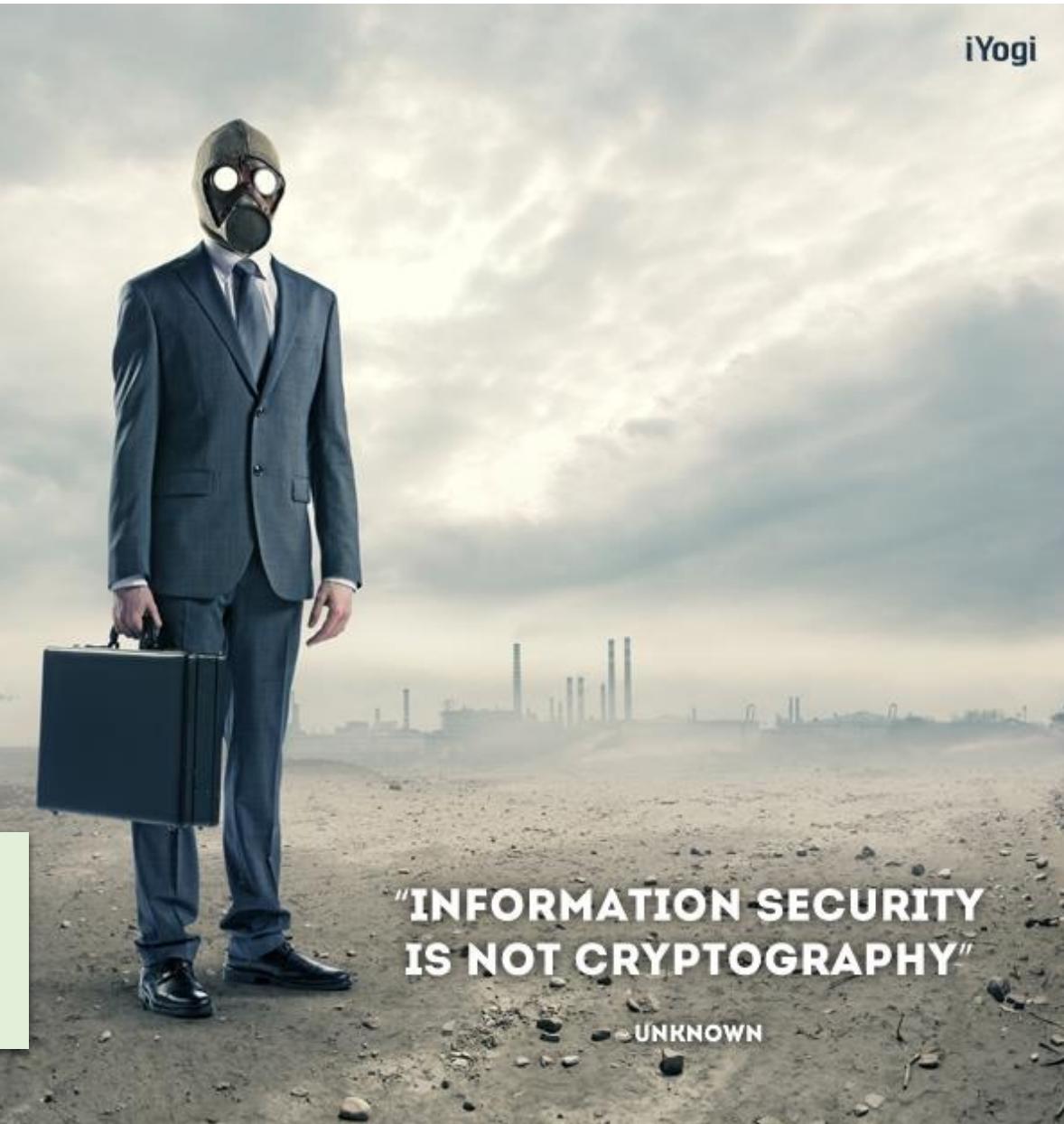


IMD0703 – Segurança de Redes

Funções Hash.

Nota: Alguns dos slides apresentados aqui são compilações de slides do livro *Criptografia e segurança de redes: princípios e práticas* - William Stallings, gentilmente disponibilizados pela editora PEARSON para fins educacionais.





Objetivo desta aula (Cont.)

