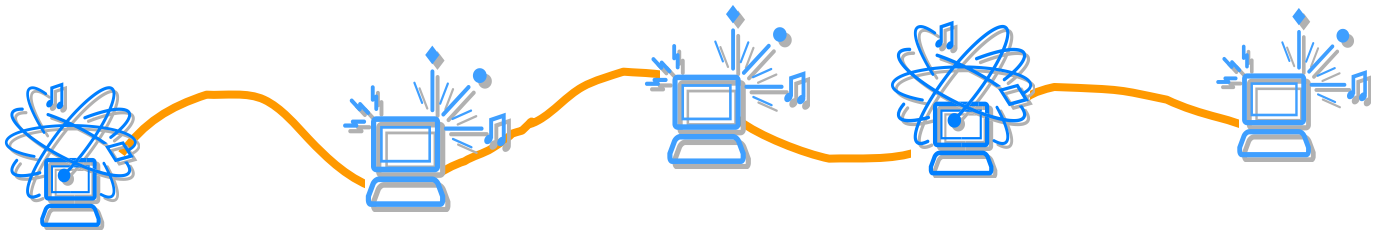
Segurança em Redes
Autenticação



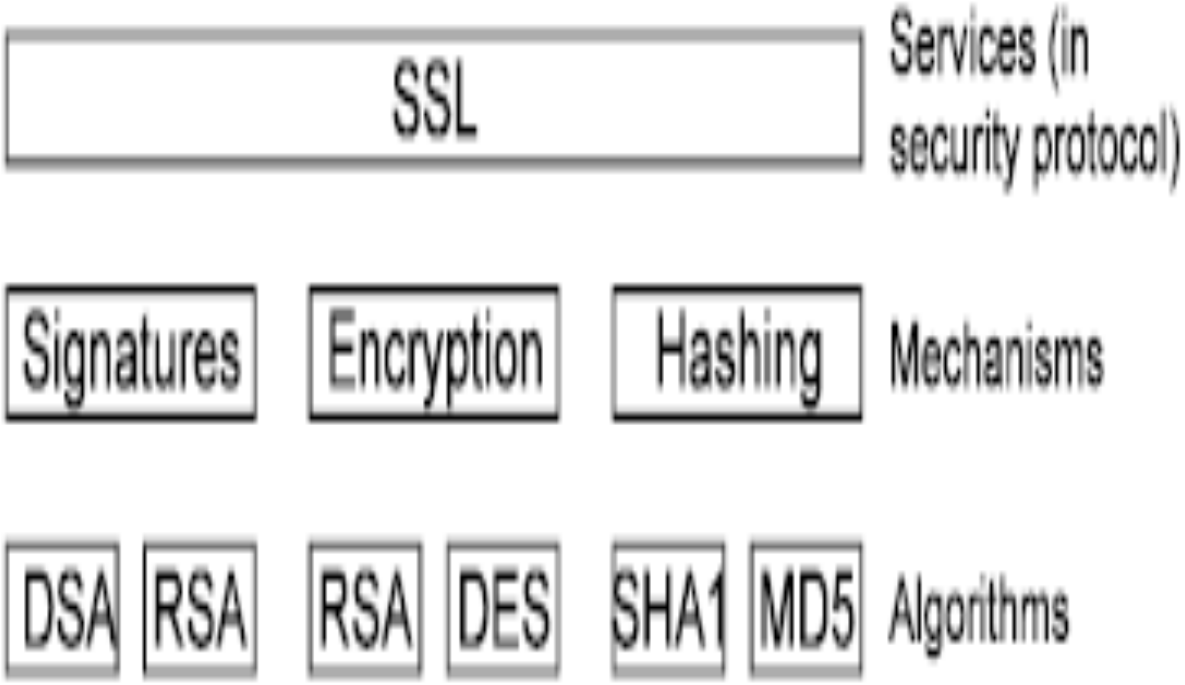
Redes de Comunicação

Departamento de Engenharia da Electrónica e Telecomunicações
e de Computadores

Instituto Superior de Engenharia de Lisboa

Serviços, mecanismos e algoritmos

Um protocolo de segurança pode providenciar um ou mais serviços



- Serviços são construídos com mecanismos
- Mecanismos são implementados usando algoritmos

Autenticação dos utilizadores

- Alguma coisa que o utilizador sabe (por ex. uma *password*)
- Alguma coisa que o utilizador tem na sua posse (por ex. um *smart card*)
- Alguma coisa que o utilizador é ou que faz parte (biométrica, por ex. impressão digital)
- Por vezes outra informação:
  - Localização do utilizador

- Usualmente são necessários pelo menos dois deste métodos para uma autenticação de um utilizador ser considerada forte.

