# Resumo de Criptografia



julho de 2016

# Sumário

1. Introdução
2. Encriptação em bloco
3. Modos de operação
4. Funções Hash
5. Autenticação (*Message Authentication Codes*)
6. O canal seguro
7. Chaves públicas: Diffie-Hellman
8. Protocolos criptográficos
9. Servidores de chaves
10. Certificados e PKI
11. Protocolos de rede

(Total de 210 slides)

# 1. Introdução

## Motivação

• Nos primórdios das redes de computadores praticamente não haviam problemas de segurança.

• Atualmente segurança é um problema crítico.

• A segurança atualmente é vista por vários ângulos diferentes:

Político, Administrativo, Gerencial, Tecnológico, Pessoal, ...

# 1. Introdução

## Infraestrutura

Usuários comuns (residenciais) de energia elétrica assumem que esta infraestrutura é uma facilidade básica e óbvia.

Usuários industriais de energia elétrica não assumem isto!

Usuários comuns de combustíveis (gasolina, alcool) também assumem tacitamente esta infraestrutura.

Usuários industriais não!

# 1. Introdução

Usuários residenciais de computadores e da Internet tentam assumir que esta (nova) tecnologia é uma infraestrutura básica da vida moderna, como outros eletrodomésticos. Os fabricantes de dispositivos também !

Usuários de maior porte, comerciais e industriais, sabem que têm que ter departamentos técnicos ou terceirizar este serviço.

# 1. Introdução

## Necessidade de uma infraestrutura de segurança

A segurança de computadores é muito mais complexa do que simplesmente usar computadores satisfatoriamente.

- Ela possui uma parte tecnológica complexa.
- Assim como em outros ramos da segurança, ela possui uma parte humana e pessoal muito complexa.

**Isto exige uma infraestrutura específica de segurança.**

# 1. Introdução

Uma infraestrutura de segurança de computadores exige um investimento específico, um planejamento, uma gerência e uma metodologia bem definida.

Além disto, as necessidades, as ameaças e as medidas de proteção não são as mesmas para todos.

**Não existe padronização das medidas de segurança !**

# 1. Introdução

Conclusão:

Este curso não pode dar uma receita para implementar segurança de computadores em uma organização...

Mas uma escola de medicina também não dá receitas para seus alunos tratarem os futuros pacientes (ainda bem!). Nem as escolas de engenharia ou de administração.

Mas existe um corpo de conhecimento sobre o assunto!

# 1. Introdução

Ao longo deste curso, veremos muitas referências a livros, artigos, *sites*, etc.

Alguns são bons, outros médios, alguns ruins...

Para começar, primeiro o melhor:

**An Introduction to Computer Security: The NIST Handbook**

National Institute of Standards and Technology

Technology Administration

U.S. Department of Commerce

# 1. Introdução

Apesar de ser uma publicação de outubro de 1995, ainda é a publicação atual do NIST.

O manual coloca o problema da segurança de computadores no contexto correto de **dar suporte à missão da organização**.

Neste contexto, trata-se mais de uma atividade de administração da organização.

# 1. Introdução

O manual do NIST não é um manual técnico de computação ou criptografia. **É um manual de administração!**

Ele deve ser lido como tal, por uma pessoa com conhecimento de administração!

Existe uma outra publicação mais nova, um pouco diferente:

**Pauline Bowen, Joan Hash, Mark Wilson, "Information Security Handbook: A Guide for Managers", NIST Special Publication 800-100, National Institute of Standards and Technology, 2006.**

# 1. Introdução

Em segundo lugar, existem os aspectos tecnológicos da segurança de computadores.

Existem livros que são tecnicamente precisos, profundos e datalhados. Mas podem ser muito difíceis de ler, e.g: Oded Godreich (http://www.wisdom.weizmann.ac.il/~oded/)

# 1. Introdução

Mas existem livros muito bons de serem lidos e com um nível matemático compreensível, sem serem absolutamente superficiais.

Meu preferido:

**Niels Fergusson & Bruce Schneier, "Practical cryptography", Wiley, 2003.**

Além de cobrir o assunto de criptografia, este livro transmite de uma maneira muito inspirada como funciona a mente de um profissional da área de segurança de computadores!

# 1. Introdução

Também existe uma segunda edição, revisada e aumentada, deste livro:

**Niels Ferguson, Bruce Schneier, Tadayoshi Kohno, "Cryptography Engineering: Design Principles and Practical Applications", John Wiley & Sons, 2010.**

# 1. Introdução

Um outro livro texto, bastante completo, e que aconselho, é:

**Jie Wang, "Computer network security: theory and practice", Higher Education Press, Beijing e Springer-Verlag GmbH, Berlin Heidelberg, 2009.**

# 1. Introdução

Podemos finalmente apresentar o modelo básico de segurança, que consiste de quatro componentes principais:

- Criptosistemas,
- Firewalls,
- Programas antimaliciosos (antivirus et al),
- Sistemas de deteção de intrusões.

Neste curso, veremos basicamente aspectos de criptosistemas.

# 2. Encriptação em bloco

**"A encriptação é o objetivo original da criptografia"** (segundo Fergusson & Schneier).

Também, segundo os mesmos autores, é uma tradição o uso de nomes próprios para os papéis nas transações criptográficas:

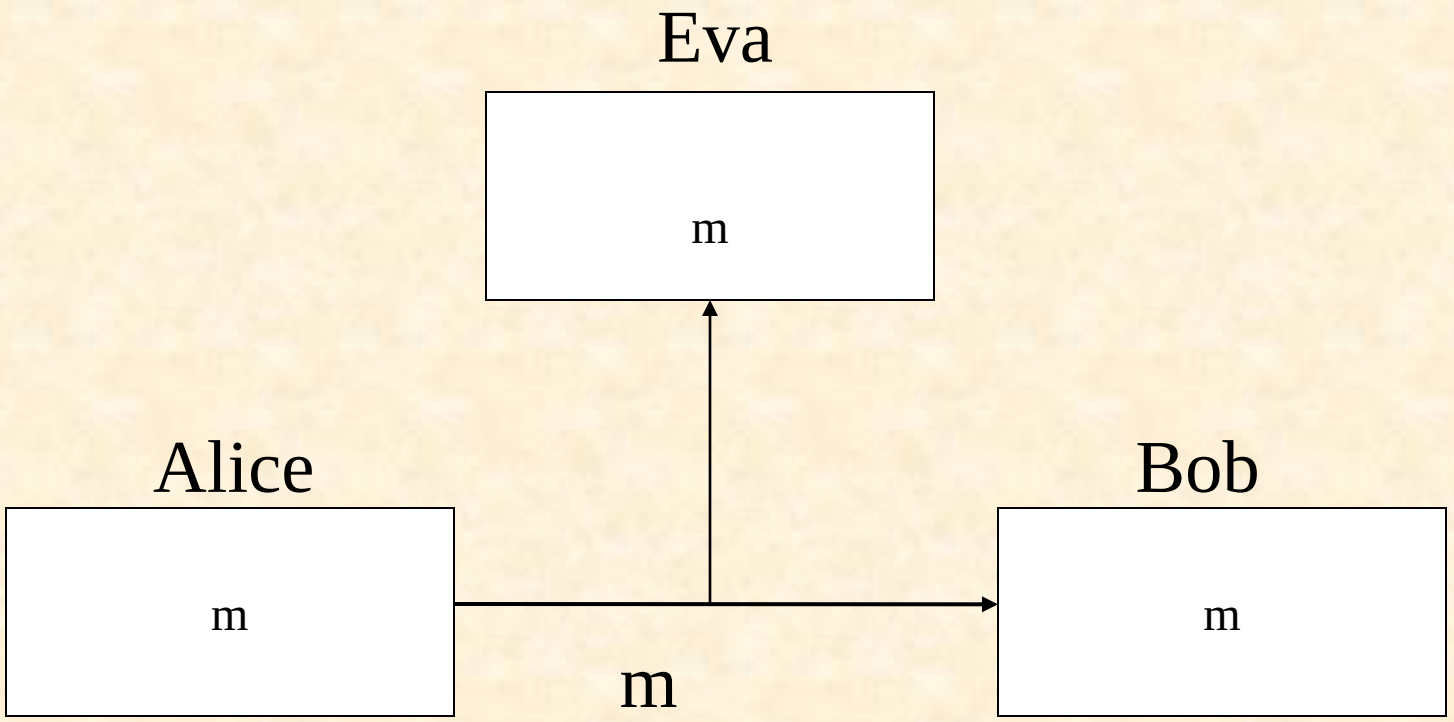
**A**: Alice – o iniciador ou transmissor

