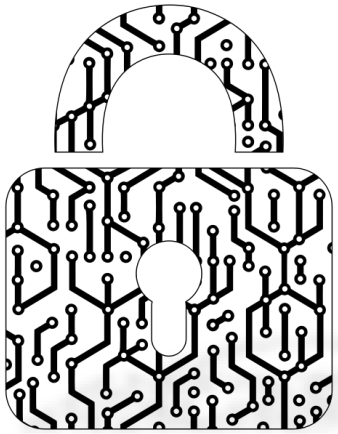


Blockchain, Criptomoedas & Tecnologias Descentralizadas

Construções criptográficas avançadas com hashes

Prof. Dr. Marcos A. Simplicio Jr. – mjunior@larc.usp.br
Escola Politécnica, Universidade de São Paulo

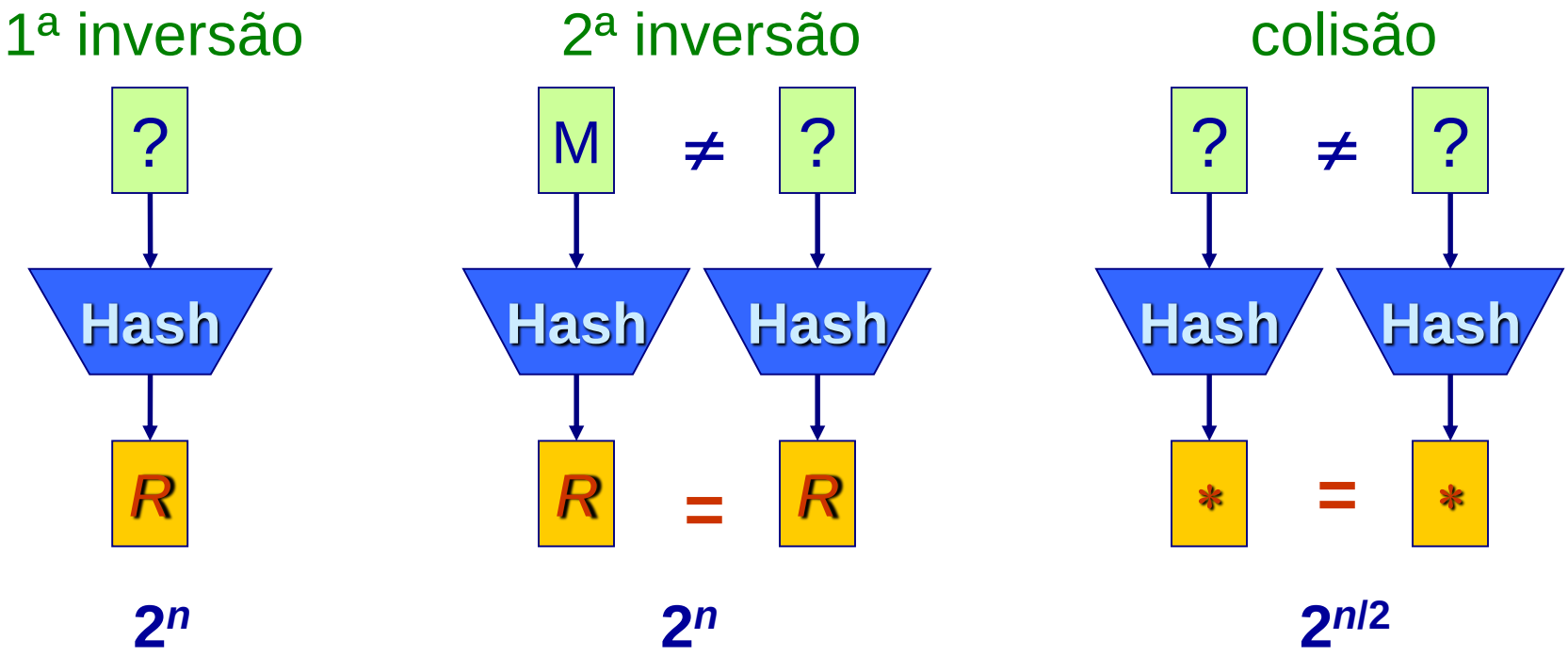
Objetivos

- Algumas construções avançadas explorando versatilidade de funções de hash
  - Mecanismos de compromisso (*commitment*)
    - Protocolos de sorteio justo
  - Cadeias de hash (*hash chains*)
    - Autenticação, assinaturas baseadas em hash
  - Merkle Trees
    - Densas, ordenadas, esparsas

Relembrando...

- Funções de hash: propriedades de segurança

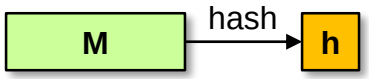
➔ E o que dá pra fazer com isso?



