UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Verificação formal de contratos inteligentes na Ethereum

Gustavo Oliveira Dias

Qualificação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP
SÃO CARLOS

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Gustavo Oliveira Dias

Verificação formal de contratos inteligentes na Ethereum

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, para o Exame de Qualificação, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional.

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Adenilso da Silva Simão

USP – São Carlos
Março de 2021

Gustavo Oliveira Dias

Formal verification on Ethereum smart contracts

Monograph submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – as part of the qualifying exam requisites of the Computer and Mathematical Sciences Graduate Program, for the degree of Master in Science.

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Adenilso da Silva Simão

USP – São Carlos
March 2021

RESUMO

DIAS, O. D. **Verificação formal de contratos inteligentes na Ethereum**. 2021. 57 p. Monografia (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2021.

A tecnologia blockchain, promovida pela criptomoeda Bitcoin, ficou conhecida pela sua capacidade de realizar o gerenciamento de posse descentralizado de criptomoedas por meio de um livro-razão distribuído. Com a introdução da Ethereum, foi possível a implantação de contratos inteligentes, que são programas de computador que expressam cláusulas, condições e acordos estabelecidos entre as partes envolvidas. Uma vez implementados, os contratos inteligentes executam de forma autônoma e descentralizada por meio da Máquina Virtual Ethereum, sem a necessidade de intermediação. Por meios dos contratos inteligentes expandiu-se as áreas de aplicação da blockchain, que passou a abranger também aplicações descentralizadas, Organizações Autônomas Descentralizadas, governança, finanças, cuidados médicos, entre outras áreas. Solidity é a linguagem de programação desenvolvida para construção de contratos inteligentes para execução na Ethereum. Contudo, vulnerabilidades encontradas no código dos contratos ocasionaram diversas perdas financeiras. Devido à imutabilidade da blockchain, uma vez implantado, o contrato não pode ser alterado, o que aumentou a preocupação com o desenvolvimento de contratos livres de erros. Este trabalho tem como objetivo propor uma abordagem para verificação formal de contratos inteligentes. Para isso, as seguintes etapas devem ser concretizadas: (i) escolher quais vulnerabilidades serão atacadas; (ii) selecionar o tipo de abordagem relacionada à verificação formal que será utilizada; (iii) Realizar uma análise acerca da viabilidade da abordagem escolhida; (iv) definir as estratégias de implementação e experimentação para avaliação da proposta.

Palavras-chave: Blockchain, Livro-razão distribuído, Contrato inteligente, Verificação formal, Vulnerabilidades.

ABSTRACT

DIAS, O. D. **Formal verification on Ethereum smart contracts**. 2021. 57 p. Monografia (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2021.

Blockchain technology, pioneered by Bitcoin cryptocurrency, became known for its ability to perform decentralized ownership management of cryptocurrencies through a Distributed Ledger Technology. By the release of Ethereum, it was possible to implement smart contracts, which are computer programs that express clauses, conditions and agreements established among the involved parties. Once implemented, smart contracts execute in an autonomous and decentralized way through the Ethereum Virtual Machine, without the need for intermediation. Through smart contracts, the application areas of blockchain were expanded, which covers decentralized applications, Autonomous Decentralized Organizations, governance, finance, healthcare, among other areas. Solidity is the programming language developed for building smart contracts for execution at Ethereum. However, vulnerabilities found in the contract code caused several financial losses. Due to blockchain immutability, once deployed the contract cannot be changed, which increases the concern with the development of error-free contracts. This work aims to propose an approach for formal verification of smart contracts. To achieve this, the following steps must be performed: (i) choose which vulnerabilities will be attacked; (ii) select the type of approach related to formal verification that will be used; (iii) carry out a feasibility analysis of the chosen approach; (iv) define the implementation and experimentation strategies for the proposal evaluation.

Keywords: Blockchain, Distributed Ledger Technology, Smart contract, Formal verification, Vulnerabilities.

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Motivação 12
1.2 Objetivos gerais e específicos 13
1.3 Organização do trabalho 14

2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 Sistemas distribuídos e a tecnologia Blockchain 15
2.1.1 Tipos de redes blockchain *20*
2.1.2 Processo de validação na Blockchain *21*
2.1.3 Escolha do histórico de transações *24*
2.1.4 Criptografia e autorização de transações *25*
2.1.5 Bitcoin como um sistema de transição de estados *27*
2.2 Blockchain Ethereum 27
2.2.1 Contas Ethereum *29*
2.2.2 Transações e mensagens *30*
2.2.3 Função de transição de estados *31*
2.2.4 Blocos *32*
2.2.5 Validação *34*
2.2.6 Aplicações *35*
2.2.6.1 Sistemas de tokens *36*
2.2.6.2 Organizações Autônomas Descentralizadas *37*
2.2.6.3 Cuidados médicos e serviços de saúde *38*
2.2.7 Arquitetura em camadas *38*
2.3 Contratos inteligentes 39
2.3.1 Vulnerabilidades e ataques *42*

3 REVISÃO BIBLIOGRÁFICA 45

4 TRABALHOS RELACIONADOS 47

5 MÉTODO 49

6 CONSIDERAÇÕES FINAIS 51

REFERÊNCIAS 53

CAPÍTULO

1

INTRODUÇÃO

Blockchain é o nome dado à tecnologia subjacente utilizada em diversas plataformas de gerenciamento descentralizado de posse de bens digitais baseada em livro-razão distribuído (do inglês, *Distributed Ledger Technology* (DLT)) (KANNENGIESSER *et al.*, 2020). Essa tecnologia tem como principais características o armazenamento descentralizado e distribuído, a imutabilidade, a transparência e a dispensa da necessidade de confiança em uma terceira parte (FAN *et al.*, 2020; Dinh *et al.*, 2018). O primeiro caso de êxito na aplicação da blockchain foi proposto por Nakamoto (2008), que apresentou a Bitcoin, uma criptomoeda gerada e gerenciada de forma distribuída e sem entidades centralizadoras (ZHANG; XUE; LIU, 2019). A geração e o gerenciamento de posse de unidades de Bitcoin são realizados por uma rede de nós conectados auto-gerenciáveis que trabalham para manter a integridade do sistema (Dinh *et al.*, 2018).

A geração e gerenciamento de posse de criptomoedas é uma entre diversas aplicações possíveis baseadas na DLT, ou seja, é apenas um fim para um meio (DRESCHER, 2017). Desde o seu surgimento, a blockchain tem passado por diversas transformações, o que possibilitou sua aplicação em diversas áreas do conhecimento, como finanças, governo, Internet das Coisas (do inglês, Internet of Things (IoT)), inteligência artificial, saúde, entre outras (SWAN, 2015; MAESA; MORI, 2020; ZHU *et al.*, 2019; Salah *et al.*, 2019; AGUIAR *et al.*, 2020).

Um fator crucial para impulsionar o avanço das DLTs foi a introdução dos contratos inteligentes (CI) (MAESA; MORI, 2020). A plataforma baseada em DLT, Ethereum, proposta por Buterin (2014), possibilitou a execução de CIs de forma descentralizada em uma rede ponto-a-ponto (do inglês, *peer-to-peer* (P2P)). Um contrato inteligente consiste em um conjunto de cláusulas e condições, que são definidas entre as partes envolvidas e expressas por meio de uma linguagem de programação (ZHENG *et al.*, 2020). Depois de escrito, o contrato é implantado em uma blockchain e executado de forma autônoma, automática, e imutável (ZHENG *et al.*, 2020; KANNENGIESSER *et al.*, 2020).

Aplicações que executam sobre a plataforma Ethereum geralmente envolvem movimenta-

ções de grandes quantias de sua criptomoeda nativa, o Ether. Assim, essas aplicações tornaram-se alvos de diversos ataques que causaram transtornos e graves perdas financeiras (ATZEI; BARTOLETTI; CIMOLI, 2017; CHEN *et al.*, 2020a). Um dos ataques mais conhecidos aconteceu em 2016 contra o The DAO (sigla para *Decentralized Autonomous Organization*), um projeto de *crowdfunding* que arrecadou cerca de 150 milhões de dólares (CHEN *et al.*, 2020a). Neste ataque, um contrato malicioso explorou uma falha no código e transferiu cerca de 3,6 milhões de Ether para sua conta, o equivalente a 50 milhões de dólares (CHEN *et al.*, 2020a; SIEGEL, 2020; ATZEI; BARTOLETTI; CIMOLI, 2017).

Grande parte dos ataques deve-se à exploração de vulnerabilidades encontradas nos CIs (CHEN *et al.*, 2020a; ATZEI; BARTOLETTI; CIMOLI, 2017; LIU; LIU, 2019). Devido à imutabilidade da blockchain, uma vez implantados, os CIs não podem ser alterados, e, portanto, não há como corrigir possíveis erros e vulnerabilidades contidos no código, o que ressalta a necessidade de identificá-las na fase de pré-implantação (VACCA *et al.*, 2020; DIKA; NOWOSTAWSKI, 2018). Os CIs são geralmente escritos em Solidity [1], uma linguagem de programação de alto nível, Turing-completa, e desenvolvida especialmente para escrever CIs para a plataforma Ethereum (VARELA-VACA; QUINTERO, 2021). Segundo Atzei, Bartoletti e Cimoli (2017) parte desses erros são ocasionados pelo desalinhamento que há entre a semântica da linguagem Solidity e a intuição dos desenvolvedores.

1.1 Motivação

Motivado por riscos à segurança das aplicações baseadas em CIs, diversas estratégias foram utilizadas no intuito de mitigar os riscos envolvidos, como exposto nos trabalhos de Liu e Liu (2019), Chen *et al.* (2020a), Sayeed, Marco-Gisbert e Caira (2020) e Singh *et al.* (2020). Segundo Dika e Nowostawski (2018), a forma mais efetiva para identificação de vulnerabilidades antes da implementação dos CIs é por meio da contratação de serviços de auditoria. Porém, tais serviços podem ser muito custosos para pequenas empresas e desenvolvedores individuais (DIKA; NOWOSTAWSKI, 2018). No estudo de Chen *et al.* (2020a), ressalta-se que as duas melhores formas de prevenir-se de vulnerabilidades consistem na escrita de contratos livres de erros por meio de boas práticas de programação, e, em seguida, na utilização de analisadores ou verificadores de código.

Na pesquisa de Almakhour *et al.* (2020), as abordagens para verificação de CIs são separadas em dois aspectos: (i) verificação formal para correção; (ii) e detecção de vulnerabilidades para garantia de segurança. A verificação formal para correção consiste na representação formal do programa por meio de métodos matemáticos, denominado o processo de modelagem do programa. Uma vez modelado, propriedades que representam a ocorrência de vulnerabilidades ou de erros lógicos são definidas, e então um processo de verificação é executado em busca

[1] Solidity documentation. <https://docs.soliditylang.org/en/develop/index.html>

de possíveis violações das propriedades (ALMAKHOUR *et al.*, 2020; SINGH *et al.*, 2020). A detecção de vulnerabilidades para garantia de segurança baseia-se na definição de padrões de vulnerabilidades conhecidas para que, por meio de ferramentas, se execute uma análise sobre o código para então detectá-las (ALMAKHOUR *et al.*, 2020).

Tanto a garantia de segurança de CIs quanto a própria tecnologia blockchain representam áreas de pesquisa relativamente novas e emergentes (CHEN *et al.*, 2020a; KANNENGIESSER *et al.*, 2020). Portanto, ainda não há uma abordagem ou ferramenta padronizada para garantia de segurança dos CIs. Além disso, também há limitações nas abordagens existentes. As técnicas de verificação formal costumam exigir conhecimento especializado para modelagem matemática dos contratos, além de limitações de tempo e memória (CHEN *et al.*, 2020a). Já as ferramentas para análise de código apresentam taxas consideráveis de falsos positivos e falsos negativos, e mostraram-se ineficientes para contratos complexos (KIM; LEE, 2020).

Este trabalho é motivado pelos problemas relacionados a exploração de vulnerabilidades em CIs escritos em Solidity na blockchain Ethereum, assim como pelas limitações presentes na abordagens existentes para mitigação destes problemas. Esta pesquisa tem o propósito de explorar este tema, tendo como base a seguinte questão de pesquisa:

"Como detectar vulnerabilidades em contratos inteligentes escritos na linguagem Solidity na fase de pré-implementação?"

1.2 Objetivos gerais e específicos

Guiado pela questão de pesquisa, este trabalho tem como objetivo propor uma abordagem para verificação formal de CIs escritos na linguagem Solidity por meio da técnicas de *model checking* e análise de código. Para complementar esse objetivo principal, há os seguintes objetivos específicos:

- Escolher o método adequado para modelagem do contratos e para representação das vulnerabilidades;
- Definir as vulnerabilidades que devem ser detectadas;
- Determinar a técnica de análise de código utilizada para complementar o processo de verificação;
- Implementar o método de verificação;
- Definir as estratégias para experimentação e avaliação da proposta.

1.3 Organização do trabalho

Este documento foi organizado da seguinte forma. No Capítulo 2 é apresentado o referencial teórico e os conceitos empregados neste trabalho. O Capítulo 3 expõe como a pesquisa foi conduzida e quais foram os métodos aplicados para levantar os trabalhos e responder às questões de pesquisa. O Capítulo 4 retrata os trabalhos da literatura cujos os temas estão associados à abordagem desta pesquisa. O Capítulo 5 retrata os detalhes da abordagem proposta.

CAPÍTULO

2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos relacionados com a tecnologia blockchain e a forma como cada um deles contribui para a manutenção de suas propriedades. Apesar de existirem diversas variações entre as blockchains, este capítulo tem como foco as plataformas Bitcoin e Ethereum, a primeira por ter sido a pioneira da tecnologia blockchain e a mais conhecida até os dias atuais, e a última por estar diretamente relacionada com este trabalho. Além de expor os aspectos estruturais e funcionais da Ethereum, este capítulo também descreve sobre contratos inteligentes e vulnerabilidades existentes, fornecendo um conjunto de conceitos e informações necessárias para o entendimento da proposta deste trabalho.