**B**: Bob – o ouvinte ou receptor

**C**: Charlie (ou Eva / *Enemy*) – o espião, atacante ou adversário

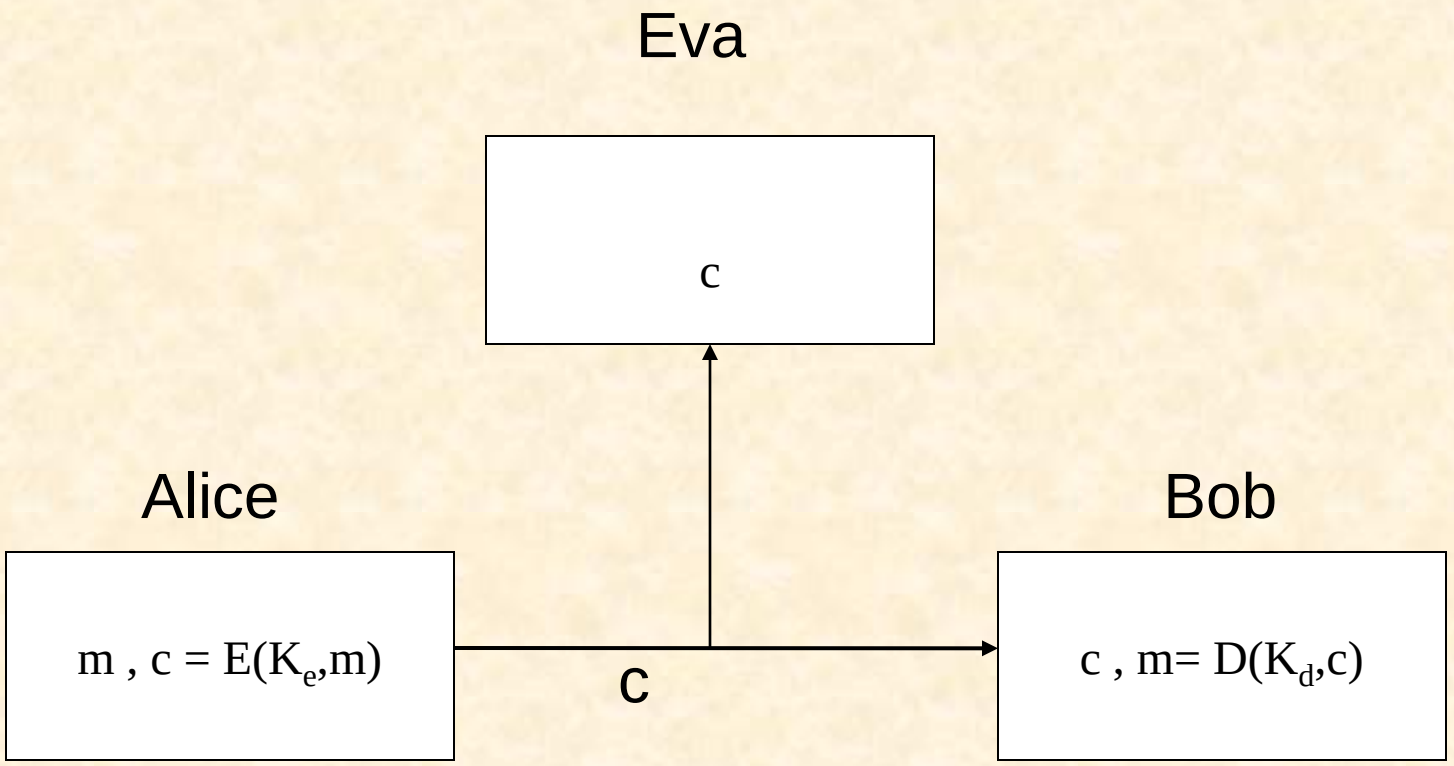
Na verdade, Alice e Bob são só papéis. Um agente real frequentemente assume os dois papéis ao longo do tempo.

# 2. Encriptação em bloco



# 2. Encriptação em bloco

Para evitar a espionagem de Eva, Alice e Bob trocam mensagens cifradas, ou encriptadas.



# 2. Encriptação em bloco

A função de encriptação

$$c = E(K_e, m)$$

gera um texto encriptado, ou cifrado, **c** (*cyphertext*) a partir do texto limpo **m** (*plaintext*) e de uma chave de encriptação $\mathbf{K_e}$.

Obviamente, supomos que o texto cifrado é suficientemente ininteligível para que Eva não consiga recuperar o texto original a partir do texto cifrado.

# 2. Encriptação em bloco

A função de decriptação

$$m = D(K_d, c)$$

regenera o texto limpo original **m**, a partir do texto cifrado **c** e de uma chave de decriptação $\mathbf{K_d}$.

As chaves de encriptação e decriptação podem ser a mesma ou serem chaves diferentes.

As funções E e D possuem implementações que são descritas por um algoritmo.

# 2. Encriptação em bloco

**“Princípio de Kerckhoff”**

Eva precisaria saber duas coisas para recuperar o texto original:

- O algoritmo de decriptação,
- A chave de decriptação $K_d$.

O princípio de Kerckhoff diz que a segurança do sistema de encriptação deve depender **somente** do segredo das chaves e **NÃO**(!) deve depender do segredo do algoritmo de decriptação.

# 2. Encriptação em bloco

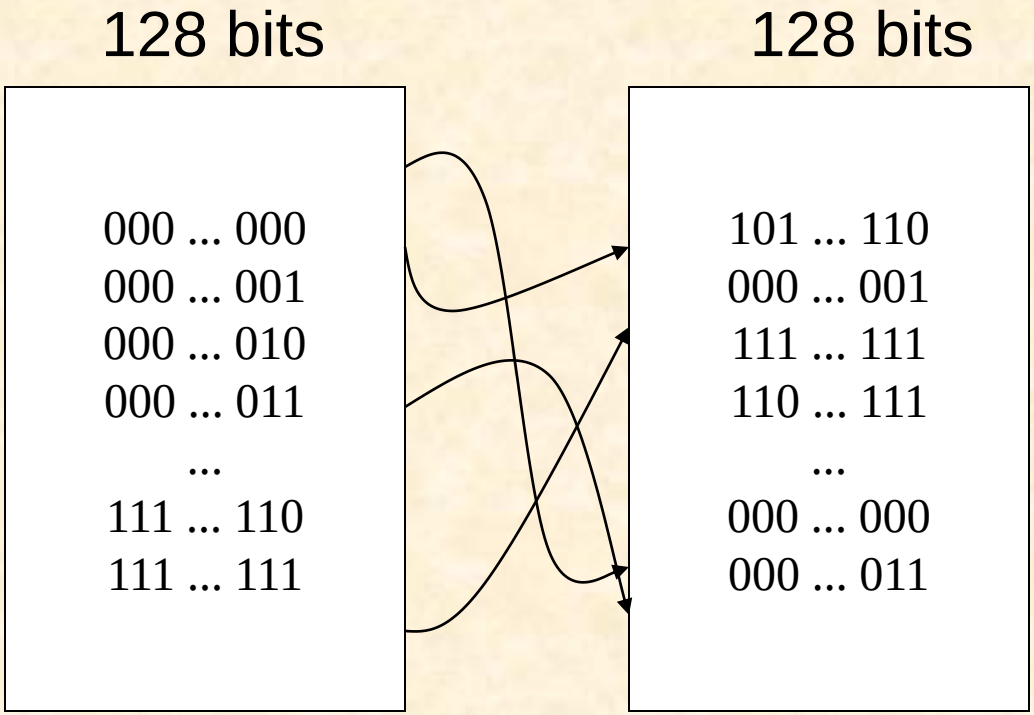
O primeiro esquema de encriptação que veremos é a encriptação em bloco (*Block cyphers* – criptografia plana ou criptografia simétrica).

A criptografia em bloco possui funções de encriptação e decriptação para textos que têm um tamanho fixo: o tamanho do bloco.

Atualmente utilizam-se blocos de 128 bits (16 bytes).

# 2. Encriptação em bloco

A idéia de encriptação em bloco pode ser mostrada através duas tabelas de mesmo tamanho:




permutações: $2^{128}$ !

# 2. Encriptação em bloco

Para um tamanho de bloco de 64 bits, o tamanho da tabela (com os dados) seria de 150 milhões de Terabytes.

Para um tamanho de bloco de 128 bits (16 bytes) o tamanho da tabela seria de aproximadamente $5 \times 10^{39}$ bytes.

Das $2^{128}$ ! Permutações, só estamos interessados nas permutações que são, num certo sentido, aleatórias.

# 2. Encriptação em bloco

Mesmo que somente 0,001% do total de permutações sejam aleatórias, o número ainda é absurdamente grande.

A chave pode ser um número de 128 bits (ou 256) que é utilizado pelos algoritmos de encriptação e decriptação para gerar uma linha de uma das tabelas, para uma dada permutação.

A chave pode ser vista como um identificador da permutação.

# 2. Encriptação em bloco

Para uma chave de 128 bits, existem $2^{128} = 3,4 \times 10^{38}$ permutações correspondentes (chaves possíveis).

Um número insignificante comparado com o número total de permutações das linhas da tabela (que é o fatorial disto).

# 2. Encriptação em bloco

Para efeito de comparação, vamos imaginar um bloco de 8 bits, um único byte. O conteúdo de tal bloco poderia ser, por exemplo, um código ASCII estendido.

Nossa tabela tem então somente 256 linhas.

Desta existem $256! = 8{,}6 \times 10^{506}$ permutações (um número impressionante!).

Uma chave de 64 bits pode identificar somente $2^{64} = 1{,}8 \times 10^{19}$ permutações destas. Isto é uma fração completamente desprezível, que pode corresponder somente as permutações realmente aleatórias.

# 2. Encriptação em bloco

O primeiro método de criptografia moderno (pós-guerra) é o DES (Data Encryption Standard).

(ver http://pt.wikipedia.org/wiki/Data_Encryption_Standard)

Trata-se de um método selecionado pelo governo dos EUA em 1976 que foi (e ainda é) largamente utilizado no mundo inteiro.

Em janeiro de 1999, a *distributed.net* e a *Electronic Frontier Foundation* juntas violaram uma chave DES em 22 horas e 15 minutos.

# 2. Encriptação em bloco

Uma variante, mais segura, chamada DESede (ou DES-EDE, 3DES ou Triple-DES) ainda se encontra em uso até hoje.

# 2. Encriptação em bloco

Em novembro de 2001, o NIST (National Institute of Standards and Technology) anunciou um novo padrão chamado AES (Advanced Encryption Standard).

Ver http://pt.wikipedia.org/wiki/AES

# 2. Encriptação em bloco

Existem outros problemas básicos que Alice e Bob têm que enfrentar, que são:

- Autenticação

- Criptografia com chave pública

- Assinatura digital

- PKI – *Public Key Infrastructure*

# 2. Encriptação em bloco

## Autenticação

O inimigo, Eva, pode não só espionar a comunicação entre Alice e Bob, como também pode tentar alterar as mensagens.

Para resolver este problema existe a Autenticação.

Quando Alice envia uma mensagem, ela calcula e também envia um código de autenticação da mensagem (*Message Authentication Code* - MAC).

O MAC tem um comprimento fixo. Ele é calculado a partir da mensagem m e de uma chave criptográfica de autenticação, Ka: a = h(Ka,m)

# 2. Encriptação em bloco

## Autenticação

Uma boa função de autenticação produz sempre um MAC diferente, de tamanho fixo, para todas as mensagens, de diferentes comprimentos.

Além disto, esta função não pode ser invertida.

Bob também possui a chave criptográfica de autenticação, que ele vai usar para recalcular o MAC, a partir da mensagem recebida.

Para evitar que Eva reenvie uma mensagem antiga com o respectivo MAC, utiliza-se também um esquema de numeração de mensagens.

# 2. Encriptação em bloco

## Criptografia com chave pública

A distribuição de chaves secretas (simétricas) é um problema difícil que foi resolvido com o desenvolvimento da criptografia com chaves públicas e privadas.

