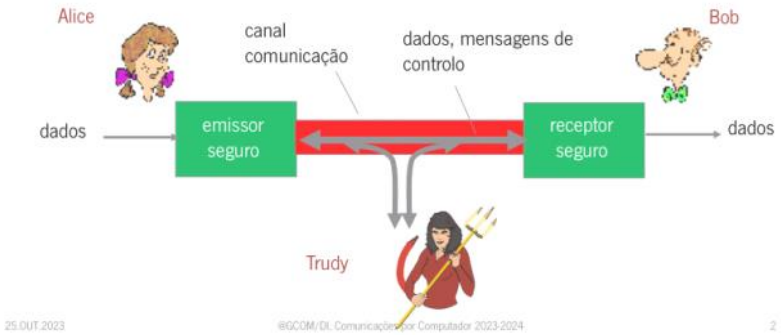
Segurança em Redes

23 de janeiro de 2024 17:24

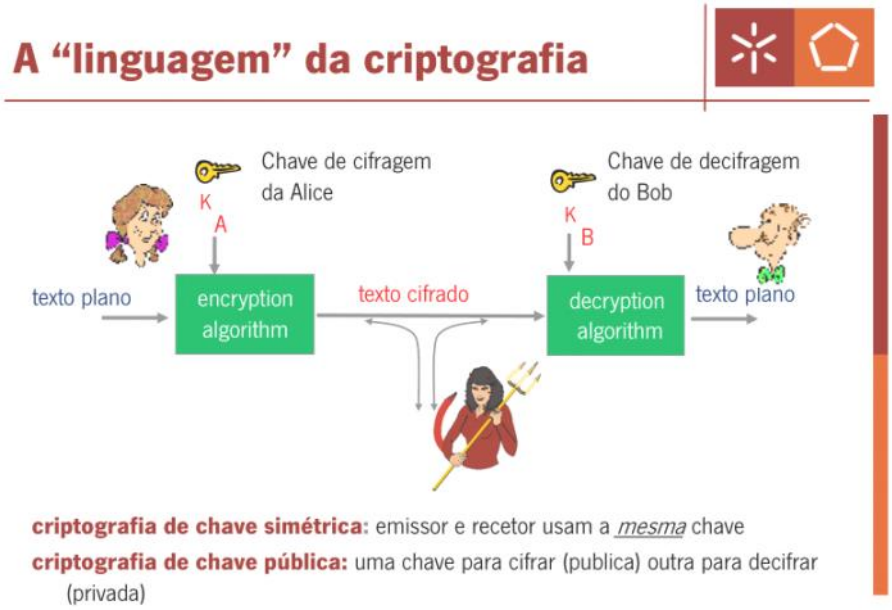


O que a Trudy pode fazer:

- espionagem: interceção indevida de mensagens
- inserção de mensagens numa conexão
- disfarce: pode fingir (spoof) endereços de origem nos pacotes desviar sessoes (hijacking): "tomar conta" de conexoes que estao a decorrer, remover o emissor ou o recetor, colocando-se no lugar destes
- negaçao de serviço: impedir premeditadamente que um serviço seja usado por outros (sobrecarregando-o de algum modo)

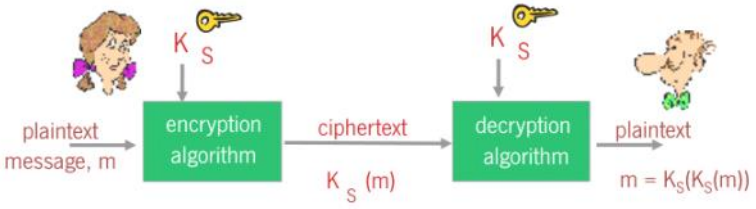
Propriedades ded uma comunicação segura

- Confidencialidade: só o emissoer e o recetor devem "perceber" o conteúdo das mensagens
- Autenticação: confirmar identidade um do outro
- Integridade das mensagens: garantir que a mensagem não foi alterada
- Não repúdio: evidências que impeçam intervenientes de negar comunicação
- Acesso e disponibilidade: serviços devem estar acessíveis e com disponibilidade para os seus utilizadores



Criptografia de chave simétrica

- ataque baseado no texto cifrado: Trudy tem textos cifrados que pode analisar
- duas abordagens: força bruta, análise estatística
- ataque baseado num texto conhecido: Trudy conhece o texto original e o texto cifrado
- ataque baseado num texto previamente escolhido: Trudy consegue obter uma versão cifrada de um texto escolhido por ela



Chave simétrica: Alice e Bob conhecem a mesma chave (simétrica) K_S

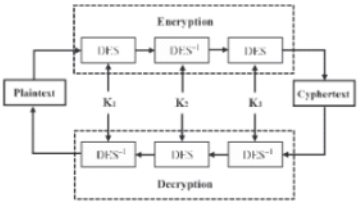
- Ex: conhecem o padrão de substituição do alfabeto! (ou a máquina de escrever, como a famosa ***Enigma*** da 2ª guerra mundial)

- **Pergunta: Como podem eles combinar a chave?**

m ---> Ks (m) ---> m = Ks(Ks(m))

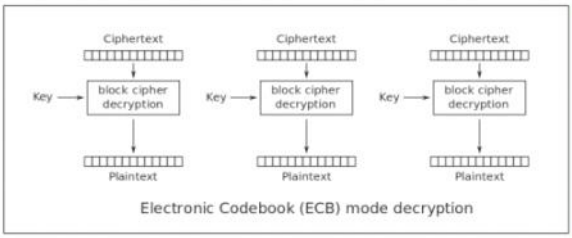
Algoritmos mais usados

- **DES – Data Encryption Standard** (fraco)
  - Chaves de 56 bits que processam blocos de 64 bits de cada vez
  - Quebra-se por força bruta em **menos de 1 dia**!
- **3-DES (3 x DES)**
  - Usa 3 chaves DES sequencialmente
    Cifra com K1, decifra com K2, cifra com K3
  - Chaves:
    56 bit (todas iguais, compatível DES),
    112 bit (k1=K3) ou 168 bit (3 distintas)
  - Prevê-se que possa ser usado até ao ano 2030..

