Graduação em Ciência da Computação

ALGORAND E OUTROS MECANISMOS DE CONSENSO EM PUBLIC LEDGER

Por

FELIPE TORRES MAGALHÃES

Trabalho de Graduação

Universidade Federal de Pernambuco
secgrad@cin.ufpe.br
www.cin.ufpe.br/~secgrad

RECIFE/2017

Felipe Torres Magalhães

ALGORAND E OUTROS MECANISMOS DE CONSENSO EM PUBLIC LEDGER

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Ruy José Guerra Barretto de Queiroz

RECIFE
2017

Agradecimentos

Agradeço ao meu orientador, professor Ruy de Queiroz, pelo seu apoio e orientação durante a realização desse trabalho. Agradeço aos meus colegas e aos amigos que fiz na universidade, por dividirem as angústias e alegrias do curso comigo. Por fim, agradeço a minha família pelo apoio dado em toda minha vida, em especial a minha mãe, Diana Torres Barros da Silva. Muito obrigado!

Resumo

Public ledgers são sequências de dados que podem ser lidas e escritas por qualquer um. Tais estruturas são invioláveis, de modo a impedir qualquer modificação a alguma informação já escrita. Devido a essas características, *public ledgers* são comumente implementadas para assegurar transações. Entre essas implementações, a mais utilizada é a blockchain, registro público de transações da popular criptomoeda, Bitcoin. Entretanto, o mecanismo de consenso - método utilizado para decidir quais pagamentos são adicionados à blockchain - gera um enorme desperdício de energia, é lento e estratifica a comunidade da moeda digital. É neste contexto que surge Algorand, *public ledger* com um novo mecanismo de consenso, proposta pelo professor Silvio Micali do MIT. O objetivo deste trabalho é trazer um estudo sobre este novo mecanismo, analisando e assertando sua viabilidade como mecanismo de consenso, além de outras alternativas.

Palavras-chave: Algorand, Bitcoin, *blockchain*, consenso, *public ledger*

Abstract

Public ledgers are sequences of data that can be read and written by anybody. They are tamper-proof, in order do prevent any modifications to some information already written. Because of those characteristics, public ledgers are commonly used to secure transactions. Among those implementations, the most widely used is the blockchain, public record of transactions of the popular cryptocurrency Bitcoin. However, the consensus mechanism - method used to decide which payments are added to the blockchain - creates a huge waste of energy, is slow and stratifies the digital currency community. It's in this context that Algorand emerges, a public ledger with a new consensus mechanism, proposed by MIT professor Silvio Micali. The objective of this work is to present a study about this new mechanism, analyzing and asserting its feasibility as a consensus mechanism, besides other alternatives.

Keywords: Algorand, Bitcoin, blockchain, consensus, public ledger

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

Sumário

1

Introdução

1.1 Contextualização e Motivação

Em computação, *public ledgers* (ou registros públicos) são sequências de dados invioláveis que podem ser lidas e escritas por qualquer um (MICALI (2016)). A qualidade de inviolabilidade é importante, pois ela expressa que, uma vez que alguma informação seja escrita, esta jamais poderá ser apagada ou modificada, sem que seja notado. Tal característica é assegurada através de mecanismos criptográficos (NAKAMOTO (2008)). Normalmente, é preciso que alguém ou um grupo de usuários decida quais informações serão adicionadas ao registro. Outro fator importante, é que cada unidade de dados aparece na ordem em que é escrito (NAKAMOTO (2008)), conferindo a essas estruturas a qualidade de registro histórico. Esse conjunto de recursos torna *public ledgers* ideais para assegurar transações de qualquer tipo.

Entre diversas implementações de *public ledgers*, a mais famosa é o registro de transações da moeda digital Bitcoin, conhecida como *blockchain* NARAYANAN et al. (2016). Bitcoin foi conceitualizada em 2008 por Satoshi Nakamoto e lançada em 2009 porém, suas origens datam da década de 1980, com o movimento *cypherpunk* (NARAYANAN et al. (2016)). Os *cypherpunks* eram um grupo de criptógrafos ativistas que advogavam o uso de criptografia e outras tecnologias de segurança como meio para mudança social e política (NARAYANAN et al. (2016)). O *white paper* de Bitcoin tem como referências propostas desse grupo de pessoas - a maioria eram sistemas de pagamento digital descentralizados (NAKAMOTO (2008)). Hoje, Bitcoin é a criptomoeda mais utilizada no mundo (NARAYANAN et al. (2016)). A *blockchain* consiste em uma "corrente" de blocos, onde cada bloco contém uma série de informações como transações (pagamentos realizados com a moeda), e um apontador para o bloco anterior, de maneira que, caso algum agente malicioso deseje modificar um desses blocos, para benefício próprio, seria necessário alterar todos os anteriores (NAKAMOTO (2008)). Assim como os sistemas nos quais se inspirou, Bitcoin é uma criptomoeda descentralizada, ou seja, não existe uma única entidade validando pagamentos e decidindo quais blocos devem ser adicionados à *blockchain*. Em vez disso, a rede de usuários de Bitcoin precisa chegar a um acordo sobre quais blocos serão adicionados. Os usuários chegam a esse acordo, através de um mecanismo de consenso (NARAYANAN et al. (2016)).

Para decidir quais blocos devem ser adicionados a *blockchain*, Bitcoin implementa um mecanismo de consenso chamado *proof-of-work* (NAKAMOTO (2008)). Nessa abordagem, a rede considera válida a sequência de blocos com o maior trabalho aplicado. No caso de Bitcoin, a prova do trabalho requer que usuários utilizem o poder de suas CPUs para calcular um valor específico (NAKAMOTO (2008)). Esta espécie de quebra-cabeça requer alto poder computacional e aquele que conseguir resolver primeiro, tem seu bloco adicionado à *blockchain*. Mais detalhes sobre esse processo serão dados à frente. Entretanto, a abordagem *proof-of-work* gera diversos problemas: gera um enorme desperdício de energia, é lento e estratifica a comunidade da moeda digital - vamos discutir esses problemas no próximo capítulo. Por causa disso, outros mecanismos de consenso tem sido propostos com o intuito de resolver esses problemas. Recentemente, Algorand foi proposto pelo professor do *Massachusetts Institute of Technology* (MIT), Silvio Micali. Vamos estudar esse novo sistema, comparando com o mecanismo utilizado por Bitcoin e atestar sua viabilidade.

1.2 Objetivos deste trabalho

Este trabalho possui dois objetivos:

1. Fornecer um estudo sobre mecanismos de consenso já propostos e utilizados por criptomoedas, assim como Alogrand, a nova proposta de Silvio Micali;

2. Analisar e assertar a viabilidade de Algorand como mecanismo de consenso.

1.3 Estrutura e organização do documento

No capítulo 2 veremos as principais tecnologias utilizadas por criptomoedas: suas primitivas criptográficas, além dos tipos de mecanismos de consenso já propostos e implementados. O capítulo 3 cobre o estudo de Algorand. Veremos suas principais características e que tecnologias foram utilizadas para alcançar essas propriedades. No capítulo 4, apresentamos uma análise desse sistema e sua viabilidade como mecanismo de consenso. E no capítulo 5, apresentamos nossas conclusões sobre esse trabalho.

2

Tecnologias de criptomoedas

Neste capítulo, veremos quais as principais tecnologias utilizadas por criptomoedas. O capítulo é dividido em 3 seções: primitivas criptográficas, estruturas de dados e mecanismos de consenso. Além destes, também incluímos um capítulo para discutir os problemas presentes no mecanismo de consenso de Bitcoin.

2.1 Primitivas criptográficas

Tipicamente, esquemas de *public ledger*, utilizam funções hash e assinaturas digitais para assegurar a integridade e autenticidade de seus registros.

2.1.1 Função hash criptográfica

Uma função hash criptográfica H toma como entrada uma mensagem m de qualquer tamanho, e produz uma saída h de tamanho fixo. Atualmente, uma das funções mais utilizadas - principalmente no meio das criptomoedas - é a SHA-256, que faz parte da família de funções criptográficas *Secure Hash Algorithm 2* (SHA-2) (NIST (2015)). A SHA-256 possui esse nome, pois é a variante que retorna valores h de tamanho 256 bits. Para ser considerada segura, tal função deve possuir as seguintes propriedades (SMART (2004)):

1. Resistência à pré-imagem: dado um valor h deve ser difícil encontrar qualquer mensagem m tal que $h = H(m)$;
2. Resistência à colisão: deve ser difícil encontrar duas mensagens m_1 e m_2 tais que $H(m_1) = H(m_2)$;
3. Resistência à segunda pré-imagem: dada uma entrada m_1 deve ser difícil encontrar m_2 tal que $H(m_1) = H(m_2)$.

2.1.2 Assinaturas digitais

Permite a usuários autenticarem informação sem o comprometimento de dados sigilosos. Um esquema de assinatura digital consiste de 3 funções: uma função geradora de par de chaves

G; uma de assinatura S; e uma verificadora V.

Um usuário i utiliza um parâmetro de segurança k, suficientemente grande, para a partir de G, gerar dois pares de *strings* de tamanho k bits:

$$G(k) = (pk_i, sk_i).$$

Essas *strings* são denominadas chaves onde, pk_i é a chave pública, e sk_i a chave privada. Como os nomes indicam, sk_i é um valor privado que só o usuário i deve conhecer, enquanto pk_i pode ser revelado a público sem comprometer sk_i, ou seja, nenhum usuário $j \neq i$ deve ser capaz de calcular sk_i a partir de pk_i em tempo hábil.

