Segurança da Informação

Funções *Hash*

Prof. Evandro Luiz C. Macedo
IC/UFRJ
evandro@ravel.ufrj.br

Tópicos da Aula

- Funções hash
- Aplicações de funções hash
- SHA-512
- MD5

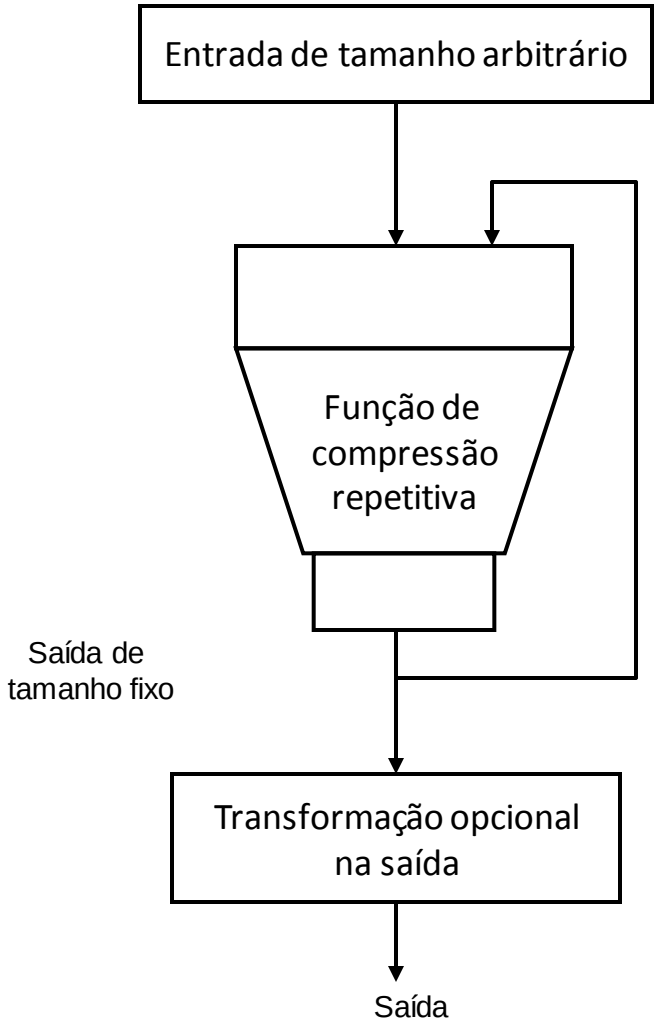
Funções *Hash*

- Uma função *Hash* é uma função que mapeia uma entrada de comprimento arbitrário (*bits*, *bytes* ou caracteres) em um resultado de tamanho fixo

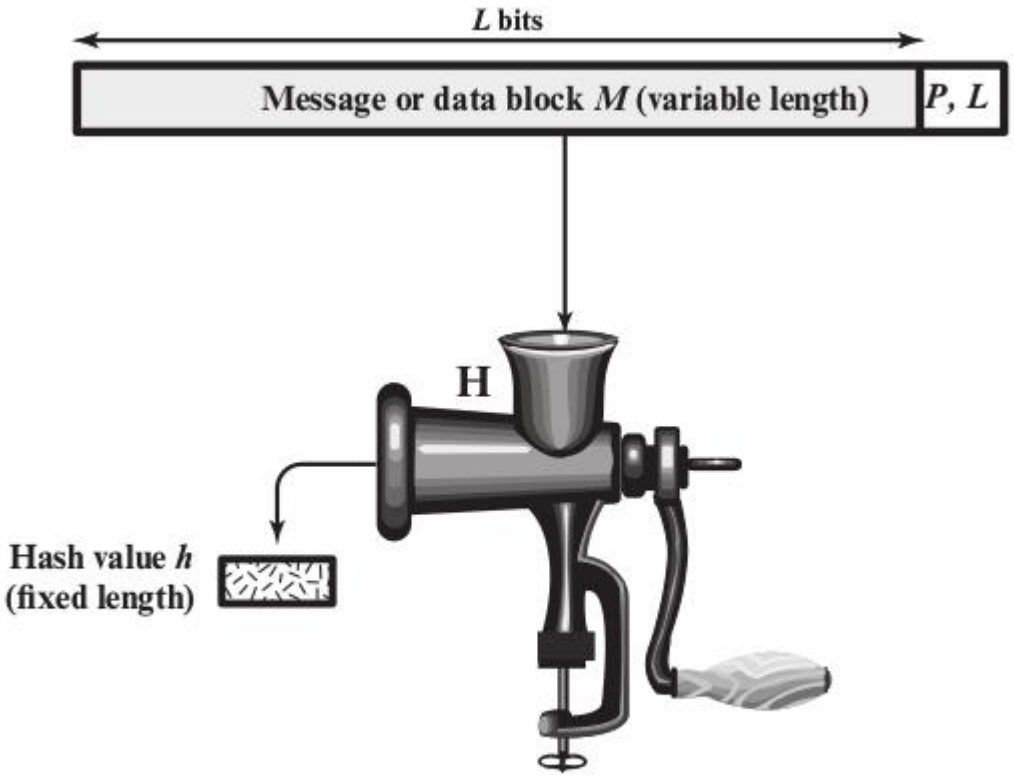
- O resultado das funções *Hash* mais usadas estão entre 128 e 512 bits

