# Ergo: Uma Plataforma Resiliente Para Dinheiro Contratual

Desenvolvedores de Ergo, https://ergoplatform.org/

14 de Maio de 2019
v1.0

**Resumo**

Apresentamos Ergo, um novo e flexível protocolo blockchain. Ergo é projetada para o desenvolvimento de aplicações descentralizadas com o foco principal em prover um jeito eficiente, seguro e fácil de implementar contratos financeiros.

Para atingir este objetivo, Ergo inclui várias melhorias técnicas e econômicas a soluções blockchain existentes. Toda moeda em Ergo é protegida por um programa em ErgoScript, que é uma linguagem de programação poderosa, amigável a protocolos, e baseada em Protocolos-Σ. Ao usar ErgoScript, podemos codificar as condições sobre as quais moedas podem ser usadas: quem pode gastá-las, quando, sob quais condições externas, para quem, e assim por diante.

Suporte estendido para nós leves torna Ergo amigável para usuários finais, porque permite a execução de contratos em *hardware* comuns não-confiáveis. Para ser usável no longo-prazo, Ergo segue a abordagem de sobrevivência – usa soluções amplamente pesquisadas na academia que não resultarão em riscos de segurança no futuro, enquanto também prevenindo degradação de performance ao longo do tempo com um novo modelo econômico. Finalmente, Ergo possui um protocolo auto-alterável que permite a absorção de novas ideias e auto-melhoria no futuro.

## 1 Introdução

Começando mais de dez anos atrás com Bitcoin [1], tecnologia blockchain tem até agora provado ser uma forma segura de publicamente manter um livro-razão de transações e eliminar, até certo ponto, intermediários, como instituições financeiras tradicionais. Mesmo após atingir um valor de mercado de U$300 bilhões em 2017 [2], nenhum ataque severo foi realizado na rede Bitcoin apesar do alto potencial de retorno. Esta resiliência das criptomoedas e o empoderamento financeiro e auto-determinação que elas prometem trazer é alcançada por uma combinação de algoritmos criptográficos modernos e arquitetura descentralizada.

Porém, essa resiliência vem com um custo e ainda não foi provada para sistemas existentes no longo-prazo em uma escala econômica global. Para usar uma blockchain sem confiar em terceiros, os participantes da rede devem verificar uns aos outros ao fazer o *download* e processamento de todas as transações na rede, utilizando recursos da rede. Além de utilização da rede, o processamento de transações também utilizam recursos computacionais, especialmente se a linguagem transacional é suficientemente flexível. Finalmente, os participantes da blockchain precisam manter uma significativa quantidade de dados em seu armazenamento local e os requisitos de armazenamento estão crescendo rapidamente. Além disso, certos dados devem ser mantidos em memória. Então, o processamento de transações utiliza vários recursos de centenas de milhares de computadores espalhados pelo mundo e o consumo desses recursos é pago pelos usuários regulares na forma de taxas de transações [3]. Apesar do generoso subsídio de recompensa de bloco em alguns sistemas existentes, essas taxas ainda podem ser muito altas de tempos em tempos [4]. Devido a isso, mesmo após existir por mais de dez anos, tecnologia blockchain ainda está primariamente sendo usada em aplicações financeiras, onde a vantagem da alta segurança supera a desvantagem dos altos custos de transação.

Apesar do exemplo básico de moeda, o outro uso de blockchains é o da criação de aplicativos descentralizados. Tais aplicativos utilizam a habilidade da plataforma fundamental para escrever contratos inteligentes [5] implementando sua lógica através de uma linguagem de programação específica para blockchains. Uma maneira de classificar blockchains em termos de sua habilidade de escrever contratos inteligentes é baseado em se elas são *baseadas em UTXO* (e.g., Bitcoin) ou *baseadas em contas* (e.g., Ethereum) [6]. Criptomoedas baseadas em Contas , como Ethereum, introduz contas especiais para contratos, controladas por códigos de programação, que podem ser invocadas por novas transações que entram na rede. Embora a abordagem permitir computações arbitrárias, a implementação de condições de gasto complexas podem levar a *bugs* nos programas, como um em um "simples" contrato multi-assinatura em Ethereum que causou uma perda de U$150 milhões em 2017 [7]. Em criptomoedas baseadas em UTXO, toda moeda possui um *script* associado e, para gastar aquela moeda, deve-se satisfazer as condições estabelecidas no *script*. Implementar tais condições de proteção é muito mais fácil com o modelo UTXO, mas realizar uma computação arbitrária que seja Turing-completa é bastante complexo [8]. Porém, a maioria dos contratos financeiros não requer Turing-completeza [9]. Ergo é baseada no modelo UTXO e fornece uma maneira conveniente para implementação de aplicativos financeiros que cubram uma esmagadora maioria de casos de uso de blockchains públicas.

Embora o componente contratual seja importante para construção de aplicações descentralizadas, também é essencial que a blockchain sobreviva no longo-prazo. Plataformas blockchain com foco em aplicações possuem somente alguns poucos anos de existência, já que a área como um todo ainda é recém-criada. Como tais plataformas já encontraram problemas com degradação de performance ao longo do tempo [10, 11], sua sobrevivência de longo-prazo é questionável. Até mesmo blockchains baseadas em UTXO e com foco em ser meio de troca

ainda não provaram ser completamente resilientes no longo-prazo sob condições dinâmicas, porque temos apenas 10 anos de histórico até este momento. Soluções para a sobrevivência de longo-prazo incluem conceitos tais como nós leves com requisitos mínimos de armazenamento [12], componente de taxa de aluguel de armazenamento para prevenção o inchaço dos nós completos [3], e protocolos auto-alteráveis que podem se adaptar ao ambiente dinâmico e melhorar a si mesmos sem intermediários [13]. O que é preciso é uma combinação de várias ideias científicas para consertar esses problemas, enquanto também provendo um jeito para mais melhorias sem quaisquer mudanças drásticas. E é exatamente isso que Ergo tenta fazer.

## 2 Visão Ergo

O protocolo Ergo é muito flexível e pode ser mudado no futuro pela comunidade. Nesta seção, definimos os princípios mais importantes que devem ser seguidos em Ergo e que podem ser referidos como o "Contrato Social de Ergo". No caso de violação intencional de qualquer um desses princípios, o protocolo resultante não deve ser chamado de Ergo.

- *Descentralização em Primeiro Lugar.* Ergo deve ser tão descentralizada quanto possível: qualquer parte (líderes sociais, desenvolvedores de *software*, fabricantes de *hardware*, mineradores, fundos, etc.) que o comportamento ausente ou malicioso pode afetar a segurança da rede deve ser evitada. Se qualquer uma dessas partes surgir durante a vida de Ergo, a comunidade deve considerar formas de diminuir o nível de impacto desses agentes.

- *Criada para Pessoas Comuns.* Ergo é uma plataforma para pessoas ordinárias e seus interesses não devem ser infringidos em favor de grandes partes. Em particular, isto significa que centralização de mineração deve ser prevenida e pessoas regulares devem ser capazes de participar no protocolo ao executar um nó completo e minerar blocos (embora com uma pequena probabilidade).

- *Plataforma para Dinheiro Contratual.* Ergo é a camada-base para aplicações que serão construídas em cima da rede. É apropriado para diversos aplicativos, mas seu foco principal é fornecer uma forma fácil, segura e eficiente para implementar contratos financeiros.

- *Foco no Longo-prazo.* Todos os aspectos do desenvolvimento de Ergo devem ser focados na perspectiva de longo-prazo. Em qualquer momento, Ergo deve ser capaz de sobreviver por séculos sem esperadas bifurcações duras, melhorias de *software* ou *hardware* ou alguma outra mudança inesperada. Como Ergo é projetada como uma plataforma, aplicativos construídos em cima de Ergo devem também ser capazes de sobreviver no longo-prazo. Essa resiliência e sobrevivência de longo-prazo pode também possibilar Ergo ser uma boa reserva de valor.