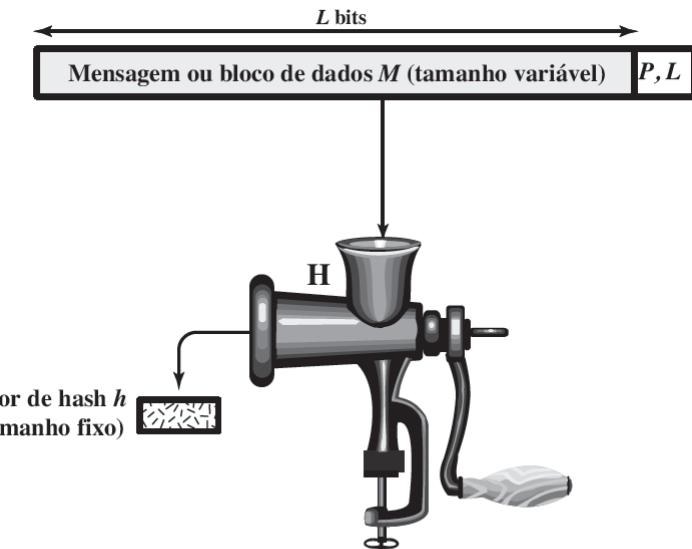
- Geral
 - Apresentar uma visão geral dos princípios básicos e aplicações de funções de Hash
- Específicos
 - Apresentar uma visão geral e exemplos de funções Hash

Funções Hash

- Motivação
 - Necessidade de privacidade, autenticidade e integridade de mensagens
 - Criptografia de chave pública não oferece agilidade em Assinaturas Digitais
 - Criptografia é insuficiente para garantir integridade de dados
 - Surgiu a necessidade de se criar mecanismos conhecidos como Digital Timestamping
 - Solução apresentada: Função Hash

Funções Hash

- Uma função de **hash** aceita uma mensagem de tamanho variável M como entrada e produz um valor de hash de tamanho fixo $h = H(M)$
- Função de apenas um sentido (*One-way*)
 - $d=h(m)$, mas $h'(d) \neq m$
 - Computacionalmente impraticável encontrar a mensagem a partir de seu resumo (*digest*)
- Quando uma função hash é usada para fornecer autenticação de mensagem, o valor dela é chamado de **resumo de mensagem**
 - Normalmente, a autenticação de mensagem é alcançada usando um **código de autenticação de mensagem (MAC)**, do inglês *Message Authentication Code*) ou função de hash chaveada
 - MACs são usados entre duas partes que compartilham uma chave secreta para autenticar as mensagens trocadas

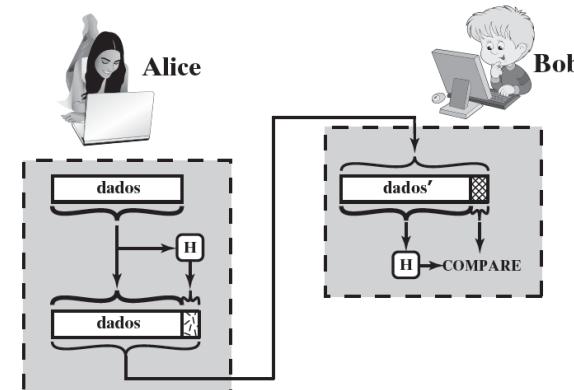


P, L = preenchimento mais campo de tamanho

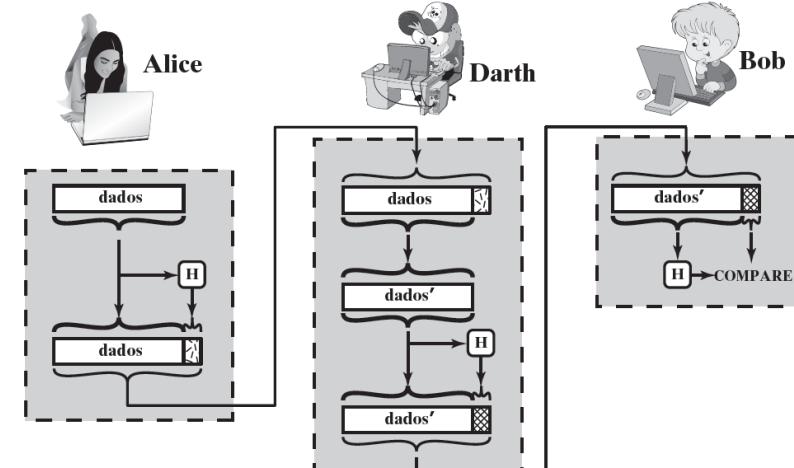
Ataque contra função de hash

- Ataque *man-in-the-middle*
 - Valor de hash gerado por Alice deve ser protegido!!!

