



Henrique Chaves Lopes

# **Diferentes Funções de Prova de Trabalho em uma Blockchain**

# Introdução

Blockchain criada por Satoshi Nakamoto em 2008.

## Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto  
satoshin@gmx.com  
www.bitcoin.org

**Abstract.** A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work forming a record that cannot be changed without redoing

# Introdução

**JORNAL DA USP**



 PORTAL DA USP  FALE CONOSCO  WHATSAPP  ENVIE UMA PAUTA  PODCASTS  RÁDIO USP  TV USP  USP NEWS  NEWSLETTE

QUALIDADES  CIÊNCIAS  CULTURA  DIVERSIDADE  EDUCAÇÃO INSTITUCIONAL  RÁDIO USP  TECNOLOGIA UNIVERSIDADE   BUSCA

Início > Ciências > Tecnologia blockchain pode reduzir risco de roubos e fraudes na área da saúde

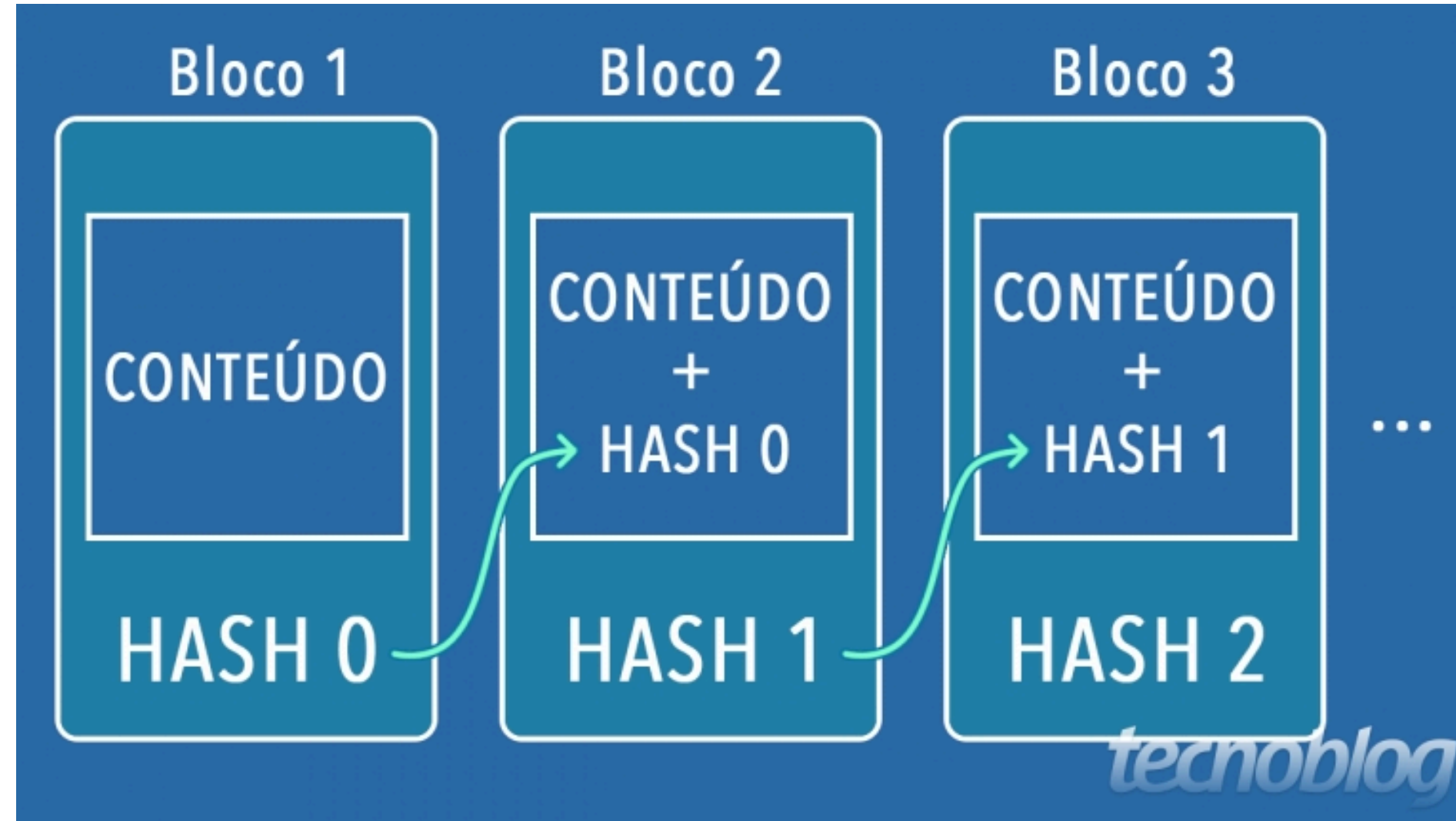
## Tecnologia blockchain pode reduzir risco de roubos e fraudes na área da saúde