O problema é que a criptografia com chaves públicas-privadas é muito mais lenta do que a criptografia com chaves secretas (simétricas).

Por isto, a criptografia com chaves públicas é usada para distribuir as chaves secretas, que então são usadas para criptografar e autenticar as mensagens.

# 2. Encriptação em bloco

## Assinatura digital

A assinatura digital é a versão baseada em criptografia pública-privada da autenticação de mensagens feita com chaves secretas.

Como normalmente qualquer um pode ter a chave pública de Alice para verificar que esta assinou a mensagem com sua chave privada, esta assinatura serviria como um forma de garantir a não repudiação.

Infelizmente, devido à possibilidade de vírus e invasões, não se pode garantir que Alice realmente tenha assinado a mensagem (para fins legais).

# 2. Encriptação em bloco

## PKI – *Public Key Infrastructure*

A idéia de ter uma infraestrutura para distribuir com segurança as chaves públicas levou ao desenvolvimento de formas de PKI e das Autoridades de Certificados (*Certificate Authority* - CA).

Apesar da idéia funcionar em várias situações, um dos principais problemas é o da responsabilidade financeira.

Por exemplo, a VeriSign limita-se na maioria dos casos a uma responsabilidade de até US$100.00.

Isto pode ser suficiente para garantir a compra de um livro, mas não para a compra de um carro.

# 3. Modos de operação

A cifragem básica que vimos encripta blocos de tamanho fixo (e relativamente pequenos).

Para encriptar dados de maior tamanho, deve-se usar modos de operação da encriptação em blocos que mantenham a confidencialidade dos dados.

O primeiro problema é o de encriptar dados que não sejam múltiplos do tamanho do bloco.

Para isto, precisa-se dividir o conjunto de dados em blocos e completar o último bloco de maneira a poder encriptar e descriptar de forma unívoca.

# 3. Modos de operação

## Complementação (*Padding*)

Para garantir que a encriptação-descriptação seja unívoca, não basta preencher o último bloco com um valor escolhido.

Um método simples sugere que se adicione um bit 1 depois do dado do último bloco (incompleto) e que depois se adicione tantos zeros quantos necessários para prenche-lo.

Se o último bloco estiver completamente cheio, um novo bloco, só com o complemento, deve ser adicionado.

# 3. Modos de operação

## ECB – *Electronic codebook*

Supondo que o último bloco já foi completado, o modo ECB diz que os blocos devem ser encriptados sequencialmente com a chave secreta.

Este modo é bastante inseguro, pois dados repetidos serão encriptados igualmente, dando ao atacante esta informação.

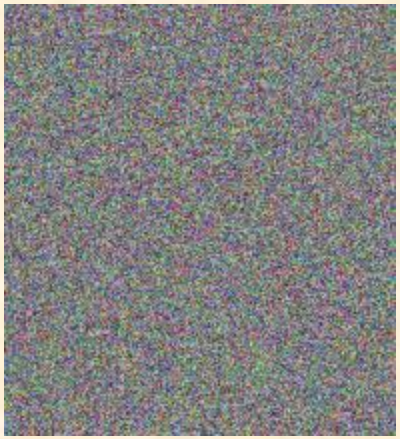
Um exemplo gráfico marcante desta fraqueza pode ser visto com uma imagem encriptada por ECB e encriptada por outro modo, como mostrado a seguir:

# 3. Modos de operação

Figura original

ECB

Outros modos



# 3. Modos de operação

## CBC – *Cipher Block Chaining*

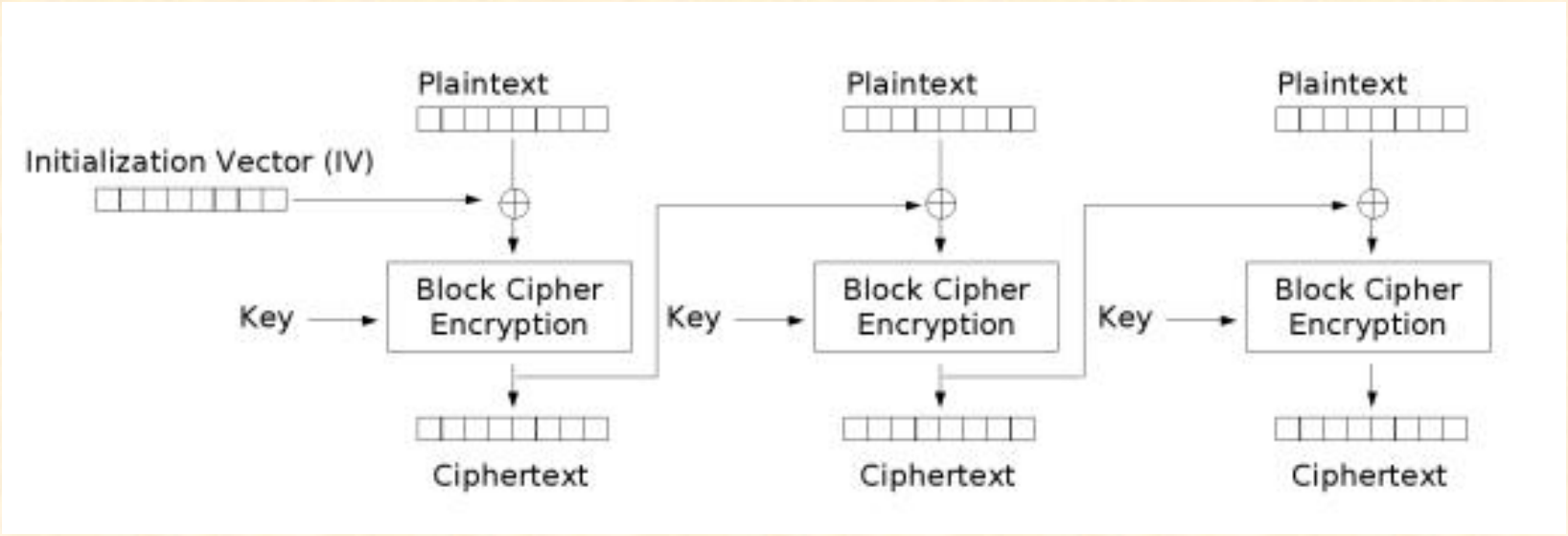
Neste modo, cada um dos k blocos de texto limpo é combinado por ou-exclusivo com o bloco cifrado anterior antes de ser cifrado:

$$Ci = E( K , P_i \oplus C_{i-1} ) \text{ para } i = 1, 2, 3, \ldots , k$$

O problema agora é determinar um valor para $C_0$, chamado de "Vetor de Iniciação" (IV – *Initialization Vector*).

Não se deve iniciar todas as mensagens com o mesmo IV, pois isto daria uma informação ao atacante.

# 3. Modos de operação



# 3. Modos de operação

Uma das maneiras de garantir uma melhor confidencialidade ao sistema é utilizar na geração do IV um número que só se use, garantidamente, uma única vez, em todo o sistema: um **nonce** – *number used once*.

O IV é gerado encriptando-se o *nonce* com a criptografia de bloco e a mesma chave do resto da mensagem.

O *nonce* é frequentemente gerado por um sistema de numeração de mensagens do próprio sistema.

# 3. Modos de operação

## OFB – *Output FeedBack*

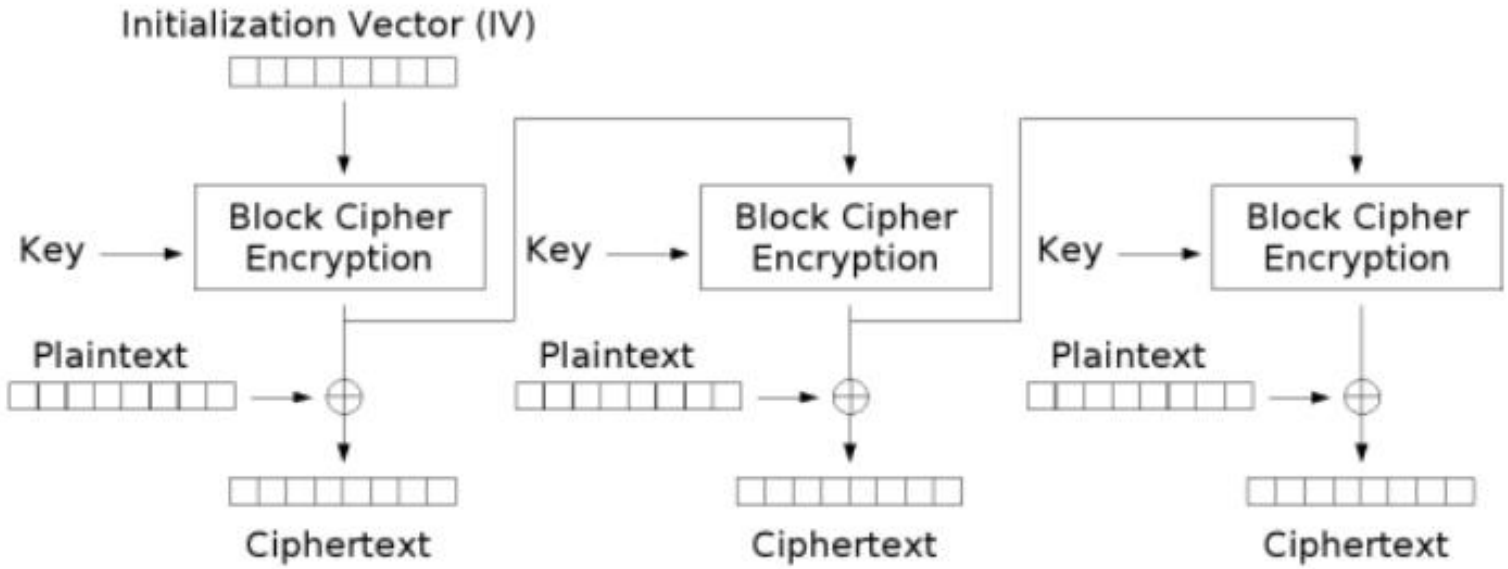
Este modo é diferente do anterior porque os dados não são criptografados  com a criptografia em bloco.

Ao invés disto, o IV é encriptado para gerar uma criptograma que é usado em um ou-exclusivo com o texto limpo para gerar o texto cifrado.