- *Sem Autorização e Aberta.* O protocolo Ergo não restringe ou limita qualquer categoria de uso. Ele deve permitir que qualquer um se junte à rede e participe no protocolo sem qualquer ação preliminar. Ao contrário do sistema financeiro tradicional, nenhum resgate, lista negra ou outras formas de discriminação deve ser possível no âmago do protocolo Ergo. Por outro lado, desenvolvedores de aplicativos estão livres para implementar qualquer lógica que queiram, assumindo responsabilidade pela ética e legalidade da sua aplicação.

# 3 Protocolo de Consenso Autolykos

O componente principal de qualquer sistema blockchain é seu protocolo de consenso e Ergo utiliza um protocolo de consenso Prova-de-Trabalho (PoW, sigla em Inglês) que é único, desenvolvido pelo time de Ergo e chamado de *Autolykos*. Apesar de extensa pesquisa em possíveis alternativas, o protocolo PoW original com a regra de cadeia mais longa ainda está em alta demanda devido a sua simplicidade, alta garantia de segurança e afabilidade a clientes leves. Porém, uma década de testes extensos revelou diversos problemas com a ideia original de "uma CPU, um voto".

O primeiro problema conhecido em um sistema PoW é o desenvolvimento de *hardware* especializado (ASICs, sigla em Inglês), que permite que um pequeno grupo de mineradores equipados com ASICs resolvam os quebra-cabeças de PoW ordens de magnitude mais rápida e eficientemente que qualquer um. Esse problema pode ser resolvido com a ajuda de esquemas PoW que usam bastante memória RAM e reduzem a disparidade entre ASICs e *hardware* comerciais. A abordagem mais promissora aqui é usar esquemas PoW assimétricos que utilizam basante RAM e que requerem significativamente menos memória para verificar uma solução do que para encontrá-la [14, 15].

A segunda ameaça conhecida à descentralização de uma rede PoW é que até mesmo grandes mineradores tendem a se unir em *pools* de mineração, levando a uma situação em que apenas alguns operadores de *pools* (5 em Bitcoin, 2 em Ethereum no momento da escrita deste artigo) controlam mais de 51% do poder computacional. Embora este problema já ter sido discutido na comunidade, nenhuma solução prática foi implementada antes de Ergo.

O protocolo PoW de Ergo, Autolykos [16], é o primeiro protocolo de consenso que é tanto intenso em uso de memória RAM quanto resistente à *pools*. Autolykos é baseado no *problema da soma dos subconjuntos*: um minerador tem que encontrar $k = 32$ elementos em uma lista $R$ pré-definida de tamanho $N = 2^{26}$ (que possui um tamanho de 2 Gb), tal que $\sum_{j \in J} r_j - sk = d$ está no intervalo $\{-b, \ldots, 0, \ldots, b \mod q\}$. Elementos da lista $R$ são obtidos como um resultado de uma computação de mão-única do índice $i$, duas chaves públicas $pk, w$ de mineradores e *hash* do cabeçalho do bloco $m$ como $r_i = H(i||M||pk||m||w)$, onde $H$ é uma função *hash* que retorna os valores em $\mathbb{Z}/q\mathbb{Z}$ e $M$ é uma grande mensagem estática que é usada para tornar mais lentos os cálculos de *hashs*. Além disso, um conjunto de índices de elementos $J$ deve ser obtido através

da função pseudo-aleatória e de mão única $genIndexes$, que previne possíveis otimizações de pesquisa de soluções.

Então, partimos do princípio que a única opção para um minerador é usar o simples método de força-bruta apresentado no Algoritmo 1 para criar um bloco válido.

**Algoritmo 1** Mineração de bloco

**Input**: próxima *hash* de cabeçalho de bloco $m$, par de chaves $pk = g^{sk}$
Gerar aleatoriamente um novo par de chaves $w = g^x$
Calcular $r_{i \in [0,N)} = H(i||M||pk||m||w)$
**while** $true$ **do**
    $nonce \leftarrow$ rand
    $J := genIndexes(m||nonce)$
    $d := \sum_{j \in J} r_j \cdot x - sk \mod q$
    **if** $d < b$ **then**
        **return** $(m, pk, w, nonce, d)$
    **end if**
**end while**

Note que, muito embora o processo de mineração utilize chaves privadas, a solução em si somente contém chaves públicas. Verificação da solução é feita pelo Algoritmo 2.

**Algoritmo 2** Verificação de solução

**Input**: $m, pk, w, nonce, d$
requer $d < b$
requer $pk, w \in \mathbb{G}$ e $pk, w \neq e$
$J := genIndexes(m||nonce)$
$f := \sum_{j \in J} H(j||M||pk||m||w)$
requer $w^f = g^d \cdot pk$

Essa abordagem impede a formação de *pools* de mineração, porque a chave secreta $sk$ é necessária para mineração: uma vez que qualquer minerador de *pool* encontre a solução correta, ele(a) pode usar essa chave secreta para roubar a recompensa de bloco. Por outro lado, é seguro revelar uma solução única, já que contém somente chaves públicas e revela uma única relação linear entre 2 segredos $sk, w$.

Uso intensivo de memória RAM vem do fato de que Algoritmo 1 requer a manutenção da lista $R$ inteira para a execução do principal *loop*. Cada elemento de lista ocupa 32 bytes, então a lista completa de $N$ elementos ocupa $N \cdot 32 = 2Gb$ de memória para $N = 2^{26}$. Um minerador pode tentar reduzir requisitos de memória ao calcular esses elementos "em tempo real" sem mantê-los em memória. Porém, o minerador precisa calcular a mesma *hash* $H$ múltiplas vezes (aproximadamente $10^4$ vezes em GPUs modernas), portanto reduzindo eficiência e lucro.

Calcular a lista $R$ também é uma tarefa computacionalmente bastante pesada: nossa implementação inicial [17] consome 25 segundos em uma Nvidia GTX 1070 para preencher todos os $2^{26}$ elementos da lista. Essa parte, porém, pode ser otimizada se um minerador também guardar uma lista de *hashes* $u_{i\in[0,N)} = H(i||M||pk)$ não finalizadas em memória, consumindo 5 Gigabytes a mais. Em tal caso, o trabalho para calcular *hashes* não finalizadas deve ser feito somente uma vez durante a inicialização da mineração, enquanto finalizá-los e preencher a lista $R$ para o novo cabeçalho consome somente alguns milissegundos (aproximadamente 50 ms em uma Nvidia GTX 1070).

O parâmetro-alvo $b$ está embutido no quebra-cabeças e é ajustado a atual *taxa de hash* da rede via um algoritmo de ajuste de dificuldade [18] para manter intervalos de tempo entre blocos perto de 2 minutos. Este algoritmo tenta prever a taxa de *hash* de uma próxima época de 1024 blocos baseado em dados das últimas 8 épocas via o conhecido *método de mínimos quadrados lineares*. Isso torna as previsões melhores do que aquelas de um algoritmo normal de ajuste de dificuldade e também torna ataques de "coin-hopping" menos lucrativos.

# 4 Estado de Ergo

Para verificar uma nova transação, um cliente de criptomoedas não usa o livro-razão com todas as transações que aconteceram antes. Ao invés disso, usa uma "fotografia instantânea" do "estado" do livro-razão a partir de seu histórico. Na documentação da implementação do Bitcoin Core, o que estão nesta fotografia são as moedas únicas ativas (i.e., UTXOs), e uma transação destrói algumas moedas e cria outras novas. Em Ethereum, essa fotografia se dá pelas contas de longa-duração e uma transação modifica o balanço monetário e armazenamento interno de algumas contas. Também em Ethereum, diferentemente de Bitcoin, a representação de uma fotografia é fixada dentro do protocolo, porque uma compilação de autenticação da fotografia é escrita no cabeçalho do bloco.

Ergo segue o *design* UTXO de Bitcoin e representa as fotografias usando moedas únicas. Ergo se diferencia de Bitcoin, além de valor monetário e *script* de proteção, ao incluir em suas moedas únicas (chamadas *box*) dados definidos pelos usuários. Similar a Ethereum, um bloco em Ergo também armazena uma compilação de autenticação, camada de *stateRoot*, do estado global após a adição do bloco.