Figura 11.2 Ataque contra função de hash.



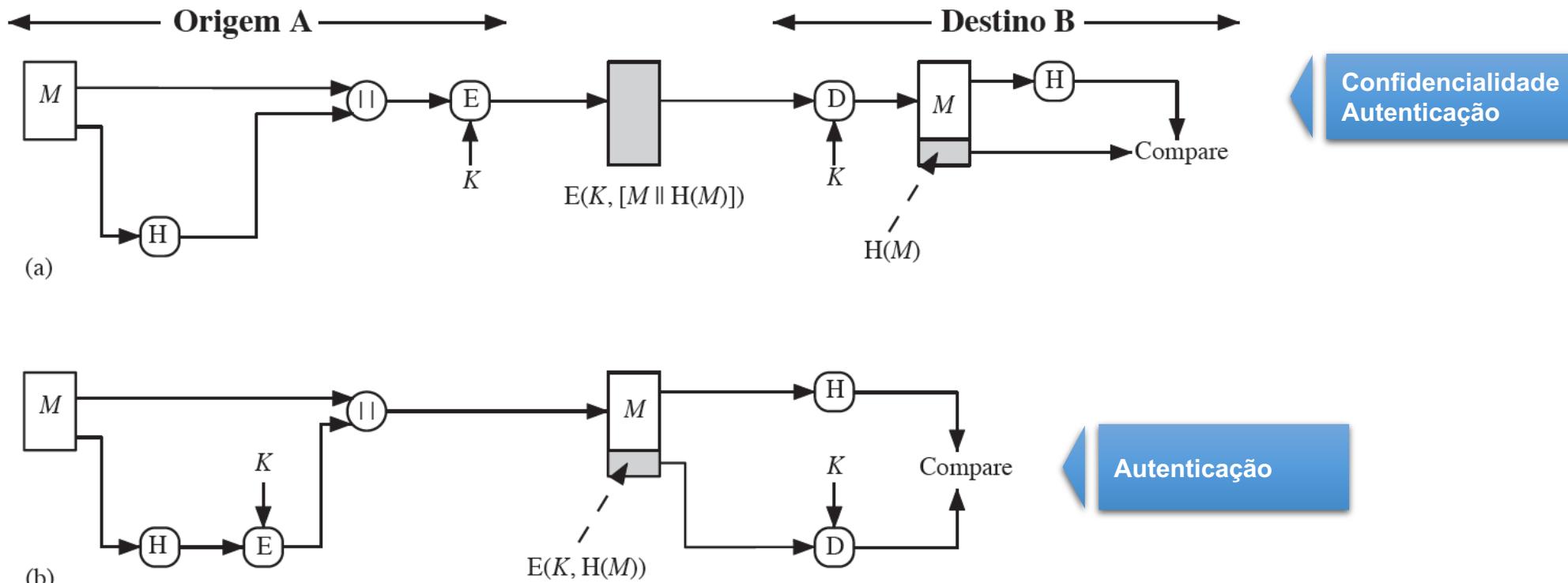
(a) Uso da função de hash para verificar integridade de dados