- Funções *Hash* também são chamadas ***message digest*** e o resultado é chamado de ***digest*** ou ***fingerprint***

Funções *Hash*



Funções *Hash*



P, L = padding plus length field

Funções *Hash*

- Notação:

$$\mathbf{h(M) = x}$$

- M é a mensagem, de comprimento arbitrário (porém finito) e **x** é o resultado, de comprimento fixo
- Nosso interesse é nas chamadas funções *Hash* criptográficas
- A diferença entre as funções *Hash* criptográficas e as funções *Hash* “comuns” está nos requisitos adicionais

Funções *Hash*

- O **primeiro requisito** é que ela seja uma **função não invertível**, ou "unidirecional" (*one-way*)
- Isto quer dizer que *é fácil* (e eficiente), dada a mensagem m, *calcular h(m)*
- Por outro lado, dado x, tal que x = h(m), *não deve ser possível encontrar um **m** tal que*

x = h(m)

Funções *Hash*

- O **segundo requisito** é que uma função *Hash* criptográfica deve ser ***resistente a colisões*** (*collision-free)*
- Embora exista uma infinidade de colisões possíveis, já que existe uma infinidade de mensagens para um número finito de resultados, elas devem ser praticamente impossíveis de serem encontradas
- Quer dizer, dado um vetor (de *bits*) m_1, é computacionalmente intratável (tempo proporcional a $\exp(m_1)$) achar outro vetor m_2, tal que:

$$\mathbf{h(m_1) = h(m_2)}$$

Funções *Hash*

- O **terceiro requisito** que uma função *Hash* criptográfica deve ter é a **resistência forte a colisões**
- Neste caso, é computacionalmente intratável, com tempo proporcional a $\exp(|m_1| + |m_2|)$, achar dois vetores m_1 e m_2, tal que:

$$\mathbf{h(m_1) = h(m_2)}$$

Funções *Hash*

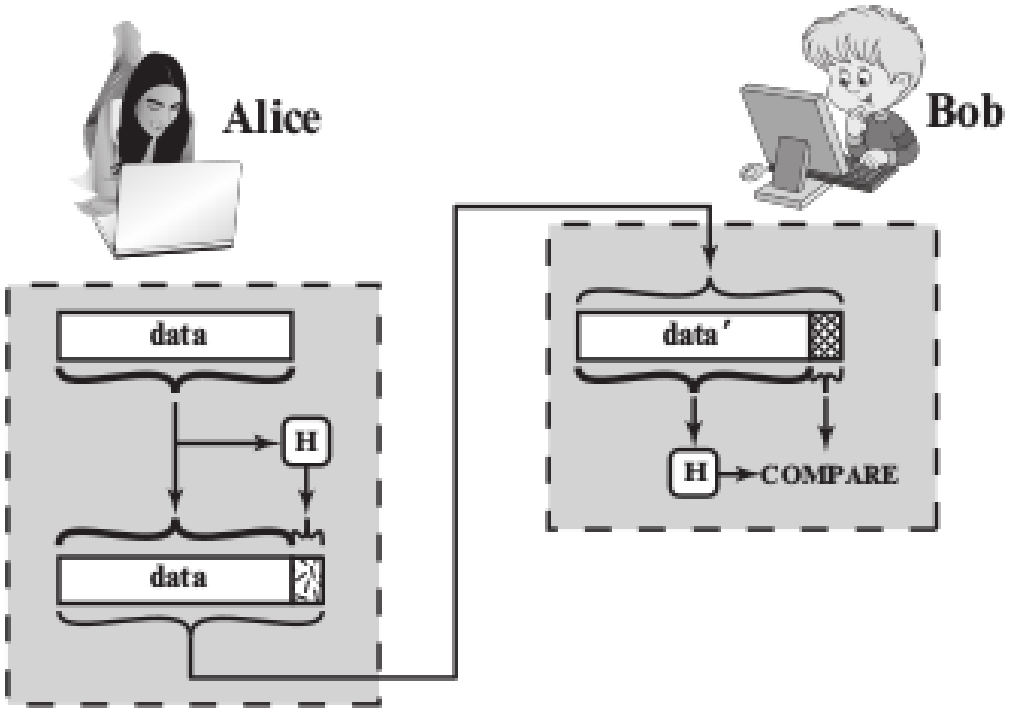
- O **quarto requisito** é que uma função *Hash* criptográfica deve ser uma **função aleatória**
- Isto significa que o resultado não deve dar nenhuma informação sobre as mensagens às quais a função foi aplicada

Funções *Hash*

- Com o aumento da quantidade dos dados, tornou-se desejável reduzir o tamanho de uma mensagem de modo a manter um mapeamento único entre a mensagem e o resultado do mapeamento
- Esse mapeamento pode ser usado para garantir a integridade do dado ou a sua origem, dando lugar às duas vertentes principais dos algoritmos de *hash*:
  - **MAC** (*Message Authentication Code*)
  - **MDC** (*Modification Detection Code*)