Um usuário i utiliza sk_i para, a partir da função de assinatura S, assinar uma mensagem m. Como essas funções recebem e dão como saída *strings* binárias de tamanho fixo, o usuário i, na verdade, assina o hash de m, $H(m)$ e produz uma *string* de k bits:

$$S(H(m), sk_i) = sig_{sk_i}(m).$$

Dizemos que $sig_{sk_i}(m)$ é a mensagem m assinada digitalmente pelo usuário i, ou $sig_i(m)$ por simplicidade. Visto que a chave pública de i, pk_i, é conhecida, qualquer usuário pode verificar uma mensagem, alegadamente, assinada por i utilizando a função de validação V. Tal algoritmo deve dar como resposta $TRUE$, caso a assinatura seja válida, ou *FALSE* caso contrário. (MICALI (2016)) dita 3 propriedades necessárias para esse esquema, se $s = sig_i(m)$:

1. Assinaturas legítimas são sempre aceitas pela função verificadora, ou seja, $V(pk_i, m, s) = TRUE$;

2. Assinaturas digitais devem ser computacionalmente inviáveis de forjar, isto é, qualquer usuário sem conhecer sk_i, não é capaz de encontrar uma assinatura s tal que $V(pk_i, m, s) = TRUE$ para uma mensagem m nunca assinada por i em tempo hábil, a não ser com probabilidade negligenciável;

3. Unicidade, ou seja, é inviável que um usuário $j \neq i$ ache pk', m, s' tal que $s \neq s'$ e $V(pk', m, s) = V(pk', m, s') = TRUE$ em tempo hábil.

Além destas propriedades, desde que sua assinatura $sig_i(m)$ seja legítima, um outro usuário, j, que recebe a assinatura, possui três importantes propriedades de segurança garantidas (SMART (2004)):

Autenticidade: o receptor j deve ser capaz de confirmar que a assinatura $sig_i(m)$ foi feita pelo emissor i;

Integridade: qualquer alteração feita à mensagem m durante sua transmissão, causa a assinatura $sig_i(m)$ a não corresponder à mensagem alterada;

Não-repudiação: o emissor *i* não pode repudiar a autenticidade de sua assinatura $sig_i(m)$.

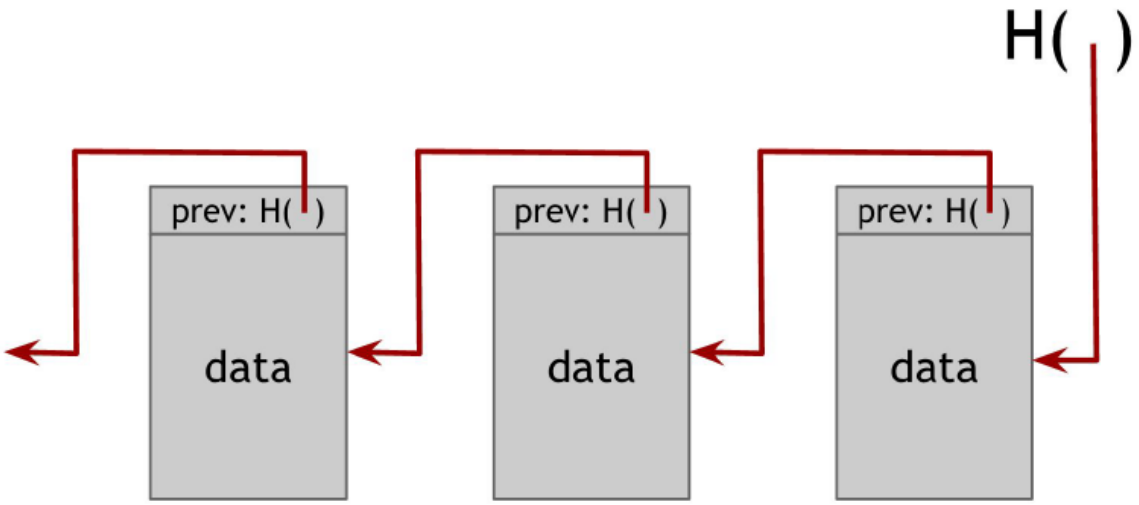
Vale ressaltar, que essas propriedades devem se manter desde que o usuário não revele sua chave privada. Além disso, esquemas de assinatura digital possuem uma propriedade implícita, a da irrecuperabilidade. Esta diz que não é possível recuperar uma mensagem *m* a partir da assinatura digital $sig_i(m)$, por um usuário *i*. Por isso, caso *i* deseje que sua assinatura seja validada por outros, ele precisa fornecer a mensagem *m* também.

2.2 Ponteiros hash e suas estruturas de dados

Nesta seção, veremos quais tecnologias são utilizadas por Bitcoin, e outras criptomoedas, para organizar seus dados. A principal técnica empregada nas estruturas de dados são ponteiros hash. Assim como um ponteiro normal, ponteiros hash servem como meio para acessar informação, visto que são apontadores para algum dado. Porém, diferentemente dos ponteiros usuais, um ponteiro hash também aponta para um hash criptográfico desta informação. Por causa disso, um ponteiro hash também serve como meio para comprovar que uma informação não foi alterada (NARAYANAN et al. (2016)). Além disso, podemos construir diversos tipos de estruturas de dados a partir de ponteiros hash, assim como os usuais (NARAYANAN et al. (2016)). Veremos duas dessas estruturas nesta seção: a *blockchain*, e a árvore de Merkle.

Uma *blockchain* é semelhante a uma lista encadeada, onde encontramos uma série de blocos, cada um contendo algum dado e um apontador para o bloco anterior, exceto que, na *blockchain*, este apontador é um ponteiro hash (NARAYANAN et al. (2016)). Assim, cada bloco é capaz de não só nos informar qual bloco o precede, como também, uma proteção à integridade dos dados armazenados no bloco anterior. A *blockchain* também contém a cabeça da lista, um ponteiro hash para o bloco mais recente, e o primeiro bloco da lista é conhecido como bloco gênesis (NARAYANAN et al. (2016)). Por causa desta configuração, *blockchains* podem ser utilizadas como um "registro de evidência de violação"(NARAYANAN et al. (2016)). Ou seja, elas não só funcionam como uma estrutura de dados, em que podemos adicionar novos blocos de dados ao final desse registro, como também podem nos indicar caso algum desses blocos seja alterado.

Considere que um adversário deseja alterar algum bloco da *blockchain*, de modo a não ser notado por alguém que conheça apenas a cabeça da lista. Caso o adversário altere os dados de um bloco *k*, o valor de seu hash seria diferente do encontrado no ponteiro do bloco seguinte, $k+1$, visto que estamos estatisticamente protegidos devido à propriedade 3 de uma função hash criptográfica. Ou seja, o adversário teria que alterar também o ponteiro do bloco $k+1$. Porém, fazendo isso, eles estaria alterando o valor desse bloco, e seria preciso fazer alterações no ponteiro do bloco seguinte, e assim por diante. Eventualmente, o adversário precisaria alterar a cabeça da lista, mas como esse valor é conhecido, tal violação aos dados da *blockchain* será detectada.

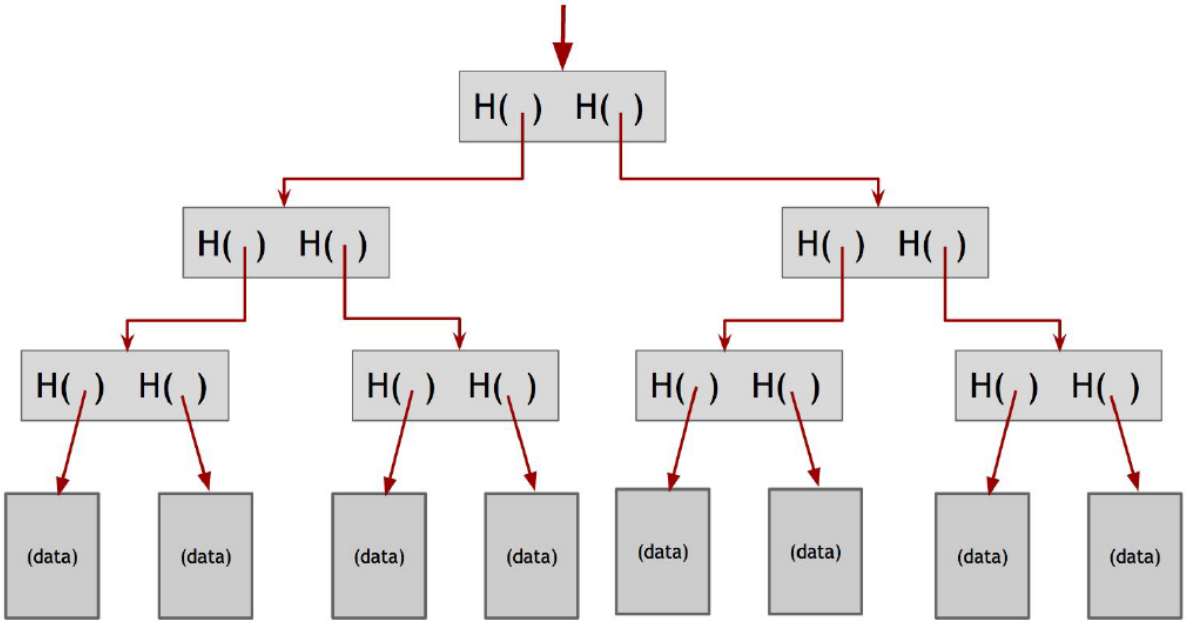


Fonte: NARAYANAN et al. (2016)

Figura 2.1: A *blockchain* é uma sequência de blocos contendo dados e um ponteiro hash para o bloco anterior. A *blockchain* também possui um ponteiro hash para o bloco mais recente, conhecido como cabeça da lista.

