Diferentes Funções de Prova de Trabalho em uma Blockchain

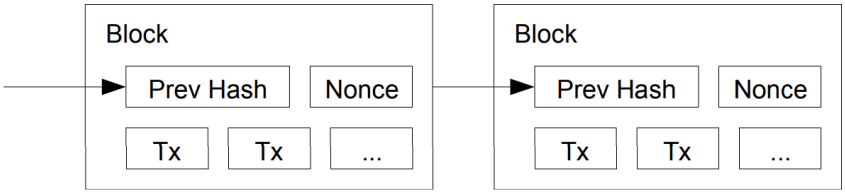
Henrique Chaves Lopes | RA 11202130079

1. Introdução

O advento da blockchain, uma tecnologia proposta no artigo publicado por Satoshi Nakamoto (cuja identidade é desconhecida) em 2008, representou uma ruptura no modo como dados sensíveis, como o registro de transações financeiras, são armazenados. Mas, para além das famosas criptomoedas, cujo surgimento foi concomitante à invenção da blockchain, novas aplicações da tecnologia estão sendo desenvolvidas, inclusive no campo da saúde. Fica, portanto, cada vez mais evidente a grande relevância deste tópico.

As blockchains são, em poucas palavras, bancos de dados descentralizados e confiáveis, implementados sobre redes nas quais cada nó representa um usuário. Na blockchain de Nakamoto, não há requisitos para participar da rede (*permissionless*) e cada usuário tem uma cópia do registro de transações. Conjuntos de transações são guardados em blocos cujos cabeçalhos referenciam, por meio de um hash, o bloco anterior na *chain* (corrente ou cadeia), mantendo a ordem cronológica dos eventos. Embora seja uma rede *permissionless*, e possam haver usuários mal-intencionados, a confiabilidade da blockchain é assegurada por seu algoritmo de consenso e seus mecanismos de provas criptográficas, desde que a maior parte do poder computacional da rede esteja nas mãos de usuários honestos. Pois todo usuário pode minerar blocos — i.e., criar novos blocos e adicioná-los na cadeia — caso resolva uma tarefa computacional, conhecida como prova de trabalho (*Proof of Work*, PoW), mais rápido que os demais. O algoritmo de consenso, por sua vez, garante que todos os nós estejam trabalhando sobre a mesma cadeia, e não sobre um *fork* (bifurcação) gerado por atrasos na rede ou quando dois blocos são minerados ao mesmo tempo.

Figura 1. Diagrama de uma blockchain



Fonte: Satoshi Nakamoto

Dada a sua importância central nas blockchains, o foco deste trabalho são os mecanismos de prova de trabalho. Uma blockchain foi implementada e diferentes funções de prova de trabalho foram testadas. Assim algumas considerações puderam ser feitas.

2. Descrição do código

A implementação da blockchain foi feita com a linguagem de progamação Python, de acordo com o guia explicitado abaixo, nas referências bibliográficas deste trabalho. O código fonte contém a classe **`Blockchain`**, onde foram definidos:

- O método de inicialização, onde cria-se o bloco de gênese.
- O método **`new_block`** que cria um novo bloco e cujos parâmetros são: o hash do bloco anterior e o *nonce* (número) da prova de trabalho.
- O método **`new_transaction`** que cria uma nova transação e cujos parâmetros são: o endereço do pagador, o endereço do recebedor e o valor da transação.
- O método **`hash`** que calcula o hash do bloco passado como parâmetro.
- O método **`last_block`** que retorna o último bloco da blockchain.
- O método **`proof_of_work`** que recebe, como parâmetro, o último bloco e calcula a prova de trabalho.
- O método **`valid_proof`** que valida uma prova de trabalho e cujos parâmetros são: o *nonce* do bloco anterior, o *nonce* atual que será verificado e o hash do bloco anterior.

Os métodos **`proof_of_work`** e **`valid_proof`** serão modificados de acordo com os testes realizados.

Em outro arquivo, usando a biblioteca Flask do Python, construiu-se uma *Application Programming Interface* (API), permitindo a comunicação entre blockchain e usuário. O usuário pode enviar requisições do protocolo *Hyper Text Transfer Protocol* (HTTP) para URLs específicas do servidor, neste caso local, para realizar cada tarefa desejada, como criar uma nova transação ou minerar um bloco.

3. Hashcash (SHA-256)

A função de mineração usada no Bitcoin é uma variação do hashcash, proposto para conter *spam* de e-mails e ataques de negação de serviço em 1997 por Adam Back. Nesta função, um número (*nonce*) é incrementado, ou gerado randomicamente, até que o hash de uma string

concatenando o próprio *nonce*, o hash e o *nonce* do bloco anterior na blockchain, começe com uma determinada quantidade de zeros. O algoritmo de hash usado é o SHA-256.