Custo do ataque: hash de n bits

Commitment

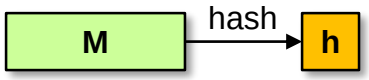
- Mecanismo de compromisso (*commitment*) p/ dados M



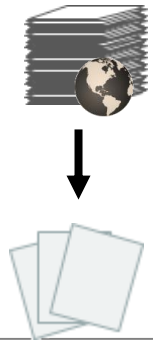
- Procedimento:
  - Alice revela a Bob h = Hash(M)
  - Posteriormente, Alice revela M' satisfazendo Hash(M') = h
  - Bob se convence que M = M'
    - Assumindo resistência a colisão
- Diz-se que Alice "se comprometeu" com M ao relevar seu hash: depois disso, ela não pode apresentar M' ≠ M
  - Mas até Alice revelar M, seu valor permanece secreto
  - Se M tem baixa entropia (e.g., 1 dígito), o comprometimento pode ser feito sobre (r , M), onde r é um número aleatório

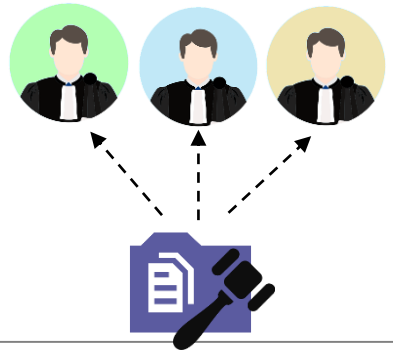
Commitment: uso

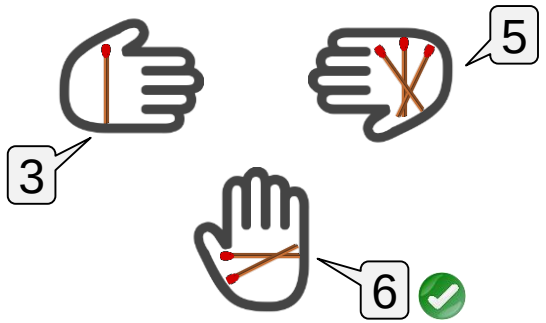
- Mecanismo de compromisso (*commitment*) p/ dados M



- **Sorteio justo** entre n nós.
  - *Integridade*: resistente a tentativas de manipulação
  - *Verificabilidade*: todos os nós devem ser capazes de chegar ao mesmo resultado usando as informações recebidas (não requer confiança em terceiros)
  - Ex.: jogos em cenário distribuído; sorteios em aplicações críticas (e.g., sorteio de juizes do supremo); “purrinha”



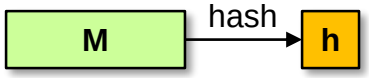




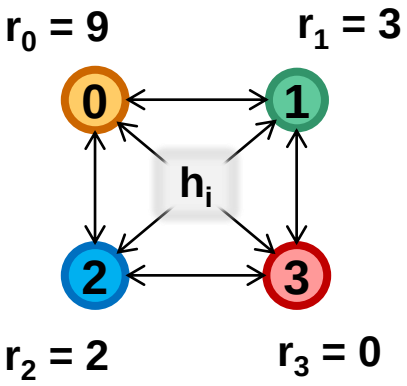
Vide [SSL+14]

Commitment: uso

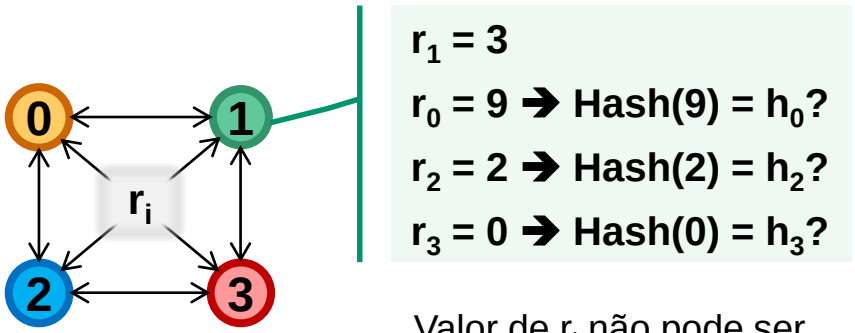
- Mecanismo de compromisso (*commitment*) p/ dados M



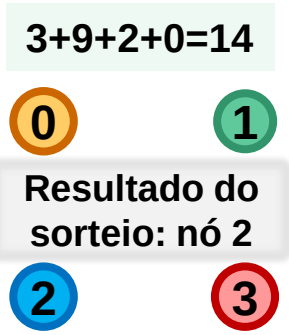
- **Sorteio justo** entre n nós. Ex.: n = 4
  - Requisito: todos os nós da rede são conhecidos
  - *Commit*: nó N_i gera r_i aleatório e faz broadcast de $h_i = Hash(r_i)$
  - *Reveal*: após receber todos os $h_{j \neq i}$, N_i faz broadcast de r_i
  - *Sorteio*: nó sorteado é N_k, onde $k = Soma(r_i) \bmod n$



