Universidade da Beira Interior
Faculdade de Engenharia
Departamento de Informática

Segurança Informática
Aula 4
Licenciatura em Engenharia Informática
Licenciatura em Informática Web

Computer Security
Lecture 4
Degree in Computer Science and Engineering
Degree in Web Informatics

Sumário
Definição de funções de *hash* criptográficas e estudo das suas propriedades. Códigos de autenticação de mensagens como forma de garantia de integridade e segurança contra manipulação de dados. Breve referência ao conceito de criptografia autenticada.

Summary
Definition of cryptographic hash functions and study of their properties. Message Authentication Codes (MACs) as a means to guarantee the integrity and security against manipulation of the data. Brief mention to the concept of authenticated encryption.

1 Funções de *Hash* Criptográficas
Cryptographic Hash Functions

1.1 Introdução
Introduction

Na literatura da especialidade, os três pilares em que assenta a segurança da informação são a **confidencialidade**, a **integridade** e a **disponibilidade**[1]. Basicamente, a ideia é que o sistema que se quer construir deve garantir o seguinte:

- **[confidencialidade]** os dados não podem ser acedidos por quem não lhes pertence;
- **[integridade]** quando acedidos em momentos diferentes ou por entidades diferentes, os dados encontram-se como inicialmente foram deixados ou transmitidos;
- **[disponibilidade]** e que aplicação dos mecanismos envolvidos no preenchimento das condições anteriores (ou qualquer outras do sistema) não impedem que sejam acedidos em tempo útil por quem de direito.

Para além destes três pilares, **podem-se enumerar vários requisitos a que a criptografia moderna tenta responder** atualmente, nomeadamente:

1. Confidencialidade;
2. Integridade;
3. Autenticação;
4. Anonimato;
5. Não Repúdio;
6. etc.

[1] Estes 3 pilares são mais conhecidos pelo acrónimo CIA, da expressão inglesa ***C**onfidentiality, **I**ntegrity and **A**vailability*.

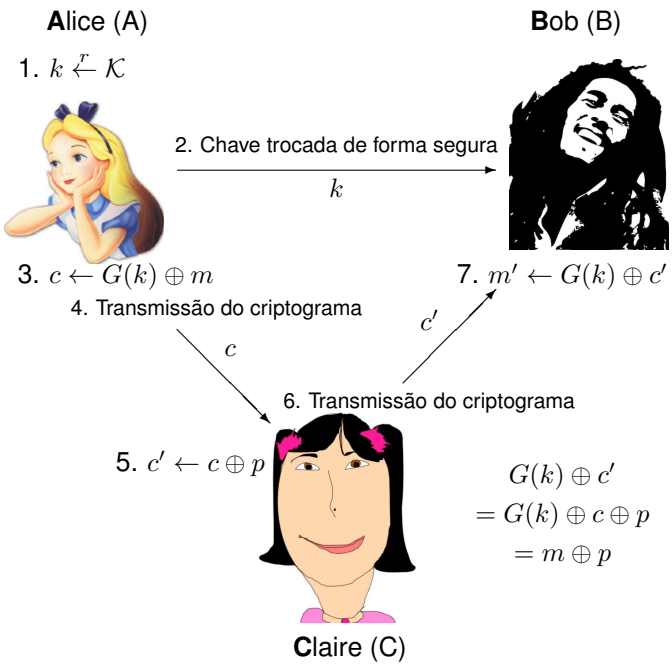
Estes requisitos são **por vezes referidos de forma ortogonal aos conceitos referidos antes**, embora alguns estejam relacionados. Anteriormente foram já discutidas algumas ferramentas da criptografia que permitem assegurar a confidencialidade dos dados, neste caso as cifras de chave simétrica. Contudo, também foi demonstrado que, **por si só, essas cifras não garantem todos os requisitos de segurança**, especialmente se forem usadas cifras de chave simétrica contínuas, em que o criptograma é maneável.