Além disto, o primeiro criptograma é usado como entrada para o próximo passo de encriptação e gerar um novo criptograma, como mostrado a seguir:

# 3. Modos de operação

$$C_i = P_i \oplus O_i$$
$$P_i = C_i \oplus O_i$$
$$O_i = E_K(O_{i-1})$$
$$O_0 = \mathrm{IV}$$




Output Feedback (OFB) mode encryption

# 3. Modos de operação

Modos deste tipo são chamados de modos de fluxo (*Stream modes*), pois não precisam de complementação (*padding*). Eles geram um número de bits cifrados exatamente igual ao número de bits do texto limpo.

Além disto, a decriptação é feita pelo mesmo algoritmo, o que economiza na sua implementação em hardware ou em software.

# 3. Modos de operação

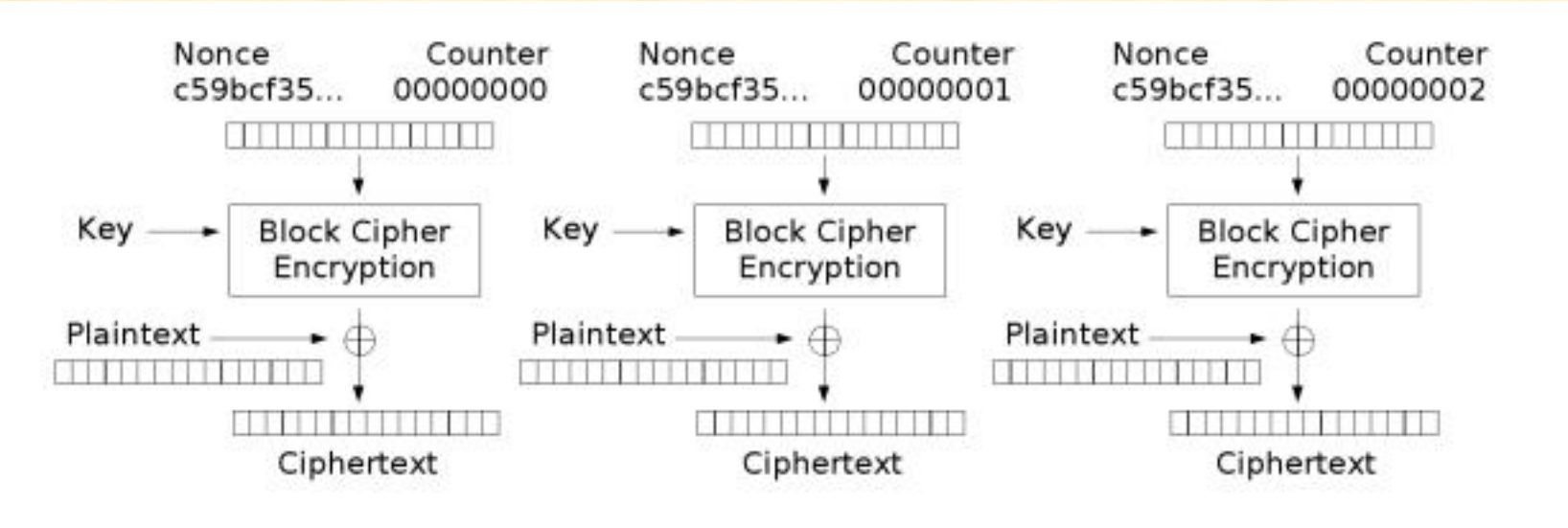
## CTR – *Counter*

Este modo é semelhante ao OFB, sendo um modo stream.

A diferença é que o valor usado para gerar o criptograma que vai ser combinado (XOR) com o texto limpo é formado por um *nonce* concatenado com um contador.

Um esquema deste modo é mostrado a seguir:

# 3. Modos de operação



# 4. Funções Hash

Uma função Hash é uma função que mapeia uma entrada de comprimento arbitrário (bits, bytes ou caracteres) em um resultado de tamanho fixo.

Atualmente, o resultado das funções Hash mais usadas estão entre 128 e 512 bits.

Funções Hash também são chamadas *message digest* e o resultado é chamado de *digest* ou *fingerprint*.

# 4. Funções Hash

Usaremos a notação:

$$h(m) = x$$

onde m é a mensagem, de comprimento arbitrário (porém finito) e x é o resultado, de comprimento fixo.

Nosso interesse é nas chamadas funções Hash criptográficas.

A diferença entre as funções Hash criptográficas e as funções Hash “comuns” está nos requisitos adicionais.

# 4. Funções Hash

O primeiro requisito é que ela seja uma função não inversível, ou “unidirecional”.

Isto quer dizer que é facil (e eficiente), dada a mensagem m, calcular h(m).

Por outro lado, dado x, tal que x = h(m), não deve ser possível encontrar um m tal que

x = h(m).

# 4. Funções Hash

O segundo requisito é que uma função Hash criptográfica deve ter é a “resistência a colisões”.

Embora exista uma infinidade de colisões possíveis, já que existe uma infinidade de mensagens para um número finito de resultados, elas devem ser praticamente impossíveis de serem encontradas.

# 4. Funções Hash

Quer dizer, dado um vetor (de bits) $m_1$ tal que length($m_1$) ≤ M, é computacionalmente intratável (tempo proporcional a exp($m_1$) ) achar um outro vetor $m_2$, com length($m_2$) ≤ M também), achar um outro vetor $m_2$, com length($m_2$) ≤ M, tal que

$h(m_1) = h(m_2)$.

# 4. Funções Hash

O terceiro requisito é que que uma função Hash criptográfica deve ter é a “resistência **forte** a colisões”.

Neste caso, dado um vetor (de bits) $m_1$ tal que length($m_1$) ≤ M, é computacionalmente intratável, com tempo proporcional a exp(length($m_1$)+length($m_2$)), achar um outro vetor $m_2$, com length($m_2$) ≤ M, tal que

$h(m_1) = h(m_2)$.

# 4. Funções Hash

O quarto requisito é que uma função Hash criptográfica deve ter é de ser uma função aleatória.

Isto significa que o resultado não deve dar nenhuma informação sobre as mensagens às quais a função for aplicada.

# 4. Funções Hash

Apesar de parecer a primeira vista que o problema de se definir uma implementação para funções Hash seja mais fácil que a implementação de encriptação em blocos (chave simétrica secreta), na verdade é o contrário.

Existe muito mais estudo e conhecimento sobre a encriptação em blocos do que sobre funções Hash criptográficas.

# 4. Funções Hash

A função Hash criptográfica ideal atende a estes quatro critérios.

Obviamente esta NÃO é uma definição formal.

Assim como no caso da encriptação em bloco, também supomos que seremos capazes de reconhecer uma função não ideal quando a virmos.

# 4. Funções Hash

Definimos então um ataque a uma função Hash como um método não trivial de distinguir uma função Hash dada de uma função Hash ideal.

O ataque típico a uma função Hash é o ataque de aniversário (ou qualquer outro de colisões).

Para uma função com resultado de n bits, o atacante poderia gerar $2^{n/2}$ resultados e testar colisões.

# 4. Funções Hash

As funções Hash mais utilizadas são:

- MD5
- SHA-1
- SHA-2

A função MD5 é a mais antiga e já foram identificadas fraquezas nela. Não é uma função aconselhada, embora ainda seja largamente utilizada.

É uma função de 128 bits.

# 4. Funções Hash

A função SHA-1 (*Secure Hash Algorithm*) foi criada pela NSA e padronizada pelo NIST.

Ela veio substituir a SHA (ou SHA-0) e reparar uma fraqueza que foi identificada.

É uma função de 160 bits, supostamente mais segura que a MD5.

Entretanto também foram encontradas fraquezas nela, que levaram a criação do novo padrão.

# 4. Funções Hash

A SHA-2 é um conjunto de 3 funções:

- SHA-256, uma função de 256 bits,
- SHA-384, uma função de 384 bits,
- SHA-512, uma função de 512 bits.

Entretanto, o NIST iniciou em 2006 o processo de definição (por concorrência) da família SHA-3. A família SHA-3 foi lançada pelo NIST em 5 de agosto de 2015.

# 4. Funções Hash

Todas as funções mencionadas são funções com implementações iterativas.

Isto quer dizer que o cálculo destas funções divide a entrada em uma sequência de blocos de tamanho fixo: $m_1$, $m_2$, ... $m_k$, utilizando algum esquema de preenchimento (*padding*) para o último bloco.

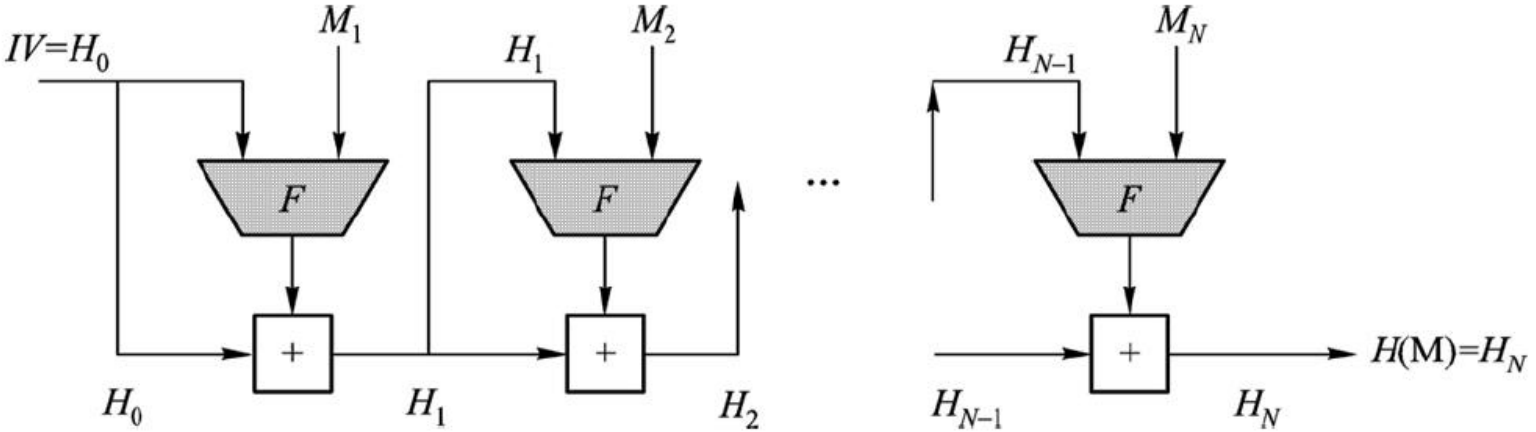
Os blocos são processados em sequência, usando uma função de compressão $h'(m_i)$ (ou F, na figura).