1. *Commit*: $h_i = Hash(r_i)$



2. *Reveal*: broadcast de r_i

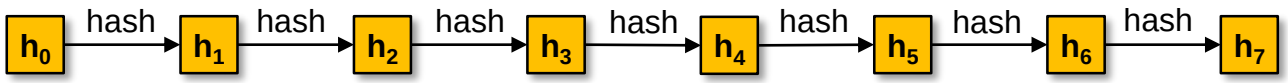


3. Sorteio: 14 mod 4 = 2

Vide [SSL+14]

Hash Chains

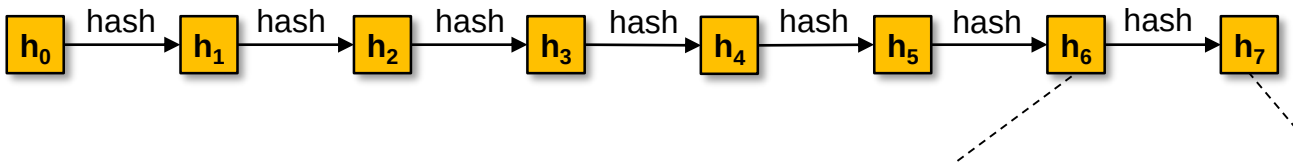
- Ligação criptográfica forte entre hashes sucessivos
  - Aproveita não-inversibilidade (resistência à 1a inversão)



- Construção:
  - h_0 aleatório: $h_0 \xleftarrow{\$} \{0,1\}^*$
  - Para todo $0 < i < n$: calcular $h_i = Hash(h_{i-1})$
  - Valor secreto: h_0 ; valor público: h_{n-1}
- Propriedades da construção:
  - Conhecimento de h_i não permite a terceiros determinar h_{i-1}
  - Apenas dono do valor secreto h_0 correspondente a h_{n-1} é capaz de revelar h_i satisfazendo $h_{n-1} = Hash^{n-i-1}(h_i)$
    - Mas fazê-lo tem característica de “uso único”: após h_i ser revelado, outro usuário poderia fingir ser o dono de h_0 revelando o mesmo h_i

Hash Chains: usos

- Ligação criptográfica forte entre hashes sucessivos
  - Aproveita não-inversibilidade (resistência à 1a inversão)



Ex.: 1º acesso:

senha usada por usuário: S

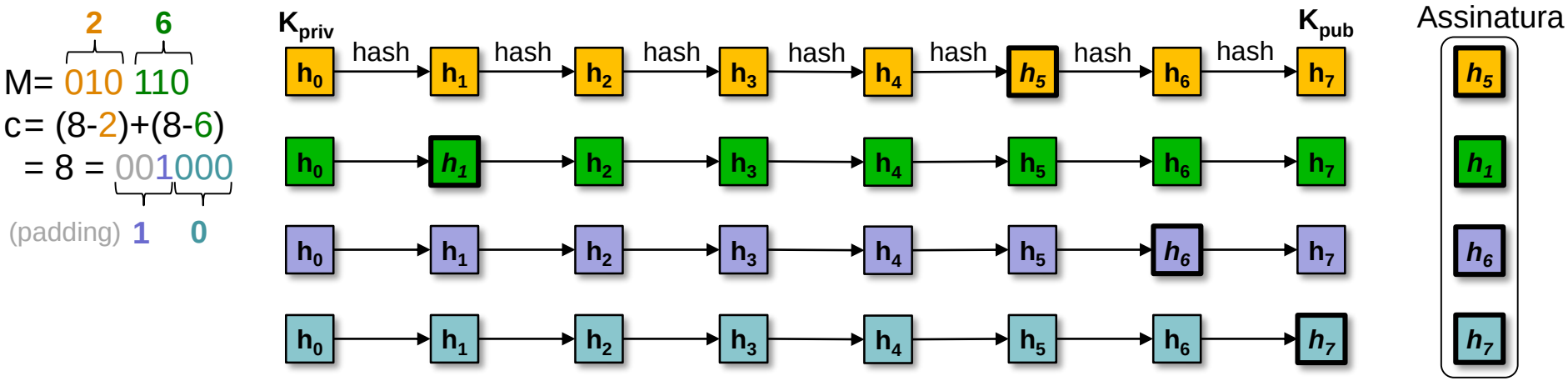
senha registrada no servidor: R

verificação: Hash(S) = R?
Se sucesso: atualização de registro R ← S

- **Autenticação de usuários**:
  - Servidor gera senha aleatória h_0, calcula cadeia de hashes de tamanho n, e fornece lista [h_0… h_{n-1}] a usuário
  - Para cada acesso i, usuário revela senha de uso único h_{n-i-1}
    - Obs.: vulnerável a ataque Man-in-the-Middle se usado em canal desprotegido, pois atacante pode capturar e usar hash antes do usuário
  - Ex.: S/KEY (1995) [H95]
  - Ex. (variante p/ micropagamentos): [RR96][MCY19]

