiliza el proceso de creación del bloque, validando él mismo las transacciones de su *pool,* con los registros indexados en su base de datos de UTXO.

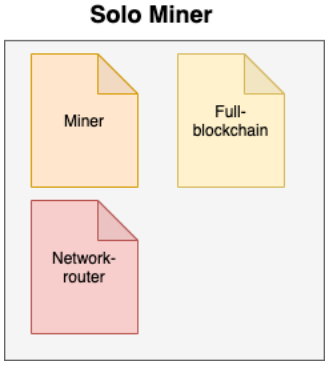


Ilustración 7-3 Nodo solo Miner

Full blockchain

Por último, existen los nodos *Full-blockchain,* los cuales son los encargados, entre otras tareas, de dar soporte a nodos SPV: por ejemplo, cuando estos últimos requieren un *merkle-branch* para la validación de una transacción en un bloque (ver 5.2.4).

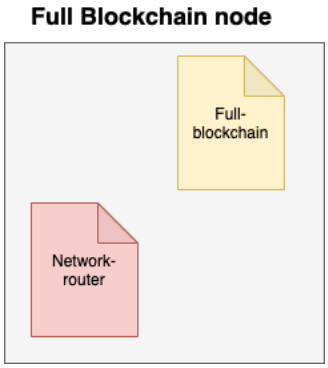


Ilustración 7-4 Nodos Full-blockchain

7.2. Tipos de nodos en *Bitcoin* extendido

Como se ha mencionado anteriormente, la red *Bitcoin* es el subconjunto de una mayor, en la cual se incluyen diversos tipos de protocolos especializados que se comunican con el protocolo P2P *Bitcoin*.

Esta red consta con una infraestructura en la cual hay una serie de nodos que actúan como *gateway* entre la red P2P Bitcoin y otra serie de nodos que realizan funciones especializadas, como puede ser la resolución del *hash-puzzle* o *proof of work*. Los nodos que operan como *gateways* son habitualmente conocidos como *gateway-routers* y tienen la siguiente estructura:

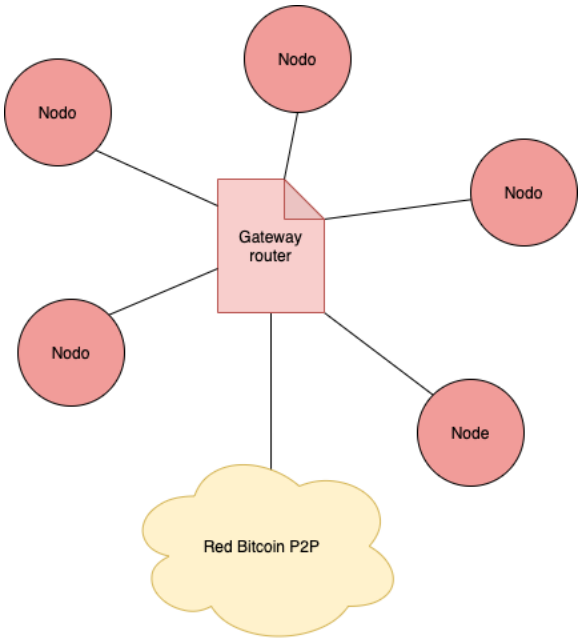


Ilustración 7-5 Gateway router

Un ejemplo de *gateway-router* puede ser los *clusters* mineros bajo protocolo *Stratum*.

Protocolo Stratum:

Este protocolo sirve para interconectar nodos de un *pool* de mineros. Este *pool* tiene como objetivo aunar nodos para que cada uno de ellos resuelva combinaciones de diferentes *hashes* para resolver el *hash-puzzle* o *proof of work*.

En las primeras versiones de *Bitcoin*, se usaba el antiguo protocolo *getwork,* servido mediante servidores HTTP*,* y en el cual, simplemente, se enviaban a un minero una serie de bloques de datos (no confundir con bloques del *blockchain*) que debían ser procesados, y, además, un *target.* Si después de calcular el *hash*, el resultado estaba por debajo de los rangos indicados en el *target*, el minero habría resuelto el *hash-puzzle* o *proof of work*.

El protocolo Stratum surge de la necesidad de proveer al proceso de minado en *cluster* de un protocolo propio y no depender de servidores HTTP, los cuales no fueron diseñados para el proceso de minado. Mediante el uso de este protocolo, el papel del servidor HTTP se elimina, dando al nodo el control absoluto de los mensajes que envía, sin preocuparse de aspectos como el *long-pooling* o balanceo de carga.

El protocolo *Stratum* simplemente crea una conexión TCP/IP entre los nodos mineros, donde la deserialización y serialización de datos se realiza mediante una API JSON.

Todos estos elementos (*Full Node, Full-Blockchain, Solo Miner, SPV, Gateway-router,* etc.) interoperan en la red *Bitcoin* extendida.

7.3. Descubrimiento de red

Para que un nuevo nodo sea capaz de conectarse a la red, hace falta que el nuevo nodo conozca, al menos, un nodo en la red. Ahora bien, ¿cómo encuentra este primer nodo?

Existen dos métodos fundamentales: 1) Descubrimiento por dirección previamente conocida 2) Descubrimiento por nodos semillas DNS (*Domain Name System*).

1) Descubrimiento por dirección previamente conocida

Supóngase que un nodo se ha desconectado por un largo periodo de tiempo, pero que tiene almacenado algunas direcciones IP de nodos que ha ido conociendo en sus anteriores conexiones. El nodo intentará conectar con estos nodos y, si alguno de ellos sigue en línea, podrá conectarse a la red.

2) *Descubrimiento por nodos semillas DNS.*

Por el contrario, si no conoce ningún nodo o los nodos que conocía ya no están disponibles, tendrá que realizar el descubrimiento por nodos semilla DNS:

El proceso de descubrimiento DNS consiste en peguntar a una serie de servidores DNS, los cuales almacenan una lista de direcciones IP de nodos *Bitcoin* estables a lo largo del tiempo. La selección de las direcciones en este listado, se realiza de una forma totalmente aleatoria,

de entre un conjunto de direcciones conocidas. Actualmente, el cliente de referencia *Bitcoin Core* dispone de 5 servidores DNS, cada uno de ellos, con diferentes tipos de propiedades y de implementaciones, lo cual ofrece una alta garantía de fiabilidad para el proceso de descubrimiento.

El *handshake* de conexión entre nodos se realiza mediante una conexión TCP/IP al puerto 8333.

