UNIVERSIDADE FEDERAL
DE SANTA CATARINA

ine
CTC • UFSC
Informática e Estatística

INE 5680
Segurança da Informação e de Redes

Criptografia – (Hash, MAC) + Criptografia autenticada (AE – *Authenticaded Encryption*)

Profa: Carla Merkle Westphall
carla.merkle.westphall@ufsc.br

Funções hash criptográficas

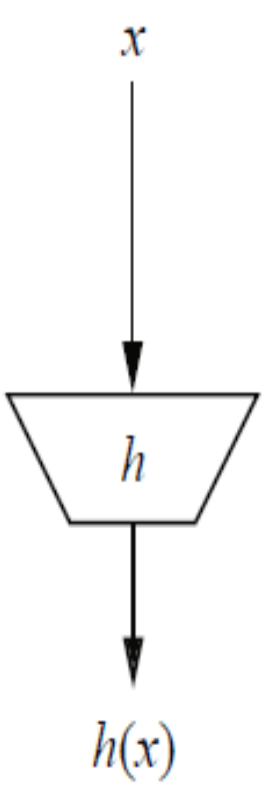
Funções Hash criptográficas

- transformam uma msg de entrada em uma saída chamada hashcode, resultado do hash, valor hash ou somente hash
- Mapeia strings de tamanho finito arbitrário em string de tamanho fixo (n bits)
- Garantem sentido unidirecional da função (one-way): sabendo o valor do hash é computacionalmente inviável encontrar a mensagem que deu origem àquele valor (sentido inverso da função é difícil)
- Garantem resistência contra “second pre-image”: computacionalmente inviável encontrar uma 2ª msg que possui o mesmo valor hash que uma 1ª msg conhecida
- Garantem resistência a colisão: muito difícil encontrar duas mensagens que geram o mesmo valor de saída da função hash

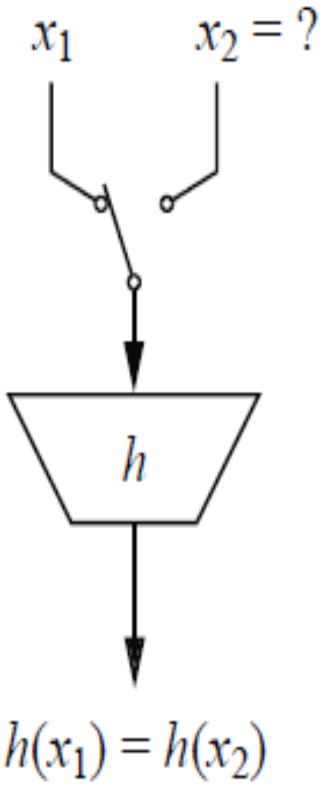
Dois tipos básicos de hash:

- Sem chave (só a msg é parâmetro de entrada)
- Com chave (msg e chave são parâmetros de entrada)

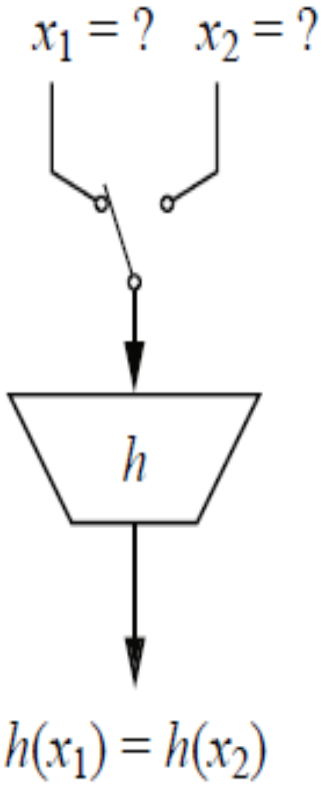
Funções hash criptográficas - propriedades



preimage resistance



second preimage resistance



collision resistance

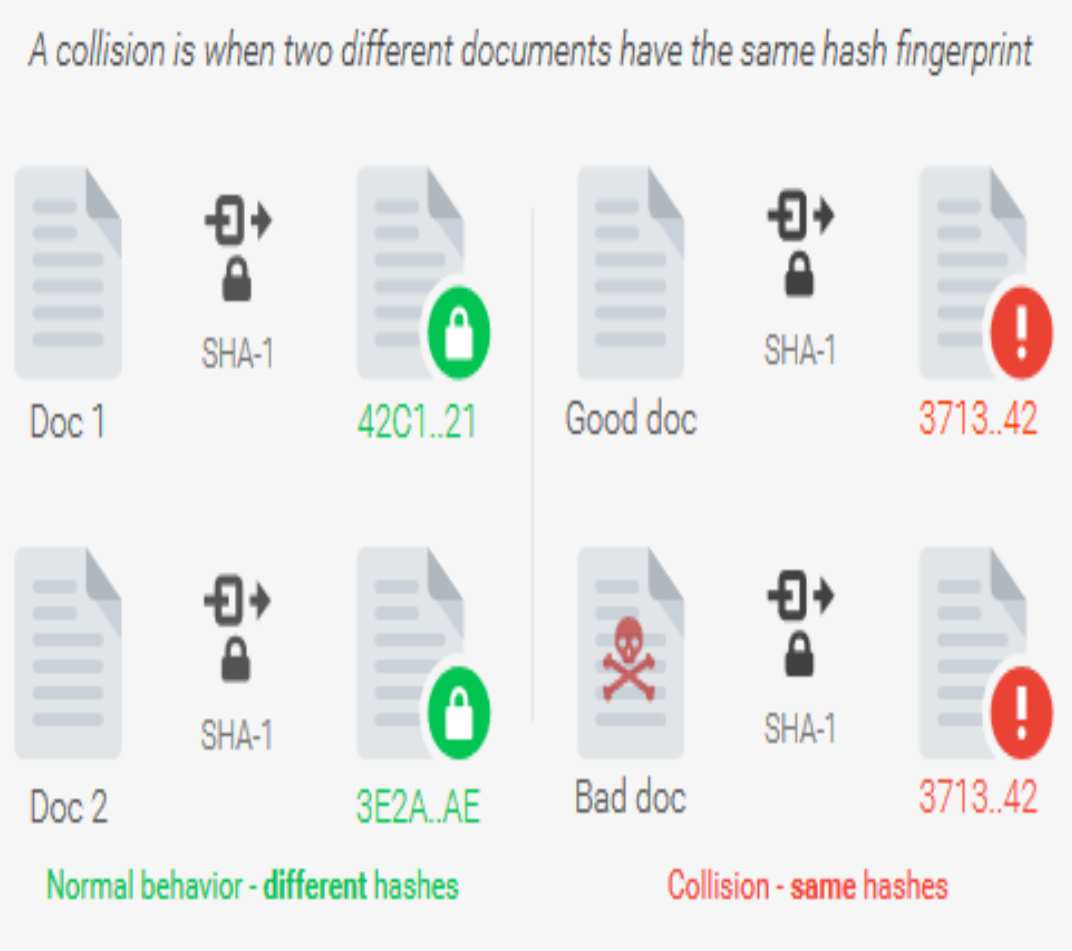
Tabela 11.1 Requisitos para função de hash criptográfica H.

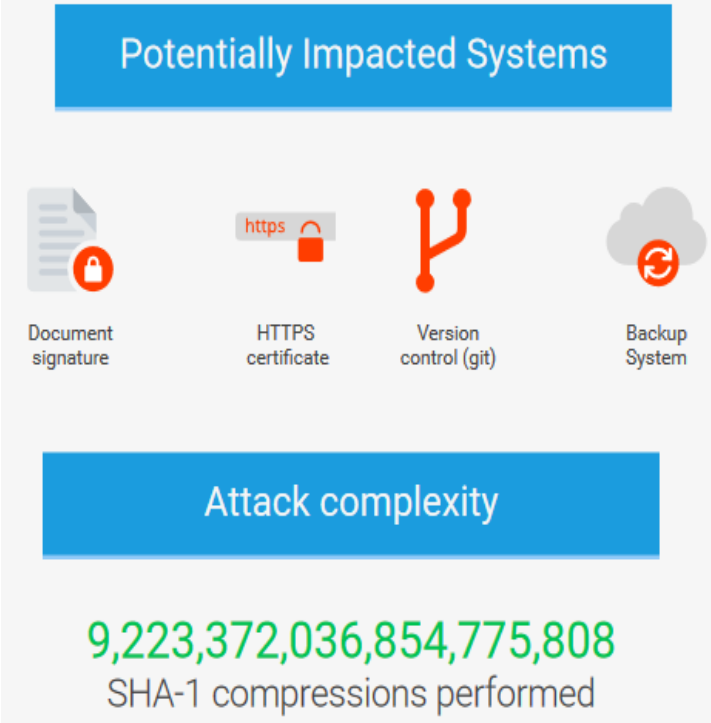
Requisito	Descrição
Tamanho de entrada variável	H pode ser aplicado em um bloco de dados de qualquer tamanho.
Tamanho da saída fixo	H produz uma saída de tamanho fixo.
Eficiência	$H(x)$ é relativamente fácil de calcular para qualquer valor de x informado, através de implementações tanto em hardware quanto 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.

