**Capítulo 6  
Arquitetura do Bitcoin**

O Bitcoin foi a primeira aplicação da tecnologia blockchain e iniciou uma revolução com a introdução da primeira moeda digital totalmente descentralizada. Provou ser notavelmente seguro e valioso como moeda digital, apesar de ser altamente volátil. A invenção do Bitcoin também despertou grande interesse na academia e na indústria, abrindo muitas novas áreas de pesquisa. Neste capítulo, introduziremos o Bitcoin em detalhes.

Especificamente, neste capítulo, concentraremos nossa atenção nos fundamentos do Bitcoin, como as transações são construídas e utilizadas, estruturas de transações, endereços, contas e mineração, nos seguintes tópicos:

* Introdução ao Bitcoin
* Chaves criptográficas
* Endereços
* Transações
* Blockchain
* Mineradores
* Rede
* Carteiras

Vamos começar com uma visão geral do Bitcoin.

**Introdução ao Bitcoin**  
Desde sua introdução em 2008 por Satoshi Nakamoto, o Bitcoin ganhou imensa popularidade e é atualmente a moeda digital de maior sucesso no mundo, com bilhões de dólares investidos nele.

Sua popularidade também é evidente pelo alto número de usuários e investidores, pelas notícias diárias relacionadas ao Bitcoin e pelas muitas startups e empresas que oferecem exchanges online baseadas em Bitcoin. Agora ele também é negociado como futuros de Bitcoin na Chicago Mercantile Exchange (CME).

Em 2008, o Bitcoin foi introduzido em um artigo chamado *Bitcoin: A Peer-to-Peer Electronic Cash System*. Este artigo está disponível em <https://bitcoin.org/en/bitcoin-paper>. Acredita-se que o nome do autor, Satoshi Nakamoto, seja um pseudônimo, pois a verdadeira identidade do inventor do Bitcoin é desconhecida e é objeto de muita especulação.

A primeira ideia-chave introduzida no artigo foi a de dinheiro eletrônico puramente P2P que não precisa de um banco intermediário para transferir pagamentos entre pares. No entanto, o Bitcoin pode ser definido de várias maneiras: é um protocolo, uma moeda digital e uma plataforma. É uma combinação de uma rede P2P, protocolos e software que facilita a criação e o uso da moeda digital. Os nós nessa rede P2P se comunicam entre si usando o protocolo Bitcoin.

O Bitcoin é construído sobre décadas de pesquisa. Várias ideias e técnicas da criptografia e da computação distribuída, como árvores de Merkle, funções hash e assinaturas digitais, foram usadas para projetar o Bitcoin. Outras ideias como BitGold, b-money, hashcash e carimbo do tempo criptográfico também forneceram parte da base para a invenção do Bitcoin. Ideias de muitos desses desenvolvimentos foram usadas de forma engenhosa no Bitcoin para criar a primeira moeda verdadeiramente descentralizada. O Bitcoin resolve vários problemas historicamente difíceis relacionados ao dinheiro eletrônico e sistemas distribuídos, incluindo:

* O problema dos generais bizantinos
* Ataques Sybil
* O problema do gasto duplo

O problema do gasto duplo surge quando, por exemplo, um usuário envia moedas para dois usuários diferentes ao mesmo tempo, e elas são verificadas independentemente como transações válidas. O problema do gasto duplo é resolvido no Bitcoin usando um livro-razão distribuído (o blockchain) onde cada transação é registrada permanentemente, e por meio da implementação de um mecanismo de validação e confirmação de transações.

Neste capítulo, examinaremos os diversos atores e componentes da rede Bitcoin e como eles interagem para formá-la:

* Chaves criptográficas
* Endereços
* Transações
* Blockchain
* Mineradores
* Rede
* Carteiras

Primeiramente, examinaremos as chaves e endereços que são usados para representar a propriedade e transferência de valor na rede Bitcoin.

Leitores interessados podem ler mais sobre os futuros de Bitcoin em <http://www.cmegroup.com/trading/bitcoin-futures.html>.

**Chaves criptográficas**

Na rede Bitcoin, a posse de bitcoins e a transferência de valor por meio de transações dependem de chaves privadas, chaves públicas e endereços. A Criptografia de Curvas Elípticas (ECC) é usada para gerar pares de chaves públicas e privadas na rede Bitcoin. Já cobrimos esses conceitos no Capítulo 4, *Criptografia Assimétrica*, e aqui veremos como as chaves privadas e públicas são usadas na rede Bitcoin.

**Chaves privadas no Bitcoin**  
As chaves privadas devem ser mantidas em segurança e normalmente residem apenas do lado do proprietário. As chaves privadas são usadas para assinar digitalmente transações, provando a posse de bitcoins.

As chaves privadas são fundamentalmente números de 256 bits escolhidos aleatoriamente no intervalo especificado pela recomendação da curva ECDSA SECP256K1. Qualquer número de 256 bits escolhido aleatoriamente de 0x1 até 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364140 é uma chave privada válida.

As chaves privadas geralmente são codificadas usando o formato Wallet Import Format (WIF), a fim de torná-las mais fáceis de copiar e usar. É uma forma de representar a chave privada de tamanho completo em um formato diferente. O WIF pode ser convertido em chave privada e vice-versa. Por exemplo, considere a seguinte chave privada:

A3ED7EC8A03667180D01FB4251A546C2B9F2FE33507C68B7D9D4E1FA5714195201

Quando convertida para o formato WIF, ela aparece assim:

L2iN7umV7kbr6LuCmgM27rBnptGbDVc8g4ZBm6EbgTPQXnj1RCZP

Além disso, o formato mini chave privada é às vezes usado para criar uma chave privada com no máximo 30 caracteres, permitindo o armazenamento onde o espaço físico é limitado. Por exemplo, gravação em moedas físicas ou codificação em códigos QR resistentes a danos. O código QR é mais resistente a danos porque mais pontos podem ser usados para correção de erros e menos para codificação da chave privada.

Uma chave privada codificada usando o formato mini chave privada às vezes também é chamada de *minikey*. O primeiro caractere da mini chave privada é sempre a letra maiúscula S. Uma mini chave privada pode ser convertida em uma chave privada de tamanho normal, mas uma chave de tamanho normal existente não pode ser convertida em uma mini chave. Esse formato foi usado nos bitcoins físicos da Casascius. O cliente principal do Bitcoin também permite a criptografia da carteira que contém as chaves privadas.

Leitores interessados podem fazer experimentações usando a ferramenta online disponível em:  
<http://gobittest.appspot.com/PrivateKey>

Os códigos QR usam correção de erros Reed-Solomon. A discussão sobre o mecanismo de correção de erros e seus detalhes subjacentes está fora do escopo deste livro, mas os leitores são incentivados a pesquisar sobre a correção de erros em códigos QR, se tiverem interesse.

**Chaves públicas no Bitcoin**  
Todos os participantes da rede podem ver as chaves públicas no blockchain. As chaves públicas são derivadas das chaves privadas devido à sua relação matemática especial. Uma vez que uma transação assinada com a chave privada é transmitida na rede Bitcoin, as chaves públicas são usadas pelos nós para verificar se a transação foi de fato assinada com a chave privada correspondente. Esse processo de verificação comprova a posse do Bitcoin.

O Bitcoin usa ECC baseado no padrão SECP256K1. Mais especificamente, ele utiliza o Algoritmo de Assinatura Digital com Curvas Elípticas (ECDSA) para garantir que os fundos permaneçam seguros e só possam ser gastos pelo legítimo proprietário. Se precisar revisar os conceitos relevantes da criptografia, você pode consultar o Capítulo 4, *Criptografia Assimétrica*, onde a ECC foi explicada. As chaves públicas podem ser representadas em formato não compactado ou compactado e são, fundamentalmente, coordenadas x e y em uma curva elíptica. A versão compactada das chaves públicas inclui apenas a parte x, já que a parte y pode ser derivada dela.

A razão pela qual a versão compactada das chaves públicas funciona é que, se o gráfico da ECC for visualizado, revela-se que a coordenada y pode estar abaixo ou acima do eixo x, e como a curva é simétrica, apenas a localização no campo primo precisa ser armazenada. Se y for par, então seu valor está acima do eixo x; se for ímpar, está abaixo do eixo x. Isso significa que, ao invés de armazenar x e y como chave pública, apenas x precisa ser armazenado com a informação sobre se y é par ou ímpar.

Inicialmente, o cliente Bitcoin usava chaves não compactadas, mas a partir do cliente Bitcoin Core versão 0.6, chaves compactadas passaram a ser usadas como padrão. Isso resultou em uma redução de quase 50% do espaço usado para armazenar chaves públicas no blockchain.

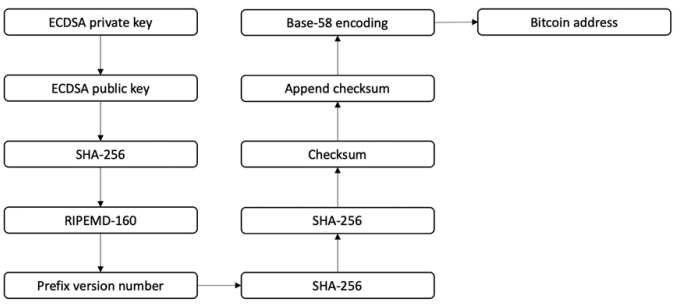
As chaves são identificadas por vários prefixos, descritos a seguir:

* **Chaves públicas não compactadas** usam 0x04 como prefixo. As chaves públicas não compactadas têm 65 bytes de comprimento. Elas são codificadas como inteiros não assinados de 256 bits em big-endian (32 bytes), que são concatenados e finalmente precedidos por um byte 0x04. Isso significa 1 byte para o prefixo 0x04, 32 bytes para o inteiro x e 32 bytes para o inteiro y, totalizando 65 bytes.
* **Chaves públicas compactadas** começam com 0x03 se a parte y de 32 bytes (256 bits) da chave pública for ímpar. Têm 33 bytes de comprimento, sendo 1 byte usado pelo prefixo 0x03 (indicando y ímpar) e 32 bytes usados para armazenar a coordenada x.
* **Chaves públicas compactadas** começam com 0x02 se a parte y de 32 bytes (256 bits) da chave pública for par. Têm 33 bytes de comprimento, sendo 1 byte usado pelo prefixo 0x02 (indicando y par) e 32 bytes usados para armazenar a coordenada x.

Tendo falado sobre chaves privadas e públicas, vamos agora passar para outro aspecto importante do Bitcoin: os endereços derivados das chaves públicas.

**Endereços**

O diagrama a seguir mostra como um endereço é gerado, desde a geração da chave privada até a saída final do endereço Bitcoin:



**Figura 6.1: Geração de endereço no Bitcoin**

No diagrama anterior, há várias etapas:

1. Na primeira etapa, temos uma chave privada ECDSA gerada aleatoriamente.
2. A chave pública é derivada da chave privada ECDSA.
3. A chave pública é hashada usando a função hash criptográfica SHA-256.
4. O hash gerado na etapa 3 é novamente hashado usando a função hash RIPEMD-160.
5. O número da versão é prefixado ao hash RIPEMD-160 gerado na etapa 4.
6. O resultado produzido na etapa 5 é hashado usando a função hash criptográfica SHA-256.
7. SHA-256 é aplicado novamente.
8. Os primeiros 4 bytes do resultado da etapa 7 são o checksum do endereço.
9. Esse checksum é anexado ao hash RIPEMD-160 gerado na etapa 4.
10. A sequência de bytes resultante é codificada em uma string codificada Base58, aplicando a função de codificação Base58.
11. Finalmente, o resultado é um endereço típico de Bitcoin.

**Endereços típicos de Bitcoin**  
Os endereços de Bitcoin têm entre 26 e 35 caracteres e começam com os dígitos 1 ou 3. Um endereço típico de Bitcoin se parece com a seguinte string:

15ccPQG3PQXcj7fhgmWAHN7SQ7JBvfNFGb

Os endereços também são comumente codificados em um código QR para fácil distribuição. O código QR do endereço Bitcoin anterior é mostrado na imagem a seguir:



**Figura 6.2: Código QR do endereço Bitcoin 15ccPQG3PQXcj7fhgmWAHN7SQ7JBvfNFGb**

Atualmente, existem dois tipos de endereços, os tipos mais comumente usados P2PKH e P2SH (ambos definidos mais adiante neste capítulo), começando com os números 1 e 3, respectivamente. Nos primeiros dias, o Bitcoin usava o método de pagamento direto Pay-to-Pubkey, que agora foi substituído pelo P2PKH. Esses tipos serão explicados mais adiante no capítulo. No entanto, o Pay-to-Pubkey direto ainda é usado no Bitcoin para endereços coinbase. Os endereços não devem ser reutilizados; caso contrário, podem surgir problemas de privacidade e segurança.