Hash Chains: usos

- Ligação criptográfica forte entre hashes sucessivos
  - Aproveita não-inversibilidade (resistência à 1a inversão)

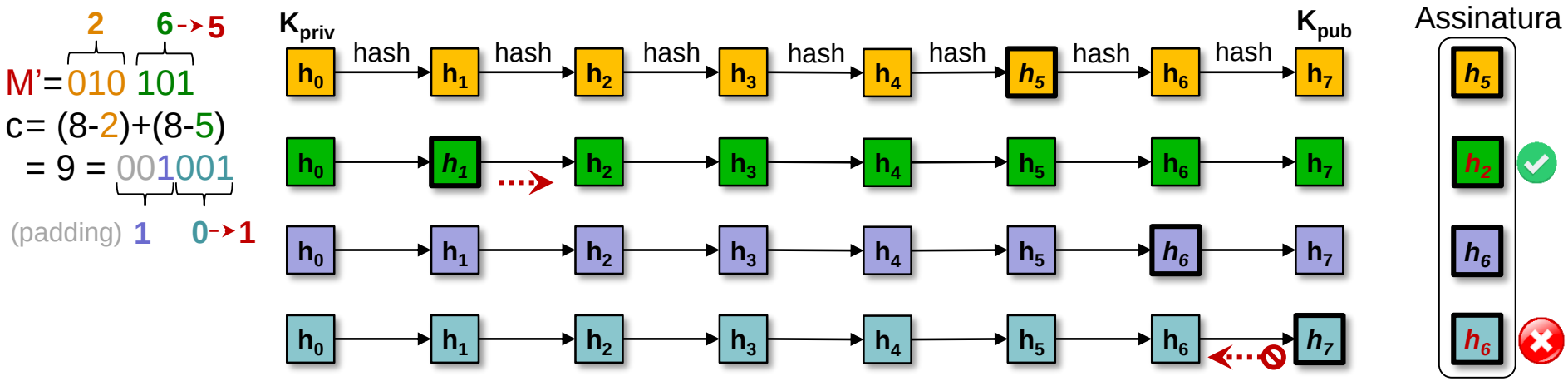


- **Assinaturas digitais** com hashes: WOTS (1979)
  - Criar cadeias de tamanho $n=2^w$; usar conjunto de h_{n-1} como chave pública e de h_0 como chave privada (uso único)
  - Para assinar valor x, revelar h_{n-x-1}
    - Assinatura de $M = (m_1 \mid m_2 \mid \ldots \mid m_t \mid c)$, onde c é um checksum: assinar cada m_i e c, agrupados em conjuntos de w bits, com uma cadeia distinta

Vide: [BBD09]

Hash Chains: usos

- Ligação criptográfica forte entre hashes sucessivos
  - Aproveita não-inversibilidade (resistência à 1a inversão)

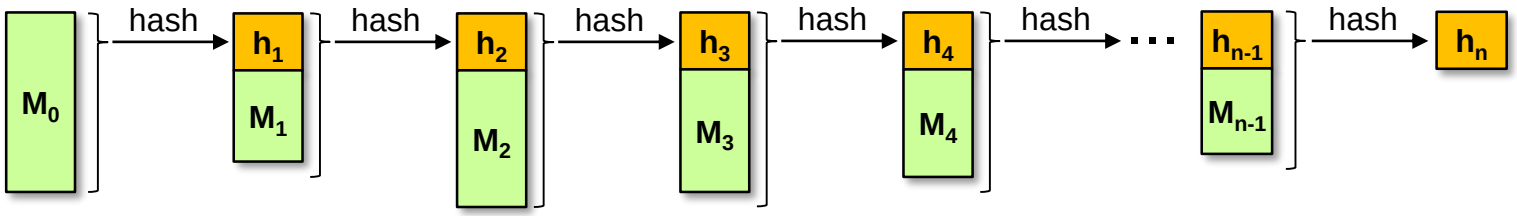


- **Assinaturas digitais** com hashes: WOTS (1979)
  - Criar cadeias de tamanho $n=2^w$; usar conjunto de h_{n-1} como chave pública e de h_0 como chave privada (uso único)
  - Para assinar valor x, revelar h_{n-x-1}
    - Assinatura de $M = (m_1 \mid m_2 \mid \ldots \mid m_t \mid c)$, onde c é um checksum: assinar cada m_i e c, agrupados em conjuntos de w bits, com uma cadeia distinta
    - **Obs.**: embora h_{n-x} revele os hashes $[h_{n-x+1}, h_n]$, checksum não permite que esses valores sejam usados na assinatura de mensagem $M' \neq M$

Vide: [BBD09]