Hashes sem chave

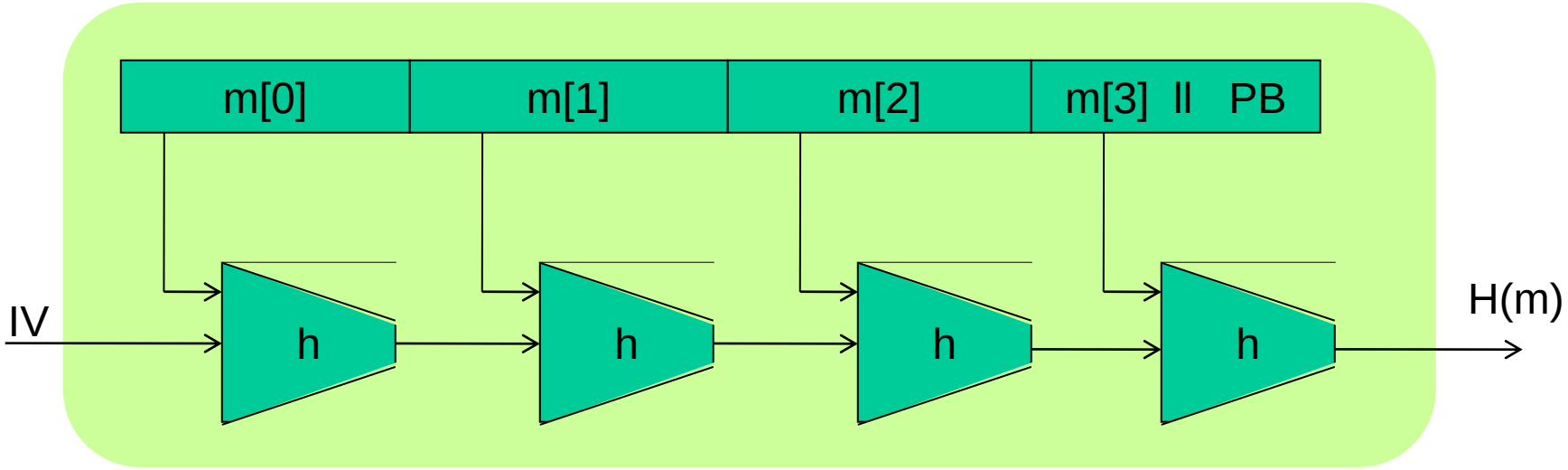
- **Família SHA (Secure Hash Algorithm)**
  - SHA-2 – SHA-256, SHA-512
  - SHA-3 - 224/256/384/512 bits – nome anterior era Keccak; estrutura interna bem diferente do restante da família SHA
- **Não usar, já estão quebrados: MD5 128 bits, SHA-1 160 bits**
- **Crypto++ (**https://www.cryptopp.com/**):**
  - BLAKE2b, BLAKE2s
  - SHA-1, SHA-2, SHA-3
  - Keccack (F1600), SHAKE (128/256), SipHash, Tiger, RIPEMD (128/160/256/320), SM3, WHIRLPOOL

Quebra do SHA-1 na prática: https://shattered.io/





Construção Merkle-Damgard



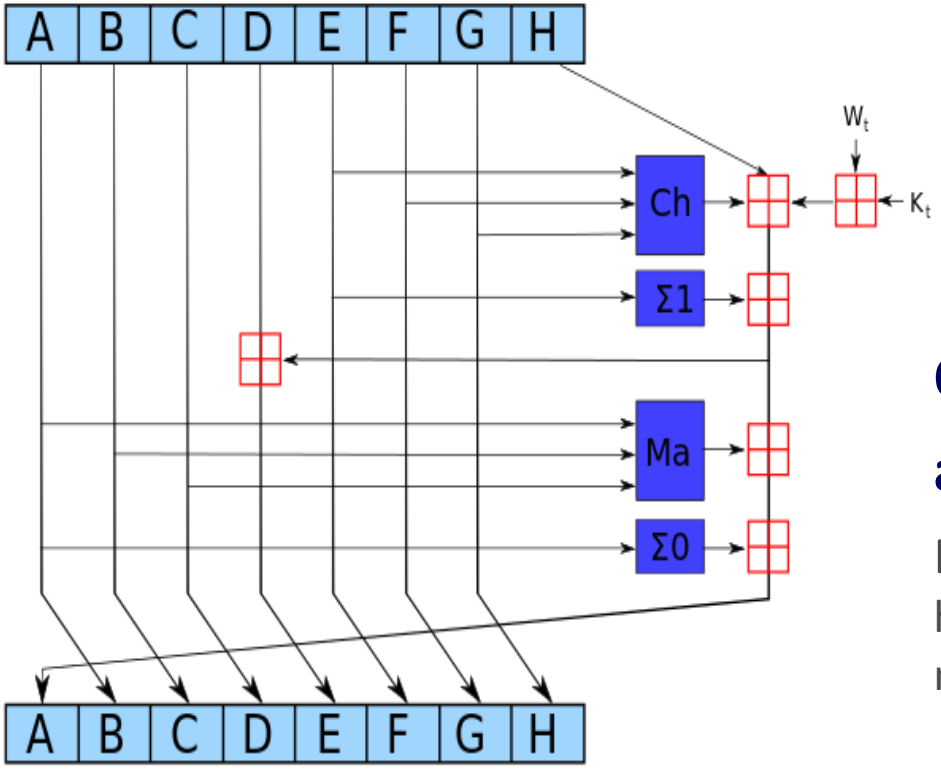
h(t, m[i]): função de compressão **H(m) é o hash de m**

- **Entrada (mensagem) é segmentada em uma série de blocos de tamanho igual. Blocos são processados pela função hash, que tem uma função de compressão no seu "núcleo".**
- **O nome desse projeto iterativo é construção Merkle–Damgard.**
- **PB: padding block** 1000…0 ll msg len (64 bits)

Se não usar PB deve-ser adicionar outro bloco

SHA-256

- Usa Merkle-Damgard
- Função de compressao de Davies-Meyer
- Cada bloco tem 512 bits (16 palavras de 32 bits)



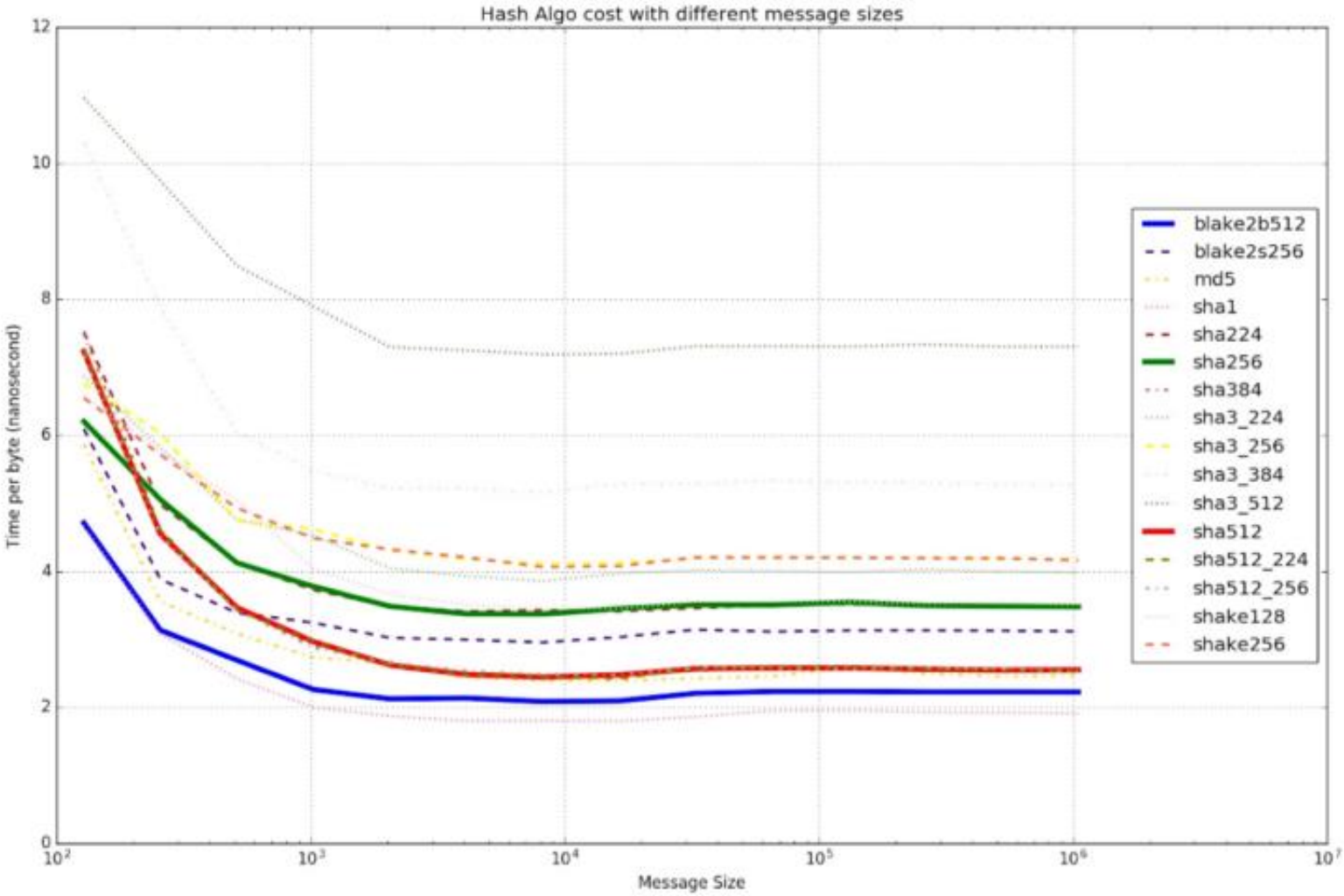
$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$
$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

Quadrado vermelho:

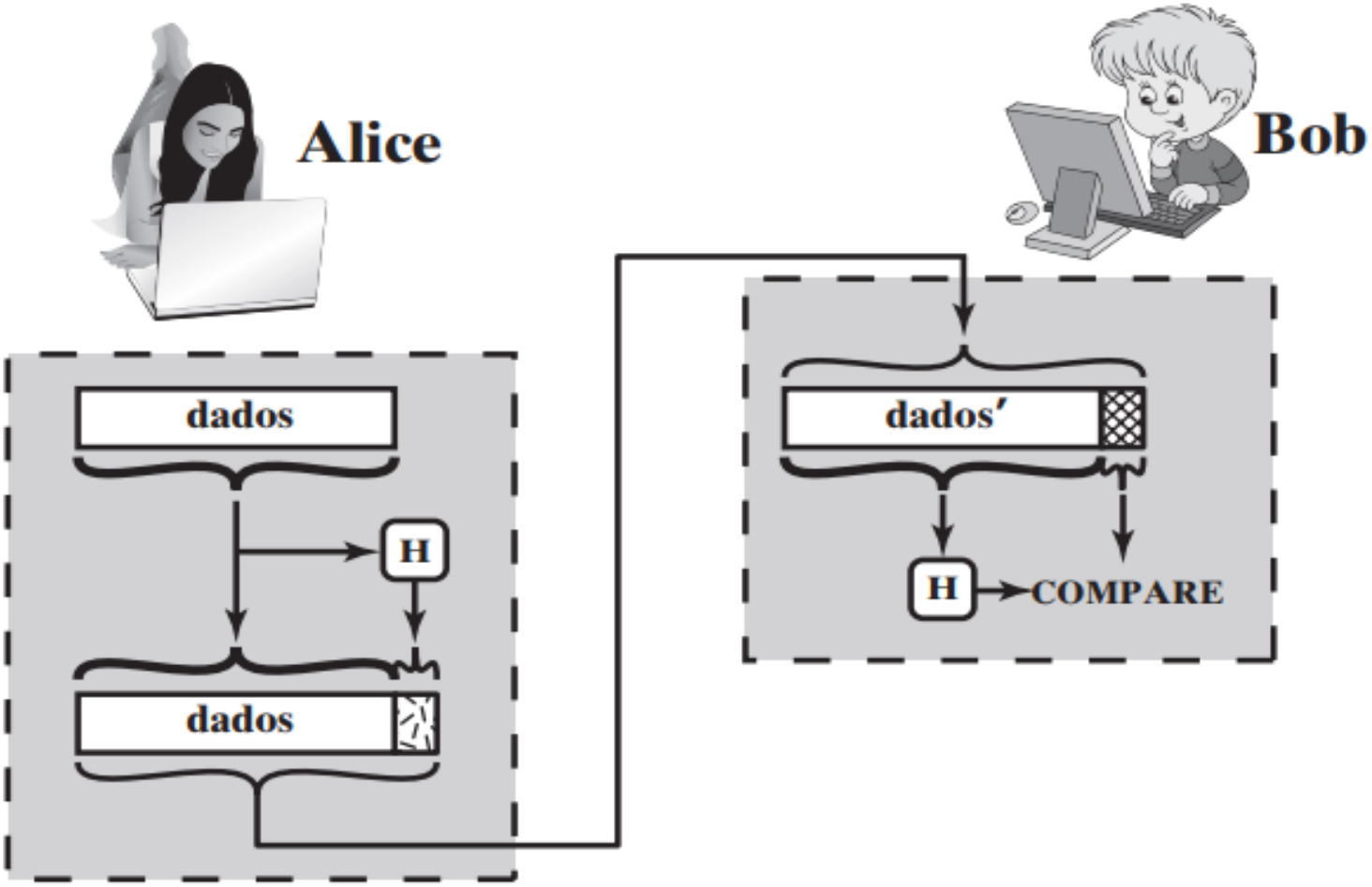
adição módulo 2^{32}

Exemplo com valores:
http://csrc.nist.gov/groups/ST/toolkit/documents/Examples/SHA256.pdf

Desempenho: https://medium.com/logos-network/benchmarking-hash-and-signature-algorithms-6079735ce05

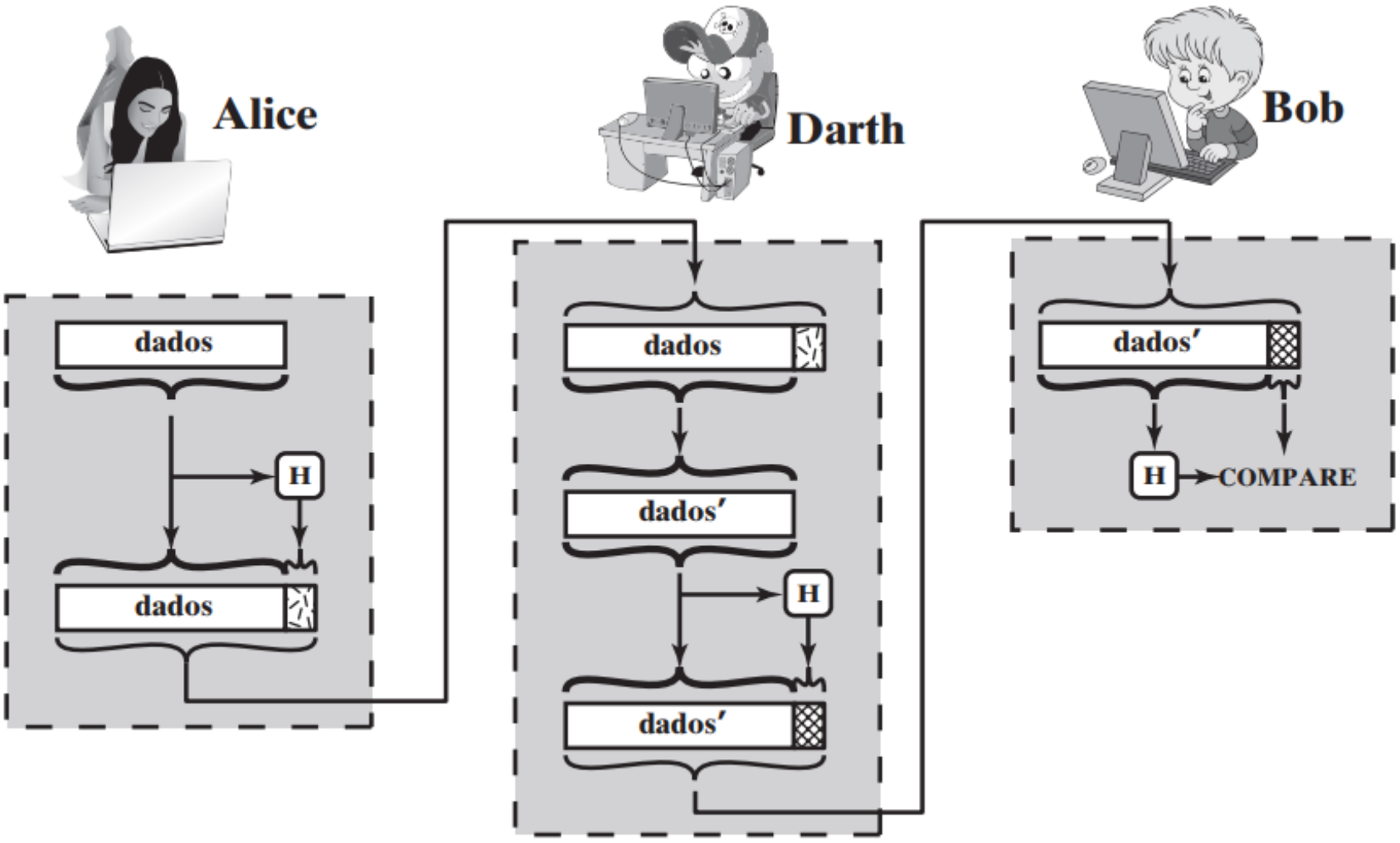


Uso do hash



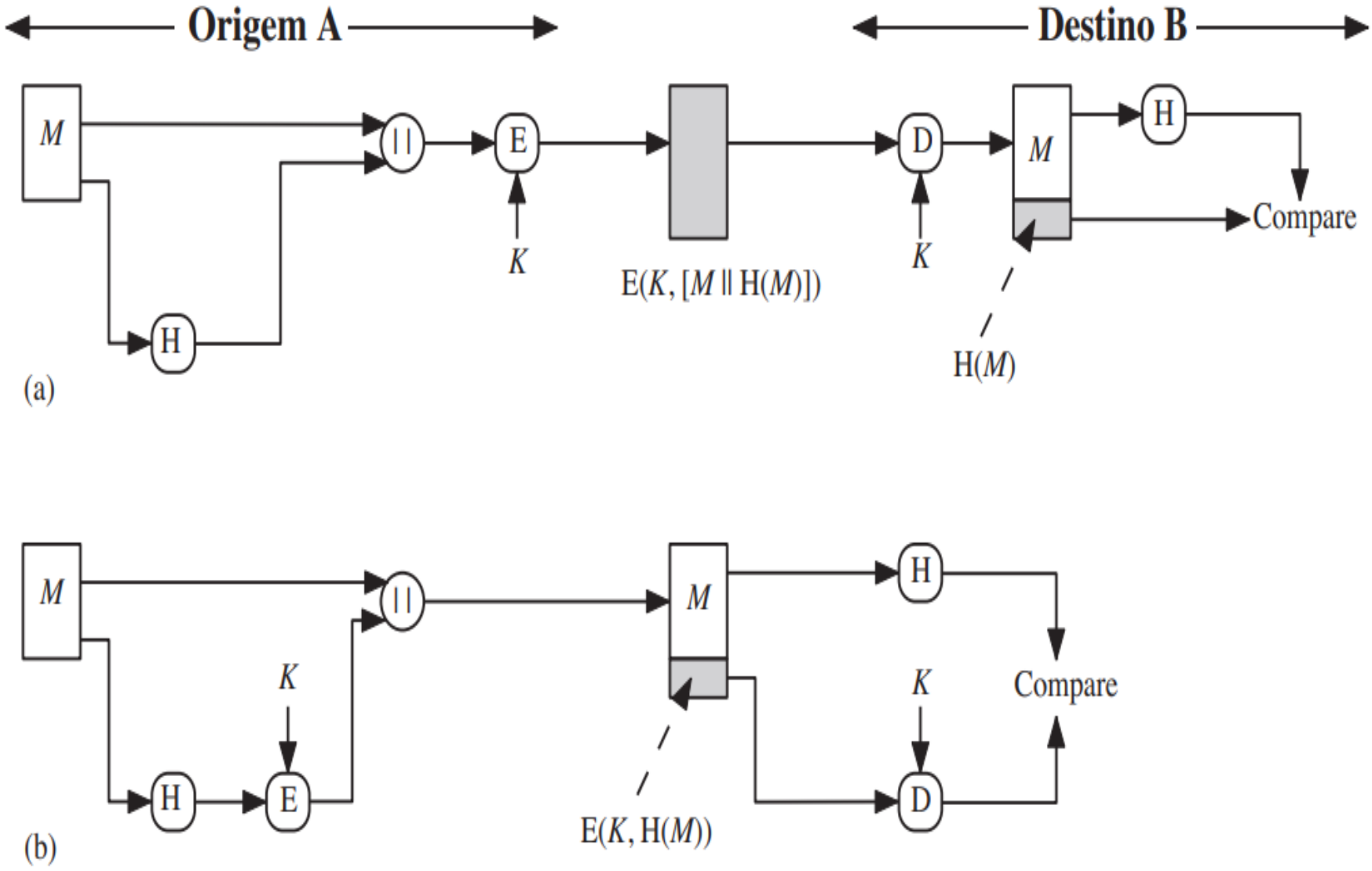
(a) Uso da função de hash para verificar integridade de dados

