Universidade de São Paulo – USP
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Blockchain e Ethereum

Aplicações e Vulnerabilidades

Trabalho de Formatura Supervisionado (MAC0499)

Frederico Lage Ferreira
Orientador: Professor Doutor Routo Terada

São Paulo, 27 de novembro de 2017.

Resumo

Em 2008, ao mesmo tempo que a indústria bancária e financeira enfrentava uma de suas maiores crises, a primeira criptomoeda descentralizada estava germinando na *Internet*. Ao mesmo tempo que a confiança da população em entidades reguladoras e autoridades certificadas despencava até o fundo do poço, um protocolo de confiança digital surpreendentemente simples foi publicado em uma lista de e-mails. A criação do *Bitcoin* trouxe consigo a ideia do *blockchain* e suas potenciais aplicações. Qualquer interação entre dois ou mais participantes, que envolva algum tipo de troca de valor e que dependa de algum protocolo de confiança para garantir que não ocorram fraudes, de repente tem uma potencial solução usando um *blockchain*. Foi tendo em vista essa ampla gama de possíveis casos de uso que um desenvolvedor decidiu criar o *Ethereum* para servir como uma plataforma geral de implementação e execução de aplicações descentralizadas baseadas em *blockchain*. Este trabalho visa não apenas explicar os principais conceitos e funcionamento de *blockchains* em geral e do *Ethereum* especificamente, mas explorar principalmente como estes sistemas lidam com a questão da segurança digital.

Palavras-chave: *blockchain, Ethereum, Smart Contracts*, aplicações descentralizadas.

Abstract

In 2008, just as the banking and financial industries were facing one of their largest crises, the first decentralized cryptocurrency was taking root on the Internet. Just as the populace's trust on regulating entities and certificate authorities were plummeting to an all-time low, a surprisingly simple digital trust protocol was published on an email list. The creation of Bitcoin brought with itself the idea of the blockchain and its potential applications. Any interaction between two or more parties, that depended on some trust protocol to ensure no frauds occurred, suddenly had a potential solution using a blockchain. It was by keeping that wide array of possible use cases that a developer decided to create Ethereum to serve as a general platform for the development and execution of decentralized applications based on blockchain. This paper intends to not only explain the main concepts and workings of blockchains in general and Ethereum specifically, but mostly explore how these systems handle the matter of digital security.

Keywords: blockchain, Ethereum, Smart Contracts, decentralized applications.

Sumário

1 – Introdução

Desde o fim da década de 2000, poucas tecnologias têm causado tanto furor quanto o *blockchain* e as criptomoedas. Além de invadirem um território que era anteriormente o monopólio único dos bancos centrais nacionais, permitiram que qualquer pessoa com acesso à internet disponha de meios para armazenar e trocar valores sem depender de moeda física. Embora muitos já tenham essa possibilidade a bastante tempo com seus cartões de crédito e débito e com suas ferramentas de internet *banking*, ainda havia uma parcela significativa da população mundial sem acesso a métodos de pagamento digitais. As criptomoedas mudaram isso.

Mas as mudanças não param por aí. Embora a ideia de criar um dinheiro digital seja quase tão antiga quanto a própria internet, foi apenas com a inovação do *blockchain* que os principais problemas inerentes à confiança em uma entidade central foram contornados. E então percebeu-se que este mecanismo de confiança implementado pelo *blockchain* tinha potencial para ser muito mais do que só um livro-razão de uma moeda virtual.

Como acontece com inovações assim, há aqueles que se empolgam excessivamente. Não é difícil encontrar artigos onde o autor exalta a ideia do *blockchain* como se fosse a nova descoberta da roda, que vai revolucionar tudo, que vai levar à *Web* 3.0, que vai criar sistemas seguros de identidade digital, que vai permitir resolver os problemas da corrupção, da pobreza e dos *spams* de e-mail. É possível sim idealizar soluções para estes problemas que se beneficiariam bastante da estrutura de um *blockchain*, porém esta não se trata de uma bala de prata que milagrosamente resolve tudo.

A implementação de sistemas em *blockchain* requer que se resolvam vários problemas técnicos não triviais, principalmente em se tratando de garantir a segurança do sistema. Portanto, este trabalho visa apresentar as soluções existentes, explicando seu funcionamento e seus potenciais pontos de falha. Em particular, decidiu-se focar no *Ethereum* ao invés do *Bitcoin* já que a gama de aplicações da criptomoeda original é significativamente menor que a do seu ambicioso descendente.

2 – *Blockchain*

2.1 – O que é *blockchain*

Como o próprio nome indica, um *blockchain* é uma cadeia de blocos, onde cada bloco contém informações. Mais especificamente, esse termo é utilizado para identificar um tipo de banco de dados distribuído em uma rede de vários participantes, no qual não há entidade central controladora e onde nenhum participante é mais confiável que qualquer outro, dependendo, portanto de um consenso descentralizado. A consequência direta dessa descentralização é que o banco de dados deixa de ter um ponto central de falhas e vulnerabilidades. Um potencial *hacker* teria que atacar todos (ou no mínimo a maioria) dos participantes para conseguir realizar qualquer alteração significativa, o que se torna cada vez mais impraticável à medida que o número destes participantes aumenta.

2.2 - Histórico

Desde a década de 90, a ideia de uma série de blocos criptografados de informação vinha sendo desenvolvida por *Stuart Haber* e *W. Scott Stornetta* como uma solução para o problema de marca temporal de documentos digitais. A ideia era usar funções de *hash* criptográficas para garantir não apenas a ordem das informações, mas também que informações anteriores não fossem adulteradas.

No final dos anos 2000, enquanto desenvolvia o *Bitcoin*, Satoshi Nakamoto[1] implementou o primeiro *blockchain* para servir como livro-razão da sua moeda virtual. O objetivo de implementar este livro-razão inviolável era resolver o problema do gasto duplo de moedas virtuais, isto é, evitar que uma certa quantidade de moeda fosse gasta duas vezes em situações distintas.

Desde então, uma imensa variedade de aplicações dessa ideia vem sendo desenvolvidas e muitas outras ainda estão sendo idealizadas. Organizações

[1] Satoshi Nakamoto na verdade é um pseudônimo usado por uma pessoa (ou grupo de pessoas) que, apesar de muita especulação, ainda se mantém anônima(s) até a presente data.

autônomas, sistemas de votação, de gerenciamento de identidade virtual, de gerenciamento de direitos autorais, de governança, de reputação, de registros imobiliários e muito mais.

2.3 - Alguns conceitos básicos

2.3.1 – Árvores de Merkle

A Árvore de Merkle é uma estrutura de dados voltada para permitir a fácil verificação da presença de uma certa informação em um certo local.

Inicialmente tendo uma grande quantidade de informação, esta será quebrada em várias partes, mantidas em ordem. A seguir, aplica-se uma função de *hash* a cada uma destas partes. O próximo passo consiste em, seguindo a ordem original, concatenar estes *hashes* dois a dois (ou três a três ou mais, dependendo do tipo de árvore que se deseja obter) e aplicar a função de *hash* a esta concatenação. O resultado será o nó de uma árvore e os *hashes* que o originaram, seus filhos. Esse procedimento é repetido com todas as partes e depois com os nós gerados pelas partes, e assim sucessivamente, até que se obtenha o nó raiz. Ou seja, é uma árvore construída de baixo para cima.