Sistema de rastreabilidade de medicamentos que impede alteração de registros garantiria maior segurança para empresas e pacientes na cadeia nacional de suprimentos

 29/05/2023 - Publicado há 2 anos  Atualizado: 31/05/2023 às 13:28

Texto: Ivan Conterno

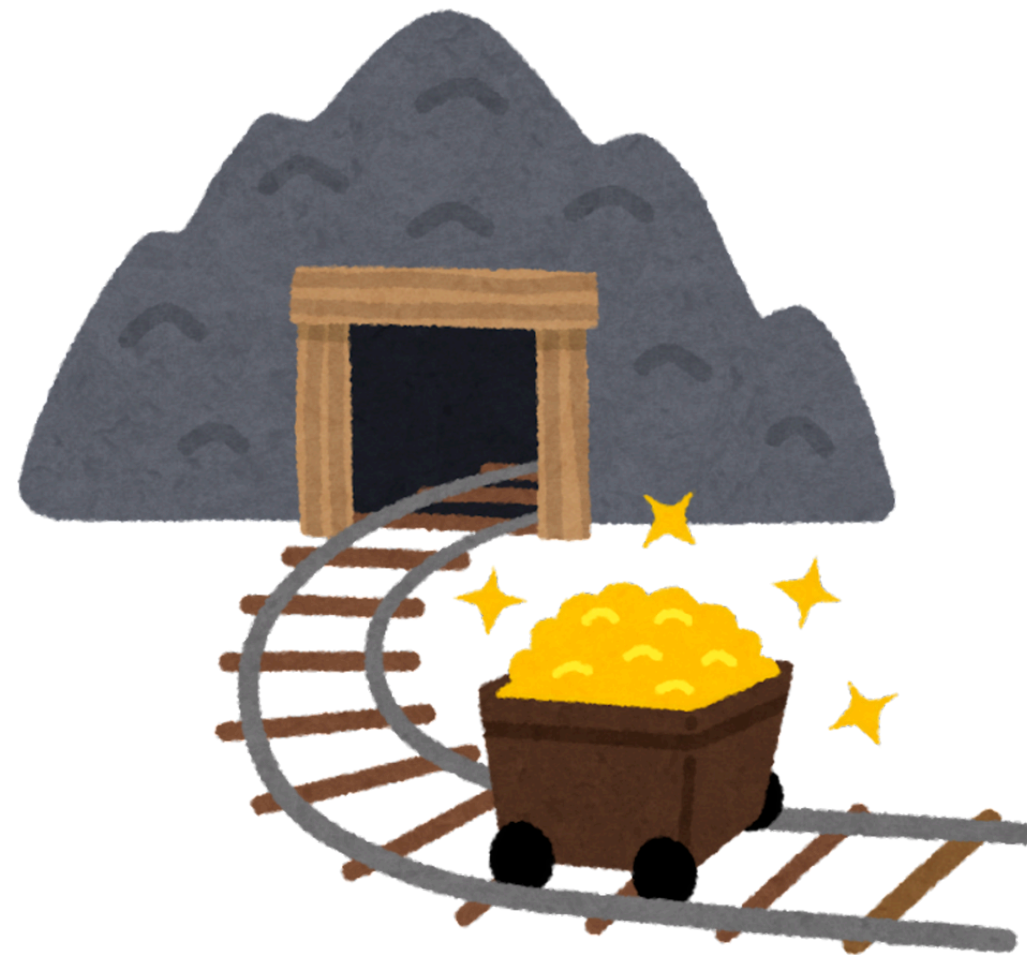
# Introdução



*Fonte: Tecnoblog*

# Introdução

Para minerar blocos, o usuário deve realizar uma tarefa computacionalmente complexa, denominada prova de trabalho (*Proof-of-Work*, PoW).



# Descrição do código

```
class Blockchain:
    def __init__(self):
        self.current_transactions = []
        self.chain = []

        # Cria o bloco de gênese
        self.new_block(previous_hash=1, proof=100)

    def new_block(self, proof, previous_hash=None):
        """
        Cria um novo bloco na blockchain

        :param proof: <int> A prova dada pelo algoritmo de prova de trabalho
        :param previous_hash: <string> Hash do último bloco
        :return: <dict> Novo bloco
        """
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.current_transactions,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