Evitar a reutilização de endereços contorna parcialmente questões de anonimato, mas o Bitcoin possui outros problemas de segurança também, como maleabilidade de transações, ataques Sybil, ataques de corrida (*race attacks*) e mineração egoísta (*selfish mining*), todos os quais exigem abordagens diferentes para serem resolvidos. A maleabilidade de transações foi resolvida com a atualização soft-fork chamada SegWit do protocolo Bitcoin. Esse conceito será explicado mais adiante no capítulo.



**Figura 6.3: Do site bitaddress.org, uma chave privada e endereço Bitcoin em uma carteira de papel**

Os endereços de Bitcoin são codificados usando a codificação Base58Check. Essa codificação é usada para limitar a confusão entre vários caracteres, como 0/O ou I/l, pois eles podem parecer os mesmos em diferentes fontes. A codificação basicamente pega os arrays de bytes binários e os converte em strings legíveis por humanos. Essa string é composta utilizando um conjunto de 58 símbolos alfanuméricos. Mais explicações e a lógica podem ser encontradas no arquivo fonte base58.h no código-fonte do Bitcoin:

/\*\*

\* Por que base-58 ao invés de codificação padrão base-64?

\* - Não queremos caracteres 0OIl que parecem iguais em algumas fontes e

\* poderiam ser usados para criar dados visualmente idênticos.

\* - Uma string com caracteres não alfanuméricos não é tão facilmente aceita como entrada.

\* - O e-mail geralmente não quebra linha se não houver pontuação para quebrar.

\* - O duplo clique seleciona toda a string como uma palavra se for toda alfanumérica.

\*/

**Endereços avançados de Bitcoin**  
Além dos tipos comuns de endereços no Bitcoin, também existem alguns tipos avançados disponíveis:

* **Endereços de vaidade (Vanity addresses):** Como os endereços de Bitcoin são baseados em codificação Base58, é possível gerar endereços que contenham mensagens legíveis por humanos e sejam personalizados. Um exemplo é 1BasHiry2VoCQCdX6X64oxvKRuf7fW6qGr — observe que o endereço contém o nome *BasHir*. Endereços de vaidade são gerados usando um método de força bruta. Existem vários serviços online que oferecem esse serviço. Mais detalhes sobre o mecanismo e um link para um programa de linha de comando independente estão disponíveis em:  
  <https://en.bitcoin.it/wiki/Vanitygen>.
* **Endereços multiassinatura (Multisig):** Como o nome sugere, esses endereços requerem várias chaves privadas. Em termos práticos, isso significa que, para liberar as moedas, um número definido de assinaturas é necessário. Isso também é conhecido como *M de N multisig*. Aqui, **M** representa o limite ou número mínimo de assinaturas exigidas de **N** chaves para liberar os bitcoins. Lembre-se de que discutimos esse conceito no Capítulo 4, *Criptografia Assimétrica*.

Com esta seção, concluímos nossa introdução aos endereços no Bitcoin. Na próxima seção, apresentaremos as transações do Bitcoin, que são o aspecto mais fundamental e importante do Bitcoin.

**Transações**

As transações estão no cerne do ecossistema Bitcoin. As transações podem ser tão simples quanto apenas enviar alguns bitcoins para um endereço Bitcoin, ou podem ser bastante complexas, dependendo dos requisitos.

Uma transação de Bitcoin é composta por vários elementos:

* **ID da transação:** Um identificador único de transação com 32 bytes de comprimento.
* **Tamanho:** Este é o tamanho da transação em bytes.
* **Peso:** Esta é uma métrica dada para o tamanho do bloco e da transação desde a introdução da versão soft-fork SegWit do Bitcoin.
* **Tempo:** Este é o tempo em que o bloco contendo esta transação foi minerado.
* **Incluído no bloco:** Mostra o número do bloco na blockchain no qual a transação está incluída.
* **Confirmações:** Número de confirmações feitas por mineradores para esta transação.
* **Entrada total:** Número total de entradas na transação.
* **Saída total:** Número total de saídas da transação.
* **Taxas:** Taxa total cobrada.
* **Taxa por byte:** Representa a taxa total dividida pelo número de bytes da transação; por exemplo, 10 Satoshis por byte.
* **Taxa por unidade de peso:** Para transações legadas, é calculada usando o número total de bytes × 4. Para transações SegWit, é calculada combinando marcador SegWit, flag e campo witness como uma unidade de peso, e cada byte dos outros campos como quatro unidades de peso.
* **Índice de entrada:** Número de sequência da entrada.
* **Índice de saída:** Número de sequência da saída.
* **Endereço de saída:** É para onde os bitcoins estão indo.
* **ID da transação anterior:** ID da transação anterior cujo(s) resultado(s) é/são usado(s) como entrada(s) nesta transação.
* **Índice da saída anterior:** Índice da saída anterior que mostra qual saída foi usada como entrada nesta transação.
* **Valor:** Quantidade de bitcoins.
* **Endereço de entrada:** Endereço de onde vem a entrada.
* **Pkscript:** Script de desbloqueio da(s) entrada(s).
* **SigScript:** Assinatura para desbloquear a entrada.
* **Witness:** Witness desta transação — usado apenas no SegWit.

Cada transação é composta por pelo menos uma entrada e uma saída. Entradas podem ser vistas como moedas sendo gastas que foram criadas em uma transação anterior, e saídas como moedas sendo criadas. Se uma transação estiver cunhando (minerando) novas moedas ao invés de gastar moedas criadas anteriormente, então não há entrada, e, portanto, nenhuma assinatura é necessária. Essa transação é chamada de transação coinbase.

**Transações coinbase**  
Uma transação coinbase ou transação de geração é sempre criada por um minerador e é a primeira transação em um bloco. Ela é usada para criar novas moedas. Inclui um campo especial, também chamado de *coinbase*, que atua como entrada para a transação coinbase. Essa transação também permite até 100 bytes de armazenamento arbitrário de dados.

Uma entrada de transação coinbase tem o mesmo número de campos que uma entrada de transação comum, mas a estrutura contém o tamanho dos dados da coinbase e os campos em vez do tamanho e campos do script de desbloqueio. Além disso, não possui um ponteiro de referência para a transação anterior. Essa estrutura é mostrada na tabela a seguir:

| **Tamanho do Campo** | **Descrição** |
| --- | --- |
| Hash da transação | 32 bytes — Definido como todos zeros, pois nenhuma referência de hash é usada |
| Índice da saída | 4 bytes — Definido como 0xFFFFFFFF |
| Tamanho dos dados coinbase | 1–9 bytes — 2–100 bytes |
| Dados | Variável — Quaisquer dados |
| Número de sequência | 4 bytes — Definido como 0xFFFFFFFF |

Por outro lado, se uma transação deve enviar moedas para outro usuário (um endereço Bitcoin), então ela precisa ser assinada pelo remetente com sua chave privada. Nesse caso, uma referência também é necessária para a transação anterior para mostrar a origem das moedas. As moedas são saídas de transações não gastas representadas em Satoshis.

**O ciclo de vida da transação**

Agora, vamos examinar o ciclo de vida de uma transação Bitcoin. As etapas do processo são as seguintes:

1. Um usuário/remetente envia uma transação usando um software de carteira ou alguma outra interface.
2. O software da carteira assina a transação usando a chave privada do remetente.
3. A transação é transmitida para a rede Bitcoin usando um algoritmo de inundação (*flooding*), que é um algoritmo para distribuir dados para cada nó na rede.
4. Nós de mineração (mineradores) que estão ouvindo as transações verificam e incluem essa transação no próximo bloco a ser minerado. Logo antes das transações serem colocadas no bloco, elas são colocadas em um buffer especial de memória chamado *pool de transações*.
5. Em seguida, começa a mineração, que é o processo pelo qual a blockchain é protegida e novas moedas são geradas como recompensa para os mineradores que gastam os recursos computacionais apropriados. Uma vez que um minerador resolve o problema de Prova de Trabalho (*Proof of Work*, PoW), ele transmite o bloco recém-minerado para a rede. Os nós verificam o bloco e o propagam adiante, e as confirmações começam a ser geradas.
6. Finalmente, as confirmações começam a aparecer na carteira do receptor e, após aproximadamente três confirmações, a transação é considerada finalizada e confirmada. No entanto, três a seis é apenas o número recomendado; a transação pode ser considerada final mesmo após a primeira confirmação. A ideia central por trás de esperar por seis confirmações é que a probabilidade de gasto duplo é virtualmente eliminada após seis confirmações.

O Bitcoin, sendo uma moeda digital, possui várias denominações. A menor denominação do Bitcoin é o Satoshi, que equivale a 0,00000001 BTC.

Quando uma transação é criada por um usuário e enviada à rede, ela acaba em uma área especial em cada cliente de software do Bitcoin. Essa área especial é chamada de *pool de transações*. Também conhecida como *memory pool*, os pools de transações são criados na memória local (RAM do computador) por nós (clientes Bitcoin) para manter uma lista temporária de transações que ainda não foram adicionadas a um bloco. Os mineradores pegam as transações desses pools de memória para criar blocos candidatos. Os mineradores selecionam transações do pool depois que elas passam pelas verificações de validade e verificação. Apresentamos como as transações do Bitcoin são validadas na próxima seção.

**Validação de transações**  
Esse processo de verificação é realizado pelos nós do Bitcoin. Há três coisas principais que os nós verificam ao validar uma transação:

1. Que as entradas da transação não foram gastas anteriormente. Esta etapa de validação impede o gasto duplo ao verificar se as entradas da transação ainda não foram usadas por outra pessoa.
2. Que a soma das saídas da transação não é maior do que a soma total das entradas. No entanto, as somas de entrada e saída podem ser iguais, ou a soma da entrada (valor total) pode ser maior do que a soma total das saídas. Essa verificação assegura que não sejam criados novos bitcoins do nada.
3. Que as assinaturas digitais são válidas, o que garante que o script é válido.

Para enviar transações na rede Bitcoin, o remetente precisa pagar uma taxa aos mineradores. A seleção de quais transações serão escolhidas é baseada na taxa e na posição delas na ordem de transações do pool. Os mineradores preferem selecionar transações com taxas mais altas.

**Taxas de transação**

As taxas de transação são cobradas pelos mineradores. A taxa cobrada depende do tamanho e do peso da transação. As taxas de transação são calculadas subtraindo a soma das saídas da soma das entradas:

**taxa = soma(entradas) – soma(saídas)**

As taxas são usadas como um incentivo para os mineradores, a fim de encorajá-los a incluir as transações dos usuários no bloco que estão criando. Todas as transações acabam no pool de memória, de onde os mineradores selecionam as transações com base em sua prioridade para inclusão no bloco proposto. O cálculo de prioridade será apresentado mais adiante neste capítulo; no entanto, do ponto de vista da taxa de transação, uma transação com taxa mais alta será escolhida mais rapidamente pelos mineradores. Existem regras diferentes com base nas quais a taxa é calculada para vários tipos de ações, como envio de transações, inclusão em blocos e retransmissão pelos nós.

As taxas não são fixadas pelo protocolo do Bitcoin e não são obrigatórias; mesmo uma transação sem taxa será processada eventualmente, mas pode demorar muito tempo. No entanto, isso já não é mais prático devido ao alto volume de transações e investidores competindo na rede Bitcoin; portanto, é aconselhável sempre fornecer uma taxa. O tempo necessário para confirmação de transações geralmente varia de 10 minutos a mais de 12 horas em alguns casos. O tempo de transação também depende da atividade da rede. Se a rede estiver muito ocupada, naturalmente as transações levarão mais tempo para serem processadas.

Em determinados momentos do passado, as taxas de Bitcoin estavam tão altas que, mesmo para transações pequenas, uma taxa elevada era cobrada. Isso se devia ao fato de que os mineradores são livres para escolher quais transações desejam verificar e adicionar a um bloco, e naturalmente selecionam as que têm taxas mais altas. Um alto número de usuários criando milhares de transações também contribuiu para causar essa situação de taxas elevadas, pois as transações competiam para serem selecionadas primeiro, e os mineradores escolhiam as com maiores taxas. Essa taxa também é geralmente estimada e calculada automaticamente pelo software de carteira do Bitcoin antes de enviar a transação.

Mineração e mineradores são conceitos que veremos um pouco mais adiante neste capítulo, na seção sobre *Mineração*.