# 4. Funções Hash

O resultado de cada etapa é guardado como um estado intermediário, que é combinado para formar o próximo estado com o próximo bloco.

O primeiro bloco utiliza um vetor de iniciação (IV - fixo) e o último estado é o resultado

# 4. Funções Hash



# 4. Funções Hash

Um exemplo com a biblioteca org.apache.commons.codec (em JAVA):

```
package commonssha;
import org.apache.commons.codec.binary.Base64;
import org.apache.commons.codec.digest.DigestUtils;
public class Main {
  public static void main(String[] args) {
    try{
      byte[] digest =
        DigestUtils.sha256("Teste de SHA256...".
                                    getBytes("ASCII"));
      System.out.println("-- Digest: " +
          (new String(Base64.encodeBase64(digest))));
    }catch(Exception e){
      e.printStackTrace();
    }
  }
}
```

# 4. Funções Hash

Observe que para mostrar o resultado, uma sequência de 256 bits binários, usamos a codificação Base64, que codifica dados binários em ASCII.

A codificação Base64 NÃO é criptográfica e é reversível.

# 4. Funções Hash

Todas as funções Hash mostradas são funções não ideais, passíveis de um ataque.

Por isto, elas devem ser utilizadas de uma forma que evite os ataques conhecidos.

O primeiro tipo de ataque  é conhecido como “extensão do comprimento”.

# 4. Funções Hash

Suponhamos que uma mensagem m seja dividida em blocos $m_1$, $m_2$, ... $m_k$,

Seja uma outra mensagem m' que seja dividida em blocos $m_1$, $m_2$, ... $m_k$, $m_{k+1}$, com

$m_1$, $m_2$, ... $m_k$, iguais nas duas mensagens.

Neste caso, h(m) é o resultado intermediário depois de k blocos no cálculo de h(m').

# 4. Funções Hash

Então, $h(m') = h'(h(m), m_{k+1})$.

Se a função Hash for usada para autenticar a mensagem m, isto permitiria que um atacante modificasse a mensagem e o valor da função Hash, sem que o receptor percebesse a diferença.

(*Download* de um arquivo, p.ex.)

# 4. Funções Hash

O outro tipo de ataque é  o de uma colisão de parte da mensagem.

Ele se aplica quando se procura autenticar uma mensagem m com uma chave de autenticação X:

$$h( m \| X )$$

O atacante pode usar um ataque de aniversário para procurar uma colisão  com uma mensagem m' tal que h(m) = h(m').

# 4. Funções Hash

Isto pode ser feito com um custo de $2^{n/2}$, o que pode ser razoável, dependendo de n.

Achada a mensagem m', teremos

$$h( m \,||\, X ) = h( m' \,||\, X )$$

para qualquer X.

# 4. Funções Hash

Um exemplo de solução *ad hoc* para este problema pode ser visto em

http://www.jasypt.org/howtoencryptuserpasswords.html

A idéia desta solução é baseada em:

• Adicionar um número aleatório à mensagem (pelo menos 8 bytes, conhecido como *salt*)

• Iterar a função Hash um bom número de vezes (de 2 a 1000 vezes)

# 5. Autenticação (MAC)

O objetivo da autenticação é evitar que o atacante manipule as mensagens (mesmo sem compreende-las).

Um MAC é uma função:

$$a = MAC( K, m )$$

Onde K é uma chave simétrica secreta e m é a mensagem.

# 5. Autenticação (MAC)

Um MAC produz um resultado de tamanho fixo e é uma função não inversível. Também deve ser uma função aleatória.

O transmissor envia agora a mensagem (possivelmente criptografada) e o valor MAC correspondente.

A diferença em relação a uma função Hash é que agora o transmissor e o receptor compartilham a chave K.

# 5. Autenticação (MAC)

O receptor pode então verificar se o MAC recebido corresponde realmente à mensagem e que foi enviado por alguém **que compartilha a chave secreta !**

A estrutura de uma função MAC é muito semelhante à de uma função Hash iterativa.

# 5. Autenticação (MAC)

Ela pode ser descrita por:

$$H_0 = IV$$
$$H_i = E_k( m_i \oplus H_{i-1})$$
$$MAC = H_n$$

Onde $H_i$ , i = 0, 1, 2, ... n, é o estado, mas a função $E_K$ é a função de encriptação com a chave simétrica K e $m_i$ , com i = 1, 2, ... n, são os blocos em que a mensagem m foi dividida.

# 5. Autenticação (MAC)

## HMAC

Este é um MAC muito utilizado e que baseia-se no uso de uma função Hash.

Ele é definido por:

$$(K,m) \rightarrow \mathbf{SHA256(\ (K \oplus a) \ ||\ SHA256(\ (K \oplus b) \ ||\ m)\ )}$$

Onde a e b são constantes definidas no padrão e que introduzem alguma aleatoriedade.

# 5. Autenticação (MAC)

Na prática, a troca de mensagens entre Alice (cliente) e Bob (servidor) precisa ser acompanhada de outras informações, tais como:

- Uma numeração de sequência das mensagens,
- Uma descrição do sentido (Alice-Bob ou Bob-Alice),
- A data e o tempo (timestamp),
- Uma descrição da versão do software utilizado,
- Uma descrição dos campos utilizados na mensagem propriamente dita,
- Etc.

# 5. Autenticação (MAC)

Para evitar algum tipo de ataque, este cabeçalho d também deve ser autenticado:

$$a = MAC( d \,||\, m )$$

Como é o cabeçalho que complementa o significado da mensagem, diz-se:

Princípio de Horton

"Autentique o significado, não a forma"

# 5. Autenticação (MAC)

É devido ao princípio de Horton, que a autenticação nos níveis mais baixos de protocolo (IPsec) não é suficiente.

Além da autenticação nos níveis mais baixos de protocolo, é necessário autenticar nos níveis mais altos (aplicação) também.

# 6. O canal seguro

Com o material que já foi apresentado, podemos finalmente atacar o primeiro problema prático: um canal seguro para Alice e Bob trocarem mensagens.

Primeiramente, precisaremos formalizar um pouco o problema.

# 6. O canal seguro

Um pouquinho de requisitos:

- Os papéis
- A chave secreta
- Datagramas ou fluxo
- Os requisitos de segurança
- A ordem de autenticação e encriptação
- Aspectos de implementação

# 6. O canal seguro

## Os papéis

Vamos supor que a comunicação é bidirecional.

Se não fizessemos esta suposição, i.e. um canal unidirecional, a comunicação nos dois sentidos precisaria de dois canais uniderecionais. Neste caso, precisaríamos considerar a posteriori as dependências de segurança entre os dois canais.

Quando supomos um canal bidirecional, todas estas as dependências já são consideradas.

# 6. O canal seguro

Vamos supor que o canal é assimétrico, i.e. Os dois lados podem ser identificados.

Como um dos lados toma a iniciativa, este será chamado de cliente e o outro de servidor.

Ainda existe o terceiro agente: o atacante (Eva).

O atacante pode fazer “análise do tráfego”, i.e. observar a temporização e o tamanho das mensagens.

# 6. O canal seguro

O atacante pode ainda destruir, modificar e inserir mensagens.

## A chave secreta

Para manter o segredo das mensagens tanto o cliente como o servidor devem conpartilhar uma chave secreta. Supomos que o atacante não conhece esta chave secreta (assim como ninguém mais).

# 6. O canal seguro

Quando o servidor recebe uma mensagem e a decripta corretamente com a chave secreta, ele sabe que a mensagem foi enviada por alguém no papel de cliente e que conhece a chave secreta.

O mesmo acontece no sentido inverso.

Além disto, cada vez que o canal for iniciado, ele vai utilizar uma chave nova, diferente de outras utilizadas anteriormente.

# 6. O canal seguro

Neste caso, NÃO vamos tratar do problema de como a chave secreta compartilhada é obtida pelos dois agentes.

Isto será visto mais adiante com chaves assimétricas (Pública/Privada).

Na verdade, sugere-se que sejam criadas 4 chaves a partir da chave secreta, para os dois sentidos, para autenticação e encriptação (com $SHA_d$, double).

# 6. O canal seguro

## Datagramas ou fluxo

Vamos utilizar somente datagramas.

Supondo-se o uso dos protocolos da internet (TCP/IP) vamos usar, por exemplo, UDP.

Também se poderia utilizar mensagens em TCP, mas a chamada característica de comunicação segura (*reliable*) do TCP não o é num sentido criptográfico.

Não existe protocolo seguro num contexto criptográfico, com um atacante ativo.

# 6. O canal seguro

## Os requisitos de segurança

O cliente deseja enviar uma sequência de mensagens $m_1$, $m_2$, ..., $m_n$ para o servidor (e analogamente no sentido inverso).

As mensagens são processadas pelo programa do canal seguro e enviadas.

O servidor recebe as mensagens e as processa, gerando a sequência $m'_1$, $m'_2$, ..., $m'_p$

# 6. O canal seguro

O atacante não consegue compreender nada das mensagens observadas no canal, exceto pela análise do tráfego.

Se o atacante manipular as mensagens (destruir, modificar, incluir) estas serão descartadas e o servidor receberá somente uma subsequência da sequência original.

Além disto, esta subsequência estará na ordem correta e o servidor saberá exatamente quais mensagens foram descartadas

# 6. O canal seguro

Além da ordem ser mantida na subsequência, não haverá mensagens duplicadas, mensagens falsas ou alteradas.

Para obter isto, a solução óbvia é usar um esquema de numeração das mensagens.

Por outro lado, a confirmação e o pedido de retransmissão de mensagens não é um problema do protocolo criptográfico e pode (e deve) ser tratado no nível acima do canal seguro.

# 6. O canal seguro

## A ordem de autenticação e encriptação

Um canal seguro deve usar tanto encriptação como autenticação.

Schneier, no livro de referência, argumenta que poder-se-ia escolher uma ordem ou outra.

Contudo, ele acaba aconselhando, para algumas situações, autenticar primeiro e encriptar depois.