El *handshake* de conexión entre dos nodos de la red se realiza de la siguiente manera:

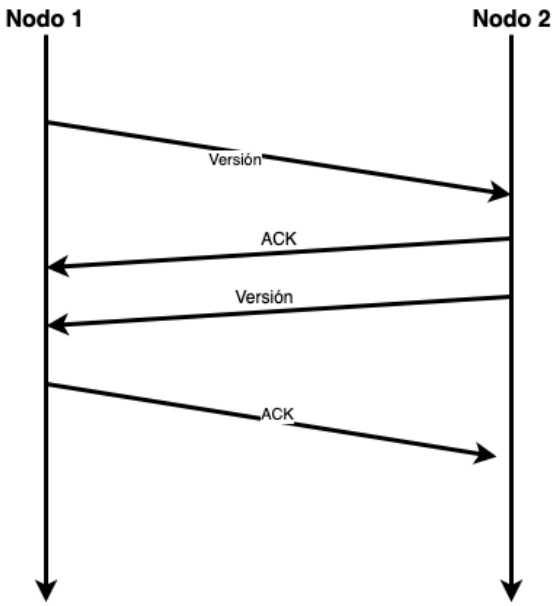


Ilustración 7-6 Handshake de descubrimiento

El *payload* de los mensajes de *version* tienen el siguiente formato:

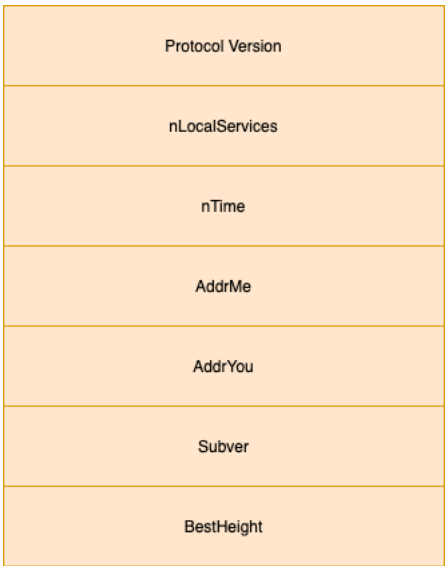


Ilustración 7-7 Formato del payload de versión

1) ***Protocol Version*:** Versión del protocolo P2P *Bitcoin* que el nodo opera.
2) ***nLocalServices*:** Lista de los servicios soportados por el nodo.
3) ***AddrMe*:** Dirección IP del nodo emisor del mensaje *version.*
4) ***AddreYou:*** Dirección IP del nodo al que se envía el mensaje *version.*
5) ***Subver:*** Tipo de *software* que se ejecuta en el nodo.
6) ***BestHeight:*** La altura del *blockchain* del nodo descubridor.

Una vez que el nodo ya tiene otro con el que conectar, puede enviar los mensajes *addr* y *getaddr:*

El mensaje *addr* envía la dirección del nodo que está realizando el proceso de descubrimiento a los nodos con los que esté conectado. Los nodos que reciben este mensaje envían la dirección del nodo a los demás nodos con los que estos también estén conectados.

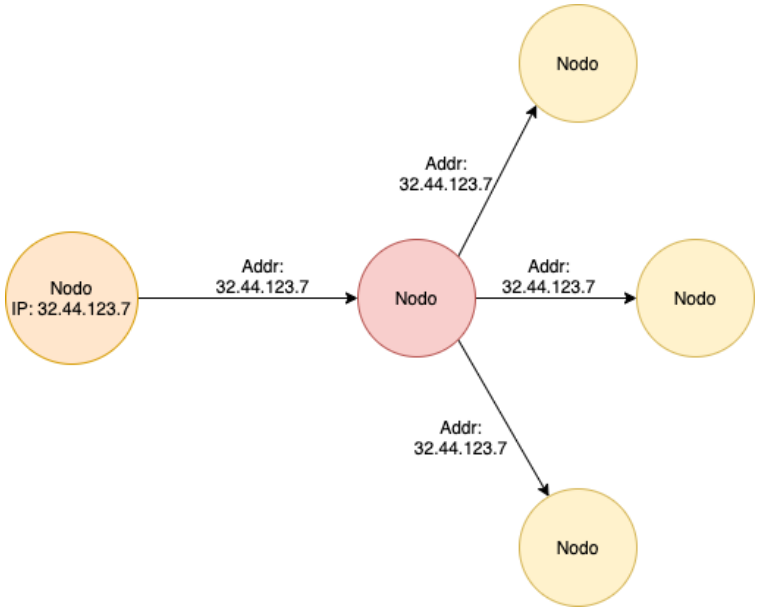


Ilustración 7-8 Proceso de descubrimiento: addr

Por el contrario, el mensaje *getaddr*, el nodo descubridor solicita la lista de los nodos con los que un nodo esté conectado:

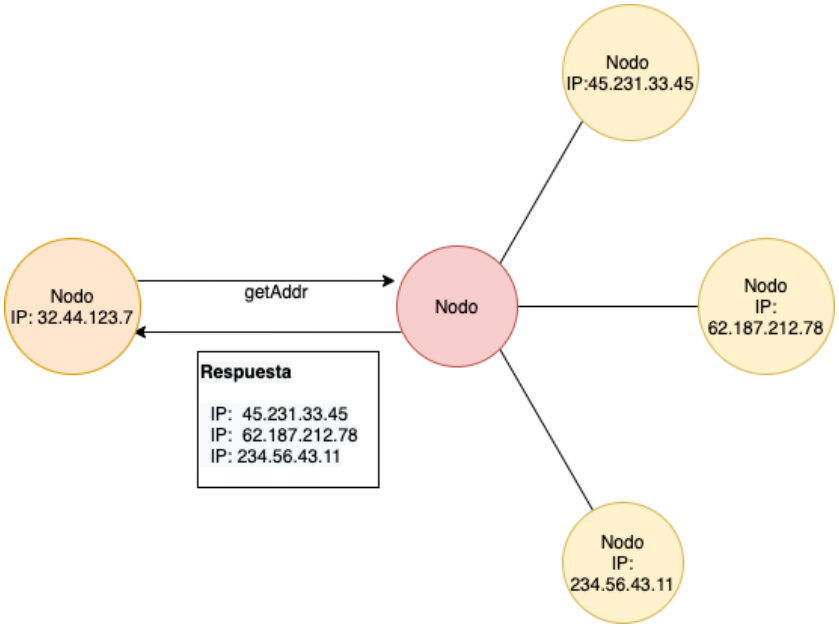


Ilustración 7-9 Proceso de descubrimiento: getaddr

7.4. Actualización del *blockchain* local de un nodo

En 7.3, se explica la forma por la cual un nodo descubre a otros en la red P2P. En este apartado se explica cuáles son los mecanismos para, una vez establecidas estas conexiones, indicar cómo un nodo, con una cadena de bloques desactualizada, consigue ponerse a la par de la última altura de bloque conocida.

El proceso de actualización es el descrito en Ilustración 7-10.

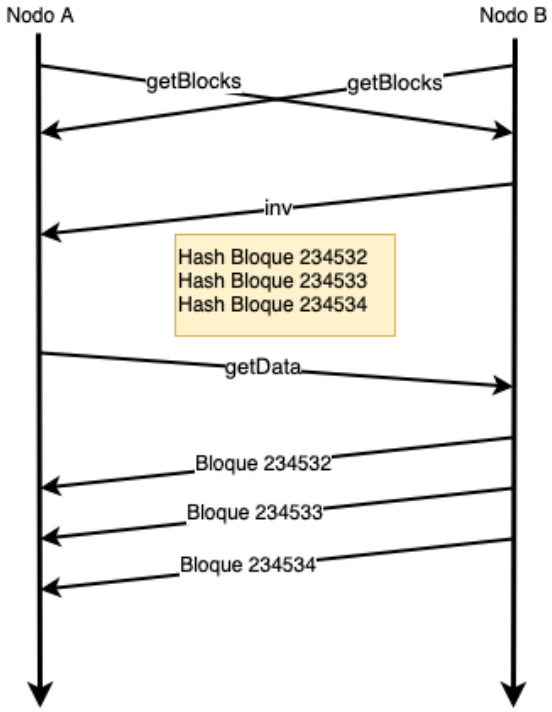


Ilustración 7-10 Proceso de actualización del blockchain

A continuación, se presenta una explicación para cada uno de los tipos de mensajes envueltos en la secuencia descrita en Ilustración 7-10.