**A estrutura de dados da transação**

Uma transação, em alto nível, contém metadados, entradas e saídas. As transações são combinadas para criar o corpo de um bloco. A estrutura geral de dados da transação é mostrada na tabela a seguir:

| **Campo** | **Tamanho** | **Descrição** |
| --- | --- | --- |
| Número da versão | 4 bytes | Especifica as regras a serem usadas pelos mineradores e nós para o processamento da transação. Existem duas versões de transações, ou seja, 1 e 2. |
| Flag | 2 bytes ou nenhum | Sempre 0001 ou ausente, usado para indicar a presença de dados witness. |
| Contador de entrada | 1–9 bytes | O número (um inteiro positivo) de entradas incluídas na transação. |
| Lista de entradas | Variável | Cada entrada é composta por vários campos. Estes incluem:• O hash da transação anterior• O índice da transação anterior• Comprimento do script da transação• Script da transação• Número de sequência |
| Contador de saída | 1–9 bytes | Um inteiro positivo representando o número de saídas. |
| Lista de saídas | Variável | Saídas incluídas na transação. Este campo representa o(s) destinatário(s) dos bitcoins. |
| Lista de witnesses | Baseado no número de entradas ou nenhum | Witnesses — 1 para cada entrada. Ausente se o campo Flag estiver ausente. |
| Tempo de bloqueio (Lock time) | 4 bytes | Este campo define o tempo mais cedo em que uma transação se torna válida. É um timestamp Unix ou a altura do bloco. |

A seguir, temos um exemplo de transação decodificada:



Como mostrado na transação decodificada anterior, há várias estruturas que compõem uma transação. Todos esses elementos serão descritos agora.

**Metadados**  
Esta parte da transação contém valores como o tamanho da transação, o número de entradas e saídas, o hash da transação e um campo *locktime*. Cada transação possui um prefixo especificando o número da versão. Esses campos são mostrados no exemplo anterior como locktime, size, weight e version.

**Entradas**  
Geralmente, cada entrada (vin) gasta uma saída anterior. Cada saída é considerada uma *saída de transação não gasta* (UTXO — *Unspent Transaction Output*) até que uma entrada a consuma. Uma UTXO pode ser gasta como entrada de uma nova transação. A estrutura de dados da entrada da transação é explicada na tabela a seguir:

| **Campo** | **Tamanho** | **Descrição** |
| --- | --- | --- |
| Hash da transação | 32 bytes | O hash da transação anterior com a UTXO |
| Índice da saída | 4 bytes | Este é o índice da saída da transação anterior, como a UTXO a ser gasta |
| Tamanho do script | 1–9 bytes | O tamanho do script de desbloqueio |
| Script de desbloqueio | Variável | O script de entrada (*ScriptSig*), que satisfaz os requisitos do script de bloqueio |
| Número de sequência | 4 bytes | Geralmente desativado ou contém o tempo de bloqueio — o valor desativado é representado por 0xFFFFFFFF |

Na transação decodificada anterior, as entradas são definidas sob a seção "inputs": [.

**Saídas**  
As saídas (vout) têm três campos e contêm instruções para o envio de bitcoins. O primeiro campo contém a quantidade de Satoshis, enquanto o segundo campo contém o tamanho do script de bloqueio. Por fim, o terceiro campo contém um script de bloqueio que define as condições que devem ser atendidas para que a saída possa ser gasta. Mais informações sobre gastos de transações usando scripts de bloqueio e desbloqueio e geração de saídas são discutidas mais adiante nesta seção.

A estrutura de dados da saída da transação é explicada na tabela a seguir:

| **Campo** | **Tamanho** | **Descrição** |
| --- | --- | --- |
| Valor | 8 bytes | O número total (em inteiros positivos) de Satoshis a serem transferidos |
| Tamanho do script | 1–9 bytes | Tamanho do script de bloqueio |
| Script de bloqueio | Variável | Script de saída (*ScriptPubKey*) |

Na transação decodificada anterior, duas saídas são mostradas sob a seção "vout":[.

**Verificação**  
A verificação é realizada usando a linguagem de script do Bitcoin, onde as assinaturas criptográficas das transações são verificadas quanto à validade, todas as entradas e saídas são verificadas, e a soma de todas as entradas deve ser igual ou maior do que a soma de todas as saídas.

Tendo coberto o ciclo de vida da transação e sua estrutura de dados, vamos agora falar sobre os scripts usados para realizar essas transações.

**A linguagem Script**  
O Bitcoin utiliza uma linguagem simples baseada em pilha chamada *Script* para descrever como os bitcoins podem ser gastos e transferidos. Ela não é Turing-completa e não possui laços (*loops*) para evitar quaisquer efeitos indesejáveis de scripts longos ou travados na rede Bitcoin. Essa linguagem de script é baseada em uma sintaxe semelhante à linguagem de programação Forth e utiliza notação polonesa reversa, na qual cada operando é seguido por seus operadores. Ela é avaliada da esquerda para a direita usando uma pilha do tipo “último a entrar, primeiro a sair” (*LIFO*).

Os scripts são compostos por dois componentes: elementos e operações. Os scripts utilizam várias operações (*opcodes*) ou instruções para definir seu comportamento. Os elementos representam simplesmente dados, como assinaturas digitais. Os *opcodes* também são conhecidos como palavras, comandos ou funções. Versões anteriores do software do nó Bitcoin tinham alguns *opcodes* que não são mais usados devido a falhas descobertas em seu design.

As várias categorias de *opcodes* de script incluem: constantes, controle de fluxo, pilha, lógica bit a bit, corte (*splice*), aritmética, criptografia e tempo de bloqueio (*lock time*).

Um script de transação é avaliado combinando o *ScriptSig* e o *ScriptPubKey*:

* **ScriptSig** é o script de desbloqueio, fornecido pelo usuário que deseja desbloquear a transação.
* **ScriptPubKey** é o script de bloqueio, parte da saída da transação, e especifica as condições que precisam ser satisfeitas para desbloquear e gastar a saída.

Veremos uma execução de script em detalhes em breve.

**Opcodes**

Em um computador, um *opcode* é uma instrução para realizar alguma operação. Por exemplo, ADD é um *opcode*, usado para adição de inteiros em CPUs Intel e várias outras arquiteturas. Da mesma forma, no design do Bitcoin, foram introduzidos *opcodes* que realizam várias operações relacionadas à verificação de transações do Bitcoin.

Descrições de alguns dos *opcodes* mais comumente usados estão listadas na tabela a seguir, extraída do Guia do Desenvolvedor Bitcoin:

| **Opcode** | **Descrição** |
| --- | --- |
| **OP\_CHECKSIG** | Recebe uma chave pública e uma assinatura e valida a assinatura do hash da transação. Se corresponder, então TRUE é empilhado na pilha; caso contrário, FALSE. |
| **OP\_EQUAL** | Retorna 1 se as entradas forem exatamente iguais; caso contrário, retorna 0. |
| **OP\_DUP** | Duplica o item no topo da pilha. |
| **OP\_HASH160** | A entrada é hashada duas vezes, primeiro com SHA-256 e depois com RIPEMD-160. |
| **OP\_VERIFY** | Marca a transação como inválida se o valor no topo da pilha não for verdadeiro. |
| **OP\_EQUALVERIFY** | Igual ao OP\_EQUAL, mas executa OP\_VERIFY em seguida. |
| **OP\_CHECKMULTISIG** | Essa instrução recebe a primeira assinatura e a compara com cada chave pública até encontrar uma correspondência, repetindo o processo até todas as assinaturas serem verificadas. Se todas forem válidas, retorna 1; caso contrário, 0. |
| **OP\_HASH256** | A entrada é hashada duas vezes com SHA-256. |
| **OP\_MAX** | Retorna o maior valor entre duas entradas. |

Existem muitos *opcodes* na linguagem de script do Bitcoin, e cobrir todos eles está fora do escopo deste livro. No entanto, todos os *opcodes* estão declarados no arquivo script.h no cliente de referência do Bitcoin, disponível em:  
<https://github.com/Bitcoin/Bitcoin/blob/0cda5573405d75d695aba417e8f22f1301ded001/src/script/script.h#L53>

Há vários scripts padrão disponíveis no Bitcoin para lidar com a verificação e a transferência de valor da origem ao destino. Esses scripts variam de muito simples a bastante complexos, dependendo dos requisitos da transação.

**Scripts de transações padrão**  
Transações padrão são avaliadas usando os testes IsStandard() e IsStandardTx(), e somente aquelas que passam nesses testes são permitidas para serem transmitidas ou mineradas na rede Bitcoin.

No entanto, transações não padrão também são permitidas na rede, desde que passem nas verificações de validade:

* **Pay-to-Public-Key Hash (P2PKH):** P2PKH é o tipo de transação mais comumente usado e é utilizado para enviar transações para endereços Bitcoin. O formato desse tipo de transação é mostrado a seguir:

ScriptPubKey: OP\_DUP OP\_HASH160 <pubKeyHash> OP\_EQUALVERIFY OP\_CHECKSIG

ScriptSig: <sig> <pubKey>

Os parâmetros *ScriptPubKey* e *ScriptSig* são concatenados e executados. Um exemplo será apresentado em breve nesta seção, onde isso será explicado em mais detalhes.

* **Pay-to-Script Hash (P2SH):** P2SH é usado para enviar transações para um hash de script (isto é, endereços começando com 3) e foi padronizado no BIP16. Além de passar o script, o script de resgate (*redeem script*) também é avaliado e deve ser válido. O modelo é mostrado a seguir:

ScriptPubKey: OP\_HASH160 <redeemScriptHash> OP\_EQUAL

ScriptSig: [<sig>…<sign>] <redeemScript>

* **MultiSig (Pay to MultiSig):** O script de transação multiassinatura do tipo M de N é um tipo de script complexo em que é possível construir um script que exige várias assinaturas para ser válido e liberar uma transação. Várias transações complexas como *escrow* (custódia) e depósitos podem ser construídas usando esse script. O modelo é mostrado aqui:

ScriptPubKey: <m> <pubKey> [<pubKey> . . . ] <n> OP\_CHECKMULTISIG

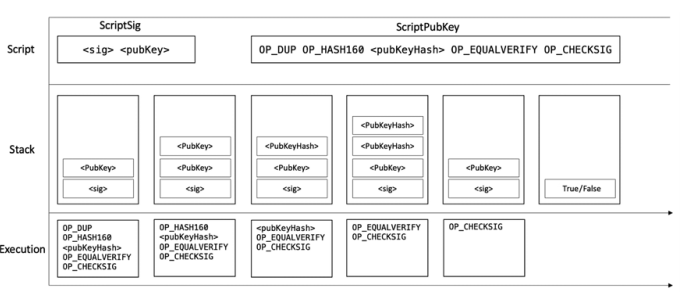
ScriptSig: 0 [<sig> . . . <sign>]

Multiassinatura bruta (Raw multisig) está obsoleta, e a multiassinatura agora geralmente faz parte do script de resgate P2SH, mencionado no ponto anterior.

* **Dados nulos / OP\_RETURN:** Esse script é usado para armazenar dados arbitrários no blockchain mediante o pagamento de uma taxa. O limite da mensagem é de 40 bytes. A saída desse script é irredimível, pois o OP\_RETURN falhará na validação em qualquer caso. *ScriptSig* não é necessário nesse caso. O modelo é muito simples e é mostrado a seguir:

OP\_RETURN <data>

A execução do script P2PKH é mostrada no seguinte diagrama:



**Figura 6.4: Execução do script P2PKH**

No diagrama anterior, temos um script padrão P2PKH apresentado na parte superior, que mostra tanto a parte de desbloqueio (*ScriptSig*) quanto a de bloqueio (*ScriptPubKey*) do script.

O script de desbloqueio é composto pelos elementos <sig> e <pubkey>, que fazem parte de todas as entradas de transação. O script de desbloqueio satisfaz as condições exigidas para consumir a saída. O script de bloqueio define as condições que precisam ser atendidas para gastar os bitcoins. As transações são autorizadas pela execução conjunta de ambas as partes.

Focando novamente na Figura 6.4, vemos que no meio há uma visualização da pilha onde os elementos de dados são empilhados e desempilhados. Na parte inferior, mostramos a execução do script. Este diagrama mostra a execução passo a passo do script com seus resultados na pilha.