Aplicações de Funções *Hash*

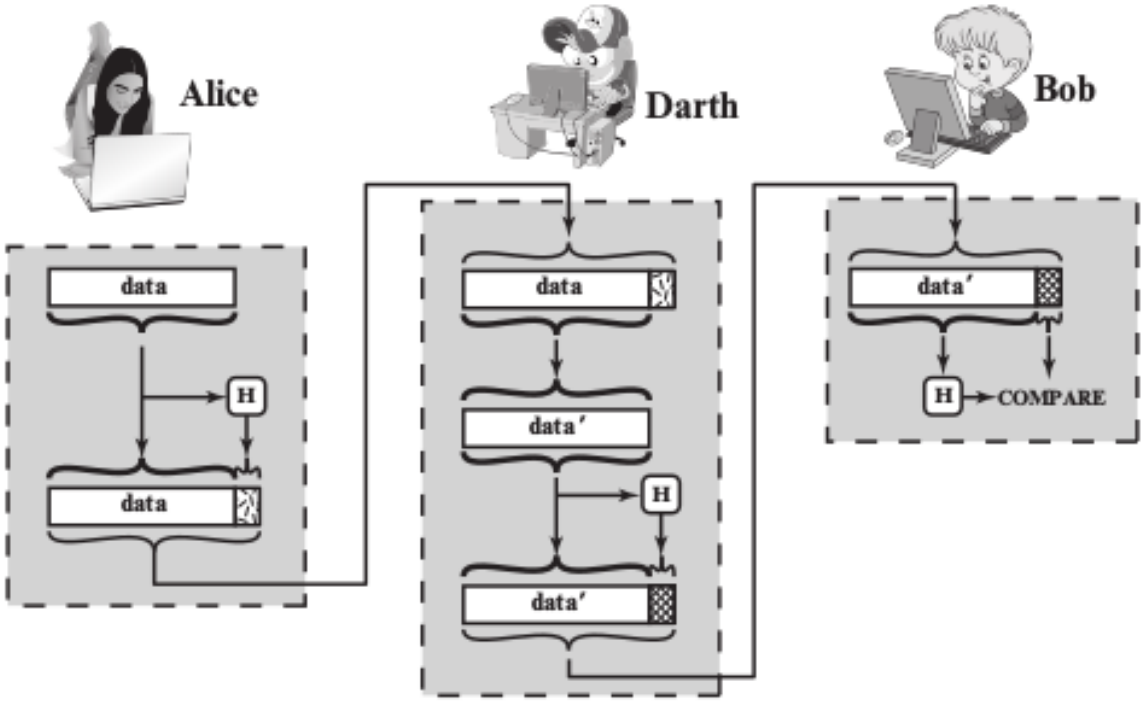
- Integridade de Mensagens



(a) Use of hash function to check data integrity

Aplicações de Funções *Hash*

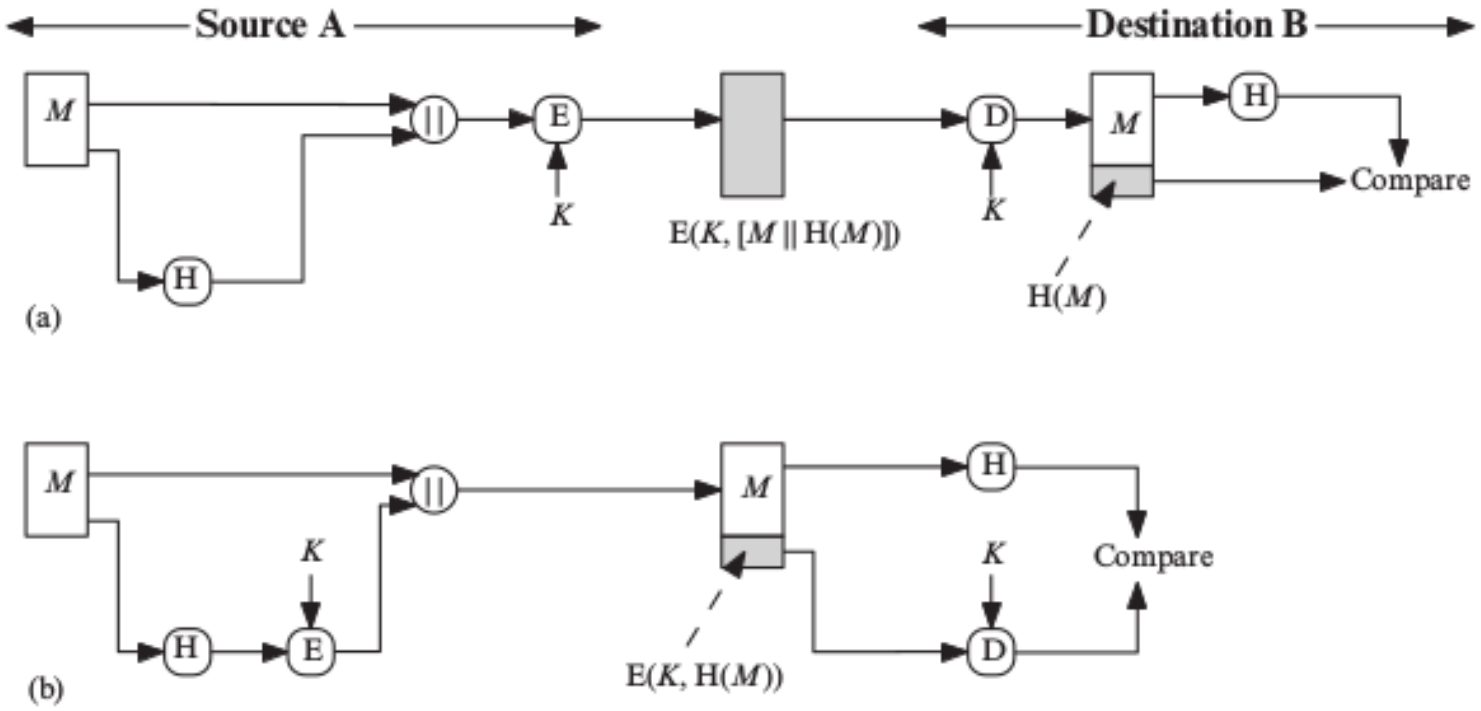
- Integridade de Mensagens



(b) Man-in-the-middle attack

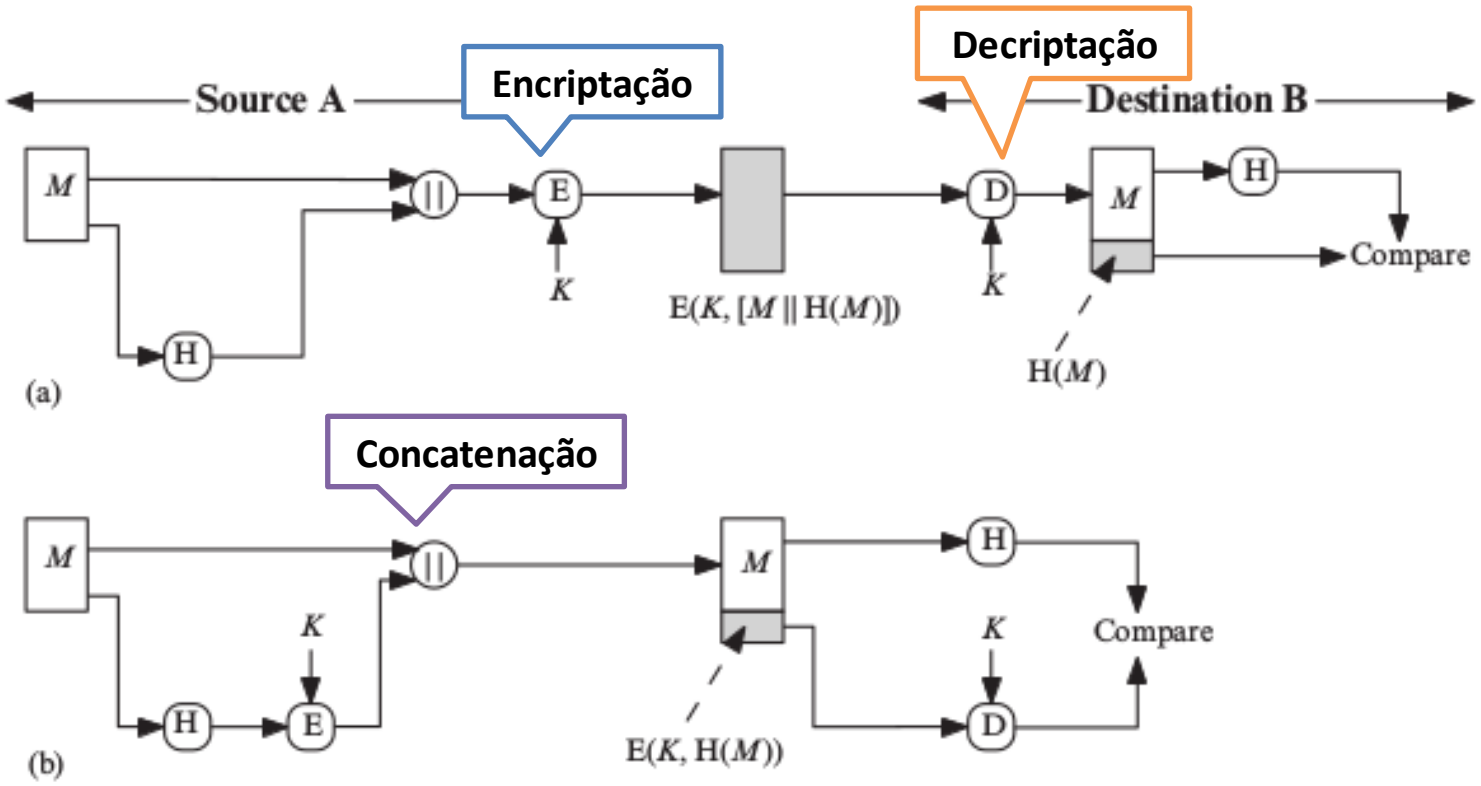
Aplicações de Funções *Hash*

- Autenticação de Mensagens



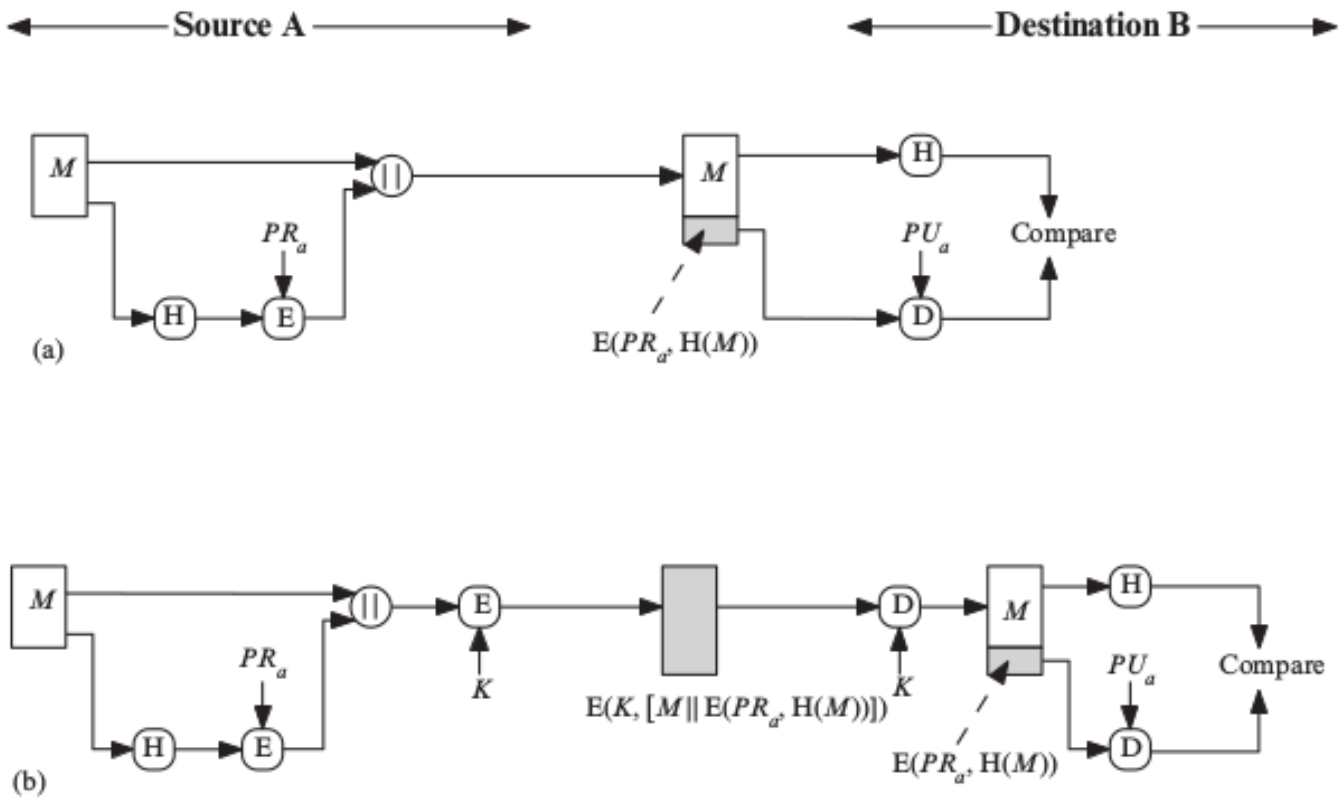
Aplicações de Funções *Hash*

- Autenticação de Mensagens



Aplicações de Funções *Hash*

- Assinatura Digital



MDC

- As MDCs (*Modification Detection Code*) também são conhecidas como código manipuladores de erro, ou código de integridade de mensagens (MICs)

- Possuem duas divisões básicas:

  OWHF = *One Way Hash Function*

  CRHF = *Collision Resistant Hash Functions*

MAC

- O propósito das MAC (*Message Authentication Code*) é assegurar a origem da mensagem

- As MACs possuem dois parâmetros importantes e funcionalmente diferentes:

  1 - a mensagem de entrada

  2 - **a chave secreta**

MAC

- Um MAC é uma função:

$$a = MAC(K, m)$$

- Onde K é uma chave simétrica secreta e m é a mensagem
- O receptor pode então verificar se o MAC recebido corresponde realmente à mensagem e se foi enviado por alguém **que compartilha da mesma chave secreta!**

Ataque às Funções *Hash*

- Definimos então um **ataque a uma função *Hash*** como um método não trivial de distinguir uma função *Hash* de uma função *Hash* ideal
- O ataque típico a uma função *Hash* é o **ataque de aniversário**
- Para uma função com resultado de n *bits*, o atacante poderia gerar $2^{n/2}$ resultados e testar colisões

Funções *Hash*

- Função Ou-exclusivo (XOR)
  - Uma das funções *Hash* mais simples

Mensagem: 0 1 1 1 1 0 1 0
Chave: 0 1 0 1 0 1 0 1
XOR: 0 0 1 0 1 1 1 1

Mensagem Cifrada: 0 0 1 0 1 1 1 1
Chave: 0 1 0 1 0 1 0 1
XOR: 0 1 1 1 1 0 1 0

XOR

00 = 0
01 = 1
10 = 1
11 = 0

Funções *Hash*

- Função Ou-exclusivo (XOR)
  - Uma das funções *Hash* mais simples

Mensagem: 0 1 1 1 1 0 1 0
Chave: 0 1 0 1 0 1 0 1
XOR: 0 0 1 0 1 1 1 1

Mensagem Cifrada: 0 0 1 0 1 1 1 1
Chave: 0 1 0 1 0 1 0 1
XOR: 0 1 1 1 1 0 1 0

XOR

00 = 0
01 = 1
10 = 1
11 = 0

Problema
Tamanho da chave!