Mensaje GET_BLOCKS

En el proceso de descubrimiento, los nodos intercambian el mensaje `version`, en el cual el campo *BestHeight* indica la altura del *blockchain* local de cada uno de los nodos. Una vez se realiza lo anterior, ambos nodos intercambian el mensaje `getBlocks`, en el cual se envía el *hash* de la cabecera del último bloque de cada nodo. Mediante esta información, uno de ellos será capaz de identificar el *hash* del otro nodo en su *blockchain* local, ya que podrá localizarlo en bloques anteriores de su copia local de la cadena de bloques, escaneando y comparando el *hash* de la cabecera del bloque con los ya incluidos en la copia local del nodo.

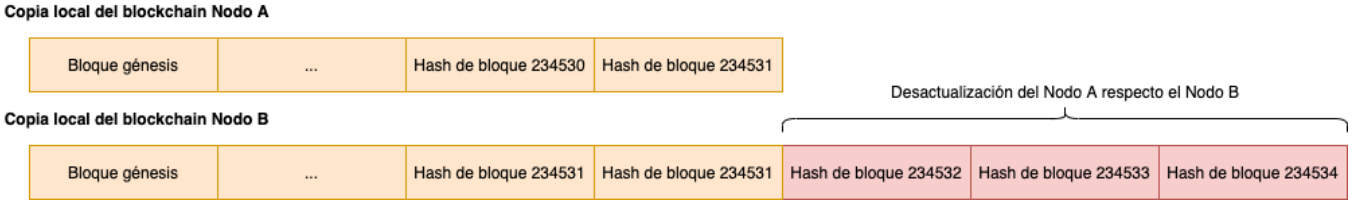


Ilustración 7-11 Blockchain desactualizado

Mensaje `INV`

Cuando los nodos han identificado cuál es el nodo que tiene la cadena de bloques desactualizada, el nodo con la altura mayor envía un mensaje de tipo `INV` (inventario). Este mensaje inventario contiene una lista con los *hashes* de las cabeceras de los bloques que le faltan al nodo con la copia atrás del *blockchain*.

Para el ejemplo, se enviarían los *hashes* de las siguientes cabeceras de bloques:

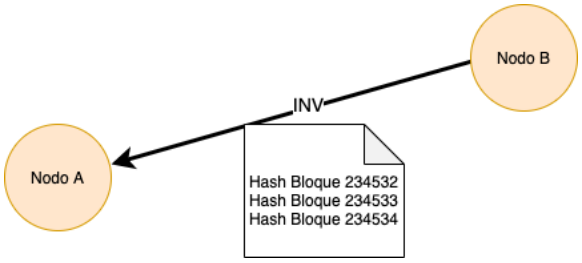


Ilustración 7-12 Mensaje INV

Cabe destacar que, si la desactualización fuese superior, por ejemplo, de 100.000 bloques, el nodo B solo respondería con un inventario de 500 *hashes* como máximo. El nodo A, una vez finalizado el proceso de actualización para los primeros 500 bloques de 100.000, tendría que repetirlo hasta actualizar la copia de la cadena por completo

Mensaje GET_DATA

Una vez el nodo recibe el mensaje de inventario, envía el mensaje `getData`. Este mensaje solicita el envío por parte del nodo con la cadena de boques de altura mayor, de los bloques que le falta para tener una copia actualizada. En el ejemplo el nodo B respondería con 3 bloques.

Cabe destacar que para no saturar la conexión entre *peers* el nodo que solicita los bloques, deberá llevar un contador de todas las peticiones que estén en tránsito y todavía no hayan sido satisfechas. Este número no debe sobrepasar el límite indicado por `MAX_BLOCKS_IN_TRANSIT_PER_PEER`. Esta variable es configurable según el ancho de banda de cada nodo. Si se sobrepasa dicho valor, el nodo solicitante deberá solicitar estos bloques a otros nodos de la red.

7.5. Solicitud de *Merkle-branch*

La solicitud de *merkle-branch* se realiza cuando se requiere validar una transacción en un nodo SPV, el cual no dispone de una copia completa del *blockchain*. Como se sabe, estos nodos solo disponen de las cabeceras, las cuales contienen una raíz del *merkle-tree*.

El proceso de solicitud se expone en la Ilustración 7-13:

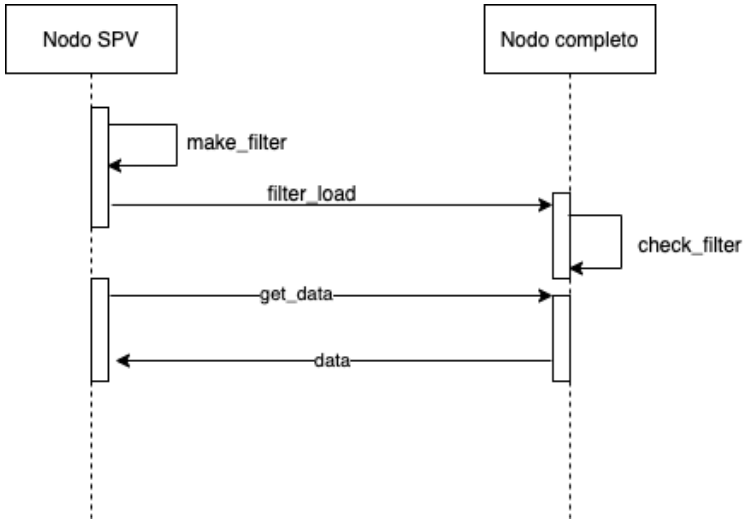


Ilustración 7-13 Diagrama de secuencia solicitud de merkle-branch

Para la validación de una transacción dentro de un bloque, lo que realizará un nodo SPV es emitir un mensaje *filter_load* el cual contiene un *filtro-bloom.*

Filtros *bloom*

Un *filtro-bloom* es un mecanismo por el cual se permite realizar una búsqueda sin proveer mucho de detalle del elemento por el cual se pregunta. Un *filtro-bloom* tiene dos elementos fundamentales: una lista de funciones *hash* y un *array* de bits.

El funcionamiento de los filtros es bastante sencillo:

1) Una transacción se toma como valor de entrada de cada una de las funciones *hash* que estén especificadas en la lista de funciones *hash*.
2) Estas funciones *hash* devuelven un *output*, el cual no es más que un índice indicando una posición dentro del rango del tamaño del *array*.
3) Se introduce un bit a 1 en la posición indicada. Si un *output* da un índice en el cual su valor ya está a 1, se mantiene.
4) Se repite este proceso para el total de transacciones que se quieran filtrar

La motivación fundamental por la cual se utilizan estas herramientas es por la privacidad. Es una búsqueda probabilística, en la que a más posiciones marcadas a 1 en el *array*, a más tamaño del *array*, y a más funciones de *hash* usadas, menor será la información que se proporciona de la transacción o transacciones que se soliciten.

