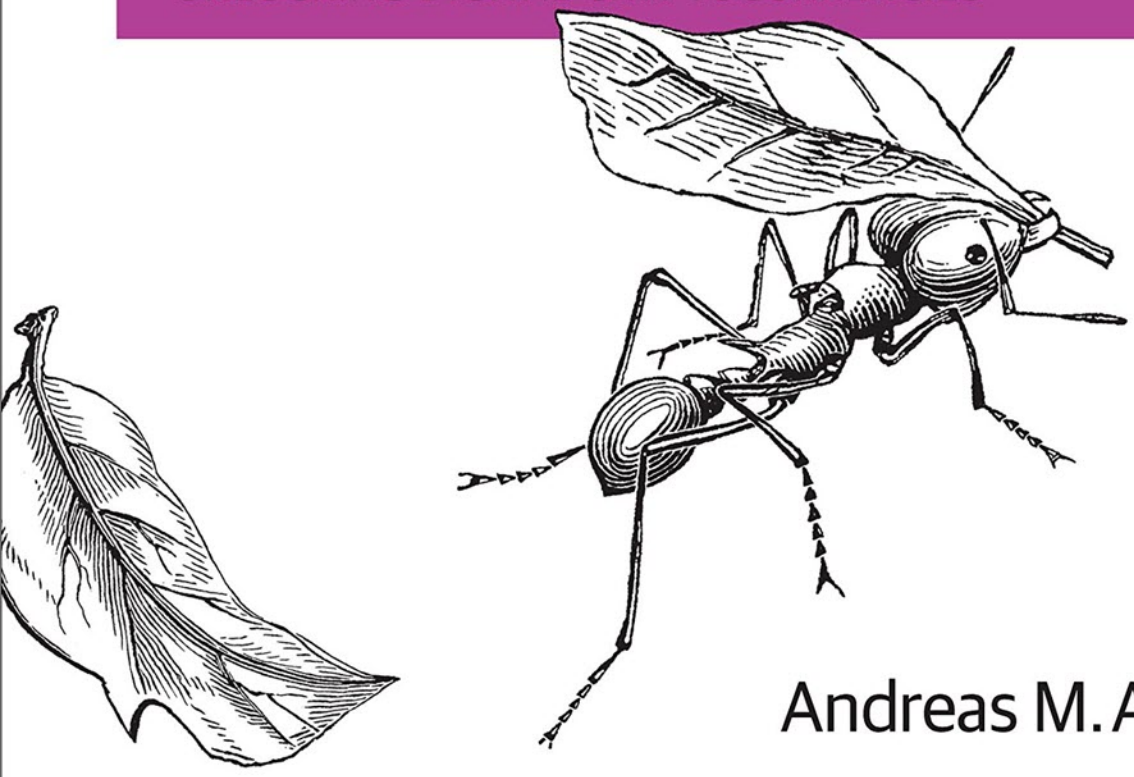
O'REILLY®

Mastering Bitcoin

UNLOCKING DIGITAL CRYPTOCURRENCIES



Andreas M. Antonopoulos

Elogios para 'Mastering Bitcoin'

> "Quando eu falo sobre bitcoin para o público em geral, às vezes me perguntam 'mas como que isso realmente funciona?' Agora eu tenho uma ótima resposta para essa pergunta, porque qualquer um que ler o *Mastering Bitcoin* terá um entendimento aprofundado de como ele funciona e estará bem preparado para desenvolver a nova geração de incríveis aplicativos de criptomoedas."
>
> — Gavin Andresen, Cientista Chefe da Bitcoin Foundation

> "As tecnologias do Bitcoin e da blockchain estão se tornando peças fundamentais na construção da próxima geração da internet. Os melhores e mais brilhantes profissionais do Vale do Silício estão trabalhando nisso. O livro do Andreas irá ajudá-lo à juntar-se à revolução do software no mundo das finanças."
>
> — Naval Ravikant, Co-fundador da AngelList

> "*Mastering Bitcoin* é a melhor referência técnica sobre o bitcoin atualmente disponível. E o bitcoin provavelmente será visto retrospectivamente como a tecnologia mais importante dessa década. Como tal, esse livro é um item indispensável para qualquer desenvolvedor de software, especialmente aqueles interessados em construir aplicativos com o protocolo bitcoin. Altamente recomendado."
>
> — Balaji S. Srinivasan (@balajis), General Partner

> “A invenção da Blockchain do Bitcoin representa uma plataforma completamente nova, que irá possibilitar um ecosistema tão amplo e diverso quanto a própria Internet. Como um dos proeminentes líderes da ideologia, Andreas Antonopoulos é a escolha perfeita para escrever esse livro."
>
> — Roger Ver, Empreendedor e Investidor Bitcoin

More trading books in
https://t.me/tradingpdfgratis

Índice

Prefácio

Escrevendo o Livro do Bitcoin

A primeira vez que ouvi falar em bitcoin foi em meados de 2011. Minha reação imediata foi mais ou menos essa "Pfft! Dinheiro de nerd!" e eu ignorei-o por mais seis meses, sem compreender a sua importância. Esta é uma reação que eu tenho observado com frequência entre muitas das pessoas mais inteligentes que conheço, o que me dá algum consolo. A segunda vez que me deparei com bitcoin, em uma lista de discussão, eu decidi ler o seu "manual de instruções" oficial, o white paper escrito por Satoshi Nakamoto, para ver do que se tratava. Ainda me lembro do momento em que eu terminei de ler aquelas nove páginas, quando eu percebi que bitcoin não era simplesmente uma moeda digital, mas uma rede de confiança que também poderia servir de base para aplicações muito mais avançadas do que apenas moedas. Após constatar que o bitcoin não é dinheiro, mas sim uma rede de confiança descentralizada, comecei uma viagem de quatro meses para devorar cada pedaço de informação que eu poderia encontrar sobre o assunto. Eu me tornei obcecado e encantado, gastando 12 ou mais horas por dia colado ao monitor, lendo, escrevendo, codificando e aprendendo o máximo que pude. Após pular muitas refeições, saí desse período de obsessão 9 quilos mais magro e determinado a dedicar-me a trabalhar com bitcoin.

Dois anos depois, após criar várias pequenas startups para explorar serviços e produtos relacionados ao bitcoin, eu decidi que estava na hora de escrever meu primeiro livro. O Bitcoin foi um tópico que me levou a um frenesi de criatividade e consumiu meus pensamentos; Foi a tecnologia mais empolgante que eu encontrei desde que conheci a Internet. Estava na hora de compartilhar minha paixão sobre essa incrível tecnologia com uma audiência mais ampla.

Público Alvo

Esse livro foi escrito principalmente para programadores. Se você sabe alguma linguagem de programação, esse livro irá ensiná-lo como as moedas criptográficas funcionam, como utilizá-las e como desenvolver softwares que trabalhem com elas. Os primeiros capítulos também são adequados como uma introdução aprofundada ao bitcoin para não-programadores, que queiram entender o funcionamento interno do bitcoin e das criptomoedas.

Convenções Usadas Neste Livro

As seguintes convenções tipográficas são usadas neste livro:

Itálico

Indica novos termos, URLs, endereços de e-mail, nomes e extensões de arquivos.

Largura constante

Usada para listagem de programas, assim como dentro de parágrafos para se referir a elementos de programas como variáveis e nomes de funções, banco de dados, tipos de dados, variáveis de

ambiente, declarações e palavras-chave.

`Largura constante em negrito`
: Mostra comandos ou outro texto que deveria ser digitado literalmente pelo usuário.

Largura constante em itálico
: Mostra um texto que deveria ser substituído por valores fornecidos pelo usuário, ou valores determinados pelo contexto.

TIP	Esse ícone é usado em dicas, sugestões ou notas em geral.

WARNING	Esse ícone indica uma mensagem de aviso ou cuidado.

Exemplos de Códigos

Os exemplos são ilustrados em Python, C++ e usando uma linha de comando de um sistema operacional do tipo Unix, como Linux ou Mac OS X. Todos os snippets de códigos estão disponíveis no GitHub repository repositório GitHub no subdiretório *code* do repositório principal. Você pode fazer um fork do código do livro, testar exemplos de códigos ou enviar correções via GitHub.

Todos os snippets de códigos podem ser replicados na maioria dos sistemas operacionais com uma instalação mínima dos compiladores e interpretadores das linguagens correspondentes. Quando necessário, nós providenciaremos as instruções básicas da instalação e exemplos passo-a-passo do resultado dessas instruções.

Alguns dos snippets de códigos e do output do código foram reformatados para a impressão. Em todos esses casos, as linhas foram divididas por uma barra invertida (\), seguida por um caractere de nova linha. Ao transcrever os exemplos, remova estes dois caracteres e una as linhas novamente, obtendo resultados idênticos aos mostrados no exemplo.

Todos os snippets de códigos, sempre que possível, usam valores e cálculos reais, de maneira que você possa construir de exemplo a exemplo e observar os mesmo resultados em qualquer código que você escrever para calcular os mesmos valores. Por exemplo, as chaves privadas e seus endereços e chaves públicas correspondentes são reais. As amostras de transação, blocos e referências à blockchain foram realmente introduzidas na blockchain do bitcoin e fazem parte do ledger público, para que você possa revisá-las em qualquer sistema bitcoin.

Agradecimentos

Este livro representa o esforço e contribuições de muitas pessoas. Agradeço por toda ajuda que recebi de amigos, colegas e até desconhecidos, que se juntaram a mim nessa tarefa de escrever o livro técnico definitivo sobre criptomoedas e bitcoin.

É impossível fazer uma distinção entre a tecnologia Bitcoin e a comunidade bitcoin — e este livro é um produto tanto dessa comunidade quanto é sobre a tecnologia. Meu trabalho nesse livro foi encorajado,

comemorado. apoiado e recompensado por toda a comunidade bitcoin desde o seu início até o fim. Mais do que tudo, esse livro me permitiu ser uma parte de uma comunidade maravilhosa por dois anos e não posso agradecer suficientemente por eu ter sido aceito por essa comunidade. Há um número imenso de pessoas para ser mencionadas pelo nome — pessoas que encontrei em conferências, eventos, seminários, meetups, encontros de pizza e pequenas reuniões, assim como tantos que se comunicam comigo via Twitter, Reddit, bitcointalk.org e pelo GitHub e que impactaram esse livro de alguma forma. Cada ideia, analogia, pergunta, resposta e explicação que você encontrar nesse livro foi de algum modo inspirada, testada ou melhorada através da interação com a comunidade. Muito obrigado a todos pelo apoio; esse livro não teria acontecido sem vocês. Serei eternamente grato.

A jornada para se tornar um autor começa, é claro, muito antes do primeiro livro. Minha língua nativa (e também na escola) era o grego, e por isso tive que fazer um curso emergencial de inglês escrito ainda no meu primeiro ano de universidade. Sou muito grato a Diana Kordas, minha professora de inglês escrito, que muito me ajudou a construir a confiança e as habilidades que precisei naquele ano. Mais para frente, já como profissional, desenvolvi minhas habilidades em escrita técnica sobre data centers, escrevendo para a revista *Network World*. Meu agradecimentos a John Dix e John Gallant, que me deram meu primeiro trabalho como colunista na *Network World*, ao meu editor Michael Cooney e meu colega Johna Till Johnson, que editaram minhas colunas e as fizeram publicáveis. Escrever 500 palavras por semana durante quatro anos me deu experiência suficiente para eventualmente considerar a me tornar um autor. Obrigado à Jean de Vera por ter me encorajado a tornar-me um autor e por sempre acreditar e insistir que eu tinha um livro dentro de mim.

Obrigado também àqueles que me apoiaram quando enviei à O'Reilly minha proposta de livro, ao enviarem referências e revisarem o esboço. Especificamente, obrigado a John Gallant, Gregory Ness, Richard Stiennon, Joel Snyder, Adam B. Levine, Sandra Gittlen, John Dix, Johna Till Johnson, Roger Ver, and Jon Matonis. Um obrigado especial ao Richard Kagan e Tymon Mattozko, que revisaram as primeiras versões da proposta e a Matthew Owain Taylor, que fez a editoração da proposta.

Obrigado ao Cricket Liu, autor do título *DNS and BIND*, que me apresentou à O'Reilly. Outro obrigado para Michael Loukides e Allyson Macdonald da O'Reilly, que trabalharam por meses, ajudando na confecção desse livro. A Allyson foi especialmente paciente quando os prazos eram perdidos e as entregas atrasavam quando a vida fazia sua intervenção em nossa agenda planejada.

Os primeiros rascunhos dos primeiros capítulos foram os mais difíceis, pois o bitcoin é um assunto difícil de ser desvendado. Cada vez que eu puxava um fio sobre a tecnologia bitcoin, eu tinha que puxar o novelo inteiro. Eu fiquei travado repetidas vezes, assim como um pouco desanimado - enquanto lutava para fazer um tópico de fácil entendimento e criar uma narrativa ao redor de um assunto tão denso tecnicamente. Eventualmente, decidi contar a estória do bitcoin através de estórias de pessoas que usavam a criptomoeda e todo o livro ficou fácil de ser escrito. Devo meus agradecimentos ao meu amigo e mentor, Richard Kagan, que me ajudou a desvendar a estória e a superar os momentos de "bloqueio de escritor" e a Pamela Morgan, que revisou os primeiros rascunhos de cada capítulo e fez as perguntas difíceis, com o propósito de torná-los melhores. Obrigado também aos desenvolvedores do grupo San Francisco Bitcoin Developers Meetup e a Tariq Lewis, co-fundador do grupo, por me ajudar a testar o material inicial.

Durante o desenvolvimento do livro, eu disponibilizei os primeiros rascunhos via GitHub e convidei o público para comentar. Mais de uma centena de comentários, sugestões, correções e contribuições me foram enviadas em resposta. Tais contribuições foram reconhecidas e agradecidas publicamente em Lançamento do Rascunho Inicial (Contribuições no GitHub). Obrigado especial para Minh T. Ngyuen, que se voluntariou para gerenciar as contribuições no GitHub e muitas outras que ele próprio adicionou. Obrigado também ao Andrew Naugler pelo desenho do infográfico.

Uma vez que o livro foi rascunhado, ele passou por diversas rodadas de revisão técnica. Obrigado ao Cricket Liu e Lorne Lantz pelas extensas revisões, comentários e apoio.

Vários desenvolvedores de bitcoin contribuíram com exemplos de códigos, revisões, comentários e encorajamento. Obrigado a Amir Taaki e Eric Voskuil pelos exemplos de snippets de código e muitos comentários de valor; a Vitalik Buterin e Richard Kiss pela ajuda com a matemática da curva elíptica e contribuições com o código; Gavin Andresen pelas correções, comentários e encorajamento; Michalis Kargakis pelos comentários, contribuições e escrita btcd; e a Robin Inge pelos envios de erratas, melhorando a segunda impressão do livro.

Eu devo o meu amor pelas palavras e livros à minha mãe, Theresa, que me criou em uma casa com livros enfileirados em cada parede. Minha mãe também me deu meu primeiro computador em 1982, mesmo ela sendo uma tecnófoba assumida. Meu pai, Menelaos, um engenheiro civil que recém publicou seu primeiro livro aos 80 anos de idade, foi quem me ensinou o pensamento lógico e analítico, bem como o amor pela ciência e engenharia.

Obrigado a todos por me apoiarem durante toda esta jornada.

Lançamento do Rascunho Inicial (Contribuições no GitHub)

Muitos contribuidores enviaram comentários, correções e adições para a versão inicial no GitHub. Muito obrigado a todos por suas contribuições para esse livro. Abaixo, uma lista de contribuidores notáveis no GitHub, incluindo seus IDs em parênteses:

- Minh T. Nguyen, editor de contribuição no GitHub (enderminh)
- Ed Eykholt (edeykholt)
- Michalis Kargakis (kargakis)
- Erik Wahlström (erikwam)
- Richard Kiss (richardkiss)
- Eric Winchell (winchell)
- Sergej Kotliar (ziggamon)
- Nagaraj Hubli (nagarajhubli)
- ethers
- Alex Waters (alexwaters)
- Mihail Russu (MihailRussu)

- Ish Ot Jr. (ishotjr)
- James Addison (jayaddison)
- Nekomata (nekomata-3)
- Simon de la Rouviere (simondlr)
- Chapman Shoop (belovachap)
- Holger Schinzel (schinzelh)
- effectsToCause (vericoin)
- Stephan Oeste (Emzy)
- Joe Bauers (joebauers)
- Jason Bisterfeldt (jbisterfeldt)
- Ed Leafe (EdLeafe)

More trading books in
https://t.me/tradingpdfgratis

Edição Aberta

Essa é a edição aberta do "Mastering Bitcoin", publicado para traduções sob a licença Creative Commons Atribuição-CompartilhaIgual (CC-BY-SA). Essa licença permite que você leia, compartilhe, copie, imprima, venda ou reutilize esse livro ou partes dele, desde que você:

- Utilize a mesma licença (Compartilha-Igual)
- Inclua atribuição

Atribuição

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

Copyright 2016, Andreas M. Antonopoulos LLC

Tradução

Se você estiver lendo esse livro em um idioma que não seja o inglês, ele foi traduzido por voluntários. As seguintes pessoas contribuíram para essa tradução:

- André Torres (@Criptonauta) - Coordenação e tradução / Rodrigo Castilhos - Revisão e tradução
- Fernando Bitti Loureiro
- Fernando Paladini, Anderson Juhasc, Paulo Gomes

Glossário Rápido

Este glossário rápido contém muitos dos termos relacionados ao bitcoin que serão usadas durante todo o livro. Recomendamos que favorite essa seção para ter uma referência rápida, caso necessário.

endereço

Um endereço bitcoin se parece com 1DSrfJdB2AnWaFNgSbv3MZC2m74996JafV. Ele consiste de uma sequência de letras e números começando com um "1" (número um). Assim como você pede para alguém enviar um email para seu endereço de email, você pediria a outras pessoas para enviarem bitcoin para seu endereço bitcoin.

bip

Bitcoin Improvement Proposals (Propostas de Melhoria Bitcoin). Um conjunto de propostas que membros da comunidade bitcoin têm feito para melhorar o bitcoin. Por exemplo, BIP0021 é uma proposta para melhorar a estrutura do Identificador Uniforme de Recursos (URI) do bitcoin.

bitcoin

O nome da unidade monetária (a moeda), a rede e o software.

bloco

Um agrupamento de transações, marcadas com um registro de tempo e uma impressão digital do bloco anterior. O cabeçalho do bloco é codificado para produzir uma prova de trabalho, assim validando as transações. Blocos válidos são adicionados à blockchain através do consenso da rede.

blockchain

Uma lista de blocos validados, cada um ligado ao seu predecessor até chegar ao bloco gênesis.

confirmações

Uma vez que uma transação é incluída em um bloco, ela recebe uma confirmação. Assim que *outro* bloco é minerado na mesma blockchain, a transação tem duas confirmações, e assim continua. Seis ou mais confirmações são consideradas como prova suficiente de que a transação não pode ser desfeita.

dificuldade

Um parâmetro que afeta toda a rede e controla o quanto de esforço computacional é necessário para produzir uma prova de trabalho.

meta de dificuldade

Uma dificuldade na qual toda a computação na rede encontrará blocos aproximadamente a cada 10 minutos.

calibragem da meta de dificuldade

Um recálculo da meta de dificuldade que afeta toda a rede e ocorre a cada 2.106 blocos, levando em consideração o poder de hashing dos 2.106 blocos anteriores.

taxas

O emissor de uma transação frequentemente inclui uma taxa para a rede pelo processamento de uma determinada transação. A maioria das transações requer uma taxa mínima de 0.5 mBTC.

hash

Uma impressão digital de alguma entrada binária.

bloco gênesis

O primeiro bloco na blockchain, usado para iniciar a criptomoeda.

minerador

Um nó da rede que encontra uma prova de trabalho válida para novos blocos, através do uso repetido de hash.

rede

Uma rede ponto-a-ponto (peer-to-peer) que propaga transações e blocos para cada nó de bitcoin na rede.

Prova-de-Trabalho

Em inglês, Proof-of-Work (PoW). Uma parte de um dado que requer um esforço computacional considerável para ser encontrada. No bitcoin, mineradores devem encontrar uma solução numérica para o algoritmo SHA-256 que esteja em conformidade com a meta da rede, a meta de dificuldade.

recompensa

Uma quantidade de moedas incluída pela rede, em cada novo bloco, como recompensa ao minerador que encontrou a solução da Prova-de-Trabalho. Atualmente, a recompensa é de 25BTC por bloco.

chave secreta (chave privada)

O número secreto que destrava os bitcoins enviados para um determinado endereço. Uma chave secreta se parece com 5J76sF8L5jTtzE96r66Sf8cka9y44wdpJjMwCxR3tzLh3ibVPxh.

transação

Em termos simples, uma transferência de bitcoins de um endereço para outro. Mais precisamente, uma transação é uma estrutura de dados assinada que expressa uma transferência de valor. Transações são transmitidas pela rede bitcoin, coletadas por mineradores, incluídas nos blocos e tornadas permanentes na blockchain.

carteira

Um software que contém todos seus endereços bitcoins e suas chaves secretas. Usada para enviar, receber e armazenar seus bitcoins.

Introdução

O que é Bitcoin?

Bitcoin é um conjunto de conceitos e tecnologias que formam a base de um ecossistema de dinheiro digital. As unidades de moeda chamadas bitcoins são usadas para armazenar e transmitir valor entre os participantes na rede Bitcoin. Os usuários Bitcoin comunicam-se entre si utilizando o protocolo bitcoin principalmente através da Internet, mas outras formas de rede também podem ser usadas. A implementação da pilha do protocolo bitcoin, está disponível como software de código aberto, pode ser executada em uma ampla variedade de dispositivos de computação, incluindo laptops e smartphones, o que torna a tecnologia de fácil acesso.

Os usuários podem transferir bitcoins através da rede para fazer as mesmas coisas que as moedas convencionais podem fazer, incluindo compra e venda de bens, envio de dinheiro a pessoas e organizações ou mesmo a extensão de crédito. Os bitcoins podem ser comprados, vendidos ou trocados por outras moedas em casas de câmbio especializadas - as populares exchanges. De certo modo, o Bitcoin é o dinheiro perfeito para a Internet, pois é rápido, seguro e sem fronteiras.

Ao contrário das moedas tradicionais, os bitcoins são inteiramente virtuais. Não há moedas físicas ou mesmo moedas digitais por si só. As moedas de bitcoin se subentendem como transações que transferem valor de um remetente a um destinatário. Os usuários de bitcoin possuem chaves que lhes permitem provar a posse de transações na rede bitcoin, desbloqueando o valor (em bitcoins) a ser gasto e o transferindo para um novo destinatário. Essas chaves geralmente são armazenadas em uma carteira digital no computador ou smartphone de cada usuário. A posse da chave que desbloqueia uma transação é o único pré-requisito para gastar os bitcoins, pondo o controle inteiramente nas mãos de cada usuário.

Bitcoin é um sistema distribuído ponto-a-ponto (peer-to-peer ou P2P). Como tal, não existe um servidor "central" ou ponto de controle. Os bitcoins são criados (gerados) através de um processo chamado de "mineração", que consiste em competir para encontrar soluções para um problema matemático enquanto se processam transações de bitcoins. Qualquer participante na rede bitcoin (ou seja, qualquer usando um dispositivo que execute a implementação completa de protocolo Bitcoin) pode ser um minerador, bastando utilizar o poder de processamento de seu computador para verificar e registrar transações. Em média, a cada 10 minutos alguém é capaz de validar as transações dos últimos 10 minutos, sendo recompensado com bitcoins novinhos em folha. Essencialmente, a mineração de bitcoins descentraliza as funções de emissão de moeda e de compensação tipicamente atribuídas a um banco central, dessa forma substituindo a necessidade de qualquer banco central.

O protocolo bitcoin contêm algoritmos que regulam a função de mineração através da rede. A dificuldade da tarefa de processamento que os mineradores devem realizar — registrar com sucesso um bloco de transações na rede bitcoin — ajusta-se dinamicamente de tal forma que, em média, alguém é bem-sucedido a cada 10 minutos, independentemente de quantos mineradores (e CPUs) estejam trabalhando na tarefa a qualquer momento. O protocolo também reduz à metade, a cada 4 anos, a taxa com que novos bitcoins são criados, limitando, assim, o número total de bitcoins que serão

criados a um máximo de 21 milhões de moedas. O resultado é que o número de bitcoins em circulação segue uma curva previsível que alcançará 21 milhões no ano de 2140. Devido à taxa decrescente de emissão, em longo prazo a moeda bitcoin é deflacionária. Além disso, bitcoin não pode ser inflacionado "imprimindo" novo dinheiro além da taxa de emissão já esperada.

Nos bastidores, bitcoin é também o nome do protocolo, de uma rede e de uma inovação digital distribuída. A moeda bitcoin é, na verdade, apenas a primeira aplicação desta invenção. Como um programador, eu vejo o bitcoin parecido com a Internet do dinheiro, uma rede para propagar valor e proteger a posse de ativos digitais através da computação distribuída. Há muito mais no bitcoin do que se enxerga à primeira vista.

Neste capítulo inicial ensinaremos alguns dos principais conceitos e termos, além de como baixar o software necessário e como usar o bitcoin para transações simples. Nos capítulos seguintes iremos desembrulhar as camadas de tecnologia que tornam possível o bitcoin e examinar o funcionamento interno da rede e do protocolo bitcoin.

Moedas Digitais Antes do Bitcoin

O surgimento de uma moeda digital viável está intimamente relacionado à evolução da criptografia. Isso não é algo surpreendente quando se leva em consideração os desafios fundamentais envolvidos no uso de bits para a representação de um valor que pode ser trocado por bens e serviços. Duas perguntas básicas feitas por qualquer um que aceite dinheiro digital são:

1. Posso confiar que o dinheiro é autêntico e não falsificado?
2. Posso estar seguro de que ninguém vai reclamar que esse dinheiro lhe pertence e não a mim? (também conhecido como o problema do "gasto duplicado")

Os emissores do dinheiro em papel estão o tempo todo enfrentando o problema da falsificação através do uso de papéis e tecnologias de impressão cada vez mais sofisticados. O dinheiro físico resolve facilmente o problema do gasto duplicado, pois a mesma nota em papel não pode estar em dois lugares ao mesmo tempo. É claro que o dinheiro convencional é frequentemente armazenado e transmitido de forma digital. Nestes casos, os problemas de falsificação e de gastos duplicados são tratados pela compensação de todas as transações eletrônicas através de autoridades centrais que detêm uma visão global da moeda em circulação. No caso do dinheiro digital, que não pode se beneficiar de tintas especiais ou marcas holográficas, a criptografia proporciona a base para confiar na legitimidade de um valor que um usuário afirma possuir. Especificamente, as assinaturas digitais criptográficas permitem a um usuário assinar um ativo digital ou transação provando a posse do ativo. Com a arquitetura apropriada, as assinaturas digitais também podem ser usadas para resolver o problema do gasto duplicado.

No final dos anos 1980, quando a criptografia começou a se tornar mais acessível e entendida, muitos pesquisadores começaram a tentar usá-la para construir moedas digitais. Estes projetos pioneiros emitiam dinheiro digital, normalmente lastreados por uma moeda nacional ou um metal precioso - como o ouro.

Apesar destas moedas digitais pioneiras funcionarem, elas eram centralizadas e, como resultado, eram fáceis de ser atacadas tanto por governos como por hackers. As primeiras moedas digitais usavam uma central de compensação para finalizar todas as transações em intervalos regulares, da mesma forma que um sistema bancário tradicional. Infelizmente, em muitos casos essas moedas digitais que surgiam se tornavam um alvo dos governos preocupados e eventualmente desapareciam. Algumas falharam em quebras espetaculares quando a companhia responsável era liquidada de repente. Para ser robusta contra a intervenção de opositores, fossem governos legítimos ou elementos criminosos, uma moeda descentralizada digital se tornava necessária para evitar um único ponto de ataque. Este sistema é o Bitcoin, projetado para ser completamente descentralizado e livre de qualquer autoridade central ou ponto de controle que possa ser atacado ou corrompido.

O Bitcoin representa o auge de décadas de pesquisa em criptografia e sistemas distribuídos e inclui quatro inovações chaves reunidas em uma combinação única e poderosa. O Bitcoin consiste em:

- Uma rede peer-to-peer descentralizada (o protocolo bitcoin)
- Um registro público de transações (a blockchain ou cadeia de blocos)
- Uma emissão de moeda descentralizada, matemática e determinística (a mineração distribuída)
- Um sistema descentralizado de verificação de transações (o script de transação)

A História do Bitcoin

O Bitcoin foi inventado em 2008 com a publicação de um documento intitulado "Bitcoin: Um Sistema de Dinheiro Eletrônico Ponto-a-Ponto" ("Bitcoin: A Peer-to-Peer Electronic Cash System" em inglês), escrito por um autor sob o pseudônimo de Satoshi Nakamoto. Nakamoto combinou várias das invenções anteriores tais como b-money e HashCash para criar um sistema de dinheiro eletrônico completamente descentralizado que não dependesse de uma autoridade central para a emissão de moeda ou para a liquidação e validação de transações. A principal inovação foi usar um sistema de computação distribuído (chamado algoritmo de "prova de trabalho" ou "proof of work") para conduzir uma "eleição" global a cada 10 minutos, permitindo à rede descentralizada chegar em um *consenso* sobre o estado das transações. Isto resolve de forma elegante o problema de gasto duplicado, onde uma única unidade de moeda poderia ser gasta duas vezes. Antes do Bitcoin, o problema de gasto duplicado era uma fraqueza do dinheiro digital, e sua solução envolvia a transmissão e verificação de todas as transações através de uma entidade central.

A rede bitcoin surgiu em 2009, baseada em uma implementação de referência publicada por Nakamoto e desde então revisada por muitos outros programadores. A computação distribuída que proporciona segurança e robustez ao bitcoin cresceu exponencialmente, e agora excede a capacidade combinada de processamento dos principais supercomputadores do mundo. Em 2014, o valor de mercado do bitcoin era estimado entre 5 e 10 bilhões de dólares americanos, dependendo da taxa de câmbio entre o bitcoin e o dólar. A maior transação processada até 2014 pela rede foi de US$ 150

milhões, transmitida instantaneamente e processada sem nenhuma taxa.

Satoshi Nakamoto afastou-se do público em abril de 2011, deixando a responsabilidade pelo desenvolvimento do código e da rede nas mão de um animado grupo de voluntários. A identidade da pessoa ou pessoas por trás do bitcoin ainda é desconhecida. No entanto, nem Satoshi Nakamoto nem qualquer outra pessoa exerce controle sobre o sistema bitcoin, que opera baseado em princípios matemáticos totalmente transparentes. A invenção em si é revolucionária e já criou um novo campo de estudos nas áreas da computação distribuída, economia e econometria.

Uma Solução para um Problema de Computação Distribuída

O invento de Satoshi Nakamoto é também uma solução prática para um problema que até então não estava resolvido na computação distribuída, conhecido como o "Problema dos Generais Bizantinos". Em resumo, o problema consiste em tentar tomar uma decisão através do intercâmbio de informações sobre uma rede pouco confiável e potencialmente comprometida. A solução de Satoshi Nakamoto, que utiliza o conceito de prova de trabalho (proof-of-work) para alcançar o consenso sem uma autoridade central confiável, representa um enorme avanço na ciência de computação distribuída e possui amplas aplicações além da ser um meio de pagamento. Tal solução pode ser usada para alcançar consenso em redes descentralizadas para provar a honestidade de eleições, loterias, registros de bens, notarização digital e mais.

Usos do Bitcoin, Seus Usuários e Suas Histórias

Bitcoin é uma tecnologia usada para representar dinheiro, que é fundamentalmente uma linguagem para a troca de valor entre pessoas. Vamos conhecer as histórias de pessoas que estão usando bitcoin e alguns dos usos mais comuns da moeda e do protocolo. Iremos reutilizar essas histórias ao longo do livro para ilustrar os usos do dinheiro digital na vida real e como eles se tornaram possíveis por meio das várias tecnologias que são partes do bitcoin.

Varejo de baixo valor nos Estados Unidos

A Alice mora na área norte da baía da Califórnia. Ela ouviu falar sobre o bitcoin através dos seus amigos e quer começar a usá-lo. Iremos acompanhar a história de como ela aprende a respeito do bitcoin, adquire algumas moedas e então gasta alguns de seus bitcoins para comprar uma xícara de café no Bob’s Café em Palo Alto. Esta história irá nos apresentar ao software, às casas de câmbio e transações básicas desde a perspectiva de um consumidor do varejo.

Varejo de produtos de alto valor nos Estados Unidos

A Carol é dona de uma galeria de arte em San Francisco. Ela vende pinturas caras por bitcoin. Esta história nos vai apresentar os riscos de um ataque de consenso "51%" para varejistas de produtos de alto valor.

Serviços de contratos internacionais

O Bob, o dono da cafeteria de Palo Alto, está montando um novo website. Ele contratou um

programador web indiano, o Gopesh, que mora em Bangalore, Índia. O Gopesh aceitou ser pago em bitcoin. Esta história vai examinar o uso do bitcoin para a terceirização, contratos de serviços e transferências bancárias internacionais.

Doações beneficentes

A Eugênia é a diretora de uma instituição de caridade para crianças nas Filipinas. Recentemente ela descobriu o bitcoin e quer usá-lo para alcançar um grupo completamente diferente de doadores locais e estrangeiros para financiar sua instituição de caridade. Ela também tem investigado formas de usar o bitcoin para rapidamente distribuir os fundos nas áreas necessitadas. Esta história irá mostrar o uso do bitcoin para a angariação de fundos através de fronteiras e moedas e o uso de um registro contábil aberto para a transparência de organizações de caridade.

Importação e exportação

O Mohammed é um importador de eletrônicos em Dubai. Ele vem tentando usar o bitcoin para comprar eletrônicos dos Estados Unidos e da China para importação aos Emirados Árabes Unidos e assim acelerar o processo de pagamentos para as importações. Esta história irá mostrar como o bitcoin pode ser usado para grandes pagamentos internacionais B2B entre negócios de grande porte atados a mercadorias físicas.

Minerando bitcoins

O Jing é um estudante de engenharia de computação em Shanghai. Ele possui uma aparelhagem de "mineração" para minerar bitcoins, usando suas habilidades de engenharia para complementar sua renda. Esta história irá examinar a base "industrial" do bitcoin: o equipamento especializado usado para proteger a rede bitcoin e emitir nova moeda.

Cada uma dessas histórias se baseia em pessoas reais e indústrias reais que atualmente usam bitcoin para criar novos mercados, novas indústrias e soluções inovadoras para os problemas econômicos globais.

Como Começar

Para participar da rede bitcoin e começar a usar a moeda, tudo que um usuário precisa fazer é baixar um programa ou usar um aplicativo web. Como o bitcoin é um padrão, há muitas implementações do software de cliente bitcoin. Também há uma implementação de referência, conhecida como o cliente Satoshi, que é gerenciado como um projeto de código aberto por uma equipe de desenvolvedores e provém da implementação original escrita por Satoshi Nakamoto.

Os três principais tipos de clientes bitcoin são:

Cliente completo

Um cliente completo, ou "nó completo", armazena todo o histórico de transações de bitcoins (cada uma das transações de todos os usuários, desde o começo), gerencia as carteiras dos usuários e pode iniciar transações diretamente na rede bitcoin. Isto é similar a um servidor de email independente, no sentido de que ele trata de todos os aspectos do protocolo sem depender de quaisquer outros servidores ou serviços de terceiros.

Cliente compacto

Um cliente compacto armazena a carteira do usuário, mas depende de servidores mantidos por terceiros para ter acesso às transações e à rede Bitcoin. O cliente compacto não guarda uma cópia completa de todas as transações e portanto precisa confiar nos servidores de terceiros para validar transações. É similar a um cliente de email autônomo que se conecta a um servidor de email para acessar uma caixa de emails, no sentido de que depende de um terceiro para interagir com a rede.

Cliente web

Os clientes web são utilizados através de um navegador web e armazenam a carteira do usuário em um servidor mantido por um terceiro. Isso é similar ao webmail no sentido em que eles dependem completamente de um servidor de terceiros.

Clientes móveis

("smartphones, clientes para bitcoin"Os clientes móveis para smartphones, tais como aqueles baseados no sistema Android, podem operar tanto como clientes completos, quanto como compactos ou web. Alguns clientes móveis se sincronizam com um cliente web ou de PC, proporcionando assim uma carteira multiplataforma entre múltiplos dispositivos, mas com uma fonte comum de fundos.

A escolha do cliente bitcoin depende de quanto controle o usuário quer sobre os fundos. Um cliente completo irá oferecer o máximo nível de controle e independência do usuário, mas, em compensação, deixa a responsabilidade pelos backups e pela segurança nas mãos do usuário. No outro extremo de opções, um cliente web é o mais fácil de configurar de de usar, mas, em compensação, introduz um risco adicional, já que a segurança e o controle são compartilhados com o usuário e o dono do serviço web. Se um serviço de carteira web é comprometido, como muitos já foram, os usuários podem perder todos os seus fundos. Por outro lado, se os usuários tiverem um cliente completo sem os backups adequados, eles podem perder todos os seus fundos por causa de um contratempo do computador.

Para os propósitos deste livro, demonstraremos o uso de uma variedade de clientes bitcoin que podem ser baixados, desde a implementação de referência (o cliente Satoshi) até as carteiras web. Alguns dos exemplos vão necessitar o uso do cliente de referência, que além de ser um cliente completo também expõe APIs de acesso à carteira, rede e serviços de transações. Se você planeja explorar as interfaces programáticas de acesso no sistema bitcoin, você irá precisar do cliente de referência.

Início Rápida

Alice, a quem apresentamos na seção Usos do Bitcoin, Seus Usuários e Suas Histórias não é uma usuária técnica e só recentemente ouviu falar do bitcoin através de um amigo. Ela começa sua jornada visitando o website oficial bitcoin.org, onde encontra uma ampla seleção de clientes bitcoin. Seguindo o conselho do site bitcoin.org, ela escolhe o cliente bitcoin compacto Multibit.

Alice segue um link do site bitcoin.org para baixar e instalar a Multibit no PC dela. Multibit está disponível para computadores Windows, Mac OS e Linux.

WARNING	Uma carteira bitcoin deve ser protegida por uma senha ou frase. Há muitos criminosos tentando quebrar senhas fracas, então tenha o cuidado de selecionar uma que não possa ser facilmente decifrada. Use uma combinação de caracteres maiúsculos e minúsculos, números e símbolos. Evite usar informação pessoal como datas de nascimento ou nomes de times de futebol. Evite quaisquer palavras facilmente encontradas em dicionários, em qualquer idioma. Se puder, use um gerador de senhas para criar uma senha completamente aleatória que tenha no mínimo 12 caracteres de comprimento. Lembre-se: bitcoin é dinheiro e pode ser transferido instantaneamente para qualquer lugar do mundo. Se não for bem protegido, ele pode ser facilmente roubado.

Assim que a Alice terminar de baixar e instalar o aplicativo Multibit, ela o executa e é saudada pela tela de Boas-Vindas, como mostrado na A tela de boas-vindas do cliente bitcoin Multibit.

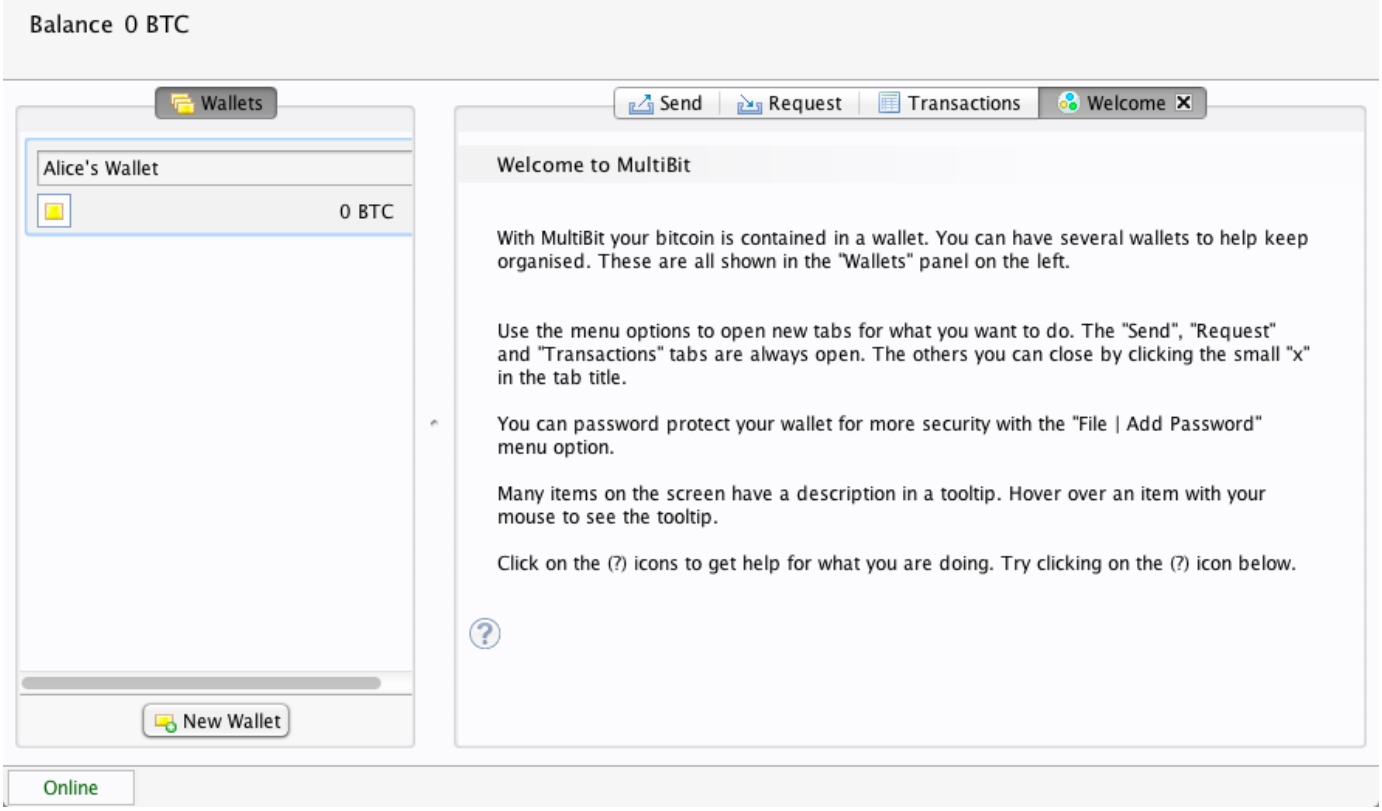


Figure 1. A tela de boas-vindas do cliente bitcoin Multibit

Multibit automaticamente cria uma carteira e um novo endereço bitcoin para Alice, que Alice pode ver clicando na aba Solicitar em O novo endereço bitcoin da Alice, na aba Solicitar do cliente Multibit.

More trading books in
https://t.me/tradingpdfgratis

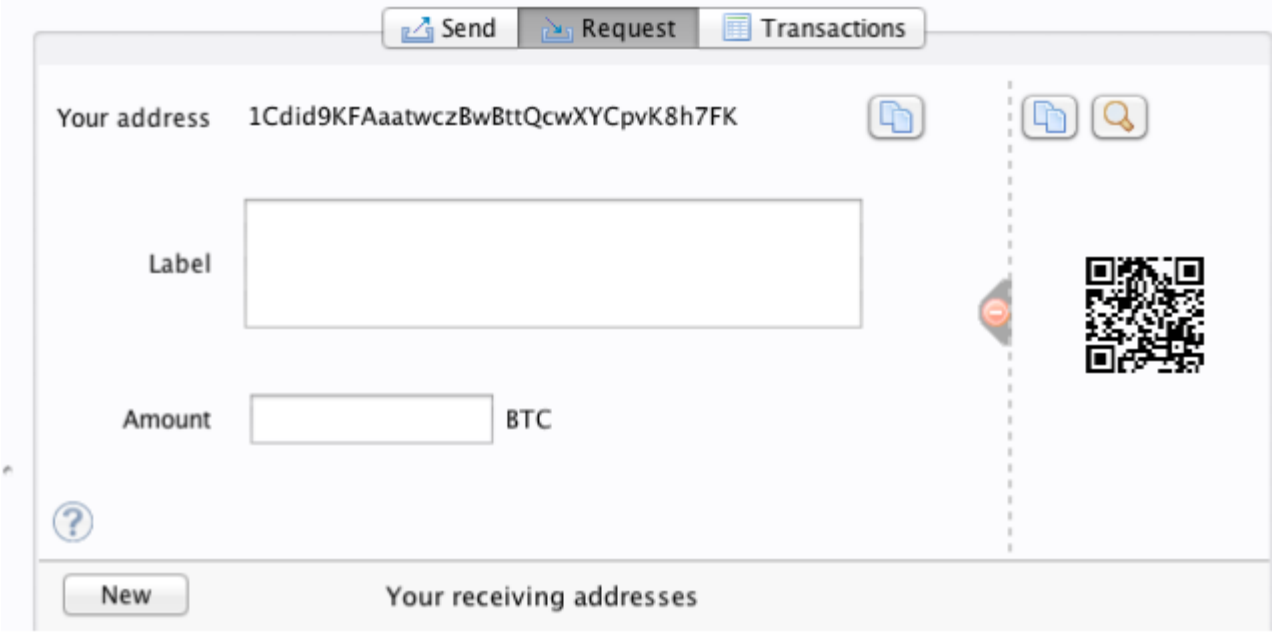


Figure 2. O novo endereço bitcoin da Alice, na aba Solicitar do cliente Multibit

A parte mais importante desta tela é o *endereço bitcoin* da Alice. Assim como um endereço de email, a Alice pode compartilhar este endereço e qualquer um pode usá-lo para mandar dinheiro diretamente à carteira dela. Na tela aparece uma longa sequência de letras e números: 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK. Junto ao endereço bitcoin da carteira está um código QR, uma forma de código de barras que contém a mesma informação, mas em um formato que pode ser escaneado pela câmera de um smartphone. O código QR é a imagem quadrada que contém pequenos quadrados preto e brancos no lado direito da janela. A Alice pode copiar o endereço bitcoin ou o código QR clicando no botão copy junto de cada um deles. Ao clicar no próprio código QR ele será ampliado, podendo facilmente ser escaneado pela câmera de um smartphone.

A Alice pode também imprimir o código QR como uma forma de passar facilmente seu endereço a outras pessoas sem que eles tenham de se lembrar de digitar uma longa sequência de letras e números.

TIP

Os endereços bitcoin começam sempre com o dígito 1 ou 3. Assim como endereços de email, eles podem ser compartilhados com outros usuários bitcoin que podem usá-los para mandar bitcoin diretamente a sua carteira. Ao contrário dos endereços de email, você pode criar novos endereços à vontade, e todos eles direcionarão os fundos para sua carteira. Uma carteira é simplesmente uma coleção de endereços e as chaves que desbloqueiam os fundos que estão nela. Você pode aumentar a sua privacidade usando um endereço diferente para cada transação. Não há nenhuma limitação na quantidade de endereços que um usuário pode criar.

Agora a Alice está pronta para começar a usar sua nova carteira bitcoin.

Obtendo Os Seus Primeiros Bitcoins

Ainda não é possível comprar os bitcoins em um banco ou casa de câmbio de moedas estrangeiras. Em 2014, ainda era difícil adquirir bitcoins na maior parte dos países. Há algumas casas de câmbio

especializadas onde você pode comprar e vender bitcoin pagando com a sua moeda local. Estas operam online como bolsas de criptomoedas e incluem:

Bitstamp

Uma bolsa de criptomoedas européia que permite comprar várias divisas inclusive euros (EUR) e dólares americanos (USD) através de transferência bancária.

Coinbase

Baseada nos EUA, é uma carteira e uma plataforma de bitcoin onde comerciantes e consumidores podem fazer transações em bitcoin. Coinbase torna fácil comprar e vender bitcoin, permitindo aos usuários se conectarem às suas contas bancárias nos EUA através do sistema ACH (Automated Clearing House).

Bolsas de criptomoedas como essas operam na interseção entre moedas nacionais e criptomoedas. Assim elas estão sujeitas às normas nacionais e internacionais e, com frequência, são específicas a um determinado país ou região econômica e se especializam nas moedas nacionais daquela região. Sua escolha de bolsas de criptomoedas será específica para moeda nacional que você usa e limitada às exchanges que operam dentro da jurisdição legal de seu país. De maneira similar a uma conta bancária, pode levar vários dias ou semanas para configurar as contas necessárias com estes serviços pois eles requerem várias formas de identificação para atender às exigências das regulações bancárias KYC (know your customer ou conheça seu cliente) e AML (anti-money laundering ou combate à lavagem de dinheiro). Assim que você tiver uma conta em um exchange bitcoin, você pode comprar e vender bitcoins rapidamente assim como você faria com uma moeda estrangeira em uma conta de corretagem.

Você pode encontrar uma lista mais completa em bitcoin charts, que é um site que mostra as cotações e outros dados de mercado obtidos de dezenas de bolsas de criptomoedas.

Como um novo usuário. há outras quatro formas de conseguir bitcoins:

- Encontre um amigo que tenha bitcoins e compre dele diretamente. Muitos usuários de bitcoin começam dessa forma.
- Use um serviço de classificados como localbitcoins.com para encontrar um vendedor na sua área para comprar os bitcoins pagando pessoalmente em dinheiro.
- Venda um produto ou serviço por bitcoin. Se você for um programador, venda as suas habilidades de programação.
- Use um caixa eletrônico de bitcoin na sua cidade. Você pode encontrar o mais perto de você consultando em um mapa online da CoinDesk.

Alice foi apresentada ao bitcoin por um amigo e portanto ela tinha uma maneira fácil de conseguir os seus primeiros bitcoins enquanto espera que sua conta em uma exchange de criptomoedas na Califórnia seja verificada e ativada.

More trading books in
https://t.me/tradingpdfgratis

Enviando e Recebendo Bitcoins

Depois de criar a sua carteira bitcoin, Alice agora está pronta para receber fundos. A carteira gera aleatoriamente uma chave privada (descrita em mais detalhes em [private_keys]) junto com o endereço bitcoin correspondente. Nesse ponto, o endereço bitcoin dela ainda não é conhecido pela rede bitcoin, nem "registrado" em qualquer parte do sistema bitcoin. O endereço bitcoin dela é simplesmente um número que corresponde a uma chave que ela pode usar para controlar o acesso aos fundos. Não há uma conta ou associação entre aquele endereço e uma conta. Até o momento em que este endereço esteja referenciado como o destinatário de um valor em uma transação publicada no ledger ou registro contábil de bitcoin (a blockchain), ele é simplesmente parte da vasta quantidade de possíveis endereços considerados "válidos" em bitcoin. A partir do momento em que esteja associado com uma transação, ele se torna parte dos endereços conhecidos na rede e a Alice poderá comprovar o saldo dela no registro público.

A Alice encontrou-se com o amigo dela, o Joe, que a apresentou ao bitcoin, em um restaurante local para que eles possam trocar alguns dólares e colocar bitcoins na conta dela. Ela trouxe um papel com o endereço dela e o código QR impressos conforme aparecem na carteira bitcoin. Não há nenhuma informação que deva ser protegida, desde um ponto de vista de segurança, no endereço bitcoin. Ele pode ser publicado em qualquer lugar sem nenhum risco de segurança à conta da Alice.

A Alice quer trocar somente 10 dólares por bitcoin, para que assim ela não arrisque muito dinheiro nessa nova tecnologia. Ela dá ao Joe uma nota de $10 e o papel impresso com seu endereço para que o Joe possa lhe mandar o montante equivalente em bitcoin.

Em seguida, Joe tem que descobrir a taxa de câmbio para que ele possa dar a quantidade certa de bitcoins à Alice. Há centenas de aplicativos e páginas web que informar a taxa de mercado atual. Eis alguns dos mais populares:

Bitcoin Charts

Um serviço de listagem de dados de mercado que informa a taxa de câmbio do bitcoin em diversas exchanges em todo o planeta, nas diferentes moedas locais

Bitcoin Average

Um site que permite, de forma simples, ver a média ponderada dos volumes negociados em cada moeda.

ZeroBlock

Um aplicativo grátis para Android e iOS que mostra o preço do bitcoin em diferentes bolsas de criptomoedas (procure por ZeroBlock, um aplicativo de preço de mercado do bitcoin para Android e iOS)

Bitcoin Wisdom

Outro serviço de listagem de dados de mercado

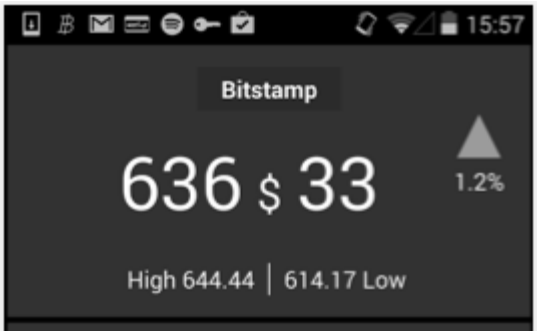


Figure 3. ZeroBlock, um aplicativo de preço de mercado do bitcoin para Android e iOS

Usando um dos aplicativos ou sites recém listados, Joe determina o preço do bitcoin como aproximadamente 100 dólares por bitcoin. Nesse momento, ele deveria dar a Alice 0.10 bitcoin, também chamado de 100 millibits, em troca dos 10 dólares que ela lhe deu.

Uma vez que Joe determinou um preço justo para a troca, ele abre um aplicativo de carteira em seu celular e seleciona "enviar" bitcoin. Por exemplo, se estiver usando a carteira da Blockchain em um telefone Android, ele veria uma tela pedindo duas informações, como mostrado em A tela de envio de bitcoin da carteira móvel Blockchain.

- O endereço bitcoin de destino para a transação
- A quantidade de bitcoins para enviar

No campo para inserir o endereço bitcoin, há um pequeno ícone que se parece com um código QR. Isso permite que Joe escaneie o código de barras com a câmera de seu smartphone para que ele não tenha que digitar o endereço bitcoin da Alice (1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK), o que seria algo grande e difícil de se digitar. Joe toca no ícone do código QR e ativa a câmera para escanear o código QR da carteira impressa que a Alice trouxe consigo. O aplicativo de carteira mobile preenche o endereço bitcoin e Joe pode verificar que o código foi escaneado corretamente ao comparar alguns dígitos com o endereço impresso pela Alice.

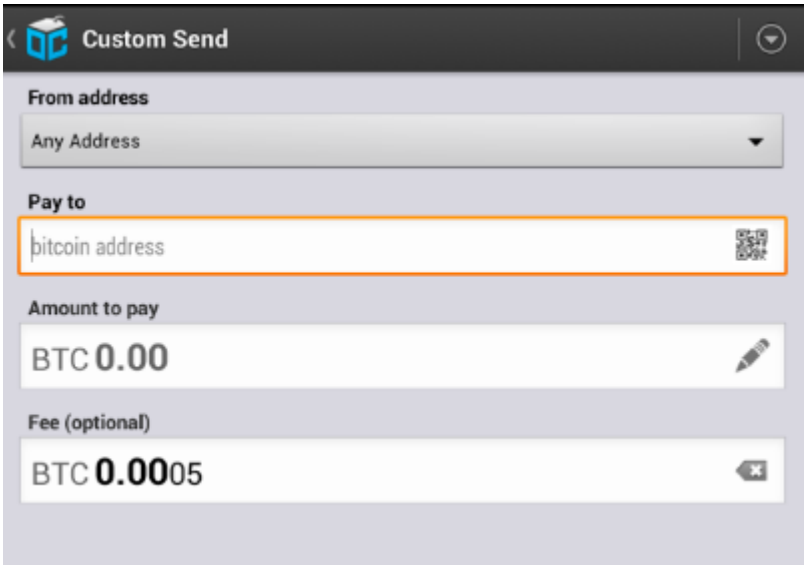


Figure 4. A tela de envio de bitcoin da carteira móvel Blockchain

Então o Joe digita o valor em bitcoins da transação, 0,10 bitcoin. Ele confere com cuidado para ter certeza de que digitou a quantia correta, pois ele está a ponto de transmitir dinheiro e qualquer erro pode sair muito caro. Finalmente ele aperta Send para transmitir a transação. A carteira móvel do Joe constrói a transação que assigna 0,10 bitcoin ao endereço da Alice, gerando os fundos da carteira do Joe e assinando a transação com as chaves privadas dele. Isso informa a rede bitcoin que o Joe autorizou uma transferência de valor de um de seus endereços para o novo endereço da Alice. À medida que a transação se transmite conforme o protocolo peer-to-peer, ela rapidamente se propaga pela rede bitcoin. Em menos de 1 segundo, a maioria dos nós com melhor conexão na rede recebem a transação e vêem o endereço da Alice pela primeira vez.

Se a Alice tiver um smartphone ou um laptop com ela, também será capaz de ver a transação. O registro contábil do bitcoin — um arquivo que não pára de crescer e que guarda cada uma das transações em bitcoin que já ocorreram desde o início — é público, o que significa que tudo que ela tem de fazer é olhar seu próprio endereço e ver se quaisquer fundos foram mandados para ele. Ela pode fazer isso facilmente no site blockchain.info, digitando o endereço dela no campo de busca. O website lhe vai mostrar uma page listando todas as transações de e para aquele endereço. Se a Alice estiver olhando essa página, vai ver uma atualização que mostra uma nova transação transferindo 0,10 bitcoin para o saldo dela logo depois do Joe apertar Send.

Confirmações

Inicialmente, o endereço da Alice vai mostrar a transação do Joe como "Transação não Confirmada." Isto significa que a transação já se propagou pela rede, mas ainda não foi incluída no registro contábil de transações do bitcoin, conhecido como a blockchain (cadeia de blocos). Para ser incluída, a transação deve ser "escolhida" por um minerador e incluída em um bloco de transações. Quando um novo bloco é criado, em aproximadamente 10 minutos, as transações dentro do bloco passam a ser aceitas como "confirmadas" pela rede e então podem ser gastas. A transação é vista instantaneamente por todos, mas só se torna "confiada" por todos quando está incluída em um novo bloco minerado.

A Alice agora é a orgulhosa dona de 0,10 bitcoin que ela pode gastar. No próximo capítulo, observaremos sua primeira compra com bitcoin e examinaremos em maiores detalhes as tecnologias de transação e propagação envolvidas.range="endofrange", startref="ix_ch01-asciidoc1")

Como Funciona o Bitcoin

Transações, Blocos, Mineração e a Blockchain

O sistema bitcoin, diferente dos tradicionais sistemas bancários e de pagamentos, é baseado em uma confiança descentralizada. Ao invés de uma autoridade central confiável, no Bitcoin a confiança é alcançada como uma propriedade emergente das interações dos diferentes participantes no sistema bitcoin. Nesse capítulo, iremos examinar o bitcoin através do rastreamento de uma transação através do sistema bitcoin e observar como ela se torna "confiável" e aceita pelo mecanismo de consenso distribuído da rede bitcoin para ser finalmente gravada na blockchain - o livro-razão distribuído que contém todas as transações.

Cada exemplo é baseado em uma transação real ocorrida na rede bitcoin, simulando as interações entre os usuários (Joe, Alice e Bob) ao enviarem fundos de uma carteira para outra. Iremos usar um site *blockchain explorer* para visualizar cada etapa. Um site block explorer (ou explorador de blockchain) é um aplicativo web que opera como um motor de busca de operações de bitcoin, que permite ao usuário verificar transações, endereços e blocos, além de ver as relações e fluxos entre eles.

Alguns dos exploradores de blockchain mais populares:

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

Cada um destes sites possui um sistema de busca que pode verificar um endereço, hash de transação ou número do bloco e encontrar o dado equivalente na rede bitcoin e na blockchain. Com cada exemplo, iremos fornecer uma URL que o levará diretamente para a entrada relevante, de forma que você possa estudar o assunto detalhadamente.

Visão Geral do Bitcoin

No diagrama de visão geral mostrado em Visão Geral do Bitcoin, vemos que o sistema bitcoin consiste de usuários com carteiras contendo chaves, transações que são propagadas pela rede e mineradores que produzem (através de computação competitiva) o consenso da blockchain - que é o registro oficial de todas as transações. Nesse capítulo, rastrearemos uma transação enquanto ela viaja através da rede e examinaremos as interações entre cada parte do sistema bitcoin. Os capítulos subsequentes investigarão a tecnologia por trás das carteiras, da mineração e do sistema de transações.

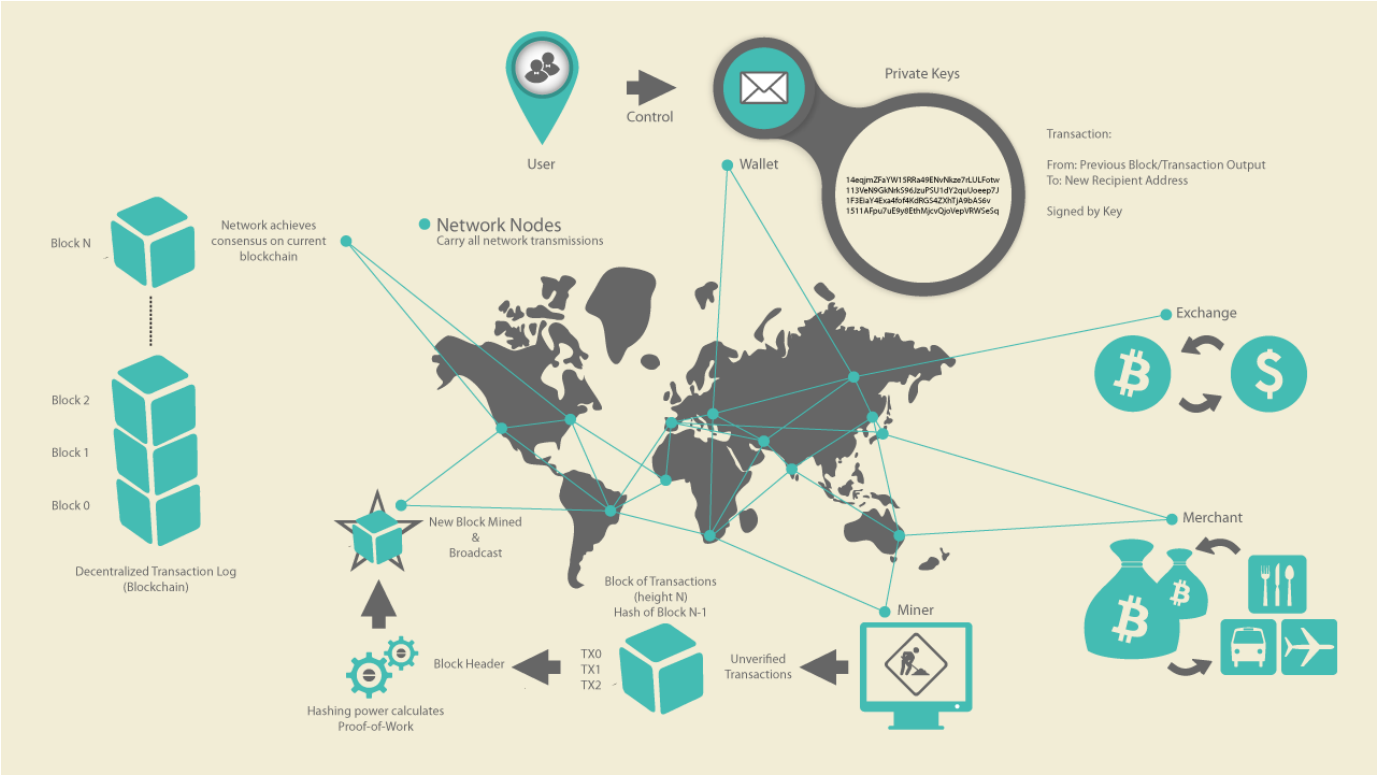


Figure 1. Visão Geral do Bitcoin

Comprando uma Xícara de Café

Alice, apresentada no capítulo anterior, é uma nova usuária que acabou de obter seu primeiro bitcoin. Em [getting_first_bitcoin], Alice encontrou com seu amigo, Joe, para trocar algum dinheiro por bitcoin. A transação criada por Joe alocou 0,10 BTC na carteira de Alice. Agora, ela irá fazer sua primeira compra, um transação de varejo, comprando uma xícara de café na cafeteria do Bob, em Palo Alto, Califórnia. A cafeteria do Bob recém começou a aceitar pagamentos em bitcoin, ao adicionar a opção de pagamentos por bitcoin no sistema do seu ponto de vendas. Os preços na cafeteria são listados na moeda local (dólares americanos), mas no caixa, os clientes agora contam com a opção de pagar tanto em dólares quanto em bitcoin. Alice faz seu pedido - uma xícara de café - e Bob entra a transação em seu sistema de vendas. O sistema do ponto de vendas fará a conversão do preço total em dólares para bitcoins, tendo como referência a cotação do momento, e apresenta o valor final nas duas moedas, bem como um código QR contendo uma *requisição de pagamento* para essa transação (ver Código QR de solicitação de pagamento (Dica: Tente escanear esse código!)):

```
Total:
$1.50 USD
0,015 BTC
```

Figure 2. Código QR de solicitação de pagamento (Dica: Tente escanear esse código!)

O código QR de solicitação de pagamento codifica a seguinte URL, definida em BIP0021:

```
bitcoin:1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA?
amount=0.015&
label=Bob%27s%20Cafe&
message=Compra%20no%20Bob%27s%20Cafe

Componentes da URL

Um endereço bitcoin: "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
Valor do pagamento (amount): "0.015"
Um rótulo para o endereço do destinatário (label): "Bob's Cafe"
Uma descrição para o pagamento (message): "Compra no Bob's Cafe"
```

TIP Ao contrário de um código QR que simplesmente contém um endereço de bitcoin como destinatário, um QR code com uma requisição de pagamento contém uma URL codificada a qual contém múltiplos parâmetros: um endereço de pagamento, um valor de pagamento e uma descrição genérica como "Bob’s Cafe". Isso permite que um aplicativo de carteira bitcoin preencha as informações usadas para enviar o pagamento enquanto mostra uma descrição intuitiva para o usuário. Você pode escanear o código QR acima com um aplicativo de carteira bitcoin para ver o que a Alice veria.

Bob diz: "A conta deu 1,50 dólares, ou 15 millibits."

A Alice então usa o smartphone dela para escanear o código de barras mostrado na tela do Bob. O smartphone dela mostra um pagamento de 0,0150 BTC para o Bob’s Cafe e ao clicar em Enviar ela autoriza o pagamento. Dentro de alguns segundos (aproximadamente o mesmo tempo que leva uma autorização de cartão de crédito), o Bob visualiza a transação em seu caixa, completando a transação.

Nas próximas seções, examinaremos essa transação em maiores detalhes, veremos como a carteira da Alice a construiu, como ela foi propagada através da rede, como ela foi verificada e, finalmente, como o Bob pode gastar a quantia recebida em novas transações futuras.

NOTE

A rede bitcoin pode fazer transações em valores fracionários, por exemplo, desde milli-bitcoins (1/1.000 de um bitcoin) até um satoshi (1/100.000.000 de um bitcoin). Ao longo deste livro nós iremos usar o termo bitcoin para se referir a qualquer quantidade na moeda bitcoin, desde a menor unidade possível (1 satoshi) até o número máximo (21.000.000) de bitcoins que podem ser minerados.

Transações Bitcoin

Em termos simples, uma transação informa para a rede que o dono de uma quantidade de bitcoins autorizou a transferência de alguns destes bitcoins para outro dono. O novo dono agora pode gastar esses bitcoins ao criar uma nova transação que autoriza a transferência para um outro dono, e assim por diante, em uma cadeia de posse de bitcoins.

As transações são como linhas em um "registro contábil" (ledger) de dupla entrada. Em termos simples, cada transação contém um ou mais "inputs" (entradas), que são débitos em uma conta bitcoin. No outro lado da transação, existem um ou mais "outputs" (saídas) que são créditos adicionados em uma conta bitcoin. A soma dos inputs e outputs (débitos e créditos) não necessariamente resultam na mesma quantia. Ao invés disso, os outputs são um pouco maiores do que os inputs, e essa diferença se dá devido à "taxa de transação", que é um pequeno pagamento coletado pelo minerador que inclui a transação no registro contábil do bitcoin (a blockchain). Uma transação bitcoin é mostrada como uma entrada no registro contábil em Transação como um registro contábil de entrada-dupla.

A transação também contém uma prova de posse para cada quantia de bitcoins (inputs) que é transferida, na forma de uma assinatura digital assinada pelo dono, que pode ser validada por qualquer pessoa, de maneira independente. Usando a terminologia do bitcoin, "gastar" é assinar uma transação que transfere um valor (de uma transação prévia) para um novo dono, o qual é identificado através de um endereço bitcoin.

TIP

Transações movimentam valores a partir de *inputs de transação* para *outputs de transação*. Um input é o lugar de onde vem o valor da moeda, geralmente um output de uma transação prévia. Um output de transação designa um novo dono para o valor ao associá-lo com um *script de travamento*. Esse script de travamento exige uma assinatura ou outra forma de validação (*script de destravamento*) para a retirada dos fundos em transações futuras. Outputs de uma transação podem ser usados como inputs em uma nova transação, dessa maneira criando uma cadeia de posses à medida que o valor é movido de um endereço para outro (ver Uma cadeia de transações, onde o output de uma transação é o input da próxima transação).

Transaction as Double-Entry Bookkeeping

Inputs	Value	Outputs	Value
Input 1	0.10 BTC	Output 1	0.10 BTC
Input 2	0.20 BTC	Output 2	0.20 BTC
Input 3	0.10 BTC	Output 3	0.20 BTC
Input 4	0.15 BTC		
Total Inputs:	0.55 BTC	Total Outputs:	0.50 BTC

	Inputs	*0.55 BTC*
-	*Outputs*	*0.50 BTC*
	Difference	*0.05 BTC (implied transaction fee)*

Figure 3. Transação como um registro contábil de entrada-dupla

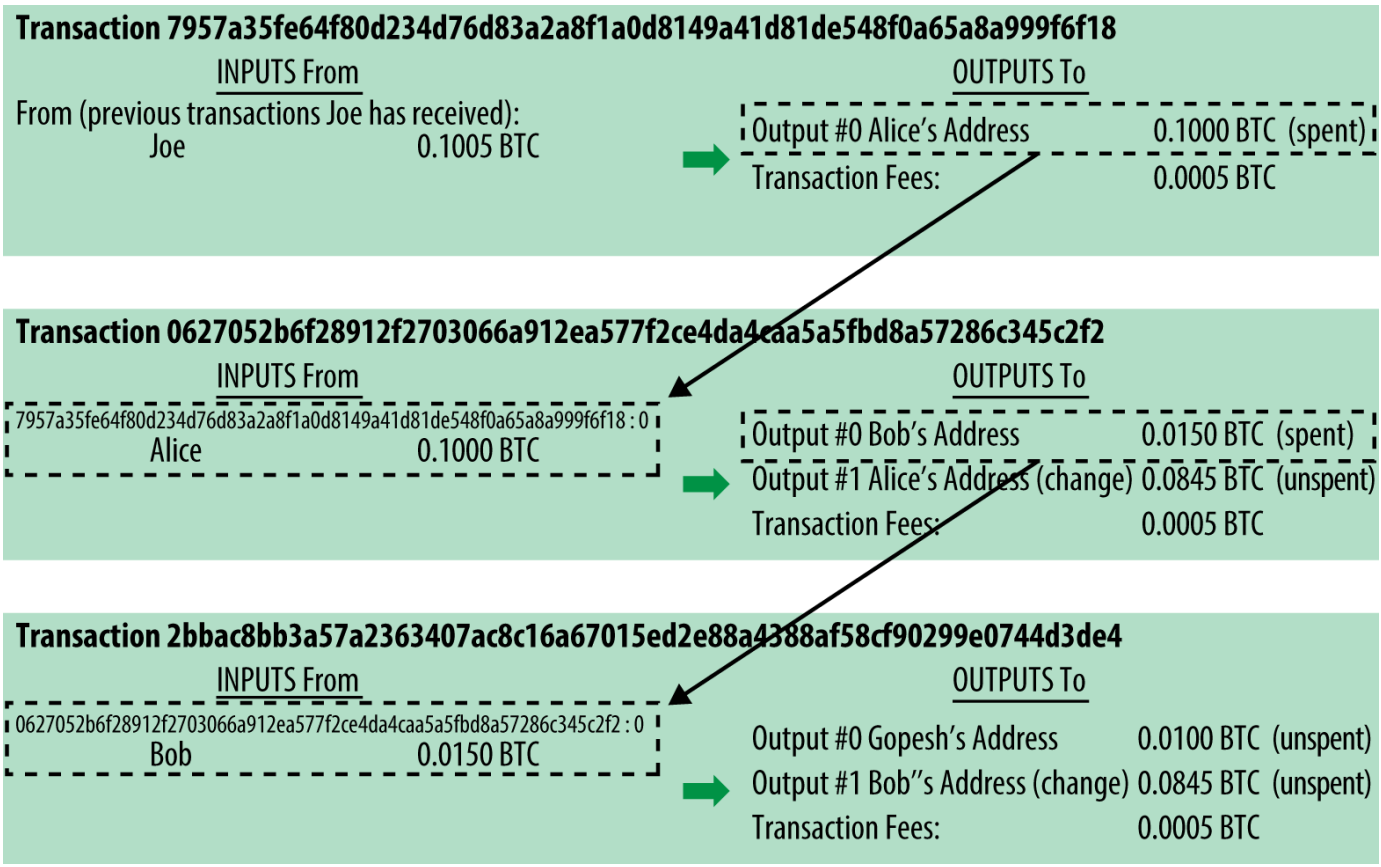


Figure 4. Uma cadeia de transações, onde o output de uma transação é o input da próxima transação

O pagamento da Alice para o Bob's Cafe usa uma transação prévia como seu input. No capítulo anterior, a Alice recebeu bitcoins do amigo dela em troca de dinheiro. Aquela transação continha um número de bitcoins "trancados" (alienados) com a chave da Alice. Sua nova transação para o Bob's Cafe utiliza a transação prévia como um input e cria novos outputs para pagar pela xícara de café e receber o troco. As transações formam uma cadeia, onde os inputs da última transação correspondem aos outputs das transações anteriores. A chave da Alice fornece a assinatura que desbloqueia estes outputs de transações prévios, desta maneira provando à rede bitcoin que ela é a dona dos fundos. Ela vincula seu pagamento pelo café ao endereço do Bob, desta maneira "alienando" este output com o requisito de que Bob produza uma assinatura, liberando essa quantidade de bitcoins para ser gasta. Isso representa a transferência de valor entre Alice e Bob. Essa cadeia de transações, do Joe para a Alice, e dela para o Bob, é ilustrada em Uma cadeia de transações, onde o output de uma transação é o input da próxima transação.

Formas Comuns de Transação

A forma mais comum de transação é um pagamento simples de um endereço para outro, que frequentemente inclui algum "troco" que é devolvido para o dono original. Esse tipo de transação possui um input e dois outputs, e é mostrada em A forma mais comum de transação..

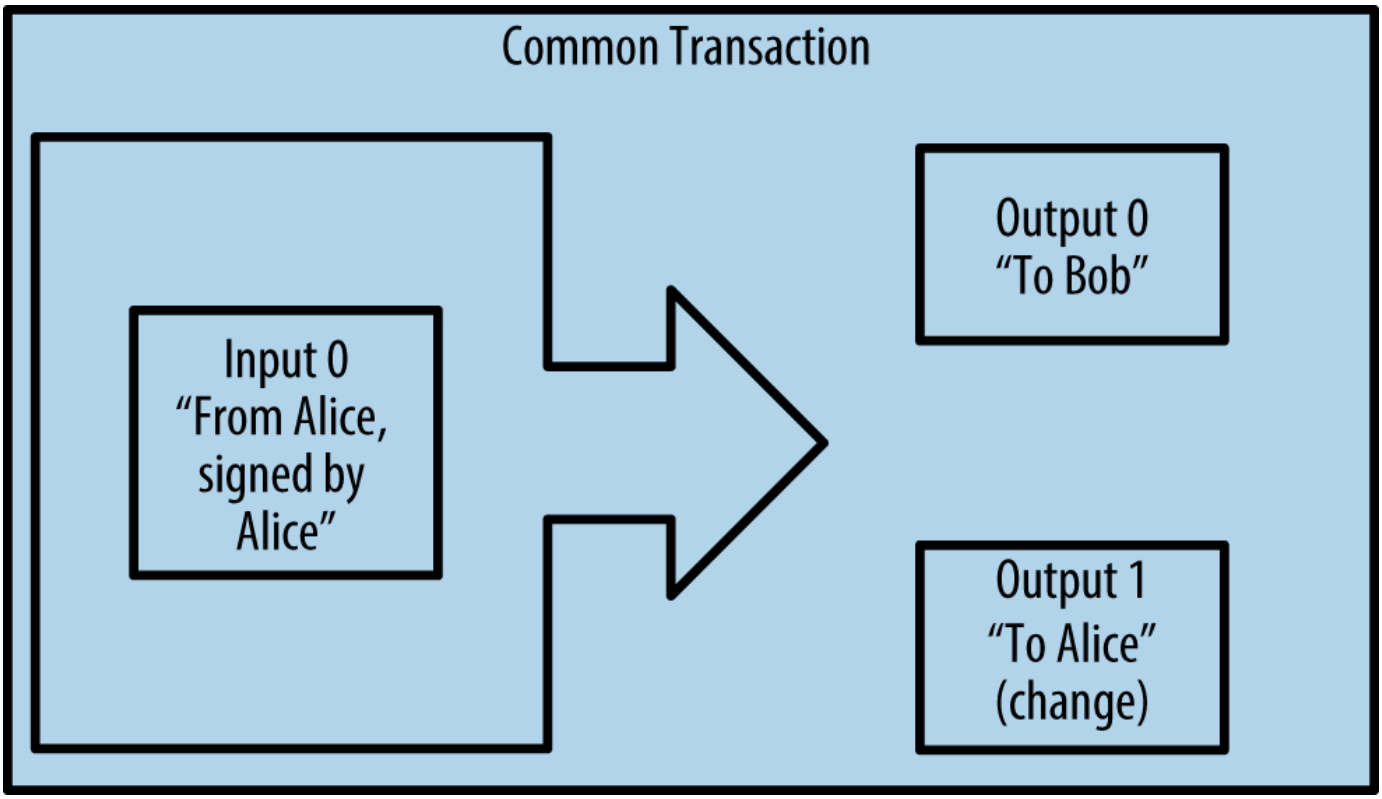


Figure 5. A forma mais comum de transação

Outra forma comum de transação é uma que agrega múltiplos inputs em um único output (ver Transação agregadora de fundos). Isso representa o equivalente no mundo real a uma troca de uma pilha de moedas e notas por uma nota de valor maior. As transações deste tipo são às vezes geradas pelos aplicativos de carteira para limpar vários valores pequenos que foram recebidos como troco pelos pagamentos efetuados.

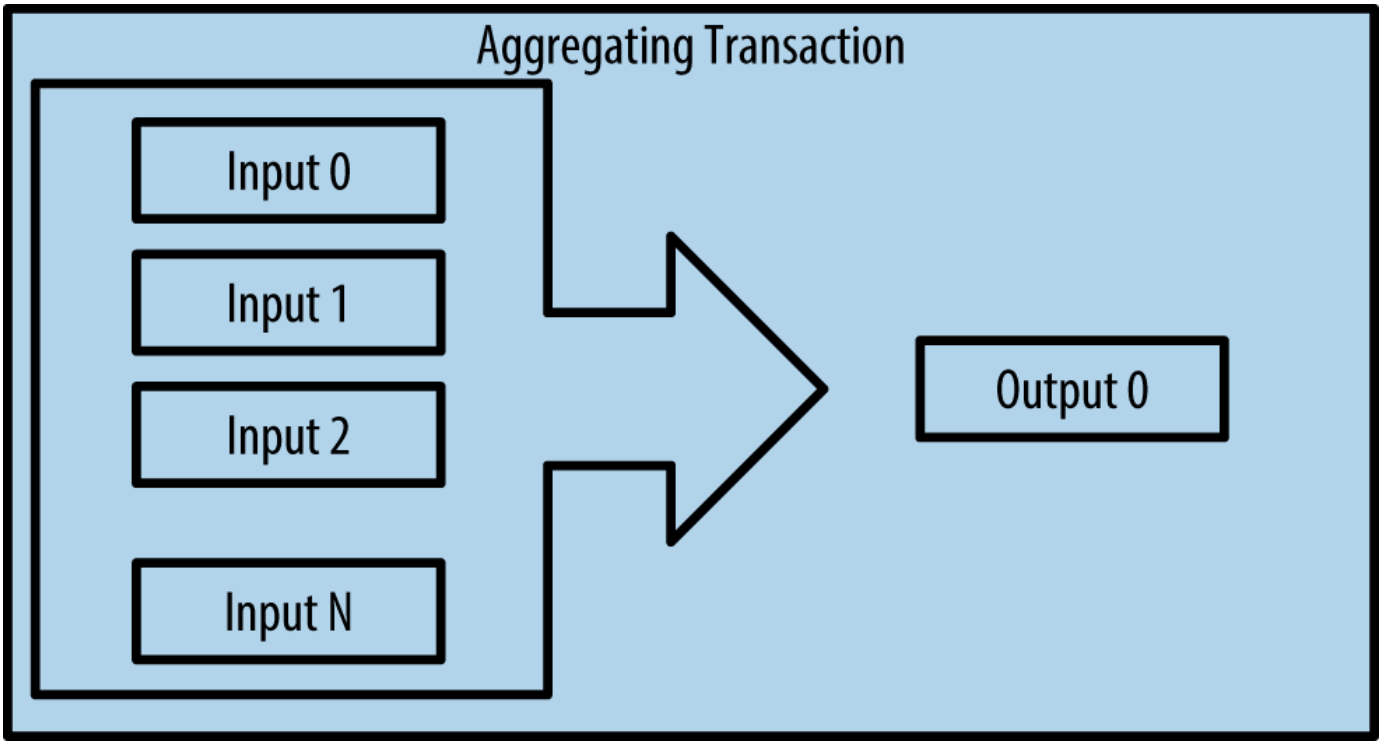


Figure 6. Transação agregadora de fundos

Finalmente, outra forma de transação frequentemente vista no registro contábil do bitcoin é uma transação que distribui um input para múltiplos outputs, que representam múltiplos destinatários (ver Transação de distribuição de fundos). Este tipo de transação às vezes é usada por entidades comerciais para distribuir fundos, como, por exemplo, ao processar folhas de pagamento para múltiplos colaboradores.

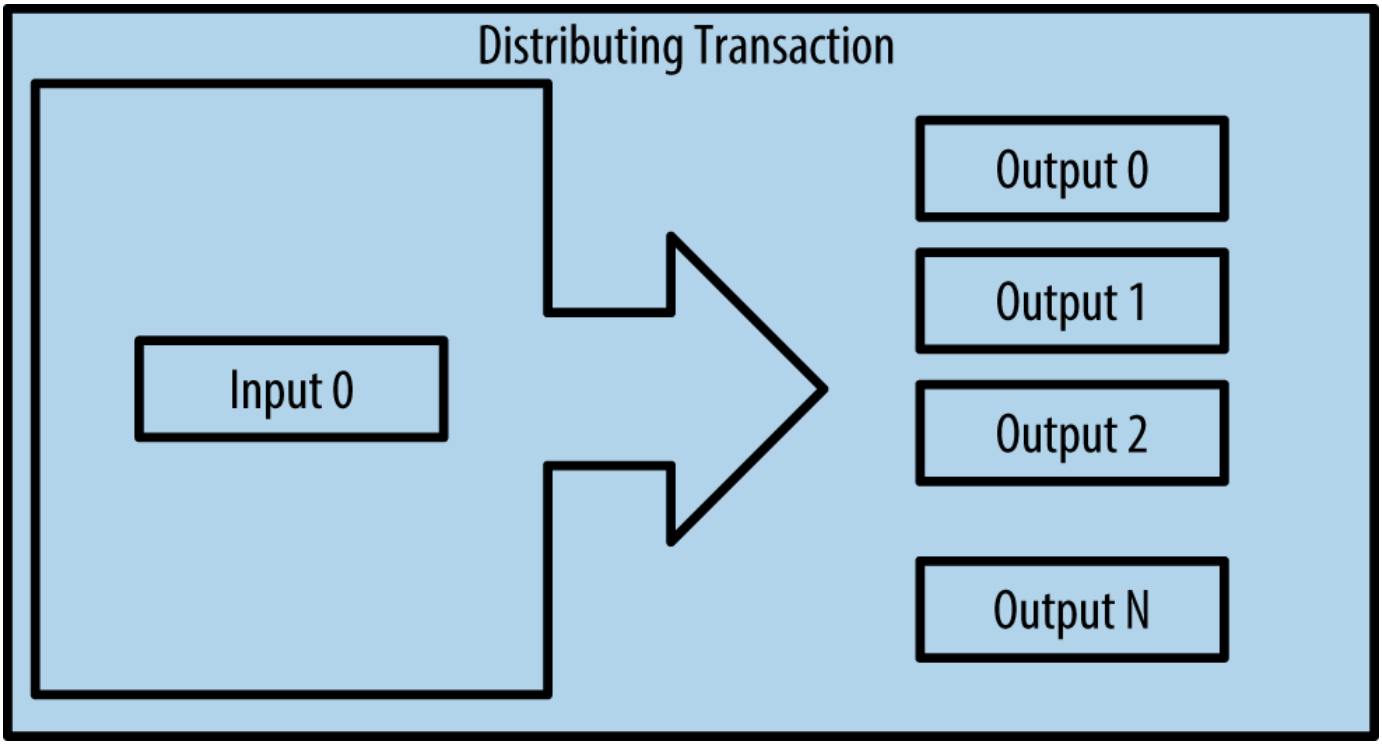


Figure 7. Transação de distribuição de fundos

Construindo uma Transação

O aplicativo de carteira contém toda a lógica para selecionar os inputs e outputs apropriados para construir uma transação com os dados especificados pela Alice. Ela só precisa fornecer os dados de destino e uma quantia: o seu aplicativo de carteira faz todo o resto, sem que ela sequer veja os detalhes. Outro aspecto importante, é que o aplicativo de carteira também pode construir transações mesmo estando completamente offline. Da mesma maneira que você pode preencher um cheque em casa para depois depositá-lo em um envelope no banco, uma conexão com a rede bitcoin não é necessária para que uma transação seja construída e assinada. A transação só precisa ser enviada para a rede quando a pessoa quiser efetuá-la.

Recebendo os Inputs Certos

O aplicativo de carteira da Alice terá primeiro que achar os inputs que podem pagar pela quantia que ela quer enviar para o Bob. A maioria dos aplicativos de carteira mantém um pequeno banco de dados de "outputs de transações não gastos" que são trancados (alienados) com as próprias chaves da carteira. Logo, a carteira de Alice iria conter uma cópia do output da transação do Joe, que foi criada na troca pelo dinheiro (ver [getting_first_bitcoin]). Um aplicativo de carteira de bitcoin que roda como um cliente de índice completo na verdade contém uma cópia de cada output não gasto de todas as transações presentes na blockchain. Isso permite que a carteira construa inputs de transação, além de verificar rapidamente se as transações que chegam tem inputs corretos. No entanto, como um cliente de índice completo ocupa muito espaço de armazenamento em disco, a maioria das carteiras roda clientes "leves" que mantém somente o registro dos outputs não gastos do usuário.

Se a wallet não mantiver uma cópia dos outputs de transação não-gastos, ela pode fazer uma requisição à rede bitcoin para solicitar essa informação, usando as APIs (ou Interfaces de Programação de Aplicações) que os diferentes fornecedores colocam à disposição, ou fazendo uma requisição a um nó de índice completo usando um API de bitcoin JSON RPC. Consultando todos os outputs não gastos do endereço de bitcoin da Alice mostra que todos os outputs não-gastos para o endereço de bitcoin de Alice mostram uma requisição API RESTful, construído como um comando HTTP GET para uma URL específica. Essa URL irá retornar todos os outputs de transação não gastos para um endereço, fornecendo para qualquer aplicativo a informação necessária para construir inputs de transação de tal forma que os bitcoins sejam gastos. Nós usamos um simples cliente HTTP de linha de comando *cURL* para solicitarmos a resposta.

Example 1. Consultando todos os outputs não gastos do endereço de bitcoin da Alice

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

Example 2. Resposta à consulta

```
{

    "unspent_outputs":[

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

    ]
}
```

A resposta no Resposta à consulta mostra um output não-gasto (um que ainda não foi resgatado) sob a posse do endereço de Alice 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK. A resposta inclui uma referência à transação na qual esse valor não-gasto está contido (o pagamento do Joe) e seu valor em satoshis, 10 milhões, equivalente a 0,10 bitcoin. Com essa informação, o aplicativo carteira de Alice pode construir uma transação para transferir o valor para o endereço do novo dono.