Funções *Hash*

- O espaço de *hashes* possíveis é:
  - 2^n
- Se temos mais informação, por exemplo, cada primeiro bit de um octeto é sempre zero
  - Em um *hash* de 128 *bits*, qual a efetividade desse *hash*?
  - Em outras palavras, qual o espaço de *hashes* possíveis?

Funções *Hash*

- O espaço de *hashes* possíveis é:
  - 2^n
- Se temos mais informação, por exemplo, cada primeiro bit de um octeto é sempre zero
  - Em um *hash* de 128 *bits*, qual a efetividade desse *hash*?
  - Em outras palavras, qual o espaço de *hashes* possíveis?
    - 2^{-112}

Funções *Hash*

- As funções *Hash* mais utilizadas são:
  - MD5
  - * SHA-1
  - SHA-256
  - SHA-3

- A função MD5 é a mais antiga e já foram identificadas fraquezas nela
- Não é uma função aconselhada, embora ainda seja largamente utilizada
- É uma função de 128 bits
- A SHA-1 também já foi quebrada (em 2015), usa 160 *bits*

Funções *Hash*

- A função **SHA-1** (*Secure Hash Algorithm*) foi criada pela NSA e padronizada pelo NIST
- Ela veio substituir a SHA (ou SHA-0) e reparar uma fraqueza que foi identificada
- É uma função de **160 bits**, supostamente mais segura que a MD5
- Entretanto também foram encontradas fraquezas nela, que levaram a criação do novo padrão

Funções *Hash*

- A **SHA-2** é um conjunto de 3 funções:
  - SHA-256, uma função de 256 bits
  - SHA-384, uma função de 384 bits
  - SHA-512, uma função de 512 bits

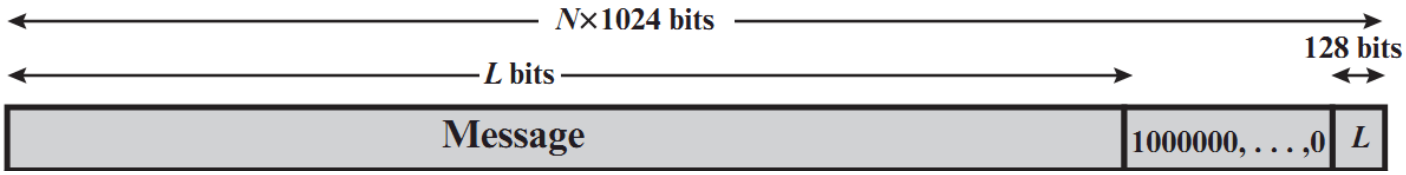
- A família **SHA-3** também já está desenvolvida, considerando uma nova abordagem matemática, diferente de seus antecessores

SHA-512

- Recebe uma entrada de no máximo 2^{128} *bits* e produz uma saída de 512 *bits*
- A entrada é processada em blocos de 1024 *bits*
- O algoritmo conta com 5 etapas:
  1. *Append Padding*
  2. *Append length*
  3. *Initialize hash buffer*
  4. *Process message blocks*
  5. *Output*

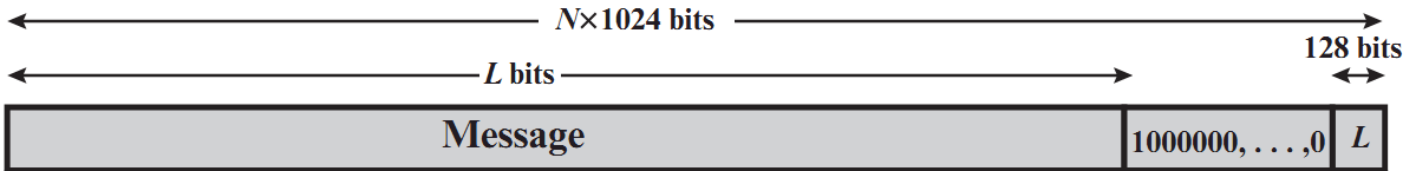
SHA-512

1. *Append Padding*
   - A mensagem é completada com 100...00 de acordo com o necessário para completar o tamanho de um bloco de 1024 *bits*



SHA-512

2. *Append length*
   - Um bloco de 128 *bits* é adicionado ao final da mensagem
   - O bloco contém o tamanho original da mensagem



SHA-512

3. *Initialize hash buffer*
   - Um *buffer* de 512 bits é usado para armazenar os resultados intermediários e final
   - O *buffer* é representado por 8 x *64-bit* blocos