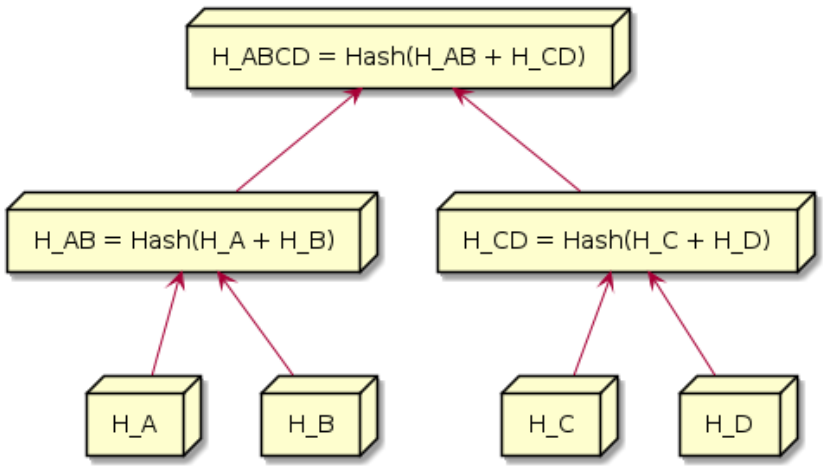

Figura 1: Uma Árvore de Merkle com quatro partes de informação (A, B, C e D)

O motivo para se utilizar esta estrutura é que ela permite um procedimento chamado Prova de Merkle. Neste, um cliente deseja verificar que uma certa informação se encontra em uma certa posição da árvore. O cliente precisa possuir o *hash* da raiz fornecido por alguma fonte confiável e a parte que está sendo inquirida precisa enviar apenas os *hashes* complementares ao longo do ramo que leva da folha até a raiz (por exemplo, na figura acima, para provar que B está na posição dele, bastaria enviar H_A e H_CD, o que permitiria verificar que a raiz obtida é a mesma esperada). Pode parecer desnecessário em uma árvore pequena como a ilustrada acima, mas na prática ela reduz para $\log n$ (onde *n* é a quantidade de folhas na árvore) a quantidade de *hashes* necessárias para a prova.

2.3.2 – Chaves assimétricas

Um dos conceitos fundamentais da criptografia moderna, o uso de pares de chave, uma pública e uma privada, tem uma ampla gama de utilidades. Explicar detalhadamente como funcionam está além do escopo deste trabalho, mas seu funcionamento básico é simples. O algoritmo gera um par de chaves, uma pública que será de conhecimento geral e uma privada, que deve ser guardada em segredo pelo seu dono.

O algoritmo gera tais chaves de modo que, caso se use a chave pública para encriptar uma mensagem, apenas o possuidor da chave privada será capaz de desencriptá-la. Alternativamente, o dono da chave privada pode utilizá-la para gerar uma assinatura em sua mensagem e qualquer um com a chave pública e a mensagem pode verificar se a assinatura é legítima.

2.3.3 – *Hashes*

Outro conceito importante da criptografia utilizado pelo *blockchain* é o de *hashes*. Uma função de *hash* aceita como entrada dados de um tamanho arbitrário e devolve um valor em tamanho fixo (que é chamado de *hash*). Em particular, funções de *hash* criptográficas são escolhidas de modo que seja fácil calcular o resultado da

sua aplicação em alguma informação, porém que seja extremamente difícil deduzir tal informação original caso se conheça apenas a função e o *hash* resultante.

2.4 – Estrutura básica de um *blockchain*

Cada computador conectado à rede de um *blockchain* (chamado de nó) contém uma cópia completa deste banco de dados e realiza trabalhos de validar e transmitir as transações aos demais nós. A principal característica que o distingue é que uma vez que uma informação é registrada ela nunca será apagada ou modificada. O "passado" de um *blockchain* é imutável, apenas o "presente" continua sendo registrado e, a qualquer momento, é possível rever toda a "história" desse banco de dados, o que não apenas resulta em uma grande confiabilidade, mas torna trivial qualquer eventual auditoria. O ponto negativo disso é que o espaço em disco ocupado por tais informações está constantemente crescendo.

Em um *blockchain* há dois tipos básicos de registros: blocos e transações. Começando com um bloco inicial (chamado de bloco gênese) que registra o estado inicial do banco de dados, seguido pelos blocos subsequentes, cada qual contém um grupo de transações já validadas. Cada um deles (exceto o bloco gênese) contém um *hash* do bloco anterior, criando o encadeamento entre eles e garantindo a integridade da informação, já que é impossível alterar blocos antigos sem alterar todos os blocos subsequentes, o que seria percebido pelos demais nós da rede. Portanto, um *blockchain* pode ser entendido como um estado inicial seguido de um certo número de funções de transição, agrupadas em blocos. De fato, o estado atual do banco de dados está contido em um *blockchain* apenas de maneira abstrata, sendo necessário que cada nó determine tal estado partindo do inicial e aplicando as subsequentes transações.

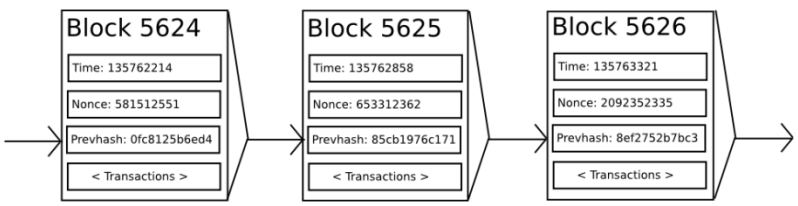

Figura 2: Uma representação visual de um blockchain

Os blocos são criados através de um processo conhecido como mineração. Um nó pega um certo número (geralmente delimitado pelo consenso da maioria) de transações já validadas e as inclui no bloco. Em seguida, deve criar o cabeçalho do bloco. Cada *blockchain* pode determinar diferentes informações que devem compor este cabeçalho, mas via de regra ele deve conter o *hash* do bloco anterior para criar o encadeamento e uma marca de tempo indicando quando foi criado. Uma vez concluído o bloco, obtém-se seu *hash* que é gravado juntamente dele para identificá-lo e para ser incluído no cabeçalho do próximo.

Um outro passo comum na criação deste cabeçalho consiste em organizar as transações em uma Árvore de Merkle e gravar a raiz desta árvore no cabeçalho. Isso permite que se faça um *hash* apenas do cabeçalho para servir como assinatura do bloco (já que a raiz de Merkle é um *hash* das transações) e permite a verificação de transações específicas usando uma Prova de Merkle.

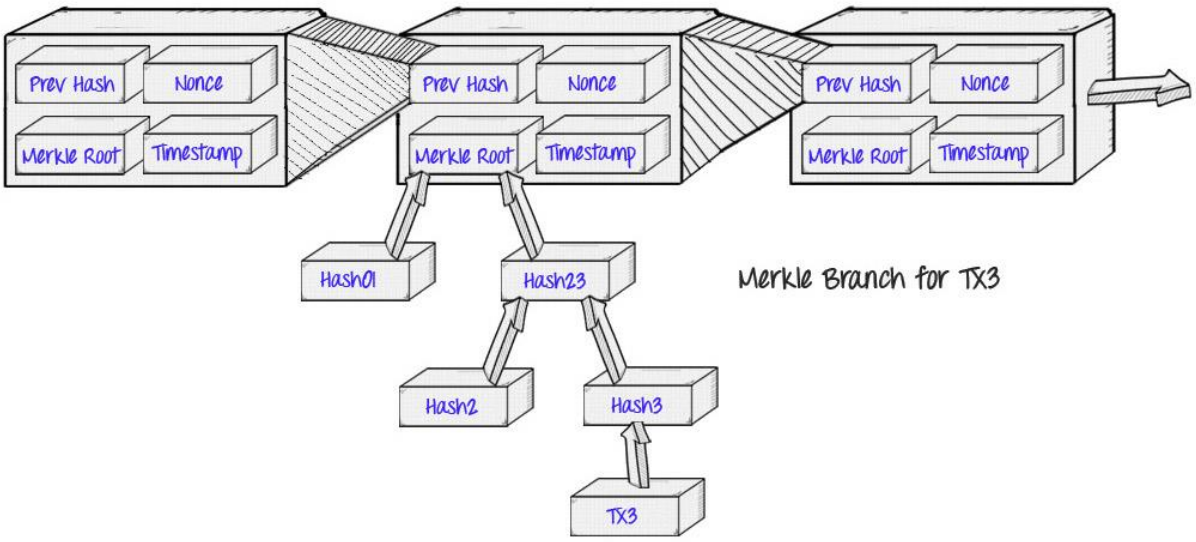

Figura 3: As transações estão nas folhas da Árvore Merkle, mas apenas a raiz fica no cabeçalho, acelerando o processo de hash.

