# Otimização do Desempenho de Hashes com ParallelHash para Aceleração do Hyperledger Fabric

**João Pedro Correa Crozariolo**[1], **Alexandre Augusto Giron**[1]

[1] Universidade Tecnológica Federal do Paraná (UTFPR)
Toledo – PR – Brazil

jcrozariolo@alunos.utfpr.edu.br, alexandregiron@utfpr.edu.br

***Abstract.*** *Hyperledger Fabric's integrity relies on hash functions, but sequential standards like SHA-2 create bottlenecks in large-data operations like ledger snapshot verification. This work investigates replacing SHA-2 with the ParallelHash family by implementing and benchmarking ParallelHash128/256 in Go, demonstrating significant throughput gains. The practical integration challenges are also analyzed, identifying hardcoded limitations in Fabric's source code that hinder crypto agility. The findings offer a roadmap for enhancing Fabric's performance using modern cryptographic primitives.*

***Resumo.*** *A integridade do Hyperledger Fabric depende de funções de hash, mas padrões sequenciais como o SHA-2 criam gargalos em operações com grandes volumes de dados, como a verificação de snapshots. Este trabalho investiga a substituição do SHA-2 pela família ParallelHash, por meio da implementação e teste das funções ParallelHash128/256 em Go, o que demonstrou ganhos de vazão significativos. Adicionalmente, são analisados os desafios práticos de integração, com a identificação de limitações no código-fonte do Fabric que dificultam a crypto agility. Os resultados oferecidos criam um roteiro para otimizar o desempenho do Fabric com primitivas criptográficas modernas.*

## 1. Introdução

Blockchain é usada em aplicações com alta demanda de integridade e rastreabilidade. O *Hyperledger Fabric* [Linux Foundation 2023], mantido pela Linux Foundation, destaca-se entre os frameworks permissionados, com uso em rastreamento de cadeias de suprimento, contratos interinstitucionais e gestão de identidades. Nesse contexto, funções de hash criptográficas garantem a integridade do sistema no encadeamento de blocos, validação de transações e verificação de estados.

A eficiência dessas funções de hash é um fator crítico para a escalabilidade da plataforma. Padrões estabelecidos como o SHA-2, apesar de seguros, possuem uma natureza inerentemente sequencial. Embora transações individuais sejam tipicamente pequenas, essa característica sequencial resulta em gargalos de desempenho durante operações que manipulam grandes volumes de dados, como na verificação de snapshots do ledger, no processamento de dados off-chain ou na sincronização inicial de novos pares na rede. Diante desse cenário, torna-se fundamental explorar alternativas de hashes paralelizáveis.

Este trabalho foca na análise da família *ParallelHash*, especificada na NIST SP 800-185 [Dang 2016], contextualizando seu desempenho frente a outras alternativas modernas, como o BLAKE3. Contudo, a otimização do desempenho é apenas

parte da solução. A substituição de um algoritmo criptográfico fundamental em um sistema distribuído é uma tarefa complexa que evidencia a necessidade de *crypto agility* [National Institute of Standards and Technology (NIST) 2023], a capacidade de uma plataforma migrar suas primitivas criptográficas de forma segura e eficiente. A falta dessa agilidade representa uma barreira prática significativa, que é também investigada neste artigo.

Assim, as principais contribuições deste trabalho são: (i) a implementação em Go das funções *ParallelHash128* e *ParallelHash256*, com variantes otimizadas para paralelismo com *goroutines* [The Go Authors 2024]; (ii) uma análise de desempenho sistemática que valida os ganhos de vazão em cenários de dados volumosos; e (iii) a análise dos desafios práticos e barreiras à sua integração no *Hyperledger Fabric*, fornecendo um diagnóstico sobre a *crypto agility* da plataforma.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta os fundamentos técnicos; a Seção 3 descreve a metodologia experimental e propõe a integração do *ParallelHash* no *Hyperledger Fabric*; a Seção 4 discute os resultados preliminares obtidos; e a Seção 5 apresenta as conclusões e trabalhos futuros.