(b) Ataque *man-in-the-middle*

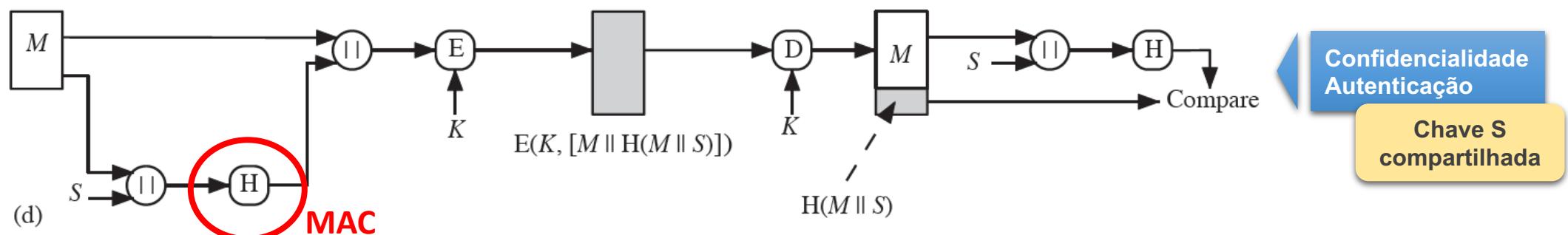
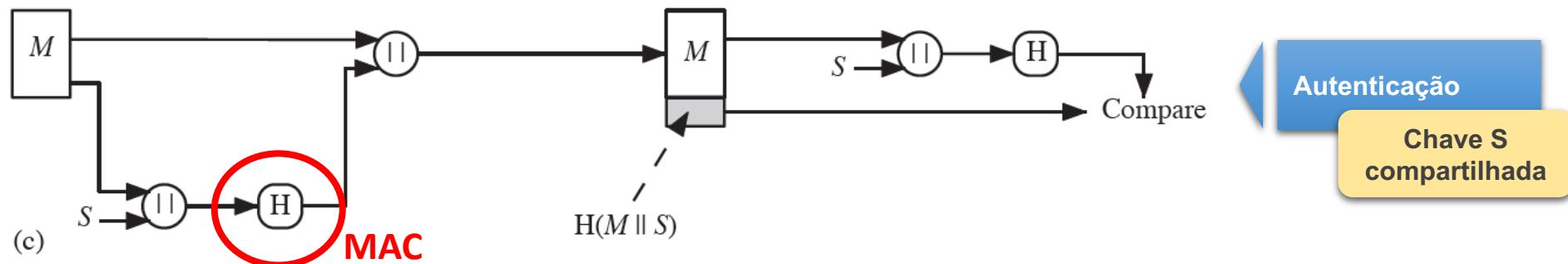
Aplicações de funções de hash criptográficas

- Exemplos simplificados do uso de uma função de hash para **autenticação de mensagem**:



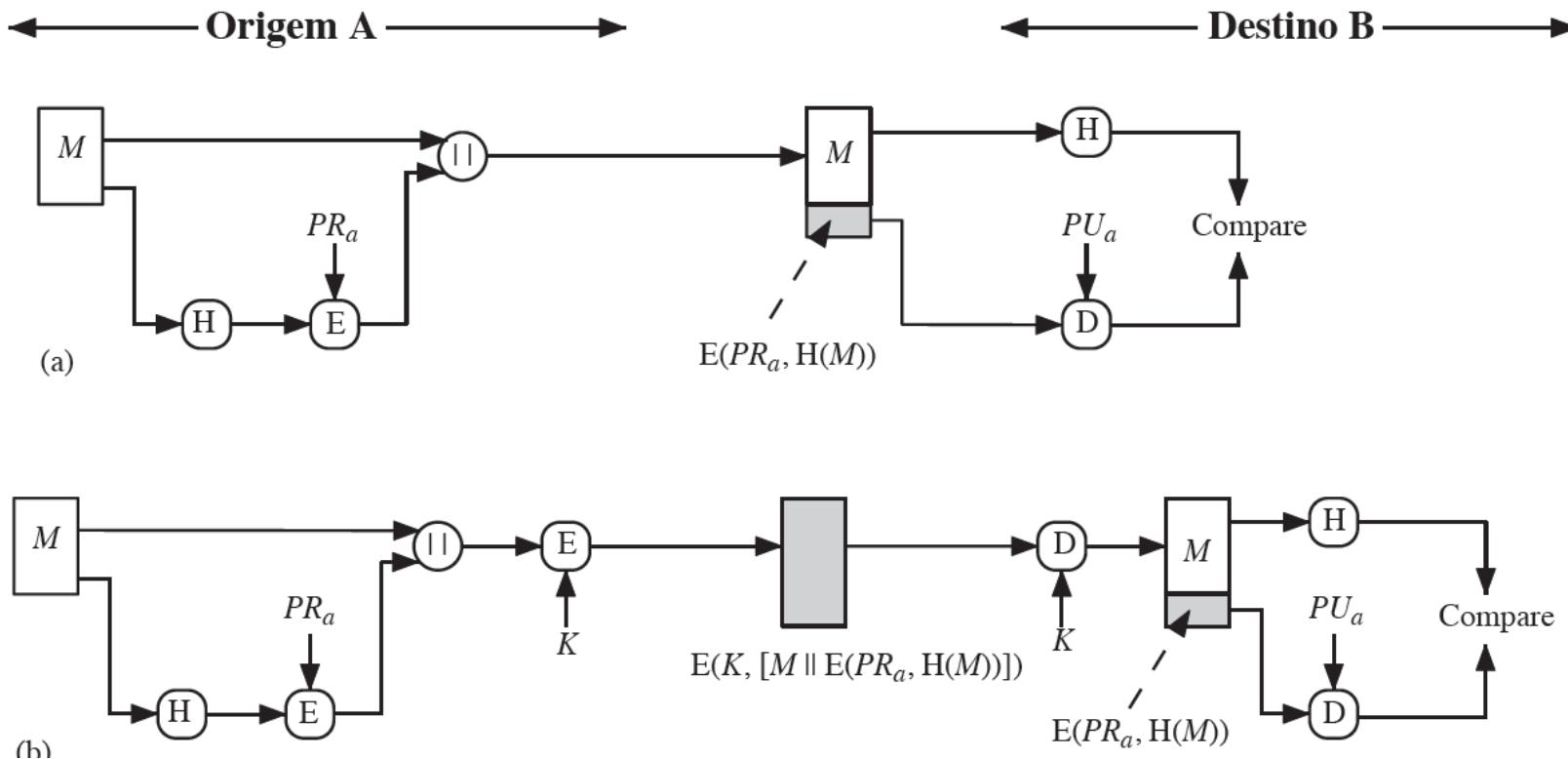
Aplicações de funções de hash criptográficas