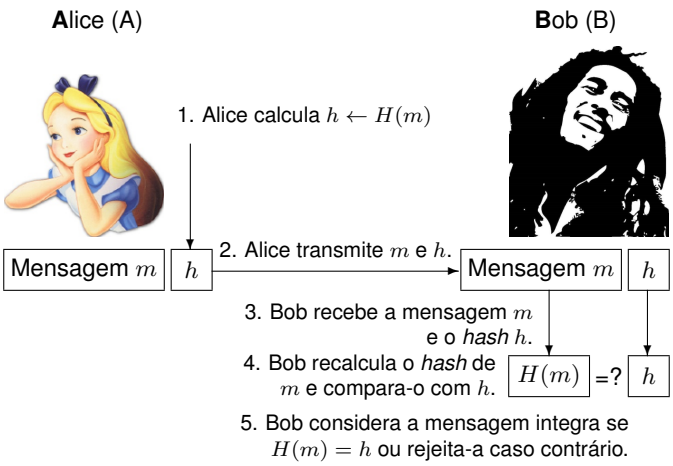
A figura incluída em baixo, repetida aqui por conveniência, demonstra **o problema associado à maneabilidade** do criptograma, que constitui **uma motivação para o estudo do tema desta aula**. Aparentemente, no ataque representado, **o problema surge do facto do Bob não conseguir distinguir se um criptograma foi alterado em trânsito** ou, por outras palavras, se veio inalterado da sua fonte inicial ou não. É um problema de integridade e autenticação.

O problema abordado nas próximas duas grandes secções é então o da integridade, não o da confidencialidade. Assim, considere que **a única preocupação é de construir mecanismos que garantam a integridade dos dados na transmissão**. Para isso, vão ser introduzidos os conceitos de função de *hash*, *Message Authentication Codes* (MACs) e, mais tarde, de cifra autenticada.

1.2 Definição de Função de *Hash* Criptográfica
Definition of Cryptographic Hash Function

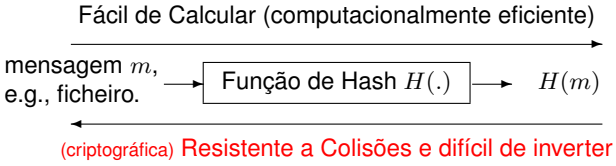
Sem a qualificação de *criptográfica*, uma **função de**





hash –também designada por **função dispersão** ou **função resumo** em português– é apenas uma função $H(m)$, **definida do espaço de mensagens $\mathcal{M}$ com tamanho arbitrariamente grande (mas finito) para uma sequência de bits $\{0,1\}^n$ com tamanho fixo n**. Portanto, esta sequência de bits determina um espaço potencialmente **muito** mais pequeno que $\mathcal{M}$ (i.e., $|\mathcal{M}| >> 2^n$).

Uma função de *hash* é **tipicamente uma função não injetiva** (i.e., a vários *inputs* diferentes pode corresponder o mesmo valor de *hash*), **compressora** (o seu retorno tem sempre o mesmo tamanho, independentemente do tamanho do *input*) **e fácil de computar** (i.e., dado o algoritmo da função $H(.)$ e a mensagem m, é computacionalmente eficiente calcular $H(m)$):



O *Ciclic Redundancy Check* 32 (CRC32) é um exemplo de uma função de *hash* normal, bem como qualquer operação com módulo. O CRC32 é, de resto, utilizado para verificar a integridade de tramas *Ethernet* enviadas entre dois nós de rede, sendo calculado pelo emissor e anexado à mensagem transmitida. O recetor também calcula o CRC32 da mensagem, comparando-o com o recebido e rejeitando ou aceitando a mensagem conforme sejam diferentes ou iguais. **A utilização de funções de *hash* para deteção de erros** ilustra-se na figura seguinte. Note-se que **esta construção não é segura contra ataques de homem no meio ativos**, já que qualquer adversário consegue capturar uma mensagem, alterar essa mensagem e calcular novo valor de resumo, e enviá-lo ao recetor, que verifica e aceita a mensagem.

Para permitirem assegurar a integridade ou **preencher outros requisitos de segurança para uma mensagem**, estas funções devem ainda possuir **propriedades adicionais**, nomeadamente **resistência a colisões**.