Uma caixa em Ergo é feita de registros (e nada além de registros). Tal caixa pode conter 10 registros denominados $R_0, R_1, \ldots, R_9$, dos quais os quatro primeiros são preenchidos com valores obrigatórios e o resto pode conter dados arbitrários ou estarem vazios.

- $R_0$ *(valor monetário).* Quantidade de Erg travada nesta caixa.
- $R_1$ *(script guardião).* *Script* em série protegendo esta caixa.
- $R_2$ *(tokens).* Uma caixa pode carregar múltiplos *tokens*. Esses registro contém um vetor de pares $(identificador\_do\_token \rightarrow quantidade)$ travados nesta caixa.

- $R_3$ *(informação de transação).* Contém (1) altura de criação declarada (deve ser não mais do que a altura atual de um bloco que contenha a transação), (2) um identificador único da transação que criou esta caixa, e (3) o índice desta caixa da caixa de saída daquela transação.

- $R_4 - R_9$ *(dados adicionais).* Contém dados arbitrários definidos por usuários.

Objetos únicos imutáveis (como no modelo UTXO do Bitcoin) possuem algumas vantagens sobre as contas mutáveis e de longa-duração de Ethereum. Primeiro, dão proteção mais fácil e mais segura contra ataques de "replay" ou "reordenamento". Em segundo lugar, é mais fácil de processar transações em paralelo, pois elas não modificam o estado dos objetos que acessam. Além disso, uma transação ou modifica o estado do sistema exatamente como a previsto, ou simplesmente não modifica o estado do sistema (sem qualquer possibilidade de efeitos colaterais resultantes de exceções do tipo "sem balanço para taxas", questões de reentrada, etc). Finalmente, parece mais fácil construir clientes completamente sem estado usando moedas únicas [19] (muito embora pesquisas nesta área ainda estejam em seus estados iniciais).

Uma grande crítica das moedas únicas é que esse modelo não parece adequado para aplicações descentralizadas não-triviais. Porém, Ergo superou este problema e mostrou que essa afirmativa é falsa ao demonstrar muitos protótipos de aplicações não-triviais construídas usando Ergo (ver Seção 7).

O protocolo Ergo fixa a representação de fotografia do livro-razão na forma de caixas não destruídas pelas transações anteriores. Ou seja, um minerador deve manter uma estrutura de dados autenticados do tipo "árvore de Merkle" construída em cima do conjunto UTXO e deve incluir uma pequena compilação (apenas 33 bytes) dessa estrutura em cada cabeçalho de bloco. Essa compilação deve ser calculada *após* a adição do bloco. Essa estrutura de dados autenticada é construída em cima de uma árvore AVL+ [12] que, assim como uma árvore de *hash* regular, permite a geração de provas de existência ou não-existência de elementos específicos na árvore. Então, usuários mantendo a árvore completa são capazes de gerar provas que suas caixas estão não-gastas e uma pequena compilação de 33 bytes é suficiente para verificar essas provas. Porém, ao contrário de árvores de *hash* regulares, uma árvore AVL+ também permite a criação de provas de modificações na árvore que permitem que verificadores computem a nova compilação da árvore. Mineradores de Ergo são obrigados a gerar provas de modificações de blocos e uma *hash* dessa prova é incluída no cabeçalho do bloco junto com a compilação do estado resultante. Portanto, clientes leves que só mantenham uma pequena compilação do atual estado são capazes de verificar um bloco completo – eles podem verificar que todas as caixas gastas foram removidas do estado, todas as caixas criadas foram adicionadas ao estado e nenhuma outra mudança foi feita.

Árvores AVL+ permitem a construção de dicionários autenticados eficientes que reduzem o tamanho da prova e aceleram verificações entre 1,4 e 2,5 vezes em comparação com soluções anteriores, tornando-as mais adequadas para

aplicações em criptomoedas. Por exemplo, nossas provas são em torno de 3 vezes menores que provas em uma árvore Merkle-Patricia usada em Ethereum para o mesmo objetivo (ver Figura 1).

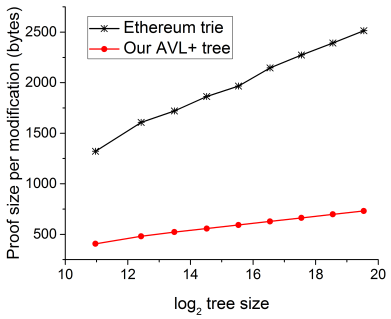


Figura 1: Comparação de tamanho de provas com uma árvore Merkle-Patricia

Finalmente, provas para múltiplas transações em um único bloco são comprimidas juntas, reduzindo seu comprimento total por um fator próximo de 2:

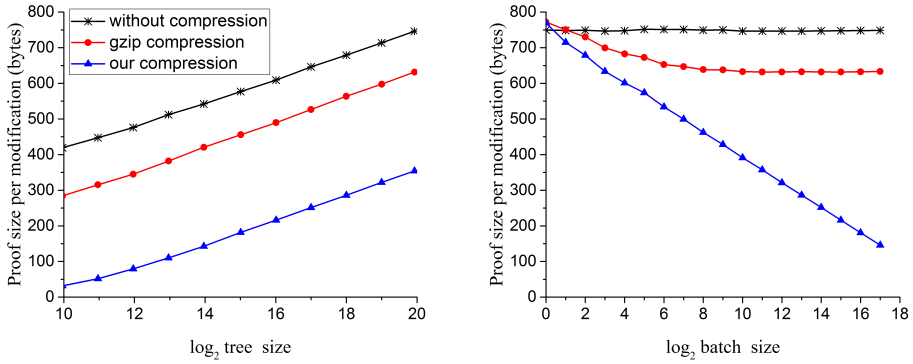


Figura 2: Esquerda: tamanho de prova por modificação para 2.000 transações como função do tamanho de árvore inicial $n$. Direita: tamanho de prova por modificação para uma árvore com $n = 1.000.000$ de chaves como função do tamanho de *batch* $B$. Em ambos os casos, metade das modificações foram inserções de novos pares (chave, valor) e a outra metade foi feita de valores para chaves existentes.

Então, estado em Ergo fornece uma forma segura e eficiente de provar existência e não-existência de certos elementos em si, além de provas de modificações em árvores. Essas operações de árvores são suportadas pela linguagem de contratos inteligentes de Ergo, deste movo dando a habilidade de implementar contratos sofisticados, como aqueles discutidos na Seção 7.

# 5 Resiliência e Sobrevivência

Sendo uma plataforma para dinheiro contratual, Ergo deve também dar suporte a contratos de longo-prazo por um período pelo menos tão longo quanto a expectativa de vida de uma pessoa. Porém, mesmo plataformas de contratos inteligentes novas e já existentes estão sofrendo problemas com degradação de performance e adaptabilidade a condições extremas. Isso leva a uma situação em que a criptomoeda depende de um pequeno grupo de desenvolvedores para fornecer uma solução em forma de bifurcação dura, ou a criptomoeda não sobrevive. Por exemplo, a rede Ethereum foi iniciada com uma algoritmo de consenso em Prova-de-Trabalho com um plano de mudar para Prova-de-Delegação num futuro próximo. Contudo, atrasos no desenvolvimento da Prova-de-Delegação tem levado a diversas soluções em bifurcação dura [20] e a comunidade ainda é forçada a depender de promessas dos desenvolvedores sobre implementar a próxima bifurcação dura.