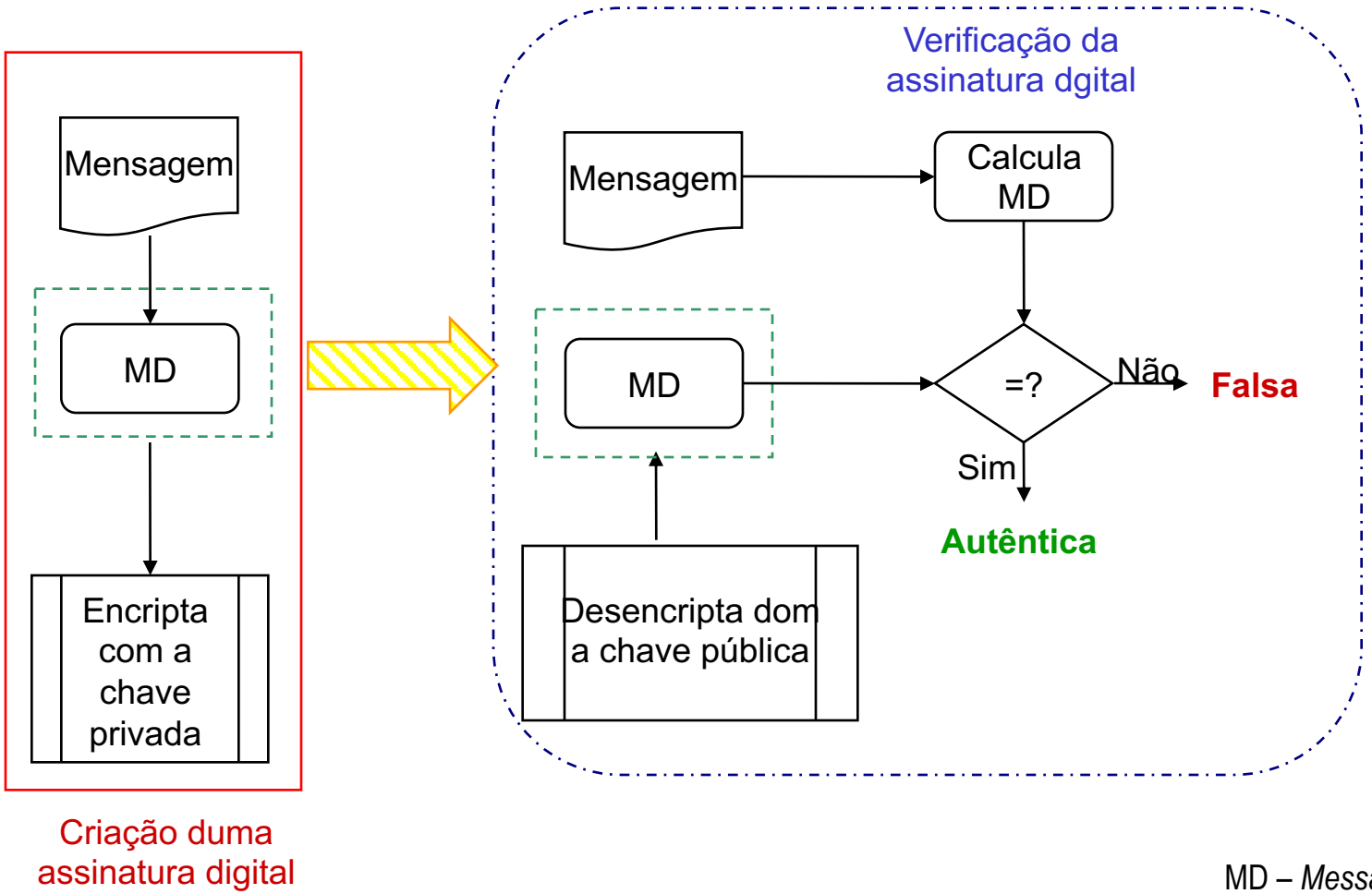
Autenticação e assinaturas digitais

Uma das áreas que normalmente provoca maior confusão na segurança em redes é a da autenticação e o tema com ela relacionado - **assinaturas digitais**.

- A autenticação de mensagens preocupa-se com:
  - **Proteger a integridade da mensagem**
  - **Validar a identidade do originador da mensagem**
  - **Não-repudiação da origem**

- Pretende ser o equivalente eletrónico da assinatura manual e, para isso, um autenticador, assinatura ou código de autenticação da mensagem é enviado com a mensagem.

Exemplo de assinatura digital



MD – *Message Digest*

Formas de autenticação de mensagens

- Existem várias formas de autenticação, entre elas:
  - Encriptação simétrica
  - Encriptação assimétrica
  - Funções de *Hash*
  - MAC – *Message Authentication Code* [MAC = F(K, M)]
  - Outros esquemas de assinatura

Autenticação com cifra de chave simétrica

- Pode ser-se levado a pensar que a encriptação garante a autenticação e a integridade mas tal, por si só, não é verdade. A encriptação não foi estudada para providenciar integridade de dados, desta forma não se deve assumir que a utilização de encriptação dá sempre garantias de integridade.

- Não devem ser confundidas encriptação e autenticação. O fim com que foram criados os respectivos algoritmos teve um fim específico em mente pelo que foram para isso optimizados.

Autenticação com cifra de chave simétrica

- Se uma mensagem for encriptada usando uma chave de sessão conhecida apenas de quem envia e do destinatário então a mensagem pode também ser autenticada mutuamente
  - Desde que apenas o originador e o destinatário possam ter criado a chave
  - Qualquer interferência irá corromper a mensagem (assumindo que possui redundância suficiente para detectar a alteração)
  - Isto não fornece protecção contra a repudiação dado que é impossível provar quem criou a mensagem (qual dos possuidores da chave secreta)
  - Integridade nem sempre garantida pelos algoritmos de cifra. Depende dos algoritmos e da forma como se utilizam.

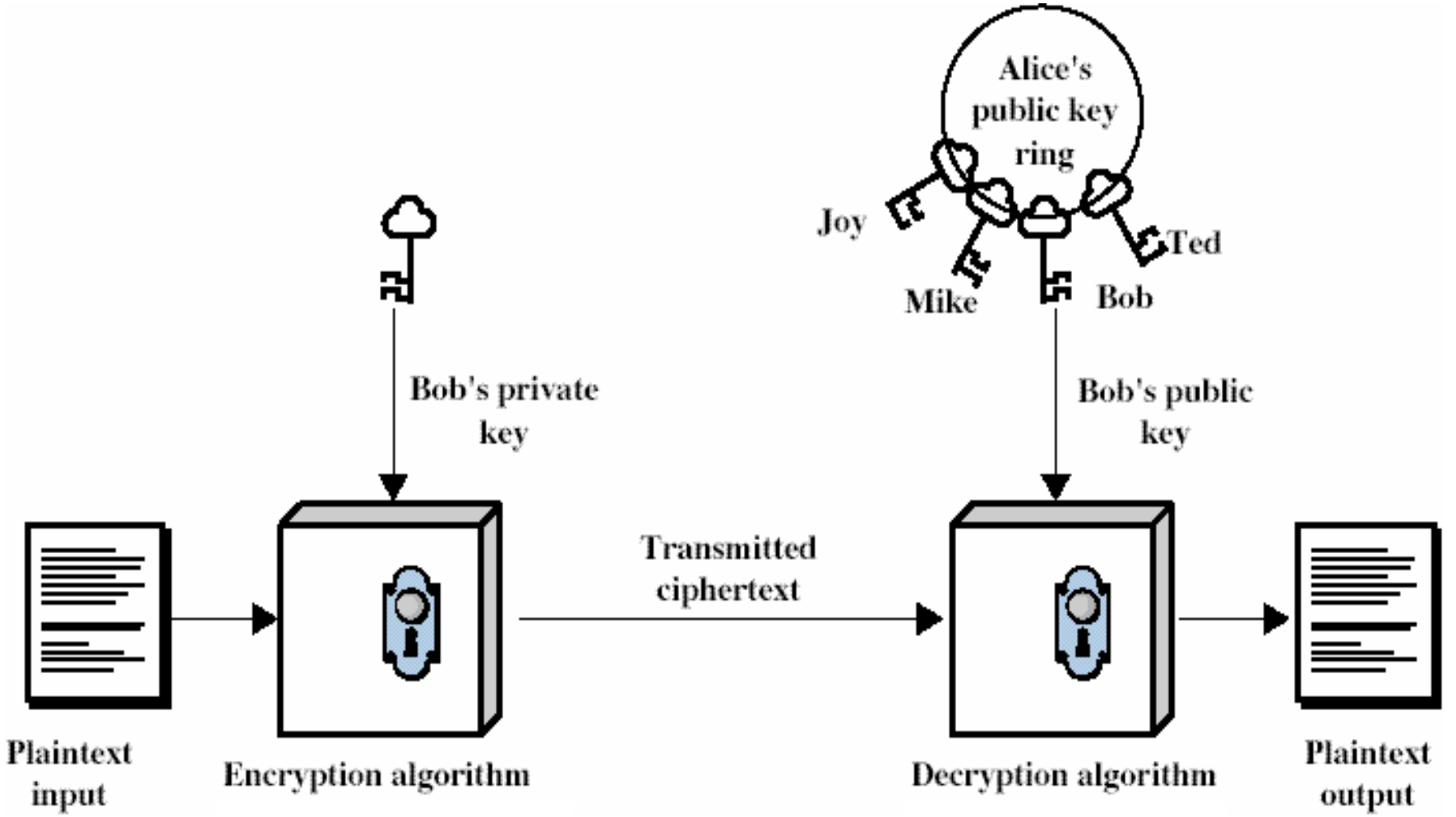
Autenticação com cifra de chave simétrica

- A autenticação da mensagem pode ser realizada usando os modos normalizados de utilização de uma cifra de bloco
  - Às vezes não se pretende enviar a mensagem cifrada
  - Pode-se usar os modos CBC ou CFB e enviar apenas o bloco final, desde que este dependa de todos os bits anteriores da mensagem
  - Não é necessária função de *hash* dado que este método aceita qualquer dimensão à entrada e produz uma saída de dimensão fixa
  - Usualmente utiliza um IV (vector inicial) conhecido
  - A Austrália utilizou um esquema destes na norma AS8205
  - A maior desvantagem é o pequeno tamanho do MAC resultante, dado que 64 bits é provavelmente muito pequeno para resistir a ataques de força bruta, entre outros.