TIP Veja a transação de Joe para Alice.

Como você pode ver, a carteira de Alice contém bitcoins suficientes em um output não-gasto isolado para pagar pela xícara de café. Caso não contivesse, o aplicativo carteira de Alice teria que "vasculhar" uma pilha de pequenos outputs não-gastos, como se estivesse pegando as moedas em uma bolsa, até encontrar o suficiente para poder pagar o café. Em ambos os casos, pode haver uma necessidade de receber algum troco de volta, que é o assunto que iremos ver na próxima seção, quando o aplicativo carteira cria os outputs da transação (pagamentos).

Criando os Outputs

Um output de transação é criado na forma de um script que cria uma alienação no valor a ser transferido, de maneira que o valor só pode ser regastado se uma solução for apresentada ao script. De maneira simplificada, o output da transação de Alice irá conter um script que diz algo como "Esse output é pagável para aquela pessoa que conseguir apresentar uma assinatura para a chave correspondente ao endereço público de Bob". Como somente o Bob possui a carteira com as chaves correspondentes àquele endereço, somente a carteira de bob pode apresentar a assinatura para resgatar esse output. A Alice ao fazer uma exigência de assinatura do Bob, ela está fazendo uma "alienação" ao valor de output.

Essa transação também incluirá um segundo output, porque os fundos de Alice estão na forma de um output de 0,10 BTC, que é dinheiro demais para a transação de 0,015 BTC pela xícara de café. A Alice precisará de 0,085 BTC de troco. O pagamento do troco da Alice é criado pela carteira de Alice na mesma transação que o pagamento do Bob. Essencialmente, a carteira de Alice divide seus fundos em dois pagamentos: um para o Bob, e outro de volta para si mesma. Ela pode então usar o output do troco em uma transação no futuro, gastando-o mais tarde.

Finalmente, para que a transação seja processada pela rede em tempo hábil, o aplicativo de carteira da Alice irá adicionar uma pequena taxa. Isso não está explícito na transação: isso está implícito na diferença entre os inputs e os outputs. Se ao invés de receber 0,085 de troco, Alice cria somente 0,0845 como um segundo output, haverá 0,0005 (metade de um milibitcoin) restantes. O input de 0,10 BTC não é totalmente gasto com os dois outputs, porque ele irá se somar até menos do que 0,10. A diferença resultante é a *taxa de transação* que é coletada pelo minerador como um pagamento por ter incluído a transação em um bloco e adicionar esse bloco no ledger da blockchain.

A transação resultante pode ser vista usando um aplicativo web explorador de blockchain, como visto em Transação de Alice para o Bob's Cafe.

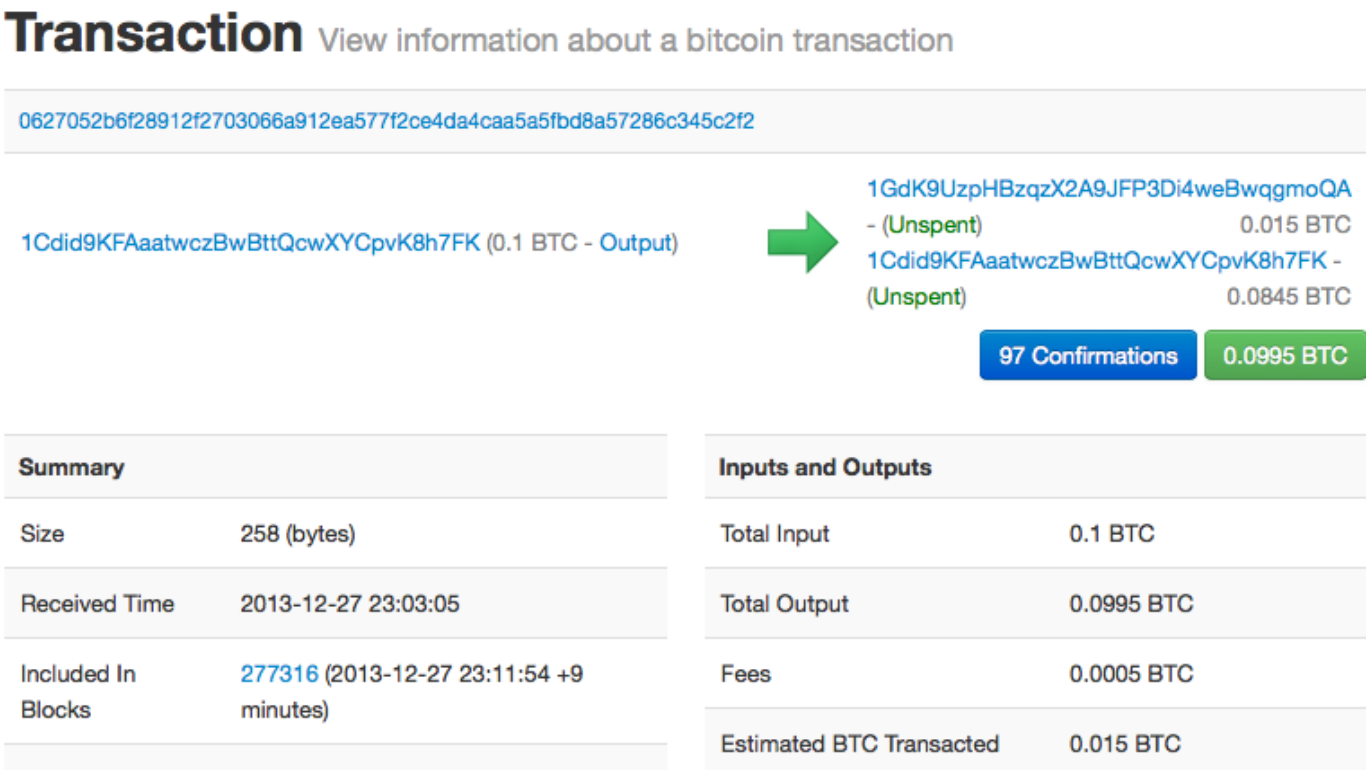


Figure 8. Transação de Alice para o Bob's Cafe

TIP Veja a transação de Alice para o Bob's Cafe.

Adicionando uma Transação ao Registro (Ledger)

A transação criada pelo aplicativo de carteira da Alice tem 258 bytes de comprimento e contém todas as informações necessárias para confirmar a sua posse dos fundos e para designar novos donos. Agora,

a transação deve ser transmitida para rede bitcoin, onde ela se tornará parte do ledger distribuído (da blockchain). Na próxima seção, iremos ver como a transação torna-se parte de um novo bloco e como o bloco é "minerado". Por fim, iremos ver como o novo bloco, após ser adicionado à blockchain, torna-se cada vez mais confiável conforme novos blocos são adicionados posteriormente à ele.

Transmitindo a transação

Como a transação contém toda a informação necessária para que seja processada, não importa como ou onde ela é transmitida para a rede bitcoin. A rede bitcoin é uma rede ponto-a-ponto (P2P), com cada cliente bitcoin participando ao se conectar a múltiplos outros clientes bitcoins. A proposta da rede bitcoin é propagar as transações e os blocos para todos os participantes.

Como ela se propaga

A carteira da Alice pode enviar a nova transação para qualquer um dos outros clientes bitcoins se ela estiver conectada através de uma conexão de Internet: por cabo, WiFi ou móvel. A sua carteira não tem que obrigatoriamente estar conectada diretamente à carteira do Bob ou usar a conexão de internet oferecida pela cafeteria, embora essas opções também sejam possíveis. Qualquer nó (outro cliente) na rede bitcoin que receber uma transação válida que não tenha sido vista anteriormente, irá propagá-la imediatamente para outros nós com os quais está ligado. Logo, a transação rapidamente é propagada através da rede ponto-a-ponto (P2P), atingindo uma grande percentagem dos nós dentro de poucos segundos.

A visão do Bob

Se a wallet do Bob estiver diretamente conectada à wallet da Alice, o aplicativo pode ser o primeiro nó a receber a transação. Entretanto, mesmo que a carteira de Alice envie a transação através de outros nós, a transação chegará à carteira do Bob dentro de pouco segundos. A carteira de Bob irá identificar imediatamente a transação de Alice como um pagamento porque ela contém outputs que são resgatáveis pelas chaves do Bob. A carteira de Bob também pode verificar independentemente que a transação é bem formada, utiliza inputs previamente não-gastos e contém taxas de transação suficientes para ser incluída no próximo bloco. Neste momento Bob pode esperar, com um alto grau de probabilidade, que a transação será em breve incluída em um bloco e será confirmada.

TIP

Uma ideia erroneamente difundida é a de que as transações bitcoin, para serem "confirmadas", exigem uma espera de 10 minutos por um novo bloco, ou de até 60 minutos por seis confirmações. Embora essas confirmações sejam uma garantia de que a transação foi aceita por toda a rede, a espera por elas é desnecessária para itens de pequeno valor, como uma xícara de café. Ao aceitar uma transação de pequeno valor como comprovadamente válida, o comerciante estará correndo um risco menor do que quando recebe um pagamento de cartão de crédito feito sem assinatura ou carteira de identidade, algo que é rotineiramente feito hoje em dia.

Mineração de Bitcoin

A transação foi propagada na rede bitcoin. Ela só vai tornar-se parte de ledger compartilhado (a *blockchain*) quando for verificada e incluída em um bloco, através de um processo chamado *mineração*. Veja [ch8] para uma explicação mais detalhada.

O sistema de confiança do bitcoin é baseado em computação. As transações são agrupadas em *blocos*, o que requer uma enorme quantidade de processamento para prová-las, mas apenas uma pequena quantidade de processamento para verificá-las como previamente provadas. O processo de mineração do bitcoin possui dois propósitos:

- A mineração cria novos bitcoins em cada bloco, quase como um banco central imprimindo novas moedas e notas. A quantidade de bitcoin criada por bloco é fixa e diminui com o tempo.
- A mineração cria confiança ao garantir que as transações sejam confirmadas somente se poder de processamento suficiente for dedicado ao bloco que as contém. Mais blocos requerem mais processamento, o que significa maior confiança.

Uma boa maneira de descrever a mineração é como um jogo de sudoku, gigantesco e competitivo, que reinicia cada vez que alguém encontra uma solução e cuja dificuldade se ajusta automaticamente, de maneira que leve cerca de 10 minutos para que uma solução seja encontrada. Imagine um sudoku gigantesco, com milhares de colunas e linhas de tamanho. Se eu mostrar para você um sudoku completo, você pode verificar rapidamente que ele está corretamente preenchido. No entando, se o sudoku tiver apenas alguns quadrados preenchidos e o resto estiver vazio, levará muito trabalho para resolvê-lo! A dificuldade do sudoku pode ser ajustada ao mudar o seu tamanho (mais ou menos linhas ou colunas), mas o sudoku ainda pode ser verificado de maneira rápida, mesmo que ele seja muito grande. O "quebra-cabeças" usado no bitcoin é baseado em um hash criptográfico, que exibe características semelhantes: ele é assimetricamente difícil de resolver, mas fácil de verificar, e sua dificuldade pode ser ajustada.

Em [user-stories], nós apresentamos o Jing, um estudante de engenharia da computação de Shanghai. Ele está participando da rede bitcoin como um minerador. À cada 10 minutos em média, Jing se une a milhares de outros mineradores para uma corrida global para achar uma solução para um bloco de transações. Encontrar a tal solução, também chamada de prova de trabalho, requer quadrilhões de operações de hashing por segundo ao longo de toda a rede bitcoin. O algoritmo para a prova de trabalho envolve fazer hashing com o cabeçalho do bloco e um número aleatório com um algoritmo criptográfico SHA256 até que a solução correspondente a um determinado padrão surja. O primeiro minerador a encontrar uma solução ganha a rodada da competição e publica o bloco na blockchain.

Jing começou a minerar em 2010 usando um computador destktop muito rápido para achar provas de trabalho adequadas para novos blocos. Conforme mais mineradores começaram a se juntar à rede bitcoin, a dificuldade do problema cresceu rapidamente. Logo em seguida, Jing e outros mineradores fizeram upgrade para um hardware mais especializado, como placas com unidades de processamento gráfico (GPUs) dedicadas de alta performance, como as placas de vídeo utilizadas para jogos de desktop ou videogames. Nesse momento, a dificuldade está tão alta que só é rentável minerar com circuitos integrados específicos para a aplicação (ASIC), que é essencialmente centenas de algoritmos de

mineração impressos em hardware, rodando em paralelo em um único chip de silício. Jing também se uniu ao "mining pool", que é como uma mina coletiva que permite que vários participantes compartilhem seus esforços e recompensas. Jing agora roda duas máquinas ASIC ligadas a USB para minerar bitcoins 24 horas por dia. Ele paga seus custos de eletricidade com a venda dos seus bitcoins minerados, obtendo algum lucro dos seus bitcoins. Seu computador roda uma cópia do bitcoind, um cliente bitcoin de referência, como um backend para seu software de mineração especializado.

Minerando Transações em Blocos

Uma transação transmitida pela rede não é verificada até que ela se torna parte do ledger distribuído global, a blockchain. A cada 10 minutos em média, os mineradores geram um novo bloco que contém todas as transações que ocorreram desde o último bloco. As novas transações estão constantemente sendo adicionadas à rede pelas carteiras e outros aplicativos dos usuários. Quando elas são vistas pelos nós da rede bitcoin, elas são adicionadas a um pool temporário de transações não-verificadas que é mantida por cada nó. Ao construir um novo bloco, os mineradores adicionam as transações não-verificadas deste pool para um novo bloco, e tentam resolver um problema (prova de trabalho) muito difícil (também conhecido como prova-de-trabalho) para provar a validade deste novo bloco. O processo de mineração é explicado em maiores detalhes em [mining].

As transações são adicionadas ao novo bloco, recebendo prioridade as transações que possuem as maiores taxas de transação, além de alguns outros critérios. Cada minerador inicia o processo de mineração de um bloco de transação tão logo ele recebe o bloco anterior da rede, sabendo que ele perdeu a rodada anterior da competição. Ele imediatamente cria um novo bloco, preenche-o com transações e impressões digitais do bloco anterior, e começa a calcular a prova-de-trabalho para o novo bloco. Cada minerador inclui uma transação especial em seu novo bloco, que paga uma recompensa de novos bitcoins recém criados (atualmente 25 BTC por bloco), que serão enviados para o endereço bitcoin do minerador. Se ele encontra uma solução que torna o bloco válido, ele "ganha" essa recompensa porque seu bloco é adicionado à blockchain e a transação especial de recompensa que ele incluiu se torna gastável. Jing, que participa de um pool de mineração, programou seu software para criar novos blocos que designam uma recompensa para um endereço de pool. Desta maneira, uma parte da recompensa recebida é distribuída entre Jing e outros mineradores, de acordo com a quantidade de trabalho que cada um contribuiu na última rodada.

A transação de Alice foi incluída na rede e adicionada no pool de transações não-verificadas. Como ela tinha taxas de transação suficientes, ela foi incluída em novo bloco gerado pela pool de mineração do Jing. Aproximadamente cinco minutos após a transação ter sido inicialmente transmitida pela carteira de Alice, o equipamento de mineração ASIC do Jing encontrou uma solução para o bloco e publicou-o como bloco #277316, contendo outras 419 transações. O equipamento de mineração ASIC do Jing publicou o novo bloco na rede bitcoin, onde outros mineradores o validaram e iniciaram uma nova rodada da corrida para gerar o próximo bloco.

Você pode ver o bloco que inclui a transação de Alice.

Alguns minutos mais tarde, um novo bloco, #277317, é minerado por outro minerador. Como esse novo bloco é baseado no bloco anterior (#277316) que continha a transação de Alice, ele adicionou ainda

mais processamento computacional neste bloco anterior, desta maneira fortalecendo a confiança nas transações contidas no bloco. Logo, após esse processamento adicional do bloco contendo a transação de Alice, considera-se que a transação da Alice contida no bloco recebeu uma "confirmação". Cada que é bloco minerado após um bloco anterior contendo transações, gera uma confirmação adicional para cada uma destas transações. Conforme os blocos se empilham um sobre os outros, torna-se exponencialmente mais difícil de se reverter a transação, dessa maneira tornando-a cada vez mais confiável pela rede.

No diagrama em Transação de Alice incluída no bloco #277316 podemos ver o bloco #277316, que contém a transação de Alice. Abaixo dele há 277316 blocos (incluindo o bloco #0), ligados uns aos outros, formando uma corrente de blocos (blockchain) que se estende até o seu bloco inicial (#0), também conhecido como *bloco gênese*. Ao longo do tempo, a "altura" da pilha de blocos aumenta, aumentando a dificuldade de processamento computacional necessário para cada bloco e para toda a corrente. Os bloco minerados após o bloco que contém a transação de Alice são considerados uma garantia adicional, já que eles receberam mais processamento computacional em uma corrente cada vez maior. Por convenção, considera-se irrevogável o bloco que já recebeu seis ou mais confirmações, porque seria necessária uma imensa capacidade de poder computacional para invalidar ou recalcular seis blocos. Nós iremos examinar em mais detalhes o processo de mineração e a maneira como ele constrói a confiança no [ch8].

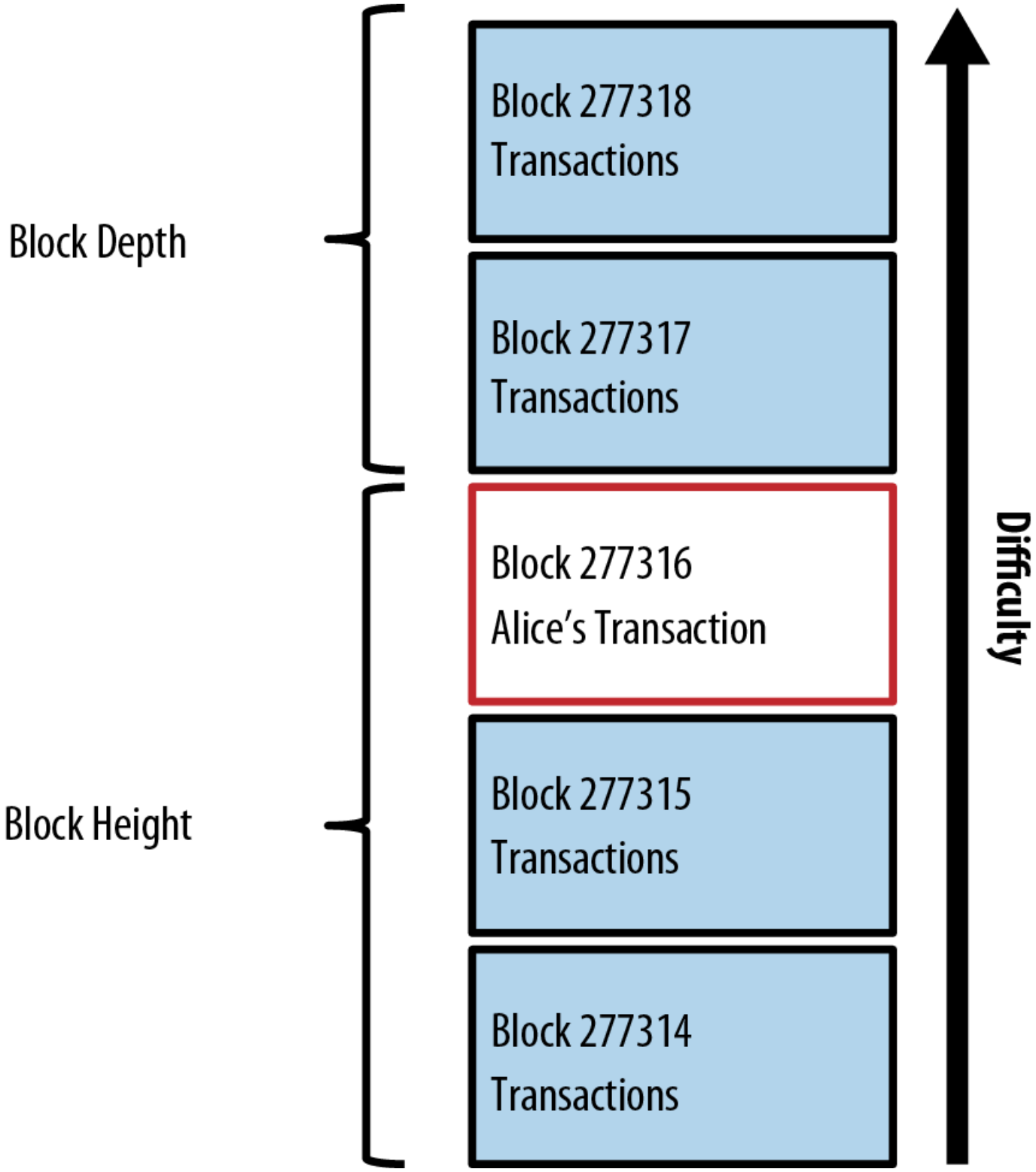


Figure 9. Transação de Alice incluída no bloco #277316

Gastando a transação

Agora que a transação da Alice foi incorporada à blockchain como parte de um bloco, ela faz parte do registro contábil distribuído do bitcoin e está visível para todos as aplicações bitcoin. Cada cliente bitcoin pode verificar independentemente que a transação é válida e que seus fundos podem ser

gastos. Clientes de índice completo (full-index) podem rastrear a origem dos fundos desde o início, ou seja, o momento em que os bitcoins foram gerados em um bloco, e, progredindo de transação a transação, até chegarem ao endereço do Bob. Clientes leves (lightweight) podem fazer uma verificação simplificada de pagamento (ver [spv_nodes]) ao confirmar que a transação está presente na blockchain e que vários blocos foram minerados após ela, garantindo que ela foi aceita pela rede como válida.

O Bob agora pode gastar o output desta e de outras transações, ao criar suas próprias transações que usam esses outputs como inputs e os designam para um novo dono. Por exemplo, Bob pode pagar um fornecedor ao transferir, para este novo dono, o valor do pagamento da xícara de café da Alice. Mais provavelmente, o software de bitcoin do Bob irá agregar vários pequenos pagamentos em um pagamento maior, talvez concentrando em uma única transação todo o lucro em bitcoins obtidos na loja em um dia. Isso moveria todos os pagamentos para um endereço único, usado como uma conta de "checking" geral da loja. Para ver um diagrama de uma transação agregadora, leia Transação agregadora de fundos.

À medida que o Bob gasta os pagamentos que recebeu de Alice e outros clientes, ele estende a cadeia de transações, que por sua vez são adicionadas ao ledger global do blockchain para que todos possam ver e confiar. Vamos assumir que o Bob paga seu webdesigner Gopesh em Bangalore para desenvolver um novo site. Agora a cadeia de transações irá ficar parecida como na figura Transação da Alice fazendo parte de uma cadeia de transação do Joe para o Gopesh.

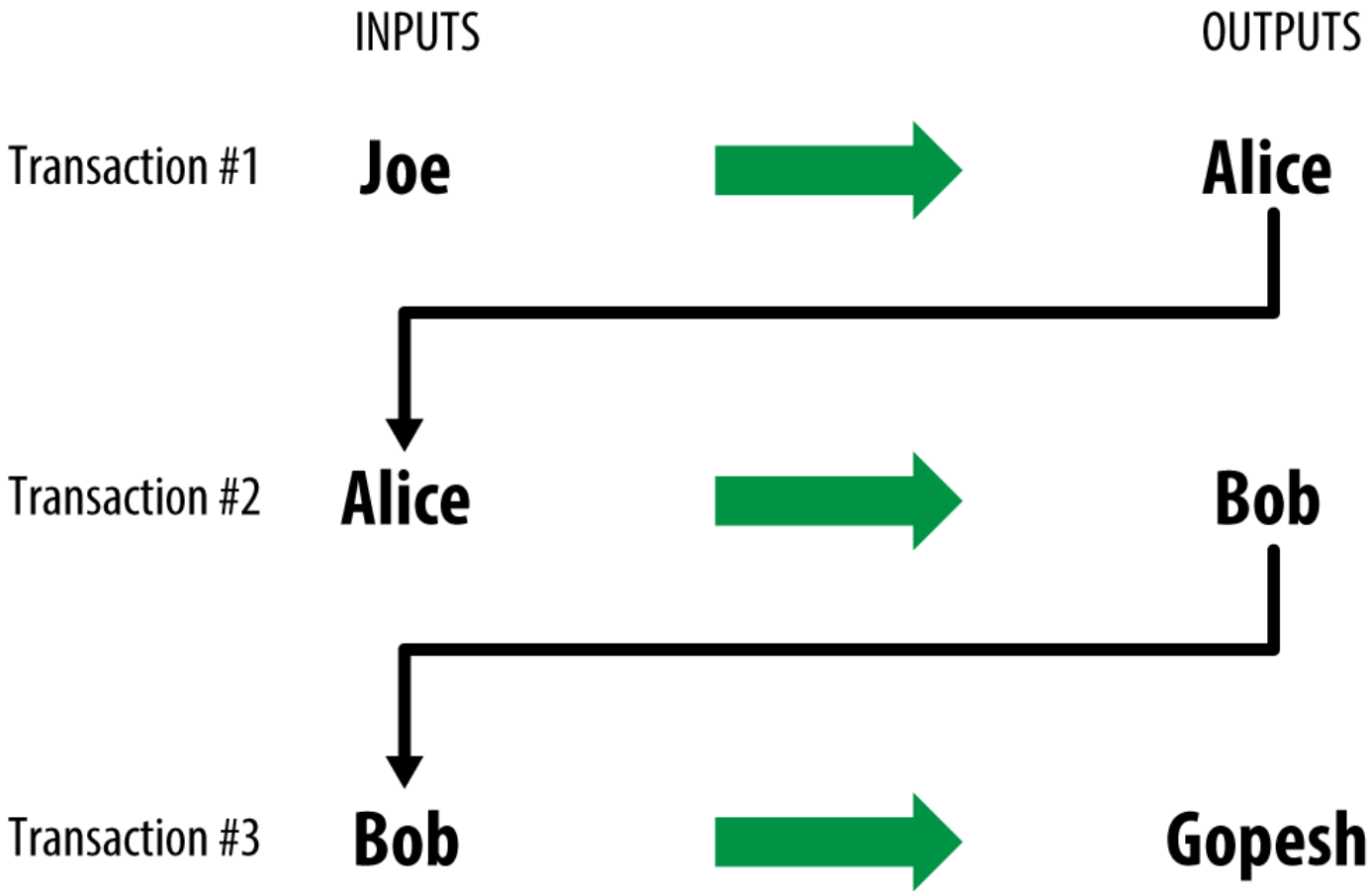


Figure 10. Transação da Alice fazendo parte de uma cadeia de transação do Joe para o Gopesh

O Cliente Bitcoin

Bitcoin Core: A Implementação de Referência

Em http://www.bitcoin.org, você pode fazer o download do *Bitcoin Core*, o cliente de referência do bitcoin, também conhecido como o "cliente Satoshi". O cliente de referência implementa todos os aspectos do sistema bitcoin, incluindo carteiras, mecanismo de verificação de transações com uma cópia completa de toda a blockchain (o ledger de transações), e um nodo completo da rede ponto-a-ponto do bitcoin.

Em Bitcoin's Choose Your Wallet page, selecione Bitcoin Core para baixar o cliente de referência. Dependendo do seu sistema operacional, você irá fazer o download de um instalador executável. Para o Windows, ele um arquivo ZIP ou um executável .exe. Para Mac OS, ele é um arquivo de imagem de disco .dmg. As versões do Linux incluem um pacote PPA para Ubuntu ou um arquivo tar.gz. A página bitcoin.org que lista os clientes bitcoins recomendados é mostrada em Escolhendo um cliente bitcoin em bitcoin.org.

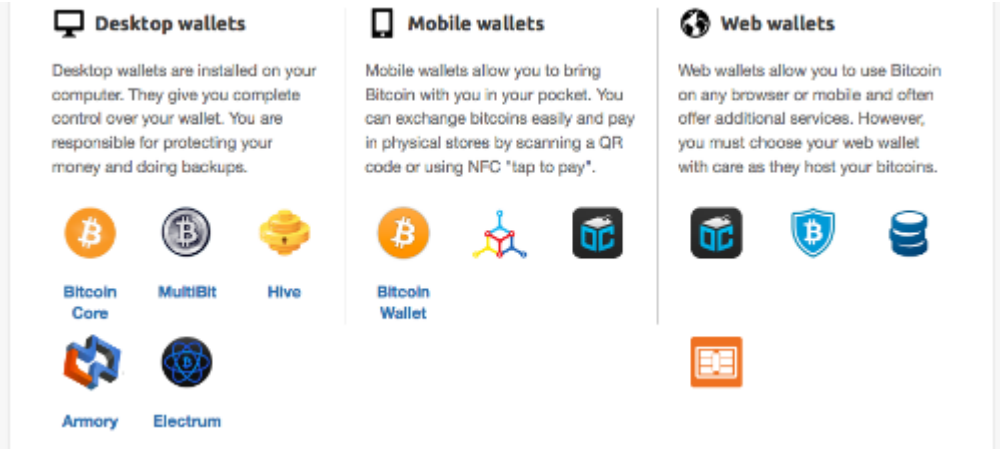


Figure 1. Escolhendo um cliente bitcoin em bitcoin.org

Executando o Bitcoin Core pela Primeira Vez

Se você baixar um pacote de instalação, como um .exe, .dmg ou PPA, você pode instalá-lo da mesma maneira que qualquer aplicativo em seu sistema operacional. Para Windows, execute o .exe e siga a instalação passo-a-passo. Para Mac OS, execute o .dmg e arraste o ícone Bitcoin-QT para sua pasta *Aplicativos*. Para Ubuntu, clique duas vezes sobre o PPA em seu Explorador de Arquivos e ele irá abrir o gerenciador de pacotes para instalar o pacote. Uma vez que instalação estiver completa, você terá um novo aplicativo chamado Bitcoin-Qt em sua lista de aplicativos. Dê um duplo clique nesse ícone para iniciar o cliente bitcoin.

Na primeira vez que você executa do Bitcoin Core, ele iniciará o download da Blockchain em um processo que pode demorar vários dias (veja [bitcoin-qt-firstload]). Deixe ele executando em segundo plano até que ele exiba "Sincronizado" e não mais exiba "Fora de Sincronia" próximo ao valor do saldo.

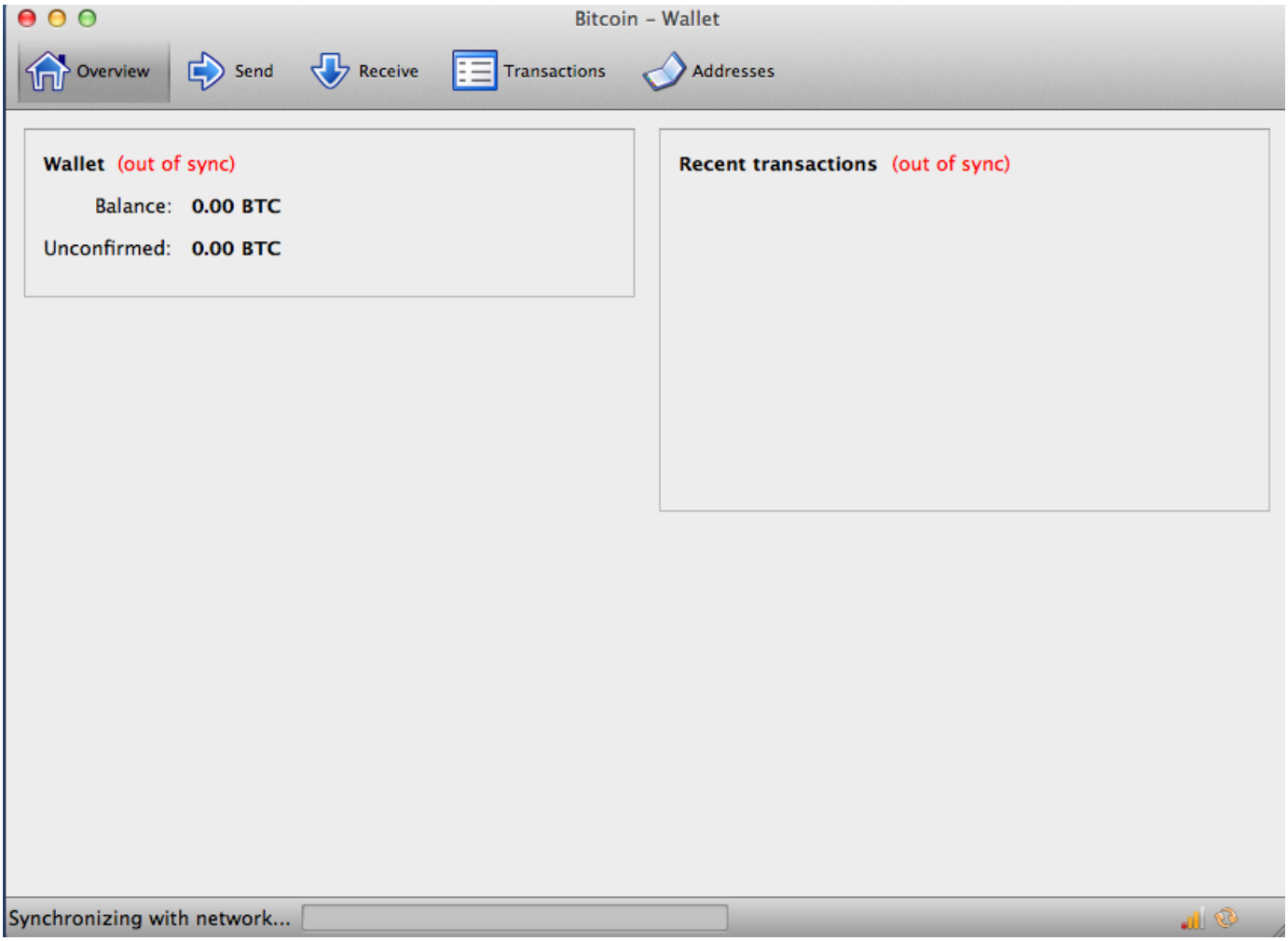


Figure 2. A tela do Bitcoin Core durante a inicialização da blockchain

TIP

O Bitcoin Core mantém uma cópia completa do registro de transações (a blockchain), que contém cada transação que já ocorreu na rede bitcoin desde seu início em 2009. Esse conjunto de dados tem vários gigabytes de tamanho (cerca de 34 GB em maio de 2015) e seu download é feito de maneira incremental ao longo de vários dias. O cliente não poderá processar transações ou atualizar o saldo da conta a menos que a blockchain completa seja baixada. Durante esse tempo, o cliente irá exibir o texto "não sincronizado" próximo ao saldo da conta e mostrará "Sincronizando" no rodapé. Certifique-se de que você tenha espaço em disco, conexão rápida à internet e tempo suficientes para completar a sincronização inicial.

Compilando o Bitcoin Core a partir do Código-Fonte

Para desenvolvedores, ainda há a opção de baixar o código-fonte completo como um arquivo ZIP ou clonar a fonte oficial do repositório do GitHub. Em GitHub bitcoin page, selecione Download ZIP na barra lateral. Você também pode usar a linha de comando git para criar uma cópia local do código fonte em seu sistema. No exemplo a seguir, nós estamo clonando o código fonte de uma linha de comando do tipo Unix, em Linux ou MaC OS:

```
$ git clone https://github.com/bitcoin/bitcoin.git
Clonando para 'bitcoin'...
remote: Contando objetos: 31864, concluído.
remote: Comprimindo objetos: 100% (12007/12007), concluído.
remote: Total 31864 (delta 24480), re-utilizados 26530 (delta 19621)
Recebendo objetos: 100% (31864/31864), 18.47 MiB | 119 KiB/s, concluído.
Resolvendo deltas: 100% (24480/24480), concluído.
$
```

TIP As instruções e o output resultante podem variar de versão para versão. Siga os passos da documentação que acompanha o código mesmo que eles sejam diferentes das instruções que você vê aqui, e não se surpreenda se o output exibido em sua tela seja levemente diferente dos exemplos desse livro.

Quando a operação de git clone completar, você terá uma cópia local completa do repositório do código-conte na sua pasta *bitcoin*. Modifique essa pasta ao digitar cd bitcoin no prompt:

```
$ cd bitcoin
```

Por padrão, a cópia local será sincronizada com o código mais recente, que pode ser uma versão beta ou instável do bitcoin. Antes de compilar o código, selecione a versão específica ao checar uma *tag* de lançamento . Isso irá sincronizar a cópia local com um snapshot específico do respositório do código identificado por uma tag palavra-chave. As tags são usadas pelos desenvolvedores para marcar lançamentos específicos do código através de um número de versão. Primeiro, para encontrar as tags disponíveis, nós usaremos o comando git tag:

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

[... muitas outras tags ...]

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

A lista de tags exibe todas as versões do bitcoin já lançadas. Por convenção, *release candidates*, são

planejados para testes e contém o sufixo "rc". Versões estáveis que possam ser executadas em sistemas de produção não possuem sufixo. Da lista existente, selecione a versão mais recente, que até este momento era v0.10.2. Para sincronizar o código local com esta versão, use o comando git checkout.

```
$ git checkout v0.9.0rc1
Note: checking out 'v0.9.0rc1'.

HEAD is now at 15ec451... Merge pull request #3605
$
```

O código-fonte inclui uma documentação, que pode ser encontrada em vários arquivos. Veja a documentação principal localizada em *README.md* na pasta bitcoin ao digitar more README.md no prompt e usando a barra de espaço para ler a próxima página. Nesse capítulo, iremos fazer o build do cliente de bitcoin na linha de comando, também conhecido como bitcoind no Linux. Veja as instruções para compilar o cliente bitcoind em linha de comando na sua plataforma ao digitar more doc/build-unix.md. Instruções alternativas para Mac OS X e Windows podem ser encontradas na pasta *doc*, como *build-osx.md* ou *build-msw.md*, respectivamente.

Analise cuidadosamente os pré-requisitos da versão, presentes na primeira parte da documentação do mesmo. Estas são as bibliotecas que devem estar presentes em seu sistema antes que você possa iniciar a compilação do bitcoin. Se estes pré-requisitos estiverem ausentes, o processo irá falhar. Logo, você pode instalá-los e então continuar o processo de compilação de onde você parou. Assumindo que os pré-requisitos estão instalados, você inicia o processo de compilação, gerando um conjunto de scripts de construção que utilizam o script *autogen.sh*.

TIP

O processo de build do Bitcoin Core foi modificado para usar o sistema autogen/configure/make a partir da versão 0.9. As versões mais antigas usam um Makefile simples e funcionam um pouco diferente do exemplo demonstrado a seguir. Siga as instruções para a versão que você quer compilar. O autogen/configure/make introduzido na 0.0 provavelmente será o sistema de build usado para todas as versões futuras do código e é o sistema demonstrado nos exemplos a seguir.

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

O script *autogen.sh* cria um conjunto de scripts de configuração automática que irão interrogar seu sistema a descobrir as configurações corretas e garantir que você tenha todas as bibliotecas necessárias para compilar o código. O mais importante desses é o script configure que oferece várias

opções diferentes para customizar o processo de build. Digite ./configure --help para ver as várias opções:

```
$ ./configure --help

`configure' configura o Bitcoin Core 0.9.0 para se adaptar a muitos tipos de sistemas.

Uso: ./configure [OPÇÃO]... [VAR=VALOR]...

Para atribuir as variáveis de ambiente (ex.: CC, CFLAGS...), especifique-as como
VAR=VALOR. Consulte abaixo descrições de algumas variáveis úteis.

As opções padrão estão especificadas em parênteses.

Configuração:
  -h, --help exibe essa ajuda e sai
      --help=short exibe opções específicas para esse pacote
      --help=recursive exibe a ajuda curta (short) de todos os pacotes incluídos
  -V, --version exibe as informações de versão e sai

[... muitas outras opções e variáveis são exibidas abaixo ...]

Funções opcionais:
  --disable-option-checking  ignore unrecognized --enable/--with options
  --disable-FEATURE não incluir FEATURE (mesmo que --enable-FEATURE=no)
  --enable-FEATURE[=ARG]  incluir FEATURE [ARG=yes]

[... mais opções ...]

Use essas variáveis para sobrescrever as escolhas feitas pelo `configure' ou para ajudar
a encontrar livrarias e programas com nomes/localizações não-padrões.

Informe bugs para <info@bitcoin.org>.

$
```

O script configure permite que você habilite ou desabilite certas funções do bitcoind através do uso das flags --enable-FEATURE e --disable-FEATURE, onde FEATURE é substituída pelo nome de uma função, como listado no output do help. Nesse capítulo, nós iremos construir o cliente bitocind com todas as funções padrões. Nós não iremos usar as flags de configurações, mas você deveria revisá-las para entender quais funções opcionais fazem parte do cliente. A seguir, rode o script configure para automaticamente descobrir todas as bibliotecas necessárias e criar um script de build customizado para o seu sistema:

```
$ ./configure
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes

[... muitos outros recursos do sistema são testados ...]

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

Se tudo der certo, o comando configure terminará criando os scripts de build customizados que nos permitirão compilar o bitcoind. Se houver qualquer livrarias faltando ou erros, o comando configure irá terminar com um erro ao invés de criar os script de build. Se um erro ocorrer, mais provavelmente será devido a uma biblioteca faltando ou incompatível. Revise a documentação do build novamente e certifique-se que você instalou os pré-requisitos que estão em falta. Então execute o configure novamente e veja se isso corrige o erro. Em seguida, você irá compilar o código fonte, um processo que pode levar até uma hora para ser completado. Durante o processo de compilação você deverá ver um output a cada poucos segundos ou minutos, ou um erro se algo der errado. O processo de compilação pode ser reiniciado a qualquer momento se for interrompido. Digite make para iniciar a compilação:

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

[... muitas outras mensagens de compilação ...]

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

Se tudo ocorrer dentro do esperado, o bitcoind agora está compilado. O passo final é instalar o executável do bitcoind no caminho do sistema usando o comando make:

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

Você pode confirmar que o bitcoin está corretamente instalado ao perguntar ao sistema pelo caminho dos dois executáveis, como demonstrado a seguir:

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

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

A instalação padrão do bitcoind o salva em */usr/local/bin*. Quando você rodar o bitcoind pela primeira vez, ele irá lembrá-lo para criar um arquivo de configuração com uma senha forte para a interface JSON-RPC. Execute o bitcoind ao digitar bitcoind no terminal:

```
$ bitcoind
Erro: Para usar a opção "-server", você precisa definir uma rpcpassword no arquivo de
configuração:
/home/ubuntu/.bitcoin/bitcoin.conf
É recomendado que você use a seguinte senha aleatória:
rpcuser=bitcoinrpc
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
(você não precisa se lembrar dessa senha)
O usuário e senha NÃO DEVEM ser iguais.
Se o arquivo não existir, crie um com permissões de arquivo owner-somente leitura
Também recomenda-se definir um alertnotify, para que você seja notificado sobre
problemas;
por exemplo: alertnotify=echo %s | mail -s "Alerta Bitcoin" admin@foo.com
```

Edite o arquivo de configuração em seu editor de preferência e defina os parâmetros, substituindo a senha por uma senha forte como recomendado pelo bitcoind. *Não* use a senha mostrada aqui. Crie um arquivo no interior da pasta *.bitcoin* de maneira que ela fique nomeada como *.bitcoin/bitcoin.conf* e insira um usuário e senha:

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

Enquanto você estiver editando esse arquivo de configuração, você pode querer definir algumas outras opções, como a txindex (ver Índice do Banco de Dados de Transações e a Opção txindex). Para uma listagem completa das opções disponíveis, digite bitcoind --help.

Agora, execute o cliente Bitcoin Core. Ao ser executado pela primeira vez, ele irá reconstruir a blockchain do bitcoin ao fazer o download de todos os blocos. Ela é um arquivo de vários gigabytes e levará em média dois dias para ser completamente baixada. Você pode diminuir o tempo de inicialização da blockchain ao fazer o download de uma cópia parcial dela usando um cliente BitTorrent a partir de SourceForge.

Execute o bitcoind em segundo plano com a opção -daemon:range="endofrange", startref="ix_ch03-

asciidoc3")

```
$ bitcoind -daemon

Bitcoin version v0.9.0rc1-beta (2014-01-31 09:30:15 +0100) _(N.T. A versão será diferente
de acordo com a versão baixada no GitHub)_
Using OpenSSL version OpenSSL 1.0.1c 10 May 2012
Default data directory /home/bitcoin/.bitcoin
Using data directory /bitcoin/
Using at most 4 connections (1024 file descriptors available)
init message: Verifying wallet...
dbenv.open LogDir=/bitcoin/database ErrorFile=/bitcoin/db.log
Bound to [::]:8333
Bound to 0.0.0.0:8333
init message: Loading block index...
Opening LevelDB in /bitcoin/blocks/index
Opened LevelDB successfully
Opening LevelDB in /bitcoin/chainstate
Opened LevelDB successfully

[... mais mensagens de inicialização ...]
```

Usando a API JSON-RPC do Bitcoin Core a partir da Linha de Comando

O cliente Bitcoin Core implementa uma interface JSON-RPC que também pode ser acessada ao se utilizar o ajudante da linha de comando bitcoin-cli. A linha de comando nos permite experimentar interativamente com as capacidades que também estão disponíveis programaticamente através da API. Para iniciar, invoque o comando help para ver uma lista dos comandos bitcoin RPC disponíveis:

```
$ bitcoin-cli help
addmultisigaddress nrequired ["key",...] ( "account" )
addnode "node" "add|remove|onetry"
backupwallet "destination"
createmultisig nrequired ["key",...]
createrawtransaction [{"txid":"id","vout":n},...] {"address":amount,...}
decoderawtransaction "hexstring"
decodescript "hex"
dumpprivkey "bitcoinaddress"
dumpwallet "filename"
getaccount "bitcoinaddress"
getaccountaddress "account"
getaddednodeinfo dns ( "node" )
getaddressesbyaccount "account"
```

```
getbalance ( "account" minconf )
getbestblockhash
getblock "hash" ( verbose )
getblockchaininfo
getblockcount
getblockhash index
getblocktemplate ( "jsonrequestobject" )
getconnectioncount
getdifficulty
getgenerate
gethashespersec
getinfo
getmininginfo
getnettotals
getnetworkhashps ( blocks height )
getnetworkinfo
getnewaddress ( "account" )
getpeerinfo
getrawchangeaddress
getrawmempool ( verbose )
getrawtransaction "txid" ( verbose )
getreceivedbyaccount "account" ( minconf )
getreceivedbyaddress "bitcoinaddress" ( minconf )
gettransaction "txid"
gettxout "txid" n ( includemempool )
gettxoutsetinfo
getunconfirmedbalance
getwalletinfo
getwork ( "data" )
help ( "command" )
importprivkey "bitcoinprivkey" ( "label" rescan )
importwallet "filename"
keypoolrefill ( newsize )
listaccounts ( minconf )
listaddressgroupings
listlockunspent
listreceivedbyaccount ( minconf includeempty )
listreceivedbyaddress ( minconf includeempty )
listsinceblock ( "blockhash" target-confirmations )
listtransactions ( "account" count from )
listunspent ( minconf maxconf  ["address",...] )
lockunspent unlock [{"txid":"txid","vout":n},...]
move "fromaccount" "toaccount" amount ( minconf "comment" )
ping
sendfrom "fromaccount" "tobitcoinaddress" amount ( minconf "comment" "comment-to" )
sendmany "fromaccount" {"address":amount,...} ( minconf "comment" )
sendrawtransaction "hexstring" ( allowhighfees )
sendtoaddress "bitcoinaddress" amount ( "comment" "comment-to" )
```

```
setaccount "bitcoinaddress" "account"
setgenerate generate ( genproclimit )
settxfee amount
signmessage "bitcoinaddress" "message"
signrawtransaction "hexstring" (
[{"txid":"id","vout":n,"scriptPubKey":"hex","redeemScript":"hex"},...]
["privatekey1",...] sighashtype )
stop
submitblock "hexdata" ( "jsonparametersobject" )
validateaddress "bitcoinaddress"
verifychain ( checklevel numblocks )
verifymessage "bitcoinaddress" "signature" "message"
walletlock
walletpassphrase "passphrase" timeout
walletpassphrasechange "oldpassphrase" "newpassphrase"
```

Obtendo Informações do Status do Cliente Bitcoin Core

Comandos: getinfo

O comando RPC getinfo do Bitcoin exibe informações básicas sobre o estado do nó da rede bitcoin, da carteira e do banco de dados da blockchain. Use bitcoin-cli para rodá-lo:

```
$ bitcoin-cli getinfo
```

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

Os dados retornam em JavaScript Object Notation (JSON), um formato que pode ser facilmente "consumido" por todas as linguagens de programação mas que também é bastante fácil de ler por uma

pessoa. Entre esses dados nós vemos os números de versão do cliente de software (90000), protocolo (70002) e carteira (60000) bitcoin. Nós vemos o saldo atual contido na carteira, que é zero. Nós vemos a altura atual do bloco, nos mostrando quantos blocos esse cliente conhece (286216). Nós também vemos várias estatísticas sobre a rede bitcoin e as configurações relacionadas a esse cliente. Nós iremos explorar essas configurações em mais detalhes no resto desse capítulo.

TIP	Levará algum tempo, talvez mais de um dia, para o cliente bitcoind "alcançar" a altura da blockchain atual ao baixar os blocos de outros clientes bitcoin. Você pode verificar o seu progresso usando getinfo para ver o número de blocos conhecidos.

Configuração e Criptografia da Carteira

Comandos: encryptwallet, walletpassphrase

Antes de você proceder com a criação de chaves e outros comandos, você deve primeiro criptografar a carteira com uma chave. Para esse exemplo, você irá utilizar o comando encryptwallet com a senha "foo". Obviamente, substitua "foo" por uma senha forte e complexa!

```
$ bitcoin-cli encryptwallet foo
carteira criptografada; Parando servidor bitccoin, reinicie para rodar com carteira
criptografada. A pool de chaves foi liberada, você precisa fazer um novo backup.
$
```

Você pode verificar se a carteira foi criptografada ao executar getinfo novamente. Dessa vez você irá perceber uma nova entrada chamada unlocked_until. Ela é um contador mostrando por quanto tempo a senha de descriptografia da carteira será armazenada na memória, mantendo a carteira desbloqueada. Inicialmente ela será definida como zero, significando que a carteira está bloqueada:

```
$ bitcoin-cli getinfo
```

```
{
    "version" : 90000,

#[... outras informações...]

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

Para desbloquear a carteira, use o comando walletpassphrase, que carrega dois parâmetros—a senha e o número de segundos até que a carteira seja bloqueada novamente de maneira automático (um timer regressivo):

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

Você pode confirmar que a carteira está desbloqueada e ver o tempo restante ao executar getinfo novamente:

```
$ bitcoin-cli getinfo
```

```
{
    "version" : 90000,

#[... outras informações ...]

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

Backup da Carteira, Dump em texto-puro e Restauração

Comandos: backupwallet, importwallet, dumpwallet

A seguir nós iremos criar um arquivo de backup da carteira e restauraremos a carteira a partir desse arquivo. Use o comando backupwallet para fazer o backup, fornecendo o nome do arquivo como parâmetro. Aqui nós iremos fazer backup da carteira para o arquivo *wallet.backup*:

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

Agora, para recuperar o arquivo de backup, utilize o comando importwallet. Se sua carteira estiver bloqueada, você precisará desbloquea-la antes (ver walletpassphrase na seção anterior) para importar o arquivo de backup:

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

O comando dumpwallet pode ser usado para fazer um dump da carteira em um arquivo de texto de leitura fácil:

```
$ bitcoin-cli dumpwallet wallet.txt
$ more wallet.txt
# Dump da carteira criado por Bitcoin v0.9.0rc1-beta (2014-01-31 09:30:15 +0100)
# * Criado em 2014-02- 8dT20:34:55Z
# * O melhor bloco na época do backup era 286234
(0000000000000000f74f0bc9d3c186267bc45c7b91c49a0386538ac24c0d3a44),
# minerado em 2014-02- 8dT20:24:01Z

KzTg2wn6Z8s7ai5NA9MVX4vstHRsqP26QKJCzLg4JvFrp6mMaGB9 2013-07- 4dT04:30:27Z change=1 #
addr=16pJ6XkwSQv5ma5FSXMRPaXEYrENCEg47F
Kz3dVz7R6mUpXzdZy4gJEVZxXJwA15f198eVui4CUivXotzLBDKY 2013-07- 4dT04:30:27Z change=1 #
addr=17oJds8kaN8LP8kuAkWTco6ZM7BGXFC3gk
[... muitas outras chaves ...]

$
```

Endereços da Carteira e Recebendo Transações

Comandos: getnewaddress, getreceivedbyaddress, listtransactions, getaddressesbyaccount, getbalance

O cliente referência do bitcoin mantém um pool de endereços, o tamanho do qual é exibido através do keypoolsize quando você usa o comando getinfo. Esses endereços são gerados automaticamente e podem ser utilizados como endereços públicos para receber pagamentos ou como endereços de troco. Para gerar um desses endereços, use o comando getnewaddress:

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

Agora nós podemos usar esse endereço para enviar uma pequena quantidade de bitcoins para nossa carteira bitcoind a partir de uma carteira externa (assumindo que você tem alguns bitcoins em uma exchange, carteira web ou outra carteira bitcoind em outro lugar). Para esse exemplo, nós enviaremos 50 milibits (0,050 bitcoin) para o endereço anterior.

Agora nós podemos requisitar o cliente bitcoind para a quantia recebida por esse endereço, e especificar quantas confirmações são necessárias antes que uma quantia seja contabilizada no saldo. Para esse exemplo, nós iremos especificar zero confirmações. Alguns segundos após enviar o bitcoin de outra carteira, nós iremos ver isso refletido na carteira. Nós usamos getreceivedbyaddress com o endereço e o número de confirmações definido para zero (0):

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

Se nós omitirmos o zero do final desse comando, nós iremos ver somente as quantias que tiveram pelo

menos minconf confirmações, onde minconf é o parâmetro para o número mínimo de confirmações antes que uma transação seja listada no saldo. O parâmetro minconf é especificado no arquivo de configuração do bitcoind. Como a transação enviando esse bitcoin só foi enviada nos últimos poucos segundos, ela ainda não foi confirmada e portanto nós iremos ver ele listas um saldo zero:

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

As transações recebidas pela carteira também podem ser exibidas usando o comando listtransactions:

```
$ bitcoin-cli listtransactions
```

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

Nós podemos listar todos os endereços contidos na carteira usando o comando getaddressesbyaccount:

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

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

Finalmente, o comando getbalance mostrará o saldo total da carteira, somando todas as transações confirmadas com pelo menos minconf confirmações:

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

TIP	Se a transação ainda não foi confirmada, o saldo que getbalance retornará será de zero. A opção de configuração "minconf" determina o número mínimo de confirmações que são necessárias antes de uma transação aparecer no saldo.

Explorando e Decodificando as Transações

Comandos: gettransaction, getrawtransaction, decoderawtransaction

Agora nós iremos explorar a transação que chega e que foi listada previamente usando o comando gettransaction. Nós podemos coletar a transação através de seu hash de transação, mostrado em txid anteriormente, com o comando gettransaction:

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

TIP Os IDs de transações não são "oficiais" até que a transação tenha sido confirmada. A ausência de um hash de transação na blockchain não significa que a transação não foi processada. Isso é conhecido como "maleabilidade da transação", porque hashes de transação podem ser modificados antes da confirmação em um bloco. Após a confirmação, o txid é imutável e oficial.

A forma de transação mostrada com o comando gettransaction é a forma simplicada. Para adquirir o código de transação completo e decodificá-lo, nós iremos usar dois comandos: getrawtransaction e decoderawtransaction. Primeiro, o getrawtransaction recebe o *hash da transação (txid)* como um parâmetro e retorna a transação completa como uma string hexadecimal "raw", exatamente como ela existe na rede bitcoin:

Para decodificar essa string hexadecimal, use o comando decoderawtransaction. Copie e cole o hexadecimal como o primeiro parâmetro de decoderawtransaction para obter o conteúdo completo interpretado como uma estrutura de dados JSON (por questões de formatação a string hexadecimal é encurtada no seguinte exemplo):

A decodificação da transação mostra todos os componentes desta transação, incluindo os inputs e outputs da transação. Nesse caso nós vemos que a transação que creditou nosso novo endereço com 50 milibits usou um input e gerou dois outputs. O input para essa transação foi o output de uma transação previamente confirmada (mostrada como o vin txid iniciando com d3c7). Os dois outputs correspondem ao crédito de 50 milibit e a um output com o troco de volta ao remetente.

Podemos explorar mais a fundo a blockchain ao examinarmos a transação anterior referenciada pelo seu txid nesta transação usando os mesmos comandos (ex: gettransaction). Saltando de transação em transação nós podemos seguir uma corrente de transações de volta na medida em que as moedas são transmitidas entre os endereços de seus donos.

Assim que a transação que recebemos é confirmada ao ser incluída em um bloco, o comando

gettransaction irá retornar informações adicionais, mostrando o *hash do bloco (identificador)* no qual a transação foi incluída:

Aqui, nós vemos a nova informação nas entradas blockhash (o hash do bloco no qual a transação foi incluída), e blockindex com valor 18 (indicando que nossa transação foi a 18ª transação naquele bloco).

Índice do Banco de Dados de Transações e a Opção txindex

Por padrão, o Bitcoin Core constrói um banco de dados contendo *apenas* as transações relacionadas à carteira do usuário. Se você quiser ter acesso a *qualquer* transação com comandos como gettransaction, você tem que configurar o Bitcoin Core para construir um índice completo de transações, que pode ser realizado com a opção txindex. Defina txindex=1 no arquivo de configuração do Bitcoin Core (geralmente encontrado em seu diretório sob *.bitcoin/bitcoin.conf*). Assim que mudar esse parâmetro, você precisará reiniciar o bitcoind e esperar que ele reconstrua o índice.

Explorando os Blocos

Comandos: getblock, getblockhash

Agora que nós sabemos em qual bloco nossa transação foi incluídas, nós podemos fazer uma requisição nesse bloco. Nós usaremos o comando getblock com o hash do bloco como parâmetro:

O bloco contém 367 transações e como você pode ver, a 18ª transação listada (9ca8f9...) é o txid de um dos 50 milibits creditados em nosso endereço. A entrada altura nos informa que esse é o 286.384º bloco na blockchain.

Nós também podemos adquirir um bloco através de sua altura de bloco usando o comando getblockhash, que recebe a altura do bloco como parâmetro e retorna o hash do bloco para aquele bloco:

Aqui, nós solicitamos o hash do bloco do "bloco gênese", o primeiro bloco minerado por Satoshi Nakamoto, na altura zero. A solicitação desse bloco mostra:

Os comandos getblock, getblockhash e gettransaction podem ser usados para explorar o banco de dados da blockchain, programaticamente.

Criando, Assinando e Enviando Transações Baseadas em Outputs Não-gastos

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

As transações do Bitcoin são baseadas no conceito de gastar "outputs", que são o resultado de transações prévias, para criar uma corrente de transações que transfere a posse de um endereço para outro endereço. Nossa carteira agora recebeu uma transação que designou um desses outputs para

nosso endereço. Uma vez confirmada, nós podemos gastar esse output.

Primeiro, nós utilizaremos o comando listunspent para mostrar todos os outputs *confirmados* não-gastos em nossa carteira:

```
$ bitcoin-cli listunspent
```

Nós vemos que a transação 9ca8f9... criou um output (com o índice vout 0) designado para o endereço 1hvzSo... para a quantidade de 50 milibits, que até esse momento já recebeu sete confirmações. As transações usam outputs criados anteriormente como seus inputs ao usá-los como referência através do txid e índice vout anterior. Nós agora iremos criar uma transação que irá gastar o vout zero da txid 9ca8f9... como seu input e designá-la como um novo output que envia um valor para um novo endereço.

Primeiramente, vamos olhar ao output específico em maiores detalhes. Nós usamos gettxout para obter os detalhes desse output não-gasto. Os outputs de transação são sempre referenciados pelo txid e vout, e esses são os parâmetros que nós passamos para o gettxout:

O que nós vemos aqui é o output que designou 50 milibits para nosso endereço 1hvz.... Para gastar esse output nós iremos criar uma nova transação. Primeiro, vamos fazer um endereço para o qual nós iremos enviar o dinheiro:

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

Nós enviaremos 25 milibits para o novo endereço 1LnfTn... que nós acabamos de criar em nossa carteira. Em nossa nova transação, nós iremos gastar o output de 50 milibit e enviar 25 milibits para esse novo endereço. Como nós temos que gastar *todo* o output da transação anterior, nós também devemos gerar algum troco. Nós iremos gerar o troco de volta para o endereço 1hvz..., enviando o troco de volta para o endereço do qual o valor se originou. Finalmente, nós também teremos que pagar uma taxa para essa transação. Para pagar a taxa, nós iremos reduzir o output do troco em 0,5 milibits, e retornar 24,5 milibits em troco. A diferença entre a soma dos novos outputs (25 mBTC + 24,5 mBTC = 49,5 mBTC) e o input (50 mBTC) será coletada como uma taxa de transação pelos mineradores.

Nós usaremos createrawtransaction para criar essa transação. Como parâmetros para createnewtransaction nós forneceremos o input da transação (o output de 50 milibits não-gastos de nossa transação confirmada) e os dois outputs de transações (dinheiro enviado para o novo endereço e troco enviado de volta para o endereço anterior):

O comando createrawtransaction produz uma string hexadecimal raw que codifica os detalhes da transação que nós fornecemos. Vamos confirmar que tudo está correto ao decodificar essa string raw usando o comando decoderawtransaction:

Isso parece correto! Nossa nova transação "consome" o output não gasto de nossa transação

confirmada e então gasta-o em dois outputs, um de 25 milibits para nosso novo endereço e outro de 24,5 milibits como troco de volta para o endereço original. A diferença de 0,5 milibits representa a taxa de transação e será creditada ao minerador que encontrar o bloco que inclui nossa transação.

Como você pode perceber, a transação contém um scriptSig vazio porque ainda não foi assinada. Sem uma assinatura, a transação não tem sentido; nós ainda não provamos que nós *possuímos* o endereço que contém o output não-gasto. Ao assinar, nós destravamos o bloqueio no output e provamos que nós somos donos desse output e que podemos gastá-lo. Nós usaremos o comando signrawtransaction para assinar a transação. Esse comando usa a string hexadecimal da transação raw como parâmetro:

TIP	Um carteira criptografada deve ser desbloqueada antes que uma transação seja assinada, pois a assinatura exige acesso às chaves secretas contidas no interior da carteira.

O comando signrawtransaction retorna outra transação raw codificada em hex. Nós decodificaremos ela para ver o que mudou, com o comando decoderawtransaction:

Agora, o input usado na transação contém um scriptSig, que é uma assinatura digital provando a posse do endereço 1hvz... e removendo a trava no output de maneira que ele possa ser gasto. A assinatura faz com que essa transação seja verificável por qualquer nó na rede bitcoin.

Agora está na hora de enviarmos a transação recém-criada para a rede. Nós faremos isso através do comando sendrawtransaction, que recebe a string hexadecimal raw produzida pelo signrawtransaction. Essa é a mesma string que nós recém decodificamos:

O comando sendrawtransaction retorna um *hash de transação (txid)* assim que a transação é enviada para a rede. Nós podemos agora consultar esse ID da transação com gettransaction:

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


Como anteriormente, nós também podemos examinar isso em maiores detalhes usando os comandos getrawtransaction e decodetransaction. Esses comandos irão retornar exatamente a mesma string hexadecimal que nós produzimos e codificamos anteriormente, logo antes de enviá-la à rede.

Clientes Alternativos, Bibliotecas e Toolkits

Além do cliente de referência (bitcoind), outros clientes e bibliotecas podem ser usados para interagir com a rede bitcoin e as estruturas de dados. Eles são implementados em várias linguagens de

programação, oferecendo aos programadores interfaces nativas em suas próprias linguagens.

Implementações alternativas:

libbitcoin

Bitcoin Cross-Platform C++ Development Toolkit

bitcoin explorer

Ferramenta Linha de comando do Bitcoin

bitcoin server

Bitcoin Full Node e Query Server

bitcoinj

Uma biblioteca Java de um cliente full-node

btcd

Um cliente bitcoin full-node em linguagem Go

Bits of Proof (BOP)

Uma implementação Java enterprise-class do bitcoin

picocoin

Uma implementação C de uma biblioteca de cliente lightweight para bitcoin

pybitcointools

Uma biblioteca Python para bitcoin

pycoin

Outra biblioteca bitcoin em Python

Muitas outras bibliotecas existem em várias outras linguagens de programação e muitas mais são criadas a todo momento.

Libbitcoin e Bitcoin Explorer

A biblioteca libbitcoin é um kit de ferramentas de desenvolvimento C++ de plataforma cruzada que suporta o nó de servidor completo-libbitcoin e a ferramenta de linha de comando Bitcoin Explorer (bx).

Os commandos bx oferecem muitas das mesmas capacidades que os comandos do cliente bitcoind que foram ilustrados nesse capítulos. Os comandos bx também oferecem algumas ferramentas de administração e de manipulação de chaves que não são oferecidas pelo bitcoind, incluindo chaves determinísticas tipo 2 e codificação mnemônica de chave, além de endereço e pagamento camuflados (stealth), e suporte a query.

Instalando o Bitcoin Explorer

Para usar o Bitcoin Explorer, simplesmente faça o download do executável assinado para seu sistema operacional. Os builds estão disponíveis para a mainnet e testnet para Linux, OS X e Windows.

Digite bx sem nenhum parâmetro para mostrar a lista de todos os comandos disponíveis (veja [appdx_bx]).

O Bitcoin Explorer também fornece um instalador para fazer builds a partir de código-fontes em Linux e OS X, assim como projetos VIsual Studio para WIndows. Também pode-se fazer build manual dos código-fontes usando-se Autotools. Esses também instalam a dependência de biblioteca libbitcoin.

TIP	O Bitcoin Explorer oferece muitos comandos úteis para codificação e decodificação de endereços, e conversão para e de diferentes formatos e representações. Use-os para explorar os vários formatos como Base16 (hex), Base58, Base58Check, Base64, etc.

Instalando a Libbitcoin

A biblioteca libbitcoin fornece um instalador para building fazer builds a partir de códigos-fonte em Linux e OS X, assim como em projetos Visual Studio para Windows. Também é possível fazer manualmente o build dos códigos-fonte usando-se Autotools.

TIP	O instalador do Bitcoin Explorer instala o bx e a biblioteca libbitcoin, se você fez o build do bx a partir das fontes, você pode pular essa etapa.

pycoin

A biblioteca Python *pycoin*, originalmente escrita e mantida por Richard Kiss, é uma livraria baseada em Python que suporta a manipulação de chaves e transações bitcoin, e até mesmo suporta a linguagem de script suficiente para lidar adequadamente com transações não-padrões.

A biblioteca pycoin suporta tanto o Python 2 (2.7.x) quanto o Python 3 (após 3.3), e vem com algumas utlidades de linhas de comando, ku e tx. Para instalar o pycoin 0.42 sob o Python 3 em um ambiente virtual (venv), utilize o seguinte:

```
$ python3 -m venv /tmp/pycoin
$ . /tmp/pycoin/bin/activate
$ pip install pycoin==0.42
Baixando/descompactando pycoin==0.42
  Baixando o pycoin-0.42.tar.gz (66kB): 66kB baixados
  Executando o setup.py (path:/tmp/pycoin/build/pycoin/setup.py) egg_info para o
pacote pycoin

Instalando pacotes coletados: pycoin
  Executando instalação setup.py para pycoin

    Instalando tx script para /tmp/pycoin/bin
    Instalando cache_tx script para /tmp/pycoin/bin
    Instalando bu script para /tmp/pycoin/bin
    Instalandog fetch_unspent script para /tmp/pycoin/bin
    Instalando block script para /tmp/pycoin/bin
    Instalando spend script para /tmp/pycoin/bin
    Instalando ku script para /tmp/pycoin/bin
    Instalando genwallet script para /tmp/pycoin/bin
pycoin instalado com sucesso
Limpando...
$
```

Aqui está um exemplo de script Python para adquirir e gastar alguns bitcoins usando a biblioteca pycoin:

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

from pycoin.key import Key

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

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

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

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

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

dst_address = get_address("destination")

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

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

Para exemplos usando os utilitários de linha de comando ku e tx, veja [appdxbitcoinimpproposals].

btcd

btcd é uma implementação de bitcoin full-node programado em Go. Ela atualmente baixa, valida e serve a blockchain usando as regras exatas (incluindo bugs) para aceitação de blocos como a implementação de referência, bitcoind. Ela também propriamente transmite blocos recém-minerados, mantém um pool de transações e transmite transações individuais que ainda não foram incluídas em um bloco. Ela garante que todas as transações individuais admitidas à pool sigam as regras exigidas e também inclui uma vas maioria de verificações mais estreitas que filtram as transações baseadas nas necessidades dos mineradores (transações "padrão")

Uma diferença importante entre o btcd e o bitcoind é que o btcd não inclui a funcionalidade de carteira, e isso foi uma decisão de projeto muito intencional. Isso significa que você não pode fazer ou receber pagamentos diretamente com o btcd. Essa funcionalidade é fornecida pelos projetos btcwallet e btcgui, os quais estão ambos sob desenvolvimento ativo. Outras diferenças notáveis entre o btcd e o bitcoind incluem suporte no btcd para requisições HTTP POST (como no bitcoind) e os Websockets preferidos, e o fato de que as conexões RPC do btcd são habilitadas a TLS por padrão.

Instalando btcd

Para instalar o btcd para Windows, baixe e execute o arquivo msi disponível em GitHub, ou execute a seguinte linha de comando no Linux, assumindo que você já tenha instalado a Go language:

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

Para atualizar o btcd para a última versão, simplesmente execute:

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

Controlando o btcd

btcd tem várias opções de configurações, que podem ser vistas ao executar:

```
$ btcd --help
```

O btcd já inclui alguns itens como o btcctl, que é um utilitário de linha de comando que pode ser usado tanto para controlar e requisitar o btcd através de RPC. O btcd não habilita seu servidor RPC por padrão; você deve configurar pelo menos tanto o usuário quanto a senha do RPC nos seguintes arquivos de configuração:

- *btcd.conf*:

```
[Opções da Aplicação]
rpcuser=myuser
rpcpass=SomeDecentp4ssw0rd
```

- *btcctl.conf*:

```
[Opções da Aplicação]
rpcuser=myuser
rpcpass=SomeDecentp4ssw0rd
```

Ou se você quiser sobrescrever os arquivos de configuração a partir da linha de comando:

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

Para uma lista das opções disponíveis, execute o seguinte:

```
$ btcctl --help
```

Chaves, Endereços, Carteiras

Introdução

A posse do bitcoin é estabelecida através de *chaves digitais*, *endereços bitcoin* e *assinaturas digitais*. Na realidade, as chaves digitais não são armazenadas na rede, mas ao invés disso, são criadas e armazenadas pelo usuário em um arquivo, um banco de dados simples, chamado de *wallet* (carteira). As chaves digitais em uma carteira de um usuário são completamente independentes do protocolo bitcoin e podem ser geradas e gerenciadas pelo software de carteira do usuário sem qualquer referência à blockchain ou acesso à internet. As chaves permitem muitas das interessantes propriedades do bitcoin, incluindo confiança descentralizada e controle, atestação de posse e o modelo de segurança por prova criptográfica.

Toda transação de bitcoin requer uma assinatura válida para ser incluída na blockchain, as quais só podem ser geradas com chaves digitais válidas; portanto, qualquer um com uma cópia destas chaves tem o controle sobre os bitcoins daquela conta. Chaves são apresentadas em pares que consistem em uma chave privada (secreta) e uma chave pública. Imagine que a Chave Pública é similar ao número de uma conta bancária e a Chave Privada similar a um PIN secreto ou uma assinatura em um cheque que provê controle sobre a conta. Estas chaves digitais são raramente vistas pelos usuários do bitcoin. Para a maioria, elas são armazenadas dentro de arquivos de carteiras e gerenciadas por um software de carteira bitcoin.

Na porção de pagamento de uma transação bitcoin, a chave pública do destinatário é representada pela sua impressão digital, denominada *endereço bitcoin*, que é usada da mesma maneira como o nome de um beneficiário em um cheque (ou seja, "Pague à ordem de"). Na maioria dos casos, um endereço bitcoin é gerado a partir de uma chave pública correspondente. No entanto, nem todos os endereços bitcoin representam chaves públicas; eles também podem representar outros beneficiários, tais como scripts, como veremos mais adiante neste capítulo. Desta forma, os endereços bitcoin abstraem o destinatário dos fundos, tornando destinos de transação flexíveis, semelhante a cheques em papel: um único instrumento de pagamento pode ser usado para depósito em contas pessoais, depósito em contas da empresa, pagar contas ou converter em dinheiro. O endereço bitcoin é a única representação das chaves que os usuários vão rotineiramente ver porque esta é a parte que eles precisam compartilhar com o mundo.

Neste capítulo iremos apresentar Carteiras, as quais contém Chaves Criptográficas. Nós iremos aprender como as chaves são geradas, armazenadas e gerenciadas. Iremos revisar os vários formatos de codificação usados para representar chaves privadas e públicas, endereços e endereços baseados em script. Finalmente iremos considerar usos especiais para as chaves: para assinar mensagens, comprovar propriedade, criar endereços exclusivos e carteiras em papel.

Criptografia de Chave Pública e Criptomoeda

A criptografia de chave pública foi inventada nos anos 1970s e é uma base matemática para a segurança da computação e informação.

Desde a invenção da criptografia de chave pública, muitas funções matemáticas adequadas foram descobertas, como exponenciação de números primos e multiplicação de curva elíptica. Essas funções matemáticas são praticamente irreversíveis, significando que elas são fáceis de calcular em uma direção, e inviáveis de serem calculadas na direção oposta. Baseada nessas funções matemáticas, a criptografia permite a criação de segredos digitais e assinaturas digitais que não podem ser esquecidas. O bitcoin usa multiplicação de curva elíptica como base para sua criptografia de chave pública.

No bitcoin, nós usamos criptografia de chave pública para criar um par de chaves que controla o acesso aos bitcoins. O par de chave consiste em uma chave privada e — derivada dessa chave — uma chave pública única. A chave pública é usada para receber os bitcoins, e a chave privada é usada para assinar transações para gastar esses bitcoins.

Existe uma relação matemática entre a chave pública e a privada que permite que a chave privada seja usada para gerar assinaturas nas mensagens. Essa assinatura pode ser validada em relação à chave pública, sem a necessidade de se revelar a chave privada.

Ao gastar bitcoins, o atual dono dos bitcoins apresenta sua chave pública e uma assinatura (diferente a cada vez, mas criada a partir da mesma chave privada) em uma transação para gastar esses bitcoins. Através da apresentação da chave pública e da assinatura, todos na rede bitcoin podem verificar e aceitar a transação como válida, confirmando que a pessoa que está transferindo os bitcoins realmente os possui no momento da transferência.

TIP	Na maioria das implementações de carteira, as chaves privadas e públicas são armazenadas juntas como um *par de chaves*, por conveniência. No entanto, a chave pública pode ser calculada a partir da chave privada, então também é possível se armazenar apenas a chave privada.

Chaves Privada e Pública

Uma carteira bitcoin contém um grupo de pares de chaves, cada um consistindo de uma chave privada e uma chave pública. A chave privada (k) é um número, geralmente escolhido ao acaso. A partir da chave privada, nós usamos multiplicação em curva elíptica, uma função criptográfica de um único sentido, para gerar a chave pública (K). A partir da chave pública (K), nós iremos usar uma função hash criptográfica de um sentido para gerar um endereço bitcoin (A). Nessa seção, nós iniciaremos com a geração da chave privada, analisaremos a matemática de curva elíptica que é usada para torná-la em uma chave pública e, finalmente, a geração do endereço bitcoin a partir da chave pública. A relação entre a chave privada, chave pública e endereço bitcoin está demonstrada em Chave privada, chave pública e endereço bitcoin.

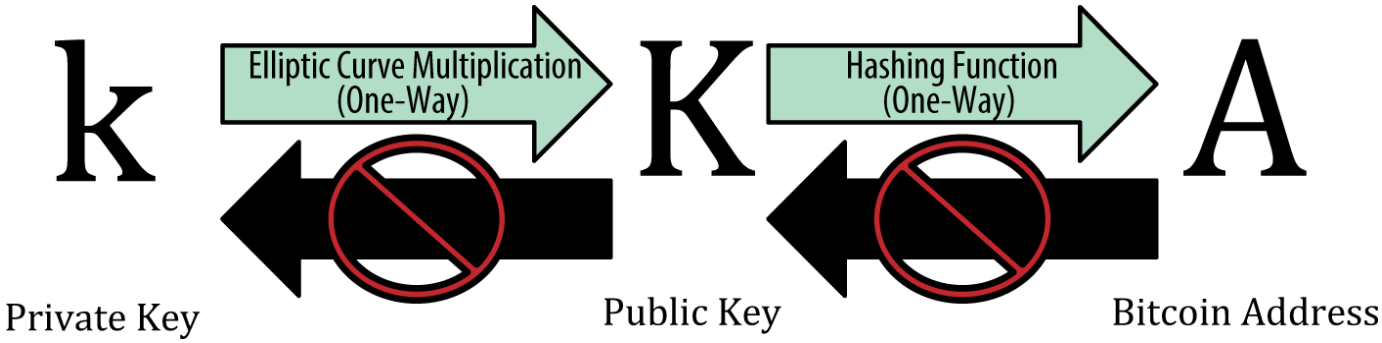


Figure 1. Chave privada, chave pública e endereço bitcoin

Chaves Privadas

Uma chave privada nada mais é do que um número, escolhido ao acaso. A posse e o controle da chave privada é tudo o que o usuário precisa para controlar todos os fundos associados ao endereço bitcoin correspondente. A chave privada é usada para criar assinaturas que são necessárias para se gastar bitcoins ao comprovar a posse dos fundos usados em uma transação. A chave privada deve sempre ser mantida em segredo, pois revelá-la a terceiros é equivalente a fornecê-los o controle sobre todos os bitcoins protegidos por aquela chave. Também deve ser feito backup da chave privada, além de protegê-la de perdas acidentais, pois ao ser perdida a chave não pode ser recuperada, e todos os fundos protegidos por ela também serão perdidos para sempre.

TIP	A chave privada bitcoin é apenas um número. Você pode escolher suas chaves privadas aleatoriamente usando uma moeda, um lápis e um papel: jogue a moeda 256 vezes e você terá os dígitos binários de uma chave privada aleatória que você pode usar em uma carteira bitcoin. A chave pública pode então ser gerada a partir de sua chave privada.

Gerando uma chave privada a partir de um número aleatório

O primeiro e mais importante passo na geração de chaves é encontrar uma fonte segura de entropia, ou aleatoriedade. Criar uma chave bitcoin é essencialmente a mesma coisa que pedir para alguém "Escolha um número entre 1 e 2^{256}". O método exato que você usa para escolher esse número não importa, contanto que não seja previsível ou repetível. O software Bitcoin baseia-se em geradores de números aleatórios do sistema operacional subjacente para produzir 256 bits de entropia (aleatoriedade). Normalmente, o gerador de números aleatórios do sistema operacional é inicializado por uma fonte humana de aleatoriedade, razão pela qual você pode ser convidado para mexer o mouse por alguns segundos. Para quem é paranóico, nada é melhor do que jogar dados e depois anotar os resultados com caneta e papel.

Mais precisamente, a chave privada pode ser qualquer número entre 1 e n - 1, onde n é uma constante ($n = 1{,}158 * 10^{77}$, ligeiramente menor que 2^{256}), definido como a ordem da curva elíptica usada no bitcoin (veja Explicando a Criptografia de Curva Elíptica). Para criar tal chave, nós escolhemos aleatoriamente um número de 256 bits e verificamos se ele é menor do que n - 1. Em termos de programação, isto é geralmente obtido alimentando-se uma seqüência maior de bits aleatórios, coletados a partir de uma fonte de aleatoriedade criptograficamente segura, ao algoritmo de hash SHA256 que irá produzir convenientemente um número de 256 bits. Se o resultado for inferior a n - 1,

temos uma chave privada adequada. Caso contrário, nós simplesmente tentamos novamente com outro número aleatório.

TIP Não escreva o seu próprio código para criar um número aleatório, nem utilize um gerador de números aleatório "simples" oferecido pela sua linguagem de programação. Use um gerador de números pseudo-aleatórios criptograficamente seguro (CSPRNG) com uma semente com fonte de entropia suficiente. Estude a documentação da biblioteca geradora de números aleatórios que você escolher para se certificar de que é criptograficamente segura. O correto emprego do CSPRNG é crítico para a segurança das chaves.

Abaixo está demonstrada uma chave privada (k) gerada aleatoriamente mostrada em formato hexadecial (256 dígitos binários mostrados como 64 dígitos hexadecimais, cada um com 4 bits):

```
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
```

TIP O tamanho do espaço possível de Chaves Privadas existentes, 2^256 é de tamanho imcompreensível. É aproximadamente 10^77 na escala decimal. Estima-se que o universo visível contenha 10^80 átomos.

Para gerar uma nova chave com o cliente Bitcoin Core (veja [ch03_bitcoin_client]), Use o comando getnewaddress. Por motivos de segurança é exibida somente a chave pública, não a chave privada. Para pedir ao bitcoind para expor a chave privada, use o comando dumpprivkey. O comando dumpprivkey mostra a chave privada em um formato checksum-codificado em Base58 chamado de *Wallet Import Format* (WIF), que vamos examinar com mais detalhes em Formatos de chave privada. Aqui está um exemplo da criação e exibição uma chave privada usando esses dois comandos:

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

O comando dumpprivkey abre a carteira e extrai a chave privada que foi gerada pelo comando getnewaddress. Não é possível que o bitcoind descubra a chave privada a partir da chave pública, a menos que ambas estejam armazenadas na carteira.

TIP O comando dumpprivkey não está gerando uma chave privada a partir de uma chave pública, já que isso é impossível. O comando simplesmente revela a chave privada que a carteira já conhece e que foi gerada através do comando getnewaddress.

Você também pode usar a ferramenta de linha de comando Bitcoin Explorer (ver [libbitcoin]) para gerar e mostrar chaves privadas com os comandos seed, ec-new and ec-to-wif:

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

Chaves Públicas

A chave pública é calculada a partir da chave privada usando uma multiplicação de curva elíptica, que é irreversível: \(K = k * G\) onde *k* é a chave privada, *G* é um ponto constante chamado de *ponto gerador* e *K* é a chave pública resultante. A operação inversa, conhecida como "encontrando o logaritmo discreto"—calculando *k* se você sabe o *K*—é tão difícil quanto tentar todos os possíveis valores de k, ou seja, uma busca de força bruta. Antes de nós demonstrarmos como gerar uma chave pública a partir de uma chave privada, vamos analisar a criptografia de curva elíptica em um pouco mais de detalhes.

Explicando a Criptografia de Curva Elíptica

A criptografia de curva elíptica é um tipo de criptografia de chave pública ou assimétrica baseada em um problema logarítimico discreto que é expressado pela adição e multiplicação nos pontos de uma curva elíptica.

Uma curva elíptica é um exemplo de uma curva elíptica, similar à utilizada pelo bitcoin.

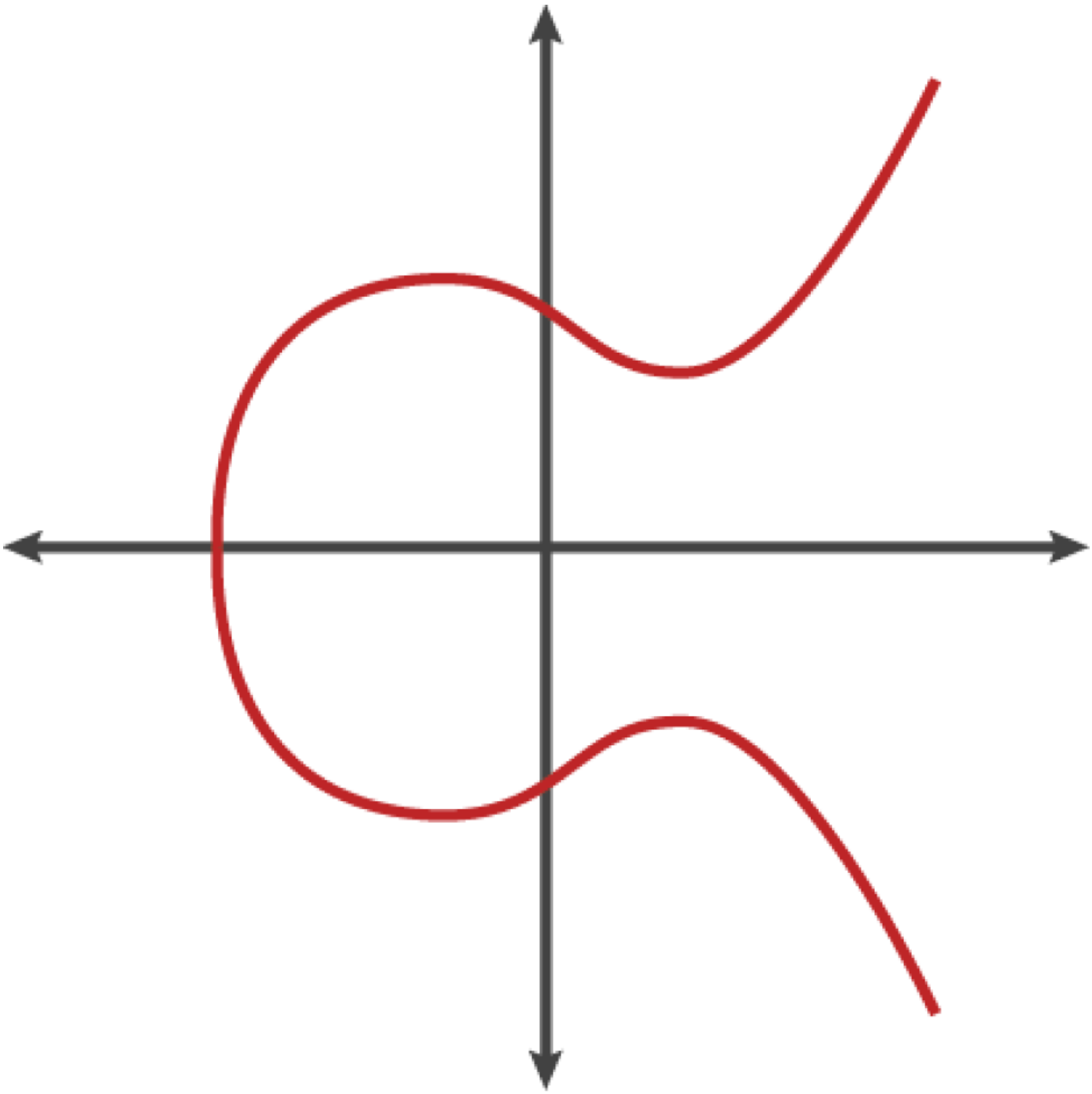


Figure 2. Uma curva elíptica

O Bitcoin usa uma curva elíptica específica e um conjunto de constantes matemáticas, como definido em um padrão chamado secp256k1, estabelecido pelo Instituto Nacional de Padronização e Tecnologia (NIST). A curva secp256k1 é definida pelas seguintes funções, que produzem uma curva elíptica:

or

O *mod p* (módulo de número primo) indica que essa curva está sobre um campo finito de números primos *p* também escrito em formato latex:[\(\mathbb{F}_p\)], onde p = $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$, um número primo muito grande.

Como a curva é definida sobre um campo finito de ordem prima ao invés de baseada em números reais, ela parece um padrão de pontos espalhados em duas dimensões, o que a torna difícil de ser visualizada. Contudo, a matemática é idêntica àquela de curvas elípticas sobre números reais. Como um exemplo, Criptografia de curva elíptica: visualizando uma curva elíptica sobre F(p), com p=17 mostra a mesma curva elíptica sobre um campo muito menor, de ordem prima 17, mostrando um padrão de pontos em uma grade. A curva elíptica secp256k1 pode ser imaginada como um padrão muito mais complexo de pontos em uma enorme malha.

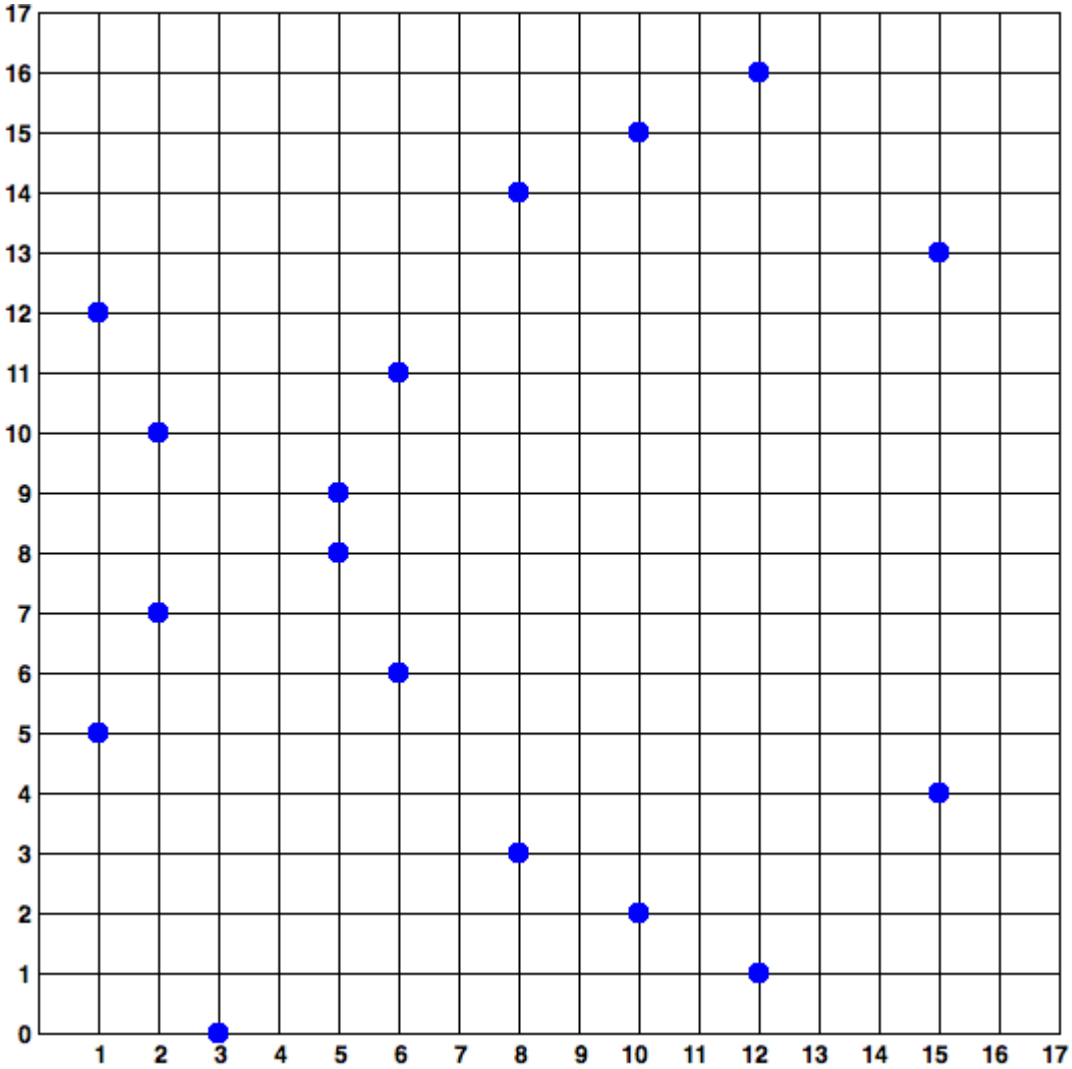


Figure 3. Criptografia de curva elíptica: visualizando uma curva elíptica sobre F(p), com p=17

Então, por exemplo, a seguir está um ponto P com coordenadas (x,y) que é um ponto na curva secp256k1. Você pode verificar isso por conta própria usando Python:

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

```
Python 3.4.0 (padrão, Mar 30 2014, 19:23:13)
[GCC 4.2.1 Compatível com Apple LLVM 5.1 (clang-503.0.38)] no darwin
Digite "help", "copyright", "credits" ou "license" para maiores informações.
>>> p =
115792089237316195423570985008687907853269984665640564039457584007908834671663
>>> x = 55066263022277343669578718895168534326250603453777594175500187360389116729240
>>> y = 32670510020758816978083085130507043184471273380659243275938904335757337482424
>>> (x ** 3 + 7 - y**2) % p
0
```

Na matemática de curva elíptica, existe um ponto chamado "ponto no infinito", que de certo modo corresponde ao papel do 0 em uma adição. Em computadores, é frequentemente representado por x = y = 0 (que não satisfaz a equação da curva elíptica, mas é um caso isolado que pode ser facilmente verificado).

Há também um operador de passe[+], chamado "adição", que possui umas propriedades similares à adição tradicional dos números reais que as crianças no ensino fundamental aprendem. Dados dois pontos P_1 e P_2 na curva, há um terceiro ponto $P_3 = P_1 + P_2$, também na curva elíptica.

Geometricamente, esse terceiro ponto P_3 é calculado ao desenhar uma linha entre P_1e P_2. Essa linha fará a intersecção com a curva elíptica exatamente em um lugar adicional. Chame esse ponto $P_{\#}$' = (x,y). Então reflita no eixo x para obter P_3 = (x, -y).

Existem vários casos especiais que explicam a necessidade de um "ponto no infinito."

Se P_1 e P_2 são o mesmo ponto, a linha "entre" P_1 e P_2 deveria estender para ser a tangente na curva no ponto P_1. Essa tangente fará a intersecção com a curva em um novo ponto exato. Você pode usar técnicas de cálculo para determinar a inclinação da linha tangente. Essas técnicas curiosamente funcionam, mesmo que estejamos restringindo nosso interesse para pontos na curva com duas coordenadas inteiras!

Em alguns casos (ex: se P_1 e P_2 têm os mesmos valores x mas diferentes valores y), a linha tangente será exatamente vertical, que no caso P3 = "ponto no infinito".

Se P_1 é o "ponto no infinito", então a soma $P_1 + P_2 = P_2$. De modo similar, se P_2 é o ponto no infinito, então $P_1 + P_2 = P_1$. Isso mostra como o ponto no infinito faz o papel do 0.

De forma que + é associativo, que significa que (A + B + C = A + (B + C). O que significa que podemos escrever A + B + C sem parênteses sem qualquer ambiguidade.

Agora que definimos adição, podemos definir multiplicação no modo padrão que estende adição. Para um ponto P na curva elíptica, se k é um número inteiro, então kP = P + P + P +... + P (k vezes). Note que k é algumas vezes confundido como um "expoente" nesse caso.

Gerando uma Chave Pública

Começando com uma chave privada na forma de um número *k* aleatoriamente gerado, o multiplicamos por um ponto predeterminado na curva, chamado de o *ponto gerador G* para produzir outro ponto em outro lugar da curva, que será a chave pública *K* correspondente. O ponto gerador é especificado como parte do padrão secp256k1 e é sempre o mesmo para todas as chaves bitcoin:

onde k é a chave privada, G é o ponto gerador e K é a chave pública resultante, um ponto na curva. Como o ponto gerador é sempre o mesmo para todos os usuários bitcoin, uma chave privada k multiplicada por G sempre resultará na mesma chave pública K. A relação entre k e K é fixa, mas apenas pode ser calculada em uma direção, de k para K. É por isso que um endereço bitcoin (derivado de K) pode ser compartilhado com qualquer um sem que a chave privada (k) do usuário seja revelada.

TIP Uma chave privada pode ser convertida em uma chave pública, mas uma chave pública não pode ser convertida de volta em uma chave privada porque a matemática só funciona em um único sentido.

Implementando a multiplicação da curva elíptica, podemos usar a chave privada k que foi gerada anteriormente e multiplicá-la pelo ponto gerador G para encontrarmos a chave pública K:

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

A Chave Pública K é definida como o ponto K = (x,y):

```
K = (x, y)

onde,

x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Para visualizar a multiplicação de um ponto com um número inteiro, usaremos uma curva elíptica mais simples sobre números reais — lembre-se, a matemática é a mesma. Nosso objetivo é descobrir o múltiplo kG de um ponto gerador G. Isso é o mesmo que adicionar G a si mesmo, k vezes seguidas. Nas curvas elípticas, adicionar um ponto a si mesmo é o equivalente a desenhar uma linha tangente no ponto e encontrar onde ela intersecciona novamente a curva, refletindo, então, o ponto no eixo x.