            'proof_of_work': self.proof_of_work(block['proof']),
        }

        # Reseta a atual lista de transações
        self.current_transactions = []

        self.chain.append(block)
        return block

    def new_transaction(self, sender, recipient, amount):
        """
        Cria uma nova transação para ir no próximo bloco minerado

        :param sender: <string> Endereço do pagador
        :param recipient: <string> Endereço do receptor
        :param amount: <float> Valor
        :return: <int> O índice do bloco que guardará esta transação
        """
        self.current_transactions.append({
            'sender': sender,
            'recipient': recipient,
            'amount': amount,
        })

        return self.last_block['index'] + 1

    @staticmethod
    def hash(block):
```

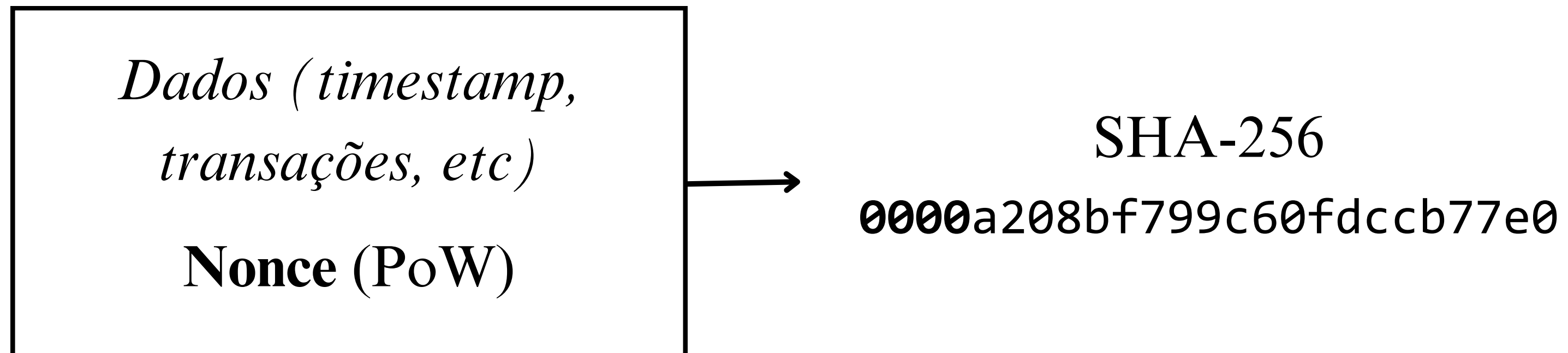
Classe Blockchain com os métodos:

- init
- new\_block
- new\_transaction
- last\_block
- hash, proof\_of\_work, valid\_proof

Comunicação via API.