## 2. Fundamentação Teórica

### 2.1. Hyperledger Fabric e Funções de Hash

O *Hyperledger Fabric* [Linux Foundation 2023] é um framework de blockchain permissionado para aplicações corporativas, que garante integridade via consenso, assinaturas digitais e funções de hash. Sua arquitetura opera com nós *endorsers*, *committers* e *orderers* sob o modelo *execute-order-validate*, que permite maior paralelismo e desempenho que modelos tradicionais [Androulaki et al. 2018]. A integridade do sistema é ancorada em hashes como SHA-2 e SHA-3 [of Standards and Technology 2015], cuja natureza sequencial limita o desempenho em operações com grandes volumes de dados. Como alternativa, a especificação NIST SP 800-185 [Dang 2016] propõe a família *ParallelHash*, que processa blocos de dados em paralelo.

### 2.2. Trabalhos Relacionados

A otimização de primitivas criptográficas no Fabric foi abordada por Holcomb et al. [Holcomb et al. 2021], que identificaram o gargalo de hashes sequenciais e sugeriram o uso de funções paralelas. Entre as alternativas modernas, destaca-se também a BLAKE3 [**?**] por sua alta performance. Nosso trabalho avança essa discussão ao focar no *ParallelHash*, por ser um padrão NIST, e ao realizar um benchmark prático no ecossistema Go, preenchendo uma lacuna na literatura.

Adicionalmente, a simples substituição de algoritmos conecta-se ao desafio prático da *crypto agility* [National Institute of Standards and Technology (NIST) 2023]. A análise das barreiras para alcançar essa agilidade em uma plataforma como o Fabric é uma contribuição chave do nosso trabalho, diferenciando-o de estudos que focam apenas em benchmarks de desempenho.

## 3. Metodologia Experimental e Análise de Integração

Este trabalho propõe a substituição das funções de hash sequenciais utilizadas no Fabric por algoritmos paralelizáveis da família *ParallelHash*, definidos na especificação

SP 800-185 do NIST [Dang 2016]. As funções *ParallelHash128* e *ParallelHash256*, instanciadas via *cSHAKE*, preservam as propriedades criptográficas da SHA-3 [of Standards and Technology 2015] e permitem ganhos de desempenho por meio da divisão da entrada em blocos processáveis em paralelo.

A implementação foi realizada em Go, com versões sequenciais e paralelas usando `goroutines`, conforme descrito pelos autores do Go [The Go Authors 2024], preenchendo uma lacuna prática da linguagem. A lógica da versão paralela segue o pseudocódigo da Figura 1.

| **Pseudocódigo** |
|---|
| 1. Dividir $M$ em blocos $M_i$ de tamanho $B$ |
| 2. Para cada bloco $M_i$, em paralelo: |
| 2.1 Calcular $Z_i = \mathsf{cSHAKE128}(M_i, 256, , )$ |
| 3. Concatenar todos os $Z_i$ em $Z$ |
| 4. Concatenar $Z$ com $\mathsf{right\_encode}(n)$ e $\mathsf{right\_encode}(L)$ |
| 5. Retornar $\mathsf{cSHAKE128}(Z, L, "ParallelHash", S)$ |

**Figura 1. Pseudocódigo simplificado da função `ParallelHash128Goroutines`.**

Para avaliar o desempenho, as implementações propostas foram comparadas com padrões NIST consolidados. É importante notar que os algoritmos selecionados (SHA3-256, SHA3-512, SHAKE128, SHAKE256) possuem diferentes níveis de segurança e tamanhos de saída. A intenção não é realizar uma comparação de desempenho estritamente ”justa”entre níveis de segurança distintos, mas sim mapear a performance do ParallelHash em relação a um amplo espectro de funções padrão usadas na prática. A análise de resultados na Seção 4 agrupará as comparações por níveis de segurança equivalentes para uma discussão mais equitativa. As variantes desenvolvidas são: PH128/PH256 (sequenciais) e PH128-G/PH256-G (paralelas com `goroutines`).

Os testes consideraram entradas de 1KB a 1GB, geradas via `/dev/urandom`[1] para evitar compressibilidade, e executados em máquina virtual dedicada no Google Cloud (e2-standard-4, 4 vCPUs, 16GB de RAM) [Platform 2023]. Cada configuração foi avaliada com 200 repetições, regenerando arquivos a cada 20 execuções. Métricas como vazão (MB/s), tempo médio (ns/op), desvio padrão, mínimo e máximo foram extraídas usando `pprof` e scripts em `Python`.