Criptografia de curva elíptica: Visualizando a multiplicação de um ponto G por um número inteiro k em uma curva elíptica mostra o processo de derivar G, 2G e 4G, enquanto operação geométrica na curva.

TIP A maior parte das implementações bitcoin utilizam a biblioteca criptográfica OpenSSL para fazer a matemática de curva elíptica. Por exemplo, para derivar a chave pública, é utilizada a função EC_POINT_mul().

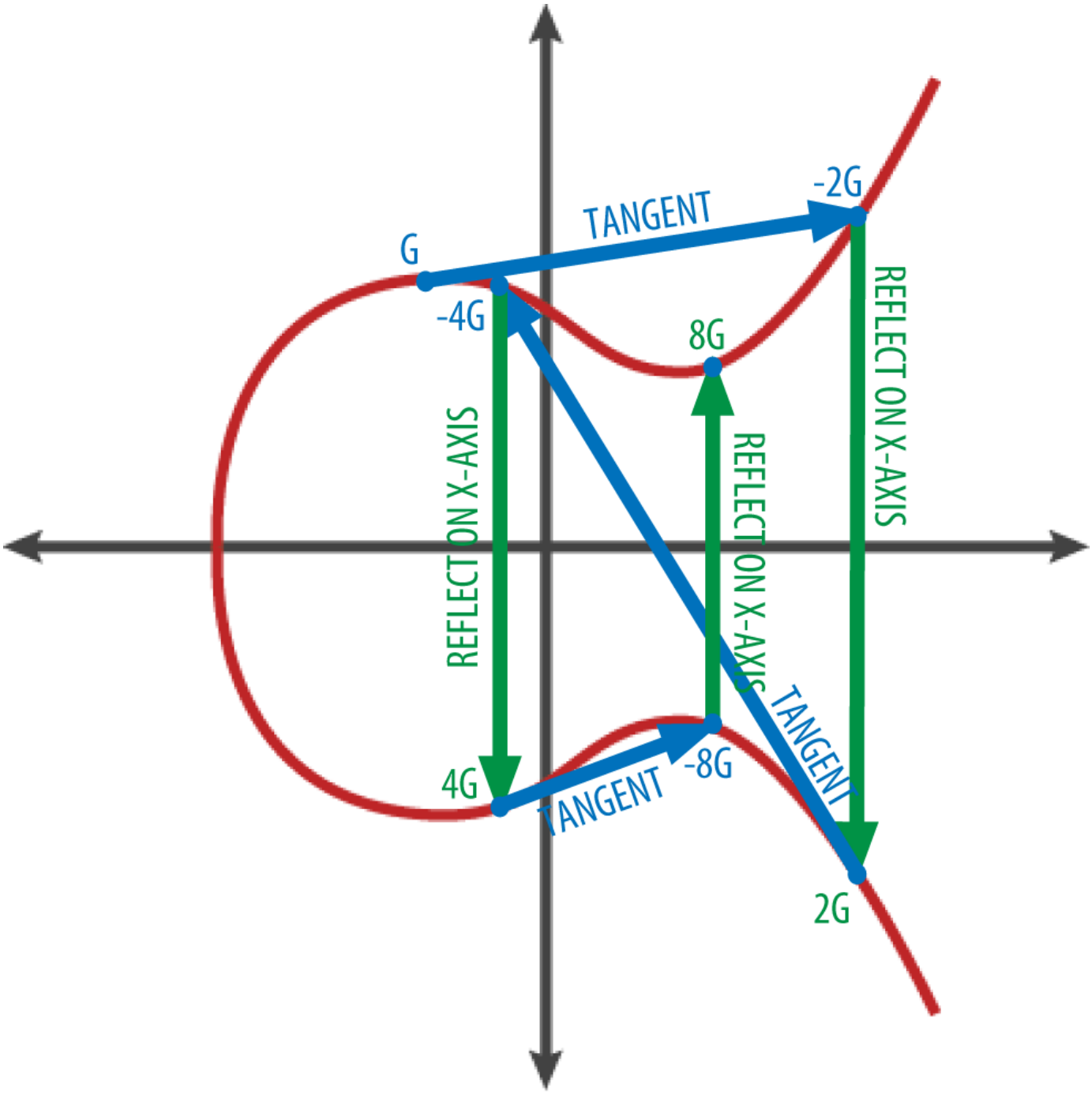


Figure 4. Criptografia de curva elíptica: Visualizando a multiplicação de um ponto G por um número inteiro k em uma curva elíptica

Endereços Bitcoin

Um endereço bitcoin é uma string de dígitos e caracteres que pode ser compartilhada com qualquer pessoa que queira lhe enviar dinheiro. Os endereços produzidos a partir de chaves públicas consistem

em uma string de números e letras, iniciando com o dígito "1". Aqui está um exemplo de um endereço bitcoin:

```
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

O endereço bitcoin é o que mais comumente aparece como "destinatário" dos fundos em uma transação. Se fôssemos comparar uma transação bitcoin a um cheque de papel, o endereço bitcoin é o beneficiário, que é o que escrevemos na linha "à ordem de". Num cheque de papel, aquele beneficiário pode algumas vezes ser o nome de um possuidor de conta bancária, mas pode também incluir corporações, instituições, e até mesmo dinheiro. E por cheques de papel não precisarem especificar uma conta, ao invés disso, usam um nome abstrato como receptor dos fundos, isto torna cheques de papel muito flexíveis como instrumentos de pagamento. As transações bitcoin usam uma abstração similar - o endereço bitcoin - para torná-las muito flexíveis. Um endereço bitcoin pode representar outra coisa, como um script de pagamento, como veremos em [p2sh]. Por hora, vamos examinar o caso simples, um endereço bitcoin que representa uma chave pública, e é derivado dela.

O endereço bitcoin é derivado da chave pública através do uso de hashing criptográfico de uma via. Um "algoritmo de hashing", ou simplesmente "algoritmo de hash" é uma função injetiva (de uma via) que produz uma impressão digital ou "hash" a partir de uma entrada de tamanho arbitrário. Funções de hash criptográficas são extensivamente usadas em bitcoin: nos endereços bitcoin, nos endereços de scripts, e no algoritmo de prova-de-trabalho da mineração. Os algoritmos utilizados para produzir um endereço bitcoin a partir de uma chave pública são Secure Hash Algorithm (SHA) e o RACE Integrity Primitives Evaluation Message Digest (RIPEMD), especificamente SHA256 e RIPEMD160.

A partir da chave pública K, nós computamos o hash SHA256, e então computamos o hash RIPEMD160 do resultado, produzindo um número de 160 bits (20 bytes).

onde K é a chave pública e A é o endereço bitcoin resultante.

TIP Um endereço bitcoin *não* é a mesma coisa que uma chave pública. Endereços bitcoin são derivados a partir de uma chave pública utilizando-se uma função de sentido único.

Os endereços bitcoin são quase sempre apresentados aos usuários em uma codificação chamada "Base58Check" (veja Codificação Base58 e Base58Check), que usa 58 caracteres (um sistema numérico de Base58) e um checksum para ajudar a legibilidade humana, evitar ambiguidade, e proteger contra erros em transcrição e digitação de endereços. O Base58Check também é utilizado de várias outras formas no bitcoin, sempre que há a necessidade de um usuário ler e transcrever corretamente um número, tal como um endereço bitcoin, uma chave privada, uma chave encriptada, ou um hash de um script. Na próxima seção, examinaremos a mecânica da codificação e decodificação Base58Check, e as represetnações resultantes. Chave pública para endereço bitcoin: conversão de uma chave pública em um endereço bitcoin ilustra a conversão de uma chave pública em um endereço bitcoin.

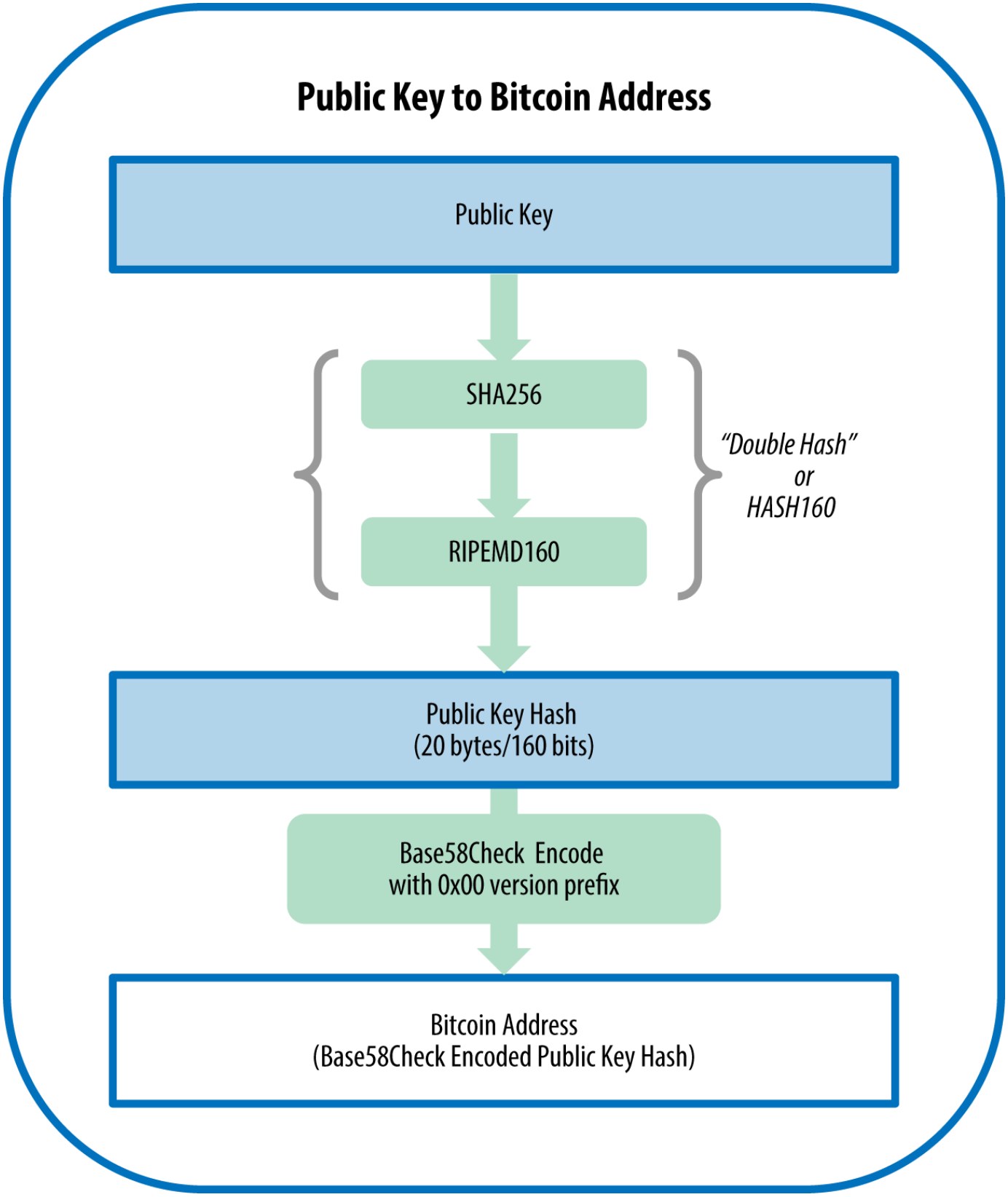


Figure 5. Chave pública para endereço bitcoin: conversão de uma chave pública em um endereço bitcoin

Codificação Base58 e Base58Check

A fim de representar números grandes de uma forma compacta, utilizando poucos símbolos, muitos sistemas de computador utilizam uma mistura alfa-númerica com base (ou raiz) maior do que 10. Por

exemplo, enquanto o sistema decimal tradicional utiliza os 10 numerais de 0 a 9, o sistema hexadecimal utiliza 16, com as letras de A até F como os seis símbolos adicionais. Um número representado no formato hexadecimal é menor ao equivalente em decimal. Ainda mais compacto, a representação Base64 utiliza 26 letras em caixa baixa, 26 letras em caixa alta, 10 numerais e mais dois caracteres como "+" e "/" para transmitir dados binários sobre mídias baseadas em texto, como o email. A Base64 é normalmente utilizada para anexar arquivos binários em emails. A Base58 é a codificação binária baseada em texto desenvolvida para o bitcoin e utilizada e muitas outras criptomoedas. Ela oferece um equilíbrio entre uma representação compacta, leitura e detecção de erro e prevenção. A Base58 é um subconjunto da Base64, utilizando caixa alta, caixa baixa, letras e números porém omitindo alguns caracteres que são frequentemente confundidos e podem parecer idênticos quando mostrados com certas fontes. Sendo específico, a Base58 é a Base64 sem o 0 (número zero), O (o maiúsculo), l (ele minúsculo), I (i maiúsculo), e os símbolos "\+" e "/". Ou, mais simplesmente, é um conjunto de letras maiúsculas e minúsculas mais os números, sem os quatro caracteres mencionados.

Example 1. O alfabeto Base58 do bitcoin

```
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
```

Para acrescentar segurança extra contra erros de digitação ou transcrição, o Base58Check é um formato de codificação Base58, frequentemente usado em bitcoin, que tem um código de verificação de erros embutido. O checksum é composto de quatro bytes adicionais, acrescidos ao final dos dados sendo codificados. O checksum é derivado do hash dos dados codificados e, portanto, poder ser usado para detectar e prevenir erros de transcrição e digitação. Quando o software decodificador se depara com um código Base58Check, ele calcula o checkum dos dados e o compara com o checksum incluído no código. Se os dois não corresponderem, tem-se uma indicação de que o erro foi introduzido e o dado Base58Check é inválido. Por exemplo, isso previne um endereço bitcoin mal digitado de ser aceito pelo software da carteira como um destino válido, um erro que, do contrário, resultaria na perda de fundos.

Para converter dados (um número) no formato Base58Check, primeiro adicionamos um prefixo aos dados, chamados de "byte de versão", que serve para identificar facilmente o tipo de dado codificado. Por exemplo, no caso de um endereço bitcoin, o prefixo é zero (0x00 em hexa), enquanto o prefixo usado para codificar uma chave privada é 128 (0x80 em hexa). Uma lista de prefixos comuns é apresentada em Base58Check prefixo de versão e exemplos de resultados codificados.

Em seguida, computamos o checksum "duplo-SHA", o que significa que aplicamos o algoritmo de hash SHA256 duas vezes no resultado anterior (prefixo e dados):

```
checksum = SHA256(SHA256(prefix+data))
```

Pegamos apenas os primeiros quatro bytes do hash de 32 bytes resultante (hash do hash). Esses quatro bytes servem como código de verificação de erro, ou checksum. O checksum é concatenado ao final.

O resultado é composto de três itens: um prefixo, os dados, e um checksum. Esse resultado é codificado

usando o alfabeto Base58 descrito previamente. Base58Check encoding: um formato para codificar dados do bitcoin de maneira inequívoca, versionada e verificada. ilustra o processo de codificação Base58Check.

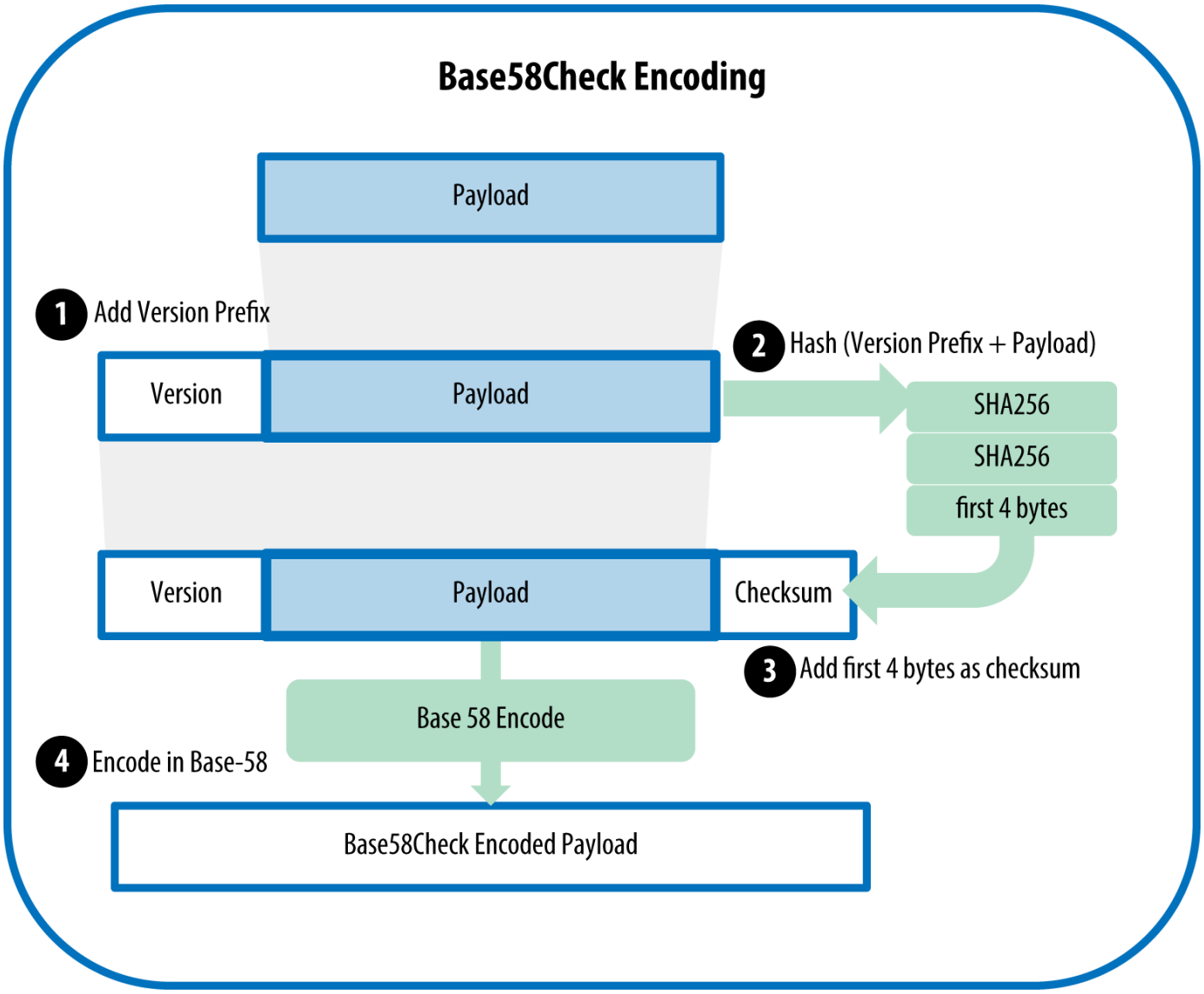


Figure 6. Base58Check encoding: um formato para codificar dados do bitcoin de maneira inequívoca, versionada e verificada.

No bitcoin, a maior parte dos dados apresentados ao usuário é codificada em Base58Check para torná-los mais compactos, fáceis de ler e para facilitar a detecção de erros. A versão de prefixo na codificação Base58Check é usada para criar formatos facilmente distinguíveis, que quando são codificados em Base58 contém caracteres específicos no início do payload codificado em Base58Check. Esses caracteres facilitam a identificação pelas pessoas do tipo de dado que está sendo codificado e como utilizá-lo. Isso é o que diferencia, por exemplo, um endereço bitcoin codificado em Base58Check que começa com o número 1 de uma chave privada codificada em Base58Check no formato WIF que inicia com o número 5. Alguns exemplos de versão de prefixo e os caracteres Base58 resultantes são demonstrados em Base58Check prefixo de versão e exemplos de resultados codificados.

Table 1. Base58Check prefixo de versão e exemplos de resultados codificados

Type	Prefixo de versão (hex)	Base58 prefixo de resultado
Endereço Bitcoin	0x00	1
Endereço Pay-to-Script-Hash	0x05	3
Endereço Bitcoin Testnet	0x6F	m ou n
Chave Privada WIF	0x80	5, K ou L
Chave Privada Criptografada em BIP38	0x0142	6P
Chave Pública Estendida em BIP32	0x0488B21E	xpub

Vamos ver o processo completo de criação de um endereço bitcoin, desde a chave privada, até a chave pública (um ponto na curva elíptica), para um endereço que sofre hash duplo e, finalmente, a codificação Base58Check. O código C++ em Criando um endereço bitcoin codificado em Base58Check a partir de uma chave privada demonstra todo o processo passo-a-passo, desde a chave privada até o endereço bitcoin codificado em Base58Check. O código de exemplo usa a livraria libbitcoin que foi apresentada em [alt_libraries] para algumas funções auxiliares.

Example 2. Criando um endereço bitcoin codificado em Base58Check a partir de uma chave privada

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

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

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

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

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

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

O código usa uma chave privada pré-definida, de maneira que ele produz o mesmo endereço bitcoin

cada vez que é executado, como demonstrado em Compilando e executando o código addr.

Example 3. Compilando e executando o código addr

```
# Compila o código addr.cpp
$ g++ -o addr addr.cpp $(pkg-config --cflags --libs libbitcoin)
# Roda o executável addr
$ ./addr
Public key: 0202a406624211f2abbdc68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa
Address: 1PRTTaJesdNovgne6Ehcdu1fpEdX7913CK
```

Formatos de Chave

Tanto as chaves privadas quanto as chaves públicas podem ser representadas em diversos formatos diferentes. Todas essas representações codificam o mesmo número, apesar de elas parecerem ser diferentes. Esses formatos são primariamente usados para tornar fácil para as pessoas lerem e transcreverem as chaves sem que cometam erros.

Formatos de chave privada

A chave privada pode ser representada em vários formatos diferentes, os quais todos correspondem ao mesmo número de 256 bits. A Representações de chave privada (formatos de codificação) demonstra três formatos comumente usados para representar chaves privadas.

Table 2. Representações de chave privada (formatos de codificação)

Tipo	Prefixo	Descrição
Hex	Nenhum	64 dígitos hexadecimais
WIF	5	Codificação em Base58Check: Base58 com prefixo de versão de 128 e soma de verificação (checksum) de 32-bit
Comprimida-WIF	K ou L	Como acima, com a adição do sufixo 0x01 antes da codificação

Exemplo: Mesma chave, diferentes formatos demonstra a chave privada nesses três formatos.

Table 3. Exemplo: Mesma chave, diferentes formatos

Formato	Chave Privada
Hex	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd

Formato	Chave Privada
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Comprimida-WIF	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Todas essas representações são diferentes maneiras de se exibir o mesmo número, a mesma chave privada. Elas parecem diferente, mas qualquer um desses formatos pode ser facilmente convertido para qualquer outro formato.

Nós usamos o comando wif-to-ec do Bitcoin Explorer (ver [libbitcoin]) para demonstrar que ambas as chaves WIF representam a mesma chave privada:

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

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

Decodificando do Base58Check

Os comandos do Bitcoin Explorer (ver [libbitcoin]) tornam fácil a escrita de scripts shell e "pipes" de linha de comando que manipulam as chaves, endereços e transações bitcoin. Você pode usar o Bitcoin Explorer para decodificar o formato Base58Check na linha de comando.

Nós usamos o comando base58check-decode para decodificar a chave não-comprimida:

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

O resultado contém a chave como a carga, o prefixo 128

Note que o "payload" da chave comprimida é acrescido com o sufixo 01, sinalizando que a chave pública derivada será comprimida.

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

Codificando de hex para Base58Check

Para codificar em Base58Check (o oposto do comando anterior), nós usamos o comando base58check-encode do Bitcoin Explorer (ver [libbitcoin]) e fornecemos a chave privada em hexadecimal, seguida pelo Wallet Import Format (WIF) com prefixo de versão 128:

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

Codificando de hex (chave comprimida) para Base58Check

Para codificar em Base58Check como uma chave privada "compactada" (veja Chaves privadas comprimidas), nós concatenamos o sufixo 01 à chave hexa e em seguida codificamos da forma descrita acima:

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

O formato comprimida-WIF resultante começa com um "K". Isso denota que a chave privada contida tem um sufixo de "01" e será usada para produzir apenas chaves públicas comprimidas (ver Chaves públicas comprimidas).

Formatos de chave pública

As chaves públicas também são apresentadas de diferentes maneiras, principalmente como chaves públicas *comprimidas* ou *não-comprimidas*.

Como vimos anteriormente, a chave pública é um ponto na curva elíptica que consiste de um par de coordenadas (x,y). Ela geralmente é apresentada com o prefixo 04 seguida por dois números de 256 bits, um para a coordenada x do ponto, e outro para a coordenada y. O prefixo 04 é usado para distinguir as chaves públicas não-comprimidas das chaves públicas comprimidas que começam com 02 ou 03.

Aqui está a chave pública gerada pela chave privada que nós criamos anteriormente, mostrada como

as coordenadas x e y:

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

Aqui está a mesma chave pública mostrada como um número de 520 bits (130 dígitos hexadecimais) com o prefixo 04 seguido pelas coordenadas x e y, como, por exemplo, 04 x y:

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

Chaves públicas comprimidas

As chaves públicas comprimidas começaram a ser usadas no bitcoin para reduzirem o tamanho das transações e para conservarem o espaço em disco que os nodos usam para armazenar a blockchain, que é o banco de dados do bitcoin. A maioria das transações incluem a chave pública, que é necessária para se validar as credenciais do dono e poder gastar os bitcoins. Cada chave pública requer 520 bits (prefixo \+ x \+ y), que quando são multiplicados por várias centenas de transações por bloco, ou dezenas de milhares de transações por dia, acabam sendo uma grande quantidade de dados adicionais na blockchain.

Como nós vimos na seção Chaves Públicas, uma chave pública é um ponto (x,y) em uma curva elíptica. Como a curva expressa uma função matemática, um ponto na curva representa uma solução para a equação e, portanto, se nós soubermos a coordenada *x*, nós poderemos calcular a coordenada *y* ao resolver a equação $y^2 \bmod p = (x^3 + 7) \bmod p$. Isso nos permite armazenar somente a coordenada *x* do ponto da chave pública, omitindo a coordenada *y* e reduzindo o tamanho da chave e o espaço necessário para armazená-la em 256 bits. Uma redução de quase 50% do tamanho de cada transação economiza a utilização de muito espaço ao longo do tempo!

Enquanto as chaves públicas não-comprimidas tem um prefixo 04, as chaves públicas comprimidas começam com os prefixos 02 ou 03. Existe um motivo pelo qual só existem dois prefixos possíveis: como o lado esquerdo da equação é y^2, isso significa que a solução para y é uma raiz quadrada, que pode ter um valor positivo ou negativo. Visualmente, isso significa que a coordenada *y* resultante pode estar acima ou abaixo do eixo x. Como você pode ver no gráfico da curva elíptica em Uma curva elíptica, a curva é simétrica, sendo refletida pelo eixo x, como se fosse um espelho. Então, enquanto nós podemos omitir a coordenada *y*, nós temos que armazenar o *sinal* do y (positivo ou negativo), ou, em outras palavras, nós temos que nos lembrar se ele estava acima ou abaixo do eixo x, pois cada uma dessas opções representa um ponto diferente e uma chave pública diferente. Quando se calcula a curva elíptica em aritmética binária no campo finito da ordem prima p, a coordenada *y* ou é par ou é ímpar, o que corresponde ao sinal positivo/negativo explicado anteriormente. Portanto, para distinguir entre os dois valores possíveis de y, nós armazenamos a chave pública comprimida com o prefixo 02 se o y é par, e 03 se ele é ímpar, permitindo que o software deduza corretamente a coordenada *y* a partir da coordenada *x* e descomprima da chave pública para as coordenadas completas do ponto. A compressão de chave pública é ilustrada em Compressão de chave pública.

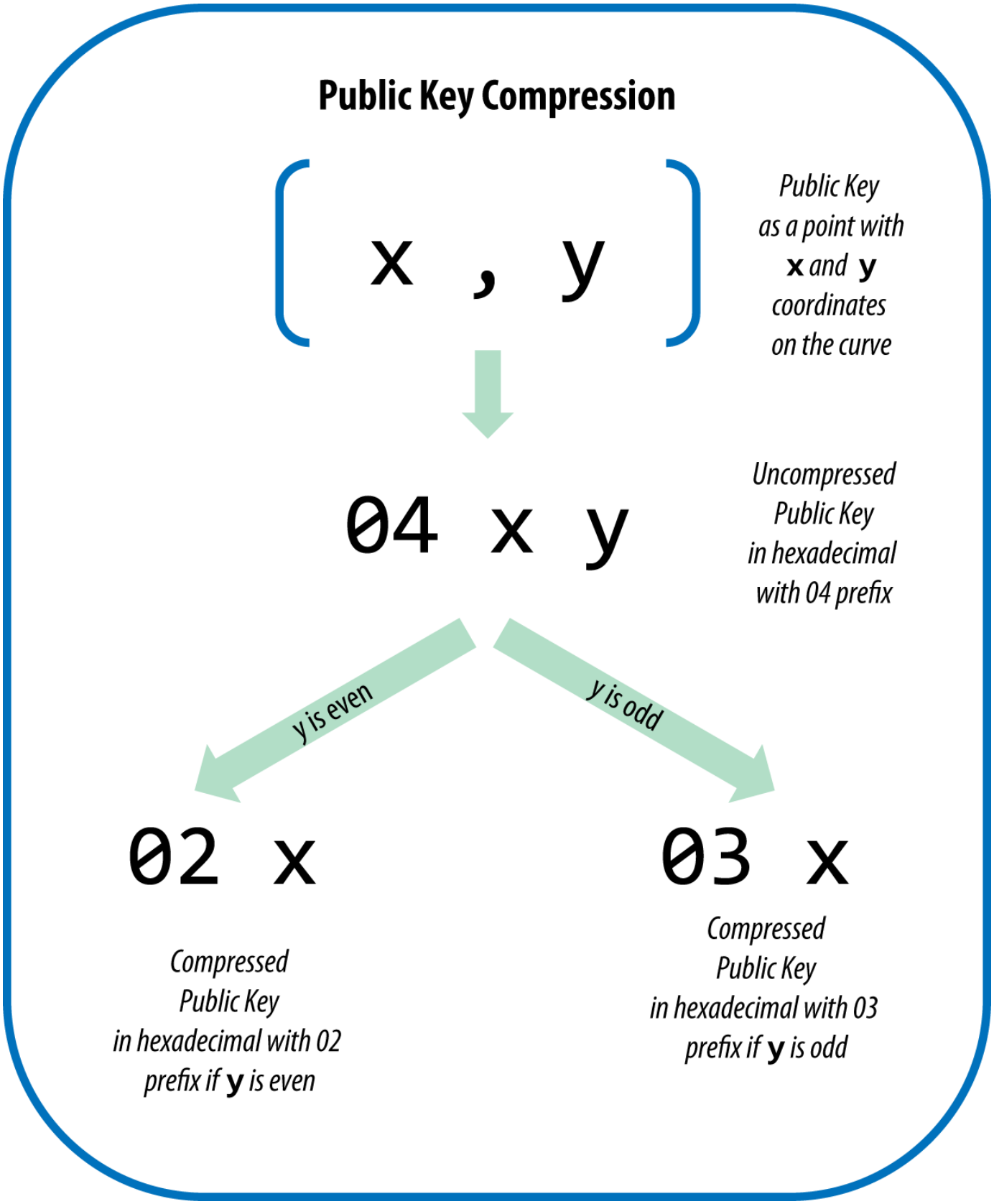


Figure 7. Compressão de chave pública

Aqui está a mesma chave pública gerada anteriormente, exibida como uma chave pública comprimida armazenada em 264 bits (66 dígitos hexadecimais) com o prefixo 03 indicando que a coordenada *y* é ímpar:

```
K = 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
```

Essa chave pública comprimida corresponde à mesma chave privada, significando que ela é gerada a partir da mesma chave privada. Entretanto, ela parece ser diferente da chave pública não-comprimida. É importante citar que, se nós convertermos essa chave pública comprimida para um endereço bitcoin usando uma função de hash duplo (RIPEMD160(SHA256(K))) ela irá produzir um endereço bitcoin *diferente*. Isso pode parecer confuso, porque isso significa que uma chave privada é capaz de produzir uma chave pública que é expressa em dois formatos diferentes (comprimido e não-comprimido), que produzem dois endereços bitcoin diferentes. Entretanto, a chave privada é idêntica para ambos os endereços bitcoin.

As chaves públicas comprimidas estão gradualmente se tornando o padrão nos clientes bitcoins, o que está causando um impacto significativo na redução do tamanho das transações e, portanto, da blockchain. Entretanto, nem todos os clientes tem suportes já tem suporte às chaves públicas comprimidas. Os clientes mais modernos que suportam as chaves públicas comprimidas tem que lidar com as transações vindas de clientes antigos que não suportam as chaves públicas comprimidas. Isso é especialmente importante quando uma aplicação de carteira está importanto chaves privadas a partir de outra aplicação de carteira bitcoin, pois a carteira nova precisa escanear a blockchain para achar as transações correspondentes a essas chaves importadas. Quais endereços bitcoins a carteira bitcoin deve escanear? Os endereços bitcoin produzidos pelas chaves públicas não-comprimidas, ou os endereços bitcoins produzidos pelas chaves públicas comprimidas? Ambos são endereços bitcoin válidos, e podem ser assinados pela chave privada, mas eles são endereços diferentes!

Para resolver esse problema que acontece quando as chaves privadas são exportadas a partir de uma carteira, o Wallet Import Format que é usado para representá-las é implementado de maneira diferente nas carteiras bitcoin mais modernas, para indicar que essas chaves privadas foram usadas para produzir chaves públicas *comprimidas* e, portanto, endereços bitcoin *comprimidos*. Isso permite que a carteira que as está importando possa distinguir as chaves privadas que se originam das carteiras antigas e das modernas, e buscar a blockchain pelas transações com endereços bitcoin correspondendo às chaves públicas não-comprimidas ou comprimidas, respectivamente. Nós iremos aprender como isso funciona em maiores detalhes, na próxima seção.

Chaves privadas comprimidas

Ironicamente, o termo "chave privada comprimida" pode confundir, pois quando uma chave privada é exportada como uma chave comprimida-WIF, na verdade ela é um byte *maior* do que uma chave privada "não-comprimida". Isso acontece porque ela tem o sufixo 01 adicional, o que significa que ela vem de numa carteira mais nova e que só deveria ser usado para produzir chaves públicas comprimidas. As chaves privadas não são comprimidas e não há como comprimi-las. O termo "chave privada comprimida" realmente quer dizer "chave privada a partir da qual as chaves públicas comprimidas devem ser derivadas", enquanto o termo "chave privada não-comprimida" quer dizer "chave privada a partir da qual as chaves públicas não-comprimidas devem ser derivadas". Você só deve se referir ao formato de exportação como "comprimida-WIF" ou "WIF" e, para evitar maiores confusões, evite se referir à chave privada como "comprimida".

Lembre-se que esses formatos não *são* intercambiáveis. Em uma carteira moderna que implementa as chaves públicas comprimidas, as chaves privadas só serão exportadas como comprimidas-WIF (com um prefixo K ou L). Se a carteira é uma implementação antiga e não usa chaves públicas comprimidas, as chaves só serão exportadas como WIF (com o prefixo 5). O objetivo é sinalizar para a carteira que está importando essas chaves privadas se ela precisa pesquisar na blockchain as chaves públicas comprimidas ou não-comprimidas e os endereços.

Se uma carteira bitcoin é capaz de implementar chaves públicas comprimidas, ela irá usar esse tipo de chave em todas as transações. As chaves privadas na carteira serão usadas para derivar os pontos de chave pública na curva, que serão comprimidas. As chaves públicas comprimidas serão usadas para produzir endereços bitcoin, que serão usados nas transações. Quando se exporta chaves privadas a partir de uma carteira mais moderna, que implementa as chaves públicas comprimidas, o Wallet Import Format é modificado, com a adição de um sufixo 01 de um byte para chave privada. A chave privada resultante, codificada em Base58Check, é chamada de "Comprimida WIF" e inicia com a letra K ou L, ao invés de iniciar com o número "5", que é o que acontece com as chaves codificadas em WIF (não-comprimidas) das carteiras mais antigas.

Exemplo: Mesma chave, diferentes formatos mostra a mesma chave, codificada nos formatos WIF e Comprimida-WIF.

Table 4. Exemplo: Mesma chave, diferentes formatos

Formato	Chave Privada
Hex	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Comprimida-Hex	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD_01_
Comprimida-WIF	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

TIP

O termo "chaves privadas comprimidas" é um termo mal empregado. Elas não são comprimidas; ao invés disso, o formato comprimida-WIF significa que elas só deveriam ser usadas para derivar chaves públicas comprimidas e seus endereços bitcoins correspondentes. Ironicamente, uma chave privada "comprimida-WIF" é um byte maior porque ela tem adicionalmente o sufixo 01 para distingui-la de uma chave "não-comprimida".

Implementando Chaves e Endereços em Python

A livraria bitcoin mais completa em Python é a pybitcointools feita pelo Vitalik Buterin. Em Geração de chaves e endereços e formatação com a livraria pybitcointools, nós usamos a livraria pybitcointools

(importada como "bitcoin") para gerar e exibir as chaves e endereços em vários formatos.

Example 4. Geração de chaves e endereços e formatação com a livraria pybitcointools

```
import bitcoin

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

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

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

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

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

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

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

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

# Generate bitcoin address from public key
```

```
print "Bitcoin Address (b58check) is:", bitcoin.pubkey_to_address(public_key)

# Generate compressed bitcoin address from compressed public key
print "Compressed Bitcoin Address (b58check) is:", \
    bitcoin.pubkey_to_address(hex_compressed_public_key)
```

Executando key-to-address-ecc-example.py mostra o output após executar esse código.

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

Um script demonstrado a matemática da curva elíptica usada para as chaves bitcoin é outro exemplo, usando a livraria Python ECDSA para a matemática da curva elíptica e sem usar quaisquer livrarias bitcoin especializadas.

Example 6. Um script demonstrado a matemática da curva elíptica usada para as chaves bitcoin

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

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

curve = curve_secp256k1
generator = generator_secp256k1

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

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

    return convert_to_int(byte_array)
```

```
def get_point_pubkey(point):
    if point.y() & 1:
        key = '03' + '%064x' % point.x()
    else:
        key = '02' + '%064x' % point.x()
    return key.decode('hex')

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


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

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

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

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

Instalando a livraria Python ECDSA e executando o script ec_math.py demonstra o output que é produzido quando se executa esse script.

NOTE

O exemplo acima usa os.urandom, que reflete um gerador de números aleatórios criptograficamente seguro (CSRNG) fornecido pelo sistema operacional subjacente. No caso de um sistema operacional do tipo UNIX, como o Linux, ele adquire isso a partir de /dev/urandom; e no caso do Windows, a partir de CryptGenRandom(). Se uma fonte adequada de aleatoriedade não for encontrada, o NotImplementedError será gerado. Enquanto o gerador de número aleatório usado aqui é usado para fins demonstrativos, ele _não é apropriado para a geração de chaves bitcoins com qualidade suficiente para serem usadas em produção, já que eles não é implementado com segurança suficiente.

Example 7. Instalando a livraria Python ECDSA e executando o script ec_math.py

```
$ # Instala o administrador de pacotes Python PIP
$ sudo apt-get install python-pip
$ # Instala a livraria Python ECDSA
$ sudo pip install ecdsa
$ # Executa o script
$ python ec-math.py
Secret:
38090835015954358862481132628887443905906204995912378278060168703580660294000
EC point:
(70048853531867179489857750497606966272382583471322935454624595540007269312627,
105262206478686743191060800263479589329920209527285803935736021686045542353380)
BTC public key: 029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a1522873
```

Carteiras

As carteiras são recipientes para chaves privadas, geralmente implementadas como arquivos estruturados ou bancos de dados simples. Outro método de se fazer chaves é através de *geração determinística de chave*. Nesse método, cada chave privada nova é derivada a partir de uma chave privada prévia utilizando-se uma função de hash de via única, que as ligam em sequência. Contanto que você possa recriar essa sequência, você só precisa da primeira chave (conhecida como uma *semente* ou chave *mestre*) para gerar todas elas. Nessa seção nós iremos examinar os diferentes métodos de geração de chaves e as estruturas de carteira que são construídas com elas.

TIP

As carteiras Bitcoin não contém bitcoins, elas contém chaves. Cada usuário possui uma carteira que contém chaves. As carteiras na verdade são chaveiros que contém pares de chaves privadas e públicas (veja Chaves Privada e Pública). Os usuários assinam as transações com as chaves, provando que eles são donos dos outputs da transação, ou seja, provando que são donos de seus bitcoins. Os bitcoins são armazenados na blockchain na forma de outputs de transações (frequentemente denominados como vout ou txout).

Carteiras não-determinísticas (aleatórias)

Nos primeiros clientes bitcoin, as carteiras eram simplesmente coleções de chaves privadas geradas aleatoriamente. Esse tipo de carteira é chamado de *carteira não-determinística tipo 0*. Por exemplo, o cliente Bitcoin Core faz uma geração inicial de 100 chaves privadas aleatórias quando é iniciado pela primeira vez e gera mais chaves conforme a necessidade, usando cada chave apenas uma única vez. Esse tipo de carteira é apelidada de "Apenas um Monte de Chaves" (em inglês, "Just a Bunch of Keys", ou JBOK), e tais carteiras estão sendo substituídas por carteiras determinísticas pois elas são difíceis de se administrar, fazer backup e importações. A desvantagem das chaves aleatórias é que se você gerar muitas delas você precisa manter cópias de todas elas, ou seja, é necessário fazer backups frequentes da carteira. Todas as chaves precisam de backup, ou os fundos que elas controlam serão

irrevogavelmente perdidos se a carteira se tornar inacessível. Isso entra em conflito direto com o princípio de se evitar a reutilização de endereços, utilizando-se cada endereço bitcoin para uma única transação. A reutilização de endereços reduz a privacidade pois associa múltiplas transações e endereços uns com os outros. Uma carteira não-determinística tipo-0 é uma escolha ruim, especialmente se você quer evitar a reutilização de endereço porque isso significa ter que administrar mais chaves, o que cria a necessidade de backups mais frequentes. Embora o cliente Bitcoin Core inclua uma carteira Tipo-0, o uso desse tipo de carteira é desaconselhado pelos desenvolvedores do Bitcoin Core. demonstra uma carteira não-determinística, contendo uma coleção solta de chaves aleatórias.

Carteiras Determinísticas (que usam semente)

As carteiras determinísticas, ou carteiras que usam "sementes", contém chaves privadas que são todas derivadas a partir de uma semente comum, a semente-mestre, através do uso de uma função de hash de sentido único. A semente-mestre é um número gerado aleatoriamente que é combinado com outros dados, como um número índice ou "código de cadeia" (ver Carteiras Determinísticas Hierárquicas (BIP0032/BIP0044)) para derivar as chaves privadas. Em uma carteira determinística, basta ter a semente para se recuperar todas as chaves derivadas, e portanto é necessário apenas um único backup feito no momento de sua criação. A semente também é a única coisa necessária para importar e exportar a carteira, permitindo a fácil migração de todas as chaves do usuário entre diferentes implementações de carteira.

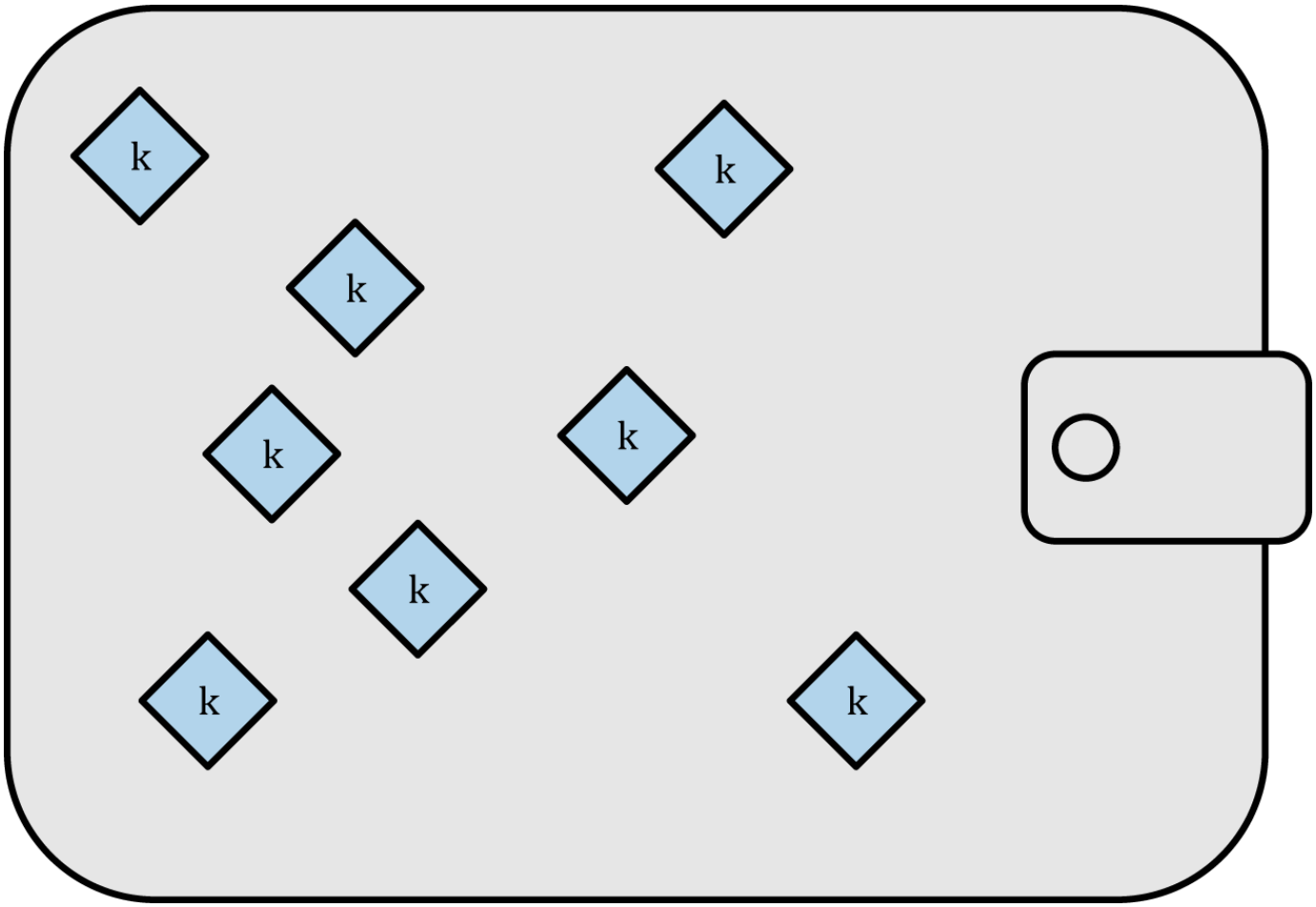


Figure 8. Carteira tipo-0 não-determinística (aleatória): uma coleção de chaves geradas aleatoriamente

Códigos Mnemônicos de Palavras

Os códigos mnemônicos são sequências (listas) de palavras em inglês que representam (codificam) um número aleatório que é usado como semente para se derivar uma carteira determinística. A sequência de palavras é a única coisa necessária para se recriar a semente e, a partir dela, se recriar a carteira e todas as suas chaves derivadas. Um aplicativo de carteiro que implementa carteiras determinísticas com código mnemônico irá mostrar ao usuário uma sequência de 12 a 24 palavras quando ele criar a carteira pela primeira vez. Essa sequência de palavras é a o backup da carteira e pode ser usado para recuperar e recriar todas as chaves no mesmo aplicativo ou em qualquer aplicativo de carteira compatível. As palavras dos códigos mnemônicos tornam mais fácil a criação de um backup das carteiras, porque elas são fáceis de ler e de se anotar corretamente, quando comparadas a uma sequência de números aleatórios.

Os códigos mnemônicos são definidos na Proposta de Melhoria do Bitcoin 39 (ver [bip0039]), atualmente com o status de Rascunho. Note que a BIP0039 é uma proposta em estudo e ainda não é um padrão. Especificamente, existe um padrão diferente, com um conjunto diferente de palavras, usado pela carteira Electrum, que já existia antes do BIP0039. O BIP0039 é usado pela carteira Trezor e algumas outras poucas carteiras, mas é incompatível com a implementação da Electrum.

O BIP0039 define a criação de um código mnemônico e de uma semente da seguinte maneira:

1. Cria uma sequência aleatória (entropia) de 128 a 256 bits.
2. Cria uma soma de verificação (checksum) da sequência aleatória tomando alguns do primeiros poucos bits de seu hash SHA256.
3. Adiciona a soma de verificação (checksum) no final da sequência aleatória.
4. Divide a sequência em seções de 11 bits, usando-as para indexar um dicionário de 2048 palavras pré-definidas.
5. Produz 12 a 24 palavras representando o código mnemônico.

Códigos mnemônicos: entropia e comprimento da palavra mostra a relação entre o tamanho dos dados de entropia e o comprimento em palavras dos códigos mnemônicos.

Table 5. Códigos mnemônicos: entropia e comprimento da palavra

Entropia (bits)	Soma de verificação (checksum) (bits)	Entropia+checksum	Comprimento de palavras
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

O código mnemônico representa de 128 a 256 bits, que são usados para derivar uma semente maior (de 512-bit) através da função de extensão de chave PBKDF2. A semente resultante é usada para criar uma carteira determinística e todas as suas chaves derivadas.

As tabelas <xref linkend="table_4-6" xrefstyle="select: labelnumber"/> e <xref linkend="table_4-7" xrefstyle="select: labelnumber"/> demonstram alguns exemplos de códigos mnemônicos e as sementes que eles produzem.

código mnemônico com entropia de .128-bit e a semente resultante

Input de entropia (128 bits)	0c1e24e5917779d297e14d45f14e1a1a
Mnemônico (12 palavras)	army van defense carry jealous true garbage claim echo media make crunch
Semente (512 bits)	3338a6d2ee71c7f28eb5b882159634cd46a898463e9d2d0980f8e80dfbba5b0fa0291e5fb888a599b44b93187be6ee3ab5fd3ead7dd646341b2cdb8d08d13bf7

código mnemônico com entropia de .256-bit e a semente resultante

Input de entropia (256 bits)	2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c
Mnemônico (24 palavras)	cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige
Semente (512 bits)	3972e432e99040f75ebe13a660110c3e29d131a2c808c7ee5f1631d0a977fcf473bee22fce540af281bf7cdeade0dd2c1c795bd02f1e4049e205a0158906c343

Carteiras Determinísticas Hierárquicas (BIP0032/BIP0044)

As carteiras determinísticas foram desenvolvidas para facilitar a derivação de várias chaves a partir de uma única "semente". A forma mais avançada de carteiras determinísticas é a *carteira determinística hierárquica* ou *carteira HD*, que é definida pelo padrão BIP0032. As carteiras HD contêm chaves derivadas em uma estrutura de árvore, de tal forma que uma chave pai pode derivar uma sequência de chaves filhas, cada uma das quais pode derivar uma sequência de chaves netas, e assim por diante, a uma profundidade infinita. Essa estrutura em árvore é ilustrada em Carteira hierárquica determinística (HD) tipo-2: uma árvore de chaves geradas a partir de uma única semente.

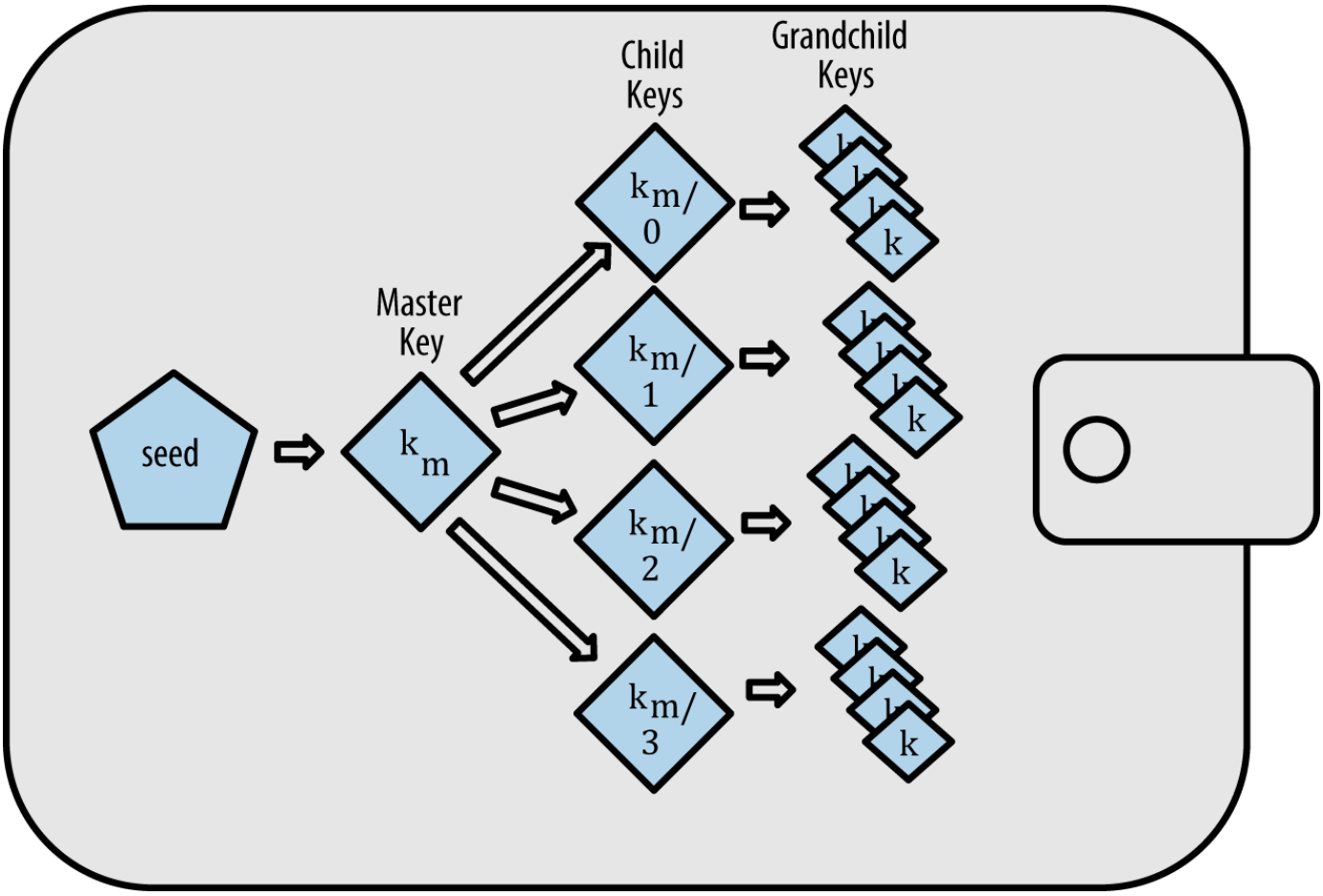


Figure 9. Carteira hierárquica determinística (HD) tipo-2: uma árvore de chaves geradas a partir de uma única semente

TIP	Se você estiver implementando uma carteira bitcoin, ela deve ser desenvolvida como uma carteira HD seguindo os padrões BIP0032 e BIP0044.

Carteiras HD fornecem duas grandes vantagens em comparação com chaves randômicas (não determinísticas). Primeiro, a estrutura em árvore pode ser usada para expressar significado organizacional adicional, por exemplo quando um ramo específico de subchaves é usado para pagamentos recebidos e outro ramo é usado para receber troco de pagamentos realizados. Ramos ou chaves também podem ser usados num cenário corporativo, alocando ramos diferentes a departamentos, subsidiárias, funções específicas, ou centros de custo.

A segunda vantagem de carteiras HD é que os usuários podem criar uma sequência de chaves públicas sem precisar ter acesso às chaves privadas correspondentes. Isso permite que carteiras HD sejam utilizadas num servidor inseguro, ou numa aplicação de somente recebimento, emitindo uma chave pública diferente para cada transação. As chaves públicas não precisam ser previamente carregadas ou derivadas de antemão, assim o servidor não detém as chaves privadas que possam gastar os fundos.

Criação de carteira HD a partir de semente

Carteiras HD são criadas a partir de uma única *semente raiz*, que é um número randômico de 128, 256, ou de 512 bits. Tudo mais na carteira HD é derivado deterministicamente a partir da sua semente raiz, o que torna possível recriar a carteira HD completa a partir daquela semente em qualquer carteira HD compatível. Isso faz com que seja fácil realizar backup, restaurar, exportar e importar carteiras HD contendo milhares ou até mesmo milhões de chaves, simplesmente transferindo apenas a semente raiz. A semente raiz é mais frequentemente representada por uma *sequência mnemônica de palavras*, tal como descrito na seção anterior Códigos Mnemônicos de Palavras, para tornar mais fácil para as pessoas transcrever e armazenar.

O processo de criar as chaves e códigos de encadeamento mestre para uma carteira HD é demonstrado em Criando chaves mestres e código de cadeia a partir da semente raiz.

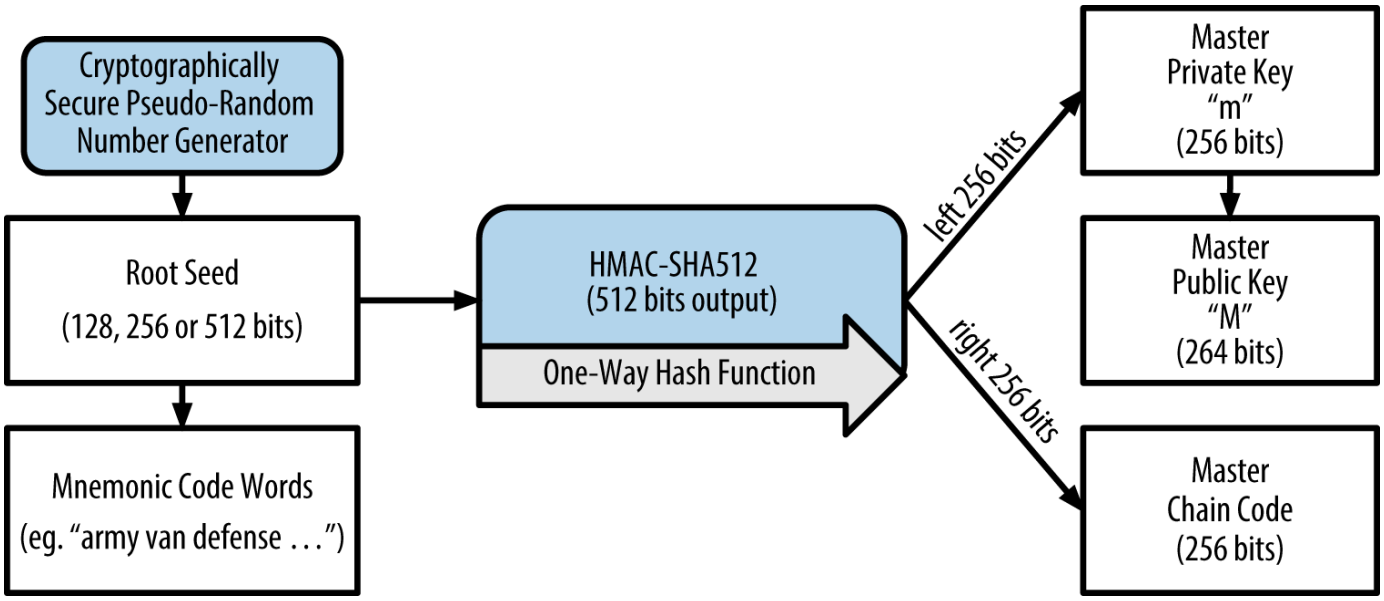


Figure 10. Criando chaves mestres e código de cadeia a partir da semente raiz

A semente raiz é usada como input no algoritmo HMAC-SHA512 e o hash resultante é usado para criar uma *chave privada mestre* (m) e um *código de corrente mestre*. A chave privada mestre (m) então gera a chave pública mestre correspondente (M), usando o processo normal de multiplicação da curva elíptica m * G que nós vimos anteriormente nesse capítulo. O código de corrente é usado para criar entropia na função que cria as chaves filhas a partir das chaves pais, como nós iremos ver na próxima seção.

Derivação da chave privada filha

As carteiras hierárquicas determinísticas usam uma função de *derivação de chave filha* ("child key derivation", CKD) para derivar chaves filhas a partir das chaves pais.

As funções de derivação de chave filha são baseadas em uma função de hash de sentido único, que combina:

- Uma chave privada pai ou uma chave pública (chava não-comprimida ECDSA)
- Uma semente chamada de código de corrente (256 bits)
- Um número índice (32 bits)

O código da corrente é usado para introduzir dados aparentemente aleatórios ao processo, fazendo com que o índice não seja suficiente para derivar outras chaves filhas. Logo, o fato de se ter uma chave filha não torna alguém capaz de descobrir suas chaves irmãs, a menos que a pessoa tenha também o código da corrente. A semente do código da corrente inicial (no nível da raiz da árvore) é feito a partir de dados aleatórios, enquando os códigos de corrente subsequentes são derivadas de cada código de corrente pai.

Esses três ítens são combinados e são usados para fazer um hash, gerando as chaves filhas.

A chave pública pai, o código de corrente e o número índice são combinados e sofrem hash com o algoritmo HMAC-SHA512 para produzir um hash de 512-bit. O hash resultante é dividido em duas metades. A metade da direita de 256 bits do output do hash se torna o código de corrente da filha. A metade da esquerda de 256 bits do hash e o número índice são adicionados à chave privada pai para produzir a chave privada filha. Em [CKDpriv], nós vimos isso ilustrado com o índice definido em 0 para produzir a 0º (primeiro no índice) filha do pai.

1. Estendendo uma chave privada pai para criar uma chave privada filha image::images/msbt_0411.png["ChildPrivateDerivation"]

Mudar o índice nos permite estender a chave pai e criar a outra chave filha na sequência. Por exemplo, Filha 0, Filha 1, Filha 2, etc. Cada chave pai pode ter 2 bilhões de chaves filha.

Repetindo o processo em um nível abaixo da árvore, cada filha pode virar um pai e criar suas próprias filhas, em um número infinito de gerações.

Usando as chaves filhas derivadas

As chaves privadas filhas são indistinguíveis das chaves não-determinísticas (aleatórias). Como a função de derivação é uma função de via única, a chave filha não pode ser usada para se encontrar a chave pai. A chave filha também não pode ser usada para se encontrar nenhuma chave irmã. Se você tem a chave filha *n*, você não é capaz de descobrir suas irmãs, como a filha n–1 ou a filha n+1, assim como nenhuma outra filha que faça parte da sequência. Somente a chave pai e o código de cadeia podem derivar todas as filhas. Sem o código de corrente da filha, a chave filha também não pode ser usada para derivar nenhuma chave neta. Você precisa ter tanto a chave privada filha quando o código de corrente da filha para poder iniciar um novo ramo e derivar as chaves netas.

Então para que serve uma chave filha privada isolada? Ela pode ser usada para fazer uma chave pública e um endereço bitcoin. Ou seja, ela pode ser usada para assinar transações que gastem qualquer valor que aquele endereço tenha recebido.

TIP Uma chave privada filha, a chave pública e o endereço bitcoin correspondentes são todos indistinguíveis de chaves e endereços criados aleatoriamente. Ninguém sabe que elas fazem parte de uma sequência, exceto a função de carteira HD que as criou. Uma vez criadas, elas operam exatamente como chaves "normais".

Chaves estendidas

Como nós vimos anteriormente, a função de derivação de chave pode ser usada para criar chaves filhas em qualquer nível da árvore, baseando-se em três entradas (inputs): uma chave, um código de corrente e o índice da filha desejada. Os dois ingredientes essenciais são a chave a o código de cadeia, que combinados são chamados de *chave estendida*. O termo "chave estendida" também poderia ser considerado como uma "chave estendível", pois esse tipo de chave pode ser usado para derivar chaves filhas.

As chaves estendidas são armazenadas e representadas basicamente como uma concatenação da chave de 256-bit e do código de corrente de 256-bit em uma sequência de 512-bit. Existem dois tipos de chaves estendidas. A chave priva estendida é a combinação de uma chave privada e do código de cadeia, e pode ser usada para derivar as chaves privadas filhas (e, a partir delas, as chaves públicas filhas). Uma chave pública estendida é uma chave pública e um código de corrente, que pode ser usada para criar as chaves públicas filhas, como descrito em Gerando uma Chave Pública.

Imagina uma chave estendida como a raiz de um ramo na estrutura de árvore da carteira HD. Tendo a raiz do ramo, você pode derivar o resto do ramo. A chave privada estendida pode criar um ramo completo, enquanto a chave pública estendida só é capaz de criar um ramo de chaves públicas.

TIP Uma chave estendida consiste de uma chave privada ou chave pública e um código de corrente. Uma chave estendida pode criar chaves filhas, gerando a sua própria remificação na estrutura em árvore. Compartilhar uma chave estendida dá acesso a toda a ramificação.

As chaves estendidas são codificadas com Base58Check, para exportar e importar facilmente entre

diferentes carteiras compatíveis com o BIP-0032. A codificação Base58Check para as chaves estendidas usa um número de versão especial que resulta no prefixo "xprv" e "xpub" quando codificada em caracteres Base58, para torná-los mais facilmente reconhecíveis. Como a chave estendida é de 512 ou 513 bits, ela é muito mais longa do que qualquer um dos strings codificados em Base58Check que nós já vimos anteriormente.

Esse é um exemplo de uma chave privada estendida, codificada em Base58Check:

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMK
Uga5biW6Hx4tws2six3b9c
```

Essa é a chave pública estendida correspondente, também codificada em Base58Check:

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJunSDMst
weyLXhRgPxdp14sk9tJPW9
```

Derivação de chaves públicas filhas

Como mencionado anteriormente, uma característica muito útil das carteiras determinísticas hierárquicas é a habilidade de derivar chaves públicas filhas a partir de chaves públicas pais, *sem* que seja necessário saber as chaves privadas. Isso fornece duas maneiras de derivar um chave pública filha: a partir da chave privada filha, ou diretamente a partir da chave pública pai.

Portanto, uma chave pública estendida pode ser usada para derivar todas as chaves *públicas* (e somente as chaves públicas) naquela ramificação da estrutura da carteira HD.

Esse atalho pode ser usado para criar implantações somente-chave-pública muito seguras, onde um servidor ou aplicação possui uma cópia da chave pública estendida e nenhuma chave privada. Esse tipo de implantação pode produzir um número infinito de chaves públicas e endereços bitcoin, mas não pode gastar nenhum dos bitcoins que forem enviados para esses endereços. Enquanto isso, em outro servidor mais seguro, a chave privada estendida pode derivar todas as cahves privadas correspondentes para assinar as transações e gastar os bitcoins.

Uma utilização comum dessa solução é a instalação de uma chave pública estendida em um servidor web que seja utilizada em um serviço de comércio eletrônico. O servidor web pode usar a função de derivação de chave pública para cirar um novo endereço bitcoin a cada transação (por exemplo, para um carrinho de compras de um consumidor). O servidor web não irá ter nenhuma chave privada que seriam vulneráveis a roubo. Sem as carteiras HD, a única maneira de se fazer isso seria gerar milhares de endereços bitcoin em um outro servidor seguro, e então enviá-los antecipadamente para o servidor de comércio eletrônico. Essa abordagem não é prática, pois requer uma manutenção constante para evitar que as chaves do servidor de comércio eletrônico sejam todas gastas e "acabem".

Outra aplicação comum dessa solução é o uso de carteiras de harware ou de armazenamento frio. Nesse cenário, a chave privada estendida pode ser armazenada em uma carteira de papel ou em um

dispositivo de hardware (como a carteira de hardware Trezor), enquanto a chave pública estendida pode ser mantida online. O usuário pode criar endereços de "recebimento" à vontade, pois as suas chaves privadas estão armazenadas de maneira segura em um ambiente offline. Para gastar os fundos, o usuário pode usar a chave privada estendida em um cliente bitcoin que assine a transação online, ou ao assinar as transações em um dispositivo de carteira de hardware (por exemplo, o Trezor). Estendendo uma chave pública pai para criar uma chave pública filha ilustra o mecanismo para estender uma chave pública pai para derivar as chaves públicas filhas.

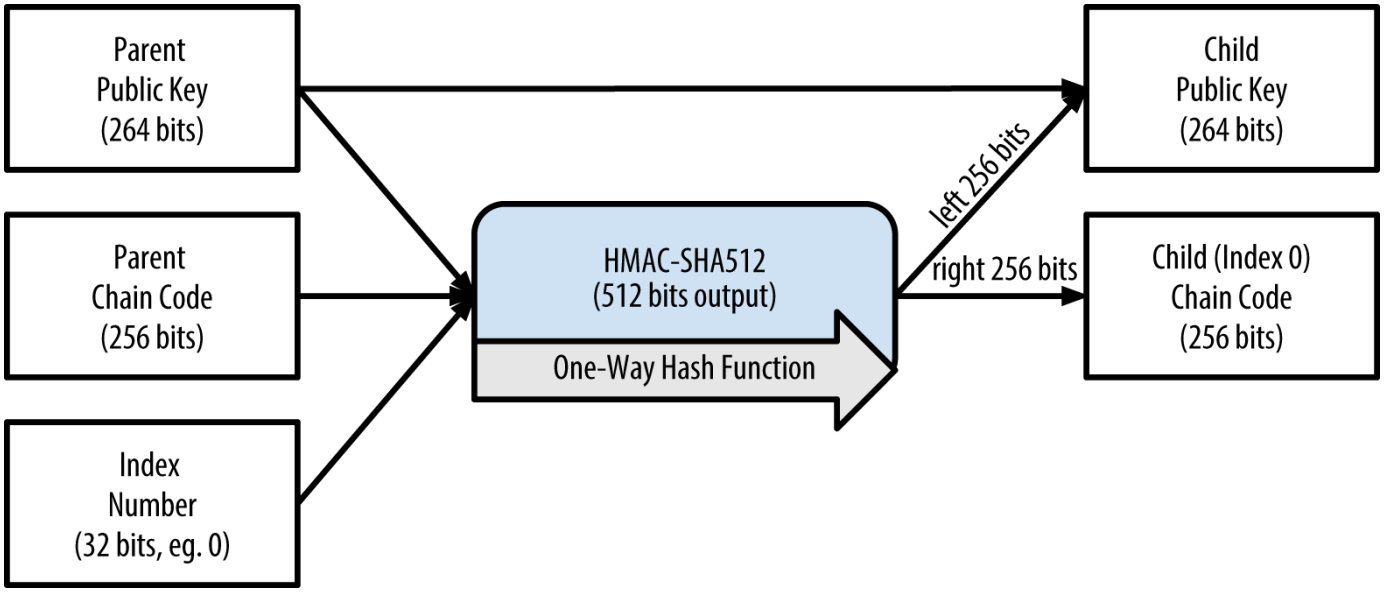


Figure 11. Estendendo uma chave pública pai para criar uma chave pública filha

Derivação endurecida da chave filha

A habilidade de derivar um ramo de chaves públicas a partir de uma chave pública estendida é muito útil, mas isso traz um risco em potencial. Ter acesso a uma chave pública estendida não dá acesso ás chaves privadas filhas. No entanto, como a chave pública estendida contém o código de corrente, se a chave privada filha é conhecida, ou vazou de alguma maneira, ela pode ser usada com o código de corrente para derivar todas as outras chaves privadas filhas. Uma única chave privada filha vazada, junto com um código de corrente pai, é suficiente para revelar todas as chaves privadas de todas as filhas. Pior ainda, a chave privada filha junto com o código de corrente pai pode ser usado para deduzir a chave privada pai.

Para evitar esse risco, as carteiras HD usam uma função de derivação alternativa chamada *derivação endurecida* ("hardened derivation"), que "desfaz" a relação entre a chave pública pai e o código de corrente filho. A função de derivação endurecida usa a chave privada pai para derivar o código de corrente filho, ao invés da chave pública pai. Isso cria um "firewall" na sequência pai/filha, com um código de corrente que não pode ser usado para comprometer uma chave privada pai ou irmã. A função de derivação endurecida parece quase idêntica à derivação de chave privada filha normal, com a exceção de que a chave privada pai é usada como input da função de hash, ao invés da chave pública pai, como demonstrado no diagrama em Derivação endurecida de uma chave filha; omitindo a chave pública pai.

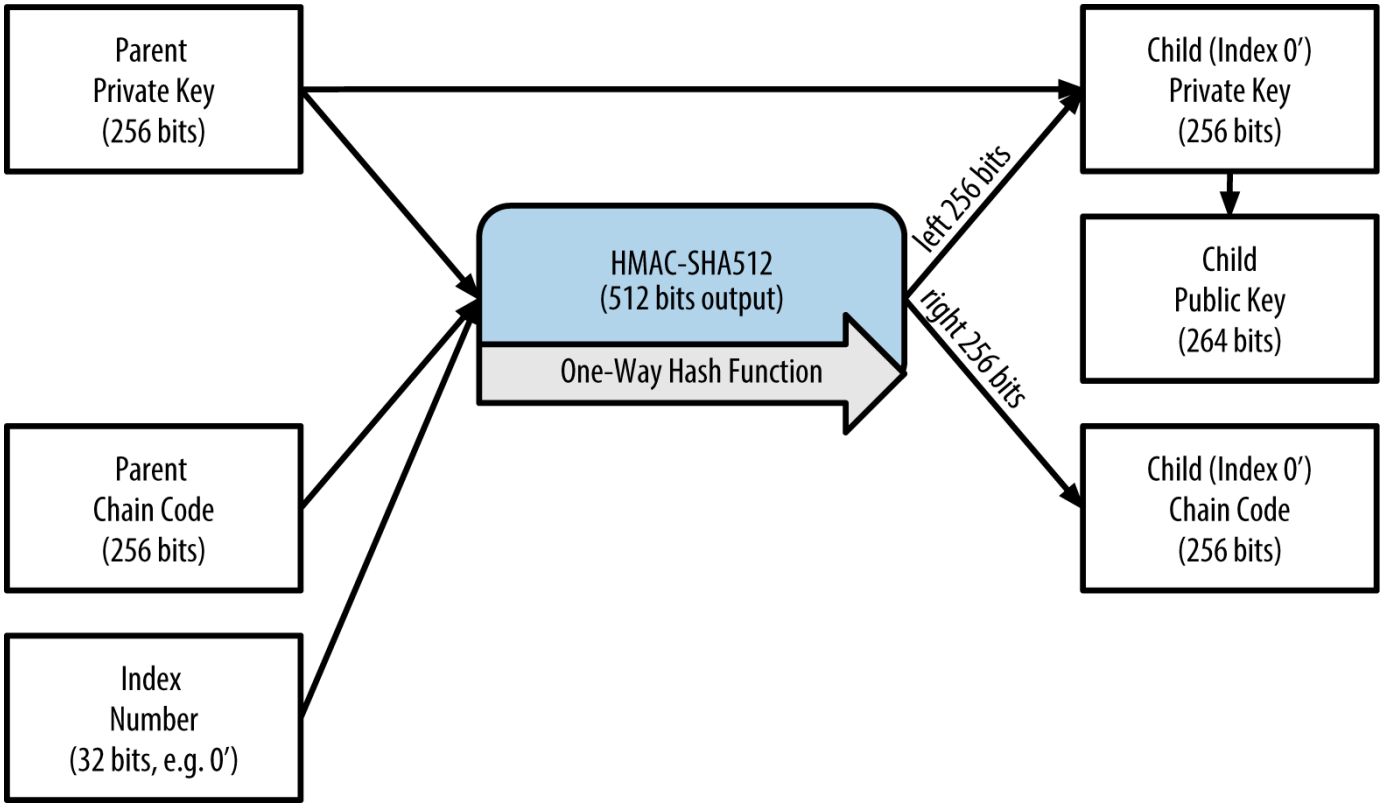


Figure 12. Derivação endurecida de uma chave filha; omitindo a chave pública pai

Quando a função de derivação privada endurecida é utilizada, a chave privada filha e o código de cadeia resultantes são completamente diferentes do que iriam resultar a partir de uma função de derivação normal. O "ramo" de chaves resultante poderia ser usada para produzir chaves públicas estendidas que não são vulneráveis, pois o código de corrente que elas contém não pode ser hackeado para revelar as chaves privadas. A derivação endurecida é portanto usada para criar uma lacuna (um "gap") na árvore acima do nível onde as chaves públicas estendidas são usadas.

Em termos simples, se você quer usar a conveniência de uma chave pública estendida para derivar ramos das chaves públicas, sem se expor ao risco de deixar o código de corrente vazar, você deveria derivá-la a partir de um pai endurecido, ao invés de um pai normal. Como uma prática recomendada, as filhas do nível-1 das chaves mestre são sempre derivadas através da derivação endurecida, para evitar o comprometimento das chaves mestre.

Números de índice para a derivações normal e endurecida

O número de índice usado na função de derivação é um número de 32-bit. Para diferenciar facilmente entre as chaves derivadas através da função de derivação normal das chaves derivadas através da derivação endurecida, esse número índice é dividido em duas faixas. Os números de índice entre 0 e 2^{31}–1 (0x0 a 0x7FFFFFFF) são usados *apenas* para a derivação normal. Os números de índice entre 2^{31} e 2^{32}–1 (0x80000000 a 0xFFFFFFFF) são usados *apenas* para a derivação endurecida. Portanto, se o número de índice é menor do que 2^{31}, isso significa que a filha é normal, enquanto se o número de índice é maior ou igual a 2^{31}, a filha é endurecida.

Para tornar o número de índice mais fácil de ler e exibir, o número de índice para as filhas

endurecidas é exibido iniciando do zero, mas com um apóstrofo. A primeira chave filha normal é portanto exibida como um 0, enquanto a primeira chave filha endurecida (índice 0x80000000) é exibida como <markup>0'</markup>. Na sequência, a segunda chave endurecida teria o índice 0x80000001 e seria exibida como 1', e assim por diante. Quando você se deparar com um índice de carteira HD i', isso significa 2³¹+i.

Identificador (caminho) da chave de uma carteira HD

As chaves em uma carteira HD são identificadas a partir de uma convenção de nomenclatura de "caminho", com cada nível sendo separado por uma barra (/) (ver Exemplos de caminhos de carteira HD). As chaves privadas derivadas a partir da chave privada mestre iniciam com "m". As chaves públicas derivadas a partir da chave pública mestre iniciam com "M". Portanto, a primeira chave privada filha da chave privada mestre é m/0. A primeira chave pública filha é M/0. A segunda neta da primeira filha é m/0/1, e assim por diante.

Os "antepassados" de uma chave são lidos da direita para a esquerda, até você chegar na chave mestre a partir da qual ela foi derivada. Por exemplo, o identificador m/x/y/z descreve a chave que é a zª filha da chave m/x/y, que é a yª filha da chave m/x, que é a xª filha da chave m.

Table 6. Exemplos de caminhos de carteira HD

Caminho HD	Chave descrita
m/0	A primeira (0) chave privada a partir da chave privada mestre (m)
m/0/0	A primeira chave privada neta da primeira filha (m/0)
m/0'/0	A primeira neta normal da primeira filha *endurecida* (m/0')
m/1/0	A primeira chave privada neta da segunda filha (m/1)
M/23/17/0/0	A primeira chave pública tataraneta da primeira bisneta da 18ª neta da 24ª filha

Navegando as estrutura de árvore da carteira HD

A estrutura de árvore da carteira HD oferece uma flexibilidade imensa. Cada chave pai estendida pode ter 4 bilhões de filhas: 2 bilhões de filhas normais e 2 bilhões de filhas endurecidas. Cada uma dessas filhas pode ter outras 4 bilhões de filhas, e assim por diante. A árvore pode ter a profundidade que for necessária, com um número infinito de gerações. Com toda essa flexibilidade, entretanto, se torna bastante difícil navegar nessa árvore infinita. É especialmente difícil transferir carteiras HD entre implementações, porque as possibilidades de organização interna em ramos e subramos são infinitas.

Duas Propostas de Melhorias para o Bitcoin (BIPs) oferecem uma solução para essa complexidade, ao criar alguns padrões propostos para a estrutura das árvores das carteiras HD. A BIP0043 propõe o uso

do índice da primeira filha endurecida como um identificador especial que representa o "propósito" da estrutura de árvore. Baseando-se na BIP0043, uma carteira HD deveria usar apenas um ramo de nível-1 da árvore, com o número de índice identificando a estrutura e o espaço de nomes para o resto da árvore ao definir o seu propósito. Por exemplo, uma carteira HD usando apenas o ramo m/i'/ se destina a um propósito específico, e esse propósito é identificado pelo número índice "i".

Estendendo essa especificação, a BIP0044 propõe uma estrutura de múltiplas contas como número de "proposta" 44' sob a BIP0043. Todas as carteiras HD seguindo a estrutura BIP0044 são identificads pelo fato de que elas somente usaram um ramo da árvore: m/44'/.

O BIP0044 especifica a estrutura como consistindo de cinco níveis de árvore pré-definidos:

m / purpose' / coin_type' / account' / change / address_index

A "proposta" ("purpose") do primeiro nível é sempre definida como 44'. O "tipo_de_moeda" ("coin_type") do segundo nível especifica o tipo de criptomoeda, permitindo a existência de carteiras HD que usem múltiplas moedas, onde cada moeda possui a sua própria subárvore sob o segundo nível. Existem três moedas definidas até hoje: a Bitcoin é m/44'/0', a Bitcoin Testnet é <markup>m/44'/1'</markup>; e a Litecoin é <markup>m/44'/2'</markup>.

O terceiro nível da árvore é a "conta" ("account"), que permite que os usuários subdividam suas carteiras em subcontas separadas lógicas, para fins de contabilidade e organização. Por exemplo, uma carteira HD pode conter duas "contas" bitcion: <markup>m/44'/0'/0'</markup> e <markup>m/44'/0'/1'</markup>. Cada conta é a raiz de sua própria subárvore.

No quarto nível, "troco" ("change"), uma carteira HD tem duas subárvores, uma para criar endereços de recebimento e outra para criar endereços de troco. Note que enquanto os níveis anteriores usavam derivação endurecida, esse nível usa derivação normal. Isso acontece para permitir que esse nível da árvore exporte chaves públicas estendidas para uso em ambientes não-seguros. Os endereços utilizáveis são derivados pela carteira HD como crianças no quarto nível, tornando o quinto nível da árvore o "índice_de_endereço" ("address_index"). Por exemplo, o terceiro endereço de recebimento para pagamentos bitcoin na conta primária seria M/44'/0'/0'/0/2. Exemplos de estrutura de carteira HD da BIP0044 demonstra mais alguns exemplos.

Table 7. Exemplos de estrutura de carteira HD da BIP0044

Caminho HD	Chave descrita
M/44'/0'/0'/0/2	A terceira chave pública de recebimento para a conta bitcoin primária
M/44'/0'/3'/1/14	A décima quinta chave pública para endereço de troco para a quarta conta bitcoin
m/44'/2'/0'/0/1	A segunda chave privada na conta principal Litecoin, para assinar transações

Experimentando com carteiras HD usando Bitcoin Explorer

Usando a ferramenta de linha de comando Bitcoin Explorer que foi apresentada em [ch03_bitcoin_client], você pode fazer um teste gerando e estendendo chaves determinísticas da BIP0032, assim como exibi-las em diferentes formatos:

```
$ bx seed | bx hd-new > m # cria uma nova chave privada mestre a partir da semente e
a armazena no arquivo "m"
$ cat m # mostra a chave privada mestre estendida
xprv9s21ZrQH143K38iQ9Y5p6qoB8C75TE71NfpyQPdfGvzghDt39DHPFpovvtWZaRgY5uPwV7RpEgHs7cvdg
fiSjLjjbuGKGcjRyU7RGGSS8Xa
$ cat m | bx hd-public # gera a chave pública estendida M/0
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJunS
DMstweyLXhRgPxdp14sk9tJPW9
$ cat m | bx hd-private # gera a chave privada estendida m/0
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6G
oNMKUga5biW6Hx4tws2six3b9c
$ cat m | bx hd-private | bx hd-to-wif # mostra a chave privada de m/0 como uma WIF
L1pbvV86crAGoDzqmgY85xURkz3c435Z9nirMt52UbnGjYMzKBUN
$ cat m | bx hd-public | bx hd-to-address # mostra o endereço bitcoin de M/0
1CHCnCjgMNb6digimckNQ6TBVcTWBAmPHK
$ cat m | bx hd-private | bx hd-private --index 12 --hard | bx hd-private --index 4 #
generate m/0/12'/4
xprv9yL8ndfdPVeDWJenF18oiHguRUj8jHmVrqqD97YQHeTcR3LCeh53q5PXPkLsy2kRaqgwoS6YZBLatRZRy
UeAkRPe1kLR1P6Mn7jUrXFquUt
```

Chaves e Endereços Avançados

Nas próximas seções nós iremos ver as formas avançadas de chaves e endereços, como chaves privadas, scripts e endereços de múltiplas assinaturas criptografados, além de endereços vanity e carteiras em papel.

Chaves Privadas Criptografadas (BIP0038)

As chaves privadas sempre tem que ser secretas. A necessidade da *confidencialidade* das chaves privadas é uma obviedade que é bastante difícil de se conseguir na prática, porque isso gera conflito com outro fator de segurança importante que é a *disponibilidade*. Manter a chave privada de maneira secreta é muito difícil quando você precisa armazenar backups de uma chave privada para evitar perdê-la. Uma chave privada armazenada em uma carteira que está criptografada por uma senha pode ser segura, mas é necessário fazer um backup da carteira. Às vezes, os usuários precisam mover suas chaves de uma carteira para outra — para fazer atualizações ou substituir o software de carteira, por exemplo. Os backups das chaves privadas também podem ser armazenados em um papel (veja Carteiras em papel) ou em uma mídia de armazenamento externo, como um pendrive USB. Mas o que

acontece se o backup for roubado ou perdido? Esses objetivos de segurança conflitantes levaram à criação de um padrão conveniente e portátil de criptografar as chaves privadas de uma maneira que possa ser entendida por muitos clientes e carteiras bitcoins, que foi padronizado pela Proposta de Melhoria do Bitcoin 38 ou BIP0038 (veja [bip0038]).

A BIP0038 propõe um padrão comum para criptografar as chaves privadas com uma frase secreta e codificá-las com Base58Check, de maneira que elas possam ser armazenadas de maneira segura em backups, movidas seguramente entre carteiras ou mantidas sob qualquer outras condições onde a chave possa ser exposta. O padrão para criptografia usa o Padrão Avançado de Criptografia (Advanced Encryption Standard, AES), um padrão estabelecido pelo Instituto Nacional de Padrões e Tecnologia (National Institute of Standards and Technology, NIST) e usado amplamente em implementações de criptografia de dados para aplicações comerciais e militares.

Um esquema de criptografia usando a BIP0038 usa como input uma chave privada de bitcoin, geralmente codificada em Wallet Import Format (WIF), como uma string Base58Check com um prefixo de "5". Adicionalmente, o esquema de criptografia com BIP0038 recebe uma frase secreta—uma senha longa—geralmente composta por várias palavras ou uma string complexa de caracteres alfanuméricos. O resultado do esquema de criptografia com BIP0038 é uma chave privada criptografada em Base58Check que começa com o prefixo 6P. Se você enxergar uma chave que inicia com 6P, isso significa que ela é criptografada e requer uma senha para poder convertê-la (descriptografá-la) de volta em uma chave privada formatada em WIF (com prefixo 5) que pode ser usada em qualquer carteira. Muitos aplicativos de carteira agora reconhecem as chaves privadas criptografadas em BIP0038 e irão solicitar do usuário uma senha para descriptografá-las e importar a chave. Aplicativos de terceiros, como o Bit Address (Aba Detalhes da Carteira), que é muito útil e baseado em browser, podem ser usados para descriptografar as chaves BIP0038.

As chaves criptografadas em BIP0038 são mais utilizadas em carteiras de papel que podem ser usadas para fazer backup de chaves privadas em um pedaço de papel. Se o usuário escolher uma frase secreta forte, uma carteira de papel com chaves criptogradas em BIP0038 é uma maneira excelente e muito segura de se criar um armazenamento offline de bitcoin (também conhecido como "armazenamento frio").

Teste as chaves criptografadas em Exemplo de chave privada criptograda em BIP0038 usando o site bitaddress.org para ver como é possível obter a chave descriptografada inserindo-se a frase secreta.

Table 8. Exemplo de chave privada criptograda em BIP0038

Chave Privada (WIF)	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Frase secreta	MinhaFraseSecretaDeTeste
Chave Criptografada (BIP0038)	6PRTHL6mWa48xSopbU1cKrVjpKbBZxcLRRCdctLJ3z5yxE87MobKoXdTsJ

Hash Pay-to-Script (P2SH) e Endereços de múltiplas assinaturas (Multi-Sig)

Como nós sabemos, os endereços bitcoin tradicionais começam com o número "1" e são derivados a partir da chave pública, que é derivada a partir da chave privada. Embora qualquer pessoa pode enviar bitcoins para um endereço "1", esse bitcoin só pode ser gasto ao se apresentar a assinatura da chave privada e o hash da chave pública correspondentes.

Os endereços bitcoin que começam como número "3" são endereços de hash-de-pagamento-para-script (pay-to-script hash, P2SH), e eles às vezes são erroneamente chamados de endereços de múltiplas assinaturas ou multi-sig. Ao invés de definir o dono da chave pública como o hash, eles definem os beneficiários de uma transação bitcoin como o hash de um script. Essa funcionalidade foi incluída em Janeiro de 2012 com a Proposta de Melhoria do Bitcoin 16, ou BIP0016 (ver [bip0016]), e vem sendo amplamente adotada, pois proporciona uma oportunidade de adicionar funcionalidade para o próprio endereço em si. Ao contrário das transações que "enviam" fundos para endereços bitcoins "tradicionais" que começam com 1, também conhecidos como "hash-de-pagamento-para-chave-pública" ("pay-to-public-key-hash", P2PKH), os fundos enviados para endereços que começam com "3" exigem como prova de posse algo além da apresentação de um hash de chave pública e de uma assinatura de chave privada. As exigências são definidas dentro do script, no momento em que o endereço é criado, e todos os inputs desse endereço serão entravados com as mesmas exigências.

Um endereço com hash pay-to-script é criado a partir de um script de transação, que define quem pode gastar um output de transação (para maiores detalhes, veja [p2sh]). A codificação de um endereço com hash pay-to-script envolve a utlização da mesma função de hash duplo que é usada durante a criação de um endereço bitcoin, porém ela é aplicada no script ao invés de ser aplicada na chave pública:

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

O "hash de script" resultante é codificado em Base58Check com um prefixo de versão de 5, que resulta em um endereço codificado que se inicia com um 3. Um exemplo de endereço P2SH é 3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM, que pode ser derivado usando os comandos script-encode, sha256, ripemd160 e base58check-encode (ver [libbitcoin]) do Bitcoin Explorer, como demonstrado a seguir:

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

TIP O P2SH não é necessariamente o mesmo que uma transação de múltiplas assinaturas padrão. Um endereço P2SH *geralmente* representa um script de múltiplas assinaturas, mas ele também pode representar um script codificando outros tipos de transações.

Endereços de múltiplas assinaturas e P2SH

Atualmente, a implementação mais comum da função P2SH é o script de endereço de múltiplas assinaturas. Como o nome sugere, o script subjacente exige mais do que uma assinatura para provar a posse e, portanto, gastar os fundos. A função de múltiplas assinaturas do bitcoin é projetada para exigir M assinaturas (também conhecido como o "limiar") de um total de N chaves, também conhecido como um assinatura múltipla M-de-N, onde M é igual ou menor do que N. Por exemplo, Bob, o dono da cafeteria de [ch01_intro_what_is_bitcoin] poderia usar um endereço de múltiplas assinaturas exigindo 1-de-2 assinaturas a partir da chave que o pertence e uma chave que pertence à sua esposa, garantindo que qualquer um deles poderia fazer uma assinatura para gastar um output de transação travado em seu endereço. Isso seria semelhante a uma "conta conjunta" em um banco tradicional, onde qualquer um do casal poderia gastar com uma única assinatura. Ou Gopesh, o web designer que o Bob contratou para criar um website, poderia ter um endereço de múltiplas assinaturas do tipo 2-de-3 para o seu negócio, para garantir que nenhum fundo pode ser gasto a menos que dois de seus sócios assinem uma transação.

Nós iremos explorar como criar transações que gastam fundos a partir de endereços P2SH (e de múltiplas assinaturas) em [transactions].

Endereços Vanity

Endereços vanity são endereços bitcoin válidos que contém mensagens possíveis de serem lidas por pessoas. Por exemplo, 1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33 é um endereço válido que contém as letras formando a palavra "Love" como as primeiras letras de Base-58. Os endereços vanity exigem a criação e testagem de bilhões de chaves privadas candidatas, até que uma derive um endereço bitcoin o padrão desejado. Embora existem algumas otimizações no algoritmo de geração vanity, o processo essencialmente envolve selecionar uma chave privada aleatoriamente, derivar a chave pública, derivar o endereço bitcoin e verificar se ele corresponde ao padrão vanity desejado, repetindo esse processo bilhões de vezes até que uma correspondência seja encontrada.

Assim que uma correspondência de um endereço vanity com um padrão desejado for encontrada, a chave privada a partir da qual ele foi derivado agora pode ser usada pelo dono para gastar seus bitcoins da mesma maneira que qualquer outro endereço. Os endereços vanity não são mais ou menos seguros do que qualquer outro endereço. Eles dependem da mesma Curva Elíptica de Criptografia (ECC) e Algoritmo de Hash Seguro (SHA), como qualquer outro endereço. Não é mais fácil descobrir a chave privada de um endereço começando com um padrão vanity, em comparação com qualquer outro endereço.

No [ch01_intro_what_is_bitcoin], nós apresentamos a Eugênia, uma diretora de um centro de caridade para crianças nas Filipinas. Vamos dizer que a Eugênia está organizando uma campanha para arrecadar fundos em bitcoin e quer usar um endereço bitcoin vanity para divulgar em sua campanha. A Eugênia irá criar um endereço vanity que comece com "1Kids" para divulgar a campanha do centro de caridade. Vamos ver como esse endereço vanity será criado e o que isso significa para a segurança do centro de caridade da Eugênia.

Gerando endereços vanity

É importante notar que um endereço bitcoin é simplesmente um número representado por símbulos no alfabeto Base58. Uma busca por um padrão como "1Kids" pode ser vista como uma buscar por um endereço na faixa de 1Kids11111111111111111111111111111 até 1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzz. Existem aproximadamente 58^{29} (aproximadamente $1,4 * 10^{51}$) endereços nessa faixa, todos começando com "1Kids". A A amplitude de endereços vanity iniciando com "1Kids" mostra a faixa de endereços que tem o prefixo 1Kids.

Table 9. A amplitude de endereços vanity iniciando com "1Kids"

De	1Kids11111111111111111111111111111
	1Kids11111111111111111111111111112
	1Kids11111111111111111111111111113
	...
Até	1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzzz

Vamos ver o padrão "1Kids" como um número e ver quão frequentemente nós encontraríamos esse padrão em um endereço bitcoin (ver A frequência de um padrão vanity (1KidsCharity) e a média de tempo-para-descobrir em um PC desktop). Um computador desktop PC médio, sem nenhum hardware especializado, é capaz de buscar cerca de 100.000 chaves por segundo.

Table 10. A frequência de um padrão vanity (1KidsCharity) e a média de tempo-para-descobrir em um PC desktop

Comprimento	Padrão	Frequência	Tempo médio de busca
1	1K	1 em 58 chaves	< 1 milisegundo
2	1Ki	1 em 3.364	50 milisegundos
3	1Kid	1 em 195.000	< 2 segundos
4	1Kids	1 em 11 milhões	1 minuto
5	1KidsC	1 em 656 milhões	1 hora
6	1KidsCh	1 em 38 bilhões	2 dias
7	1KidsCha	1 em 2,2 trilhões	3–4 meses
8	1KidsChar	1 em 128 trilhões	13–18 anos
9	1KidsChari	1 em 7 quadrilhões	800 anos
10	1KidsCharit	1 em 400 quadrilhões	46.000 anos
11	1KidsCharity	1 em 23 quintilhões	2,5 milhões de anos

Como você pode ver, a Eugênia não irá criar um enderço vanity "1KidsCharity" tão cedo, mesmo que

ela tivesse acesso a milhares de computadores. Cada caractere adicional aumenta a dificuldade em um fator de 58. Os padrões com mais de sete caracteres são geralmente encontrados utilizando-se hardware especializado, como desktops montados de maneira customizada com múltiplas placas de vídeo (GPUs). Esses hardwares geralmente são equipamentos antigos que eram usados para a mineração do bitcoin, mas se tornaram obsoletos, e agora são usados para buscar endereços vanity. As buscas por endereços vanity em hardwares usando múltiplas placas de vídeo são muito mais rápidas do que as realizadas em um computador desktop comum.

Outra maneira de encontrar um endereço vanity é pagando para um pool de mineradores vanity, como por exemplo o Vanity Pool. Um pool é um serviço que permite que aqueles que tenham hardware com placas de vídeo (GPU) possam ganhar bitcoin por buscarem endereços vanity para outros usuários. Por uma pequena quantia (0,01 bitcoin, ou aproximadamente 5 dólares, na época que esse livro foi escrito), a Eugênia pode pagar para alguém fazer a busca de um endereço vanity com sete caracteres em poucas horas, ao invés de ter que usar seu próprio computador durante meses.

A geração de um endereço vanity é um exercício de força-bruta: são criadas chaves aleatórias, e verifica-se cada uma se corresponde ao padrão desejado. As chaves são criadas repetidamente até que se consiga atingir o padrão desejado. Minerador de endereço vanity mostra um exemplo de um "minerador vanity", um programa projetado para buscar endereços vanity, escrito em C++. O exemplo usa a livraria libbitcoin, que nós apresentamos em [alt_libraries].

Example 8. Minerador de endereço vanity

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

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

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

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

    // Loop continuously...
    while (true)
    {
```

```
        // Generate a random secret.
        bc::ec_secret secret = random_secret(engine);
        // Get the address.
        std::string address = bitcoin_address(secret);
        // Does it match our search string? (1kid)
        if (match_found(address))
        {
            // Success!
            std::cout << "Found vanity address! " << address << std::endl;
            std::cout << "Secret: " << bc::encode_hex(secret) << std::endl;
            return 0;
        }
    }
    // Should never reach here!
    return 0;
}

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

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

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

```
    return true;
}
```

NOTE O exemplo acima usa std::random_device. Dependendo da implementação ele pode repletir um gerador de número aleatório criptograficamente seguro fornecido pelo sistema operacional. No caso de um sistema operacional do tipo Unix, como o Linux, ele o obtém a partir de /dev/urandom. O gerador de número aleatório utilizado aqui é usado apenas para fins demonstrativos, *não* sendo apropriado para gerar chaves de bitcoin com qualidade suficiente para ser usadas em produção, já que ele não é implementado com segurança suficiente.

O código de exemplo deve ser compilado usando um compilador C e vinculado à livraria libbitcoin (que deve ser previamente instalada no sistema). Para executar o exemplo, rode o executável vanity-miner++ sem nenhum parâmetro (ver Compilando e executando o exemplo de minerador vanity) e ele irá tentar encontrar um endereço vanity que se inicie com "1kid".

Example 9. Compilando e executando o exemplo de minerador vanity

```
$ # Compila o código com g++
$ g++ -o vanity-miner vanity-miner.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Executa o exemplo
$ ./vanity-miner
Endereço vanity encontrado! 1KiDzkG4MxmovZryZRj8tK81oQRhbZ46YT
Secret: 57cc268a05f83a23ac9d930bc8565bac4e277055f4794cbd1a39e5e71c038f3f
$ # Executa-o novamente para um resultado diferente
$ ./vanity-miner
Endereço vanity encontrado! 1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUFn
Secret: 7f65bbbbe6d8caae74a0c6a0d2d7b5c6663d71b60337299a1a2cf34c04b2a623
# Use "time" para ver quanto tempo demora para se encontrar um resultado
$ time ./vanity-miner
Endereço vanity encontrado! 1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM
Secret: 2a802e7a53d8aa237cd059377b616d2bfcfa4b0140bc85fa008f2d3d4b225349

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

O código de exemplo irá levar alguns segundos para encontrar uma correspondência ao padrão de três caracteres "kid", como nós podemos ver quando usamos o comando time do Unix para medir o tempo de execução. Mude o padrão search no código-fonte e veja como pode ser mais demorado para se conseguir padrões de quatro ou cinco caracteres!

Segurança do endereço vanity

Os endereços vanity podem ser usados tanto para melhorar *quanto* para burlar medidas de segurança; eles são literalmente uma faca de dois gumes. Quando usado para melhorar a segurança, um endereço diferenciado torna mais difícil para que hackers substituam o seu próprio endereço, fazendo com que seus consumidores a pagarem para eles, ao invés de para você. Infelizmente, os endereços vanity também tornam possível que qualquer um cria um endereço que se *pareça* com qualquer endereço aleatório, ou mesmo outro endereço vanity, dessa maneira enganando seus consumidores.

A Eugênia poderia divulgar um endereço gerado aleatoriamente (por exemplo, 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy) para que as pessoas enviem suas doações. Ou ela poderia gerar um endereço vanity que começa com 1Kids, para torná-lo mais atraente.

Em ambos os casos, um dos riscos de se usar um endereço fixo único (ao invés de um endereço dinâmico separado para cada doador) é que o ladrão pode ser capaz de se infiltrar em seu site e substituí-lo pelo seu próprio endereço, desviando as doações que você receberia. Se você divulgou seu endereço de doação em vários lugares diferentes, seus usuários podem inspecionar visualmente o endereço antes de fazer um pagamento para se assegurarem que ele é o mesmo que eles viram em seu site, e-mail ou flyer. No caso de um endereço aleatório como 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy, o usuário comum talvez inspecionará apenas os primeiros caracteres "1J7mdg" e estará satisfeito se o endereço corresponder. Usando um gerador de endereços vanity, alguém pode roubá-lo ao rapidamente criar um endereço que se pareça com o verdadeiro, com apenas os primeiros caracteres iguais, como demonstrado em Gerando endereços vanity para corresponder a um endereço aleatório.

Table 11. Gerando endereços vanity para corresponder a um endereço aleatório

Endereço Aleatório Original	1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
Vanity (correspondência de 4 caracteres)	1J7md1QqU4LpctBetHS2ZoyLV5d6dShhEy
Vanity (correspondência de 5 caracteres)	1J7mdgYqyNd4ya3UEcq31Q7sqRMXw2XZ6n
Vanity (correspondência de 6 caracteres)	1J7mdg5WxGENmwyJP9xuGhG5KRzu99BBCX

Um endereço vanity aumenta a segurança? Se Eugenia gerar o endereço vanity 1Kids33q44erFfpeXrmDSz7zEqG2FesZEN, os usuários provavelmente irão olhar para a palavra com padrão vanity e alguns caracteres além, por exemplo percebendo a parte "1Kids33" do endereço. Isso iria forçar um hacker a gerar um endereço vanity que corresponde a pelo menos seis caracteres (dois a mais), tendo que se esforçar 3.364 vezes mais (58 × 58) do que o que a Eugenia teve que se esforçar para gerar o seu endereço vanity de quatro caracteres. Essencialmente, o esforço que a Eugenia teve que fazer (ou pagar para um pool vanity fazer) acaba exigindo que o hacker produza um padrão vanity mais longo. Se a Eugenia pagar para um pool gerar um endereço vanity de 8 caracteres, o hacker teria que fazer um esforço para produzir um endereço de 10 caracteres, que é inviável de ser feito em um computador pessoal e muito caro para ser obtido, até mesmo em equipamentos de mineração vanity ou em um pool de vanity. O que tem um preço acessível para a Eugenia se torna muito caro para o hacker, especialmente se a recompensa potencial pela fraude não é alta o suficiente para cobrir o custo da geração do endereço vanity.

Carteiras em papel

As carteiras em papel são chaves privadas de bitcoin impressas em papel. Frequentemente as carteiras em papel também incluem o endereço bitcoin para conveniência, mas isso não é necessário, já que eles podem ser derivados a partir da chave privada. As carteiras em papel são uma maneira muito efetiva para se criar backups ou armazenamento offline de bitcoins, também conhecidos como "armazenamento frio". Como mecanismo de backup, uma carteira em papel pode fornecer segurança contra a perda da chave devido a um problema no computador como uma falha no disco rígido, roubo ou o apagamento acidental. Como mecanismo de "armazenamento frio", se as chaves da carteira de papel forem geradas offline e nunca forem armazenadas em um computador ou dispositivo, elas estarão muito mais protegidas contra hackers, key-loggers e outras ameaças onlines.

As carteiras em papel possuem muitos formatos, tamanhos e estilos, mas basicamente elas são apenas uma chave privada e um endereço impressos no papel. A A forma mais simples de uma carteira em papel—uma impressão contendo um endereço bitcoin e a sua chave privada. mostra a forma mais simples de carteira em papel.

Table 12. A forma mais simples de uma carteira em papel—uma impressão contendo um endereço bitcoin e a sua chave privada.

Endereço Público	Chave Privada (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

Carteiras de papel podem ser geradas facilmente utilizando uma ferramenta como um gerador JavaScript local em *bitaddress.org*. Essa página contém todo o código necessário para gerar chaves e carteiras de papel, mesmo quando o computador estiver completamente desconectado da internet. Para usá-la, salve a página HTML em seu disco local ou em um drive USB externo. Desconecte-se da internet e abra o arquivo em um navegador. É ainda melhor se você fizer um boot do seu computador usando um sistema operacional primitivo, como um sistema operacional Linux que faça boot a partir de CD-ROM. Quaisquer chaves geradas offline com essa ferramenta podem ser impressas com uma impressora local usando-se um cabo USB (não use wi-fi), logo, criando uma carteira de papel cujas chaves existem somente no papel e que nunca foram armazenadas em nenhum sistema online. Guarde essas carteiras de papel em um cofre à prova de incêndios e "envie" alguns bitcoins para os endereços bitcoin contidos nelas, para implementar uma solução simples de "armazenamento frio (cold)", porém muito efetiva. Um exemplo de uma carteira de papel simples originada em bitaddress.org demonstra uma carteira de papel gerada a partir do site bitaddress.org.

Figure 13. Um exemplo de uma carteira de papel simples originada em bitaddress.org

A desvantagem desse sistema simples de carteira de papel é que as chaves impressas são vulneráveis a roubos. Um ladrão que tenha acesso ao papel poderá roubar ou fotografar as chaves, podendo gastar os bitcoins vinculados aos endereços. Uma sistema mais avançado de armazenamento de carteira em papel usa as chaves privadas criptografadas em BIP0038. As chaves impressas na carteira de papel são protegidas por uma senha que o dono memorizou. Sem essa senha, as chaves criptografadas são inúteis. No entanto, elas ainda são melhores que uma carteira protegida por uma senha, pois essas chaves nunca estiveram online e ainda precisam ser fisicamente extraídas de um cofre ou outro armazenamento físico seguro. Um exemplo de carteira de papel criptografada no site bitaddress.org. A senha é "test". mostra uma carteira de papel com uma chave privada criptografada (BIP0038) criada no site bitaddress.org.

Figure 14. Um exemplo de carteira de papel criptografada no site bitaddress.org. A senha é "test".

WARNING	Embora você possa depositar fundos várias vezes em uma carteira de papel, o correto é você sacar todos os fundos uma única vez, gastando tudo. Isso acontece porque no processo de destravamento e de gastar os fundos algumas carteiras podem gerar um endereço de troco se você gastar menos que toda a quantia contida na carteira de papel. Adicionalmente, se o computador que você usa para assinar a transação estiver comprometido (hackeado), você correrá o risco de expor a chave privada. Ao gastar todo o saldo da carteira de papel uma única vez, você reduz o risco de sua chave ser comprometida. Se você precisa gastar apenas uma pequena quantidade, envie os fundos remanescentes para uma nova carteira de papel na mesma transação.

As carteiras em papel tem vários designs e tamanhos, além de diferentes características. Algumas são feitas para serem dadas de presente, outras são temáticas, como as de Natal e Ano Novo. Outras são projetadas para armazenar em um cofre do banco com a chave privada escondida em algum lugar, seja através de adesivos opacos com raspadinhas, ou adesivos dobrados e lacrados. As figuras <xref linkend="paper_wallet_bpw" xrefstyle="select: labelnumber"/> a <xref linkend="paper_wallet_spw" xrefstyle="select: labelnumber"/> mostram vários exemplos de carteiras em papel com diferentes características de segurança e backup.

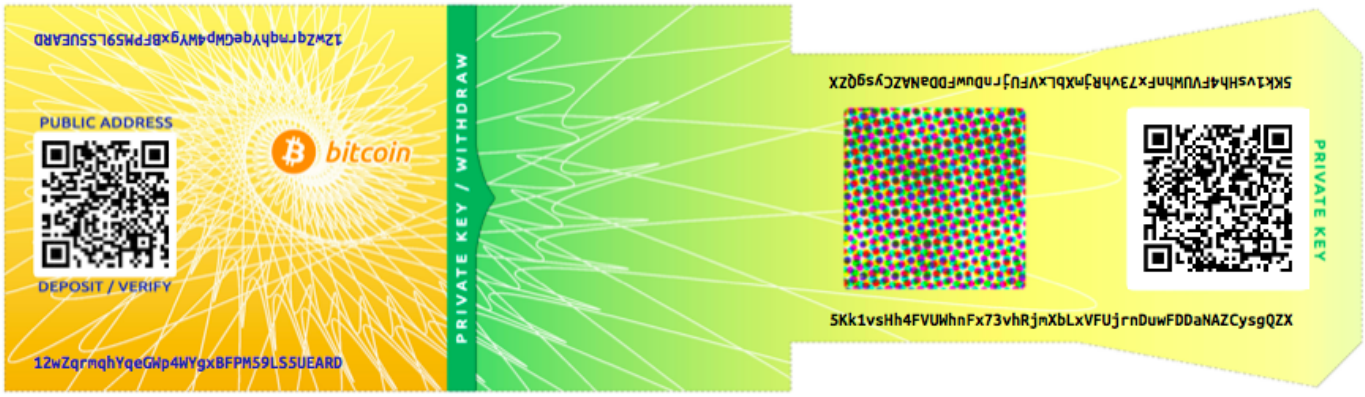


Figure 15. Um exemplo de uma carteira em papel feita em bitcoinpaperwallet.com com a chave privada impressa numa aba dobrável.

Figure 16. Uma carteira de papel do site bitcoinpaperwallet.com com a chave privada escondida.

Outros designs apresentam cópias adicionais da chaves e dos endereços, na forma de canhotos

destacáveis semelhantes a canhotos de ingressos, permitindo que você armazene múltiplas cópias para protegê-las do fogo, enchentes e outros desastres naturais.

Figure 17. Um exemplo de uma carteira em papel com um canhoto contendo cópias adicionais das chaves como backup

Transações

Introdução

As transações são a parte mais importante do sistema bitcoin. Todo o restante no bitcoin é projetado para garantir que as transações possam ser criadas, propagadas na rede, validades e por mim adicionadas ao registro global das transações (a blockchain). As transações são estruturas de dados que codificam a transferência de valor entre os participantes no sistema bitcoin. Cada transação é uma entrada pública na blockchain do bitcoin, o registro geral de dupla-entrada.

Neste capítulo, nós iremos examinar as várias formas de transações, o que elas contêm, como criá-las, como elas são verificadas e como elas se tornam parte do registro permanente de todas as transações.

Ciclo de Vida da Transação

O ciclo de vida de uma transação começa com a criação da transação, também conhecido como *originação*. A transação é então assinada com uma ou mais assinaturas indicando a autorização para gastar os fundos que são indicados pela transação. A transação é então transmitida na rede bitcoin, onde cada nodo (participante) da rede a valida e a propaga até que ela atinja (quase) todos os nodos na rede. Por fim, a transação é verificada por um nodo minerador e é incluída em um bloco de transações que é registrado na blockchain.

Ao ser registrada na blockchain e confirmada por um número subsequente suficiente de blocos (confirmações), a transação é uma parte permanente do registro do bitcoin e é aceita como válida por todos os participantes. Os fundos alocados para um novo dono através da transação agora podem ser gastos em uma nova transação, estendendo a cadeia de possa e iniciando novamente o ciclo de vida de uma transação.

Criando Transações

Pode ser útil pensar em uma transação da mesma maneira que um cheque de papel. Como um cheque, a transação é um instrumento que expressa a intenção de se transferir dinheiro e ela não é visível ao sistema financeiro até que seja enviada para execução/depósito. Como um cheque, quem origina a transaçõa não tem a obrigação de ser aquele que assina a transação.

As transações podem ser criadas online ou offline por qualquer pessoa, mesmo se a pessoa que estiver criando a transação não seja a pessoa que assinará a autorização para gastar os fundos. Por exemplos, um funcionário de contabilidade poderá processar cheques pagáveis pela assinatura do CEO. De maneira similar, um funcionário de contabilidade pode criar transações bitcoin e então fornecê-las para que o CEO aplique suas assinaturas digitais para torná-las válidas. Enquanto um cheque tradicional refere-se a uma conta bancária específica como a fonte para os fundos, uma transação bitcoin refere-se a uma transação prévia como sua fonte, ao invés de uma conta bancária.

Assim que a transação for criada, ela é assinada pelo dono (ou donos) dos fundos de origem. Ela é

propriamente formada e assinada, a transação assinada agora é válida e contém toda a informação necessária para executar a transferência dos fundos. Finalmente, a transação válida tem que atingir a rede bitcoin para que ela possa ser propagada até atingir um minerador para incluí-la no registro público (a blockchain).

Transmitindo Transações para a Rede Bitcoin

Primeiro, uma transação precisa chegar à rede bitcoin para que ela seja propagada e incluída na blockchain. Basicamente, uma transação bitcoin é simplesmente 300 a 400 bytes de dados e precisa atingir qualquer um das dezenas de milhares de nodos bitcoin. Aqueles que enviam não precisam confiar nos nodos que eles usam para transmitir a transação, contanto que eles usem mais de um para garantir a sua propagação. Os nodos não precisam confiar em quem envia, nem precisam saber a "identidade" de quem envia. Como a transação é assinada e não contém informações confidenciais, chaves privadas ou credenciais, ela pode ser transmitida publicamente usando qualquer transporte de rede que seja conveniente. Ao contrário das transações de cartões de crédito, por exemplo, que contém informações sensíveis e só podem ser transmitidas em redes criptografadas, uma transação bitcoin pode ser enviada em qualquer rede. Contanto que a transação possa atingir um nodo bitcoin que a propague para rede bitcoin, não interessa a maneira como ela será transportada para o primeiro nodo.

Portanto, as transações bitcoin podem ser transmitidas para a rede bitcoin através de redes inseguras como WiFi, Bluetooth, NFC, Chirp, códigos de barras ou ao copiar-se e colar a partir de um formulário na internet. Em casos extremos, uma transação bitcoin poderia ser transmitida através de um pacote de rádio, relay satélite ou em ondas curtas usando transmissão burst, espectro amplo ou pulando frequências para evitar detecção e interferências. Uma transação bitcoin poderia ser codificada até mesmo como smileys (emoticons) e postada em um fórum público ou enviada como uma mensagem de texto ou uma mensagem de chat Skype. O Bitcoin transformou o dinheiro em uma estrutura de dados, tornando praticamente impossível que alguém seja impedido de criar e executar uma transação bitcoin.

Propagando Transações na Rede Bitcoin

Assim que a transação é enviada para qualquer nodo conectado à rede bitcoin, a transação será validada por aquele nodo. Se for válida, o nodo irá propagá-la para outros nodos com os quais ele está conectado, e uma mensagem de sucesso irá retornar na mesma hora para quem originou a transação. Se a transação for inválida, o nodo irá rejeitá-la e retornará na mesma hora uma mensagem de rejeição para quem originou a transação.

A rede bitcoin é uma rede par-a-par, o que significa que cada nó bitcoin está conectado a alguns outros nós bitcoin que ele descobre durante a inicialização através do protocolo par-a-par. Toda a rede forma uma emaranhado frouxamente conectado sem uma topologia fixa ou qualuqer estrutura, tornando todos os nós pares iguais. As mensagens, incluindo transações e blocos, são propagadas a de cada nó a todos os pares aos quais ele está conectado, um processo chamado "flooding" (inundação). Uma transação recém validada e injetada em qualquer nó da rede será enviada a todos os nós conectados a ele (vizinhos), cada um dos quais enviará a transação a todos os seus vizinhos, e assim por diante. Dessa maneira, dentro de alguns segundos uma transação válida será propagada numa onda

exponencialmente expandida através da rede até que todos os nós da rede a recebam.

A rede bitcoin é projetada para propagar transações e blocos para todos os nodos de uma maneira eficiente e flexível que seja resistente a ataques. Para prevenir spam, ataques DOS ou outros ataques maliciosos contra o sistema bitcoin, cada nodo valida independentemente cada transação antes de propagá-la adiante. Uma transação mal formada não irá passar por um nodo sequer. As regras através das quais as transações são validadas serão explicadas em maiores detalhes em [tx_verification].

Estrutura da Transação

Uma transação é uma *estrutura de dados* que codifica uma transferência de valor a partir de uma fonte de fundos, chamada de *input*, para um destino, chamado de *output*. Os inputs e outputs de transação não são relacionados a contas ou identidades. Ao invés disso, você deveria imaginá-los como quantidades de bitcoin—pedaços de bitcoin—que são bloqueados com uma senha secreta específica que somente o dono, ou a pessoa que conhece a senha, pode desbloqueá-los. Uma transação contém vários campos, como demonstrado em A estrutura de uma transação.

Table 1. A estrutura de uma transação

Tamanho	Campo	Descrição
4 bytes	Versão	Especifica quais regras essa transação irá seguir
1–9 bytes (VarInt)	Input Counter	Quantos inputs serão incluídos
Variable	Inputs	Um ou mais inputs de transação
1–9 bytes (VarInt)	Output Counter	Quantos outputs são incluídos
Variable	Outputs	Um ou mais outputs de transação
4 bytes	Locktime	Data e hora no formato Unix (timestamp) ou número de bloco

Locktime da Transação

Locktime (tempo de travamento), também conhecido como nLockTime, devido à variável com este nome utilizada no cliente de referência, define o tempo mais precoce que uma transação é valida e pode ser transmitida na rede ou adicionada à cadeia de blocos (blockchain). Ele é definido como zero na maioria das transações, indicando uma propagação e execução imediatas. Se o locktime é diferente de zero e abaixo de 500 milhões, ele é interpretado como uma altura de bloco, significando que a transação não é válida e que ela não será transmitida ou incluída na cadeia de blocos antes que a altura de bloco especificada seja atingida. Se ele for acima de 500 milhões, ele é interpretado como uma timestamp Unix Epoch (segundos desde 1-Jan-1970) e a transação não é valida até que tal tempo seja antingido. As transação com locktime especificando um bloco ou um tempo no futuro precisam ser mantidas pelo sistema de origem e transmitidas à rede blockchain apenas após elas se tornarem válidas. O uso do locktime é o equivalente a assinar um cheque pré-datado.

Inputs e Outputs das Transações

A matéria-prima principal de uma transação bitcoin é um *output de transação não-gasto*, ou UTXO. Os UTXOs são pedaços indivisíveis da moeda bitcoin vinculados a um dono específico, registrados na blockchain, e reconhecidos como unidades de moeda pela rede. A rede bitcoin rastreia todos os UTXOs disponíveis (não-gastos), que atualmente são milhões. Quando um usuário recebe um bitcoin, a quantia é registrada na blockchain como um UTXO. Portanto, o bitcoin de um usuário pode estar disperso como UTXOs entre centenas de transações e centenas de blocos. Como efeito, não existe algo como um armazenamento do saldo de um endereço ou conta bitcoin; existem apenas UTXOs dispersos, vinculados a seus respectivos donos. O conceito de saldo de um usuário de bitcoin é algo criado pelos aplicativos de carteira. A carteira calcula o saldo do usuário ao escanear a blockchain e ao somar todos os UTXOs que pertecem àquele usuário.

TIP	Não há contas correntes ou saldos no bitcoin; existem apenas *outputs de transações não-gastos* (UTXO) espalhados na blockchain.

Um UTXO pode ter um valor arbitrário denominado como um múltiplo de satoshis. Assim como os dólares podem ser divididos para duas casas decimais (os centavos), os bitcoin podem ser dividos até oito casas decimais (os satoshis). Embora o UTXO possa ser qualquer valor arbitrário, uma vez criado ele é indivisível assim como uma moeda que não pode ser cortada pela metade. Se um UTXO é maior do que o valor desejado de uma transação, ele precisa ser totalmente consumido e um troco deve ser gerado na transação. Em outras palavras, se você possui 20 UTXOs de bitcoin, e você quer pagar 1 bitcoin, a sua transação precisa consumir todos os 20 UTXOs de bitcoin e produzir duas saídas (outputs): uma que paga 1 bitcoin para o recipiente e outra que paga 19 bitcoins em troco de volta para a sua carteira. Como consequência disso, a maioria das transações de bitcoins irão gerar troco.

Imagine um consumidora comprando uma bebida de $1,50, pegando sua carteira e tentando achar uma combinação de moedas e notas para atingir o valor de $1,50. A consumidora poderá escolher o

troco exato se disponível (uma nota de um dólar e duas moedas de 25 centavos), ou uma combinação de pequenas denominações (6 moedas de 25 centavos), ou, se necessário, uma unidade maior como uma nota de 5 dólares. Se ela der dinheiro demais, digamos $5, ao dono da loja, ela esperará um troco de $3,50, o qual ela irá colocar em sua carteira e o terá disponível para transações futuras.

De maneira semelhante, uma transação bitcoin precisa ser criada a partir de um UTXO do usuário em quaisquer denominações que o usuário tenha disponível. Os usuários não podem dividir um UTXO pela metade, da mesma maneira que eles não podem cortar uma nota de um dólar pela metade e usá-la no mercado. O aplicativo de carteira do usuário tipicamente irá selecionar entre os UTXOs disponíveis do usuário várias unidades para compor uma quantia maior ou igual ao valor de transação desejado.

Assim como na vida real, a aplicação bitcoin possui várias estratégias para satisfazer a quantia de compra: combinar várias unidades menores, procurar pelo troco exato ou usar uma unidade única maior que o valor da transação e receber o troco. Toda essa administração completa dos UTXOs gastáveis é feita de maneira automática pela carteira do usuário, e os usuários sequer tomam conhecimento dela. A administração das UTXOs só é relevante se você for um programador construindo transações cruas (raw) a partir de UTXOs.

Os UTXOs consumidos por uma transação são chamados de entradas (inputs) de transação, e os UTXOs criados por uma transação são chamados de saídas (outputs) de transação. Dessa maneira, partes de valor do bitcoin movem adiante de dono para dono em uma cadeia de transações que consome e cria UTXOs. As transações consomem UTXOs quando os desbloqueiam com a assinatura do dono atual, e criam UTXOs quando os vinculam ao endereço bitcoin do novo dono.

A exceção para a cadeia de output e input é um tipo especial de transação chamada transação *coinbase*, que é a primeira transação em cada bloco. Essa transação é inserida no bloco pelo minerador "vencedor" e cria novos bitcoins que serão pagos para o minerador como uma recompensa por ter conseguido minerar o bloco. É assim que a oferta monetária do bitcoin é criada durante o processo de mineração, como veremos no [ch8].

TIP	E o que vem primeiro? Inputs ou outputs, a galinha ou o ovo? Falando estritamente, os outputs vem primeiro porque as transações coinbase, que geram os novos bitcoins, não tem inputs e criam outputs a partir do nada.

Outputs de Transações

Toda transação bitcoin cria saídas (outputs), que são gravadas no registro do bitcoin (a cadeia de blocos). Quase todas essas saídas (outputs), com uma exceção (ver Output de dados (OP_RETURN)) criam partes de bitcoin gastáveis chamadas _outputs de transação não-gastos ou UTXO (do inglês Unspent Transaction Outputs), que são então reconhecidas por toda a rede e se tornam disponíveis para que o dono gaste em uma transação futura. Quando se envia um bitcoin para alguém, na verdade se está criando uma output de transação não-gasto (UTXO) registrado para o endereço dessa pessoa e tornando-o disponível para ser gasto.

Os UTXOs são monitorados por todos os nodos completos de bitcoin como um conjunto de dados

chamado de *conjunto UTXO* ("UXTO set") ou *pool UTXO*, que é armazenado em um banco de dados. As novas transações consomem (gastam) um ou mais desses outputs que estão no conjunto UTXO.

Os outputs de transações consistem de duas partes:

- Uma quantidade de bitcoin, denominados em *satoshis*, a menor unidade bitcoin
- Um *script de travamento*, também conhecido como uma "oneração" que "trava" essa quantia ao especificar os requisitos que precisam ser preenchidos para poder se gastar o output.

A linguagem de script da transação, usada no script de travamento mencionado anteriormente, é discutida em detalhes em Scripts de Transação e Linguagem de Script. A estrutura de um output de transação demonstra a estrutura de um output de transação.

Table 2. A estrutura de um output de transação

Tamanho	Campo	Descrição
8 bytes	Quantia	Quantia em Bitcoin designada em satoshis (10^{-8} bitcoin)
1-9 bytes (VarInt)	Locking-Script Size	Comprimento do script de travamento em bytes, para seguir
Variable	Locking-Script	Um script definindo as condições necessárias para se gastar o output

Em Um script que chama o API da blockchain.info para encontrar o UTXO relacionado a um endereço, nós usamos o API da blockchain.info para encontrar os outputs não-gastos (UTXO) de um endereço específico.

Example 1. Um script que chama o API da blockchain.info para encontrar o UTXO relacionado a um endereço

```
# get unspent outputs from blockchain API

import json
import requests

# example address
address = '1Dorian4RoXcnBv9hnQ4Y2C1an6NJ4UrjX'

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

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

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

Ao executar o script, nós enxergamos uma lista de IDs de transação, uma vírcula, um número de índice do output de transação não-gasta (UTXO) específico e o valor daquele UTXO em satoshis. O script de travamento não é demonstrado no output em Executando o script get-utxo.py.

Example 2. Executando o script get-utxo.py

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

Condições para gastar (entravamentos)

Os outputs de transação associam uma quantia específica (em satoshis) a um script específico de travamento ou de *oneração* que define a condição que deve ser preenchida para que aquela quantia possa ser gasta. Na maioria dos casos, o script de travamento irá travar o output para um endereço bitcoin específico, transferindo a posse dessa quantia para o novo dono. Quando a Alice pagou o Bob's Cafe por uma xícara de café, a transação dela criou um output de 0,015 bitcoins *onerados* ou travador ao endereço de bitcoin da cafeteria. Esse output de 0,015 bitcoin foi registrado na blockchain e se tornou parte do conjunto de Output de Transação Não-gasta, ou seja, ele apareceu na carteira do Bob como parte de um saldo disponível. Quando o Bob resolver gastar essa quantia, a transação dele irá liberar essa oneração, destravando o output ao forncer um script de destravamento que contenha uma assinatura da chave privada do Bob.

Inputs de transações

Em termos simples, os inputs de transação apontam para os UTXOs. Eles apontam para um UTXO específica através da referência ao hash da transação e o número da sequência onde o UTXO está registrado na blockchain. Para gastar um UTXO, um input de transação também inclui scripts de destravamento para satisfazer as condições de gasto definidas pelo UTXO. O script de destravamento é geralmente uma assinatura que comprova a posse do endereço bitcoin que está no script de travamento.

Quando os usuários fazem um pagamento, a carteira deles constrói uma transação ao selecionar a partir do UTXO disponível. Por exemplo, para fazer um pagamento de 0,015 bitcoin, o aplicativo de carteira pode selecionar um UTXO de 0,01 e um UTXO de 0,005, usando ambos para somar a quantia de pagamento desejada.

Em Um script para calcular quantos bitcoins totais serão emitidos, nós demonstramos o uso do algoritmo "greedy" para selecionar a partir dos UTXOs disponíveis, a fim de fazer um pagamento de uma quantia específica. No exemplo, os UTXOs disponíveis são fornecidos como uma array constante, mas, na realidade, os UTXOs disponíveis seriam adquiridos com uma chamada RPC ao Bitcoin Core, ou a um API de terceiro como demonstrado em Um script que chama o API da blockchain.info para encontrar o UTXO relacionado a um endereço.

Example 3. Um script para calcular quantos bitcoins totais serão emitidos

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

from sys import argv

class OutputInfo:

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

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

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

```
def main():
    unspent = [

OutputInfo("ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167", 1,
8000000),

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

OutputInfo("b2affea89ff82557c60d635a2a3137b8f88f12ecec85082f7d0a1f82ee203ac4", 0,
10000000),

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

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

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

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

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

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

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

Se nós executarmos os script *select-utxo.py* sem um parâmetro, ele irá tentar construir um conjunto de UTXOs (e troco) para um pagamento de 55.000.000 satoshis (0,55 bitcoin). Se você fornecer uma quantia alvo de pagamento como um parâmetro, o script irá selecionar os UTXOs para fazer aquela quantia alvo de pagamento. Em Executando o script select-utxo.py, nós executamos o escript tentando fazer um pagamento de 0,5 bitcoin ou 50.000.000 satoshis.

Example 4. Executando o script select-utxo.py

```
$ python select-utxo.py 50000000
Para uma quantia de transação de 50000000 Satoshis (0.500000 bitcoin), utilize:
([<7dbc497969c7475e45d952c4a872e213fb15d45e5cd3473c386a71a1b0c136a1:0 with 25000000
Satoshis>, <7f42eda67921ee92eae5f79bd37c68c9cb859b899ce70dba68c48338857b7818:0 with
16100000 Satoshis>,
<6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf:0 with 16050000
Satoshis>], 'Change: 7150000 Satoshis')
```

Assim que o UTXO é selecionado, a carteira passa a produzir scripts de destravamento contendo as assinaturas para cada um dos UTXOs, dessa maneira tornando-os gastáveis, ao satisfazer as condições de seus scripts de travamento. A carteira acrescenta essas referências dos UTXOs e os scripts de destravamento como inputs da transação. A estrutura de um input de transação demonstra a estrutura de um input de transação.

Table 3. A estrutura de um input de transação

Tamanho	Campo	Descrição
32 bytes	Transaction Hash	Apontador para a transação contendo o UTXO para ser gasto
4 bytes	Output Index	O número índice do UTXO para ser gasto; o primeiro é 0
1-9 bytes (VarInt)	Unlocking-Script Size	Comprimento do script de destravamento em bytes, para seguir
Variável	Unlocking-Script	Um script que preenche os critérios (condições) para o script de travamento do UTXO
4 bytes	Sequence Number	Funcionalidade de substituição de transação, atualmente desabilitada, definida como 0xFFFFFFFF

NOTE

O número de sequência é usado para substituir uma transação antes da expiração do locktime da transação, que é uma funcionalidade atualmente desabilitada no bitcoin. A maioria das transações definem o valor do número de sequência para o valor inteiro máximo (0xFFFFFFFF), que é ignorado pela rede bitcoin. Se a transação possuir um locktime diferente de zero, para que o locktime seja habilitado, pelo menos um de seus inputs deve ter um número de sequência menor que 0xFFFFFFFF .

Taxas de Transação

A maioria das transações incluem taxas de transação, que remuneram os mineradores bitcoin por manterem a segurança da rede. A mineração, as taxas e as recompensas coletadas pelos mineradores são discutidas em maiores detalhes em [ch8]. Essa seção examina como as taxas de transação são incluídas em uma transação típica. A maioria das carteiras calcula e inclui automaticamente as taxas de transação. Entretanto, se você estiver construindo transações através de uma programação, ou usando uma interface de linha de comando, você deve contabilizar isso manualmente e incluir essas taxas.

As taxas de transação servem como um incentivo para incluir (minerar) a transação no próximo bloco e também como um desestimulador contra transações de "spam" ou qualquer tipo de abuso contra o sistema, ao impor um pequeno custo em cada transação. As taxas de transação são coletadas pelo minerador que minerar o bloco que registra a transação na blockchain.

As taxas de transação são calculadas de acordo com o tamanho da transação em kilobytes, e não no valor da transação em bitcoins. De modo geral, as taxas de transação são definidas baseadas na forças do mercado na rede bitcoin. Os mineradores priorizam transações baseados em muitos critérios diferentes, incluindo taxas, e podem até mesmo processar transações de graça sob certas circunstâncias. As taxas de transações afetam a prioridade do processamento, significando que uma transação com taxas suficientes é mais provável de ser incluída no próximo bloco mais minerado, enquanto uma transação sem taxa ou com taxa insuficiente pode ser atrasada, processada em uma base de melhor-esforço após alguns blocos, ou nem chegar a ser processada. As taxas de transação não são obrigatórias, e as transações sem taxas podem chegar a ser processadas; no entanto, a inclusão de taxas de transação incentiva uma maior prioridade de processamento.

Ao longo do tempo, houve evoluções na maneira como as taxas são calculadas e no efeito que elas tem na prioridade das transações. Inicialmente, as taxas de transação eram fixas e constantes ao longo da rede. Gradualmente, a estrutura de taxas foi flexibilizada, de maneira que ela pudesse ser influenciada por forças do mercado, baseando-se na capacidade da rede e no volume de transações. Atualmente, a taxa de transação mínima é fixada em 0,0001 bitcoin ou um décido de um milibitcoin por kilobyte, recentemente reduzida de um mili-bitcoin. A maioria das transações tem menos de um kilobyte; no entanto, aquelas que contém múltiplos inputs ou outputs podem ser maiores. Em revisões futuras do protocolo bitcoin, espera-se que as aplicações em carteira irão utilizar análise estatística para calcular a taxa mais apropriada para ser incluída em uma transação, de acordo com as taxas médias da transações recentes.

O algoritmo atualmente utilizado pelos mineradores para priorizar a inclusão no bloco das transações de acordo com suas taxas é examinado em detalhes no [ch8].

Adicionando Taxas às Transações

A estrutura de dados das transações não tem um campo para taxas. Ao invés disso, infere-se que as taxas são a diferença entre a soma dos inputs e a soma dos outputs. Qualquer valor em excesso que permanece após todos os outputs terem sido deduzidos de todos os inputs é a taxa que é coletada pelos mineradores.

As taxas de transação são considerada como a sobra dos inputs subtraídos dos outputs:

```
Taxas = Soma(Inputs) - Soma(Outputs)
```

Isso é um elemento relativamente confuso das transações, e é um ponto importante para se compreender, porque se você está construindo as suas próprias transações você deve se certificar que você não incluiu uma taxa muito grande ao subutilizar os inputs. Isso significa que você deve contabilizar todos os inputs, se necessário para criar o troco, caso contrário você irá acabar dando uma gorjeta grande demais para os mineradores!

Por exemplo, se você consumir um UTXO de 20 bitcoins para fazer um pagamento de 1 bitcoin, você precisa incluir um output de troco de 20 bitcoins de volta para a sua carteira. Caso contrário, os 19 bitcoins que sobrarem serão contados como uma taxa de transação e serão coletados pelo minerador que minerar a sua transação em um bloco. A sua transação vai receber alta prioridade e o minerador vai ficar muito feliz, mas provavelmente não era isso que você queria fazer.

WARNING	Se você se esquecer de adicionar um output de troco em uma transação construída manualmente, você pagará a taxa de transação com o valor do troco. Pode ser que a sua real intenção não seria dizer "pode ficar com troco!"

Vamos ver como isso funciona na prática, ao ver novamente a compra do café da Alice. A Alice quer gastar 0,015 bitcoin para pagar pelo café. Para garantir que essa transação será processada prontamente, ela terá que incluir uma taxa de transação, que digamos que seja de 0,001. Isso significa que o custo total da transação será de 0,016. A carteira dela portanto deve usar um conjunto de UTXOs cuja soma dê 0,016 bitcoin ou mais e, se necessário, terá que criar o troco. Digamos que a carteira dela tem uma UTXO de 0,2 bitcoin disponível. Portanto ela precisará consumir esse UTXO, criar um output para o Bob’s Cafe de 0,015 e um segundo output com 0,184 bitcoin de troco de volta para a própria carteira dela, deixando 0,001 bitcoin não-alocado, que é uma taxa implícita pela transação.

Agora vamos ver um cenário diferente. Eugênia, nossa diretora do centro de caridade para crianças nas Filipinas, completou uma campanha para arrecadação de fundos para comprar livros escolares para as crianças. Ela recebeu milhares de pequenas doações de pessoas de todos os lugares do mundo, totalizando 50 bitcoins, então sua carteira está cheia de pagamentos muito pequenos (UTXOs). Agora ela que comprar centenas de livros escolares da editor local, pagando em bitcoin.

Ao tentar construir uma única transação de pagamento com uma quantia maior, o aplicativo de carteira da Eugenia deve utilizar o conjunto de UTXOs disponíveis, que é composto por várias quantias menores. Isso significa que a transação resultante irá utilizarmais de uma centena de UTXOs de pequeno valor como input, e apenas um único output, que será o pagamento à editora do livro. Uma transação com tantos inputs terá mais do que um kilobyte, talvez 2 a 3 kilobytes. Como resultado, ela irá exigir uma taxa maior do que a taxa mínima da de rede, que é 0,0001 bitcoin.

O aplicativo de carteira da Eugenia irá calcular a taxa apropriada ao medir o tamanho da transação e multiplicando-o pela taxa por kilobyte. Muitas carteiras irão pagar taxas extras para transações maiores para assegurar que a transação seja processada prontamente. A taxa não é maior porque a

Eugenia está gastando mais dinheiro, mas porque a transação dela é mais complexa e tem um tamanho maior—a taxa é independente do valor em bitcoins da transação.

Encadeamento de Transações e Transações Órfãs

Como nós já vimos, as transações formam uma cadeia, onde uma transação gasta os outputs da transação anterior (conhecida como transação pai) e cria outputs para uma transação subsequente (conhecida como transação filha). Às vezes uma cadeia inteira de transações que são dependentes umas das outras—por exemplo, uma pai, uma filha e uma neta—são criadas ao mesmo tempo, para preencher um fluxo complexo de transações que exige que as filhas sejam assinadas antes da transação pai ser assinada. Por exemplo, essa é a técnica usada em transações CoinJoin, onde várias pessoas agrupam transações para protegerem a sua privacidade.

Quando várias transações são transmitidas na rede, elas nem sempre chegam na mesma ordem. Às vezes, a transação filha pode chegar antes da pai. Nesse caso, os nodos que enxergarem a filha primeiro irão ver que ela tem uma referência a uma transação pai que ainda não é conhecida. Ao invés de rejeitar a filha, eles a colocam em uma pool temporária para aguardar a chegada de sua transação pai e propagá-la para todos os outros nodos. A pool de transações sem transações-pais é conhecida como a *pool de transações órfãs*. Assim que uma transação pai chega, quaisquer órfãs que tenham uma referência ao UTXO criado pela pai são liberadas, revalidadas recursivamente, e então toda a corrente de transações pode ser incluída em um pool de transações, pronto para ser minerado em um bloco. As correntes de transações podem ser arbitrariamente longas, com qualquer número de gerações transmitidas simultaneamente. O mecanismo que mantém as órfãs em um pool de órfãs garante que transações válidas não sejam rejeitadas apenas porque a transação pai delas se atrasou e que no final a corrente que elas pertecem será reconstruída na ordem correta, independente da ordem de chegada.

Existe um limite para o número de transações órfãs armazenadas na memória, para prevenir um ataque de negação de serviço (ataque DoS) contra os nodos do bitcoin. O limite é definido como MAX_ORPHAN_TRANSACTIONS no código-fonte do cliente referência do bitcoin. Se o número de transações órfãs na pool exceder MAX_ORPHAN_TRANSACTIONS, uma ou mais transações órfãs selecionadas aleatoriamente serão removidas da pool, até que o tamanho da pool volte a estar dentro dos limites.

Scripts de Transação e Linguagem de Script

Os clientes bitcoin validam as transações ao executar um script, escrito em uma linguagem de script parecida com Forth. Tanto o script de travamento (de oneração) colocado em um UTXO quanto o script de destravamento que geralmente contém uma assinatura são escritos nessa linguagem de script. Quando uma transação é validada, o script de destravamento em cada in put é executado junto com o script de travamento correspondente para ver se ele satisfaz a condição para se gastar.

Atualmente, a maioria das transações processadas pela rede bitcoin possuem a forma "Alice paga Bob" e são baseadas no mesmo script chamado script de Pagamento-para-Hash-de-Chave-Pública ("Pay-to-Public-Key-Hash script"). Entretanto, o uso de scripts para travar outputs e destravar inputs significa que através do uso de linguagem de programação, as transações podem conter um número infinito de

condições. As transações bitcoin não são limitadas à forma e ao padrão "Alice paga Bob".

Essa é apenas a ponta do iceberg de possibilidade que podem ser expressadas com essa linguagem de script. Nessa seção, nós iremos demontrar os componentes da linguagem de scripting de transações bitcoin e mostraremos como isso pode ser usado para expressar condições complexar para se gastar e como essas condições podem ser satisfeitas através de scripts de destravamento.

TIP	A validação da transação bitcoin não é baseada em um padrão estático, ao invés disso, ela é obtida através da execução de uma linguagem de script. Essa linguagem permite que uma variedade quase infinita de condições seja expressa. Essa é a maneira através da qual o bitcoin recebe o poder do "dinheiro programável".

Construção do Script (Travamento + Destravamento)

O mecanismo de validação da transação do Bitcoin depende de dois tipos de scripts para validar as transações: um script de travamento e um de destravamento.

Um script de travamento é uma oneração colocada em um output, e especifica as condições que devem ser preenchidas para se gastar esse output no futuro. Historicamente, o script de travamento era chamado de *scriptPubKey*, pois ele geralmente continha uma chave pública ou um endereço bitcoin. Nesse livro nós nos referimos a ele como um "script de travamento", para reconhecer a gama muito maior de possibilidades dessa tecnologia de scripting. Na maioria das aplicações bitcoin, aquilo que nós nos referimos como um script de travamento aparecerá no código-fonte como scriptPubKey.

Um script de destravamento é um script que "resolve", ou satisfaz, as condições que foram colocadas em um ouput por um script de travamento, permitindo que o output seja gaso. Os scripts de destravamento fazem parte de todos os inputs de transação, e na maioria das vezes eles contém uma assinatura digital produzida pela carteira do usuário a partir de sua chave privada. Historicamente, o script de destravamento é chamado de *scriptSig*, porque ele geralmente contém um assinatura digital. Na maioria das aplicações bitcoins, o código-fonte se refere ao script de destravamento como scriptSig. Nesse livro, nós nos referimos a ele como um "script de destravamento" para reconhecer a gama muito maior de exigências dos scripts de travamento, pois nem todos os scripts de destravamento precisam conter assinaturas.

Todos os clientes bitcoin irão validar as transações ao executar os scripts de travamento e destravamento juntos. Para cada input na transação, o software de validação irá primeiro adquirir as UTXOs referenciadas pelo input. Esses UTXOs contém um script de travamento definindo as condições necessárias para gastá-los. O software de validação irá então adquirir o script de travamento contido no input que está tentanto gastar essas UTXOs e irá executar os dois scripts.

No cliente bitcoin original, os scripts de destravamento e travamento são concatenados e executados em sequência. Por questões de segurança, isso foi modificado em 2010, devido a uma vulnerabilidade que permitia que um script de destravamento mal formado adicionasse dados no stack e corrompesse o script de travamento. Na implementação atual, os scripts são executados separadamente com o stack sendo transferido entre as duas execuções, como descrito a seguir.

Primeiro, o script de destravamento é executando, usando o mecanismo de execução do stack. Se o script de destravamento foi executado sem erros (por exemplo, ele não possui operadores "pendentes" faltando), o stack principal (e não o stack alternativo) é copiado e o script de travamento é executado. Se o resultado da execução do script de travamento com os dados do stack copiados a partir do script de destravamento for "TRU", o script de destravamento teve sucesso em resolver as condições impostas pelo script de travamento e, portanto, o input é uma autorização válida para gastar o UTXO. Se qualquer resultado diferente de "TRUE" permanecer após a execução do script combinado, o pinput é inválido porque ele não conseguiu satisfazer as condições de gasto impostas no UTXO. Note que o UTXO está permanentemente registrado nablockchain, e portanto é invariável e não é afetado por múltiplas tentativas de se gastá-lo por referência em uma nova transação. Somente uma transação válida que satisfaz corretamente as condições do UTXO resulta em um UTXO sendo marcado como "gasto" e removido do conjunto de UTXOs disponíveis (não-gastos).

Combinando scriptSig e scriptPubKey para avaliar um script de transação é um exemplo de scripts de destravamento e travamento para o tipo mais comum de transação bitcoin (um pagamento para um hash de endereço público), demonstrando o script cominando resultando da concatenação dos scripts de destravamento e travamento antes da validação do script.

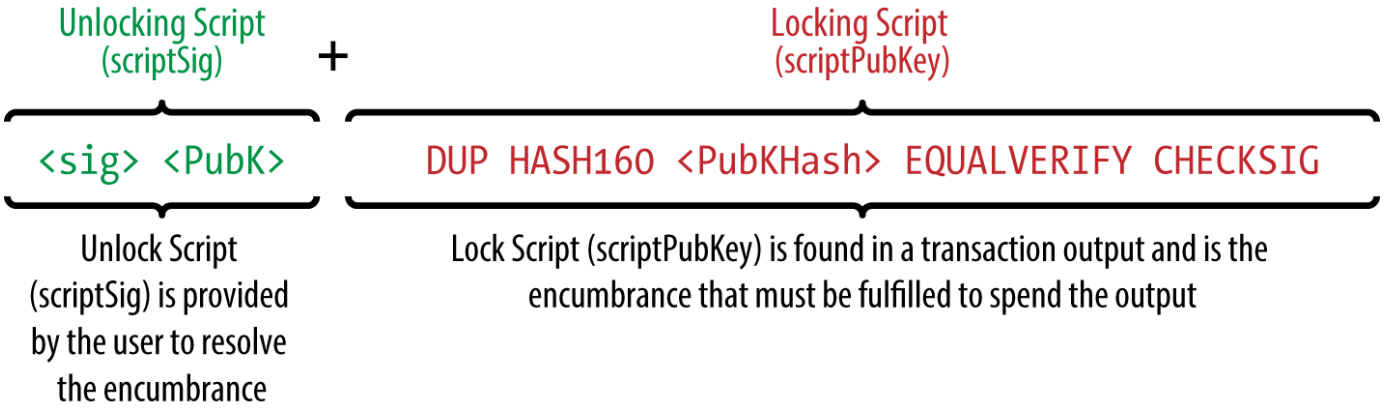


Figure 1. Combinando scriptSig e scriptPubKey para avaliar um script de transação

Linguagem de Script

A linguagem de script das transações bitcoin, chamada de *Script*, é uma linguagem de execução de notação polonesa invertida semelhante a Forth baseada em stack. Se isso soa estranho, você provavelmente não estudou as linguagens de programação dos anos 1960. O script é uma linguagem muito simples que foi projetada para ser limitada em escopo e executável em vários hdarwares, talvez tão simples quando um dispositivo embutido, como uma calculadora de bolso. Ele requer processamento mínimo e não pode fazer coisas que muitas linguagens de programação modernas conseguem. No caso do dinheiro programável, isso é uma funcionalidade de segurança deliberada.

A linguagem de script do bitcoin é chamada de linguagem baseada em stack porque ela usa uma estrutura de dados chamada de *stack* (pilha). Um stack é uma estrutura de dados muito simples, que pode ser imaginada como uma pilha de cartas de um baralho. Um stack permite duas operações: empilhar ("push") e desempilhar ("pop"). "Empilhar" ("push") adiciona um item no topo da pilha. Desempilhar ("pop") retira o item que está no topo da pilha.

A linguagem de scripting executa o script ao processar cada idem da esquerda para a direita. Números (constantes de dados) são empurrados para a stack. Os operadores empilham (push) ou desempilham (pop) um ou mais parâmetros do stack, atuam neles e podem empurrar um resultado para pilha. Por exemplo, OP_ADD irá desempilhar dois itens do stack, adicioná-los e empurrar a soma resultante para o stack.

Os operadores condicionais avaliam uma condição, produzindo um resultado booleano de TRUE (VERDADEIRO) ou FALSE (FALSO). Por exemplo, OP_EQUAL desempilha (pop) dois itens do stack e empurra TRUE (TRUE representado pelo número 1) se eles forem iguais ou FALSE (representado por zero) se eles não forem iguais. Scripts de transação de bitcoin geralmente contém um operador condicional, de maneira que eles possam produzir o resultado TRUE que significa uma transação válida.

Em [simplemath_script], o script 2 3 OP_ADD 5 OP_EQUAL demonstra o operador de adição aritimética OP_ADD, adicionando dois números e colocando o resultado no stack, seguido pelo operador condicional OP_EQUAL, que verifica que a soma resultante é igual a 5. Para simplificar, o prefixo OP_ é omitido no exemplo passo-a-passo.

O script a seguir é um pouco mais complexo, que calcula 2 + 7 – 3 + 1. Note que quando o script contém vários operadores em uma linha, o stack permite que os resultados de um operadores ajam no próximo operador;

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

Tente você mesmo validar o script anterior usando papel e caneta. Quando a execução do script terminar, você deve terminar com o valor TRUE no stack.

Apesar de a maioria dos scripts de travamento se referir a um endereço bitcoin ou a uma chave pública, portanto exigindo a prova de posse para gastar os fundos, o script não precisa ser tão complexo. Qualquer combinação de scripts de travamento e destravamento que resultar em um valor TRUE é válida. Essa aritmética simples que nós usamos como exemplo da linguagem de script é também um script de travamento válido que pode ser usado para travar um output de transação.

Usando parte do script de exemplo aritmético como script de travamento:

```
3 OP_ADD 5 OP_EQUAL
```

que pode ser satisfeita através de uma transação contendo um input com o script de destravamento:

```
2
```

O software de validação combina os scripts de bloqueio e desbloqueio e o script resultante é:

```
2 3 OP_ADD 5 OP_EQUAL
```

Como nós vimos no exemplo passo-a-passo em [simplemath_script], quando o script é executado, o resultado é OP_TRUE, fazendo a transação válida. Não apenas esse é um script de travamento válido, como também o UTXO resultante poderia ser gasto por qualquer pessoa que tivesse as habilidades aritiméticas para saber que o número 2 satisfaz o script.

1. Script de validação do bitcoin fazendo uma conta matemática simples image::images/msbt_0502.png["TxScriptSimpleMathExample"]

TIP	As transações são válidas se o resultado no topo do stack é TRUE (definido como {0x01}), qualquer valor diferente de zero ou se o stack estiver vazio após a execução do script. As transações são inválidas se o valor do topo do stack for FALSE (um valor vazio de comprimento zero, definido como {}) ou se a execução do script for suspensa por um operador, como o OP_VERIFY, o OP_RETURN ou um terminador condicional como o OP_ENDIF. Veja [tx_script_ops] para detalhes.

A Incompletude de Turing

A linguagem de script da transação bitcoin contém muitos operadores, mas é deliberadamente limitada de uma maneira importante—não existem loops ou capacidades de controle de fluxo complexa além do controle de fluxo condicional. Isso garante que a linguagem não é *Turing completa*, o que significa que os scripts tem uma complexidade limitada e tempos de execução previsíveis. O script não é uma linguagem de finalidades gerais. Essas limitações garantem que a linguagem não pode ser usada para criar um loop infinito ou outra forma de "bomba lógica" que poderia ser embutida em uma transação de uma maneira que cause um ataque de negação de serviço contra a rede bitcoin. Lembre-se, cada transação é validada por todos os nodos completos da rede bitcoin. Uma linguagem limitada previne que o mecanismo de validação da transação seja usado como uma vulnerabilidade.

Verificação sem Monitoração de Estado

A linguagem de script de transação do bitcoin não exige a monitoração de estado, ou seja, não existe um estado anterior à execução do script, ou estado salvo após a execução do script. Portanto, toda a informação que é necessária para se executar um script está contida no interior do script. Um script será executado de maneira previsível em qualquer outro sistema. Se o seu sistema verifica um script, você pode ter certeza que todos os outros sistemas na rede bitcoin também verificarão esse script, ou seja, uma transação válida será válida para todos na rede, e todos saberão disso. Essa previsibilidade de resultados é um benefício essencial do sistema bitcoin.

Transações padrão

Nos primeiros anos do desenvolvimento do bitcoin, os desenvolvedores apresentaram algumas limitações nos tipos de scripts que poderia ser processados pelo cliente referência. Essas limitações são

codificadas em uma função chamada isStandard(), que define cinco tipos de transações "padrão" (standard). Essas limitações são temporárias e podem ser que já tenham sido removidas no momento em que você está lendo esse livro. Até então, os cinco tipos padrões de scripts de transação são os únicos que serão aceitos pelo cliente de referência e pela maioria dos mineradores que executam o cliente de referência. Embora seja possível criar uma transação não-padrão contendo um script que não faz parte dos tipos padrões, você deve encontrar um minerador que não siga essas limitações para minerar essa transação em um bloco.

Verifique o código-fonte do cliente Bitcoin Core (a implementação de referência) para ver o que é atualmente permitido como um script de transação válido.

Os cinco tipos padrões de scripts de transação são pagamento-para-hash-de-chave-pública (pay-to-public-key-hash, P2PKH), chave-pública (public-key), múltiplas-assinaturas (multi-signature, limitado a 15 chaves), pagamento-para-hash-de-script (pay-to-script-hash, P2SH) e output de dados (data output, OP_RETURN), que são descritos em maiores detalhes nas seções a seguir.

Pay-to-Public-Key-Hash (P2PKH)

A vasta maioria das transações processadas na rede bitcoin são transações P2PKH. Elas contém um script de travamento que onera o output com um hash de chave pública, mais comumente conhecido como um enderço bitcoin. As transações que pagam um endereço bitcoin contém scripts P2PKH. Um output travado por um script P2PKH pode ser destravado (gasto) ao se apresenar a chave pública e a assinatura digital criada pela chave privada correspondente.

Por exemplo, vamos examinar novamente o pagamento da Alice para o Bob’s Cafe. A Alice fez um pagamento de 0,015 bitcoin para o endereço bitcoin do Bob’s Cafe. Esse output de transação teria um script de travamento da seguinte forma:

```
OP_DUP OP_HASH160 <Hash da Chave Publica do Cafe> OP_EQUAL OP_CHECKSIG
```

O Hash da Chave Pública do Café é equivalente ao endereço bitcoin do café, sem a codificação Base58Check. A maioria das aplicações iria mostrar o *hash da chave pública* em codificação hexadecimal e não no formato conhecido de endereço bitcoin codificado em Base58Check que começa com um "1".

O script de travamento anterior pode ser satisfeito com um script de destavamento desse jeito:

```
<Assinatura do Cafe> <Chave Publica do Cafe>
```

Os dois scripts juntos iriam formar o seguinte script combinado de validação:

```
<Assinatura do Cafe> <Chave Publica do Cafe> OP_DUP OP_HASH160
<Hash da Chave Publica do Cafe> OP_EQUAL OP_CHECKSIG
```

Quando executado, esse script combinado irá avaliar para TRUE se, e somente se, o script de destravamento preencher as condições definidas pelo script de travemento. Em outras palavras, o resultado será TRUE se o script de destravaemento tiver uma assinatura válida da chave privada do café que corresponde ao hash da chave pública definida como uma oneração.

As figuras <xref linkend="P2PubKHash1" xrefstyle="select: labelnumber"/> e <xref linkend="P2PubKHash2" xrefstyle="select: labelnumber"/> demonstram (em duas partes) uma execução passo-a-passo do script combinado, que irá comprovar que essa é uma transação válida.

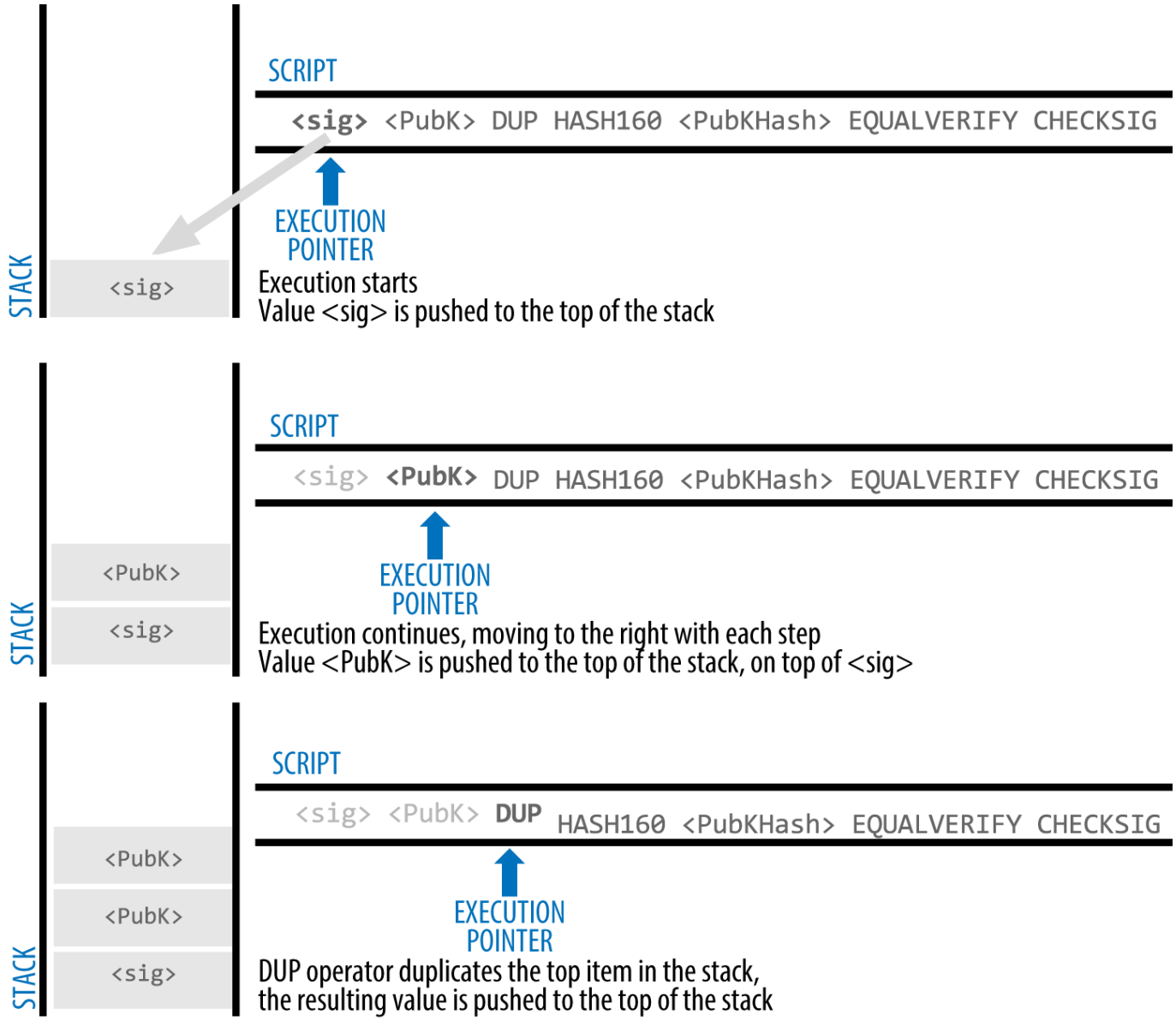


Figure 2. Avaliando um script para uma transação P2PKH (Parte 1 de 2)

Pay-to-Public-Key

Pagamento-para-chave-pública é uma forma mais simples de um pagamento bitcoin do que um pagamento-para-hash-de-chave-pública. Com esse tipo de script, a própria chave pública é armazenada no script de travamento, ao invés de um hash-de-chave-pública como é feito com o P2PKH, que é muito menor. O pagamento-para-hash-de-chave-pública foi inventado pelo Satoshi para tornar os endereços

bitcoins mais curtos, facilitando o uso. Atualmente o pagamento-para-chave-pública é visto com maior frequência em transações coinbase, geradas por softwares de mineração mais antigos que não foram atualizados para utilizarem o P2PKH.

Um script de travamento de pagamento para chave pública fica dessa maneira:

```
<Chave Pública A> OP_CHECKSIG
```

O script de destravamento correspondente que deve ser apresentado para destravar esse tipo de output é uma assinatura simples, como essa:

```
<Assinatura a partir da Chave Privada A>
```

O script combinado, que é validado pelo software de validação de transação, é:

```
<Assinatura a partir da Chave Privada A> <Chave Pública A> OP_CHECKSIG
```

Esse script é uma invocação simples do operador CHECKSIG, que valida a assinatura como pertencendo à chave correta e retorna TRUE para o stack.

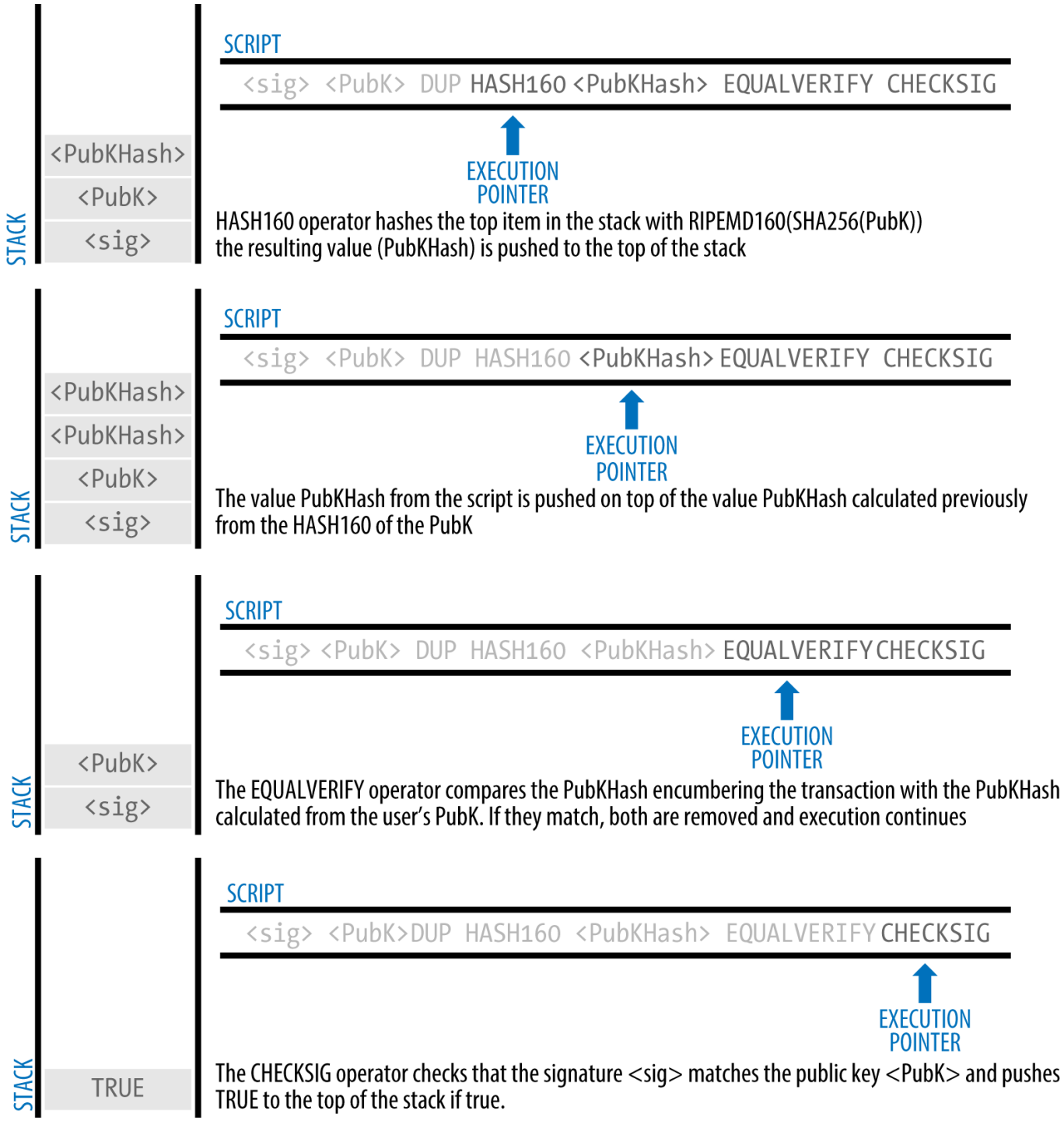


Figure 3. Avaliando um script para uma transação P2PKH (Parte 2 de 2)

Múltiplas assinaturas

Scripts de múltiplas assinaturas definem uma condição onde N chaves públicas são registradas no script e pelo menos M dessas devem fornecer assinaturas para liberar a oneração. Isso também é conhecido como um esquema M-de-N, onde N é o número total de chaves e M é o número de assinaturas necessárias para a validação. Por exemplo, um script de múltiplas assinaturas 2-de-3 é um onde três chaves públicas são listadas como assinadores em potencial e pelo menos dois deles devem ser usados para criar assinaturas para uma transação válida que gaste os fundos. Atualmente, os

scripts de múltiplas assinaturas padrão podem fazer múltiplas assinaturas desde 1-de-1 a 15-de-15, ou qualquer combinação nessa faixa. A limitação de 15 chaves listadas já deve ter sido removida no momento em que esse livro for publicado, então verifique a função isStandard() para ver o que está sendo atualmente aceito pela rede.

A forma geral usada por um script de travamento para definir uma condição de múltiplas assinaturas M-de-N é

```
M <Chave Pública 1> <Chave Pública 2> ... <Chave Pública N> N OP_CHECKMULTISIG
```

onde N é o número total de chaves públicas listadas e M é o limiar de assinaturas necessárias para passar a saída.

Um script de travamento definindo uma condição de 2-de-3 múltiplas assinaturas fica desse jeito:

```
2 <Chave Pública A> <Chave Pública B> <Chave Pública C> 3 OP_CHECKMULTISIG
```

O script de bloqueio anterior pode ser satisfeito com um script de desbloqueio contendo pares de chaves públicas e assinaturas:

```
OP_0 <Assinatura B> <Assinatura C>
```

ou qualquer combinação de duas assinaturas a partir das chaves privadas correspondendo às três chaves públicas listadas.

NOTE O prefixo é exigido OP_0 devido a um bug na implementação original do CHECKMULTISIG, que faz com que um item a mais seja retirado do stack. Isso é ignorado pelo CHECKMULTISIG, que é usado apenas para resolver esse problema.

Os dois scripts juntos formariam o script de validação combinado:

```
OP_0 <Assinatura B> <Assinatura C> 2 <Chave Pública A> <Chave Pública B> <Chave Pública
C> 3 OP_CHECKMULTISIG
```

Quando executado, o script combinado será avaliado como TRUE se, e somente se, o script de desbloqueio coincidir com as condições definidas pelo script de bloqueio. Neste caso, a condição é se o script de desbloqueio tem uma assinatura válida entre as duas chaves privadas que correspondem a duas das três chaves públicas definidos como um estorvo.

Output de dados (OP_RETURN)

"ledger, storing unrelated information in")A blockchain, ou seja, todos os registros do bitcoin, que são

distribuídos e contém data e hora, possui usos potenciais muito além de pagamentos. Muitos desenvolvedores tentaram usar a linguagem de script de transação para tirar vantagem da segurança e da resiliência do sistema para aplicações como serviços de cartórios digitais, certificados de ações e contratos smart. As primeirs tentativas de se usar a linguagem de script do bitcoin para essas finalidades envolveram criar outputs de transação que registravam dados na blockchain; por exemplo, para registrar uma impressão digital de um arquivo de uma maneira que qualquer um pudesse estabelecer uma prova-de-existência daquele arquivo em uma data específica através de uma referência àquela transação.

O uso da blockchain do bitcoin para armazenar dados não-relacionados ao bitcoin é um assunto controverso. Muitos desenvolvedores consideram esse uso abusivo e querem desencorajá-lo. Outros acreditam que isso é uma demonstração das capacidades poderosas da tecnologia bitcoin e querem encorajar esse tipo de experiência. Aqueles que são contra a inclusão de dados não relacionados a pagamento argumentam que isso causa um "inchaço na blockchain", causando um fardo para aqueles rodando nodos completos do bitcoin, que tem que lidar com o custo do armazenamento em disco para os dados que a blockchain não foi projetada para armazenar. Além disso, essas transações criam UTXOs que não podem ser gastos, usando endereços bitcoins destinários como um campo de formulário livre de 20 bytes. Como o endereço é usado para dados, ele não corresponde a uma chave privada e o UTXO resultante *jamais* pode ser gasto; ele é um pagamento falso. Essas transações que jamais podem ser gastas portanto nunca são removidas do conjunto UTXO e fazem com que o tamanho do banco de dados UTXO cresça ou "inche" infinitamente.

Na versão 0.9 do cliente Bitcoin Core, um comprometimento foi atingido com a introdução do operador OP_RETURN. O OP_RETURN permitiu que os desenvolvedores adicionassem 80 bytes de dados de não-pagamento para um output de transação. Entretanto, ao contrário do uso de UTXOs falsos, o operador OP_RETURN cria um output que é *comprovadamente não-gastável* de maneira explícita, que não precisa ser registrado no conjunto UTXO. Os outputs OP_RETURN são registrados na blockchain, então eles consomem espaço em disco e contribuem para o aumento do tamanho da blockchain, mas eles não são armazenados no conjunto UTXO e portanto eles não ficam enchendo a pool de memória UTXO e não são um fardo para os nodos completos, pois não exigem mais memórias RAM caras.

O script OP_RETURN fica assim:

```
OP_RETURN <data>
```

A parte de dados é limitada a 80 bytes e a mais frequentemente representa um hash, como o output do algoritmo SHA256 (32 bytes). Muitas aplicações colocam um prefixo na frente dos dados para ajudar a identificar a aplicação. Por exemplo, o serviço de cartório digital Proof of Existence usa o prefixo de 8 bytes "DOCPROOF", que é codificado em ACII como 44f4350524f4f46 em hexadecimal.