Por tanto, un nodo SPV computará el filtro para todas las transacciones por las que desee obtener su correspondiente *merkle-branch,* y envía el resultado en un mensaje `filter_load` (el filtro contiene las funciones de *hash* y el *array* de resultado, de manera que el nodo completo pueda recrearlo). Este mensaje no contiene una información explicita de la transacción, sino que establece una forma por la cual puede que haya transacciones que se ajusten al patrón especificado con una cierta probabilidad.

Cabe destacar que, a más privacidad (más transacciones y más funciones de *hash* del *filtro-bloom*), menos exacta será esta búsqueda. Se estarán devolviendo transacciones las cuales se ajusten al patrón, pero que quizás no sean la/s solicitada/s. Eso sí, de esta forma no se compromete los datos de las direcciones, *scripts* o movimientos implicados.

Los nodos completos comprueban todas las salidas que se ajusten al patrón indicado y se mantendrá a la espera de recibir un mensaje `getData`. El nodo SPV emitirá un mensaje `getData` para solicitar el resultado computado por el nodo completo. El nodo completo le responderá con la cabecera, *merkle-branch* y transacción que ha satisfecho el patrón indicado en el *filtro-bloom*.

7.6. Actualización del protocolo

Las actualizaciones del protocolo son unas de las acciones más complejas a las que se puede someter a la red. La actualización en una red descentralizada, en la cual no existe un servidor central que pueda forzar la actualización, puede levar a escenarios indeseados que tienen que ser evitados bajo toda circunstancia.

Cuando se introduce una actualización el protocolo *Bitcoin* se pueden dar dos situaciones:

- ***Hard-fork*:** Se dice que una actualización provoca un *hard-fork* cuando una actualización implementa funcionalidades no permitidas en versiones anteriores del protocolo. Esto genera que se creen dos bifurcaciones (*forks* en inglés), lo cual produce dos cadenas de bloques alternativas: una bajo las restricciones de la versión antigua, y la otra bajo las de la nueva. Esta situación es inaceptable, debido a que, básicamente, se está echando de la red a los nodos que no actualicen. Por tanto, las actualizaciones que puedan provocar un *hard-fork*, deben de evitarse o, si no es posible, modificarse de manera que solo creen un *soft-fork*.

- ***Soft-fork*:** Las actualizaciones *soft-fork* implementan restricciones más estrictas a las ya existentes. Por ejemplo, un caso de *soft-fork* es la implementación de una BIP. Los *soft-forks* requieren que una mayoría de los nodos de la red hayan actualizado a la nueva versión. De esta forma, los nodos que no hayan actualizado podrán seguir operando en la red y proponiendo bloques. Estos nuevos bloques, generados por nodos con una versión obsoleta, serán rechazos. Puede que el nodo no entienda el porqué de que los demás nodos estén rechazando sus bloques, pero está situación le hará comprobar si es debido a que existe una nueva versión de protocolo, y, por tanto, se verá forzado a actualizar.

8. Minería

El proceso de minería es el instrumento por el cual se crean nuevos *Bitcoins*, pero más importante aún, es el instrumento que proporciona un sistema basado en el consenso implícito para el mantenimiento y el correcto funcionamiento de la red.

Una red monetaria descentralizada no es solamente compleja desde el punto de vista tecnológico, también, lo es desde el punto de vista sociológico: como se sabe, la red *Bitcoin* se apoya en la tecnología del *blockchain* para mantener un registro completo de todos los movimientos o transacciones que se realizan entre nodos. Pero no existe una cadena central y única, son los *peers* de la red, los que se encargan de mantener una copia local de la cadena de bloques. Cada nodo se encarga de validar, incluir, desechar o trasmitir un bloque a los demás. Por tanto, para que el sistema funcione adecuadamente, tiene que haber una herramienta, la cual permita que los nodos de la red favorezcan los comportamientos honestos y penalicen los deshonestos. Es aquí, donde entra el consenso implícito. Es esta la herramienta que permite que *Bitcoin* opere de la forma deseada y se cree una cadena de bloques consensuada.

El consenso implícito es un complejo sistema, basado en ingeniería social, que proporciona a la red una herramienta de incentivos que premia a los nodos participantes con ciertas recompensas. Para obtener estas recompensas, los nodos se tienen que ajustar a las reglas del protocolo vigente, o por el contrario serán excluidos de la red, no por ninguna autoridad central, ni nada del estilo, sino por los demás nodos de la red.

Para que el consenso implícito sea exitoso, tiene que haber una mayoría de nodos honestos, los cuales, sí siguen las pautas definidas por el protocolo. En el caso de que la mayoría de los nodos fueran deshonestos, la red fracasaría, haciendo que los demás nodos honestos perdiesen la confianza en la misma. Este tipo de ataques al protocolo se denominan “ataques 51%”, pero por suerte, en el estado actual de la red, es altamente improbable que se de tal ataque (dadas las características del *proof of work*).

Uno de los elementos más esenciales del consenso implícito es la prueba de trabajo o *proof of work* en inglés. El *proof of work* es el mecanismo por el cual se demuestra que un nodo ha realizado cierta tarea (bajo los parámetros del protocolo), la cual le ha llevado un gran esfuerzo, computacional, en resolverlo. Técnicamente, el *proof of work* demuestra que el nodo minero ha estado tratando de resolver el *hash-puzzle* o algoritmo de *proof of work* durante un periodo de tiempo determinado (10 minutos de media), probando trillones de combinaciones *hash* hasta que una de estas combinaciones haya caído por debajo de un objetivo determinado (*target*).

Una vez el minero está en condiciones de aportar una prueba de trabajo válida, realizará las acciones necesarias para incluir información en un nuevo bloque. Una vez el bloque se ha creado, el nodo lo retransmitirá a los nodos con los cuales esté conectado. Estos nodos retransmitirán el bloque mediante un algoritmo de *flooding,* y en cuestión de segundos, el bloque habrá llegado a prácticamente la totalidad de la red.

El proceso de minería, como se ha mencionado, es un proceso basado en incentivos que premia el buen comportamiento de los nodos honestos de la red. Estos incentivos se obtienen de dos formas fundamentales: transacciones *coinbase*, las cuales crean nuevos *Bitcoins,* y el cobro de comisiones por transacción.

El ritmo de emisión no es fijo, sino que es decreciente. En concreto, la cantidad de emisión de *Bitcoins* se reduce a la mitad cada 210.000 bloques. Cuando *Bitcoin* empezó a entrar en funcionamiento, la cantidad que se recogía en una transacción *coinbase* era de 50 BTC. Este proceso de emisión decreciente provoca que la masa monetaria *Bitcoin* tenga un límite superior. Este límite es de 2.099.999.997 *Satoshis,* o casi 21 millones BTC. Este hecho hace que, en un futuro (aproximadamente 2140), se hayan emitido todos los BTC posibles. A partir de entonces, el sistema de incentivos se basará exclusivamente en las comisiones cobradas a las transacciones que un minero incluye en bloque.

Las comisiones de transacciones (ver 6.5) aseguran el cobro de una cierta cantidad BTC por kB de transacción. Actualmente, las ganancias de los mineros procedentes de comisiones, es un porcentaje ínfimo de sus ingresos. Se espera que, a medida que la emisión de masa monetaria *Bitcoin* se vaya contrayendo, el papel de las transacciones cobre más importancia.

Estos son los elementos fundamentales sobre los que se apoya la minería. En el presente capitulo, se detallará cada uno de los pasos que hay que tomar para lograr el consenso implícito en la red, y cómo se desarrolla el proceso de minado o la competición minera, por parte de los nodos mineros de la red.