Ataque do *man-in-the-middle*

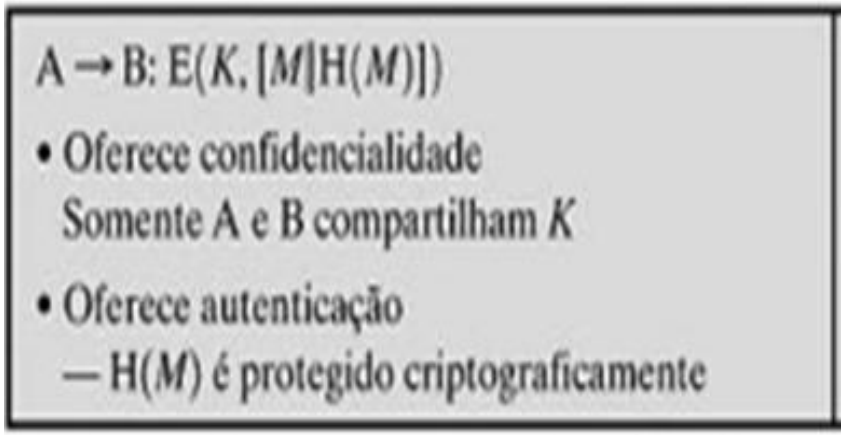


(b) Ataque *man-in-the-middle*

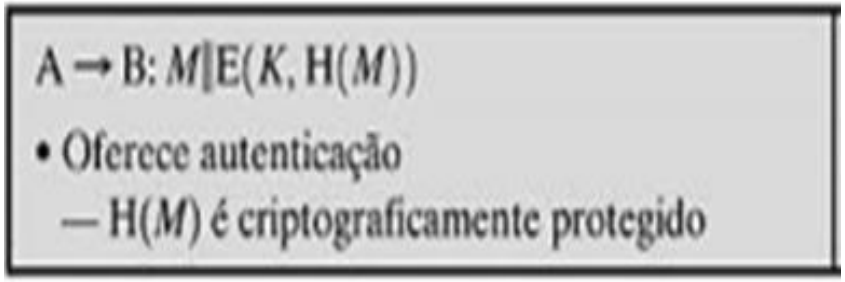
Figura 11.3 Exemplos simplificados do uso de uma função de hash para autenticação de mensagem.



Uso básico do Hash sem chave (Stallings)



(a) Criptografia de mensagem mais código de hash

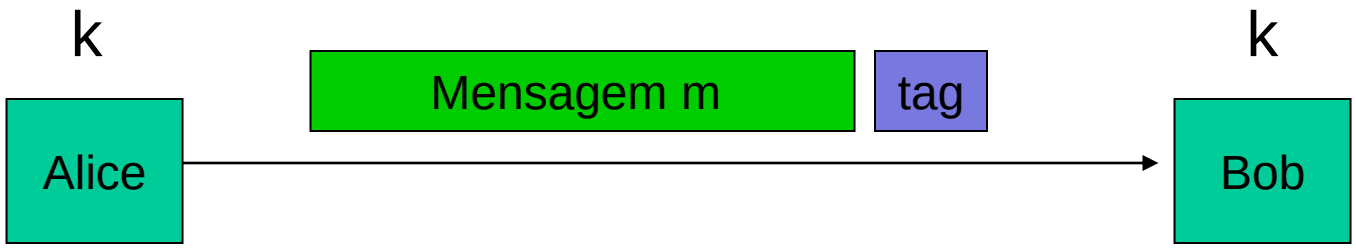


(b) Criptografia do código de hash — chave secreta compartilhada

Hashes com chave – MACs

- **MAC (*Message authentication code*) – autenticador de mensagem. Também chamado de MAC, tag, código autenticador de msg.**
- **Objetivo: fornecer integridade, sem confidencialidade**
- **Pode ser construído com:**
  - Cifras de bloco
  - Funções hash sem chave
- **CMAC e CBC-MAC (ISO/IEC 9797-1), HMAC (ISO/IEC 9797-2)**
  - **CMAC**
  - **CBC-MAC** (bancos – ANSI X9.9, X9.19, FIPS 186-3)
  - **HMAC** (FIPS 198-1, Protocolos da Internet: SSL, IPsec, SSH, …)
- **PMAC – Parallel MAC**

MAC (Message Authentication Code): hash COM CHAVE



Gerar tag:
tag ← S(k, m)

Verificar tag: ?
V(k, m, tag) = `sim'

Def: MAC I = (S,V) definido sobre (K,M,T) um par de algoritmos :

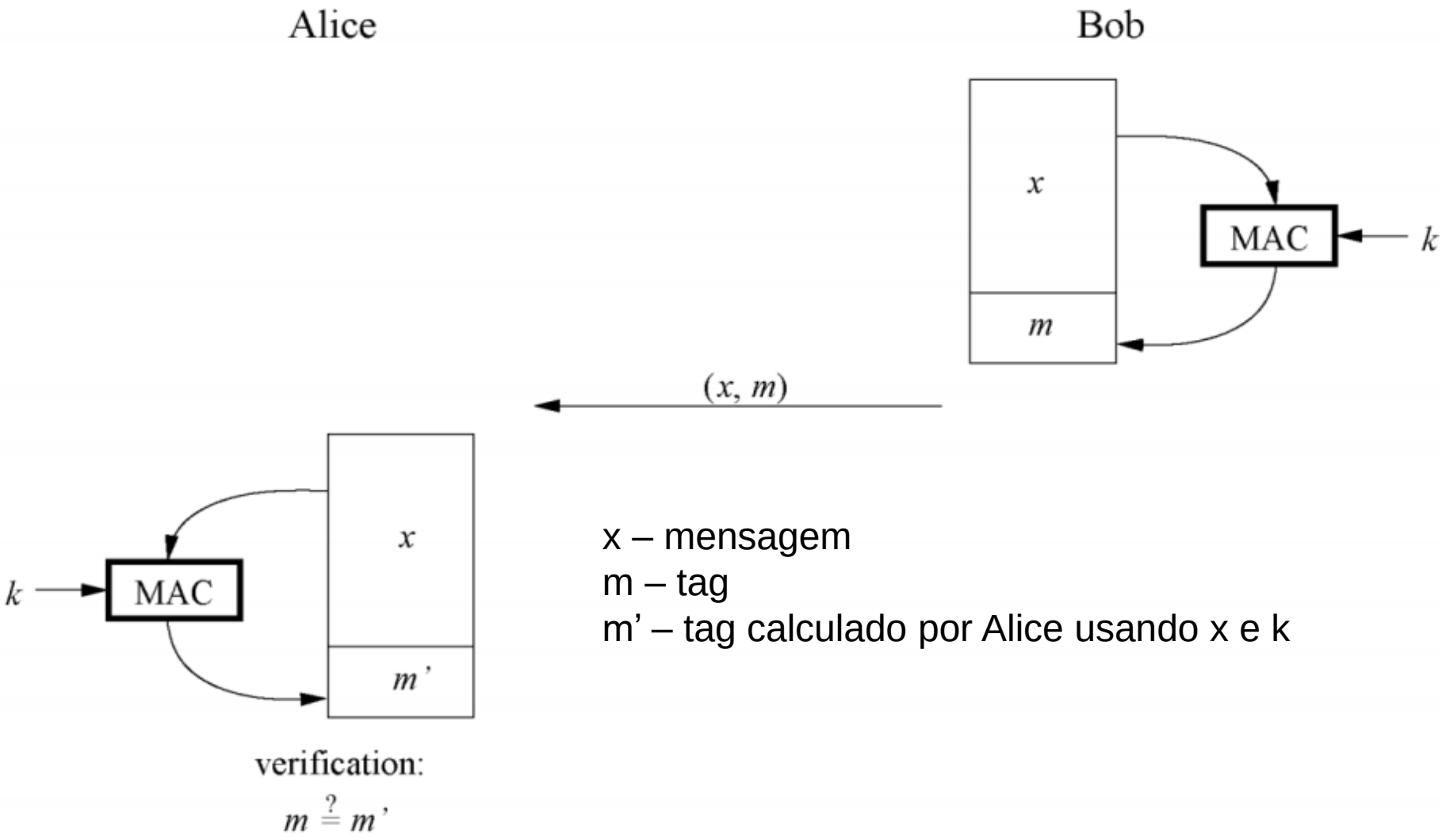
- S(k,m) – saída é t em T
- V(k,m,t) saída é `sim' ou `nao'

MAC é seguro quando:

⇒ um atacante não consegue produzir um tag válido para uma nova msg

nota: checksum sem chave (CRC) é um MAC inseguro !!