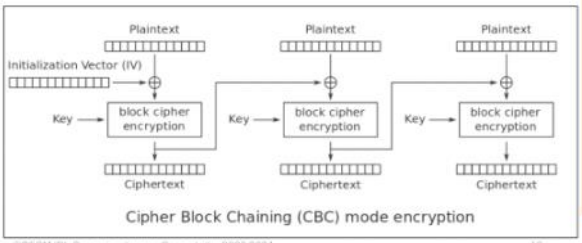


- **AES – Advanced Encryption Standard**
  - Veio em 2001 para substituir o velho DES
  - Processa blocos de 128 bits de cada vez
  - Chaves de 128, 192 ou 256 bits de tamanho
  - Pela força bruta, o que no DES demora um segundo a quebrar ...
  - ... demorará **149 triliões de anos** no AES!!!

- **ECB:**
  - O mesmo padrão na entrada, produz a mesma saída!
  - $m_{(i)}$ = "HTTP/1.1" → C(i) = ""k329aM02" qualquer que seja (i)

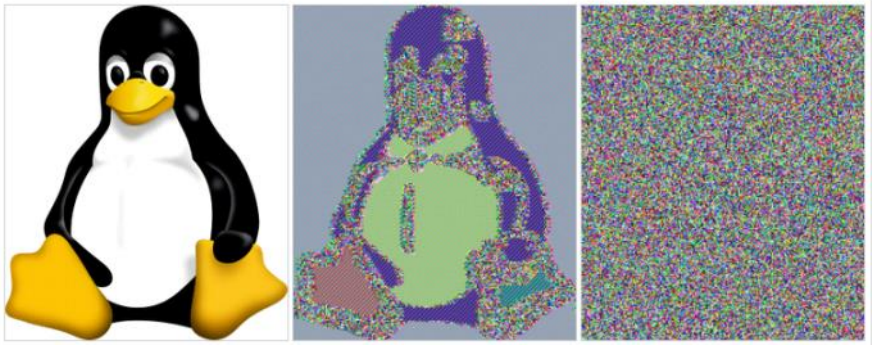


- **CBC:**
  - Antes de cifrar, mistura (XOR) o bloco de texto de entrada com o bloco anterior cifrado
  - $C_{(0)}$ = IV (Vetor inicial enviado às claras)
  - $C_{(i)} = K_S (m_{(i)} \text{ XOR } C_{(i-1)})$



Imagens: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

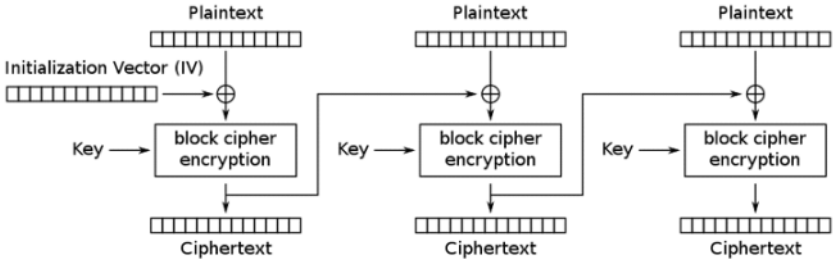
IV must be a cryptographic nonce (used just once under the same key)



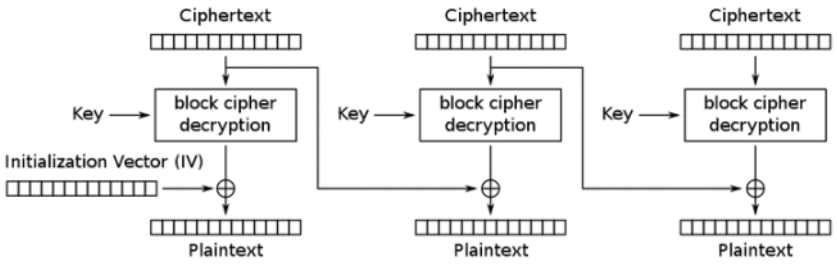
Original image | Using ECB allows patterns to be easily discerned | Modes other than ECB result in pseudo-randomness

Cipher block chaining (CBC) [edit]

Ehrsam, Meyer, Smith and Tuchman invented the cipher block chaining (CBC) mode of operation in 1976.[23] In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block.



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

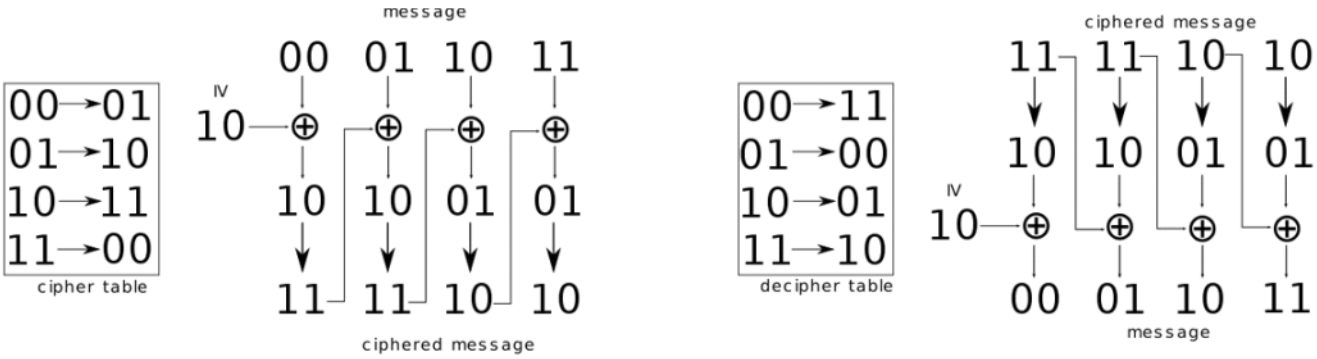
If the first block has index 1, the mathematical formula for CBC encryption is