De maneira geral, uma árvore de Merkle é uma árvore binária com ponteiros hash, e foi assim nomeada em referência a seu inventor, Ralph Merkle (NARAYANAN et al. (2016)). Uma árvore de Merkle é composta por blocos de dados, que compreendem as folhas da árvore - elementos do nível mais profundo. Esses blocos são agrupados em pares, e para cada par, construímos uma estrutura contendo ponteiros hash para cada um dos blocos desse par, que formam o próximo nível da árvore. Em seguida, sempre que possível, formamos pares desses ponteiros, e para cada par, criamos um apontador hash. Isto é feito até chegarmos em um único bloco, a raiz, no nível mais alto da árvore (NARAYANAN et al. (2016)). Assim como a *blockchain*, árvores de Merkle também possuem a propriedade apropriada para que sirvam como um registro de evidência de violação, porém, apresentam outras propriedades interessantes.

Uma dessas propriedades é a prova de associação em que, caso se deseje provar que algum bloco de dados pertença à árvore, tudo que se precisa fazer é mostrar o caminho da raiz até o bloco em questão. Se conhecermos a raiz da árvore, podemos verificar os hashes do caminho até o bloco e comprovar se ele é um membro da árvore ou não. Além disso, por se tratar de uma árvore binária, caso n seja o número de nós, só precisamos passar por $\log n$ nós para essa comprovação. E considerando que cada passo requer apenas uma computação de hash, a prova de associação leva $O(\log n)$ para ser feita (NARAYANAN et al. (2016)). Uma outra propriedade, é a prova de não-associação. Esta requer uma árvore de Merkle ordenada, em que as folhas estão ordenadas segundo alguma função de ordenação - seja alfabética, lexicográfica, numérica, etc (NARAYANAN et al. (2016)). Assim, podemos provar que algum bloco não é membro de uma árvore, mostrando o caminho para o bloco anterior ao bloco em questão e o posterior. Caso esses dois itens sejam consecutivos, isto serve como prova que o bloco em questão não é membro da



Fonte: NARAYANAN et al. (2016)

Figura 2.2: Representação gráfica de uma árvore de Merkle. As folhas deste tipo de árvore são blocos que armazenam informação.

árvore, pois caso fosse, encontrar-se-ia entre os dois blocos encontrados (NARAYANAN et al. (2016)).

Em nosso contexto, cada bloco da *blockchain* inclui um ponteiro hash que é a raiz de uma árvore de Merkle, e cada folha dessa árvore diz respeito a uma transação. Outros tipos de informação inclusos nos blocos da *blockchain* e das árvores de Merkle de outras criptomoedas podem variar, mas todas costumam implementar as mesmas estruturas apresentadas nesta seção.

2.3 Mecanismos de consenso

Nesta seção veremos os tipos de mecanismos de consenso já propostos e implementados por criptomoedas. Podemos dividi-los em dois tipos: *proof-of-work* e *proof-of-stake*.

2.3.1 Proof-of-work

A abordagem *Proof-of-Work* (PoW), como já mencionado, é a mais difundida entre as *blockchains*, e ela prioriza o ramo da *blockchain* com o maior "trabalho" aplicado. Esse trabalho diz respeito a quanto processamento foi posto para realizar alguma tarefa, normalmente, na forma de algum quebra-cabeça criptográfico. A difusão de PoW pode ser atribuída à popularidade de Bitcoin, com seu consenso de Nakamoto. Antes de Bitcoin, propostas de outras criptomoedas já tinham sido lançadas, mas nenhuma vingou como a criação de Nakamoto. Após seu lançamento, tivemos um *boom* de criptomoedas, também conhecidas como *altcoins* (NARAYANAN et al. (2016)). Muitas delas, vieram de alterações ao núcleo de Bitcoin - ou seja, às suas regras e protocolos - e por isso, acabaram herdando a mesma abordagem PoW.

Como a maioria das *altcoins* vieram de alterações a Bitcoin, elas utilizam um teste computacional semelhante ao consenso de Nakamoto: os usuários utilizam um valor representativo de um bloco contendo pagamentos de usuários da rede para, a partir disso, calcular um valor específico. Para incentivar que seus usuários estejam sempre procurando por novos blocos, mecanismos PoW costumam implementar um sistema de recompensa, normalmente com o usuário que descobriu o bloco sendo recompensado com alguma quantia da moeda do sistema. Como visto no capítulo anterior, essa abordagem gera alguns problemas, e o mais preocupante é a sua baixa escalabilidade, isto é, a rede Bitcoin, por exemplo, possui uma taxa de processamento de pagamentos muito baixa, se comparado com outros sistemas de pagamento digital, como PayPal, e principalmente com as formas mais tradicionais, como Mastercard e Visa (NARAYANAN et al. (2016)). O consenso de Nakamoto, e seus derivados, utilizam uma abordagem PoW orientada a processamento, em que os seus testes computacionais requerem um alto poder de processamento para serem resolvidos em tempo hábil. Há uma outra abordagem, orientada a memória, cujos quebra-cabeças envolvem várias operações de escrita e leitura em memória RAM.

Exemplos de criptomoedas que utilizam a abordagem PoW incluem: Namecoin, criptomoeda cujo objetivo é fornecer uma versão descentralizada do Domain Name System (DNS) - que mapeia nomes de domínio a endereços IP -, atribuindo o nome de um domínio da web - diferentemente do valor de uma moeda - a uma chave pública; Dogecoin, uma *altcoin*, que apesar de não fornecer muito em inovação, teve bastante sucesso em seu lançamento, devido a suas campanhas de marketing humorosas; e Litecoin, uma *altcoin* que utiliza um quebra-cabeça baseado em memória e, recentemente, ativou com sucesso o *Segregated Witness* (SegWit) (CoinDesk (2017)). SegWit é um mecanismo que promete resolver o problema da escalabilidade de Bitcoin e criptomoedas semelhantes, implementando modificações na estrutura do bloco, o que torna operações de validação mais rápidas (BitcoinCore (2016)).

2.3.2 Proof-of-stake

A abordagem *Proof-of-Stake* (PoS) tem sido a principal alternativa a *proof-of-work*. Mecanismos de consenso que implementam esta abordagem são tipicamente caracterizados por colocar um maior poder de decisão nos usuários com maior concentração de moedas (NARAYANAN et al. (2016)). Como o nome indica, a "prova de risco" envolve em usuários aplicarem dinheiro no sistema para poderem participar do seu mecanismo de consenso. Assim, usuários com mais moedas deste sistema - "maior risco" - terão mais chances de serem escolhidos para a determinação de qual bloco deve ser adicionado ao registro. Diferentemente de mecanismos da abordagem PoW no qual os usuários disputam entre si para descobrirem qual bloco será adicionado ao registro - prática apelidada de "mineração" - em PoS, como não há um quebra-cabeça envolvido, os usuários participantes simplesmente geram um novo bloco - a prática de "forjar" um bloco (NARAYANAN et al. (2016)).

O exemplo mais famoso de uma criptomoeda *proof-of-stake* é a Peercoin. Também conhecida como PPCoin foi a primeira criptomoeda PoS mas, faz isso através de um abordagem híbrida PoW/PoS (NARAYANAN et al. (2016)). Além disso, Peercoin possui também uma outra característica marcante: a existência de administradores que assinam certos blocos, que funcionam como *checkpoints* (NARAYANAN et al. (2016)). Esse tipo de medida é para evitar o aparecimento de bifurcações na *blockchain* mas acaba causando um outro tipo de problema. A existência de usuários com esse tipo de poder significa que o sistema não é verdadeiramente descentralizado. Além disso, não há como garantir que o sistema seja seguro sem a interferência desses usuários.

Uma outra criptomoeda que merece ser mencionada nesta seção é Ethereum (Ethereum Project (2015)). Como criptomoeda, distingue-se das outras, pois possui uma premissa ambiciosa: fornecer uma *blockchain* para qualquer tipo de aplicação. Em Ethereum, ao pagar uma certa taxa, um usuário pode incluir o código-fonte de sua aplicação, ou contrato, na *blockchain* através de uma transação. Esse contrato é executado por uma máquina virtual, apelidada de EVM, e outros usuários podem utilizá-lo, de acordo com regras estabelecidas pelo usuário dono do contrato (NARAYANAN et al. (2016)). Lançado com uma implementação PoW semelhante ao consenso de Nakamoto, recentemente, Ethereum adotou uma abordagem híbrida PoW/PoS (Ethereum Project (2016)). Os mineradores de Ethereum devem continuar a validar blocos pela abordagem PoW, porém, algum bloco - um a cada cem - são validados pela abordagem *proof-of-stake*. A intenção dos desenvolvedores de Ethereum é fazer a transição completa para PoS e este é o primeiro passo em busca disso (Ethereum Project (2016)).

2.4 Problemas no mecanismo de consenso de Bitcoin

Como visto anteriormente, apesar da *blockchain* ser a implementação de *public ledger* mais utilizada, e por consequência, seu mecanismo de consenso também, ela não é livre de problemas. O mecanismo de consenso utilizado em Bitcoin é conhecido como "consenso de Nakamoto"(NARAYANAN et al. (2016)), que tira seu nome de Satoshi Nakamoto, criador do Bitcoin. O consenso de Nakamoto é uma implementação da abordagem *proof-of-work*. A abordagem implementada por Bitcoin é notória por gerar um enorme desperdício de energia, por sua lentidão em confirmar novos blocos adicionados, e por estratificar a comunidade entre usuários que apenas fazem transações e os que se dedicam a minerar novos blocos (MICALI (2016)). Precisamos entender melhor este processo para sabermos por que e como isso acontece.

Em Bitcoin, o processo no qual usuários adicionam novos blocos à *blockchain* é chamado de mineração, e os usuários que participam disso são mineradores (NARAYANAN et al. (2016)). Para começar a minerar, é preciso se tornar um nó na rede de Bitcoin, ou seja, rodar o cliente de Bitcoin em um computador. Ao longo do tempo, transações com a moeda são feitas, e estas são propagadas pela rede - um nó manda para seus nós adjacentes transações efetuadas por ele e outras que ele recebeu. Um nó que deseja minerar um bloco deve validar transações recebidas,

seguindo as diretrizes do protocolo de Bitcoin (NARAYANAN et al. (2016)). O conjunto de transações que passarem por essa validação é utilizado pelo nó minerador para formar um bloco.