Figura 2 e 3. Implementação do hashcash em Python

```python
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
```

```python
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"
```

Fonte: De autoria própria

A quantidade de zeros representa o nível de dificuldade da operação, visto que o processamento exigido e o tempo de execução aumentam exponencialmente com essa restrição. A exemplo do Bitcoin, a dificuldade de sua prova de trabalho é ajustada de tempos em tempos, de modo que o tempo de mineração seja sempre próximo de 10 minutos. O primeiro teste realizado envolveu minerar um bloco em diferentes dificuldades do algoritmo. O *hardware* usado foi um dispositivo particular com um processador Intel i5 da 13ª geração. Na tabela 1 vêem-se os resultados.

Tabela 1. Teste de dificuldades do hashcash

Dificuldade	**Tempo de mineração**
4 zeros	0,80s
5 zeros	4,98s
6 zeros	66,62s
7 zeros	806,40s ≈ 13,44 min

Fonte: De autoria própria

É importante ressaltar que, do ponto de vista energético, este algoritmo tem se provado muito ineficiente. Atualmente, para minerar seus blocos, a rede de bitcoins consome mais energia elétrica do que uma gama de países. Por isso, a tecnologia tem sido criticada por ambientalistas.

4. Primecoin

O Primecoin surgiu como uma alternativa ecologicamente responsável do Bitcoin. Sua principal diferença em relação à blockchain original está na aplicação do conceito de prova de trabalho útil. Neste conceito, todo o poder computacional empenhado na mineração de novos blocos é direcionado para resolver problemas que possuem algum valor para a sociedade. No caso da Primecoin, os mineradores devem buscar, como prova de trabalho, cadeias de números primos que sejam classificadas como cadeias de Cunningham de primeira ordem, de segunda ordem ou cadeias bi-gêmeas. Assim, as informações geradas servem para a pesquisa acadêmica em áreas como a Matemática e a Física.

Uma cadeia de Cunningham de primeira ordem é uma sequência de números primos que obedece à fórmula: (p, $2p + 1$, $4p + 3$, $8p + 7$, ...). Abaixo, encontra-se uma possível implementação em Python de um algoritmo capaz de encontrar uma de tais cadeias.

Figura 4. Implementação da prova de trabalho do Primecoin em Python

```python
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
```

Fonte: De autoria própria

A dificuldade deste algoritmo pode ser ajustada alterando o tamanho da sequência de números primos que deve ser encontrada. O segundo teste realizado, como o anterior, comparou o tempo de mineração de um bloco nos diferentes níveis de dificuldade do algoritmo.

Tabela 2. Teste de dificuldades do primecoin

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

Fonte: De autoria própria

O tempo de mineração, no *hardware* testado, excede 30 minutos quando a dificuldade é aumentada para 7 números. É difícil para o computador verificar a primalidade de números muito grandes. Obserava-se, então, que apesar da prova de trabalho ter utilidade neste caso, o consumo de energia é tão exacerbado quanto no caso do Bitcoin.

5. Scrypt

O scrypt é uma função de derivação de chave baseada em senha (*Password-Based Key Derivation Function*), ou seja, um algoritmo capaz de gerar chaves criptográficas seguras a partir de uma entrada, neste caso, uma senha. Scrypt foi projetado em 2009 por Colin Percival com o propósito de ser resistente à ataques de força bruta realizados por *hardwares* especializados, como ASICs (*Application Specific Integrated Circuit* ou circuito integrado de aplicação específica). Sendo assim, o algoritmo consome mais recursos, como memória, para o tornar custoso em operações de grande escala.

O fato de o scrypt demandar certo esforço computacional e ser resistente a ASICs atraiu a atenção de desenvolvedores de criptomoedas, pois, em meados de 2011, GPUs e ASICs haviam se tornado a maioria dos mineradores da rede Bitcoin. Desta forma, scrypt foi adaptado para funcionar como um algoritmo hash alternativo ao SHA-256, podendo ser usado como função de prova de trabalho em blockchains. Algumas criptomoedas, como Litecoin e Dogecoin, utilizam este algoritmo. Abaixo encontra-se uma possível implementação em Python.

Figura 5. Implementação do scrypt em Python

```python
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
```

Fonte: Autoria própria

Como no hashcash, a quantidade de zeros exigida no hash controla a dificuldade de mineração neste algoritmo. No entanto, neste caso também podemos controlar os parâmetros da função scrypt, que estão associados ao uso de memória. Em seguida são apresentados os resultados do terceiro teste.