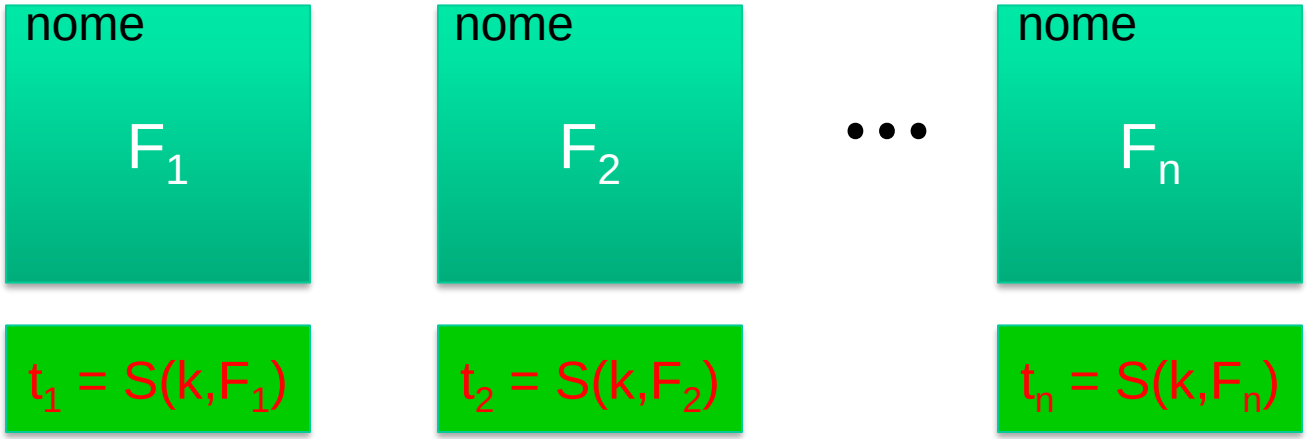
MAC (Message Authentication Code): hash COM CHAVE



x – mensagem
m – tag
m' – tag calculado por Alice usando x e k

Exemplo: proteger sistema de arquivos

Suponha que na instalação o sistema calcule:



k derivada da senha do usuário

Mais tarde um vírus infecta o sistema e modifica os arquivos do sistema

Usuário entra com um sistema limpo e fornece sua senha

- Então: MAC seguro ⇒ todos os arquivos modificados serão detectados

Figura 12.4 Usos básicos do código de autenticação de mensagem (MAC).

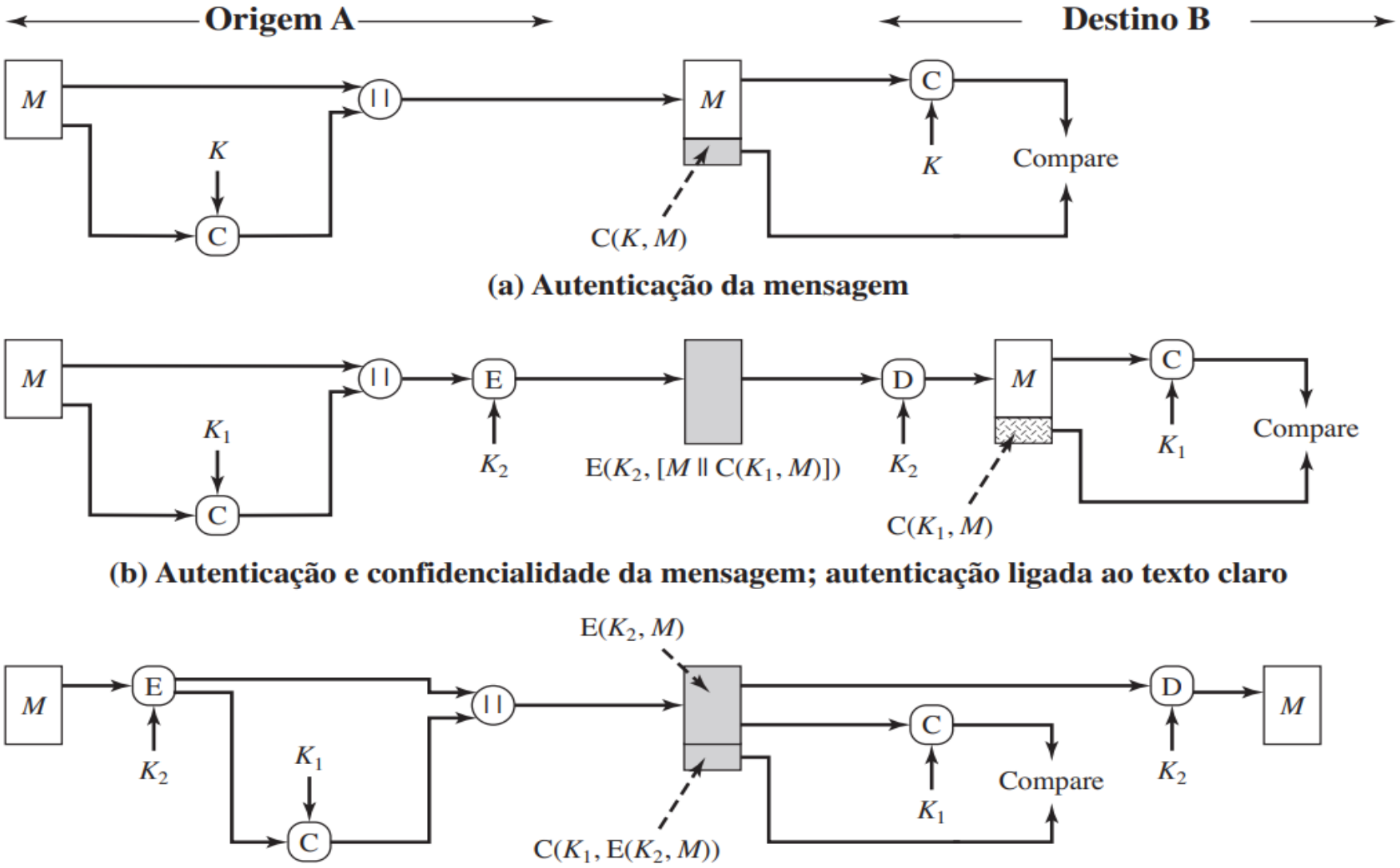


Tabela 11.2 Usos básicos do código de autenticação de mensagens C (ver Figura 12.4

$A \rightarrow B: M \| C(K, M)$

- Oferece autenticação
  — Somente A e B compartilham K

(a) Autenticação da mensagem

$A \rightarrow B: E(K_2, [M \| C(K, M)])$

- Oferece autenticação
  — Somente A e B compartilham K_1
- Oferece confidencialidade
  — Somente A e B compartilham K_2

(b) Autenticação e confidencialidade da mensagem: autenticação ligada ao texto claro

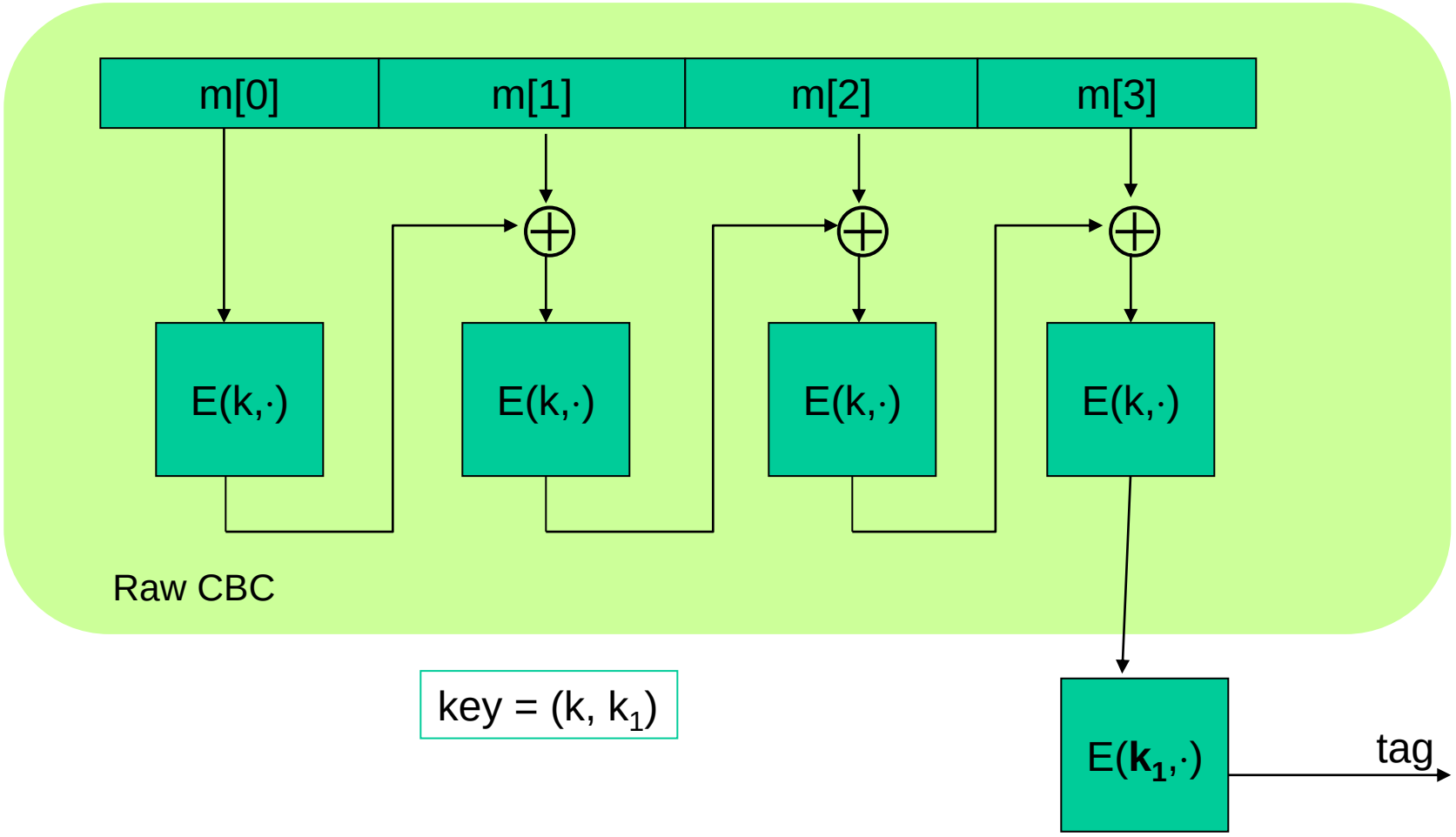
$A \rightarrow B: E(K_2, M) \| C(K_1, E(K_2, M))$