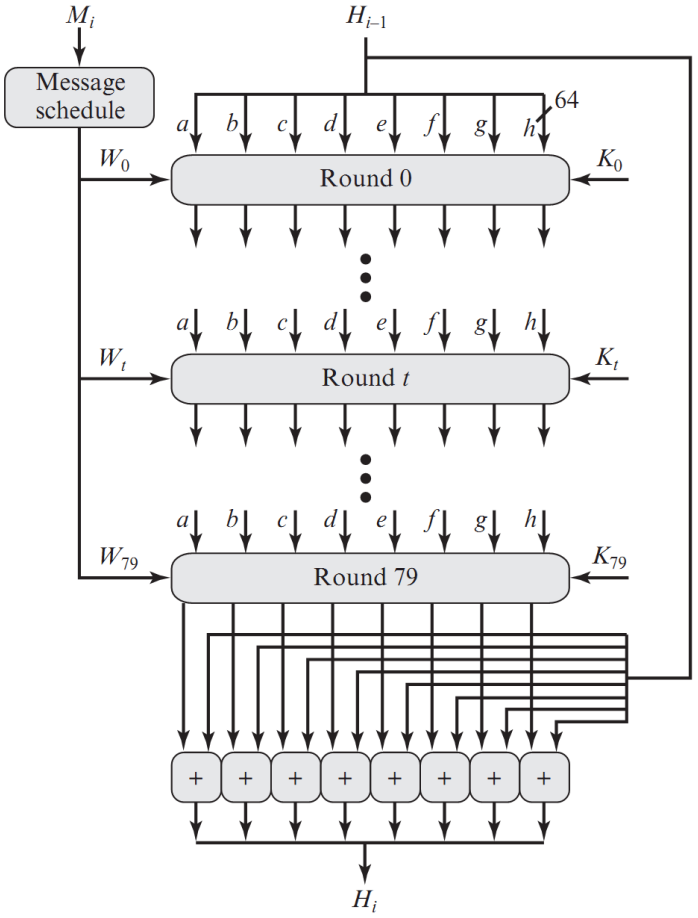
```
a = 6A09E667F3BCC908    e = 510E527FADE682D1
b = BB67AE8584CAA73B    f = 9B05688C2B3E6C1F
c = 3C6EF372FE94F82B    g = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1    h = 5BE0CD19137E2179
```

SHA-512

4. *Process message blocks*
   - Consiste em um módulo central (F) para o algoritmo que é executado **80 vezes**
   - Cada *round* atualiza o conteúdo do *buffer* de acordo com a entrada fornecida