Na Seção 2.1 são introduzidos os fundamentos da tecnologia blockchain. Na Seção 2.2 são abordados os conceitos e as particularidades inerentes à blockchain Ethereum. Na Seção 2.3 é discutido sobre o ciclo de execução dos contratos inteligentes, vulnerabilidades presentes em contratos inteligentes, e ataques ocorridos que exploraram essas vulnerabilidades.

2.1 Sistemas distribuídos e a tecnologia Blockchain

Ao implementar um sistema de *software*, uma decisão fundamental a ser tomada diz respeito à sua arquitetura, isto é, o modo como seus componentes são organizados e se relacionam. As principais abordagens são as arquiteturas distribuída e centralizada (TANEMBAUM; STEEN, 2007).

Em uma arquitetura centralizada, os componentes do sistema são dispostos em torno de um componente central e estão conectados a eles. Esses componentes centrais podem ser servidores, bancos de dados, centrais de controle, centrais de processamento ou qualquer elemento que concentre serviços ou recursos computacionais. Em contraste, em sistemas distribuídos não há nenhum elemento central para coordenação ou controle, de modo que seus elementos formam uma rede de componentes conectados que, de alguma forma, trabalham para manter o

funcionamento do sistema (TANEMBAUM; STEEN, 2007).

Quando os componentes de um sistema estão distribuídos entre vários computadores conectados, os recursos desses computadores são agregados ao sistema como um todo e disponibilizados diretamente a outros nós. Assim, os participantes do sistema atuam tanto como fornecedores quanto consumidores de recursos. Esse tipo de sistema distribuído é conhecido como sistema ponto a ponto (TANEMBAUM; STEEN, 2007).

Em sistemas centralizados, recursos como processamento e armazenamento são delegados a apenas um computador central ou um conjunto específico de computadores. Podem haver também abordagens híbridas, nas quais os recursos computacionais são usados de forma distribuídas, mas alguns serviços, como gerenciamento de comunicação, geração de relatórios e tarefas de coordenação, por exemplo, são desempenhados por algum componente específico. Quando um sistema ponto a ponto não possui nenhum elemento central para controle ou coordenação, então este é considerado também como um sistema ponto a ponto puramente distribuído (TANEMBAUM; STEEN, 2007).

Em sistemas distribuídos, a integração de recursos e serviços entre os componentes envolvidos, também chamados de nós do sistema, agrega diversos benefícios. Nesses sistemas, a capacidade total de processamento resulta da combinação da capacidade de processamento de todos os nós conectados. Assim, o poder computacional do sistema pode expandir-se se acordo com a demanda, bastando apenas conectar mais computadores ao sistema, o que agrega ao sistema a capacidade de expandir-se naturalmente. Desta forma, é comum que sistemas distribuídos tenham um poder de processamento maior do que um supercomputador individual. Com vários nós integrando o sistema, simplifica-se também a manutenção, pois substituir ou desativar computadores individuais de um sistema distribuído pode ser feito sem um impacto significativo no sistema como um todo. Este último fator eleva a confiabilidade no funcionamento do sistema, que pode continuar operando normalmente mesmo que máquinas individuais falhem (TANEMBAUM; STEEN, 2007; DRESCHER, 2017).

Apesar dos benefícios, em uma arquitetura distribuída é necessário lidar com uma série de questões. Sem entidades centrais para coordenar seus membros, a coordenação em um sistema distribuído deve ser feita pelos próprios nós participantes do sistema. Para que essa coordenação possa ser desempenhada é necessário um protocolo de comunicação, e lidar com os custos de processamento envolvidos no envio, recepção e processamento de mensagens. Essa necessidade constante de comunicação resulta em uma dependência das redes, que têm seus próprios desafios e adversidades. Como a comunicação em rede implica no envio e compartilhamento de dados críticos, entidades não confiáveis podem utilizar indevidamente a rede a fim de acessar e explorar informações, o que compromete a integridade do sistema (TANEMBAUM; STEEN, 2007; DRESCHER, 2017).

Resolver um problema de computação e prover serviços para usuários de computadores envolve a escrita de programas e *softwares*. Quando se trata de sistemas distribuídos, esse

processo implica em lidar com problemas de coordenação, comunicação, utilização de redes e segurança (DRESCHER, 2017). A forma como se lida com esses problemas impacta diretamente na integridade de um sistema.

A integridade é um aspecto esperado em qualquer sistema de *software*, e engloba três componentes: integridade de dados; integridade comportamental; e segurança. A integridade de dados assegura que os dados usados e mantidos pelo sistema são completos, corretos e livres de contradições. A integridade comportamental garante que o sistema executa suas funções e rotinas de acordo com o esperado, sem falhas ou erros de lógica. A segurança atesta que o acesso às funcionalidades, dados dos usuários e dados internos do sistema estão restritos somente aos usuários autorizados (TANEMBAUM; STEEN, 2007).

A relação entre sistemas ponto a ponto puramente distribuídos e a tecnologia blockchain está em seu uso para prover e manter a integridade desses sistemas (DRESCHER, 2017). No trabalho pioneiro de Nakamoto (2008), foi proposto uma solução para manter a integridade em uma versão ponto a ponto e puramente distribuída de um dinheiro eletrônico (ou criptomoeda) para pagamentos *online* diretamente entre as duas partes envolvidas na transação, sem a necessidade de uma terceira parte intermediadora, como uma instituição financeira. A solução foi proposta com o objetivo de evitar o problema do gasto duplo, quando uma das partes envolvidas na transação executa a mesma transferência duas vezes para obter benefícios indevidos.

Para que os participantes desse sistema de pagamentos confiem na integridade do sistema, é preciso acessar o histórico das transações realizadas. Desta forma, o próprio participante, chamado também de nó participante, pode conferir a integridade das transações. Contudo, é necessário também que todos os nós tenham acesso ao mesmo histórico de transações.

Em empresas, utiliza-se um livro-razão para lançamento de registros contábeis para elaboração de relatórios financeiros (MARION, 1985). Na estrutura de dados de uma blockchain, este livro-razão fica disposto por meio uma cadeia de blocos interligados (i.e., uma lista encadeada) que armazenam todo o histórico de transações. Cada nó da rede é responsável por manter uma versão atualizada do histórico de transações, assim como preservar a integridade dessas informações.

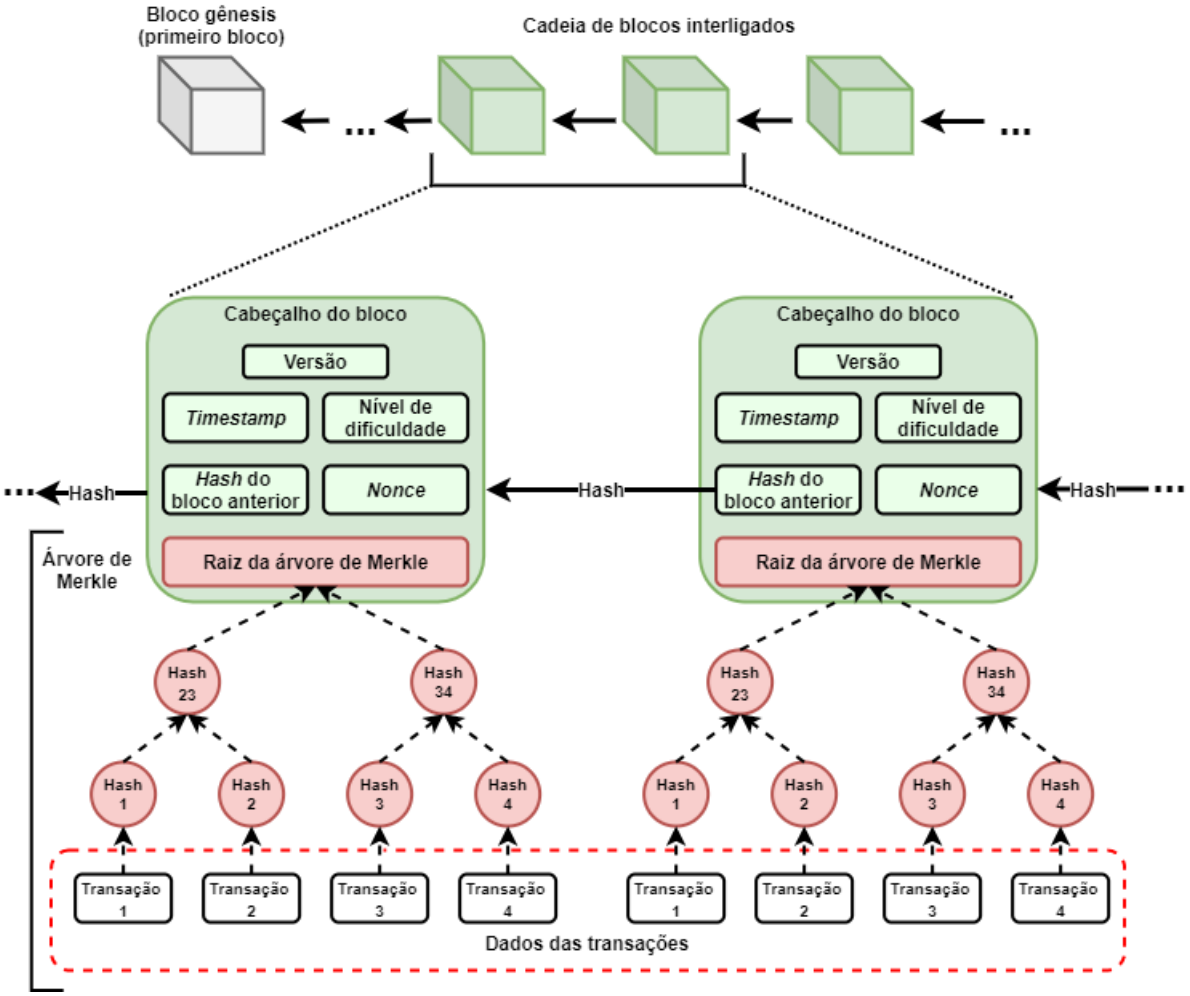
Para garantir a disponibilidade do mesmo histórico de transações para todos os nós. Sempre que uma nova transação ocorre, as informações sobre essa transação, como as contas envolvidas, a quantia transferida e o horário da transação, são transmitidas entre todos os nós da rede. Os nós participantes da rede, conhecidos também como mineradores, coletam uma determinada quantidade de transações e as englobam em um componente chamado de bloco.

As transações englobadas em um bloco são organizadas na forma de uma árvore de Merkle. Nessa estrutura de dados do tipo árvore cada transação é alocada em um nó folha, e os demais nós armazenam referências do código *hash* gerado a partir dessas transações. Um código *hash* é gerado por meio de algum algoritmo criptográfico de geração de código *hash*,

como o SHA-3 (DWORKIN, 2015) e o Keccak256 (BERTONI *et al.*, 2011). Esses algoritmos recebem e processam dados de entrada, e então geram um código formado por uma sequência de caracteres e dígitos, de forma que, para duas ou mais entradas distintas, a chance de um código *hash* igual ser gerado é extremamente baixa. Outro ponto importante é que, apenas com a posse de um código gerado, não há como se obter os dados de entrada do qual o código se originou.

Na Figura 1 é ilustrada a estrutura da blockchain como um conjunto de blocos interligados em que, a partir dos blocos, as informações das transações podem ser acessadas por meio da raiz da árvore de Merkle.

Figura 1 – Estrutura da blockchain



Fonte: Aguiar *et al.* (2020), Dinh *et al.* (2018).

Para cada bloco é criado um cabeçalho, que passa a integrar o bloco. As informações presentes no cabeçalho podem variar de acordo com a rede blockchain. Na rede Bitcoin (NAKAMOTO, 2008), um cabeçalho é formado por cinco elementos:

- **Versão:** Número da versão do protocolo de regras de validação a ser seguido;

- ***Hash* do bloco anterior:** Isto é, o código *hash* obtido a partir dos dados do cabeçalho do último bloco presente na blockchain no momento da construção do próximo bloco;
- **Raiz da árvore de Merkle:** Em um bloco, apenas a raiz da árvore de Merkle é armazenada. Desta forma, qualquer tentativa de adulterar ou adicionar indevidamente uma transação na árvore referenciada pelo bloco irá invalidar a referência de *hash* da raiz, alterando também o valor do *hash* do bloco;
- ***Timestamp*:** Horário atual referente ao momento em que o bloco está sendo criado. Esta informação é essencial para manter a ordenação dos blocos no histórico de transações mantido por cada nó da rede;
- **Nível de dificuldade:** Número que indica o nível de dificuldade do quebra-cabeça computacional que deve ser resolvido pelos mineradores na disputa pela criação do próximo bloco. Este item influencia diretamente no tempo e esforço computacional necessário para a criação de um bloco;
- ***nonce*:** Quando o *nonce* é acrescendo ao cabeçalho do bloco, o código *hash* obtido a partir do cabeçalho deve ser iniciado por uma quantidade predefinida de zeros, que é indicada pelo nível de dificuldade.

Ao se criar um bloco, o minerador forma primeiro um bloco preliminar contendo os dados dos itens 1 ao 5. Para se obter o *nonce* é necessário realizar uma quantidade massiva de tentativas com o intuito de encontrar a sequência de caracteres e dígitos que satisfaça o nível de dificuldade predefinido. Essa tarefa é conhecidas como mineração, e gera uma disputa entre os nós da rede pela criação do próximo bloco. Nessa disputa, aqueles que possuem computadores mais robustos e com maior capacidade de processamento têm maiores chances de ganhar. A descoberta do *nonce* também é referida neste trabalho como um quebra-cabeça computacional ou quebra-cabeça de *hash*.

O *nonce* é um item necessário em blockchains nas quais, assim como na *Bitcoin*, utilizam um algoritmo de consenso com regras para criação e validação de blocos conhecido como *proof-of-work* (NAKAMOTO, 2008).