Hashes encadeados

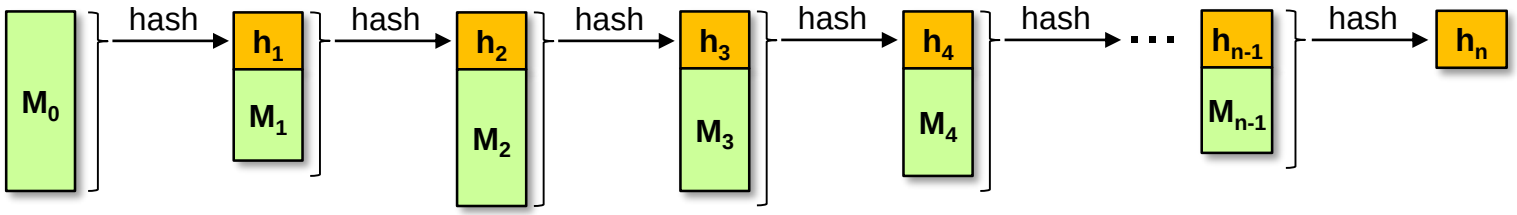
- Permite verificar uma sequência de dados
  - Uma espécie de combinação de mecanismo de compromisso com hash chains



- Construção:
  - Calcular $h_1 = Hash(M_0)$
  - Para todo $1 < i < n$: calcular $h_i = Hash(h_{i-1}, M_{i-1})$
  - Valor final h_n permite verificar a cadeia completa $[M_0 , M_{n-1}]$
    - E revelar h_n cria um compromisso com essa cadeia

Hashes encadeados

- Permite verificar uma sequência de dados
  - Uma espécie de combinação de mecanismo de compromisso com hash chains



- Qual o **custo** para provar que um M_i qualquer está em uma cadeia contendo n hashes?
  - Pior caso: provar que M_0 está na cadeia ➔ **O(n)** hashes; **O(n)** de espaço (requer acesso à cadeia complete)
- **Desafio**: reduzir esse custo assintótico
  - Dica: a cadeia precisa mesmo ser unidimensional…?

Árvore de Merkel

Fonte: https://www.nwzonline.de/hintergrund/auch-zu-politikern-kommt-der-weihnachtsmann_a_6,0,1832470651.html#

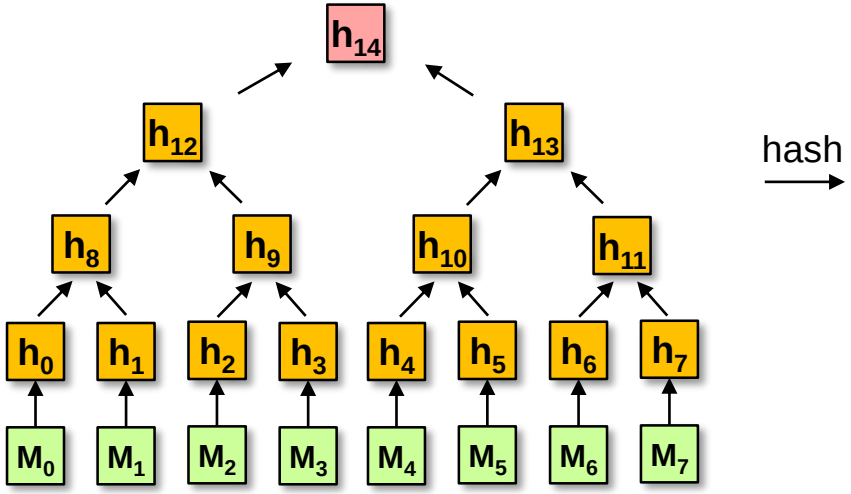
Oops...
Na verdade
eu quis dizer

~~Merkel~~ Árvore de Merkle (1989)

- Permite verificar uma sequência de dados
- Construção: árvore binária
  - Folhas h_i: hash do dado M_i
    - Pode-se usar M_i diretamente, mas M_i grande eleva custo das provas
  - Nós internos: hash dos filhos direito e esquerdo

➔ Raíz permite verificar sequência completa de dados

Fonte: pt.wikipedia.org/wiki/Ralph_Merkle



Vide: [BBD09]

Árvore de Merkle (1989)

- Provar que M_i está em uma árvore de raíz r: fornecer nós irmãos do caminho de M_i até r
  - Ex.: prova p/ M_3 ➔ fornecer $\{h_2, h_8, h_{13}\}$ e r=h_{14};
  - Verificador:

    $v_3 = Hash(M_3)$

    $v_9 = Hash(h_2, v_3)$

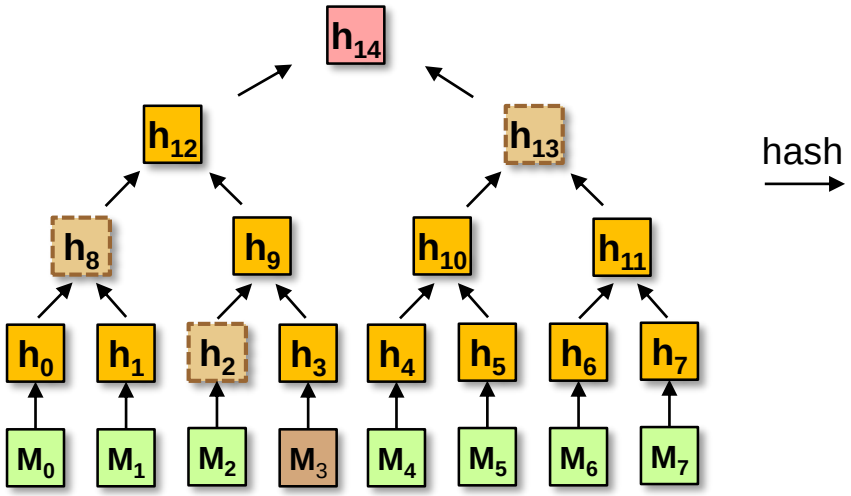
    $v_{12} = Hash(h_8, v_9)$

    $v_{14} = Hash(v_{12}, h_{13})$

    aceitar se $v_{14} = h_{14}$

➔ Custo: $O(\log_2 n)$ hashes

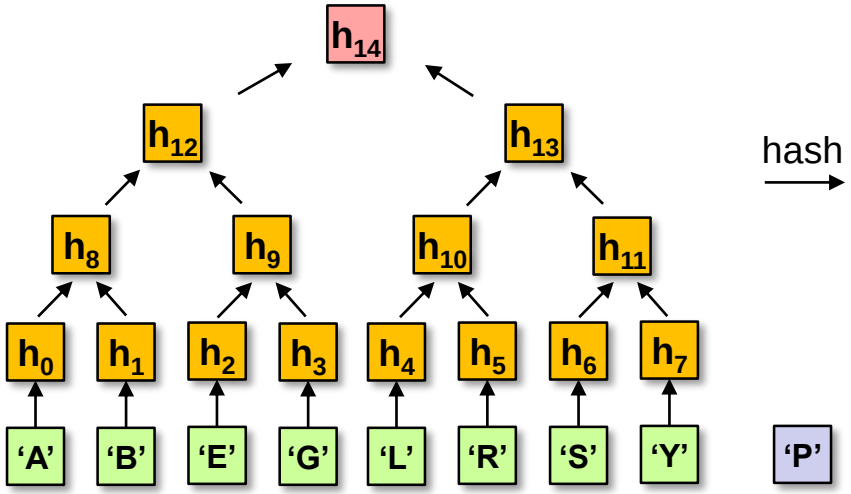
$O(\log_2 n)$ de espaço



Vide: [BBD09]

Árvore de Merkle (1989)

- Mas e se eu quiser provar que M_x **não** está em uma árvore de raíz r?
  - Pode estar em **qualquer lugar** da árvore: requer leitura da árvore completa...
    - Custo: **O(n) hashes** para verificar integridade da árvore; depois, **O(n) comparações** p/ cada prova, i.e., **O(n) de espaço**.
  - Desafio: obter maior eficiência no “depois”

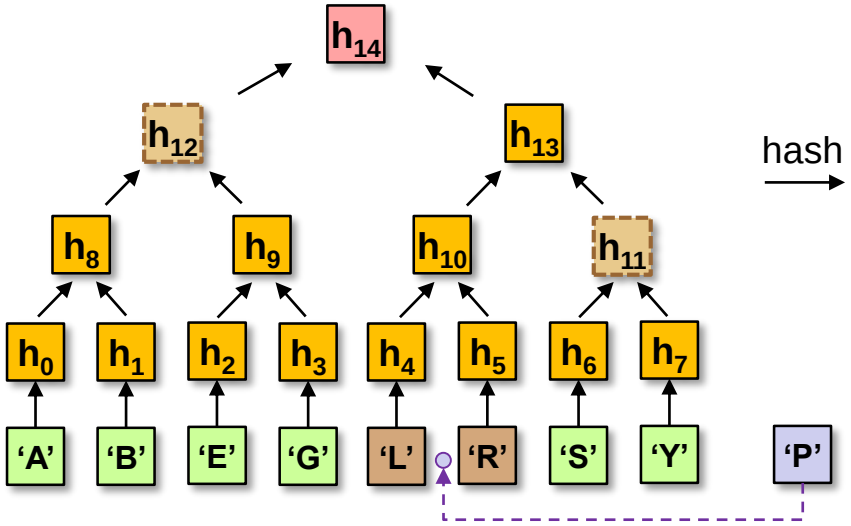


Vide: [LK12]

Árvore de Merkle (1989)

- Mas e se eu quiser provar que M_x **não** está em uma árvore de raíz r?
  - **Árvore ordenada**: basta verificar que M_x não está no local esperado se estivesse na árvore...
    - Custo: **O(n) hashes** para verificar integridade da árvore; depois, prova de pertencimento de **2 nós** da árvore