La explicación de cómo alcanzar el consenso, se realizará siguiendo el diagrama que se presenta a continuación:

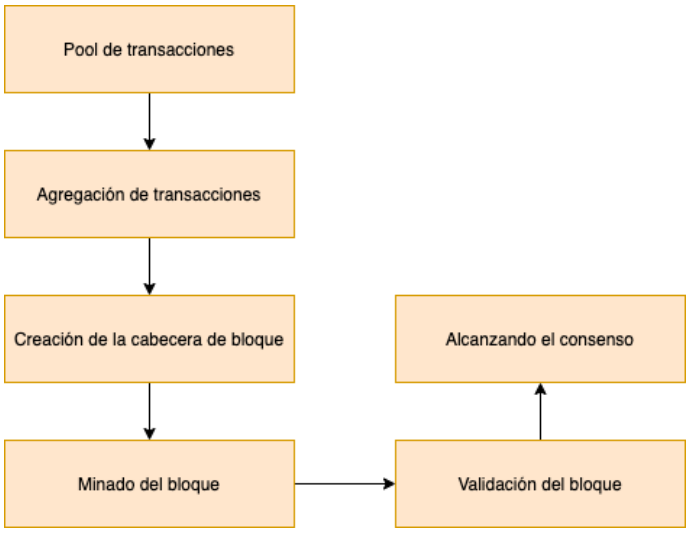


Ilustración 8-1 Diagrama general del proceso de minado

8.1. Pool de transacciones

El *pool* de transacciones o *mempool* es una memoria volátil que se encarga de recoger las transacciones que se emiten en la red. Las transacciones que se incluyen son aquellas se han validado bajo un cierto número de parámetros, y no han recibido ninguna confirmación.

Se dice que una transacción ha sido confirmada cuando se introduce en un bloque, y, otros nodos, construyen sobre ese bloque. Por ejemplo, se dice que una transacción que, ha sido incluida en un bloque sobre el cual se han construido dos bloques más, tiene 3 confirmaciones.

Por tanto, las transacciones que no han recibido confirmaciones aún no han sido incluidas en ningún bloque. Estas transacciones entrarán a formar parte del *mempool* de un nodo, si cumplen una serie determinada de condiciones; algunas de las más relevantes son las siguientes:

- Sintaxis de la transacción correcta.
- Entrada y salidas definidas.
- Tamaño inferior a `MAX_BLOCK_SIZE`.
- *Hash* de la transacción diferente de 0 y -1.
- Tamaño de transacción superior a 100 *bytes*.
- El valor de firmas requeridas para liberar una salida es menor que 15.
- El *script* de desbloqueo solo puede introducir datos a la pila.
- El *script* de bloqueo debe ser uno de acuerdo con los especificados en el apartado 6.4.
- Comprobar si alguna de las entradas referencia a alguna de las salidas de una transacción, ya referenciada en el *pool*.
- Si se trata de una transacción *coinbase*, debe tener, al menos, 100 confirmaciones.
- Cada entrada de la transacción debe referenciar alguna salida conocida en la cadena de bloques.
- Comprobar que el rango de los fondos *Bitcoin* incluidos en las salidas y entradas esté dentro de 0 a 21 millones de BTC.
- Comprobar que el total del valor de las entradas no sea inferior al valor total de las salidas (sin contar la transacción *coinbase*), de lo contrario, se estarían creando *Bitcoins*.
- Si el nodo lo desea, puede aceptar transacciones con una comisión inexistente. O, por el contrario, si la transacción no tiene comisión alguna, podrá rechazarla.

Cabe destacar dos aspectos relevantes:

1) El *mempool* es volátil, por lo cual los datos no persisten si se corta la corriente al sistema del nodo. Por tanto, puede que una transacción que se haya retransmitido no llegue a formar parte nunca del *blockchain*. Por ello, la aplicación de cartera deberá comprobar que las transacciones que hayan retrasmitido, se hayan incluido en la cadena de bloques, y, en caso de que no se hayan incluido, volver a retransmitirlas.

2) Todos los nodos de la red implementan el *mempool,* aunque no estén minando un nuevo bloque. El *mempool* es, por así decirlo, una especie de filtro por el cual se

validan las transacciones, y si éstas son satisfactoriamente validadas, se retransmitirán a los demás nodos con los que esté conectado el nodo que las recibe.

8.2. Agregación de transacciones

Un minero, además de estar a la escucha de transacciones de los demás nodos, también está pendiente de la recepción de nuevos bloques. Si un minero recibe un nuevo bloque por parte de otro minero, significará que ha perdido la carrera minera para esa altura. Cuando un minero recibe y válida un nuevo bloque, la carrera del cálculo de la prueba de trabajo (*proof of work*) del siguiente bloque comienza, reiniciando y desechando el trabajo que hubiese hecho hasta entonces.

Cuando se reinicia el proceso de la carrera minera, el nodo minero introduce transacciones a su bloque candidato. La inclusión de estas transacciones en el bloque, se hace según un parámetro de prioridad, el cual se define por la siguiente ecuación:

$$Prioridad = \frac{Valor\ de\ la\ entrada * Edad\ de\ la\ entrada}{Tamaño\ de\ transacción}$$

Ecuación 8-1

- El valor de la entrada representa el valor en Satoshis.
- La edad de transacción indica la profundidad de la transacción, es decir, el número de bloques que han sido construidos sobre el bloque en el que se introdujo la UTXO.
- El tamaño de transacción indica el tamaño en *bytes* de la transacción. Como se sabe, este valor tiene que ser superior a los 100 *bytes*.

Se considera que una transacción tiene una prioridad alta si tiene un valor superior a 57.600:

$$Prioridad\ Alta = \frac{100.000 * 144}{250} = 57.600$$

Los primeros 50 kB se rellenan con transacciones de prioridad alta. Una vez cubierto este cupo, las transacciones que se incluyen son aquellas que suponen una mayor comisión para el minero. Este proceso de agregación continúa hasta alcanzar el `MAX_BLOCK_SIZE`.

Por tanto, las transacciones no se priorizan por su comisión, sino por su edad, tamaño y valor. Esto permite que las transacciones sin comisión, se puedan procesar. Solamente tendrán que esperar lo suficiente hasta que alcancen la edad necesaria para que sea considerada como una de alta prioridad.

8.3. Creación de la cabecera bloque

Una vez las transacciones ya han sido incluidas, se da paso a la creación de la cabecera del bloque candidato.

Este proceso consiste en rellenar los campos especificados en Ilustración 5-3.

- El campo *version* incluirá la versión del protocolo *Bitcoin* que ejecute el nodo minero.
- El campo *hash de bloque anterior* recogerá el *hash-pointer* del anterior bloque, almacenado en la copia local de la cadena de bloques del nodo.
- La *raíz de merkle* calculará, según los métodos indicados en Ilustración 5-7 y Ilustración 5-8, dependiendo de si el número de transacciones sea par o impar.
- El campo *hora* indicará el instante de tiempo en formato *timestamp* del segundo en el que bloque fue creado. Cabe destacar que este valor puede ser hasta dos horas superior, debido a las imprecisiones que se pueden dar por los usos horarios.
- El *objetivo de la dificultad* marcará la dificultad que ha sido empleada para resolver el *hash-puzzle* o algoritmo de *proof of work*.
- El *nonce* es un número entero utilizado para variar las salidas del algoritmo usado para resolver el reto minero. Se inicializa a 0.