Assim que o *nonce* é adicionado ao bloco, este é então transmitido pela rede para que todos os nós possam acessá-lo e participar do processo de validação do bloco. Caso o bloco seja aceito no processo de validação, então cada nó adiciona o bloco válido à própria cópia da estrutura de dados blockchain e o minerador é recompensado pelo gasto que teve (NAKAMOTO, 2008).

Como o *hash* gerado nas ramificações da árvore de Merkle depende diretamente do conteúdo das transações, qualquer alteração em uma transação invalida as referências de *hash* dos nós da ramificação da qual a transação pertence, inclusive o nó raiz. Com uma modificação no valor de *hash* da árvore de Merkle, o valor de *hash* do bloco também é alterado. Modificar

o valor de *hash* do bloco invalida a referência de *hash* que aponta para o cabeçalho do bloco modificado, invalidando, assim, toda a estrutura de dados.

Desta forma, tentar fraudar dados de transação manipulados envolve uma série de operações custosas. Primeiro deve-se reescrever a árvore de Merkle à qual a transação manipulada pertence. Após isso, é necessário reescrever o cabeçalho do bloco a qual a raiz da árvore de Merkle reescrita pertence, o que requer a solução do quebra-cabeça de *hash* para obtenção de um novo *nonce*. Consequentemente, todos os cabeçalhos até o final da estrutura de dados da blockchain precisam ser reescritos, o que inclui encontrar o *nonce* de cada um. Este processo é propositalmente complexo e se faz necessário para manter os dados consistentes e íntegros. Isso atribui à tecnologia blockchain a propriedade de imutabilidade.

2.1.1 Tipos de redes blockchain

O funcionamento de uma blockchain varia de acordo com a forma como se lida com questões como imutabilidade, permissões, velocidade de processamento de transações e gerenciamento de participantes no processo de validação. Por isso, existem diferentes variações em blockchains de acordo com as necessidades das aplicações. Baseada nas características descritas por Ahmed *et al.* (2019) e Sankar, Sindhu e Sethumadhavan (2017), esses tipos de blockchains podem ser classificados com segue:

- **Blockchain pública:** A participação no processo de criação e validação de blocos, assim como o acesso ao histórico de transações, é aberto à todos os nós da rede. Todos os nós são livres para se juntar ou deixar a rede como desejarem. Os exemplos mais conhecidos de blockchain pública são a Bitcoin e a Ethereum, em que os mineradores validam as transações e recebem criptomoedas como recompensa pelo esforço em resolver o quebra-cabeça de *hash*;
- **Blockchain privada:** Não é publicamente acessível e possui uma estrutura centralizada. Regras para restrição de acesso e criação de blocos são estabelecidas, controladas e fiscalizadas por uma entidade central. Blockchains privadas são projetadas especialmente para empresas e companhias, como, por exemplo, a blockchain da Ripple (SCHWARTZ *et al.*, 2014).
- **Blockchain de consórcio:** Nem todos os nós participantes possuem os mesmo direitos de validação de transações e participação no processo de consenso. Apenas o conjunto de nós servidores previamente selecionados podem controlar uma blockchain de consórcio e validar blocos, como acontece nas blockchains Hyperledger Fabric (ANDROULAKI *et al.*, 2018) e Corda (BROWN *et al.*, 2016).

Devido à sua natureza aberta para participação, uma blockchain pública é definida também como uma blockchain não permissiva. Já as blockchains privadas e de consórcio

concedem direitos de acesso aos dados e validação de transações à nós específicos, e são, portanto, classificadas como blockchains permissivas. Há também casos como o da Ethereum, que, mesmo sendo uma blockchain utilizada como pública, também é possível configurar ambientes privados com controle de acesso. Maiores detalhes sobre a rede Ethereum são tratados na Seção 2.2.

2.1.2 Processo de validação na Blockchain

Um fator fundamental no êxito da Blockchain foi sua capacidade de garantir integridade e confiança em um ambiente de sistemas ponto a ponto puramente distribuídos, onde há um número ilimitado de nós conectados sem nenhum nível de confiança pré-estabelecidos entre estes. Em uma rede de blocos com informações que podem ser produzidas por qualquer nó conectado, há o risco iminente de inserção de informações falsas e maliciosas. Para garantir a confiança de que os blocos na blockchain são legítimos, é necessário verificar a validade de um novo bloco antes deste ser inserido na rede.

Por meio de um protocolo previamente estabelecido, os nós são responsáveis por chegar a um consenso para validar cada inserção, seguindo a ordem na qual as transações ocorrem. Para incentivar os nós a manterem a integridade das transações, são definidos mecanismos de incentivo, assim como formas de punição para os nós que tentam inserir ou validar transações maliciosas. Em uma blockchain, as regras que regem esse protocolo são definidas por um algoritmo de consenso.

Neste trabalho, o termo "protocolo de consenso" é usado para se referir de forma generalizada ao processo de tomada de decisão coletiva entre os nós para validação de novos blocos, um procedimento pertinente em qualquer rede blockchain. Contudo, em cada rede blockchain, esse protocolo pode ser composto por regras distintas. Cada conjunto específico de regras para estabelecimento de um protocolo de consenso é referido neste trabalho como um algoritmo de consenso, que pode apresentar diversas variações.

Um algoritmo de consenso é elaborado com o objetivo de garantir que todos os nós da rede concordem com o histórico da transações que compõem os blocos da rede, que será comum à todos, formando assim a rede blockchain (Xiao *et al.*, 2020). Desta forma, os nós são estimulados a participar do processo de validação. Além de proporcionar um ambiente participativo para criação e validação dos blocos, os algoritmos de consenso propõe formas de recompensar os nós honestos, isto é, aqueles que trabalham para manter a integridade da rede e não agem de forma maliciosa.

Para atingir seu objetivo, um algoritmo de consenso deve ser projetado baseado em determinados requerimentos que visam garantir a integridade em uma rede blockchain. No trabalho de Xiao *et al.* (2020) são definidos 4 requerimentos:

- **Terminação**: Para cada nó honesto, uma nova transação pode ser descartada ou aceita na blockchain, juntamente do conteúdo de um bloco;

- **Concordância**: Cada nova transação e seu respectivo bloco pode ser aceito ou descartado por todos os nós honestos. Cada nó honesto deve atribuir ao bloco aceito o mesmo número de sequência. Este requerimento controla a disposição na ordem correta dos blocos e transações;

- **Validade**: Se cada nó recebe o mesmo bloco válido, então este bloco é aceito e inserido na blockchain;

- **Integridade**: Para cada nó honesto, todas as transações aceitas devem ser consistentes entre si. Cada bloco aceito deve ser gerado e inserido na blockchain em ordem cronológica, ligado ao último bloco da rede por meio de sua referência de *hash*.

Os requerimentos de terminação e validade representam a propriedade de vivacidade da rede, pois fazem com que todos os nós honestos participem do processo de recebimento e integração de novos blocos, agregando valor ao processo de consenso. A concordância garante que todos os nós tenham acesso à mesma estrutura de dados blockchain. Assim, a sequência de blocos e transações deve ser a mesma para todos os participantes. Por meio do requerimento de integridade se estabelece a exatidão da origem de transações e blocos, visando evitar ataques como o gasto duplo, o que reforça a consistência e segurança da rede (Xiao *et al.*, 2020; BOURAGA, 2021).

Cada protocolo de consenso pode dispor de diferentes mecanismos para cumprir com os requerimentos citados, e a forma de implementar cada um desses varia de acordo com a rede blockchain. Contudo, cinco componentes básicos podem ser destacados, como feito por Xiao *et al.* (2020):

- **Proposta do bloco**: Geração dos blocos e incorporação das provas da execução deste processo;

- **Propagação da informação**: Disseminação dos blocos a transações pela rede;

- **Validação do bloco**: Checagem dos blocos para produção de provas da geração dos blocos e da validade das transações;

- **Finalização do bloco**: Atingir um acordo para aceitação dos blocos válidos;

- **Mecanismo de incentivo**: Atribuir recompensas para os nós honestos participantes e geração de criptomoedas.

Cada blockchain pode utilizar uma variação de diferentes algoritmos de consenso. Dentre os principais algoritmos de consenso estão o *Proof-of-Work* (PoW), *Proof-of-Stake* (PoS) e *Practical Byzantine Fault Tolerance* (PBFT).

- ***Proof-of-Work***: O algoritmo *Proof-of-Work* tem entre seus principais mecanismos a competição entre os nós participantes do processo de validação (chamados de mineradores) para resolução de um quebra-cabeça criptográfico. O nó que encontrar uma solução primeiro obtém o direito de validar o bloco, que é então criado, dissipado pela rede de nós para que todos os participantes possam verificar sua validade, e, por fim, o nó é adicionado à blockchain. Para estimular a participação honesta dos nós no processo de mineração e compensar os custos financeiros envolvidos neste processo, visto que o esforço computacional exigido tem como consequência um alto consumo de energia, algoritmos de PoW utilizados em blockchains como Bitcoin e Ethereum oferecem uma recompensa ao vencedor. Esta recompensa é feita por meio da obtenção da posse, por parte do minerador, de uma quantidade da moeda virtual utilizada como incentivo na rede, que pode posteriormente ser convertida em valor monetário (i.e., alguma moeda fiduciária) (NAKAMOTO, 2008; BUTERIN, 2014; DRESCHER, 2017). O alto esforço computacional e gasto energético despendido pelos mineradores agrega integridade aos blocos, pois não é vantajoso para um nó malicioso ter um alto gasto para resolução do quebra-cabeça criptográfico de um bloco contendo transações fraudadas e correr o risco iminente do bloco ser rejeitado no processo de validação entre os nós da rede. Por outro lado, o gasto computacional elevado para manutenção da integridade da *blokchain* pode restringir as condições de acesso dos usuários ao processo de mineração, além de aumentar o tempo para inclusão das transações, limitando questões práticas de implantação e uso de sistemas, como escalabilidade e performance (BOURAGA, 2021);

- ***Proof-of-Stake***: O algoritmo PoS foi proposto inicialmente por King e Nadal (2012) com o intuito de mitigar a dependência do alto consumo de energia e recursos computacionais do PoW (OAPOS;DWYER, 2014). No PoS, os nós que se candidatam para participar da criação dos blocos, chamados de validadores, investem uma quantia da criptomoeda vigente na blockchain. Esta quantia também é referida como valor de participação, e funciona como uma conta bloqueada com um saldo que representa o comprometimento do validador em manter a integridade da rede. Quanto maior o valor, maior a chance do validador ser selecionado para criar o próximo bloco. Enquanto no PoW a chance do minerador criar um bloco é proporcional ao seu poder computacional, no PoS a chance é proporcional ao valor de participação investido pelo validador (Xiao *et al.*, 2020; Dinh *et al.*, 2018);

- ***Practical Byzantine Fault Tolerance***: Baseado no trabalho de Castro, Liskov *et al.* (1999), o PBFT é adotado na blockchain por meio de dois tipos de nós, o cliente e o servidor. O nó cliente envia um bloco aos nós servidores, e se o bloco for validado por um número suficiente de nós, então este é adicionado à blockchain. Este processo de validação das transações do PBFT consiste em cinco etapas: (i) o nó cliente envia o bloco proposto para os servidores; (ii) os servidores transmitem o bloco para outros servidores, que devem

avaliar uma série de condições relacionadas à validade do bloco e chegar a um consenso sobre sua aceitação; (iii) Se o bloco for aceito, o segundo grupo de servidores enviam uma mensagem aos outros nós indicando que o bloco está pronto. Assim que esta mensagem é verificada e validada por um número suficiente de nós, estes entram em fase de "entrega"; (iv) após realizar a confirmação, cada nó transmite uma mensagem para a rede para atestar sua ação; e (v) o nó servidor que enviou o bloco recebe a resposta, seja o bloco validado ou não (BOURAGA, 2021; Xiao *et al.*, 2020; AHMED *et al.*, 2019; ZHANG; LEE, 2020).

Apesar de ser crucial para o sucesso de blockchains como Bitcoin e Ethereum, o algoritmo PoW impõe limitações de escalabilidade, performance e participação na rede. Essas questões motivaram pesquisadores e empresas a desenvolver alternativas para mitigação desses problemas, como os algoritmos PoS e PBFT.

Em dezembro de 2020 começou a primeira fase de implantação da *Ethereum* 2.0 [1], uma nova rede blockchain que utiliza o algoritmo de consenso PoS. Com isso, pretende-se aumentar a velocidade de validação das transações e integração dos blocos, expandindo a escalabilidade e otimizando a performance das aplicações.

O algoritmo PBFT é utilizado pela *Hyperledger Fabric* [2], uma plataforma blockchain privada desenvolvida pela Fundação Linux (ANDROULAKI *et al.*, 2018). Assim como o PoS, o PBFT também proporciona economia de energia para validação e integração das transações. Variações do PBFT também são empregadas nas blockchains Stellar (MAZIERES, 2015) e Ripple (SCHWARTZ *et al.*, 2014) (AHMED *et al.*, 2019; Xiao *et al.*, 2020; ZHANG; LEE, 2020).

2.1.3 Escolha do histórico de transações

O funcionamento da blockchain exige um ritmo de trabalho dos mineradores no qual, em algum momento, estes sempre estarão concentrados em alguma das seguintes tarefas: analisar um novo bloco criado por algum nó da rede; ou se esforçar para criar o próximo bloco que, posteriormente, será analisado pelos demais nós.

A capacidade de transmissão e entrega de novos blocos sofre grande influência da capacidade da entrega de mensagens de rede. Por consequência, vários nós podem terminar de construir um bloco em um pequeno intervalo de tempo. Esses blocos são transmitidos pela rede e coletados pelos nós em momentos distintos. Assim, os nós da rede não terão informações idênticas à sua disposição ao mesmo tempo (DRESCHER, 2017).