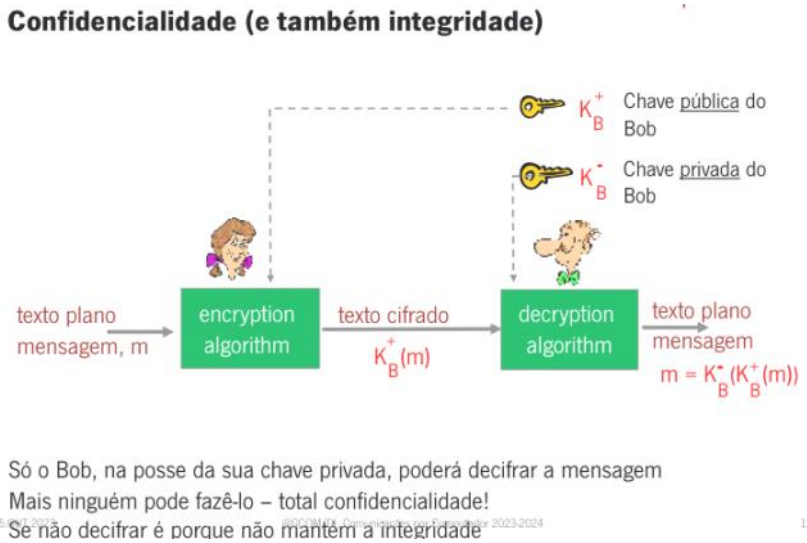
$$C_i = E_K(P_i \oplus C_{i-1}),$$
$$C_0 = IV,$$

while the mathematical formula for CBC decryption is

$$P_i = D_K(C_i) \oplus C_{i-1},$$
$$C_0 = IV.$$

Example [edit]





a mensagem é cifrada com a chave publica Kb+, e só pode ser decifrada com outra chave, a Kb-, que é a chave privada, apenas o recetor a possui (em principio)

Assim,

1. Kb-(Kb+(m)) = m
2. deverá ser impossível obter a chave privada a partir da chave pública !!

Operation [edit]

The RSA algorithm involves four steps: key generation, key distribution, encryption, and decryption.

A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d, and n, such that with modular exponentiation for all integers m (with $0 \leq m < n$):

$$(m^e)^d \equiv m \pmod{n}$$

and that knowing e and n, or even m, it can be extremely difficult to find d. Here the symbol ≡ denotes modular congruence: i.e. both $(m^e)^d$ and m have the same remainder when divided by n.

In addition, for some operations it is convenient that the order of the two exponentiations can be changed: the previous relation also implies

$$(m^d)^e \equiv m \pmod{n}.$$

RSA involves a *public key* and a *private key*. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time by using the private key. The public key is represented by the integers n and e, and the private key by the integer d (although n is also used during the decryption process, so it might be considered to be a part of the private key too). m represents the message (previously prepared with a certain technique explained below).

n, e --> public key
d --> private key
m --> message

https://en.wikipedia.org/wiki/RSA_(cryptosystem)

1. choose two large prime numbers, kept secret
2. compute n = p * q
   - *n* is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
   - *n* is released as part of the public key.
3. Compute $\lambda(n)$, where λ is Carmichael's totient function. Since $n = pq$, $\lambda(n) = \operatorname{lcm}(\lambda(p), \lambda(q))$, and since p and q are prime, $\lambda(p) = \varphi(p) = p - 1$, and likewise $\lambda(q) = q - 1$. Hence $\lambda(n) = \operatorname{lcm}(p - 1, q - 1)$.
4. Choose an integer e such that $2 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; that is, e and $\lambda(n)$ are coprime (if the only positive integer that is a divisor of both of them is 1)
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$;

Encryption [edit]

After Bob obtains Alice's public key, he can send a message M to Alice.

To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c, using Alice's public key e, corresponding to

$$c \equiv m^e \pmod{n}.$$

This can be done reasonably quickly, even for very large numbers, using modular exponentiation. Bob then transmits c to Alice. Note that at least nine values of m will yield a ciphertext c equal to m,[a] but this is very unlikely to occur in practice.

Decryption [edit]

Alice can recover m from c by using her private key exponent d by computing

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

Given m, she can recover the original message M by reversing the padding scheme.

Example [edit]

Here is an example of RSA encryption and decryption. The parameters used here are artificially small, but one can also use OpenSSL to generate and examine a real keypair.

1. Choose two distinct prime numbers, such as
   $p = 61$ and $q = 53$.
2. Compute $n = pq$ giving
   $n = 61 \times 53 = 3233$.
3. Compute the Carmichael's totient function of the product as $\lambda(n) = \operatorname{lcm}(p-1, q-1)$ giving
   $\lambda(3233) = \operatorname{lcm}(60, 52) = 780$.
4. Choose any number $2 < e < 780$ that is coprime to 780. Choosing a prime number for e leaves us only to check that e is not a divisor of 780.
   Let $e = 17$.
5. Compute d, the modular multiplicative inverse of $e \pmod{\lambda(n)}$, yielding
   $d = 413$,
   as $1 = (17 \times 413) \bmod 780$.