Autenticação com chaves assimétricas

- **Esquemas de assinaturas com chave pública** – a chave privada assina (cria) a assinatura e a chave pública verifica assinatura
- Apenas o dono (da chave privada) pode criar a assinatura digital, pelo que pode ser utilizado para verificar quem criou a mensagem
- Qualquer pessoa que conheça a chave pública pode verificar a validade da assinatura (desde que confie na identidade do dono da chave pública – **problema da distribuição de chaves**)
- Normalmente não se assina toda a mensagem (duplicando a quantidade de informação original, mas **apenas um *hash*** da mensagem)
- **As assinaturas digitais fornecem não-repudiação da origem** dado que o algoritmo de cifra assimétrica é utilizado na sua criação, e havendo que assegurar que *timestamps* e as redundâncias sejam incorporadas na assinatura

Utilização de criptografia de chave assimétrica para autenticação



Utilização de criptografia de chave assimétrica para autenticação

- A cifra assimétrica presta-se à utilização como mecanismo de autenticação quando a mensagem é cifrada com a chave privada e decifrada com a chave pública.
- Tem o problema da cifra assimétrica ser muito lenta.
- Pode ser utilizada cifrando-se não a mensagem mas apenas um *hash* da mensagem.

Exemplo RSA

Tendo em conta o algoritmo RSA e *p*=11 e *q*=3, determine: Uma chave pública {*e, n*} e uma privada {*d, n*}

1 – calcular $n = p \times q = 11 \times 3 = 33$

2 – Calcular Totiente o phi de *n:* $\phi(n) = (p - 1) \times (q - 1) = (11-1) \times (3-1) = 10 \times 2 = 20$

3 – Escolher *e*

- um número primo não divisor do Totiente de phi (20)
- *começar a tentar: 1, 3, 5, 7, 9, 11 …*
- *Qual o primeiro não divisor de 20 ? -> 3 -> logo fazemos e = 3*

3.1 – Validar pelo máximo divisor comum (gdc) com *p*

- *calcular: gdc(e , p-1) = gdc (3 , 10) = 1 --> ou seja, 3 e 10 não têm fatores em comum que não o 1*

3.2 – Validar pelo máximo divisor comum (gdc) com *q*

- *calcular: gdc(e , q-1) = gdc (3 , 2) = 1 --> ou seja, 3 e 2 não têm fatores em comum que não o 1*

Logo, validamos: gdc(e, ϕ) = gdc (e, (p-1), (q-1)) = gdc (3,20) = 1 ou seja, 3 e 20 não têm fatores comuns que não o 1

Portanto *e = 3*

4 – Só falta calcular *d.* Usamos $ed = 1 \bmod \phi \rightarrow d = 1/e \bmod \phi \rightarrow 1/3 \bmod 20$.

i - ou seja, procurar um numero tal que ϕ divida por (ed-1).

ii - testamos *d=1*, *d=2*, … e chegamos a *d=7*.

iii - *o resultado é 20 = (3x7 – 1) = 20 , logo satisfaz 4.i*

iv – portanto o nosso d = 7

5 – Chaves: pública {*e, n*} = {*3, 33*} e uma privada {*d, n*} = {*7,33*}

Exemplo Cifra

Questão: Uma mensagem secreta foi interceptada durante uma acção de vigilância eletrónica.

Do processo de criptoanálise resulta a identificação da cifra como sendo uma cifra de César com Shift igual a 3.

A mensagem encriptada é "KDSSBFLSKHU."

Qual é a mensagem original?

HAPPYCIPHER (right)

EXMMVZFMEBO (left)

Exemplo Cifra 2

Questão: Uma mensagem secreta foi detectada num documento.

Do processo de criptoanálise não resulta a identificação do tipo de cifra/codificação.

A mensagem encriptada é
"RVhFTVBMTyBCQVNFNjQgU0lNUExFUw=="

1 – Qual é o tipo de cifra/codificação?
2 - Qual é a mensagem original?

EXEMPLO BASE64 SIMPLES (BASE64)

Hashing: Ferramenta final

- A encriptação procura obscurecer o texto em claro com uma chave, mas de maneira a que o texto possa ser recuperado

- As **funções de *hash*** produzem um resultado de dimensão fixa, independentemente da dimensão dos dados de entrada

- A ideia é o *hash* mudar substancialmente mesmo que só seja alterado um único bit do texto de entrada
  - Semelhante ao *checksum* no que respeita a garantir a integridade dos dados
  - Depende da função de *hash* ser unidireccional

Autenticação com funções de *hash*

- É habitualmente assumido que a função de *hash* é pública e sem chave
- Os CRC tradicionais não satisfazem os requisitos anteriores (porquê?)
- O comprimento deve ser grande de maneira a resistir a ataques tipo "data de nascimento/*birthday*" (64 bits é agora assumido como muito pequeno, 128-512 propostos)

Função de *hash*

- Cria uma “impressão digital” de uma mensagem”

Qualquer um pode alterar os dados e calcular uma novo valor de *hash*

- *O hash tem de ser protegido de alguma forma*

Tipos de *hash*

- **"Normalizado"**
  - A mensagem dá entrada na função de *hash* (ex. SHA-1)
  - A função de *hash* calcula de acordo com a norma
  - A mesma mensagem produz sempre o mesmo resultado (*hash*)

- **Com chave, ou *hash* seguro**
  - A mensagem é uma das entradas da função de *hash*
  - A chave secreta é outra entrada
  - A saída depende de ambos, chave secreta e mensagem

Autenticação com funções de *hash*

- As funções de *hash* são utilizadas para representar mensagens de qualquer dimensão num bloco de dimensão fixa, normalmente para serem assinados posteriormente por um algoritmo de assinatura digital
- **Uma boa função de *hash* H deve possuir as seguinte propriedades:**
  - **H deve destruir toda a relação de forma existente levando à impossibilidade do cálculo do *hash* de duas mensagens combinadas dados os respectivos *hashes* individuais.**
  - **H deve ser calculado sobre toda a mensagem**
  - **H deve ser uma função unidireccional de maneira a que a mensagem não possa ser descoberta a partir da sua assinatura**
  - **Deve ser computacionalmente irrealizável dados uma mensagem e o seu valor de *hash* descobrir outra mensagem com o mesmo valor de *hash***
  - **Deve resistir a ataques da data de nascimento (*birthday atacks*) (encontrar duas mensagens com o mesmo valor de *hash*, talvez iterando através de permutações menores de duas mensagens)**

Propriedades das funções de *hash*

- A finalidade de uma função de *hash* é produzir uma "impressão digital"
- **Propriedades duma função de *hash* H:**
  1. H pode ser aplicada a um bloco de dados de qualquer dimensão
  2. H produz uma saída de comprimento fixo
  3. H(x) é fácil de calcular para qualquer x
  4. Para qualquer bloco x, sabendo h, é computacionalmente irrealizável achar x tal que H(x) = h
  5. Para qualquer bloco x, é computacionalmente irrealizável achar y<>x com H(y)=H(x)
  6. É computacionalmente irrealizável achar um par (x, y) tal que H(x)=H(y)

Termos utilizados no *hash*

- **Unidireccional**
  - *H(x)* é unidireccional se, sabendo h, for computacionalmente irrealizável achar *x* tal que *H(x)* = *h*
  - i.e. *H(x)* é difícil de inverter

- **Livre de colisões**
  - Livre de colisões fraco: Dado *x*, ser computacionalmente irrealizável achar *y* em que $y \neq x$, tal que $H(x) = H(y)$
  - Livre de colisões forte: Ser computacionalmente irrealizável achar quaisquer duas mensagens *x* e *y* tal que $H(x) = H(y)$

Onde são necessárias as propriedades do *hash*?