O primeiro problema comum de sobrevivência é que, em busca de popularidade, desenvolvedores tendem a implementar soluções *ad-hoc* sem pesquisas e testes prévios. Tais soluções inevitavelmente levam a *bugs*, que então levam a correções apressadas, que levam a correções dessas correções, e assim por diante. Tudo isso torna a rede inconsistente e até mesmo menos segura. Um notável exemplo é a criptomoeda IOTA, que implementou várias soluções de escalabilidade, incluindo sua própria função *hash* e estrutura DAG, que a permitiu alcançar grande popularidade e valor de mercado. Porém, uma análise detalhada dessas soluções revelou múltiplos problemas sérios, incluindo ataques práticos que permitem roubo de tokens [21, 22]. Uma subsequente bifurcação dura [23] consertou esses problemas ao trocar sua função *hash* para a bem-conhecida função *SHA3*, deste modo confirmando a inutilidade desses tipos de inovações. A abordagem de Ergo aqui é a de usar soluções estáveis e bem conhecidas, mesmo que elas levem a uma lentidão nas inovações de curto-prazo. A maioria das soluções usadas em Ergo são formalizadas em artigos apresentados em conferências [12, 18, 3, 8, 24, 25] e tem sido amplamente discutidos na comunidade.

Um segundo problema que descentralização (e, portanto, sobrevivência) enfrenta é a falta de clientes leves seguros e agnósticos. Ergo tenta corrigir esse problema da tecnologia blockchain sem criar outros novos. Como Ergo é uma blockchain PoW, ela facilmente permite a extração de um pequeno cabeçalho do conteúdo do bloco. Esse cabeçalho sozinho permite a validação do trabalho feito no bloco e uma cadeia somente de blocos é o suficiente para selecionar a melhor cadeia e sincronização com a rede. Uma cadeia feita somente de cabeçalhos, apesar de muito menos que a blockchain inteira, ainda cresce linearmente com o tempo. Pesquisas recentes sobre clientes leves demonstram uma maneira para clientes leves se sincronizarem com a rede ao fazer o *download* de uma quantidade ainda menor de dados, deste modo destravando a habilidade de se juntar a rede usando *hardware* de baixo-custo e não-confiáveis, como telefones móveis [26, 27]. Ergo usa um estado autenticado 4 e, para transações incluídas em um bloco, um cliente pode fazer o *download* de uma prova da sua correção.

Portanto, independentemente de tamanho, um usuário regular com um telefone móvel pode se juntar a rede e começar a usar Ergo com a mesma garantia de segurança de um nó completo.

Leitores podem notar um terceiro problema em potencial em que, muito embora o suporte a clientes leves resolva a questão para usuários de Ergo, ele não soluciona o problema para mineradores de Ergo, que ainda precisam manter o estado inteiro para validação eficiente de transações. Em sistemas blockchain existentes, os usuários podem colocar dados arbitrários nesse estado. Esses dados, que duram para sempre, cria muita "poeira" e seu tamanho aumenta indefinidamente com o tempo [28]. Um grande tamanho de estado leva a sérias questões de segurança, porque quando o estado não cabe na memória RAM, um adversário pode gerar transações cuja validação se torna muito lenta devido ao necessário acesso ao armazenamento do minerador. Isto pode levar a ataques de negação de serviço, como o que ocorreu em Ethereum em 2016 [29]. Além do mais, o medo que a comunidade tem desses tipos de ataque, junto com o problema do "inchaço de estado" sem qualquer tipo de compensação para mineradores ou usuários para hospedarem o estado, podem ter prevenido soluções de escalabilidade que de outro modo poderiam ter sido implementadas (tal como tamanhos de bloco maiores, por exemplo). Para prevenir isso, Ergo possui um componente de aluguel de armazenamento: se uma saída permanece no estado por 4 anos sem ser consumida, um minerador pode cobrar uma pequena taxa para todo *byte* mantido no estado.

Essa ideia, que é similar a serviços comuns de armazenamento em nuvem, só foi proposta bem recentemente no contexto de criptomoedas [30] e tem diversas consequências importantes. Primeiro, garante que mineração de Ergo sempre será estável, diferentemente de Bitcoin e outras moedas PoW, onde mineração pode se tornar instável após o fim das emissões [31]. Segundo, crescimento do tamanho do estado se torna controlável e previsível, deste modo ajudando mineradores de Ergo a administrar seus requisitos de *hardware*. Terceiro, ao colecionar taxas de armazenamento de caixas obsoletas, mineradores podem retorar moedas à circulação e, portanto, prevenir o decrescimento estável da oferta circulante devido a chaves perdidas [32]. Todos esses efeitos devem dar suporte à sobrevivência de longo-prazo de Ergo, tanto tecnica quando economicamente.

Um quarto desafio vital para sobrevivência é que as mudanças no ambiente e demanda externos e colocados no protocolo. Um protocolo deve adaptar deve se adaptar a infraestrutura de *hardware* que sempre muda, novas ideias para melhorar segurança ou escalabilidade que emergem ao longo do tempo, a evolução de casos de uso, e etc. Se todas as regras forem fixas e sem qualquer habilidade de serem alteradas de uma forma descentralizada, mesmo uma simples mudança de constante pode levar a debates acalorados e divisões na comunidade. Por exemplo, a discussão do limite de tamanho de bloco em Bitcoin levou a sua repartição em diversas moedas independentes. Em contraste, o protocolo Ergo é auto-alterável e é capaz de se adaptar a um ambiente variável. Em Ergo, parâmetros como tamanho de bloco podem ser mudados sob demanda via votação de mineradores. No começo de cada época votante de 1024 blocos, um minerador propõe mudanças de até 2 parâmetros (e.g. um aumento de tama-

nho de bloco e uma diminuição do fator de taxa de armazenamento). Durante o resto da época, mineradores votam para aprovar ou rejeitar as mudanças. Se uma maioria dos votos dentro de uma época der suporte à mudança, os novos valores são escritos na seção de extensão do primeiro bloco da nova época, e a rede começa a usar os valores atualizados para mineração e validação de blocos.

Para absorver mudanças mais fundamentais, Ergo segue a abordagem de *bifurcabilidade suave* que permite mudar significativamente o protocolo, enquanto mantendo nós antigos operacionais. No começo de uma época, um minerador pode também propor votos para uma mudança mais fundamental (e.g., adicionar uma nova instrução em ErgoScript) descrevendo regras de validação afetadas. Votações para tais mudanças drásticas continuam por 32.768 blocos e requerem pelo menos 90% de votos "Sim" para serem aceitas. Uma vez sendo aceita, um longo período de ativação de 32.768 blocos começa a dar tempo a nós obsoletos par atualizarem sua versão de *software*. Se um *software* de nós ainda estiver desatualizado no período de ativação, então ele pula as verificações especificadas mas continuam a validar todas as regras já conhecidas. Uma lista de mudanças prévias de bifurcação suave é registrada na extensão para permitir que clientes leves de qualquer versão de *software* se juntem à rede e se emparelhem às atuais regras de validação. Uma combinação de bifurcabilidade suave com o protocolo de votação permite mudanças de quase todos os parâmetros da rede, exceto as regras de PoW que são responsáveis pela votação em si.

# 6 *Tokens* Nativos de Ergo

A plataforma Ergo tem seu *token* nativo, que é chamado de Erg e é divisível em até $10^9$ unidades menores, nanoErgs (um nanoErg é um bilionésimo de um Erg). Ergs são importantes para a estabilidade e segurança da plataforma Ergo por diversas razões discutidas abaixo.

Durante a fase inicial da vida de Ergo, os mineradores irão receber a recompensa em Ergs de acordo com um calendário de emissão de *token* pré-definido e que é parte definitiva do código (ver 6.1 para mais detalhes). Essas moedas irão incentivar mineradores a participar da rede Ergo, segurando-a de ataques baseados em *hashrate*, e.g. os conhecidos ataques de 51% [33].

A emissão de Erg será finalizada dentro de apenas oito anos, e após isso os mineradores receberão Ergs somente vindas das taxas. Muito embora ajustáveis durante o tempo através de votação na rede pelos mineradores, tamanho de bloco em Ergo e máximo custo computacional em qualquer momento será limitado, e portanto mineradores são forçados a escolher somente um subconjunto de transações da *mempool* durante períodos de alto demanda. Taxas ajudarão mineradores a arranjar as transações, prevenindo ataques *spam* enquanto permitindo que mineradores incluam transações de usuários honestos em blocos.