# 6. O canal seguro

Vale a pena relembrar que, usando o princípio de Horton, deve-se autenticar conjuntamente o cabeçalho e a mensagem propriamente dita.

**Aspectos de implementação**

Schneier faz uma crítica das várias linguagens de programação disponíveis e tanto C, C++ como JAVA apresentam fraquezas.

Os sistemas operacionais Windows e Unix também não são perfeitos.

# 6. O canal seguro

Aparentemente sobraram Assembly e a possibilidade de modificar e corrigir alguma versão de Unix/Linux.

Na verdade, as críticas que ele faz são muito estritas. Na prática, temos que nos arranjar com o que é disponível.

Para testar estas idéias, vamos utilizar JAVA e Windows (ou Linux).

# 6. O canal seguro

A numeração das mensagens deve ser feita de forma monotônica e sem repetições.

Em JAVA, o uso de inteiros de precisão simples (*int* – 32 bits com sinal) permite usar a sequência de +1 a +2.147.483.647 ($2^{31}$-1, com o 0 para representar que nehuma mensagem ainda foi enviada ou recebida)

Uma sessão não pode então enviar mais do que 2.147.483.647 mensagens.

# 6. O canal seguro

Para autenticar a mensagem, Schneier sugere usar HMAC-SHA-256.

A entrada desta função é composta do número de sequência *i*, dos dados auxiliares $d_i$ (cabeçalho) e da mensagem propriamente dita $m_i$.

Um formato possível para o cálculo do MAC é:

$$a_i = MAC( i \,||\, length(d_i) \,||\, d_i \,||\, m_i )$$

# 6. O canal seguro

Para encriptar a mensagem e a autenticação, sugere-se AES com o modo CTR.

Como o modo CTR precisa de um ***nonce***, o número de sequência da mensagem pode ser usado para tal.

Como cada bloco AES tem 16 bytes, o tamanho total máximo de uma mensagem é de $15 \times 2^{31}$ bytes (aprox. 30 GBytes).

# 6. O canal seguro

O modo CTR basicamente vai fazer o XOR de um bloco de texto limpo com uma chave variável criada a partir da chave secreta, de um contador de um nonce.

O bloco de texto limpo é a concatenação do bloco cabeçalho-mensagem com a autenticação:

$$t_i = d_i \,||\, m_i \,||\, a_i$$

# 6. O canal seguro

A sequência de chaves variáveis é dada por:

$$k_j : \; E_K( j \,||\, i \,||\, \text{zeros} )$$

$E_K$ é a função de encriptação do AES com a chave secreta K,

j , o número do bloco, é o contador do CTR,

i é o nonce (o número da mensagem),

o resto do bloco (de 16 bytes) é completado com zeros.

# 6. O canal seguro

Cada chave variável é combinada com o texto limpo para criar o bloco cifrado a ser enviado:

$$c_i = t_i \oplus k_i$$

Na verdade, por razões práticas, o quadro do datagrama a ser enviado possui o número da mensagem e o bloco cifrado:

$$i \,||\, c_i$$

# 6. O canal seguro

Quando se utiliza a Internet, datagramas UDP podem chegar ao destino fora de ordem.

Para evitar que isto aconteça, sem a intervenção de um atacante, alguma técnica pode (e deve) ser usada para reordenar as mensagens recebidas (pelo menos parcialmente).

# 6. O canal seguro

A recepção, desencriptação e a verificação da autenticidade seguem passos análogos.

# 7. Chaves públicas: Diffie-Hellman

A criptografia em blocos, como a AES, e seus os modos de operação provêm uma maneira de enviar mensagens com um nível de privacidade bastante bom.

O uso de autenticação, como HMAC baseado em SHA-256, permite construir um canal de comunicação seguro.

# 7. Chaves públicas: Diffie-Hellman

Entretanto, ele depende basicamente do uso de uma chave secreta que deve ser compartilhada.

A distribuição desta chave constitui um grande problema.

Para resolve-lo, pode-se utilizar o esquema descrito a seguir.

# 7. Chaves públicas: Diffie-Hellman

A (Alice) deseja enviar mensagens criptografadas para B (Bob) usando uma chave secreta K (que ainda não foi escolhida).

Para isto, A envia para B (por um correio confiável) uma caixa forte, com uma aldraba e contendo no seu interior um cadeado aberto.

A mantem consigo a chave deste cadeado.

# 7. Chaves públicas: Diffie-Hellman

B recebe a caixa, coloca dentro desta a chave secreta K que será compartilhada com A, tranca a caixa com o cadeado e a devolve por correio para A.

A abre a caixa e pega a chave secreta K, que agora pode ser usada para trocar mensagens cifradas (e autenticadas) de forma eficiente e segura.

# 7. Chaves públicas: Diffie-Hellman

Este esquema funciona se o correio não fizer um ataque do tipo chamado *man-in-the-middle* (ou *bucket-brigade*):

Neste caso, o correio substitui o cadeado aberto na caixa por um outro cadeado seu antes de entregar a caixa a B.

Quando ele recebe a caixa de B, ele abre o cadeado, lê a chave K e coloca o cadeado original, trancando-o antes de entregar a caixa a A.

# 7. Chaves públicas: Diffie-Hellman

A defesa contra este tipo de ataque é autenticar o cadeado, de uma forma que ele não possa ser substituído por outro.

Um protocolo baseado nesta idéia foi proposto em 1976 por Whitfield Diffie e Martin Hellman (Diffie-Hellman - DH) e deu origem a um ramo da criptografia conhecido como PKC – Public Key Cryptography.

# 7. Chaves públicas: Diffie-Hellman

Em 1977, Ronald Rivest, Adi Shamir e Leonard Adleman apresentaram um sistema semelhante, com uma implementação, que se tornou conhecido (e patenteado) como RSA.

Em termos matemáticos, podemos traduzir a paródia da caixa com o cadeado da maneira descrita a seguir.

# 7. Chaves públicas: Diffie-Hellman

A e B usam uma função $f_0$

$$y = f_0( k , x )$$

tal que, dados k e y, é computacionalmente muito difícil (intratável) calcular x.

O valor k é um inteiro positivo, parâmetro do algoritmo, e que é conhecido publicamente.

# 7. Chaves públicas: Diffie-Hellman

A e B também usam uma outra função $f_1$

$$y = f_1( z , x )$$

que tem a seguinte propriedade:

$$f_1( f_0 ( z , y ) , x ) = f_1( f_0 ( z , x ) , y )$$

# 7. Chaves públicas: Diffie-Hellman

A seleciona um número inteiro positivo $\mathbf{x_a}$ , aleatório, como sua **chave privada**, calcula $y_a = f_0( k , \mathbf{x_a} )$ e envia $y_a$ (sua chave pública) para B.

Por outro lado,

B seleciona um número inteiro positivo $\mathbf{x_b}$ , aleatório, como sua **chave privada**, calcula $y_b = f_0( k , \mathbf{x_b} )$ e envia $y_b$ (sua chave pública) para A.

# 7. Chaves públicas: Diffie-Hellman

Se A calcular $K_1 = f_1( y_b , x_a )$, enquanto

B calcular $K_2 = f_1( y_a , x_b )$, teremos

$$f_1( y_b , \mathbf{x_a} ) = f_1(f_0( k , \mathbf{x_b} ), \mathbf{x_a} ) =$$
$$f_1(f_0( k , \mathbf{x_a} ), \mathbf{x_b} ) = f_1( y_a , \mathbf{x_b} )$$

Daí temos que

$$K_1 = K_2 = K$$

Quer dizer, tanto A como B puderam calcular uma chave secreta compartilhada K.

# 7. Chaves públicas: Diffie-Hellman

No protocolo DH original, as chaves públicas e privadas NÃO são utilizadas para encriptar uma mensagem arbitrária, mas sim para calcular uma chave secreta compartilhada K.

Esta chave secreta compartilhada K é que será utilizada para trocar mensagens em um canal seguro.

# 7. Chaves públicas: Diffie-Hellman

O protocolo DH original baseia-se numa teoria sobre grupos multiplicativos módulo **p**.

No protocolo DH as funções $f_0$ e $f_1$ são definidas por:

$$f_0(p,g;x) = g^x \bmod p$$
$$f_1(y,x) = y^x \bmod p$$

onde p e g são parâmetros do algoritmo conhecidos publicamente.

x e y são inteiros positivos.

# 7. Chaves públicas: Diffie-Hellman

A escolhe um número positivo, aleatório $X_a$ tal que $X_a < p$.

$X_a$ é a chave privada de A.

A então calcula $Y_a = f_0(p,g;X_a) = g^{Xa} \bmod p$.

$Y_a$ é a chave pública de A.

# 7. Chaves públicas: Diffie-Hellman

B também escolhe um número positivo, aleatório $X_b$ tal que $X_b < p$.

$X_b$ é a chave privada de B.

B então calcula $Y_b = f_0(p,g;X_b) = g^{Xb} \bmod p$.

$Y_b$ é a chave pública de B.

# 7. Chaves públicas: Diffie-Hellman

Sabe-se que:

$(b^x \bmod p)^y \bmod p = (b^x)^y \bmod p = b^{xy} \bmod p$

# 7. Chaves públicas: Diffie-Hellman

Então, pode-se ver que

$$K_a = Y_b^{Xa} \bmod p = (g^{Xb} \bmod p)^{Xa} \bmod p = g^{XaXb} \bmod p$$

$$K_b = Y_a^{Xb} \bmod p = (g^{Xa} \bmod p)^{Xb} \bmod p = g^{XbXa} \bmod p$$

e que então

$$K_a = K_b = K,$$

a chave secreta compartilhada.

# 7. Chaves públicas: Diffie-Hellman

A dificuldade computacional que garante a confidencialidade está no cálculo de $X_a$ dada a equação

$$Y_a = g^{Xa} \bmod p \qquad \text{(i.e. dados } Y_a \text{, } g \text{ e } p\text{)}.$$

Como **g**, **p** e $Y_a$ são inteiros, a solução desta equação é dada pelo logaritmo discreto.

É sabido que não existe um algoritmo eficiente para calcular o logaritmo discreto.

Todos os algoritmos conhecidos requerem um tempo exponencial com o tamanho do grupo discreto definido por **p** e **g** (gerador).