- Oferece autenticação
  — Usando K_1
- Oferece confidencialidade
  — Usando K_2

(c) Autenticação e confidencialidade da mensagem: autenticação ligada ao texto cifrado

Construção 1: ECBC (Encrypt CBC-MAC)

- Construir um MAC a partir de um cifrador de bloco
- Tamanho da mensagem é variável



Construção 1: ECBC (Encrypt CBC-MAC)

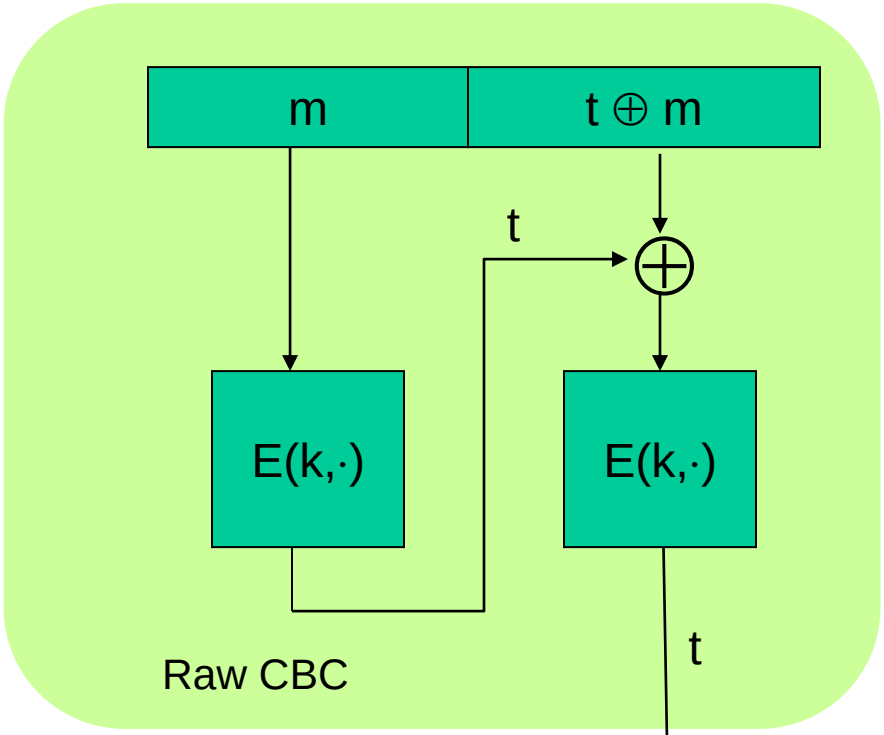
- **Por que o último passo de cifragem do ECBC?**
  - CBC (também conhecido como Raw-CBC) não é um MAC seguro:
    - Dado um tag na mensagem m o atacante pode deduzir o tag de algum mensagem m'
    - Como ?

Atacante faz:

1. Escolhe uma msg qualquer com apenas um bloco $m \in X$
2. Pede um tag para m. Obtém $t = F(k,m)$
3. Anuncia t como um MAC forjado para uma msg de 2-blocos $(m, t \oplus m)$

Verdade: $\text{rawCBC}(k, (m, t \oplus m)) = F(k, F(k,m) \oplus (t \oplus m)) = F(k, t \oplus (t \oplus m)) = t$

Construção 1: ECBC (Encrypt CBC-MAC)



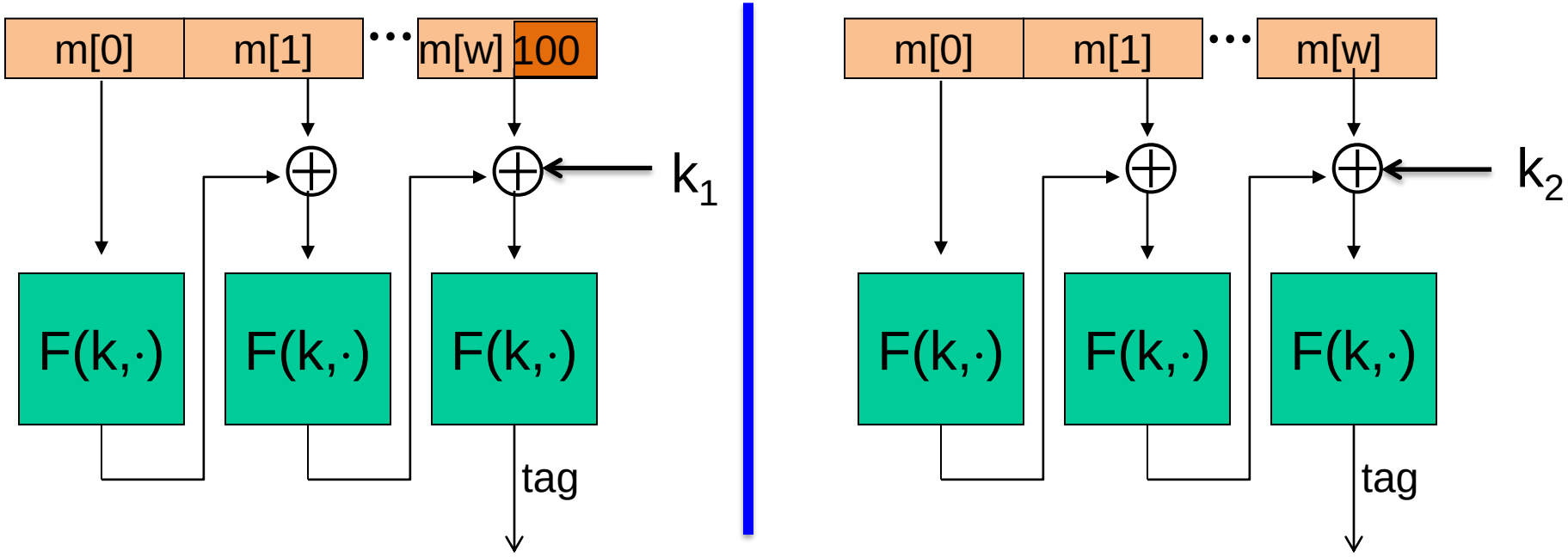
PASSO IMPRESCINDÍVEL:

key = (k)
mensagem de dois blocos = (m, t ⊕ m)

Construção 2: CMAC (Padrão NIST)

Variação do CBC-MAC onde key = (k, k_1, k_2)

- k_1, k_2 derivadas de k
- Sem cifragem final mas faz um xor final com chave
- Sem bloco dummy (ambiguidade resolvida pelo uso de k_1 ou k_2)



Construção 3: HMAC (Hash-MAC)

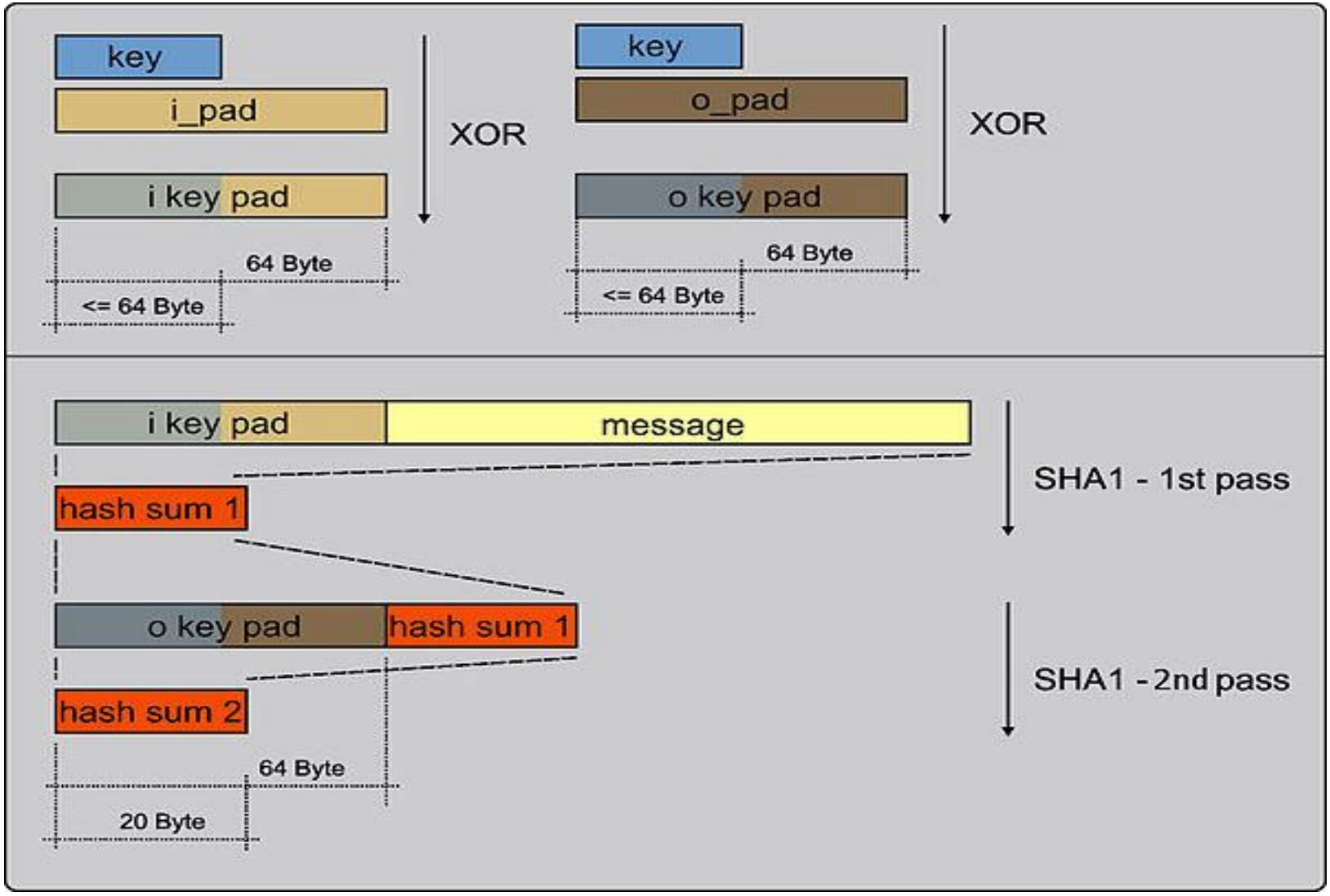
- Amplamente usado na Internet
- Construído com funções hash