- Exemplos simplificados do uso de uma função de hash para **autenticação de mensagem**:



Aplicações de funções de hash criptográficas

- Exemplos simplificados de **assinaturas digitais**:



Aplicações de funções de hash criptográficas

- Outras aplicações incluem:
 - Arquivo de senha de mão única
 - Ex: O arquivo `/etc/shadow`
 - Os primeiros caracteres da senha indicam o algoritmo de hash usado (varia entre distribuições)
 - \$1 = Algoritmo de hash MD5
 - \$2 = Algoritmo de hash Blowfish
 - \$2a= Algoritmo de hash eksblowfish
 - \$5 = Algoritmo de hash SHA-256
 - \$6 = Algoritmo de hash SHA-512
 - Detecção de **intrusão** e detecção de **vírus**
 - Calcula hash(Arquivo) para cada arquivo e armazena de forma segura
 - Quando necessário, recalcula hash(Arquivo) e compara com o valor armazenado para detectar alterações
 - Usada para construir uma **função pseudoaleatória (PRF)** ou gerador de número **pseudoaleatório (PRNG)**

Aplicações de funções de hash criptográficas

- Outras aplicações incluem (Cont.):
 - Usada para construir uma **função pseudoaleatória (PRF)** ou gerador de número **pseudoaleatório (PRNG)**
 - Digital Timestamping
 - Data e hora de criação/assinatura do documento entram no cálculo do valor hash
 - Útil para proteção de propriedade intelectual

Conceitos e Princípios

- A entrada (mensagem, arquivo, etc) de uma função de hash é vista como uma sequência de blocos de n bits
- A entrada é processada, um bloco de cada vez, em um padrão iterativo para produzir uma função de hash de n bits
- Para um valor de hash $h = H(x)$, dizemos que x é a **pré-imagem** de h
- Uma colisão ocorre se tivermos $x \neq y$ e $H(x) = H(y)$
 - Como funções hash são usadas para garantir a integridade, colisões são indesejáveis

Requisitos e Segurança

- Requisitos para função de hash criptográfica H:

Requisito	Descrição
Tamanho de entrada variável	H pode ser aplicado em um bloco de dados de qualquer tamanho.
Tamanho da saída fixo	H produz uma saída de tamanho fixo.
Eficiência	$H(x)$ é relativamente fácil de calcular para qualquer valor de x informado, através de implementações tanto em hardware quanto em software.
Resistência à pré-imagem (propriedade de mão única)	Para qualquer valor de hash h informado, é computacionalmente impossível encontrar y , de modo que $H(y) = h$.
Resistência à segunda pré-imagem (resistência à colisão fraca)	Para qualquer bloco x informado, é computacionalmente impossível encontrar $y \neq x$ com $H(y) = H(x)$.
Resistência à colisão forte	É computacionalmente impossível encontrar qualquer par (x, y) , de modo que $H(x) = H(y)$.
Pseudoaleatoriedade	A saída de H atende os testes padrão de pseudoaleatoriedade.