# Hashcash

Originalmente criado para conter *spam* de e-mails e ataques de negação de serviço.



# Hashcash

```
def proof_of_work(self, last_block):  
    """ ...  
  
    last_proof = last_block['proof']  
    last_hash = self.hash(last_block)  
  
    proof = 0  
    while self.valid_proof(last_proof, proof, last_hash) is False:  
        proof += 1  
  
    return proof
```

```
def valid_proof(last_proof, proof, last_hash):  
    """ ...  
  
    guess = f'{last_proof}{proof}{last_hash}'.encode()  
    guess_hash = hashlib.sha256(guess).hexdigest()  
    return guess_hash[:4] == "0000"
```



# Hashcash

*Tabela 1. Teste de dificuldades do hashcash*

<b>Dificuldade</b>	<b>Tempo de mineração</b>
4 zeros	0,80s
5 zeros	4,98s
6 zeros	66,62s
7 zeros	806,40s $\approx$ 13,44 min

# Hashcash



🔍 Buscar



Energia  
e Ciência

Parceiros



Clima

Energia

Finanças

Sociedade

Biodiversidade

Colunas e Blogs

Últimas Notícias

Quem somos

Manifesto

COP

## Mineradoras de bitcoin gastaram a mesma quantidade de energia que a Austrália no último ano

Órgão da Universidade de Cambridge estima que o gasto para minerar moedas virtuais como o bitcoin representa entre 0,2% a 0,9% da demanda global de energia

Por Gabriela Guido, da Época NEGÓCIOS

10/02/2024 08h01 · Atualizado há 10 meses



# Primecoin

Implementa o conceito de prova de trabalho útil.  
Descobre novas cadeias de Cunningham e bi-gêmeas.

$$p_2 = 2p_1 + 1,$$

$$p_3 = 4p_1 + 3,$$

$$p_4 = 8p_1 + 7,$$

$$\vdots$$

$$p_i = 2^{i-1}p_1 + (2^{i-1} - 1)$$

# Primecoin

Largest known Cunningham chain of length  $k$  (as of 17 March 2023<sup>[2]</sup>)

$k$ ↕	Kind ↕	$p_1$ (starting prime) ▼	Digits ↕	Year ↕	Discoverer ↕
13	1st	106680560818292299253267832484567360951928953599522278361651385665522443588804123392×61# − 1	107	2014	Primecoin ( <a href="#">block 368051</a> ↗)
13	2nd	38249410745534076442242419351233801191635692835712219264661912943040353398995076864×47# + 1	101	2014	Primecoin ( <a href="#">block 539977</a> ↗)
14	2nd	5819411283298069803200936040662511327268486153212216998535044251830806354124236416×47# + 1	100	2014	Primecoin ( <a href="#">block 547276</a> ↗)
12	1st	288320466650346626888267818984974462085357412586437032687304004479168536445314040×83# − 1	113	2014	Primecoin ( <a href="#">block 558800</a> ↗)
11	1st	73853903764168979088206401473739410396455001112581722569026969860983656346568919×151# − 1	140	2013	Primecoin ( <a href="#">block 95569</a> ↗)
14	1st	4631673892190914134588763508558377441004250662630975370524984655678678526944768×47# − 1	97	2018	Primecoin ( <a href="#">block 2659167</a> ↗)

*Fonte: Wikipedia*

# Primecoin

```
def proof_of_work(self):
    """Busca por uma cadeia de Cunningham válida de acordo com a dificuldade."""
    difficulty = 4
    while True:
        base_number = random.randint(2, 10**5)
        chain = self.find_cunningham_chain(base=base_number, length=difficulty)
        if len(chain) == difficulty:
            return chain

def find_cunningham_chain(self, base, length):
    chain = []
    while len(chain) < length:
        if self.is_prime(base):
            chain.append(base)
        else:
            break
        base = 2 * base + 1
    return chain

@staticmethod
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
```

# Primecoin

*Tabela 2. Teste de dificuldades do primecoin*

<b>Dificuldade</b>	<b>Tempo de mineração</b>	<b>Sequência</b>
5 números	0,05s	<i>66749, 133499, 266999, 533999, 1067999</i>
6 números	0,42s	<i>63419, 126839, 253679, 507359, 1014719, 2029439</i>

# Scrypt


Originalmente uma KDF criada para ser resistente à ASICs.