Lembre-se que não existe um "script de destravamento" que corresponde ao OP_RETURN que pudesse ser usado para "gastar" um output OP_RETURN. O objetivo do OP_RETURN é que você não possa gastar o dinheiro travado naquele output, e, portanto, ele não precisa ser armazenado no conjunto UTXO como potencialmente gastável—o OP_RETURN é *comprovadamente não-gastável*. O+OP_RETURN+ é geralmente um output com uma quantia de zero bitcoin, porque qualquer bitcoin que for vinculado a

esse output será efetivamente perdido para sempre. Se um OP_RETURN for encontrado pelo software de validação de script, ele resultará imediatamente na suspensão do script de validação e na marcação da transação como inválida. Portanto, se você acidentalmente referenciar um output OP_RETURN como um input em uma transação, aquela transação será inválida.

Uma transação padrão (que preenche as verificações isStandard()) pode ter apenas um output +OP_RETURN. Entretanto, o output OP_RETURN único pode ser combinado em uma transação com outputs de qualquer outro tipo.

Duas novas opções de linha de comando foram adicionadas no Bitcoin Core na versão 0.10. A opção datacarrier controla a transmissão e mineração de transações OP_RETURN, cujo padrão é definido como "1", para permiti-las. A opção datacarriersize recebe um argumento numérico especificando o tamanho máximo em bytes para os dados do OP_RETURN, cujo padrão é 40 bytes.

NOTE O OP_RETURN foi inicialmente proposto com um limite de 80 bytes, mas o limite foi reduzido para 40 bytes quando a funcionalidade foi lançada. Em fevereiro de 2015, na versão 0.10 do Bitcoin Core, o limite foi novamente elevado para 80 bytes. Os nodos podem decidir transmitir ou minerar OP_RETURN, ou apenas transmitir e minerar OP_RETURN contendo menor que 80 bytes de dados.

Pay-to-Script-Hash (P2SH)

O pagamento-para-hash-de-script (P2SH) foi apresentado em 2012 como um novo tipo poderoso de transação que simplifica e muito o uso de scripts complexos de transação. Para explicar a necessidade do P2SH, vamos dar uma olhada em um exemplo prático.

Em [ch01_intro_what_is_bitcoin], nós apresentamos o Mohammed, um importador de eletrônicos que mora em Dubai. A companhia do Mohammed usa a funcionalidade de múltiplas assinaturas do bitcoin para suas contas corporativas. Os scripts de múltiplas assinaturas são um dos usos mais comuns das capacidades avançadas de scripting do bitcoin e são uma funcionalidade muito poderosa. A companhia do Mohammed usa um script de múltiplas-assinaturas para todos os pagamentos de clientes, conhecidos em termos de contabilidade como "contas recebíveis", ou CR. Com o esquema de múltiplas assinaturas, quaisquer pagamentos feitos pelos consumidores são travados de uma maneira que eles exigem pelo menos duas assinaturas para serem liberados, do Mohammed para um de seus sócios ou do seu advogado que possui uma chave de backup. Um esquema de múltiplas assinaturas desse tipo oferece controle de governança corporativa e protege contra roubo, desvio de fundos ou perdas.

O script resultante é bastante longo e se parece com isso:

```
2 <Chave Pública do Mohammed> <Chave Pública do Parceiro1> <Chave Pública do Parceiro2>
<Chave Pública do Parceiro3> <Chave Pública do Advogado> 5 OP_CHECKMULTISIG
```

Apesar de os scripts de múltiplas assinaturas serem uma funcionalidade poderosa, eles podem ser difíceis de se usar. Em relação ao script anterior, o Mohammed teria que comunicar esse script para cada um de seus consumidores antes do pagamento. Cada consumidor teria que usar um software de

carteira bitcoin especial com a habilidade de criar scripts de transação customizados, e cada consumidor teria que entender como criar uma transação usando scripts customizados. Além disso, a transação resultante seria cerca de cinco vezes maior do que uma transação de pagamento simples, pois esse script contém chaves públicas muito longas. O fardo dessa transação extra-grande teria que ser carregado pelo consumidor na forma de taxas. Finalmente, um script grande de transação como esse seria carregado no conjunto UTXO na RAM de cada nodo completo, até que ele fosse gasto. Todos esses problemas fazem com que o uso de scripts de outputs complexos sejam difíceis na prática.

O pagamento-para-hash-de-script (P2SH) foi desenvolvido para resolver essas dificuldades práticas e para fazer a utilização de scripts complexos de uma maneira tão fácil quanto um pagamento para um endereço bitcoin. Com pagamentos P2SH, o script de travamento complexo é substituído pela sua impressão digital, que é um hash criptográfico. Quando uma transação tentanto gastar o UTXO é apresentada mais tarde, ela deve conter o script que corresponde ao hash, além do script de destravamento. Em termos simples, P2SH significa "pagar para um script que corresponde a esse hash, um script que será apresentado mais tarde quando esse output for gasto".

Nas transações P2SH, o script de travamento que é substituído por um hash é chamado de *script de resgate* ("*redeem script*"), pois ele é apresentado ao sistema na hora do resgate ao invés de ser apresentado como um script de travamento. Script complexo sem P2SH demonstra o script sem P2SH e Script complexo como P2SH demonstra o mesmo script codificado com P2SH.

Table 4. Script complexo sem P2SH

Script de Travamento	2 ChavePub1 ChavePub2 ChavePub3 ChavePub4 ChavePub5 5 OP_CHECKMULTISIG
Script de Destravamento	Ass1 Ass2

Table 5. Script complexo como P2SH

Script de Resgate	2 ChavePub1 ChavePub2 ChavePub3 ChavePub4 ChavePub5 5 OP_CHECKMULTISIG
Script de Travamento	OP_HASH160 <hash de 20 bytes do script de resgate> OP_EQUAL
Script de Destravamento	Ass1 Ass2 script de resgate

Como você pode ver nas tabelas, com P2SH, o complexo script que detalha as condições para gastar o output (o script de resgate) não é apresentado ao script de travamento. Ao invés disso, somente um hash disso está no script de travamento e o próprio scrit de resgate é apresentado depois, como parte do script de destravamento quando o output é gasto. Isso passa o fardo das taxas e da complexidade do pagador para o recipiente (que gasta) da transação.

Vamos dar uma olhada na companhia do Mohammed, um script complexo de múltiplas assinaturas e os scripts P2SH resultantes.

Primeiro, o script de múltiplas assinaturas que a companhia de Mohammed usa para todos os pagamentos que recebe de seus clientes:

```
2 <Chave Pública do Mohammed> <Chave Pública do Parceiro1> <Chave Pública do Parceiro2>
<Chave Pública do Parceiro3> <Chave Pública do Advogado> 5 OP_CHECKMULTISIG
```

Se os marcadores forem substituídos por chaves públicas de verdade (demonstradas aqui como números de 520 bits começando com 04), você verá que esse script se torna muito longo:

```
2
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7EEC6984
D83F1F50C900A24DD47F569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308
EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D78D0E34224858008E8B49047E632
48B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA7D4420D7E1A9162F0279CFC1
0F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD7149B255382ED7F78E946580657EE6FDA162A187543A9
D85BAAA93A4AB3A8F044DADA618D087227440645ABE8A35DA8C5B73997AD343BE5C2AFD94A5043752580AFA1E
CED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF518C022DD618DA774D207D1
37AAB59E0B000EB7ED238F4D800 5 OP_CHECKMULTISIG
```

Todo esse script pode, no entanto, ser representado por um hash criptográfico de 20 bytes, ao aplicar-se primeito o algoritmo de hashing SHA256 e então aplicando-se o algoritmo RIPEMD160 no resultado. O hash de 20 bytes do script anterior é:

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

Uma transação P2SH trava o output nesse hash, e não no script maior, através do script de travamento:

```
OP_HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e OP_EQUAL
```

o qual, como você pode ver, é muito menor. Ao invés de "pagar para esse script de múltiplas assinaturas de 5 chaves", a transação equivalente P2SH é "pagar para um script com esse hash". Um cliente fazendo um pagamento à companhia do Mohammed precisa apenas incluir no seu pagamento esse script de travamento muito menor. Quando o Mohammed quiser gastar esse UTXO, eles precisam apresentar o script de resgate original (aqueles cujo hash travou o UTXO) e as assinaturas necessárias para destravá-lo, dessa maneira:

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG>
```

Os dois scripts são combinados em dois estágios. Primeiro, o script de resgate é verificado em relação ao script de travamento, para garantir que os hashes correspondem:

```
<2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG> OP_HASH160 <redeem scriptHash> OP_EQUAL
```

Se o hash do script de resgate corresponder, o script de destravemento é executado por conta própria, para destavar o script de resgate:

```
<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 OP_CHECKMULTISIG
```

Endereços Pay-to-script-hash

Outra parte importante da funcionalidade P2SH é a habilidade de codificar um hash de script como um endereço, que foi definido na BIP0013. Os endereços P2SH são codificações Base58Check do hash de 20 bytes de um script, da mesma maneira que endereços bitcoin são codificações Base58Check do hash de 20 bytes de uma chave pública. Os endereços P2SH usam o prefixo de versão "5", que resulta em endereços codificados em Base58Check que iniciam com um "3". Por exemplo, o script complexo do Mohammed, "hashado" e codificado em Base58Check, como um endereço P2SH se torna 39RF6JqABiHdYHkfChV6USGMe6Nsr66Gzw. Agora, o Mohammed pode fornecer o seu "endereço" para seus consumidores e eles podem usá-lo em quase qualquer carteira bitcoin para fazer um pagamento simples, como se ele fosse um endereço bitcoin. O prefixo 3 lhes dá a dica de que esse é um tipo especial de endereço, um que corresponde a um script, ao invés de uma chave pública, mas fora isso ele funciona exatamente da mesma maneira que um pagamento para um endereço bitcoin.

Os endereços P2SH escondem toda essa complexidade, de maneira que a pessoa que está fazendo o pagamento não enxerga o script.

Benefícios do pay-to-script-hash

A funcionalidade pagar-para-hash-de-script oferece os seguintes benefícios em comparação ao uso direto de scripts complexos para travar outputs:

- Scripts complexos são substituídos por impressões digitais menores no output da transação, tornando a transação menor.
- Os scripts podem ser codificados como um endereço, de maneira que o pagador e a carteira do pagador não precisem fazer modificações complexas para implementar o P2SH.
- O P2SH faz com que o fardo de se construir o script seja do recipiente, e não do pagador
- O P2SH passa o fardo do armazenamento de dados para o script longo do output (que está no conjunto UTXO) para o input (armazenado na blockchain)
- O P2SH faz com que o fardo do armazenamento de dados para o script longo deixe de estar no tempo presente (pagamento) e passe a estar no tempo futuro (quando ele é gasto).
- O P2SH faz com que o custo da taxa de transação de um script longo deixe de ser do pagador e passe a ser do recipiente, que tem que incluir o longo script de resgate para poder gastá-lo.

Script de Resgate e validação isStandard

Antes da versão 0.9.2 do cliente Bitcoin Core, o pagamento-para-hash-de-script era limitado a tipos padrões de scripts de transação bitcoin, pela função isStandard(). Isso significa que o script de resgate

apresentado na transação de gasto poderia ser de apenas um dos tipos padrões: P2PK, P2PKH ou de múltiplas assinaturas, excluindo o OP_RETURN e o próprio P2SH.

A partir da versão 0.9.2 do cliente Bitcoin Core, as transações P2SH podem conter qualquer script válido, tornando o padrão P2SH muito mais flexível e permitindo a experimentação de muitos tipos novos e complexos de transações.

Note que você não consegue colocar um P2SH dentro de um script de resgate P2SH, porque a especificação P2SH não é recursiva. Você também não consegue usar o OP_RETURN em um script de resgate porque, por definição, o OP_RETURN não pode ser resgatado.

Como o script de regaste não é apresentado para a rede até que você tente gastar o output P2SH, se você travar um output com o hash de uma transação inválida, ela será processada independente disso. Entretanto, você não será capaz de gastá-lo, pois a transação de gasto, que inclui o script de resgate, não será aceita, pois o script é inválido. Isso cria um risco, porque você pode travar um bitcoin em um P2SH que não poderá ser gasto mais tarde. A rede irá aceitar a oneração P2SH mesmo que ela corresponda a um script de resgate inválido, pois o hash do script não dá nenhuma indicação do script que ele representa.

WARNING	Os scripts de travamento P2SH contém o hash de um script de resgate, que não dá dicas do conteúdo do script de resgate. A transação P2SH será considerada válida e será aceita mesmo se o script de resgate for inválido. Dessa maneira, você pode travar bitcoins acidentalmente, que não poderão ser gastos mais tarde.

A Rede Bitcoin

A Arquitetura de Rede Ponto-a-Ponto

O Bitcoin é estruturado como uma arquitetura de rede ponto-a-ponto em cima da Internet. O termo ponto-a-ponto, ou P2P (do inglês peer-to-peer), significa que os computadores que participam da rede são pontos uns para os outros, que eles são todos iguais, que não há nodos "especiais" e que todos os nodos compartilham o trabalho de fornecer serviços na rede. Os nodos da rede se interconectam em uma rede mesh com uma topologia "plana". Não há nenhum servidor, nenhum serviço centralizado ou hierarquia na rede. Os nodos em uma rede ponto-a-ponto tanto fornecem quanto consomem serviços ao mesmo tempo com a reciprocidade atuando como o incentivo para a participação. Redes ponto-a-ponto são inerentemente resilientes, descentralizadas e abertas. O exemplo proeminente de uma arquitetura de rede P2P foi a Internet em seu início, onde os nodos na rede IP eram iguais. A arquitetura de Internet hoje é mais hierárquica, mas o Protocolo da Internet ainda mantém sua essência de topologia plana. Além do bitcoin, a aplicação mais difundida e de maior sucesso das tecnologias P2P é o compartilhamento de arquivos, com o Napster sendo o pioneiro e o BitTorrent como a evolução mais recente da arquitetura.

A arquitetura de rede P2P do Bitcoin é muito mais do que uma escolha de topologia. O bitcoin é projetado como um sistema de dinheiro digital ponto-a-ponto, e a arquitetura da rede é tanto um reflexo e uma base fundamental dessa característica chave. Descentralização do controle é um princípio chave do projeto e ela só pode ser obtida e mantida através de uma rede de consenso P2P descentralizada.

O termo "rede bitcoin" refere-se à coleção de nodos executando o protocolo ponto-a-ponto (P2P) bitcoin. Além do protocolo P2P bitcoin, existem outros protocolos como o Stratum, que são usados para mineração e carteiras leves ou móveis. Esses protocolos adicionais são fornecidos por servidores de roteamento de gateway que acessam a rede bitcoin usando o protocolo P2P bitcoin, e então estendem essa rede aos nodos executando outros protocolos. Por exemplo, os servidores Stratum se conectam os nodos mineradores Stratum através do protocolo Stratum à rede bitcoin principal e fazem um bridge do protocolo Statum com o protocolo P2P bitcoin. Nós usamos o termo "rede bitcoin estendida" para referirmos à rede geral que inclui o protocolo P2P bitcoin, protocolos de mineração-pool, o protocolo Stratum e qualquer outros protocolos relacionados que conectem os componentes do sistema bitcoin.

Tipos e Papéis dos Nodos

Embora os nodos na rede P2P do bitcoin são iguais, eles podem assumir diferentes papéis dependendo da funcionalidade que eles estejam suportando. Um nodo bitocin é uma coleção de funções: roteamento, o banco de dados da blockchain, mineração e serviços de carteira. Um nodo completo com todas essas quatro funções é demonstrado em Um nodo da rede Bitcoin com todas as quatro funções: carteira, minerador, banco de dados completo da blockchain e roteador da rede.

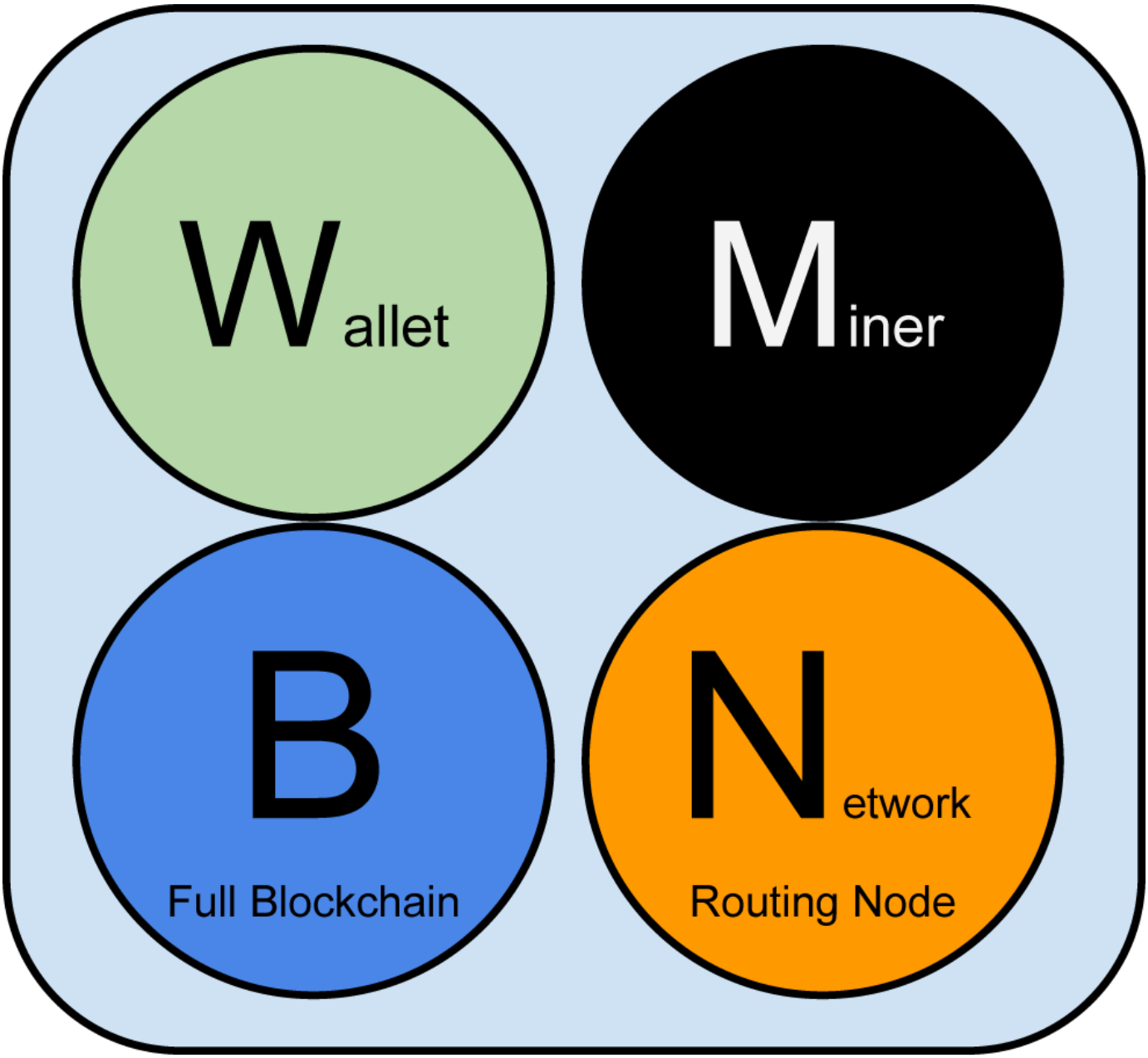


Figure 1. Um nodo da rede Bitcoin com todas as quatro funções: carteira, minerador, banco de dados completo da blockchain e roteador da rede

Todos os nodos incluem a função de roteamento para participar na rede e podem incluir outras funcionalidades. Todos os nodos validam e propagam as transações e blocos, e descobrem e mantém conexões com os pontos. No exemplo do nodo completo em Um nodo da rede Bitcoin com todas as quatro funções: carteira, minerador, banco de dados completo da blockchain e roteador da rede, a função de roteamento é indicada por um círculo laranja chamado de "Nodo de Roteamento de Rede."

Alguns nodos, chamados de nós completos, também mantém uma cópia completa e atualizada do blockchain. Nodos completos podem verificar de maneira autônoma e autoritária qualquer transação sem referência externa. Alguns nodos mantêm somente uma parte do blockchain e verifica transações utilizando um método chamado *verificação de pagamento simplificada*, ou SPV. Estes nodos são conhecidos como SPV ou nodos peso-leve. No exemplo nodos completos da figura, a função base de dados blockchain nodo completo está indicado por um círculo azul denominado "Blockchain

Completo". Em A rede bitcoin estendida mostrando vários tipos de nodos, gateways e protocolos, nodos SPV estão representados sem o círculo azul, indicando que estes não têm uma cópia completa do blockchain.

Os nodos de mineração competem para criar novos blocos ao utilizarem hardware especializado para resolver os algoritmos de prova-de-trabalho. Alguns nodos de mineração também são nodos completos, mantendo uma cópia completa da blockchain, enquanto outros são nodos peso leve (lightweight) que participam de um pool de mineração e dependem de um servidor de pool para manter um nodo completo. A função de mineração é demonstrada no nodo completo como um círculo preto chamado de "Minerador."

As carteiras de usuários podem fazer parte de um nodo completo, que é o que geralmente ocorre em clientes desktop do bitcoin. Cada vez mais as carteiras de usuário são nodos VSP, especialmente aquelas sendo executadas em dispositivos com poucos recursos como smartphones. A função carteira é demonstrada Um nodo da rede Bitcoin com todas as quatro funções: carteira, minerador, banco de dados completo da blockchain e roteador da rede em como um círculo verde chamado de "Carteira"

Além dos tipos de nodos principais no protocolo P2P bitcoin, existem servidores e nodos executando outros protocolos, como protocolos especializados em mineração-pool e protocolos de acesso de clientes com aplicativos leves (lightweight).

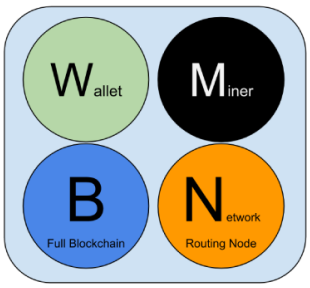
Diferentes tipos de nodos na rede bitcoin estendida mostra os tipos mais comuns de nodos na rede bitcoin estendida.

A Rede Bitcoin Estendida

A rede bitcoin principal, executando o protocolo P2P bitcoin, consistem entre 7.000 e 10.000 nodos ativos rodando várias versões do cliente bitcoin de referência (Bitcoin Core) e algumas centenas de nodos rodando várias outras implementações do protoclo P2P bitcoin, como o BitconJ, Libbitcoin e btcd. Uma pequena porcentagem dos nodos na rede bitcoin P2P também são nodos mineradores, competindo em um processo de mineração, validando transações e criando novos blocos. Várias grandes companhias participam da rede bitcoin ao rodas clientes de nodo completos no cliente Bitcoin Core, com cópias completas da blockchain e um nodo da rede, mas sem funções de mineração ou de carteira. Esses nodos atuam como roteadores de borda da rede, permitindo que vários outros serviços (exchanges, carteiras, exploradores de bloco, processamento de pagamentos a comerciantes) sejam construídos em seu topo.

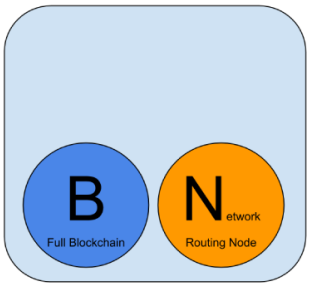
A rede bitcoin estendida inclui a rede executando o protocolo P2P bitcoin, descrita anteriormente, assim como nodos executando protocolos especializados. Ligados a essa rede P2P bitcoin principal existem vários servidores pool e gateways de protocolo que conectam nodos executando outros protocolos. Esses nodos usando outros protocolos são em sua maioria nodos de pool de mineração (ver [ch8])e clientes de carteira leves (lightweight), que não carregam uma cópia completa de blockchain.

A rede bitcoin estendida mostrando vários tipos de nodos, gateways e protocolos demonstra a rede bitcoin estendida com os vários tipos de nodos, servidores gateway, roteadores edge e clientes de carteira, além dos vários protocolos que eles usam para conectar-se uns com os outros.



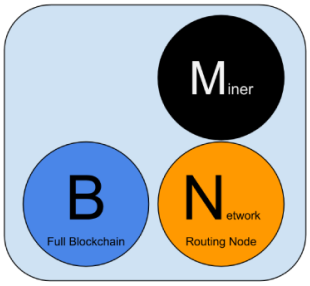
Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.



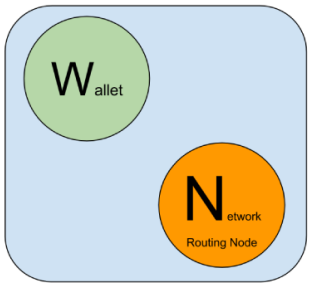
Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.



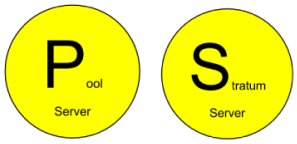
Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.



Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.

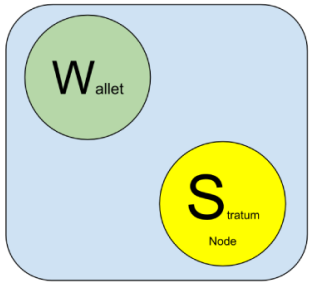


Pool Protocol Servers

Gateway routers connecting the bitcoin P2P network to nodes running other protocols such as pool mining nodes or Stratum nodes.

Mining Nodes

Contain a mining function, without a blockchain, with the Stratum protocol node (S) or other pool (P) mining protocol node.



Lightweight (SPV) Stratum wallet

Contains a Wallet and a Network node on the Stratum protocol, without a blockchain.

Figure 2. Diferentes tipos de nodos na rede bitcoin estendida

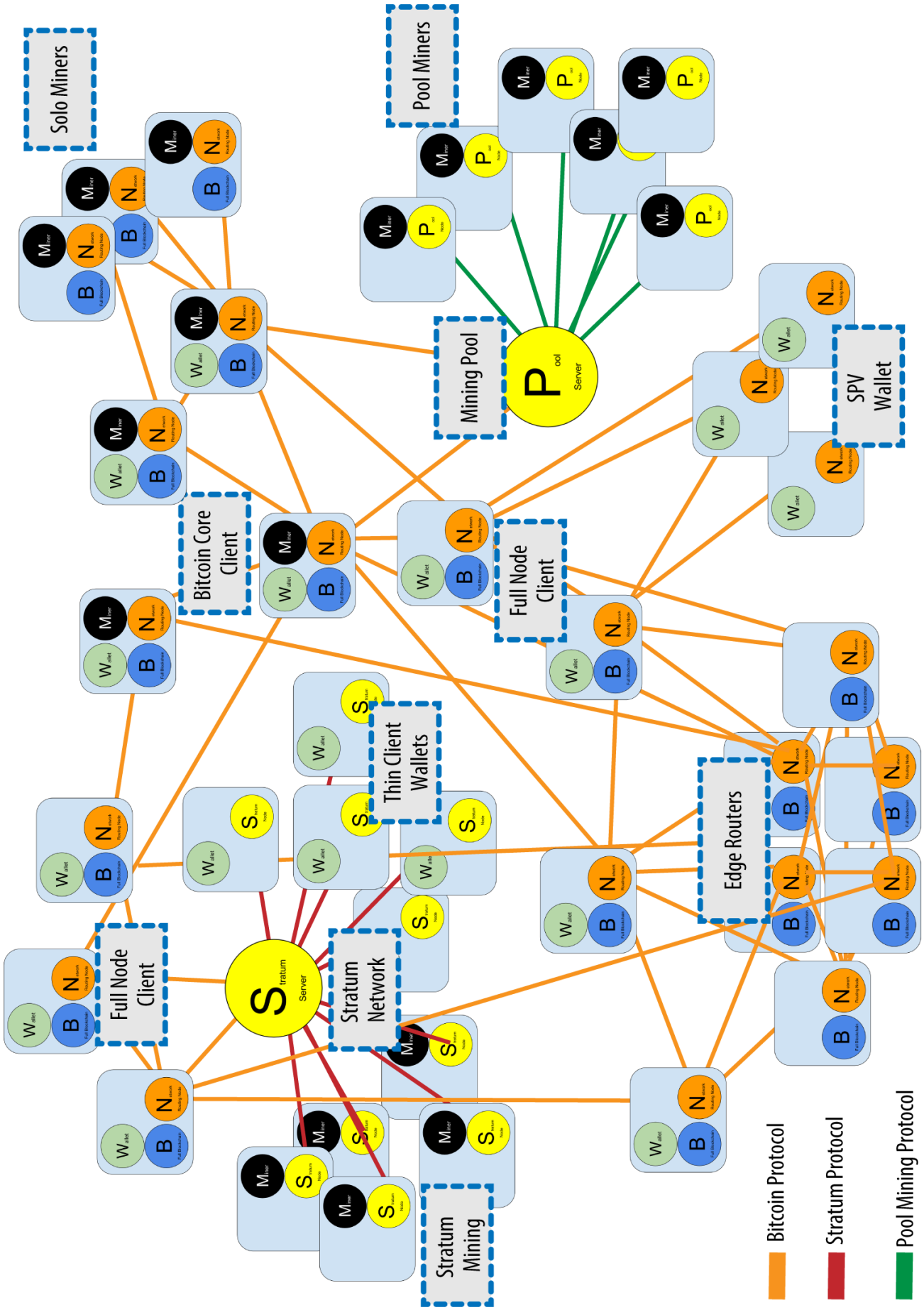

Solo Miners
Pool Miners
Mining Pool
Pool Server
SPV Wallet
Bitcoin Core Client
Full Node Client
Thin Client Wallets
Edge Routers
Stratum Server
Stratum Network
Stratum Mining
Bitcoin Protocol
Stratum Protocol
Pool Mining Protocol

Figure 3. A rede bitcoin estendida mostrando vários tipos de nodos, gateways e protocolos

Descoberta da Rede

Quando um novo nodo é ligado, ele deve descobrir outros nodos bitoins na rede para que possa participar. Para iniciar esse processo, um nodo novo deve descobrir pelo menos um nodo existente na rede para conectar-se a ele. A localização geográfica dos outros nodos é irrelevante; a topologia da rede bitcoin não é definida geograficamente. Logo, qualquer nodo bitcoin existente pode ser selecionado aleatoriamente.

Para se conectar a um ponto conhecido, os nodos estabelecem uma conexão TCP, geralmente na porta 8333 (a porta geralmente conhecida como uma das usadas pelo bitcoin), ou a uma porta alternativa caso tenha sido fornecida. Ao estabelecer a conexão, o nodo iniciará um "aperto de mão" (ver O aperto de mãos inicial entre os pontos) ao transmitir uma mensagem de versão, que contém basicamente informações de identificação, incluindo:

PROTOCOL_VERSION

Uma constante que define a versão do protocolo P2P do bitcoin através da qual o cliente se "comunica" (ex: 70002)

nLocalServices

Uma lista dos serviços locais suportados pelo nodo, atualmente apenas NODE_NETWORK

nTime

A hora atual

addrYou

O endereço IP do nodo remoto da maneira que é visto por esse nodo

addrMe

O endereço IP do nodo local, da maneira que é descoberto pelo nodo local

subver

Uma sub-versão mostrando o tipo de software sendo executado nesse nodo (ex: "/Satoshi:0.9.2.1/")+

BestHeight

A altura dos blocos da blockchain deste nodo

(Veja GitHub para um exemplo da mensagem de rede version.)

O nodo ponto responde com verack para reconhecer e estabelecer uma conexão, e opcionalmente envia sua própria mensagem versão caso ele desejar estabelecer uma conexão recíproca e se conectar como um ponto.

Como um nodo novo encontra seus pontos (peers)? O primeiro método é solicitar o DNS usando várias

"sementes (seeds) de DNS," que são servidores DNS que fornecem uma lista de endereços IP dos nodos bitcoin. Algumas dessas sementes DNS fornecem uma lista estática de endereços IPs de nodos bitcoin de escuta. Algumas dessas sementes DNS são implementações customizadas de BIND (Berkeley Internet Name Daemon) que retornam um conjunto aleatório a partir de uma lista de endereços de nodo bitcoin coletada por um crawler ou um nodo bitcoin de longa-execução. O cliente Bitcoin Core contém os nomes de cinco sementes DNS diferentes. A diversidade de donos e a diversidade de implementação das diferentes sementes DNS oferece um alto nível de confiabilidade para o processo de bootstrapping inicial. No cliente Bitcoin Core, a opção para usar as sementes DNS é controlada pela opção switch -dnsseed (definida como 1 por padrão, para usar a semente DNS).

De maneira alternativa, um nodo bootstrapping que não saiba nada da rede deve receber o endereço IP de pelo menos um nodo bitcoin, a partir do qual ele poderá estabelecer conexões através de novas introduções. O argumento de linha de comando -seednode pode ser usado para conectar a um nodo somente para introduções, utilizando-o como uma semente (seed). Após o nodo semente inicial ser usado para formar as introduções, o cliente irá se desconectar dele e usará os novos pontos recém-descobertos.

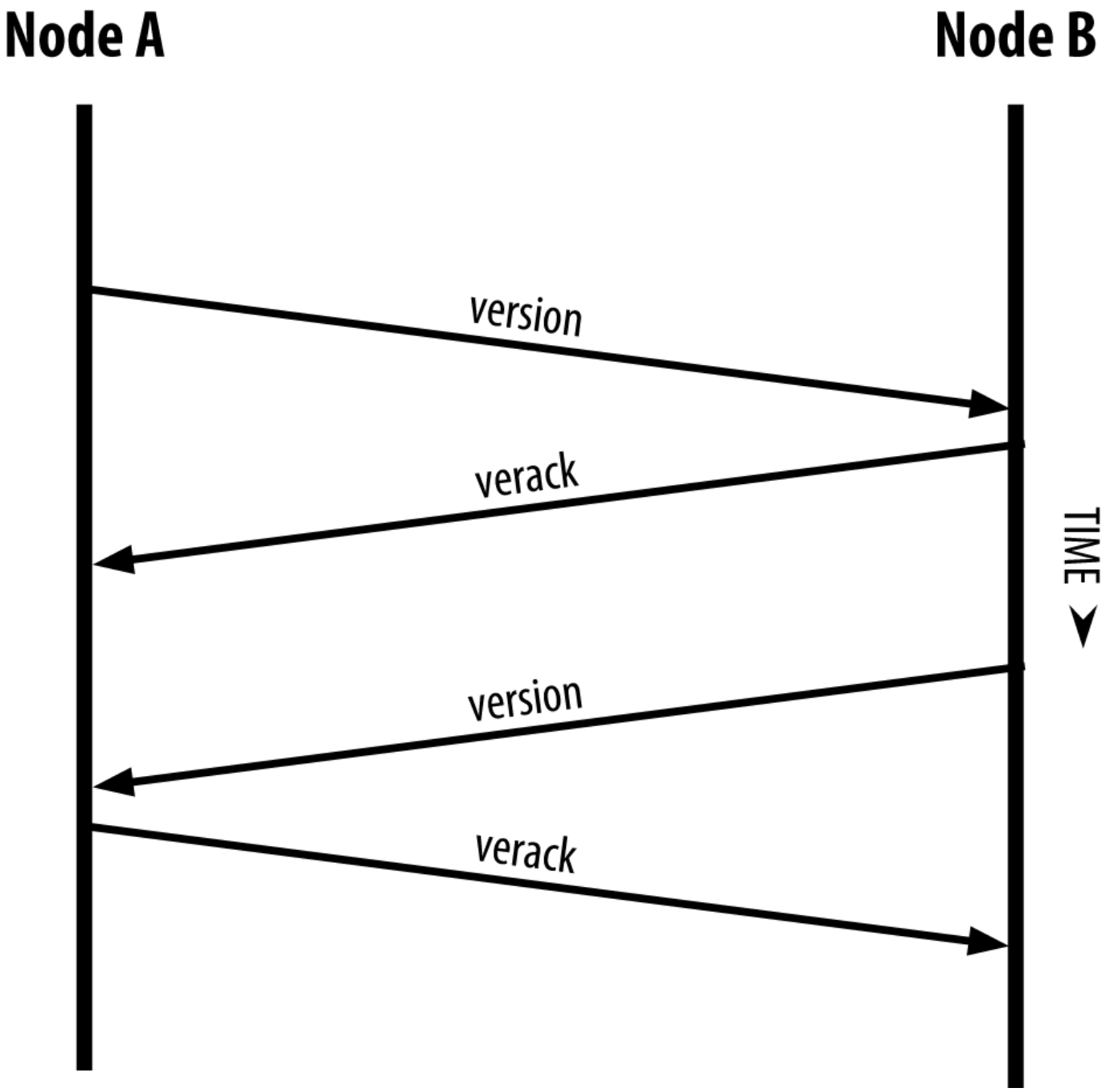


Figure 4. O aperto de mãos inicial entre os pontos

Assim que uma ou mais conexões são estabelecidas, o novo nodo irá enviar uma mensagem addr contendo seu próprio endereço IP para seus vizinhos. Seus vizinhos irão, por sua vez, retransmitir a mensagem addr para seus vizinhos, garantindo que os novos nodos conectados se tornem bem conhecidos e melhor conectados. Adicionalmente, o novo nodo conectado pode enviar getaddr para os vizinhos, solictando-os que retornem uma lista de endereços IP de seus pontos. Dessa maneira, um nodo pode encontrar pontos para conectar-se e divulgar sua existência na rede para que outros nodos o encontrem. Propagação e descoberta do endereço demonstra o protocolo de descoberta de endereço.

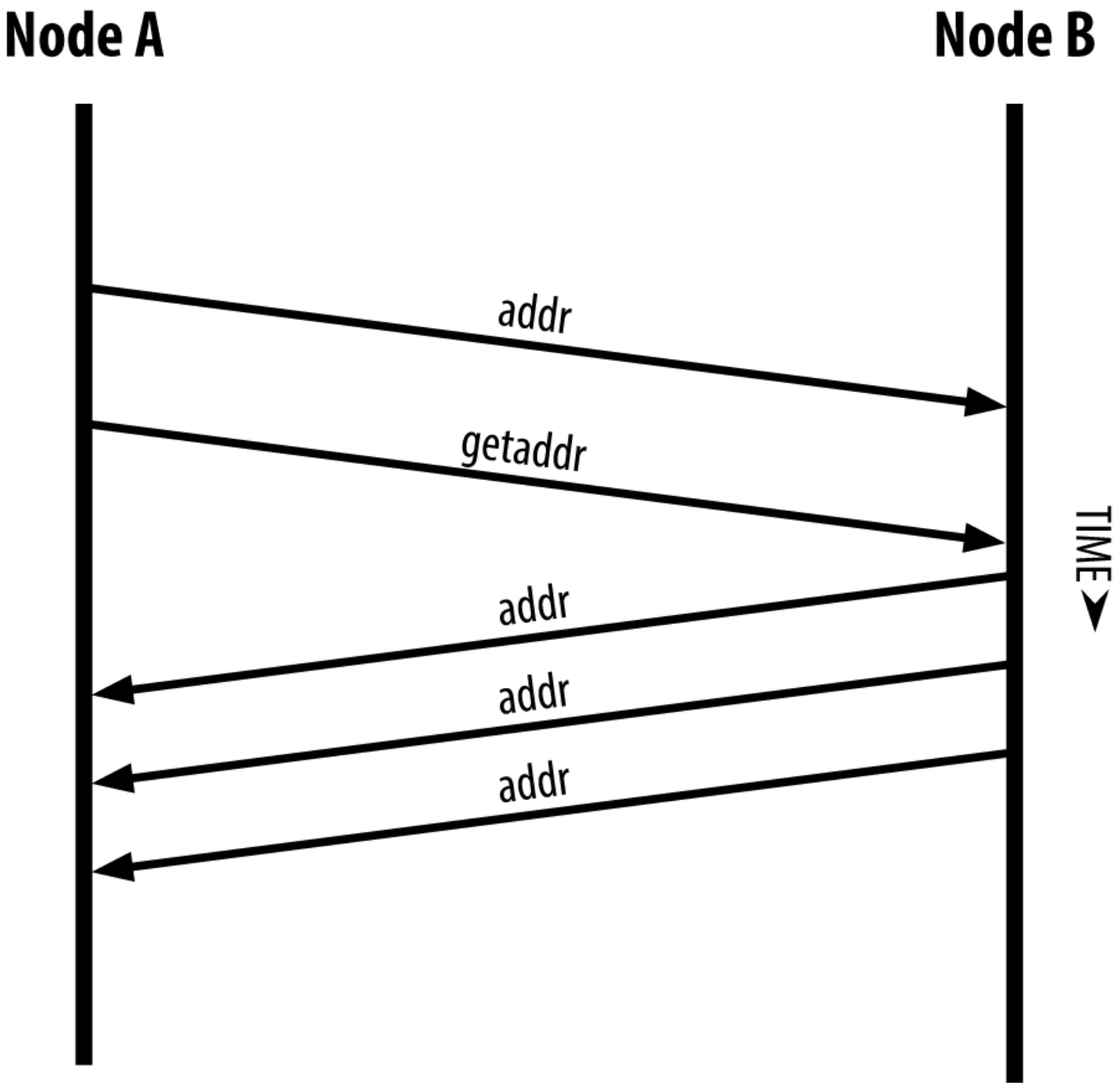


Figure 5. Propagação e descoberta do endereço

Um nodo precisa conectar-se a alguns poucos peers diferentes para estabelecer vários caminhos na rede bitcoin. Os caminhos não são confiáveis—os nodos ficam online e offline—e então o nodo precisa continuar a descobrir novos nodos à medida que ele perde conexões antigas, assim como auxilia outros nodos quando eles fazem bootstrap. Somente uma conexão é necessária para fazer bootstrap, porque o primeiro nodo pode oferecer apresentações para seus nodos pontos e esses pontos podem oferecer novas apresentações subsequentes. Conectar-se a mais de alguns nodos também é desnecessária e desperdiça recursos da rede. Após fazer bootstrap, um nodo irá se lembrar de suas conexões de sucesso mais recentes com pontos, de maneira que se reiniciado ele poderá rapidamente reestabelecer conexões com sua rede de pontos prévia. Se nenhum dos antigos pontos responder à sua solicitação de conexão, o nodo pode usar os nodos sementes para fazer bootstrap novamente.

Em um nodo executando o cliente Bitcoin Core, você pode listar as conexões com os peers através do comando getpeerinfo:

```
$ bitcoin-cli getpeerinfo
```

```
[
    {
        "addr" : "85.213.199.39:8333",
        "services" : "00000001",
        "lastsend" : 1405634126,
        "lastrecv" : 1405634127,
        "bytessent" : 23487651,
        "bytesrecv" : 138679099,
        "conntime" : 1405021768,
        "pingtime" : 0.00000000,
        "version" : 70002,
        "subver" : "/Satoshi:0.9.2.1/",
        "inbound" : false,
        "startingheight" : 310131,
        "banscore" : 0,
        "syncnode" : true
    },
    {
        "addr" : "58.23.244.20:8333",
        "services" : "00000001",
        "lastsend" : 1405634127,
        "lastrecv" : 1405634124,
        "bytessent" : 4460918,
        "bytesrecv" : 8903575,
        "conntime" : 1405559628,
        "pingtime" : 0.00000000,
        "version" : 70001,
        "subver" : "/Satoshi:0.8.6/",
        "inbound" : false,
        "startingheight" : 311074,
        "banscore" : 0,
        "syncnode" : false
    }
]
```

Para desativar a administração automática dos pontos e especificar uma lista de endereços IP, os usuários podem usar a opção -connect=<EndereçoIP> e especificar um ou mais endereços IP. Se essa opção for utilizada, o nodo irá conectar-se somente aos endereços IP selecionados, ao invés de automaticamente descobrir e manter conexões com pontos.

Se não houver tráfico em uma conexão, os nodos irão periodicamente enviar uma mensagem para manter a conexão. Se um nodo não se comunicar em uma conexão por mais de 90 minutos, assume-se que ele esteja desconectado e um novo ponto será procurado. Logo, a rede dinamicamente ajusta-se aos nodos transitórios e aos problemas da rede, e pode crescer e diminuir organicamente conforme necessário, sem a necessidade de um controle central.

Nodos completos

Os nodos completos são nodos que mantém uma blockchain completa com todas as transações. Mais precisamente, eles provavelmente deveriam ser chamados de "nodos com a blockchain completa". Nos anos iniciais do bitcoin, todos os nodos eram nodos completos e atualmente o cliente Bitcoin Core é um nodo com a blockchain completa. Nos últimos dois anos, no entanto, novas formas de clientes bitcoins foram introduzidas que não precisam manter uma blockchain completa, mas são executandos como clientes leves ("lightweight"). Nós examinaremos esses clientes em mais detalhes na próxima seção.

Os nodos com a blockchain completa mantém uma cópia completa e atualizada da blockchain do bitcoin com todas as transações, a qual eles independentemente construiram e verificaram, iniciando desde o primeiro bloco (bloco gênese) e adicionando até o último bloco conhecido da rede. O nodo com a blockchain completa pode verificar independentemente e com autoridade qualquer transação sem depender de qualquer outro nodo ou fonte de informação. O nodo completo depende da rede para receber atualizações sobre novos blocos de transações, as quais ele verifica e incorpora em sua cópia local da blockchain.

Executar um nodo com a blockchain completa proporciona a experiência bitcoin pura: verificação independente de todas as transações, sem a necessidade de dependência ou confiança em qualquer outro sistema. É fácil dizer se você está executando um nodo completo porque ele requer mais de 20 gigabytes de armazenamento persistente (espaço em diasco) para armazenar a blockchain completa. Se você precisa de muito espaço e ele leva dois a três dias para sincronizar com a rede, você está executando um nodo completo. Esse é o preço que se paga para uma liberdade completa e total independência de uma autoridade central.

Existem algumas poucas implementações alternativas dos clientes de bitcoin com a blockchain completa, construídas usando-se diferentes linguagens de programação e arquiteturas de software. No entanto, a implementação mais comum é o cliente de referência Bitcoin Core, também conhecido como o cliente Satoshi. Mais de 90% dos noso na rede bitcoin executam várias versões do Bitcoin Core. Ele é identificado como "Satoshi" na string sub-version enviada na mensagem version e é mostrado pelo comando getpeerinfo como nós vimos anteriormente; por exemplo, /Satoshi:0.8.6/.

Trocando o "Inventário"

A primeira coisa que um nodo completo fará assim que se conectar aos pontos é tentar construir uma blockchain completa. Se ele for um nodo recém criado que não tenha nenhuma parte da blockchain, ele só terá um bloco, o bloco gênese, que é incluído estaticamente no software do cliente. Iniciando com o bloco #0 (o bloco gênese), o novo nodo terá que fazer o download de centenas de milhares de blocos para sincronizar-se com a rede e re-estabelecer a blockchain completa.

O processo de sincronização da blockchain é iniciado a partir da mensagem version, pois ela contém o BestHeight, a altura atual (número de blocos) da blockchain de um nodo. Ao ver a mensagem version de seus pontos, um nodo saberá quantos blocos cada um deles tem, e será capaz de comparar com o número de blocos que existem em sua própria blockchain. Os nodos pareados irão trocar uma mensagem getblocks, a qual contém o hash (impressão digital) do bloco mais alto de suas blockchains locais. A seguir, um ponto será capaz de identificar que o hash recebido pertence ao bloco que não está no topo, mas que pertence a um bloco antigo, logo deduzindo que sua blockchain local é mais comprida do que a de seus pontos.

O ponto que tem a blockchain mais longa possui mais blocos que o outro nodo e pode identificar quais blocos o outro nodo precisa para ficar "em dia". Ele irá identificar os primeiros 500 blocos para compartilhar e transmitirá seus hashes usando uma mensagem inv (inventory) . O nodo que estiver com esses blocos faltando irá então recebê-los ao emitir uma série de mensagens getdata solicitando os dados completos dos blocos e identificando os blocos solicitados usando os hashes da mensagem inv.

Vamos assumir, por exemplo, que um nodo tenha apenas o bloco gênese. Ele irá então receber uma mensagem inv de seus pontos contendo os hashes dos próximos 500 blocos na cadeia. Ele irá começar a solicitar blocos de todos os seus pontos conectados, espalhando a carga e garantindo que ele não sobrecarregue nenhum de seus pontos com requisições. O nodo mantém um registro de quantos blocos estão "em trânsito" por conexão com ponto, significando os blocos que ele solicitou mas não recebeu, certificando-se que ele não exceda um limite (MAX_BLOCKS_IN_TRANSIT_PER_PEER). Dessa maneira, se ele precisar de muitos blocos, ele irá somente irá solicitar novos quando as solicitações prévias forem completadas, permitindo que os pontos controlem o ritmo de updates e não sobrecarregando a rede. À medida que cada bloco é recebido, ele é adicionado à blockchain, como nós iremos ver em [blockchain]. À medida que a blockchain é gradultamente construída, mais blocos são solicitados e recebidos, e o processo continua até que o nodo alcance o resto da rede.

Esse processo de comparar a blockchain local com os pontos e adquirir quaisquer blocos em falta acontece sempre que um bloco fica offline por algum período de tempo. Tenha um nodo ficado offline por alguns minutos e esteja com alguns pouco blocos em falta, ou tenha ficado um mês offeline e esteja com milhares de blocos em falta, ele começa ao enviar getblocks, recebeuma resposta inv e inicia baixando os blocos remanescentes. Nodo sincronizando a blockchain ao adquirir blocos de um ponto demonstra o inventário e o protocolo de propagação dos blocos.

Nodos de Verificação Simplificada de Pagamento (VSP)

Nem todos os nodos tem a habilidade de armazenar a blockchain completa. Muitos clientes bitcoin são projetados para serem executados em dispositivos com limitações de armazenamento e energia, como smartphones, tablets ou sistemas embutidos. Para esses dispositivos, um método de *verificação simplificada de pagamento* é usado para permití-los que operem sem terem que armazenar a blockchain completa. Esses tipos de clientes são chamados clientes VSP (em inglês, SPV) ou clientes leves. Conforme a adoação do bitcoin aumenta, o nodo VSP está se tornando a forma mais comum de nodo bitcoin, especialmente para carteiras bitcoin.

Os nodos VSP baixam apenas os cabeçalhos dos blocos e não baixam as transações incluídas em cada

bloco. A cadeia de blocos resultante, sem as transações, é 1.000 vezes menor do que a blockchain completa. Os nodos VSP não podem construir um figura completa de todas as UTXOs que estão disponíveis para serem gastas porque eles não conhecem todas as transações da rede. Os nodos VSP verificam as transações usando uma metodologia levemente diferente, a qual se baseia nos pontos que fornecem, sob demanda, consultas de apenas algumas partes relevantes da blockchain.

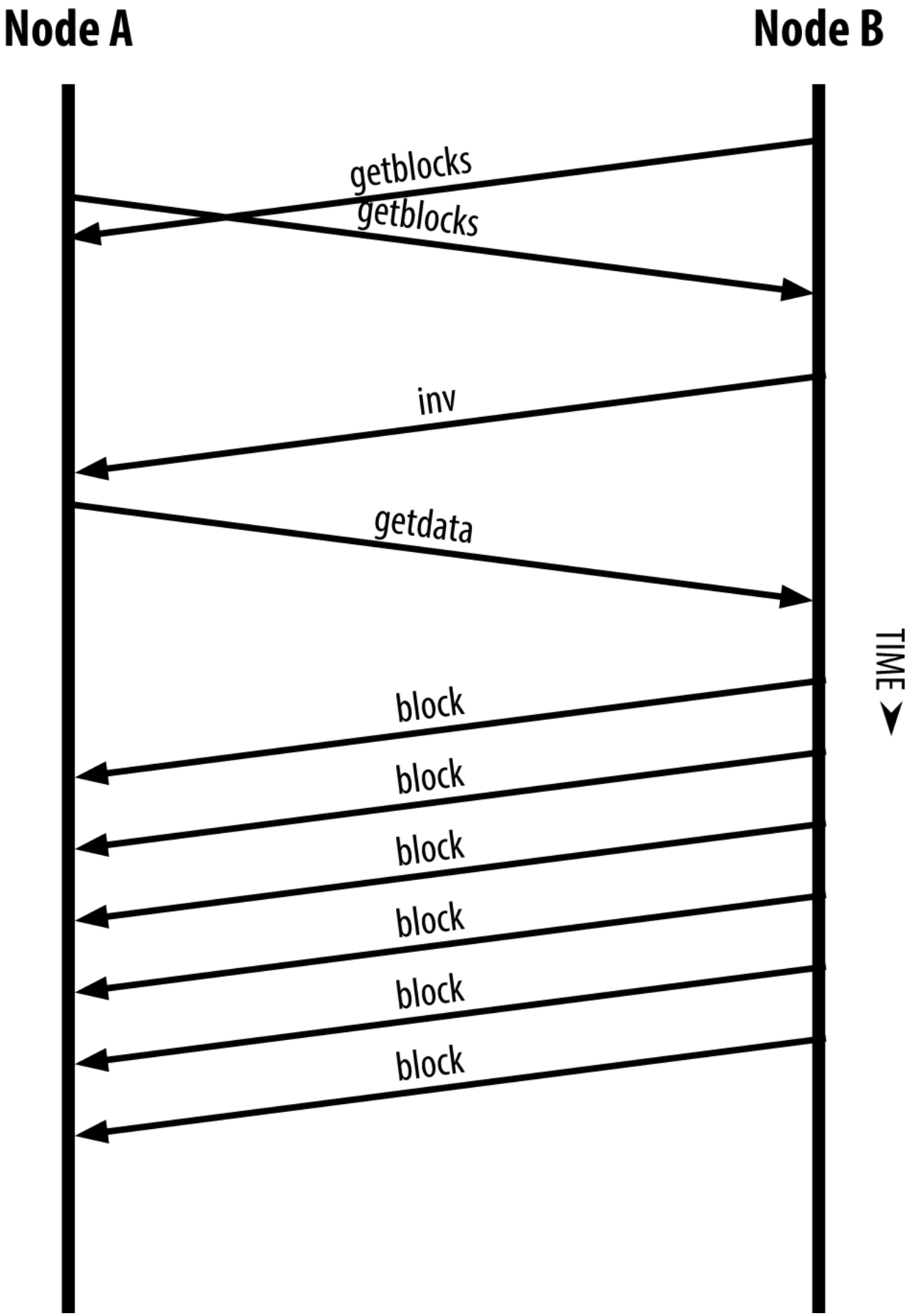

Node A
Node B
getblocks
getblocks
inv
getdata
block
block
block
block
block
block
TIME

Figure 6. Nodo sincronizando a blockchain ao adquirir blocos de um ponto

Como uma analogia, um nodo completo é como um turista em uma cidade estranha, equipado com um mapa detalhado de cada rua e cada endereço. Em comparação, um nodo de VSP é como um turista em uma cidade estranha perguntando a desconhecidos aleatórios na rua por orientações de como chegar a um lugar, enquanto ele sabe apenas o nome de uma avenida principal. Embora ambos os turistas possam verificar a existência de uma rua ao visitá-la, o turista sem o mapa não sabe o que se passa em cada rua colateral e não sabe quais outras ruas existem. Posicionado na frente da Rua da Igreja 23, o turista sem o mapa não tem como saber se existem uma dúzia de outros endereços "Rua da Igreja 23" na cidade e se ele está no endereço correto. A melhor chance de um turista sem mapa é perguntar para um número suficiente de pessoas e torcer para que alguns deles não estejam tentando lhe passar a perna.

A verificação simplificada de pagamento verifica as transações através de referências às suas *profundidades* na blockchain, ao invés de sua *altura*. Enquanto um nodo com a blockchain completa irá construir uma cadeia completamente verificada de milhares de blocos e transações que siga pela blockchain (retrospectivamente no tempo) até o bloco gênese, um nodo de VSP irá verificar a cadeia de todos os blocos (mas não todas as transações) e irá ligar essa cadeia à transação de interesse.

Por exemplo, ao examinar uma transação no bloco 300.000, um nodo completo segue todos os 300.000 blocos até o bloco gênese e constrói um banco de dados completo de UTXO, estabelecendo a validade da transação ao confirmar que a UTXO ainda não foi gasta. Um nodo de VSP não pode validar se a UTXO ainda não foi gasta. Ao invés disso, o nodo de VSP irá estabelecer uma conexão entre a transação e o bloco que a contém, usando um *caminho de Merkle* (veja [merkle_trees]). Então, o nodo de VSP aguarda até ver os seis blocos do 300.0001 até o 300.006 empilhados sobre o topo do bloco ontendo a transação e verifica-a ao estabelecer sua profundidade sob os blocos 300.006 a 300.001. O fato de que outros nodos na rede aceitaram o bloco 300.000 e então fizeram o trabalho necessário para produzir mais seis blocos no seu topo é a prova, através de proxy, que a transação não foi um gasto duplo.

Um nodo de VSP não pode ser convencido de que uma transação existe em um bloco quando a transação de fato não existe. O nodo de VSP estebelece que uma transação existe em um bloco ao solicitar um caminho de merkle como prova e ao validar a prova de trabalho na cadeia de blocos. Entretanto, a existência de uma transação pode ser "escondida" de um nodo de VSP. Um nodo de VSP pode provar definitivamente que uma transação existe, mas não pode verificar que uma transação, como um gasto duplo de uma mesma UTXO, não existe porque ele não tem um registro de todas as transações. Essa vulnerabilidade pode ser usada em um ataque de negação de serviço (DOS) ou para um ataque de gasto duplo contra os nodos de VSP. Para defender-se contra isso, um nodo de VSP precisa se conectar aleatoriamente a vários nodos, para aumentar a probabilidade de que ele esteja em contato com pelo menos um nodo honesto. Essa necessidade de se conectar aleatoriamente significa que os nodos de VSP também são vulneráveis a ataques de particionamento de rede ou ataques Sybil, onde eles são conectados a nodos ou redes fakes e não tem acesso a nodos honestos ou à rede bitcoin verdadeira.

Para a maioria das funções práticas, os nodos VPS bem-conectados são seguros o suficiente, demonstrando um equilíbrio ideal entre necessidade de recursos, praticidade e segurança. Para segurança infalível, no entanto, nada é superior do que executar um nodo com a blockchain completa.

TIP	Um nodo que tem a blockchain completa verifica a transação ao fazer uma pesquisa em sua cadeia inteira de milhares de blocos, para se certificar de que a UTXO já não foi gasta previamente, enquanto um nodo VSP verifica ao pesquisar quão profundo o bloco está "enterrado" sob os vários blocos acima dele.

Para receber os cabeçalhos dos blocos, os nodos de VSP usam uma mensagem getheaders ao invés de getblocks. O ponto que responder irá enviar até 2.000 cabeçalhos de blocos usando uma mensagem headers única. O processo é o mesmo que o utilizado por um nodo completo receber os blocos completos. Os nodos de VSP também definem um filtro na conexão com os pontos, para filtrar a transmissão de blocos futuros e transações enviadas pelos pontos. Quaisquer transações de interesse são recebidas usando uma requisição getdata. Em resposta, os pontos geram uma mensagem tx contendo as transações. Nodo SPV sincronizando os cabeçalhos dos blocos mostra a sincronização dos cabeçalhos dos blocos.

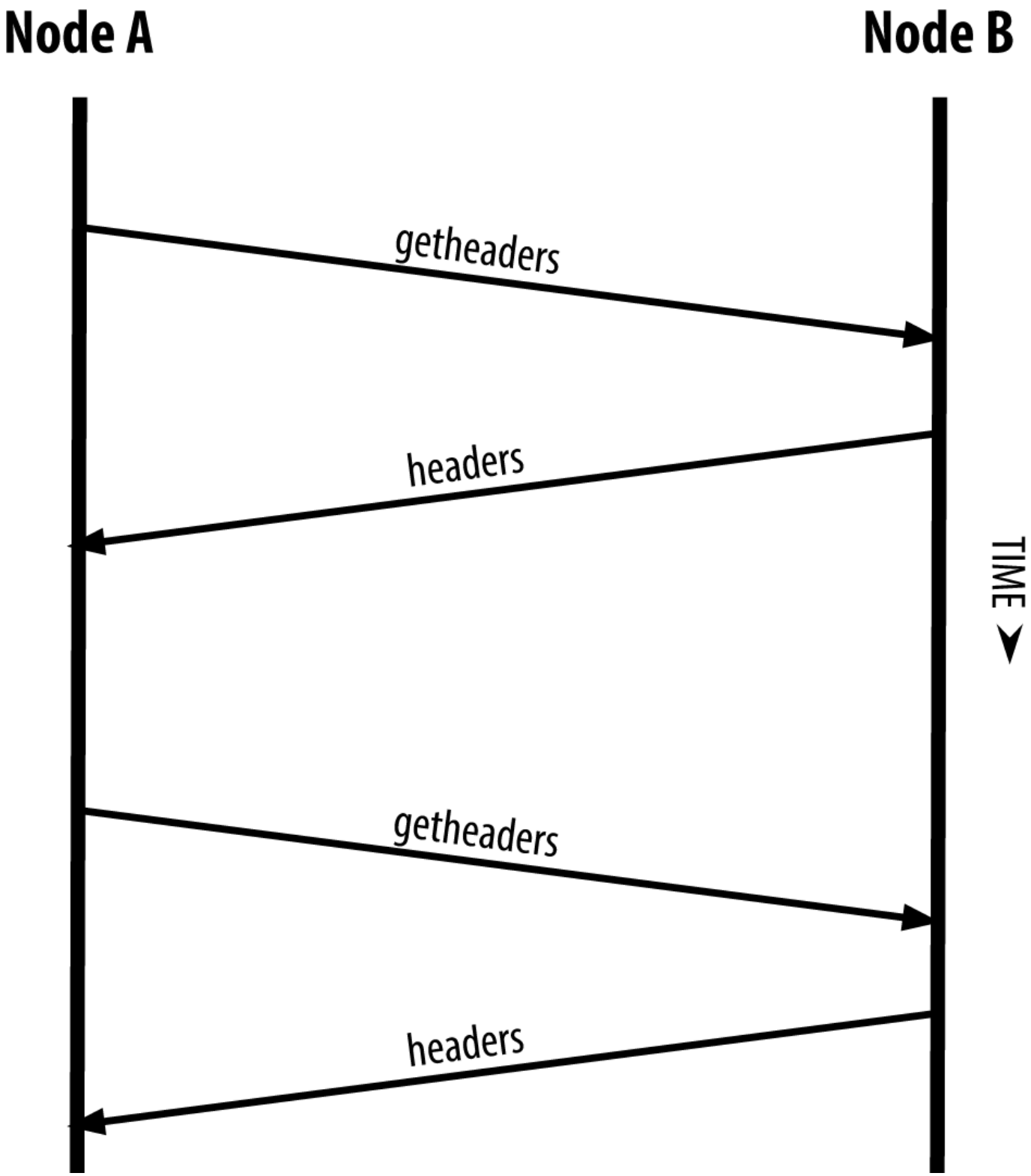


Figure 7. Nodo SPV sincronizando os cabeçalhos dos blocos

Como os nodos de VSP precisam baixar transações específicas para verificá-las seletivamente, eles também criam um risco de privacidade. Ao contrário dos nodos com a blockchain completa, que baixam todas as transações no interior de cada bloco, as solicitações por dados específicos que são feitas pelos nodos de VSP podem inadvertidamente revelar os endereços em suas carteiras. Por exemplo, um terceiro monitorando uma rede poderia monitorar todas as transações solicitadas por

uma carteira em um nodo de VSP e usá-las para associar endereços bitcoins com o usuário dessa carteira, destruindo a privacidade do usuário.

Logo após a introdução dos nodos VSP/peso leve, os desenvolvedores bitcoin adicionaram uma funcionalidade chamada *filtros bloom* para resolver os riscos de privacidade dos nodos VSP. Os filtros bloom permitem que os nodos VSP recebam um conjunto de transações sem que revelem precisamente quais endereços eles estão interessados, através de um mecanismo de filtros que utiliza probabilidades ao invés de padrões fixos.

Filtros Bloom

Um filtro bloom é um filtro de busca probabilístico, uma maneira de descrever um padrão desejado sem especificá-lo exatamente. Os filtros bloom oferecem uma maneira eficiente de expressar um padrão de brusca enquanto protegem a privacidade. Eles são usados pelos nodos VSP para requisitar seus pontos por transações coincidindo com um padrão específico, sem revelar exatamente quais endereços eles estão procurando.

Na analogia prévia, uma turista sem um mapa está perguntando por orientações para um endereço específico, na "Rua Church 23". Se ela perguntar por orientações para estrangeiros nessa rua, ela inadvertidamente revelará seu destino. Um filtro bloom é como se perguntasse, "Existem ruas nessa vizinhança cujo nome termina em R-C-H?" Uma pergunta como essa revela um pouco menos sobre o destino desejado do que perguntando por "Rua Church 23". Usando essa técnica, a turista poderia especificar o endereço desejado em mais detalhes como "termina em U-R-C-H" ou menos detalhe como "termina em H". Ao variar a precisão de sua busca, a tursta revela mais ou menos informações, às custas de resultados mais ou menos específicos. Se ela perguntar um padrão menos específico, ela receberá muito mais endereços possíveis e terá uma privacidade maior, mas muitos dos resultados serão irrelevantes. Se ela perguntar por um padrão muito específico, ela receberá poucos resultados, mas perderá em privacidade.

Os filtros bloom servem essa função ao permitir que um nodo VSP especifique um padrão de busca para transações que possa ser refinado de acordo com precisão ou privacidade. Um filtro bloom mais específico irá produzir resultados precisos, mas às custas de revelar quais endereços são usados na carteira do usuário. Um filtro bloom menos específico irá produzir mais dados sobre mais transações, muitas irrelevantes para o nodo, mas permitirá ao nodo que mantenha uma privacidade maior.

Um nodo de VSP irá inicializar um filtro bloom como "vazio" e nesse estado o filtro bloom não irá corresponder a nenhum padrão. O nodo de VSP irá então fazer uma lista de todos os endereços em sua carteira e criará um padrão de b usca que corresponda ao output da transação correspondente a cada endereço. Geralmente, o padrão de busca é um script pay-to-public-key-hash que é o script de locking esperado que estará presente em qualquer transação pagando ao public-key-hash (endereço). Se o nodo de VSP estiver registrando o saldo de um endereço P2SH, o padrão de busca será um script pay-to-script-hash. O nodo de VSP adiciona então cada um dos padrões de busca ao filtro bloom, de maneira que o filtro bloom possa reconhecer o padrão de busca se ele estiver presente em uma transação. Por fim, o filtro bloom é enviado ao ponto e o ponto o utiliza para corresponder as transações para transmissão para o nodo de VSP.

Os filtros bloom são implementados como uma array de tamanho variável de N dígitos binários (um campo bit) e um número variável de M funções hash. As funções hashes são projetadas para sempre produzir um output entre 1 e N, correspondendo à array de dígitos binários. As funções hash são geradas deterministicamente, de maneira que qualquer nodo implementando um filtro bloom sempre usará as mesmas funções hash e receberá os mesmos resultados para um input específico. Ao escolher filtros bloom de diferentes comprimentos (N) e um número diferente (M) de funções hash, o filtro bloom pode ser melhorado, variando o nível de acurácia, e, portanto, de privacidade.

Em Um exemplo de um filtro bloom simplístico, com um campo de 16-bit e três funções hash, nós usamos um array muito pequeno de 16 bits e um conjunto de três funções hash para demonstrar como os filtros bloom funcionam.

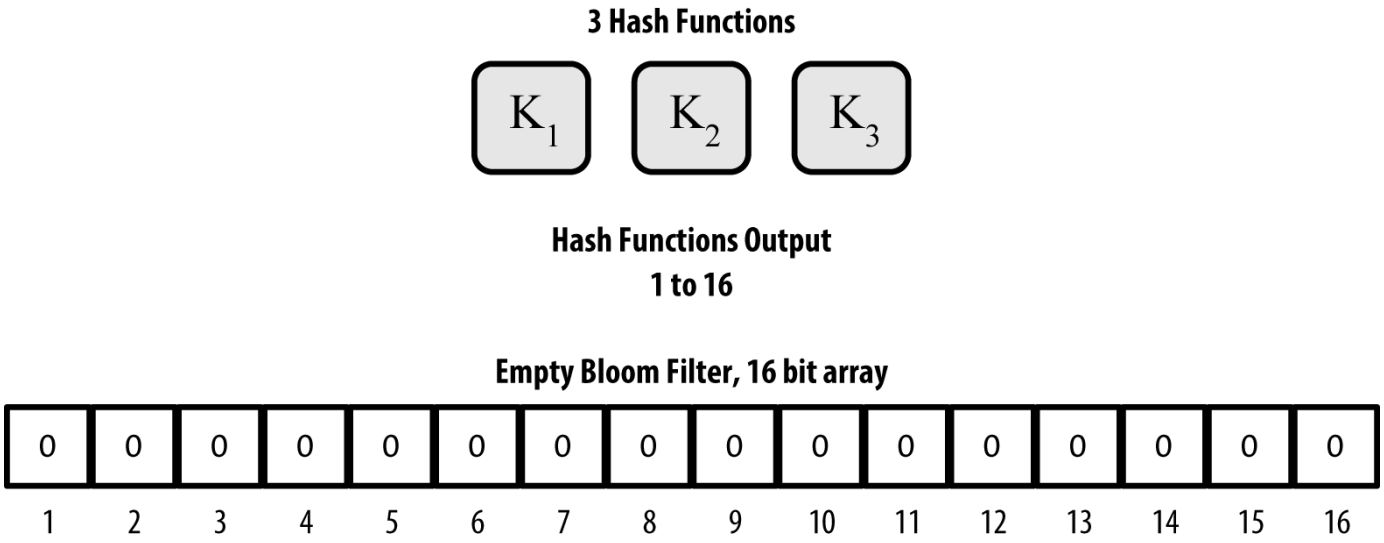


Figure 8. Um exemplo de um filtro bloom simplístico, com um campo de 16-bit e três funções hash

O filtro bloom é inicializado de maneira que a array de bits seja toda de zeros. Para adicionar um padrão ao filtro bloom, o padrão é "hashado" por cada função hash em cada vez. A aplicação da primeira função de hash nos resultados do input resulta em um número entre 1 e N. O bit correspondente na array (indexado de 1 a N) é encontrado e definido como 1, logo, registrando o output da função hash. Logo, a próxima função hash é usada para definir outro bit, e assim por diante. Uma vez que todas as funções hash M forem aplicadas, o padrão de busca será "registrado" no filtro bloom como M bits que mudaram de 0 para 1.

Adicionando um padrão "A" para o nosso filtro bloom simples é um exemplo da adição de um padrão "A" para o filtro bloom simples mostrado em Um exemplo de um filtro bloom simplístico, com um campo de 16-bit e três funções hash.

Adicionar um segundo padrão é tão simples quanto repetir esse processo. O padrão é "hashado" por cada função hash de cada vez e o resultado é registrado ao definir os bits como 1. Note que à medida que o filtro bloom é preenchido com mais padrões, um resultado de função hash pode coincidir com um bit que já esteja definido como 1, neste caso o bit não é modificado. Em essência, à medida que mais padrões gravam em bits sobreponentes, o filtro bloom começa a se tornar saturado com mais bits definidos como 1 e a acurácica do filtro diminui. É por isso que o filtro é uma estrutura de dados

probabilística—ele se torna menos acurado conforme mais padrões são adicionados. A acurácica depende do número de funções hash (M). Uma array de bit maior e mais funções hashes podem registrar mais padrões com maior acurácica. Uma array de bit menor ou menos funções hash irá registrar menos padrões e terá menor acurácia.

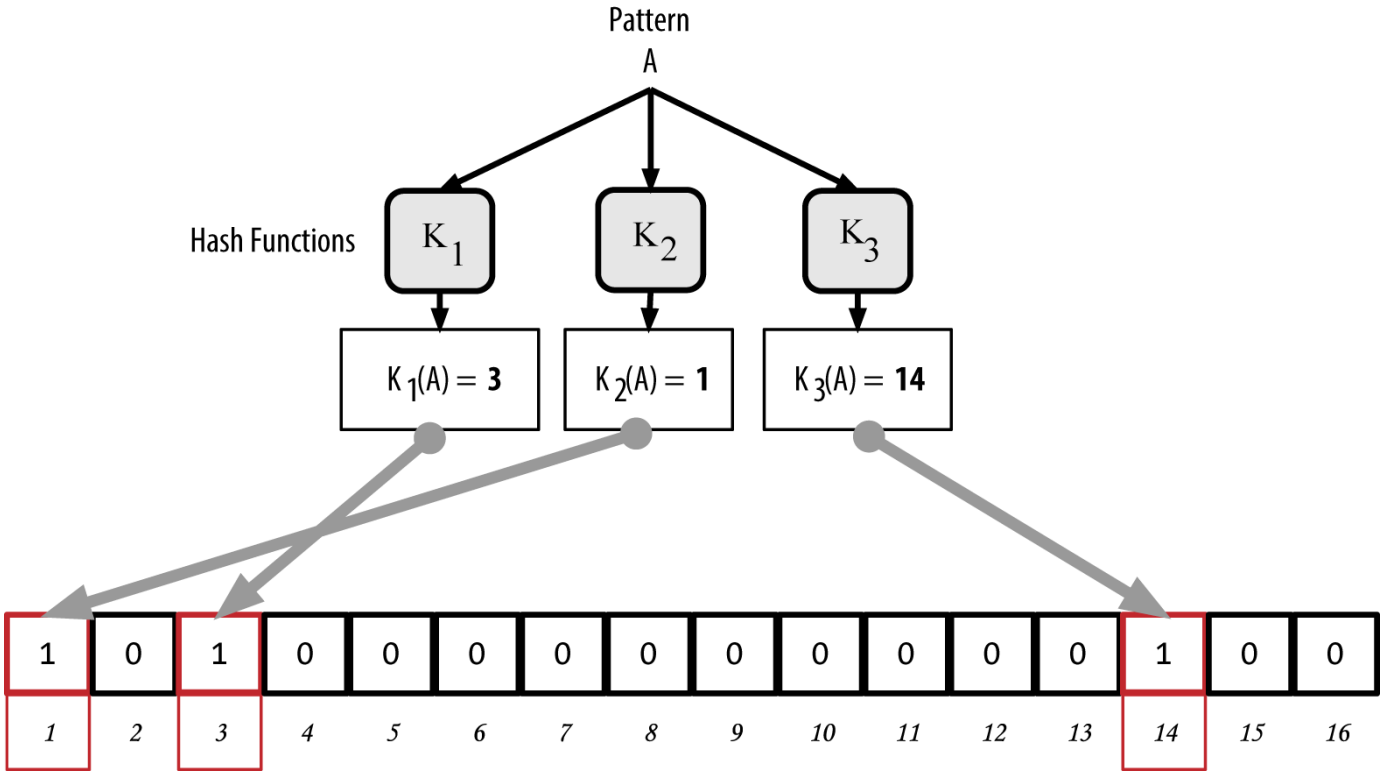


Figure 9. Adicionando um padrão "A" para o nosso filtro bloom simples

Adicionando um segundo padrão "B" para o nosso filtro bloom simples é um exemplo da adição de um segundo padrão "B" para o filtro bloom simples.

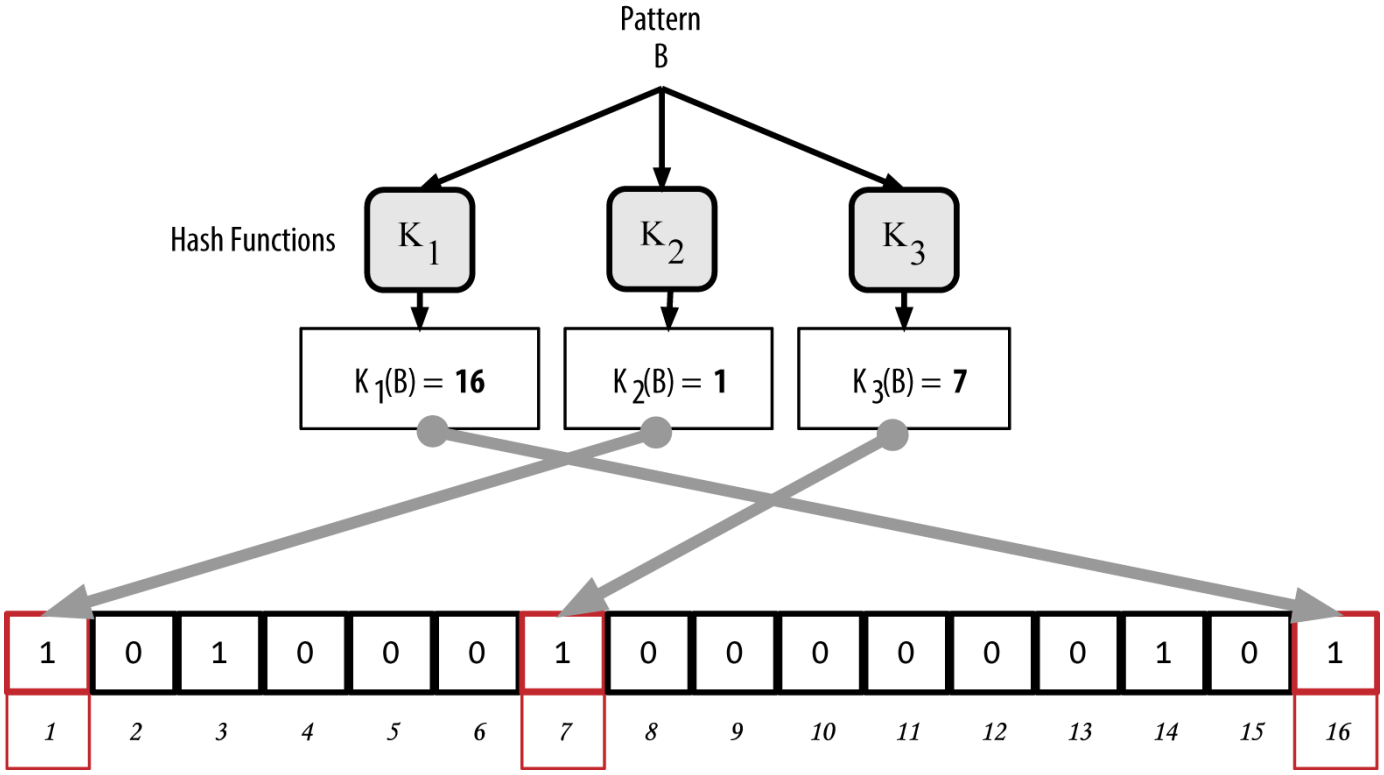


Figure 10. Adicionando um segundo padrão "B" para o nosso filtro bloom simples

Para testar se um padrão faz parte de um filtro bloom, o padrão é "hashado" por cada função hash e o padrão bit resultante é testado contra a bit array. Se todos os bits indexados pelas funções hash são definidas para 1, então o padrão é *provavelmente* registrado no filtro bloom. Como os bits podem ser definidos devido a sobreposição de múltiplos padrões, a resposta não é certa, ao invés disso, ela é probabilística. Em termos simples, uma correspondência positiva em filtro bloom é um "Talvez, Sim."

Testando a existência de um padrão "X" no filtro bloom. O resultado é uma correspondência positiva probabilística, significando "Talvez." é um exemplo de teste da existência do padrão "X" no filtro bloom simples. Os bits correspondentes estão definidos como 1, então o padrõa é provavelmente uma correspondência.

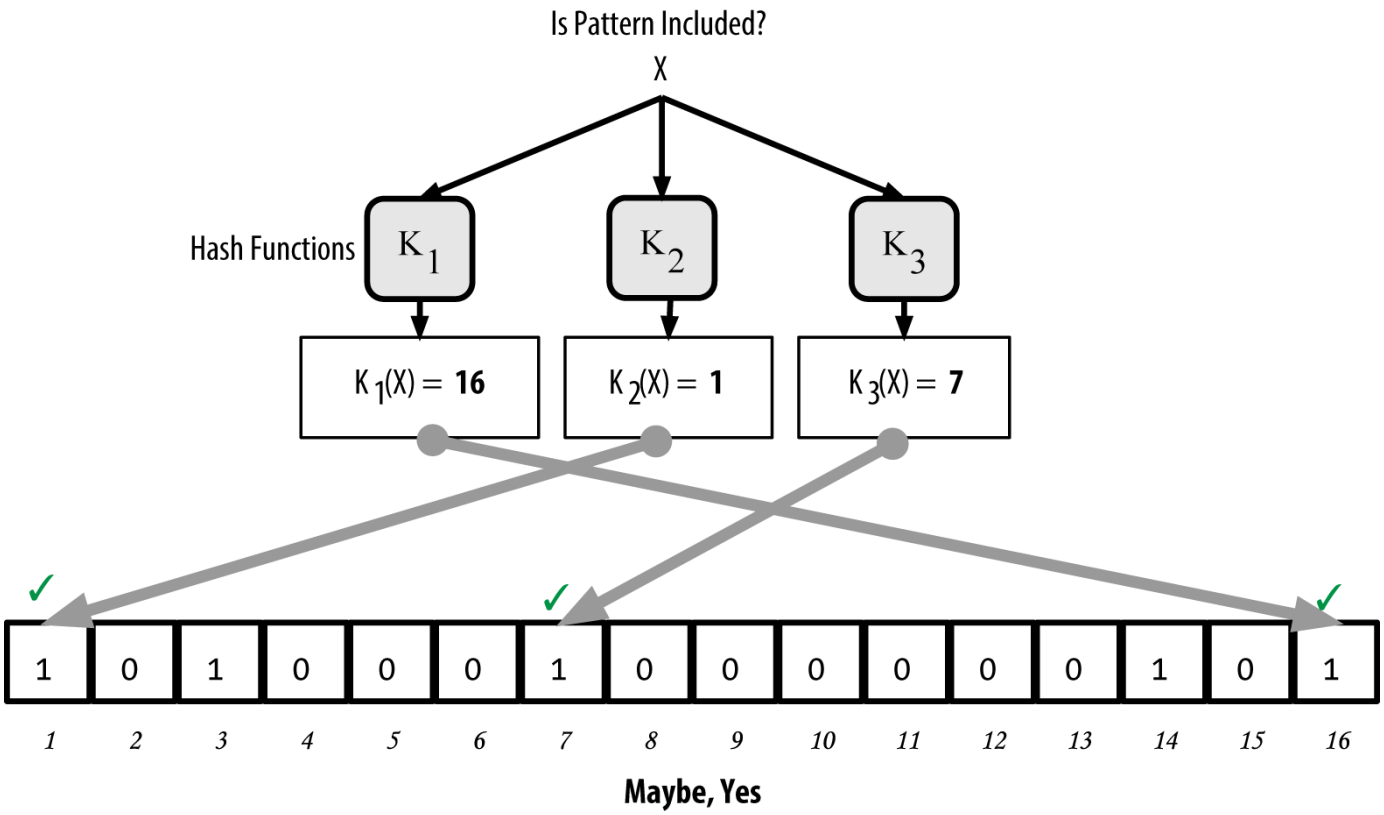


Figure 11. Testando a existência de um padrão "X" no filtro bloom. O resultado é uma correspondência positiva probabilística, significando "Talvez."

Por outro lado, se um padrão for testado contra um filtro bloom e qualquer um dos bits estiver definido como 0, isso prova que o padrão não foi registrado no filtro bloom. O resultado negativo não é uma probabilidade, ele é uma certeza. Em termos simples, uma correspondência negativa em um filtro bloom é um "Definitivamente Não!"

Testando a existência do padrão "Y" no filtro bloom. O resultado é uma correspondência negativa definitiva, significando "Definitivamente Não!" é um exemplo de um teste da existência do padrão "Y" em um filtro bloom simples. Um dos bits correspondentes está definido como 0, então o padrão definitivamente não é uma correspondência.

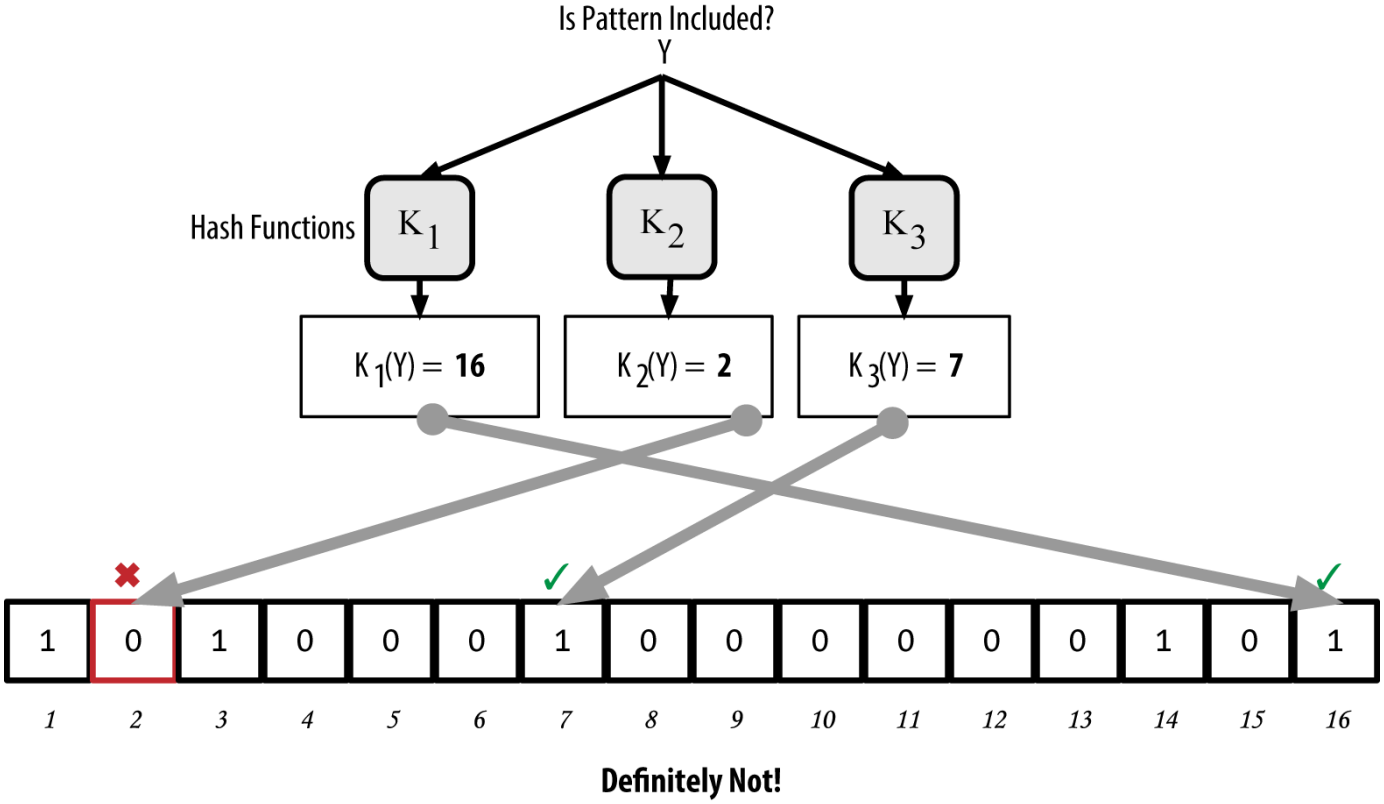


Figure 12. Testando a existência do padrão "Y" no filtro bloom. O resultado é uma correspondência negativa definitiva, significando "Definitivamente Não!"

A implementação de filtros bloom do Bitcoin é descrita em Bitcoin Improvement Proposal 37 (BIP0037). Veja [appdxbitcoinimpproposals] ou acesse o GitHub.

Filtros Bloom e Atualizações de Inventário

Filtros Bloom são usados para filtrar as transações (e os blocos que as contém) que um nodo VPS recebe de seus pontos. Os nodos VPS irão criar um filtro que corresponde somente aos endereços contidos na carteira do nodo VPS. O nodo VPS irá então enviar uma mensagem filterload para o ponto, contendo o filtro bloom para ser usado na conexão. Após um filtro ser estabelecido, o ponto irá então etestar cada output da transação contra o filtro bloom. Somente as transações que correspondem ao filtro serão enviadas para o nodo.

Em resposta a uma mensagem getdata vindo do nodo, os pontos irão enviar uma mensagem merkleblock que contém somente os cabeçalhos de bloco para os blocos correspondentes ao filtro e um caminho merkle (ver [merkle_trees]) para cada transação correspondente. O ponto também enviará mensagens tx contendo as transações que correspondem ao filtro.

O nodo definindo o filtro bloom pode adicionar padrões ao filtro de maneira interativa ao enviar uma mensagem filteradd. Para limpar o filtro bloom, o nodo pode enviar uma mensagem filterclear. Como não é possível remover um padrão de um filtro bloom, um nodo tem que limpar e reenviar um novo filtro bloom se um padrão não é mais desejado.

Pools de Transações

Quase todo nodo na rede bitcoin mantém uma lista temporária de transações não-confirmadas chamada de *pool de memória*, *mempool* ou *pool de transações*. Os nodos usam esse pool para manter um acompanhamento das transações que são conhecidas pela rede mas que ainda não foram incluídas na blockchain. Por exemplo, um nodo contendo uma carteira de usuário utilizará um pool de transação para acompanhar os pagamentos para essa carteira que foram recebidos na rede, mas que ainda não foram confirmados.

As transações são recebidas e verificadas, sendo adicionadas ao pool de transações e transmitidas aos nodos vizinhos para serem propagadas para a rede.

Algumas implementações de nodos também mantém um pool separado de transações órfãs. Caso um input de transação referir-se a uma transação que ainda não é conhecida, como um pai desconhecido, a transação órfã será armazenada temporariamente no pool órfão até que a transação pai chegue.

Quando uma transação é adicionada ao pool de transações, verifica-se o pool órfão para quaisquer órfãos que sejam referenciados para esses outputs de transação (seus filhos). Quaisquer órfãos correspondentes são então validados. Se válidos, eles são removidos do pool órfão e adicionados ao pool de transação, completando a cadeia que iniciou com a transação pai. Na presença de uma transação recém-adicionada, que não é mais uma órfã, o processo é repetido recursivamente em busca de quaisquer outros descendentes, até que não se encontre mais nenhum descendente. Através desse processo, a chegada de uma transação pai desencadeia uma reconstrução em cascata de uma cadeia completa de transações interdependentes ao reunir os órfãos com seus pais ao longo de toda a cadeia.

Tanto o pool de transação quanto o pool de órfãs (quando implementado) são armazenados na memória local e não são salvos em um armazenamento persistente; ao invés disso, eles são populados dinamicamente a partir das mensagens de rede que chegam. Quando um nodo é iniciado, ambas as pools são esvaziadas e são gradualmente populadas com as novas transações recebidas na rede.

Algumas implementações do cliente bitcoin também mantém um banco de dados de UTXO (um pool de UTXO), que é o conjunto de todos os outputs da blockchain que não foram gastos. Embora o nome "pool de UTXO" pareça semelhante ao pool de transações, ele representa um conjunto de dados diferente. Ao contrário dos pools de transação e órfãs, o pool UTXO não é inicializado vazio, ao invés disso contém milhões de entradas de outputs de transações não gastos, incluindo alguns antigos, que datam de 2009. O pool UTXO pode ser armazenado na memória local ou como uma tabela de banco de dados indexada em um armazenamento persistente.

Enquanto a transação e os pools órfãos representam uma perspectiva local de um nodo isolado, e podem variar significativamente de nodo para nodo dependendo de quando o nodo foi iniciado ou reiniciado, a pool de UTXO representa o consenso emergente da rede e logo irá variar pouco entre os nodos. Além disso, a transação e as pools órfãs contém somente transações não-confirmadas, enquanto a pool UTXO contém somente outputs confirmados.

Mensagens de Alerta

As mensagens de alerta são uma função raramente utilizada, mas mesmo assim são implementadas na maioria dos nodos. As mensagens de alerta são um "sistema de alerta de emergências" do bitcoin, um meio através do qual os desenvolvedores do bitcoin core podem enviar uma mensagem de texto de emergência para todos os nodos bitcoin. Essa funcionalidade foi implementada para permitir que a equipe de desenvolvedores do core possa notificar todos os usuários bitcoin de problemas graves na rede bitcoin, como um bug crítico que exija a ação do usuário para ser corrigido. O sistema de alerta só foi utilizado poucas vazes, mais notavelmente no início de 2013, quando um bug crítico de banco de dados causou uma bifurcação de múltiplos blocos na blockchain do bitcoin.

As mensagens de alerta são propagadas pela mensagem alert. A mensagem de alerta contém vários campos, incluindo:

ID

Um identificação do alerta, de maneira que alertas duplicados possam ser detectados

Expiration

Uma hora a partir da qual o alerta expira

RelayUntil

Uma hora após a qual o alerta não deve mais ser transmitido

MinVer, MaxVer

A faixa de versões do protocolo bitcoin a qual esse alerta se aplica

subVer

A versão do software de cliente a qual esse alerta se aplica

Priority

Um nível de prioridade para o alerta, atualmente não sendo utilizado

Os alertas são assinados criptograficamente por uma chave pública. A chave privada correspondente é controlada por alguns membros selecionados do time de desenvolvimento do core. A assinatura digital garante que alertas falsos não sejam propagados na rede.

Cada nodo que receber essa mensagem de alerta irá verificá-la, checar pela expiração e propagá-la para todos os seus pontos, dessa maneira garantindo a rápida propagação através de toda a rede. Além de propagar o alerta, os nodos podem implementar uma função de interface de usuário para apresentar o alerta para o usuário.

No cliente Bitcoin Core, o alerta é configurado com a opção da linha de comando -alertnotify, que especifica um comando para ser executado quando um alerta for recebido. A mensagem de alerta é passada como um parâmetro para o comando alertnotify. Mais comumente, o comando alertnotify é definido para gerar uma mensagem de e-mail para o administrador do nodo, contendo a mensagem de

alerta. O alerta também é exibido como uma caixa de diálogo pop-up na interface gráfica do usuário (bitcoin-Qt), caso ela esteja sendo executada.

Outras implementações do protocolo bitcoin podem manejar o alerta em maneiras diferentes. Muitos sistemas de mineração de bitcoin que usam hardware não implementam a função de mensagem de alerta porque eles não tem uma interface de usuário. É fortemente recomendado que os mineradores que executem esses sistemas de mineração inscrevam-se nesses alertas através de um operador de pool de mineraçõ ou ao executar um nodo de peso leve (lightweight) apenas com o propósito de ter a função do alerta.

A Blockchain

Introdução

A estrutura de dados da blockchain é uma lista ordenada de blocos de transações, com cada bloco sendo ligado ao bloco anterior. A blockchain pode ser armazenada como um arquivo simples ou em um banco de dados simples. O cliente Bitcoin Core armazena os metadados da blockchain usando o banco de dados do LevelDB do Google. Os blocos são interligados de frente para trás, ou seja, cada um se refere ao bloco anterior na corrente. A blockchain é frequentemente visualizada como um empilhamento vertical, com os blocos empilhados uns sobre os outros e com o primeiro bloco servindo como fundação que suporta a pilha. A visualização dos blocos empilhados uns sobre os outros resulta no uso de termos como "altura" para se referir à distância em relação ao primeiro bloco, e "topo" ou "ponta" para se referir ao bloco adicionado mais recentemente.

Cada bloco contido na blockchain é identificado no cabeçalho do bloco por um hash, que é gerado utilizando-se o algoritmo criptográfico de hash SHA256. Cada bloco também contém uma referência ao bloco anterior, conhecido como o bloco *pai*, através do campo "hash do bloco anterior (previous block hash)" que existe no cabeçalho do bloco. Em outras palavras, cada bloco contém o hash de seu bloco-pai no interior de seu próprio cabeçalho. A sequência de hashes ligando cada bloco ao seus pai cria uma corrente que pode ser seguida retrogradamente até o primeiro bloco que já foi criado, que é conhecido como o *bloco gênese*.

Embora um bloco tenha apenas um bloco "pai", ele pode temporariamente ter múltiplos blocos "filhos". Cada um dos blocos "filhos" refere-se ao mesmo bloco "pai" e contém o mesmo hash (o hash do bloco "pai") no campo "hash do bloco anterior". Múltiplos blocos filhos surgem quando há uma "bifurcação" da blockchain, uma situação temporária que ocorre quando diferentes blocos são descobertos quase que simultaneamente por diferentes mineradores (veja [forks]). No final, somente um bloco filho se tornará parte da blockchain e a "bifurcação" deixará de existir. Apesar de cada bloco poder ter mais que um filho, cada bloco pode ter somente um pai. Isso ocorre porque um bloco possui apenas um único campo de "hash do bloco anterior", que é uma referência ao seu bloco pai único.

O campo "hash do bloco anterior" está dentro do cabeçalho do bloco e portanto afeta o hash do bloco *atual*. A identidade de um filho muda se a identidade de seu pai mudar. Quando o pai é modificado de qualquer maneira, o hash do pai muda. O hash modificado do pai exige uma mudança no apontador "hash do bloco anterior" do filho. Isso por sua vez faz com que o hash do filho mude, o que requer uma mudança no apontador do neto, o que por sua vez muda o (hash do) neto, e assim por diante. Esse efeito dominó garante que, assim que um bloco tenha muitas gerações o sucedendo, ele não pode ser modificado sem que haja um novo cálculo forçado de todos os blocos subsequentes. Como um novo cálculo exigiria um processamento computacional enorme, a existência de uma longa corrente de blocos faz com que a história profunda da blockchain seja imutável, o que é uma característica chave para a segurança do bitcoin.

Uma maneira de imaginar a blockchain seria como um solo, onde os blocos seriam camadas de uma formação geológica, ou como uma amostra do núcleo de uma geleira. As camadas da superfície podem

mudar com as estações, ou mesmo serem destruídas antes de terem tempo para se assentarem. Mas quanto mais profundo escavarmos, veremos que maior será a estabilidade das camadas geológicas. Quando você escavar algumas centenas de metros de profundidade, você estará olhando para uma fotografia do passado que permaneceu intocada por milhões de anos. Na blockchain, pode acontecer de os poucos blocos mais recentes tenham que ser revisados/corrigidos, caso haja um novo cálculo da corrente devido a uma bifurcação. Os seis blocos do topo são como os centímetros mais superficiais do solo. Mas quanto mais fundo você penetrar na blockchain, além dos seis blocos, se cada vez menor se torna a probabilidade desses blocos se modificarem. Após 100 blocos de profundidade há tanta estabilidade que a transação coinbase - a transação contendo os bitcoins recém-minerados - pode ser gasta. Alguns milhares de blocos atrás (um mês) e a blockchain é uma história estabelecida, para todos os fins práticos. Apesar de o protocolo sempre permitir que uma corrente seja desfeita por uma corrente mais comprida, e apesar de sempre existir a possibilidade de qualquer bloco ser revertido, a probabilidade de ocorrência de um evento como esse diminui à medida que o tempo passo, até que ela se torne ínfima.

Estrutura de um Bloco

Um bloco é uma estrutura contendo dados que agrega as transações para a inclusão no registro público, a blockchain. O bloco é composto por um cabeçalho, contendo metadados, e por uma longa lista de transações que constituem a maior parte de seu tamanho. O cabeçalho do bloco tem 80 bytes, enquanto uma transação em média tem pelo menos 250 bytes e um bloco em média contém mais de 500 transações. Um bloco completo, com todas as transações, consequentemente é 1.000 vezes maior do que o cabeçalho do bloco. A estrutura de um bloco descreve a estrutura de um bloco.

Table 1. A estrutura de um bloco

Tamanho	Campo	Descrição
4 bytes	Tamanho do Bloco	O tamanho do bloco, em bytes, após esse campo
80 bytes	Cabeçalho do Bloco	Vários campos formam o cabeçalho do bloco
1-9 bytes (VarInt)	Contador de Transações	Quantas transações seguem
Variável	Transações	As transações registradas nesse bloco

Cabeçalho (header) do Bloco

O cabeçalho do bloco consiste de três conjuntos de metadados de bloco. Primeiro, existe uma referência ao hash do bloco anterior, que conecta esse bloco ao bloco anterior na blockchain. O segundo conjunto de metadados, a *dificuldade*, a *data e hora (timestamp)* e o *nonce*, está relacionado à competição da mineração, como serão detalhados no [ch8]. A terceira parte dos metadados é a raiz da árvore de merkle, uma estrutura de dados usada para resumir de maneira eficiente todas as transações contidas no bloco. A estrutura do cabeçalho do bloco descreve a estrutura de um cabeçalho

de bloco.

Table 2. A estrutura do cabeçalho do bloco

Tamanho	Campo	Descrição
4 bytes	Versão	Um número de versão para servir como referência nas atualizações de software/protocolo.
32 bytes	Hash do Bloco Anterior	Uma referência ao hash do bloco anterior (bloco pai) na blockchain
32 bytes	Raiz de Merkle	Um hash da raiz da árvore de merkle das transações desse bloco
4 bytes	Data e Hora (timestamp)	O momento aproximado em que este bloco foi criado (em segundos, usando Unix Epoch)
4 bytes	Dificuldade Alvo	O alvo de dificuldade do algoritmo de prova-de-trabalho deste bloco
4 bytes	Nonce	Um contador usado para o algoritmo de prova-de-trabalho

O nonce, a dificuldade alvo e a data e hora (timestamp) são usados no processo de mineração e serão discutidos em maiores detalhes no [ch8].

Identificadores dos Blocos: Hash do Cabeçalho do Bloco e Altura do Bloco

O identificador primário de um bloco é o seu hash criptográfico, uma impressão digital eletrônica que é criada ao se fazer um duplo hashing do cabeçalho do bloco através do algoritmo SHA256. O hash resultante de 32-bytes é chamado de *hash do bloco*, mas é mais precisamente o *hash do cabeçalho do bloco*, porque apenas o cabeçalho do bloco é usado para computá-lo. Por exemplo, 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f é o hash do bloco do primeiro bloco bitcoin que já foi criado. O hash do bloco identifica o bloco de maneira única e sem ambiguidades, e pode ser independentemente derivado por qualquer nodo que fizer um hashing do cabeçalho do bloco.

Note que o hash do bloco não está incluído no interior da estrutura de dados do bloco, nem quando o bloco é transmitido na rede, nem quando ele é armazenado em um armazenamento persistente de um nodo como parte da blockchain. Ao invés disso, o hash do bloco é processado por cada nodo assim que o bloco é recebido vindo da rede. O hash do bloco pode ser armazenado em uma tabela de banco de

dados separada como parte dos metadados do bloco, para facilitar a indexação e uma coleta mais rápida de blocos a partir do disco.

Uma segunda maneira de se identificar um bloco é através de sua posição na blockchain, que é chamada de altura do bloco. O primeiro bloco criado está na altura de bloco 0 (zero) e é o mesmo bloco que foi referência anteriomente ao seguinte hash de bloco 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f. Logo, um bloco pode ser identificado de duas maneiras: usando as referências do hash do bloco ou da altura do bloco. Cada bloco que é adicionado a seguir "em cima" daquele bloco inicial está uma posição "mais alta" na blockchain, como se fossem caixas empilhadas uma sobre as outras. A altura do bloco em 1º de Janeiro de 2014 era aproximadamente 278.000, significando que havia 278.000 blocos empilhados sobre o primeiro bloco que foi criado em Janeiro de 2009.

Ao contrário do hash do bloco, a altura do bloco não é um identificador único. EMbora um bloco individual sempre terá uma altura de bloco específica e fixa, o inverso nem sempre é verdade - a altura do bloco nem sempre identifica um bloco individual. Dois ou mais blocos podem ter a mesma altura de bloco, competindo pela mesma posição na blockchain. Esse cenário é discutido em detalhes na seção [forks]. A altura do bloco também não faz parte da estrutura de dados do bloco; ela não é armazenada no interior do bloco. Cada nodo identifica dinamicamente uma posição do bloco (altura) na blockchain quando ele é recebido da rede bitcoin. A altura do bloco pode também ser armazenada como metadados em uma tabela de banco de dados indexada para uma leitura mais rápida.

TIP	Um *hash do bloco* de um determinado bloco sempre identifica um bloco único individualmente. Um bloco também sempre tem uma *altura do bloco* específica. Entretanto, nem sempre uma altura de bloco específica pode identificar um bloco único. Na verdade, dois ou mais blocos podem competir por uma posição única na blockchain.

O Bloco Gênese

O primeiro bloco na blockchain é chamado de bloco gênese e foi criado em 2009. Ele é o ancestral comum de todos os blocos na blockchain, significando que se você iniciar em qualquer bloco e seguir a cadeia retrogradamente no tempo, você irá atigir o bloco gênese no final.

Cada nodo sempre começa com uma blockchain de pelo menos um bloco porque o bloco gênese está estaticamente codificado no interior do software do cliente bitcoin, de maneira que ele não pode ser alterado. Cada nodo sempre "conhece" o hash do bloco gênese e sua estruura, o tempo exato em que ele foi criado e até mesmo a transação única que ele contém. Logo, cada nodo possui o ponto inicial da blockchain, uma "raiz" segura a partir da qual ele pode construir uma blockchain de confiança.

Veja o bloco gênese codificado estaticamente no interior do cliente Bitcoin Core, em chainparams.cpp.

O seguinte hash identificador pertence ao bloco gênese:

```
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Você pode procurar por esse hash do bloco em qualquer site explorador de blocos, como o blockchain.info, e você irá encontrar uma página que descreverá o conteúdo desse bloco, com uma URL contendo o hash:

https://blockchain.info/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

https://blockexplorer.com/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

Usando o cliente de referência Bitcoin Core na linha de comando:

```
$ bitcoind getblock 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

```
{
    "hash" : "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
    "confirmations" : 308321,
    "size" : 285,
    "height" : 0,
    "version" : 1,
    "merkleroot" : "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
    "tx" : [
        "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
    ],
    "time" : 1231006505,
    "nonce" : 2083236893,
    "bits" : "1d00ffff",
    "difficulty" : 1.00000000,
    "nextblockhash" : "00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"
}
```

O bloco gênese contém uma mensagem escondida em seu interior. O input da transação coinbase contém o texto "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks.", que poderia ser traduzida como "The Times 03/Jan/2009 Chanceler está prestes a realizar um segundo resgate dos bancos." Essa mensagem tinha a intenção de oferecer prova da data mais precoce em que esse bloco foi criado, ao usar como referência a manchete de um jornal Britânico, o *The Times*. Ela também serve como um lembrete da importância de um sistema monetário independente, com o lançamento do bitcoin ocorrendo ao mesmo tempo em que ocorria uma crise monetária internacional sem precedentes. A mensagem foi embutida no primeiro bloco por Satoshi Nakamoto, o criador do bitcoin.

Conectando os Blocos na Blockchain

Os nodos bitcoin completos mantém uma cópia local da blockchain, iniciando pelo bloco gênese. A cópia local da blockchain é constantemente atualizada à medida que novos blocos são encontrados e são usados para estender a corrente. Assim que um nodo recebe os blocos vindo da rede, ele irá validar

esses blocos e então ligá-los à blockchain existente. Para estabelecer essa ligação, o nodo irá examinar o cabeçalho do bloco vindo da rede e procurar pelo "hash do bloco anterior".

Vamos assumir, por exemplo, que o nodo tem 277.314 blocos na cópia local da blockchain. O último bloco que o nodo conhece é o bloco 277.314, com um cabeçalho do bloco contendo o hash 00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249.

O nodo bitcoin então recebe um novo bloco da rede, que então o analisa da seguinte maneira:

```
{
    "size" : 43560,
    "version" : 2,
    "previousblockhash" :
        "00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
    "merkleroot" :
        "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
    "time" : 1388185038,
    "difficulty" : 1180923195.25802612,
    "nonce" : 4215469401,
    "tx" : [
        "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",

 #[... muitas outras transações omitidas ...]

        "05cfd38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
    ]
}
```

Olhando para esse novo bloco, o nodo encontra o campo previousblockhash, que contém o hash de seu bloco pai. Ele é um hash conhecido ao nodo, aquele do último bloco na corrente, que está na altura 277.314. Logo, esse novo bloco é um bloco filho do último bloco na corrente e estende a blockchain existente. O nodo adiciona esse novo bloco ao fim da corrente, aumentando o comprimento da blockchain, que agora tem uma nova altura de 277.315. Blocos ligados em uma cadeia, pela referência ao hash do cabeçalho do bloco anterior mostra a corrente de três blocos, ligados através das referências contidas no campo previousblockhash.

Árvores de Merkle

Cada bloco na blockchain do bitcoin contém um resumo de todas as transações no bloco, usando uma *árvore de merkle*.

Uma *árvore de merkle*, também conhecida como uma árvore de hash binário, é uma estrutura de dados usadas para resumir eficientemente e verificar a integridade da grandes conjuntos de dados. As árvores de merkle são árvores binárias contendo hashes criptográficos. O termo "árvore" é usado na ciência da computação para descrever uma estrutura de dados ramificada, mas essas árvores

geralmente são exibidas de cabeça para baixo com a "raiz" no topo e com as "folhas" na porção inferior de um diagrama, como você verá nos exemplos a seguir.

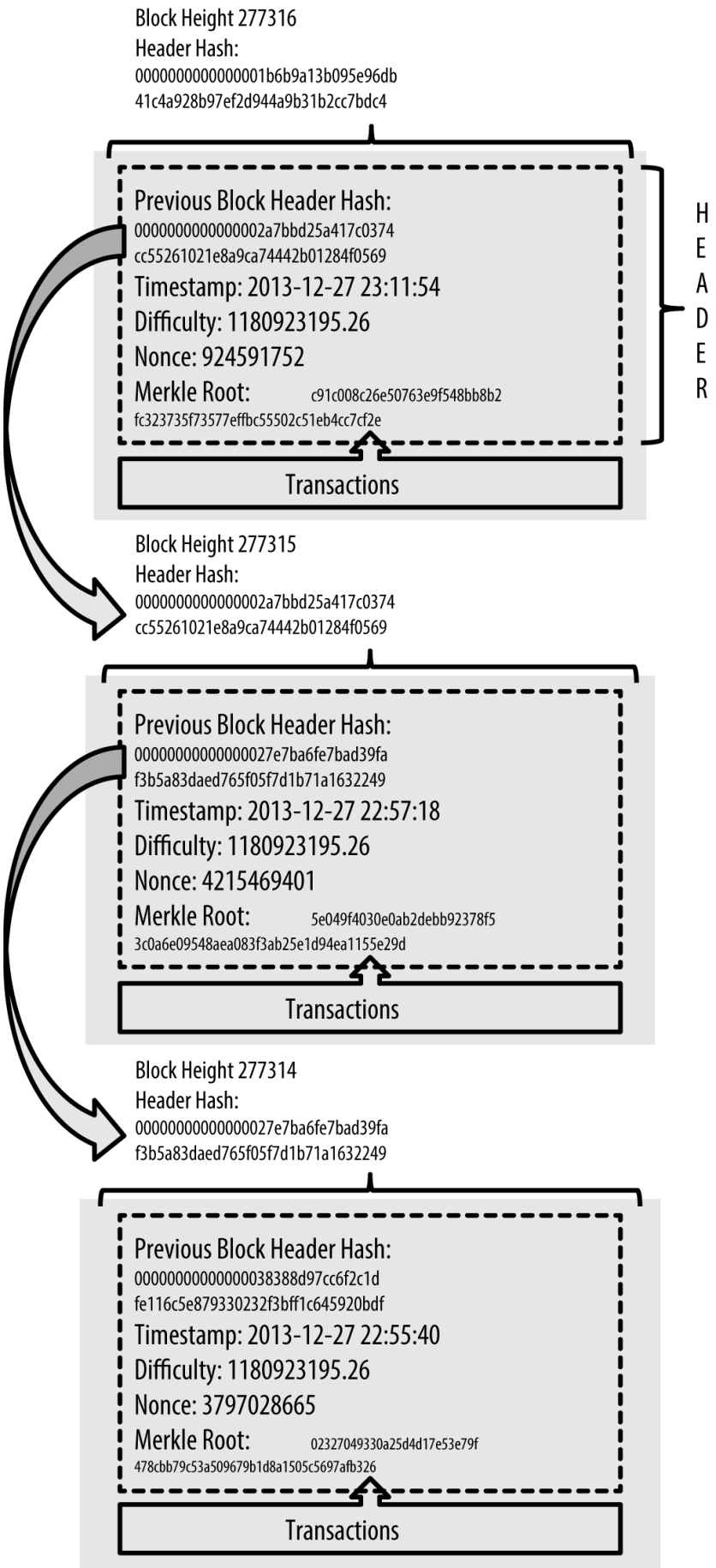

Block Height 277316
Header Hash:
0000000000000001b6b9a13b095e96db
41c4a928b97ef2d944a9b31b2cc7bdc4
Previous Block Header Hash:
0000000000000002a7bbd25a417c0374
cc55261021e8a9ca74442b01284f0569
Timestamp: 2013-12-27 23:11:54
Difficulty: 1180923195.26
Nonce: 924591752
Merkle Root: c91c008c26e50763e9f548bb8b2
fc323735f73577effbc55502c51eb4cc7cf2e
Transactions
HEADER
Block Height 277315
Header Hash:
0000000000000002a7bbd25a417c0374
cc55261021e8a9ca74442b01284f0569
Previous Block Header Hash:
00000000000000027e7ba6fe7bad39fa
f3b5a83daed765f05f7d1b71a1632249
Timestamp: 2013-12-27 22:57:18
Difficulty: 1180923195.26
Nonce: 4215469401
Merkle Root: 5e049f4030e0ab2debb92378f5
3c0a6e09548aea083f3ab25e1d94ea1155e29d
Transactions
Block Height 277314
Header Hash:
00000000000000027e7ba6fe7bad39fa
f3b5a83daed765f05f7d1b71a1632249
Previous Block Header Hash:
0000000000000000038388d97cc6f2c1d
fe116c5e879330232f3bff1c645920bdf
Timestamp: 2013-12-27 22:55:40
Difficulty: 1180923195.26
Nonce: 3797028665
Merkle Root: 02327049330a25d4d17e53e79f
478cbb79c53a509679b1d8a1505c5697afb326
Transactions

Figure 1. Blocos ligados em uma cadeia, pela referência ao hash do cabeçalho do bloco anterior

As árvores de Merkle são usadas no bitcoin para resumir todas as transações em um bloco, produzindo uma impressão digital eletrônica geral de todo o conjunto de transações, fornecendo um processo muito efeiciente para verificar se uma transação foi incluída em um bloco. Uma árvore de Merkle é construída através do hashing recursivo de pares de nodos até que haja apenas um hash, conhecido como a *raiz* ou *raiz de merkle*. O algoritmo de hash criptográfico usado nas árvores de merkle do bitcoin é o SHA256 aplicado duas vezes, também conhecido como SHA256-duplo.

Quando N elementos de dados sofrem hashing e são resumidos em uma árvore merkle, você pode verificar para ver se qualquer elemento de dados foi incluído na árvore com no máximo 2*log~2~(N) cálculos, o que demonstra que a árvore merkle é uma estrutura de dados muito eficiente.

A árvore de merkle é construída de baixo para cima. No exemplo a seguir, nós iniciamos com quatro transações, A, B, C e D, que formam as *folhas* da árvore de Merkle, como demonstrado em Calculando os nodos em uma árvore de merkle. As transações não são armazenadas na árvore de merkle; ao invés disso, seus dados são "hashed" e o hash resultante é armazenado em cada nodo folha como H_A, H_B, H_C e H_D:

```
H~A~ = SHA256(SHA256(Transação A))
```

Os pares consecutivos de nodos folhas são então resumidos em um nodo pai, ao se concatenar dois hashes e fazendo um hashing dos dois juntos. Por exemplo, para construir um nodo pai H_{AB}, os dois hashes de 32-bytes dos filhos são concatenados para criar uma string de 64-bytes. Essa string sofre então um duplo hashing para produzir o hash do nodo pai:

```
H~AB~ = SHA256(SHA256(H~A~ + H~B~))
```

O processo continua até que haja apenas um nodo no topo, o nodo conhecideo como a raiz Merkle. Esse hash de 32-bytes é armazenado no cabeçalho do bloco e resume todos os dados em todas as quatro transações.

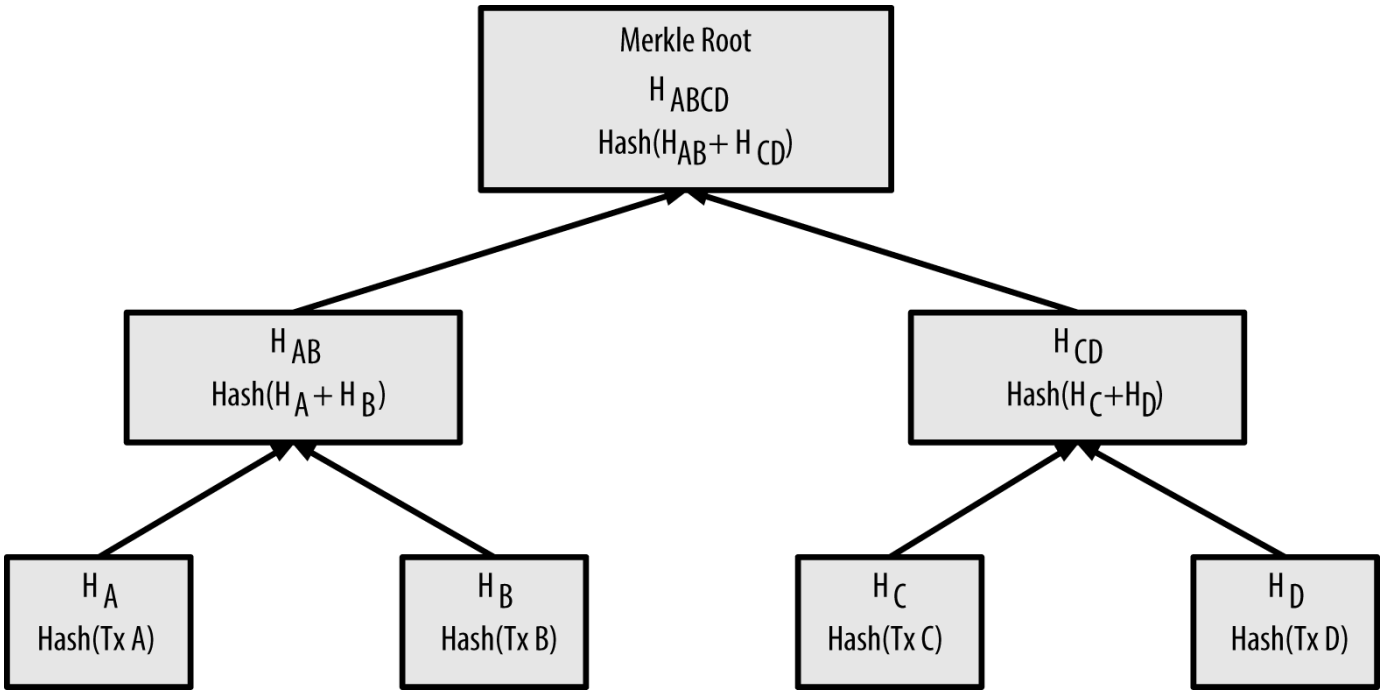


Figure 2. Calculando os nodos em uma árvore de merkle

Como a árvore de merkle é uma árvore binária, ela precisa de um número par de nodos folha. Se existir um número ímpar de transações para ser resumido, o hash da última transação será duplicado para criar uma *árvore equilibrada*, ou seja, uma árvore que contém um número par de nodos folhas. Isso é demonstrado em A duplicação de um elemento de dados gera um número par de elementos de dados, onde a transação C é duplicada.

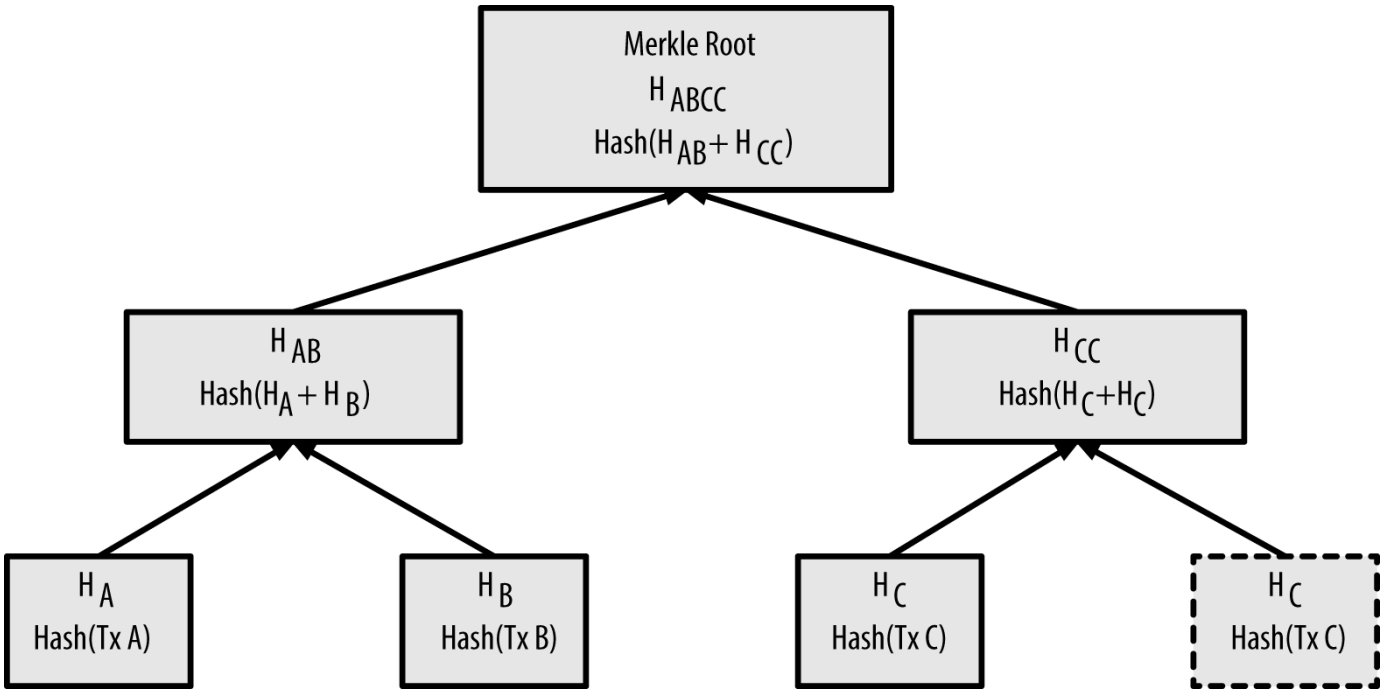


Figure 3. A duplicação de um elemento de dados gera um número par de elementos de dados

O mesmo método para construir uma árvore a partir de quatro transações pode ser generalizado para construir árvores de qualquer tamanho. No bitcoin é comum que haja várias centenas a mais de

milhares de transações em um único bloco, que podem ser resumidas exatamente da mesma maneira, ao produzir apenas 32 bytes de dados como uma árvore de merkle única. Em Uma árvore de merkle resumindo muitos elementos de dados, você verá uma árvore construída a partir de 16 transações. Note que embora a raiz aparente ser maior do que os nodos-folha no diagrama, ela tem exatamente o mesmo tamanho, apenas 32 bytes. Independente de um bloco ter apenas uma ou centenas de milhares de transações, a raiz de merkle sempre as resume em 32 bytes.

Para provar que uma transação específica está incluída em um bloco, um nodo precisa apenas produzir log~2~(N) hashes de 32 bytes, constituindo um *caminho de autenticação* ou *caminho merkle* conectando a transação específica à raiz da árvore. Isso é especialmente importante à medida que o número de transações cresce, porque o logaritmo de base 2 do número de transações aumenta muito mais lentamente. Isso permite que os nodos de bitcoin produzam de maneira eficiente caminhos de 10 ou 12 hashes (320-384 bytes), que podem fornecer uma prova de uma transação individual em um meio de milhares de transações contidas em um bloco com tamanho de megabytes.

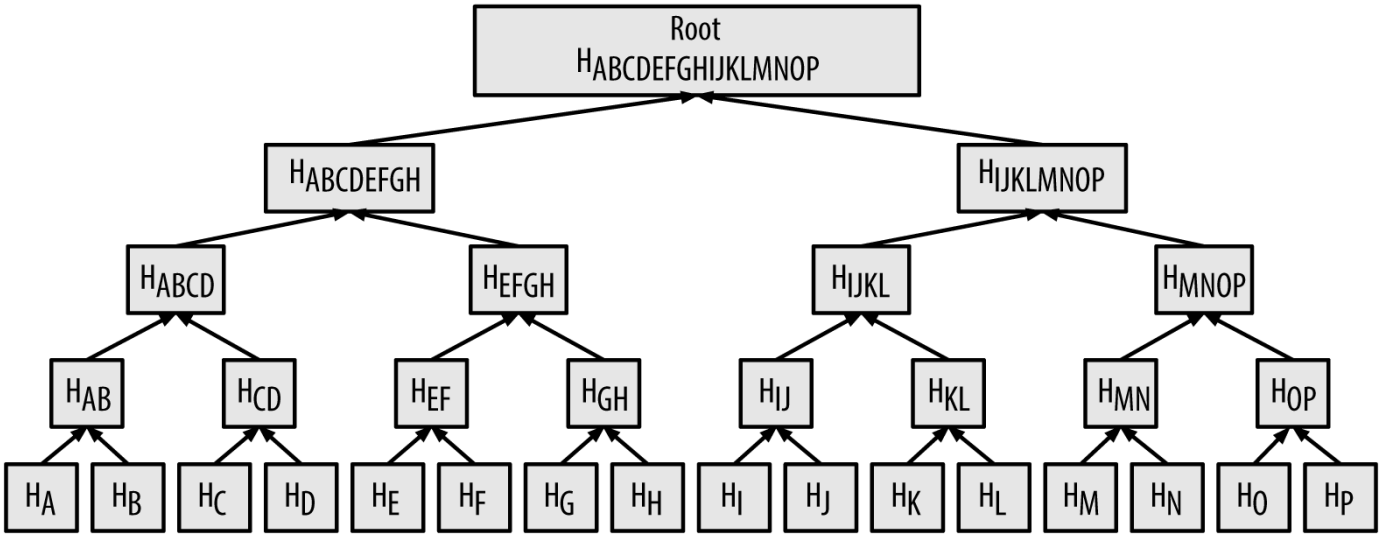


Figure 4. Uma árvore de merkle resumindo muitos elementos de dados

Em Um trajeto de merkle usado para provar a inclusão de um elemento de dados, um nodo pode provar que uma transação K é incluída em um bloco ao produzir um caminho merkle que tenha apenas quatro hashes de 32-bytes (128 bytes no total). O caminho consiste em quatro hashes (identificados em azul em Um trajeto de merkle usado para provar a inclusão de um elemento de dados) H_L, H_{IJ}, H_{MNOP} e $H_{ABCDEFGH}$. Com esses quatro hashes fornecidos como um caminho de autenticação, qualquer nodo pode provar que H_K (identificado em verde no diagrama) está incluso na raiz de merkle ao computar quatro hashes pair-wise H_{KL}, H_{IJKL}, $H_{IJKLMNOP}$, e a raiz da árvore de merkle (contornada com uma linha pontilhada no diagrama).

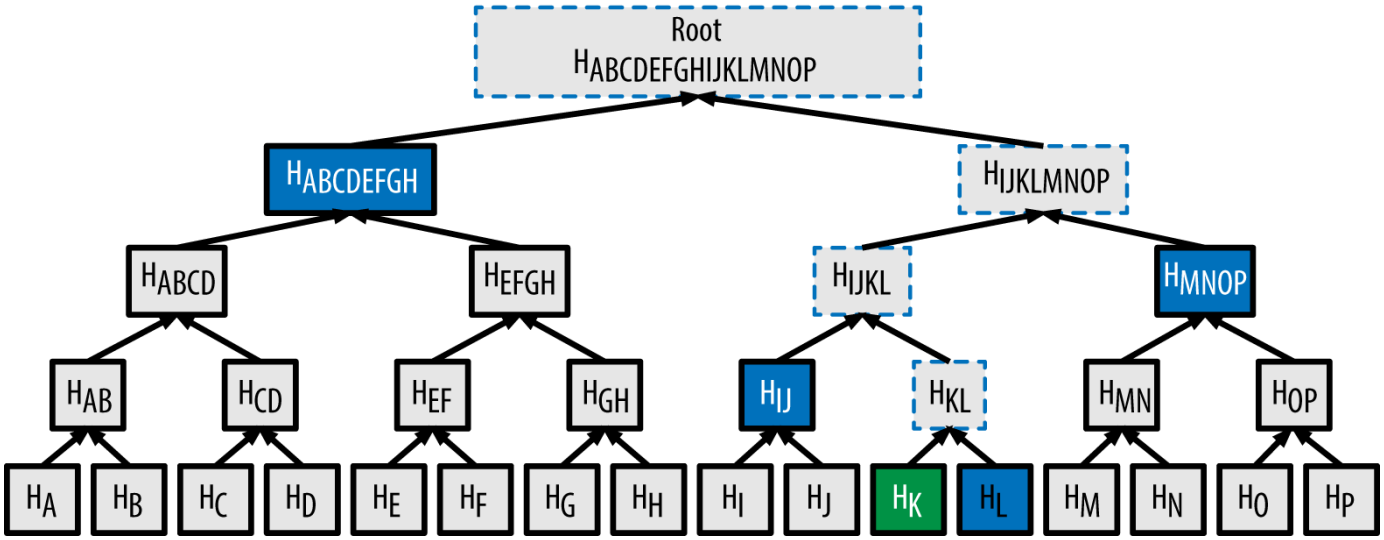


Figure 5. Um trajeto de merkle usado para provar a inclusão de um elemento de dados

O código em Construindo uma árvore de Merkle demonstra o processo de criação de uma árvore de merkle desde os hashes dos nodos folha até a raiz, usando a livraria libbitcoin para algumas funções auxiliares.

Example 1. Construindo uma árvore de Merkle

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

bc::hash_digest create_merkle(bc::hash_list& merkle)
{
    // Stop if hash list is empty.
    if (merkle.empty())
        return bc::null_hash;
    else if (merkle.size() == 1)
        return merkle[0];

    // While there is more than 1 hash in the list, keep looping...
    while (merkle.size() > 1)
    {
        // If number of hashes is odd, duplicate last hash in the list.
        if (merkle.size() % 2 != 0)
            merkle.push_back(merkle.back());
        // List size is now even.
        assert(merkle.size() % 2 == 0);

        // New hash list.
        bc::hash_list new_merkle;
        // Loop through hashes 2 at a time.
        for (auto it = merkle.begin(); it != merkle.end(); it += 2)
        {
            // Join both current hashes together (concatenate).
```

```
            bc::data_chunk concat_data(bc::hash_size * 2);
            auto concat = bc::make_serializer(concat_data.begin());
            concat.write_hash(*it);
            concat.write_hash(*(it + 1));
            assert(concat.iterator() == concat_data.end());
            // Hash both of the hashes.
            bc::hash_digest new_root = bc::bitcoin_hash(concat_data);
            // Add this to the new list.
            new_merkle.push_back(new_root);
        }
        // This is the new list.
        merkle = new_merkle;

        // DEBUG output ------------------------------------
        std::cout << "Current merkle hash list:" << std::endl;
        for (const auto& hash: merkle)
            std::cout << "  " << bc::encode_hex(hash) << std::endl;
        std::cout << std::endl;
        // -------------------------------------------------
    }
    // Finally we end up with a single item.
    return merkle[0];
}

int main()
{
    // Replace these hashes with ones from a block to reproduce the same merkle root.
    bc::hash_list tx_hashes{{

bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000000"),

bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000011"),

bc::hash_literal("0000000000000000000000000000000000000000000000000000000000000022"),
    }};
    const bc::hash_digest merkle_root = create_merkle(tx_hashes);
    std::cout << "Result: " << bc::encode_hex(merkle_root) << std::endl;
    return 0;
}
```

Compilando e executando o código de exemplo merkle mostra o resultado da compilação e execução do código de merkle

Example 2. Compilando e executando o código de exemplo merkle

```
$ # Compilar o código merkle.cpp
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Rodar o executável merkle
$ ./merkle
Hash list de merkle atual:
  32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27006
  30861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7540c4

Hash list de merkle atual:
  d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

Resultado: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3
```

A eficiência da árvore de merkle se torna óbvia na medida que ela cresce em escala. Eficiência da árvore de Merkle mostra a quantidade de dados que precisa ser trocada como um caminho de merkle para provar que uma transação faz parte de um bloco.

Table 3. Eficiência da árvore de Merkle

Número de transações	Tamanho aprox. do bloco	Tamanho do caminho (hashes)	Tamanho do caminho (bytes)
16 transações	4 kilobytes	4 hashes	128 bytes
512 transações	128 kilobytes	9 hashes	288 bytes
2.048 transações	512 kilobytes	11 hashes	352 bytes
65.535 transações	16 megabytes	16 hashes	512 bytes

Como você pode ver na tabela, enquanto o tamanho do bloco aumenta rapidamente, de 4 KB com 16 transações até um tamanho de bloco de 16 MB, para comportar 65.535 transações, o caminho merkle necessário para provar a inclusão de uma transação cresce muito mais lentamente, de 128 bytes para somente 512 bytes. Com árvores merkle, um nodo pode fazer apenas o download dos cabeçalhos dos blocos (80 bytes por bloco) e ainda ser capaz de identificar a inclusão de uma transação em um bloco ao adquirir um pequeno caminho merkle a partir de um nodo completo, sem armazenar ou transmitir a vasta maioria da blockchain, que pode ter vários gigabytes de tamanho. Os nodos que não mantém uma blockchain completa, chamados de nodos de verificação simplificada de pagamento (nodos VSP), usam caminhos merkle para verificar as transações sem ter que fazer o download dos blocos completos.

Árvores de Merkle e Verificação Simplificada de Pagamento (VSP)

As árvores de Merkle são muito usadas por nodos VSP. Os nodos VSP não tem todas as transações e não fazem o download de todos os blocos, fazem apenas dos cabeçalhos dos blocos. Para verificar que uma transação foi incluída em um bloco, sem ter que fazer download de todas as transações no bloco, eles usam um caminho de autenticação, ou um caminho merkle.

Considere, por exemplo, que um nodo de VSP esteja interessado em receber pagamentos para um endereço contido em sua carteira. O nodo VSP irá estabelecer um filtro bloom em suas conexões aos pontos para limitar as transações para somente aquelas que contenham os endereços de interesse. Quando um ponto vê uma transação que corresponde ao filtro bloom, ele irá enviar para aquele bloco uma mensagem merkleblock. A mensagem merkleblock contém o cabeçalho do bloco assim como o caminho merkle que liga a transação de interesse à raiz merkle no bloco. O nodo VSP pode usar esse caminho merkle para conectar a transação ao bloco e verificar que a transação foi incluída no bloco. O nodo VSP também usa o cabeçalho do bloco para ligar o bloco ao resto da blockchain. A combinação dessas duas ligações, entre a transação e o bloco, e entre o bloco e a blockchain, prova que a transação foi registrada na blockchain. No final das contes, o nodo VSP terá recebido menos que um kilobyte de dados que é mais do que mil vezes menos do que um bloco completo (cerca de 1 megabyte atualmente).

Mineração e Consenso

Introdução

Mineração é o processo através do qual um novo bitcoin é adicionado à oferta de dinheiro. A mineração também serve para proteger o sistema bitcoin contra transações fraudulentas ou transações gastando a mesma quantia de bitcoin mais de uma vez, o que é conhecido como gasto duplo. Os mineradores fornecem poder de processamento à rede bitcoin em troca da oportunidade de serem recompensados em bitcoin.

Mineradores validam novas transações e as gravam no Livro Razão global. Um novo bloco, contendo transações que ocorreram desde o último bloco é "minerado" em média a cada 10 minutos, assim adicionando estas transações a Cadeia de Blocos. Transações que se tornam parte de um bloco e são adicionadas a Cadeia de Blocos são consideradas "confirmadas", o que permite os novos proprietários de bitcoin a gastarem os bitcoins recentemente recebidos nestas transações.

Mineradores recebem dois tipos de recompensa pela mineração: novas moedas criadas a cada novo bloco, e taxas de transação em todas as transações incluídas no bloco. Para ganhar essa recompensa, os mineradores competem entre si para resolver um complexo problema matemático baseado em um algoritmo de hash criptográfico. A solução para o problema, chamada de proof-of-work (prova de trabalho), é incluída no novo bloco e serve como prova de que o minerador gastou uma significativa quantidade de esforço computacional. A competição para resolver o algoritmo de proof-of-work a fim de obter as recompensas e o direito de registrar transações no blockchain é a base do modelo de segurança da rede bitcoin.

O processo de geração de novas moedas é chamado de mineração porque a recompensa é projetada para simular retornos cada vez menores, da mesma maneira que a mineração de metais preciosos. A oferta monetária do bitcoin é criada através da mineração, semelhante à maneira como um banco central emite dinheiro através da impressão de cédulas (notas de dinheiro). A quantia de bitcoin recém-criado que um minerador pode acrescentar em cada bloco diminui a cada 4 anos em média (ou precisamente a cada 210.000 blocos). No início do bitcoin, em janeiro de 2009, 50 bitcoins eram gerados a cada bloco, e esse número caiu pela metade (25 bitcoins por bloco) em novembro de 2012. Ele irá cair pela metade (12,5 bitcoins por bloco) no ano 2016. Como é baseada nessa fórmula, a remuneração da mineração do bitcoin diminuirá exponencialmente até aproximadamente o ano de 2140, quando todos os bitcoins (20,99999998 milhões) terão sido emitidos. Após 2140, mais nenhum bitcoin será criado.

Os mineradores de bitcoin também ganham taxas das transações. Cada transação pode incluir uma taxa de transação, na forma de um excedente de bitcoins entre as entradas (inputs) e saídas (outputs) da transação. O minerador de bitcoin que vencer irá receber as taxas das transações que estão incluídas no bloco que ele minerou. Atualmente, as taxas representam 0,5% ou menos da renda de um minerador de bitcoin, ou seja, a maior parte da renda vem dos bitcoins recém-minerados. No entanto, à medida que a recompensa diminui ao longo do tempo e o número de transações por bloco aumenta, uma proporção maior da renda dos mineradores virá das taxas das transações. Após 2140, tudo o que os mineradores irão ganhar virá na forma de taxas de transações.

A palavra "mineração" pode causar confusão. Pela analogia com a extração se metais preciosos, ela foca nossa atenção na recompensa pela mineração, ou seja, os novos bitcoins gerados em cada bloco. Embora a mineração seja incentivada por essa recompensa, a proposta primária da mineração não é a recompensa ou a geração de novas moedas. Se você ver mineração somente como o processo através do qual as moedas são criadas, você está confundindo os meios (incentivos) como um objetivo do processo. Mineração é o processo principal da câmara de compensação descentralizada, através do qual as transações são apuradas e validadas. A mineração garante a segurança do sistema bitcoin e permite a existência de um consenso em toda a rede sem a necessidade de uma autoridade central.

Mineração é a invenção que torna o bitcoin especial, é um mecanismo de segurança descentralizado que é a base para o dinheiro digital ponto-a-ponto. A recompensa por moedas recém-mineradas e as taxas de transações são um esquema de incentivo que alinha as ações dos mineradores com a segurança da rede, enquanto simultaneamente implementa a oferta monetária.

Nesse capítulo, nós iremos primeiramente examinar a mineração como um mecanismo de oferta monetária e então olharemos para a mais importante função da mineração: o mecanismo de consenso emergergente descentralizado subjacente da segurança do bitcoin.

A Economia do Bitcoin e a Criação de Moedas

Os bitcoins são minerados durante a criação de cada bloco em uma taxa fixa e que só diminui. Cada bloco, gerado a cada 10 minutos em média, contém bitcoins completamente novos, que foram criados do nada. A cada 210.000 blocos, ou aproximadamente a cada 4 anos, a taxa de emissão da moeda é diminuída em 50%. Nos primeiros 4 anos de operação da rede, cada bloco continha 50 bitcoins novos.

Em novembro de 2012, a taxa de emissão de bitcoins foi diminuída para 25 bitcoins/bloco, e ela irá reduzir novamente para 12,5 bitcoins/bloco no bloco 420.000, que só será minerado aproximadamente em 2016. A taxa de novas moedas diminui exponencialmente em 64 "divisões por dois" até o bloco 13.230.000 (que será minerado aproximadamente no ano 2137), quando ela atingirá a unidade mínima de 1 satoshi. Finalmente, após 13,44 milhões de blocos, aproximadamente no ano de 2140, quase 2.099.999.997.690.000 de satoshis, ou quase 21 milhões de bitcoins, terão sido emitidos. Depois disso, os blocos não irão conter novos bitcoins, e os mineradores serão recompensados apenas com as taxas de transações. Oferta de moedas bitcoin ao longo do tempo, que é baseada em uma taxa de emissão com decréscimo geométrico demonstra o número total de bitcoins em circulação ao longo do tempo, na medida que a emissão da moeda diminui.

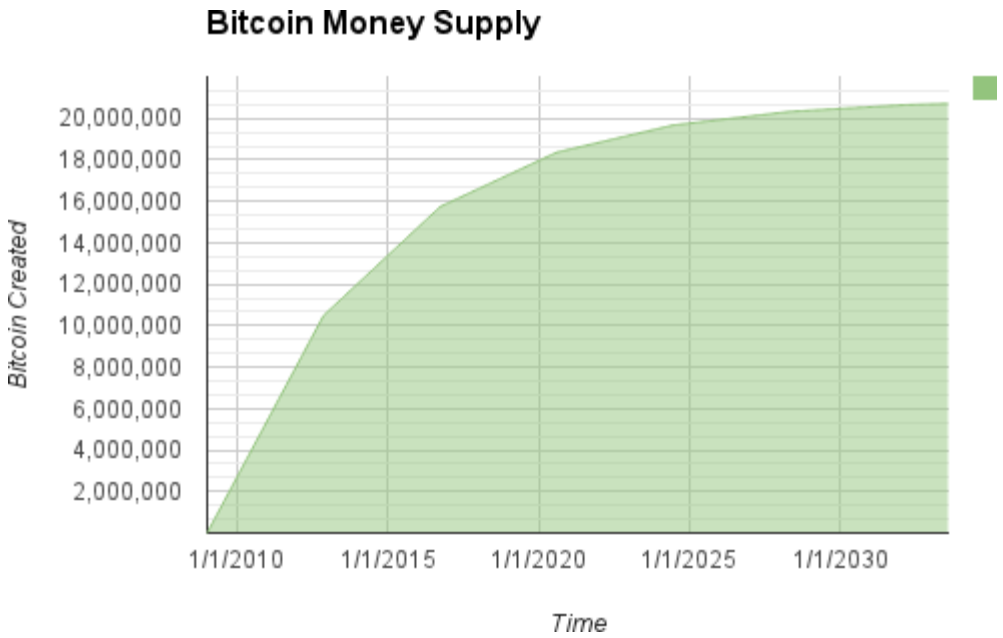


Figure 1. Oferta de moedas bitcoin ao longo do tempo, que é baseada em uma taxa de emissão com decréscimo geométrico

NOTE	O número máximo de "coins" mineradas é o *limite superior* das recompensas de mineração possíveis do bitcoin. Na prática, um minerador pode intencionalmente minerar um bloco recebendo menos do que a recompensa completa. Tais blocos já foram minerados e mais podem ser minerados no futuro, resultando em uma menor emissão total de moedas.

No código de exemplo em Um script para calcular quanto de bitcoins totais serão emitidos, nós calculamos a quantidade total de bitcoins que será emitida.

Example 1. Um script para calcular quanto de bitcoins totais serão emitidos

```
# Original block reward for miners was 50 BTC
start_block_reward = 50
# 210000 is around every 4 years with a 10 minute block interval
reward_interval = 210000

def max_money():
    # 50 BTC = 50 0000 0000 Satoshis
    current_reward = 50 * 10**8
    total = 0
    while current_reward > 0:
        total += reward_interval * current_reward
        current_reward /= 2
    return total

print "Total BTC to ever be created:", max_money(), "Satoshis"
```

Executando o script max_money.py mostra o output produzido ao executar-se esse script.

Example 2. Executando o script max_money.py

```
$ python max_money.py
Total de BTCs que serão criados: 2099999997690000 Satoshis
```

A emissão decrescente e finita cria uma oferta monetária fixa que resiste à inflação. Ao contrário das moedas fiduciárias, que podem ser impressas em números infitos por um banco central, o bitcoin jamais sofrerá inflação causada por impressão de moedas.

Dinheiro deflacionário

A consequência mais importante e debatida de uma emissão monetária fixa e descrescente é que a moeda tenderá a ser inerentemente *deflacionária*. A deflação é o fenômeno da perda de valor devido a um defasamento entre a oferta e a demanda que aumenta o valor (e a taxa de câmbio) de uma moeda. A deflação dos preços, o contrário da inflação, significa que o dinheiro tem mais poder de compra com o passar do tempo.

Muitos economistas argumentam que uma economia deflacionária é um desastre e deveria ser evitada a todo custo. Isso porque em um período de rápida deflação, as pessoas tendem a guardar dinheiro ao invés de gastá-lo, na esperança de que os preços irão cair. Tal fenômeno ocorrer durante a "Década Perdida" do Japão, quando um colapso completo da demanda levou a moeda para um espiral deflacionário.

Os especialistas em Bitcoin argumentam que a deflação não é algo ruim por si só. Ao invés disso, os economistas associam a deflação a um colapso na demanda porque este foi o único exemplo de deflação que nós temos disponível para estudar. Em uma moeda fiduciária com a possibilidade de impressão ilimitada, é muito difícil de se entrar em uma espiral deflacionária a menos que exista um colapso completo na demanda e um desinteresse em imprimir dinheiro. A deflação do bitcoin não é causada por um colapso na demanda, mas por uma oferta restrita previsível.

Na prática, se tornou evidente que o instinto de acumular causado pela moeda deflacionária pode ser superado através de descontos dos vendedores, até que o desconto supere o instinto de acumular do comprador. Como o vendedor também está motivado a acumular, o desconto se torna o preço de equilíbrio no qual os dois instintos de acumular se correspondem. Com descontos de 30% no preço do bitcoin, a maior parte dos comerciantes de bitcoin não estão tendo dificuldades em superar o instinto de acumular e estão gerando renda. Ainda não se sabe se o aspecto deflacionário da moeda é realmente um problema quando ele não é conduzido por uma rápida retração econômica.

Consenso Descentralizado

No capítulo anterior, nós aprendemos sobre a blockchain, o registro público (lista) global de todas as transações, que todos participantes da rede bitcoin aceitam como um registro oficial das posses dos bitcoins.

Mas como é possível que todas as pessoas na rede concordem em uma "verdade" única universal sobre quem é dono do que, sem ter que confiar em alguém? Todos os sistemas de pagamento tradicionais dependem de um modelo de confiança que possui uma autoridade central fornecendo um serviço de câmara de compensação, basicamente verificando e compensando todas as transações. O Bitcoin não possui uma autoridade central no entanto de certa forma todos os nodos completos possuem uma cópia completo do registro público, no qual eles podem confiar como um registro confiável. A blockchain não é criada por uma autoridade central, mas é criada independentemente por cada nodo

da rede. De alguma maneira, cada nodo na rede, agindo com as informações que são transmitidas através das conexões de rede inseguras, conseguem chegar à mesma conclusão e fabricar o mesmo registro público que todos os outros nodos. Essa capítulo examina o processo através do qual a rede bitcoin atinge um consenso global sem a necessidade de uma autoridade central.

A principal invenção de Satoshi Nakamoto é o mecanismo descentralizado para o *consenso emergente*. Diz-se emergente, pois o consenso não é atingido de maneira explícita — não existe uma eleição ou um momento fixo no qual o consenso ocorre. Ao invés disso, o consenso é um artefato que emerge da interação assíncrona de milhares de nodos independentes, todos seguindo regras simples. Todas as propriedades do bitcoin, incluindo a moeda, as transações, os pagamentos e o modelo de segurança que não depende de confiança ou de uma autoridade central, derivam dessa invenção.

O consenso descentralizado do Bitcoin emerge da interação de quatro processos que ocorrem independentemente nos nodos pela rede:

- Verificação independente de cada transação, realizada por cada nodo completo, que é baseada em uma extensa lista de critérios
- Agregação independente dessas transações em blocos novos pelos nodos mineradores, associado à demonstração de que um processo computacional foi realizado através de um algoritmo de prova-de-trabalho.
- Verificação independente dos novos blocos por cada nodo e inclusão deles em uma cadeia
- Seleção independente, por cada nodo, da corrente de blocos com a maior computação acumulada demonstrada através da prova de trabalho

Nas próximas seções nós iremos examinar os processos e como eles interagem para criar uma propriedade emergente de consenso amplo na rede que permite que qualquer nodo monte a sua própria cópia do registro global, público, confiável e seguro.

Verificação Independente de Transações

Em [transactions], nós vimos como o software de carteira cria transações ao coletar UTXO, fornecendo os scripts de destravamento apropriados e então construindo novos outputs designados para um novo dono. A transação resultando é então enviada para os nodos da rede bitcoin que estão mais próximos, para que possa ser propagada ao longo de toda a rede bitcoin.

Entretanto, antes de propagar as transações para seus nodos vizinhos, cada nodo bitcoin que recebe uma transação irá primeiro verificá-la. Isso garante que somente transações válidas serão propagadas para a rede, enquanto as transações inválidas serão logo descartadas pelo primeiro nodo que as encontrarem.

Cada nodo verifica todas as transações contra uma lista de verificação (checklist) com vários critérios:

- A sintaxe da transação e a estrutura de dados devem estar corretas.
- Nem a lista de entradas (inputs), nem a de saídas (outputs) estão vazias.

- O tamanho da transação em bytes é menor do que o MAX_BLOCK_SIZE.
- Cada valor de output, assim como o total, deve estar dentro da faixa de valores permitida (menos do que 21 milhões de bitcoins e mais do que 0 bitcoins).
- Nenhum dos inputs tem hash=0, N=–1 (transações coinbase não devem ser transmitidas).
- O nLockTime é menor ou igual ao INT_MAX.
- O tamanho da transação em bytes é maior ou igual a 100.
- O número de operações de assinatura contidas na transação é menor do que o limite de operações de assinatura
- O script de destravamento (scriptSig) só pode empurrar números para o stack, e o script de travamento (scriptPubkey) deve coincidir com as formas isStandard (isso rejeita transações "nonstandard", não-padrões).
- Deve existir uma transação correspondente no pool, ou em um bloco na corrente principal.
- Para cada input, se o output referenciado existir em qualquer outra transação no pool, a transação deve ser rejeitada.
- Para cada input, buscar no ramo principal e no pool de transações pelo output de transação referenciado. Se o output de transação estiver faltando para qualquer input, essea será uma transação órfã. Adiciona ao pool de transações órfãs, se uma transação correspondente já não estiver no pool.
- Para cada input, se o output de transação referenciado for um output coinbase, ele deve ter pelo menos COINBASE_MATURITY (100) confirmações.
- Para cada entrada (input), a saída (output) de referência deve existir e não pode ter sido gasta previamente.
- Ao usar os outputs de transações referenciados para adquirir os valores de input, verificar que cada valor de input, assim como a soma, estão na faixa permitida de valores (menos que 21 milhões de bitcoin, maior do que 0).
- Rejeitar se a soma dos valores de entrada (input) seja menor do que a soma dos valores de saída (output).
- Rejeitar se a taxa de transação seria muito baixa para entrar em um bloco vazio.
- Os scripts de destravamento para cada input devem ser validados em relação aos scripts de travamento dos outputs correspondentes.

Essas condições podem ser vistas em detalhes nas funções AcceptToMemoryPool, CheckTransaction e CheckInputs no cliente de referência bitcoin. Note que as condições mudam ao longo do tempo, para lidar com novos tipos de ataques de negação de serviço (DOS) ou às vezes para tornar algumas regras menos rígidas, com o objetivo de incluir mais tipos de transação.

Ao verificar cada transação independentemente à medida que ela é recebida e antes de propagá-la, cada nodo constroi um pool de transações válidas (mas ainda não-confirmadas) conhecido como *pool de transações*, *pool de memória* ou *mempool*.

Nodos Mineradores

Alguns dos nodos da rede bitcoin são nodos especializados chamados de *mineradores*. Em [ch01_intro_what_is_bitcoin] nós apresentamos o Jing, um estudante de engenharia da computação em Shangai, na China, que é um minerador bitcoin. O Jing ganha bitcoins por ter um "mining rig," "equipamento de mineração", que é um sistema de hardware computadorizado projetado para minerar bitcoins. O hardware especializado de mienração do Jing é conectado a um servidor executando um nodo bitcoin completo. Ao contrário do Jing, alguns mineradores mineram sem ter um nodo completo, como nós veremos em Pools de Mineração. Assim como todos os outros nodos completos, o nodo de Jing recebe e propaga transações não-confirmadas na rede bitcoin. O nodo do Jing, no entanto, também agrega essas transações em blocos novos.

O nodo do Jing fica monitorando os novos blocos que são propagados na rede bitcoin, assim como todos os nodos fazem. No entanto, a chegada de um novo bloco tem um significado especial para um nodo de mienração. A competição entre os mineradores efetivamente termina com a propagação de um novo bloco, agindo como se fosse o anúncio de um vencedor. Para os mineradores, receber um novo bloco significa que alguma outra pessoa ganhou a competição e que eles perderam. No entanto, o fim de uma rodada de competição também é o início da próxima rodada. O novo bloco não é apenas uma bandeira quadriculada, que marca o final da corrida; ele também é o disparo de largada na corrida para o próximo bloco.

Agregando Transações em Blocos

Após validas as transações, um nodo bitcoin irá adicionálas ao *pool de memória*, ou *pool de transações*, onde as transações aguardam até elas serem incluídas (mineradas) em um bloco. O nodo do Jing coleta, valida e transmite novas transações assim como qualquer outro nodo. Ao contrário dos outros nodos, entretanto, o nodo do Jing irá agregar essas transações em um *bloco candidato*.

Vamos seguir os blocos que foram criados durante o período em que a Alice comprou uma xícara de café do Bob's Cafe (ver [cup_of_coffee]). A transação da Alice foi incluída no bloco 277.316. Para fins de demonstrar os conceitos nesse capítulo, vamos assumir que o bloco foi minerado pelo sistema de mineração do Jing e vamos seguir a transação da Alice à medida que ela é integrada a esse novo bloco.

O nodo de mineração do Jing contém uma cópia local da blockchain, a lista de todos os blocos criados desde o início do sistema bitcoin em 2009. No momento em que a Alice compra a xícara de café, o nodo do Jing montou uma cadeia até o bloco 277.314. O nodo do Jing está monitorando novas transações, tentanto minerar um novo bloco e também monitorando os blocos descobertos pelos outros nodos. À medida que o nodo do Jing minera, ele recebe o bloco 277.315 através da rede bitcoin. A chegada desse bloco significa o fim da competição pelo bloco 277.315 e o início da competição para a criação do bloco 277.316.

Durante os últimos 10 minutos, enquanto o nodo do Jing procurava uma solução para o bloco 277.315, ele também estava coletando transações para se preparar para o próximo bloco. Agora ele já coletou algumas centenas de transações no pool de memória. Ao receber o bloco 277.315 e validando-o, o nodo do Jing também irá verificar todas as transações no pool de memória e removerá qualquer uma que já

tiver sido incluída no bloco 277.315. Quaisquer transações que permanecerem no pool de memória são transações não-confirmadas e estão aguardando para serem registradas em um novo bloco.

O nodo do Jing imediatamente constroi um novo bloco vazia, um candidato para o bloco 277.316. Esse bloco é chamado de bloco candidato pois ele ainda não é um bloco válido, já que ele não contém uma prova de trabalho válida. O bloco só se torna válido se o minerador tiver sucesso em encontrar uma solução para o algoritmo de prova de trabalho.

Idade da transação, Taxas e Prioridade

Para construir um bloco candidato, o nodo de bitcoin do Jing seleciona as transações a partir do pool de memória ao aplicar uma métrica de prioridade para cada transação e ao adicionar primeiro as transações com maior prioridade. As transações são priorizadas de acordo com a "idade" do UTXO que está sendo gasto em suas entradas (inputs), permitindo que inputs antigos e de alto valor recebam maior prioridade que inputs novos e de menores valores. As transações que tem prioridade podem ser enviadas sem nenhuma taxa, se houver espaço suficiente no bloco.

A prioridade de uma transação é calculada como a soma dos valores e da idade dos inputs dividida pelo tamanho total da transação:

```
Prioridade = Soma (Valor da entrada * Idade da entrada) / Tamanho da Transação
```

Nessa equação, o valor de um input é medido na unidade de base, satoshis (1/100m, a centésima milionésima parte de um bitcoin). A idade de um UTXO é o número de blocos que já foram minerados desde que a UTXO foi registrada na blockchain, medindo a quantos blocos de "profundidade" ela está na blockchain. O tamanho da transação é medido em bytes.

Para que uma transação seja considerada de "alta prioridade", sua prioridade deve ser maior do que 57.600.000, o que corresponde a um bitcoin (100 milhões de satoshis), tenha um dia de idade (144 blocos), em uma transação com tamanho total de 250 bytes:

```
Alta Prioridade > 100.000.000 satoshis * 144 blocos / 250 bytes = 57.600.000
```

Os primeiros 50 kilobytes do espaço de uma transação em um bloco são reservados para transações de alta prioridade. O nodo do Jing irá preencher os primeiros 50 kilobytes, dando prioridade para as transações de alta prioridade primeiro, independente das taxas. Isso permite que as transações de alta prioridade sejam processadas mesmo que elas não tenham taxa alguma.

O nodo de mineração do Jing preenche então o resto do bloco até o tamanho máximo de bloco (MAX_BLOCK_SIZE no código), com transações que carregam pelo menos a taxa mínima, priorizando aquelas que tem a maior taxa por kilobyte de transação.

Se houver algum espaço remanescente no bloco, o nodo de mineração do Jing pode escolher preenchê-lo com transações sem taxas. Alguns mineradores decidem minerar as transações sem taxas baseando-

se em um melhor esforço. Outros mineradores podem decidir ignorar as transações sem taxas.

Quaisquer transações que sobrarem no pool de memória, após o bloco ter sido preenchido, permanecerão no pool para inclusão no próximo bloco. À medida que as transações permanecem no pool de memória, a "idade" de suas entradas (inputs) aumenta, pois o UTXO que elas gastam se aprofunda cada vez mais na blockchain que vem recebendo novos blocos acima dele. Como a prioridade da transação depende da idade de suas entradas (inputs), as transações remanescentes na pool irão envelhecer e, portanto, aumentarão de prioridade. Por fim, uma transação pode atingir uma prioridade alta o suficiente para ser incluída grauitamente em um bloco.

As transações Bitcoin não possuem um tempo limite de expiração. Uma transação que é válida agora será válida para sempre. No entanto, se uma transação só for propagada na rede uma única vez, ela irá persistir contanto que ela seja mantida em um pool de memória de um nodo minerador. Quando um nodo minerador é reiniciado, o seu pool de memória é apagado, porque ele é uma forma de armazenamento transitório, não-persistente. Embora uma transação válida possa ter sido propagada pela rede, se ela não for executada ela pode chegar a não residir no pool de memória de nenhum minerador. Espera-se que o software de carteira retransmita tais transações ou as reconstrua com taxas maiores se elas não forem executadas com sucesso em uma quantia de tempo razoável.

Quando o nodo do Jing agrega todas as transações originadas do pool de memória, o novo bloco candidato tem 418 transações com um total de taxa de transações de 0,09094928 bitcoin. Você pode ver esse bloco na blockchain usando a interface em linha de comando do cliente Bitcoin Core, como demonstrado em Bloco 277,316.

```
$ bitcoin-cli getblockhash 277316
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4

$ bitcoin-cli getblock
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

Example 3. Bloco 277,316

```
{
    "hash" : "0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
    "confirmations" : 35561,
    "size" : 218629,
    "height" : 277316,
    "version" : 2,
    "merkleroot" :
"c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e",
    "tx" : [
        "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
        "b268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877c503fbe",

        ... 417 outras transações ...

       ],
    "time" : 1388185914,
    "nonce" : 924591752,
    "bits" : "1903a30c",
    "difficulty" : 1180923195.25802612,
    "chainwork" : "0000000000000000000000000000000000000000000000934695e92aaf53afa1a",
    "previousblockhash" :
"0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569",
    "nextblockhash" :
"000000000000000010236c269dd6ed714dd5db39d36b33959079d78dfd431ba7"
}
```

A Transação de Geração

A primeira transação adicionada ao bloco é uma transação special, chamada de *transação de geração* ou *transação coinbase* Essa transação é construída no nodo de Jing e é a sua recompensa pelos seus esforços de mineração. O nodo de Jing cria a transação de geração como um pagamento para a sua própria carteira: "Page para o endereço de Jing 25.09094928 bitcoin." A quantia total da recompensa que o Jing coleta por minerar um bloco é a soma da recompensa da transação de geração (25 novos bitcoins) com as taxas de transação (0.09094928) de todas as transações incluídas no bloco, como demonstrado em Geração da transação:

```
$ bitcoin-cli getrawtransaction
d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f 1
```

Example 4. Geração da transação


```
{
    "hex" :
"01000000010000000000000000000000000000000000000000000000000000000000000000ffffffff0f
03443b0403858402062f503253482ffffffff0110c08d9500000000232102aa970c592640d19de03ff6f
329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac00000000",
    "txid" : "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "coinbase" : "03443b0403858402062f503253482f",
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 25.09094928,
            "n" : 0,
            "scriptPubKey" : {
                "asm" :
"02aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21OP_CHECKSIG",
                "hex" :
"2102aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac",
                "reqSigs" : 1,
                "type" : "pubkey",
                "addresses" : [
                    "1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N"
                ]
            }
        }
    ],
    "blockhash" : "0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
    "confirmations" : 35566,
    "time" : 1388185914,
    "blocktime" : 1388185914
}
```


Ao contrário das transações comuns, a transação de geração não consome (gasta) UTXO como inputs. Ao invés disso, ela possui um único input, chamado de *coinbase*, que cria bitcoins a partir do nada. A transação de geração possui um único output, que é pago a um endereço bitcoin do próprio minerador. O output da transação de geração envia o valor de 25,09094928 bitcoins para o endereço bitcoin do minerador, nesse caso para 1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N.

Recompensa da Coinbase e Taxas

Para construir uma transação de geração, o nodo do Jing primeiro calcula a quantia total de taxas de transação ao somar todos os inputs e outputs das 418 transações que foram acrescentadas ao bloco. As taxas são calculadas assim:

```
Total de Taxas = Soma(Inputs) - Soma(Outputs)
```

No bloco 277.316, o total de taxas de transação é de 0,09094928 bitcoins.

A seguir, o nodo do Jing calcula a recompensa correta para o novo bloco. A recompensa é calculada baseando-se na altura do bloco, iniciando em 50 bitcoins por bloco e reduzindo pela metade a cada 210.000 blocos. Como esse bloco está na altura 277.316, a recompensa correta é de 25 bitcoins.

O cálculo pode ser visto na função GetBlockSubsidy no cliente Bitcoin Core, como demonstrado em Calculando a recompensa do bloco — Função GetBlockSubsidy, Cliente Bitcoin Core, main.cpp.

Example 5. Calculando a recompensa do bloco — Função GetBlockSubsidy, Cliente Bitcoin Core, main.cpp

```
CAmount GetBlockSubsidy(int nHeight, const Consensus::Params& consensusParams)
{
    int halvings = nHeight / consensusParams.nSubsidyHalvingInterval;
    // Força a recompensa do bloco para zero quando a mudança para a direita (right
shift)está indefinida
    if (halvings >= 64)
        return 0;

    CAmount nSubsidy = 50 * COIN;
    // O subsídio é dividido pela metade a cada 210.000 blocos, o que irá acontecer,
em média, a cada 4 anos.
    nSubsidy >>= halvings;
    return nSubsidy;
}
```

O subsídio inicial é calculado em satoshis ao se multiplar 50 com a constante COIN (100.000.000 satoshis). Isso define a recompensa inicial (nSubsidy) em 5 bilhões de satoshis.

A seguir, a função calcula o número de halvings que ocorreram ao dividir a altura do bloco atual pelo intervalo de halving (SubsidyHalvingInterval). No caso do bloco 277.316, com um intervalo de halving a cada 210.000 blocos, o resultado é 1 halving.

O número máximo permitido de halvings (divisões da recompensa pela metade) é 64, então o código impõe uma recompensa zero (retornando apenas o valor das taxas) se as 64 halvings forem excedidas.

A seguir, a função usa o operador de deslocamento binário à direita ("right-shift") para dividir a recompensa (nSubsidy) por dois a cada rodada de halving. No caso do bloco 277.316, isso faria um deslocamento binário à direita da recompensa de 5 bilhões de satoshi uma vez (um halving) e resultaria em 2,5 bilhões de satoshis, ou 25 bitcoins. O operador de deslocamento binário à direita é usado porque ele é mais eficiente na divisão por dois do que a divisão de números inteiros ou a divisão de ponto flutuante.

Finalmente, a recompensa coinbase (nSubsidy) é adicionada às taxas de transação (nFees), e retorna-se o valor da soma.

Estrutura da Geração da Transação

Com esses cálculos, o nodo do Jing constroi então a transação de geração para pagar 25,09094928 bitcoins para ele próprio.

Como você pode ver em Geração da transação, a transação de geração possui um formato especial. Ao invés de um input de transação especificando um UTXO prévio para gastar, ela possui um input "coinbase". Nós examinamos os inputs de transação em [tx_in_structure]. Vamos comparar o input de uma transação comum com o input de uma transação de geração. A estrutura de um input de transação "normal" demonstra a estrutura de uma transação comum, enquando A estrutura da geração de um input de transação demonstra a estrutura de um input de uma transação de geração.

Table 1. A estrutura de um input de transação "normal"

Tamanho	Campo	Descrição
32 bytes	Hash da Transação	Apontador para a transação contendo o UTXO para ser gasto
4 bytes	Índice do Output	O número índice do UTXO para ser gasto, o primeiro é 0
1-9 bytes (VarInt)	Tamanho do Script de Destravamento	Comprimento do script de destravamento em bytes, para seguir
Variável	Script de Destravamento	Um script que preenche os critérios (condições) para o script de travamento da UTXO
4 bytes	Número Sequencial	Funcionalidade de substituição de transação, atualmente desabilitada, definida como 0xFFFFFFFF

Table 2. A estrutura da geração de um input de transação

Tamanho	Campo	Descrição
32 bytes	Hash da Transação	Todos os bits são zeros: Não é uma referência a um hash de transação
4 bytes	Índice do Output	Todos os bits são um: 0xFFFFFFFF
1-9 bytes (VarInt)	Tamanho dos Dados Coinbase	Comprimento dos dados coinbase, de 2 a 100 bytes
Variável	Dados Coinbase	Dados arbitrários usados para o nonce extra e tags de mineração nos blocos v2, deve começar com a altura do bloco
4 bytes	Número Sequencial	Definido para 0xFFFFFFFF

Em uma transação de geração, os dois primeiros campos são definidos para valores que não representam uma referência UTXO. Ao invés de um "Hash de Transação", o primeiro campo é preenchido com 32 bytes, todos definidos para zero. O "Índice de Output" é preenchido com 4 bytes todos definidos para 0xFF (255 em decimais). O "Script de Destravamento" é substituído pelos dados coinbase, um campo de dados arbitrários usado pelos mineradores.

Dados Coinbase

As transações de geração não possuem um campo de script de destravamento (também conhecido como scriptSig). Ao invés disso, o campo é substituído pelos dados coinbase, que devem ter entre 2 e 100 bytes. Exceto pelos primeiros poucos bytes iniciais, o resto dos dados coinbase podem ser usados pelos minerados da maneira que eles quiserem; eles são dados arbitrários.

No bloco gênese, por exemplo, o Satoshi Nakamoto adicionou o texto "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks" ("The Times 03/Jan/2009 Chanceler prestes a conceder segundo resgate aos bancos") nos dados coinbase, usando isso como prova da data e para transmitir uma mensagem. Atualmente os mineradores usam os dados da coinbase para adicionar valores nonce extra e strings identificando o pool de mineração, como nós veremos nas seções a seguir.

Os primeiros poucos bytes da coinbase costumavam ser arbitrários, mas hoje em dia não são mais. A partir da Proposta de Melhoria do Bitcoin 34 (BIP0034), os blocos versão-2 (blocos com o campo de versão definido como 2) devem conter o índice da altura do bloco como uma operação "push" de script no início do campo coinbase.

No bloco 277.315 nós vemos que a coinbase (ver Geração da transação), que está no campo "Unlocking Script" ou scriptSig do input da transação, contém o valor hexadecimal 03443b0403858402062f503253482f. Vamos decodificar esse valor.

O primeiro byte, 03, instrui o motor de execução do script a empurrar os próximos três bytes para o stack do script (ver [tx_script_ops_table_pushdata]). Os próximos três bytes, 0x443b04, são a altura do

bloco codificada em formato little-endian (ao contrário, o byte menos significante primeiro). A ordem reversa dos bytes e o resultado é 0x043b44, que é 277.316 em decimais.

Os próximos poucos dígitos hexadecimais (03858402062) são usados para codificar um *nonce* extra (ver A Solução do Nonce Extra), ou um valor aleatório, usado para achar uma solução de prova de trabalho adequada.

A parte final dos dados coinbase (2f503253482f) é a string /P2SH/ codificada em ASCII, que indica que o nodo minerador que minerou esse bloco suporta a melhoria hash-pagamento-para-script (P2SH) definida pela BIP0016. A introdução da capacidade P2SH exigiu um "voto" pelos mineradores para apoiar o BIP0016 ou o BIP0017. Aqueles que apoiaram a implementação do BIP0016 iriam incluir /P2SH/ em seus dados coinbase. Aqueles que apoiaram a implementação BIP0017 do P2SH iriam incluir a string p2sh/CHV em seus dados coinbase. A BIP0016 foi eleita como a vencedora, e muitos mineradores continuaram a incluir a string /P2SH/ em sua coinbase para indicar que eles apoiavam essa funcionalidade.

Extração dos dados coinbase a partir do bloco gênese usa a livraria libbitcoin apresentada em [alt_libraries] para extrair os dados coinbase a partir do bloco gênese, exibindo a mensagem do Satoshi. Note que a livraria libbitcoin contém uma cópia estática do bloco gênese, portanto o código exemplo pode adquirir o bloco gênese diretamente da livraria.

Example 6. Extração dos dados coinbase a partir do bloco gênese

```
/*
  Display the genesis block message by Satoshi.
*/
#include <iostream>
#include <bitcoin/bitcoin.hpp>

int main()
{
    // Create genesis block.
    bc::block_type block = bc::genesis_block();
    // Genesis block contains a single coinbase transaction.
    assert(block.transactions.size() == 1);
    // Get first transaction in block (coinbase).
    const bc::transaction_type& coinbase_tx = block.transactions[0];
    // Coinbase tx has a single input.
    assert(coinbase_tx.inputs.size() == 1);
    const bc::transaction_input_type& coinbase_input = coinbase_tx.inputs[0];
    // Convert the input script to its raw format.
    const bc::data_chunk& raw_message = save_script(coinbase_input.script);
    // Convert this to an std::string.
    std::string message;
    message.resize(raw_message.size());
    std::copy(raw_message.begin(), raw_message.end(), message.begin());
    // Display the genesis block message.
    std::cout << message << std::endl;
    return 0;
}
```

Nós compilamos o código com o compilador GNU C++ e rodamos o executável resultante, como demonstrado em Compilando e executando o código de exemplo satoshi-words.

Example 7. Compilando e executando o código de exemplo satoshi-words

```
$ # Compila o código
$  g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Roda o executável
$ ./satoshi-words
^D    <GS>^A^DEThe Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

Construindo o Cabeçalho (Header) do Bloco

Para construir o cabeçalho do bloco, o nodo minerador precisa preenchê-lo em seis campos, como listado em A estrutura do cabeçalho do bloco.

Table 3. A estrutura do cabeçalho do bloco

Tamanho	Campo	Descrição
4 bytes	Versão	Um número de versão para servir como referência nas atualizações de software/protocolo.
32 bytes	Hash do Bloco Anterior	Uma referência ao hash do bloco anterior (bloco pai) na blockchain
32 bytes	Raiz de Merkle	Um hash da raiz da árvore de merkle das transações desse bloco
4 bytes	Data e Hora (timestamp)	O momento aproximado em que este bloco foi criado (em segundos, usando Unix Epoch)
4 bytes	Dificuldade Alvo	O alvo de dificuldade do algoritmo de prova-de-trabalho deste bloco
4 bytes	Nonce	Um contador usado para o algoritmo de prova-de-trabalho

No momento em que o bloco 277.316 foi minerado, o número de versão descrevendo a estrutura do bloco é a versão 2, que é codificada em formato little-endian em 4 bytes como 0x02000000.

A seguir, o nodo de mineração precisa adicionar o "Hash do Bloco Anterior". Esse é o hash do cabeçalho do bloco 277.315, o bloco anteriormente recebido pela rede, que o nodo do Jing aceitou e selecionou como pai do bloco candidato 277.316. O hash do cabeçalho do bloco 277.315 é:

```
0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

O próximo passo é fazer um resumo de todas as transações com uma árvore de merkle, para adicionar a raiz de merkle ao cabeçalho do bloco. A transação de geração é listada como a primeira transação do bloco. Então, mais 418 transações são adicionadas após ela, totalizando 419 transações no bloco. Como nós vimos em [merkle_trees], deve haver um número par de nodos "folhas" na árvore, de maneira que a última transação seja duplicada, criando 420 nodos, cada um contendo o hash de uma transação. Os hashes das transações são então combinados, em pares, criando cada nível da árvore, até que todas as

transações sejam resumidas em um nodo único na "raiz" da árvore. A raiz da árvore de merkle é um resumo de todas as transações em um valor único de 32 bytes, que você pode ver listado como "merkle root" em Bloco 277,316, e aqui:

```
c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e
```

O nodo de mineração irá então adicionar uma estampa de tempo de 4-bye, codificada como uma estampa de tempo Unix "Epoch", que é baseada no número de segundos decorridos desde à meia noite de 1º de janeiro de 1970 UTC/GMT. O tempo 1388185914 é igual a sexta-feira, 27 Dez 2013, 23:11:54 UTC/GMT.

O nodo então preenche a dificuldade alvo, que define a dificuldade prova-de-trabalho exigida para tornar esse bloco válido. A dificuldade é armazenada no bloco utilizando-se a métrica "bits de dificuldade" ("difficulty bits"), que é uma codificação mantissa-expoente do alvo. A codificação tem um expoente de 1 byte, seguido por uma mantissa (coeficiente) de 3 bytes. No bloco 277.316, por exemplo, o valor dos bits de dificuldade é 0x1903a30c. A primeira parte 0x19 é um expoente hexadecimal, enquanto a próxima parte, 0x03a30c, é o coeficiente. O conceito de um alvo de dificuldade é explicado em Alvo de Dificuldade e Reajustando o Alvo e a representação "bits de dificuldade" é explicada em Representação da Dificuldade.

O campo final é o nonce criptográfico, que é começa com zero.

Com todos os outros campos preenchidos, o cabeçalho do bloco agora está completo e o processo de mineração pode começar. O objetivo agora é encontar um valor para o nonce que resulta em um hash de cabeçalho de bloco que é menor do que a dificuldade alvo. O nodo de mineração terá que testar bilhões ou trilhões de valores de nonce até que encontre um nonce que satisfaça as exigências.

Minerando o Bloco

Agora que um bloco candidato foi construído pelo nodo do Jing, é hora do equipamento de mineração do Jing minerar o bloco, ou seja, encontrar uma solução para o algoritmo de prova de trabalho que fará com que o bloco seja válido. Ao longo desse livro, nós estudamos quais são as funções de hash criptográficas usadas em vários aspectos do sistema bitcoin. A função de hash SHA256 é a função usada no processo de mineração do bitcoin.

Explicando de maneira simples, a mineração é o processo de se ficar fazendo hashing do cabeçalho do bloco de maneira repetida, até que o hash resultante corresponda a um alvo específico. Os resultados da função de hash não podem ser determinados previamente, nem é possível criar um padrão que irá produzir um valor de hash específico. Essa funcionalidade das funções de hash significa que a única maneira de se produzir um resultado de hash que corresponde a um alvo específico é através de sucessivas tentativas, modificando aleatoriamente o input até que o hash resultante deseja apareça por acaso.

Algoritmo de Prova-de-Trabalho

Um algoritmo de hash pega um input de dados de comprimento arbitrário e produz um resultado determinístico de comprimento fixo, uma impressão digital do input. Para cada input específico, o hash resultante sempre será o mesmo e pode ser facilmente calculado e verificado por qualquer pessoa que implementar o mesmo algoritmo de hash. A característica chave de um algoritmo de hash criptográfico é que é virtualmente impossível encontrar dois inputs diferentes que produzem a mesma impressão digital. Como complemento, também é virtualmente impossível selecionar-se um input de uma maneira a produzir uma impressão digital desejada, a menos que fique se tentando inputs aleatórios.

Com SHA256, o output sempre tem 256 bits de comprimento, independente do tamanho do input. Em Exemplo de SHA256, nós iremos usar o intérprete Python para calcular o hash SHA256 da frase, "I am Satoshi Nakamoto."

Example 8. Exemplo de SHA256

```
$ python
```

```
Python 2.7.1
>>> import hashlib
>>> print hashlib.sha256("I am Satoshi Nakamoto").hexdigest()
5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e
```

Exemplo de SHA256 demonstra o resultado de se calcular o hash da frase "I am Satoshi Nakamoto": 5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e. Esse número é o *hash* ou a *compilação* ("*digest*") da frase e ele é dependente de cada parte da frase. A adição de uma única letra, uma pontuação ou qualquer outro caractere irá produzir um hash diferente.

Agora, se nós mudarmos a frase, nós deveríamos ver hashes completamente diferentes. Vamos experimentar isso ao adicionar um número ao final de nossa frase, usando o script Python simples em SHA256 Um script para gerar muitos hashes ao fazer iteração com um nonce.

Example 9. SHA256 Um script para gerar muitos hashes ao fazer iteração com um nonce

```
# example of iterating a nonce in a hashing algorithm's input

import hashlib

text = "I am Satoshi Nakamoto"

# iterate nonce from 0 to 19
for nonce in xrange(20):

    # add the nonce to the end of the text
    input = text + str(nonce)

    # calculate the SHA-256 hash of the input (text+nonce)
    hash = hashlib.sha256(input).hexdigest()

    # show the input and hash result
    print input, '=>',  hash
```

A execução desse exemplo irá produzir os hashes de várias frases, que serão diferentes devido à adição de um número no final do texto. Ao incrementar o número, nós podemos obter hashes diferentes, como demonstrado em Output SHA256 de um script para gerar muitos hashes fazendo iteração com um nonce.

Example 10. Output SHA256 de um script para gerar muitos hashes fazendo iteração com um nonce

```
$ python hash_example.py
```

```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

Cada frase produz um resultado de hash completamente diferente. Eles parecem ser completamente aleatórios, mas você é capaz de reproduzir exatamente os mesmos resultados desse exemplo em qualquer computador com Python e ver exatamente os mesmos hashes.