8.4. Minado del bloque

El minado de un bloque consiste en encontrar una entrada para la cual se obtenga, después de ser procesada por la función de *hashing* SHA-256, una salida menor que un *target* especificado.

La gracia de utilizar las funciones de *hash* para este propósito es que, una vez encontrado dicho valor, es inmediato, para otros nodos, comprobar su validez, ya que lo único que se tendría que realizar es volver a computar el valor de salida mediante el valor de entrada ganador. Esta es una gran ventaja si se tiene cuenta que permite, a cualquier nodo de la red, verificar la validez de este parámetro de entrada; simplemente, se tendrá que especificar en la cabecera del bloque candidato.

Es aquí donde entran en juego los campos *nonce* y *dificultad* sobre los cuales se ha pasado de puntillas cuando se especificaban los campos de la cabecera de un bloque.

Debe entenderse que una función de *hash* produce un valor de salida diferente sin más que variar la entrada, por mínima que sea esta modificación (es decir, la modificación de un bit en un valor de entrada, dará un valor de salida totalmente diferente). Con esto en mente se define lo siguiente:

En el proceso de obtención de un *proof of work* válido, un nodo computará tantos valores *hash* diferentes como sean necesarios, incrementando en una unidad el valor del *nonce* [14] especificado en la cabecera, hasta encontrar uno que sea menor que el *target* o dificultad predefinida en el campo *dificultad* de la cabecera del bloque.

El valor de entrada que toma la función SHA-256 para el calculo del valor de salida, es el de la cabecera del bloque concatenándolo con el valor *nonce*:

$$SHA256(cabecera\ del\ bloque\ candidato\ ||\ nonce)$$

Durante el proceso de obtención de un valor que satisfaga la prueba del trabajo, un minero habitualmente incrementa en una unidad el valor del *nonce,* para comprobar si, con esa modificación, se obtiene un *hash* válido.

El *target* o dificultad indica la cota superior del conjunto de *hashes* válidos que satisfacen el *proof of work*. Esta dificultad no es fija y se ajusta en función de la capacidad de cómputo del conjunto de todos los nodos mineros de la red. Exactamente, esta dificultad se ajusta cada 2016 bloques, lo que, a un ritmo medio de 10 minutos por bloque, supone intervalos de tiempo de, aproximadamente, dos semanas (20160 minutos). La dificultad se recalcula mediante el uso de la siguiente expresión:

$$Nueva\ dificultad\ =\ Antigua\ dificultad\ *\ (Tiempo\ total\ de\ minado\ /\ Tiempo\ esperado)$$

Ecuación 8-2

- **Tiempo total de minado:** Tiempo que transcurre desde el minado del primer bloque de los 2016, hasta el último. Este parámetro es fácil de calcular mediante el campo *hora* en la cabecera de bloque.

- **Tiempo esperado:** Como se sabe, la convención *Bitcoin* de los 10 minutos establece que un bloque deba ser minado cada 10 minutos. A 10 minutos por bloque, y un total de 2016 bloques, se obtiene un tiempo esperado de 20160 minutos.

La forma por la cual se representa el parámetro de dificultad es mediante un formato predefinido para el valor del campo dificultad en la cabecera de bloque. Este formato se detalla a continuación.

[14] Nota: Como se verá, esta modificación del nonce en una unidad no es suficiente dada la capacidad de computo del hardware actual dedicado a la resolución del reto minero.

Tómese, por ejemplo, el valor `0x1B0404CB` (*Little endian):*[15]

El primer byte indica que un exponente numérico:

$$Exponente = 0x1B\ (Little\ endian) \text{ à } 27\ decimal$$

Los siguientes 3 bytes indica un coeficiente numérico:

$$Coeficiente = 0x0404CB\ (Little\ endian) \text{à} 13304836\ decimal$$

Una vez obtenidos los valores del coeficiente y exponente se calculan el *target* o dificultad:

$$dificultad = coeficiente * 2^{(8*(exponente-3)}$$

Ecuación 8-3

Sustituyendo para el ejemplo:

$$dificultad = coeficiente * 2^{(8*(exponente-3)}$$

$$dificultad = 13304836 * 2^{(8*(27-3)} = 13304836 * 2^{(8*(27-3))} = 13304836 * 2^{192} = 6277101735386680763835789423207666416102355444464034512896$$

En hexadecimal (*little endian*):

$$Dificultad = 0000000000000008f70fa40ad4983e849043ca29503006b4eb88x0$$

Por tanto, con el valor de dificultad del ejemplo, el minero que encuentre un valor, cuyo *hash* sea menor que el especificado, habrá resuelto el reto minero y podrá acreditarlo mediante la inclusión del valor del *nonce* usado en la cabecera del bloque candidato.

Cabe destacar que la simple variación del parámetro *nonce* es insuficiente dada la capacidad de procesamiento actual de los mineros. Los mineros, con su poder de computo actual, pueden calcular todos los valores del *nonce* posibles sin encontrar una solución menor que el *target,* en mucho menos que 1 segundo. Esto es, debido, otra vez, a la relación del tamaño del subconjunto de valores por debajo del valor de dificultad y el conjunto total del espacio de llegada. Para el algoritmo SHA-256, el espacio de llegada tiene 2^{256} combinaciones posibles, y el subconjunto de valores "ganadores" (valores por debajo del valor de dificultad), es mucho menor que el conjunto total del espacio de llegada, lo cual puede causar que después de agotar

[15] En contraposición al formato *Big endian*, en *Little endian,* los bits siguen un orden de significancia creciente en relación a su posición.

todos los valores del nonce posibles, no se encuentre ningún *proof of work*. La variación del *nonce* puede que no sea suficiente, por el simple hecho de que el número de casos favorables, en comparación con el número total de posibilidades (2^{256}) es ínfimo, y el número de combinaciones que el nonce proporciona es limitado.

Por ejemplo, para la anterior dificultad, el número total de combinaciones posibles es de:

$$Combinaciones\ ganadoras\ totales = 24245342955548033679360000000$$

En cambio, el número total de combinaciones posibles es de:

$$Combinaciones\ totales = 2^{256} = 115792089237316195423570985008687907853269984665640564039457584007913129639936$$

Luego, la probabilidad (casos favorables entre casos totales) de encontrar un valor ganador es:

$$P(Ganador) = 0.00000000000000000000000000000000000000000000000002 = 2 * 10^{-49}$$

Como se sabe, el campo *nonce* en la cabecera de un bloque tiene 4 bytes, lo cual permite el cálculo de un total de 2^{32} combinaciones diferentes de *hash.*

Por lo anterior, se sabe que la probabilidad de que una combinación sea la ganadora es de $2 * 10^{-49}$, por tanto, la probabilidad de que alguna de entre 2^{32} sea correcta, no es más que la probabilidad de que, o bien la primera sea correcta, o la segunda, o la tercera… y así hasta la 256ª.

$$P(Alguna\ de\ entre\ las\ 2^{32}\ combinaciones) = 2 * 10^{-49} * 2^{32} \approx 3{,}8 * 10^{-30}$$

Por tanto, es altamente improbable, dada la dificultad actual, que para las 2^{32} combinaciones, alguna sea menor que el valor de dificultad. Esta probabilidad aumentará en función del número total de combinaciones que se puedan calcular.