The **public key** is $(n = 3233, e = 17)$. For a padded plaintext message m, the encryption function is

$$\begin{aligned} c(m) &= m^e \bmod n \\ &= m^{17} \bmod 3233. \end{aligned}$$

The **private key** is $(n = 3233, d = 413)$. For an encrypted ciphertext c, the decryption function is

$$\begin{aligned} m(c) &= c^d \bmod n \\ &= c^{413} \bmod 3233. \end{aligned}$$

$c^d \bmod 3233$
↳ Very hard to find d

For instance, in order to encrypt $m = 65$, one calculates

$$c = 65^{17} \bmod 3233 = 2790.$$

To decrypt $c = 2790$, one calculates

$$m = 2790^{413} \bmod 3233 = 65.$$

A seguinte propriedade é muito útil:

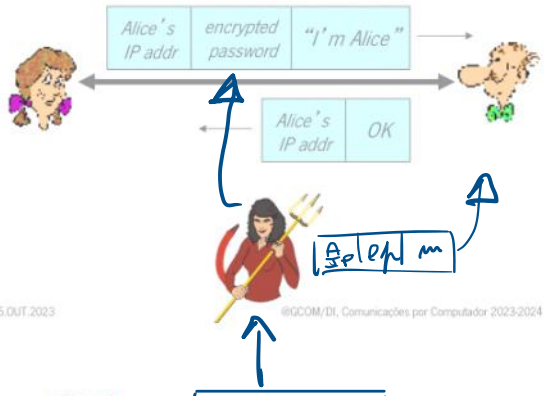
$$\underbrace{K_B^-(K_B^+(m))}_{\text{Usar a } \textit{chave pública} \text{ e depois a } \textit{privada}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{Usar a } \textit{chave privada} \text{ e depois a } \textit{pública}}$$

O resultado é o mesmo!

Propriedade matemática (explicação no livro):
- $(m^d \bmod n)^e \bmod n = m^{de} \bmod n = m^{ed} \bmod n = (m^e \bmod n)^d \bmod n$
- $n = pq$ (p e q numeros primos)

AUTENTICAÇÃO

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



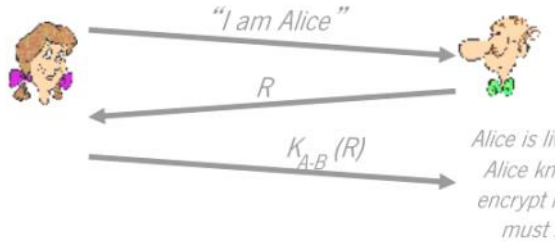
Failure scenario??

25.OUT.2023 ©GCOM/DI, Comunicações por Computador 2023-2024 19

Goal: avoid playback attack

nonce: number (R) used only once-in-a-lifetime

ap4.0: to prove Alice "live", Bob sends Alice nonce, R. Alice must return R, encrypted with shared secret key



Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

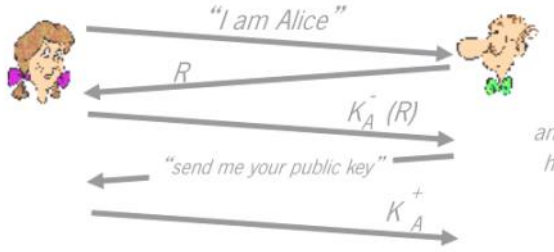
Failures, drawbacks?

23 ©GCOM/DI, Comunicações por Computador 2023-2024 20

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

ap5.0: **use nonce, public key cryptography**

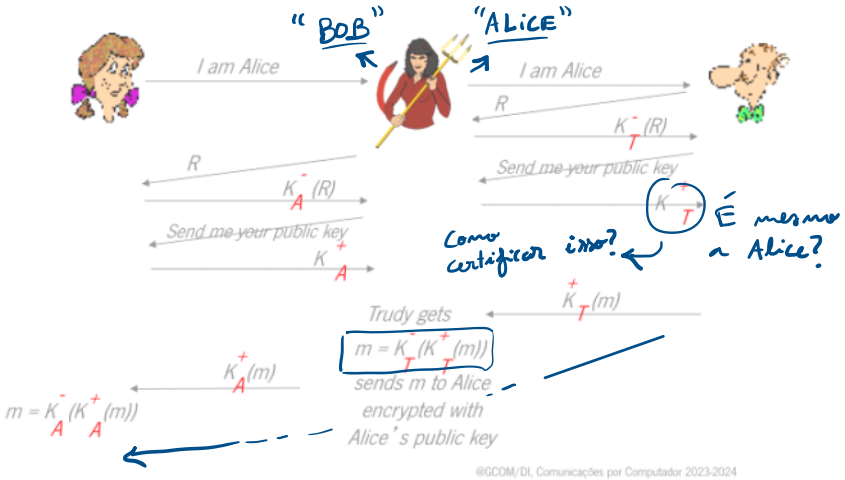


Bob computes $K_A^+(K_A^-(R)) = R$ and knows only Alice could have the private key, that encrypted R such that

$K_A^+(K_A^-(R)) = R$

Alice cifra R com a sua chave privada
Bob decifra a mensagem com a chave pública da Alice
Caso seja == R, a identidade da Alice é confirmada

man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



Infra-estrutura de chaves públicas (PKI)

Problema Chaves Simétricas:

- Como é que duas entidades estabelecem um segredo (a chave secreta) usando apenas a rede?

Solução:

- Centro de distribuição de chaves que seja de confiança e actua como intermediário entre as entidades

Problema Chaves Públicas

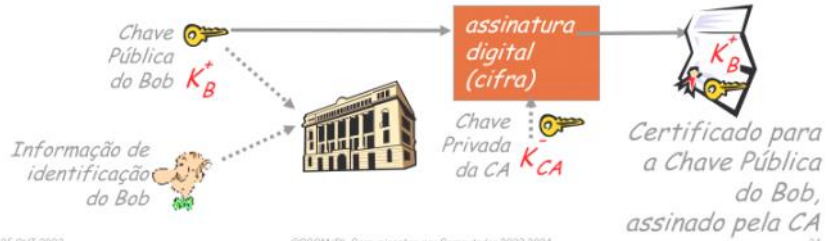
- Quando se obtém a chave pública da Alice ou do Bob na rede (e-mail, web, etc) como sabemos que são mesmo deles e não do intruso?