- As *passwords* do UNIX são guardadas na forma de *hash*
  - Sentido único: Difícil recuperar as *passwords*
  - Resistência fraca às colisões: Difícil encontrar outra *password* que tenha um *hash* com o mesmo valor
- Integridade na distribuição de software
  - Resistência fraca às colisões
- Lances nos leilões
  - Alice quer fazer o lance B, envia H(B), mais tarde revelará B
  - Sentido único: Os concorrentes rivais não devem poder obter B
  - Resistência às colisões: Alice não deve ser capaz de alterar a sua decisão e oferecer B' tal que H(B)=H(B')

Funções comuns de *hash*

- MD5
  - 128 bits à saída
  - Desenhado por Ron Rivest, muito usado
  - Resistência às colisões quebrada (verão de 2004)
- RIPEMD-160
  - Variante a 160 bits do MD-5
- SHA-1 (*Secure Hash Algorithm*)
  - 160 bits à saída
  - Norma do governo do EUA
    - Também é utilizado como algoritmo de *hash* na norma de assinaturas digitais *Digital Signature Standard* (DSS)
    - O governo dos EUA aconselha a passagem para o SHA-2 e a deixar de se usar o SHA-1 a partir do fim de 2010
- SHA-2 ”família de funções de *hash* (SHA-224, SHA-256, SHA-384 and SHA-512)

Ciclo de vida dos *hashes* mais comuns

Life cycles of popular cryptographic hashes (the "Breakout" chart)

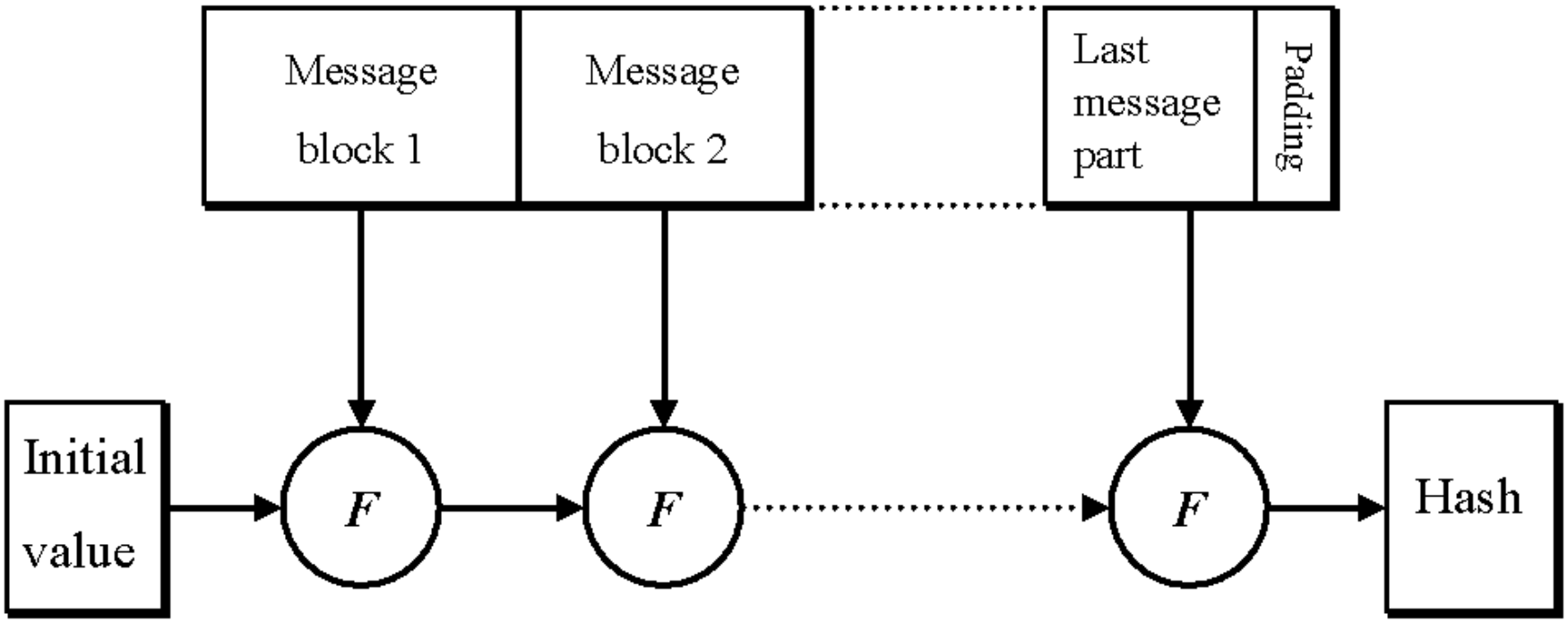
Function	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
Snefru	Unbroken	Unbroken	Unbroken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken
MD4	Unbroken	Weakened	Weakened	Weakened	Weakened	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken	Broken
MD5			Unbroken	Unbroken	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Broken	Broken	Broken	Broken	Broken	Broken
MD2			Unbroken	Unbroken	Unbroken	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Broken	Broken	Broken	Broken	Broken	Broken
RIPEMD			Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Broken	Broken	Broken	Broken	Broken	Broken
HAVAL-128			Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Weakened	Weakened	Weakened	Weakened	Broken	Broken	Broken	Broken	Broken	Broken
SHA-0				Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened	Broken	Broken	Broken	Broken	Broken	Broken
SHA-1						Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Weakened	Weakened	Weakened	Weakened	Weakened	Weakened
RIPEMD-128 [1]							Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken
RIPEMD-160							Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken
SHA-2 family											Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken	Unbroken

Key Unbroken Weakened Broken

[1] Note that 128-bit hashes are at best 2^64 complexity to break; using a 128-bit hash is irresponsible based on sheer digest length.

(http://valerieaurora.org/monkey.html)

Exemplo de função de *Hash*



Source: RSA Laboratories, Inc.