### 3.1. Desafios para Adoção do ParallelHash no Fabric

Embora o Fabric permita certa flexibilidade na escolha de algoritmos de hash em alguns componentes, como na configuração do *Membership Service Provider* (MSP) para identidades digitais [Linux Foundation 2023], uma análise do código-fonte revela limitações importantes para a adoção plena de algoritmos alternativos.

Em particular, observou-se que:

- Em partes críticas, como em `txutils.go`, a função SHA256 encontra-se fixada diretamente no código, sem possibilidade de configuração dinâmica [Hyperledger Fabric Developers 2024].

[1]Linux manpage: `https://man7.org/linux/man-pages/man4/urandom.4.html`

- O mecanismo de assinatura utilizado no consenso, baseado no paradigma *hash-then-sign*, também presume o uso de SHA256 de forma hardcoded [Hyperledger Fabric Developers 2024].
- Funções de verificação de assinaturas no cluster de ordering nodes seguem a mesma limitação [Hyperledger Fabric Developers 2024].
- Outras ferramentas auxiliares, como o `cryptogen`, empregam SHA256 diretamente em suas operações internas [Hyperledger Fabric Developers 2024].

Esses fatores indicam que, apesar do suporte a *crypto agility* em áreas específicas, uma substituição completa do algoritmo de hash no Fabric exigiria modificações profundas no código-fonte, ou a adoção de mecanismos de transição mais sofisticados, como recalculação de hashes ou hard-fork controlado, conforme abordado em trabalhos relacionados [Holcomb et al. 2021].

Assim, propõe-se como diretriz para futuras implementações a criação de uma camada de configuração de algoritmo de hash a nível de protocolo, de forma que a escolha do algoritmo possa ser governada dinamicamente pela rede sem necessidade de alterações disruptivas no código.

## 4. Resultados e Discussão

Esta seção apresenta os resultados dos benchmarks de desempenho, obtidos com 200 execuções por combinação de algoritmo e tamanho de entrada. A Figura 2 exibe a vazão média (MB/s) para entradas de 1KB e 1GB. Conforme antecipado na Seção 3, a análise a seguir agrupa os algoritmos por níveis de segurança equivalentes para uma comparação equitativa.

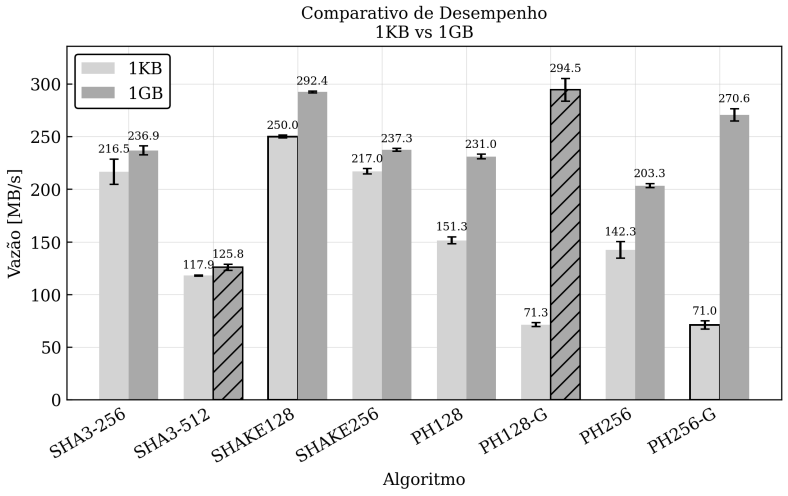


**Figura 2. Vazão média (MB/s) com desvio padrão para entradas de 1KB e 1GB.**

### 4.1. Análise por Nível de Segurança

Nível de 128 bits (PH128/PH128-G vs. SHAKE128): Para entradas pequenas (1KB), o SHAKE128 apresenta a maior vazão, superando as variantes do ParallelHash. Este resultado é esperado, pois a sobrecarga computacional para inicializar as *goroutines* e gerenciar os blocos paralelos do PH128-G não é compensada pelo baixo volume de dados.

Em contrapartida, no cenário com entradas grandes (1GB), o PH128-G demonstra sua vantagem, alcançando a maior vazão entre todos os algoritmos testados. Esse ganho é particularmente relevante para os cenários de uso intensivo de dados no Fabric, como a verificação de snapshots do ledger.