Solução:

- Autoridade de Certificação (CA) de confiança (*trusted certification authority*)

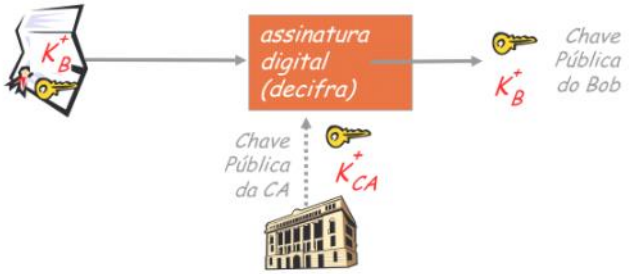
Autoridades de Certificação

- *Autoridade de Certificação (CA): associa a chave pública a uma determinada entidade, E*
- *E (pessoa, máquina,..) regista a sua chave pública na CA*
  - *E tem de fornecer uma provada de identidade a CA*
  - *CA cria um certificado digital associando E à sua chave pública*
  - *certificado contém a chave pública de E assinada digitalmente pela CA que assim assegura que "esta é a chave pública de E"*



- *Quando a Alice quer obter a chave pública do Bob:*
  - *obtem o certificado do Bob (dele mesmo ou doutros sítios).*
  - *aplica a chave pública da CA ao certificado para verificar a validade do certificado e extrair de lá a chave pública do Bob*

certificado do bob
aplicar chave publica da CA
extrair chave publica do bob



Problema: como confiar no CA?
Certificados de raiz (root certificates) instalados com as máquinas (windows, linux, etc.) - momento decisivo para a criptografia de chave publica

Assinatura Digital

Alice recebe uma mensagem do Bob, e quer garantir que:

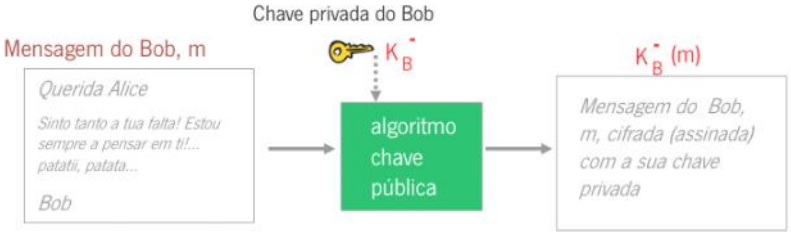
- a mensagem veio originalmente do Bob
- a mensagem não foi alterada (mantém-se íntegra) desde que foi enviada pelo Bob até ter sido lida pela Alice
- Bob não pode negar (não repúdio) que enviou a mensagem: Alice (recetor) consegue provar a qualquer um que foi o Bob que enviou e que não poderia ter sido mais ninguém, nem mesmo a própria Alice!

"Assinatura Digital"
garante integridade, autenticação do originador, não repúdio do originador

- Técnica criptográfica que se assemelha à assinatura manual...

Método 1: usando só criptografia de chave pública

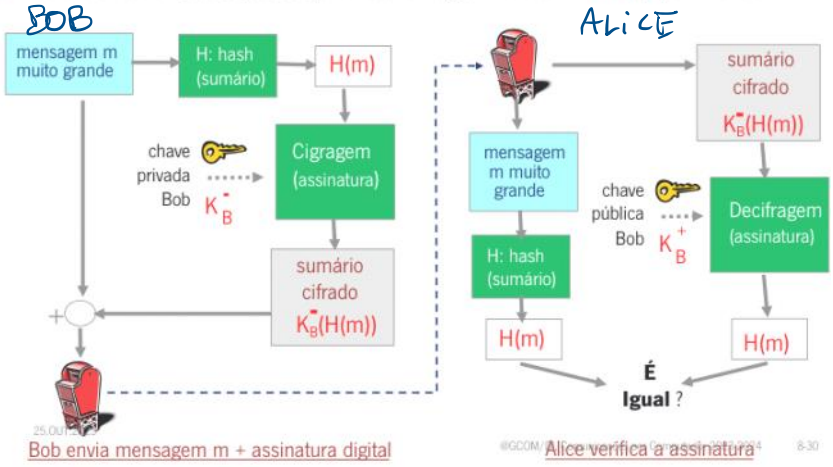
- Podemos usar a *chave privada* para assinar digitalmente a mensagem m
- Bob "assina" m cifrando-a com a sua chave privada, criando assim uma mensagem assinada K_B (m)



- Garante Autenticação do originador, não repúdio do originador e integridade
- Mas... não usado utilizado na prática por questões de desempenho! Muito lento!

Método 2: criptografia de chave pública e uso de função Hash

- Podemos usar a *chave privada* para assinar digitalmente um Hash(m) em vez de m



Bob envia mensagem m + assinatura digital

Alice verifica a assinatura

m_1 muito grande

↳ hash $H(m_2)$

↳ cifragem com K_B^-

Bob $\xrightarrow{m_1, K_B^-(H(m_2))}$ Alice

$K_B^+(K_B^-(H(m_2))) = H(m_2)$

$H(m_1)$

$H(m_1) == H(m_2)$?

↳ Sim ⇒ Assinatura comprovada

Algoritmos mais usados

- MD5 ⇝ Já não é adequado
- SHA-1, -2, -3

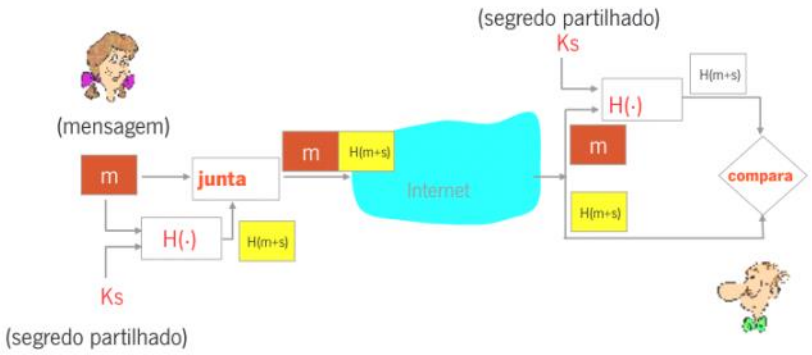
Garantias

- Só o Bob pode ter assinado m, pois só ele conhece a sua chave privada
- Mais ninguém poderia ter assinado m
- A mensagem que foi assinada foi m e não um m' qualquer
- Qualquer um pode verificar isso: basta pegar na chave pública de Bob e decifrar a assinatura
- Garante ainda o não repúdio – mesmo em tribunal! – pois Bob não poderá negar ter usado a sua chave privada

Integridade e Autenticação da Origem

MAC – Message Authentication Code

Envia o sumário da mensagem e do segredo juntos (m+s)



Usa apenas uma função de Hash criptográfica!
Não garante o não repúdio, pois se o segredo é partilhado, Alice e Bob conhecem a chave, e qualquer um deles pode produzir o Hash, não sendo por isso uma Assinatura Digital.