Para evitar que ocorram ataques ao sistema neste processo, é preciso que exista algum mecanismo de consenso que incentive aos mineradores serem bem-intencionados. O mais comum, usado pelo *Bitcoin* e pelo Ethereum (embora este segundo tenha planos de mudar) é a Prova de Trabalho (*Proof of Work* em inglês), que tem como objetivo garantir que o criador do bloco dedicou uma certa quantidade

de recursos de tempo e energia enquanto criava tal bloco, de modo que seja economicamente desvantajoso tentar gerar uma cadeia de blocos falsa.

No *Bitcoin*, tal prova consiste em adicionar um *nonce* (isto é, um número arbitrário que só será usado uma vez) ao cabeçalho e então obter o *hash* do cabeçalho usando o algoritmo SHA256. Se o resultado obtido for menor do que um certo valor alvo, a prova é aceita. Caso contrário, muda-se o *nonce* e tenta-se novamente (como esse número é de apenas 32 bits, é comum que os valores possíveis se esgotem, o que exige que alguma outra parte do cabeçalho, como a marca de tempo, seja alterada para tentar novamente). Como o SHA256 é um algoritmo pseudoaleatório com resultado imprevisível, não há qualquer maneira de se obter uma Prova de Trabalho válida exceto por tentativa e erro alterando o *nonce* a cada tentativa. O valor alvo é redefinido a cada 2016 blocos para aumentar ou diminuir a dificuldade, de modo que seja criado em média um bloco a cada dez minutos. Para incentivar os mineradores, estes adicionam uma transação ao bloco no qual se recompensam (atualmente é com 12.5 BTC) pela criação do mesmo antes de iniciar a criação do cabeçalho. Assim, todo dia milhões de nós competem entre si na mineração de blocos a fim de ganhar esta recompensa e auditam uns aos outros para garantir que ninguém está “roubando”.

Existem outros mecanismos de consenso possíveis como a Prova de Participação (*Proof of Stake*) e a Prova de Queima (*Proof of Burn*). Na primeira, os nós são selecionados de maneira aleatória para minerar um bloco, mas tem sua probabilidade aumentada quando atendem certos critérios (por exemplo, o critério de “idade da moeda” aumenta a chance de quem vem guardado uma certa unidade de valor a mais tempo). Já a prova de queima propõe que o criador do bloco queime um pouco da sua criptomoeda (o que, na prática, consiste em enviar o valor para um endereço do qual não é possível recuperá-lo) para provar que não está mal-intencionado.

A validação de um novo bloco segue um procedimento simples: verifica-se que o bloco anterior referenciado existe e é válido; verifica-se a marca temporal do novo bloco, que deve ser maior que a do anterior e não pode estar muito distante no futuro (diferentes *blockchains* determinam diferentes limites); verificam-se quaisquer informações adicionais que cada *blockchain* específico coloca no seu cabeçalho;

verifica-se o mecanismo de consenso do novo bloco; dado o estado do banco de dados calculado até o bloco anterior, aplicam-se (em ordem) todas as transações registradas no novo bloco a este estado, obtendo uma variedade de estados intermediários; se a aplicação de alguma transação resultar em um estado intermediário inválido, a verificação falhou; caso contrário, a verificação foi bem sucedida e o nó guarda para si o novo estado atual do *blockchain*, com o novo bloco ao final da cadeia. Esta validação é feita por cada nó quando recebe a informação de que um novo bloco foi criado. É por isso que o sistema funciona a base do consenso da maioria. Se em qualquer etapa da criação de um bloco o minerador fizer algo que vá contra o consenso, seu novo bloco não será aceito pelos demais, tendo então desperdiçado tempo e energia.

Por se tratar de uma rede distribuída, frequentemente acontece de dois ou mais blocos diferentes, todos legítimos, serem criados quase que simultaneamente e adicionados à mesma posição no *blockchain* em nós diferentes da rede. Isso cria um *fork* temporário do banco de dados, no qual duas ou mais versões competem para serem aceitas. Cada rede deve ter definido algum algoritmo que permita avaliar estas versões e decidir uma pontuação para elas, decidindo então a vencedora. Por exemplo, em *blockchains* que usam Prova de Trabalho, tal pontuação é calculada como o trabalho total acumulado em uma cadeia. Blocos criados que acabam ficando de fora da rede após este “desempate” são chamados blocos órfãos. Em geral, estes algoritmos são desenvolvidos de modo tal que essa pontuação continue crescendo à medida que se adicionam novos blocos, de modo que a possibilidade de um certo bloco se tornar órfão diminui exponencialmente quanto mais a cadeia cresce.

2.5 – Efeitos do *blockchain*

O uso da Prova de Trabalho acabou gerando um fenômeno interessante. Como a mineração bem-sucedida de um bloco é valiosa (e se torna mais ainda a medida que a criptomoeda se valoriza), passaram-se a desenvolver processadores especializados em minerá-los. Os chamados *ASIC*s (*Application-Specific Integrated Circuit*, ou Circuito Integrado de Aplicação Especifica em português) são

desenvolvidos para paralelizar e otimizar ao máximo o processo de mineração, dando aos seus usuários uma vantagem significativa sobre outros mineradores. De fato, atualmente grande parte da mineração de *Bitcoins* tem sido feita na China, onde grandes centros de computação com *ASIC*s tiram vantagem da energia subsidiada pelo governo do país.

Além da corrida pelo ouro, houve também um grande interesse por parte de empreendedores que viram no *blockchain* uma tecnologia que tem potencial para mudar paradigmas tanto quanto a *Internet* mudou na década de 90. Diversas empresas já estão desenvolvendo seus *blockchains* particulares para não ficarem de fora da onda.

Inclusive, isso gerou algumas discordâncias quanto à definição de *blockchain*. Existem aqueles que argumentam que para que seja considerado um *blockchain* o sistema deve ser público e aberto, caso contrário acaba sendo nada mais que um banco de dados pesado e incômodo.

3 - *Ethereum*

No *Bitcoin*, é possível que uma unidade de valor seja possuída não por uma chave pública, mas sim por script expressado em uma linguagem simples baseada em pilhas, o qual realiza alguma ação com esta unidade de acordo com sua programação. Porém tal linguagem é limitada e, principalmente, não é Turing completa (por exemplo, não possui *loops*). Algumas outras limitações incluem: inabilidade de uma unidade de valor se fragmentar; impossibilidade de a unidade ter vários estados intermediários (no *Bitcoin*, uma unidade está gasta ou não-gasta); e inacessibilidade pela unidade a algumas informações do *blockchain*.

Vitalik Buterin era um dentre muitos programadores envolvidos com o projeto do *Bitcoin* no início da década de 2010. Ele argumentava que uma linguagem de scripts mais robusta deveria ser criada e implementada ao sistema da criptomoeda, mas não conseguiu obter apoio para levar sua ideia adiante. Antecipando que futuramente surgiriam muitas ideias que se beneficiariam da estrutura segura de um *blockchain*, mas que provavelmente não seriam grandes o bastante para ter uma rede suficientemente ampla, decidiu criar uma nova plataforma dedicada ao desenvolvimento de aplicações descentralizadas e publicou em 2013 o white paper que descrevia o *Ethereum*. Seguindo-se um processo muito bem-sucedido de *crowdfunding* (quando criadores ou desenvolvedores buscam uma grande quantidade de investidores pequenos para financiar seus projetos), foi criada a *Ethereum Foundation* e uma equipe começou a implementar o sistema idealizado pelo seu criador.