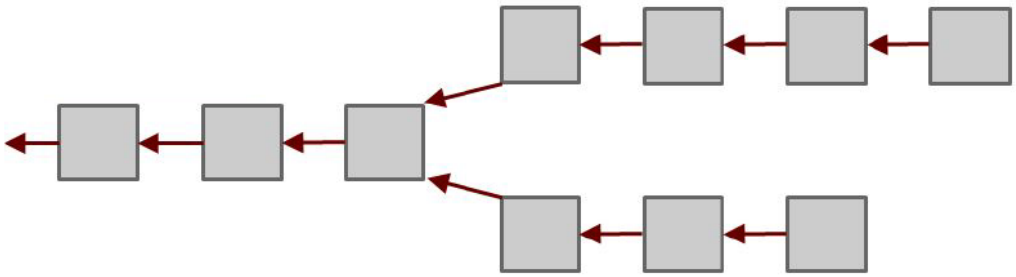
Desse modo, é provável que cada nó minerador acabe criando um bloco com transações distintas. Então, como decidir qual bloco deve fazer parte da *blockchain*? A cada dez minutos, os nós mineradores começam uma disputa para decidir qual bloco será incluso (NAKAMOTO (2008)). A partir de uma representação na forma de *string* - uma cadeia de caracteres - desse bloco, concatenado a um valor arbitrário chamado *nonce*, é calculada uma nova *string* através de uma função hash criptográfica (NARAYANAN et al. (2016)). A *string* que representa o bloco é uma concatenação do hash do bloco anterior, e todas as *strings* de identificação das transações do novo bloco, conhecida como *txid* (NARAYANAN et al. (2016)). Como vimos na subseção 2.1.1, pequenas mudanças em um desses valores de entrada resultam em saídas bastante diferentes, além da propriedade 1 que torna difícil inverter uma função hash criptográfica.

Bitcoin está interessado em um valor alvo específico. Assim, o nó minerador que encontrar este valor, tem seu bloco adicionado à *blockchain*. Por isso, a necessidade de utilizar valores arbitrários de *nonce* no cálculo. Entretanto, encontrar um valor $m = nonce \,||\, ponteiro\ hash \,||\, txids$ (a operação || significa concatenação) é uma tarefa muito difícil. De fato, a tarefa é tão difícil que, até algum nó minerador achar o valor desejado, terá gasto uma enorme quantidade de energia (NARAYANAN et al. (2016)). Note que toda essa energia é considerada um desperdício, visto que apenas o cálculo do valor correto é o que adiciona um bloco novo ao registro.

Como dito anteriormente, para realizar tal tarefa, é necessário máquinas com poder de processamento muito alto. Equipamentos assim custam caro, e combinado com o gasto da energia necessária para achar um bloco, usuários mineradores precisam ter um alto poder aquisitivo. Isso é ainda mais claro hoje, com a presença de mineradoras de Bitcoin, estabelecimentos montados com máquinas de grande poder computacional em ambientes altamente refrigerados com o propósito de minerar novos blocos. Claro, tanto interesse tem uma razão clara: Bitcoin implementa um sistema de recompensa para aqueles usuários que conseguirem minerar um bloco (NAKAMOTO (2008)). A cada novo bloco, uma quantidade específica de BTC (unidade de moeda de Bitcoin) é criado. Essas moedas são destinadas ao nó que minerou o bloco. Por isso, dependendo do local onde é minerado - países mais frios e/ou onde energia é mais barata - minerar Bitcoin pode ser uma atividade lucrativa, desde que se tenha os meios.

Um outro problema causado pelo mecanismo de consenso da *blockchain* é a lentidão com que pagamentos são confirmados. Bitcoin dita que um bloco novo deve ser descoberto a cada 10 minutos. Para tal, a dificuldade de encontrar o valor h é ajustada de acordo com a capacidade da rede de computar hash, medida em hash por segundo (hash/s) (NARAYANAN et al. (2016)). Espera-se, então, que pagamentos sejam confirmados a cada 10 minutos. Porém, na prática, não é isso o que acontece. Em Bitcoin, é comum ocorrer bifurcações na *blockchain*, conhecidas como *fork*. Um *fork* ocorre quando, em um dado momento, há mais de um ramo (do inglês, *branch*) de blocos, de mesma quantidade (ou altura), disputando para se tornar o maior ramo. Em Bitcoin, dizemos que o ramo válido é aquele de maior altura, ou o com maior trabalho

realizado - número de hashes computadas.



Fonte: NARAYANAN et al. (2016)

Figura 2.3: Representação gráfica de um *fork*: usuários consideram ramos distintos da *blockchain* como sendo o oficial.

Um *fork* ocorre por diversas razões, seja pela atividade de algum usuário malicioso, ou por disputa entre nós mineradores. O tipo mais grave ocorre quando um grupo de nós utiliza um cliente distinto do outro. Isso implica em diferenças no protocolo de validação, causando nós a enxergarem blocos válidos por um grupo como sendo inválidos e vice-versa. Esta bifurcação é chamada de *hard fork* (NARAYANAN et al. (2016)). Um outro tipo, mais brando, ocorre quando nós não estão de acordo com o(s) bloco(s) adicionado(s) à *blockchain*, mas todos os usuários enxergam ambos os ramos como válidos. Este tipo de bifurcação é conhecida como *soft fork* (NARAYANAN et al. (2016)). Neste trabalho, quando utilizarmos a expressão *fork*, estamos nos referindo a sua forma mais branda, por ser um problema pertinente ao nosso escopo. Devido a esse problema, é recomendado que após realizar algum pagamento em Bitcoin, o usuário deve esperar de 30 até 60 minutos para a sua confirmação (NARAYANAN et al. (2016)). Este costuma ser o tempo necessário para que um ramo consiga uma vantagem de pelo menos 3 blocos à frente de outro.

3

Algorand

3.1 Introdução

Neste capítulo, estudaremos Algorand como *public ledger* e sua implementação de mecanismo de consenso. Em seu trabalho *Algorand, The Efficient Public Ledger*, Silvio Micali, professor do MIT, descreve um novo método de atingir consenso em um *public ledger* descentralizado de maneira segura, rápida e econômica, tanto energetica quanto financialmente.

Micali primeiro define as primitivas criptográficas utilizadas, alguns termos e métodos ou mecanismos que serão utilizados ou aprimorados para o seu *public ledger*. Por questões didáticas, Micali começa definindo uma versão mais simples - e menos segura - de seu mecanismo de consenso. Em seguida, essa versão é aprimorada, até chegar em seu estágio final - e mais seguro. O nível de segurança dessas versões está de acordo com o poder do adversário que se pretende enfrentar. Um adversário com muitas limitações requer medidas de segurança mais simples e assim por diante. Nesse trabalho, iremos analisar apenas a versão final e mais segura de Algorand, visto que este é o sistema proposto por Micali.

Primeiro, iremos apontar algumas condições sob as quais Algorand atua tais como o tipo de rede, tamanho dos blocos, primitivas utilizadas, etc. Depois, apresentaremos as abordagens implementadas por Algorand em seu mecanismo de consenso. Exploraremos como Algorand atinge suas metas de velocidade, segurança e economia, entre outras ideias para garantir que o sistema funcione - como, por exemplo, métodos de recompensa, e garantia de que os usuários honestos prevalecerão sobre os maliciosos. No próximo capítulo, iremos estabelecer se Algorand é ou não um mecanismo de consenso viável para criptomoedas.

3.2 Principais propriedades

Nesta seção, vamos listar e explicar as principais propriedades de Algorand (como *blockchain* e mecanismo de consenso). Silvio Micali propõe um sistema "sem permissão", ou seja, novos usuários podem entrar no sistema a qualquer momento. Isso mostra-se um problema, visto que, Micali deseja construir um sistema resistente a um adversário dinâmico e altamente corruptor. Tal adversário, em princípio, seria capaz de corromper instantaneamente qualquer usuário a qualquer momento, e coordená-los perfeitamente - inclusive todos os usuários

verificadores, assim que forem revelados (MICALI (2016)). Para garantir a segurança, Algorand assume que o adversário não é capaz de introduzir usuários maliciosos ao sistema e que até 1/3 do dinheiro da rede está em posse de usuários maliciosos (MICALI (2016)).

Algorand utiliza o mesmo tamanho de mensagens que Bitcoin - cerca de 1KB para pagamentos e 1MB para blocos MICALI (2016) - assim, a velocidade de propagação dos blocos em ambas as redes deve ser a mesma - cerca de 1 segundo para pagamentos alcançarem 95% dos nós da rede e 1 minuto para blocos (MICALI (2016)). Este tempo é conhecido como latência da transmissão de bloco. Como já foi dito anteriormente, em Bitcoin, um bloco novo é minerado a cada 10 minutos NAKAMOTO (2008). Esta é a latência da geração de bloco. Entretanto, como há uma chance moderada de ocorrência de *fork*, usuários costumam esperar que um bloco esteja a pelo menos 3 níveis de altura na corrente. Por causa disso, a latência verdadeira acaba sendo cerca de 30 minutos - e não 10. Já em Algorand, desde que pelo menos 2/3 do dinheiro esteja no controle de usuários honestos, como não há *forks*, a não ser com uma chance negligenciável, e a computação necessária para gerar um bloco é mínima, a latência da geração de bloco é próxima da de propagação (MICALI (2016)). Logo, a latência total coincide com a de propagação.