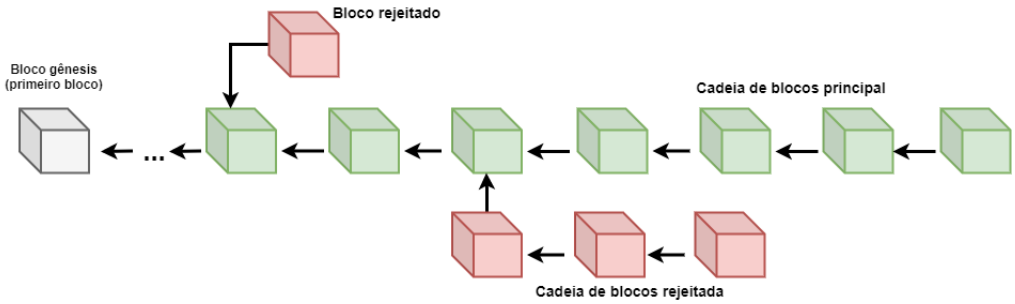
Quando um nó coleta um sua caixa de entrada mais de um nó com o mesmo valor de referência do *hash* do bloco anterior, então uma ramificação, referida também como um *fork*, é criada, já que esses blocos possuem o mesmo bloco pai. Essas ramificações podem formar

[1] <https://github.com/ethereum/eth2.0-specs>
[2] <https://www.hyperledger.org/use/fabric>

uma cadeia com diversos blocos. Dessa forma, a estrutura de dados da blockchain pode ser vista como uma árvore, porém, quando ocorre um *fork*, os nós da rede devem escolher apenas uma ramificação para compor a cadeia de blocos principal. Na Bitcoin é definido que, sempre que ocorrer um *fork*, deve-se escolher a cadeia mais longa, ou, em caso de empate, mantém-se aquela que foi recebida primeiro. (DRESCHER, 2017; SOMPOLINSKY; ZOHAR, 2015).

Em situações de decisão como essa, o protocolo da Bitcoin estabelece que a ramificação escolhida pela maioria dos nós é considerada como parte da cadeia de blocos principal, como ilustrado na Figura 2. Este procedimento é estabelecido pelo protocolo de consenso da blockchain e visa manter a integridade da blockchain por meio do consenso alcançado entre os nós participantes. Porém, deve-se considerar a premissa de que sempre haverá mais de 50% de participantes dispostos a agir de forma honesta (NAKAMOTO, 2008).

Figura 2 – Cadeia de blocos de uma blockchain



Fonte: Monrat, Schelén e Andersson (2019).

2.1.4 Criptografia e autorização de transações

Em uma blockchain pública qualquer nó pode criar transações e submetê-las para validação, enquanto que em uma blockchain privada essa atividade pode ser autorizada à apenas um conjunto de nós. Independente do acesso de leitura e escrita concedido, é essencial que apenas o proprietário legítimo de uma conta possa transferir o direito de propriedade ou de posse associado à sua conta (e.g., uma quantia de criptomoeda) para outra conta.

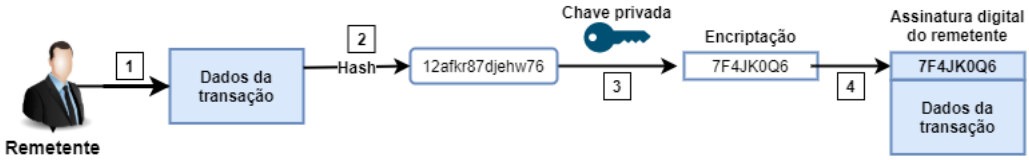
Para garantir que somente o proprietário legítimo transfira a posse, é utilizada uma assinatura digital. Para isso, é aplicada a criptografia de curva elíptica (KOBLITZ, 1987), na qual são utilizadas técnicas de *hash* e criptografia assimétrica por meio do par de chaves que cada nó detém, uma chave pública e outra privada. A chave privada fica disponível apenas para seu proprietário, e é utilizada para criptografar informações, transformando-as em um texto cifrado. Por se tratar de uma criptografia assimétrica, não há como se obter a informação original por meio do texto cifrado resultante. A única forma de descriptografar esse texto e obter novamente a informação original é utilizando a chave pública correspondente, que é única para cada proprietário e representa o identificador de sua conta. A chave pública é compartilhada com

todos, assim, qualquer nó pode usá-la para se certificar de que a transação foi autorizada por quem cedeu a posse (DRESCHER, 2017).

A assinatura é utilizada em duas situações: na assinatura de uma transação; e na verificação de uma transação (DRESCHER, 2017). Na Figura 3 é ilustrado um exemplo em que o proprietário da conta que cede a posse realiza a assinatura da transação por meio de dos passos a seguir:

1. Descreve a transação com todas a informações necessárias, exceto a assinatura;
2. Gera o valor de *hash* dos dados de transação;
3. Utiliza sua chave privada para gerar o valor de *hash* da transação a partir do valor gerado no passo 2. Esse processo é chamado de encriptação;
4. Adiciona o texto cifrado criado no item 3 à transação como sua assinatura digital.

Figura 3 – Processo de assinatura digital de uma transação



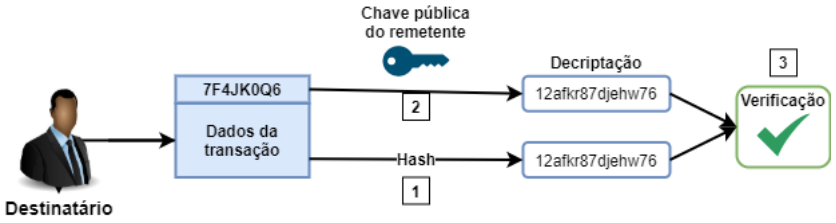
Fonte: Ahmed *et al.* (2019).

O processo de verificação de uma transação é ilustrado na Figura 4, no qual o nó verificador executa os seguintes passos:

1. Cria o valor de *hash* a partir dos dados da transação a ser verificada, com exceção da assinatura;
2. Utiliza a chave pública da conta que está cedendo a posse para descriptografar a assinatura digital da transação. Esse processo é chamado de decriptação;
3. Compara o valor do *hash* gerado no passo 1 com o valor obtido no passo 2. Se ambos forem idênticos, então indica que a transação foi autorizada pelo proprietário da chave privada, que corresponde à chave pública que está cedendo a posse (i.e., o identificador da conta). Caso os valores não sejam idênticos, então conclui-se que o proprietário da chave privada não autorizou a transação, que é descartada.

Um valor de *hash* criptográfico é único para cada transação. Analogamente, a associação entre uma chave pública e uma privada também é único. Essa característica faz com que as assinaturas digitais sejam apropriadas para servir como prova de que o proprietário da chave privada usada para criar a assinatura digital realmente concorda com o conteúdo da transação (DRESCHER, 2017).

Figura 4 – Processo de verificação da assinatura digital de uma transação



Fonte: Ahmed *et al.* (2019).

2.1.5 Bitcoin como um sistema de transição de estados

Do ponto de vista técnico, um livro-razão para gerenciamento de posses de criptomoedas, como o da Bitcoin, pode ser considerado como um sistema de transição de estados. Este estado consiste no status da posse de todos os bitcoins existentes, e uma mudança de estados acontece sempre que uma solicitação de transferência de posse da moeda é aceita, alterando o status de posse das mesmas. No sistema bancário tradicional, seria como transferir uma quantia de dinheiro de uma conta para outra, alterando o estado destas contas, que passam a ter um novo valor de saldo (BUTERIN, 2014).

O estado da Bitcoin é representado pelo conjunto de todas as saídas de transações não gastas, denominadas como *unspent transaction outputs* (UTXO), que representam todas as moedas já mineiradas. Para cada UTXO é atribuído o endereço de seu proprietário, isto é, a chave pública criptográfica de um nó. Cada transação é composta por ao menos uma entrada e saída. Cada entrada contém uma referência para uma UTXO existente e a assinatura digital criptográfica de quem detém sua posse. Uma saída possui uma nova UTXO a ser adicionada ao novo estado (BUTERIN, 2014).

2.2 Blockchain Ethereum

A Ethereum (BUTERIN, 2014) é uma das mais conhecidas implementações da tecnologia blockchain. Ela é definida como uma plataforma de computação distribuída composta por uma rede de computadores que operam de forma descentralizada, autônoma e democrática (WOOD *et al.*, 2014). Embora também lide com geração e gerenciamento de posse de sua criptomoeda, o Ether, essa é apenas uma parte do que a plataforma é capaz de prover.

O funcionamento da Ethereum baseia-se na implantação de contratos inteligentes, que são programas de computador que, uma vez implantados, executam automaticamente e obrigatoriamente de acordo a lógica definida em sua programação. Por meio desses programas é possível estabelecer um acordo entre duas ou mais partes envolvidas, que se comprometem a cumprir com as regras estabelecidas expressas em código (CHEN *et al.*, 2020b).

Na Ethereum, as transações são disparadas por meio de mensagens, que podem conter

instruções que causam a alteração na estado da blockchain (WOOD *et al.*, 2014). Isso acontece, por exemplo, quando um nó executa uma função de um contrato inteligente que altera o valor de algum atributo. Essas transações são coletadas pelos nós para formação dos blocos e são estruturadas por meio de uma variação da árvore de Merkle denominada árvore de Patricia Markle, também chamada de *trie*. Esta árvore opera de forma semelhante à arvore de Merkle na garantia de imutabilidade dos dados, pois nela as informações das transações também ficam armazenadas nos nós folhas, assim como os demais nós também são formados pela criptografia do conteúdo de seus nós filhos. Assim, qualquer alteração nos dados de uma transação causa alterações sucessivas nos nós da mesma ramificação, inclusive a raiz.

Contratos inteligentes são geralmente escritos em linguagens de auto nível e Turing-completas, sendo Solidity a linguagem mais utilizada na plataforma Ethereum. A compilação do contrato resulta em um *bytecode*, um código de baixo nível composto por instruções e argumentos. O *bytecode* é então executado nos nós da rede por meio da Máquina Virtual Ethereum (MVE), uma máquina virtual Turing-completa (CHEN *et al.*, 2020b; SYED *et al.*, 2019).

A Ethereum foi elaborada por Buterin (2014) para ser um protocolo alternativo para criação de DApps. Na plataforma *State of The DApps* (STATE..., 2021) há mais de 3800 DApps contabilizados, sendo que destes, pouco mais de 3 mil utilizam a Ethereum. Nas DApps, geralmente o *front-end* é implementado como uma aplicação *web*, enquanto que o *back-end* é implementado por um ou mais contratos inteligentes (HEWA; YLIANTTILA; LIYANAGE, 2021). Os números envolvendo a plataforma ajudam a dimensionar o tamanho de sua popularidade. Em 2021 o valor de mercado da Ethereum superou 210 bilhões de dólares, sendo a segunda maior plataforma em valor de mercado, atrás apenas da Bitcoin (COINMARKETCAP..., 2021). Também, de acordo com a plataforma Etherscan, há ao menos 2 milhões de contratos inteligentes já executados (ETHERSCAN..., 2021).

Devido às tecnologias e técnicas que compõe a Ethereum, as aplicações que a utilizam dispõe de uma série de propriedades (BUTERIN, 2014; HEWA; YLIANTTILA; LIYANAGE, 2021), tais como:

- Descentralização: Eliminação da necessidade de confiança em uma terceira parte reguladora para execução da lógica do contrato;
- Imutabilidade: Uma vez executado, o código não pode ser alterado, assim como as transações resultantes da interação entre os contratos e os nós;
- Persistência dos dados: Uma vez inseridas na blockchain, as informações contidas em um bloco estarão sempre disponíveis;
- Execução autônoma: A execução de condições programadas e fluxo de eventos a serem realizados são disparados automaticamente conforme o sistema blockchain atinge um determinado estado, garantindo a autonomia da execução. O estado no qual uma ação é

disparada é definido na programação do contrato inteligente, em comum acordo com todas as partes envolvidas;

- Acurácia: Assim que o contrato inteligente é executado, confia-se que as condições programadas serão cumpridas. A acurácia da execução do que foi programado é garantida por meio da transparência envolvida na execução autônoma, pois assim, vieses humanos e erros que podem acontecer em uma execução centralizada são evitados.

A seguir, na Seção 2.2.1, são abordados os tipos de contas que operam na plataforma Ethereum. Detalhes sobre as transações e mensagens usadas para a comunicação entre as contas são tratados na Seção 2.2.2. Na Seção 2.2.3 é apresentada a função de transição de estados da Ethereum. Detalhes sobre a formação dos blocos e seu processo de validação são discutidos nas Seções 2.2.4 e 2.2.5, respectivamente. Alguns exemplos de aplicações baseadas em contratos inteligentes que executam sobre a plataforma Ethereum são discutidos na Seção 2.2.6.

2.2.1 Contas Ethereum

Diferente do modelo de representação de estados da Bitcoin, que é baseado no estado das moedas mineiradas, na Ethereum, o estado da blockchain é definido pelo estado das contas. O estado de todas as contas define o estado da blockchain, que é atualizado sempre que um novo bloco é adicionado. As contas são necessárias para que haja interação dos usuários com a blockchain por meio das transações (ETHEREUM..., 2018). Uma conta pode ser de dois tipos: Contas de Propriedade Externa (CPE); e contas de contrato (CC).

Uma CPE é usada para armazenar os fundos do usuário em Wei, que é a menor sub-denominação de um Ether, sendo um Ether equivalente a 10^{18} Wei. As CPEs são associadas e controladas por uma chave privada, e são necessárias para que um cliente possa participar da rede. As CCs são controladas pelo código de um *bytecode* executável, que é gerado na compilação de um contrato inteligente. Em uma CPE pode-se enviar mensagens para outras CPEs ou para uma CC, basta criar uma mensagem, assinar digitalmente a transação e transmiti-lá para na rede. Sempre que uma CC recebe uma mensagem seu código é ativado. Uma mensagem enviada à uma CC tem o intuito de executar alguma função em seu código. Essa função pode executar alguma operação de leitura ou escrita em seu armazenamento interno (i.e., suas variáveis), ou até mesmo criar e executar outro contrato inteligente. Embora um contrato inteligente possa ser criado por uma CPE ou uma CC, uma CPE não pode ser criada por uma conta (BUTERIN, 2014; CHEN *et al.*, 2020a).