En un primer momento, el anterior problema se solucionó incluyendo la posibilidad de modificar el valor del *timestamp* de la creación del nodo candidato. Pero como se ha mencionado anteriormente, este *timestamp* solo puede variar 2 horas hacia delante en tiempo. A medida que la potencia de calculo progresó, esta modificación también se volvió insuficiente. Cuando los mineros alcanzaron la capacidad de computo de 4GHash/s, todos los posibles valores de variación se agotaban en menos de 1 segundo. La solución final fue poder modificar ciertos valores en las transacciones *coinbase*:

Recuerde que la transacción *coinbase* es una transacción en la cual sus salidas no consumen entrada alguna. Esto posibilita la inclusión de ciertos valores en el campo de datos de una transacción *coinbase.* El campo de datos de una transacción es de una longitud variable de entre 2 y 100 bytes. Por tanto, los mineros empezaron a usar este parámetro para realizar modificaciones en el campo de datos, lo cual, después, generaba un *hash* distinto en el campo raíz de *merkle* de la cabecera de bloque.

Recuerde que la raíz de *merkle* se calcula mediante el *merkle-tree* del conjunto de todas las transacciones que el bloque recoge. La transacción *coinbase* es tomada como entrada por el primer nodo hoja del árbol, y se sabe que la más mínima modificación de un bit en algún nodo hoja, producirá una raíz de *merkle* totalmente diferente.

De esta manera se soluciona la limitación, y teniendo en cuenta que el hardware especializado está alcanzando los limites indicados en la *Ley de Moore* se considera una solución a largo plazo.

8.5. Validación del bloque

Una vez se ha obtenido un *proof of work*, el minero está en disposición de incluir el *nonce* correspondiente en la cabecera del bloque. Después, retransmite el nuevo bloque a la red, y los nodos que reciban el nuevo bloque comenzarán a validarlo, y, si la validación se realiza satisfactoriamente, podrán incluirlo en sus copias locales del *blockchain*.

Durante el proceso de validación de un nuevo bloque, se realizan las siguientes acciones:

- Sintaxis del bloque válida.
- El *hash* de la cabecera y *nonce* es menor que el especificado en el campo de dificultad.
- El campo hora, el cual recoge un *timestamp* de la creación del bloque, por parte del minero que ha propuesto el bloque, tiene un valor menor que 2 horas en el futuro.
- Se comprueba que el tamaño total del bloque esté dentro de los límites indicados por `MAX_BLOCK_SIZE`.
- Las transacciones incluidas en el bloque son válidas. Este proceso se realiza de acuerdo con lo descrito en 5.2.3. Recuérdese que, en caso de que el nodo sea un SPV, tendrá que realizar lo indicado en **Error! Reference source not found.**.

Esta validación individual es la que garantiza que no se realicen falsificaciones, introducción de bloques no válidos, transacciones inválidas, etc. Un bloque que no cumpla alguna de las características anteriormente descritas, será descartado, y el nodo deshonesto habrá malgastado sus recursos de computación (los cuales son muy costosos), para nada.

8.6. Alcanzando el consenso

Este es el último paso que un bloque, recién incluido, tendrá que completar para que persista en la cadena de bloques de larga duración.

Obtener un *proof of work* válido puede que no sea suficiente para lograr que el nuevo bloque minado perdure en la rama principal *blockchain.* Pero ¿qué es esto de rama principal?

Existe la posibilidad de que dos bloques sean minados en instantes de tiempo muy próximos entre sí. Esto genera que se creen dos bloques (totalmente válidos) los cuales son incluidos en copias locales de la cadena de bloques de los nodos en la red. Según la proximidad topológica de la red y conforme al modelo de propagación de bloques, esta situación generará que un porcentaje determinado de nodos que considerarán un bloque el nuevo, y otros que consideren el otro. Inmediatamente después de que cada bloque sea introducido en una de las dos versiones de la cadena de bloques, se reiniciará el proceso de minado. En cualquier caso, el bloque en llegar en segundo lugar, para cada una de las dos versiones del *blockchain,* se ignora.

La Ilustración 8-2 muestra la situación anteriormente descrita. Dos bloques C y D, perfectamente válidos, son minados en dos instantes de tiempo similares y cada uno prolonga la cadena principal, creando una bifurcación:

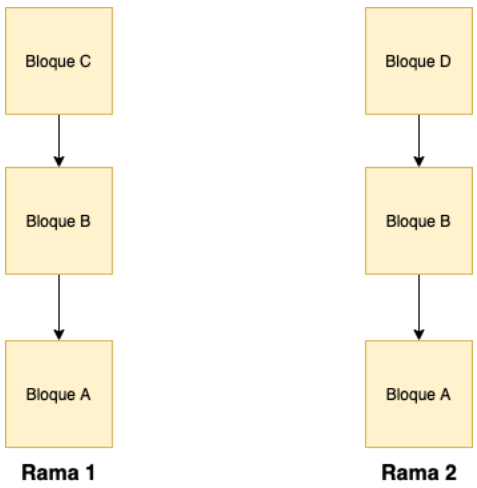


Ilustración 8-2 Bifurcación del blockchain

Esta situación hace que se creen dos ramas totalmente válidas del *blockchain* en la red. Ahora bien, ¿cómo se soluciona esta incoherencia del estado del *blockchain*?

Por norma general, los nodos prolongarán y se construirán sobre la cadena con la mayor dificultad acumulada, la cual se calcula mediante la suma de todos los *nonce* de los bloques en la cadena. Aquella cadena con una dificultad acumulada mayor, será la elegida sobre la que se seguirá construyendo.

Para entender cómo se materializa esta estrategia de resolución del estado de incoherencia del *blockchain* el proceso continuará de la siguiente forma:

Una vez cada bloque se introduce en las dos ramas, los nodos mineros de cada versión reinician el proceso de cálculo del *proof of work* para el siguiente bloque a introducir en cada una de las versiones. Este paso es crucial. Los mineros de cada versión empezarán a minar, hasta que uno de ellos dé con una solución que cumpla el *proof of work*. Cuando el nodo minero retransmita el siguiente bloque, la dificultad acumulada de una rama será superior a la otra, haciendo que la de menor valor sea descartada y considerada como rama secundaria.

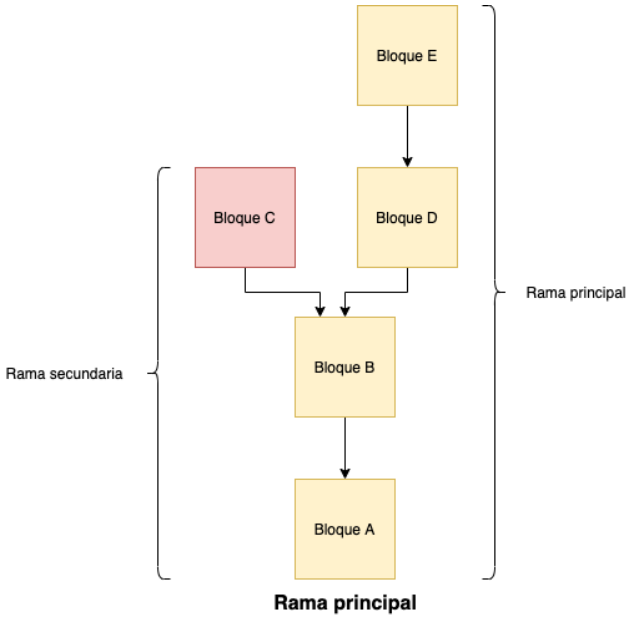


Ilustración 8-3 Resolución de la bifurcación

Se podría dar el caso en el que en ambas ramas se repitiera la situación en la que los dos bloques minasen en tiempos muy próximos dos bloques diferentes. Aunque esta situación es altamente improbable, no supondría más problema que tener que esperar a que el siguiente bloque produjera un desempate.