Apesar de recursos computacionais e da rede, uma transação utiliza armazenamento ao aumentar o tamanho do estado. Em criptomoedas existentes, um elemento do estado, i.e. um UTXO em blockchains baseadas em UTXO, chamado de "caixa" em Ergo, uma vez criado vive possivelmente para sempre, sem

qualquer compensação para mineradores e alguns usuários que devem manter esse estado em memória RAM. Isso leva a um desalinhamento de incentivos e continuamente aumenta o tamanho do estado. Em contraste, Ergo possui uma componente de aluguel de armazenamento que periodicamente cobra Erg dos usuários por todo *byte* incluído no estado. Esse aluguel de armazenamento está tornando o sistema mais estável ao limitar o tamanho do estado ou assegurar compensação devida por maiores tamanhos de estado, retornando moedas perdidas à circulação e fornecendo uma recompensa aos mineradores que é estável e previsível.

Logo, como é uma plataforma de dinheiro contratual, Ergo é adequada para construção de aplicações e sistemas monetários. Porém, participação em um sistema deste tipo requer o uso do *token* nativo de Ergo, Erg, também com o objetivo de pagar aluguel de armazenamento e taxas de transação que fornecerão aos mineradores fortes incentivos recorrentes para garantir a segurança da rede com adequado poder de *hash*. Usuários, pela sua vez, serão altamente incentivados a adquirir, usar e guardar Ergs se eles encontrarem aplicações em Ergo que sejam de alto valor.

## 6.1 Emissão

Todos os *tokens* Erg que entrarão em circulação no sistema são apresentados no estado inicial, que consiste de 3 caixas:

- *Prova de Ausência de Pré-mineração.* Esta caixa contém exatamente uma Erg e está protegida por um *script* que previne que ela seja gasta por qualquer um. Logo, é uma caixa de longa duração que permanecerá no sistema até que a componente de aluguel de armazenamento a destrua. Seu propósito principal é provar que a mineração de Ergo não foi iniciada privadamente por alguém antes da data de lançamento anunciada. Para fazer isso, registros adicionais dessa caixa contém as últimas manchetes da mídia (The Guardian, Vedomosti, Xinhua), assim como os últimos identificadores de bloco de Bitcoin e Ethereum. Assim, a mineração de Ergo não poderia ter começado antes de certos eventos do mundo real e do espaço de criptomoedas.

- *Tesouro.* Esta caixa contém 4.330.791,5 Ergs que serão usadas para financiar o desenvolvimento de Ergo. Seu *script* protetor [34] consiste de duas partes.

  Primeiro, garante que somente uma porção pré-determinada do valor da caixa é destravada. Durante os blocos 1 e 525.599 (2 anos), 7,5 Ergs serão liberadas todo bloco. Então, durante os blocos 525.600 e 590.399 (3 meses), 4,5 Ergs serão liberadas por bloco. Finalmente, durante os blocos 590.400 e 655.199 (3 meses), 1,5 Ergs serão liberadas por bloco. Essa regra garante a presença de fundos para o desenvolvimento de Ergo por 2,5 anos e, em qualquer momento, recompensas não excedem 10% do número total de moedas em circulação.

Segundo, essa caixa possui uma proteção personalizada contra gastos inesperados. Inicialmente, ela requer que a transação de gasto seja assinada por pelo menos duas ou três chaves secretas que estão sob controle dos membros iniciais da equipe. Quando eles gastam a caixa, eles estão livres para modificar essa parte do *script* como desejarem, e.g. adicionando novos membros para proteger os fundos da fundação.

Durante o primeiro ano, esses fundos serão usados para cobrir o *token* EFYT pré-lançado. Depois, eles serão distribuídos de uma forma descentralizada via um sistema de votação comunitário que está sob desenvolvimento.

- *Recompensas para Mineradores.* Esta caixa contém 93.409.132 Ergs que serão coletadas por mineradores de bloco como uma recompensa pelo seu trabalho. Seu *script* protetor [35] requer que a transação de gasto possua exatamente duas saídas com as seguintes propriedades:

  – A primeira saída deve ser protegida pelo mesmo *script* e o número de Ergs nela deve igualar as recompensas restantes. Durante os blocos 1 e 655.199, um minerador será capaz de coletar 67,5 Ergs dessa caixa. Durante os blocos 655.200 e 719.999 (3 meses), um minerador será capaz de coletar 66 Ergs. A partir daí, a recompensa por bloco será reduzida em 3 Ergs a cada 64.800 blocos (3 meses) até chegar a zero no bloco 2.080.800.

  – A segunda saída deve conter as moedas restantes e deve ser protegida pela seguinte condição: ela pode ser gasta por um minerador que tenha resolvido o quebra-cabeças PoW do bloco e não antes que 720 blocos depois do bloco atual.

Todas essas regras resultam na seguinte curva que denota o número de moedas em circulação em função do tempo:

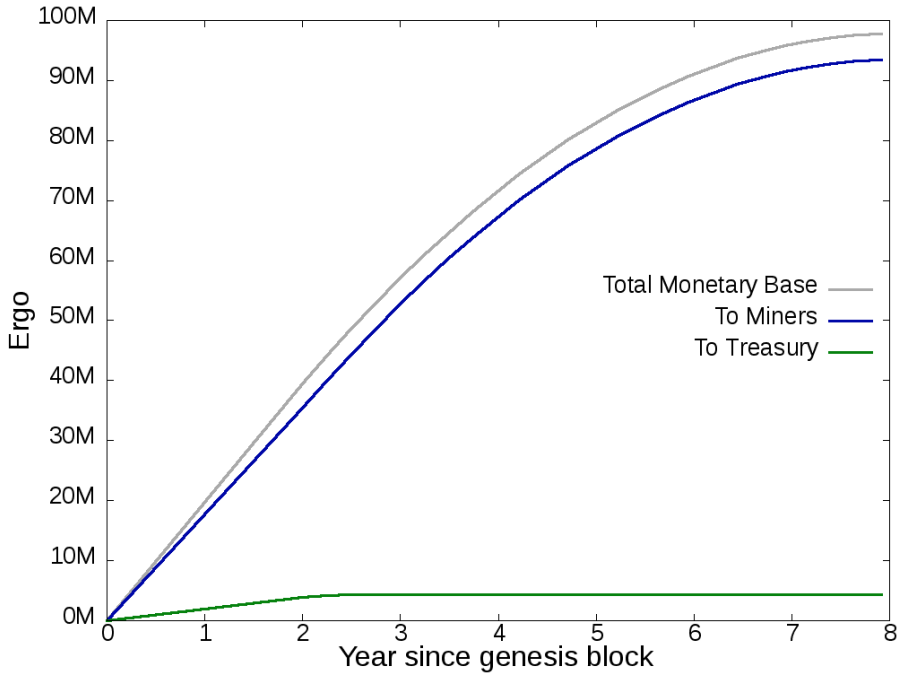


Figura 3: Curva de emissão de Ergo

# 7 Dinheiro Contratual

Na nossa opinião, a esmagadora maioria dos casos de uso de blockchains públicas (mesmo aqueles que afirmam fornecer um computador mundial descentralizado de uso geral) é voltada para aplicações financeiras, que não necessitam de Turing-completeza. Por exemplo, se um oráculo escreve dados não-financeiros na blockchain (e.g. temperatura), esses dados são geralmente usados em um contrato financeiro. Uma outra observação trivial que fazemos é que muitas aplicações usam *tokens* digitais com mecanismos de funcionamento diferentes dos *tokens* nativos.

Para um desenvolvedor de aplicativos, a Plataforma Ergo oferece *personalizados* (que são cidadãos de primeira-classe) e uma linguagem de domínio específico para escrever condições protetoras de caixas, a fim de implementar aplicações financeiras seguras e flexíveis. Aplicativos em Ergo são definidos em termos de *scripts* de proteção construídos nas caixas, que também podem conter dados envolvidos na execução. Usamos o termo *dinheiro contratual* para definir Ergs (e *tokens* secundários) cujos usos são limitados por um contrato. Isso se aplica também para todos os *tokens* em existência na plataforma, porque qualquer caixa com esses conteúdos (Ergs, *tokens*, dados) está limitada por um contrato.