Método padronizado: HMAC

$$S(k, m) = H(k \oplus opad \,\|\, H(k \oplus ipad \,\|\, m))$$

- *H* é o hash criptográfico
- k é a chave secreta (padded com zero até tamanho do bloco)
- ‖ significa concatenação, ⊕ significa ou exclusivo (XOR)
- *opad* é o padding (0x5c5c5c…5c5c, constante hexa de tamanho de um bloco)
- *ipad* é o padding (0x363636…3636, constante hexa de tamanho de um bloco)

Geração HMAC SHA-1



Criptografia Autenticada

- ***Authenticated Encryption (AE): combinação da criptografia simétrica com MAC***
- **Objetivo: garantir confidencialidade, INTEGRIDADE e autenticidade (da origem dos dados)**
- **Se a msg precisa integridade mas não precisa confidencialidade:**

  → **usar um MAC**

- **Se a msg precisa de ambos – CONFIDENCIALIDADE e INTEGRIDADE:**

  → **usar modos de criptografia autenticada**

Modos de Criptografia Autenticada

- **Encrypt-and-Mac (E&M):** Usado no SSH. Não é "strongly unforgeable" mas pode se tornar melhor com algumas modificações.

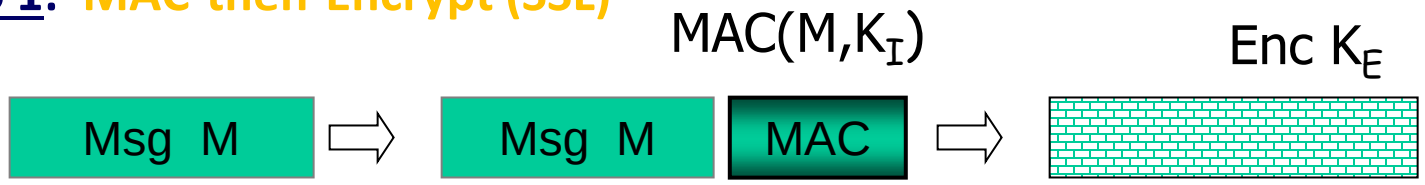
- **Mac-then-Encrypt (MtE):** Usado no SSL/TLS. Não é "strongly unforgeable", mas a implementação de SSL/TLS é strongly unforgeable de acordo com Krawczyk.

- **Encrypt-then-Mac (EtM):** O método padrão ISO/IEC 19772:2009. É o método com MAIS ALTA DEFINIÇÃO DE SEGURANÇA. Usado no Ipsec. É "strongly unforgeable".

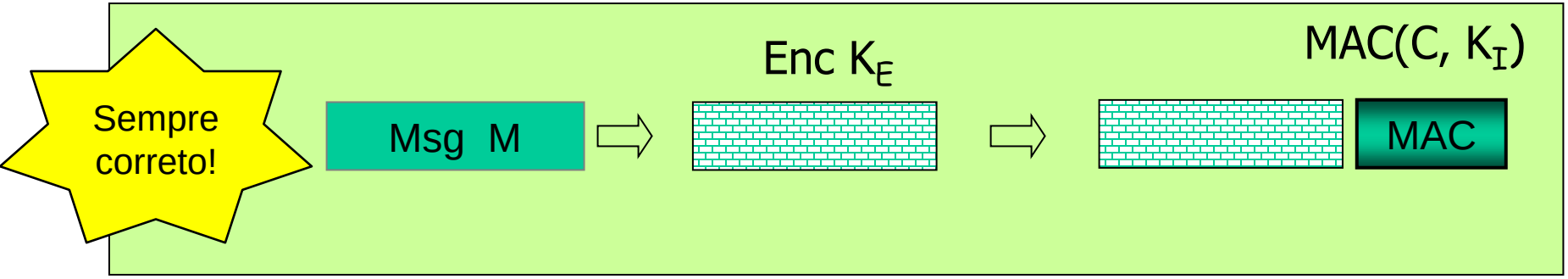
Combinando MAC e Cifragem

Chave de cifragem K_E Chave MAC = K_I

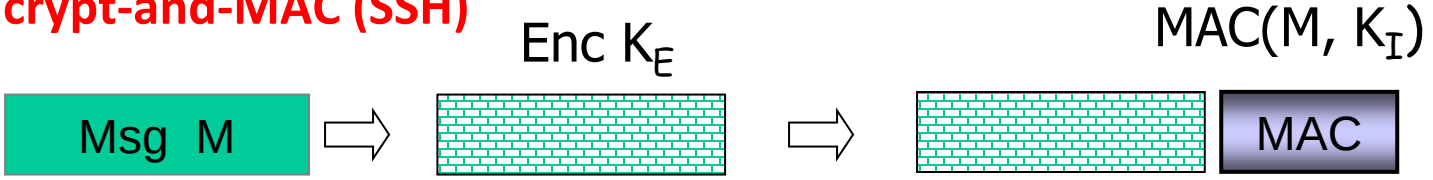
Opção 1: MAC-then-Encrypt (SSL)



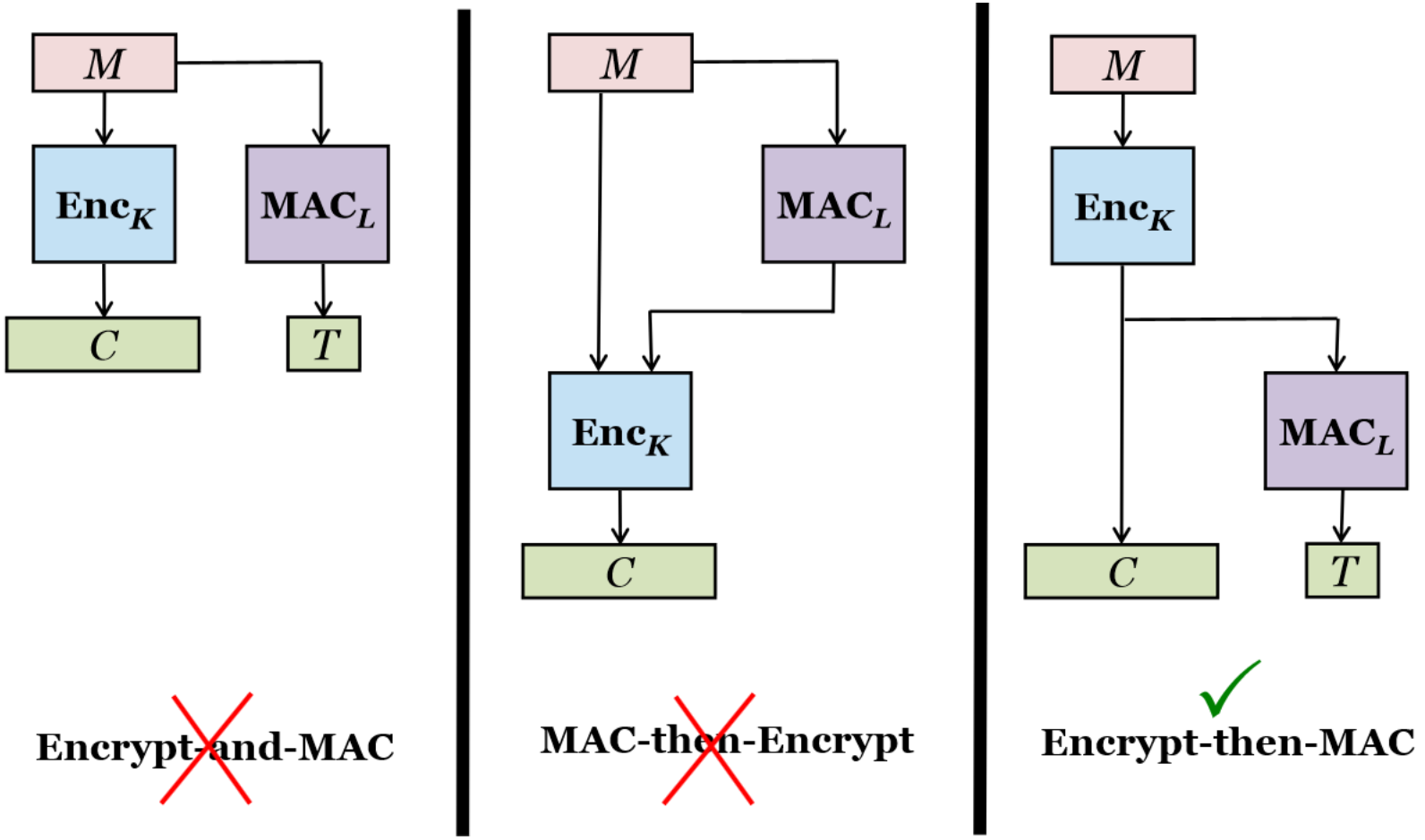
Opção 2: Encrypt-then-MAC (IPsec)



Opção 3: Encrypt-and-MAC (SSH)



Combinando MAC e Cifragem



Workshop on Real-World Cryptography , Stanford University, Jan, 9-11, 2013
https://crypto.stanford.edu/RealWorldCrypto/slides/phil.pdf

Comentários

1. **Encrypt-then-MAC: sempre fornece AE**

2. **MAC-then-encrypt: pode ser inseguro em ataques de CCA (chosen-ciphertext attacks)**

   entretanto: quando se usa modos rand-CTR ou rand-CBC esse modo fornece AE

   para o modo rand-CTR, one-time MAC é suficiente

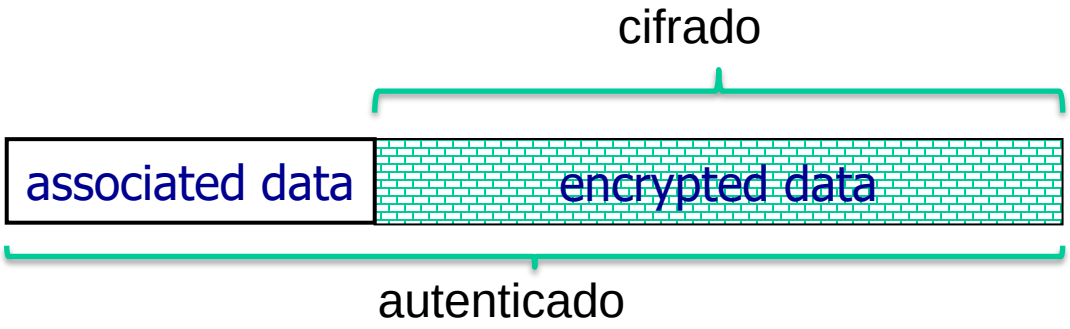
Padrões

- **GCM: cifragem em modo CTR e MAC CW-MAC**

  (acelerado com a instruções em hardware da Intel PCLMULQDQ)

- **CCM: CBC-MAC então cifragem em modo CTR (802.11i)**
- **EAX: cifragem modo CTR então CMAC**

Todos suportam AEAD: (auth. enc. with associated data).
Todos baseados em nonce.



http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html#01

GCM

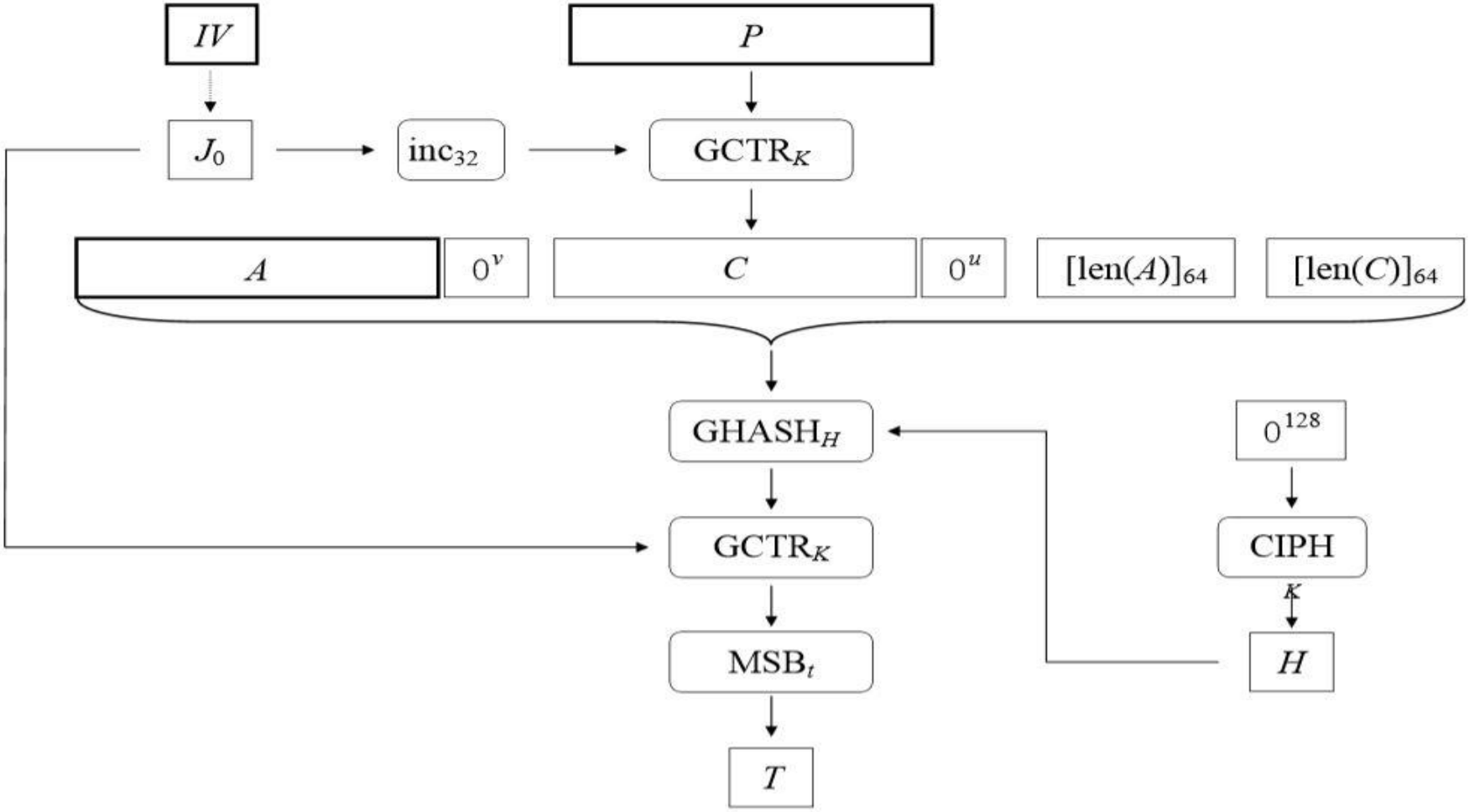
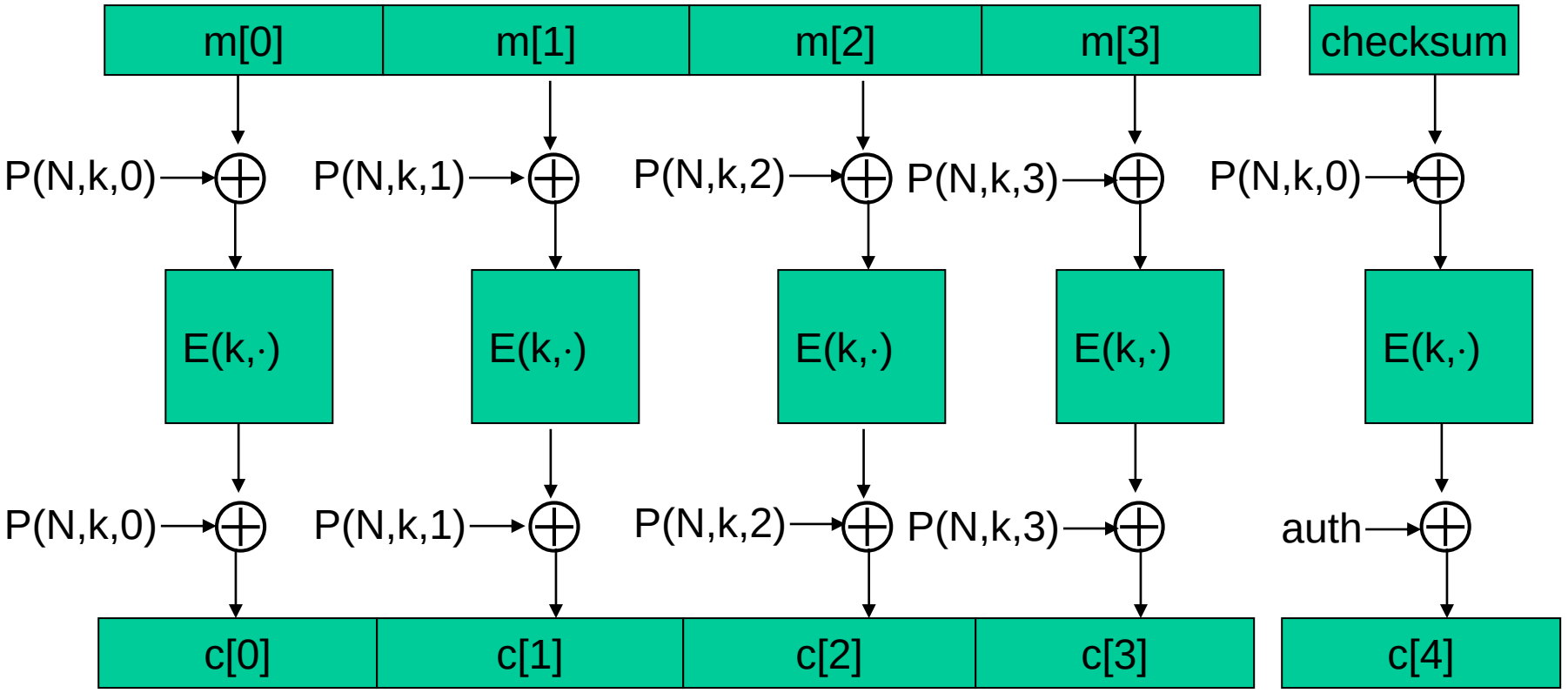


Figure 3: $\text{GCM-AE}_K(IV, P, A) = (C, T)$.

OCB (Offset codebook mode)

Forma mais eficiente de criptografia autenticada (AE)



Desempenho: Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

Cifra	tamanho código	Velocidade (MiB/s)		
AES/GCM	**largo****	**108**	AES/CTR	139
AES/CCM	**menor**	**61**	AES/CBC	109
AES/EAX	**menor**	**61**		
			AES/CMAC	109
AES/OCB		**129***	HMAC/SHA1	147

* Extrapolado para resultados de Ted Kravitz ** máquina não-Intel