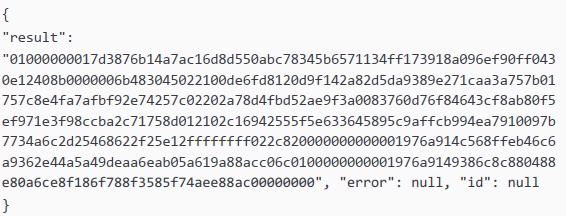
Agora vejamos como este script é executado:

1. No primeiro passo dos elementos de dados, <sig> e <pubkey> são colocados na pilha.
2. O item do topo da pilha, <pubkey>, é duplicado devido à instrução OP\_DUP, que duplica o item superior.
3. Após isso, a instrução OP\_HASH160 é executada, produzindo o hash de <pubkey>, que está no topo da pilha.
4. <pubkeyhash> é então empurrado para a pilha. Neste ponto, temos dois hashes na pilha: o gerado como resultado da execução de OP\_HASH160 sobre <pubkey> do script de desbloqueio, e o outro fornecido pelo script de bloqueio.
5. Agora a instrução OP\_EQUALVERIFY é executada e verifica se os dois elementos no topo (isto é, os hashes) são iguais ou não. Se forem iguais, o script continua; caso contrário, ele falha.
6. Por fim, OP\_CHECKSIG é executado para verificar a validade das assinaturas dos dois elementos superiores da pilha. Se a assinatura for válida, a pilha conterá o valor verdadeiro (True), isto é, 1; caso contrário, falso (False), ou seja, 0.

Todas as transações são codificadas em formato hexadecimal antes de serem transmitidas pela rede Bitcoin. Uma transação de exemplo pode ser recuperada usando o comando bitcoin-cli no nó Bitcoin executando na rede principal da seguinte forma:

$ bitcoin-cli getrawtransaction "d28ca5a59b2239864eac1c96d3fd1c23b747f0ded8f5af0161bae8a616b56a1d"

A saída é mostrada em formato hexadecimal:



O sistema de script é bastante limitado e só pode ser usado para programar uma única coisa — a transferência de bitcoins de um endereço para outros. No entanto, há certa flexibilidade ao criar esses scripts, o que permite impor certas condições para o gasto dos bitcoins. Esse conjunto de condições pode ser considerado uma forma básica de contrato financeiro.

**Contratos**

Contratos são scripts Bitcoin que usam o blockchain do Bitcoin para impor um acordo financeiro. Esta é uma definição simples, mas com consequências de longo alcance, pois permite que os usuários criem programaticamente contratos complexos que podem ser usados em muitos cenários do mundo real. Contratos permitem o desenvolvimento de plataformas totalmente descentralizadas, independentes e de risco reduzido ao impor programaticamente diferentes condições para o desbloqueio de bitcoins. Com as garantias de segurança fornecidas pelo blockchain do Bitcoin, é quase impossível contornar essas condições.

Como alternativa ao software cliente instalado localmente, também pode-se usar um serviço online disponível aqui:  
<https://chainquery.com/bitcoin-cli/getrawtransaction>

Observe que isso não é o mesmo que *smart contracts*, que permitem a escrita de programas arbitrários no blockchain. Discutiremos isso mais adiante no Capítulo 8, *Contratos Inteligentes*.

Vários contratos, como *escrow* (custódia), arbitragem e canais de micropagamentos podem ser construídos usando a linguagem de script do Bitcoin. A implementação atual da linguagem Script é minimalista, mas ainda assim é possível desenvolver diversos tipos de contratos complexos. Por exemplo, poderíamos estabelecer que a liberação de fundos só será permitida quando várias partes assinarem a transação, ou que os fundos só poderão ser liberados após certo tempo decorrido. Ambos os cenários podem ser realizados usando *multisig* e opções de tempo de bloqueio da transação.

Embora a linguagem de script do Bitcoin possa ser usada para criar contratos complexos, ela é bastante limitada e está longe de ser equivalente aos contratos inteligentes, que são construções Turing-completas e permitem o desenvolvimento de programas arbitrários. No entanto, recentemente houve alguns avanços nessa área. Uma nova linguagem de contratos inteligentes para o Bitcoin foi anunciada, chamada **Miniscript**, que permite uma abordagem mais estruturada para escrever scripts no Bitcoin. Embora o Script do Bitcoin suporte combinações de diferentes condições de gasto, como *time locks* e *hash locks*, não é fácil analisar scripts existentes ou construir novos scripts complexos. O **Miniscript** torna mais fácil escrever regras de gasto complexas. Ele também facilita a verificação da correção do script. Atualmente, o Miniscript oferece suporte a **P2WSH** e **P2SH-P2WSH**, e suporte limitado a **P2SH**.

**Erros em transações**

Embora a construção e validação de transações sejam geralmente processos seguros e sólidos, algumas vulnerabilidades existem no Bitcoin. A seguir, apresentamos duas vulnerabilidades importantes no Bitcoin que foram exploradas de forma notória:

* **Maleabilidade de transação** é um ataque no Bitcoin que foi introduzido devido a um bug na implementação do Bitcoin. Por causa desse bug, tornou-se possível para um adversário alterar o ID da transação, resultando em um cenário onde parece que uma certa transação não foi executada. Isso pode permitir cenários onde depósitos ou saques duplos possam ocorrer. Isso foi corrigido por meio do BIP62, que corrigiu várias causas da maleabilidade. Mais detalhes podem ser encontrados aqui:  
  <https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki>
* **Transbordamento de valor (value overflow):** Este incidente é um dos eventos mais conhecidos na história do Bitcoin. Em 15 de agosto de 2010, foi descoberta uma transação que criou aproximadamente 184 bilhões de bitcoins. Esse problema ocorreu devido a um bug de transbordamento de inteiros, onde o campo de valor no código do Bitcoin foi definido como um inteiro com sinal em vez de um inteiro sem sinal. Esse bug significava que o valor também poderia ser negativo, e resultava em uma situação onde as saídas eram tão grandes que o valor total resultava em um transbordamento. Para a lógica de validação no código do Bitcoin, tudo parecia estar correto, e parecia que a taxa também era positiva (após o transbordamento). Esse bug foi corrigido rapidamente por meio de um soft fork (mais sobre isso na seção Blockchain).

Mais informações sobre o Miniscript estão disponíveis em: <http://bitcoin.sipa.be/miniscript/>  
Mais informações sobre a vulnerabilidade de transbordamento estão disponíveis em: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-5139>

A pesquisa em segurança em geral, e especificamente no mundo do Bitcoin (criptomoedas/blockchain), é um assunto fascinante; talvez alguns leitores se inspirem nos exemplos de vulnerabilidade nos links a seguir e embarquem em uma jornada para descobrir mais vulnerabilidades.

Um exemplo de um problema crítico recentemente resolvido no Bitcoin, que permaneceu não descoberto por bastante tempo, pode ser encontrado em: <https://bitcoincore.org/en/2019/11/08/CVE-2017-18350/>

Outro exemplo de um bug crítico de inflação e negação de serviço, descoberto em 17 de setembro de 2018 e corrigido rapidamente, está detalhado em: <https://bitcoincore.org/en/2018/09/20/notice/>

Talvez ainda existam bugs que ainda não foram descobertos!

**Blockchain**  
A blockchain do Bitcoin pode ser definida como um livro-razão público e distribuído contendo um registro ordenado, com carimbo de data e imutável de todas as transações da rede Bitcoin. As transações são coletadas pelos mineradores e agrupadas em blocos para mineração. Cada bloco é identificado por um hash e é vinculado ao seu bloco anterior referenciando o hash do bloco anterior em seu cabeçalho.

**Estrutura**

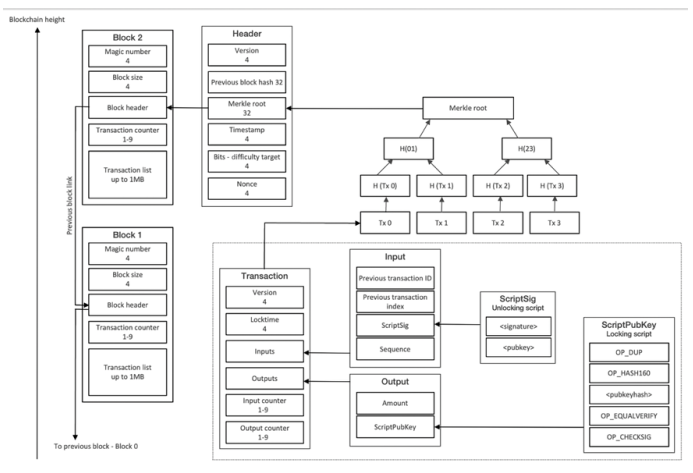
A estrutura de dados de um bloco do Bitcoin é mostrada na tabela a seguir:

| **Campo** | **Tamanho** | **Descrição** |
| --- | --- | --- |
| Tamanho do bloco | 4 bytes | O tamanho do bloco. |
| Cabeçalho do bloco | 80 bytes | Inclui os campos do cabeçalho do bloco, descritos na próxima seção. |
| Contador de transações | Variável | Contém o número total de transações no bloco, incluindo a transação coinbase. O tamanho varia de 1 a 9 bytes. |
| Transações | Variável | Todas as transações no bloco. |

O cabeçalho do bloco mencionado na tabela anterior é uma estrutura de dados que contém vários campos. Essa estrutura é mostrada na tabela a seguir:

| **Campo** | **Tamanho** | **Descrição** |
| --- | --- | --- |
| Versão | 4 bytes | O número da versão do bloco que dita as regras de validação a serem seguidas. |
| Hash do cabeçalho do bloco anterior | 32 bytes | Um hash duplo SHA-256 do cabeçalho do bloco anterior. |
| Hash da raiz de Merkle | 32 bytes | Um hash duplo SHA-256 da árvore de Merkle de todas as transações incluídas no bloco. |
| Timestamp | 4 bytes | Este campo contém o tempo aproximado de criação do bloco no formato de tempo Unix. Mais precisamente, é o tempo no qual o minerador começou a hashear o cabeçalho (o tempo do local do minerador). |
| Alvo de dificuldade | 4 bytes | A dificuldade atual da rede/bloco. |
| Nonce | 4 bytes | Um número que os mineradores alteram repetidamente para produzir um hash inferior ao alvo de dificuldade. |

Como mostrado no diagrama a seguir, uma blockchain é uma cadeia de blocos onde cada bloco é vinculado ao bloco anterior referenciando o hash do cabeçalho do bloco anterior. Esse encadeamento garante que nenhuma transação possa ser modificada a menos que o bloco que a registra e todos os blocos subsequentes também sejam modificados. O primeiro bloco não é vinculado a nenhum bloco anterior e é conhecido como o **bloco gênese**:



**Figura 6.5: Uma visualização de uma blockchain, bloco, cabeçalho de bloco, transações e scripts**

À esquerda, são mostrados os blocos começando de baixo para cima. Cada bloco contém transações e cabeçalhos de bloco, que são ampliados à direita. Na parte superior, primeiro, o cabeçalho do bloco é ampliado para mostrar vários elementos dentro dele. Em seguida, à direita, o elemento *raiz de Merkle* do cabeçalho do bloco é mostrado ampliado, revelando como a raiz de Merkle é construída.

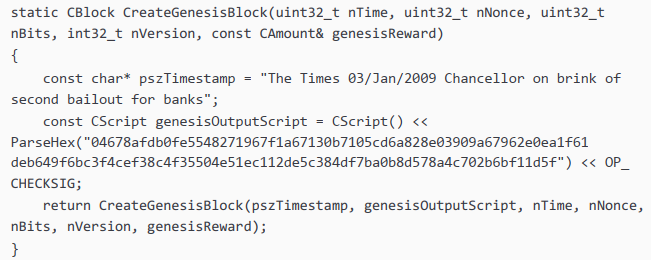
Mais abaixo no diagrama, as transações também são ampliadas para mostrar sua estrutura e os elementos que contêm. Observe também que as transações são detalhadas ainda mais para mostrar como são os scripts de bloqueio e desbloqueio. O tamanho (em bytes) de cada campo do bloco, cabeçalho e transação também é exibido como um número abaixo do nome do campo.

**O bloco gênese**

O bloco gênese é o primeiro bloco na blockchain do Bitcoin. Ele foi minerado por Satoshi Nakamoto em 3 de janeiro de 2009. Sua altura de bloco é 0, e ele contém a seguinte mensagem codificada como parte do script coinbase da transação:

“The Times 03/Jan/2009 Chancellor on brink of second bailout for banks”

A seguinte representação do código do bloco genesis pode ser encontrada no arquivo chainparams.cpp disponível em <https://github.com/Bitcoin/Bitcoin/blob/master/src/chainparams.cpp>:

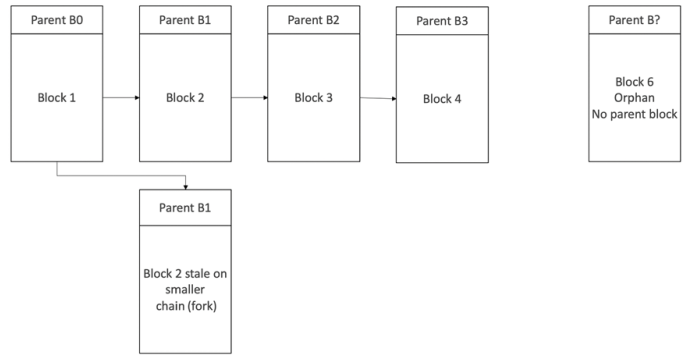


A altura do bloco é o número de blocos anteriores a um bloco específico na blockchain. O PoW é usado para proteger a blockchain. Cada bloco contém uma ou mais transações, a primeira das quais é a transação da Coinbase. Há uma condição especial para transações da Coinbase que impede que sejam gastas até que pelo menos 100 blocos tenham passado, para evitar uma situação em que o bloco possa ser declarado obsoleto posteriormente.