Outra principal característica é o que Micali chama de honestidade preguiçosa - do inglês *lazy honesty*. Algorand trabalha com o pressuposto de que, não mais de 1/3 do dinheiro da rede está em mãos maliciosas. Entretanto, esperar que 2/3 dos usuários, além de serem honestos, estarão sempre online e participando ativamente do protocolo, é um pressuposto muito forte. Seja porque as máquinas estão desligadas ou por desconexões da rede de outro tipo, pode ser que a maioria dos usuários da rede sejam honestos, porém, a maior parte dos usuários ativos no protocolo sejam maliciosos. Por causa disso, Algorand consegue garantir a segurança de sua *blockchain* com uma noção mais relaxada de honestidade, onde usuários honestos podem ser preguiçosos. Em termos mais formais, dizemos que um usuário é honesto porém preguiçoso se ele segue as instruções necessárias quando é convocado para participar do protocolo, e isto ocorre raramente - uma vez por semana, por exemplo - com aviso prévio, e o usuário possivelmente sendo recompensado por participar (MICALI (2016)).

Além dessas características, Micali apresenta um novo acordo bizantino como principal mecanismo para escolha de novos blocos. Um acordo bizantino (*Byzantine Agreement* (BA)) é um protocolo de comunicação que permite a um conjunto de participantes, cada um de posse de um valor inicial, possivelmente diferente, concordarem em um único valor v (MICALI (2016)). Tal acordo é alcançado por todos os participantes honestos - que seguem o protocolo - mesmo na presença de participantes maliciosos - que desviam dos passos do protocolo, coordenadamente ou não. Em Algorand, os valores iniciais são blocos e um número suficiente de usuários participantes deve chegar a um consenso sobre qual bloco, dentre os iniciais, será adicionado ao registro (MICALI (2016)). Algorand faz uso de um acordo bizantino como substituto à abordagem *Proof-of-Work*, porém esta versão é mais rápida que as anteriores e com uma nova propriedade, a saber, substituição de participantes, que confere ao protocolo maior segurança (MICALI (2016)).

Algorand utiliza um sistema de votação através de uma implementação própria de

acordo bizantino. Entretanto, é inviável que todos os usuários da rede participem da votação. Não só tornaria o mecanismo lento, como também inviabilizaria sua escalabilidade. Para solucionar esse problema, Micali utiliza um método que ele apelida de "sorteio"criptográfico (do inglês, *cryptographic sortition*). Algorand escolhe, para o protocolo, um subconjunto de participantes, bem menor que o conjunto de todos os usuários (MICALI (2016)). Um subconjunto de participantes diferente deve ser escolhido para participar de cada rodada de escolha de bloco para evitar concentração de poder (MICALI (2016)). Uma rodada é a unidade lógica na qual Algorand está organizado. Para indicar que algum conjunto ou valor pertence a uma rodada, utilizamos um r sobrescrito. Por exemplo, dizemos que o bloco de uma rodada r é B^r, e que esse bloco é escolhido na rodada anterior, $r-1$. Para que a escolha seja aleatória, é utilizada uma função criptográfica para determinar automaticamente - sem troca de mensagens-, a partir do bloco anterior, o conjunto de verificadores encarregado de escolher o próximo bloco (MICALI (2016)). Para evitar que usuários maliciosos afetem a composição do bloco anterior - manipulando algumas de suas transações - são utilizadas entradas adicionais para garantir a aleatoriedade do conjunto de verificadores (MICALI (2016)).

3.3 Técnicas utilizadas

Nesta seção, veremos quais técnicas foram utilizadas e algumas desenvolvidas por Micali, para satisfazer as propriedades de Algorand discutidas mais adiante.

3.3.1 Usuários e chaves públicas

Em Algorand, assim como em Bitcoin, a identificação de usuários é feita por meio de chaves públicas. Cada chave pública aponta para uma quantidade de moedas do sistema, e quando um usuário deseja fazer algum pagamento, este ocorre transferindo uma certa quantidade de moedas de uma chave pública para outra. Para que um pagamento seja válido, há um conjunto de regras quanto a este procedimento que precisam ser seguidas. Devido à natureza sem permissão de Algorand, uma chave pública i junta-se ao sistema assim que uma outra chave j, já no sistema, faz um pagamento a i. Note que, um mesmo usuário, isto é, pessoa, pode possuir mais de uma chave pública no sistema.Vamos nos referir a PK como o conjunto de todas as chaves públicas do sistema.

3.3.2 A quantidade Q^r

Uma das variáveis mais importantes de Algorand, é a quantidade Q^r. O mecanismo de consenso, consiste de um líder ℓ^r, encarregado de gerar um bloco candidato B^r, e este será votado, seguindo um protocolo BA, pelo conjunto de verificadores selecionados, SV^r. Tanto ℓ^r, quanto SV^r poderão ser determinados por qualquer outro usuário da rede, sem troca de mensagem, a partir de informações obtidas do bloco anterior, B^{r-1} (MICALI (2016)). Esta é uma abordagem

delicada, visto que qualquer pagamento que o adversário conseguir incluir em B^{r-1}, pode lhe dar controle sobre SV^r e ℓ^r. Para garantir a imprevisibilidade de um bloco B^r, a quantidade Q^r é construída e a cada rodada, atualizada separadamente do bloco (MICALI (2016)).

A quantidade Q^r deve satisfazer os seguintes requisitos (MICALI (2016)):

1. Ser univocamente determinada por r e conhecida por todos os usuários ao final da rodada r;

2. Ser um valor aleatório representado por uma cadeia curta e independentemente selecionado;

3. Ser totalmente imprevisível, antes do final da rodada r, a não ser pelo líder ℓ^r - o único capaz de computar ℓ^r, no início de r, mas não antes disso.

Para satisfazer os três requerimentos, assumimos que toda rodada r tem um líder ℓ^r e que ele computa e revela, antes do fim de r, sua credencial e a sua assinatura de Q^{r-1}, $SIG_{\ell^r}(Q^r - 1)$ (MICALI (2016)). Calculamos Q^r como:

$$Q^r = H(SIG_{\ell^r}(Q^{r-1}), r-1),$$

e o adicionamos ao bloco B^r. Assim, satisfazemos os três requisitos, pois: uma vez em B^r, ele é determinado pela rodada e conhecido por todos (1); por ser a saída de uma função hash, é um valor aleatório curto e selecionado independentemente (2); e de seu cálculo, somente ℓ^r o conhece, antes de revelá-lo, desde que assim o faça, antes do final da rodada r (3).

3.3.3 Os verificadores

Os membros de SV^r participarão de um protocolo BA, e este consiste de várias etapas. Para evitar que um verificador seja corrompido durante o procolo, a cada passo, é escolhido um novo conjunto de verificadores (MICALI (2016)). $SV^{r,s}$ é o conjunto de usuários verificadores da rodada r em um passo s. $SV^{r,s}$ precisa ser escolhido aleatória e independentemente de $SV^{r,s-1}$, podendo até ter cardinalidade diferente, além de garantir que membros de $SV^{r,s-1}$ não saibam quem serão os participantes do próximo passo e nem troquem informações sobre o estado do protocolo. Esta propriedade de substituição de participantes é essencial para atingir a segurança contra o adversário idealizado (MICALI (2016)).

A escolha dos membros verificadores é feita para cada passo s de uma rodada r, entre os membros de uma rodada passada arbitrária, a rodada $r-k$. Algorand utiliza um mecanismo de seleção privada, *Secretive Verifier Selection* (SVS) que permite a todos os usuários descobrirem se pertencem ou não a $SV^{r,s}$, de maneira secreta, e permite a esses usuários provarem aos outros que este é o caso. O mecanismo $SVS^{-k,p}$, para determinar $SV^{r,s}$, através da quantidade Q^{r-1}, é definido da seguinte maneira (MICALI (2016)): seja k um inteiro positivo e p um valor real em

$[0,1]$. O mecanismo $SVS^{-k,p}$ permite definir um conjunto de verificadores de um passo s em uma rodada r, com parâmetro de *look-back* k e probabilidade p como:

$$SV^{r,s} \doteq \{i \in PK^{r-k} : .H(\sigma_i^{r,s}) \leq p\}$$

em que $\sigma_i^{r,s}$ denota $SIG_i(r,s,Q^{r-1})$, a credencial $r-s$ de i, se $i \in SV^{r,s}$. Note que o símbolo "." que precede a função H acima, é um ponto decimal utilizado para marcar o início de uma expansão binária de um número entre 0 e 1 (MICALI (2016)). Ou seja, $.H(\sigma_i^{r,s})$ é um número de 256 bits no intervalo $[0,1]$.

3.3.4 O líder

Assim como os verificadores, um usuário precisa aprender que é um líder de maneira secreta e privada. Caso contrário, o adversário poderia corromper esses usuários assim que um novo bloco for eleito, com a revelação da quantidade Q^{r-1}. Com o mecanismo SVS e a capacidade de gerar credenciais, ao aprender que é um líder, o usuário constrói e propaga um bloco candidato com suas credenciais ao sistema (MICALI (2016)). O adversário descobre a identidade do líder, mas qualquer chance de corrompê-lo é inútil, visto que o bloco candidato daquela rodada já foi propagado.

Algorand não garante que cada rodada tenha um líder único, nem que, caso exista, ele saiba que o é. Algorand garante que um usuário descobre, secretamente, que é um líder em potencial, e que, caso haja algum líder, ele é implicitamente único. Para essa descoberta, o usuário precisa, primeiro calcular sua credencial, $.H(\sigma_i^{r,s})$. Daí, dizemos que um usuário $i \in PK^{r-s}$ é um líder em potencial de uma rodada r se $.H(SIG_i(r,1,Q^{r-1})) \leq 1/|PK^{r-k}|$ e é líder de uma rodada r se $.H(SIG_i(r,1,Q^{r-1})) \leq .H(SIG_j(r,1,Q^{r-1}))$ para todo líder em potencial j. Uma rodada pode ficar sem líder caso não tenha nenhum líder em potencial, mas isso ocorre com probabilidade negligenciável (MICALI (2016)). Fora isso, caso não haja empates improváveis, uma rodada r tem um líder único ℓ^r e denotamos sua credencial, $SIG_{\ell^r}(r,1,Q^{r-1})$, por $\tau_{\ell^r}^r$ ou simplesmente τ^r.