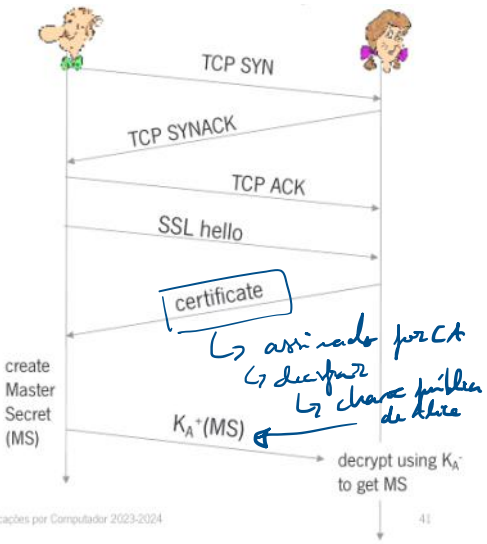
Segurança: TLS

- **Nível 4: TLS (Transport Layer Security) – muitas vezes ainda designado pelo nome inicial: Secure Sockets Layer (SSL)**
  - Segurança ao nível de transporte para qualquer aplicação TCP
  - Usado, por exemplo, no acesso a servidores HTTP, IMAP, SMTP
  - Serviços de segurança:
    - Autenticação do servidor e, opcionalmente, do cliente
    - Confidencialidade dos dados
  - **Autenticação do servidor**:
    - Cliente conhece chaves públicas de autoridades de certificação de sua confiança (CA)
    - Obtem certificado do servidor emitido por uma CA sua conhecida
    - Extrai chave pública do certificado depois de verificada validade
- **Nível 4: TLS (Transport Layer Security)**
  - **Confidencialidade (cifragem dos dados da sessão)**
    - Cliente gera chave de sessão, cifra-a com a chave pública do servidor e envia-a ao servidor
    - Servidor decifra a chave de sessão usando a sua chave privada
    - Ambos – cliente e servidor – na posse da chave de sessão, podem cifrar todos os dados trocados...
  - **Autenticação do cliente** pode ser feita com base em certificados do cliente
    - Opcional, porque a maioria dos clientes não possui chave pública e chave privada com certificado verificável

1. *Handshake* inicial:

- Bob estabelece conexão TCP com Alice
- Autentica Alice usando o certificado assinado por uma CA
- Cria chave mestra, encripta-a (usando a chave pública da Alice), e envia-a à Alice

Incompleto: a troca de um "nonce" não está ilustrada



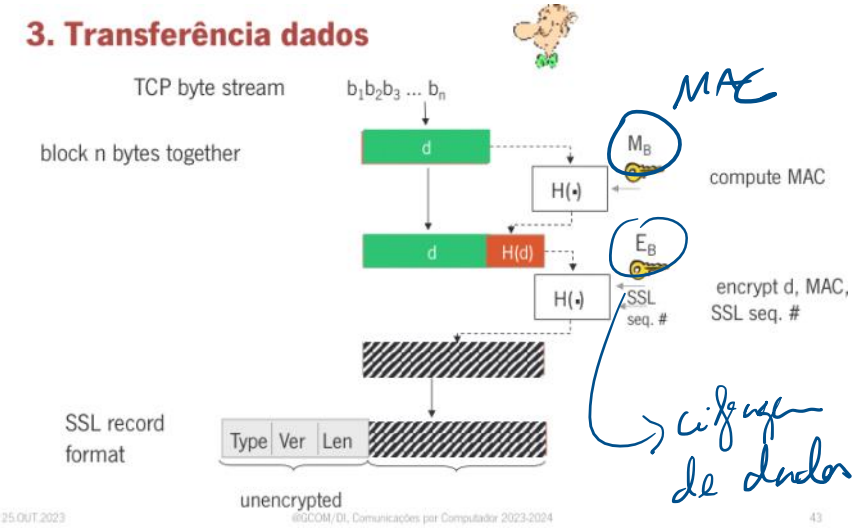
2. Cálculo das chaves:

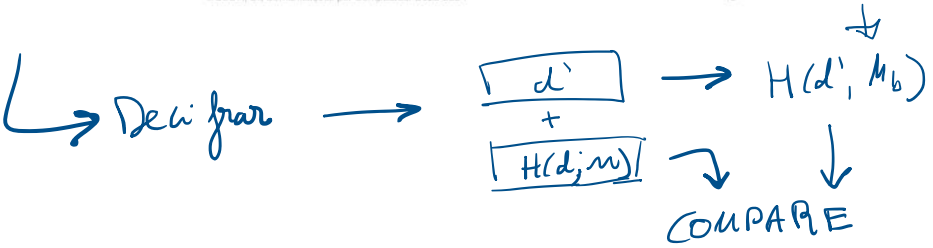
- Alice e Bob usam a chave mestra (MS) para gerar 4 chaves:
  - $\mathbf{E_B}$: *Bob->Alice* chave de cifragem de dados
  - $\mathbf{E_A}$: *Alice->Bob* chave de cifragem de dados
  - $\mathbf{M_B}$: *Bob->Alice* chave MAC (*Message Authentication Code*)
  - $\mathbf{M_A}$: *Alice->Bob* chave MAC (*Message Authentication Code*)
- Os algoritmos (cifragem e MAC) são negociados entre a Alice e o Bob
- Porquê 4 chaves?