Uma função de *hash* $H(m)$, definida do espaço de mensagens $\mathcal{M}$ para $\{0,1\}^n$, diz-se **criptográfica** se for **computacionalmente inviável obter quaisquer duas mensagens m_0 e m_1 com o mesmo valor de *hash***, i.e.,

$$H(m) : \mathcal{M} \rightarrow \{0,1\}^n$$

e

$$\forall m_0, m_1 \in \mathcal{M} \text{ and } \forall \text{ Algoritmos eficientes } A$$
$$P[A \text{ devolver uma colisão para } H] < \epsilon \leq 2^{-80}.$$

1.3 Propriedades de Funções de *Hash* Criptográficas

Properties of Cryptographic Hash Functions

Prova-se que a propriedade da resistência a colisões determina outras duas propriedades, pelo que a função de *hash* criptográfica pode ser caracterizada da seguinte forma:

- **Compressora e fácil de computar**;
- **Resistência à descoberta de um texto original**– dado qualquer valor de hash $h \in \{0,1\}^n$, é computacionalmente inviável encontrar uma mensagem $m \in \mathcal{M}$ que lhe dê origem;
- **Resistência à descoberta de um segundo texto original**– dado uma mensagem $m_0 \in \mathcal{M}$, é computacionalmente inviável encontrar outra mensagem $m_1 \in \mathcal{M}$ com o mesmo valor de hash da primeira, i.e. $H(m_0) = H(m_1)$;
- **Resistência à colisão**– é computacionalmente inviável encontrar quaisquer duas mensagens $m_0, m_1 \in \mathcal{M}$ com o mesmo valor de *hash*.

Num tom coloquial, pode-se dizer que a função de *hash* criptográfica fornece uma segurança probabilística: é

que **é muito pouco provável que um valor de *hash* derive de uma mensagem diferente daquela a que aparece associada**.

Resistência à Descoberta de um Texto Original

As primeiras **duas propriedades** enunciadas antes são refletidas no facto de que, para uma função de *hash* criptográfica, **seria necessário um ataque de força bruta**, i.e., testar uma média de 2^{n-1} mensagens m_k, **para encontrar um valor de $H(m_k)$ que fosse igual ao estabelecido ou ao da mensagem original**.

Resistência à Colisão

A propriedade **da resistência a colisões** significa que seria preciso tentar aproximadamente $1,177 \times 2^{n/2}$ pares de mensagens para encontrar duas que colidissem com probabilidade igual a 0,5. **Um ataque** para encontrar quaisquer duas mensagens com o mesmo valor de *hash* é chamado de **Ataque do Aniversário**.

A **complexidade** referida em cima deve-se ao chamado **paradoxo do aniversario**, tipicamente associado à **facilidade** com que se atinge **50% de probabilidade de encontrar duas pessoas cujo aniversário se dê no mesmo dia do ano**, bastando **23 pessoas** para que isso aconteça. Devido a isso, a tarefa de encontrar quaisquer duas mensagens com o mesmo valor de *hash* é a menos complexa das 3 enunciadas, fornecendo um limite inferior para a sua segurança. Para o *Secure Hash Algorithm* 1 (**SHA1**), cujo ***output* é de 160 bits**, esta complexidade **é de 1.177×2^{80}, embora haja já algoritmos que permitem consegui-lo com cerca de 2^{51} avaliações da função** para algumas mensagens específicas.

1.4 Construção de Funções de *Hash* Criptográfica
Construction of Cryptographic Hash Functions

Uma das melhores formas de construir funções de *hash* criptográficas é conhecida como a **construção iterativa de Merkle–Damgard**. Segundo os seus proponentes, dada uma função g compressora, que aceite dois valores de entrada de tamanho r e n bits e os transforme noutro com o tamanho de n bits, i.e.,

$$g : \{0,1\}^n \times \{0,1\}^r \rightarrow \{0,1\}^n,$$

pode-se obter uma função $H(m)$ que primeiramente aplique g a um **vetor de inicialização** (vi) e ao primeiro bloco da mensagem com r bits, e depois **a aplique iterativamente a cada bloco de r bits e ao resumo obtido no passo anterior**, **devolvendo o valor resultante para o último bloco**, conforme se formaliza e ilustra a seguir:

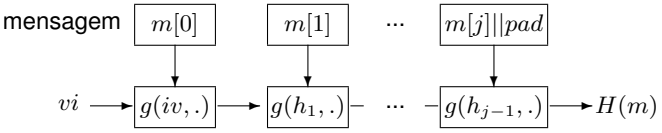
$$H : \mathcal{M} \rightarrow \{0,1\}^n,$$

em que

$$H(m) = h_j, j = |m| \div r$$

e

$$\begin{cases} h_0 = g(vi, m[0]) \text{ ou } h_0 = g(vi, m[0]||pad), \text{ se j=0} \\ h_i = g(h_{i-1}, m[i]), \text{ para } i = 1, ..., j-1 \\ h_j = g(h_{j-1}, m[j]||pad) \end{cases}$$



A grande vantagem de utilizar esta construção deve-se ao teorema que determina que **qualquer função $H : \mathcal{M} \rightarrow \{0,1\}^n$ construida a partir de $g : \{0,1\}^n \times \{0,1\}^r \rightarrow \{0,1\}^n$ conforme o esquema de Merkle–Damgard é resistente a colisões, se $g(.)$ também o for.**

Repare-se que este **resultado é extremamente útil**, pois **permite construir funções de *hash* criptográficas para mensagens de qualquer tamanho usando apenas funções mais simples**. Provar a propriedade da resistência a colisões para funções compressores como as aqui simbolizadas por g será mais simples à partida. Algumas das funções de *hash* usadas no passado (e.g., *Message Digest* 5 –MD5) e na atualidade (e.g., SHA1) elaboram precisamente nesta construção.

A **demonstração** do teorema anterior é **feita por redução ao absurdo**, e passa por considerar que **duas mensagens m_0 e m_1 diferentes** colidem para $H(.)$, demonstrando que isso **implica que $g(.)$ não é resistente a colisões** ou que **as duas mensagens são iguais**:
Temos que

$$g(h_{0,t}, m_0[t]||pad_t) = H(m_0) = H(m_1) = g(h_{1,r}, m_1[r]||pad_t).$$

Se $h_{0,t} \neq h_{1,r}$ ou $m_0[t] \neq m_1[r]$ ou $pad_{0,t} \neq pad_{1,r}$, então já encontramos uma colisão para $g(.)$■
Caso contrário, $h_{0,t} = h_{1,r}$ **e** $m_0[t] = m_1[r]$ **e** $pad_{0,t} = pad_{1,r}$. Ora, se $pad_{0,t} = pad_{1,r}$, então as mensagens são do mesmo tamanho, e

$$g(h_{0,t-1}, m_0[t-1]) = h_{0,t} = h_{1,t} = g(h_{1,t-1}, m_1[t-1]).$$

De novo, se $h_{0,t-1} \neq h_{1,t-1}$ ou $m_0[t-1] \neq m_1[t-1]$, então encontrámos uma colisão para $g(.)$, o que invalida uma das assumções.■
Caso contrário, $h_{0,t-1} = h_{1,t-1}$ e $m_0[t-1] = m_1[t-1]$ e podemos continuar a iterar até chegar ao primeiro bloco da mensagem, concluindo que:

- ou encontramos uma colisão para $g(.)$, o que é absurdo;
- ou as duas mensagens são iguais, o que é absurdo■

A grande **desvantagem desta construção é que não é paralelizável**, o que não é muito desejável nos dias de hoje. **O novo SHA3** (que elabora numa família de funções esponja chamadas **Keccak**) **não usa esta construção, de modo a facilitar implementações para processamento paralelo**.

1.5 *Message Digest* 5 (MD5)

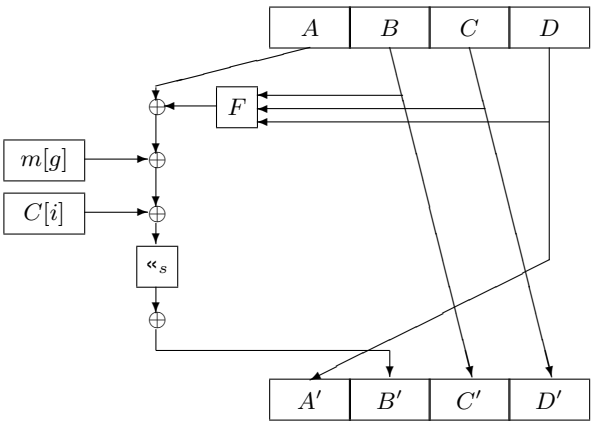
Message Digest 5 (MD5)

O **MD5** (e o MD4) produzem **resumos de 128 bits** de **mensagens com tamanho máximo de** $2^{64} - 1$ **bits** (devido a uma opção de desenho do algoritmo). Cada mensagem é preenchida até um múltiplo de 512 bits (os últimos 64 bits contêm o tamanho da mensagem).