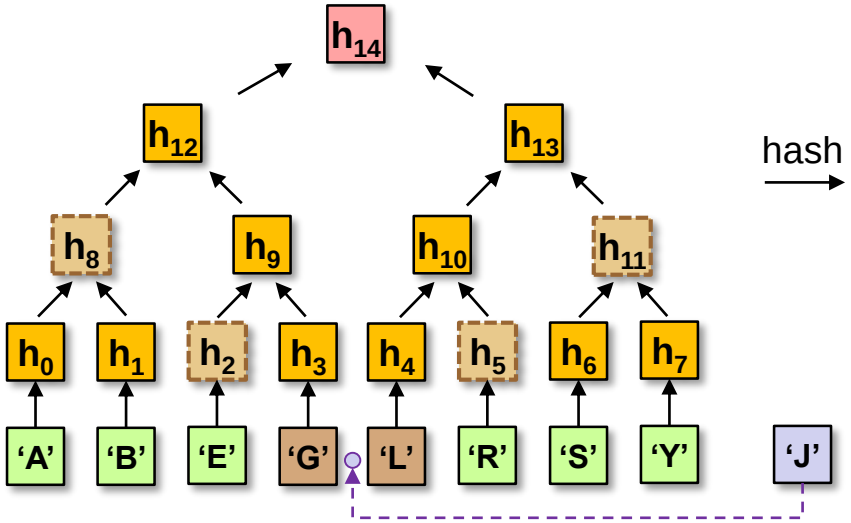
- Ex: 'P' ➔ Mostrar **folhas vizinhas** nas **posições adjacentes** a local onde estaria 'P', i.e., posições 4 e 5
  - Custo: $O(\log_2 n)$ hashes
    $O(\log_2 n)$ de espaço



Árvore de Merkle (1989)

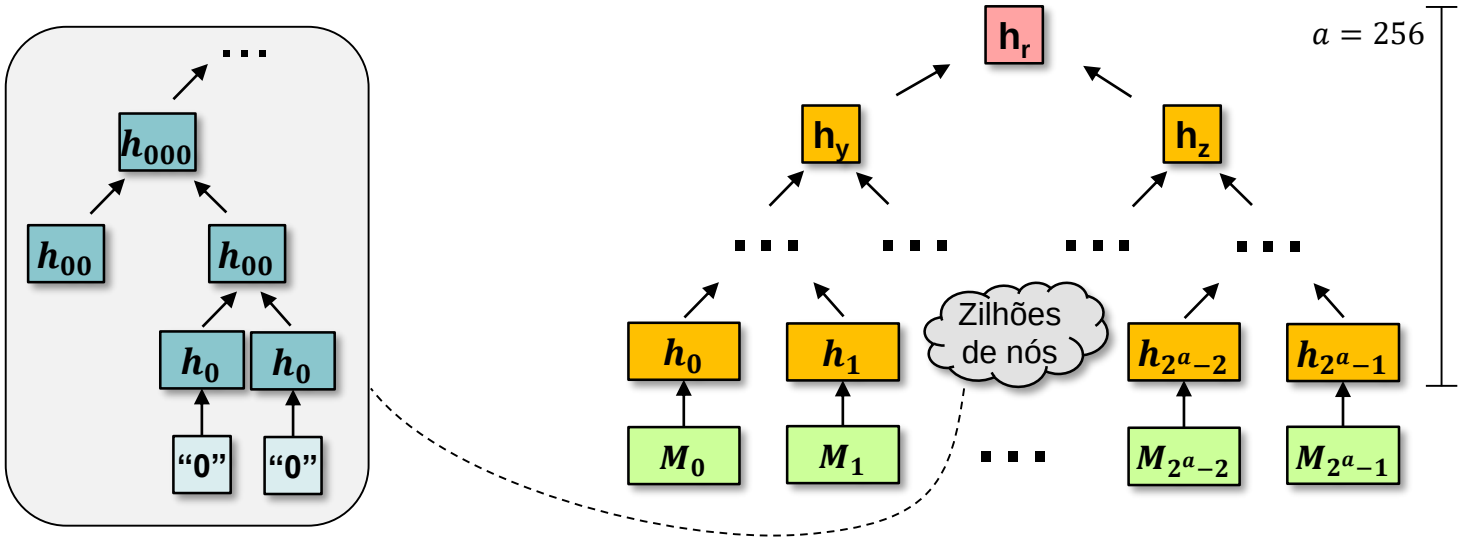
- Mas e se eu quiser provar que M_x **não** está em uma árvore de raíz r?
  - **Árvore ordenada**: basta verificar que M_x não está no local esperado se estivesse na árvore...
    - Custo: **O(n) hashes** para verificar integridade da árvore; depois, prova de pertencimento de **2 nós** da árvore

- Ex: 'J' ➔ Mostrar **folhas vizinhas** nas **posições adjacentes** a local onde estaria 'J', i.e., posições 3 e 4
  - Custo: $O(\log_2 n)$ hashes
    $O(\log_2 n)$ de espaço