O estado global da Ethereum é definido pelo estado de todas as contas. Internamente, o estado global é obtido por meio de um mapeamento entre os endereços das contas (identificadores de 20 bytes) e o estado de cada conta (WOOD *et al.*, 2014). Ambas as contas possuem um estado dinâmico, definido por:

- ***nonce***: indica o número de transações iniciadas pelo proprietário da CPE correspondente, ou, no caso de uma CC, o número de contratos criados pela conta;
- ***balance***: saldo em Wei sob posse da CPE ou da CC;
- ***storageRoot***: Valor do *hash* da raiz da árvore de Patricia Merkle, a qual armazena o estado das variáveis do contrato associadas ao bytecode correspondente. Este atributo não é aplicável às CPEs;
- ***codeHash***: Valor do *hash* do código em *bytecode* da CC correspondente. Este atributo não é aplicável às CPEs.

As operações requisitadas em uma transação são executadas por meio da MVE, que pode seguramente verificar a identidade do remetente (i.e., uma CPE), pois, assim como na Bitcoin, as transações também são assinadas por meio da técnica de curva elíptica (ETHEREUM. . . , 2018).

2.2.2 Transações e mensagens

Na Ethereum, uma transação se refere a um pacote de dados criptograficamente assinado que armazena uma mensagem a ser enviada por uma CPE. Essa mensagem estabelece uma interação entre uma CPE e uma CC, ou outra CPE, e especifica alguma instrução a ser executada. Há dois tipos de transações: mensagens externas enviadas por uma CPE; e mensagens internas enviadas por uma CC. Ambas as mensagens podem ser usadas para transferência de Ether, e criação e execução de contratos inteligentes (WOOD *et al.*, 2014). Os dois tipos de transações possuem os seguintes atributos:

- ***nonce*:** Utilizado como um contador, pois indica o número total de transações que já foram iniciadas pelo remetente. Ressalta-se que, este item não possui relação ou função semelhante ao *nonce* utilizado nos cabeçalhos dos blocos da blockchain que implementam o algoritmo de consenso *proof-of-work*, abordado na Seção 2.1.2;
- ***gasPrice*:** Um valor em Wei a ser pago para cada unidade de *gas* utilizada na execução da respectiva transação;
- ***gasLimit*:** O valor máximo em *gas* que o remetente está disposto a pagar como taxa para o minerador que vencer a disputa pela criação do bloco no qual essa transação está inclusa;
- **Destinatário (*to*):** O endereço do destinatário da mensagem, que pode ser uma CPE ou uma CC;
- ***value*:** Valor em Wei a ser transferido ao destinatário da mensagem. Em caso de criação de contrato, indica o valor a ser depositado na nova contra criada;

- ***(v, r, s)*:** Os dados indicam a assinatura do remetente, feita por meio do Algoritmo de Assinatura Digital de Curva Elíptica (JOHNSON; MENEZES; VANSTONE, 2001).

Uma transação para criação de um contrato inteligente também possui o seguinte item:

- ***init*:** Especifica, por meio de um vetor de *bytes*, o *bytecode* para o procedimento de inicialização da conta.

O *init* é um fragmento do *bytecode* que é executado apenas na criação da contrato, e então é descartado. Quando executado, retorna o corpo do código da conta, que é um segundo fragmento de código que é executado sempre que a conta recebe uma mensagem, seja por meio de uma transação ou devido a uma execução interna do código (WOOD *et al.*, 2014).

Já transações com mensagens para transferência de Ether e execução de contratos possuem o seguinte atributo (WOOD *et al.*, 2014):

- ***data*:** Um vetor de *bytes* com dados de entrada da mensagem. Esses dados podem ser parâmetros de uma função, por exemplo.

A execução de uma transação pode resultar em um certo custo computacional. Na Ethereum esse custo é calculado em *gas*, e assim, cada tipo de operação possui um determinado custo para ser executada, que varia de acordo com a quantidade de passos computacionais envolvidos, além de um custo fixo de 5 *gas* para cada *byte* dos dados da transação (WOOD *et al.*, 2014). A utilização do *gas* como métrica é benéfica na medida que desvincula o custo computacional envolvido na execução das operações do custo do Wei, que possui valor monetário e está sujeito a variações de mercado. Assim, um cliente pode levar este último fator em consideração no momento de decidir o quanto está disposto a pagar em Wei por unidade *gas* utilizada (*gasPrice*).

O atributo *gasLimit* é essencial para evitar que estruturas de repetição consumam *gas* indefinidamente ou zerem o saldo do remetente, evitando assim maiores perdas. Contratos inteligentes são executas em uma MVE, definida por Chen *et al.* (2020b) como uma máquina quase Turing-completa. O termo "quase" refere-se ao fato de que a execução é limitada à quantidade de *gas* oferecida nas transações (CHEN *et al.*, 2020b).

2.2.3 Função de transição de estados

Na Ethereum, o estado global da blockchain é definido pelo estado das contas, seja uma CPE ou uma CC. Quando uma transação é executada, algum atributo de uma conta é alterado. Esse atributo pode ser o saldo em Ether após a realização de uma transferência, ou também o valor de uma variável de um contrato inteligente, por exemplo. Desta forma, ao executar uma transação (TX), ocorre uma transição de um estado (S) para outro estado (S') (BUTERIN, 2014).

Ao se executar de uma função (F) de transição de estado $F(S,TX) \rightarrow S'$, a seguinte sequência de passos deve ser realizada:

1. Checar se a transação contém todos os atributos necessários;
2. Checar se a assinatura digital é válida;
3. Checar se o *nonce* da transação e da conta do remetente são iguais. Em caso negativo para algum dos 3 primeiros passos, uma mensagem de erro é retornada;
4. Calcular a taxa de execução da transação ($gasLimit * gasPrice$);
5. Identificar o remetente por meio da assinatura digital;
6. Subtrair a taxa do saldo da conta e incrementar o *nonce* do remetente. Caso o saldo não seja suficiente, uma mensagem de erro é retornada;
7. Definir $gas = startGas$, e retirar a quantidade de *gas* a ser paga pelos *bytes* da transação;
8. Transferir a quantidade de Ether especificada na transação da conta do remetente para a conta do destinatário. Se a conta do destinatário não existir, então esta deve ser criada. Se a conta do destinatário for uma CC, executar o código do contrato até que a execução esteja completa, ou até atingir o limite definido em *gasLimit*;
9. Se a transferência falhar por conta de saldo insuficiente do remetente ou por *gasLimit* atingido, todos os estados alterados são revertidos, exceto o pagamento das taxas dos mineradores, que devem receber o valor em suas contas;
10. Se a transferência for bem executada, então o remetente recebe de volta a quantia de *gas* restante, e o minerador recebe a taxas referente ao *gas* consumido.

2.2.4 Blocos

Na Ethereum, os mineradores agrupam as transações em blocos. O cabeçalho de um bloco da Ethereum contém as seguintes informações:

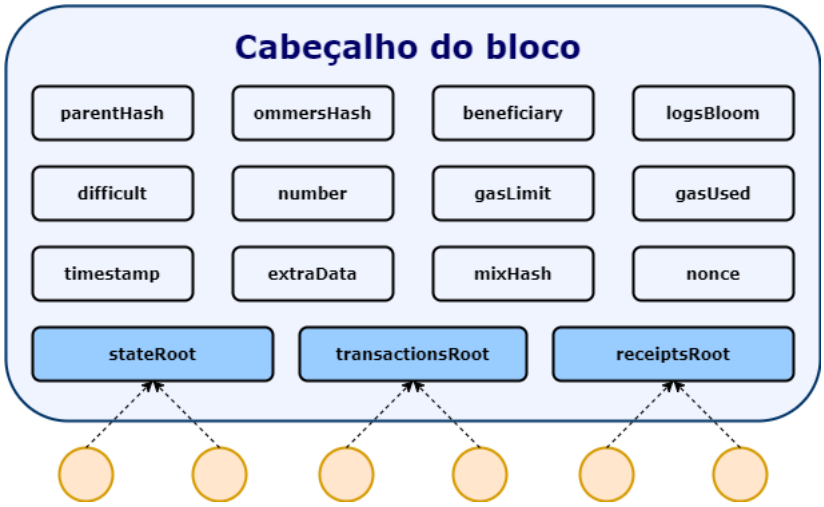
- ***parentHash*:** Valor do *hash* do cabeçalho do último bloco pai, isto é, o antecessor do bloco atual;
- ***ommersHash*:** Valor do *hash* dos cabeçalhos dos blocos cujo antecessor são iguais ao antecessor do bloco atual. Essas blocos são chamados de *ommers*;
- ***beneficiary*:** O endereço do minerador deste bloco. Assim, o minerador é identificado e recebe as taxas de mineração coletadas;

- ***stateRoot*:** Valor do *hash* da raiz da *trie* que contém os estados das transações, após todas serem executadas e finalizadas;
- ***transactionsRoot*:** Valor do *hash* da raiz da *trie* que contém as transações que compõem o bloco;
- ***receiptsRoot*:** Valor do *hash* da raiz da *trie* que contém os recibos com as informações da execução de todas as transações listadas neste bloco;
- ***logsBloom*:** Contém um *Bloom filter*, uma estrutura de dados probabilística usada para testar se um dado elemento é membro de um conjunto. Neste caso, a estrutura é usada para armazenar informações dos *logs* de entrada dos destinatários de cada transação listada no bloco;
- ***difficult*:** Representa o nível de dificuldade para mineração do bloco. Este item é ajustado dinamicamente à cada novo bloco minerado com o objetivo de manter uma média de 15 segundos para o tempo de validação de cada bloco;
- ***number*:** Número de blocos antecessores a este na estrutura de dados blockchain, considerando o primeiro bloco, chamado de bloco gênesis, como bloco zero;
- ***gasLimit*:** Limite de gastos de *gas* por bloco;
- ***gasUsed*:** Soma de todo *gas* utilizado pelas transações deste bloco;
- ***timestamp*:** O horário do início deste bloco, definido a partir do padrão *Unix*;
- ***extraData*:** Dados extras relacionados a este bloco;
- ***mixHash*:** Valor do *hash* que, quando combinado com o *nonce*, prova que um esforço computacional suficiente foi empregado para a criação deste bloco;
- ***nonce*:** Um valor que, quando combinado com o *mixHash* prova que um esforço computacional suficiente foi empregado para a criação deste bloco. É utilizado junto com o *mixHash* como parte do algoritmo de consenso PoW.

Ao se programar um contrato inteligente, pode-se definir e emissão de *logs*, que são mensagens utilizadas para rastrear quando determinados eventos acontecem durante a execução do código. Esse evento pode ser, por exemplo, uma transferência bem sucedida entre duas contas, ou a criação de um contrato. Uma entrada de *log* contém o endereço da conta responsável pelo disparo da mensagem, tópicos que representam os eventos realizados pela transação, e demais dados associados a esses eventos (SOLIDITY. . . , 2020). O *logsBloom* é o componente do bloco em que esses *logs* são armazenados.

A estrutura do bloco é ilustrada na Figura 5. Observa-se que são utilizadas três estruturas em árvore para armazenamento das informações resultantes da execução das transações: *stateRoot*; *transactionsRoot*; e *receiptsRoot*. Desta forma, pode-se rastrear em detalhes o processo de execução de cada transação, o que agrega à Ethereum transparência, auditabilidade e persistência dos dados.

Figura 5 – Estrutura do cabeçalho de um bloco da Ethereum



Fonte: Wood *et al.* (2014).

2.2.5 Validação

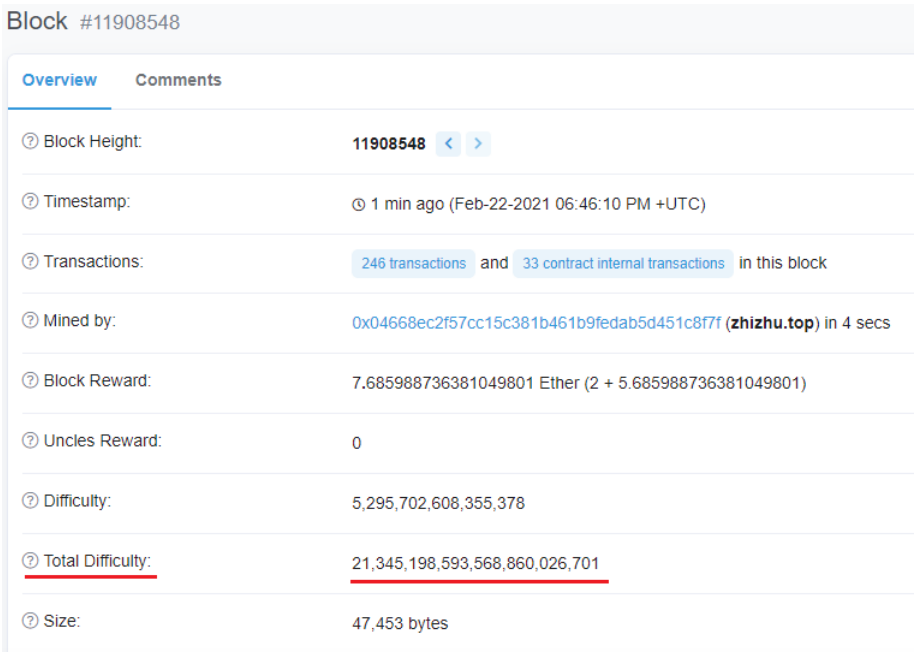
Em uma blockchain, antes de um bloco ser adicionado à cadeia de blocos, este deve passar por um processo de validação por meio de um algoritmo de consenso distribuído. Este processo visa manter a integridade do histórico de transações executadas. Desta forma, cada bloco têm sua integridade verificada pelos nós da rede. Se a maioria dos nós atestarem a integridade do bloco, então este é adicionado na rede principal.