O algoritmo **opera sobre um estado de 128 bits dividido em 4 palavras de 32 bits**, designadas por A, B, C e D, que são inicializadas com constantes. A mensagem é **dividida** em blocos de 512 bits, que são subdivididos em **peças** de 32 bits. Cada peça é combinada com A, B, C e D em 64 operações, representadas pelo esquema em baixo, onde F é uma das seguintes funções (cada função F é usada 16 vezes durante as 64 operações):

- $F(X, Y, Z) = (X \wedge Y) \vee \neg(X \wedge Z)$
- $G(X, Y, Z) = X \wedge Z \vee Y \wedge \neg Z$
- $H(X, Y, Z) = X \oplus Y \oplus Z$
- $I(X, Y, Z) = Y \oplus (X \vee \neg Z)$

onde $\wedge$, $\vee$, $\neq$ e $\oplus$ denotam os operadores lógicos E, OU, negação e OU exclusivo.



Na figura anterior, $m[g]$ representa uma determinada peça da mensagem M, enquanto que $C[i]$ se refere a uma de 64 constantes definidas no algoritmo. Finalmente, $<<_s$ denota a operação de deslocamento bit-a-bit em s posições. g e s são diferentes para cada uma das 64 operações.

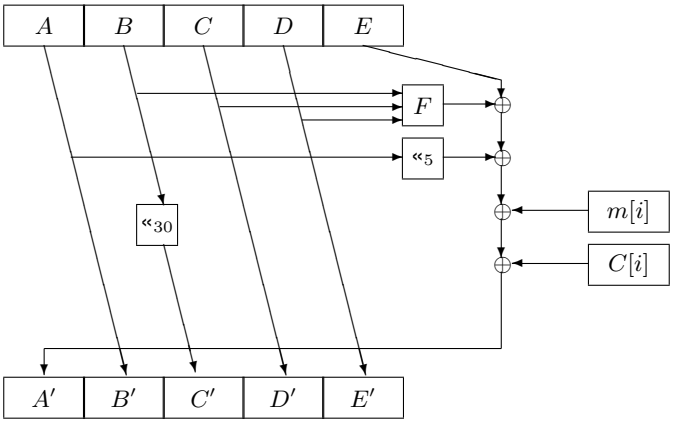
Por curiosidade, costuma indicar-se **o valor de *hash* da cadeia de caracteres vazia** em hexadecimal:

```
MD5() = d41d8cd98f00b204e9800998ecf8427e
```

1.6 *Secure Hash Algorithm* 1 (SHA1)

Secure Hash Algorithm 1 (SHA1)

O **SHA1** (e também o SHA-0) produz **um resumo de 160 bits de qualquer mensagem com tamanho máximo de** $2^{64} - 1$ **bits** (ver esquema de preenchimento usado no MD5). O SHA1 é baseado em princípios semelhantes aos que foram usados no desenho do MD4 e do MD5, mas tem um *design* mais conservador (basta ver a figura e o conjunto de funções utilizadas), conforme sugere a figura seguinte:



Para o SHA1, o conjunto de funções utilizadas é:

- $F(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$
- $G(X, Y, Z) = X \oplus Y \oplus Z$
- $H(X, Y, Z) = (X \wedge Y) \vee (Y \wedge Z) \vee (X \wedge Z)$
- $I(X, Y, Z) = X \oplus Y \oplus Z$

É **largamente utilizado em aplicações criptográficas**, nomeadamente *Transport Layer Security/Secure Sockets Layer* TLS/SSL, *Pretty Good Privacy* (PGP), *Secure Shell* (SSH), S/MIME e *Internet Protocol Security* (IPSec). A principal **motivação para a publicação e normalização do SHA1 foi o aparecimento da norma da Assinatura Digital**, no qual está incluído. Por curiosidade, a string vazia produz o seguinte valor de *hash* para o SHA1:

```
SHA1() = da39a3ee5e6b4b0d3255bfef95601890afd80709
```

A **família de funções SHA2** usa um algoritmo idêntico ao do SHA1 (ver em baixo) mas oferece a funcionalidade de **escolher o tamanho do resumo**: SHA-224, SHA-256, SHA-384, e SHA-512.