# 7. Chaves públicas: Diffie-Hellman

Chegamos então ao seguinte protocolo de troca de troca de chaves:

**A** **B**

Escolhe os parâmetros **g** e **p**

Escolhe uma chave privada $x_a$

Calcula $y_a = \mathbf{g}^{xa} \bmod \mathbf{p}$

Envia $(\mathbf{g},\mathbf{p},y_a)$ para B $\rightarrow$

# 7. Chaves públicas: Diffie-Hellman

**A** **B**

Recebe os parâmetros **g** e **p**

Escolhe uma chave privada $x_b$

Calcula $y_b = \mathbf{g}^{xb} \bmod \mathbf{p}$

← Envia $y_b$ para A

Calcula **K** = $y_a^{xb} \bmod p$

Calcula **K** = $y_b^{xa} \bmod p$

# 7. Chaves públicas: Diffie-Hellman

Para garantir a segurança contra ataques matemáticos, os parâmetros **g** e **p** devem ser escolhidos adequadamente.

**p** deve ser um número primo com mais de 300 dígitos (decimais) da forma **p = (2q + 1)** onde **q** também é um primo (grande).

**p** é então chamado de um primo seguro.

# 7. Chaves públicas: Diffie-Hellman

**g** deve ser um número que seja o gerador de um subgrupo de ordem **q** (e não **p**).

Isto faz com que não se possa nem descobrir o bit menos significativo de $x_a$ (usando o símbolo de Legendre).

As chaves $x_a$ e $x_b$ devem ser números aleatórios

# 7. Chaves públicas: Diffie-Hellman

Um exemplo de programa em JAVA que faz a troca de chaves de Deffie-Hellman é mostrado em um apêndice.

# 8. Protocolos criptográficos

A tecnologia de protocolos de comunicação em redes de computadores já se encontra razoavelmente desenvolvida.

Mas quando se trata de protocolos de comunicação em criptografia, a história é outra.

# 8. Protocolos criptográficos

Quando um agente (ou entidade) se comunica com outro num ambiente criptográfico, ele tem que assumir a paranóia da criptografia:

Ele tem que assumir que está lidando com o inimigo.

# 8. Protocolos criptográficos

Os agentes envolvidos num protocolo são identificados por seus papéis: A (Alice, cliente ou outro), B (Bob, servidor, comerciante etc), E (Eva ou *Enemy*, inimigo ou atacante).

Os indivíduos reais podem assumir estes papéis ou alternar-se entre eles.

# 8. Protocolos criptográficos

A razão mais básica que permite que um agente dialogue com outros é a confiança (*trust*).

Por exemplo, uma simples compra no comércio exige que haja confiança no outro, tanto da parte do cliente como da parte do comerciante.

# 8. Protocolos criptográficos

As razões para se ter confiança podem ser várias:

- Ética
- Reputação
- Lei
- Ameaça física
- MAD – *Mutual Assured Destruction*

# 8. Protocolos criptográficos

Mas a confiança NÃO é booleana.

Uma coisa é confiar que o seu colega vai lhe pagar o dinheiro do café no dia seguinte, outra coisa é deixar com ele o bilhete premiado da mega-sena...

A questão mais bem posta é:

"A deve confiar em B por x reais?" (ou outra escala de valores apropriada)

# 8. Protocolos criptográficos

No mundo dos negócios é mais comum falar de **risco**.

O **risco** é diretamente ligado à desconfiança, o contrário da confiança.

Por isto, é necessário “traduzir” os termos quando se fala com o pessoal de negócios,

**risco**,

ou com o pessoal de software,

**confiança**.

# 8. Protocolos criptográficos

Uma outra componente no diálogo (protocolo) entre as partes é a motivação ou incentivo.

Por exemplo, um protocolo de pagamento eletrônico não impede o comerciante de enganar o cliente. Mas ele cria uma prova de que o comerciante recebeu e não atendeu o cliente.

O comerciante tem um incentivo para não ter clientes com provas que podem ser usadas contra ele na justiça ou simplesmente arruinar sua reputação.

# 8. Protocolos criptográficos

Por outro lado, o protocolo favorece o comerciante, permitindo que ele só envie a mercadoria depois de receber a confirmação da empresa de cartão de crédito.

Os incentivos parecem ter sobretudo razões materiais.

Mas muitos ataques a computadores NÃO são feitos por razões materiais, mas por divertimento, para contar vantagem, por vingança, etc.

# 8. Protocolos criptográficos

Função dos protocolos criptográficos:

**“Minimizar a necessidade de confiança”**

Significa minimizar o número de pessoas que precisam confiar umas nas outras e o quanto elas devem confiar nas outras.

(Quando se tem garantias ou evidências da identidade e das intenções do outro, não se precisa contar tanto com a confiança)

# 8. Protocolos criptográficos

O modelo básico no projeto de protocolos criptográficos é o modelo paranóico.

Quando A (Alice) participa de um protocolo, ela assume que todos os outros participantes conspiram contra ela. Cada um dos outros participantes também pensa assim.

Sempre que houver uma excessão a esse modelo (confiança em algo ou alguém), isto **DEVE SER DOCUMENTADO**.

# 8. Protocolos criptográficos

Do ponto de vista dos negócios, os requisitos DOCUMENTADOS de confiança (em algo ou alguém) implicam num **RISCO**.

Riscos devem ser documentados e tratados de apropriadamente (seguro, responsabilidade civil, financeira, etc).

# 8. Protocolos criptográficos

## Mensagens e camadas

Uma das maneiras de especificar protocolos é através das mensagens que são trocadas entre as entidades e das computações que cada entidade deve fazer.

Para lidar com a grande complexidade que logo aparece, a solução é modularizar o problema.

A forma tradicional é dividir o protocolo em camadas.

# 8. Protocolos criptográficos

As camadas inferiores geralmente prestam serviços conhecidos, genéricos e mais documentados.

Geralmente, é somente a camada mais alta que precisa ser documentada.

# 8. Protocolos criptográficos

## A camada de transporte

Do ponto de vista da criptografia, pacotes UDP, TCP, e-mail, ou arquivos em pen-drives, todos pertencem à camada de transporte.

A camada de transporte deve também prover campos adicionais para garantir que uma sequência de bytes enviada seja idêntica à sequência recebida.

Isto pode exigir campos de tamanho da mensagem, numeração das mensagens, etc.

# 8. Protocolos criptográficos

Eventualmente, a camada de transporte pode prover confidencialidae (por encriptação), autenticação e proteção contra repetições.

É o caso do uso de um canal seguro, como mostrado em um exemplo anterior.

Isto pode facilitar o projeto do protocolo, já que podem haver menos possibilidades de ataque.

# 8. Protocolos criptográficos

## Identificação do protocolo e das mensagens

A camada imediatamente acima da camada de transporte deve identificar o protocolo, sua VERSÃO e o tipo de mensagem.

Todas estas identificações são importantes pricipalmente quando se usa autenticação, já que frequentemente autenticam-se várias mensagens  (como veremos mais a frente).

# 8. Protocolos criptográficos

## Codificação e Análise

A camada seguinte é a camada de codificação.

Geralmente, os elementos da mensagem são convertidos de sua codificação original para uma sequência de bytes.

A operação inversa, no receptor, é a análise sintática e a codificação para o formato original.

Esta análise deve ser feita de maneira INDEPENDENTE DO CONTEXTO.

# 8. Protocolos criptográficos

Algumas das técnicas usuais de codificação/decodificação são:

- ASN.1 (relativamente complexa)
- XML, com o uso de um DTD.

# 8. Protocolos criptográficos

## Os estados do protocolo

O estado (principal) de um protocolo é geralmente especificado através de uma máquina de estados.

A implementação de máquinas de estados para várias instâncias de um protocolo é geralmente feita com algum tipo de programação baseada em eventos e eventualmente multithreading ou vários processos.

# 8. Protocolos criptográficos

O tratamento de erros deve dar a menor informação possível a um atacante.

A informação mais detalhada pode ser armazenada em um histórico (*log*). Além disto, o tempo para dar uma mensagem de erro curta ao usuário não deve dar informações sobre o tipo do erro.

# 8. Protocolos criptográficos

Repetições (normais) de mensagens devem ser idênticas.

A resposta a repetições de mensagens também deve ser igual.

Finalmente, é muito difícil (impossível) saber se uma mensagem é realmente a última mensagem de um protocolo. Ele deve ser projetado para ser seguro apesar disto.

# 8. Protocolos criptográficos

## Exemplo

## Um protocolo de negociação de chaves

Fergusson & Schneier mostram no capítulo 15 o projeto iterativo de um protocolo em quatro versões.

Apresentamos a seguir somente a versão final do protocolo.

Neste, supõe-se que Alice e Bob podem autenticar as mensagens um do outro, mas os detalhes não são mostrados (serão vistos depois).

# 8. Protocolos criptográficos

Alice e Bob vão iniciar uma troca de chaves usando Diffie-Hellman para estabelecer uma chave de sessão k.

Esta chave de sessão será usada somente durante uma única sessão.

Mas para faze-lo, Alice e Bob devem se autenticar mutuamente.

# 8. Protocolos criptográficos

Depois disto, Alice e Bob usam uma chave secreta para trocar os dados através um canal seguro.

Além disto, usa-se uma convenção em que toda autenticação é feita usando-se todas as mensagens trocadas (nos dois sentidos) até aquele ponto.

# 8. Protocolos criptográficos

**Alice** **Bob**

Alice envia o tamanho mínimo de primo que ela deseja $p_{min}$ e um nonce $N_a$ que Bob deve usar na autenticação

$p_{min}$, $N_a$

# 8. Protocolos criptográficos

**Alice** **Bob**

Bob escolhe (p,q,g), calcula $X=g^x$ e autentica a mensagem anterior ($p_{min}$, $N_a$)

$(p,q,g),X,AUTH_B$

←————————————————————————

# 8. Protocolos criptográficos

**Alice** **Bob**

Alice verifica a autenticação,
verifica se (p,q,g) foram
bem escolhidos (min e máx),
verifica se X é válido,
escolhe y e calcula
$Y=g^y$ ,
autentica as mensagens anteriores

$Y, AUTH_A$

# 8. Protocolos criptográficos

**Alice** **Bob**

Bob verifica a autenticação

verifica Y,

calcula a chave da sessão

$k = SHA_d\text{-}256(Y^x)$

Alice calcula

$k = SHA_d\text{-}256(X^y)$

# 8. Protocolos criptográficos

O passo seguinte é verificar as várias visões do protocolo:

**Visão de Alice**

Alice só recebe uma única mensagem de Bob:

Ela tem a certeza que é de Bob porque a mensagem pôde ser autenticada;

Ela verifica a validade dos dados de DH e tem a certeza que somente ela e Bob têm a chave secreta.