A ideia é simples: qualquer um pode desenvolver um sistema baseado em *blockchain* e executá-lo usando o próprio ambiente do *Ethereum*, sem precisar lidar com os custos de desenvolver o seu próprio. Já existem vários sistemas que utilizam o *Bitcoin* dessa maneira, porém são limitados pelas capacidades reduzidas dos scripts deste (existe também a possibilidade de se usar meta-protocolos, porém estes têm sérios problemas de escalabilidade e perdem grande parte da segurança inata a um *blockchain*).

O *Ethereum* é, portanto, uma plataforma dedicada ao desenvolvimento e implantação de aplicações descentralizadas de alta confiabilidade. Em seu âmbito,

tais aplicações são conhecidas como *Smart Contracts* (contratos inteligentes, em inglês). O *Ethereum* utiliza um *blockchain* para executar esses contratos de maneira inviolável, uma vez que todos os nós da rede deverão entrar em consenso sobre o resultado de cada computação. Para participar da sua rede, cada nó deve possuir uma implementação da *Ethereum Virtual Machine* (EVM), uma máquina virtual Turing completa definida por Gavin Wood.

O sistema também implementa uma criptomoeda chamada *Ether,* que é usada tanto para compensar os nós por computações realizadas como para realizar a transferência de valores entre contas. Em 2016, em virtude de uma falha no projeto *The DAO*, o sistema sofreu um *hard fork* que o separou em dois *Blockchains*, o principal que continua sendo chamado apenas de *Ethereum* e um paralelo que é chamado de *Ethereum Classic.*

3.1 - Conceitos básicos

3.1.1 - Gas

A fim de evitar ataques de negação de serviço e outros tipos de *spam*, o Ethereum definiu um mecanismo de preço interno chamado *gas* (abreviação de gasolina em inglês). De maneira simplificada, ele representa o preço que se está disposto a pagar por cada computação realizada. Um passo computacional simples deve custar 1 *gas*, enquanto operações mais computacionalmente complexas devem custar valores maiores. Além disso, também é cobrada uma taxa de 5 *gas* por cada byte nos dados da transação.

Sempre que algum pedaço de código vai ser executado, a parte que está solicitando tal execução deve estabelecer a quantidade máxima de unidades de *gas* que está disposta a utilizar e qual o valor em *Ether* que irá pagar por cada unidade gasta. Parte básica da verificação inicial consiste em checar se o solicitante de fato possui a quantidade necessária de moeda e subtraí-la de sua conta para pagar as taxas de transação (isto é, o custo do *gas*). Se for necessário mais *gas* que o enviado para executar o código, os efeitos deste serão revertidos, mas o solicitante

perde o valor das taxas de transação. Por outro lado, se sobrar *gas*, o valor em excesso será devolvido ao solicitante no final da transação.

É importante ressaltar que cabe a cada nó decidir se aceita o valor do *gas* proposto, de modo que este acaba sendo definido pelas leis de oferta e demanda, já que um valor muito baixo seria ignorado e um muito alto seria desperdício de dinheiro.

Desse modo, fica inviável que um eventual atacante tente soterrar o sistema com pedidos de computação inúteis, já que perderia muito dinheiro com isso, ou seria ignorado se oferecesse muito pouco pelo valor do *gas*. Além disso, *loops* infinitos são automaticamente barrados já que não é possível enviar *gas* infinito.

3.1.2 – Contas

Os objetos básicos que compõem o estado do sistema são as **contas**, sendo as transições de estado basicamente a transferência de valor e informação entre contas. Existem dois tipos delas: contas externas (controladas por quem possuir a chave privada apropriada); e contas de contrato (controladas pelo seu próprio código).

As contas possuem 4 campos:

- Um *nonce*, para garantir que não se processe a mesma transação várias vezes;
- A atual quantidade de *Ether* possuída pela conta;
- O código do contrato, se houver;
- O armazenamento da conta (inicialmente vazio).

Uma conta externa não tem código interno e pode enviar transações de acordo com o controle do seu usuário. Uma conta de contrato por outro lado deve conter um código e irá executá-lo sempre que receber uma transação ou mensagem. Tal execução pode ler e escrever no seu armazenamento (em pares de chave-valor), enviar novas mensagens, criar novos contratos, dentre outras coisas.

3.1.3 - Transações

No *Ethereum*, uma transação é a forma como uma conta externa envia um pacote de dados a outras contas (sejam externas ou de contrato). Cada uma contém os seguintes campos:

- O destinatário.
- A assinatura do remetente.
- A quantidade de *Ether* a ser transferida do remetente ao destinatário.
- Um campo de dados opcional.
- A quantidade de *gas* enviada.
- O preço do *gas* enviado.

Como explicado anteriormente, o *gas* serve para determinar o custo das computações solicitadas. O campo de dados pode ou não conter alguma informação que será acessada por um contrato (pode ser imaginado como argumentos de entrada passados a um programa na sua inicialização ou *inputs* do usuário durante sua execução).

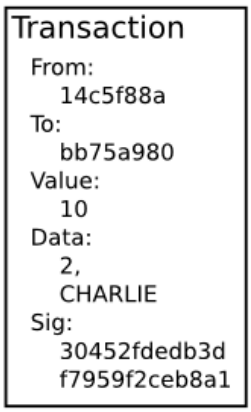

Figura 4: Um exemplo do formato de uma transação (sem a informação do gas*)*

3.1.4 - Mensagens

Enquanto contas externas se comunicam por transações, contas de contratos se comunicam (entre si) por mensagens. Estas são objetos virtuais que existem apenas no ambiente de execução do *Ethereum*. Cada uma contém:

- O remetente.
- O destinatário.
- A quantidade de *Ether* a ser transferida.
- Um campo de dados.
- A quantidade de *gas* enviada.

Desse modo, uma transação ou uma mensagem chega a uma conta de contrato, seu código é executado e este potencialmente pode ou não enviar uma mensagem à outra conta de contrato que por sua vez reagirá da maneira definida por seu código e assim sucessivamente, até que a sequência de execuções termine ou até que a quantidade de *gas* enviada pela transação original se esgote.

O preço do *gas* não está incluso, porque uma conta de contrato não “gera” *gas*, mas sim o recebe, seja de uma conta externa por transação (esta sim gera *gas*) ou de outra conta de contrato por mensagem (que por sua vez recebeu de uma transação ou de outra mensagem e assim sucessivamente).

3.2 - Transição de estado

Uma mudança de estado, necessariamente deve começar com o envio de uma transação, já que as contas de contrato só executam seu código quando são “cutucadas”. No processamento da transição, seguem-se os seguintes passos:

- Verifica-se a transação que deu início ao processo. Ela deve ter todos os campos corretamente, sua assinatura deve ser válida e o *nonce* da transação deve coincidir com o *nonce* do remetente. Caso contrário, já ocorreu algum erro.

- A taxa de transação é calculada multiplicando-se a quantidade de *gas* enviada pelo seu preço. O valor desta taxa é subtraído da conta do remetente e o *nonce* deste é incrementado (para evitar que a mesma transação seja processada duas vezes, motivo pelo qual o *nonce* foi verificado no passo anterior). Caso o remetente não tenha *Ether* o bastante para pagar a taxa, ocorreu um erro.

- Inicializado o contador com a quantidade disponível de *gas*, já se subtrai a taxa a ser paga pelo total de bytes na transação.

- Transfere-se o valor da transação do remetente ao destinatário. Se a conta destinatária ainda não existir, será criada agora. Se o remetente não possuir o valor a ser transferido, revertem-se todas as mudanças e encerra-se o processo, mas o minerador do bloco ficará com as taxas de transação (isto é, o valor do *gas* enviado, que não será devolvido).

- Se o destinatário for uma conta de contrato, executa-se o seu código até que este retorne, ou até que a quantidade de *gas* se esgote. Se o *gas* se esgotar antes do fim, também se revertem todas mudanças, exceto pelas taxas que ficam com o minerador.

- Se a execução terminar normalmente, o minerador fica com o valor do *gas* que foi gasto nas computações e o restante é devolvido ao remetente.