1.7 Eficiência Computacional

Computational Performance

De modo a obter uma visão global da *performance* dos algoritmos que implementam funções de *hash* atuais, inclui-se a tabela que **mostra a velocidade de cálculo dos valores resumo**. Segundo o autor original desta tabela[2], estes valores dizem respeito à implementação em C++ de Wei Dai na biblioteca Crypto++ 5.6.0, e foram obtidos numa máquina AMD Opteron com um processador a 2.2 GHz com sistema operativo Linux:

* Como antes referido, o melhor método para encontrar colisões no SHA1 requer apenas 2^{51} avaliações da função.

[2]Esta tabela foi adaptada de um conjunto de diapositivos de Dan Boneh.

Norma NIST Função *hash*	Tamanho	Velocidade	Segurança
SHA1	160	153 MB/seg	$2^{80}*$
SHA256	256	111 MB/seg	2^{128}
SHA512	512	109 MB/seg	2^{256}
Outras			
Whirlpool	512	57 MB/seg	2^{256}

2 Códigos de Autenticação da Mensagem

Message Authentication Codes

Como brevemente mencionado anteriormente, **por sí só, as funções de *hash* (mesmo que criptográficas) não resolvem todos os problemas relacionados com a integridade dos dados**, nomeadamente aqueles relacionados com a alteração intencional das mensagens e dos seus respetivos códigos. **O uso de uma função de *hash* endereça apenas erros aleatórios, e não intencionais**. Assim, **precisamos de um mecanismo um pouco mais poderoso** não vulnerável nesses casos. Esse mecanismo é **conhecido como *Message Authentication Code* (MAC)** ou, em português, **Código de Autenticação da Origem da Informação** ou **Código de Autenticação da Mensagem**.

As funções de *hash* são vulgarmente usadas na deteção de erros de transmissão de ficheiros na *Internet*. Por exemplo, o **BitTorrent usa funções de *hash* para garantir que os pedaços de ficheiro transmitidos por vários *peers* não chegam com erros**, pedindo a retransmissão caso tal seja detetado.

2.1 Definição de Autenticação da Mensagem

Message Authentication Code Definition

Tal como para as cifras de chave simétricas, **um MAC é um par de algoritmos** (S, V) **definidos sobre o espaço de chaves, mensagens e códigos possíveis**, $\mathcal{K}, \mathcal{M}, \mathcal{T}$, tal que

$$S : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$$

e

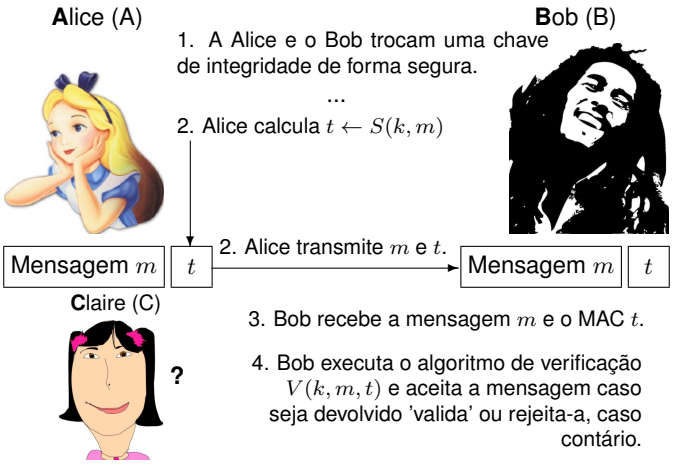
$$V : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{\text{'verifica', 'não verifica'}\},$$

em que

$$V(k, m, S(k, m)) = \text{'verifica'}.$$

$S(k, m)$ **é normalmente designada por algoritmo de assinatura, enquanto que** $V(k, m, t)$ **é conhecido como o algoritmo de verificação.** De uma maneira informal, pode-se dizer que **o algoritmo de verificação é aquele que**, dada a mensagem, a chave de integridade e o código de autenticação da mensagem, **devolve o valor 'verifica' ou 'não verifica', conforme o código corresponda ou não à mensagem a que está associado.**