O número usado como uma variável nesse cenário é chamado de *nonce*. O nonce é usado para variar o output de uma função criptográfica, nesse caso para variar a impressão digital SHA256 da frase.

Para fazer um desafio com esse algoritmo, vamos definir um alvo arbitrário: encontra uma frase que produza um hash hexadecimal que começa com um zero. Felizmente, isso não é difícil! Output SHA256 de um script para gerar muitos hashes fazendo iteração com um nonce demonstra que a frase "I am Satoshi Nakamoto13" produz o hash 0ebc56d59a34f5082aaef3d66b37a661696c2b618e62432727216ba9531041a5, que preenche nossos critérios. Levaria 13 tentativas até achá-lo. Em termos de probabilidades, se o output da função de hash é distribuído igualmente, nós esperaríamos encontrar um resultado com um zero como prefixo hexadecimal uma vez a cada 16 hashes (um em cada 16 dígitos hexadecimais 0 até F). Em termos numéricos, isso significa achar um valor de hash que é menor que 0x1000000000000000000000000000000000000000000000000000000000000000. Nós chamamos esse limite de *alvo* e o objetivo é encontrar um hash que é numericamente *menor que o alvo*. Se nós

diminuirmos o alvo, a tarefa de encontrar um hash que é menor que o alvo se torna cada vez mais difícil.

Para usarmos uma analogia, imagine um jogo onde os jogadores lançam um par de dados repetidamente, tentando tirar uma soma menor do que um alvo especificado. Na primeira rodada, o alvo da soma é 12. A menos que você tire dois seis, você ganha. Na próxima rodada o alvo é 11. Os jogadores devem tirar uma soma menor ou igual a 10 para ganhar, o que também é uma tarefa fácil. Digamos que algumas rodadas depois o alvo é 5. Agora, mais do que a metade dos lançamentos irão somar mais do que 5 e, portanto, serão inválidos. O número de lançamentos necessários para se ganhar aumenta exponencialmente, quanto menor for o alvo. Por fim, quando o alvo é 2 (o menor possível), somente um lançamento a cada 36, ou 2% deles, irá produzir um resultado vencedor.

Em Output SHA256 de um script para gerar muitos hashes fazendo iteração com um nonce, o "nonce" vencedor é 13 e esse resultado pode ser confirmado por qualquer pessoa de maneira independente. Qualquer um pode adicionar o número 13 como sufixo da frase "I am Satoshi Nakamoto" e computar o hash, verificando que ele é menor do que o alvo. A confirmação do resultado (do "nonce") também é uma prova de trabalho, pois ela comprova que nós trabalhamos para encontrar esse nonce. Enquanto a verificação exige a computação de apenas um hash, foi necessária a computação de 13 hashes até que encontrássemos um nonce que funcionasse. Se nós tivéssemos um alvo menor (uma dificuldade maior), seria necessária a computação de muito mais hashes até encontrar um nonce compatível, mas continuaria sendo necessária a computação de apenas um hash para verificá-lo. Além disso, ao ter o conhecimento do alvo, qualquer um pode estimar a dificuldade usando estatísticas, e portanto saberá quanto de trabalho foi necessário para se descobrir o "nonce".

A prova de trabalho do bitcoin é muito semelhante ao desafio demonstrado em Output SHA256 de um script para gerar muitos hashes fazendo iteração com um nonce. O minerador constroi um bloco candidado preenchido com transações. A seguir, o minerador calcula o hash do cabeçalho desse bloco e verifica se ele é menor do que o *alvo* atual. Se o hash não for menor do que o alvo, o minerador irá modificar o nonce (geralmente somando o número 1) e tentará novamente. Na dificuldade atual da rede bitcoin, os mineradores tem que fazer quadrilhões de tentativas até achar um nonce que resulte em um hash de cabeçalho de bloco baixo o suficiente.

Um algoritmo muito simplificado de prova-de-trabalho é implementado em Python em Implementação simplificada de prova-de-trabalho.

Example 11. Implementação simplificada de prova-de-trabalho

```
#!/usr/bin/env python
# example of proof-of-work algorithm

import hashlib
import time

max_nonce = 2 ** 32 # 4 billion

def proof_of_work(header, difficulty_bits):
```

```
    # calculate the difficulty target
    target = 2 ** (256-difficulty_bits)

    for nonce in xrange(max_nonce):
        hash_result = hashlib.sha256(str(header)+str(nonce)).hexdigest()

        # check if this is a valid result, below the target
        if long(hash_result, 16) < target:
            print "Success with nonce %d" % nonce
            print "Hash is %s" % hash_result
            return (hash_result,nonce)

    print "Failed after %d (max_nonce) tries" % nonce
    return nonce


if __name__ == '__main__':

    nonce = 0
    hash_result = ''

    # difficulty from 0 to 31 bits
    for difficulty_bits in xrange(32):

        difficulty = 2 ** difficulty_bits
        print "Difficulty: %ld (%d bits)" % (difficulty, difficulty_bits)

        print "Starting search..."

        # checkpoint the current time
        start_time = time.time()

        # make a new block which includes the hash from the previous block
        # we fake a block of transactions - just a string
        new_block = 'test block with transactions' + hash_result

        # find a valid nonce for the new block
        (hash_result, nonce) = proof_of_work(new_block, difficulty_bits)

        # checkpoint how long it took to find a result
        end_time = time.time()

        elapsed_time = end_time - start_time
        print "Elapsed Time: %.4f seconds" % elapsed_time

        if elapsed_time > 0:
```

```
            # estimate the hashes per second
            hash_power = float(long(nonce)/elapsed_time)
            print "Hashing Power: %ld hashes per second" % hash_power
```

Ao executar esse código, você pode definir a dificuldade desejada (em bits, a quantidade de bits inicias deve ser zero) e ver quanto tempo leva para seu computador encontrar uma solução. Em Executando o exemplo de prova de trabalho para várias dificuldades, você pode ver como isso funciona em um laptop comum.

Example 12. Executando o exemplo de prova de trabalho para várias dificuldades

```
$ python proof-of-work-example.py*
```

```
Dificuldade: 1 (0 bits)

[...]

Dificuldade: 8 (3 bits)
Iniciando a busca...
Sucesso com o nonce 9
O hash é 1c1c105e65b47142f028a8f93ddf3dabb9260491bc64474738133ce5256cb3c1
Tempo Decorrido: 0,0004 segundos
Poder de Hashing: 25.065 hashes por segundo
Dificuldade: 16 (4 bits)
Iniciando a busca...
Sucesso com o nonce 25
O hash é 0f7becfd3bcd1a82e06663c97176add89e7cae0268de46f94e7e11bc3863e148
Tempo Decorrido: 0,0005 segundos
Poder de Hashing: 52.507 hashes por segundo
Dificuldade: 32 (5 bits)
Iniciando a busca...
Sucesso com o nonce 36
O hash é 029ae6e5004302a120630adcbb808452346ab1cf0b94c5189ba8bac1d47e7903
Tempo Decorrido: 0,0006 segundos
Poder de Hashing: 58.164 hashes por segundo

[...]

Dificuldade: 4194304 (22 bits)
Iniciando a busca...
Sucesso com o nonce 1759164
O hash é 0000008bb8f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cefc3
Tempo Decorrido: 13,3201 segundos
Poder de Hashing: 132.068 hashes por segundo
```

```
Dificuldade: 8388608 (23 bits)
Iniciando a busca...
Sucesso com o nonce 14214729
O hash é 000001408cf12dbd20fcba6372a223e098d58786c6ff93488a9f74f5df4df0a3
Tempo Decorrido: 110,1507 segundos
Poder de Hashing: 129.048 hashes por segundo
Dificuldade: 16777216 (24 bits)
Iniciando a busca...
Sucesso com o nonce 24586379
O hash é 0000002c3d6b370fccd699708d1b7cb4a94388595171366b944d68b2acce8b95
Tempo Decorrido: 195,2991 segundos
Poder de Hashing: 125.890 hashes por segundo

[...]

Dificuldade: 67108864 (26 bits)
Iniciando a busca...
Sucesso com o nonce 84561291
O hash é 0000001f0ea21e676b6dde5ad429b9d131a9f2b000802ab2f169cbca22b1e21a
Tempo Decorrido: 665,0949 segundos
Poder de Hashing: 127.141 hashes por segundo
```

Como você pode ver, aumentar a dificuldade em 1 bit causa um aumento exponencial no tempo que leva para se encontrar uma solução. Se você pensar em todo o espaço numérico de 256-bits, cada vez que você limite um bit a mais para zero, você diminui o espaço de busca pela metade. Em Executando o exemplo de prova de trabalho para várias dificuldades, leva 84 milhões de tentativas de hash para se encontrar um nonce que produza um hash com 26 bits iniciais como zero. Mesmo em uma velocidade de mais de 120.000 hashes por segundo, ainda são necessários 10 minutos para encontrar essa solução em um notebook comum .

Na época em que esse livro foi escrito, a rede estava tentanto achar um bloco cujo hash do cabeçalho fosse menor que 000000000000004c296e6376db3a241271f43fd3f5de7ba18986e517a243baa7. Como você pode ver, existem vários zeros no início desse hash, significando que a faixa aceitável de hashes é muito menor, portanto é mais difícil de se encontrar um hash válido. Levará em média mais de 150 quadrilhões de cálculos hash por segundo para que a rede descubra o próximo bloco. Isso parece uma tarefa quase impossível, mas felizmente a rede já está com um poder de processamento de 100 petahashes por segundo (PH/seg), que serão capazes de encontrar um bloco em cerca de 10 minutos em média.

Representação da Dificuldade

Em Bloco 277,316, nós vimos que o bloco contém a dificuldade alvo, em uma notação chamada "bits de dificuldade" ou apenas "bits", que no bloco 277.316 tinha o valor de 0x1903a30c. Esse notação expressa a dificuldade alvo em um formato coeficiente/expoente, com os primeiros dois dígitos hexadecimais sendo o expoente e os próximos seis dígitos hexadecimais sendo o coeficiente. Nesse bloco, portanto, o

expoente é 0x19 e o coeficiente é 0x03a30c.

A fórmula para calcular a dificuldade-alvo a partir dessa representação é:

```
alvo = coeficiente * 2^(8 * (expoente – 3))
```

Usando essa fórmula, e o valor 0x1903a30c para os bits de dificuldade, nós obtemos:

```
alvo = 0x03a30c * 2^(0x08 * (0x19 - 0x03))^

=> alvo = 0x03a30c * 2^(0x08 * 0x16)^

=> alvo = 0x03a30c * 2^0xB0^
```

que em decimal é:

```
=> alvo = 238,348 * 2^176^

=> alvo = 22,829,202,948,393,929,850,749,706,076,701,368,331,072,452,018,388,575,715,328
```

transformando de volta para hexadecimal:

```
=> alvo = 0x0000000000000003A30C000000000000000000000000000000000000000000
```

Isso significa que um bloco válido para a altura 277.316 é um quer tem um hash de cabeçalho de bloco que é menor do que o alvo. Em binários esse número teria mais do que os primeiros 60 bits definidos como zero. Com esse nível de dificuldade, um único minerador processando 1 trilhão de hases por segundo (1 tera-hash por segundo ou 1 TH/seg) encontraria uma solução, em média, somente uma vez a cada 8.496 blocos ou uma vez a cada 59 dias.

Alvo de Dificuldade e Reajustando o Alvo

Como vimos anteriormente, o alvo determina a dificuldade e portanto afeta quão demora será para encontrar uma solução para o algortimo de prova-de-trabalho. Isso nos levanta questões óbvias: Por que a dificuldade é ajustável, quem a ajusta e como?

Os blocos do Bitcoin são gerados a cada 10 minutos, em média. Essa é a "frequência cardíaca" do bitcoin e é a base da frequência de emissão de moeda e a velocidade de confirmação das transações. Ela tem que permanecer constante não apenas a curto prazo, mas também a prazos muito longos, como décadas. Ao longo desse tempo, espera-se que o poder de processamento irá aumentar em um ritmo muito rápido. Além disso, o número de paricipantes na mineração e os computadores que eles usam também irá mudar constantemente. Para manter o tempo de geração de blocos em 10 minutos, a

dificuldade da mineração deve ser ajustada para contabilizar essas mudanças. De fato, a dificuldade é um parâmetro dinâmico que será ajustado periodicamente para atingir o alvo de um bloco a cada 10 minutos. Simplificando, o alvo de dificuldade é definido para qualquer poder de mineração que resulte em um intervalo de 10 minutos entre a geração dos blocos.

Como, então, tal ajusta é feito em uma rede completamente descentralizada? O reajuste do alvo da dificuldade ocorre automaticamente e em cada nodo completo de maneira independente. A cada 2.016 blocos, todos os nodos reajustam o alvo da dificuldade de prova de trabalho. A equação para reajustar o alvo mede o tempo que levou para encontrar os últimos 2.016 blocos e o compara com o tempo esperado de 20.160 minutos (duas semanas, baseando-se em um tempo desejado de 10 minutos para gerar um bloco). A razão entre os períodos de tempo atual e o desejado é calculada, e um ajuste correspondente (para cima ou para baixo) é feito à dificuldade. Simplificando: Se a rede estiver encontrando blocos mais rápido do que a cada 10 minutos, a dificuldade irá aumentar. E se a descoberta de blocos estiver mais lenta que o esperado, a dificuldade irá diminuir.

A equação pode ser resumida como:

```
Nova Dificuldade = Dificuldade Antiga * (Tempo dos últimos 2.016 blocos / 20.160 minutos)
```

Mudando a dificuldade alvo da prova-de-trabalho — CalculateNextWorkRequired() in pow.cpp demonstra o código usado no cliente Bitcoin Core.

Example 13. Mudando a dificuldade alvo da prova-de-trabalho — CalculateNextWorkRequired() in pow.cpp

```
// Limita o passo de ajuste
 int64_t nActualTimespan = pindexLast->GetBlockTime() - nFirstBlockTime;
 LogPrintf("  nActualTimespan = %d  before bounds\n", nActualTimespan);
 if (nActualTimespan < params.nPowTargetTimespan/4)
     nActualTimespan = params.nPowTargetTimespan/4;
 if (nActualTimespan > params.nPowTargetTimespan*4)
     nActualTimespan = params.nPowTargetTimespan*4;

 //Reestabelecer o alvo
 const arith_uint256 bnPowLimit = UintToArith256(params.powLimit);
 arith_uint256 bnNew;
 arith_uint256 bnOld;
 bnNew.SetCompact(pindexLast->nBits);
 bnOld = bnNew;
 bnNew *= nActualTimespan;
 bnNew /= params.nPowTargetTimespan;

 if (bnNew > bnPowLimit)
     bnNew = bnPowLimit;
```

NOTE Enquanto a calibragem da dificuldade acontece a cada 2.016 blocos, a existência de um erro off-by-one (OBOE, "erro por um") no cliente Bitcoin Core original faz com que ela seja baseada no tempo total dos 2.015 blocos anteriores (e não 2.016, como deveria ser), o que resulta em uma dificuldade 0,05% maior na modificação do alvo.

Os parâmetros Interval (2.016 blocos) e TargetTimespan (duas semanas ou 1.209.600 segundos) são definidos em *chainparams.cpp*.

Para evitar uma volatilidade extrema na dificuldade, o ajusto do alvo deve ser menor do que um fator de quatro (4) por ciclo. Se o ajuste de dificuldade necessário é maoir do que um fator de quatro, ele será ajustado até o máximo, e não mais do que isso. Qualquer ajuste adicional será realizado no próximo período de ajuste do alvo, pois o desequilíbrio irá persistir durante os próximos 2.016 blocos. Portanto, grandes discrepâncias entre o poder de hashing e a dificuldade podem levar vários ciclos de 2.016 blocos para se equilibrarem.

TIP A dificuldade de se achar um bloco bitcoin é de aproximadamente '10 minutos de processamento' para toda a rede, baseando-se no tempo que levou para encontrar os últimos 2.016 blocos, ajustada a cada 2.016 blocos.

Perceba que a dificuldade alvo é independente do número de transações ou do valor das transações. Isso significa que o alvo do poder de hashing e, portanto, a eletricidade gasta para manter a segurança

da rede bitcoin é também totalmente independente do número de transações. O bitcoin pode se expandir, atingir uma adoção maior e se manter seguro sem nenhum aumento no poder de hashing que já existe hoje. O aumento no poder de hashing representa as forças do mercado à medida que novos mineradores entram no mercado para competir pela recompensa. Enquanto houver um poder de hashing suficiente sob o controle de mineradores agindo honestamente em busca da recompensa, ele será suficiente para prevenir ataques de "assumir o controle" ("takeover") e, portanto, será suficiente para manter o bitcoin seguro.

A dificuldade-alvo é intimamente relacionada com o custo da eletricidade e a taxa de câmbio do bitcoin em relação à moeda usada para pagar pela eletricidade. Sistemas de mineração de alta performance são o mais eficientes possíveis com a geração atual que é fabricada em silício, convertendo a eletricidade em computação de hashes na taxa mais alta possível. A principal influência no mercado de mineração é o preço de um kilowatt-hora em bitcoin, porque ele determina a rentabilidade da mineração e, portanto, os incentivos para se entrar ou sair do mercado de mineração.

Minerando um Bloco com Sucesso

Como nós vimos anteriormente, o nodo do Jing construiu um bloco candidato e o preparou para a mineração. O jing possui vários equipamentos de mineração em hardware com circuitos integrados de aplicação especifica (CIAE ou ASICs), onde centenas de milhares de circuitos integrados executam a velocidades incríveis o algoritmo SHA256 em paralelo. Essas máquinas especializadas são conectadas ao seu nodo de mineração através de USB. A seguir, o nodo de mineração sendo executado no desktop do Jin transmite o cabeçalho do bloco para seu hardware de mineração, que começa as tentativas de trilhões de nonces por segundo.

Quase 11 minutos depois de começar a minerar o bloco 277.316, uma das máquinas de mineração encontra uma solução a enviar de volta ao nó de mineração. Quando inserida no cabeçalho do bloco, o nonce 4.215.469.401 produz um bloco de hash de:

```
0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

que é menor do que o alvo:

```
0000000000000003A30C0000000000000000000000000000000000000000000
```

Imediatamente, o nodo de mineração do Jing transmite o bloco para todos os seus pontos. Eles recebem, validam e então propagam o novo bloco. À medida que o bloco é disseminado na rede, cada nodo o adiciona à sua própria cópia da blockchain, estendendo-a para uma nova altura de 277.316 blocos. Conforme os nodos de mineração vão recebendo e validando o bloco, eles abandonam seus esforços de encontrar um bloco com a mesma altura e imediatamente passam a computar o próximo bloco na corrente.

Na próxima seção, nós iremos aprender sobre o processo que cada nodo usa para validar um bloco e

selecionar a cadeia mais longa, criando o consenso que forma a cadeia de blocos descentralizada (blockchain).

Validando um Novo Bloco

O terceiro passo no mecanismo de consenso do bitcoin é a validação independente de cada bloco novo, que é realizada por cada nodo da rede. À medida que os blocos recém-descobertos são propagados pela rede, cada nodo realiza uma série de testes para validá-lo antes de propagá-lo para seus pares. Isso garente que somente blocos válidos serão propagados na rede. A validação independente também garante que os mineradores que agirem honestamente terão seus blocos incorporados na blockchain, e, portanto, receberão sua recompensa. Os mineradores que agirem desonestamente terão seus blocos rejeitados e não apenas perderão sua recompensa, como também terão desperdiçado o esforço despendido para encontrar uma solução de prova-de-trabalho, ou seja, terão gasto com energia elétrica sem obter nenhuma compensação.

Quando um nodo recebe um bloco novo, ele irá validar o bloco ao verificá-lo contra uma longa lista de critérios que precisam ser preenchidos; caso contrário, o bloco será rejeitado. Esses critérios podem ser vistos no cliente Bitcoin Core nas funções CheckBlock e CheckBlockHeader e incluem:

- A estrutura do bloco de dados é sintaticamente válida.
- O hash do cabeçalho do bloco é menor do que a dificuldade de destino (garante a prova de trabalho)
- O timestamp do bloco é menos do que duas horas no futuro (permitindo erros de tempo)
- O tamanho do bloco está dentro dos limites aceitáveis
- A primeira transação (e somente a primeira) é uma transação de geração coinbase
- Todas as transações contidas no bloco são válidas usando o checklist de transações discuto em Verificação Independente de Transações

A validação independente de cada bloco novo feita por todos os nodos da rede garante que os mineradores não podem trapacear. Nas seções anteriores nós vimos como os mineradores conseguem escrever uma transação que os recompensa com os novos bitcoins criados no interior do bloco e os paga as taxas de transação. Por que os mineradores não escrevem para eles próprios uma transação com mil bitcoins de recompensa, ao invés de usar a recompensa correta? Porque todos os nodos validam os blocos de acordo com as mesmas regras. Uma transação coinbase inválida tornaria inválido todo o bloco, o que resultaria em um bloco sendo rejeitado e, portanto, a transação jamais seria incluída na blockchain. Os mineradores tem que construir um bloco perfeito, baseado nas regras compartilhadas que todos os nodos seguem, e minerá-lo com uma solução correta para a prova de trabalho. Para fazer isso, eles gastam muita energia elétrica na mineração, e se eles trapacearem, todos os seus gastos com energia elétrica e todo o seu esforço serão desperdiçados. Esse é o motivo pelo qual a validação independente é o componente chave do consenso descentralizado.

Montando e Selecionando Cadeias de Blocos

O passo final no mecanismo de consenso descentralizado do bitcoin é a união dos blocos em cadeias e a seleção da cadeia que possui a maior prova de trabalho. Assim que o nodo validar um bloco novo, ele irá tentar montar uma cadeia ligando o bloco à blockchain existente.

Os nodos mantém três conjuntos de blocos: aqueles conectados à blockchain principal, aqueles que foram ramificações da blockchain principal (correntes secundárias), e, finalmente, os blocos que não tem um pai conhecido nas correntes conhecidas (órfãos). Os blocos inválidos são rejeitados assim que qualquer critério de validação falhar e, portanto, não são incluídos em nenhuma corrente.

A "corrente principal" a qualquer momento é a corrente de blocos que tiver a maior dificuldade acumulada associada a ela. Sob a maioria das circunstâncias ela tamém será a corrente que tiver o maior número de blocos, a menos que existam outras duas correntes de comprimento semelhante e uma delas tiver mais prova de trabalho. A corrente principal também terá ramificações com blocos que são "irmãos" dos blocos que estão na corrente principal. Eles são mantidos para referência futura, caso uma dessas correntes for estendida e passar a apresentar maior dificuldade que a corrente principal. Na próxima seção (Forks (bifurcações) da Blockchain), nós iremos ver como as correntes secundárias surgem quando há uma mineração quase simultânea de blocos em uma mesma altura.

Quando um bloco novo é recebido, um nodo irá tentar acrescentá-lo na blockchain existente. O nodo irá procurar no bloco pelo campo "hash do bloco anterior", que é a referÊncia ao pai do bloco novo. Então, o nodo irá tentar encontrar esse pai na blockchain existente. Na maioria das vezes, o pai será o "topo" da corrente principal, ou seja, esse bloco novo irá estender a corrente principal. Por exemplo, o novo bloco 277.316 tem uma referência ao hash de seu bloco pai 277.315. A maioria dos nodos que recebem o 277.316 já terão o bloco 277.315 como o topo de suas corrente principal e, portanto, vincularão o bloco novo e estenderão a corrente.

Às vezes, como nós veremos em Forks (bifurcações) da Blockchain, o bloco novo estende um corrente que não está na corrente principal. Nesse caso, o nodo irá anexar o bloco novo à corrente secundária que ele estende e então irá comparar a dificuldade da corrente secundária com a dificuldade da corrente principal. Se a corrente secundária possuir uma dificuldade acumulada maior do que a da corrente principal, o nodo irá *convergir novamente* para a corrente secundária, ou seja, ele irá selecionar a corrente secundária como sua nova corrente principal, fazendo com que a corrente principal anterior se torne uma corrente secundária. Se o nodo for um minerador, ele irá construir um bloco ao estender essa corrente nova, que é mais longa.

Se um bloco válido é recebido e nenhum pai é encontrado na correntes existentes, o bloco é considerado um "órfão". Os blocos órfãos são salvos no pool de blocos órfãos, onde eles ficam até que o seus pais sejam recebidos. Assim que o bloco pai é recebido e vinculado às correntes existentes, o bloco órfão pode então sair do pool e ser vinculado a seu bloco pai, passando a fazer parte da corrente. Os blocos órfãos geralmente surgem quando dois blocos que foram minerados em um curto espaço de tempo são recebidos em ordem inversa, ou seja, o bloco filho é recebido antes do bloco pai.

Como selecionam a corrente com mais dificuldade, todos os nodos acabam atingindo um consenso

disseminado na rede. As discrepâncias temporárias entre as correntes acabam sendo resolvidas à medida que se adiciona mais prova-de-trabalho, estendendo uma das possíveis correntes. Os nodos mineradores "votam" com seu poder de mineração ao escolher qual corrente eles irão estender através da mineração do próximo bloco. Ao minerar um bloco novo e estender a corrente que eles escolheram, os mineradores estão votando nessa corrente.

Na próxima seção iremos ver como discrepâncias entre cadeias em competição (forks) são resolvidas através da seleção independente da cadeia de maior dificuldade.

Forks (bifurcações) da Blockchain

Como a blockchain é uma estrutura de dados descentralizada, as diferentes cópias dela não são sempre iguais. Os blocos podem chegar em diferentes nodos em momentos diferentes, fazendo com que os nodos tenham diferentes perspectivas da blockchain. Para resolver isso, cada nodo sempre seleciona e tenta estender a corrente de blocos que representa a maior prova de trabalho, também conhecida como a corrente mais longa, ou a corrente com a maior dificuldade acumulada. Ao somar a dificuldade registrada em cada bloco de uma corrente, um nodo pode calcular a quantia total da prova de trabalho que foi despendida para criar aquela corrente. Enquanto todos os nodos selecionarem a corrente mais longa, que tem a maior dificuldade acumulada, a rede global do bitcoin irá convergir para um estado consistente, onde as diferentes cópias são iguais. As bifurcações ocorrem devido a inconsistências temporárias entre as versões da blockchain, que são acabam sendo resolvidas com a reconvergência, à medida que mais blocos são adicionados a uma das ramificações da bifurcação.

Nos próximos diagramas, nós iremos ver como ocorre o evento de uma "bifurcação" ao longo da rede. O diagrama é uma representação simplificada do bitcoin como uma rede global. Na verdade, a topologia da rede bitcoin não é organizada geograficamente. Ao invés disso, ela forma uma rede em malha de nodos interconectados, que podem estar localizados geograficamente muito longe uns dos outros. A representação de uma topologia geográfica é uma simplificação usada com o objetivo de ilustrar uma bifurcação. Na rede bitcoin verdadeira, a "distância" entre os nodos é medida em "saltos" ("hops") de um nodo para o outro, e não em sua localização física. Para fins ilustrativos, diferentes blocos são exibidos como cores diferentes, espalhados ao longo da rede e colorindo as conexões que eles cruzam.

No primeiro diagrama (Visualização de um evento de bifurcação da blockchain—antes da bifurcação), a rede possui uma perspectiva unificada da blockchain, com o bloco azul sendo o topo da cadeia principal.

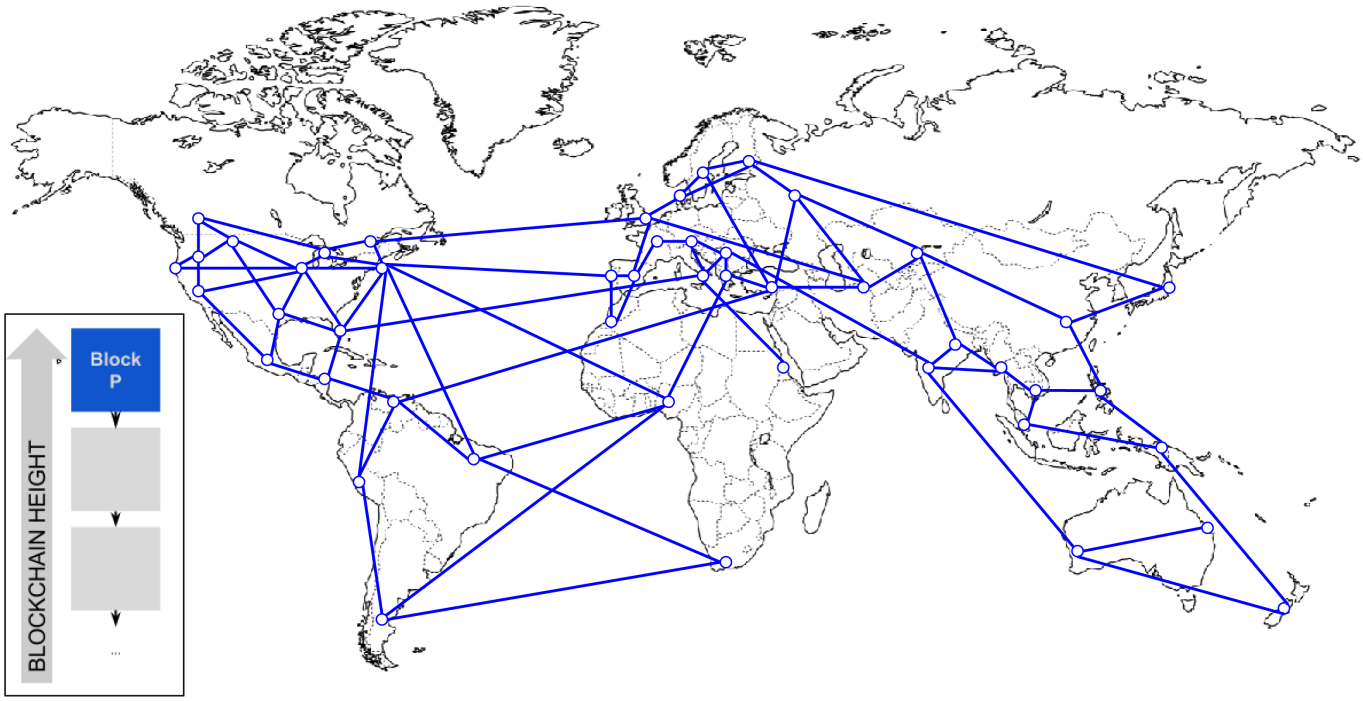


Figure 2. Visualização de um evento de bifurcação da blockchain—antes da bifurcação

Uma "bifurcação" ocorre sempre que houver dois blocos candidatos competindo para formarem a blockchain mais longa. Isso ocorre sob condições normais sempre que dois minerados resolvem o algoritmo de prova de trabalho com uma diferença de tempo muito pequena. Quando ambos os mineradores descobrem uma solução para seus respectivos blocos candidatos, eles imediatamente transmitem o seu bloco "vencedor" para seus vizinhos imediatos que começam a propagar o bloco pela rede. Cada nodo que recebe um bloco válido irá incorporá-lo à sua blockchain, estendendo a blockchain em um bloco. Se aquele nodo descobrir mais tarde outro bloco candidato estendendo o mesmo bloco pai, ele conectará o segundo candidato em uma corrente secundária. Como resultado, alguns nodos irão "enxergar" um bloco candidato primeiro, enquanto outros nodos irão enxergar o outro bloco candidato, surgindo duas versões concorrentes da blockchain.

Em Visualização de um evento de bifurcação da blockchain: dois blocos são encontrados simultaneamente, nós podemos ver dois mineradores que mineram dois blocos diferentes quase que simultaneamente. Esses dois blocos são filhos do bloco azul, e são destinados a estender a cadeia ao serem adicionados no topo do bloco azul. Para facilitar o nosso acompanhamento, um deles é visualizado como um bloco vermelho se originando do Canadá, e o outro é marcado como um bloco verde se originando da Austrália.

Vamos assumir, por exemplo, que um minerador no Canadá encontra uma solução de prova de trabalho para um bloco "vermelho" que estende a blockchain, sendo adicionado no topo do bloco pai "azul". Quase simultaneamente, um minerador australiano que também está estendendo o bloco "azul" encontra uma solução para o bloco "verde", o seu bloco candidato. Agora, existem dois blocos possíveis, um "vermelho", se originando no Canadá, e outro "verde", se originando na Austrália. Ambos os blocos são válidos, ambos os blocos contém uma solução válida para a prova de trabalho e ambos os blocos estendem o mesmo pai. Provavelmente a maioria das transações de ambos os blocos é semelhante, com apenas algumas diferenças na ordem das transações.

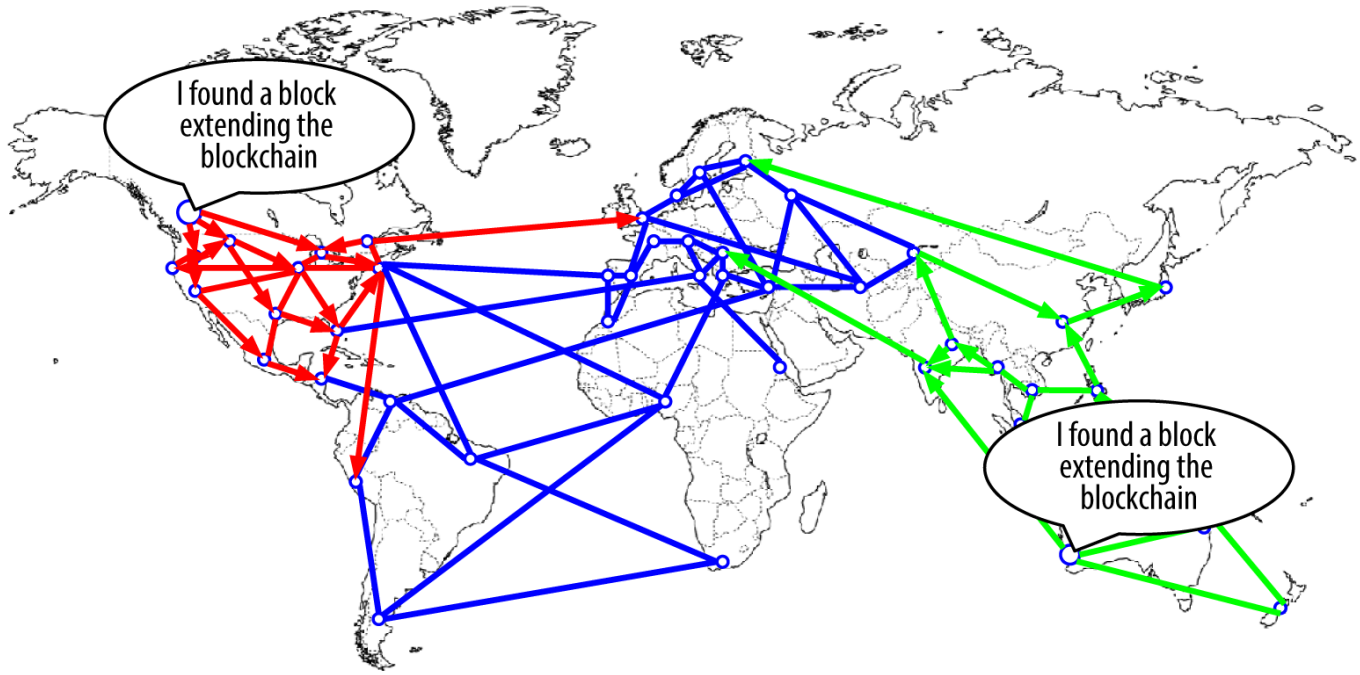


Figure 3. Visualização de um evento de bifurcação da blockchain: dois blocos são encontrados simultaneamente

À medida que os dois blocos são propagados, alguns nodos recebem o bloco "vermelho" antes e alguns recebem o bloco "verde" antes. Como demonstrado em Visualização de um evento de bifurcação da blockchain: dois blocos são propagados, dividindo a rede, a rede se bifurca em duas perspectivas diferentes da blockchain, um lado coberto com um bloco vermelho, e outro lado coberto com um bloco verde.

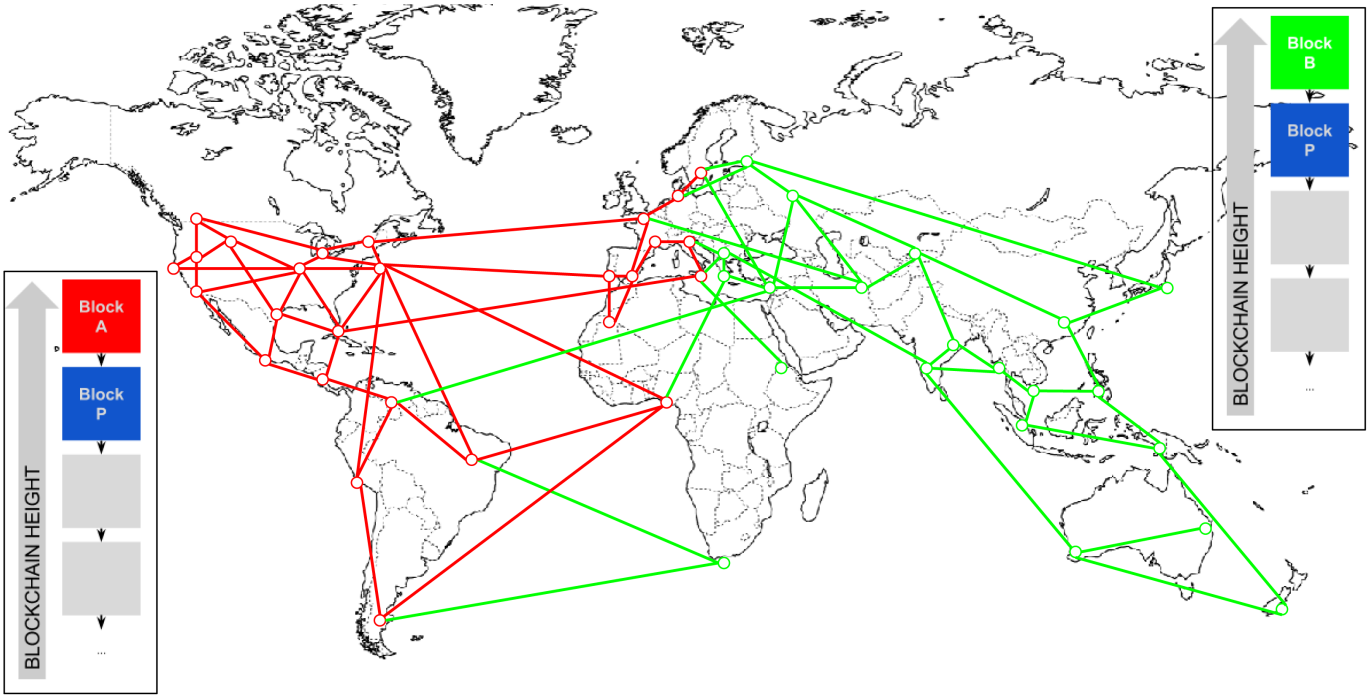


Figure 4. Visualização de um evento de bifurcação da blockchain: dois blocos são propagados, dividindo a rede

A partir desse momento, os nodos da rede bitcoin que estiverem mais próximos (topologicamente, e não geograficamente) ao nodo canadense irão ficar sabendo sobre o bloco "vermelho" antes e irão criar uma nova blockchain com a maior dificuldade acumulada, que tem o bloco "vermelho" como último bloco na corrente (por exemplo, azul-vermelho), ignorando o bloco candidato "verde" que chegar um pouco depois. Enquanto isso, os nodos mais próximos do nodo australiano irão considerar aquele bloco como o vencedor e irão estender a blockchain com o bloco "verde" como o último bloco (por exemplo, azul-verde), ignorando o bloco candidato "vermelho" quando ele chegar alguns segundos depois. Qualquer minerador que enxergar o bloco "vermelho" antes irá imediatamente construir blocos candidatos que usam o "vermelho" como referência para o pai, e começarão a tentar resolver a prova de trabalho para esses blocos candidatos. Por outro lado, os mineradores que aceitaram o bloco "verde" irão começar a construir no topo do "verde", estendendo aquela corrente.

As bifurcações quase sempre são resolvidas em um bloco. Como parte do poder de hashing da rede é dedicado a construir no topo do "vermelho" como pai, outra parte do poder de hashing é focado em construir no topo do "verde". Mesmo que o poder de mineração esteja dividido quase igualmente, é provável que um conjunto de mineradores irá encontrar uma solução e propagá-la antes que outro conjunto de mineradores tenha encontrado algumas soluções. Digamos que, por exemplo, os mineradores construindo no topo do "verde" encontra um novo bloco "rosa" que estende a corrente (por exemplo, azul-verde-rosa). Eles imediatamente propagam esse bloco novo e toda a rede o enxerga como uma solução válida, como demonstrado em Visualização de um evento de bifurcação da blockchain: um novo bloco estende uma das bifurcações.

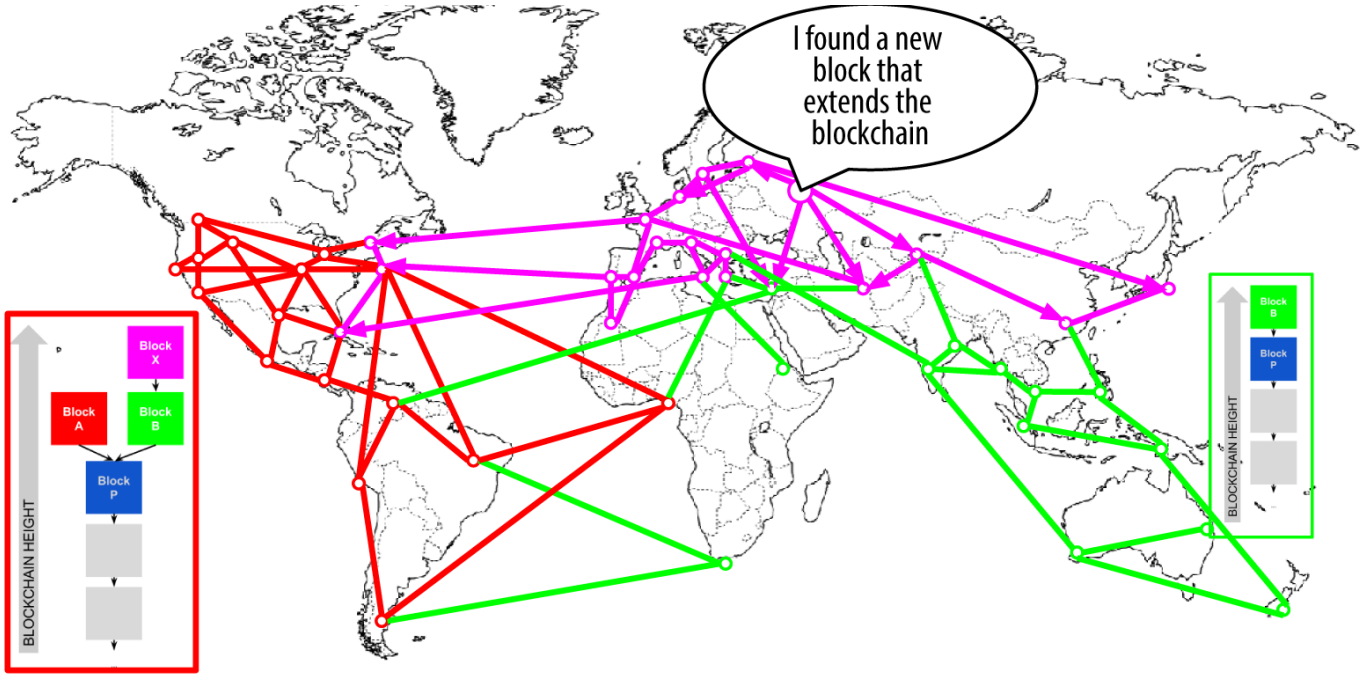


Figure 5. Visualização de um evento de bifurcação da blockchain: um novo bloco estende uma das bifurcações

Todos os nodos que escolheram o "verde" como vencedor na rodada anterior irão simplesmente estender a corrente em mais um bloco. Os nodos que escolheram o "vermelho" como vencedor, entretanto, enxergarão agora duas correntes: azul-verde-rosa e azul-vermelho. A corrente azul-verde-

rosa agora é maior (mais dificuldade acumulada) do que a corrente azul-vermelho. Como resultado, aqueles nodos irão definir a corrente azul-verde-rosa como a corrente principal e passarão a considrar a corrente azul-vermelha como sendo a corrente secundária, como demonstrado em Visualização de um evento de bifurcação da blockchain: a rede reconverge em uma nova cadeia mais longa. Isso se chama de reconvergência da corrente, pois aqueles nodos são forçados a revisar sua visão da blockchain para incorporar a nova evidência de uma corrente mais longa. Qualquer minerador que estiver trabalhando em estender a corrente azul-vermelho terá que parar o trabalho porque o seu bloco candidato é um "órfão", já que o seu pai "vermelho" não está mais na corrente mais longa. As transações contidas no "vermelho" são enfileiradas novamente para processar o próximo bloco, pois aquele bloco não está mais na corrente principal. Toda a rede reconverge para uma única blockchain azul-verde-rosa, com o "rosa" sendo o último bloco na corrente. Todos os mineradores imediatamente começam a trabalhar nos blocos candidatos que referenciam o "rosa" como seu pai, para estender a corrente azul-verde-rosa.

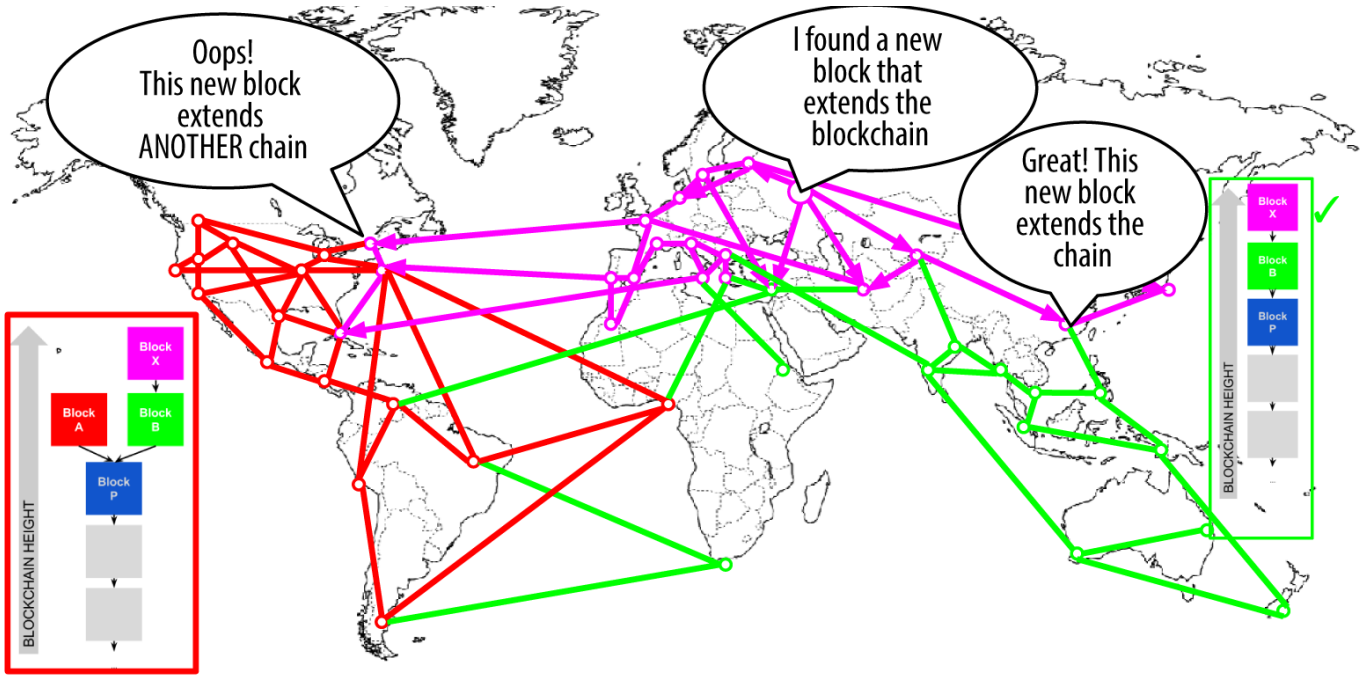


Figure 6. Visualização de um evento de bifurcação da blockchain: a rede reconverge em uma nova cadeia mais longa

É teoricamente possivel que uma bifurcação se estenda em dois blocos, se os dois blocos forem encontrados quase que simultaneamente por mineradores nos "lados" opostos da bifurcação anterior. Entretanto, a chance disso acontecer é muito baixa. Enquanto uma bifurcação de um bloco pode ocorrer semanalmente, uma bifucarcação de dois blocos é extremamente rara.

O intervalo de bloco de 10 minutos do bitcoin é um ajuste equilibrado entre tempos de confirmação rápidos (liquidação das transações) e a probabilidade de uma bifurcação. Um tempo de bloco mais rápido tornaria faria com que as transações fossem liquidadas mais rapidamente, mas isso causaria bifurcações da blockchain mais frequentes, enquanto um tempo de bloco mais lento iria diminuir o número de bifurcações, mas faria com que as transações fossem liquidadas mais lentamente.

Mineração e a Corrida do Hashing

A mineração do bitcoin é uma indústria extremamente competitiva. O poder de hashing aumentou exponencialmente a cada ano de existência do bitcoin. Em alguns anos o aumento do poder de hashing foi resultado de uma mudança completa na tecnologia de mineração, como em 2010 e 2011, quando muitos mineradores deixaram de usar a mineração com CPU (processadores comuns) e passaram a usar mineração com GPU (placas de vídeo) e mineração com arranjo de portas programável em campo (FPGA). Em 2013, a introdução da mineração com circuitos integrados de aplicação especifica (ASIC) foi mais um salto gigante no poder de mineração, ao colocar a função SHA256 diretamente no interior de chips de silício construídos especialmente para o propósito de mineração. O primeiro desses chips era capaz de fornecer, em uma pequena caixinha, um poder de mineração maior do que o poder que toda a rede bitcoin tinha em 2010.

A lista a seguir mostra o poder de hashing total da rede bitcoin, ao longo dos primeiros cinco anos de operação:

2009

0,5 MH/s–8 MH/s (crescimento de 16x)

2010

8 MH/s–116 GH/s (crescimento de 14.500x)

2011

16 GH/s–9 TH/s (crescimento de 562x)

2012

9 TH/s–23 TH/s (crescimento de 2.5x)

2013

23 TH/s–10 PH/s (crescimento de 450x)

2014

10 PH/s–150 PH/s em Agosto (crescimento de 15x)

No diagrama em, nós vemos que o poder de hashing da rede bitcoin aumento ao longo dos últimos dois anos. Como você pode ver, a competição entre os mineradores e o crescimento do bitcoin resultou em um aumento exponencial no poder de hashing (total de hashes por segundo ao longo da rede).

Figure 7. Poder de hashing total, em gigahashes por segundo, ao longo de dois anos

À medida que a quantidade de poder de hashing aplicado à mineração do bitcoin explodiu, a dificuldade cresceu para acompanhá-la. A métrica de dificuldade no gráfico mostrado em Métrica de dificuldade de mineração do Bitcoin, ao longo de dois anos é medida como a razão da dificuldade atual pela dificuldade mínima (a dificuldade do primeiro bloco).

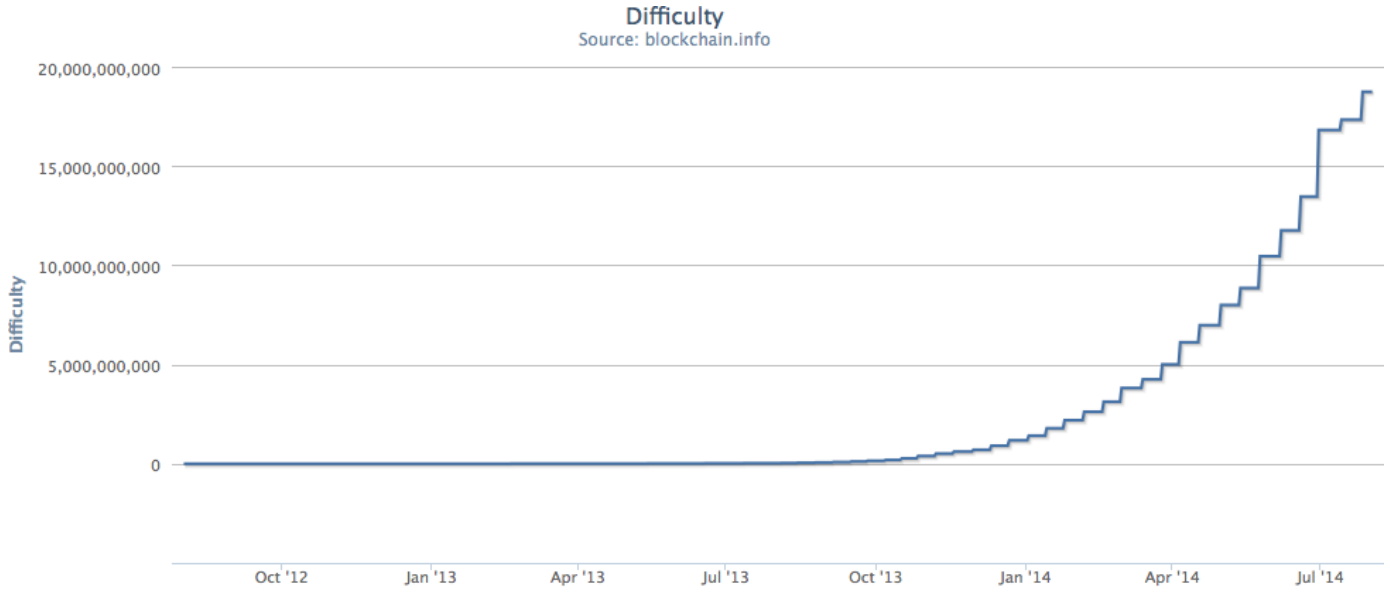


Figure 8. Métrica de dificuldade de mineração do Bitcoin, ao longo de dois anos

Nos últimos dois anos, os chips de mineração ASIC se tornaram cada vez mais densos, se aproximando da tecnologia de ponta na fabricação em silício, com um tamanho (resolução) de 22 nanômetros (nm). Atualmente, os fabricantes de ASIC estão tentando superar os fabricantes de chips de CPU, projetando chips com 16 nm, pois a rentabilidade dos mineradores está fazendo com que essa indústria se torne até mais rápida do que a indústria de informática. Não existem mais passos gigantes para se dar na mineração em bitcoin, pois a indústria atingiu o limite da Lei de Moore, que estipula que a capacidade de computação irá dobrar a cada 18 meses. Entretanto, o poder de mineração da rede continua a avançar em um ritmo exponencial, à medida que a corrida por chips de maior densidade é combinada

com uma corrida por data center de maior densidade, onde milhares desses chips podem ser empregados. Agora não é mais uma questão de quanta mineração pode ser feita em um chip, mas quantos chips podem ser espremidos em um prédio, enquanto o prédio ainda é capaz de dissipar o calor e fornecer energia adequada.

A Solução do Nonce Extra

Desde 2012, a mineração do bitcoin evoluiu para resolver uma limitação fundamental na estrutura do cabeçalho do bloco. No início do bitcoin, um minerador podia encontrar um bloco ao ficar tentando descobrir o nonce até que o hash resultante estivesse abaixo do alvo. À medida que a dificuldade cresceu, os mineradores frequentemente ciclavam por todos os 4 bilhões de valores do nonce sem encontrar um bloco. Entretanto, isso foi facilmente resolvido ao se atualizar a estampa de tempo do bloco para contabilizar pelo tempo decorrido. Como a estampa de tempo faz parte do cabeçalho do bloco, essa mudança permitiu que os mineradores fizessem suas buscas pelo valor do nonce novamente com resultados diferentes. No entanto, quando o hardware de mineração excedeu 4 GH/s, essa abordagem se tornou cada vez mais difícil, pois os valores do nonce se esgotavam em menos de um segundo. À medida que os equipamentos de mineração ASIC começaram a ampliar e exceder a taxa de hashes (TH/s), o software de mineração precisou de mais espaço para os valores de nonce. A estampa de tempo podia ser estendida um pouco, mas ficar empurrando-a muito para o futuro iria fazer com que o bloco se tornasse inválido. O cabeçalho do bloco precisava de um novo lugar nele para ser usado como fonte para os valores de nonce. A solução foi usar a transação coinbase como uma fonte para valores extra de nonce. Como o script da coinbase pode armazenar entre 2 e 100 bytes de dados, os mineradores começaram a usar esse espaço como espaço extra para o nonce, o que permitiu que eles explorassem uma faixa muito maior de valores de cabeçalhos de blocos, para encontrar blocos válidos. A transação coinbase é incluída na árvore de merkle, ou seja, qualquer mudança no script coinbase faz com que a raiz de merkle mude. Os 8 bytes extras do nonce, mais os 4 bytes do nonce "padrão" permitem aos mineradores explorarem um total de 2^{96} (8 seguidos de 28 zeros) possibilidades *por segundo* sem ter que modificar a estampa de tempo. Se, no futuro, os mineradores conseguirem ciclar através de todas essas possibilidades, eles poderão então modificar a estampa de tempo. Também há mais espaço no script da coinbase para futuras expansões do espaço extra de nonce.

Pools de Mineração

Nesse ambiente altamente competitivo, os mineradores individuais trabalhando sozinho (também conhecido como mineradores solo) não tem chance de vencer. A probabilidade de um deles encontrar um bloco é tão baixa, que não compensa os custos que eles terão com eletricidade e hardware, ou seja, acaba virando uma aposta, como ganhar numa loteria. Nem mesmo o sistema de mineração ASIC mais rápido é capaz de competir com os sistemas comerciais que empilham dezenas de milhares desses chips em galpões próximos de estações de energia hidrelétrica. Os mineradores agora colaboram para formar pools de mineração, agrupando o seu pode de hashing e compartilhando as recompensas entre os milhares de participantes da pool. Ao participar de uma pool, os mineradores recebem uma pequena cota da recompensa total, mas tipicamente são recompensados diariamente, reduzindo as incertezas.

Vamos dar uma olhada em um exemplo específico. Assuma que um minerador comprou hardware de mineração com uma taxa de hashing combina de 6.000 gigahashes por segundo (GH/s), ou 6 TH/s. Em agosto de 2014, esse equipamento custava aproximadamente 10.000 dólares. O hardware consome 3 kilowatts (kW) de eletricidade quando está ligado, 72 kw-horas por dia, a um custo de 7 ou 8 dólares por dia, em média. Na dificuldade atual do bitcoin, o minerador será capaz de minerar sozinho um bloco aproximadamente a cada 155 dias, ou a cada 5 meses. Se o minerador encontrar um único bloco nesse período de tempo, a recompensa de 25 bitcoins, com a taxa de câmbio de 600 dólares por bitcoin, iria resultar em um pagamento único de 15.000 dólares, que irá cobrir todos os custos do hardware e da eletricidade consumidos durante todo o período de tempo, deixando um rendimento líquido de cerca de 3.000 dólares. Entretanto, a chance de encontrar um bloco em um período de cinco meses depende da sorte do minerador. Ele pode encontrar dois blocos em cinco meses e ter um lucro muito grande. Ou ele pode não encontrar nenhum bloco em 10 meses e ter prejuízo financeiro. Pior ainda, se a taxa de crescimento atual do poder de hashing se mantiver, a dificuldade do algoritmo de prova-de-trabalho do bitcoin provavelmente irá aumentar significativamente nesse período, ou seja, o minerador terá, no máximo, seis meses para equilibrar seus gastos com seu lucro, antes que o hardware esteja efetivamente obsoleto e tenha que ser substituído pode um hardware de mineração mais poderoso. Se esse minerador participar de um pool de mineração, ao invés de aguardar por uma chance única a cada cinco meses de ganhar 15.000 dólares, ele será capaz de ganhar aproximadamente 500 a 750 dólares por semana. Os pagamentos regulares do pool de mineração o ajudarão a amortizar os custos do harware e da eletricidade ao longo do tempo, sem fazer com que ele assuma um risco enorme. O harware ainda estará obsoleto em seis a nove meses, e o risco ainda será alto, mas pelo menos o rendimento será regular e confiável naquele período.

Os pools de mineração coordenam centenas a milhares de mineradores, através de protocolos especializados de mineração em pool. Cada minerador configura o seu equipamento de mineração para se conectar ao servidor do pool, após criar uma conta no pool. O hardware de mineração do minerador permanece conectado ao servidor do pool enquanto está minerando, sincronizando seus esforços com os dos outros mineradores. Portanto, os mineradores do pool compartilham o esforço necessário para minerar um bloco, e, então, comparilham as recompensas da mineração.

Os blocos com sucesso pagam a recompensa para um endereço bitcoin da pool, ao invés de pagar individualmente para os mineradores. O servidor do pool fará pagamentos periodicamente para os endereços bitcoin dos mineradores, assim que a sua cota da recompensa atingir um valor mínimo. Tipicamente, o servidor do pool cobra uma taxa percentual das recompensas por fornecer o serviço de mineração em pool.

Os mineradores que paricipam de uma pool dividem o trabalho de buscar uma solução para um bloco candidato, cada um deles ganhando "cotas" pela sua contribuição com a mineração. O pool de mineração define um alvo de dificuldade menor para ganhar uma cota, tipicamente mais do que 1.000 vezes mais fácil que a dificuldade da rede bitcoin. Quando alguém do pool minera um bloco com sucesso, a recompensa é destinada ao pool e então é redistribuída para todos os mineradores, de acordo com o número de cotas que cada um deles contribuiu para o esforço coletivo.

Qualquer minerador pode participar das pools, grande ou pequeno, amador ou profissional. Uma pool portanto irá tem alguns participantes com uma única pequena máquina de mineração, e outros com

uma garagem repleta de hardware de mineração de última geração. Alguns irão minerar com algumas dezenas de kilowatts de eletricidade, outros irão executar um data center consumindo um megawatt de energia. Como que a pool de mineração quantifica as contribuições individuais, para que as recompensas sejam distribuídas de maneira justa, sem a possibilidade de trapaças? A solução é usar o algoritmo de prova-de-trabalho do bitcoin para medir a contribuição de cada minerador da pool, mas definir uma dificuldade menor, de maneira que até mesmo os menores mineradores da pool ganhem uma cota com frequência suficiente para que valha a pena contribuir para a pool. Ao definir uma dificuldade menor para se ganhar as cotas, a pool mede a quantidade de trabalho que é feita por cada minerador. Cada vez que um minerador da pool encontra um hash de cabeçalho de bloco que é menor do que a dificuldade da pool, ele comprova que ele fez o trabalho de hashing para encontrar esse resultado. Mais importante, o trabalho para encontrar as cotas contribui, de uma maneira que pode ser mensurada estatisticamente, para o esforço de toda a pool para encontrar um hash menor do que o alvo da rede bitcoin. Milhares de mineradores tentando encontrar hashes de valor baixo uma hora acabarão encontrado um baixo o suficiente que satisfaça o alvo da rede bitcoin.

Vamos voltar à analogia do jogo de dados. Se os jogadores estiverem lançando os dados com o objetivo de tirar menos do que quatro (a dificuldade geral da rede), uma pool definiria um alvo mais fácil, contanto quantas vezes os jogadores da pool conseguiram tirar menos do que oito. Quando os jogadores da pool tirarem menos que oito (o alvo da cota da pool), eles ganham cotas, mas eles não ganham o jogo porque eles não atingiram o alvo do jogo (menos do que quatro). Os jogadores da pool irão atingir o alvo mais fácil da pool muito mais frequentemente, frequentemente ganhando muitas cotas, mesmo que eles não atinjam o alvo mais difícil de se ganhar o jogo. De vez em quando, um dos jogadores da pool irá lançar dois dados combinados e irá tirar menos do que quatro, fazendo com que a pool vença. Então, os ganhos podem ser distribuídos para os jogadores da pool baseando-se nas cotas que eles ganharam. Apesar de o alvo de oito ou menos não ser uma vitória, ele é uma maneira justa de se medir os lançamentos de dados dos jogadores, e isso ocasionalmente produz um lançamento menor do que quatro.

De maneira semelhante, um pool de mineração irá definir uma dificuldade do pool que irá garantir que um minerador individual do pool possa achar, com frequência, os hashes dos cabeçalhos de bloco que são menores do que a dificuldade do pool, ganhando cotas. De vez em quando, uma dessas tentativas irá produzir um hash de cabeçalho de bloco que é menor do que o alvo da rede bitcoin, tornando-o um bloco válido, e toda a pool ganhará.

Pools gerenciadas

A maioria das pools de mineração são "gerenciadas", ou seja, existe uma companhia ou indivíduo executando um servidor de pool. O dono do servidor da pool é chamado de *operador da pool*, e ele cobra dos mineradores uma taxa percentual dos lucros.

O servidor da pool executa software especializado e um protocolo de mineração em pool que coordena as atividades dos mineradores da pool. O servidor da pool é também conectado a um ou mais nodos completos de bitcoin e tem acesso direto à cópia completa do banco de dados da blockchain. Isso permite que o servidor da pool valide blocos e transações em nome dos mineradores da pool, aliviando-os do fardo de se executar um nodo completo. Para os mineradores da pool, isso é uma consideração importante, pois um nodo completo exige um computador dedicado com pelo menos 15 a

20 GB de armazenamento permanente (disco rígido) e pelo menos 2 GB de memória (RAM). ALém disso, o software bitcoin sendo executado no nodo completo precisa ser monitorado, mantido e atualizado frequentemente. Qualquer período de inatividade causado por uma falta de manutenção ou falta de recursos irá comprometer os rendimentos do minerador. Para muitos mineradores, a habilidade de minerar sem ter que executar um nodo completo é outro grande benefício de se associar a uma pool gerenciada.

Os mineradores da pool se conectam ao servidor da pool usando um protocolo de mineração como o Stratum (STM) ou o GetBlockTemplate (GBT). Um padrão mais antigo chamado GetWork (GWK) já é obsoleto desde 2012, porque ele não suporta de maneira fácil a mineração em taxas de hash maiores que 4 GH/s. Tanto o protocolo STM quanto o GBT criam *modelos* de blocos que contém um modelo de um cabeçalho de bloco candidato. O servidor da pool constrói um bloco candidato ao agregar as transações, acrescentando uma transação coinbase (com espaço extra de nonce), calculando a árvore de merkle e vinculando ao hash do bloco anterior. O cabeçalho do bloco candidato é enviado para cada um dos mineradores da pool como um modelo. Cada minerador da pool minera o bloco modelo, em uma dificuldade menor do que a dificuldade da rede bitcoin, e envia quaisquer resultados de sucesso de volta ao servidor do pool para ganhar cotas.

P2Pool

As pools gerenciadas criam a possibilidade de se trapacear o operador da pool, que pode direcionar os esforços da pool para transações de duplo-gasto ou blocos inválidos (ver Ataques de Consenso). Além disso, os servidores de pool centralizados representam um ponto único de falha. Se o servidor da pool ficar inativo ou for lentificado por ataques de negação de serviço (DoS), os mineradores da pool não conseguem minerar. Em 2011, para resolver esses problemas causados pela centralização, um novo método de mineração em pool foi proposto e implementado: a P2Pool é uma pool de mineração ponto-a-ponto, sem um operador central.

A P2Pool funciona ao descentralizar as funções do servidor de pool, implementando um sistema paralelo parecido com a blockchain chamado de *corrente de cotas* ("*share chain*"). Uma corrente de cotas é uma blockchain que é executada em uma dificuldade menor do que a blockchain do bitcoin. A corrente de cotas permite que os mineradores da pool colaborem em uma pool descentralizada, ao minerar as cotas na corrente de cotas em uma taxa de um bloco de cotas a cada 30 segundos. Cada um dos blocos na corrente de cotas registra uma cota proporcional de recompensa para os mineradores da pool que contribuírem com o trabalho, recebendo as cotas do bloco de cotas anterior. Quando um dos blocos de cotas também atinge o alvo de dificuldade da rede bitcoin, ele é propagado e incluído na rede bitcoin, recompensando todos os mineradores da pool que contribuíram com todas as cotas que precederam o bloco de cotas vencedor. Essencialmente, ao invés de ter um servidor de pool mantendo um registro das cotas e recompensas dos mineradores da pool, a corrente de cotas permite que todos os mineradores da pool mantenham um registro de todas as cotas utilizando um mecanismo de consenso descentralizado, semelhante ao mecanismo de consenso descentralizado do bitcoin.

A mineração em P2Pool é mais complexa do que a mineração em pool porque ela exige que os mineradores da pool executem um computador dedicado com espaço em disco, memória e banda de internet suficientes para suportarem um nodo completo de bitcoin e o software do nodo da P2Pool. Os mineradores P2Pool conectam o seu hardware de mineração ao seu nodo local de P2Pool, que simula

as funções de um servidor pool ao enviar modelos de bloco para o hardware de mineração. Os mineradores do P2Pool constróem seus próprios blocos candidatos, agregando as transações da mesma maneira que os mineradores solo fazem, mas então ele mineram colaborativamente na corrente compartilhada. O P2Pool é uma abordagem híbrida que tem a vantagem de ter pagamentos muito mais detalhados do que a mineração solo, mas sem dar muito controle ao operador da pool, como acontece nas pools gerenciadas.

Recentemente, a participação em P2Pool aumentou significativamente, à medida que a concentração de mineração em pools de mineração atingiram níveis que geraram um temor de um ataque de 51% (ver Ataques de Consenso). A continuação do desenvolvimento do protocolo P2Pool mantém a expectativa de remover a necessidade de se executar um nodo completo e, portanto, tornando a mineração descentralizada ainda mais fácil de se utilizar.

Apesar de a P2Pool reduzir a concentração de poder dos operadores das pools de mineração, ela é vulnerável a ataques de 51% contra a própria cota da corrente. Uma adoção muito maior da P2Pool não resolve o risco que o bitcoin tem de sofrer um ataque de 51%. No entanto, a P2Pool torna o bitcoin mais robusto de maneira geral, como parte de um ecossistema de mineração diversificado.

Ataques de Consenso

O mecanismo de consenso do bitcoin é, pelo menos teoricamente, vulnerável a ataques de mineradores (ou pools) que tentarem usar o seu poder de hashing com finalidades desonestas ou destrutivas. Como nós já vimos, o mecanismo de consenso depende de se ter uma maioria de mineradores agindo honestamente por interesse próprio. Entretanto, se um minerador ou um grupo de mineradores conseguir atingir uma cota significativa de poder de mineração, eles podem atacar o mecanismo de consenso, de maneira que comprometam a segurança e a disponibilidade da rede bitcoin.

É importante citar que os ataques de consenso só podem afetar consensos futuros, ou, no máximo, os mais recentes (dezenas de blocos). O registro do bitcoin se torna mais e mais imutável à medida que o tempo passo. Apesar de que, em teoria, uma bifurcação pode ocorrer em qualquer profundidade, na prática, o poder computacional necessário para se forçar uma bifurcação muito profunda é imenso, tornando os blocos antigos praticamente imutáveis. Os ataques de consenso também não afetam a segurança das chaves privadas e do algoritmo de assinatura (ECDSA). Um ataque de consenso não é capaz de roubar bitcoin, gastar bitcoins sem assinaturas, redirecionar bitcoin ou modificar transações antigas ou registros de posse. Os ataques de consenso só são capazes de afetar os b locos mais recentes e causar distúrbios de negação de serviço na criação de blocos futuros.

Um cenário de ataque contra o mecanismo de consenso é chamado de "ataque de 51%". Nesse cenário, um grupo de mineradores, controlando uma maioria (51%) do poder de hashing total da rede, conspira para atacar o bitcoin. Com a habilidade de minerar a maioria dos blocos, os mineradores atacantes podem gerar "bifurcações" deliberadas na blockchain, gerar transações de duplo gasto ou executar ataques de negação de serviço (DoS) contra endereços ou transações específicas. Um ataque de bifurcação/duplo gasto é um ataque onde o atacante faz com que blocos já confirmados sejam invalidados ao fazer uma bifurcação em um nível abaixo deles, com uma posterior reconvergência em uma corrente alternativa. Com poder suficiente, um atacante pode invalidar seis ou mais blocos em

uma sequência, invalidando transações que antes eram consideradas imutáveis (com seis confirmações). Note que o duplo gasto só pode ser feito nas transações do próprio atacante, para as quais o atacante pode produzir uma assinatura válida. Fazer um duplo gasto da própria transação é rentável quando, ao invalidar uma transação, o atacante puder receber um pagamento irreversível ou um produto sem ter que pagar por isso.

Vamos examinar um exemplo prático de um ataque de 51%. No primeiro capítulo, nós analisamos uma transação entre Alice e Bob por uma xícara de café. Bob, o dono da cafeteria, está disposto a aceitar pagamentos pelas suas xícaras de café sem esperar por uma confirmação (a inclusão da transação em um bloco minerado), porque o risco de um duplo gasto com uma xícara de café é baixo se comparado com a conveniência de se oferecer um serviço rápido ao consumidor. Isso é semelhante à prática das cafeterias que aceitam pagamentos de cartão de crédito sem uma assinatura do cliente em valores abaixo de 25 dólares, porque o risco de um reembolso de um pagamento com cartão de crédito é baixo, enquanto o custo de atrasar um transação para se obter uma assinatura é comparativamente maior. Em contraste, vender um item mais caro por bitcoins aumenta o risco de um ataque de duplo gasto, onde o comprador transmite uma transação competidora que gasta os mesmos inputs (UTXO) e cancela o pagamento do comerciante. Um ataque de duplo gasto pode ocorrer de duas formas: ou antes de uma transação ser confirmada, ou se o ataque utilizar uma bifurcação da blockchain para desfazer vários blocos. Um ataque de 51% permite que os atacantes façam duplos gastos de suas próprias transações em uma nova corrente, portanto desfazendo a transação correspondente na corrente antiga.

Em nosso exemplo, Mallory, um atacante malicioso, vai até a galeria da Carol e compra um lindo tríptico de pinturas retratando o Satoshi Nakatomo como Prometeus. A Carol vende as pinturas "O Grande Fogo" para o Mallory por 250.000 dólares em bitcoin. Ao invés de aguardar por seis ou mais confirmações da transação, a Carol embala e entrega as pinturas para o Mallory após uma única confirmação ter ocorrido. O Mallory tem um cúmplice, o Paul, que opera uma grande pool de mineração, e o seu cúmplice faz um ataque de 51% assim que a transação do Mallory é incluída em um bloco. O Paul direciona a pool de mineração para reminerar a mesma altura de bloco em que está o bloco contendo a transação do Mallory, substituindo o pagamento do Mallory para a Carol por uma transação que faz um duplo gasto do mesmo input do pagamento do Mallory. A transação de duplo gasto consome o mesmo UTXO e o paga de volta para a carteira do Mallory, ao invés de pagar para a Carol, essencialmente permitindo que o Mallory permaneça com os seus bitcoins. O Paul então direciona a pool de mineração para minerar um bloco adicional, para tornar a corrente contendo a transação de duplo gasto maior do que a corrente original (causando uma bifurcação abaixo do bloco contendo a transação do Mallory). Quando a bifurcação da blockchain se resolver em favor da nova corrente (mais longa), a transação de duplo gasto substituirá o pagamento original à Carol. A Carol agora está sem as suas três pinturas e também não tem nenhum pagamento bitcoin. Durante toda essa atividade criminosa, os participantes da pool de mineração do Paul permanecem sem estar cientes da tentativa de duplo gasto, pois a mineração que eles fazem é automatizada e eles não conseguem monitorar todas as transações ou blocos.

Para proteger contra esse tipo de ataque, um comerciante vendendo itens de alto valor deve esperar por pelo menos seis confirmações antes de entregar o produto ao comprador. Outra alternativa seria o comerciante usar uma conta de múltiplas assinaturas como caução, novamente esperando por várias confirmações após que a conta caução receba os fundos. Quanto mais confirmações tiver a transação,

mais difícil será invalidá-la com um ataque de 51%. Para itens de alto valor, o pagamento com bitcoin ainda será conveniente e eficiente mesmo que o comprador tenha que aguardar por 24 horas para a entrega, o que corresponderia a aproximadamente 144 confirmações.

Além de um ataque de duplo-gasto, o outro cenário para um ataque de consenso é a negação de serviço para participantes bitcoin específicos (endereços bitcoin específicos). Um atacante com uma maioria de poder de mineração pode simplesmente ignorar transações específicas. Se elas forem incluídas em um bloco minerado por outro minerador, o atacante pode deliberadamente fazer uma bifurcação e reminerar aquele bloco, novamente excluindo as transações específicas. Esse tipo de ataque pode resultar em uma negação de serviço sustentada contra um endereço ou conjunto de endereços específicos, que se manterá enquanto o atacante controlar a maioria do poder de mineração.

Apesar do nome, o cenário de uma ataque de 51% não exige um poder de hashing de 51%. De fato, esse ataque pode ser tentado com uma pequena porcentagem de poder de hashing. O limite de 51% é simplesmente o nível em que esse tipo de ataque tem sucesso quase garantido. Um ataque de consenso é basicamente um cabo-de-guerra para o próximo bloco e o grupo "mais forte" tem maior probabilidade de vencer. Com menos poder de hashing, a probabilidade de sucesso é reduzida, pois outros mineradores controlam a geração de alguns blocos com o seu poder "honesto" de mineração. Uma maneira de se olhar isso é que, quanto mais poder de hashing um atacante tem, mais longa será a ramificação da bifurcação que ele poderá criar deliberadamente, mais blocos no passado recente ele poderá invalidar ou mais blocos no futuro ele poderá controlar. Grupos de pesquisa de segurança usaram modelos estatísticos para alegar que vários tipos de ataques de consenso já são possíveis a partir de apenas 30% de poder de hashing.

O aumento maciço no poder de hashing total tornou o bitcoin impermeável a ataques feitos por um minerador isolado. Não há mais como um minerador solo controlar mais do que uma pequena percentagem do poder total de mineração. Entretanto, a centralização do controle causada pelas pools de mineração trouxe o risco da existência de ataques lucrativos sendo feitos pelo operador do pool de mineração. O operador do pool de uma pool gerenciada controla a construção dos blocos candidatos e também controla as transações que são incluídas nos blocos. Isso dá ao operador da pool o poder de excluir transações ou acrescentar transações de gasto duplo. Se esse abuso de poder for feito de uma maneira limitada e sutil, um operador de pool seria capaz de lucrar com um ataque de consenso, sem ser descoberto.

Entretanto, nem todos os atacantes são motivados pelo lucro. Um cenário potencial de ataque seria um onde o atacante pretende derrubar a rede bitcoin sem que ele tenha a possibilidade de lucrar com esse ataque. Um ataque malicioso que tenha o objetivo de destruir o bitcoin iria exigir um investimento enorme e um planejamento secreto, mas poderia ser lançado por um atacante com muitos recursos financeiros, muito provavelmente patrocinado por um governo. Alternativamente, um atacante com muitos recursos financeiros poderia atacar o consenso do bitcoin ao acumular simultaneamente hardware de mineração, comprometendo operadores de pool e atacando outras pools com negação de serviço. Todos esses cenários são teoricamente possíveis, mas se tornam cada vez menos praticáveis à medida que o poder de hashing total da rede bitcoin continua a crescer exponencialmente.

Sem dúvida, um ataque de consenso grave poderia arruinar a confiância no bitcoin a curto prazo, possivelmente causando uma queda significativa no preço. No entanto, a rede e o software bitcoin

estão evoluindo constantemente, e a comunidade bitcoin iria tomar medidas contrárias aos ataques de consenso, tornando o bitcoin mais fortalecido, protegido e robusto do que nunca.

Cadeias, Moedas e Aplicações Alternativas

Bitcoin foi o resultado de 20 anos de pesquisa em sistemas distribuídos e moedas, trazendo uma nova tecnologia revolucionária: o mecanismo de consenso descentralizado baseado em prova de trabalho. Essa invenção no núcleo do bitcoin marcou o início de uma onda de inovações em moedas, serviços financeiros, economia, sistemas distribuídos, sistemas de votação, administração corporativa e contratos.

Neste capítulo iremos examinar muitos ramos das invenções do bitcoin e do blockchain: as cadeias alternativas, moedas e aplicações construídas desde a introdução dessa tecnologia em 2009. Principalmente iremos olhar para as moedas alternativas, ou *alt coins*, que são moedas digitais implementadas que usam o mesmo padrão de design que o bitcoin, mas com uma blockchain e uma rede completamente separadas.

Para cada alt coin mencionada nesse capítulo, 50 ou mais não serão mencionadas, provocando gritos de raiva em seus criadores e fãs. A proposta desse capítulo não é avaliar ou qualificar as alt coins, ou mesmo mencionar as mais significativas baseando-se em alguma avaliação subjetiva. Ao invés disso, nós iremos destacar alguns exemplos que mostram a amplitude e variedade do ecosistema, mostrando a primeira de cada tipo de inovação ou diferenciação significativa. Alguns dos exemplos mais interessantes de alt coins são de fato fracassos completos do ponto de vista monetário. Isso talvez as torne ainda mais interessantes de se estudar e destaca o fato que esse capítulo não é para ser usado como um guia de investimento.

Com novas moedas sendo lançadas a cada dia, seria impossível não deixar de falar de alguma moeda importante, talvez uma que mude a história. A taxa de inovação é o que faz com que esse espaço seja tão empolgante e garante que esse capítulo já será incompleto e desatualizado assim que ele for publicado.

Uma Taxonomia de Moedas e Cadeias Alternativas

O Bitcoin é um projeto de código-aberto, e seu código tem sido usado como a base para muitos outros projetos de software. A forma mais comum de software gerado a partir do código-fonte do bitcoin são moedas descentralizadas alternativas, ou *alt coins*, que usam os mesmos blocos de construção básicos para implementar as moedas digitais.

Existem várias camadas de protocolos implementadas sobre a blockchain do bitcoin. Essas *meta coins*, *meta chains* ou *apps de blockchain* usam a blockchain como uma plataforma de aplicações ou estendem o protocolo bitcoin ao adicionar camadas de protocolos. Alguns exemplos incluem Colored Coins, Mastercoin, NXT e Counterparty.

Na próxima seção, examinaremos algumas alt coins notáveis, como a Litecoin, Dogecoin, Freicoin, Primecoin, Peercoin, Darkcoin e Zerocoin. Essas alt coins são notáveis por questões históricas ou porque elas são bons exemplos para um tipo específico de inovação de alt coin, e não porque elas são mais valiosas ou porque são as "melhores" alt coins.

Além das alt coins, existem várias implementações alternativas da blockchain que não são "moedas", que eu as chamo de *alt chains*. Essas alt chains implementam um algoritmo de consenso e um registro distribuído como uma plataforma para contratos, registro de nomes e outras aplicações. As alt chains usam os mesmos blocos básicos de construção e às vezes também usam uma moeda ou um token como mecanismo de pagamento, mas sua proposta primária não é ser moeda. Nós iremos apresender sobre a Namecoin e o Ethereum como exemplos de alt chains.

Por fim, existem vários aspirantes a bitcoin que oferecem moeda digital ou redes de pagamento digital, mas sem usar um registro descentralizada ou mecanismo de consenso baseado em prova de trabalho, como o Ripple e outros. Essas tecnologias não-blockchain estão fora do âmbito desse livro e não serão tratadas nesse capítulo.

Plataformas Meta Coin

Plataformas meta coin e meta chains são camadas de software implementados no topo do bitcoin, seja implementando uma moeda-dentro-de-uma-moeda, ou uma sobrecamada de plataforma/protocolo dentro do sistema bitcoin. Essas camadas funcionais estendem o protocolo principal do bitcoin e adiciona funcionalidades e capacidade de codificar dados adicionais dentro das transações e endereços de bitcoin. As primeiras implementações de meta coins (meta-moedas) usaram vários hacks para adicionar metadados à blockchain do bitcoin, tais como usar um endereço bitcoin para codificar dados ou utilizar campos de transações vazios (por ex: o campo sequência de transação) para codificar metadados sobre a camada de protocolo adicional. Já que a introdução do opcode do script de transação OP_RETURN, as meta moedas são capazes de gravar metadados mais diretamente na blockchain, e a maioria está migrando para esse uso.

Colored Coins

Colored coins ou *Moedas Coloridas* é um meta protocolo que superpõe a informação de pequenas quantidades de bitcoin. Uma moeda "colorida" é uma quantidade de bitcoin que teve seu propósito redefinido para expressar outro ativo. Imagine, por exemplo, pegar uma nota de R$1 e colocar um selo que diz "Esse é 1 certificado de ações de uma ação da Acme Ltda." Agora a nota de R$1 serve para dois propósitos: É uma nota de dinheiro e também um certificado de ações. Como ela é mais valiosa como uma ação, você não vai querer usá-la para comprar bala, de forma que ela não é mais útil como dinheiro. Moedas coloridas funcionam da mesma forma, convertendo uma pequena e específica quantidade de bitcoin em um certificado negociável que representa outro ativo. O termo "color" ou "cor" refere à ideia de dar um significado especial através da adição de um atributo tal como uma cor — é uma metáfora, não uma associação a uma cor real. Não há cores nas moedas coloridas.

Moedas coloridas são gerenciadas por carteiras especializadas que gravam e interpretam os metadados anexados aos bitcoins coloridos. Usando tal carteira, o usuário irá converter uma quantidade de bitcoins comuns para moedas coloridas ao adicionar uma etiqueta que possui um significado especial. Por exemplo, a etiqueta poderia representar certificados de ações, cupons, propriedades reais, commodities ou tokens colecionáveis. Fica a critério do usuário de moedas coloridas designar e interpretar o significado da "cor" associada com moedas específicas. Para colorir as moedas, o usuário define o metadado associado, tal como o tipo de emissão, se pode ser subdividida

ou não. um símbolo e descrição ou outra informação relacionada. Uma vez colorida, essas moedas podem ser compradas e vendidas, subdivididas e agregadas, além de poder receber dividendos. As moedas coloridas também podem ser "descoloridas" ao se remover a associação especial e assim, ter seu valor novamente em bitcoin.

Para demonstrar o uso das colored coins, nós criamos um conjunto de 20 colored coins com o símbolo "MasterBTC" que representa cupons para uma cópia gratuite desse livro demonstrado em [example_9-1]. Cada unidade de MasterBTC, representada por essas colored coins, agora pode ser vendida ou doada para qualquer usuário bitcoin com uma carteira compatível com colored-coins, que agora poderá transferí-las para outros ou regastá-las com a editora para receber uma cópia gratuita desse livro. Esse exemplo das colored coins pode ser visto aqui.

1. O perfil de metadados das colored coins registrado como um cupom para uma cópia grátis do livro