Cabe destacar que o passo de execução pode ser bastante longo, já que a execução do código do contrato pode enviar mensagens para outros contratos, que irão executar os seus códigos e assim por diante, até que tudo se encerre, ou até que o *gas* acabe. Outro detalhe importante é que caso este esgotamento ocorra durante uma computação por efeito de uma mensagem, apenas os efeitos daquela mensagem serão necessariamente revertidos, já que a conta que enviou a mensagem pode ainda ter mais *gas* que não havia sido enviado.

3.3 - A *Ethereum Virtual Machine*

Para garantir que todos os contratos sejam executados na mesma maneira em qualquer nó, cada um destes deve possuir uma implementação da EVM definida no *Yellow Paper* de Gavin Wood. Embora existam várias linguagens de programação disponíveis para o *Ethereum*, todas são compiladas para a mesma linguagem de *bytecode* baseada em pilhas conhecida como *EVM code*. Sua operação é bastante semelhante à de computadores modernos, onde se tem uma série de instruções básicas em sequência, um *program counter* (pc) que aponta para a próxima instrução a ser executada e é incrementado ao final de cada instrução e a execução segue até que: as instruções acabem (fim do programa); até que se detecte um erro (falha no programa); ou se encontre uma instrução de parada ou de retorno (programa concluído ou interrompido).

Porém, diferentemente de computadores modernos, a EVM não tem registradores, mas apenas três tipos de espaços onde guardar informações:

- Uma pilha com as operações típicas de *push* e *pop*.
- A memória, que é organizada como um *array* de bytes que teoricamente pode ser expandido indefinidamente.
- O armazenamento do contrato (o campo possuído por todas as contas, onde pares chave-valor são guardados).

Os dois primeiros são temporários (como a memória RAM de um computador) e *resetados* ao final de cada computação. Já o último é persistente até que seja mudado (equivalente ao disco rígido de um computador).

O *EVM code* possui as instruções típicas de linguagens de *assembly* para operações matemáticas básicas, comparações e lógicas bit a bit, armazenamento na memória, uso da pilha e movimentação do *program counter (jumps).* Além destas, também possui instruções para determinar a quantidade de *gas* disponível e o seu preço, para enviar mensagens para outra conta, para criar uma nova conta e para obter vários tipos de informações sobre o ambiente, como por exemplo: o bloco atual (informações do cabeçalho); o código de uma conta (podendo inclusive copiá-lo para a memória); a transação ou mensagem que iniciou a atual execução (remetente,

balanço e etc); e o valor enviado. Existe também uma instrução especificamente para aplicar o algoritmo de *hash* criptográfico SHA3 e uma variedade de instruções para propósitos de *log* (isto é, registrar eventos ao longo da execução do programa). Ao final, a execução deste código pode retornar um *array* de bytes como saída.

3.4 - Mineração no *Ethereum*

A mineração no *Ethereum* consiste em acumular uma certa quantidade de transações, limitadas por valor de *gas* máximo que cada bloco pode ter e aplicá-las, obtendo os próximos estados. O *blockchain* do *Ethereum* possui a particularidade de guardar todo o estado atual em cada bloco, embora utilize recursos para minimizar o espaço gasto com isso.

O cabeçalho de um bloco no *Ethereum* possui, além das informações básicas, as raízes de três árvores de Merkle: a primeira, como no *Bitcoin*, para armazenar todas as transações contidas no bloco; a segunda contendo o estado atual, que utiliza uma estrutura modificada conhecida como árvore Patricia; a terceira e última é uma árvore de recibos que, na prática, são informações mostrando o efeito de cada transação.

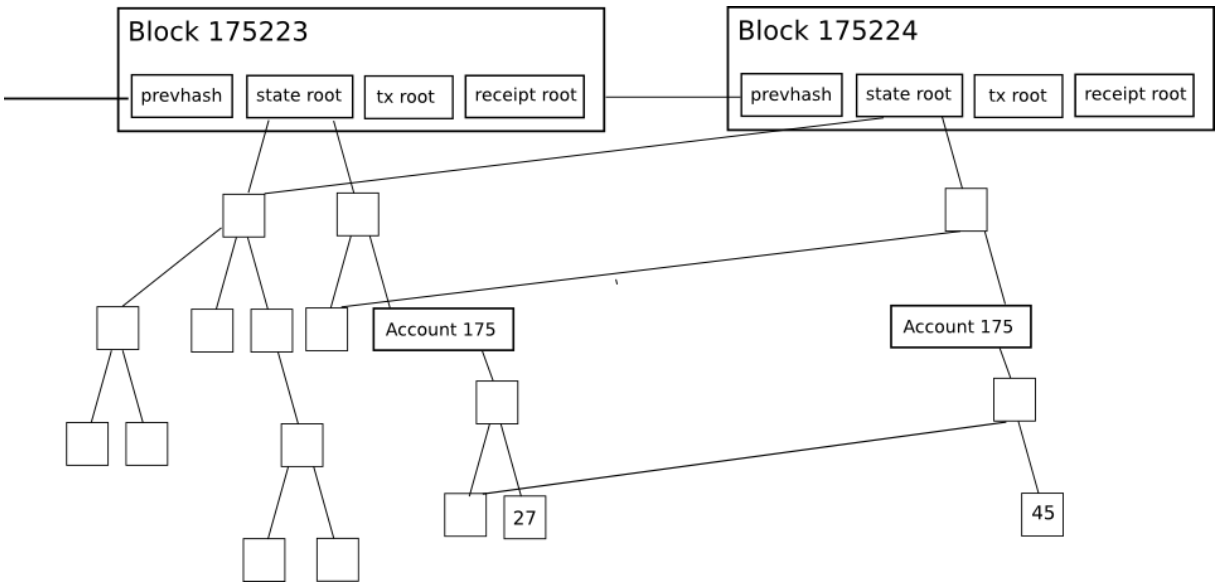

Figura 5: As 3 raízes nos cabeçalhos do Ethereum

O *Ethereum* ainda utiliza a Prova de Trabalho como mecanismo de consenso, porém existem planos para mudar para um sistema misto de Prova de Trabalho e Prova de Participação. O algoritmo de Prova de Trabalho usado é o *ethash*, que é bastante semelhante ao usado pelo *Bitcoin*, porém com algumas modificações que tornam mais difícil a criação de ASICs voltadas para o *Ethereum*.

Outra diferença importante está no fato de que a criação de blocos é bem mais rápida no *Ethereum*, sendo que o grau de dificuldade da mineração é ajustado para manter a taxa de criação em cerca de 12 a 15 segundos por bloco.

O processo de validação de um bloco inclui checar todas as computações desencadeadas por cada transação, de modo que no final, cada nó terá realizado praticamente o mesmo trabalho que o minerador (exceto a prova de trabalho, cuja verificação é mais simples), embora apenas este receba as taxas de transação e a recompensa pela mineração.

3.5 - Linguagens

Para desenvolver e escrever contratos no *Ethereum*, existem linguagens de mais alto nível que são compiladas para *EVM code* antes de serem incluídas em uma conta de contrato. Originalmente, haviam as 3 seguintes linguagens:

- Mutan, uma linguagem baseada em Go (linguagem de programação criada pelo Google em 2009) mas que acabou sendo abandonada em 2015;
- LLL (Lisp Like Language) que, como a própria sigla indica, é uma linguagem de mais baixo nível semelhante a Lisp. Embora pouco usada, continua sendo suportada pela versão atual;
- Serpent, baseada em Python, é de bem alto nível e bastante usada.

Posteriormente, foi criada uma quarta linguagem:

- Solidity, atualmente a mais utilizada, é uma linguagem orientada a objetos de alto nível baseada em C++ e Javascript. Foi desenvolvida de modo a poder facilmente substituir as demais, mas sem removê-las.

Nenhuma delas requer um conhecimento muito específico para ser utilizada, apenas familiarização com suas particularidades e com as chamadas básicas do *Ethereum*.