## STRONGER KEY DERIVATION VIA SEQUENTIAL MEMORY-HARD FUNCTIONS

COLIN PERCIVAL

ABSTRACT. We introduce the concepts of memory-hard algorithms and sequential memory-hard functions, and argue that in order for key derivation functions to be maximally secure against attacks using custom hardware, they should be constructed from sequential memory-hard functions. We present a family of key derivation functions which, under the random oracle model of cryptographic hash functions, are provably sequential memory-hard, and a variation which appears to be marginally stronger at the expense of lacking provable strength. Finally, we provide some estimates of the cost of performing brute force attacks on a variety of password strengths and key derivation functions.

# Script



Informe seu CEP

asic bitcoin

☐ Somente em Rigs de Mineração

Escolha o seu plano **meli+** a partir de **R\$9,90**

[Crie a sua conta](#) [Entre](#) [Compras](#)

Buscas relacionadas: mineradora bitcoin - mineradora asic bitcoin - maquina mineradora bitcoin

Mais Categorias > Criptomoedas > Mineração de Criptomoedas > Rigs de Mineração

Asic bitcoin

40 resultados

Frete grátis

por ser sua primeira compra

Marca

Oba Decor (4)

Lilygo (4)

Antminer (3)

P&W (1)

Genérica (1)

Canaan (1)

ArtBox3d (1)

Lojas oficiais


Somente lojas oficiais (2)

Preço

Até R\$ 300 (12)

R\$300 a R\$3.000 (13)

Mais de R\$3.000 (15)




Asic De Bitcoin Antminer S19k Pro 120 Th/s E 2800w Nova

R\$ 24.500

em 12x R\$ 2.354

Frete grátis

Disponível 60 dias após sua compra




Mineradora Antminer Asic De Bitcoin Avalon 50th/s Usada

R\$ 4.900

em 10x R\$ 490 sem juros

Frete grátis por ser sua primeira compra



Mineradora Bitcoin Asic Whatsminer M21s 56th/s \*usado\*

R\$ 3.390

em 10x R\$ 339 sem juros

Frete grátis por ser sua primeira compra



# Scrypt

Usado em criptomoedas como Litecoin e Dogecoin.

```
def proof_of_work(self, last_block):
    """Busca um hash baseado em scrypt que atenda à dificuldade especificada."""
    data = f"{last_block['proof']}{self.hash(last_block)}"
    difficulty = 4
    prefix = "0" * difficulty
    proof = 0
    while True:
        salt = os.urandom(16)
        result = hashlib.scrypt(
            f"{data}{proof}".encode(),
            salt=salt,
            n=2**14, # parâmetros do scrypt
            r=8,
            p=1,
            dklen=32
        )
        if result.hex().startswith(prefix):
            return proof
        proof += 1
```

# Scrypt

*Tabela 3. Teste de dificuldades do scrypt*

<b>Dificuldade</b>	<b>Tempo de mineração</b>
2 zeros	30s
3 zeros	285,1s $\approx$ 4,75 min
4 zeros	excedeu 30 min

# Conclusão

Outros algoritmos:

- equihash
- ethash
- cuckoo cycle
- blake2b
- verifiable delay functions

# Conclusão

Outras formas de consenso, como a prova de participação  
(*Proof-of-Stake*, PoS) usada pela Ethereum 2.0.

**Obrigado.**