Função simples de *Hash* usando XOR

	bit 1	bit 2	• • •	bit *n*
block 1	b_{11}	b_{21}		b_{n1}
block 2	b_{12}	b_{22}		b_{n2}
	•	•	•	•
	•	•	•	•
	•	•	•	•
block *m*	b_{1m}	b_{2m}		b_{nm}
hash code	C_1	C_2		C_n

Um *shift* circular do valor de *hash* após o processamento de cada bloco melhoraria a função

Hash/Message digest

Função unidirecional que mapeia uma mensagem de qualquer dimensão num bloco de saída de dimensão fixa (*hash* ou *message digest*)

- **Evolução dos algoritmos**

MD2 → MD4 → MD5 → SHA → SHA1

Funções comuns de Hash

Algoritmo	MD2	MD4	MD5	SHA-1
Comprimento da saída	128 bits	128 bits	128 bits	160 bits
Dimensão do bloco	128 bits	512 bits	512 bits	512 bits
Especificação	RFC 1319	RFC 1320	RFC 1321	FIPS 180-1

Usos e beneficios das funções de *Hash*

- Verificar a integridade do bloco de dados
  - e.g. uma mensagem
- Mais rápido calcular o *hash* do que produzir a versão cifrada da mensagem
- Produz sempre uma saída de dimensão fixa conhecida
- Útil em muitas aplicações

Quantos bits?

- Se o comprimento do *hash* for n bits, demorar-se-à $O(2^{n/2})$ para se achar uma mensagem com um *hash igual* (problema do nascimento (*birthday problem*)). **Mensagem com 2^{27} variantes:**

Type 1 message

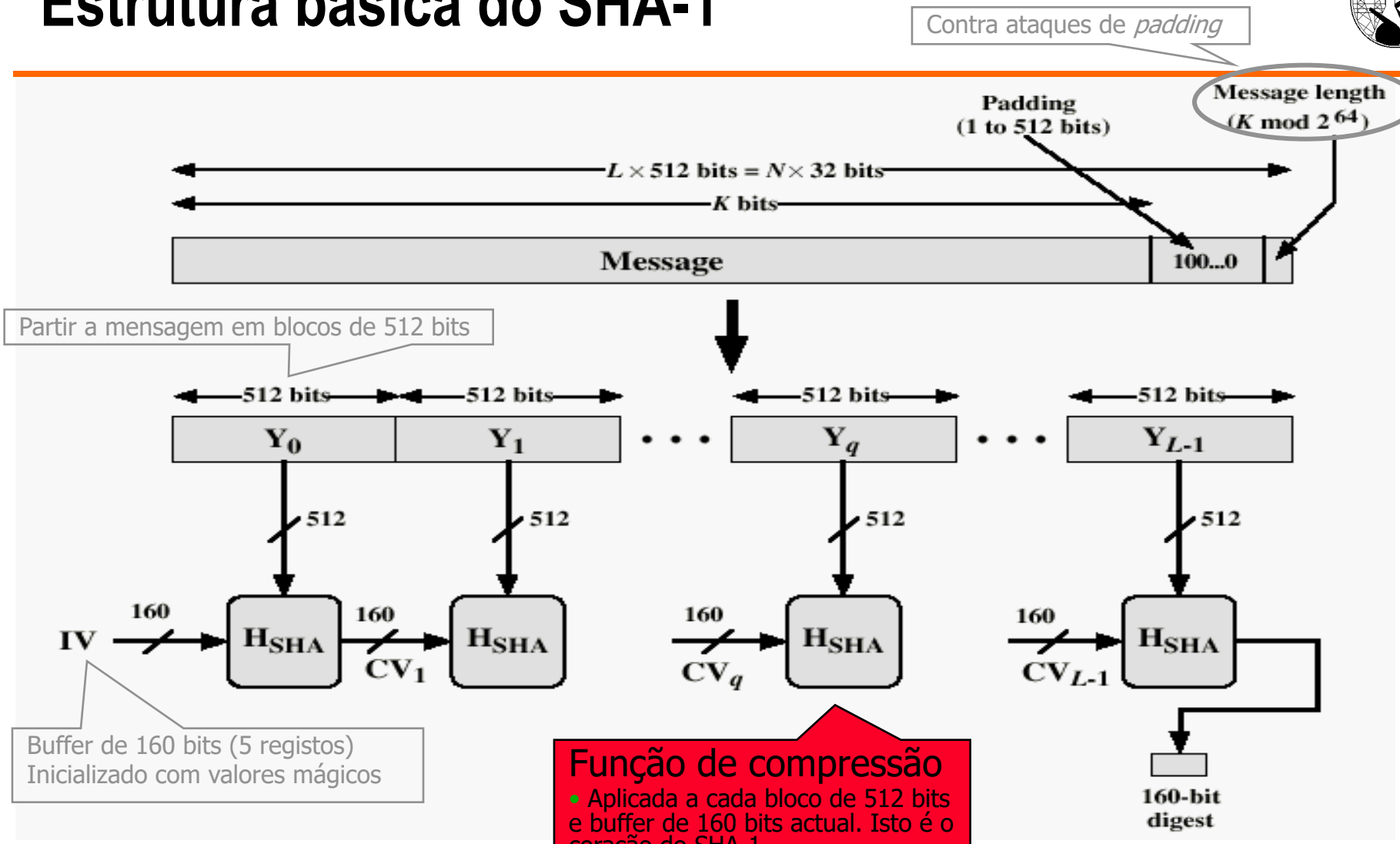
I am writing {this memo | } to {demand | request | inform you} that {Fred | Mr. Fred Jones} {must | } be {fired | terminated} {at once | immediately}. As the {July 11 | 11 July} {memo | memorandum} {from | issued by} {personnel | human resources} states, to meet {our | the corporate} {quarterly | third quarter} budget {targets | goals}, {we must eliminate all discretionary spending | all discretionary spending must be eliminated}.

{Despite | Ignoring} that {memo | memorandum | order}, Fred {ordered | purchased} {Post-Its | nonessential supplies} in a flagrant disregard for the company's {budgetary crisis | current financial difficulties}.