1. **Chave de Cifragem de Dados de Bob para Alice (EB):**
   - Esta chave é utilizada por Bob para cifrar os dados que serão enviados para Alice. A cifragem de dados visa garantir a confidencialidade da informação durante a transmissão. A chave de cifragem (EB) é derivada da chave mestra.
2. **Chave de Cifragem de Dados de Alice para Bob (EA):**
   - Similar à chave EB, mas destinada a cifrar os dados que Alice envia para Bob. Também tem como objetivo garantir a confidencialidade dos dados durante a transmissão. A chave de cifragem (EA) é derivada da chave mestra.
3. **Chave MAC de Bob para Alice (MB):**
   - Esta chave é utilizada por Bob para gerar um código de autenticação de mensagem (MAC) que é anexado aos dados enviados para Alice. A MAC serve para garantir a integridade dos dados e autenticar a origem. A chave MAC (MB) é derivada da chave mestra.
4. **Chave MAC de Alice para Bob (MA):**
   - Similar à chave MB, mas destinada a Alice para gerar um MAC para os dados que ela envia para Bob. A chave MAC (MA) é derivada da chave mestra.

Essa abordagem de usar chaves distintas para cifragem e autenticação tem a vantagem de proporcionar uma separação clara de funções. A confidencialidade dos dados é assegurada pela cifragem, enquanto a integridade e autenticação são garantidas pelo uso de códigos MAC. Separar essas funções em chaves distintas aumenta a segurança do sistema e ajuda a evitar problemas que podem surgir ao usar

3. Transferência dados

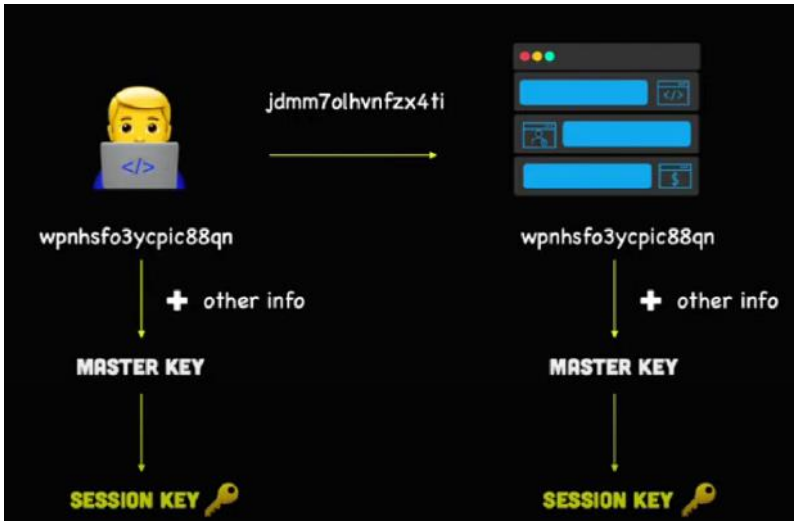




Exemplo TLS1.0-1.2/SSL3.0: 3 fases

- **Detalhadamente:**
  1. Cliente envia "**nonce**" e lista de algoritmos criptográficos que suporta
  2. Servidor escolhe da lista um algoritmo simétrico (ex: AES), um algoritmo de chave pública (ex: RSA), um algoritmo MAC (ex: HMAC_SHA256, HMAC com SHA-256), e envia ao cliente as suas escolhas, com o seu "nonce" e o seu certificado de chave pública
  3. Cliente verifica o certificado, extrai chave pública do servidor, gera um PMS (Pré-Master Secret), cifra PMS com chev pública do servidor, e envia ao servidor
  4. [paralelo] Cliente e servidor calcula de forma independente o MS (Master Secret) a partir do PMS e dos "nonces". MS é fatiado para gerar 4 chaves. Se for usado um algoritmo CBC, derivam-se também dois vetores de inicialização (um em cada sentido). Mensagens entre cliente e servidor são confidenciais e autenticadas
  5. Cliente envia um MAC de todas as mensagens de Handshake (*)
  6. Servidor envia um MAC de todas as mensagens de Handshake (*)

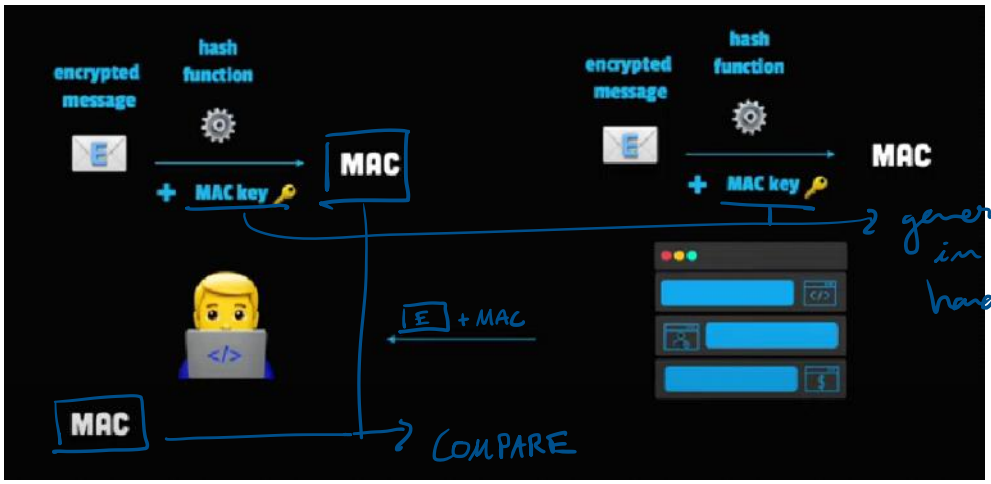
(*) evita "tampering" da lista de algoritmos, por exemplo!