Contudo, podemos distinguir entre dois tipos de Ergs contratuais. A primeira, chamada *Ergs livres*, são as que poderiam mudar seus contratos facilmente e não ter restrições nas saídas ou as outras entradas da transação de gasto. O segundo tipo é o de *Ergs limitadas*, cujos contratos necessitam que a transação de gasto possua caixas de entrada e saída com propriedades específicas.

Por exemplo, se uma caixa $A$ está protegida por apenas uma chave pública (logo fornecer uma assinatura contra uma transação de gasto é suficiente para destruí-la), o(a) dono(a) da chave pública pode gastar $A$ e transferir as Ergs para um número arbitrário de caixas de saída. Portanto, as Ergs dentro de $A$ estão livres. Em contraste, imagine que uma caixa $B$ esteja protegida por uma combinação de uma chave pública e uma condição que demanda que a transação de gasto crie uma caixa de saída com a mesma quantidade de Ergs que em $B$ e cujo *script* guardião possua a *hash* `rBMUEMuPQUx3GzgFZSsHmLMBouLabNZ4HcERm4N` (em codificação *Base58*). Neste caso, as Ergs em $B$ são Ergs limitadas.

Similarmente, podemos definir *tokens* livres e limitados. Um contrato em Ergo pode ter diversos híbridos, como Ergs limitadas e *tokens* livres ou ambos limitados sob uma chave pública e livres sob outra.

## 7.1 Preliminares para Contratos em Ergo

Enquanto em Bitcoin uma saída de transação é protegida por um programa em uma linguagem *stack-based* chamada *Script*, em Ergo uma caixa está protegida por uma fórmula lógica que combina predicados sobre um contexto com declarações criptográficas provadas via protocolos de conhecimento-zero usando conectivos E, OU e $k$-out-of-$n$. A fórmula é representada como um grafo acíclico direto digitado, cuja forma em série é escrita em uma caixa. Para destruir esta caixa, uma transação de gasto precisa fornecer argumentos (que incluem provas de conhecimento-zero) que satisfaçam a fórmula.

Contudo, na maioria dos casos, desenvolvedores provavelmente não desenvolverão contratos em termos de grafos. Em vez disso, eles gostariam de usar uma linguagem de alto-nível como ErgoScript, que apresentamos junto com o cliente de referência.

Escrever *scripts* em ErgoScript é fácil. Como um exemplo, para uma assinatura uma-dentre-duas, o *script* de proteção seria $pk_1 \| pk_2$, que significa "prove conhecimento de uma chave secreta correspondendo à chave pública $pk_1$ ou conhecimento de uma chave secreta correspondendo à chave pública $pk_2$". Temos dois documentos separados para ajudar no desenvolvimento de contratos com ErgoScript: o "Tutorial ErgoScript" [36] e o "Tutorial Avançado de ErgoScript" [37]. Logo, não entramos aqui em detalhes de desenvolvimento de contratos usando. Do contrário, fornecemos alguns exemplos motivadores nas seções a seguir.

Duas outras funcionalidades de Ergo que moldam possibilidades de contratos são:

- *Entradas de Dados:* Para ser usada em uma transação, uma caixa não

precisa ser destruída, podendo ser usada em modo somente-leitura. Neste caso, nos referimos à caixa como fazendo parte da *entrada de dados* da transação. Portanto, uma transação recebe dois conjuntos de caixas como seus argumentos, as entradas e entradas de dados, e produz um conjunto de caixas chamado *saídas*. Entradas de dados são úteis para aplicações de oráculos e contratos interagentes.

- *Tokens Personalizados:* Uma transação pode carregar tantos *tokens* quanto possíveis dentro de um limite de complexidade estimada para processá-la. Este limite é um parâmetro que é configurado por votação dos mineradores. Uma transação pode também emitir um único *token* com um identificador único que seja igual ao identificador da primeira caixa de entrada (gastável) da transação. O identificador é único ao partir do princípio da resistência à colisão de uma função *hash* base. A quantidade de *tokens* emitidas pode ser qualquer número dentro do intervalo $[1, 9223372036854775807]$. A regra de preservação fraca é seguida para *tokens*, que necessita que a quantidade total de qualquer *token* em uma saída de transação deva ser não mais do que a quantidade total daquele token nas entradas da transação (i.e., uma certa quantia de *token* pode ser destruída). Por outro lado, a regra de reserva forte é seguida para Ergs, que necessita que a quantidade total de Ergs nas entradas e saídas deve ser a mesma.

## 7.2 Exemplos de Contratos

Nesta seção, apresentamos alguns exemplos que demonstram a superioridade dos contratos em Ergo em comparação com os de Bitcoin. Os exemplos incluem usar dados provenientes de oráculos para apostas, um *misturador* não-interativo, trocas atômicas, moedas complementares e uma oferta inicial de moedas implementada utilizando a blockchain Ergo.

### 7.2.1 Um Exemplo de Oráculo

Equipados com *tokens* personalizados e entradas de dados, podemos desenvolver um simples exemplo de oráculo que também demonstra alguns padrões de design que descobrimos quando brincávamos com contratos em Ergo. Parta da hipótese de que Alice e Bob querem apostar no tempo de amanhã ao colocar dinheiro em uma caixa que se torna gastável por Alice se a temperatura de amanhã for maior do que $15^{\circ}$C e gastável por Bob se o contrário acontecer. Para entregar a temperatura à blockchain, é preciso um oráculo confiável.

Em contraste a Ethereum e suas contas de longa-duração onde um identificador de um oráculo confiável é conhecido a priori, entregar dados com caixas de uso-único é mais complicado. Para começar, uma caixa protegida por uma chave de oráculo não pode ser confiável, já que qualquer um pode ter criado tal caixa. É possível incluir dados assinados em uma caixa e verificar a assinatura do oráculo no contrato (temos um exemplo disso a seguir), mas isso é

bastante complicado. Como alternativa, uma solução com *tokens* personalizados é bastante simples.

Primeiro, um *token* identificando o oráculo deve ser emitido. No caso mais simples, a quantidade desse *token* pode ser um. Chamamos tal *token* de *um token singleton*. O oráculo cria uma caixa contendo este *token* junto com seus dados (i.e., a temperatura) no registro $R_4$ e o tempo da época UNIX no registro $R_5$. Para atualizar a temperatura, o oráculo destrói a caixa e cria uma nova com a temperatura atualizada.

Partimos da hipótese de que Alice e Bob conhecem o identificador do *token* do oráculo a priori. Com isso, eles podem conjuntamente criar uma caixa com um contrato que requer primeiro que entrada de dados (em modo somente-leitura) contenha o *token* do oráculo. O contrato extrai a temperatura e tempo da entradas de dados e decide quem recebe o pagamento. O código é tão simples quanto o mostrado abaixo:

**Algoritmo 3** Exemplo de Contrato de Oráculo

```
1: val dataInput = CONTEXT.dataInputs(0)
2: val inReg = dataInput.R4[Long].get
3: val inTime = dataInput.R5[Long].get
4: val inToken = dataInput.tokens(0)._1 == tokenId
5: val okContractLogic = (inTime > 1556089223) &&
6:           ((inReg > 15L && pkA) || (inReg ≤ 15L && pkB))
7: inToken && okContractLogic
```

Este contrato mostra como um *token singleton* pode ser usado para autenticação. Como uma possível alternativa, o oráculo pode colocar o tempo e temperatura dentro de uma caixa junto com uma assinatura nesses dados. Contudo, isso requer verificação de assinatura, que é mais complexo e caro em comparação à abordagem de *token singleton*. Além disso, o contrato mostra como entradas de dados somente-leitura podem ser úteis para acessar dados armazenados em alguma outra caixa no estado. Sem entradas de dados, um oráculo deve emitir uma caixa gastável para cada par de Alice e Bob. Com entradas de dados, o oráculo emite somente uma única caixa.