3.6 - *Smart Contracts*

A razão de existência do *Ethereum*, estes contratos inteligentes são basicamente *scripts* para transferências complexas de valor que podem conversar entre si. No seu *white paper*, *Vitalik Buterin* subdivide estes contratos em 3 tipos básicos: aplicações financeiras; aplicações parcialmente financeiras; e aplicações não financeiras.

Dentre as aplicações financeiras, estão todo tipo de coisa como fundos de investimentos e de cobertura, sistemas de bolsa de valores, derivativos e até mesmo contratos empregatícios.

O segundo tipo envolve qualquer situação na qual há uma troca de valor monetária, mas o sistema em si não é voltado para dinheiro. Um exemplo clássico é o pagamento de uma recompensa pela resolução de problemas computacionais.

Já o último tipo envolve casos como sistemas de votação e governança, que não incluem qualquer troca de valor monetário.

Qualquer um pode escrever um *Smart Contract* e colocá-lo no *Ethereum*, mas precisa ter dinheiro em *Ether* para poder pagar o valor do *gas* necessário para que este contrato seja executado.

3.6.1 - ICO

Um tipo particular de *Smart Contract* que se tornou bastante popular são os chamados ICOs (Initial Coin Offerings), uma variação baseada em criptomoeda dos IPOs (Initial Public Offerings).

Em uma IPO, uma empresa (até então privada) passa a vender publicamente ações que representam uma certa porcentagem da propriedade de si

mesma, com a intenção que investidores comprem tais ações, gerando assim capital para a empresa investir em seu crescimento. À medida que a empresa crescer, seu valor de mercado deve aumentar, aumentando assim o valor das ações (o que é lucrativo para os investidores) e assim sucessivamente, de acordo com as leis e regras do mercado de ações.

No ICO, uma empresa emite uma nova criptomoeda, a qual investidores podem comprar seja com dinheiro real ou com outra criptomoeda já estabelecida (dependendo do ICO). Assim, a empresa obtém capital para financiar seu crescimento e os investidores ganham moedas que, espera-se, irão valorizar à medida que a empresa tiver sucesso.

Curiosamente, o próprio *Ethereum* foi financiado por um processo desses, quando sua ICO angariou 18 milhões de dólares em *Bitcoin*, com um valor de 40 centavos de dólar por *Ether* (em novembro de 2017, o *Ether* já vale mais de 400 dólares, ou seja, um aumento de mais de 1000 vezes).

3.7 – *Hard Forks*

Todas as regras e protocolos acima citados estão sujeitos a mudanças. Quando uma alteração fundamental é incluída no *Ethereum*, diz-se que ocorreu um *hard fork*. O que isso significa, na prática é que a maioria concordou em seguir novas regras de um certo bloco em diante e qualquer um que tente utilizar a rede seguindo regras que não valem mais será ignorado pelo consenso da maioria.

Sempre que a *Ethereum Foundation* decide atualizar os protocolos usados, esse processo é realizado. Até outubro de 2017 o *Ethereum* já está em sua quarta versão (chamada *Metropolis vByzantium*).

Em alguns casos, é possível que um grupo suficientemente grande de usuários decida não participar do *fork* e continuem com uma rede separada própria que continua seguindo as antigas regras. Cabe destacar, porém, que não é possível que esta nova rede interaja diretamente com a original. O exemplo mais famoso é o do *Ethereum Classic* um *blockchain* que surgiu quando um grande grupo de usuários se recusou a participar do *hard fork* que reverteu as perdas do The DAO.

4 - A Segurança do *Ethereum*

4.1 - Confiança

Há várias razões para se preferir um *Smart Contract* no lugar de um contrato tradicional. Primeiramente, há a eficiência ganha quando os diversos resultados do contrato são calculados e executados instantaneamente, sem a necessidade de cartórios, advogados, transferências bancárias e etc. Mais importante, porém, está a segurança ganha em garantir que nenhuma das partes poderá faltar com as suas obrigações (tais como definidas no código do contrato), calotear outra das partes ou de alguma maneira deixar de cumprir o acordo estabelecido.

A vantagem principal de um *blockchain* é permitir a troca de informações entre partes que não se conhecem e não confiam umas nas outras, sem a necessidade de que haja uma entidade central que assume o papel de árbitro. Isto é desejável porque elimina tal árbitro como um ponto de vulnerabilidade na interação entre as partes, isto é, não é possível a nenhuma das partes subverter o árbitro em seu favor, já que este na verdade é o consenso da maioria.

O *blockchain* também permite uma vantagem quase paradoxal na qual, mesmo que as partes se mantenham anônimas entre si, ainda assim é possível garantir a identidade um do outro.

Uma vulnerabilidade que o *Ethereum* não é capaz de eliminar é a dos próprios códigos. Cada um é responsável pelos contratos que submete e dos quais participa. A falha humana é um fator impossível de ser eliminado nestes casos e eventualmente descobrem-se maneiras de explorar uma brecha no código. O caso mais famoso, a perda de cerca de 150 milhões de dólares no The DAO ocorreu por uma falha deste tipo.

Outro problema que tem ocorrido na rede é o de fraudes de investimento (esquemas Ponzi). Um contrato é criado para que as pessoas possam investir em algo que pareça promissor, elas incialmente parecem estar tendo grandes ganhos para que invistam mais e atraiam mais investidores e então o contrato acaba, alguém leva todo o dinheiro embora. Cabe destacar que nestes casos, não se trata de uma quebra do contrato como está codificado (isto continua não sendo possível),

mas sim de pessoas que entram em um contrato sem avaliar com cuidado possíveis brechas deixadas pelo criador para se aproveitar posteriormente.

A conclusão é que embora o *Ethereum* permita um grau de confiança muito mais elevado que o normal, ainda é importante lembrar-se do ditado russo *"Doveryai, no proveryai"* (Confie, mas verifique). Qualquer situação em que haja interação de pessoas trocando valor está sujeita a golpistas. O que a plataforma permite é que se codifiquem estas interações de maneira inviolável, mas a própria codificação é feita por pessoas e, portanto, não está isenta da possibilidade de falhas.

4.2 – Ataques à rede

Em se tratando dos contratos, não é possível, por exemplo, que um hacker invada o computador onde roda um programa bancário e roube dinheiro para si, já que existem N computadores rodando a EVM e seria necessário subverter no mínimo 51% deles para talvez obter algum resultado. Mesmo um ataque que conseguisse de alguma maneira enganar a maioria para obter consenso, ainda teria que lidar com a possibilidade de um *hard fork* que revertesse seu roubo.

Um ataque comum a sistemas de consenso é chamado ataque Sybil. Ele consiste em criar um grande número de identidades falsas para que essa "multidão" tente criar uma falsa maioria. É por isso que o *blockchain* implementa a Prova de Trabalho, que faz com que o consenso valorize trabalho efetivamente realizado. Mesmo que o atacante crie uma cadeia longa com informações inválidas e faça com que seus "clones" propaguem isso como se fosse o consenso, o trabalho total necessário para fazer tal subversão acaba sendo maior do que qualquer potencial recompensa, já que requereria um poder computacional e gasto de energia muito superior ao restante da rede.

Outra vulnerabilidade comum a sistemas em rede é a de sobrecarregar os canais de comunicação e os processadores com informação inútil, atrapalhando o funcionamento destes (ataques de negação de serviço). O *Ethereum* implementa o sistema de *gas* justamente para evitar este tipo de ataque, impondo um custo *a priori*

para qualquer envio de informação, além de um custo por operação computacional. Se o *gas* se esgotar, o minerador fica com toda a taxa e não terá desperdiçado seu tempo e energia em troca de nada, ao passo que a pessoa que enviou a solicitação não obterá qualquer resultado e terá perdido dinheiro. Mesmo assim, houve um caso no qual o custo computacional de várias instruções de escrita na memória foi subestimado e atacantes maliciosos se aproveitaram disso. Por um custo em *gas* relativamente baixo, enviaram as mesmas operações demoradas muitas e muitas vezes, sobrecarregando os nós que tentavam processá-las. Para corrigir isto foi necessário um *hard fork* que aumentou o custo em *gas* destas operações.

