**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:

{

"result":

"01000000017d3876b14a7ac16d8d550abc78345b6571134ff173918a096ef90ff0430e12408b0000006b483045022100de6fd8120d9f142a82d5da9389e271caa3a757b01757c8e4fa7afbf92e74257c02202a78d4fbd52ae9f3a0083760d76f84643cf8ab80f5ef971e3f98ccba2c71758d012102c16942555f5e633645895c9affcb994ea7910097b7734a6c2d25468622f25e12ffffffff022c820000000000001976a914c568ffeb46c6a9362e44a5a49deaa6eab05a619a88acc06c0100000000001976a9149386c8c880488e80a6ce8f186f788f3585f74aee88ac00000000",

"error": null,

"id": null

}

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”

Essa mensagem foi uma manchete do jornal britânico *The Times*, publicada no mesmo dia, referindo-se à crise financeira global. Ela serve como um carimbo de data (timestamp) e como uma declaração política sobre a motivação por trás do Bitcoin: criar um sistema financeiro alternativo descentralizado.

O hash do bloco gênese é:

000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

A estrutura do bloco gênese é idêntica a todos os blocos subsequentes. No entanto, ele é codificado no software Bitcoin e não possui um bloco anterior. Além disso, a transação coinbase no bloco gênese não pode ser gasta.

**Árvore de Merkle**

A árvore de Merkle é uma estrutura de dados usada para armazenar hashes de transações em um bloco. Ela é uma árvore binária onde cada folha é um hash de transação, e cada nó interno é o hash da concatenação de seus filhos. A raiz da árvore de Merkle — conhecida como *raiz de Merkle* — é armazenada no cabeçalho do bloco. Isso permite verificações eficientes de inclusão de transações, chamadas *provas de Merkle*.

A árvore de Merkle reduz significativamente a quantidade de dados que precisa ser verificada para confirmar que uma transação está incluída em um bloco. Essa verificação é crucial para nós leves (SPV — *Simplified Payment Verification*), que não armazenam toda a blockchain, mas apenas os cabeçalhos dos blocos.

**Figura 6.6: Estrutura da árvore de Merkle**

No diagrama anterior, T1 a T4 são os hashes das transações. H12 é o hash da concatenação de T1 e T2; H34 é o hash da concatenação de T3 e T4; e a raiz de Merkle é o hash da concatenação de H12 e H34.

**Mineração**

A mineração é o processo de adicionar novos blocos à blockchain. É feita por mineradores que competem para encontrar um nonce tal que o hash do cabeçalho do bloco seja inferior a um valor-alvo (definido pela dificuldade de rede). Esse processo é chamado de **prova de trabalho** (*Proof of Work*, PoW).

A mineração serve a dois propósitos:

1. **Segurança da rede:** Ao exigir que os mineradores gastem poder computacional para adicionar blocos, a rede é protegida contra ataques como gastos duplos.
2. **Emissão de novas moedas:** Novos bitcoins são criados como recompensa para o minerador que encontra o bloco válido. Essa recompensa é composta pela *recompensa de bloco* fixa (que diminui pela metade a cada 210.000 blocos — evento chamado *halving*) e pelas taxas de transação incluídas no bloco.

Atualmente, a recompensa por bloco é de 6,25 BTC (valor vigente na data de publicação desta edição), e é previsto que ela continue diminuindo até que o número máximo de 21 milhões de bitcoins seja emitido.

**Pools de mineração**

Com o aumento do número de mineradores e do poder computacional necessário para minerar um bloco, tornou-se mais difícil minerar com sucesso de forma independente. Para resolver esse problema, grupos de mineradores se organizam em *pools de mineração*. Um pool de mineração é um grupo de mineradores que trabalham juntos para encontrar um bloco. Quando o bloco é encontrado, a recompensa é dividida proporcionalmente entre todos os participantes, de acordo com a quantidade de trabalho (hashes) que contribuíram.

Esse conceito resolve o problema da variância de recompensa: enquanto um minerador solo pode esperar meses ou anos até encontrar um bloco, em um pool ele recebe pagamentos menores e mais frequentes.

Vários métodos são usados para dividir recompensas em pools de mineração. Alguns deles incluem:

* **PPS (Pay-Per-Share):** Os mineradores recebem um pagamento fixo por cada share válido enviado, independentemente de o bloco ser encontrado.
* **PPLNS (Pay-Per-Last-N-Shares):** Os mineradores são pagos com base nas últimas N ações, o que alinha melhor os incentivos com o sucesso do pool.
* **PROP (Proporcional):** As recompensas são divididas proporcionalmente entre todos os mineradores que contribuíram durante uma rodada de mineração.

Exemplos populares de pools de mineração incluem SlushPool, F2Pool, Antpool e BTC.com.

**Reorganização de blocos (Reorgs)**

Uma reorganização (reorg) de blockchain ocorre quando dois mineradores encontram blocos válidos ao mesmo tempo, resultando em uma bifurcação temporária. A rede então segue a cadeia mais longa, ou seja, aquela com o maior trabalho acumulado (medido pela dificuldade).

Quando ocorre uma reorganização, os blocos da cadeia mais curta são descartados e as transações dentro deles são devolvidas ao pool de transações. Isso pode levar a situações onde uma transação previamente confirmada é revertida. Por isso, recomenda-se aguardar múltiplas confirmações (seis ou mais) antes de considerar uma transação como final.

**Tempo de bloqueio (Locktime) e nSequence**

A linguagem de script do Bitcoin permite que transações sejam postergadas até um determinado tempo ou bloco. Isso é feito usando os campos **nLockTime** e **nSequence**.

* **nLockTime:** Especifica o tempo (timestamp ou número do bloco) antes do qual a transação não pode ser incluída em um bloco.
* **nSequence:** Usado em conjunto com **nLockTime** para indicar se uma transação é substituível antes da sua confirmação (relevante para *Replace-by-Fee* ou RBF).

Essas ferramentas permitem construir contratos temporais, como pagamentos diferidos ou condicionais.

**Resumo**

Neste capítulo, exploramos os principais componentes do sistema Bitcoin, incluindo:

* Endereços e sua geração usando funções de hash criptográficas;
* A estrutura e o ciclo de vida das transações;
* Scripts Bitcoin e como são usados para validar transações;
* A estrutura de blocos e a função das árvores de Merkle;
* Mineração, pools de mineração, reorganizações e parâmetros de tempo de bloqueio.

Esses conceitos são fundamentais para entender o funcionamento interno do Bitcoin e servem de base para tópicos mais avançados em criptomoedas e blockchain.