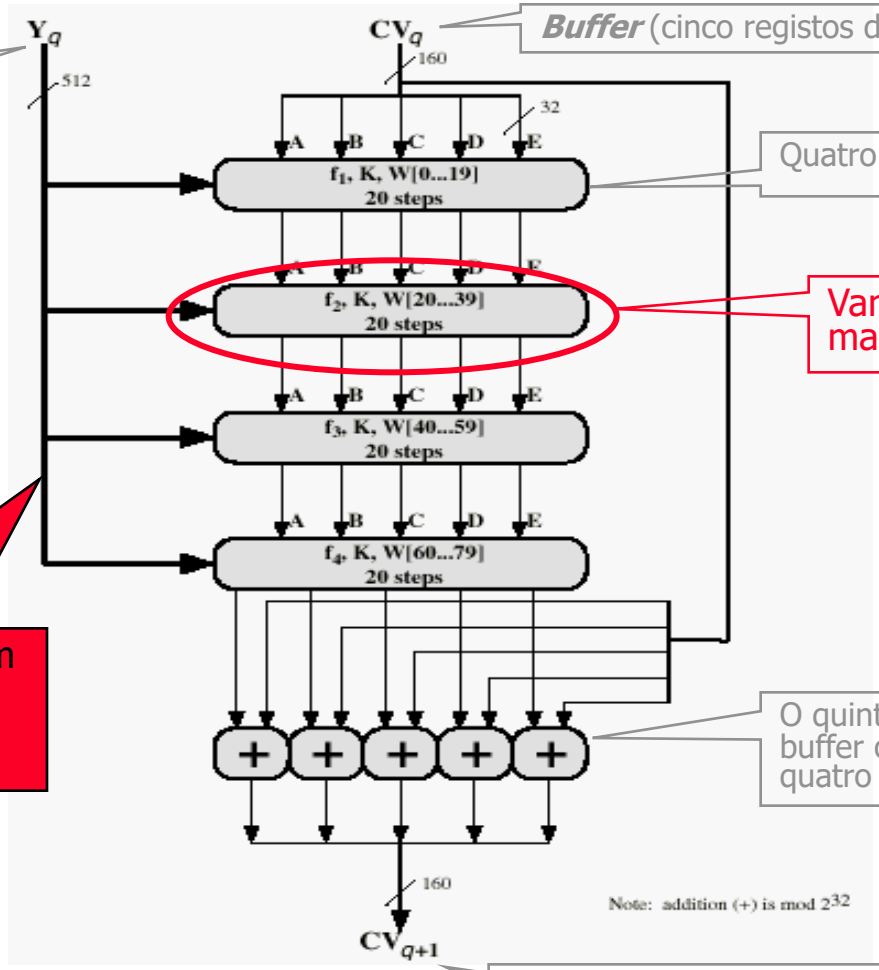
Type 2 message

I am writing {this letter | this memo | this memorandum | } to {officially | } commend Fred {Jones | } for his {courage and independent thinking | independent thinking and courage}. {He | Fred} {clearly | } understands {the need | how} to get {the | his} job {done | accomplished} {at all costs | by whatever means necessary}, and {knows | can see} when to ignore bureaucratic {nonsense | impediments}. I {am hereby recommending | hereby recommend} {him | Fred} for {promotion | immediate advancement} and {further | } recommend a {hefty | large} {salary | compensation} increase.

Estrutura básica do SHA-1



Função de compressão do SHA-1



Bloco actual da mensagem

Buffer (cinco registos de 32 bits A,B,C,D,E)

Quatro ***rounds***, 20 passos em cada

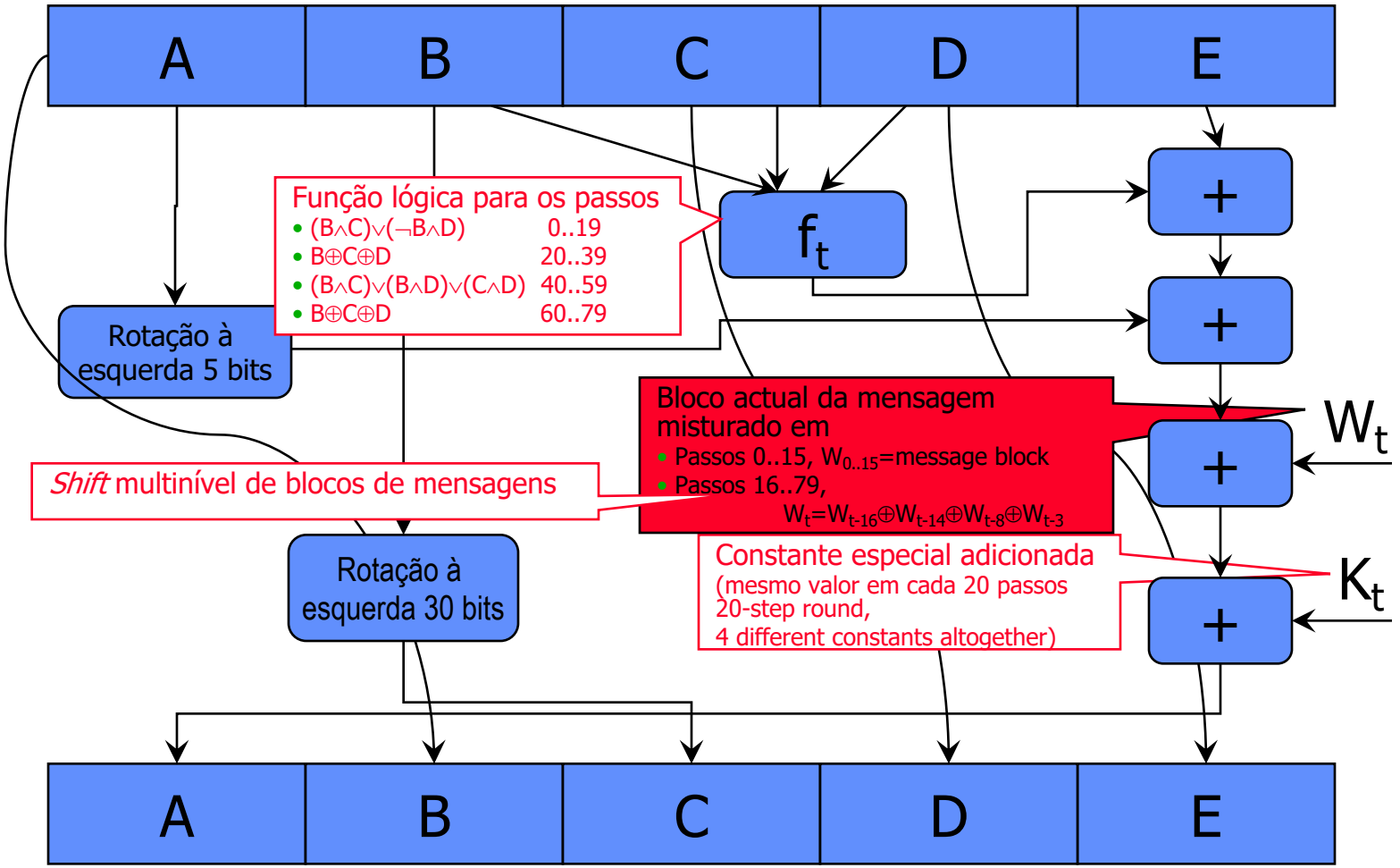
Vamos ver cada passo mais em detalhe

Muito semelhante à cifra em bloco com a mensagem a servir de chave para cada *round*

O quinto *round* acrescenta o buffer original ao resultado dos quatro *rounds*

O *buffer* contem o valor final do *hash*

Um passo do SHA-1 (80 passos no total)



Exemplo SHA1

Usando CiberChef:

1 - Escrever uma frase e calcular hash SHA-1
2 – Alterar um caracter e voltar a calcular hash.

3 – Criar um ficheiro .txt com a palavra TESTE. Calcular hash
4 – Alterar texto para teste (minúsculas). Calcular hash.

5 – No Cyberchef colocar uma sequencia binária. Calcular hash.
6 – Alterar 1 bit. Calcular hash.

7 – No ficheiro, colocar a mesma sequencia binaria. Calcular hash
8 – Alterar 1 bit e guardar ficheiro. Calcular hash.

9 – Encontrar um ficheiro qualquer e calcular a sua hash.
10 – Encontrar um outro ficheiro de tamanho maior e repetir.

Objetivo. Verificar por comparação que qualquer alteração leva a um resultado diferente e sempre com o mesmo tamanho de hash.