Assim como na Bitcoin, na Ethereum os blocos também podem ser finalizados, transmitidos e recebidos em momentos distintos pelos mineradores, gerando assim um *fork* com versões distintas do histórico de transações. Na Ethereum é utiliza uma variação do protocolo GHOST (SOMPOLINSKY; ZOHAR, 2015) para selecionar como parte da rede principal a ramificação com a maior dificuldade de bloco acumulada, enquanto que as demais sub-redes continuam existindo, mas sem fazer parte da rede principal.

Para cada ramificação, pode-se calcular a dificuldade acumulada, chamada também de "o caminho mais pesado", por meio do acesso às informações contidas no cabeçalho do último bloco adicionado. Como o cabeçalho do bloco contém a dificuldade de mineração do bloco, representada pelo campo *difficult*, basta somar recursivamente o valor da dificuldade de mineração de todos os blocos da rede, com o exceção dos bloco gênesis (WOOD *et al.*, 2014).

Na plataforma Etherscan (ETHERSCAN..., 2021) são coletadas informações sobre todos os blocos e transações incluídos na blockchain Ethereum. Na Figura 6 pode-se observar que, entre outras informações do bloco, há o campo *Total Difficulty*, sublinhado em vermelho, que mostra o valor da dificuldade acumulada na blockchain até o respectivo bloco.

Figura 6 – Dificuldade acumulada de um bloco na Ethereum



Fonte: Etherscan... (2021).

2.2.6 Aplicações

A tecnologia blockchain foi proposta inicialmente com o intuito de apoiar o desenvolvimento de criptomoedas como a Bitcoin. O êxito da Bitcoin chamou atenção tanto da academia quanto da indústria, e, posteriormente, outros tipos de criptomoedas e tecnologias baseadas na blockchain foram desenvolvidas. Como exposto por (SWAN, 2015) e (MAESA; MORI, 2020), esses avanços são classificados como *Blockchain 1.0*, *2.0* e *3.0*. blockchains aplicados ao para gerenciamento de posse de criptomoedas integram um conjunto de aplicações classificado como *Blockchain 1.0*.

Com a introdução dos contratos inteligentes, impulsionados principalmente pela blockchain Ethereum, possibilitou-se a implementação dos DApps. Desta forma, viabilizou-se o uso de sistemas descentralizados projetados para automatizar aplicações financeiras baseadas em criptomoedas, como OADs e sistemas de *tokens*. Tais aplicações são baseadas na junção entre contratos inteligentes e criptomoedas, e são definidas como *Blockchain 2.0* (MAESA; MORI, 2020).

Blockchain 3.0 é o estágio evolucionário no qual a tecnologia não se limita apenas à aplicações financeiras, mas também à áreas como cuidados médicos, ciências, inteligência

artificial, internet das coisas, governança descentralizada, entre outras (MAESA; MORI, 2020).

A seguir, nas Seções 2.2.6.1 e 2.2.6.2 são abordadas algumas áreas de aplicação da tecnologia blockchain, como sistemas de tokens e OADs, que integram a *Blockchain 2.0*. Na Seção 2.2.6.3 é discutido sobre o uso da blockchain para solucionar alguns problemas encontrados na área de cuidados médicos e serviços de saúde. Ao longo das Seções 2.2.6.1, 2.2.6.2 e 2.2.6.3 são citados alguns exemplos de sistemas relacionados com as aplicações tratadas. Como a plataforma Ethereum faz parte do foco e do escopo deste trabalho, são citados apenas exemplos de aplicações desenvolvidas sobre a Ethereum, apesar de existirem outras blockchains que executam aplicações semelhantes.

2.2.6.1 Sistemas de tokens

Um token é um ativo digital e programável gerenciado por um contrato inteligente para ser utilizado em um DApp ou algum projeto específico. Tokens são similares às criptomoedas, porém, enquanto criptomoedas como bitcoin e ether possuem uma blockchain própria para sua mineração e gerenciamento de posse, os tokens são criados sobre a estrutura de uma blockchain já existente (di Angelo; Salzer, 2020).

Tokens são usados para representar o direito sobre algo, de forma que esse direito é representado como um artefato digital, um processo conhecido como tokenização. O gerenciamento de posse do token usufrui de benefícios inerentes à estrutura de uma blockchain, como imutabilidade, persistência dos dados, descentralização, anonimato e auditabilidade (di Angelo; Salzer, 2020; Monrat; Schelén; Andersson, 2019).

Quando um artefato é tokenizado, é possível fracionar seu valor para quem se interessa em obter a posse, assim como já acontece com as criptomoedas tradicionais. Desta forma, facilita-se a entrada de investidores, resultando em um aumento de liquidez dos ativos tokenizados. Além disso, O fato de um token ser programável proporciona o gerenciamento autônomo e a concordância dos direitos dos investidores. (di Angelo; Salzer, 2020).

Um exemplo de aplicação de tokens são as *stable coins*, moedas digitais cujo valor é lastreado de acordo com alguma moeda fiduciária ou fundos de investimentos já existentes. Projetos como o Tether [3] e USD Coin [4] operam com as criptomoedas USDT e USDC, que são lastreadas pela cotação do dólar. Já o Pax Gold (CASCARILLA, 2019), opera por meio do PAXG, uma versão tokenizada do ouro físico licenciado e certificado pelas agências *London Bullion Market Association* e *London Good Delivery*. Desta forma, é possível comprar frações de onças de ouro, equivalente a aproximadamente 31 gramas de ouro, sem a necessidade de adquirir cofres ou contratar serviços de armazenamento disponíveis em bancos (CASCARILLA, 2019).

Entre outros exemplos, vale citar também tokens como o WiBX [5], oferecido como

[3] Tether: Digital money for a digital age. <https://tether.to/>
[4] USDC: the world's leading digital dollar stablecoin. <https://www.circle.com/en/usdc>
[5] WiBX: Crypto jumping coin. <https://www.wibx.io/>

recompensa em sistemas de compartilhamento de produtos, o Aave (AAVE...,), uma criptomoeda utilizada em um sistemas de finanças descentralizadas, e o MCO2 (MOSS...,), token da companhia ambiental MOSS, usado para obtenção de créditos de carbono.

A facilidade para programação e o estabelecimento de padrões para criação de tokens, como o padrão ERC-20 [6], foram fundamentais para o estabelecimento deste tipo de ativo, o qual abrange diversas aplicações. Na plataforma Etherscan [7] pode-se consultar uma lista de tokens, na qual, no momento da escrita deste trabalho, foram encontrados 362,745 tokens, considerando apenas aqueles escritos no padrão ERC-20.

2.2.6.2 Organizações Autônomas Descentralizadas

Uma OAD é uma organização desenvolvida por meio da tecnologia blockchain que pode ser gerida de forma autônoma, sem a necessidade de confiar em uma autoridade central ou estruturas hierárquicas (Wang *et al.*, 2019). Em uma OAD, todas as regras operacionais e de gerenciamento são programadas em um contrato inteligente e gravadas em uma blockchain. Assim, protocolos de consenso e tokens são utilizados como incentivo para estimular a autonomia operacional, governamental e evolucionária das organizações. Por meio da implantação de uma OAD, espera-se abolir modelos de gerenciamento tradicionais baseados em hierarquia, além de reduzir os custos das organizações com comunicação, gerenciamento e colaboração (Wang *et al.*, 2019).

Em 2016, foi lançado o primeiro OAD, chamado de *The DAO* (sigla para *Decentralized Autonomous Organization*), o maior projeto de *crowdfunding* da época, que em pouco tempo arrecadou cerca de 150 milhões de dólares. Por meio do *The DAO*, propostas de investimento eram submetidas e os participantes compravam tokens que davam direito de participação na aprovação das propostas, assim como receber parte dos lucros gerados (Wang *et al.*, 2019).

Após o *The DAO* outras OAD surgiram, como a *Aragon* [8] e a *Steemit* [9]. A *Aragon* é uma plataforma que oferece aos usuários uma infraestrutura para criação e gerenciamento de vários tipos de OADs. A *Steemit* é uma plataforma de mídias sociais baseada em blockchain que, por meio de um sistema de tokens, usuários são recompensados pela criação e curadoria de conteúdos.

Por se tratarem de plataformas com grande capacidade de arrecadação financeira, as OADs tornaram-se alvo de ataques (ATZEI; BARTOLETTI; CIMOLI, 2017). Em junho 2016 ocorreu o caso conhecido como *The DAO Attack*, no qual um participante malicioso explorou uma falha no contrato e transferiu cerca de 3,6 milhões de Ether para sua conta, o equivalente a 50 milhões de dólares (SIEGEL, 2020). Este caso teve grande repercussão e chamou a atenção

[6] ERC-20 Token standard. <https://ethereum.org/en/developers/docs/standards/tokens/erc-20/>
[7] Etherscan. Token Traker. <https://etherscan.io/tokens>
[8] Aragon: Next-level communities run on Aragon. <https://aragon.org/>
[9] Steemit. <https://steemit.com/>

da academia e da indústria, impulsionando estudos e estratégias para detecção e prevenção de vulnerabilidades em contratos inteligentes (CHEN *et al.*, 2020a; LIU; LIU, 2019).

2.2.6.3 Cuidados médicos e serviços de saúde

Com a ampla difusão da informatização de processos em variadas áreas, do acesso à *internet*, aliados à popularização de *smartphones* e computadores pessoais, um dos maiores problemas enfrentados diz respeito à proteção dos dados pessoais dos usuários. Entre esses dados pessoais estão as informações de serviços de saúde. O histórico médico contém dados sensíveis de pacientes, que precisam ser compartilhados com médicos, farmácias, seguradoras, e outras partes interessadas da área da saúde. Ao mesmo tempo, essas dados devem ser protegido contra acessos indevidos e manipulados corretamente pelos profissionais da área, que muitas vezes não têm conhecimento técnico para lidar com os dados de forma segura. Além da preocupação com a proteção dos dados, também há falta de padronização do formato desses dados, que podem enfrentar incompatibilidade no compartilhamento entre instituições médicas, profissionais da saúde, e outras partes interessadas (MAESA; MORI, 2020).

Outro problema na área da saúde diz respeito à falsificação e adulteração da composição de medicamentos. De acordo com a Organização Organização Mundial da Saúde (OMS), em 2017, 1 em cada 10 medicamentos nos países em desenvolvimento eram falsificados ou de baixa qualidade (WHO, 2017).

Diante desses problemas, diversas soluções foram propostas utilizando como base a tecnologia blockchain, que possui potencial para transformar a área de cuidados médicos e serviços de saúde McGhin *et al.* (2019), Aguiar *et al.* (2020).

A tecnologia blockchain pode oferecer uma infraestrutura adequada para integração de dados de prontuários médicos, e outros benefícios proporcionados pela integridade e imutabilidade dos dados. Uma proposta que utiliza contratos inteligentes e a estrutura da Ethereum é o sistema MedRec (EKBLAW *et al.*, 2016), utilizado para gerenciamento de registros de prontuário eletrônicos. O MedRed permite que pacientes consultem suas informações de forma acessível e oferece uma estrutura modular que facilita a interoperabilidade com sistemas já existentes, além de gerenciar questões como autenticação, confidencialidade, contabilidade e compartilhamento de dados (EKBLAW *et al.*, 2016).

Outros exemplos de aplicações da blockchain na área da saúde são relatados nos trabalhos de McGhin *et al.* (2019) e Aguiar *et al.* (2020).

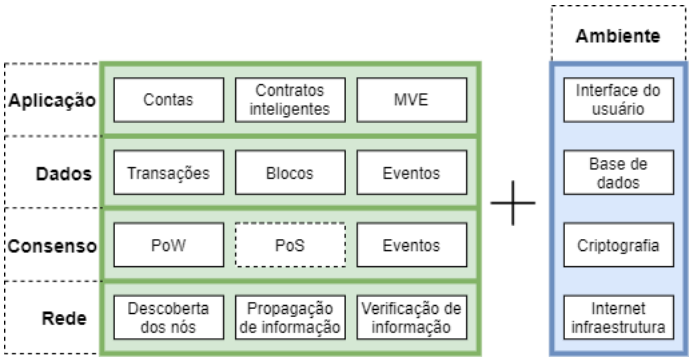
2.2.7 Arquitetura em camadas

A blockchain Ethereum foi projetada sobre uma série de conceitos, protocolos e procedimentos, que dependem de recursos computacionais e tecnológicos para funcionar. Esse conjunto

de elementos compõem a arquitetura da Ethereum. Em seu trabalho, Chen *et al.* (2020a) dividem essa arquitetura em quatro camadas: aplicação; dados; consenso; e rede.

A arquitetura em camadas conta também com elementos presentes no ambiente, relacionados com recursos tecnológicos e infraestrutura, como exposto na Figura 7. Na camada de aplicação estão as contas, que podem ser CPEs ou CCs, os contratos inteligentes, e a MVE, responsável pela execução do *bytecode* gerado na compilação do contrato. A camada de dados consiste nas informações geradas ao longo da execução dos contratos, como transações e *logs* de eventos, e no armazenamento destas, que são acessadas por meio dos blocos. Os mecanismos para validação dos blocos estão incluídos na camada de consenso, no qual um algoritmo de consenso e uma política de incentivos é utilizada para motivar os mineradores a agirem de forma honesta. A camada de rede é a responsável pela comunicação entre os participantes da rede, possibilitando a descoberta de novos nós e a propagação e verificação das informações, essencial para que cada nó possa manter seu histórico de transações atualizado (CHEN *et al.*, 2020a).

Figura 7 – Arquitetura da blockchain Ethereum e seu ambiente de execução



Fonte: Chen *et al.* (2020a).

Cada camada depende de componentes do ambiente de execução das aplicações, como uma interface *web* para interação dos usuários com as aplicações, uma base de dados para armazenamento dos dados da blockchain, mecanismos criptográficos para apoiar os protocolos de consenso, e o serviço de *Internet* que dá suporte às tarefas da camada de rede.