3.3.5 Chaves efêmeras

Algorand requer que seus usuários utilizem assinaturas digitais para diversas tarefas. O par de chaves do sistema de um usuário i, pk_i e sk_i, é utilizado para assinar: seus pagamentos; sua credencial; e o par de chaves efêmeras que serão utilizadas na troca de mensagens do protocolo de votação, caso o usuário pertença ao conjunto de verificadores, $SV^{r,s}$ (MICALI (2016)). Por isso, a cada passo do protocolo, um usuário i deve, gerar um par de chaves efêmeras $pk_i^{r,s}$ e $sk_i^{r,s}$, e utilizar a chave efêmera privada $sk_i^{r,s}$ para assinar e propagar o valor de votação, $sig_i^{r,s}$, e então, destruir $sk_i^{r,s}$. O usuário então computa e propaga, com a chave privada do sistema sk_i, sua credencial, $\sigma_i^{r,s}$, e uma autenticação da chave efêmera pública $auth(pk_i^{r,s})$, para provar que

$pk_i^{r,s}$ é de fato sua chave efêmera. Note que, como $sk_i^{r,s}$ é destruída, o par de chaves efêmeras é também de uso único. Isto é para impedir que, caso o adversário descubra a identidade de algum usuário verificador, ele o corrompa e o utilize para mandar outras mensagens em um mesmo passo. Utilizar um par de chaves de uso único, dentro de um passo do protocolo, impede que o adversário faça isso.

3.4 Protocolo

Em linhas gerais, Algorand funciona da seguinte maneira: cada usuário, utiliza o valor Q^r junto a sua assinatura digital para verificar se é o líder ℓ^r ou faz parte do conjunto de verificadores, $SV^{r,s}$. Em caso positivo, o líder prepara um bloco candidato, a partir de transações que ele escutou na rede e foram validadas por ele e, então, propaga esse bloco junto a sua assinatura. Os usuários verificadores recebem o bloco candidato, verificam se foi mandado pelo líder daquela rodada, através de sua assinatura, e iniciam o protocolo de votação. Ao final, caso o bloco candidato não receba votos o suficiente, o protocolo deve dar como saída vazio (representado pelo símbolo $\perp$), não gerando um bloco novo naquela rodada. Caso o bloco candidato receba um número suficiente de assinaturas entre os usuários verificadores, ele é propagado junto com estas, para que todos os usuários possam atestar que este é, de fato, o bloco B^r escolhido.

Veremos, agora, como o protocolo BA de Algorand funciona. Silvio Micali propõe duas novas versões: um protocolo de acordo bizantino binário, BBA_m^*, e um de *Graded-Consensus* (GC) - ou consenso graduado. Ambos são adaptações de protocolos anteriormente propostos pelo próprio Micali, em co-autoria com Pesech Feldman (FELDMAN; MICALI (1997)). Esses protocolos utilizam a notação # para indicar quantas mensagens de outros participantes, um participante i já recebeu. $\#_i^s(v)$ é o número de participantes j dos quais i recebeu o valor v em um passo s. Como i recebe exatamente uma mensagem de cada usuário j, e se n for o número total de participantes, então, para todo i e s, $\sum_v \#_i^s(v) = n$. Esses protocolos também utilizam as seguintes variáveis: t um limitante superior para o número de usuários maliciosos durante uma execução do protocolo, no sistema; r uma *string* aleatória comum entre os usuários (a quantidade Q^{r-1}, no nosso caso); x uma *string* binária; e $lsb(x)$ indica o bit menos significativo de x.

3.4.1 Protocolo GC

O protocolo GC diz respeito à primeira parte do processo de votação. Definimos GC como um protocolo de (n,t)-consenso graduado se, em cada execução com n participantes, onde no máximo t são maliciosos, todo participante honesto i dá como saída um par valor-nota (v_i, g_i), onde $g_i \in \{0,1,2\}$ satisfazendo três propriedades (MICALI (2016)):

1. Para todos os participantes honestos i e j, $|g_i - g_j| \leq 1$;

2. Para todos os participantes honestos i e j, $g_i, g_j > 0 \Rightarrow v_i = v_j$;

3. Caso $v'_i = ... = v'_n = v$ para algum valor v, então $v_i = v$ e $g_i = 2$ para todo participante honesto i.

O protocolo GC consiste de 2 passos, que dizem respeito aos passos 2 e 3 do protocolo final: no passo 2, cada participante i envia v'_i para os outros participantes; no passo 3, cada participante i envia para todos os outros participantes a *string* x, se e somente se, $\#_i^2(x) \geq 2t+1$. Então, cada participante i dá como saída o par (v_i, g_i) de acordo com as seguintes condições (MICALI (2016)):

1. Se, para o valor x, $\#_i^3(x) \geq 2t+1$ então, $v_i = x$ e $g_i = 2$;
2. Se, para o valor x, $\#_i^3(x) \geq t+1$ então, $v_i = x$ e $g_i = 1$;
3. Caso contrário, $v_i = \perp$ e $g_i = 0$.

Tanto o valor inicial v'_i quanto a *string* x é uma representação do bloco candidato B^r sendo votado. Note que a nota g_i é computada de acordo com quantos usuários enviaram o valor x para um participante i, visto que usuários maliciosos poderiam não seguir o protocolo, votando contra o bloco candidato, ou o bloco candidato possui pagamentos inválidos. Desde que os participantes do protocolo sejam honestos, incluindo o líder, então $v_i = x$ e $g_i = 2$ para todo participante i honesto. Caso um participante honesto i receba a *string* x de um número de participantes menor que o valor t, i dá como saída valor $\perp$ e nota 0 e o protocolo não gera um bloco naquela rodada (o símbolo $\perp$ denota vazio).

3.4.2 Protocolo BBA^*_m

Em linhas gerais, um protocolo de acordo bizantino binário permite seus participantes decidirem por um valor de um conjunto $V = \{0,1\}$, sem contar o valor especial $\perp$, em uma execução com até t usuários maliciosos, e onde cada participante i começa com um valor inicial $v_i \in V$ e cada participante honesto j dá como saída $out_i \in V \cup \{\perp\}$ para satisfazer as seguintes condições (MICALI (2016)):

Acordo Existe $out \in V \cup \{\perp\}$ tal que $out_i = out$, para todo participante honesto i;

Consistência Se, para algum valor $v \in V$, $v_i = v$ para todos os participantes honestos, então $out = v$.

Note que out_i é o valor da saída de um participante i e out o valor da saída do protocolo.

BBA^*_m recebe como entrada a saída da execução do protocolo GC. BBA^*_m é bastante rápido, visto que consiste em todo participante executar o mesmo passo s até chegarem a um acordo. Também requer mínima preparação, apenas um valor aleatório de conhecimento de todos - a quantidade Q^{r-1} - e ele já pode ser executado. O passo consiste em cada jogador i propagar um valor b_i e $SIG_i(Q^{r-1}, s)$ para todos os jogadores, incluindo ele mesmo. Então, a sua saída é determinada da seguinte maneira:

1. Caso $\#_i^s(0) \geq 2t+1$ então i configura $b_i = 0$;
2. Caso $\#_i^s(1) \geq 2t+1$ então i configura $b_i = 1$;
3. Caso contrário, se S_i é o conjunto de todos os participantes j dos quais i recebeu $SIG_j(Q^{r-1}, s)$, i configura $b_i = lsb(min_{j \in S_i} H(SIG_j(s)))$.

Ao juntarmos os dois protocolos, temos o protocolo final do sistema, chamado de Algorand'. Algorand' consiste de $3+m$ passos. O primeiro passo diz respeito à propagação do bloco candidato pelo líder. Os dois passos seguintes são os do protocolo GC, descrito na subseção anterior. Um usuário i honesto configura $b_i = 0$, caso $g_i = 2$ e $b_i = 1$ caso contrário. Os próximos m passos dizem respeito ao protocolo BBA_m^*. O último passo requer que todos os participantes honestos assinem e propaguem o bloco escolhido. Além disso, devido à utilização de chaves efêmeras, em cada passo, todos os participantes devem destruir a chave privada efêmera daquele passo, e enviar uma mensagem que, em geral, consiste do valor de votação escolhido e da sua assinatura desse valor, a credencial, a chave pública efêmera, e uma autenticação dessa chave efêmera, utilizando a chave privada do sistema.

4

Análises

4.1 Análise de Algorand

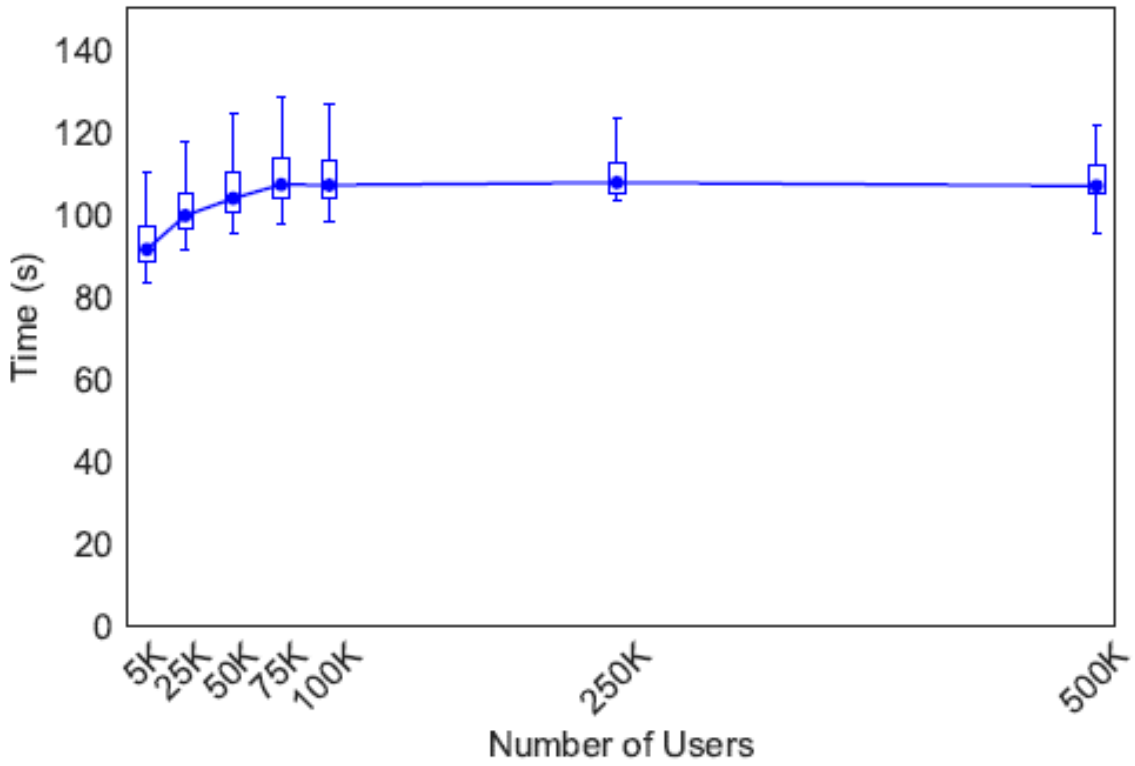
Como o próprio Micali escreveu em seu trabalho, Algorand assemelha-se mais à abordagem *Proof-of-Stake* que à *Proof-of-Work*. Como vimos, isso traz algumas vantagens a Algorand, referentes à velocidade com que pagamentos são confirmados, ao consumo de energia e poder computacional. Como nos mecanismos PoS não é necessário executar uma tarefa computacionalmente dispendiosa, o poder computacional requisitado é mínimo, e o consumo de energia baixo. Logo, usuários que desejem participar de Algorand podem fazê-lo utilizando uma máquina comum e sem se preocupar com gastos referentes a consumo de energia. Ppelo mesmo motivo, o tempo gasto com a geração do bloco é mínimo e a verdadeira latência acaba sendo a da propagação do bloco - quanto tempo leva para os nós da rede tomarem conhecimento de um novo bloco escolhido. Além disso, *forks* devem ocorrera apenas com probabilidade negligenciável. Isso mostra uma vantagem de Algorand sobre o mecanismo de consenso implementado por Bitcoin.

Entretanto, e se ocorrer um *fork* em Algorand? O artigo original de Micali não dá diretrizes sobre o que fazer em um caso como esse, porém, uma versão mais formal e detalhada, relaciona alguns procedimentos que os usuários podem seguir caso um *fork* ocorra (CHEN; MICALI (2016)):

1. Considerar o ramo mais longo - como em Bitcoin;
2. Caso não haja um ramo mais longo, considerar o ramo com menos blocos vazios na cabeça da lista;
3. Caso também haja um empate no número de blocos vazios ao final dos ramos, os usuários devem seguir o ramo cujo líder do último bloco tenha a menor credencial; caso ainda haja empate, então cujo hash do próprio bloco tenha o menor valor. Caso ainda haja empate, então os usuários devem seguir o ramo cujo bloco mais recente é o primeiro ordenado lexicograficamente.

Um outro trabalho, desta vez mais prático, conseguiu simular uma rede rodando Algorand (GILAD et al. (2017)). Utilizando SHA-256 como função hash, blocos de 1MB e até 50.000

usuários (1.000 máquinas virtuais com 50 usuários por máquina), em que 80% desses são honestos. Os resultados observados confirmam o que foi dito anteriormente: Algorand atinge uma latência de menos de 1 minuto com uma taxa 30 vezes maior de transações por segundo, comparado a Bitcoin. Algorand também mostrou escalar bem, com uma outra configuração de 500.000 usuários. Apesar de aumentar em 10 vezes o número de usuários, os resultados não sofreram alterações, mostrando que Algorand tem potencial para escalar mais (figura 4.1). Outras configurações simuladas incluem a rede com uma taxa de 35% de usuários offline (ficando mais rápida) e até 0.2% de usuários maliciosos (ficando mais lenta).



Fonte: GILAD et al. (2017)

Figura 4.1: Gráfico da latência de uma rodada de simulação de 5.000 até 500.000 usuários

Como qualquer máquina deve ser capaz de rodar Algorand, sem detrimento a outra, não há risco de ocorrer uma estratificação dos usuários como na rede de Bitcoin, onde encontramos uma divisão entre os mineradores - aplicam grande capital em máquina e consumo de energia para receber recompensas dentro do sistema - e usuários comuns - apenas recebem e realizam pagamentos - da moeda. Entretanto, Algorand pode acabar criando um outro tipo de estratificação. Em abordagens PoS é comum atrelar um peso à quantidade de moedas em posse de um usuário para a seleção dos participantes do mecanismo de consenso (NARAYANAN et al. (2016)). Desta maneira, os usuários mais ricos do sistema, acabam tendo mais chance de serem selecionados para gerar e/ou verificar os blocos que serão adicionados ao registro. Algorand não é diferente

pois também atribui um peso às moedas para selecionar o líder dos verificadores de uma rodada. Micali tenta reverter essa situação, colocando algumas limitações a Algorand. Essas limitações envolvem (MICALI (2016)):

1. Limitar cada chave pública a apenas uma unidade da moeda;
2. Limitar cada chave pública a um valor fixo;
3. Em combinação com uma dessas limitações, um usuário é visto como um conjunto de cópias de chaves públicas e cada uma delas pode ser selecionada individualmente.

Estas três sugestões vão da implementação mais simples (1) até a mais complexa (3). As duas primeiras limitam uma quantidade da moeda para uma chave pública, mas implicam que uma pessoa rica, em Algorand, é um usuário com posse de muitas chaves. A terceira dita uma maneira de garantir rotatividade dentro de um certo número de rodadas, assim como passos. Note que essa rotatividade é para garantir que um mesmo usuário não seja escolhido mais de uma vez por meio de chaves diferentes, dentro de um certo número de rodadas e passos. Essa rotatividade não garante que usuários com poucas moedas sejam escolhidos. Então, apesar de não estratificar sua base de usuários com relação à posse de dinheiro no mundo real, Algorand ainda confere um maior poder de decisão aos usuários com maior posse de moeda digital.

Entretanto, há uma lógica em conferir a usuários com mais dinheiro maior decisão de poder. Primeiro, precisamos explicar de onde vem o valor dado a uma criptomoeda como Bitcoin. Para estabelecer o valor de uma criptomoeda, é preciso alcançar 3 tipos de consenso (NARAYANAN et al. (2016)):

Consenso sobre as regras: Os usuários devem concordar em todos os conjuntos de regras da criptomoeda: o que torna uma transição válida, os protocolos e os formatos de dados envolvidos.

Consenso sobre história: Todos os usuários devem entrar em consenso sobre o que está e o que não está na *blockchain* - isto é, sobre quais transações ocorreram.

Consenso sobre o valor da moeda: Por fim, os usuários precisam concordar que as moedas tem valor. Particularmente, eles precisam acreditar que se alguém adquirir uma moeda hoje, esta pessoa tem que ser capaz de trocá-la, depois, por algo de valor.

Este último tipo de consenso depende, entre outros, do quão bem vista é uma moeda por usuários ativos e em potencial. Imagine que uma criptomoeda tem um histórico de falha de segurança, em que pagamentos inválidos e até prejudiciais a alguns usuários tenham sido efetivamente adicionados à *blockchain*. Esse tipo de acontecimento causa desconfiança na moeda, o que afasta usuários em potencial e provoca o abandono daqueles em atividade. Uma criptomoeda sem confiança e sem usuários não tem valor.

Agora, voltemos à lógica por trás de conceder aos usuários com mais posse de moeda um maior poder de decisão sobre a validação de transações. Esses usuários, por estarem de posse de muitas moedas, estão interessados em garantir o consenso de valor da moeda. Logo, espera-se que esses usuários sigam à risca o protocolo e validem as transações adequadamente, visto que são os usuários com mais a perder no caso de uma desvalorização da criptomoeda. Entretanto, um mecanismo de consenso ideal deve dar as mesmas oportunidades de participação na validação de pagamentos para todos os seus usuários. Principalmente, se este envolver um sistema de recompensa por participação. Em Algorand, Micali não especifica um valor ou modo de recompensar os participantes do mecanismo de consenso, mas sugere que todos os envolvidos na escolha do bloco, incluindo o líder ℓ^r e os membros do grupo SV^r devem ser recompensados. Isso, junto a uma abordagem que dá maior poder de decisão aos usuários em posse de mais moedas, pode acabar por centralizar a rede nesse grupo: os usuários mais ricos são escolhidos como verificadores com maior frequência, sendo recompensados mais vezes e ficando mais ricos.

4.2 Outras alternativas

Apesar de Algorand fazer avanços, em relação a Bitcoin, quanto à velocidade e ao consumo de energia, seu mecanismo de consenso mostra-se potencialmente centralizado em usuários com maior poder aquisitivo. Se o *Proof-of-Work* de Bitcoin acaba por fazer isso de maneira indireta, a abordagem de *Proof-of-Stake* Algorand faz o mesmo diretamente. Esse é um ponto muito sensível na comunidade das criptomoedas, visto que uma das principais motivações da criação de Bitcoin era justamente criar um sistema de transações descentralizado, sem a necessidade de um terceiro confiável para validar transações - como bancos e governos NARAYANAN et al. (2016). Por isso, Bitcoin opta por sacrificar escalabilidade para trazer um sistema descentralizado - apesar desse poder de decisão acabar caindo nas mãos daqueles com máquinas com maior poder de processamento. Algorand parece tentar resolver o problema da escalabilidade sacrificando a descentralização da rede. Vale ressaltar, porém, que Silvio Micali conseguiu propor um sistema menos centralizado que outras criptomoedas PoS, como Peercoin, por exemplo.

Vale notar que, propostas para solucionar esses problemas de Bitcoin não partem apenas de fontes externas da comunidade, mas também de seus próprios desenvolvedores. *Mining pool* é um mecanismo que permite a usuários com máquinas mais modestas serem recompensados pelo trabalho efetuado durante a mineração de um bloco, mesmo que eles não descubram o valor do bloco necessário (NARAYANAN et al. (2016)). Os usuários participantes de um *mining pool* trabalham em conjunto para minerar um mesmo bloco candidato - cada um trabalhando com valores de *nonce* diferentes. Caso algum usuário do *mining pool* consiga minerar o bloco, o valor da recompensa é distribuído entre todos os nós usuário, uniforme ou proporcionalmente de acordo com o quanto cada um contribuiu de trabalho (hash/segundos), dependendo da implementação de cada *mining pool* (NARAYANAN et al. (2016)).

Uma outra proposta dos desenvolvedores de Bitcoin, dessa vez para resolver o problema da escalabilidade, é o SegWit, mencionado na subseção 2.3.2. O *Segregated Witness* funciona tratando um outro problema de Bitcoin, o da maleabilidade de um bloco. Entre vários dos campos que formam um bloco, há dois em especial: o de entrada, contendo as chaves públicas do remetente da transação, e o de saída, contendo as chaves do destinatário da transação (NAKAMOTO (2008)). O problema é que o campo de entrada também possui a assinatura digital dos remetentes, utilizada para validar que essas transações estão sendo feitas por cada um dos remetentes (NAKAMOTO (2008)). Naturalmente, essas assinaturas precisam ser verificadas e essas verificações pesam muito na rede, atrasando a validação dos pagamentos (BitcoinCore (2016)). O que SegWit faz é separar o campo de entrada - contendo as assinaturas e chaves públicas - para um bloco próprio, chamado bloco estendido (BitcoinCore (2016)). O conjunto dessas assinaturas é conhecido como "testemunhas de transação" (BitcoinCore (2016)). Daí o nome "testemunha segregada". O bloco contendo as transações ganha espaço livre para conter mais pagamentos, o que deve aumentar a taxa de transações (BitcoinCore (2016)).

Aumentar a quantidade de transações por bloco é apenas um dos benefícios, visto que SegWit deve melhorar a escalabilidade de Bitcoin de outras maneiras além de corrigir alguns bugs e melhorar a segurança (BitcoinCore (2016)). SegWit foi implementado para funcionar como um *soft fork* para que, mesmo aqueles usuários que não desejem aderir à nova versão do cliente de Bitcoin, continuem enxergando blocos de SegWit como válidos. Porém, para que SegWit seja ativado, é preciso que 95% dos usuários de Bitcoin passem a utilizar o novo cliente. Além disso, uma abordagem mais invasiva, obrigando os usuários de Bitcoin a utilizar SegWit, poderia dividir a comunidade, causando um *hard fork* o que acabaria criando uma nova moeda.

Porém, nenhuma dessas soluções parece resolver um problema de grande debate em toda a discussão: o consumo de energia de Bitcoin. Muitos argumentam que o desperdício de energia é insustentável seja pela sua magnitude, ou porque seu consumo não é justificável. Já os entusiastas de Bitcoin e criptomoedas defendem que não há desperdício algum. Seu consumo é justificável para manter um sistema monetário mundial, descentralizado e seguro. Infelizmente, com as soluções para alcançar consenso em uma *public ledger* parece existir um compromisso entre eficiência e descentralização. Cabe aos desenvolvedores e usuários escolherem qual característica deve ser priorizada

5

Conclusões

Neste trabalho, estudamos como sistemas que implementam *public ledgers* chegam a um consenso sobre quais informações são válidas, em especial, Bitcoin. Vimos como sua rede funciona, como as transações acontecem nela e como a segurança delas é garantida. Vimos como os usuários validam essas transações, através de mecanismos de consenso e suas diferentes abordagens. Nesse contexto, identificamos os problemas que a abordagem utilizada por Bitcoin, *Proof-of-Work*, possui e estudamos uma nova alternativa a esta abordagem, Algorand.

Algorand é uma nova *public ledger*, proposta por Silvio Micali, com o intuito de ser mais escalável que Bitcoin. O novo mecanismo de consenso, baseado em acordo bizantino e concebido também por Micali, é mais rápido, e por isso capaz de processar mais pagamentos em menos tempo, além de consumir menos energia. Entretanto, apesar de ser uma solução menos custosa com respeito ao consumo de energia e equipamento, Algorand põe o poder de decisão nos usuários em posse de mais moedas do sistema.

Algorand porém, é apenas um dos vários esforços de escalar Bitcoin, e criptomoedas em geral. Uma outra criptomoeda que busca alternativas para escalar sua rede é Ethereum. Originalmente, Ethereum implementava um mecanismo de consenso semelhante ao de Nakamoto, mas recentemente trocou por uma abordagem híbrida de *Proof-of-Stake*. Os desenvolvedores de Bitcoin também reconhecem os problemas em sua *blockchain*, e tentam corrigi-los com novas propostas, como o SegWit, uma nova adição ao mecanismo de consenso que promete deixar a rede mais rápida e até mais segura. Porém, qualquer modificação deve ser feita sem sacrificar a principal característica dessas criptomoedas: descentralização, a capacidade da rede de tomar decisões quanto ao que deve ser adicionado a sua *blockchain* sem a necessidade - ou interferência - de um terceiro confiável.

BitcoinCore. **Segregated Witness Benefits**. Acessado em 10/06/2017, https://bitcoincore.org/en/2016/01/26/segwit-benefits/.

CHEN, J.; MICALI, S. Algorand. **CoRR**, [S.l.], 2016. Disponível em https://arxiv.org/abs/1607.01341.

CoinDesk. **Litecoin Successfully Activates SegWit**. Acessado em 10/06/2017, http://www.coindesk.com/litecoin-successfully-activates-long-debated-segwit-upgrade/.

Ethereum Project. **Ethereum Project**. Acessado em 10/06/2017, https://www.ethereum.org/.

Ethereum Project. **Proof-of-stake FAQ - Ethereum Wiki**. Acessado em 10/06/2017, https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ.

FELDMAN, P.; MICALI, S. An Optimal Probabilistic Algorithm for Byzantine Agreement. **SIAM Journal on Computing**, [S.l.], 1997.

GILAD, Y. et al. Algorand: scaling byzantine agreements for cryptocurrencies. **IACR Cryptology ePrint Archive 2017: 454**, [S.l.], 2017. Disponível em http://www.mit.edu/~yossigi/Algorand.pdf.

MICALI, S. ALGORAND, The Efficient Public Ledger. **CoRR**, [S.l.], 2016. Disponível em https://arxiv.org/abs/1607.01341v1.

NAKAMOTO, S. **Bitcoin**: a peer-to-peer electronic cash system. Disponível em https://bitcoin.org/bitcoin.pdf.

NARAYANAN, A. et al. **Bitcoin and Cryptocurrency Technologies**: a comprehensive introduction. [S.l.]: Princeton University Press, 2016.

NIST. **FIPS PUB 180-4, Secure Hash Standard**. Gaithersburg, MD: National Institute of Standards and Technology, 2015.

SMART, N. P. **Cryptography**: an introduction. [S.l.]: Mcgraw-Hill College, 2004.

Apêndice

A

Termos e criptomoedas mencionadas

Tabela A.1: Lista de termos utilizados no trabalho.

Termo	Definição
Acordo bizantino	Protocolo que permite a um sistema com n participantes chegarem a um acordo sobre um determinado valor, na presença de até t participantes maliciosos, isto é, membros que não seguem as regras do protocolo
Blockchain	Lista encadeada de blocos de dados que implementam ponteiros hash. Frequentemente utilizadas para assegurar transações, são a principal estrutura de dados de uma criptmoeda. Sinônimo de *public ledger*
Criptomoeda	Tipo de moeda digital, é um meio de troca que implementa técnicas criptográficas para assegurar transações
Hard fork	Bifurcação na *blockchain* em que grupos distintos de usuários se dividem sobre qual ramo é válido
Soft fork	Bifurcação na *blockchain* em que um grupo dos usuários passa a enxergar um dos ramos como inválido
Mecanismo de consenso	Método pelo qual uma criptomoeda decide quais transações devem ser adicionadas à sua *public ledger*
Proof-of-Work	Mecanismo de consenso em que usuários precisam resolver algum quebra-cabeça criptográfico para validar transações
Proof-of-Stake	Mecanismo de consenso em que usuários são escolhidos para validar transações, de acordo com a sua riqueza

Tabela A.2: Lista de criptomoedas mencionadas no trabalho, assim como os mecanismos de consenso utilizados e o seus símbolos.

Moeda	Símbolo	Mecanismo de consenso
Bitcoin	BTC	PoW
Dogecoin	DOGE, XDG	PoW
Ethereum	ETH	PoW e PoS
Litecoin	LTC	PoW
Namecoin	NMC	PoW
Peercoin	PPC	PoS e PoW