Outras funções seguras de *hash*

	SHA-1	**MD5**	**RIPEMD-160**
Comprimento do *hash*	160 bits	128 bits	160 bits
Unidade básica de processamento	512 bits	512 bits	512 bits
Número de passos	80 (4 rounds of 20)	64 (4 rounds of 16)	160 (5 paired rounds of 16)
Comprimento máximo da mensagem	2^{64}-1 bits		

Autenticação com MAC

- O MAC (***Message Authentication Code***) é gerado através de um algoritmo que depende da mensagem e de uma chave conhecida apenas de quem envia e do destinatário [**MAC = F(*Key*, *Message*)**]

- A mensagem pode ser de qualquer comprimento

- O MAC pode ser de qualquer dimensão mas na maioria dos casos é de dimensão fixa, requerendo a utilização de uma função de *hash* para reduzir a mensagem à dimensão necessária

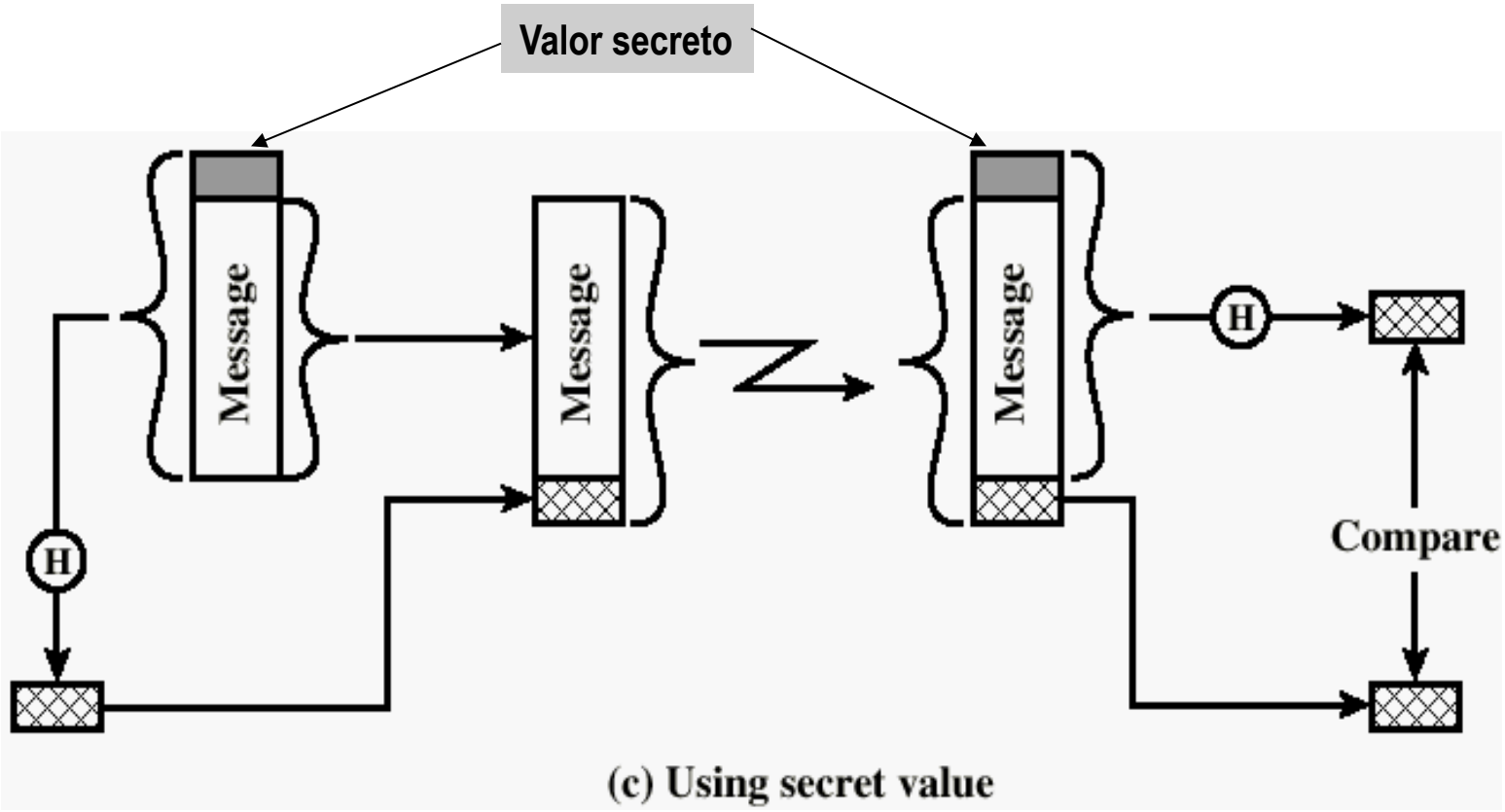
- A necessidade de precaução contra as retransmissões de mensagens e respectivos MACs implica a utilização de números de sequência nas mensagens, *timestamp* e valores aleatórios negociados.

MAC

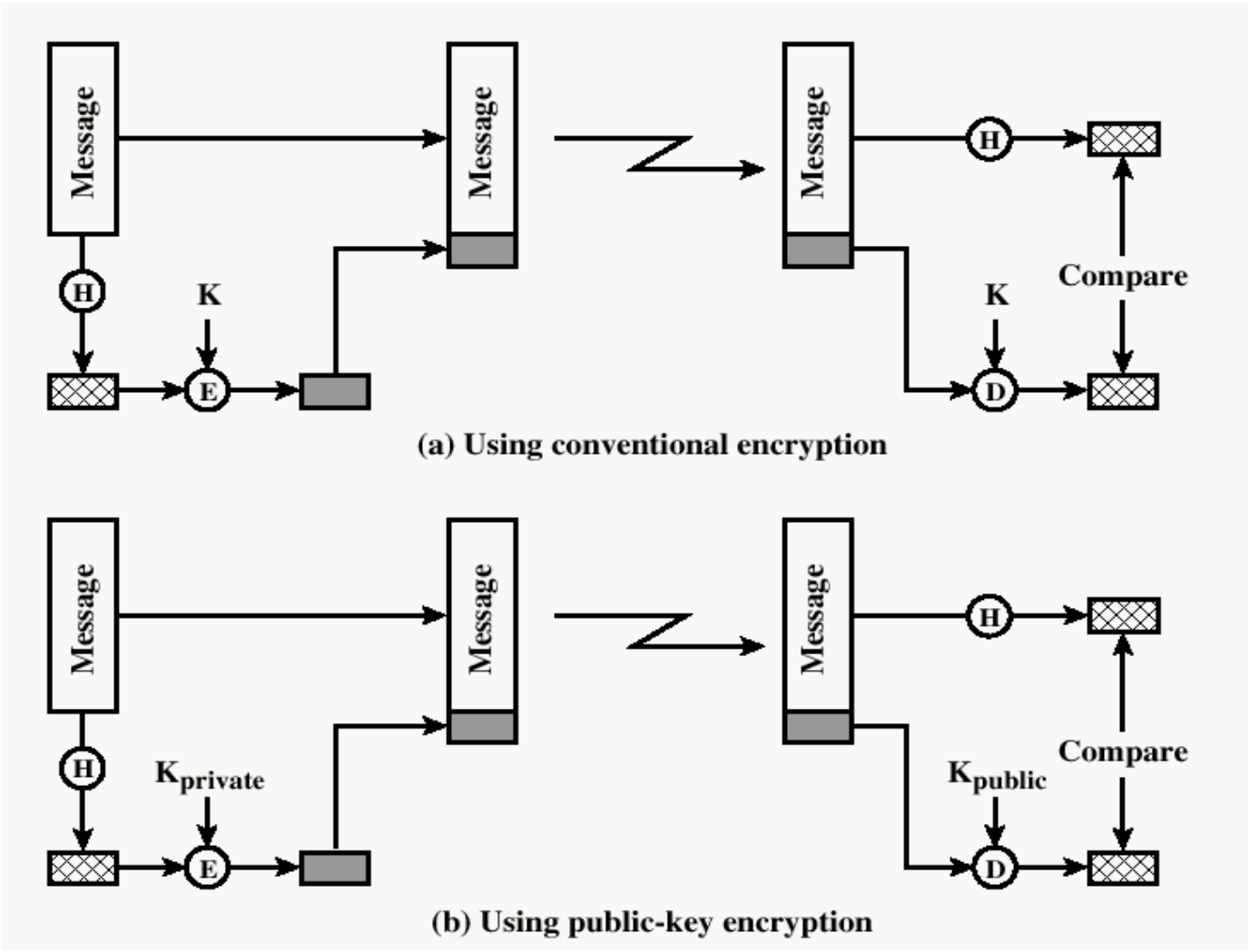
- MD(M)?
- MD(K_{AB}|M): Apenas aqueles que conhecem o “segredo” é que podem verificar