Secure Hash Algorithm (SHA)

- Em anos recentes, a função hash mais utilizada
- Desenvolvido pelo *National Institute of Standards and Technology (NIST)* e publicado como um padrão de processamento de informações federais em 1993
- Quando foram descobertos pontos fracos no **SHA-0**, uma nova versão (o **SHA-1**) foi lançada em 1995
- SHA é baseado na função de hash MD4
- Produz um valor de hash de 160 bits
- Em 2002, o NIST produziu uma versão revisada do padrão, definindo três novas versões com tamanhos de valor de hash de 256 bits (**SHA-256**), 384 bits (**SHA-384**) e 512 bits (**SHA-512**)
 - Coletivamente, estas versões são conhecidas com **SHA-2**
- **SHA-3** foi resultado de um concurso do NIST lançado em 2007 e somente em 2015 o NIST anunciou que o SHA-3 se tornaria um padrão de hash

Secure Hash Algorithm (SHA)

- Comparação de parâmetros do SHA:

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Tamanho do resumo da mensagem	160	224	256	384	512
Tamanho da mensagem	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Tamanho do bloco	512	512	512	1024	1024
Tamanho da word	32	32	32	64	64
Número de etapas	80	64	64	80	80

Secure Hash Algorithm (SHA)

- Comparação de parâmetros do SHA:

Tabela 11.5 Parâmetros do SHA-3.

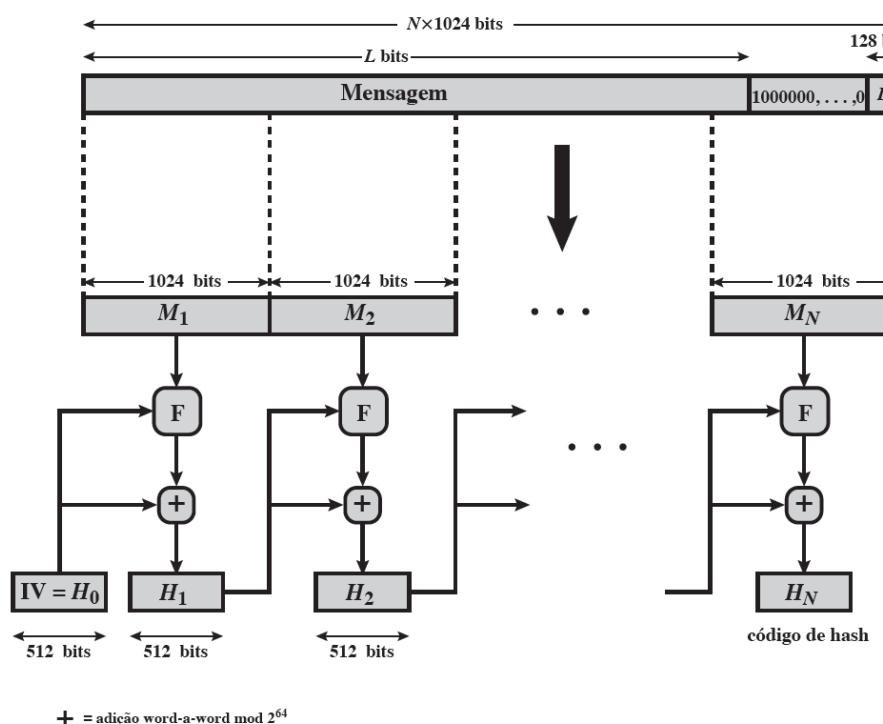
Tamanho do resumo da mensagem	224	256	384	512
Tamanho da mensagem	nenhum máximo	nenhum máximo	nenhum máximo	nenhum máximo
Tamanho do bloco (taxa de bits r)	1152	1088	832	576
Tamanho da word	64	64	64	64
Número de rodadas	24	24	24	24
Capacidade c	448	512	768	1024
Resistência à colisão	2^{112}	2^{128}	2^{192}	2^{256}
Resistência à segunda pré-imagem	2^{224}	2^{256}	2^{384}	2^{512}

Nota: todos os tamanhos e níveis de segurança são medidos em bits.