Na Figura 7, nota-se que que, na camada de consenso, o retângulo contendo o algoritmo PoS está com o contorno pontilhado. Isto deve-se ao fato da rede Ethereum 2.0, que opera com o algoritmo de consenso PoS, estar em fase inicial de implementação. Futuramente, de acordo com o planejamento do projeto, espera-se que a Ethereum seja englobada pela Ethereum 2.0.

2.3 Contratos inteligentes

No trabalho de Szabo (1997) foi proposta pela primeira vez a ideia de um contrato inteligente cujas cláusulas são escritas em programas de computador e executadas automaticamente sem a necessidade de confiar em uma terceira parte reguladora. Anos depois, por

meio da tecnologia blockchain, os contratos inteligentes puderam ser de fato implementados, impulsionados por plataformas como Ethereum, Hyperledger Fabric, Corda e Stellar (ZHENG *et al.*, 2020).

No desenvolvimento de um contrato inteligente, as cláusulas contratuais estabelecidas em comum acordo entre as partes envolvidas são expressas por meio de programas de computador executáveis. Esses programas são normalmente escritos em linguagens de programação de alto nível, como a linguagem Solidity (SOLIDITY...,). Na plataforma Ethereum, independente da linguagem, os contratos inteligentes são sempre convertidos em um *bytecode*, uma linguagem de baixo nível que é executada na MVE.

Na linguagem Solidity, um contrato é similar à um objeto. Cada contrato possui atributos e funções que podem ter seus controles de acesso definidos por modificadores. O controle lógico das condições estabelecidas pelas cláusulas podem ser definidos por meio de estruturas de controle, como *if*, *if-else*, *for*, etc.

Um exemplo de contrato escrito na linguagem Solidity é exposto na Figura 8. Neste exemplo, a conta que implementa o contrato pode atribuir algum saldo para si ou para outras contas, e esses valores atribuídos podem ser transferidos para outras contas.