SHA-512

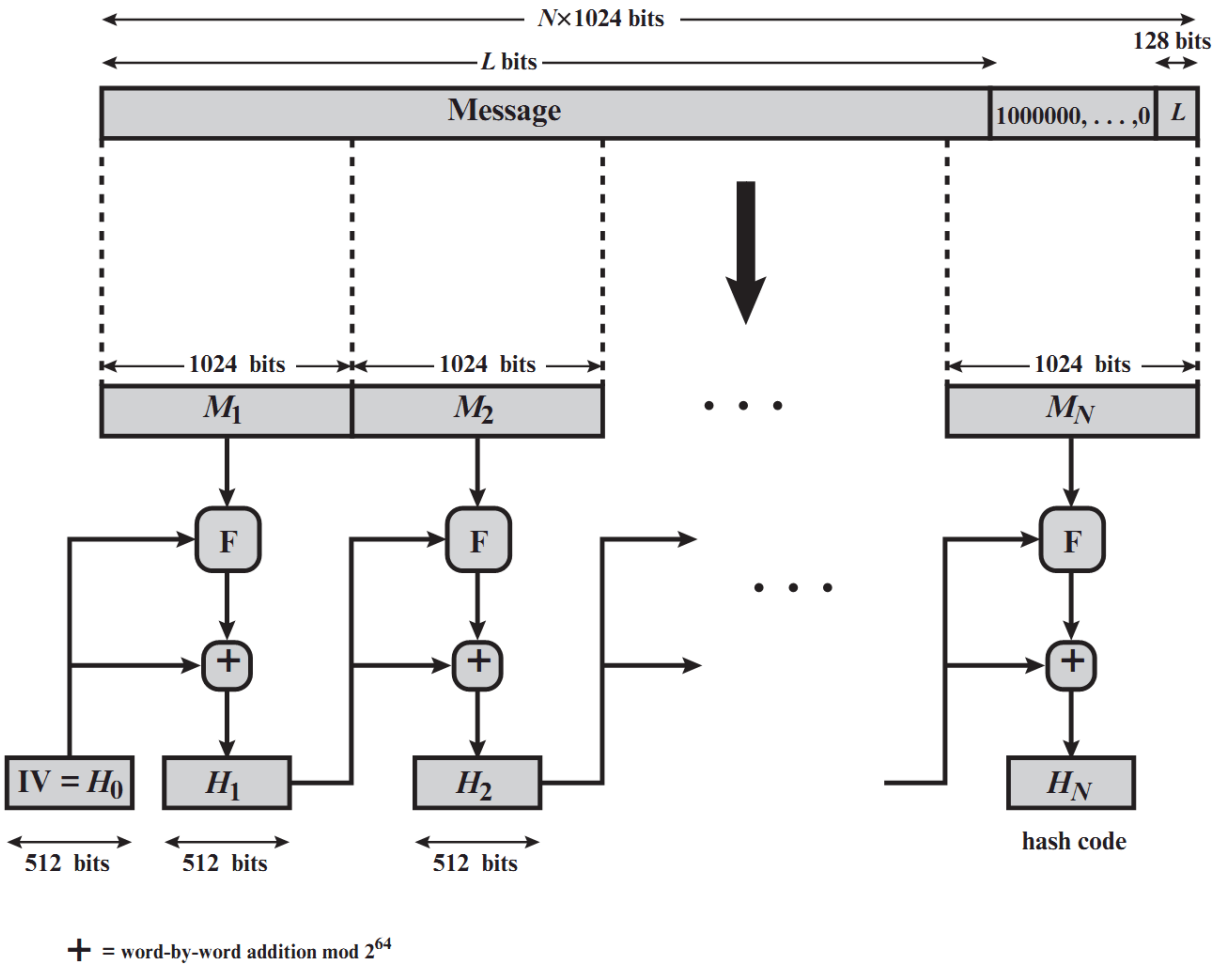
4. *Process message blocks*



SHA-512

5. *Output*
   - Cada passagem do algoritmo gera uma saída intermediária
   - O último passo gera o *hash* final de 512 *bits*

SHA-512


N×1024 bits
128 bits
L bits
Message
1000000, . . . ,0
L
1024 bits
1024 bits
1024 bits
M1
M2
MN
F
F
F
IV = H0
H1
H2
HN
512 bits
512 bits
512 bits
hash code
+ = word-by-word addition mod 2^64

SHA-512

- O SHA-512 tem a propriedade de cada *bit* do *hash* ser função de cada *bit* da entrada
- As repetições da função F produzem resultados bem embaralhados, o que reduz a probabilidade de colisão
- Se nenhuma vulnerabilidade estiver oculta no SHA-512, então a resistência à colisão é da ordem de 2^{256} operações necessárias para quebrá-lo

MD5

- ***Message Digest 5***
  - Produz um *hash* de 128 *bits*
  - Divide uma entrada de 512 *bits* em 16 x *32-bit* blocos
  - É feito um *padding* para alcançar um tamanho múltiplo de 512 *bits*
  - Também é adicionado um bloco contendo o tamanho da entrada original

MD5

- ***Message Digest 5***
  - Quatro variáveis de 32 *bits* são inicializadas (como *buffers*)

A = 0x01234567

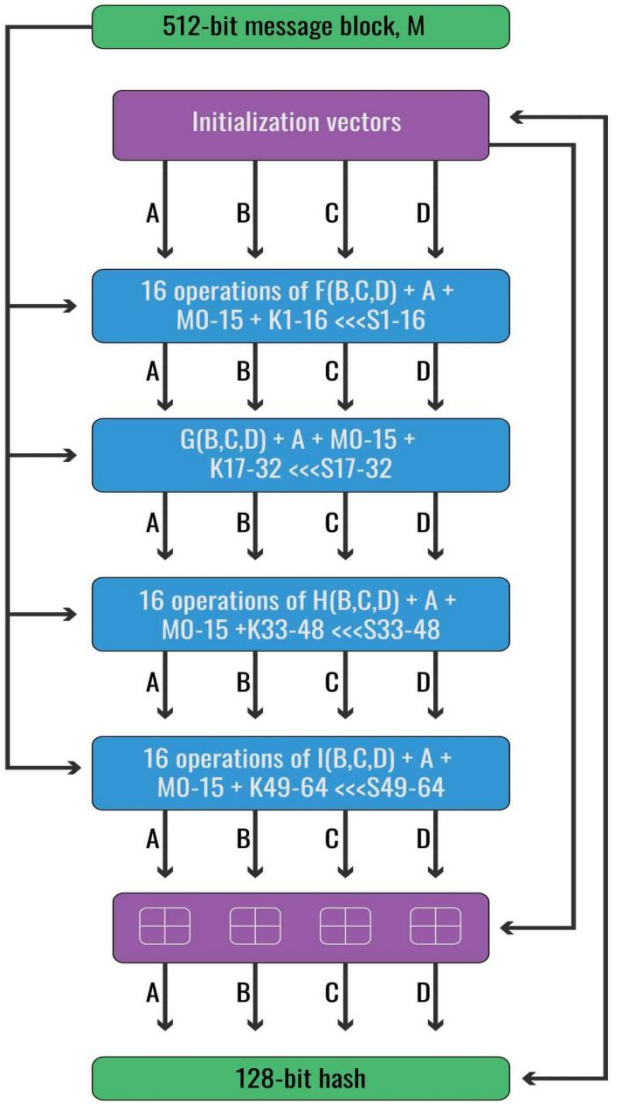
B = 0x89abcdef

C = 0xfedcba98

D = 0x76543210

MD5

- ***Message Digest 5***
  - O MD5 considera 4 *rounds* (MD4 considera 3 *rounds*)
  - Para cada *round*, uma operação lógica diferente é realizada 16 vezes



Referências

- Livro do Stalings
- Livro do Schneier