Além disso, mesmo sem um ataque malicioso o *Ethereum* ainda está limitado pela capacidade computacional de seus nós (o chamado problema da escalabilidade), de modo que se houver uma quantidade muito grande de transações, mesmo que bem-intencionadas, pode ocorrer um certo grau de lentidão indesejável. Problemas de escalabilidade como este são enfrentados por quaisquer *blockchain* que se tornem suficientemente grandes e recentemente (25 de novembro de 2017) *Buterin* apresentou algumas novas propostas para lidar com isso, mas que ainda não foram totalmente validadas e nem implementadas.

4.3 - Criptografia

Por envolver dinheiro, é natural que se preocupe com roubos. Cada conta possui seu saldo, e a criptomoeda já atingiu valores superiores a 400 dólares por *Ether*. Para garantir que apenas o dono de uma conta irá gastar seu conteúdo, usa-se o sistema de chave pública-privada. De uma maneira simplificada, cada conta externa é identificada por uma chave pública ao passo que seu “dono” deve possuir a chave privada equivalente. Sempre que alguém emite uma transação, esta deve conter uma assinatura gerada com a chave privada correta, o que permite que se verifique, usando a chave pública conhecida, que o possuidor da chave foi quem enviou aquela assinatura.

O *Ethereum* usa o algoritmo ECDSA (*Elliptic Curve Digital Signature Algorithm*, ou Algoritmo de Assinatura Digital de Curva Elíptica em português), uma variante do DSA. Para uma chave pública de 512 bits, como a usada pelo *Ethereum*,

esse algoritmo faz com que sejam necessárias no máximo 2^{256} operações (da ordem de 10^{77}) para se descobrir a chave privada, ou seja, é inviável descobrir por força bruta (e se fosse, valeria mais a pena gastar tal poder computacional para minerar blocos do que para roubar *Ether*). Naturalmente, isso não elimina a possibilidade de falha humana (permitir que se roube a sua chave privada) e nem há garantias de que o algoritmo seja inquebrável, mas se isto acontecer, um *hard fork* ocorreria para passar a utilizar outro tipo de encriptação.

Um dos principais elementos estruturais do *blockchain* são os *hashes*. Deles depende o encadeamento entre os blocos, a integridade das informações nas árvores de Merkle e as Provas de Trabalho. Tudo isso depende do fato dos algoritmos utilizados (quase todos são variações do SHA) serem suficientemente resistentes a serem quebrados. Até o presente momento, ao longo da elaboração deste trabalho, tais algoritmos ainda se mantém viáveis, mas é sempre possível que ocorra alguma nova descoberta criptográfica.

Outra possibilidade no horizonte é o desenvolvimento de computadores quânticos capazes de deixar grande parte destes algoritmos “comendo poeira”. Naturalmente, já existem estudos sobre algoritmos de criptografia pós-quântica, mas cabe à rede implementar este tipo de algoritmo no momento apropriado, ou serem derrubados pela inovação tecnológica.

4.4 – Centralização

É irônico que este seja um problema enfrentado pelo *Ethereum* quando um *blockchain* é, por sua natureza, direcionado a descentralização. Entretanto, diferentemente do criador do *Bitcoin*, *Vitalik Buterin* e a Ethereum *Foundation* não se afastaram do projeto uma vez concluído. Pelo contrário, continuam sendo os principais desenvolvedores e “guardiões” do *Ethereum*. O que isso ocasiona é um grau de concentração de poder em *Buterin* e na fundação que é visto por alguns “puristas do *blockchain*” como uma violação do princípio da descentralização. O exemplo mais claro, argumentam, foi o *hard fork* que reverteu as perdas do *The DAO*, projeto encabeçado por membros da fundação próximos a *Buterin*. O fato de perdas de outros projetos terem sido ignoradas ao passo que essa recebeu

tratamento especial é visto como a prova de que a fundação exerce um alto grau de controle sobre a rede que deveria ser autônoma e descentralizada.

Não surpreendentemente, o *Ethereum* possui um grande número de redes derivadas privadas, uma vez que muitos negócios perceberam a utilidade de uma rede deste tipo, porém não estavam dispostos a se submeter às vontades da fundação, decidindo então fazer seus próprios *forks* e fechá-los. Os principais críticos são justamente aqueles que acabaram criando o *Ethereum Classic* que ainda é aberto, mas que não aceita que a fundação tenha esse poder.

Além disso, *Buterin* acabou gerando um certo culto de personalidade ao seu redor, o que acaba lhe dando um poder extra de influenciar os usuários do *Ethereum*. Ele até mesmo já afirmou "brincando" que era um ditador benevolente do sistema.

4.5 – Questões econômicas

O fato de ser uma criptomoeda não torna o *Ethereum* imune aos problemas inerentes a qualquer sistema econômico. Especulação e trocas de informação privilegiada são alguns exemplos de problemas que ele já enfrenta.

Recentemente, houve uma explosão no número de ICOs implementadas no *Ethereum* (a ponto de alguns desenvolvedores afirmarem jocosamente que as aplicações distribuídas foram esquecidas). *Startups* têm conseguido milhões de dólares em *Ether* para se financiar, a ponto de não precisarem gastar tudo que ganham. Como o *Ether* está valorizando, vale a pena para elas guardarem um pouco deste em suas contas, o que aumenta ainda mais o valor deste e incentiva mais empresas a acumularem a criptomoeda. Alguns economistas identificam nisso maus sinais, indícios de que uma bolha está crescendo e pode estourar em breve. Basta que uma destas *Startups* quebre, ou de alguma maneira falhe em fazer valer o dinheiro investido e, de repente, todo mundo pode querer liquidar seu *Ether* para algo mais certo, o que faria o preço despencar vertiginosamente.

Os críticos da centralização inclusive apontam isso como sendo culpa de Buterin e da fundação, que vêm incentivando e aconselhando alegremente todas

estas ICOs, mesmo quando muitas delas se mostram mais gananciosas do que precisariam ser.

5 – O Caso do The DAO

5.1 – Histórico

Uma das muitas aplicações do *Ethereum* é a criação de organizações autônomas descentralizadas (em inglês, *descentralized autonomous organization* ou DAO). Na prática, codificam-se as regras e o aparato de decisões de uma organização (através de *Smart Contracts*), eliminando a necessidade de papelada e administradores, criando uma estrutura com controle descentralizado. Uma vez codificada, há um período de ICO no qual a organização recebe fundos à medida que prospectivos participantes compram fichas que representam sua participação na organização. Terminado esse período de financiamento, a organização começa a operar, sendo que os participantes da organização utilizam suas fichas para votar em diferentes propostas de como gastar o dinheiro da organização.

Várias destas já foram desenvolvidas, mas houve uma que ganhou bastante proeminência e acabou ficando conhecida como **The** DAO. Idealizada como um fundo de investimento de capital de risco, seu período de financiamento começou em 30 de abril de 2016, programado para terminar em 28 dias depois. Em menos de 15 dias, The DAO já tinha mais de 100 milhões de dólares em fundos e ao final do período de financiamento, tinha um saldo de mais de 150 milhões e mais de 11 mil participantes. Ainda em maio, um paper foi publicado por Dino Mark, Vlad Zamfir, e Emin Gün Sirer onde estes apontavam várias falhas com potencial para serem exploradas por alguém mal-intencionado e pediam que o início das operações fosse adiado.

Entretanto, mesmo enquanto estas vulnerabilidades e potenciais soluções para elas eram discutidas pela comunidade, The DAO entrou em operação. No dia 17 de junho de 2016 um ataque foi realizado que combinava várias das falhas já identificadas e foi capaz de retirar 3,6 milhões de *Ether* (o que na época correspondia a um valor de cerca de 50 milhões de dólares) da organização. Uma particularidade do código fez com que este dinheiro tivesse que esperar em uma conta separada por 28 dias, período durante o qual a comunidade debateu o que fazer. Em todo caso, o preço do *Ether* despencou, fato do qual o atacante pode ter

se aproveitado apostando contra a valorização da criptomoeda, lucrando independentemente de poder ficar com o dinheiro roubado ou não.