Tabela 3. Teste de dificuldades do scrypt

Dificuldade	Tempo de mineração
2 zeros	30s
3 zeros	285,1s ≈ 4,75 min
4 zeros	excedeu 30 min

Fonte: Autoria própria

Apesar da proposta original, atualmente existem *hardwares* especializados em minerar cripto-moedas baseadas em scrypt. Além disso, como nos outros testes realizados, é possível verificar que esta função de prova de trabalho também consome muita energia.

6. Conclusão

Além das funções testadas, existem outras inúmeras. Para citar algumas: equihash, ethash, cuckoo cycle, blake2b e verifiable delay functions. Cada uma com sua proposta particular para resolver os problemas do uso eficiente de energia e dos dispositivos especializados. E, vale

ressaltar, existem também outras formas de consenso que não dependem de prova de trabalho. Como exemplo, a criptomoeda Ethereum 2.0 passou a utilizar, no lugar da prova de trabalho, o consenso baseado em prova de participação (*Proof-of-Stake*, PoS).

Sendo assim, o presente trabalho deixou evidente como o processo constante de evolução da segurança se desdobrou, e continua acontecendo, no contexto das blockchains.

Referências

NAKAMOTO, Satoshi. Bitcoin: a peer-to-peer electronic cash system. **Bitcoin**, 2008. Disponível em: https://bitcoin.org/bitcoin.pdf. Acesso em: 12 de dez. de 2024.

CONTERNO, Ivan. Tecnologia blockchain pode reduzir risco de roubos e fraudes na área da saúde. **Jornal da USP**, São Paulo, 29 de mai. de 2023. Disponível em: https://jornal.usp.br/ciencias/tecnologia-blockchain-pode-reduzir-risco-de-roubos-e-fraudes-na-area-da-saude/. Acesso em: 10 de dez. de 2024.

GAYVORONSKAYA, T.; MEINEL, C. **Blockchain**: hype or innovation. Springer, 2020.

TRISTIANTO, Bimo Putro. Build your own blockchain in Python: a practical guide. **Medium**, 2022. Disponível em: https://bimoputro.medium.com/build-your-own-blockchain-in-python-a-practical-guide-f9620327ed03. Acesso em: 10 de dez. de 2024.

Proof of work. **Wikipedia**, 2024. Disponível em: https://en.wikipedia.org/wiki/Proof_of_work. Acesso em: 12 de dez. de 2024.

Hashcash. **Wikipedia**, 2024. Disponível em: https://en.wikipedia.org/wiki/Hashcash. Acesso em: 12 de dez. de 2024.

BACK, Adam. Hashcash - a denial of service counter-measure. **Hashcash**, 2002. Disponível em: http://www.hashcash.org/papers/hashcash.pdf. Acesso em: 13 de dez. de 2024.

HUYNH, A. N. Q. et al. Energy consumption and Bitcoin market. **Asia-Pacifc Financial Markets**, 29, 79-93, 2022. Disponível em: https://link.springer.com/article/10.1007/s10690-021-09338-4. Acesso em: 13 de dez. de 2024.

GUIDO, Gabriela. Mineradoras de bitcoin gastaram a mesma quantidade de energia que a Austrália no último ano. **Um Só Planeta**, 10 de fev. de 2024. Disponível em: https://umsoplaneta.globo.com/energia/noticia/2024/02/10/mineradoras-de-bitcoin-gastaram-a-mesma-quantidade-de-energia-que-a-australia-no-ultimo-ano.ghtml. Acesso em: 13 de dez. de 2024.

Primecoin. **Wikipedia**, 2024. Disponível em: https://pt.wikipedia.org/wiki/Primecoin. Acesso em: 13 de dez. de 2024.

KING, Sunny. Primecoin: cryptocurrency with prime number proof-of-work. **Primecoin**, 2013. Disponível em: https://web.archive.org/web/20131103035723/http://primecoin.org/static/primecoin-paper.pdf. Acesso em: 13 de dez. de 2024.

Scrypt. **Wikipedia**, 2024. Disponível em: https://pt.wikipedia.org/wiki/Scrypt. Acesso em: 14 de dez. de 2024.

PERCIVAL, Colin. Stronger key derivation via sequential memory-hard functions. **Tarsnap**, 2009. Disponível em: https://www.tarsnap.com/scrypt/scrypt.pdf. Acesso em: 14 de dez. de 2024.