### 7.2.2 Um Exemplo de *Misturador*

Privacidade é importante para uma moeda digital, mas implementá-la pode ser custoso ou requerer uma configuração confiável. Logo, é desejável encontrar uma forma barata de misturar moedas. Como um primeiro passo nesse sentido, oferecemos um protocolo não-interativo de mistura entre dois usuários, Alice e Bob. O protocolo funciona assim:

1. Alice cria uma caixa que demanda que a transação de gasto satisfaça certas condições. Após isso, Alice só observa a blockchain; nenhuma interação com Bob é necessária.

2. Bob cria uma transação gastando a caixa de Alice junto com uma de suas caixas para gerar duas saídas contendo *scripts* idênticos, mas dados diferentes. Tanto Alice quanto Bob podem gastar somente uma das duas saídas, mas um observador decide qual saída pertence a quem, porque elas parecem indistinguíveis.

Por simplicidade, não consideramos taxas no exemplo. A ideia de misturador é similar à de trocas de chave Diffie-Hellman não-interativas. Primeiro, Alice gera um valor secreto $x$ (um número enorme) e publica a o valor público correspondente $gX = g^x$. Ela exige que Bob gere um número secreto $y$ e inclua dentro de cada saída dois valores $c_1$, $c_2$, onde um valor é igual a $g^y$ e o outro igual a $g^{xy}$. Bob usa uma moeda aleatória para escolher sentidos para $\{c_1, c_2\}$. Sem acesso aos segredos, um observador externo não pode adivinhar com probabilidade melhor que $\frac{1}{2}$ se $c_1$ é igual a $g^y$ ou a $g^{xy}$. Isso parte da hipótese de que a primitiva criptográfica que usamos possui uma certa propriedade, que é a de que a Decisão Diffie-Hellman (DDH) é difícil. Para destruir uma caixa de saída, uma prova deve ser dada de que ou $y$ é conhecido tal que $c_2 = g^y$, ou $x$ é conhecido tal que $c_2 = c_1^x$. O contrato da caixa de Alice verifica se $c_1$ e $c_2$ estão bem formados. Os trechos de código para moeda de Alice e a saída da transação de mistura são fornecidos em Algoritmos 4 e 5, respectivamente. Como ErgoScript atualmente não dá suporte para provas de conhecimento de algum $x$ tal que $c_2 = {c_1}^x$ para $c_1$ arbitrário, então provaremos uma declaração um pouco mais longa, mas que é suportada: provaremos conhecimento de $x$ tal que $gX = g^x$ e $c_2 = {c_1}^x$. Isto é chamado de *proveDHTuple*.

**Algoritmo 4** *Script* de entrada de Alice

```
1:  val c1 = OUTPUTS(0).R4[GroupElement].get
2:  val c2 = OUTPUTS(0).R5[GroupElement].get
3:
4:  OUTPUTS.size == 2 &&
5:  OUTPUTS(0).value == SELF.value &&
6:  OUTPUTS(1).value == SELF.value &&
7:  blake2b256(OUTPUTS(0).propositionBytes) == fullMixScriptHash &&
8:  blake2b256(OUTPUTS(1).propositionBytes) == fullMixScriptHash &&
9:  OUTPUTS(1).R4[GroupElement].get == c2 &&
10: OUTPUTS(1).R5[GroupElement].get == c1 && {
11:     proveDHTuple(g, gX, c1, c2) ||
12:     proveDHTuple(g, gX, c2, c1)
13: }
```

**Algoritmo 5** *Script* de Saída da Transação de Mistura

```
1: val c1 = SELF.R4[GroupElement].get
2: val c2 = SELF.R5[GroupElement].get
3: proveDlog(c2) || // ou c2 é g^y
4: proveDHTuple(g, c1, gX, c2) // ou c2 é u^y = g^xy
```

Direcionamos o leitor à Ref. [37] para ver uma prova de indistinguibilidade das saídas e detalhes do porquê Alice e Bob podem gastar somente suas respectivas moedas.

### 7.2.3 Mais Exemplos

Nesta seção, brevemente descrevemos mais alguns exemplos, junto com referências para os documentos que mostram seus detalhes e códigos.

**Trocas Atômicas** Trocas atômicas entre-cadeias entre Ergo e qualquer blockchain que dá suporte a pagamentos para pré-imagens de *hash*ou em SHA-256 ou em Blake2b-256 e bloqueios de tempo podem ser feitos de forma similar àquela proposta por Bitcoin [38]. Uma implementação alternativa em Ergo é fornecida em Ref. [36]. Como Ergo também possui *tokens* personalizados, trocas atômicas na blockchain Ergo (e.g. Erg-para-*token* ou *token*-para-*token*) também é possível. Uma implementação para isso pode também pode ser encontrada em [36].

**Financiamento Coletivo** Consideramos o cenário mais simples de financiamento coletivo. Neste exemplo, um projeto de financiamento coletivo com uma chave pública conhecida é considerado bem-sucedido se pode coletar saídas não-gastas com um valor total não menor que uma certa quantia antes de uma certa altura. Um patrocinador de um projeto cria uma caixa de saída protegida pela seguinte declaração: a caixa pode ser gasta se a transação de gasto possuir a primeira caixa de saída protegida pela chave do projeto e quantidade não menor que a quantidade alvo. Então, o projeto pode coletar (em uma única transação) as maiores caixas de saída de patrocinadores com um valor total não menor que a quantidade alvo (é possível coletar até 22.000 saídas, que é o suficiente até mesmo para uma grande campanha de financiamento coletivo). Para as saídas restantes, é possível construir transações subsequentes. O código pode ser encontrado em Ref. [36].

**O Sistema de Negociação de Câmbio Local** Aqui, brevemente demonstramos um Sistema de Negociação de Câmbio Local (LETS, sigla em Inglês) em Ergo. Em um sistema desses, um(a) membro(a) da comunidade pode emitir moedas comunitárias contraindo dívida pessoal. Por exemplo, se Alice possuir saldo zero é estiver comprando algo por 5 *tokens* comunitários de Bob, cujo saldo também é zero, o saldo dela após a negociação seria de $-5$ *tokens*, enquanto o saldo de Bob seria de 5 *tokens*. Então, Bob pode comprar algo usando

seus 5 *tokens*, por exemplo, de Carol. Normalmente, em tais sistemas, há um limite em saldos negativos para se evitar abusos.

Como uma comunidade digital é vulnerável a ataques Sybil [39], algum mecanismo é preciso para prevenir tais ataques, onde nós Sybil criam dívidas. A solução mais simples é usar um comitê de administradores confiáveis que aprovam novos membros da comunidade. Uma solução agnóstica, porém mais complexa, é usar depósitos de seguro feitos em Ergs. Por simplicidade, consideremos aqui a abordagem de comitês.

Este exemplo contém dois contratos interagentes. Um *contrato de administração* mantém uma lista de membros da comunidade, e um(a) novo(a) membro(a) pode ser adicionado se alguma condição de administração é satisfeita (por exemplo, uma assinatura limiar é fornecida). Um(a) novo(a) membro(a) é associado a uma caixa contendo um *token* que identifica o(a) membro(a). Esta caixa, que contém o *contrato de membro(a)*, é protegida por um *script* de câmbio especial que necessita que a transação de gasto faça uma troca justa. Pularemos o código correspondente, que pode ser encontrado em um artigo separado [40].

O que este contrato mostra, em contraste com o exemplo anterior, é que ao invés de armazenar a lista de membros, somente uma curta compilação de uma árvore AVL+ autenticada pode ser incluída na caixa. Isto permite uma redução em requisitos de armazenamento para o estado. Uma transação que faça leitura ou modificação da lista de membros deve fornecer uma prova para operações de leitura ou modificação da árvore AVL+. Portanto, economizar espaço no armazenamento do estado leva a transações maiores, mas esse problema de escalabilidade é mais fácil de resolver.