Houveram aqueles que argumentaram que embora fosse um ataque antiético, não era ilegal já que estava habilitado pelo código, outros queriam que o dinheiro fosse devolvido e outros queriam que a organização fosse desfeita. Por fim, em 20 de julho de 2016 houve a decisão por parte da *Ethereum Foundation* em fazer um *hard fork* que reverteu os efeitos do roubo, basicamente rebobinando esta parte do *blockchain*. Uma grande parcela de usuários considerou isso inaceitável e um grupo deles decidiu ignorar o *fork* e manter a cadeia anterior, que passou a ser conhecida como *Ethereum Classic*.

5.2 – Como funcionava o The DAO

Durante o período de financiamento, investidores compraram *The DAO Tokens* (TDT), as fichas que representavam sua participação na organização. O contrato principal, onde estes fundos foram investidos funcionava como uma fábrica de subcontratos (conhecidos com *child-DAO*s, ou DAOs filhos, em português).

Toda instância do The DAO tinha um curador, uma conta (que poderia ou não ser controlada por vários usuários) e que era responsável por vetar propostas mal-intencionadas submetidos à organização (por exemplo, um bloco com mais de 50% dos TDT que submetesse uma proposta que beneficiaria apenas a seus membros). Na instância principal, o curador era composto por 11 usuários.

Uma vez que uma proposta passasse pelo crivo do curador, ela era incluída na lista de votação (pagando uma taxa por isso, a fim de evitar *spam*). Os participantes então votariam sim ou não na proposta (com o período de votação mínimo de 14 dias), de acordo com a sua avaliação pessoal se aquele era um investimento que valia a pena. Cada voto tinha seu peso determinado pela quantidade de TDT possuída pelo votante. Ao final, se a maioria votasse sim e o quórum mínimo fosse atingido (este variando entre 20 e 53%, de acordo com o tamanho da proposta), a proposta era aprovada.

A fim de evitar um fluxo muito rápido de entrada e saída de fundos, o procedimento para deixar o The DAO era um pouco complexo. O participante deveria propor a criação de uma *child-DAO* tendo apenas a si mesmo como curador. A proposta em si não importa já que, independentemente de qualquer votação, passados um mínimo de 7 dias, aqueles que votaram sim ganham o direito de se separar da The DAO e transferir seus fundos para a *child-DAO* criada. Feito isso, é necessário esperar o período de 27 dias de financiamento inerente ao contrato original, ou seja, o processo todo leva no mínimo 34 dias. Passado todo esse tempo, o novo curador da sua própria DAO submete uma proposta para pagar todo o balanço desta para sua conta externa, reobtendo finalmente seu investimento.

Foi no meio desse procedimento extremamente complexo que o principal furo foi encontrado. Denominado *Stalking Attack* (ataque de perseguição, em inglês), ele se aproveita do fato de que a *child-DAO* criada para retirar os fundos entra em um período de financiamento como qualquer outra. O atacante pode, portanto, comprar TDTs dessa organização filha e, se obtiver pelo menos 53% das fichas, será capaz de bloquear qualquer proposta do curador para extrair seus próprios fundos. O atacante também não pode pegar o dinheiro para si, já que teoricamente o curador não aprovaria tais propostas, mas é nesse ponto que o erro humano beneficia o atacante, já que o processo todo é muito confuso. Mesmo que a vítima perceba o que aconteceu e tente sair desse subcontrato (criando um sub subcontrato, que levaria mais 34 dias para ter resultado) é extremamente fácil para o atacante persegui-lo, repetindo o processo, ao passo que a vítima precisa passar pelo custoso processo de criar uma *child-DAO* que provavelmente será bloqueada novamente. Isso permite chantagens e pedidos de resgate por parte do atacante.

O ataque que resultou na perda dos 50 milhões foi uma combinação de várias vulnerabilidades, incluindo esta, mas todas são produto de falhas na organização lógica do The DAO. Não foram *bugs* no código nem no *Ethereum*, nem quebras criptográficas, mas sim apenas uma falha dos programadores em antecipar execuções maliciosas do sistema que criaram.

6 – Conclusão

Não é sem motivo que a tecnologia *blockchain* causou tanto furor. A possibilidade de se automatizar de maneira segura a interação entre partes que não necessariamente confiam umas nas outras é valiosíssima. Entretanto, não se pode esquecer que a tecnologia está constantemente evoluindo e o que é uma solução fenomenal hoje pode se tornar inútil amanhã. Além disso, o fator humano continua sendo um ponto vulnerável constante em todos estes sistemas. Afinal, não importa quantas camadas de abstração e protocolos de comunicação, no final temos pessoas por trás de tudo, codificando os contratos, procurando maneiras de violá-los e ganhar dinheiro com isso, tentando se manter às suas convicções ou se deslumbrando com a fama.

As perspectivas são bastante positivas em se tratando da quantidade de soluções em potencial que podem ser criadas no *Ethereum*, mas este ainda precisa amadurecer em vários aspectos para se tornar a plataforma que ambiciona ser. Por enquanto, o *Bitcoin* continua sendo a aposta mais segura para quem quiser investir em criptomoedas e sistemas baseados em *blockchain*.

Referências Bibliográficas

1. TAPSCOTT, Don; TAPSCOTT, Alex. ***Blockchain Revolution*: Como a tecnologia por trás do *Bitcoin* está mudando o dinheiro, os negócios e o mundo.** São Paulo: SENAI-SP Editora, 2016.

2. BAYER, Dave; HABER Stuart; Stornetta W. Scott. **Improving the Efficiency and Reliability of Digital Time-Stamping.**
Disponível em: https://link.springer.com/chapter/10.1007/978-1-4613-9323-8_24

3. Vários autores. **Bitcoin Developer Reference**
Disponível em: https://bitcoin.org/en/developer-reference

4. BUTERIN, Vitalik. **White Paper.**
Disponível em: https://github.com/Ethereum/wiki/wiki/White-Paper

5. WOOD, Gavin. **Yellow Paper.**
Disponível em: https://Ethereum.github.io/yellowpaper/paper.pdf

6. Whale Panda. **I was wrong about Ethereum.**
Disponível em: https://medium.com/startup-grind/i-was-wrong-about-Ethereum-804c9a906d36

7. MARK, Dino; ZAMFIR, Vlad; GÜN SIRER, Emin. **A Call for a Temporary Moratorium on The DAO.**
Disponível em: http://hackingdistributed.com/2016/05/27/dao-call-for-moratorium/

8. ROSIC, Ameer. **What is Ethereum? A Step-by-Step Beginners Guide.**
Disponível em: https://blockgeeks.com/guides/what-is-Ethereum/

9. Vários autores. **Introduction to Smart Contracts.**
Disponível em: http://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html

10. NARAYANAN, Arvind; BONNEAU, Joseph; FELTEN, Edward; MILLER, Andrew; GOLDFEDER, Steven. Bitcoin and Cryptocurrency Technologies. Disponível em: https://d28rh4a8wq0iu5.cloudfront.net/bitcointech/readings/princeton_bitcoin_book.pdf

11. LEWIS, Antony. **A gentle introduction to blockchain technology.** Disponível em: https://bitsonblocks.net/2015/09/09/a-gentle-introduction-to-blockchain-technology/

12. BAUERLE, Nolan. **What is the Difference Between a Blockchain and a Database?** Disponível em: https://www.coindesk.com/information/what-is-the-difference-blockchain-and-database/

13. BUTERIN, Vitalik. **Merkling in Ethereum**. Disponível em: https://blog.Ethereum.org/2015/11/15/merkling-in-Ethereum/