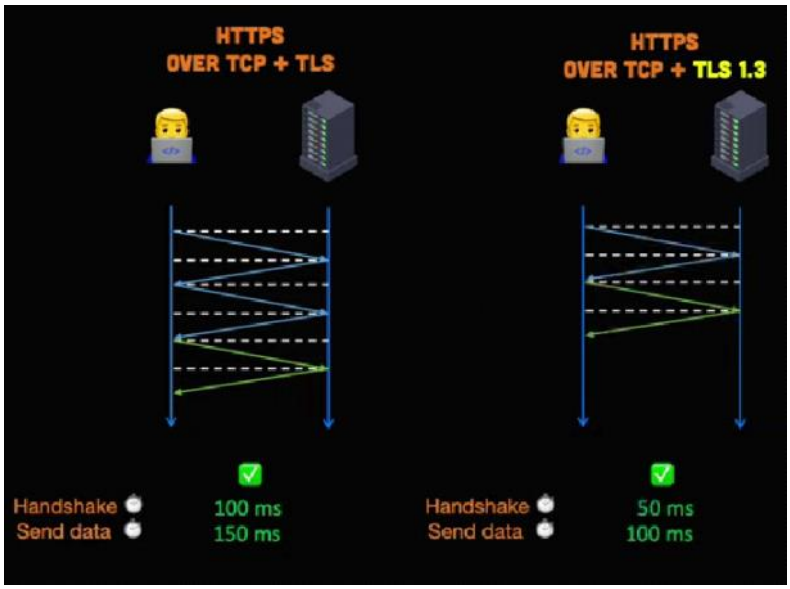
How TLS Works?

→ More efficient
⇒ FASTER

MAC



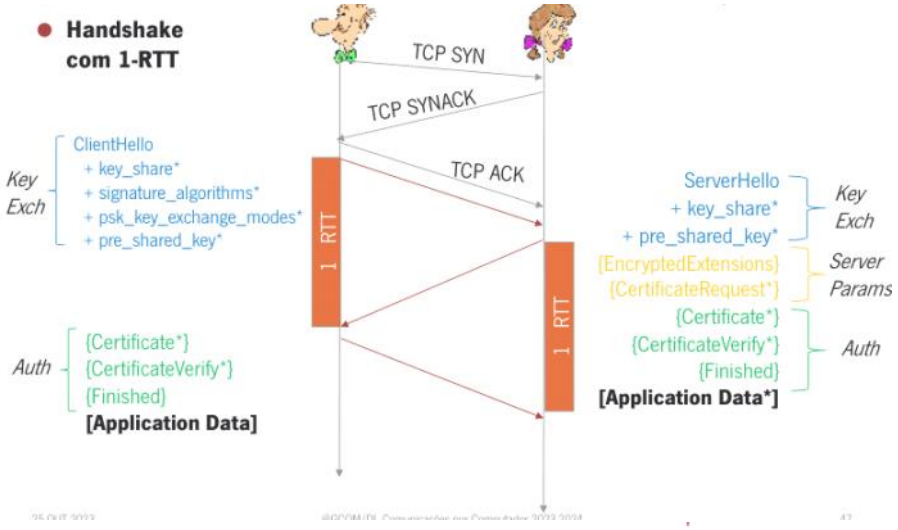
generated in the handshake



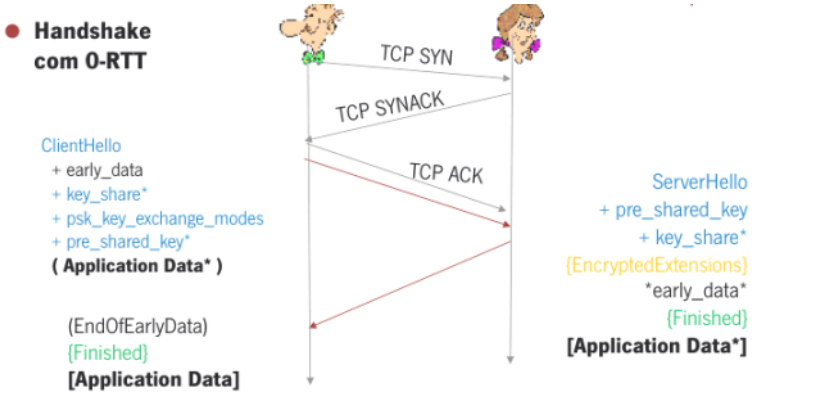
Session resumption

Segurança: TLS 1.3 (2020)

- **Objetivos principais**
  - Segurança melhorada → falhas detetadas, cifras obsoletas, etc.
  - Melhoria de desempenho → Reduzir o *handshake* inicial a 1-RTT ou 0-RTT
- **Principais diferenças para o TLS 1.2**
  - Remover mecanismos e algoritmos obsoletos (Hash: MD5, SHA-1; Simétricos: DES, 3DES, AES-CBC, RC4; chave pública: RSA estático, Diffie-Helman estático...)
  - Todas as mensagens de handshake a seguir ao ServerHello são cifradas
  - Novas funções para derivar chaves
  - Reduzir o handshake inicial (evitar a negociação do primeiro RTT)
  - Retoma de uma sessão com ou sem manutenção de estado do lado do servidor
  - Permitir um modo especial com 0 – RTT !!



- **Certificate**
  - Certificado de chave pública da entidade
- **CertificateVerify**
  - Assinatura digital sobre todo o *handshake*, feita com a chave privada associada à chave publica do certificado.
- **Finished**
  - MAC (Message Authentication Code) sobre todo o handshake que confirma as chaves e algoritmos usados
- **EncryptedExtensions**
  - Primeira mensagem cifrada, com as chaves deduzidas, e contendo lista de extensões
- **CertificateRequest**
  - Pedido de certificado do cliente



Quando cliente e servidor partilham uma Pre-SharedKey, obtida externamente ou num handshake anterior, os clientes podem enviar dados no primeiro segmento (extensão *early_data*), usando o PSK para autenticar o servidor e encriptar os dados. Semelhante à retomada de uma sessão