**Blocos obsoletos e órfãos**

Blocos obsoletos são blocos antigos que já foram minerados. Mineradores que continuam trabalhando nesses blocos devido a uma bifurcação, onde a cadeia mais longa (cadeia principal) já progrediu além desses blocos, são considerados como trabalhando em um bloco obsoleto. Em outras palavras, esses blocos existem em uma cadeia mais curta e não fornecerão nenhuma recompensa aos seus mineradores. Blocos órfãos são um conceito ligeiramente diferente.

Seus blocos pais são desconhecidos. Como seus pais são desconhecidos, eles não podem ser validados. Esse problema ocorre quando dois ou mais mineradores descobrem um bloco quase ao mesmo tempo. Esses são blocos válidos e foram descobertos corretamente em algum momento no passado, mas agora não fazem mais parte da cadeia principal. A razão para isso ocorrer é que, se houver dois blocos descobertos quase ao mesmo tempo, aquele com uma quantidade maior de Prova de Trabalho (PoW) será aceito e aquele com uma quantidade menor de trabalho será rejeitado. Semelhante aos blocos obsoletos, eles não fornecem nenhuma recompensa aos seus mineradores. Podemos ver esse conceito visualmente no diagrama a seguir:



**Bifurcações (Fork):**

Na introdução anterior aos blocos obsoletos, introduzimos um novo termo: bifurcação. Uma bifurcação é uma condição que ocorre quando existem duas versões diferentes da blockchain. É aceitável em algumas condições e prejudicial em outras. Devido à natureza distribuída do Bitcoin, bifurcações na rede podem ocorrer inerentemente. Em casos em que dois nós anunciam simultaneamente um bloco válido, isso pode resultar em uma situação em que há duas blockchains com transações diferentes. Essa é uma situação indesejável, mas pode ser resolvida pela rede Bitcoin apenas aceitando a cadeia mais longa.

Nesse caso, a cadeia menor será considerada órfã. Se um adversário conseguir controlar 51% da taxa de hash da rede (poder computacional), ele poderá impor sua própria versão do histórico de transações. Existem diferentes tipos de bifurcações que podem ocorrer em uma blockchain:

• Bifurcações temporárias

• Bifurcações suaves

• Bifurcações rígidas

Bifurcações temporárias podem ocorrer naturalmente no protocolo Bitcoin quando dois blocos são minerados aproximadamente ao mesmo tempo e uma estrutura em forma de árvore começa a emergir com dois ou mais ramos. No entanto, devido à regra de escolha da cadeia, que impõe a seleção da cadeia mais longa entre todos os ramos, a cadeia mais curta é removida. A cadeia mais longa significa a cadeia que possui o maior trabalho acumulado (dificuldade).

No caso de uma bifurcação sof, um cliente que optar por não atualizar para a versão mais recente que suporta o protocolo atualizado ainda poderá trabalhar e operar normalmente. Nesse caso, blocos novos e anteriores são aceitáveis, tornando a bifurcação sof compatível com versões anteriores. Os mineradores precisam apenas atualizar para o novo software cliente sof-fork para utilizar as novas regras do protocolo. Atualizações planejadas não necessariamente criam bifurcações, pois todos os usuários já devem ter atualizado o software.

Uma bifurcação dura, por outro lado, invalida blocos válidos anteriormente e exige que todos os usuários atualizem. Novos tipos de transação são às vezes adicionados como um soft fork, e quaisquer alterações, como alterações na estrutura do bloco ou grandes mudanças no protocolo, resultam em um hard fork. À medida que o Bitcoin evolui e novas atualizações e inovações são introduzidas, a versão associada aos blocos também muda. Essas versões introduzem vários parâmetros de segurança e novos recursos.

**Propriedades**:

O Bitcoin é uma cadeia de blocos em constante crescimento e está aumentando de tamanho. O tamanho atual da blockchain do Bitcoin é de aproximadamente 432 GB. O gráfico em https://www.blockchain.com/charts/blocks-size mostra o tamanho atual e histórico. À medida que a cadeia cresce e mais mineradores são adicionados à rede, a dificuldade da rede também aumenta. A dificuldade da rede se refere a uma medida de quão difícil é encontrar um novo bloco, ou em outras palavras, quão difícil é encontrar um hash abaixo do alvo fornecido.

Novos blocos são adicionados à blockchain aproximadamente a cada 10 minutos, e a dificuldade da rede é ajustada dinamicamente a cada 2.016 blocos (aproximadamente a cada duas semanas) para manter uma adição constante de novos blocos à rede.

A dificuldade da rede é calculada usando a seguinte fórmula:



A dificuldade anterior representa o valor-alvo antigo, e 2016 \* 10 é o tempo total necessário para gerar os 2.016 blocos anteriores. A dificuldade de rede significa essencialmente a dificuldade dos mineradores em encontrar um novo bloco; ou seja, o quão difícil é o quebra-cabeça de hashing agora. Na próxima seção, discutiremos a mineração, que explicará como o quebra-cabeça de hashing é resolvido.

**Mineração**

A mineração é um processo pelo qual novos blocos são adicionados à blockchain. Este processo consome muitos recursos devido aos requisitos da Prova de Trabalho (PoW), onde os mineradores competem para encontrar um número menor que o alvo de dificuldade da rede. Essa dificuldade em encontrar o valor correto (também às vezes chamada de quebra-cabeça matemático) existe para garantir que os mineradores tenham gasto os recursos exigidos antes que um novo bloco proposto possa ser aceito. Os mineradores cunham novas moedas resolvendo o problema de PoW, também conhecido como o problema de inversão parcial de hash. Este processo consome uma grande quantidade de recursos, incluindo poder computacional e eletricidade. Este processo também protege o sistema contra fraudes e ataques de gasto duplo, ao mesmo tempo em que adiciona mais moeda virtual ao ecossistema do Bitcoin.

Aproximadamente um novo bloco é criado (minerado) a cada 10 minutos para controlar a frequência da geração de bitcoins. Essa frequência precisa ser mantida pela rede Bitcoin. Ela está codificada no cliente Bitcoin Core para controlar a “oferta monetária”.

Aproximadamente 144 blocos, ou seja, 1.728 bitcoins, são gerados por dia. O número de moedas reais pode variar por dia; no entanto, o número de blocos permanece, em média, em 144 por dia. O fornecimento de Bitcoin também é limitado. Em 2140, todos os 21 milhões de bitcoins terão finalmente sido criados, e nenhum novo bitcoin poderá ser criado após isso. Os mineradores de Bitcoin, no entanto, ainda poderão lucrar com o ecossistema cobrando taxas de transação.

Uma vez que um nó se conecta à rede Bitcoin, há várias tarefas que um minerador realiza:

1. **Sincronizar com a rede**: Quando um novo nó entra na rede Bitcoin, ele baixa a blockchain solicitando blocos históricos de outros nós. Isso é mencionado aqui no contexto do minerador, mas não é necessariamente uma tarefa exclusiva de mineradores. Outros nós, como nós completos que não necessariamente mineram, também se sincronizam com a rede para atualizar seu banco de dados local da blockchain.
2. **Validação de transações**: Transações transmitidas na rede são validadas por nós completos ao verificar e validar assinaturas e saídas.
3. **Validação de blocos**: Mineradores e nós completos podem começar a validar blocos recebidos avaliando-os com base em certas regras. Isso inclui a verificação de cada transação no bloco juntamente com a verificação do valor do nonce.
4. **Criação de um novo bloco**: Mineradores propõem um novo bloco combinando transações transmitidas na rede após validá-las.
5. **Executar PoW**: Esta tarefa é o núcleo do processo de mineração, e é aqui que os mineradores encontram um bloco válido resolvendo um quebra-cabeça computacional. O cabeçalho do bloco contém um campo nonce de 32 bits, e os mineradores devem variar repetidamente o nonce até que o hash resultante seja menor que um alvo predefinido.
6. **Obter recompensa**: Uma vez que um nó resolve o quebra-cabeça de hash (PoW), ele imediatamente transmite os resultados, e outros nós o verificam e aceitam o bloco. Há uma pequena chance de que o bloco recém-cunhado não seja aceito por outros mineradores na rede devido a um conflito com outro bloco encontrado aproximadamente ao mesmo tempo, mas, uma vez aceito, o minerador é recompensado com bitcoins e quaisquer taxas de transação associadas.

Os mineradores são recompensados com novas moedas se e quando descobrem novos blocos resolvendo a PoW. Os mineradores recebem taxas de transação em troca das transações em seus blocos propostos. Novos blocos são criados em uma taxa fixa aproximada de a cada 10 minutos.

A taxa de criação de novos bitcoins diminui em 50% a cada 210.000 blocos, o que equivale a aproximadamente 4 anos. Quando o Bitcoin começou em 2009, a recompensa de mineração era de 50 bitcoins. Após cada 210.000 blocos, a recompensa do bloco é reduzida pela metade. Em novembro de 2012, caiu para 25 bitcoins. Atualmente, desde maio de 2020, é de 6,25 bitcoins por bloco. A próxima redução pela metade está prevista para ocorrer no início de 2024, o que reduzirá a recompensa para 3,125 BTC. Esse mecanismo está codificado no Bitcoin para regular e controlar a inflação e limitar a oferta de bitcoins. Para que os mineradores ganhem a recompensa, eles devem mostrar que resolveram o quebra-cabeça computacional. Isso é chamado de PoW.

**Prova de Trabalho (PoW)**

Esta é a prova de que recursos computacionais suficientes foram gastos para construir um bloco válido. A PoW baseia-se na ideia de que um nó aleatório é selecionado toda vez para criar um novo bloco. Neste modelo, os nós competem proporcionalmente à sua capacidade de computação para serem selecionados. A fórmula a seguir resume o requisito de PoW no Bitcoin:

**H (N || P\_hash || Tx || Tx || … Tx) < Alvo**

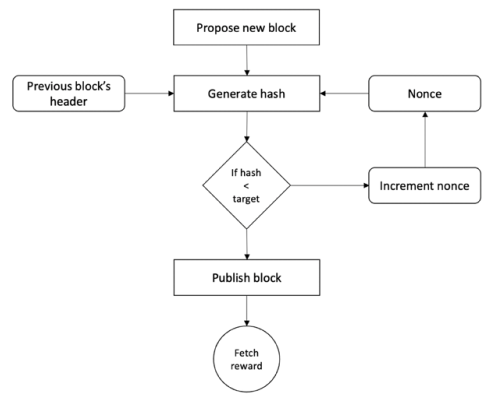
Aqui, **N** é um nonce, **P\_hash** é o hash do bloco anterior, **Tx** representa as transações no bloco, e **Alvo** é o valor de dificuldade-alvo da rede. Isso significa que o hash dos campos concatenados acima mencionados deve ser menor que o valor-alvo do hash.

A única maneira de encontrar este nonce é por método de força bruta. Uma vez que um determinado padrão com certo número de zeros seja alcançado por um minerador, o bloco é imediatamente transmitido e aceito por outros mineradores.

O algoritmo de mineração consiste nos seguintes passos:

1. O cabeçalho do bloco anterior é recuperado da rede Bitcoin.
2. Um conjunto de transações transmitidas na rede é montado em um bloco a ser proposto.
3. O hash duplo do cabeçalho do bloco anterior é calculado, combinado com um nonce e o novo bloco proposto, usando o algoritmo SHA-256.
4. Verifica-se se o hash resultante é menor que o nível atual de dificuldade (o alvo). Se for, então a PoW foi resolvida. Como resultado da PoW bem-sucedida, o bloco descoberto é transmitido para a rede e os mineradores recebem a recompensa.
5. Se o hash resultante não for menor que o nível atual de dificuldade (alvo), então o processo é repetido após incrementar o nonce.

À medida que a taxa de hash da rede Bitcoin aumentou, a quantidade total do nonce de 32 bits se esgotava muito rapidamente. Para resolver esse problema, a solução de extra nonce foi implementada, na qual a transação coinbase é usada para fornecer uma faixa maior de nonces a serem pesquisados pelos mineradores. Este processo é visualizado no fluxograma a seguir:

****

*Figure 6.7: Mining process*

A dificuldade da mineração aumenta com o tempo, e os bitcoins que antes podiam ser minerados por um computador portátil com uma CPU agora exigem centros de mineração dedicados para resolver o quebra-cabeça de hash. O nível de dificuldade atual pode ser consultado por meio da interface de linha de comando do Bitcoin com o seguinte comando:



Isso gera algo como o seguinte:

**36.835.682.546.787,98**

Esse número representa o nível de dificuldade de PoW da rede Bitcoin. Recorde-se das seções anteriores que os mineradores competem para encontrar a solução de um problema. Esse número, de fato, mostra o quão difícil é encontrar um hash inferior ao alvo de dificuldade da rede. Todos os blocos minerados com sucesso devem conter um hash que seja menor que esse número-alvo. Este número é atualizado a cada duas semanas ou a cada 2.016 blocos para garantir que, em média, o tempo de geração de bloco de 10 minutos seja mantido. A dificuldade da rede Bitcoin aumentou de forma aproximadamente exponencial.

**Sistemas de mineração**

Com o tempo, os mineradores de Bitcoin utilizaram vários métodos para minerar bitcoins. Como o princípio central por trás da mineração é baseado no algoritmo SHA-256 duplo, com o tempo, especialistas desenvolveram sistemas sofisticados para calcular o hash de maneira cada vez mais rápida. A seguir, uma análise dos diferentes tipos de métodos de mineração usados no Bitcoin e como eles evoluíram com o tempo.

**CPU**

A mineração por CPU foi o primeiro tipo de mineração disponível no cliente original do Bitcoin. Os usuários podiam até usar laptops ou computadores de mesa para minerar bitcoins. A mineração por CPU não é mais lucrativa e agora métodos de mineração mais avançados, como mineração baseada em ASIC, são usados. A mineração por CPU durou apenas cerca de um ano desde a introdução do Bitcoin, e logo outros métodos foram explorados e testados pelos mineradores.

**GPU**

Devido ao aumento da dificuldade da rede Bitcoin e à tendência geral de encontrar métodos mais rápidos para minerar, os mineradores começaram a usar GPUs ou placas gráficas disponíveis nos PCs para realizar a mineração. As GPUs oferecem cálculos mais rápidos e paralelizados, que geralmente são programados usando a linguagem OpenCL.

A dificuldade da rede Bitcoin aumentou de forma bastante significativa nos últimos anos.  
O gráfico de dificuldade mais recente está disponível aqui:  
<https://www.blockchain.com/charts/difficulty>

A taxa de hash aumenta ao longo do tempo e atualmente está em 266 EH. Para ver o valor mais recente, um gráfico está disponível aqui:  
<https://www.blockchain.com/charts/hash-rate>

Isso acabou se tornando uma opção mais rápida em comparação com as CPUs. Os usuários também utilizavam técnicas como overclocking para obter o máximo benefício da potência da GPU. Além disso, a possibilidade de usar várias placas gráficas em paralelo aumentou a popularidade do uso de placas gráficas para mineração de Bitcoin. No entanto, a mineração por GPU possui algumas limitações, como superaquecimento e a exigência de placas-mãe especializadas e hardware extra para abrigar várias placas gráficas. Por outro lado, as placas gráficas se tornaram bastante caras devido ao aumento da demanda, o que impactou jogadores e usuários de softwares gráficos.

**FPGAs**

Mesmo a mineração por GPU não durou muito, e logo os mineradores encontraram outra forma de realizar a mineração usando FPGAs (Field Programmable Gate Arrays – Matrizes de Portas Programáveis em Campo).

Um FPGA é basicamente um circuito integrado que pode ser programado para realizar operações específicas. Os FPGAs geralmente são programados em linguagens de descrição de hardware (HDLs), como Verilog e VHDL. O SHA-256 duplo rapidamente se tornou uma tarefa de programação atrativa para programadores de FPGA e vários projetos de código aberto também foram iniciados. Os FPGAs ofereciam um desempenho muito melhor em comparação com as GPUs; no entanto, questões como acessibilidade, dificuldade de programação e a exigência de conhecimento especializado para programar e configurar FPGAs resultaram em uma vida curta para a era FPGA da mineração de Bitcoin.

Hardware de mineração como o minerador X6500, Ztex e Icarus foi desenvolvido durante a época em que a mineração por FPGA era lucrativa. Vários fabricantes de FPGA, como Xilinx e Altera, produzem hardware e placas de desenvolvimento FPGA que podem ser usados para programar algoritmos de mineração.

**ASICs**

Os ASICs foram projetados para realizar operações SHA-256. Esses chips especiais foram vendidos por vários fabricantes e ofereciam uma taxa de hash muito alta. A chegada dos ASICs resultou na eliminação rápida dos sistemas baseados em FPGA para mineração.

Isso funcionou por algum tempo, mas devido ao rápido aumento do nível de dificuldade da mineração, ASICs de unidade única não são mais lucrativos. Com o fator de dificuldade atual (em maio de 2022), se um minerador conseguir produzir uma taxa de hash de 100 trilhões de hashes por segundo (TH/s), ele pode esperar ganhar 0,0004374 BTC (cerca de 12 dólares) por dia, e cerca de 4.456,61 dólares por ano, o que é muito baixo em comparação com o investimento necessário para obter o equipamento capaz de produzir 100 TH/s, aproximadamente 20.000 dólares. Incluindo os custos operacionais como eletricidade, isso se torna nada lucrativo.

Agora, centros de mineração profissionais usando milhares de unidades ASIC em paralelo estão oferecendo contratos de mineração para usuários realizarem mineração em seu nome. Não há limitação técnica — um único usuário pode operar milhares de ASICs em paralelo — mas isso exigirá data centers dedicados e hardware; portanto, o custo para um indivíduo pode se tornar proibitivo.

Deve-se observar que a mineração por GPU ainda é lucrativa para algumas outras criptomoedas até certo ponto, porque a dificuldade da rede é muito menor do que a do Bitcoin.

**Pools de mineração**

Um pool de mineração se forma quando um grupo de mineradores trabalha junto para minerar um bloco. O gerente do pool recebe a transação coinbase se o bloco for minerado com sucesso, e então é responsável por distribuir a recompensa para o grupo de mineradores que investiram recursos para minerar o bloco. Isso é mais lucrativo do que a mineração solo, onde apenas um único minerador tenta resolver a função de inversão parcial de hash (quebra-cabeça de hash), porque nos pools de mineração a recompensa é paga a cada membro do pool independentemente de eles (ou mais especificamente, seu nó individual) terem resolvido o quebra-cabeça ou não.

Existem vários modelos que um gerente de pool de mineração pode usar para pagar aos mineradores, como o modelo de pagamento por participação (*pay-per-share*) e o modelo proporcional. No modelo de pagamento por participação, o gerente do pool paga uma taxa fixa a todos os mineradores que participaram do exercício de mineração, enquanto no modelo proporcional, a participação é calculada com base na quantidade de recursos computacionais gastos para resolver o quebra-cabeça de hash.

A centralização da mineração pode ocorrer se um pool conseguir controlar mais de 51% da rede ao gerar mais de 51% da taxa de hash da rede Bitcoin. Como discutido anteriormente na introdução, um ataque de 51% pode resultar em ataques bem-sucedidos de gasto duplo e pode impactar o consenso e, de fato, até impor outra versão do histórico de transações na rede Bitcoin. Esse evento aconteceu uma vez na história do Bitcoin quando a GHash.io, um grande pool de mineração, conseguiu adquirir mais de 51% da capacidade da rede. Soluções teóricas, como a PoW de duas fases (*two-phase PoW*), foram propostas na academia para desincentivar grandes pools de mineração.

Esse esquema introduz um segundo quebra-cabeça criptográfico que resulta em pools de mineração revelando suas chaves privadas ou fornecendo uma porção considerável da taxa de hash do seu pool de mineração, reduzindo assim a taxa de hash total do pool.

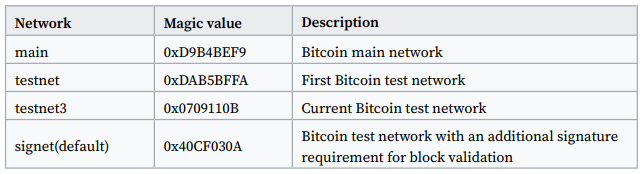
A corrida para construir mineradores eficientes está em andamento e só tende a crescer — embora não infinitamente. No entanto, há um limite para a aceleração de hardware e para as limitações físicas. Em breve, não restará espaço para nenhuma otimização *on-chip*. Ou pode se tornar extremamente desafiador alcançar quaisquer otimizações adicionais, a menos que ocorra alguma mudança fundamental na forma como o silício do hardware é desenvolvido.

**Rede**

A rede Bitcoin é uma rede ponto-a-ponto (P2P) onde os nós realizam transações. Eles verificam e propagam transações e blocos. Como acabamos de ver, nós chamados mineradores também produzem blocos. Um nó completo da rede Bitcoin executa quatro funções. São elas: carteira, minerador, blockchain e roteamento de rede.

Muitos pools comerciais agora existem e fornecem contratos de serviço de mineração via nuvem e interfaces web fáceis de usar. Os mais usados são AntPool, ViaBTC, F2Pool e Poolin.  
Uma comparação da taxa de hash dos principais pools de mineração está disponível aqui:  
<https://blockchain.info/pools>

O seguinte artigo descreve uma proposta de PoW em duas fases, a fim de desincentivar grandes pools de mineração do Bitcoin:  
<http://hackingdistributed.com/2014/06/18/how-to-disincentivize-large-bitcoin-mining-pools>



**Tipos de mensagens**

Existem 27 tipos de mensagens de protocolo no total, mas esse número provavelmente aumentará com o tempo à medida que o protocolo evolui. As mensagens de protocolo mais usadas e uma explicação de cada uma estão listadas a seguir:

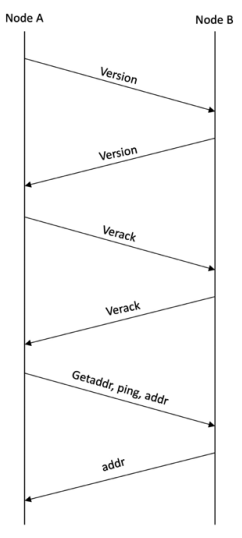
* **Version**: Esta é a primeira mensagem que um nó envia para a rede, anunciando sua versão e contagem de blocos. O nó remoto então responde com as mesmas informações, e a conexão é estabelecida.
* **Verack**: Esta é a resposta à mensagem *version*, aceitando o pedido de conexão.
* **Inv**: É usada pelos nós para anunciar seu conhecimento de blocos e transações.
* **Getdata**: Esta é uma resposta ao *inv*, solicitando um único bloco ou transação identificado pelo seu hash.
* **Getblocks**: Retorna um pacote *inv* contendo a lista de todos os blocos a partir do último hash conhecido ou até 500 blocos.
* **Getheaders**: É usada para solicitar cabeçalhos de bloco dentro de um intervalo especificado.
* **Tx**: Usada para enviar uma transação como resposta à mensagem de protocolo *getdata*.
* **Block**: Envia um bloco em resposta à mensagem *getdata*.
* **Headers**: Este pacote retorna até 2.000 cabeçalhos de bloco como resposta ao *getheaders*.
* **Getaddr**: Enviado como um pedido para obter informações sobre *peers* conhecidos.
* **Addr**: Fornece informações sobre nós na rede. Contém o número de endereços e uma lista de endereços no formato de endereço IP e número da porta.
* **Ping**: Esta mensagem é usada para confirmar se a conexão de rede TCP/IP está ativa.
* **Pong**: Esta mensagem é a resposta a uma mensagem *ping*, confirmando que a conexão de rede está ativa.

Quando um nó Bitcoin Core é iniciado, ele primeiro inicia a descoberta de todos os pares (*peers*). Isso é feito consultando *seeds* DNS que estão codificados no cliente Bitcoin Core e são mantidos por membros da comunidade Bitcoin. Essa busca retorna vários registros DNS do tipo A. O protocolo Bitcoin funciona por padrão na porta TCP 8333 para a rede principal e na porta TCP 18333 para a *testnet*.

Primeiramente, o cliente envia uma mensagem de protocolo, *version*, que contém vários campos, como a versão, serviços, *timestamp*, endereço de rede, nonce, e outros campos. O nó remoto responde com sua própria mensagem *version*, seguida pela troca de mensagens *verack* entre os dois nós, indicando que a conexão foi estabelecida.

Depois disso, mensagens *getaddr* e *addr* são trocadas para encontrar os pares que o cliente ainda não conhece. Enquanto isso, qualquer um dos nós pode enviar uma mensagem *ping* para verificar se a conexão ainda está ativa. *Getaddr* e *addr* são tipos de mensagem definidos no protocolo Bitcoin.

Esse processo é mostrado no diagrama a seguir do protocolo:

****

***Figure 6.8: Visualization of node discovery protocol***

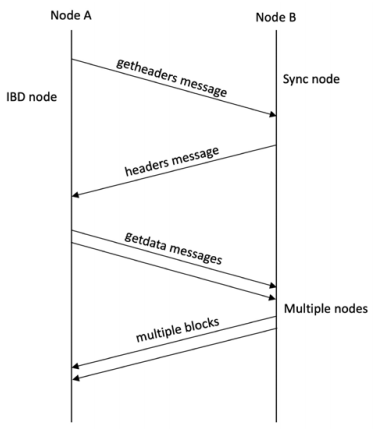
Este diagrama de sequência do protocolo de rede mostra a comunicação entre dois nós Bitcoin durante a conectividade inicial. O Nó A é mostrado no lado esquerdo e o Nó B no direito.

Os *seeds* DNS são declarados (codificados) no arquivo chainparams.cpp no código-fonte do Bitcoin, que pode ser visualizado no GitHub no seguinte link:  
<https://github.com/bitcoin/bitcoin/blob/0cda5573405d75d695aba417e8f22f1301ded001/src/chainparams.cpp#L116>

Primeiro, o Nó A inicia a conexão enviando uma mensagem *version* que contém o número da versão e o horário atual para o par remoto, Nó B. O Nó B então responde com sua própria mensagem *version* contendo o número da versão e o horário atual. O Nó A e o Nó B então trocam uma mensagem *verack*, indicando que a conexão foi estabelecida com sucesso. Uma vez que essa conexão seja bem-sucedida, os pares podem trocar mensagens *getaddr* e *addr* para descobrir outros pares na rede.

Agora, o download do bloco pode começar. Na versão 0.10.0, foi introduzido o método de download inicial de blocos chamado *headers-first*. Isso resultou em uma grande melhoria de desempenho, e a sincronização da blockchain, que costumava levar dias, passou a levar apenas algumas horas. A ideia central é que o novo nó primeiro solicita cabeçalhos de blocos aos pares e então os valida. Uma vez que isso é concluído, os blocos são solicitados em paralelo de todos os pares disponíveis. Isso ocorre porque o esqueleto da cadeia completa já foi baixado na forma da cadeia de cabeçalhos de bloco.

Neste método, quando o cliente é iniciado, ele verifica se a blockchain está totalmente sincronizada, caso a cadeia de cabeçalhos já esteja sincronizada; se não estiver, o que é o caso na primeira vez que o cliente inicia, ele solicita os cabeçalhos de outros pares usando a mensagem *getheaders*. Se a blockchain estiver totalmente sincronizada, ele escuta novos blocos via mensagens *inv*, e se já tiver uma cadeia de cabeçalhos totalmente sincronizada, então ele solicita blocos usando mensagens de protocolo *getdata*. O nó também verifica se a cadeia de cabeçalhos possui mais cabeçalhos do que blocos e, então, solicita os blocos emitindo a mensagem de protocolo *getdata*:

****

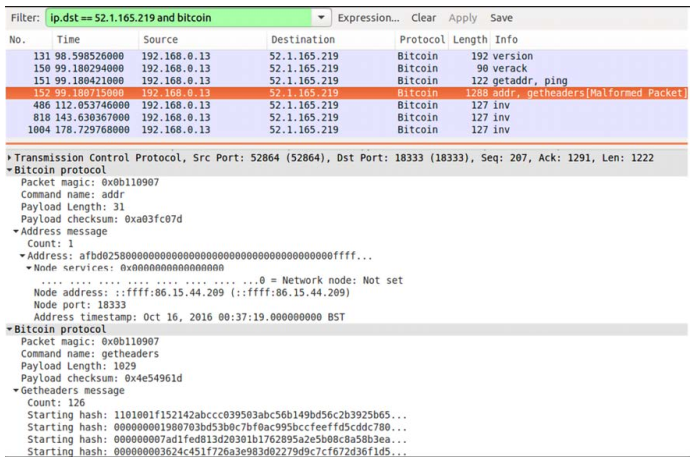
***Figure 6.9: Bitcoin Core client >= 0.10.0 header and block synchronization***

A imagem anterior mostra o processo de sincronização de blocos do Bitcoin entre dois nós da rede Bitcoin. O Nó A, mostrado no lado esquerdo, é chamado de nó de *Initial Block Download* (IBD), e o Nó B, mostrado à direita, é chamado de nó de sincronização.

Nó IBD significa que este é o nó que está solicitando os blocos, enquanto nó de sincronização significa o nó de onde os blocos estão sendo solicitados. O processo começa com o Nó A enviando primeiro a mensagem *getheaders*, que é respondida com uma resposta *getheaders* do nó de sincronização. A carga útil da mensagem *getheaders* é um ou mais hashes de cabeçalhos. Se for um nó novo, então há apenas o hash do cabeçalho do bloco gênese. O Nó de Sincronização B responde enviando até 2.000 cabeçalhos de bloco para o Nó IBD A. Após isso, o nó IBD, Nó A, começa a baixar mais cabeçalhos do Nó B e blocos de múltiplos nós em paralelo; ou seja, ele atua como o nó IBD e recebe múltiplos blocos de múltiplos nós, incluindo o Nó B. Se o nó de sincronização não tiver mais de 2.000 cabeçalhos quando o nó IBD fizer uma solicitação *getheaders*, o nó IBD envia uma mensagem *getheaders* para outros nós. Esse processo continua em paralelo até que a sincronização da blockchain esteja completa.

As chamadas de procedimento remoto (RPCs) *getblockchaininfo* e *getpeerinfo* foram atualizadas com novas funcionalidades para atender a essa mudança. Um RPC conhecido como *getchaintips* é usado para listar todos os ramos conhecidos da blockchain. Isso também inclui blocos somente com cabeçalhos. *getblockchaininfo* é usado para fornecer informações sobre o estado atual da blockchain. *getpeerinfo* é usado para listar tanto o número de blocos quanto os cabeçalhos que são comuns entre os pares.

O **Wireshark** também pode ser usado para visualizar a troca de mensagens entre pares e pode servir como uma ferramenta inestimável para aprender sobre o protocolo Bitcoin. Um exemplo disso é mostrado aqui. Este é um exemplo básico mostrando as mensagens *version*, *verack*, *getaddr*, *ping*, *addr* e *inv*. Nos detalhes, podem ser vistas informações valiosas como o tipo do pacote, nome do comando e os resultados das mensagens do protocolo:

****

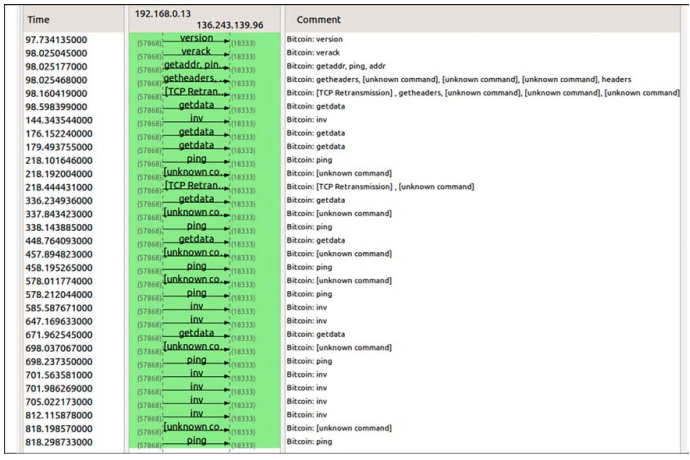
***Figure 6.10: A sample block message in Wireshark***

Um gráfico de protocolo mostrando o fluxo de dados entre os dois pares pode ser visto na captura de tela anterior. Isso pode ajudá-lo a entender quando um nó é iniciado e que tipo de mensagens são usadas. Essa visualização pode ser acessada selecionando a interface apropriada da placa de rede e, em seguida, aplicando o filtro para Bitcoin.

No exemplo a seguir, o **dissector Bitcoin** é usado para analisar o tráfego e identificar os comandos do protocolo Bitcoin. A troca de mensagens como *version*, *getaddr* e *getdata* pode ser vista, juntamente com o comentário apropriado descrevendo o nome da mensagem.

Esse pode ser um exercício muito útil para aprender sobre o protocolo Bitcoin e é recomendado que os experimentos sejam realizados na rede de teste do Bitcoin (*testnet*), onde várias mensagens e transações podem ser enviadas pela rede e depois analisadas pelo Wireshark.

A análise sendo realizada aqui pelo Wireshark mostra mensagens sendo trocadas entre dois nós. Se você olhar de perto, notará que as três primeiras mensagens mostram o protocolo de descoberta de nós que introduzimos anteriormente:

****

***Figure 6.11: Bitcoin node discovery protocol in Wireshark***

Os nós executam diferentes softwares cliente do Bitcoin. Vamos discutir isso a seguir.

**Wireshark** é uma ferramenta de análise de rede e está disponível em:  
<https://www.wireshark.org>

<https://en.bitcoin.it/wiki/Testnet>

**Software cliente**

Existem diferentes tipos de nós na rede. Os dois principais tipos de nós são os **nós completos** e os **nós SPV (verificação simples de pagamento)**.

Nódulos completos, como o nome implica, são implementações do cliente Bitcoin Core que executam as funções de carteira, mineração, armazenamento completo da blockchain e roteamento da rede. Esses nós baixam toda a blockchain; eles fornecem o método mais seguro de validar a blockchain como cliente e desempenham um papel vital na propagação de blocos. No entanto, não é necessário que todos os nós de uma rede Bitcoin executem todas essas funções. Alguns nós executam apenas funções de roteamento de rede, mas não realizam mineração nem armazenam chaves privadas (a função de carteira). Outro tipo de nó é o nó minerador solo, que pode executar mineração, armazenar blockchains completas e atuar como nós de roteamento da rede Bitcoin. Alguns nós realizam apenas funções de mineração e são chamados de **nós de mineração**.

É possível executar software SPV que executa uma carteira e função de roteamento de rede sem armazenar uma blockchain, enquanto ainda se sincroniza com a rede. Os nós SPV mantêm apenas uma cópia dos cabeçalhos de bloco da blockchain válida mais longa. Quando necessário, eles podem solicitar transações de nós completos. A verificação é realizada observando o ramo de Merkle, que liga as transações ao bloco original no qual a transação foi aceita. Isso não é muito prático e exige uma abordagem mais pragmática, que foi implementada com a **BIP37**, onde **filtros de Bloom** foram usados para filtrar apenas as transações relevantes. Vamos revisar filtros de Bloom na próxima seção.

Por fim, existem alguns nós não padronizados, mas amplamente utilizados. Esses são chamados de **servidores de protocolo de pool**. Esses nós fazem uso de protocolos alternativos, como o **protocolo Stratum**, um protocolo baseado em linhas que utiliza sockets TCP simples e JSON-RPC legível por humanos para operar e se comunicar entre nós. O Stratum é comumente usado para conectar-se a pools de mineração. Nós que apenas calculam hashes usam o protocolo Stratum para enviar suas soluções ao pool de mineração.

As informações de versionamento são codificadas no cliente Bitcoin no arquivo version.h, disponível em:  
<https://github.com/bitcoin/bitcoin/blob/0cda5573405d75d695aba417e8f22f1301ded001/src/version.h#L9>

A maioria dos protocolos na internet é baseada em linhas, o que significa que cada linha é delimitada por um caractere de retorno de carro e nova linha \r\n. Mais detalhes sobre esse protocolo estão disponíveis em:  
<https://en.bitcoin.it/wiki/Stratum_mining_protocol>

**Filtros de Bloom**

Um filtro de Bloom é uma estrutura de dados (um vetor de bits com índices) que é usada para testar a associação de um elemento de maneira probabilística. Ele fornece uma verificação probabilística com **falsos positivos**, mas **sem falsos negativos**.

Isso significa que esse filtro pode produzir uma saída em que um elemento que **não** é membro do conjunto testado seja **erroneamente considerado como presente** no conjunto. No entanto, ele **nunca** produzirá uma saída em que um elemento que de fato está presente no conjunto seja considerado como **ausente**.

Os elementos são adicionados ao filtro de Bloom após serem aplicadas várias funções hash e, em seguida, os bits correspondentes no vetor de bits são definidos como 1 nos índices resultantes. Para verificar a presença de um elemento no filtro de Bloom, as mesmas funções hash são aplicadas e, então, os bits no vetor são comparados para ver se os mesmos bits estão definidos como 1.

Observe que **nem toda função hash** (como SHA-1) é adequada para filtros de Bloom, pois é necessário que sejam **rápidas**, **independentes** e **uniformemente distribuídas**. Funções hash **não criptográficas** são usadas para filtros de Bloom, como **fnv**, **murmur** e **Jenkins**.

Esses filtros são usados principalmente por **clientes SPV** para solicitar transações e os blocos de Merkle nos quais estão interessados. Um **bloco de Merkle** é uma versão leve do bloco, que inclui um cabeçalho de bloco, alguns hashes, uma lista de sinalizadores de 1 bit e uma contagem de transações. Essas informações podem então ser usadas para construir uma **árvore de Merkle**. Isso é feito criando-se um filtro que corresponde apenas àquelas transações e blocos da cadeia completa que foram solicitados pelo cliente SPV. Uma vez que as mensagens *version* tenham sido trocadas e a conexão seja estabelecida entre os pares, os nós podem configurar filtros de acordo com suas necessidades.

Esses filtros probabilísticos oferecem diferentes graus de **privacidade** ou **precisão**, dependendo de quão estritamente ou vagamente eles tenham sido configurados. Um **filtro de Bloom estrito** filtrará apenas as transações que foram solicitadas pelo nó, mas à custa da possibilidade de revelar os endereços dos usuários para adversários, que podem correlacionar transações com seus endereços IP, comprometendo assim a privacidade.

Por outro lado, um filtro configurado de forma mais frouxa pode resultar na recuperação de mais transações não relacionadas, mas oferecerá mais **privacidade**. Além disso, para clientes SPV, os filtros de Bloom permitem o uso de **baixa largura de banda**, em contraste com o download de todas as transações para verificação.

A **BIP37** propôs a implementação dos filtros de Bloom no Bitcoin e introduziu três novas mensagens ao protocolo Bitcoin:

* **filterload**: Usada para configurar o filtro de Bloom na conexão.
* **filteradd**: Adiciona um novo elemento de dados ao filtro atual.
* **filterclear**: Exclui o filtro atualmente carregado.

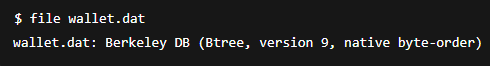
Até agora, discutimos que, em uma rede Bitcoin, existem clientes completos (nós), que executam a função de armazenar uma blockchain completa. Se você não puder executar um nó completo, então clientes SPV podem ser usados para verificar se transações específicas estão presentes em um bloco, baixando apenas os **cabeçalhos de bloco** em vez de toda a blockchain.

Às vezes, mesmo executar um nó SPV não é viável (especialmente em dispositivos com poucos recursos, como **celulares**) e a única exigência é **enviar e receber Bitcoin de alguma forma**. Para esse propósito, são usados **wallets** (software de carteira) que **não requerem nem mesmo o download dos cabeçalhos de bloco**.

Mais detalhes podem ser encontrados na especificação da **BIP37**, disponível em:  
<https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki>

**Carteiras**

O software de carteira é usado para **gerar e armazenar chaves criptográficas**. Ele executa várias funções úteis, como **receber e enviar Bitcoin**, **fazer backup de chaves** e **manter o controle do saldo disponível**. O software cliente do Bitcoin geralmente oferece ambas as funcionalidades: **cliente Bitcoin e carteira**. No disco, as carteiras do cliente Bitcoin Core são armazenadas como um arquivo do Berkeley DB:



Chaves privadas são geradas escolhendo aleatoriamente um número de 256 bits fornecido pelo software da carteira. As regras de geração são predefinidas e foram discutidas no **Capítulo 4, Criptografia Assimétrica**. As chaves privadas são usadas pelas carteiras para assinar transações de saída. As carteiras **não armazenam nenhuma moeda**. De fato, na rede Bitcoin, **moedas não existem**; em vez disso, apenas **informações de transações** são armazenadas na blockchain (mais precisamente, **UTXOs**, ou saídas não gastas), que são então usadas para calcular o número de bitcoins.

Fundamentalmente, uma carteira de criptomoeda depende de um **keystore** (repositório de chaves) para armazenar as chaves privadas. Um keystore pode ser tão simples quanto um arquivo ou tão complexo quanto um **módulo de segurança de hardware (HSM)** ou até mesmo um dispositivo portátil como uma **carteira física (hardware wallet)**. A(s) chave(s) privada(s) devem ser protegidas contra roubo. Normalmente, a criptografia do keystore e hardware seguro especializado, como um HSM, são usados para proteger as chaves. Essas chaves privadas são usadas em **assinaturas digitais** para assinar as transações como prova da propriedade das moedas.

No Bitcoin, e em geral nas criptomoedas, existem diferentes tipos de carteiras que podem ser usadas para armazenar chaves privadas. Como software, elas também oferecem algumas funções aos usuários para gerenciar e realizar transações na rede Bitcoin. Vejamos os tipos mais comuns de carteiras:

* **Carteiras não determinísticas**: Essas carteiras contêm chaves privadas geradas aleatoriamente e também são chamadas de carteiras do tipo **Just a Bunch of Keys (JABOK)**. O cliente Bitcoin Core gera algumas chaves quando é iniciado pela primeira vez e também gera chaves conforme necessário. Gerenciar muitas chaves é um processo muito difícil e propenso a erros que pode levar ao roubo e, consequentemente, à perda de moedas. Além disso, há necessidade de criar backups regulares das chaves e protegê-las adequadamente, por exemplo, criptografando-as para evitar roubo ou perda.
* **Carteiras determinísticas**: Nesse tipo de carteira, as chaves são derivadas de um valor semente (*seed*) por meio de funções hash. Esse número semente é gerado aleatoriamente e é comumente representado por **palavras mnemônicas legíveis por humanos**. As palavras mnemônicas são definidas na **BIP39**, uma proposta de melhoria do Bitcoin para geração de chaves determinísticas. Essa BIP está disponível em:  
  <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>  
  Essas frases podem ser usadas para recuperar todas as chaves e tornam o gerenciamento de chaves privadas relativamente mais fácil.
* **Carteiras determinísticas hierárquicas (HD)**: Definidas na **BIP32** e na **BIP44**, as carteiras HD armazenam chaves em uma **estrutura de árvore** derivada de uma semente.  
  A semente gera a chave pai (**chave mestra**), que é usada para gerar chaves filhas e, subsequentemente, chaves netas. A geração de chaves em carteiras HD não gera diretamente as chaves privadas; em vez disso, produz algumas informações (informações para geração de chave privada) que podem ser usadas para gerar uma sequência de chaves privadas.

A hierarquia completa de chaves privadas em uma carteira HD é facilmente recuperável se a **chave privada mestre** for conhecida. É por causa dessa propriedade que as carteiras HD são muito fáceis de manter e altamente portáveis. Existem muitas carteiras HD gratuitas e comerciais disponíveis, por exemplo:

* <https://trezor.io>
* <https://jaxx.io>
* <https://electrum.org/>
* **Carteiras cerebrais (*brain wallets*)**: A chave privada mestre também pode ser derivada de uma **única frase legível por humanos e fácil de lembrar** (daí o termo *carteiras cerebrais*). A ideia aqui é que essa **senha fácil de memorizar** seja usada para derivar a chave privada e, se usada em carteiras HD, isso pode resultar em uma **carteira HD completa derivada de uma única senha memorizada**. Isso é conhecido como carteira cerebral. Embora essa seja uma solução fácil de usar para os usuários, esse método é propenso a **ataques de adivinhação de senha**, **ataques de dicionário** e **ataques de força bruta**. Algumas técnicas, como **key stretching**, podem ser usadas para retardar o progresso feito pelo atacante.
* **Carteiras de papel (*paper wallets*)**: Como o nome indica, trata-se de uma carteira em papel com o material da chave necessário impresso nela. Exige **segurança física** para ser armazenada. As carteiras de papel podem ser geradas online a partir de vários provedores de serviço, como:  
  <https://bitcoinpaperwallet.com/>  
  <https://www.bitaddress.org/>
* **Carteiras físicas (*hardware wallets*)**: Outro método é usar um dispositivo resistente à violação para armazenar chaves. Este dispositivo resistente à violação pode ser construído sob medida. Com o advento dos telefones com **NFC (Comunicação por Campo de Proximidade)**, isso também pode ser um **elemento seguro (SE)** em telefones NFC. Trezor (<https://trezor.io>) e Ledger (<https://www.ledger.com>) são carteiras físicas de criptomoedas comumente usadas.
* **Carteiras online**: As carteiras online, como o nome sugere, são armazenadas inteiramente online e são oferecidas como um serviço, geralmente via nuvem. Elas fornecem uma interface web para que os usuários gerenciem suas carteiras e realizem várias funções, como fazer e receber pagamentos. Elas são fáceis de usar, mas exigem que o usuário **confie no provedor de carteira online**. Um exemplo de carteira online é **GreenAddress**, que está disponível em:  
  <https://greenaddress.it/en/>
* **Carteiras móveis**: Carteiras móveis, como o nome sugere, são instaladas em dispositivos móveis. Elas podem fornecer diversos métodos para realizar pagamentos, mais notavelmente a capacidade de usar câmeras de smartphones para **escanear QR codes rapidamente** e efetuar pagamentos. Há muitas empresas oferecendo essas carteiras. No entanto, não é sensato sugerir qual tipo de carteira deve ser usado, pois isso varia de acordo com **preferências pessoais** e os recursos disponíveis em cada carteira. Portanto, **não faremos nenhuma recomendação específica aqui**.

**Elementos seguros (Secure Elements)**: chip resistente à violação que pode hospedar com segurança aplicativos, dados confidenciais e chaves criptográficas. Smartphones, carteiras físicas de criptomoedas e tablets usam elementos seguros para armazenar com segurança dados confidenciais como senhas, códigos PIN, impressões digitais e, de fato, qualquer coisa que requeira confidencialidade e integridade.

**Chips NFC**: comunicação sem fio de curto alcance entre dois dispositivos em proximidade próxima, comumente usada para pagamentos por aproximação e outras aplicações seguras via smartphones.

**Escolha da carteira**

A escolha da carteira Bitcoin depende de vários fatores, como **segurança**, **facilidade de uso** e **recursos disponíveis**. Dentre todos esses atributos, a **segurança**, é claro, vem em primeiro lugar e, ao decidir qual carteira usar, **a segurança deve ser de importância máxima**.

Carteiras físicas tendem a ser mais seguras em comparação com carteiras web, por causa de seu design resistente à violação. As carteiras web, por sua própria natureza, são hospedadas em websites, os quais podem não ser tão seguros quanto um dispositivo de hardware resistente à violação. De modo geral, carteiras móveis para smartphones são bastante populares devido a uma combinação equilibrada de recursos, experiência do usuário e segurança.

**Resumo**

Este capítulo começou com uma introdução ao Bitcoin e explicou como uma transação funciona do ponto de vista do usuário. Em seguida, foi apresentada uma introdução às transações do ponto de vista técnico. Após isso, foram discutidas as chaves públicas e privadas utilizadas no Bitcoin. Na próxima seção, foram introduzidos os endereços e seus diferentes tipos, seguidos por uma discussão sobre as transações, seus tipos e usos.

Em seguida, vimos a blockchain, com uma explicação detalhada de como ela funciona e os vários componentes incluídos na blockchain do Bitcoin. Depois, foram introduzidos os processos de mineração e os conceitos relevantes, como sistemas de hardware, suas limitações e recompensas em bitcoin, juntamente com uma introdução à rede Bitcoin, seguida de uma discussão sobre descoberta de nós Bitcoin e protocolos de sincronização de blocos. Por fim, examinamos diferentes tipos de carteiras Bitcoin e discutimos os vários atributos e recursos de cada tipo.

No próximo capítulo, examinaremos alguns conceitos práticos relacionados a pagamentos em Bitcoin, clientes e programação.

Um cliente SPV, que introduzimos anteriormente, também é um tipo de carteira Bitcoin. Outros tipos de carteiras Bitcoin, especialmente carteiras móveis, são na maioria **carteiras baseadas em API** que dependem de um mecanismo onde **chaves privadas e públicas são armazenadas localmente no dispositivo onde o software da carteira está instalado**. Aqui, **servidores de backend confiáveis** são usados para fornecer dados da blockchain via APIs.