Trabalho de Engenharia de Sistemas Distribuídos

Hash em Sistemas Distribuídos

Equipe:

- Francisco Emmanuel Marcolino Lino dos Santos

- Andre Lopes de Sousa e Oliveira

- Elias Victor de Andrade Torres

- Matheus Eduardo Bezerra do Nascimento

- Anna Myllenne Araujo dos Santos

**Conceito**

Uma função hash é um algoritmo que transforma uma entrada de tamanho arbitrário (como um arquivo, uma mensagem, uma senha, ou qualquer outro tipo de dado) em uma saída de tamanho fixo, chamada de hash ou digest.

A função hash é como uma “impressão digital” do dado: cada entrada tem um hash único (ou, pelo menos, extremamente improvável de se repetir). Isso permite que possamos identificar e verificar dados sem precisar conhecer ou transmitir a informação completa.

Exemplos básicos e teóricos:

* Entrada: "Olá, mundo!"
* Hash MD5: "65a8e27d8879283831b664bd8b7f0ad4" (128 bits)
* HashSHA-256: "c0535e4be2b79ffd93291305436bf889314e4a3faec05ecffcbb7df31ad5f6a0" (256 bits)

Mesmo que seja modificada apenas uma letra, o hash muda completamente (isso é chamado de avalanche effect, que vamos detalhar depois).

**Propriedades Fundamentais de Funções Hash Criptográficas**

Para uma função hash ser considerada criptograficamente segura, ela deve atender a certas propriedades essenciais:

* Determinística
  + A mesma entrada sempre gera o mesmo hash.
  + Exemplo (ambos iguais):
    - hash\_md5("abc") -> "900150983cd24fb0d6963f7d28e17f72"
    - hash\_md5("abc") -> "900150983cd24fb0d6963f7d28e17f72"
* Velocidade no Cálculo
  + A função hash deve ser computacionalmente eficiente, mesmo para entradas grandes;
  + Importante em aplicações como verificação de integridade de arquivos ou autenticação de senha.
* Resistência à pré-imagem
  + Dado um hash H, é computacionalmente inviável descobrir a entrada X original que gerou H;
  + Exemplo prático: armazenar senhas em hash em vez de texto plano;
  + Mesmo que alguém veja o hash, ele não consegue reverter para a senha original.
* Resistência à segunda pré-imagem
  + Dado um dado X1 e seu hash H, é praticamente impossível encontrar outro dado X2 ≠ X1 que produza o mesmo hash;
  + Isso impede ataques de falsificação: não se pode criar um documento diferente que tenha o mesmo hash do documento original.
* Resistência a colisões
  + Uma colisão ocorre quando dois dados diferentes produzem o mesmo hash;
  + Funções seguras devem tornar essa possibilidade extremamente improvável;
  + Alguns algoritmos antigos, como MD5 e SHA-1, já possuem colisões conhecidas, e por isso não são recomendados para segurança.
* Avalanche Effect
  + Uma pequena mudança na entrada deve causar uma mudança drástica no hash.
  + Exemplo:
    - hash\_md5("abc") -> 900150983cd24fb0d6963f7d28e17f72
    - hash\_md5("abd") -> 7ac66c0f148de9519b8bd264312c4d64
    - Saída de tamanho fixo.
* Saída de tamanho fixo
  + Independentemente do tamanho da entrada, a saída do hash tem tamanho constante.
  + Exemplos:
    - MD5 → 128 bits
    - SHA-1 → 160 bits
    - SHA-256 → 256 bits
    - SHA3-256 → 256 bits

**Aplicações Práticas de Funções Hash**

As funções hash são essenciais em várias áreas de TI e segurança:

* Integridade de dados
  + Garantem que arquivos ou mensagens não foram alterados;
  + Exemplo: baixar um arquivo de software e verificar seu hash fornecido pelo desenvolvedor.
* Armazenamento de senhas
  + As senhas nunca são armazenadas em texto plano;
  + Apenas o hash da senha é armazenado;
  + Mesmo que o banco de dados seja comprometido, não é trivial recuperar a senha.
* Assinaturas digitais e criptografia
  + Hashes são usados para assinar digitalmente documentos;
  + Garantem que o documento não foi alterado e que a assinatura é válida.
* Blockchain e sistemas distribuídos
  + Cada bloco contém o hash do bloco anterior, formando uma cadeia;
  + Isso garante integridade e imutabilidade: qualquer alteração em um bloco quebra toda a cadeia.

**Comparação entre os principais algoritmos**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Tamanho do Hash** | **Segurança Atual** | **Observações** |
| MD5 | 128 bits | Quebrado | Colisões conhecidas, só serve para checksum ou retrocompatibilidade |
| SHA-1 | 160 bits | Inseguro | Colisões encontradas; não recomendado para segurança |
| SHA-2 | 256 bits | Seguro | Padrão de mercado; usado em SSL, blockchain e assinaturas digitais |
| SHA-3 | 256 bits | Muito Seguro | Design novo, resistente a ataques futuros; melhor avalanche effect |

**Conceitos de Distribuição e Avalanche**

* Funções hash seguras devem distribuir uniformemente os bits na saída.
* Uma pequena mudança na entrada deve alterar quase todos os bits do hash.
* Isso é fundamental para:
  + Evitar padrões previsíveis;
  + Garantir que ataques de colisão sejam difíceis;
  + Proteger dados em sistemas distribuídos.

**Hash em Sistemas Distribuídos: Conceito e Propósito**

Em sistemas distribuídos, temos múltiplos nós ou servidores que compartilham dados ou tarefas. O desafio é gerenciar esses dados de forma eficiente, segura e balanceada.

Funções hash são fundamentais porque permitem:

* Localização determinística de dados:
  + Um dado (por exemplo, o ID de um usuário ou um arquivo) pode ser transformado em um índice ou endereço lógico que determina em qual nó ele será armazenado;
  + Isso garante que todos os nós possam encontrar rapidamente o dado, sem necessidade de consultar toda a rede.
* Distribuição equilibrada (Load Balancing)
  + A saída da função hash é praticamente aleatória e uniforme, então os dados se distribuem de forma equilibrada entre os nós;
  + Isso evita que alguns nós fiquem sobrecarregados enquanto outros ficam ociosos;
  + Algoritmos como hashing consistente usam essa propriedade para minimizar o movimento de dados quando nós são adicionados ou removidos.
* Verificação de integridade e detecção de alterações
  + Ao transmitir ou replicar dados entre nós, cada bloco ou arquivo recebe um hash;
  + Quando outro nó recebe esses dados, ele calcula o hash localmente e compara com o hash original;
  + Se houver diferença, o dado foi corrompido ou alterado;
  + Isso é crítico em sistemas como HDFS, Ceph, IPFS e blockchain.
* Segurança e imutabilidade
  + Em sistemas distribuídos que lidam com transações ou blocos de dados (como blockchain), cada bloco contém:
    - O hash do bloco anterior;
    - O hash do seu próprio conteúdo.
  + Dessa forma, qualquer alteração em um bloco invalida toda a cadeia;
  + O hash garante integridade, auditabilidade e segurança criptográfica, mesmo em um sistema totalmente distribuído.
* Deduplicação de dados
  + Hashes permitem identificar arquivos ou blocos de dados duplicados sem comparar o conteúdo byte a byte;
  + Se dois arquivos diferentes têm o mesmo hash, é muito provável que sejam iguais, economizando armazenamento.