Figura 8 – Contrato escrito na linguagem Solidity para atribuição e transferência de saldo

```
pragma solidity ^0.5.9;

contract Coin {
    address public minter;
    mapping (address => uint) public balances;

    event Sent(address from, address to, uint amount);

    constructor() public{
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public  {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        if (balances[msg.sender] < amount) return;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Fonte: Solidity... (2019).

Em seu trabalho, Zheng *et al.* (2020) descreveram a utilização dos contratos inteligentes como um ciclo de vida que consiste em quatro fases: criação; implantação; execução; e conclusão. Cada fase é descrita como segue:

1. **Criação:** Essa primeira fase se inicia com a negociação entre as partes envolvidas para definição das obrigações, direitos e proibições que devem ser expressas no contrato. Em seguida, desenvolvedores e engenheiros de *software* descrevem esse acordo para alguma linguagem de programação para contratos inteligentes, tal processo para por etapas de

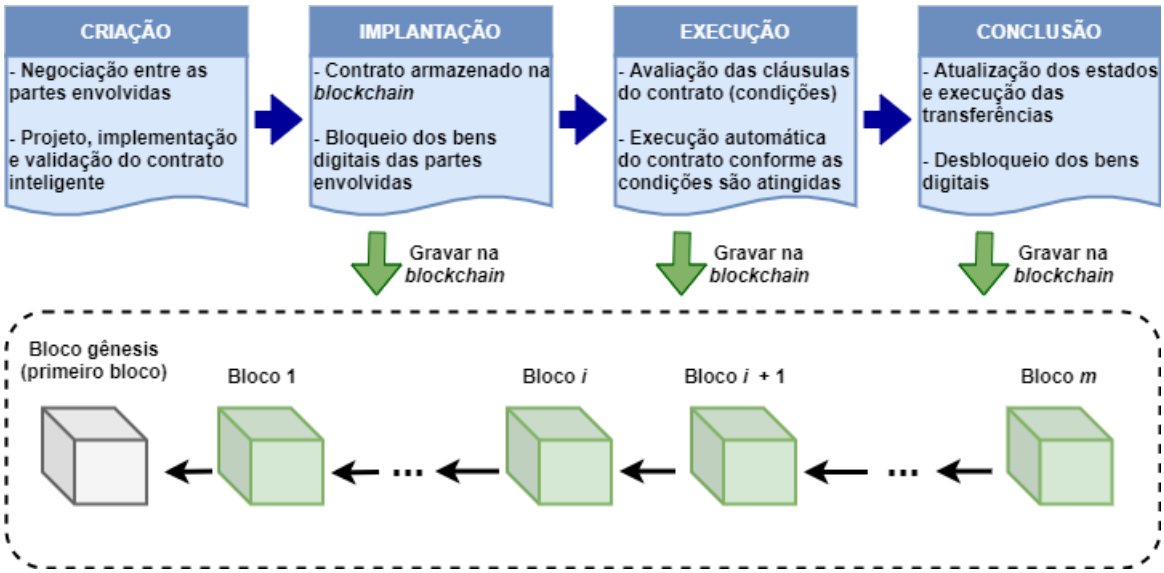
projeto, implementação e validação. A criação de contratos inteligentes é um processo interativo que pode envolver a participação de vários profissionais, como investidores, advogados e engenheiros de *software*;

2. **Implantação:** Consiste em implantar o contrato compilado na blockchain. A implantação é feita por meio de plataformas como a Go Ethereum [10], que opera sobre a blockchain Ethereum. Uma vez implantado na blockchain o contrato inteligente não pode mais ser modificado, e todas as partes envolvidas podem acessá-lo. Vale ressaltar que, nesta etapa, a partes envolvidas podem ter uma parte de seus bens digitais bloqueados. Esse bem digital pode ser uma quantidade de Ether dada como garantia de uma transferência, por exemplo. Assim, as partes envolvidas podem ser identificadas por meio de suas carteiras digitais;

3. **Execução:** Após a implantação as cláusulas contratuais são monitoradas e avaliadas. Assim, conforme as condições estabelecidas são atingidas, as operações e funções expressas no contrato são automaticamente executadas, o que gera um fluxo de transações que são executadas e validadas pelos mineradores.

4. **Conclusão:** Depois que um contrato é executado, o estado das contas das partes envolvidas é atualizado. Logo, as transições e os dados de atualização dos estados são gravados na blockchain, as transferências entre as contas são concretizadas e os bens digitais das partes envolvidas são desbloqueados. Assim, o ciclo de vida de um contrato inteligente é concluído.

Figura 9 – Ciclo de vida de um contrato inteligente baseado em 4 fases: criação, implantação, execução e conclusão



Fonte: Zheng *et al.* (2020).

[10] *Go Ethereum: Official Golang implementation of the Ethereum protocol.* <https://github.com/ethereum/go-ethereum>

O ciclo de vida dos contratos inteligentes é ilustrado na Figura 9. Nota-se que, durante as fases de implantação, execução e conclusão, uma série de transações são geradas, transmitidas, validadas e gravadas na blockchain, proporcionando a rastreabilidade e auditabilidade do contrato (ZHENG *et al.*, 2020).

Devido a imutabilidade da blockchain, um contrato implementado não pode mais ser alterado. Esta propriedade agrega integridade à tecnologia blockchain, mas também ressalta a importância da implementação de contratos livres de erros e de acordo com boas práticas, já que vulnerabilidades presentes nos contratos podem torná-los alvos de ataques.

A seguir, na Seção 2.3.1 são abordadas algumas das vulnerabilidades já encontradas em contratos inteligentes e ataques que ocorreram por meio da exploração dessas vulnerabilidades.

2.3.1 Vulnerabilidades e ataques

Aplicações desenvolvidas por meio de contratos inteligentes, como os DApps e as OADs, costumam envolver transferências e gerenciamento de grandes quantidades de bens digitais, e isso tornou-os alvo de uma série de ataques que exploraram vulnerabilidades encontradas no código desses contratos (ATZEI; BARTOLETTI; CIMOLI, 2017; LIU; LIU, 2019; CHEN *et al.*, 2020a).

Há vários fatores que tornam a implementação de contratos inteligentes propícios à erros. Segundo (ATZEI; BARTOLETTI; CIMOLI, 2017), parte desses erros são ocasionados pelo desalinhamento que há entre a semântica da linguagem Solidity e a intuição dos desenvolvedores. Apesar de alguns elementos em Solidity serem similares aos encontrados em outras linguagens, como o uso de funções, exceções e modificadores de acesso, estes não são implementados da mesma forma.

Desde os primeiros casos notórios de ataques, como o ataque ao The DAO (SIEGEL, 2020), mencionado na Seção 2.2.6.2, diversos trabalhos foram desenvolvidos com o intuito de listar e classificar os ataques e as vulnerabilidades explorados em contratos inteligentes, e também relatar os esforços despendidos na mitigação dessas ameaças.

No trabalho de Chen *et al.* (2020a) são identificadas 40 vulnerabilidades relacionadas com a blockchain Ethereum. Cada vulnerabilidade encontra-se em uma das camadas da arquitetura da Ethereum, a qual é ilustrada na Figura 7. Das vulnerabilidades identificadas, 26 estão na camada de aplicação, que engloba as contas, os contratos inteligentes e a MVE. Destas, 14 estão associadas à programação dos contratos inteligentes. Em outro trabalho, desenvolvido por Atzei, Bartoletti e Cimoli (2017), são identificadas 6 vulnerabilidades em contratos escritos na linguagem Solidity, sobre as quais 6 ataques foram realizados. Algumas dessas vulnerabilidades, assim como os respectivos ataques, são descritos no decorrer desta Seção.

Reentrada

Essa vulnerabilidade ocorre quando o contrato de um receptor externo invoca novamente uma função do tipo *callback* de outro contrato antes que este termine de executar essa função. Quando um contrato vulnerável contém uma função *callback*, um contrato externo pode invocá-la sucessivas vezes até esgotar qualquer saldo contido no contrato (CHEN *et al.*, 2020a; SAYEED; MARCO-GISBERT; CAIRA, 2020).

A vulnerabilidade da reentrada foi observada primeiramente no *The DAO Attack*, em que um contrato externo transferiu cerca de 3,6 milhões de Ether para sua conta, o equivalente a 50 milhões de dólares (SIEGEL, 2020). Apesar dos danos terem sido revertidos, isso causou uma divisão entre os mineradores da Ethereum. A maior parte dos mineradores concordaram em reverter os danos por meio de um *hard fork*, um procedimento no qual é feita uma bifurcação na cadeia de blocos, que neste caso, foi referente ao momento anterior ao ataque. Após o *hard fork*, a cadeia de blocos principal da Ethereum continuou a partir do bloco 1920000 [11], enquanto que a outra cadeia foi continuada pelos mineradores que não concordaram com a decisão, e foi denominada como Ethereum Classic [12].

Delegatecall Injection

Para facilitar o reuso de código, a MVE dispõe do código de operação (do inglês, *opcode*) `delegatecall`, usado para inserir o *bytecode* de um contrato no *bytecode* de outro contrato, que irá executá-lo por meio de uma chamada. Quando isso ocorre, o contrato que é chamado pode alterar as variáveis de estado do contrato que o invocou. Essa característica torna este último contrato vulnerável à ação de contratos maliciosos que, quando chamados, podem causar alterações para obter benefícios e transferir tokens para sua conta (CHEN *et al.*, 2020a).

O primeiro ataque a explorar essa vulnerabilidade ocorreu contra a *Parity Multsignature Wallet*, uma carteira multi-assinatura. Para se autorizar uma transação convencional de Ether, o remetente deve assinar a transação com sua chave privada. Na Ethereum, uma carteira multi-assinatura é um contrato inteligente que requer múltiplas chaves privadas para desbloquear uma carteira e autorizar transferências. Em 2017, uma vulnerabilidade em uma chamada `delegatecall` foi explorada, e cerca de 31 milhões de dólares em Ether foi subtraído da *Parity Multsignature Wallet* (CHEN *et al.*, 2020a).

[11] *Hard Fork Completed*. <https://blog.ethereum.org/2016/07/20/hard-fork-completed/>
[12] Ethereum Classic. <https://ethereumclassic.org/>

CAPÍTULO

3

REVISÃO BIBLIOGRÁFICA

CAPÍTULO

4

TRABALHOS RELACIONADOS

CAPÍTULO

5

MÉTODO

CAPÍTULO

6

CONSIDERAÇÕES FINAIS

REFERÊNCIAS

AAVE: The liquidity protocol. Acesso em: 23 fev 2021. Disponível em: <https://aave.com/>. Citado na página 37.

AGUIAR, E. J. D.; FAIÇAL, B. S.; KRISHNAMACHARI, B.; UEYAMA, J. A survey of blockchain-based strategies for healthcare. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 53, n. 2, p. 1–27, 2020. Citado nas páginas 11, 18 e 38.

AHMED, M.; ELAHI, I.; ABRAR, M.; ASLAM, U.; KHALID, I.; HABIB, M. A. Understanding blockchain: Platforms, applications and implementation challenges. In: **Proceedings of the 3rd International Conference on Future Networks and Distributed Systems**. New York, NY, USA: Association for Computing Machinery, 2019. (ICFNDS '19). ISBN 9781450371636. Disponível em: <https://doi.org/10.1145/3341325.3342033>. Citado nas páginas 20, 24, 26 e 27.

ALMAKHOUR, M.; SLIMAN, L.; SAMHAT, A. E.; MELLOUK, A. Verification of smart contracts: A survey. **Pervasive and Mobile Computing**, Elsevier, p. 101227, 2020. Citado nas páginas 12 e 13.

ANDROULAKI, E.; BARGER, A.; BORTNIKOV, V.; CACHIN, C.; CHRISTIDIS, K.; CARO, A. D.; ENYEART, D.; FERRIS, C.; LAVENTMAN, G.; MANEVICH, Y.; MURALIDHARAN, S.; MURTHY, C.; NGUYEN, B.; SETHI, M.; SINGH, G.; SMITH, K.; SORNIOTTI, A.; STATHAKOPOULOU, C.; VUKOLIć, M.; COCCO, S. W.; YELLICK, J. Hyperledger fabric: A distributed operating system for permissioned blockchains. In: **Proceedings of the Thirteenth EuroSys Conference**. New York, NY, USA: Association for Computing Machinery, 2018. (EuroSys '18). ISBN 9781450355841. Disponível em: <https://doi.org/10.1145/3190508.3190538>. Citado nas páginas 20 e 24.

ATZEI, N.; BARTOLETTI, M.; CIMOLI, T. A survey of attacks on ethereum smart contracts (sok). In: SPRINGER. **International conference on principles of security and trust**. [S.l.], 2017. p. 164–186. Citado nas páginas 12, 37 e 42.

BERTONI, G.; DAEMEN, J.; PEETERS, M.; ASSCHE, G. **The keccak SHA-3 submission. Submission to NIST (Round 3)**. 2011. Citado na página 18.

BOURAGA, S. A taxonomy of blockchain consensus protocols: A survey and classification framework. **Expert Systems with Applications**, Elsevier Ltd, v. 168, 2021. Citado nas páginas 22, 23 e 24.

BROWN, R. G.; CARLYLE, J.; GRIGG, I.; HEARN, M. Corda: an introduction. **R3 CEV, August**, v. 1, p. 15, 2016. Citado na página 20.

BUTERIN, V. [S.l.], 2014. Acesso em: 29 jan. 2021. Disponível em: <https://ethereum.org/en/whitepaper/>. Citado nas páginas 11, 23, 27, 28, 29 e 31.

CASCARILLA, C. **Pax Gold white paper**. 2019. Acesso em: 23 fev 2021. Disponível em: <https://www.paxos.com/pax-gold-whitepaper>. Citado na página 36.

CASTRO, M.; LISKOV, B. *et al.* Practical byzantine fault tolerance. In: **OSDI**. [S.l.: s.n.], 1999. v. 99, n. 1999, p. 173–186. Citado na página 23.

CHEN, H.; PENDLETON, M.; NJILLA, L.; XU, S. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 3, jun. 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3391195>. Citado nas páginas 12, 13, 29, 38, 39, 42 e 43.

CHEN, T.; LI, Z.; ZHU, Y.; CHEN, J.; LUO, X.; LUI, J. C.-S.; LIN, X.; ZHANG, X. Understanding ethereum via graph analysis. **ACM Transactions on Internet Technology**, Association for Computing Machinery, v. 20, n. 2, 2020. ISSN 15576051. Citado nas páginas 27, 28 e 31.

COINMARKETCAP: Today's Cryptocurrency Prices by Market Cap. 2021. Acesso em: 17 fev. 2021. Disponível em: <https://coinmarketcap.com/>. Citado na página 28.

di Angelo, M.; Salzer, G. Tokens, types, and standards: Identification and utilization in ethereum. In: **2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)**. [S.l.: s.n.], 2020. p. 1–10. Citado na página 36.

DIKA, A.; NOWOSTAWSKI, M. Security vulnerabilities in ethereum smart contracts. In: IEEE. **2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)**. [S.l.], 2018. p. 955–962. Citado na página 12.

Dinh, T. T. A.; Liu, R.; Zhang, M.; Chen, G.; Ooi, B. C.; Wang, J. Untangling blockchain: A data processing view of blockchain systems. **IEEE Transactions on Knowledge and Data Engineering**, v. 30, n. 7, p. 1366–1385, 2018. Citado nas páginas 11, 18 e 23.

DRESCHER, D. **Blockchain basics: a non-technical introduction in 25 steps**. [S.l.]: Apress, 2017. Citado nas páginas 11, 16, 17, 23, 24, 25 e 26.

DWORKIN, M. J. Sha-3 standard: Permutation-based hash and extendable-output functions. 2015. Citado na página 18.

EKBLAW, A.; AZARIA, A.; HALAMKA, J. D.; LIPPMAN, A. A case study for blockchain in healthcare:"medrec" prototype for electronic health records and medical research data. In: **Proceedings of IEEE open & big data conference**. [S.l.: s.n.], 2016. v. 13, p. 13. Citado na página 38.

ETHEREUM Homestead Documentation. 2018. Acesso em: 20 fev. 2021. Disponível em: <https://ethdocs.org/en/latest/>. Citado nas páginas 29 e 30.

ETHERSCAN: The Ethereum Blockchain Explorer. 2021. Acesso em: 17 fev. 2021. Disponível em: <https://etherscan.io/accounts>. Citado nas páginas 28 e 35.

FAN, C.; GHAEMI, S.; KHAZAEI, H.; MUSILEK, P. Performance evaluation of blockchain systems: A systematic survey. **IEEE Access**, IEEE, v. 8, p. 126927–126950, 2020. Citado na página 11.

HEWA, T.; YLIANTTILA, M.; LIYANAGE, M. Survey on blockchain based smart contracts: Applications, opportunities and challenges. **Journal of Network and Computer Applications**, v. 177, 2021. Citado na página 28.

JOHNSON, D.; MENEZES, A.; VANSTONE, S. The elliptic curve digital signature algorithm (ecdsa). **International journal of information security**, Springer, v. 1, n. 1, p. 36–63, 2001. Citado na página 31.

KANNENGIESSER, N.; LINS, S.; DEHLING, T.; SUNYAEV, A. Trade-offs between distributed ledger technology characteristics. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 53, n. 2, p. 1–37, 2020. Citado nas páginas 11 e 13.

KIM, K. B.; LEE, J. Automated generation of test cases for smart contract security analyzers. **IEEE Access**, IEEE, v. 8, p. 209377–209392, 2020. Citado na página 13.

KING, S.; NADAL, S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 2012. Acesso em: 05 fev. 2021. Disponível em: <https://www.peercoin.net/whitepapers/peercoin-paper.pdf>. Citado na página 23.

KOBLITZ, N. Elliptic curve cryptosystems. **Mathematics of computation**, v. 48, n. 177, p. 203–209, 1987. Citado na página 25.

LIU, J.; LIU, Z. A survey on security verification of blockchain smart contracts. **IEEE Access**, IEEE, v. 7, p. 77894–77904, 2019. Citado nas páginas 12, 38 e 42.

MAESA, D. D. F.; MORI, P. Blockchain 3.0 applications survey. **Journal of Parallel and Distributed Computing**, Elsevier, v. 138, p. 99–114, 2020. Citado nas páginas 11, 35, 36 e 38.

MARION, J. **Contabilidade basica**. [S.l.]: Atlas, 1985. ISBN 9788547210236. Citado na página 17.

MAZIERES, D. The stellar consensus protocol: A federated model for internet-level consensus. **Stellar Development Foundation**, Citeseer, v. 32, 2015. Disponível em: <https://www.stellar.org/papers/stellar-consensus-protocol>. Citado na página 24.

MCGHIN, T.; CHOO, K.-K. R.; LIU, C. Z.; HE, D. Blockchain in healthcare applications: Research challenges and opportunities. **Journal of Network and Computer Applications**, Elsevier, v. 135, p. 62–75, 2019. Citado na página 38.

Monrat, A. A.; Schelén, O.; Andersson, K. A survey of blockchain from the perspectives of applications, challenges, and opportunities. **IEEE Access**, v. 7, p. 117134–117151, 2019. Citado nas páginas 25 e 36.

MOSS: MCO2 token. Acesso em: 23 fev 2021. Disponível em: <v.fastcdn.co/u/f3b4407f/54475626-0-Moss-white-paper-eng.pdf>. Citado na página 37.

NAKAMOTO, S. **Bitcoin: A peer-to-peer electronic cash system**. [S.l.], 2008. Acesso em: 29 jan. 2021. Disponível em: <https://bitcoin.org/bitcoin.pdf>. Citado nas páginas 11, 17, 18, 19, 23 e 25.

OAPOS;DWYER, K. Bitcoin mining and its energy footprint. **IET Conference Proceedings**, Institution of Engineering and Technology, p. 280–285(5), January 2014. Citado na página 23.

Salah, K.; Rehman, M. H. U.; Nizamuddin, N.; Al-Fuqaha, A. Blockchain for ai: Review and open research challenges. **IEEE Access**, v. 7, p. 10127–10149, 2019. Citado na página 11.

Sankar, L. S.; Sindhu, M.; Sethumadhavan, M. Survey of consensus protocols on blockchain applications. In: **2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)**. [S.l.: s.n.], 2017. p. 1–5. Citado na página 20.

SAYEED, S.; MARCO-GISBERT, H.; CAIRA, T. Smart contract: Attacks and protections. **IEEE Access**, IEEE, v. 8, p. 24416–24427, 2020. Citado nas páginas 12 e 43.

SCHWARTZ, D.; YOUNGS, N.; BRITTO, A. *et al.* The ripple protocol consensus algorithm. **Ripple Labs Inc White Paper**, v. 5, n. 8, p. 151, 2014. Disponível em: <https://cryptoguide.ch/cryptocurrency/ripple/whitepaper.pdf>. Citado nas páginas 20 e 24.

SIEGEL, D. **Understanding The DAO Attack**. 2020. Acesso em: 24 fev. 2021. Disponível em: <https://www.coindesk.com/understanding-dao-hack-journalists>. Citado nas páginas 12, 37, 42 e 43.

SINGH, A.; PARIZI, R. M.; ZHANG, Q.; CHOO, K.-K. R.; DEHGHANTANHA, A. Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. **Computers & Security**, Elsevier, v. 88, p. 101654, 2020. Citado nas páginas 12 e 13.

SOLIDITY documentation. [S.l.]. Acesso em: 16 fev. 2021. Disponível em: <https://readthedocs.org/projects/solidity/>. Citado na página 40.

SOLIDITY version 0.5.9 documentation. 2019. Acesso em: 02 mar. 2021. Disponível em: <https://docs.soliditylang.org/en/v0.5.9/>. Citado na página 40.

SOLIDITY version 0.7.4 documentation. 2020. Acesso em: 21 fev. 2021. Disponível em: <https://docs.soliditylang.org/en/v0.7.4/index.html>. Citado na página 33.

SOMPOLINSKY, Y.; ZOHAR, A. Secure high-rate transaction processing in bitcoin. In: SPRINGER. **International Conference on Financial Cryptography and Data Security**. [S.l.], 2015. p. 507–527. Citado nas páginas 25 e 34.

STATE of the DApps - DApp Statics. 2021. Acesso em 16 fev. 2021. Disponível em: <https://www.stateofthedapps.com/stats>. Citado na página 28.

SWAN, M. **Blockchain: Blueprint for a New Economy**. [S.l.]: O'Reilly Media, 2015. ISBN 9781491920473. Citado nas páginas 11 e 35.

SYED, T. A.; ALZAHRANI, A.; JAN, S.; SIDDIQUI, M. S.; NADEEM, A.; ALGHAMDI, T. A comparative analysis of blockchain architecture and its applications: Problems and recommendations. **IEEE access**, IEEE, v. 7, p. 176838–176869, 2019. Citado na página 28.

SZABO, N. Formalizing and securing relationships on public networks. **First Monday**, v. 2, n. 9, Sep. 1997. Disponível em: <https://journals.uic.edu/ojs/index.php/fm/article/view/548>. Citado na página 39.

TANEMBAUM, A. S.; STEEN, M. V. **Sistemas Distribuídos-Princípios e Paradigmas**. [S.l.]: Person Prentice-Hall, 2007. Citado nas páginas 15, 16 e 17.

VACCA, A.; SORBO, A. D.; VISAGGIO, C. A.; CANFORA, G. A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. **Journal of Systems and Software**, Elsevier, p. 110891, 2020. Citado na página 12.

VARELA-VACA, Á. J.; QUINTERO, A. M. R. Smart contract languages: A multivocal mapping study. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 54, n. 1, p. 1–38, 2021. Citado na página 12.

Wang, S.; Ding, W.; Li, J.; Yuan, Y.; Ouyang, L.; Wang, F. Decentralized autonomous organizations: Concept, model, and applications. **IEEE Transactions on Computational Social Systems**, v. 6, n. 5, p. 870–878, 2019. Citado na página 37.

WHO. **Substandard and Falsified Medical Products**. 2017. Acesso em: 25 fev. 2021. Disponível em: <https://www.who.int/news-room/fact-sheets/detail/substandard-and-falsified-medical-products>. Citado na página 38.

WOOD, G. *et al.* Ethereum: A secure decentralised generalised transaction ledger. **Ethereum project yellow paper**, v. 151, n. 2014, p. 1–32, 2014. Citado nas páginas 27, 28, 29, 30, 31 e 34.

Xiao, Y.; Zhang, N.; Lou, W.; Hou, Y. T. A survey of distributed consensus protocols for blockchain networks. **IEEE Communications Surveys Tutorials**, v. 22, n. 2, p. 1432–1465, 2020. Citado nas páginas 21, 22, 23 e 24.

ZHANG, R.; XUE, R.; LIU, L. Security and privacy on blockchain. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 52, n. 3, jul. 2019. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3316481>. Citado na página 11.

ZHANG, S.; LEE, J.-H. Analysis of the main consensus protocols of blockchain. **ICT express**, Elsevier, v. 6, n. 2, p. 93–97, 2020. Citado na página 24.

ZHENG, Z.; XIE, S.; DAI, H.-N.; CHEN, W.; CHEN, X.; WENG, J.; IMRAN, M. An overview on smart contracts: Challenges, advances and platforms. **Future Generation Computer Systems**, v. 105, p. 475–491, 2020. ISSN 0167-739X. Citado nas páginas 11, 40, 41 e 42.

ZHU, Q.; LOKE, S. W.; TRUJILLO-RASUA, R.; JIANG, F.; XIANG, Y. Applications of distributed ledger technologies to the internet of things: A survey. **ACM computing surveys (CSUR)**, ACM New York, NY, USA, v. 52, n. 6, p. 1–34, 2019. Citado na página 11.


ICMC
USP
SÃO CARLOS