Secure Hash Algorithm (SHA)

- Geração de resumo da mensagem usando SHA-512

Figura 11.9 Geração de resumo da mensagem usando SHA-512.



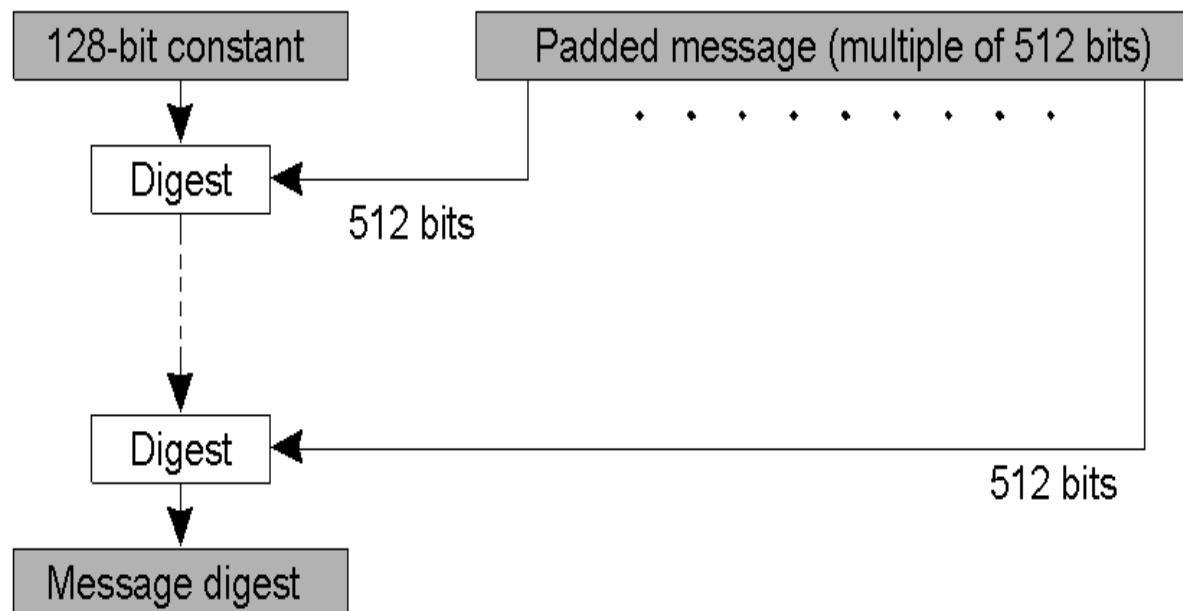
Maiores detalhes sobre o funcionamento do SHA, além de uma implementação de referência (em linguagem C) pode ser consultada na RFC4634 - <https://tools.ietf.org/html/rfc4634>

Message Digest 5 (MD5)

- O algoritmo MD5 foi desenvolvido por Ronald L. Rivest in 1991
 - Versões anteriores (MD2, MD4) apresentaram fraquezas
- De acordo com a RFC 1321
 - O algoritmo MD5 toma como entrada uma mensagem de tamanho arbitário e produz como saída um “fingerprint” ou “message digest” de 128 bits de tamanho
- MD5 é mais vulnerável a ataques de força-bruta devido a sua menor saída: 128 bits contra 160 bits do SHA
- O MD5 é mais rápido que o SHA, pois este último possui mais etapas em seu algoritmo (80 contra 64 do MD5) e um buffer maior (160 bits contra 128 bits do MD5)
- Já foram encontradas colisões para a função de compressão do MD5, enquanto que o SHA-1,2 e 3 permanecem inabaláveis