**Ofertas Iniciais de Moedas** Discutimos um exemplo de Oferta Inicial de Moedas (ICO, sigla em Inglês) que mostra como contratos multi-estágio podem ser criados em Ergo. Como a maioria dos ICOs, nosso exemplo possui três estágios. No primeiro estágio, o projeto arrecada dinheiro em Ergs. Em um segundo estágio, o projeto emite um novo *token*, cuja quantia iguala o número de nanoErgs arrecadadas no primeiro estágio. No terceiro estágio, os investidores podem sacar os *tokens* emitidos.

Note que o primeiro e o terceiro estágios tem muitas transações na blockchain, enquanto uma única transação é o suficiente para o segundo estágio. Similar ao exemplo anterior, o contrato do ICO usa uma árvore AVL+ para armazenar a lista de pares (investidor, quantia). O código completo está disponível em [41].

**Mais Exemplos** Temos ainda mais exemplos de aplicações em Ergo nas Refs. [36, 37]. Esses exemplos incluem emissão controlada por tempo, contratos em carteiras frias, o jogo pedra-papel-tesoura, e muitos outros.

# Referências

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] Bitcoin's price surpasses $18,000 level, market cap now higher than visa's. [Online]. Available: https://cointelegraph.com/news/bitcoins-price-surpasses-18000-level-market-cap-now-higher-than-visas

[3] A. Chepurnoy, V. Kharin, and D. Meshkov, "A systematic approach to cryptocurrency fees," *IACR Cryptology ePrint Archive*, vol. 2018, p. 78, 2018.

[4] Skyrocketing fees are fundamentally changing bitcoin. [Online]. Available: https://arstechnica.com/tech-policy/2017/12/bitcoin-fees-rising-high/

[5] N. Szabo, "Smart contracts," *Unpublished manuscript*, 1994.

[6] J. Zahnentferner, "Chimeric ledgers: Translating and unifying utxo-based and account-based cryptocurrencies." *IACR Cryptology ePrint Archive*, vol. 2018, p. 262, 2018.

[7] I accidentally killed it: Parity wallet bug locks $150 million in ether. [Online]. Available: https://www.ccn.com/i-accidentally-killed-it-parity-wallet-bug-locks-150-million-in-ether

[8] A. Chepurnoy, V. Kharin, and D. Meshkov, "Self-reproducing coins as universal turing machine," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, pp. 57–64.

[9] M. Jansen, F. Hdhili, R. Gouiaa, and Z. Qasem, "Do smart contract languages need to be turing complete?" 2019. [Online]. Available: https://www.researchgate.net/publication/332072371_Do_Smart_Contract_Languages_Need_to_be_Turing_Complete/download

[10] Very slow syncing on hard drive. [Online]. Available: https://github.com/ethereum/go-ethereum/issues/14895

[11] Why is my node synchronization stuck/extremely slow at block 2,306,843? [Online]. Available: https://ethereum.stackexchange.com/questions/9883/why-is-my-node-synchronization-stuck-extremely-slow-at-block-2-306-843

[12] L. Reyzin, D. Meshkov, A. Chepurnoy, and S. Ivanov, "Improving authenticated dynamic dictionaries, with applications to cryptocurrencies," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 376–392.

[13] L. Goodman, "Tezos — a self-amending crypto-ledger white paper," *URL: https://www. tezos. com/static/papers/white_paper. pdf*, 2014.

[14] A. Biryukov and D. Khovratovich, "Equihash: Asymmetric proof-of-work based on the generalized birthday problem," *Ledger*, vol. 2, pp. 1–30, 2017.

[15] Ethash. [Online]. Available: https://github.com/ethereum/wiki/wiki/Ethash/6e97c9cea49605264c6f4d1dc9e1939b1f89a5a3

[16] A. Chepurnoy, V. Kharin, and D. Meshkov, "Autolykos: The ergo platform pow puzzle," 2019. [Online]. Available: https://docs.ergoplatform.com/ErgoPow.pdf

[17] Autolykos gpu miner. [Online]. Available: https://github.com/ergoplatform/Autolykos-GPU-miner

[18] D. Meshkov, A. Chepurnoy, and M. Jansen, "Short paper: Revisiting difficulty control for blockchain systems," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2017, pp. 429–436.

[19] A. Chepurnoy, C. Papamanthou, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation," Cryptology ePrint Archive, Report 2018/968, Tech. Rep., 2018.

[20] Ethereum's blockchain is once again feeling the 'difficulty bomb' effect. [Online]. Available: https://www.coindesk.com/ethereum-blockchain-feeling-the-difficulty-bomb-effect

[21] E. Heilman, N. Narula, G. Tanzer, J. Lovejoy, M. Colavita, M. Virza, and T. Dryja, "Cryptanalysis of curl-p and other attacks on the iota cryptocurrency."

[22] G. De Roode, I. Ullah, and P. J. Havinga, "How to break iota heart by replaying?" in *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018, pp. 1–7.

[23] Iota vulnerability report: Cryptanalysis of the curl hash function enabling practical signature forgery attacks on the iota cryptocurrency. [Online]. Available: https://github.com/mit-dci/tangled-curl/blob/master/vuln-iota.md

[24] A. Chepurnoy and M. Rathee, "Checking laws of the blockchain with property-based testing," in *Blockchain Oriented Software Engineering (IWBOSE), 2018 International Workshop on*. IEEE, 2018, pp. 40–47.

[25] T. Duong, A. Chepurnoy, and H.-S. Zhou, "Multi-mode cryptocurrency systems," in *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts*. ACM, 2018, pp. 35–46.

[26] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," Cryptology ePrint Archive, Report 2017/963, 2017. Accessed: 2017-10-03, Tech. Rep., 2017.

[27] L. Luu, B. Buenz, and M. Zamani, "Flyclient super light client for cryptocurrencies," *IACR Cryptology ePrint Archive*, 2019. [Online]. Available: https://eprint.iacr.org/2019/226

[28] C. Pérez-Solà, S. Delgado-Segura, G. Navarro-Arribas, and J. Herrera-Joancomartí, "Another coin bites the dust: an analysis of dust in utxo-based cryptocurrencies," *Royal Society open science*, vol. 6, no. 1, p. 180817, 2019.

[29] Ethereum network attacker's ip address is traceable. [Online]. Available: https://www.bokconsulting.com.au/blog/ethereum-network-attackers-ip-address-is-traceable/

[30] A. Chepurnoy and D. Meshkov, "On space-scarce economy in blockchain systems." *IACR Cryptology ePrint Archive*, vol. 2017, p. 644, 2017.

[31] M. Carlsten, H. Kalodner, S. M. Weinberg, and A. Narayanan, "On the instability of bitcoin without the block reward," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 154–167.

[32] E. Krause, "A fifth of all bitcoin is missing. these crypto hunters can help," 2018.

[33] 51% attack. [Online]. Available: https://en.bitcoinwiki.org/wiki/51%25_attack

[34] Script of the ergo treasury box. [Online]. Available: https://github.com/ScorexFoundation/sigmastate-interpreter/blob/1b7b5a69035fc384b47c18ccf42b87dd95bbcf12/src/main/scala/org/ergoplatform/ErgoScriptPredef.scala#L118

[35] Script of the ergo emission box. [Online]. Available: https://github.com/ScorexFoundation/sigmastate-interpreter/blob/1b7b5a69035fc384b47c18ccf42b87dd95bbcf12/src/main/scala/org/ergoplatform/ErgoScriptPredef.scala#L74

[36] Ergoscript, a cryptocurrency scripting language supporting noninteractive zero-knowledge proofs. [Online]. Available: https://docs.ergoplatform.com/ErgoScript.pdf

[37] Advanced ergoscript tutorial. [Online]. Available: https://docs.ergoplatform.com/sigmastate_protocols.pdf

[38] T. Nolan, "Alt chains and atomic transfers," 2013. [Online]. Available: https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949

[39] Sybil attack. [Online]. Available: https://en.wikipedia.org/wiki/Sybil_attack

[40] A local exchange trading system on top of ergo. [Online]. Available: https://ergoplatform.org/blog/2019_04_22-lets/

[41] An ico example on top of ergo. [Online]. Available: https://ergoplatform.org/blog/2019_04_10-ico-example/