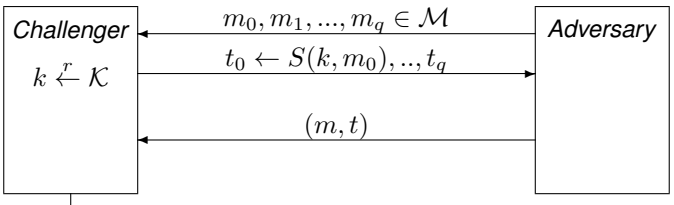
Note-se que **o fator que dá vantagem ao MAC** face a um simples mecanismo de integridade é **a inclusão de uma chave de integridade** nos algoritmos, contrariamente ao que acontecia antes. Assume-se que **apenas as entidades que estão efetivamente em comunicação possuem essa chave** e que, portanto, um **adversário mal intencionado não deverá conseguir alterar a mensagem** (ou o criptograma) **e o código simultaneamente**.



Ao receber e validar um MAC com sucesso, **o Bob fica com um elevado grau de confiança de que não houve erros na transmissão** e que **ninguém alterou a mensagem em trânsito**.

Claro que nem todas as construções de MAC são seguras. **A segurança é normalmente definida em termos do modelo de *Chosen Plaintext Attack* em que**, neste caso, o adversário tem o **poder de escolher qualquer conjunto de mensagens** $m_0, m_1, ..., m_q \in \mathcal{M}$ **e obter os seus MACs** antes do desafio. O objetivo do adversário é, neste caso, conseguir produzir **um novo par** mensagem m/MAC t (diferentes dos anteriores, i.e., $(m, t) \notin \{(m_0, t_0), ..., (m_q, t_q)\}$) **que sejam verificados corretamente** pelo desafiador. **O adversário ganha o jogo se conseguir produzir um MAC para qualquer mensagem arbitrária** (mesmo se for um MAC diferente para uma mensagem já conhecida).

O jogo pode ser esquematizado da seguinte forma:



b' = 1 se $V(k, m, t) =$ 'verifica' e $(m, t) \notin \{(m_0, t_0), ..., (m_q, t_q)\}$

b' = 0 caso contrário.

No jogo ilustrado antes, o adversário ganha se $b = 1$.

Existem **várias formas de construir MACs**. Em baixo, discutem-se as **duas** seguintes:

1. ***Encrypted Cipher Block Chaining* MAC (ECBC-MAC)**, construido a partir de funções de permutação pseudo-aleatórias seguras, e muito populares em aplicações bancárias e;
2. **MAC baseado em *Hashing***, construido a partir de

funções de *hash* criptográficas, e muito populares em protocolos da Internet.

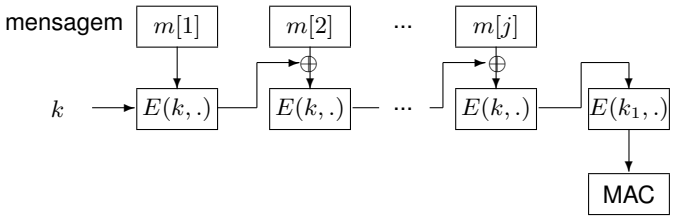
2.2 *Encrypted Cipher Block Chaining* MAC (ECBC-MAC)

Encrypted Cipher Block Chaining MAC (ECBC-MAC)

É possível **provar** que **qualquer função de permutação pseudo-aleatória segura** concretiza um **excelente MAC para mensagens com tamanho menor ou igual ao do bloco da permutação**. Por exemplo, para mensagens menores que 128 bits, o resultado de $E_{AES}(k, m)$ pode ser interpretado como um código MAC. Repare-se que, neste caso, **um adversário dificilmente ganharia o jogo acima indicado, já que esta função é reconhecimente indistinguível de uma outra função aleatória**, e o atacante não beneficiaria [3] nada por observar pares $\{(m_0, t_0), ..., (m_q, t_q)\}$ antes do desafio.