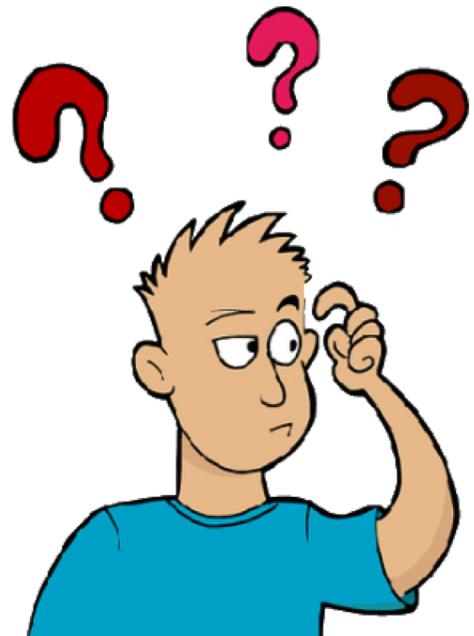
Message Digest 5 (MD5)

- Estrutura do Algoritmo



Maiores detalhes sobre o funcionamento do MD5, além de uma implementação de referência (em linguagem C) pode ser consultada na RFC1321 - <http://www.rfc-base.org/txt/rfc-1321.txt>

Alguma Questão?



Operação “Leva Jeito”



- Por conta de seus conhecimentos em segurança computacional, você foi convocado a colaborar com as investigações da operação “Leva Jeito” na qualidade de consultor técnico.
- Sua contribuição inicial será o desenvolvimento de um software que permita garantir a integridade de documentos extraídos de computadores, preservando assim a cadeia de custódia destes elementos de prova.
- O problema relatado é que documentos extraídos ou coletados em computadores de suspeitos podem potencialmente serem alterados entre a sua coleta e o seu processamento investigativo.
- Assim, a equipe da “Leva Jeito” requisitou a você o desenvolvimento de um software que permita verificar a integridade dos dados a qualquer momento.

Operação “Leva Jeito”



- Implemente o programa **guarda** que, usando de calculo de puro Hash ou HMAC, permita garantir a autenticação de um conjunto de arquivos para uma determinada pasta (recursivamente). O programa deverá ser executado em linha de comando, seguindo a sintaxe:
 - **./guarda <metodo> <opcao> <pasta> <saída>**
 - <metodo> : indica o método a ser utilizado (--hash ou --hmac **senha**)
 - <opcao>: indica a ação a ser desempenhada pelo programa
 - -i : inicia a guarda da pasta indicada em <pasta>, ou seja, faz a leitura de todos os arquivos da pasta (recursivamente) registrando os dados e Hash/HMAC de cada um e armazenando numa estrutura própria (Ex: tabela hash em uma subpasta oculta ./guarda – ou pode ser usada uma árvore B)
 - -t : faz o rastreio (tracking) da pasta indicada em <pasta>, inserindo informações sobre novos arquivos e indicando alterações detectadas/exclusões
 - -x : desativa a guarda e remove a estrutura alocada
 - <pasta> : indica a pasta a ser “guardada”
 - <saída> : indica o arquivo de saída para o relatório (-o **saída**). Caso não seja passado este parâmetro, a saída deve ser feita em tela.

Operação “Leva Jeito”



- O programa deve gerar como saída:
 - **Para a opção –i**
 - Gerar uma lista de TODOS os arquivos lidos e seu respectivo valor de hash
 - **Para a opção –t**
 - Gerar uma lista apenas dos arquivos NOVOS/ALTERADOS/REMOVIDOS. Diferenciar as categorias por cor ou apresentando o status (Exemplo: [R] /home/silvio/teste1.txt; [A] /home/silvio/dados.dat; [N] /home/silvio/novos/secreto.txt)
 - **Para a opção –x**
 - Apenas uma mensagem indicando que a operação foi completada.
- O programa deve permitir “guardar” múltiplas pastas.

Operação “Leva Jeito”



- Observações
 - Pode ser utilizada alguma biblioteca (de acordo com sua linguagem) para a implementação da função Hash embutida (MD5, SHA-1, SHA-2, etc)
 - O algoritmo HMAC e o programa principal devem ser implementados por você
 - O programa deve estar devidamente comentado!!!
 - Faça uso das boas práticas de programação
- Referências
 - <https://tools.ietf.org/html/rfc2104>
 - <https://csrc.nist.gov/csrc/media/publications/fips/198/archive/2002-03-06/documents/fips-198a.pdf>
 - <http://pt.wikipedia.org/wiki/HMAC>