Autenticação com MAC

É adicionado um **valor secreto** antes do *hash* e removido antes da transmissão



Autenticação com função de *hash* unidirecional



HMAC - *Hash Message Authentication Code*

- Usa um MAC derivado de uma função criptográfica de *hash* como o SHA-1

- Motivação:
  - As funções criptográficas de *hash* são de execução mais rápida em software do que os algoritmos como o DES
  - As livrarias de código de funções de *hash* estão bastante disponíveis
  - Não existem restrições tão apertadas à exportação, nos EUA, como existem para os algoritmos de cifra.

HMAC

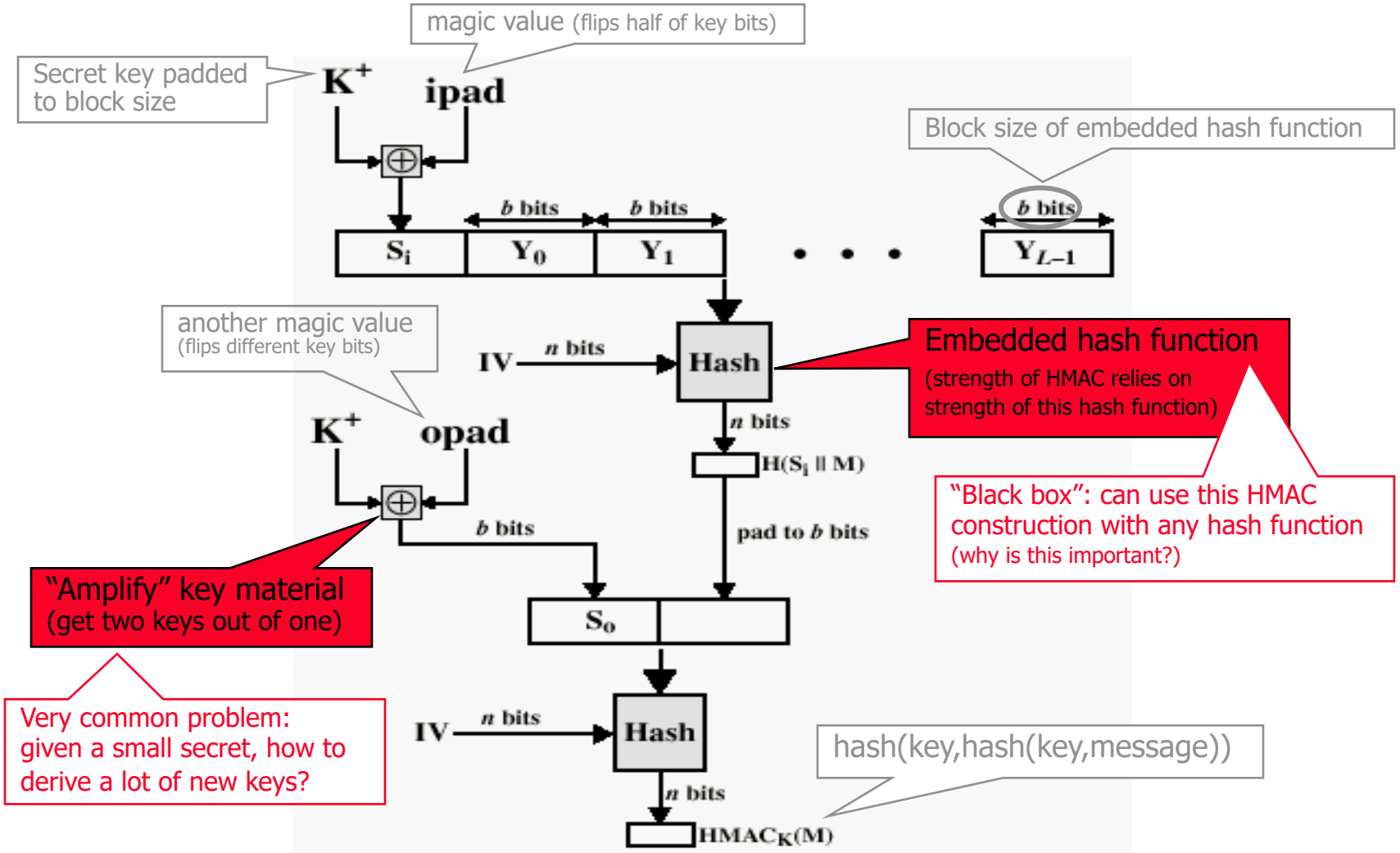
- Construir um MAC aplicando uma função de *hash* a uma mensagem e a uma chave
  - Pode também ser reutilizada cifra em vez de *hash*, mas …
  - O *hash* é mais rápido do que a cifra
  - As bibliotecas com código de funções de *hash* estão largamente disponíveis
  - Pode-se substituir facilmente uma função de *hash* por outra
  - As funções de *hash* sofrem menos restrições pelos EUA à exportação do que as de cifra
- Inventada por Bellare, Canetti, e Krawczyk (1996)
  - A força do HMAC é determinada pela análise criptográfica
- Obrigatória no IPsec, também utilizada em SSL/TLS

HMAC - *Hash Message Authentication Code*

•H(·) é uma função *hash* criptográfica
•***K*** é uma chave secreta preenchida com zeros extras à direita para entrada no bloco do tamanho da função *hash* , ou o *hash* da chave original se esta é maior que o tamanho do bloco
• ***m*** é a mensagem a ser autenticada
• ‖ denota concatenação
• ⊕ denota ou exclusivo (**XOR**)
• **opad** é o preenchimento externo (**0x5c5c5c…5c5c**, um bloco de comprimento constante hexadecimal)
• **ipad** é o preenchimento interno (**0x363636…3636**, um bloco de comprimento constante hexadecimal)
Então **HMAC**(*K*,*m*) é definido matematicamente por:

$$\mathbf{HMAC}(K,m) = \mathbf{H}\big((K \oplus \text{opad}) \,\|\, \mathbf{H}\big((K \oplus \text{ipad}) \,\|\, m\big)\big).$$

Estructura do HMAC



- In August 1991 the National Institute of Standards and Technology (NIST) proposed DSA for use in their Digital Signature Standard (DSS).
- NIST adopted DSA as a Federal standard (FIPS 186) in 1994.
- The DSA algorithm involves four operations:
  - key generation (which creates the key pair)
  - key distribution
  - signing and
  - signature verification.

Utilização do SHA-1 com o DSA