O problema agora coloca-se em como construir um MAC a partir de funções de permutação pseudo-aleatórias para qualquer tamanho da mensagem, e uma das formas de responder a este problema é conhecida como *Encrypted Cipher Block Chaining* MAC (ECBC-MAC).

Seja $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{X}$ uma **função de permutação pseudo-aleatória segura**. Então, o algoritmo de assinatura ECBC-MAC, definido para $\mathcal{K}^2 \times \mathcal{X}^j \rightarrow \mathcal{X}$ de acordo com o circuito incluído a seguir, **é um MAC seguro**:



Note-se que **esta construção aceita duas chaves de cifra**, ao contrário do que acontece para o modo de cifra respetivo. Isto acontece porque **o último bloco tem de ser cifrado uma última vez** com uma chave diferente para que este MAC seja seguro.

2.3 MAC baseado em *Hashing*

Hash Based Message Authentication Code (HMAC)

Um ***Hash Based Message Authentication Code*** (HMAC) constitui uma **forma peculiar, popular, direta e relativamente simples de combinar uma função de *hash* criptográfica com uma chave secreta para obter o MAC**. Neste caso a função de assinatura HMAC: $\mathcal{K} \times M \rightarrow \mathcal{T}$ é dada por

$$\text{HMAC}(k, m) = H((k \oplus opad)||H((k \oplus ipad)||m)),$$

onde

- $H(.)$ é uma função de *hash* criptográfica, e.g., SHA256;
- k é uma chave secreta preenchida com zeros até ao tamanho do bloco da função de *hash*;
- m é a mensagem a ser autenticada;
- $||$denota concatenação e $\oplus$ o OU exclusivo (XOR);
- $opad$ e $ipad$ são os preenchimentos externos (`0x5c5c5c...5c5c`) e internos (`0x363636...3636`), respectivamente, i.e.,

A função de verificação consiste simplesmente no recálculo do HMAC na receção e respetiva comparação:

$$\text{HMAC}_{ver}(k, m, t) = \begin{cases} \text{'verifica', se HMAC}(k, m) = t \\ \text{'não verifica', caso contrário} \end{cases}$$

O HMAC está especificado no *Request for Comments* (RFC) 2104: HMAC: *Keyed-Hashing for Message Authentication.* O HMAC é um dos MACs mais populares atualmente sendo, por exemplo, utilizados os HMAC-SHA-1 e o HMAC-MD5 nos protocolos IPsec e TLS.

3 Cifra Autenticada

Authenticated Encryption

O **conceito** específico de cifra autenticada **surgiu apenas no ano 2000**, embora existissem já algumas implementações que **combinavam**, de alguma forma, **cifragem e códigos de autenticação da mensagem**. Algumas dessas combinações estavam, contudo, erradas. A **três formas possíveis** de combinar os dois mecanismos são:

1. Calcular o MAC da mensagem, cifrar a mensagem e concatenar o criptograma com o MAC –também conhecido como *MAC and Encrypt*– usado no SSH;
2. Calcular o MAC da mensagem, concatená-lo à mensagem e cifrar o resultado desta operação – também conhecido como *MAC then Encrypt*– usado no TLS/SSL;
3. **Cifrar a mensagem, calcular o MAC do criptograma e concatenar os dois –também conhecido como *Encrypt then MAC*–** usado no IPSec.

Apesar de ser possível implementar qualquer uma das formas referidas em cima sem que isso acarrete necessariamente uma falha de segurança, **a forma considerada sempre correta é a terceira**, já que a inclusão do MAC **não degenera nunca em vazamento de informação** e o encadeamento **não é vulnerável a *timing attacks***.

Nota: o conteúdo exposto na aula e aqui contido não é (nem deve ser considerado) suficiente para total enten-

[3] Beneficiaria de forma negligenciável.

dimento do conteúdo programático desta unidade curricular e deve ser complementado com algum empenho e investigação pessoal.