Nível de 256 bits (PH256/PH256-G vs. SHA3-256/SHAKE256): Um padrão similar é observado neste grupo. Para 1KB, o SHAKE256 lidera, enquanto para 1GB, a versão paralela PH256-G se destaca, superando as alternativas sequenciais. O algoritmo SHA3-512, incluído como um ponto de referência de maior segurança, apresenta, como esperado, a menor vazão em ambos os cenários devido ao seu maior estado interno e complexidade.

### 4.2. Discussão sobre Limitações de Desempenho

Um resultado notável é que a margem de vantagem do PH128-G sobre o SHAKE128 para entradas de 1GB, embora clara, não foi tão expressiva quanto teoricamente esperado. Acredita-se que a causa seja a otimização de baixo nível no runtime da linguagem Go. O pacote padrão `crypto/sha3` inclui implementações em *assembly* para arquiteturas AMD64, que permitem ao SHAKE128 aproveitar instruções vetoriais avançadas (AVX2/AVX512). A máquina virtual utilizada nos testes (e2-standard-4) pertence a uma classe de processadores (Skylake) que suporta tais instruções [Platform 2023].

A nova implementação do ParallelHash, por ser puramente em Go e não possuir essas otimizações em *assembly*, não consegue explorar esse recurso de hardware. Essa diferença justifica o desempenho competitivo do SHAKE128 mesmo em grandes volumes e reforça um ponto crucial: a escolha de algoritmos criptográficos em cenários de alto desempenho não deve considerar apenas a complexidade teórica, mas também o nível de otimização da implementação e a arquitetura de hardware subjacente. Os resultados gerais, contudo, sugerem a viabilidade da integração do ParallelHash ao Fabric.

## 5. Conclusões

Este trabalho teve um duplo objetivo: avaliar o desempenho da família de hashes paralelos *ParallelHash* e investigar os desafios práticos para sua adoção no Hyperledger Fabric. Os resultados experimentais confirmaram que as variantes paralelas (PH128-G e PH256-G) superam os padrões sequenciais em cenários com grandes volumes de dados,validando seu potencial para otimizar o Fabric. A análise também revelou que otimizações de baixo nível (*assembly*) em bibliotecas padrão podem mitigar essa vantagem, uma consideração crucial para a escolha de algoritmos na prática.

Tão importante quanto a análise de desempenho, a investigação do código-fonte do Fabric expôs barreiras significativas à *crypto agility*. A presença de funções de hash como a SHA256 fixadas em código (”hardcoded”) em componentes críticos demonstra que o maior obstáculo para a modernização criptográfica da plataforma não é a escolha de um novo algoritmo, mas a necessidade de refatorações para permitir flexibilidade.

Como trabalhos futuros, propõe-se o desenvolvimento de um protótipo que implemente a camada de configuração de algoritmos de hash sugerida neste artigo. Tal protótipo permitiria não apenas a integração efetiva do ParallelHash, mas também de outras funções como o BLAKE3, possibilitando uma avaliação completa do impacto no desempenho.

## Referências

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A. D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolic, M., Cocco, S. W., and Yellick, J. (2018). Hyperledger fabric: A distributed operating system for permissioned blockchains. *Proceedings of the Thirteenth EuroSys Conference*, pages 1–15.

Dang, Q. (2016). Sha-3 derived functions: cshake, kmac, tuplehash and parallelhash. Technical Report NIST SP 800-185, National Institute of Standards and Technology (NIST).

Holcomb, A., Pereira, G., Das, B., and Mosca, M. (2021). Pqfabric: A permissioned blockchain secure from both classical and quantum attacks. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE.

Hyperledger Fabric Developers (2024). Hyperledger fabric source code analysis: txutils.go, signer.go, util.go, cryptogen/ca.go. `https://github.com/hyperledger/fabric`. Accessed April 2025.

Linux Foundation (2023). Hyperledger fabric documentation. `https://hyperledger-fabric.readthedocs.io/`. Accessed April 2025.

National Institute of Standards and Technology (NIST) (2023). Understanding and improving crypto agility. Technical Report NIST CSWP 39, U.S. Department of Commerce. Accessed April 2025.

of Standards, N. I. and Technology (2015). Sha-3 standard: Permutation-based hash and extendable-output functions. Technical Report FIPS PUB 202, U.S. Department of Commerce.

Platform, G. C. (2023). Machine types - google cloud. GCP Documentation. `https://cloud.google.com/compute/docs/machine-types`.

The Go Authors (2024). The go programming language. `https://golang.org/doc/`. Accessed April 2025.