Esta situación de aparición de ramas secundarias y principales provoca que los nodos afectados por la inclusión de su versión de la rama a una secundaria, pierdan todo el progreso que hubieran hecho hasta entonces. Cuantos más bloques se construyan sobre el bloque, más improbable se volverá que una rama se vuelva secundaria.

Una vez este proceso ha finalizado, el minero habrá conseguido incluir su bloque en la cadena, y se habrá alcanzado el consenso implícito en la red.

9. Instalación de cliente Bitcoin Core

En este apartado se provee de una serie de pasos para instalar el cliente de referencia *Bitcoin Core.* Téngase en cuenta que el procedimiento de instalación se realizará mediante una interfaz de línea de comandos, para lo cual requerirá tener algún sistema operativo capaz de ejecutar ordenes de *Bash*.

La instalación se realizará en un sistema basado en *Unix*.

Descarga del paquete Git

```
$ sudo apt-get install git
```

Clonado del repositorio central Bitcoin

Se recomienda que el PWD sea el directorio de entrada al sistema.

```
$ git clone https://github.com/bitcoin/bitcoin.git
```

Listado de ramas del repositorio Bitcoin

```
$ PWD=~/bitcoin
```

```
$ git tag
```

El comando muestra el *output* siguiente:

```
v0.1.5
v0.1.6test1
v0.10.0
v0.10.0rc1
v0.10.0rc2
v0.10.0rc3
v0.10.0rc4
v0.10.1
v0.10.1rc1
v0.10.1rc2
v0.10.1rc3
v0.10.2
v0.10.2rc1
v0.10.3
v0.10.3rc1
v0.10.3rc2
v0.10.4
v0.10.4rc1
v0.10.5
```

```
v0.11.0
v0.11.0rc1
```

Las versiones con el sufijo `rc` son candidatos de *releases*. Las versiones estables son aquellas sin el sufijo `rc`. Se recomienda seleccionar alguna versión estable

Selección de la rama

Se selecciona la rama que se desee. Para este caso, se elige la ultima versión estable: `v0.11.0`

```
$ git checkout v0.11.0
```

Ejecución del script autogen.sh

El *script* `autogen.sh` crea la configuración para la instalación según el sistema.

```
$ ./autogen.sh
```

Ejecución del script configure.sh

Se ejecuta el *script* `configure.sh` para generar el archivo de compilación `bitcoind`:

```
$ ./configure.sh
```

Después de ejecutar esta orden, el archivo `bitcoind` estará compilado y listo para instalar.

Ejecución de bitcoind

```
$ sudo make install
```

Cambio de contraseña

La contraseña por defecto en el proceso de instalación debe modificarse por razones de seguridad. Se edita el archivo de configuración de *Bitcoin*:

```
$ nano ~/bitcoin/bitcoin.conf
```

Se modifica los campos `rpcuser` y `rpcpassword`. Se recomienda el uso de una contraseña segura.

Ejecución bitcoind

Si desea realizar una descarga completa del *blockchain* ejecute el siguiente comando en segundo plano:

```
bitcoind &
```

10. Conclusiones

Bitcoin es la primera criptomoneda que logra llegar al público general. No por casualidad, sino por los avances tecnológicos que implementa. *Bitcoin* es un artificio tecnológico, únicamente comparable con la *World Wide Web*. Como se decía en la introducción del presente trabajo, las tecnologías que adopta no son excesivamente complejas, pero es la interconexión de todos estos módulos independientes lo que hace que *Bitcoin* sea una tecnología tan innovadora y especial. Personalmente resulta difícil creer que una única persona la diseñara. La forma en la que relaciona ingeniería social con criptografía, economía e informática es una maravilla única e inteligente.

Para un estudiante de ingeniería informática, adentrarse en este mundo solo significa disfrutar de una sutil combinación de, prácticamente, todos los conceptos básicos de esta técnica. A medida que se profundiza en sus tecnologías, todos los módulos encajan los unos con los otros. *Bitcoin* no deja nada al azar; todo tiene un porqué.

Solo el tiempo decidirá si este modelo sustituirá al de las divisas tradicionales. Personalmente creo que, a medida que avance en su ciclo de vida y su uso no derive, simplemente, en un activo especulativo, *Bitcoin* tendrá más posibilidades de convertirse en el nuevo paradigma monetario. Hasta entonces, la confianza de los grandes inversores tradicionales jugará un papel decisivo para tal fin. Queda por ver si su estructura descentralizada, suscitará la suficiente confianza para lograr que la gente se embarque en el proyecto. También queda pendiente ver si la volatilidad de su cotización es simplemente una fase por la que ha de pasar, o, por el contrario, si es simplemente un valor especulativo.

Uno de los grandes inconvenientes que se aprecian en *Bitcoin* es la dificultad de flexibilizar el modelo de implementación inicial. Por ejemplo, ¿serán suficientes los 80 *bytes* del *campo de datos* de una transacción *coinbase*? ¿Qué papel jugará la computación cuántica en el calculo del *proof of work*? ¿Qué pasa si algún algoritmo criptográfico empleado se demuestra inseguro? ¿Cómo le afectarán las acciones regulatorias gubernamentales? Todas estas preguntas son preocupantes; solo el tiempo decidirá. Lo que está claro que es una modificación de un aspecto básico como puede el cambio del algoritmo ECDSA, implementado para el proceso de firma, puede ser catastrófico.

En conclusión, creo que *Bitcoin* es una tecnología alentadora que sienta los precedentes de las técnicas a emplear en una criptomoneda. A mi parecer, esta tecnología solo triunfará si los usuarios entienden su funcionamiento. El escepticismo generado entorno a la moneda es, en gran parte, debido al desconocimiento de sus tecnologías. La sobreinformación y popularidad de *Bitcoin* son necesarias para su difusión y éxito, pero solamente tendrán sentido si los conceptos promocionados son divulgativos e informativos.

Referencias

[1] Antonopoulos, Andreas M. (April 2014). Mastering Bitcoin: Unlocking Digital Crypto-Currencies. O'Reilly Media.

[2] Arvind Narayanan. Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press.

[3] https://en.bitcoin.it/wiki/Vanitygen

[4] Antoon Bosselaers. The hash function RIPEMD-160.

[5] J.I. Farrán. Elliptic Curve Cryptography: On the Algebraic and Geometric Classifications of Projective Varieties with Applications to Coding Theory and Cryptography. University of Valladolid.

[6] J.I. Farrán. PROTOCOLOS Y COMUNICACIONES SEGURAS Criptografía de Clave Pública. University of Valladolid.

[8] Bitcoin Improvement Proposal 16: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki

[9] Bitcoin Improvement Proposal 18: https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki