CiberSegurança

MEET, MEIC, MEIM

Integridade e autenticação (funções Hash)

2021–2022

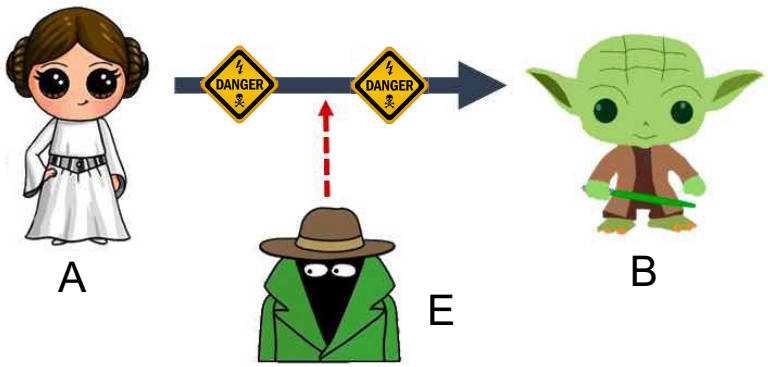
Sistemas de cifra e adversários ativos

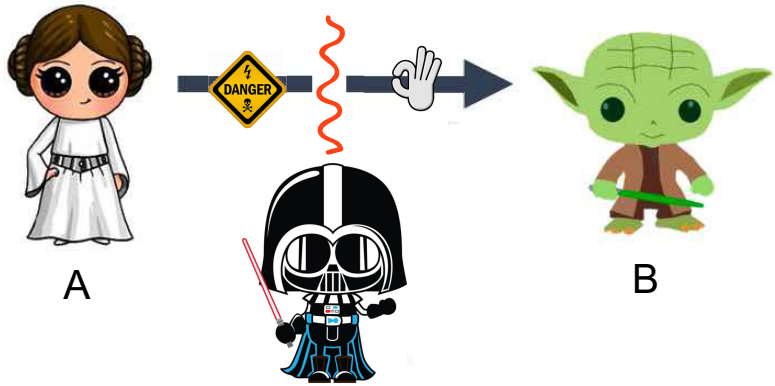
As ferramentas criptográficas apresentadas até agora (cifras de blocos, *stream*, RSA e ElGamal, Protocolo DH ou ECDH) estão orientadas principalmente a garantir a *confidencialidade* da informação, isto é, a manter o conteúdo da informação secreto para todos excepto para o destinatário da mesma.

Este tipo de segurança pressupõe um **adversário passivo**, isto é um adversário que pode eventualmente monitorizar o canal de comunicação mas não interfere na informação transmitida.

As ferramentas criptográficas anteriores não fornecem segurança na presença de um **adversário ativo**.

Adversário passivo versus adversário ativo





Integridade, autenticação e não repúdio

Recorde-se que além da confidencialidade, os outros objetivos da criptografia são:

- a **integridade da informação** (*data integrity*), isto é, que não há alteração da informação por entidades não autorizadas;
- a **autenticação** (*authentication*) das entidades que comunicam entre si e da informação (origem, conteúdo, data de envio ...);
- o **não repúdio** (*non-repudiation*), isto é, assegurar que as entidades participantes não podem negar a autoria das suas ações ou compromissos.

As primitivas criptográficas mais relevantes nas questões relacionadas com a integridade da informação, a autenticação e o não repúdio são as chamadas **funções de dispersão** ou funções *hash* e as **assinaturas digitais**.

Funções *hash*

⇝ Uma função *hash* aplica uma sequência de bits de comprimento arbitrário em sequências de comprimento fixo n, chamadas **valor de dispersão** (*hash code*, *message digest*).

⇝ O objetivo das funções *hash* é utilizar o *hash code* como uma imagem representativa (uma *impressão digital* da mensagem) de uma dada sequência de entrada.

⇝ Em criptografia são usadas funções *hash* de tipo *one-way* ou funções *hash* criptográficas.

Funções *hash* criptográficas

Uma função h que transforma sequências x de bits de comprimento arbitrário em sequências y de bits de comprimento fixo

$$y = h(x)$$

diz-se que é uma função de *hash* **criptográfica** se:

1. (*preimage resistant, one-way*), é computacionalmente fácil determinar y a partir de h e computacionalmente intratável encontrar um valor x a partir do y;
2. (*collision resistant, strong collision resistant*), é computacionalmente intratável encontrar dois valores distintos x,x' verificando que $h(x) = h(x')$;

   $\rightsquigarrow$ x, x' são quaisquer

Tipos de funções *hash* criptográficas

As funções *hash* classificam-se em:

- Funções *hash* sem chave (*unkeyed hash functions*): funções de dispersão cujo único valor de entrada a informação a ser tratada:

  códigos de deteção de modificações (MDC)

- Funções *hash* com chave (*keyed hash functions*): funções de dispersão que têm como valores de entrada a informação a ser tratada e uma dada chave secreta:

  códigos de autenticação de mensagens (MAC)

Funções de compressão

As funções *hash* são construídas normalmente a partir de **funções de compressão**, isto é, funções que transformam mensagens de tamanho fixo s-bits em sequências de tamanho fixo n-bits, com $s > n$.

As operações mais frequentes usadas nas funções de compressão (e nas funções *hash*) são:

- $\oplus$ *bitwise* XOR;
- $\vee$ *bitwise* OR;
- $\wedge$ *bitwise* AND;
- $\neg$ *bitwise* complemento;
- $\boxplus_n$ adição em 2^n;
- R^n rotação à direita de n bits;
- S^n *shift* à direita de n bits;
- $||$ concatenação de blocos de bits.

Exemplo 1

Função de compressão de 8-bits a 4-bits

Seja f uma função que transforma os blocos m de 8 bits em blocos de 4 bits do seguinte modo:

$$f(m) = R^2(x) \boxplus_4 y$$

com x, y os sub-blocos de 4 bits do array inicial m, isto é $m = x||y$.

Por exemplo, dado $m = 0111\ 1101$, consideramos $x = 0111$, $y = 1101$ e então

$$R^2(0111) = 1101$$

donde

$$f(m) = f(0111\ 1101) = 1101 \boxplus_4 1101 = 1010 \qquad (13 + 13 = 10 \bmod 2^4).$$

Exemplo 2

Função de compressão de 12-bits a 4-bits

Seja T a função de compressão de 12-bits a 4 bits definida por

$$T(m_1||m_2||m_3) = \left(S^2(m_2) \oplus S^1(m_3)\right) \boxplus_4 \left((m_1 \wedge m_2) \oplus (\ m_1 \wedge m_3)\right)$$

com m_i os sub-blocos (*palavras*) de 4-bits.

Por exemplo, considerando $m = m_1||m_2||m_3 = 1101\,1111\,0001$, tem-se

$$S^2(m_2) = S^2(1111) = 0011, \quad S^1(m_3) = S^1(1001) = 0100$$

$$(m_1 \wedge m_2) \oplus (\ m_1 \wedge m_3) = (1101 \wedge 1111) \oplus (1101 \wedge 1001) = 0100$$

donde

$$T(0101\,1111\,0001) = (0011 \oplus 0100) \boxplus_4 0100 = 0111 \boxplus_4 0100 \stackrel{*}{=} 1011$$

Construção de funções *hash*

A partir de uma função de compressão f, que transforma blocos de s-bits em blocos de n-bits com $s > n$, é possível construir funções *hash* que operam em blocos de bits de comprimento arbitrário, usando técnicas parecidas aos modos de operação em cifras por blocos.

A construção mais usada na definição de funções *hash* é a **construção de Merkle-Damgård.**

Trata-se de uma construção algorítmica que permite, a partir de uma função de compressão f resistente às colisões, construir uma função *hash* também resistente às colisões, com *input* blocos de bits de tamanho arbitrário.

Construção de Merkle-Damgård

Suponha-se que f é uma função de compressão, resistente a colisões, que transforma sequências de s bits em sequências de n bits, com $s > n$ e define-se $\ell = s - n$.

A construção de Merkle-Damgård para obter o valor *hash* de m a partir de f consta de três fases:

1. realização de um *OneZeroesPadding* da mensagem m até obter uma mensagem cujo comprimento é mútiplo de ℓ, isto é, $m = m_1 m_2 \cdots m_k$, como m_i bloco de comprimento ℓ-bits;
2. adição de um bloco extra, chamado **bloco de comprimento**, no qual se coloca a representação binária do comprimento de m à direita ($m < 2^\ell$);
3. um processo iterativo nos blocos m_i, que usa a função f e um bloco inicial H^0 com comprimento n,para obter uma sequência de blocos com n-bits, $H^1, \ldots, H^{k+1}$

   H^{k+1} é o *hash* da mensagem.

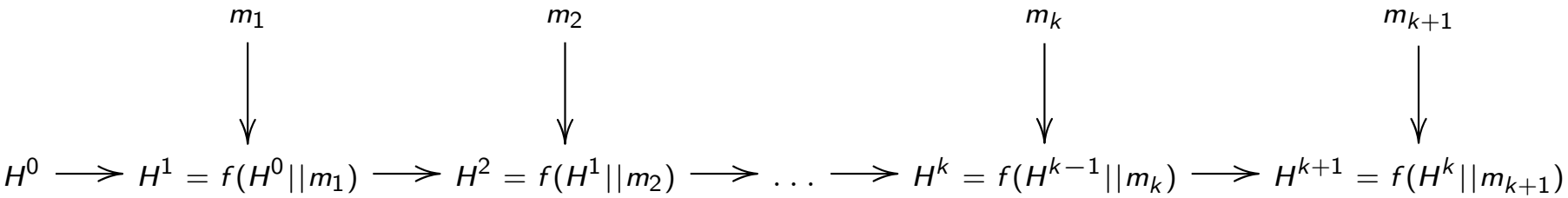
Processo iterativo na construção de Merkle-Damgård

Considera-se um bloco inicial fixo, H^0, com comprimento n. Para cada $i = 1$ até $k + 1$, define-se

$$H^i = f(H^{i-1} \| m_i)$$

com $\|$ a concatenação de bits.

A valor *hash* da mensagem m será H^{k+1}. Os valores H^0, H^1, H^2 ... costumam chamar-se **valores** *hash* **internos** ou **intermédios** (*internal state of the hash function*).



Exemplo: construção MD

Seja f a função de compressão de 12 bits a 4 bits definida por

$$f(x||y||z) = S^2(x) \boxplus_4 y \boxplus_4 (x \oplus z)$$

com x, y, z os sub-blocos de 4 bits do *array* inicial e o **hash inicial** $H^0 = 1000$.

Considere-se a mensagem $m = 001101011100111$1. Como $\ell = 12 - 4 = 8$, os blocos m_i usados na construção MD devem ter comprimento 8.
O *padding OneAndZeroes* para 8-bits da mensagem é a sequência

$$00110101\ 11001111\ 10000000$$

O comprimento da mensagem original é 8, em binário, 00001000, que concatenamos com a sequência anterior:

$$00110101\ 11001111\ 10000000\ 00001000$$

Exemplo: construção MD

Os blocos para processo iterativo são então :

$$m_1 = 00110101, \quad m_2 = 11001111, \quad m_3 = 10000000, \quad m_4 = 00001000$$

Iniciámos o processo com o *hash* inicial H^0 e m_1 e iteramos até m_4:

$$\begin{aligned} H^1 &= f(H^0||m_1) = f(1000\,00110101) = S^2(1000) \boxplus_4 0011 \boxplus_4 (1000 \oplus 0101) \\ &= 0010 \boxplus_4 0011 \boxplus_4 1101 = 0010 \end{aligned}$$

$$\begin{aligned} H^2 &= f(H^1||m_2) = f(0010\,11001111) = S^2(0010) \boxplus_4 1100 \boxplus_4 (0010 \oplus 1111) \\ &= 0000 \boxplus_4 1100 \boxplus_4 1101 = 1001 \end{aligned}$$

$$\begin{aligned} H^3 &= f(H^2||m_3) = f(1001\,10000000) = S^2(1001) \boxplus_4 1000 \boxplus_4 (1001 \oplus 0000) \\ &= 0010 \boxplus_4 1000 \boxplus_4 1001 = 0011 \end{aligned}$$

$$\begin{aligned} H^4 &= f(H^3||m_4) = f(0011\,00001000) = S^2(0011) \boxplus_4 0000 \boxplus_4 (0011 \oplus 1000) \\ &= 0000 \boxplus_4 0000 \boxplus_4 1101 = 1101 \end{aligned}$$

O *hash* da mensagem inicial, usando a construção de Merkle-Damgård e o *hash* inicial de 1000, é

$$H^4 = 1101$$

A família SHA

As **SHA**-*Secure Hash Algorithm* são um conjunto de funções *hash* criptográficas publicadas pelo NIST (National Institute of Standards and Technology) como FIPS (Federal Information Processing Standars) que incluem:

- A SHA-1, função *hash* de 160-bits, projetada em 1995 pela NSA (National Security Agency), em desuso atualmente;
- A família SHA-2, também projetada pela NSA, é composta por seis funções *hash* identificadas pelo comprimento em bits do *hash code*: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 e SHA-512/256;
- A família SHA-3, publicada em 2015 após uma competição pública, está composta por funções com o mesmo comprimento em bits do *hash* da família SHA-2: SHA-3-224, SHA-3-256, SHA-3-384, e SHA-3-512, mas com uma estrutura interna diferente.

A família SHA-2

As funções *hash* da família SHA-2 estão construídas usando a estrutura de Merkle-Damgard, a partir de uma função de compressão criptográfica definida por sua vez usando a estrutura de Davies-Meyer a partir de uma cifra por blocos.

A função SHA-256 é, provavelmente, a mais famosa de todas as funções hash criptográficas por ter sido usada na tecnologia *blockchain* (protocolo *Bitcoin* original do S. Nakamoto).

Cálculo do *hash* de uma mensagem M de comprimento arbitrário r usando SHA-256:

1. m é preenchida usando *OneAndZeroes padding*, de modo a obter um comprimento que seja múltiplo de 512 bits mas reservando os últimos 64 bits para a representação binária de r;
2. A mensagem é subdividida em blocos $m = m_1||m_2||\cdots||m_N$ cada um com comprimento 512-bits;
3. Os blocos são processados sucessivamente, iniciando com um *hash value* H^0 fixo e gerando os valores *hash* intermédios H^i (com 256-bits) através de uma cifra por blocos $\mathcal{C}$ específica do SHA-2:

$$H^i = H^{i-1} \boxplus_{32} \mathcal{C}_{m_i}(H^{i-1})$$

H^N é o *hash value* de m.

O método de compressão realizado em cada bloco usando a cifra por blocos $\mathcal{C}$:

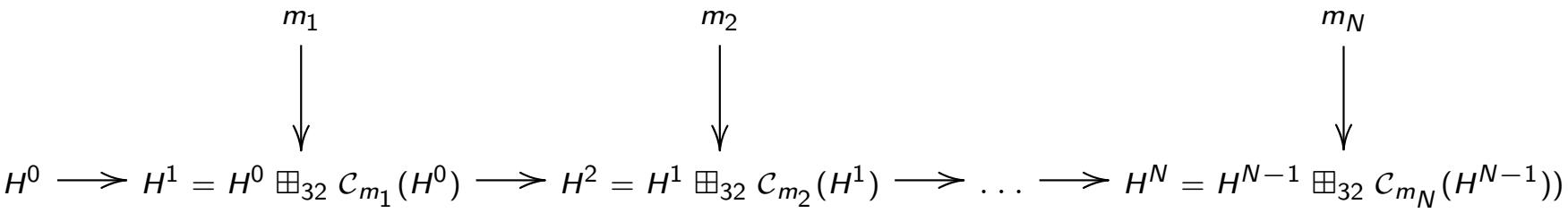
$$H^i = H^{i-1} \boxplus_{32} \mathcal{C}_{m_i}(H^{i-1})$$

é o chamado *esquema de Davies-Meyer*.

Este esquema permite, a partir de uma cifra por blocos, obter uma função de compressão e usar a construção de Merkle-Dagmärd para obter uma função *hash*:

$$H^0 \longrightarrow H^1 = H^0 \boxplus_{32} \mathcal{C}_{m_1}(H^0) \longrightarrow H^2 = H^1 \boxplus_{32} \mathcal{C}_{m_2}(H^1) \longrightarrow \ldots \longrightarrow H^N = H^{N-1} \boxplus_{32} \mathcal{C}_{m_N}(H^{N-1}))$$

Estrutura geral da SHA-256



- $m = m_1||m_2||\cdots||m_N$, os blocos m_i têm comprimento 512-bits;
- $\mathcal{C}$ é uma cifra por blocos de 256-bits com uma chave m_i de comprimento 512;
- os *hash* H^i tem comprimento 256-bits;
- a operação $\boxplus_{32}$, ou seja, a adição módulo 2^{32} deve ser aplicada por blocos (*palavras*) de 2^{32} bits;

  Dados dois *array* de 256 bits em hexadecimal, a adição é feita por blocos de 4 bytes em $\mathbf{Z}_{2^{32}}$, por exemplo:

a4 fb 58 ab	9e 76 dc f8	d9 b2 63 8c	78 dc 2b a0	8b 72 44 71	70 60 46 84	d3 f1 ad fb	98 42 f9 50
00 00 00 01	00 00 00 00	00 00 00 01	00 00 00 01	00 00 00 01	ff ff ff ff	ff ff ff ff	ff ff ff ff

- o valor *hash* inicial H^0 está formado por 8 palavras concatenadas de 32-bits:

$$H^0 = \quad H_1^0 \quad H_2^0 \quad H_3^0 \quad H_4^0 \quad H_5^0 \quad H_6^0 \quad H_7^0 \quad H_8^0$$

$$= 6A09E667\ \ BB67AE85\ \ 3C6EF37\ \ A54FF53A\ \ 510E527F\ \ 9B05688C\ \ 1F83D9AB\ \ 5BE0CD19$$

$\rightsquigarrow$ partes fracionárias das raízes quadradas dos primeiros oito primos

A cifra $\mathcal{C}$ interna no SHA-256

A cifra interna do SHA-256 é uma cifra *round* por blocos que a partir de uma chave de comprimento 512-bits (que serão os blocos do texto inicial m_i), e um texto claro de comprimento 256-bits (o valor de *hash* H_{i-1}) retorna um bloco de comprimento 256 bits:

$$\mathcal{C}_{m_i}(H^{i-1})$$

São usadas seis funções lógicas na SHA-256, todas elas operando com palavras com 32-bits de comprimento e gerando como *output* uma palavra de 32 bits.

$$\begin{aligned} F_C(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ F_M(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\ \Sigma_0(x) &= S^2(x) \oplus S^{13}(x) \oplus S^{22}(x) \\ \Sigma_1(x) &= S^6(x) \oplus S^{11}(x) \oplus S^{25}(x) \\ \sigma_0(x) &= S^7(x) \oplus S^{18}(x) \oplus R^3(x) \\ \sigma_1(x) &= S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x) \end{aligned}$$

Cálculo de $\mathcal{C}_M(H)$

- a chave M de 512 bits é dividida em 16 sub-blocos de 32 bits:

$$M_0,\ M_1, \ldots, M_{15};$$

- são definidos 64 **blocos expandidos** usando:

$$\begin{aligned} W_j &= M_j, \quad j = 0, 1, \ldots, 15 \\ W_j &= \sigma_1(W_{j-2}) \boxplus W_{j-7} \boxplus \sigma_0(W_{j-15}) \boxplus W_{j-16}, \quad j = 16, \ldots, 63 \end{aligned}$$

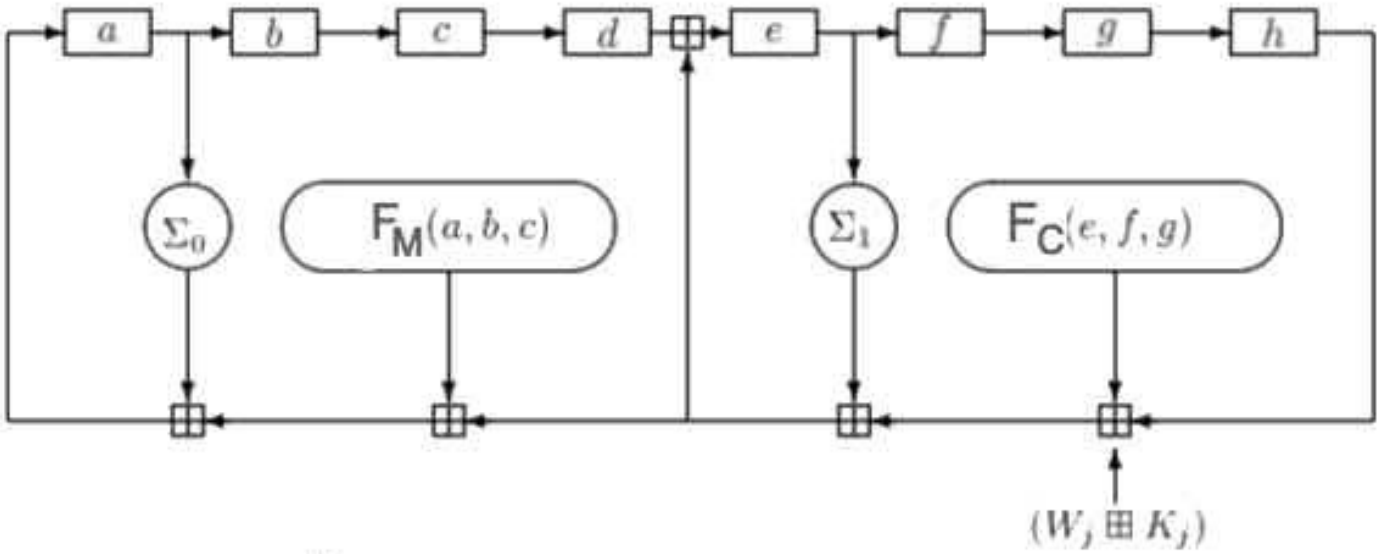
- são usadas 64 constantes, $K_0, \ldots, K_{63}$ definidas como os primeiros 32 bits da parte fraccionária das raízes cúbicas dos primeiros 64 primos;
- o texto H de comprimento 256 bits, é dividido em 8-sub-blocos de 32-bits, H_1, H_2, ..., H_8;
- são realizados 64 rounds usando os dados e funções anteriores

Sequência de 64-*rounds* definida por:

- Inicializar registros $a = H_1,\ b = H_2,\ \ldots\ h = H_8$;
- Para j=0 até 63, calcular:

$$T_1 = h \boxplus \Sigma_1(e) \boxplus F_C(e, f, g) \boxplus K_j \boxplus W_j$$
$$T_2 = \Sigma_0(a) \boxplus F_M(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d \boxplus T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 \boxplus T2$$

Cada *round* da função de compressão $\mathcal{C}$ corresponde com o percurso da figura:



Exemplos
valores *hash* com SHA-256

Exemplos de valores *hash* obtidos usando SHA-256 [1]:

$$\text{SHA}_{256}(000) = 2ac9a6746aca543af8dff39894cfe8173afba21eb01c6fae33d52947222855ef$$

$$\text{SHA}_{256}(001) = 7a3e6b16cb75f48fb897eff3ae732f3154f6d203b53f33660f01b4c3b6bc2df9$$

O valor *hash* de

$$m = 0000000000000000000000000000000000000000000000000000000000000000$$

usando SHA-256 é

$$\text{SHA}_{256}(m) = 19ebde8a753772a7361df1046fbec18c34d27dc1a20e15e8944a0d5e46629e75$$

[1] https://www.xorbin.com/tools/sha256-hash-calculator

Autenticadores de mensagens (MACs)

Um **autenticador de mensagem** ou MAC (*Message Authentication Code*) ou *tag*, é um tipo de função *hash* que recebe como entrada uma chave secreta K e uma mensagem de tamanho arbitrário m e como saída um MAC ou *etiqueta*, de tamanho fixo:

$$h_K(m)$$

Exemplos

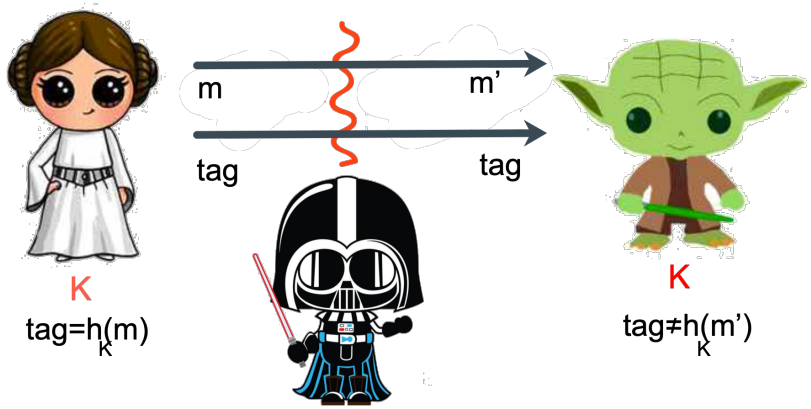
$$h_{13}(bonsdias) = 9269ef75c8591cf12d07cc95e862b573753bda8a$$

$$h_{101}(bonsdias) = 7413b59ca1d8dab126875f3e43e3bb80b815051f$$

$$h_{101}(osautenticadoresdemensagensgarantemaintegridadedosdadoseaautenticidadedoemissor) = 09254b24e355ac3cdae5b0438e8692991a48e3ad$$

Autenticadores de mensagens (MACs)

- O emissor calcula o MAC (etiqueta) associado a uma mensagem m usando a chave secreta K, $h_K(m)$ e envia ao destinatário a mensagem e o MAC;
- O destinatário recebe a mensagem m e o MAC, $h_K(m)$, calculado pelo emissor, calcula ele próprio o MAC da mensagem recebida e compara com a etiqueta.



Se o MAC que calculou ou destinatário coincide com o MAC enviado pelo emissor, está garantida a integridade dos dados e a autenticidade do emissor.

Construção HMAC

O esquema HMAC permite construir a partir de uma função *hash criptográfica* h e de uma chave secreta K, um autenticador de mensagens. O MAC obtido costuma denotar-se por HMAC-h (por exemplo HMAC-SHA256).

A definição de um HMAC precisa de uma função *hash* criptográfica h, que supomos realiza o *hash* usando uma função de compressão interna que trabalha em blocos de dados de comprimento B-bytes e proporciona um valor *hash* de comprimento ℓ .

A chave de autenticação K, pode ter qualquer tamanho até B, o comprimento interno dos blocos da função *hash* h. Para uso de chaves com tamanho inferior, é usado um padding à direita com zeros, para uso de chaves com comprimento superior, é preciso calcular primeiro o valor *hash* da chave K.

Construção HMAC

Sejam h e K a função *hash* criptográfica e a chave secreta K verificando as condições anteriores e:

- m a mensagem a ser autenticada
- opad denota o byte 0x5C repetido B vezes (*outer padding*)
- ipad denota o byte 0x36 repetido B vezes (*inner padding*);

Define-se

$$HMAC(K, m) = h((K \oplus \text{opad})||h((K \oplus \text{ipad})||m))$$

Pode consultar on-line implementações do HMAC com diferentes funções *hash* (por exemplo em https://www.freeformatter.com/hmac-generator.html).

Exemplos - HMAC-SHA256

1. $m=$ Um HMAC gera tags para etiquetar mensagens
   $K=$ 0
   $HMAC(K, m)=$ 821f34c9796e35312d3c19a40edc64860c7ea9b8ce636be95c2287a233c8b343
2. $m=$ Um HMAC gera tags para etiquetar mensagens
   $K=$ 997
   $HMAC(K, m)=$ 94c18514b5a01471d05945157d78fabe3eaf4e5f7ef9b4240a244308c5213654
3. $m=$ Um HMCM gera tags para etiquetar mensagens
   $K=$ 997
   $HMAC(K, m)=$ 8a4222bf8e889942f7091dd8970549eb186ec9a021c8eda98f25fddf9d1723df