```
{
  "source_addresses": [
    "3NpZmvSPLmN2cVFw1pY7gxEAVPCVfnWfVD"
  ],
  "contract_url":
"https://www.coinprism.info/asset/3NpZmvSPLmN2cVFw1pY7gxEAVPCVfnWfVD",
  "name_short": "MasterBTC",
  "name": "Free copy of \"Mastering Bitcoin\"",
  "issuer": "Andreas M. Antonopoulos",
  "description": "This token is redeemable for a free copy of the book \"Mastering
Bitcoin\"",
  "description_mime": "text/x-markdown; charset=UTF-8",
  "type": "Other",
  "divisibility": 0,
  "link_to_website": false,
  "icon_url": null,
  "image_url": null,
  "version": "1.0"
}
```

Mastercoin

Masteroin é uma camada de protocolo sobre o bitcoin que dá suporte a uma plataforma para várias aplicações que extendem o sistema bitcoin. Mastercoin usa a moeda MST como um token para conduzir transações Mastercoin, mas não é primariamente uma moeda. Ao invés disso, é uma plataforma para construir outras coisas, como moedas de usuário, tokens de propriedade smart, meios descentralizados de troca de ativos, e contratos. Pense na Mastercoin como um protocolo em camada de aplicação sobre a camada de transporte de transações financeiras do bitcoin, da mesma forma que HTTP roda por cima do TCP.

A Mastercoin operada primariamente através de transações enviadas de e para um endereço bitcoin especial chamado de endereço "exodus" 1EXoDusjGwvnjZUyKkxZ4UHEf77z6A5S4P), da mesma maneira que o HTTP usa uma porta TCP específica (porta 80) para diferenciar seu tráfego do resto do tráfego TCP. O protocolo Mastercoin está gradualmente se transformando, deixando de usar endereços endereços exodus especializados e multi-assinaturas para usar o operador bitcoin OP_RETURN para codificar os metadados de transação.

Counterparty

Counterparty é outra camada de protocolo implementada sobre o bitcoin. A Counterparty possibilita moedas de usuários, tokens negociáveis, instrumentos financeiros, transações descentralizadas de ativos e outras funcionalidades. A Counterpary é implementada primariamente usando o operador OP_RETURN na linguagem de script do bitcoin para registrar metadados que aprimoram as transações bitcoin com significados adicionais. A Counterparty usa a moeda XCP como um token para conduzir as transações Counterparty.

Alt Coins

A vasta maioria das alt coins são derivadas do código-fonte do bitcoin, também conhecidas como "forks" (bifurcações). Algumas são implementadas do zero, baseando-se no modelo blockchain, mas sem utilizar nenhum código-fonte do bitcoin. As alt coins e alt chains (leia mais na próxima seção) são implementações separadas da tecnologia blockchain e ambas as formas usam sua própria blockchain. Usamos termos diferentes para indicar que as alt coin são primariamente usadas como moeda, enquanto as alt chain são usadas para outras propostas, mas não como moeda primariamente.

Falando mais precisamente, o primeiro fork "alt" principal do código do bitcoin não foi uma alt coin, mas a alt chain *Namecoin*, que nós iremos discutir na próxima seção.

Baseando-se na data de divulgação, a primeira alt coin que foi uma bifurcação (fork) do bitcoin apareceu em agosto de 2011; ela foi chamada de *IXCoins*. A IXCoin modificou alguns parâmetros do bitcoin, acelerando a criação da moeda ao aumentar a recompensa para 96 moedas por bloco.

Em setembro de 2011, a *Tenebrix* foi lançada. A Tenebrix foi a primeira criptomoeda a implementar um algoritmo de prova-de-trabalho alternativo, um *scrypt* , um algoritmo originalmente projetado para reforçar senhas (resistência a ataques de força bruta). O objetivo da Tenebrix era fazer uma moeda que fosse resistente à mineração com GPUs e ASICs, ao usar um algoritmo com uso intenso de memória. A Tenebrix não teve sucesso como uma moeda, mas ela foi a base para a Litecoin, que obteve grande sucesso e disseminou centenas de clones.

A *Litecoin*, além de usar um script como algoritmo de prova-de-trabalho, também implementou um tempo de geração de blocos maior, com alvo de 2,5 minutos ao invés dos 10 minutos do bitcoin. A moeda resultante é aclamada como a "prata do bitcoin-ouro" e tem como objetivo ser uma moeda alternativa de peso leve (light-weight). Devido ao tempo de confirmação mais rápido e o limite de moedas total de 84 milhões, muitos adeptos do Litecoin acreditam que ela é mais adequada para operações de varejo do que o bitcoin.

As alt coins continuaram a proliferar-se em 2011 e 2012, baseadas em bitcoin ou no Litecoin. Em 2013, existiam 20 alt coins disputando por uma posição no mercado. No final de 2013, esse número explodiu para 200, com 2013 rapidamente se tornando o "ano das alt coins". O crescimento das alt coins continuou em 2014, com mais de 500 alt coins em existência no momento que esse livro foi escrito. Mais de metade das alt coins atuais são clones do Litecoin.

Criar uma alt coin é fácil, por isso existem agora mais de 500 delas. A maioria das alt coins diferem muito pouco do bitcoin e não oferecem nada que mereça um estudo. Muitas são de fato apenas tentativas de enriquecer seus criadores. Entre as cópias baratas e os esquemas de pump-and-dump (aumentar o valor da moeda e depois vender), existem, entretando, algumas exceções notáveis e inovações muito importantes. Essas alt coins usam abordagens radicalmente diferentes ou adicionam inovação significativa ao padrão de projeto do bitcoin. Existem três áreas principais nas quais essas alt coins se diferenciam do bitcoin:

- Política monetária diferente
- Mecanismo de consenso ou de prova de trabalho diferente
- Características específicas, como forte anonimidade

Para maiores informações, veja esse gráfico de linha do tempo das alt coins e alt chains.

Avaliando uma Alt Coin

Com tantas alt coins disponíveis, como alguém pode decidir quais são merecedoras de atenção? ALgumas alt coins tentam conseguir distribuição ampla e uso como moedas. Outras são laboratórios para experiências com recursos e modelos monetários diferentes. Muitas são apenas esquemas para enriquecimento rápido de seus criadores. Para avaliar alt coins, eu analiso suas características definidores e suas métricas de mercado.

Aqui estão algumas questões para se perguntar sobre o quanto uma alt coin se diferencia do bitcoin:

- A alt coin introduz uma inovação significativa?
- A diferença é estimuladora o suficiente para afastar os usuários do bitcoin?
- A alt coin se presta a um mercado de nicho ou aplicação interessantes?
- A alt coin conseguirá atrair mineradores suficientes para ser protegida contra ataques de consenso?

Aqui estão algumas métricas de mercado e financeiras vitais para serem consideradas:

- Qual é a capitalização de mercado total dessa alt coin?
- Quantos usuários/carteiras estima-se que a alt coin tenha?
- Quantos comerciantes aceitam a alt coin?
- Quantas transações diárias (volume) são executadas com a alt coin?
- Quanto valor é transferido diariamente?

Nesse capítulo, nós iremos nos concentrar primariamente nas características técnicas e no potencial de inovação das alt coins representadas pelo primeiro conjunto de questões.

Alternativas de Parâmetro Monetário: Litecoin, Dogecoin, Freicoin

O Bitcoin possui alguns parâmetros monetários que o fornecem características distintas de moeda de emissão-fixa deflacionária. Ele é limitado a 21 milhões de unidades principais (ou 21 quadrilhões de unidades menores), ele possui uma taxa de emissão com declínio em progressão geométrica e tem uma "frequência cardíaca" de um bloco a cada 10 minutos, que controla a velocidade de confirmação das transações e a geração da moeda. Muitas alt coins modificaram os parâmetros primários para obter políticas monetárias diferentes. Entre as centenas de alt coins, alguns dos exemplos mais notáveis incluem os seguintes.

Litecoin

Uma das primeiras alt coins, lançada em 2011, a Litecoin é a segunda moeda digital de maior sucesso após o bitcoin. Suas inovações primárias foram o uso de *script* para o algoritmo de prova-de-trabalho (herdado do Tenebrix) e seus parâmetros de moeda mais rápidos/leves.

- Tempo de geração do bloco: 2,5 minutos
- Total de moedas: 84 milhões de moedas em 2140
- Algoritmo de consenso: Prova de trabalho Scrypt
- Capitalização de mercado: $160 milhões na metade de 2014

Dogecoin

A Dogecoin foi lançada em Dezembro de 2013, baseada em um fork da Litecoin. A Dogecoin é notável porque ela possui uma política monetária de rápida emissão e um limite de moedas muito alto, para estimular o gasto e as doações. A Dogecoin também é notável porque ela começou como uma piada, mas se tornou bastante popular, com uma comunidade numerosa e ativa, antes de diminuir rapidamente em 2014.

- Tempo de geração de bloco: 60 segundos
- Total de moedas: 100.000.000.000 (100 bilhões) de Doge em 2015
- Algoritmo de consenso: Prova de trabalho Scrypt
- Capitalização de mercado: $12 milhões na metade de 2014

Freicoin

Freicoin foi introduzida em julho de 2012. Ela é uma *moeda demurrage*, o que significa que ela tem uma taxa de juros negativa para o valor armazenado. O valor guardado na Freicoin é deduzido de uma taxa APR de 4,5%, para encorajar o consumo e desencorajar a acumulação de dinheiro. A Freicoin é notável por implementar uma política monetária que é o oposto exato da política deflacionária do Bitcoin. A Freicoin não experimentou sucesso enquanto moeda, mas é um exemplo intessante da

variedade de políticas monetárias que podem ser expressadas por alt coins.

- Tempo de geração do bloco: 10 minutos
- Total de moedas: 100 milhões de moedas em 2140
- Algoritmo de consenso: Prova de trabalho em SHA256
- Capitalização de mercado: $130.000 na metade de 2014

Inovação no Consenso: Peercoin, Myriad, Blackcoin, Vericoin, NXT

O mecanismo de consenso do Bitcoin é baseado em prova-de-trabalho usando o algoritmo SHA256. As primeiras alt coins introduziram scrypt como um algoritmo alternativo de prova de trabalho, como forma de tornar a mineração mais amigável a CPUs e menos sucetível a centralização com ASICs. Desde então, inovações no mecanismo de consenso têm surgido num ritmo frenético. Várias alt coins adotaram uma variedade de algoritmos tais como scrypt, scrypt-N, Skein, Groestl, SHA3, X11, Blake, e outros. Algumas alt coins combinam múltiplos algoritmos para prova-de-trabalho. Em 2013, vimos a invenção da alternativa à prova-de-trabalho, chamada *prova de participação*, que forma a base de diversas alt coins modernas.

Prova-de-trabalho é um sistema por meio do qual os atuais donos de uma moeda podem "comprometer" moeda como um colateral para aferimento de juros. Mais ou menos como um depósito garantido, os participantes podem reservar uma porção de suas moedas possuídas, enquanto ganham um retorno do investimento em forma de novas moedas (emitidas como pagamento de juros) e taxas de transação.

Peercoin

A Peercoin foi introduzida em agosto de 2012 e foi a primeira alt coin a usar um algoritmo híbrido de prova-de-trabalho e proof-of-stake para emitir novas unidades da moeda.

- Tempo de geração do bloco: 10 minutos
- Total de moedas: Sem limite
- Algoritmo de consenso: (Híbrido) proof-of-stake com uma prova-de-trabalho inicial
- Capitalização de mercado: $14 milhões na metade de 2014

Myriad

Myriad foi introduzida em fevereiro de 2014 e é notável porque usa, simultaneamente, cinco algoritmos diferentes de prova-de-trabalho (SHA256d, Scrypt, Qubit, Skein, or Myriad-Groestl), com dificuldades variadas para cada algoritmo dependendo da participação do minerador. A intenção é tornar a Myriad imune a especialização e centralização ASIC, e também muito mais resistente a ataques de consenso, porque múltiplos algoritmos de mineração precisariam ser atacados simultaneamente.

- Geração de blocos: média de 30 segundos (alvo de 2,5 minutos através de algoritmo de mineração)

- Total de unidades: 2 bilhões em 2024
- Algoritmo de consenso: Multi-algoritmo de prova-de-trabalho
- Capitalização de mercado: $120.000 na metade de 2014

Blackcoin

A Blackcoin foi apresentada em fevereiro de 2014 e utiliza um algoritmo de consenso de proof-of-stake. Ela também é notável por ter introduzido "multipools", um tipo de pool de mineração que pode alternar entre diferentes alt coins automaticamente, dependendo do lucro de cada.

- Tempo de geração do bloco: 1 minuto
- Total de moedas: Sem limite
- Algoritmo de consenso: Proof-of-stake
- Capitalização de mercado: $3,7 milhões na metade de 2014

VeriCoin

A VeriCoin foi lançada em maio de 2014. Ela usa um algoritmo de consenso de proof-of-stake com uma taxa de juros variável que se ajusta dinamicamente baseada nas forças da oferta e demanda do mercado. Ela também foi a primeira altcoin a apresentar conversão automática para bitcoin, para realização de pagamentos em bitcoins a partir de sua carteira.

- Tempo de geração do bloco: 1 minuto
- Total de moedas: Sem limite
- Algoritmo de consenso: Proof-of-stake
- Capitalização de mercado: $1,1 milhão na metade de 2014

NXT

NXT (pronunciado "Next") é uma alt coin de prova-de-participação "pura", no sentido de que não usa mineração para prova-de-trabalho. NXT é uma implementação de criptomoeda do zero, e não uma derivação do bitcoin ou qualquer outra alt coin. NXT implementa muitas funcionalidades avançadas, incluindo um registro de nomes (similar a Namecoin), um meio de troca de ativos descentralizado (similar às Colored Coins), tendo integrada mensageria descentralizada e segura (similar a Bitmessage), e delegação de participação (para delegar a prova-de-participação a outros). Adeptos da NXT a chamam de criptomoeda de "nova-geração" ou 2.0.

- Tempo de geração do bloco: 1 minuto
- Total de moedas: Sem limite
- Algoritmo de consenso: Proof-of-stake
- Capitalização de mercado: $30 milhões na metade de 2014

Inovação com Mineração de Dupla-Proposta: Primecoin, Curecoin, Gridcoin

O algoritmo de prova-de-trabalho do Bitcoin tem uma única proposta: trazer segurança à rede bitcoin. Comparado com a segurança do sistema de pagamentos tradicional, o custo de mineração não é muito alto. Entretanto, ele foi criticado por muitos por ser pouco econômico. A geração seguinte de alt coins tentou trabalhar esse aspecto. Os algoritmos de prova-de-trabalho de dupla proposta resolve um problema "útil" específico, enquanto produzem prova de trabalho para trazer segurança à rede. O risco de adicionar um uso externo à segurança da moeda é que isso também adiciona influência externa à curva de oferta e demanda.

Primecoin

A Primecoin foi anunciada em julho de 2013. Seu algoritmo de prova-de-trabalho busca por números primos, computando cadeias de Cunningham e números primos gêmeos. Os números primos são úteis para várias disciplinas científicas. A blockchain da Primecoin contém os números primos descobertos, logo produzindo um registro público de descoberta científica em paralelo ao registro público das transações.

- Tempo de geração do bloco: 1 minuto
- Total de moedas: Sem limite
- Algortimo de consenso: Prova de trabalho com descoberta de cadeia de números primos
- Capitalização de mercado: $1,3 milhões na metade de 2014

Curecoin

A Curtecoin foi anunciada em maio de 2013. Ela combina um algoritmo de prova de trabalho com SHA256 associado a pesquisa de envolamento de proteínas através do projeto Folding@Home. O enovelamento de proteínas é uma simulação computadorizada que exige muito poder computacional, que avalia as interações bioquímicas das proteínas, sendo usada para descobrir novos alvos moleculares para medicamentos, que serão usados para curar doenças.

- Tempo de geração do bloco: 10 minutos
- Total de moedas: Sem limite
- Algoritmo de consenso: Prova de trabalho com pesquisa de enovelamento de proteínas
- Capitalização de mercado: $58.000 na metade de 2014

Gridcoin

A Gridcoin foi introduzida em outubro de 2013. Ela substitui a prova-de-trabalho baseada em scripts por subsídios à participação em computação em grid aberta BOINC. BOINC-Berkeley Open Infrastructure for Network Computing - é um protocolo aberto para computação em grid voltada para pesquisa científica, que permite aos participantes compartilhar seu tempo ocioso de ciclos de computação para uma vasta gama de computações de pesquisa acadêmica. Gridcoin usa BOINC como uma plataforma genérica de computação, ao invés de resolver problemas científicos específicos, tais

como números primos ou enovelamento de proteínas.

- Tempo de geração do bloco: 150 segundos
- Total de moedas: Sem limite
- Algoritmo de Consenso: Prova-de-trabalho (Proof-of-work) com subsídio de rede computacional BOINC
- Capitalização de mercado: $122.000 na metade de 2014

Alt Coins focadas no Anonimato: CryptoNote, Bytecoin, Monero, Zerocash/Zerocoin, Darkcoin

O Bitcoin frequentemente é caracterizado como uma moeda "anônima". De fato, é relativamente fácil conectas as identidades das pessoas aos endereços bitcoin e, usando análise de dados, conectar os ednereços uns aos outros para formar uma visão geral dos hábitos de consumo de bitcoins de uma pessoa. Várias alt coins buscaram resolver essa questão diretamente ao focar fortemente em anonimidade. A primeira dessas tentativas foi mais provavelmente a *Zerocoin*, um protocolo de meta-coin para preservar a anonimidade no topo do bitcoin, introduzida como um papel no Simpósio de Segurança e Privacidade IEEE de 2013. A Zerocoin será implementada como uma alt coin completamente separada chamada Zerocash, que ainda está em desenvolvimento enquanto esse livro estava sendo escrito. Uma abordagem alternativa para o anonimato foi lançada com a *CryptoNote* em um paper publicado em outubro de 2013. A CryptoNote é uma tecnologia de base que é implementada por vários forks de alt coin que serão discutidos a seguir. Além do Zerocash e das Cryptonotes, existem várias outras moedas anônimas independentes, como a Darkcoin, que usa endereços stealth ou rearranjo de transações para fornecer anonimato.

Zerocoin/Zerocash

A Zerocoin é uma abordagem teórica de anonimidade em moeda digital apresentada em 2013 por pesquisadores da Johns Hopkins. A Zerocash é uma implementação de alt-coin da Zerocoin que está em desenvolvimento e ainda não foi lançada.

CryptoNote

A CryptoNote é uma alt coin de implementação de referência que fornece a base para o dinheiro digital anônimo. Ela foi introduzida em outubro de 2013. Ela foi projetada para ser bifurcada em várias implementações diferentes e possui um mecanismo embutido de reinicialização que a torna inutilizável como moeda por si só. Várias alt coins surgiram a partir da CryptoNote, incluindo: Bytecoin (BCN), Aeon (AEON), Boolberry (BBR), duckNote (DUCK), Fantomcoin (FCN), Monero (XMR), MonetaVerde (MCN) e Quazarcoin (QCN). A CryptoNote também é notável por ter sido uma implementação completamente nova de cripto-moeda, não tendo sido uma bifurcação do bitcoin.

Bytecoin

Bytecoin foi a primeira implementação brotada da CryptNote, fornecendo uma moeda anônima viável baseada na tecnologia CryptoNote. Bytecoin foi lançada em Julho de 2012. Perceba que havia

previamente uma alt coin chamada Bytecoin com o símbolo monetário BTE, enquanto a Bytecoin derivada da CryptNote tem o símbolo BCN. Bytecoin usa o algoritmo de prova-de-trabalho Cryptonight, que requer acesso a ao menos 2 MB de RAM por instância, tornando-o inadequado para mineração por GPU ou ASIC. Bytecoin herda assinaturas em anel, transações irrastreáveis, e anonimato resistente à análise do blockchain do CryptoNote.

- Tempo de geração do bloco: 2 minutos
- Total de moedas: 184 bilhões de BCN
- Algoritmo de consenso: Prova de trabalho Cryptonight
- Capitalização de mervado: $3 milhões na metade de 2014

Monero

A Monero é uma outra implementação do CryptoNote. Ela possui uma curva de emissão levemente mais achatada que a Bytecoin, emitindo 80% da moeda em seus primeiros quatro anos. Ela oferece as mesmas características de anonimato herdadas da CryptoNote.

- Tempo de geração do bloco: 1 minuto
- Total de moedas: 18,4 milhões de XMR
- Algoritmo de consenso: Prova de trabalho Cryptonight
- Capitalização de mercado: $5 milhões na metade de 2014

Darkcoin

A Darkcoin foi lançada em janeiro de 2014. Ela implementa uma moeda anônima usando um protocolo de mistura de moedas em todas as transações, chamado de DarkSend. O Darkcoin também é notável por usar 11 variações de diferentes funções hash (blake, bmw, groestl, jh, keccak, skein, luffa, cubehash, shavite, simd, echo) para o algoritmo de prova de trabalho.

- Tempo de geração do bloco: 2,5 minutos
- Total de moedas: Máximo de 22 milhões de DRK
- Algoritmo de consenso: Prova de trabalho multi-algortimo e multi-variações
- Capitalização de mercado: $19 milhões na metade de 2014

Alt Chains não-moedas

Alt chains são implementações alternativas do padrão de design da blockchain, que não são primariamente utilizadas como moeda. Muitas incluem uma moeda, mas a moeda é usada como um token para alocar alguma outra coisa, como um recurso ou um contrato. A moeda, em outras palavras, não é o ponto principal da plataforma; ela é uma funcionalidade secundária.

Namecoin

Namecoin foi o primeiro fork no código bitcoin. Namecoin é um registro descentralizado de valor e plataforma de transferência usando a blockchain. Ela suporta um registro global de nome de domínio similar ao sistema de registro de nome de domínio na Internet. Namecoin é atualmente usada como um serviço de nome de domínio (DNS) alternativo, para o domínio de nível raiz .bit. Namecoin também pode ser usado para registrar nomes e pares de valores-chave em outros namespaces; para armazenar coisas como endereços de email, chaves de encriptação, certificados SSL, assinatura de arquivos, sistemas de votação, certificados de ações; e uma miríade de outras aplicações.

O sistema Namecoin inclui a moeda Namecoin (símbolo NMC), que é usada para pagar taxas de transação pelo registro e transferência de nomes. Nos preços atuais, a taxa para registrar um nome é 0,01 NMC ou aproxidamente US$0,01. Assim como no bitcoin, as taxas são coletadas pelos mineradores namecoin.

Os parâmetros básicos do Namecoin são os mesmos do bitcoin:

- Tempo de geração do bloco: 10 minutos
- Total de moedas: 21 milhões de NMC em 2140
- Algoritmo de consenso: Prova de trabalho em SHA256
- Capitalização de mercado: $10 milhões na metade de 2014

Os nomes de espaço da Namecoin não são restritos, e qualquer um pode usar qualquer nome de espaço da maneira que quiser. Entretanto, alguns nomes de espaço possui uma especificação acordada de maneira que quando ele é lido a partir da blockchain, o software a nível de aplicação sabe como ler e processar a partir disso. Se ele for mal formado, então não importa qual software você user para ler a partir de um nome de espaço específico, ele irá resultar em um erro. Alguns nomes de espaço populares são:

- d/ é o namespace de nome de domínio para domínios .bit
- id/ é o namespace para armazenar identificadores de pessoa como endereços de email, chaves PGP, etc.
- u/ é uma especificação adicional, mais estruturada, para armazenar identidades (basead em openspecs)

O cliente Namecoin é muito semelhante ao Bitcoin Core, porque ele é derivado do mesmo código fonte. Após a instalação, o cliente irá baixar uma cópia completa da blockchain Namecoin e então estará pronto para requisitar e registrar nomes. Existem três comandos principais:

name_new

Solicitar ou pré-registrar um nome

name_firstupdate

Registra um nome e torna público o registro

name_update

Modifica os detalhes ou atualiza um registro de nome

Por exemplo, para registrar o domínio mastering-bitcoin.bit, nós usaremos o comando name_new, como demonstrado a seguir:

```
$ namecoind name_new d/mastering-bitcoin
```

```
[
    "21cbab5b1241c6d1a6ad70a2416b3124eb883ac38e423e5ff591d1968eb6664a",
    "a05555e0fc56c023"
]
```

O comando name_new registra uma reivindicação no nome, ao criar um hash do nome com uma chave aleatória. As duas strings que retornam com o name_new são o hash e a chave aleatória (a05555e0fc56c023 no exemplo anterior) que podem ser usadas para fazer o registro público do nome. Uma vez que a reivindicação foi registrada na blockchain da Namecoin, ela pode ser convertida para um registro público com o comando name_firstupdate, ao fornecer-se a chave aleatória:

```
$ namecoind name_firstupdate d/mastering-bitcoin a05555e0fc56c023 "{"map": {"www":
{"ip":"1.2.3.4"}}}}"
b7a2e59c0a26e5e2664948946ebeca1260985c2f616ba579e6bc7f35ec234b01
```

Esse exemplo irá mapear o nome de domínio www.mastering-bitcoin.bit ao endereço IP 1.2.3.4. O hash que retornou é o ID da transação que pode ser usado para seguir esse registro. Você pode ver quais nomes estão registrados para você ao executar o comando name_list:

```
$ namecoind name_list
```

```
[
    {
        "name" : "d/mastering-bitcoin",
        "value" : "{map: {www: {ip:1.2.3.4}}}}",
        "address" : "NCccBXrRUahAGrisBA1BLPWQfSrups8Geh",
        "expires_in" : 35929
    }
]
```

Os registros da Namecoin precisam ser atualizados a cada 36.000 blocos (aproximadamente 200 a 250

dias). O comando name_update não possui taxas e portanto a renovação de domínios na Namecoin é gratuita. Terceiros podem fornecer o serviço de administrar o registro, renovação automática e update através de uma interface web, por uma pequena taxa. Com um serviço sendo oferecido por terceiros, você não precisa rodar um cliente Namecoin, mas você perde o controle independente de um registro de nome descentralizado oferecido pela Namecoin.

Ethereum

Ethereum é um processamento de contrato em linguagem Turing completa e uma plataforma de execução baseada na blockchain. Ele não é um clone do Bitcoin, mas sim uma implementação completamente independente tanto em estrutura quanto em design. Ethereum possui uma moeda interna chama *ether*, que é necessária para pagar a execução dos contratos. A blockchain do Ethereum registra *contratos*, que são expressos em uma linguagem Turing completa, que é de baixo-nível e com códigos em bytes. Essencialmente, um contrato é um programa que roda em cada nó no sistema Ethereum. Contratos Ethereum podem armazenar dados, enviar e receber pagamentos em ether, armazenar ether e executar uma variedade infinita (daí ser Turing completa) de ações computáveis, agindo como softwares agentes autônomos e descentralizados.

Ethereum pode implementar sistemas bastante complexos que são, de outra forma, implementados como alt-chains próprias. Por exemplo, o seguinte é um contrato de registro de nome estilo Namecoin escrito em Ethereum (ou, mais precisamente, escrito em linguagem de alto-nível que pode ser compilada para o código Ethereum):

```
if !contract.storage[msg.data[0]]: # A key ainda não foi pega?
    # Então pegue ela!
    contract.storage[msg.data[0]] = msg.data[1]
    return(1)
else:

    return(0) // Caso contrário, não fazer nada
```

Futuro das Moedas

O futuro das moedas criptográficas como um todo é ainda mais brilhante que o futuro do bitcoin. O bitcoin introduziu uma forma completamente nova de organização e consenso descentralizado que desenvolveu centenas de inovações incríveis. Essas invenções provavelmente afetarão amplos setores da economia, desde sistemas distribuídos de ciência à finança, economia, moedas, bancos centrais e administração corporativa. Muitas atividades humanas que antigamente exigiam instituições ou organizações centralizadas para funcionar como pontos de autoridade ou confiança agora podem ser descentralizada. A invenção da blockchain e o sistema de consenso irá reduzir significativamente o custo da organização e coordenaçõa em sistemas de grande escala, ao mesmo tempo que removerá oportunidades para concentração de poder, corrupção e captação regulatória.

A Segurança do Bitcoin

Manter bitcoins de maneira segura é desafiador porque o bitcoin não é a referência abstrata de um valor, da mesma forma que é o saldo em uma conta bancária. Em sua essência, o Bitcoin é como dinheiro digital ou o ouro. Você provavelmente já escutou o ditado "A posse é 90% da lei". Bem, no mundo do bitcoin, a posse é 100% da lei. A posse das chaves para desbloquear o bitcoin é equivalente a posse de notas de dinheiro ou de uma barra de um metal precioso. Você pode perdê-las, esquecer onde as guardou, ser roubado ou dar acidentalmente uma quantia errada para outra pessoa. Em cada uma dessas situações, da mesma maneira como se tivessem deixado cair notas de dinheiro na calçada, os usuários não tem a quem recorrer.

Entretanto, o bitcoin possui capacidades que o dinheiro, o ouro e as contas bancárias não têm. Como qualquer arquivo de computador, pode-se realizar um backup (cópia de segurança) de uma carteira de bitcoin contendo suas chaves. Ela pode ser armazenada em múltiplas cópias, e até mesmo impressa em papel para fazer um backup offline não-digital. Você não pode fazer um backup das suas notas de dinheiro, de ouro ou de suas contas bancárias. O bitcoin é tão diferente de tudo que já existiu, que também precisamos encarar sua segurança de uma maneira diferente daquela que já estamos acostumados.

Princípios de Segurança

O princípio fundamental do bitcoin é a descentralização e isso traz implicações importantes para a sua segurança. Um modelo centralizado - como o dos bancos tradicionais e da redes de pagamento - depende de controles de acesso e comprovações de segurança para manter o sistema livre de criminosos. Em comparação, um sistema descentralizado como o bitcoin atribui a responsabilidade e o controle aos usuários. Como a rede é baseada em prova de trabalho - e não em controle de acesso - a rede pode ser aberta, não exigindo criptografia para o tráfego de bitcoins.

Em uma rede de pagamentos tradicional, como o sistema de cartões de crédito, o pagamento é open-ended porque o sistema já contém o identificador privado do usuário (o número do cartão de crédito). Dessa forma, após a cobrança inicial, qualquer pessoa com acesso ao identificador pode repetidamente sacar fundos e fazer cobranças ao dono do cartão. Logo, a rede de pagamento deve ser criptografada em toda sua extensão (end-to-end) e deve garantir que nenhum bisbilhoteiro ou intermediários possam comprometer o tráfego de pagamento, quando em trânsito ou quando estiver armazenado. Se um criminoso burlar o controle de acesso ao sistema, ele pode comprometer as transações atuais *e* os tokens de pagamentos - que podem ser usados para criar novas transações. Ou, pior ainda, quando os dados dos consumidores são comprometidos, os consumidores são expostos a roubo de identidade e devem tomar medidas para prevenir o uso fraudulento de suas contas comprometidas.

Com o bitcoin, a história é completamente diferente. Uma transação bitcoin autoriza somente um valor específico e somente para um destinatário específico, além de não poder ser forjada ou modificada. Ela não revela nenhuma informação privada - como as identidades dos usuários - e não pode ser utilizada para autorizar pagamentos adicionais. Dessa forma, uma rede de pagamentos bitcoin não precisa ser encriptada ou protegida de bisbilhoteiros. De fato, você pode transmitir transações em bitcoin usando

um canal público aberto, como redes WiFi sem senha ou Bluetooth - sem comprometer a segurança da transação.

O modelo de segurança descentralizado do bitcoin coloca muito poder nas mãos dos usuários. E com esse poder, vem a responsabilidade de manter o sigilo das chaves. Para a maioria dos usuários isso não é uma tarefa fácil, especialmente em dispositivos comuns como smartphones ou laptops conectados a internet. Embora o modelo descentralizado do bitcoin impeça o tipo de comprometimento em massa observado com os cartões de crédito, muitos usuários não são capazes de guardar suas chaves com a devida segurança e, um a um, acabam sendo hackeados.

Desenvolvendo Sistemas Bitcoin de Maneira Segura

O princípio mais importante para os desenvolvedores em bitcoin é a descentralização. A maioria dos desenvolvedores são acostumados com modelos de segurança centralizados e podem ter a tentação de aplicar estes modelos em seus aplicativos bitcoin, com resultados desastrosos.

A segurança do Bitcoin depende do controle descentralizado das chaves e da validação independente das transações - uma atribuição dos mineradores. Se você quiser alavancar a segurança do Bitcoin, precisa se assegurar que permaneça dentro do modelo de segurança do Bitcoin. Em termos simples: não retire dos usuários o controle de suas chaves e não realize transações fora da blockchain.

Por exemplo, muitas das primeiras exchanges de bitcoin concentravam todos os bitcoins dos usuários em uma única carteira "quente" que tinha as chaves armazenadas em um único servidor. Esse tipo de design retira o controle dos usuários e centraliza o controle das chaves em um único sistema. Muitos destes sistemas foram hackeados, com consequências desastrosas para seus consumidores.

Outro erro comum é o de realizar transações fora da blockchain, numa tentativa desastrada de reduzir taxas de transação ou de acelerar o processamento das transações. Um sistema "fora da blockchain" geralmente grava as transações em um ledger interno, centralizado e somente ocasionalmente realiza uma sincronização com a blockchain do bitcoin. Essa prática, novamente, substitui a segurança descentralizada do bitcoin por uma abordagem proprietária e centralizada. Quando as transações são realizadas fora da blockchain, os ledgers centralizados com pouca segurança podem ser falsificados, possibilitando que intrusos possam, sem chamar a atenção, desviar fundos e esvaziar reservas.

A menos que você esteja preparado para investir pesado em segurança operacional, múltiplas camadas de controle de acesso e auditorias (como os bancos tradicionais fazem), você deveria pensar com muito cuidado antes de retirar os fundos do contexto de segurança descentralizada do Bitcoin. Mesmo que você tenha condições e disciplina para implementar um modelo de segurança robusto, este tipo de modelo apenas replica a fragilidade das redes financeiras tradicionais - infestadas por roubo de identidade, corrupção e fraude. Para aproveitar o modelo único de segurança descentralizada do Bitcoin, você deve evitar a tentação das arquiteturas centralizadas pois, apesar de serem familiares, acabam por fim subvertendo a segurança do Bitcoin.

A Raiz de Confiança

A arquitetura de segurança tradicional é baseada em um conceito conhecido como a *raiz de confiança*,

que é um núcleo de confiança usado como base para a segurança de todo o sistema ou aplicação. A arquitetura de segurança é desenvolvida ao redor da raiz de confiança através de uma série de círculos concêntricos, como as camadas de uma cebola, estendendo a confiança do centro para a periferia. Cada camada usa como suporte a camada interna de maior confiança, usando controles de acesso, assinaturas digitais, criptografia e outros elementos de segurança. Conforme os sistemas de software se tornam mais complexos, aumenta a chance de eles possuírem bugs, que podem torná-los vulneráveis à falhas de segurança. Como resultado, quanto mais complexo se torna um sistema de software, mais difícil se torna a tarefa de mantê-lo seguro. O conceito da raiz de confiança garante que a maior parte da confiança seja depositada na parte menos complexa do sistema, ou seja, nas partes menos vulneráveis do sistema, enquanto o software mais complexo é desenvolvido ao redor. Essa arquitetura de segurança é repetida em diferentes escalas, primeiro estabelecendo uma raiz de confiança no interior do hardware de um sistema único, para depois estender essa raiz de confiança no sistema operacional para serviços de maior nível e, finalmente, através de múltiplos servidores dispostos em camadas de círculos concêntricos com diminuição progressiva da confiança.

A arquitetura de segurança do Bitcoin é diferente. No Bitcoin, o sistema de consenso cria um ledger público de confiança que é completamente descentralizado. Uma blockchain corretamente validada usa o bloco gênese como a raiz de confiança, construindo uma corrente de confiança a partir deste bloco até o mais atual. Os sistemas Bitcoin podem e devem usar a blockchain como sua raiz de confiança. Ao projetar uma aplicação bitcoin complexa que consiste em serviços em diversos sistemas diferentes, você deve examinar cuidadosamente a arquitetura de segurança de maneira a se assegurar do local onde a confiança está sendo depositada. Em última análise, o único elemento que deve ser considerado explicitamente como confiável é uma blockchain validada em sua totalidade. Se a sua aplicação deposita confiança em qualquer elemento que não seja a blockchain, isso deveria ser motivo de preocupação, pois abre as portas para a vulnerabilidade. Um bom método para avaliar a segurança da arquitetura de sua aplicação é considerar cada componente individual e avaliar um cenário hipotético no qual este componente está completamente comprometido e sob o controle de um usuário mal intencionado. Considere cada componente de sua aplicação, um a um, e avalie os impactos na segurança geral se este componente for comprometido. Se a sua aplicação não for mais segura com o comprometimento de um desses componentes, isso demonstra que você depositou equivocadamente a confiança nestes componentes. Uma aplicação bitcoin sem vulnerabilidades deveria ser vulnerável somente a um comprometimento do mecanismo de consenso do bitcoin, significando que a sua raiz de confiança é baseada no elemento mais forte da arquitetura de segurança do bitcoin.

Os numerosos exemplos de bolsas de bitcoins que foram hackeadas servem de exemplo para enfatizar esse ponto, pois, mesmo com uma simples análise superficial, identificaríamos que o projeto e arquitetura de segurança destes serviços eram falhos. Estes serviços usaram implementações centralizadas que depositaram confiança em múltiplos componentes fora da blockchain do bitcoin, como carteiras quentes, bancos de dados de ledger centralizados, chaves de criptografia vulneráveis e esquemas similares.

Melhores Práticas na Segurança do Usuário

Por milhares de anos, os seres humanos têm feito uso de sistemas físicos de controle de segurança. Em comparação, a segurança digital tem sido usada há menos de 50 anos. Os sistemas operacionais

modernos usados para atividades gerais não são muito seguros e não são adequados para o armazenamento de dinheiro digital. Por estarem sempre conectados à internet, nossos computadores estão constantemente expostos a ameaças externas. Eles rodam milhares de componentes de softwares desenvolvidos por centenas de programadores diferentes, os quais frequentemente possuem acesso irrestrito aos arquivos do usuário. Um único bug ou código malicioso em um único software, entre os milhares instalados em seu computador, pode ser suficiente para comprometer a segurança do que você digita em seu teclado e de todos os seus arquivos, permitindo que criminosos roubem seus bitcoins armazenados em aplicativos de carteiras. Para complicar, o nível de manutenção necessária para manter o computador livre de vírus e trojans é uma tarefa difícil para a grande maioria dos usuários de computador.

Apesar de décadas de pesquisas e avanços na segurança da informação, os ativos digitais lamentavelmente ainda são vulneráveis à um adversário determinado. Mesmo os sistemas mais restritos e protegidos, utilizados em companhias de serviços financeiros, agências de inteligência e de defesa, são frequentemente burlados. O Bitcoin cria ativos digitais que possuem valor intrínseco, que podem ser roubados e transferidos para usuários criminosos de maneira instantânea e irreversível. Essa possibilidade criou um forte incentivo para a ação de hackers. Até recentemente, após terem acesso a contas bancárias/cartões de crédito, os hackers tinham que sacar, transferir ou lavar o dinheiro roubado. E, apesar da dificuldade crescente de se realizar estas atividades, elas continuam ocorrendo cada vez mais. O Bitcoin traz um novo aspecto a ser considerado nesse problema, pois agora o dinheiro roubado não precisa mais ser lavado; é um valor intrínseco contido em um ativo digital.

Felizmente, o bitcoin também criou incentivos para o aperfeiçoamento da segurança dos computadores. Se antigamente o risco de um computador ser comprometido era vago e indireto, com o bitcoin esse risco se torna claro é óbvio. O fato de manter bitcoins em um computador aumenta a conscientização dos usuários em manterem seus computadores mais seguros. Como resultado direto da proliferação e maior adoção do bitcoin e outras moedas digitais, tem se observado uma evolução nas técnicas de hacking e nas soluções de segurança. Em termos simples, agora os hackers tem um alvo muito tentador, enquanto os usuários têm um bom motivo para se defenderem.

Ao longo dos últimos três anos, como resultado direto da adoção ao bitcoin, temos observado grandes inovações na área da segurança da informação, na forma de criptografia em hardware, armazenamento de chaves e carteiras de hardware, tecnologia de múltiplas assinaturas e custódias (escrow) digitais. Nas próximas seções nós iremos examinar as melhores práticas para a segurança do usuário.

Armazenamento Físico de Bitcoins

Como a maior parte dos usuários ficam muito mais confortáveis com segurança física do que com segurança da informação, um método muito efetivo para proteger bitcoins é convertê-los para uma forma física. As chaves de bitcoins nada mais são do que longos números. Isso significa que elas podem ser armazenadas em uma forma física, como, por exemplo, impressas em um papel ou gravadas em uma moeda de metal. Dessa maneira, manter a segurança das chaves torna-se tão simples quanto manter seguro um papel com chaves de bitcoin impressas. Um conjunto de chaves de bitcoin impressas em papel é chamado de "carteira em papel" (paper wallet), e existem muitas ferramentas

gratuitas que podem ser usadas para criá-las. Eu pessoalmente mantenho a grande maioria dos meus bitcoins (99% ou mais) armazenados em carteiras de papel, criptografadas com BIP0038, com múltiplas cópias trancadas em cofres. Manter bitcoins offline é conhecido como *armazenamento frio* (cold storage) e é uma das técnicas de segurança mais efetivas. Em um sistema de armazenamento frio, as chaves são geradas e armazenadas de maneira offline (sem nunca se conectar à internet). As chaves são armazenadas em papel (impressas) ou em uma mídia digital, como um pendrive.

Carteiras em Hardware

Em longo prazo, a segurança do bitcoin terá cada vez mais a forma de carteiras em hardware à prova de falsificação. Diferente dos smartphones ou computadores de mesa, a carteira de bitcoin em hardware possuem apenas um único propósito: armazenar bitcoins de forma segura. Com interfaces limitadas e sem o risco de comprometimento por softwares de múltiplos propósitos, essas carteiras podem fornecer um nível altíssimo de segurança para os usuários comuns, não especialistas. Eu espero ver as carteiras de hardware se tornando o principal método de armazenamento de bitcoins. Para conhecer um exemplo deste tipo de carteira, pesquise sobre a carteira Trezor.

Balanceando o Risco

Embora a maioria dos usuários se preocupe, com razão, em evitar que seus bitcoins sejam roubados, existe um risco ainda maior. Arquivos de computador são constantemente perdidos. Se eles contiverem bitcoins, a perda será muito mais dolorosa. Ao usar medidas de segurança para suas carteiras de bitcoin, os usuários devem ter muito cuidado para não exagerarem na proteção e acabarem perdendo suas moedas. Em julho de 2011, uma conhecida organização, responsável por um projeto de divulgação do bitcoin, perdeu quase 7.000 bitcoins. Ao tentar prevenir roubos, os donos da organização implementaram múltiplos backups (cópias de segurança) criptografados. Infelizmente, as chaves da criptografia acabaram sendo perdidas, inutilizando os backups e, assim, perdendo uma fortuna. Da mesma forma que enterrar dinheiro no meio do deserto, proteger demais seus bitcoins pode fazer com que você nunca mais consiga achá-los.

Diversificando o Risco

Você carregaria todas as suas economias em notas de dinheiro guardadas na sua carteira? A maioria das pessoas consideraria isso imprudente, no entanto os usuários de bitcoin frequentemente mantêm todos os seus bitcoins em uma única carteira. Ao invés disso, os usuários deveriam distribuir o risco de perdê-los entre múltiplas carteiras de bitcoin, de diferentes tipos. Usuários prudentes mantém somente uma pequena porção de seus bitcoins, talvez menos de 5%, em uma carteira online ou de smartphone, para usá-los nas compras do dia-a-dia, como "trocados no bolso." O resto deveria ser dividido em diferentes formas de armazenamento, como em uma carteira em desktop e offline (armazenamento frio).

Múltiplas Assinaturas e Governança

Sempre que uma empresa ou pessoa armazena grandes quantidades de bitcoin, a opção de usar endereços com múltiplas assinaturas deveria ser considerada. Esse endereços multi-sig são mais

seguros porque exigem mais de uma assinatura para que um pagamento seja realizado. Idealmente, as chaves usadas nas assinaturas devem ser armazenadas em diferentes locais e sob o controle de diferentes pessoas. Em um ambiente corporativo, por exemplo, as chaves deveriam ser geradas independentemente e guardadas por diferentes executivos da empresa, garantindo que nenhuma pessoa sozinha consiga comprometer as economias da empresa. Os endereços com múltiplas assinaturas também podem oferecer redundância, quando uma pessoa sozinha possuir múltiplas chaves que são armazenadas em diferentes locais.

Legado

Uma importante consideração sobre segurança frequentemente ignorada é a disponibilidade - especialmente no contexto de invalidez, doença ou morte da pessoa que possui as chaves. Para evitar roubos, frequentemente os usuários de bitcoin são orientados a usarem senhas complexas e a manterem suas chaves em segurança, longe do alcance de terceiros. Infelizmente, essa prática torna quase impossível a recuperação dos bitcoins pela família do usuário. Na maioria dos casos, as famílias dos usuários sequer tem conhecimento da existência de poupanças feitas em moeda digital.

Se você tiver uma grande quantidade de bitcoins, considere a ideia de compartilhar detalhes de acesso com um familiar de confiança ou com um advogado. Uma estratégia mais complexa para o seu legado pode ser obtida utilizando endereços de múltiplas assinaturas ou através de um planejamento de herança com um advogado especializado em ativos digitais.

Conclusão

O bitcoin é uma tecnologia nova, complexa e sem precedentes. Com o passar do tempo, iremos desenvolver melhores ferramentas de segurança, bem como práticas que serão mais fáceis de serem utilizadas por pessoas comuns, não especialistas. Enquanto isso, os usuários de bitcoin podem usar muitas das dicas discutidas nesse capítulo para aproveitarem uma experiência com bitcoins segura e livre de problemas.

Appendix A: Comandos do Bitcoin Explorer (bx)

```
Uso: bx COMANDO [--help]

Info: Os comandos bx são:

address-decode
address-embed
address-encode
address-validate
base16-decode
base16-encode
base58-decode
base58-encode
base58check-decode
base58check-encode
base64-decode
base64-encode
bitcoin160
bitcoin256
btc-to-satoshi
ec-add
ec-add-secrets
ec-multiply
ec-multiply-secrets
ec-new
ec-to-address
ec-to-public
ec-to-wif
fetch-balance
fetch-header
fetch-height
fetch-history
fetch-stealth
fetch-tx
fetch-tx-index
hd-new
hd-private
hd-public
hd-to-address
hd-to-ec
hd-to-public
hd-to-wif
```

```
help
input-set
input-sign
input-validate
message-sign
message-validate
mnemonic-decode
mnemonic-encode
ripemd160
satoshi-to-btc
script-decode
script-encode
script-to-address
seed
send-tx
send-tx-node
send-tx-p2p
settings
sha160
sha256
sha512
stealth-decode
stealth-encode
stealth-public
stealth-secret
stealth-shared
tx-decode
tx-encode
uri-decode
uri-encode
validate-tx
watch-address
wif-to-ec
wif-to-public
wrap-decode
wrap-encode
```

Para maiores informações, veja a página do Bitcoin Explorer e a documentação do usuário do Bitcoin Explorer.

Exemplos de uso de comandos bx

Vamos dar uma olhada em alguns exemplos usando comandos do Bitcoin Explorer para trabalharmos com chaves e endereços:

Gera um valor "semente" aleatório usando o comando seed, que usa o gerador de números aleatórios

do sistema operacional. Passe a semente para o comando ec-new para gerar uma nova chave privada. Nós salvaremos o output padrão no arquivo *private_key*:

```
$ bx seed | bx ec-new > private_key
$ cat private_key
73096ed11ab9f1db6135857958ece7d73ea7c30862145bcc4bbc7649075de474
```

Agora, gere a chave pública a partir da chave privada usando o comando ec-to-public. Nós passaremos o arquivo *private_key* no input padrão e salvaremos o output padrão do comando em um novo arquivo *public_key*:

```
$ bx ec-to-public < private_key > public_key
$ cat public_key
02fca46a6006a62dfdd2dbb2149359d0d97a04f430f12a7626dd409256c12be500
```

Nós podemos reformatar a public_key como um endereço usando o comando ec-to-address. Nós passamos a *public_key* no input padrão:

```
$ bx ec-to-address < public_key
17re1S4Q8ZHyCP8Kw7xQad1Lr6XUzWUnkG
```

Chaves geradas dessa maneira produzem uma carteira não-determinística tipo-0. Isso significa que cada chave é gerada a partir de uma semente independente. Os comandos do Bitcoin Explorer também podem gerar chaves deterministicamente, de acordo com o BIP0032. Nesse caso, uma chave "mestre" é criada a partir da semente e então é estendida deterministicamente para produzir uma árvore de subchaves, resultando em uma carteira determinística tipo-2.

Primeiro, usaremos os seed and comandos seed e hd-new para gerar uma chave mestre que será usada como a base para derivar uma hierarquia de chaves.

```
$ bx seed > seed
$ cat seed
eb68ee9f3df6bd4441a9feadec179ff1

$ bx hd-new < seed > master
$ cat master
xprv9s21ZrQH143K2BEhMYpNQoUvAgiEjArAVaZaCTgsaGe6LsAnwubeiTcDzd23mAoyizm9cApe51gNfLMkBqkYo
WWMCRwzfuJk8RwF1SVEpAQ
```

Nós agora iremos usar o comando hd-private para gerar uma chave de "conta" dificultada e uma sequência de duas chaves privadas no interior da conta.

```
$ bx hd-private --hard < master > account
$ cat account
xprv9vkDLt81dTKjwHB8fsVB5QK8cGnzveChzSrtCfvu3aMWvQaThp59ueufuyQ8Qi3qpjk4aKsbmbfxwcgS8PYbg
oR2NWHeLyvg4DhoEE68A1n

$ bx hd-private --index 0 < account
xprv9xHfb6w1vX9xgZyPNXVgAhPxSsEkeRcPHEUV5iJcVEsuUEACvR3NRY3fpGhcnBiDbvG4LgndirDsia1e9F3DW
PkX7Tp1V1u97HKG1FJwUpU

$ bx hd-private --index 1 < account
xprv9xHfb6w1vX9xjc8XbN4GN86jzNAZ6xHEqYxzbLB4fzHFd6VqCLPGRZFsdjsuMVERadbgDbziCRJru9n6tzEWr
ASVpEdrZrFidt1RDfn4yA3
```

A seguir usaremos o comando hd-public para gerar a sequência correspondente de duas chaves públicas.

```
$ bx hd-public --index 0 < account
xpub6BH1zcTuktiFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYPFJ3vezHz
5wzaSW4FiGrseNDR4LKqTy

$ bx hd-public --index 1 < account
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWyMaMV1cn7nUPUkgQHPVXJVqsrA8xWbGQD
hohEcDFTEYMvYzwRD7Juf8
```

As chaves públicas também podem ser derivadas a partir de suas chaves privadas correspondentes ao utilizar-se o comando hd-to-public.

```
$ bx hd-private --index 0 < account | bx hd-to-public
xpub6BH1zcTuktiFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYPFJ3vezHz
5wzaSW4FiGrseNDR4LKqTy

$ bx hd-private --index 1 < account | bx hd-to-public
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWyMaMV1cn7nUPUkgQHPVXJVqsrA8xWbGQD
hohEcDFTEYMvYzwRD7Juf8
```

Nós podemos gerar um número praticamente ilimitado de chaves em uma cadeia determinística, todas derivadas de uma única semente. Essa técnica é usada em muitas aplicações de carteira para gerar chaves que podem ser usadas em back ups e restauradas com um valor único de semente. Isso é mais fácil do que ter que fazer back up de toda a carteira com todas suas chaves geradas aleatoriamente cada vez que uma nova chave for criada.

A semente pode ser codificada usando o comando mnemonic-encode.

```
$ bx hd-mnemonic < seed > words
adore repeat vision worst especially veil inch woman cast recall dwell appreciate
```

A semente pode então ser decodificada usando o comando mnemonic-decode.

```
$ bx mnemonic-decode < words
eb68ee9f3df6bd4441a9feadec179ff1
```

A codificação do mnemônico pode facilitar a gravação da semente e até mesmo a recordação dela.

Appendix A: Propostas de Melhorias ao Bitcoin

Propostas de Melhorias ao Bitcoin (ou BIP - Bitcoin improvement proposals) são documentos de design que fornecem informações para a comunidade bitcoin, ou descreve uma nova função para o bitcoin, seus processos ou ambiente.

Conforme a BIP0001 *Propósitos e Guias das BIP*, existem três tipos de BIP:

BIP Padrão (Standard)

Descreve qualquer mudança que afete a maioria ou a totalidade das implementações do bitcoin, tais como uma mudança no protocolo da rede, uma mudança nas regras de validação de transações e blocos ou qualquer mudança ou adição que afete a inter-operabilidade de aplicativos usando bitcoin.

BIP Informativa

Descreve uma questão relacionada ao design do bitcoin ou fornece guias gerais ou informações para a comunidade bitcoin, mas não propões uma nova característica ou função. BIPs informativas não necessariamente representam um consenso da comunidade bitcoin ou uma recomendação, de forma que usuários e implementadores podem ignorar ou seguir as orientações das BIPs Informativas.

BIP de Processo

Descreve um processo do bitcoin, ou propõe uma mudança de (ou um evento em um) processo. BIPs de Processo são como BIPs Padrão mas se aplicam a outras áreas que não o protocolo bitcoin em si. Elas podem propor uma implementação, mas não para o código do Bitcoin; elas frequentemente requerem o consenso da comunidade; e diferente de BIPs Informativas, as BIPs de Processo são mais do que recomendações - e usuários não são livres para as ignorar. Exemplos incluem procedimentos, guias gerais, mudanças no processo de tomada de decisão e mudanças nas ferramentas ou ambiente usado para desenvolvimento do Bitcoin. Qualquer meta-BIP também é considerada uma BIP de Processo.

As versões das Propostas de Melhorias do Bitcoin são gravadas em um repositório no GitHub. Retrato das BIPs mostra um exemplo de BIPs vigentes em 2014. Consulte o repositório oficial para informação atualizada sobre as BIPs existentes e seus conteúdos.

Table 1. Retrato das BIPs

BIP#	Link	Título	Proprietário	Tipo	Status
1	https://github.com/bitcoin/bips/blob/master/bip-0001.mediawiki	BIP Proposta e Diretrizes	Amir Taaki	Padrão	Ativo
10	https://github.com/bitcoin/bips/blob/master/bip-0010.mediawiki	Distribuição de Transação com Múltiplas Assinaturas (Multi-Sig)	Alan Reiner	Informativo	Rascunho
11	https://github.com/bitcoin/bips/blob/master/bip-0011.mediawiki	M-de-N Transações Padrões	Gavin Andresen	Padrão	Aceito
12	https://github.com/bitcoin/bips/blob/master/bip-0012.mediawiki	OP_EVAL	Gavin Andresen	Padrão	Removido
13	https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki	Formato de Endereço para pay-to-script-hash	Gavin Andresen	Padrão	Final
14	https://github.com/bitcoin/bips/blob/master/bip-0014.mediawiki	Versão de Protocolo e Agente de Usuário	Amir Taaki, Patrick Strateman	Padrão	Aceito
15	https://github.com/bitcoin/bips/blob/master/bip-0015.mediawiki	Aliases	Amir Taaki	Padrão	Removida

BIP#	Link	Título	Proprietário	Tipo	Status
16	https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki	Pagamento Para Script Hash (Pay To Script)	Gavin Andresen	Padrão	Aceito
17	https://github.com/bitcoin/bips/blob/master/bip-0017.mediawiki	OP_CHECKHASHVERIFY (CHV)	Luke Dashjr	Removido	Rascunho
18	https://github.com/bitcoin/bips/blob/master/bip-0018.mediawikilink:	hashScriptCheck	Luke Dashjr	Padrão	Rascunho
19	https://github.com/bitcoin/bips/blob/master/bip-0019.mediawiki	Transações Padrão M-de-N (Low SigOp)	Luke Dashjr	Padrão	Rascunho
20	https://github.com/bitcoin/bips/blob/master/bip-0020.mediawiki	Esquema de URI	Luke Dashjr	Padrão	Substituído
21	https://github.com/bitcoin/bips/blob/master/bip-0021.mediawiki	Esquema URI	Nils Schneider, Matt Corallo	Padrão	Aceito
22	https://github.com/bitcoin/bips/blob/master/bip-0022.mediawiki	getblocktemplate - Fundamentos	Luke Dashjr	Padrão	Aceito

BIP#	Link	Título	Proprietário	Tipo	Status
23	https://github.com/bitcoin/bips/blob/master/bip-0023.mediawiki	getblocktemplate - Mineração em Pools	Luke Dashjr	Padrão	Aceito
30	https://github.com/bitcoin/bips/blob/master/bip-0030.mediawiki	Transações duplicadas	Pieter Wuille	Padrão	Aceito
31	https://github.com/bitcoin/bips/blob/master/bip-0031.mediawiki	Mensagem pong	Mike Hearn	Padrão	Aceito
32	https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki	Carteiras Determinísticas Hierárquicas	Pieter Wuille	Informativo	Aceito
33	https://github.com/bitcoin/bips/blob/master/bip-0033.mediawiki	Nodos Stratizados	Amir Taaki	Padrão	Rascunho
34	https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki	Block v2, Altura na coinbase	Gavin Andresen	Padrão	Aceito
35	https://github.com/bitcoin/bips/blob/master/bip-0035.mediawiki	mensagem mempool	Jeff Garzik	Padrão	Aceito

BIP#	Link	Título	Proprietário	Tipo	Status
36	https://github.com/bitcoin/bips/blob/master/bip-0036.mediawiki	Serviços Customizados	Stefan Thomas	Padrão	Rascunho
37	https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki	Filtragem Bloom	Mike Hearn e Matt Corallo	Padrão	Aceito
38	https://github.com/bitcoin/bips/blob/master/bip-0038.mediawiki	Chave privada protegida por frase secreta	Mike Caldwell	Padrão	Rascunho
39	https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki	Código mnemônico para gerar chaves chaves	Slush	Padrão	Rascunho
40		Protocolo de comunicação Stratum	Slush	Padrão	Número BIP alocado
41		Protocolo de mineração Stratum	Slush	Padrão	Número BIP alocado
42	https://github.com/bitcoin/bips/blob/master/bip-0042.mediawiki	Uma oferta monetária finita para o bitcoin	Pieter Wuille	Padrão	Rascunho
43	https://github.com/bitcoin/bips/blob/master/bip-0043.mediawiki	Proposta de Campo para Carteiras Determinísticas	Slush	Padrão	Rascunho

BIP#	Link	Título	Proprietário	Tipo	Status
44	https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki	Hierarquia de Múltiplas Contas para Carteiras Carteiras	Slush	Padrão	Rascunho
50	https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki	Fork Post-Mortem da Chain de Março de 2013	Gavin Andresen	Informativo	Rascunho
60	https://github.com/bitcoin/bips/blob/master/bip-0060.mediawiki	Mensagem "versão" de Comprimento FIxo (Campo Relay-Transactions)	Amir Taaki	Padrão	Rascunho
61	https://github.com/bitcoin/bips/blob/master/bip-0061.mediawiki	mensagem P2P "reject"	Gavin Andresen	Padrão	Rascunho
62	https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki	Lidando com maleabilidade	Pieter Wuille	Padrão	Rascunho
63		Endereços Camuflados (Stealth)	Peter Todd	Padrão	Número BIP alocado
64	https://github.com/bitcoin/bips/blob/master/bip-0064.mediawiki	mensagem getutxos	Mike Hearn	Padrão	Rascunho

BIP#	Link	Título	Proprietário	Tipo	Status
70	https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki	Protocolo de pagamento	Gavin Andresen	Padrão	Rascunho
71	https://github.com/bitcoin/bips/blob/master/bip-0071.mediawiki	Tipos MIME do protocolo de pagamento	Gavin Andresen	Padrão	Rascunho
72	https://github.com/bitcoin/bips/blob/master/bip-0072.mediawiki	URIs do Protocolo de Pagamento	Gavin Andresen	Padrão	Rascunho
73	https://github.com/bitcoin/bips/blob/master/bip-0073.mediawiki	Uso do cabeçalho "Accept" com a Requisição de Pagamento URLs	Stephen Pair	Padrão	Rascunho

Appendix A: pycoin, ku e tx

A livraria Python pycoin, originalmente escrita e mantida por Richard Kiss, é uma livraria baseada em Python que suporta a manipulação de chaves e transações bitcoin, suportando até a linguagem de script suficiente para lidar apropriadamente com transações não-padrões.

A livraria pycoin suporta tanto o Python 2 (2.7.x) quanto o Python 3 (após 3.3), e vem com alguns utilitários de linha de comando úteis, o ku e o tx.

Utilitário de Chave (Key Utility, KU)

O utilitário de linha de comando ku ("key utility, utilitário de chave") é um canivete suíço para a manipulação de chaves. Ele suporta chaves BIP32, WIF e endereços (bitcoin e alt coins). A seguir, alguns exemplos:

Cria uma chave BIP32 usando fontes de entropia padrão de GPG e */dev/random*:

```
$ ku create

input           : create
network         : Bitcoin
wallet key      : xprv9s21ZrQH143K3LU5ctPZTBnb9kTjA5Su9DcWHvXJemiJBsY7VqXUG7hipgdWaU
                    m2nhnzdvxJf5KJo9vjP2nABX65c5sFsWsV8oXcbpehtJi
public version  : xpub661MyMwAqRbcFpYYiuvZpKjKhnJDZYAkWSY76JvvD7FH4fsG3Nqiov2CfxzxY8
                    DGcpfT56AMFeo8M8KPkFMfLUtvwjwb6WPv8rY65L2q8Hz
tree depth      : 0
fingerprint     : 9d9c6092
parent f'print  : 00000000
child index     : 0
chain code      : 80574fb260edaa4905bc86c9a47d30c697c50047ed466c0d4a5167f6821e8f3c
private key     : yes
secret exponent :
112471538590155650688604752840386134637231974546906847202389294096567806844862
 hex            : f8a8a28b28a916e1043cc0aca52033a18a13cab1638d544006469bc171fddfbe
wif             : L5Z54xi6qJusQT42JHA44mfPVZGjyb4XBRWfxAzUWwRiGx1kV4sP
 uncompressed   : 5KhoEavGNNH4GHKoy2Ptu4KfdNp4r56L5B5un8FP6RZnbsz5Nmb
public pair x   :
76460638240546478364843397478278468101877117767873462127021560368290114016034
public pair y   :
59807879657469774102040120298272207730921291736633247737077406753676825777701
 x as hex       : a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
 y as hex       : 843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcd625
y parity        : odd
key pair as sec : 03a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
 uncompressed   : 04a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
                    843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcd625
hash160         : 9d9c609247174ae323acfc96c852753fe3c8819d
 uncompressed   : 8870d869800c9b91ce1eb460f4c60540f87c15d7
Bitcoin address : 1FNNRQ5fSv1wBi5gyfVBs2rkNheMGt86sp
 uncompressed   : 1DSS5isnH4FsVaLVjeVXewVSpfqktdiQAM
```

Cria uma chave BIP32 a partir de uma frase secreta (passphrase):

WARNING	A frase-passe nesse exemplo é fácil demais para ser descoberta.

```
$ ku P:foo

input           : P:foo
network         : Bitcoin
wallet key      : xprv9s21ZrQH143K31AgNK5pyVvW23gHnkBq2wh5aEk6g1s496M8ZMjxncCKZKgb5j
                    ZoY5eSJMJ2Vbyvi2hbmQnCuHBujZ2WXGTux1X2k9Krdtq
public version  : xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtS
                    VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
tree depth      : 0
fingerprint     : 5d353a2e
parent f'print  : 00000000
child index     : 0
chain code      : 5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc
private key     : yes
secret exponent :
65825730547097305716057160437970790220123864299761908948746835886007793998275
 hex            : 91880b0e3017ba586b735fe7d04f1790f3c46b818a2151fb2def5f14dd2fd9c3
wif             : L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
 uncompressed   : 5JvNzA5vXDoKYJdw8SwwLHxUxaWvn9mDea6k1vRPCX7KLUVWa7W
public pair x   :
81821982719381104061777349269130419024493616650993589394553404347774393168191
public pair y   :
58994218069605424278320703250689780154785099509277691723126325051200459038290
 x as hex       : b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
 y as hex       : 826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52
y parity        : even
key pair as sec : 02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
 uncompressed   : 04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
                    826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52
hash160         : 5d353a2ecdb262477172852d57a3f11de0c19286
 uncompressed   : e5bd3a7e6cb62b4c820e51200fb1c148d79e67da
Bitcoin address : 19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
 uncompressed   : 1MwkRkogzBRMehBntgcq2aJhXCXStJTXHT
```

Adquire informações como JSON:

```
$ ku P:foo -P -j
```

```
{
   "y_parity": "even",
   "public_pair_y_hex":
"826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
   "private_key": "no",
   "parent_fingerprint": "00000000",
   "tree_depth": "0",
   "network": "Bitcoin",
   "btc_address_uncompressed": "1MwkRkogzBRMehBntgcq2aJhXCXStJTXHT",
   "key_pair_as_sec_uncompressed":
"04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f826d8b4d3010aea16ff4c1
c1d3ae68541d9a04df54a2c48cc241c2983544de52",
   "public_pair_x_hex":
"b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f",
   "wallet_key":
"xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtSVYFvXz2vPPpbXE1qpjoUFi
dhjFj82pVShWu9curWmb2zy",
   "chain_code": "5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc",
   "child_index": "0",
   "hash160_uncompressed": "e5bd3a7e6cb62b4c820e51200fb1c148d79e67da",
   "btc_address": "19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii",
   "fingerprint": "5d353a2e",
   "hash160": "5d353a2ecdb262477172852d57a3f11de0c19286",
   "input": "P:foo",
   "public_pair_x":
"81821982719381104061777349269130419024493616650993589394553404347774393168191",
   "public_pair_y":
"58994218069605424278320703250689780154785099509277691723126325051200459038290",
   "key_pair_as_sec":
"02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f"
}
```


Chave pública BIP32:

```
$ ku -w -P P:foo
xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtSVYFvXz2vPPpbXE1qpjoUFid
hjFj82pVShWu9curWmb2zy
```

Gerar uma subchave:

```
$ ku -w -s3/2 P:foo
xprv9wTErTSkjVyJa1v4cUTFMFkWMe5eu8ErbQcs9xajnsUzCBT7ykHAwdrxvG3g3f6BFk7ms5hHBvmbdutNmyg6i
ogWKxx6mefEw4M8EroLgKj
```

Subkey dificultada (hardened):

```
$ ku -w -s3/2H P:foo
xprv9wTErTSu5AWGkDeUPmqBcbZWX1xq85ZNX9iQRQW9DXwygFp7iRGJo79dsVctcsCHsnZ3XU3DhsuaGZbDh8iDk
BN45k67UKsJUXM1JfRCdn1
```

WIF:

```
$ ku -W P:foo
L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
```

Endereço:

```
$ ku -a P:foo
19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
```

Gerar várias subchaves:

```
$ ku P:foo -s 0/0-5 -w
xprv9xWkBDfyBXmZjBG9EiXBpy67KK72fphUp9utJokEBFtjsjiuKUUDF5V3TU8U8cDzytqYnSekc8bYuJS8G3bhX
xKWB89Ggn2dzLcoJsuEdRK
xprv9xWkBDfyBXmZnzKf3bAGifK593gT7WJZPnYAmvc77gUQVej5QHckc5Adtwxa28ACmANi9XhCrRvtFqQcUxt8r
UgFz3souMiDdWxJDZnQxzx
xprv9xWkBDfyBXmZqdXA8y4SWqfBdy71gSW9sjx9JpCiJEiBwSMQyRxan6srXUPBtj3PTxQFkZJAiwoUpmvtrxKZu
4zfsnr3pqyy2vthpkwuoVq
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8EskpzK
L1Y8Gk9aX6QbryA5raK73p
xprv9xWkBDfyBXmZv2q3N66hhZ8DAcEnQDnXML1J62krJAcf7Xb1HJwuW2VMJQrCofY2jtFXdiEY8UsRNJfqK6DAd
yZXoMvtaLHyWQx3FS4A9zw
xprv9xWkBDfyBXmZw4jEYXUHYc9fT25k9irP87n2RqfJ5bqbjKdT84Mm7Wtc2xmzFuKg7iYf7XFHKkSsaYKWKJbR5
4bnyAD9GzjUYbAYTtN4ruo
```

Gerar os endereços correspondentes:

```
$ ku P:foo -s 0/0-5 -a
1MrjE78H1R1rqdFrmkjdHnPUdLCJALbv3x
1AnYyVEcuqeoVzH96zj1eYKwoWfwte2pxu
1GXr1kZfxE1FcK6ZRD5sqqqs5YfvuzA1Lb
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
1Cz2rTLjRM6pMnxPNrRKp9ZSvRtj5dDUML
1WstdwPnU6HEUPme1DQayN9nm6j7nDVEM
```

Gerar os WIFs correspondentes:

```
$ ku P:foo -s 0/0-5 -W
L5a4iE5k9gcJKGqX3FWmxzBYQc29PvZ6pgBaePLVqT5YByEnBomx
Kyjgne6GZwPGB6G6kJEhoPbmyjMP7D5d3zRbHVjwcq4iQXD9QqKQ
L4B3ygQxK6zH2NQGxLDee2H9v4Lvwg14cLJW7QwWPzCtKHdWMaQz
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmTDMrqemY8UF
L2oD6vA4TUyqPF8QG4vhUFSgwCyuuvFZ3v8SKHYFDwkbM765Nrfd
KzChTbc3kZFxUSJ3Kt54cxsogeFAD9CCM4zGB22si8nfKcThQn8C
```

Verifica que isso funciona ao escolher uma string BIP32 (a que corresponde à subchave 0/3):

```
$ ku -W
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8EskpzK
L1Y8Gk9aX6QbryA5raK73p
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmTDMrqemY8UF
$ ku -a
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuqxYGpHfv9cnGj5P7e8EskpzK
L1Y8Gk9aX6QbryA5raK73p
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
```

Sim, isso parece familiar.

A partir de um exponente secreto:

```
$ ku 1

input           : 1
network         : Bitcoin
secret exponent : 1
 hex            : 1
wif             : KwDiBf89QgGbjEhKnhXJuH7LrciVrZi3qYjgd9M7rFU73sVHnoWn
 uncompressed   : 5HpHagT65TZzG1PH3CSu63k8DbpvD8s5ip4nEB3kEsreAnchuDf
public pair x   :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y   :
32670510020758816978083085130507043184471273380659243275938904335757337482424
 x as hex       : 79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex       : 483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity        : even
key pair as sec : 0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 uncompressed   : 0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
                    483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160         : 751e76e8199196d454941c45d1b3a323f1433bd6
 uncompressed   : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAMH
 uncompressed   : 1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm
```

Versão Litecoin:

```
$ ku -nL 1

input            : 1
network          : Litecoin
secret exponent  : 1
 hex             : 1
wif              : T33ydQRKp4FCW5LCLLUB7deioUMoveiwekdwUwyfRDeGZm76aUjV
 uncompressed    : 6u823ozcyt2rjPH8Z2ErsSXJB5PPQwK7VVTwwN4mxLBFrao69XQ
public pair x    :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y    :
32670510020758816978083085130507043184471273380659243275938904335757337482424
 x as hex        : 79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex        : 483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity         : even
key pair as sec  : 0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 uncompressed    : 0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
                     483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160          : 751e76e8199196d454941c45d1b3a323f1433bd6
 uncompressed    : 91b24bf9f5288532960ac687abb035127b1d28a5
Litecoin address : LVuDpNCSSj6pQ7t9Pv6d6sUkLKoqDEVUnJ
 uncompressed    : LYWKqJhtPeGyBAw7WC8R3F7ovxtzAiubdM
```

Dogecoin WIF:

```
$ ku -nD -W 1
QNcdLVw8fHkixm6NNyN6nVwxKek4u7qrioRbQmjxac5TVoTtZuot
```

A partir de um par público (na Testnet):

```
$ ku -nT
55066263022277343669578718895168534326250603453777594175500187360389116729240,even

input                    :
55066263022277343669578718895168534326250603453777594175500187360389116729240,even
network                  : Bitcoin testnet
public pair x            :
55066263022277343669578718895168534326250603453777594175500187360389116729240
public pair y            :
32670510020758816978083085130507043184471273380659243275938904335757337482424
 x as hex                :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex                :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity                 : even
key pair as sec          :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 uncompressed            :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798

483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160                  : 751e76e8199196d454941c45d1b3a323f1433bd6
 uncompressed            : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin testnet address : mrCDrCybB6J1vRfbwM5hemdJz73FwDBC8r
 uncompressed            : mtoKs9V381UAhUia3d7Vb9GNak8Qvmcsme
```

A partir de hash160:

```
$ ku 751e76e8199196d454941c45d1b3a323f1433bd6

input           : 751e76e8199196d454941c45d1b3a323f1433bd6
network         : Bitcoin
hash160         : 751e76e8199196d454941c45d1b3a323f1433bd6
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAMH
```

Como um endereço Dogecoin:

```
$ ku -nD 751e76e8199196d454941c45d1b3a323f1433bd6

input            : 751e76e8199196d454941c45d1b3a323f1433bd6
network          : Dogecoin
hash160          : 751e76e8199196d454941c45d1b3a323f1433bd6
Dogecoin address : DFpN6QqFfUm3gKNaxN6tNcab1FArL9cZLE
```

Utilitário de Transação (Transacion Utility, TX)

O utilitário de linha de comando tx irá mostrar as transações de uma maneira fácil, fazer fetch de transações base a partir do cache de transações do pycoin ou de serviços web (blockchain.info, blockr.io e biteasy.com são os suportados atualmente), mesclar transações, adicionar ou remover inputs ou outputs e assinar transações.

A seguir temos alguns exemplos.

Veja a famosa transação "pizza" [PIZZA]:

```
$ tx 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
aviso: considere definir a variável de ambiente PYCOIN_CACHE_DIR=~/.pycoin_cache para
fazer o cache das transações adquiridas através de serviços web
aviso: não foram encontrados provedores de serviço para get_tx; considere definir a
variável de ambiente
PYCOIN_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCKEXPLORER
uso: tx [-h] [-t TRANSACTION_VERSION] [-l LOCK_TIME] [-n NETWORK] [-a]
          [-i endereço] [-f caminho-para-chaves-privadas] [-g ARGUMENTO_GPG]
          [--remove-tx-in tx_in_index_to_delete]
          [--remove-tx-out tx_out_index_to_delete] [-F transaction-fee] [-u]
          [-b URL_DO_BITCOIND] [-o caminho-para-o-arquivo-output]
          argumento [argumento ...]
tx: erro: não foi possível encontrar o Tx com o id
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
```

Opa! Nós não definimos os serviços web. Vamos fazer isso agora:

```
$ PYCOIN_CACHE_DIR=~/.pycoin_cache
$ PYCOIN_SERVICE_PROVIDERS=BLOCKR_IO:BLOCKCHAIN_INFO:BITEASY:BLOCKEXPLORER
$ export PYCOIN_CACHE_DIR PYCOIN_SERVICE_PROVIDERS
```

Isso não é feito automaticamente de maneira que uma ferramenta de linha de comando não irá vazar

para um website de terceiros informações potencialmente privadas sobre as transações que você está interessado. Se você não se importar com isso, você pode inserir essas linhas em seu *.profile*.

Vamos tentar novamente:

```
$ tx 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
Version:  1  tx hash 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
159 bytes
TxIn count: 1; TxOut count: 1
Lock time: 0 (válido a qualquer momento)
Input:
  0:                           (desconhecido) de
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0
Output:
  0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik recebe 10000000.00000 mBTC
Output total 10000000.00000 mBTC
incluindo os não-gastos no dump hex, já que a transação não está completamente
assinada
010000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a4
93046022100a7f26eda874931999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e
9199238eb73f07c8f209504c84b80f03e30ed8169edd44f80ed17ddf451901ffffffff010010a5d4e8000
0001976a9147ec1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000

** não foi possível validar a transação, pois as transações-fonte estão faltando
```

A última linha aparece porque para validar as assinaturas das transações, você tecnicamente precisa das transações-fonte. Então vamos adicionar -a para acrescentar às transações uma informação de fonte:

```
$ tx -a 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
aviso: as recomendações de taxas de transação calculadas casualmente e estimativas
podem estar incorretas
aviso: a taxa de transação é menor que o valor experado de 0,1 mBTC (casualmente
calculado). A transação pode não se propagar.
Version:  1  tx hash 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
159 bytes
TxIn count: 1; TxOut count: 1
Lock time: 0 (válido a qualquer momento)
Input:
  0: 17WFx2GQZUmh6Up2NDNCEDk3deYomdNCfk de
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0 10000000.00000
mBTC  sig ok
Output:
  0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik recebe 10000000.00000 mBTC
Input total  10000000,00000 mBTC
Output total 10000000.00000 mBTC
Total de taxas        0,00000 mBTC

010000000141045e0ab2b0b82cdefaf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a4
93046022100a7f26eda874931999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e
9199238eb73f07c8f209504c84b80f03e30ed8169edd44f80ed17ddf451901ffffffff010010a5d4e8000
0001976a9147ec1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000

todos os valores de transação recebidas foram validados
```

Agora, vamos ver os outputs não-gastos para um endereço específico (UTXO). No bloco #1, nós vemos uma transação coinbase para 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX. Vamos usar fetch_unspent para encontrar todas as moedas (bitcoins) nesse endereço:

```
$ fetch_unspent 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX
a3a6f902a51a2cbebede144e48a88c05e608c2cce28024041a5b9874013a1e2a/0/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/333000
cea36d008badf5c7866894b191d3239de9582d89b6b452b596f1f1b76347f8cb/31/76a914119b098e2e9
80a229e139a9ed01a469e518e6f2688ac/10000
065ef6b1463f552f675622a5d1fd2c08d6324b4402049f68e767a719e2049e8d/86/76a914119b098e2e9
80a229e139a9ed01a469e518e6f2688ac/10000
a66dddd42f9f2491d3c336ce5527d45cc5c2163aaed3158f81dc054447f447a2/0/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/10000
ffd901679de65d4398de90cefe68d2c3ef073c41f7e8dbec2fb5cd75fe71dfe7/0/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/100
d658ab87cc053b8dbcfd4aa2717fd23cc3edfe90ec75351fadd6a0f7993b461d/5/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/911
36ebe0ca3237002acb12e1474a3859bde0ac84b419ec4ae373e63363ebef731c/1/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/100000
fd87f9adebb17f4ebb1673da76ff48ad29e64b7afa02fda0f2c14e43d220fe24/0/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/1
dfdf0b375a987f17056e5e919ee6eadd87dad36c09c4016d4a03cea15e5c05e3/1/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/1337
cb2679bfd0a557b2dc0d8a6116822f3fcbe281ca3f3e18d3855aa7ea378fa373/0/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/1337
d6be34ccf6edddc3cf69842dce99fe503bf632ba2c2adb0f95c63f6706ae0c52/1/76a914119b098e2e98
0a229e139a9ed01a469e518e6f2688ac/2000000

0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098/0/410496b538e853519c
726a2c91e61ec11600ae1390813a627c66fb8be7947be63c52da7589379515d4e0a604f8141781e622947
21166bf621e73a82cbf2342c858eeac/5000000000
```

Appendix A: Operadores de Linguagem de Script de Transação, Constantes e Símbolos

Adiciona um valor no stack demonstra os operadores para adicionar valor no stack (pilha).

Table 1. Adiciona um valor no stack

Símbolo	Valor (hex)	Descrição
OP_0 or OP_FALSE	0x00	Um array vazio é adicionado no stack
1-75	0x01-0x4b	Adiciona os próximos N bytes no stack, quando N é 1 a 75 bytes
OP_PUSHDATA1	0x4c	O próximo script byte contém N, adiciona os seguintes N bytes no stack
OP_PUSHDATA2	0x4d	Os próximos dois script bytes contém N, adiciona os seguintes N bytes no stack
OP_PUSHDATA4	0x4e	Os próximos quatro script bytes contém N, adiciona os seguintes N bytes no stack
OP_1NEGATE	0x4f	Adiciona o valor "–1" no stack
OP_RESERVED	0x50	Suspenso - Transação inválida a menos que seja encontrada em uma cláusula OP_IF não-executada
OP_1 or OP_TRUE	0x51	Adiciona o valor "1" no stack
OP_2 to OP_16	0x52 to 0x60	Para OP_N, adiciona o valor "N" no stack. Por exemplo, OP_2 adiciona "2"

Controle de fluxo condicional mostra os operadores de controle de fluxo condicionais.

Table 2. Controle de fluxo condicional

Símbolo	Valor (hex)	Descrição
OP_NOP	0x61	Não fazer nada

Símbolo	Valor (hex)	Descrição
OP_VER	0x62	Suspenso - Transação inválida a menos que seja encontrada em uma cláusula OP_IF não-executada
OP_IF	0x63	Executa as declarações a seguir se o topo do stack não for 0
OP_NOTIF	0x64	Executa as declarações a seguir se o topo do stack não for 0
OP_VERIF	0x65	Pare - Transação inválida
OP_VERNOTIF	0x66	Pare - Transação inválida
OP_ELSE	0x67	Executa apenas se as declarações anteriores não foram executadas
OP_ENDIF	0x68	Termina o bloco OP_IF, OP_NOTIF, OP_ELSE
OP_VERIFY	0x69	Verifica o topo do stack, suspende e invalida a transação se não for TRUE
OP_RETURN	0x6a	Interrompe e invalida a transação

Operações de stack demonstra os operadores usados para manipular o stack.

Table 3. Operações de stack

Símbolo	Valor (hex)	Descrição
OP_TOALTSTACK	0x6b	Exibe o item do topo do stack e o empurra para o stack alternativo
OP_FROMALTSTACK	0x6c	Exibe o item do topo do stack alternativo e o empurra para o stack
OP_2DROP	0x6d	Exibe os dois itens do topo do stack
OP_2DUP	0x6e	Duplica os dois itens no topo do stack
OP_3DUP	0x6f	Duplica os três itens no topo do stack

Símbolo	Valor (hex)	Descrição
OP_2OVER	0x70	Copia o terceiro e quarto itens do stack para o topo
OP_2ROT	0x71	Move o quinto e sexto itens no stack para o topo
OP_2SWAP	0x72	Troca os dois pares de itens no topo do stack
OP_IFDUP	0x73	Duplica o item no topo do stack se ele não for 0
OP_DEPTH	0x74	Conta os itens no stack e empurra o resultado da contagem
OP_DROP	0x75	Exibe o item do topo do stack
OP_DUP	0x76	Duplica o item no topo do stack
OP_NIP	0x77	Exibe o segundo item do stack
OP_OVER	0x78	Copia o segundo item do stack e o empurra para o topo
OP_PICK	0x79	Exibe o valor N do topo, então copia o Nº item para o topo do stack
OP_ROLL	0x7a	Exibe o valor N do topo, então move o Nº item para o topo do stack
OP_ROT	0x7b	Inverte os três itens do topo do stack
OP_SWAP	0x7c	Permuta os três itens do topo no stack
OP_TUCK	0x7d	Copia o item do topo e insere entre o topo e o segundo item.

Operações de pedaços de string demonstra as operações de string.

Table 4. Operações de pedaços de string

Símbolo	Valor (hex)	Descrição
OP_CAT	0x7e	Desativado (concatena os dois itens do topo)
OP_SUBSTR	0x7f	Desativado (retorna substring)

Símbolo	Valor (hex)	Descrição
OP_LEFT	0x80	Desativado (retorna left substring)
OP_RIGHT	0x81	Desativado (retorna right substring)
OP_SIZE	0x82	Calcula o comprimento da string do item do topo e empurra o resultado

Aritmética binária e condicionais demonstra os operadores de aritimética binária e de lógica booleana.

Table 5. Aritmética binária e condicionais

Símbolo	Valor (hex)	Descrição
OP_INVERT	0x83	Desativado (Inverte os bits do item do topo)
OP_AND	0x84	Desativado (Booleano AND dos dois itens do topo)
OP_OR	0x85	Desativado (Booleano OR dos dois itens do topo)
OP_XOR	0x86	Desativado (Booleano XOR dos dois itens do topo)
OP_EQUAL	0x87	Empurra TRUE (1) se os dois itens do topo forem exatamente iguais, caso contrário empurra FALSE (0)
OP_EQUALVERIFY	0x88	O mesmo que o OP_EQUAL, mas executa o OP_VERIFY após para suspender se não for TRUE
OP_RESERVED1	0x89	Suspenso - Transação inválida a menos que seja encontrada em uma cláusula OP_IF não-executada
OP_RESERVED2	0x8a	Suspende - Transação inválida a menos que seja encontrada em uma cláusula OP_IF não-executada

Operadores numéricos demonstra os operadores numéricos (aritméticos).

Table 6. Operadores numéricos

Símbolo	Valor (hex)	Descrição
OP_1ADD	0x8b	Adiciona 1 ao item do topo
OP_1SUB	0x8c	Subtrai 1 do item do topo
OP_2MUL	0x8d	Desativado (multiplica o item do topo por 2)
OP_2DIV	0x8e	Desativado (divide o item do topo por 2)
OP_NEGATE	0x8f	Inverte o sinal do item do topo
OP_ABS	0x90	Muda o sinal do item do topo para positivo
OP_NOT	0x91	Se o item do topo é 0 ou 1 Booleano o inverte, caso contrário retorna 0
OP_0NOTEQUAL	0x92	Se o item do topo é zero, retorna 0, caso contrário retorna 1
OP_ADD	0x93	Exibe os dois itens do topo, soma os dois e empurra o resultado
OP_SUB	0x94	Exibe os dois itens do topo, subtrai o primeiro do segundo e empurra o resultado
OP_MUL	0x95	Desativado (multiplica os dois itens do topo)
OP_DIV	0x96	Desativado (divide o segundo item pelo primeiro item)
OP_MOD	0x97	Desativado (a sobra divide o segundo item pelo primeiro item)
OP_LSHIFT	0x98	Desativada (muda o segundo item esquerdo pelo primeiro número de bits do primeiro)
OP_RSHIFT	0x99	Desativado (muda o segundo item direito pelo número de bits do primeiro)
OP_BOOLAND	0x9a	Booleano AND dos dois itens do topo
OP_BOOLOR	0x9b	Booleano OR dos dois itens do topo

Símbolo	Valor (hex)	Descrição
OP_NUMEQUAL	0x9c	Retorna TRUE se os dois itens do topo são iguais a números
OP_NUMEQUALVERIFY	0x9d	O mesmo que o NUMEQUAL, então OP_VERIFY para suspender se não for TRUE
OP_NUMNOTEQUAL	0x9e	Retorna TRUE se os dois itens do topo não são iguais a números
OP_LESSTHAN	0x9f	Retorna TRUE se o segundo item for menor que o item do topo
OP_GREATERTHAN	0xa0	Retorna TRUE se o segundo item for maior do que o item do topo
OP_LESSTHANOREQUAL	0xa1	Retorna TRUE se o segundo item for menor ou igual ao item do topo
OP_GREATERTHANOREQUAL	0xa2	Retorna TRUE se o segundo item for maior ou igual ao item do topo
OP_MIN	0xa3	Retorna o menor dos dois itens do topo
OP_MAX	0xa4	Retorna o maior dos dois itens do topo
OP_WITHIN	0xa5	Retorna TRU se o terceiro item estiver entre o segundo item (ou for igual) e o primeiro item

Operações criptográficas e de hashing demonstra os operações de função criptográfica.

Table 7. Operações criptográficas e de hashing

Símbolo	Valor (hex)	Descrição
OP_RIPEMD160	0xa6	Retorna o hash RIPEMD160 do item do topo
OP_SHA1	0xa7	Retorna o hash SHA1 do item do topo
OP_SHA256	0xa8	Retorna o hash SHA256 do item do topo
OP_HASH160	0xa9	Retorna o hash RIPEMD160(SHA256(x)) do item do topo

Símbolo	Valor (hex)	Descrição
OP_HASH256	0xaa	Retorna o hash SHA256(SHA256(x)) do item do topo
OP_CODESEPARATOR	0xab	Marca o início dos dados verificados pela assinatura
OP_CHECKSIG	0xac	Exibe uma chave pública e assinatura e valida a assinatura para os dados hashados da transação, retorna TRUE se corresponde
OP_CHECKSIGVERIFY	0xad	O mesmo que o CHECKSIG, então OP_VERIFY para suspender se não for TRUE
OP_CHECKMULTISIG	0xae	Executa o CHECKSIG para cada par de assinatura e chave pública fornecido. Todos devem corresponder. Um bug na implementação exibe um valor extra, use o prefixo OP_NOP como uma solução temporária
OP_CHECKMULTISIGVERIFY	0xaf	O mesmo que o CHECKMULTISIG, então OP_VERIFY para suspender se não for TRUE

Não-operadores demonstra os símbolos não-operadores

Table 8. Não-operadores

Símbolo	Valor (hex)	Descrição
OP_NOP1-OP_NOP10	0xb0-0xb9	Não faz nada, ignorado

Códigos OP reservados para uso interno pelo parser demonstra os códigos de operador reservado para o parser interno de script.

Table 9. Códigos OP reservados para uso interno pelo parser

Símbolo	Valor (hex)	Descrição
OP_SMALLDATA	0xf9	Representa o campo de dados pequeno
OP_SMALLINTEGER	0xfa	Representa um campo pequeno de dados numerais

Símbolo	Valor (hex)	Descrição
OP_PUBKEYS	0xfb	Representa os campos de chave pública
OP_PUBKEYHASH	0xfd	Representa o campo do hash da chave pública
OP_PUBKEY	0xfe	Representa um campo de chave pública
OP_INVALIDOPCODE	0xff	Representa qualquer código OP que não está atualmente designado