Um estudo sobre Blockchain

Igor F. Miranda
Engenharia de Computação
Universidade de Brasília
Email: igormiranda5@gmail.com

***Resumo*—The abstract goes here.**

1. Introduction

This demo file is intended to serve as a “starter file” for IEEE Computer Society conference papers produced under LaTeX using IEEEtran.cls version 1.8b and later. I wish you the best of success.

mds
August 26, 2015

2. O Bitcoin

A proposta do Bitcoin(฿) surgiu em meados de 2008 em um artigo intitulado "Bitcoin: A Peer-to-Peer Electronic Cash System"escrito por um autor sob o pseudônimo de Satoshi Nakamoto. Ele utilizou varias propostas apresentadas no b-money, Hashcash e Bitgold para criar uma moeda digital *peer-to-peer*(P2P) que não depende de uma autoridade central, ou seja, para efetuar uma transação não é necessário a existência de uma autoridade central confiável para validá-la [2].

A grande inovação proposta por Nakamoto em seu artigo foi utilizar o conceito de prova de trabalho (*proof-of-work*) para criar um consenso distribuído confiável e resolver o problema de *Double Spending*. Tal solução pode ser utilizada para alcançar um consenso em redes decentralizadas para provar a honestidade de eleições, registros, contratos e muito mais.

Para utilizar a moeda é necessário participar da rede Bitcoin e para fazer isso deve-se possuir um cliente Bitcoin. As *Bitcoin Wallets* (Carteiras Bitcoin) são os clientes mais conhecidos para participar desse sistema onde pode-se enviar, receber e "armazenar" suas moedas. Existem vários tipos e implementações de Carteiras Bitcoins, porém, a mais conhecida é o Bitcoin Core que foi derivado da implementação original de Satoshi [1].

Atualmente exitem vários tipos de carteiras Bitcoin com diferentes níveis de segurança e propósitos. Elas são classificadas de acordo com o local de armazenamento das moedas e são divididas em cinco categorias, *Desktop wallet*, *Mobile wallet*, *Web wallet*, *Hardware wallet* e *Paper wallet* [1].

As carteiras também são classificadas de acordo com a sua autonomia e o tipo de interação com a rede Bitcoin:

- *Full node*: armazena todo o histórico de transação da rede bitcoin (Blockchain), gerencia a carteira do usuário localmente e pode iniciar uma transação diretamente com a rede Bitcoin. Ele consegue validar a blockchain oferecendo completa autonomia e uma validação de transações independente, porém ele consome uma grande quantidade de espaço em disco.
- *Lightweight Client*: se conecta a um full node para ter acesso as transações da rede Bitcoin. Gerencia a carteira do usuário localmente, cria, valida e transmite as transações.
- *Third-party API client*: o usuário ira interagir com a rede Bitcoin através de uma API fornecida por um servidor. A carteira poderá ser armazenada com o próprio usuário ou no servidor, porém as transações são sempre gerenciadas pelo servidor.

Cada carteira Bitcoin possui uma par de chave pública/privada. A chave privada é tudo que o usuário necessita para controlar os fundos associados ao endereço da carteira Bitcoin e para comprovar a posse dos fundos usados em uma transação. A partir da chave publica utilizando uma função hash é gerado um endereço para a carteira Bitcoin. Esse par de chaves é essencial para fazer uma transação na rede Bitcoin.

2.1. Rede Bitcoin

Como a finalidade do Bitcoin é ser um sistema monetário descentralizado, justo, sem fronteiras e que não depende de uma autoridade central dizendo aos seus usuários como deve-se usa-lo, onde usa-lo, o sistema que o rege deve seguir os mesmo princípios.

Para conseguir alcançar tal objetivo a rede Bitcoin utiliza uma arquitetura chamada de peer-to-peer (P2P). Nesse tipo de arquitetura todos os nós da rede realizam as funções tanto de cliente quanto de servidor, sendo iguais hierarquicamente. Isso faz com que uma rede p2p seja descentralizada, pois não existe nenhum servidor central, serviço centralizado ou hierarquia na rede, e escalável.

O termo rede Bitcoin refere-se ao nós que participam da rede P2P Bitcoin. Porém, além do protocolo P2P a rede Bitcoin utiliza outros protocolos como o Stratum, que é utilizado por pools de mineração e carteiras leves ou

móveis, essas redes que utilizam um protocolo diferente do P2P Bitcoin são chamadas de redes Bitcoin estendidas. Essa redes estendidas comunicam-se com a rede Bitcoin através de servidores de roteamento de gateway que podem comunicar-se tanto com o protocolo P2P Bitcoin quanto o protocolo da rede estendida.

Tipos de nós

Existem quatro funcionalidades que um nó pode ter na rede: carteira, minerador, blockchain completo e roteamento. Além dessas quatro funcionalidades existe um funcionalidade para os nós da rede estendida que é de e roteamento pool, esse funcionalidade é responsável pela comunicam do nó com os servidores de roteamento de gateway (Servidores Pool) que se comunicam com a rede Bitcoin. Embora os nós em uma rede P2P sejam iguais hierarquicamente isso não significa que eles necessitam possuir as mesmas funcionalidades. Todos os nós da rede Bitcoin devem possuem a funcionalidade de roteamento e podem incluir outras funcionalidades. Os nós da rede Bitcoin são classificados da seguinte forma:

- Reference client: inclui as funcionalidades de minerador, carteira, blockchain completo e roteamento;
- Full node: inclui as funcionalidades de blockchain completo e roteamento;
- Minerador solo: inclui as funcionalidades de minerador, blockchain completo e roteamento;
- Carteira leve (SPV): inclui as funcionalidades de carteira e roteamento;
- Nós mineradores: inclui as funcionalidades de minerador e roteamento pool;
- Carteira leve(SPV) Stratum:: inclui as funcionalidades de carteira e roteamento pool Stratum.

Os nós que possuem a funcionalidade de blockchain completo mantém uma cópia completa e atualizada da blockchian, fazendo com que eles possam verificar de maneira autônoma e autoritária qualquer transação sem qualquer referência externa. Alguns nós mantêm apenas uma parte da blockchain verificando as transações utilizando um método chamado verificação de pagamento simplificado (SPV). Esses nós são conhecidos como SPV ou peso-leve já que não armazenam toda a blockchain.

Os nós com funcionalidade de minerar competem pela criação de novos blocos na blockchain resolvendo algoritmos de prova de trabalho. Alguns nós de mineração são nós completos que matem uma cópia da blockchian, enquanto outros são nós peso leve que participam de um pool de mineração.

As carteiras Bitcoin também podem fazer parte de um Full node para validar todas as transações da rede. Porém, cada vez mais is usuários utilizam carteiras SVP, isso vem ocorrendo porque cada vez mais os usuários vem usando carteiras em dispositivos com poucos recursos com um smartphone.

Conectando-se a rede

Para participar da rede um nó deve descobrir e conectar-se a outros nós da rede Bitcoin. Como o novo nó ainda não conhece nenhum nó da rede ele deve solicitar aos servidores DNS (DNS seeds) da rede o IP de pelo menos um nó da rede, a partir do qual pode estabelecer novas conexões.

Assim que uma ou mais conexões são estabelecidas, o novo nó deve enviar mensagens addr contendo seu endereço IP para seus vizinhos que irão retransmitir a mensagem addr recebi para seus vizinhos. Esse processo da a garantia que os novos nós serão bem conhecidos e melhor conectados. Os novos nós conectados também podem enviar mensagens getaddr para seus vizinhos, solicitando-os que retornem uma lista dos nós conectados a eles. Dessa maneira, um nó pode encontrar novos pontos para conectar-se e divulgar sua existência na rede para que outros nós possam encontra-lo.

Como os nós podem ficar offline line a qualquer momento os nós ativos precisam continuar procurando novos nós a medida que eles perdem conexões antigas.

Verificação simplificada de pagamento

2.2. Transações na rede Bitcoin

O ciclo de vida de uma transação começa em sua criação, conhecido também como originação. Após ser criada ela deve ser assinada por uma ou mais assinaturas indicando a autorização da transferência dos fundos indicados na transação. Ela então deve ser transmitida para a rede através de um processo de *flooding* (inundação). Ao chegar a um nó minerador as transações ficam em um local chamado de pool de transações, até ser incluída em um bloco da blockchain.

O remetente da transação não precisa confiar nos nós que ele utiliza para transmitir a transação, contanto que ele utilize mais de uma para garantir a propagação, e da mesma maneira os nós destinatários não precisam conhecer a identidade de que envia a transação. Como uma transação bitcoin não contem nenhum dado confidencial ela pode ser transmitida publicamente em qualquer transporte de rede conveniente.

Assim que uma transação chega em um nó ela será validada por ele. Caso ela seja válida, o nó irá propaga-la para outros nós conectados a ele, e uma mensagem de sucesso será enviada para que originou a transação. Se a transação for invalida, o nó irá rejeita-lá e irá retornar uma mensagem de rejeição para que originou a mensagem. Essa regra de não propagar as transações invalidas serve para prevenir spam, ataques DDos e outros ataques maliciosos contra o sistema bitcoin.

Estrutura de uma transação

Uma transação bitcoin tem a finalidade de informar a rede que uma carteira que controla uma fonte de fundos bitcoin, chamado de *input*, autorizou a transferência de alguns de seus bitcoins para um destino, chamado de *output*.

Esses *inputs* e *outputs* são quantias de bitcoin, pedaços indivisíveis, que são ficam bloqueados até que seu dono forneça a assinatura digital que libera essa quantia de bitcoin. Uma transação contém seis campos e sua estrutura e apresentada na Figura 1.

Campo		Tamanho
nVersion		int 4bytes
#vin		VarInt 1-9 bytes
vin[ ]	hash	uint256 32 bytes
	n	uint 4 bytes
	scriptSigLen	VarInt 1-9 bytes
	scriptSig	CScript Variable
	nSequence	uint 4 bytes
#vout		VarInt 1-3 bytes
vout[ ]	nValue	int64_t 8 bytes
	scriptPubkeyLen	VarInt 1-9 bytes
	scriptPubkey	CScript Variable
nLockTime		unsigned int 4 bytes

Figura 1. Estrutura de uma transação [3]

Cada transação pode conter mais de um *input* e *output*, e eles são armazenados nos campos *vin*[] e *vout*[] respectivamente. O tamanho desses vetores, que corresponde a quantidade de *inputs* e *outputs* de uma transação, é armazenado nos campos #vin, para o *vin*[], e #vout, para o *vout*[].

Cada *output* de transação possui 3 campos: *nValue*, *scriptPubkeyLen* e *scriptPubkey*. O campo *nValue* corresponde uma quantia de Bitcoin em uma unidade chamada Satoshi, essa unidade corresponde a maior divisão que pode-se fazer em uma unidade de Bitcoin que é de 8 casas decimais, ou seja, 1 Satoshi é equivalente 0.00000001฿. O *scriptPubkeyLen* armazena o tamanho do *scriptPubkey* que é um script de travamento do UTXO, a linguagem script do bitcoin será apresentada na seção 2.3.

Os *outputs* de transação geram pedaços indivisíveis de bitcoin gastáveis chamadas de *output* de transação não gastos (*Unspent transaction output*) ou UTXO. Ao criar um UTXO ele se torna indivisível, ou seja, seu valor não pode ser divido assim como não se pode dividir uma moeda. Esses UTXO são reconhecidos na rede como quantias de bitcoin que podem ser gastas em uma transação futura.

Os UTXOs são monitorados e armazenados na memória RAM por todos os nós da rede Bitcoin como um conjunto de dados chamado de *pool* UTXO ou UTXO *set*. Sua administrarão é feita automaticamente pela carteira Bitcoin do usuário, assim como sua quantia de Bitcoins. Essa quantia que toda carteira diz possuir não passa de todos os UTXOs associados a chave primaria daquela carteira que estão espalhados na Blockchain. Como efeito disso, não existe um armazenamento de saldo em uma carteira Bitcoin, o que existe na verdade são UTXOs dispersos na blockchain vinculados a uma chave. O conceito de saldo de uma carteira Bitcoin não passa de uma abstração da operação de busca na blockchain e a soma de todas as UTXOs pertencentes a chave primaria de uma carteira.

Os *inputs* de transação são referências aos UTXOs que serão utilizados na transação. Cada *input* possui quatro campos: *hash*, *n*, *coinbaseLen* e *nSequence*. O campo *hash* corresponde ao duplo hash da transacão no blockchain que contém o UTXO que está sendo utilizado no *input*, servindo como referência. O segundo campo armazena a posição do UTXO na transacão anterior, como um index de um vetor. O *scriptSigLen* armazena o tamanho do *scriptSig* que é o script que libera a utilização do UTXO. O *nSequence* armazena o número de sequência do input de transação. Ele foi planejado para a funcionalidade de atualização da transação antes de ser incluída em um bloco através de múltiplas assinaturas, porém essa funcionalidade foi removida para reduzir a complexidade do protocolo.

O *nLockTime* serve como uma trava de tempo para a transação. Assim que o tempo estipulado pelo *nLockTime* for ultrapassado a transação é travada é pode ser incluída em um bloco. Se o valor do *nLockTime* dor 0 significa que a transação está travada, um valor menor que $5 * 10^8$ corresponde ao número de blocos que deve-se ter para travar a transação e um valor igual ou acima de $5*10^8$ corresponde ao tempo no formato UNIX que deve-se obter.

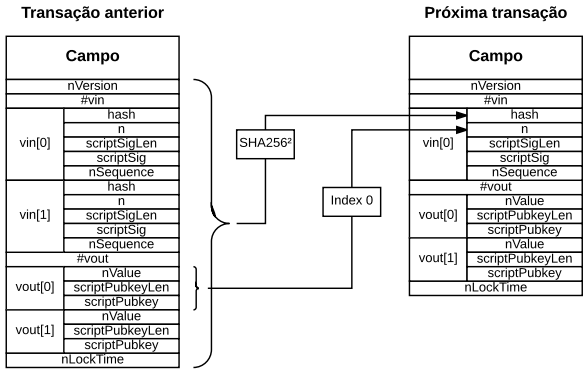

Figura 2. Relação entre transação

Pode parecer estranha a necessidade de dois *outputs*, mas como um UTXO é indivisível é necessário gerar um troco na transação, e esse segundo output é uma transação de volta para a carteira do usuário com o valor do troco que de ser gerado. Por exemplo,um usuário possui 3 UTXOs de Bitcoin e deseja fazer uma transação de 1 Bitcoin, essa transação precisará consumir todos os 3 UTXOs e irá produzir 2 outpus: uma para o destinatário da transação no valor de 1฿ e outro no valor de 2฿ menos a taxa do minerador como troco de volta para o usuário.

A taxa de transação é paga ao minerador que processou a transação em um bloco como recompensa pelo seu proces-

samento e gasto de energia é são calculadas de acordo com o tamanho da transação em kilobytes, e também servem como desestimuladoras de transações *spam*. As taxas de transação afetam a prioridade do processamento de uma transação, significando uma uma transação com uma taxa alta podem ser incluídas na blockchain mais rapidamente do que uma transação de com uma taxa muito baixa. Um taxa de transação não é obrigatória em uma transação, e as transações sem taxa podem ser processadas.

O valar dessa taxa é calculado através da subtração da soma de todos os *inputs* com a soma de todos os *outputs* ($\sum_{i=0}^{N} inputs - \sum_{i=0}^{K} outputs$), o valor dessa operação será o valor da taxa. As taxas dadas ao minerador são transferidas para ele através de uma transação criado por ele é inserida no bloco que foi minerado.

Transação coinbase

A única exceção para essa regra de cadeia de *inputs* e *output* é para um tipo especial de transação chamada *coinbase*. Essa transação é inserida no bloco minerado pelo minerador "vencedor"e tem a finalidade de inserir novos bitcoins na rede, que serão transferidos para o minerador como recompensa por ter minerado o bloco. Como essa novas moedas acabaram de ser criadas essa transação não possui um *input*. A estrutura de uma transação coinbase a apresentada na Figura 3.

Campo		Tamanho
nVersion		int 4bytes
#vin		VarInt 1-9 bytes
vin[ ]	hash	uint256 32 bytes
	n	uint 4 bytes
	coinbaseLen	VarInt 1-9 bytes
	coinbase	CScript Variable
	nSequence	uint 4 bytes
#vout		VarInt 1-3 bytes
vout[ ]	nValue	int64_t 8 bytes
	scriptPubkeyLen	VarInt 1-9 bytes
	scriptPubkey	CScript Variable
nLockTime		unsigned int 4 bytes

Figura 3. Estrutura de uma transação coinbase [3]

A única alteração na transação coinbase são os campos *coinbaseLen*, que armazena o tamanho do segundo campo que é o *coinbase*. O campo *coinbase* funciona como um script de travamanto, por isso é também conhecido por coinbase script. Ele armazena a altura que o blockchain deve ter, a partir do bloco que possui a transação coinbase, para que se possa gastar essas novas moedas. A estrutura do campo *coinbase* é apresentada na Figura 3.

Campo	Tamanho (bytes)
blockHeightLen	1
blockHeight	blockHeightLen
arbitraryData	coinbaseLen - (blockHeightLen + 1)

Figura 4. Estrutura do campo coinbase [3]

O primeiro campo *blockHeightLen* armazena o tamanho do *blockHeight* que possui a altura do bloco. O dados do campo *arbitraryData* devem ser aleatórios e podem ser utilizados no algoritmo de prova-de-trabalho.

2.3. Validando transações

As transações Bitcoin são validadas através de uma linguagem script. São utilizados dois tipos de scripts para a validação: um script de travamento e um de destravamento.

Um script de travamento é uma imposição inserida em um output especificando as condições necessárias para se gastar um UTXO no futuro. O script de travamento é conhecido como *scriptPubKey* pois ele continha uma chave publica ou endereço Bitcoin, porém atualmente utiliza-se uma gama muito maior de possibilidades de script.

Um script de destravamento satisfaz as condições que forem colocas em um UTXO por um script de travamento, permitindo que o UTXO seja gasto. Eles fazem parte de todos os inputs de transação e na maioria das vezes contém uma assinatura digital produzida por uma carteira a partir de sua chave primaria, por isso é conhecido como *scriptSig*, mas nem todos os scripts de destravamento contém uma assinatura digital.

Linguagem Script

A linguagem script das transações Bitcoin utiliza a notação polonesa invertida, baseada em pilha(*stack*). Uma pilha possui apenas duas operações, empilhar(push) e desempilhar(pop). A operação push insere um item no topo da pilha enquanto a operação pop retira um item do topo da pilha.

Ela não é uma linguagem Turing Completa, o que significa que não se pode criar loops ou algo que de alguma maneira poderia criar um ataque de negação de serviço na rede. Outra forte característica é que ela não exige um monitoramento de estado, ou seja, não existe um estado salvo após a execução do script, toda a informação necessária para executar o script está contida no script. Isso significa que uma transação valida será valida para toda a rede.

Essa linguagem script possui dois tipos de dados, constantes de dados(números) e operadores, e executa-os da esquerda para direita. Os operadores podem desempilhar(pop) constantes de dados, opera-las e empilha-las(push) novamente. Enquanto uma constantes de dados são sempre empilhadas(push).

Atualmente existem cinco tipos padrões de scripts de transação: *pay-to-public-key-hash*(P2PKH), *Pay-to-Public-Key*(P2PK), *multi-signature*, *Pay-to-script-hash*(P2SH) e

data output (OP_RETURN). Todos esse script serão abordados a seguir.

Pay-to-Public-Key-Hash

O script de transação Pay-to-Public-Key-Hash(P2PKH) utiliza um script de travamento que disponibiliza um *output* para um endereço Bitcoin, que é o hash da chave pública da carteira. Esse *output* pode ser destravado ao se apresentar a chave pública e a assinatura digital criada pela chave privada correspondente.

Para melhor entender o funcionamento desse script vamos supor que Bob deseja fazer uma transação de uma quantia de bitcoins que existe em sua carteira para a carteira de um café de sua cidade. O script de travamento do UTXO que será inserido no *input* dessa transação possui a seguinte forma.

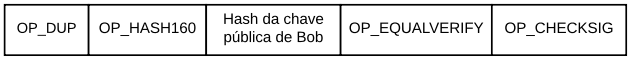

Figura 5. Exemplo de script de travamento P2PHKH

O script de destravamento que satisfaz o script de travamento P2PKH é o seguinte.

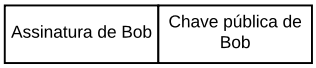

Figura 6. Exemplo de script de destravamento P2PHKH

Ao juntarmos os scripts de travamento e destravamento temos o seguinte script de validação.

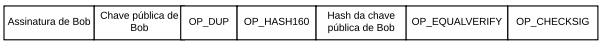

Figura 7. Exemplo de script de validação P2PHKH

Esse script de validação será executado na seguinte forma.

1) A assinatura de Bob é inserido no topo da pilha;
2) A chave pública de Bob é inserida no topo da pilha, acima da assinatura de Bob;
3) O operador OP_DUP é executado, ele duplica o dado que está no topo da pilha e o resultado é empilhado;
4) O operador OP_HASH160 é executado, ele aplica o hash RIPEMD160(SHA256) no dado que está no topo da pilha e empilha o resultado;
5) O hash da chave pública (endereco Bitcoin) de Bob é empilhado;
6) A operaćão OP_EQUALVERIFY é executada, essa operação retira da pilha dois dados mais próximos ao topo e os compara. No caso do P2PKH será comparado o hash da chave pública de Bob empilhada em 5 com o calculada em 4. Se os hashs forem igual ambos são removidos e a operação continua;
7) É executada a operação OP_CHECKSIG, ela verifica se a assinatura de Bob combina com a chave pública fornecida, se a assinatura combinar com a chave pública fornecida a operação empilha o valor TRUE na pilha. As assinaturas no Bitcoin utilizam o Algoritmo de Assinatura Digital de Curvas Elípticas(ECDSA).

Pay-to-Public-Key

O script Pay-to-Public-Key(P2PK) utiliza a chave pública no script de travamento, ao invés do hash da chave pública como no P2PKH.

O script de travamento do P2PK possui a seguinte forma.

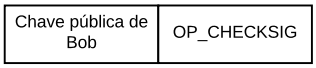

Figura 8. Exemplo de script de travamento P2PK

O script de destravamento que deve ser apresentado para se utilizar o UTXO é apenas uma assinatura a partir da chave privada de Bob.

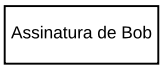

Figura 9. Exemplo de script de destravamento P2PK

Ao juntarmos os scripts de travamento e destravamento temos o seguinte script de validação.

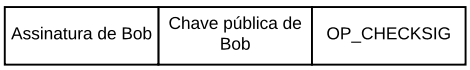

Figura 10. Exemplo de script de validação P2PK

Esse script é bem mais simples que o P2PKH, porém o seu script de travamento ocupa muito mais espaço já que a chave pública é maior que seu hash, o que dificulta seu uso. O script P2PK é mais utilizado hoje por softwares de mineração mais antigos que não foram atualizados para utilizarem o P2PHK.

Multi-signature

Esse script de transação define uma condição onde N chaves públicas são registradas no script e pelo menos M dessas chaves devem ser fornecidas para a liberação de tal UTXO. Essa condição é conhecida como um esquema M-de-N, onde N é o número total de chaves e M é o número de assinaturas necessárias para a liberação do UTXO.

O script de travamento de um script Multi-signature é da seguinte forma.

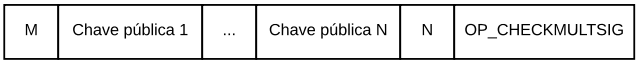

Figura 11. Exemplo de script de travamento Multi-signature

O script de destravamento Multi-signature deve apresentar M-de-N assinaturas.

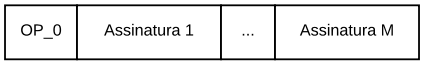

Figura 12. Exemplo de script de destravamento Multi-signature

Ao juntarmos os scripts de travamento e destravamento temos o seguinte script de validação.

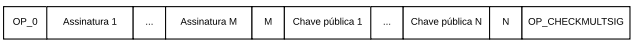

Figura 13. Exemplo de script de validação Multi-signature

Os scripts Multi-signature são os exemplos mais comuns das avançadas capacidades do script Bitcoin e são uma funcionalidade muito poderosa. Como é necessário mais de uma assinatura para liberação dos fundos, um esquema de múltiplas assinaturas como esse oferece um controle de governança corporativa e protege os fundos contra roubo, desvios ou perdas.

Apesar de todas as vantagens dos scripts Multi-signature suas desvantagens acabam inviabilizando seu uso. As transações Multi-signature são bem maiores que as transações de pagamento simples, pois elas contém chaves públicas muito longas. O fardo de uma transação extra-grande carregado pelo sistema teria que ser compensado com uma taxa de transação bem maior que as pagas pelas transações de menor tamanho, e além disso esses scripts extra-grandes teriam que ser carregados no *pool* UTXO de cada nó da rede até que seja gasto. Outro grande problema e talvez o maior é que para cada transação desse tipo teriamos um script de transação customizado, logo teríamos que ter uma carteira que possibilita a criação de tais scripts customizados e cada usuário teria que entender como criar transações com scripts customizados. Para resolver o problema dessa dificuldades práticas e fazer a utilização de scripts complexos tão fácil quanto o P2PKH foi desenvolvido o script Pay-to-Script-Hash(P2SH).

Pay-to-Script-Hash

Nos scripts Pay-to-Script-Hash(P2SH) é utilizado o hash criptográfico do script de travamento apresentado em 2.3, ou seja, o script de travamento complexo e substituído pela sua impressão digital. Esse hash criptográfico codificado na Base58Check corresponde ao endereço P2PH, assim como o hash da chave pública corresponde ao endereço Bitcoin. Com o endereco P2PH resolvemos todos os proplemas apresentados em 2.3. Agora basta fornecer o endereco P2SH para conseguir efetuar uma transacao, um processo simples, similar ao executado no script P2PKH.

Nas transações P2SH o script de travamento é o endereço P2PH e o script de travamento do Multi-signature aqui é conhecido como script de resgate(*redeem script*), pois ele deve ser apresentado na hora do resgate dos UTXOs. O script de destravamento deve conter as assinaturas necessárias para desbloquear os UTXOs, a condição M-de-N também devem ser satisfeita no P2SH. As Tabelas 1 e 2 apresentam os scripts no esquema de Mult-signature sem o P2SH e com o P2SH respectivamente.

Tabela 1. MULTI-SIGNATURE SEM P2SH

Script de travamento	M PK1 ... PKn N CHECKMULTISIG
Script de destravamento	Ass1 ... Assn

Tabela 2. MULTI-SIGNATURE COM P2SH

Script de resgate	M PK1 ... PKn N CHECKMULTISIG
Script de travamento	OP_HASH160 <endereço P2SH> OP_EQUALVERIFY
Script de destravamento	Ass1 ... Assn

Data Recording Output (OP_RETURN)

Todas as transações da rede Bitcoin são armazenas em um "banco de dados"distribuído chamado Blockchain, o funcionamento desse sistema será explicado mais adiante. Esse sistema potencial muito mais amplo do que apenas pagamentos, muitos desenvolvedores tentaram utilizar da segurança e resiliência da Blockchain para aplicações como cartórios digitais, certificados de ações e contratos *smart*. As primeiras tentativos surgiram ao tentar criar *outputs* de transação que registravam dados na Blockchain.

Quando a ideia de utilizar a Blockchain do Bitcoin consideram esse uso abusivo, enquanto outros acreditaram que isso mostrava a enorme capacidade dessa tecnologia. Os que eram contra argumentavam que isso causaria um "inchaço na blockchain", trazendo uma carga par os nós que armazenam a Blockchain em disco. Além disso, esse processo criaria falsas transações já que utilizam o campo de endereço do destinatário para armazenar dados, fazendo com que o UTXO jamais seja gasto. Essas transações que não podem ser gastas jamais seriam removidas do *pool* UTXO fazendo-o crescer infinitamente.

Para resolver esse problema na versão BitcoinCore 0.9.0 foi introduzido o operador OP_return, esse operador permitia que fosse adicionado 40 bytes de não-pagamento em um *output* de transação, na versão BitconCore 0.12.0 esse limite foi aumentado para 83 bytes. Esse operador cria *outputs* que são comprovadamente não gastáveis que não precisam ser registrados no *pool* UTXO.

2.4. Blockchain

A blockchain funciona como uma de registros distribuído e compartilhado. No bitcoin a blockchain funciona como um livro-razão onde são armazenadas todas as transações já feitas. Seu principal proposito é garantir uma ordem cronológica das transações e prevenir o problema de gasto duplo (double spending), apresentado por Hal Finney em seu whitepaper intitulado Digital Cash & Privacy.

A estrutura da blockchain é vista como um estrutura ordenada de blocos de dados com cada bloco sendo identificado pelo seu hash (SHA256). Cada bloco contém uma

referência ao bloco anterior (bloco pai) em seu cabeçalho, essa referência corresponde ao identificador do bloco anterior. Isso empoe uma ordem cronológica nos blocos e por consequência nas transações, já que não é possível criar o hash de um bloco que ainda não existe na blockchain. O único bloco que não possui a referência para seu bloco pai é o bloco gênese, pelo fato de ser o primeiro bloco da blockchain.

O bloco gênese é o ancestral comum de todos os blocos do blockchain. Todo nó começa com o bloco gênese em sua blockchain porque ele está estaticamente codificado no cliente bitcoin, de modo que ele não pode ser alterado sem recompilar o código-fonte. Como o nó conhece todos os dados da estrutura do bloco gênese, ele contém um ponto de partida da blockchain seguro, a partir do qual pode montar uma blockchain de confiança.

Devido cronologia existente entre os blocos podemos abstrair a blockchain como um empilhamento de blocos de dados, com cada novo bloco gerado sendo empilhado sobre o bloco mais recente da blockchain. Essa visualização de blocos empilhados possibilita a utilização de termos como "altura", para se referir a posição de um bloco em relação ao primeiro bloco do Blockchain (Bloco gênese), e "topo", para se referir ao bloco mais recente da blockchain.

Como cada bloco possui a o hash de seu bloco pai em seu cabeçalho isso irá influenciar no seu hash, assim, se ocorrer uma mudança em um bloco isso afetará seu hash que também afetará os blocos acima causando um efeito dominó. Esse efeito dominó garante a que um bloco que tenha algumas gerações o sucedendo não possa ser alterando, a não ser que haja um calculo forçado do hash de todos os blocos subsequentes. Essa é uma das características chave para a segurança da blockchain, pois a mudança de um bloco que já possui uma certa profundidade exigira um processo computacional enorme.

Os blocos também possuem a solução para o algoritmo de prova de trabalho. O trabalho computacional envolvido na solução do problema de prova de trabalho é utilizado como um esquema de votação entre os nós da rede para que eles concordem com a versão da blockchain. No geral, os nós concordam a versão da blockchain que envolva o maior poder computacional utilizado para ser criado.

Como os nós mineradores da rede competem na solução de prova de trabalho para incluir um bloco na blockchain, existe a possibilidade de dois blocos serem minerados ao mesmo tempo, causando um bifurcação na blockchain visto pela rede. Nesse caso os nós irão aceitar o bloco que receberem primeiro e continuar a construir a blockchain a partir daquele bloco. Assim que um novo bloco for adicionado a uma das bifurcações, aquela com maior altura será considerada a blockchain válida e as transações do bloco descartado serão adicionadas ao pool de transações.

Estrutura de um bloco

Cada bloco no blockchain do Bitcoin é composto por um cabeçalho e um payload e sua estrutura é apresentada na Tabela 3. O cabeçalho corresponde aos 6 primeiros campos e o payload aos dois últimos.

Tabela 3. ESTRUTURA DE UM BLOCO [3]

Campo	Tamanho
nVersion	int (4 bytes)
HashPrevBlock	uint256 (32 bytes)
HashMerkleRoot	uint256 (32 bytes)
nTime	unsigned int (4 bytes)
nBits	unsigned int (4 bytes)
nNonce	unsigned int (4 bytes)
#vtx	VarInt (1-9 bytes)
vtx[]	Variable

O campo *nVersion* armazena o número de versão como referências as atualizações de software/protocolo. O segundo campo *HashPrevBlock* corresponde ao duplo hash dos dados concatenados do cabeçalho bloco anterior (bloco pai), que é o identificador primário do bloco, como apresentado na Figura 14. Essa ligação entre os blocos é uma das características chaves que garante a segurança e resiliência do Blockchain e foi explicada do início dessa seção. O *HashMerkleRoot* armazena o hash da raiz da árvore de Merkle das transações desse bloco, a árvore de Merkle é explicada na seção 2.4. Os campos *nTime* e *nBits* correspondem ao tempo aproximado no formato UNIX em que o bloco foi criado e a dificuldade do algoritmo de prova-de-trabalho do bloco. O *nNonce* é uma sequência de bit aleatória e é utilizado como fonte de aleatoriedade para o algoritmo de prova-de-trabalho. Entretanto como o *nNonce* possui um tamanho muito pequeno ele não prove uma variância significante, devido a isso existem outras fontes que serão apresentadas com mais detalhes na seção **??**.

No payload temos os campos *#vtx* que armazena a quantidade de transações que o campo *vtx*[] possui. O campo *vtx*[] funciona como um vetor de transações. A estrutura de uma transações foi apresentada na seção 2.2.

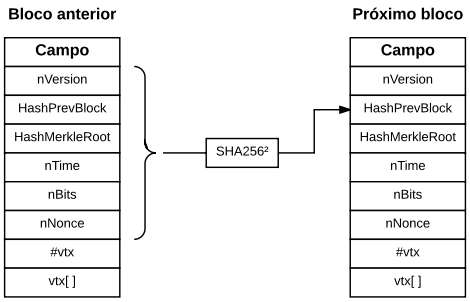

Figura 14. Referência entre blocos

Perceba que o não está incluído na estrutura de dados do bloco, nem quando o bloco é transmitido na rede e nem

quando ele é armazenado na blockchain. O hash deve ser calculado por cada nó assim que o bloco é recebido, e pode ser armazenado em uma tabela separada como parte dos metadados do bloco para facilitar a indexação e uma coleta mais rápida dos blocos em disco.

Árvode de Merkle

3. Conclusion

The conclusion goes here.

Acknowledgments

The authors would like to thank...

Referências

[1] A. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain*. O'Reilly Media, 2017.

[2] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008.

[3] K. Okupski. Bitcoin developer reference, 2016.