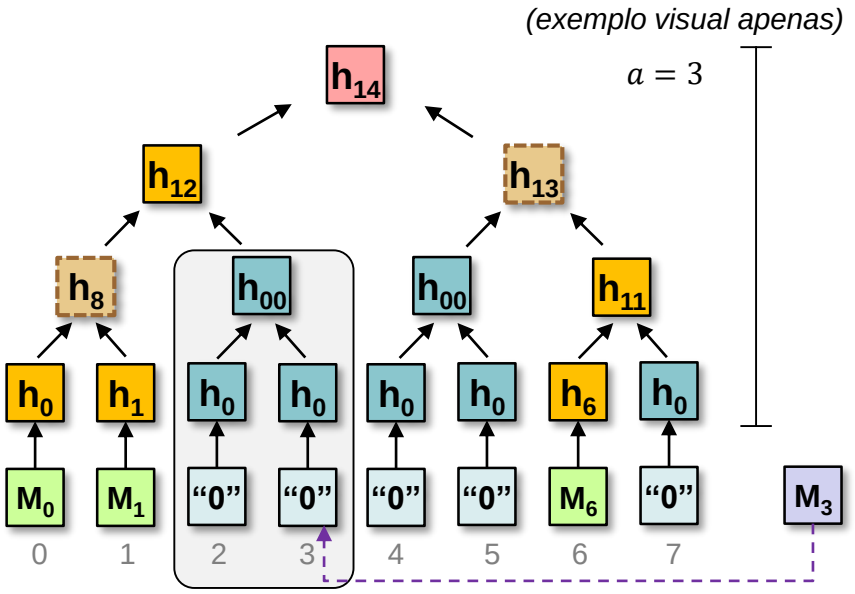
Árvore de Merkle Esparsa (2012)

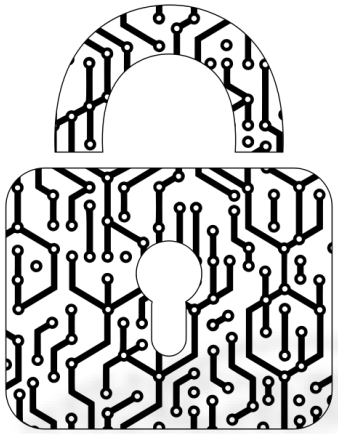
- Mas e se eu quiser provar que M_x **não** está em uma árvore de raíz r?
  - **Árvore esparsa** com altura a: apenas algumas folhas não nulas (valor especial “0”)
    - Computável mesmo para $a = 256$: pré-calcular Hash(0) e $Hash^i(0,0)$ para 0 < i < n



Árvore de Merkle Esparsa (2012)

- Mas e se eu quiser provar que M_x **não** está em uma árvore de raíz r?
  - **Árvore esparsa** com altura a: apenas algumas folhas não nulas (valor especial "0")
    - Computável mesmo para $a = 256$: pré-calcular Hash(0) e $Hash^i(0,0)$ para 0 < i < n
  - Mostrar que posição relativa a M_x é nula
    - Posição: valor de $Hash(M_x)$
    - Custo: até a hashes
      até a de espaço





Blockchain, Criptomoedas & Tecnologias Descentralizadas

Construções criptográficas avançadas com hashes

Prof. Dr. Marcos A. Simplicio Jr. – mjunior@larc.usp.br
Escola Politécnica, Universidade de São Paulo

Apoio financeiro: d24n.org <https://d24n.org>
ripple impact | university blockchain research initiative

USP

Referências

- [BBD09] D. Bernstein, J. Buchmann, E. Dahmen (2009). Post-Quantum Cryptography. Springer, Springer, Berlin, Heidelberg. ISBN 978-3-540-88701-0. DOI: https://doi.org/10.1007/978-3-540-88702-7
- [H95] N. Haller (1995). "RFC 1760: The S/KEY One-Time Password System". Internet Engineering Task Force - Network Working Group.
- [LK12] B. Laurie, E. Kasper. Revocation Transparency. Google Research, Tech. Report, 2012. Available: https://sump2.links.org/files/RevocationTransparency.pdf
- [MCY19] M. Elsheikh, J. Clark, A. Youssef (2019). Short Paper: Deploying PayWord on Ethereum. Int. Conf. on Financial Cryptography and Data Security, 82-90. URL: https://users.encs.concordia.ca/~clark/papers/2019_wtsc_ethword.pdf
- [RR96] R. Rivest, A. Shamir (1996) PayWord and MicroMint: two simple micropayment schemes. In: Security Protocols. Lecture Notes in Computer Science, vol 1189. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-62494-5_6
- [SSL+14] Simplicio Jr, M., Santos, M., Leal, R., Gomes, M., Goya, W. (2014). SecureTCG: a lightweight cheating-detection protocol for P2P multiplayer online trading card games. Security and Communication Networks, 7(12), 2412-2431. https://doi.org/10.1002/sec.952