# 8. Protocolos criptográficos

## Visão de Bob

A primeira mensagem que Bob recebe não lhe dá nenhuma confiança, mas ele responde com dados que não divulgam nenhum segredo.

A terceira mensagem pode ser autenticada como sendo de Alice. Como os parâmetros de DH foram escolhidos por ele mesmo, ele tem a certeza que somente ele e Alice têm a chave secreta.

# 8. Protocolos criptográficos

## Visão do atacante

Se o atacante observar a troca DH e os parâmetros tiverem sido bem escolhidos, ele não pode descobrir a chave secreta.

Se ele tentar modificar alguma coisa das mensagens, a autenticação impedirá que Alice ou Bob aceitem qualquer modificação.

# 8. Protocolos criptográficos

## Perda de chaves

Vamos analizar quais as consequências de perda de chaves.

Se Alice (ou Bob) perder a sua chave de autenticação, sem que ela caia em poder de um atacante, ela perde a capacidade de usar este protocolo.

Ela pode contudo continuar usando a chave de uma sessão já estabelecida para trocar dados.

# 8. Protocolos criptográficos

Se Alice perder a chave da sessão, sem que ela caia em poder de um atacante, ela precisará usar de novo o protocolo para estabelecer uma nova chave de sessão.

Se um atacante obtiver a chave de autenticação de Alice, ele pode se passar por Alice. Por isto, Alice deve informar Bob o mais rápido possível que sua chave de autenticação foi revogada!

# 8. Protocolos criptográficos

Finalmente, se o atacante conseguir se apoderar da chave de sessão (compartilhada por Alice e Bob), ele poderá se inteirar dos segredos de Alice e Bob, mas como ele não tem uma chave de autenticação, ele não poderá modificar as mensagens.

# 9. Servidores de chaves

Segundo Fergusson & Schneier (Practical Cryptography, capítulo 18):

*“At last we turn to key management. This is, without a doubt, the most difficult issue in cryptographic systems,...”*

“Finalmente chegamos ao problema de gerenciamento de chaves. Este é, sem sombra de dúvidas, o tópico mais difícil dos sistemas criptográficos,...”

# 9. Servidores de chaves

O gerenciamento de chaves é difícil porque envolve pessoas, não pela matemática.

Uma das maneiras de lidar com este problema, talvez a mais natural, é ter um servidor, **no qual se confia**, que vai guardar e entregar todas as chaves necessárias.

Este servidor vai utilizar protocolos criptográficos para entregar com segurança as chaves aos destinatários.

# 9. Servidores de chaves

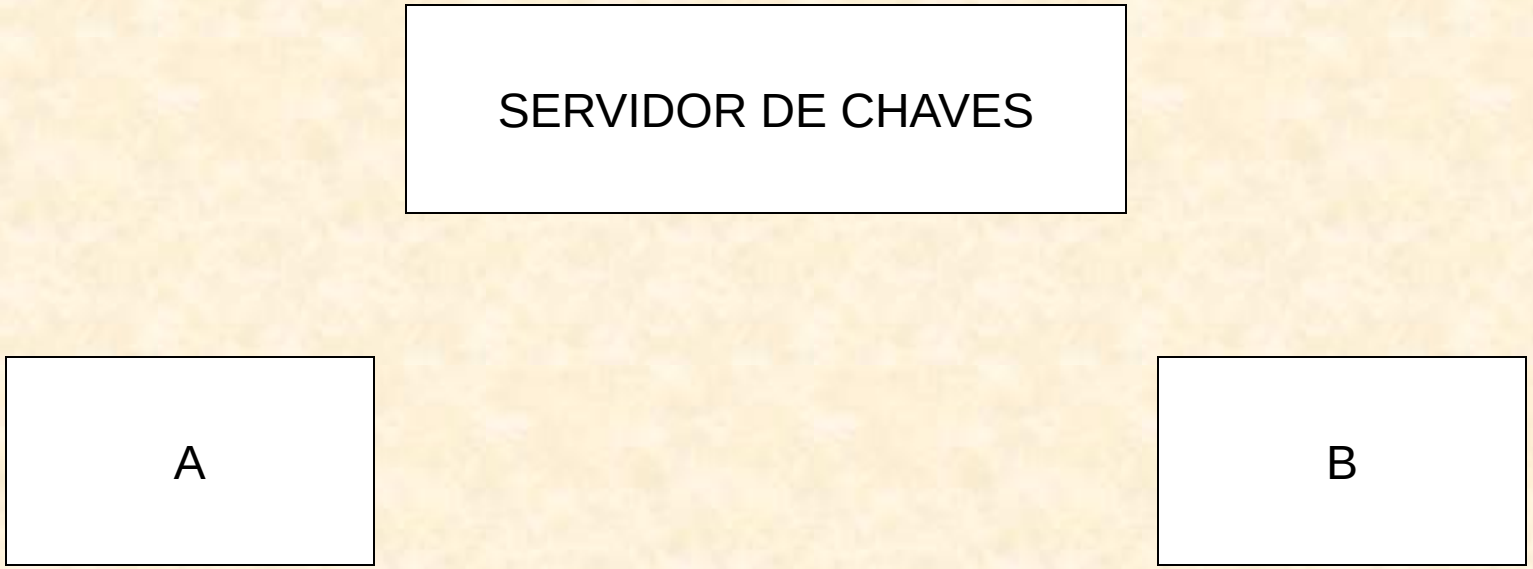
Protocolos criptográficos para este tipo de serviço SÃO MUITO DIFÍCEIS.

Vamos apresentar a idéia básica de um sistema chamado KERBEROS, que pode ser adaptada para casos mais simples.

Porém, mesmo os casos simplificados ainda são MUITO DIFÍCEIS.

# 9. Servidores de chaves

Seja um **Servidor de Chaves** que vai guardar chaves secretas de vários usuários, sejam clientes ou servidores de outros serviços.



# 9. Servidores de chaves

Suponhamos que **A** é um cliente de serviços de **B**.

**A** deseja comunicar-se com **B** de forma segura, mas **A** e **B** **não** compartilham uma chave secreta.

O protocolo de troca de chaves que vimos antes supunha que havia uma forma de autenticar **A** e **B** um para o outro, usando uma chave secreta.

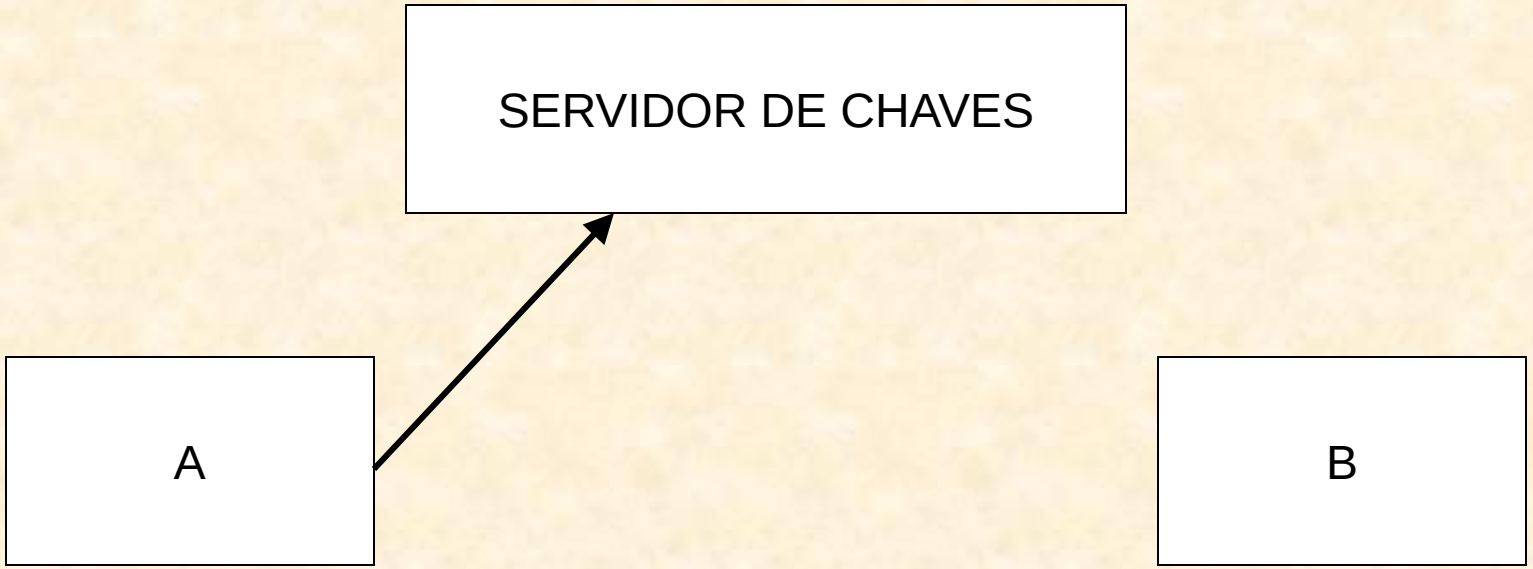
# 9. Servidores de chaves

O **Servidor de Chaves** vai resolver este problema:

Supomos que cada usuário (**A**, **B**, **C**, ...) deposita uma chave secreta no **Servidor de Chaves**. Além da chave secreta, ele normalmente coloca outras informações pertinentes, como *username*, senha, nome pessoal, nome do *host*, etc.

# 9. Servidores de chaves

Cadastramento da chave secreta e outras informações (A, B, C, ...):



# 9. Servidores de chaves

Nesta etapa,

**A** confia no **Servidor de Chaves** e o

**Servidor de Chaves** deve se assegurar da identidade de **A**.

É assim que normalmente abrimos contas em bancos, obtemos carteira de motorista e criamos contas em computadores (redes) com o administrador de rede !

# 9. Servidores de chaves

Quando **A** deseja se comunicar com **B**,

**A** diz isto ao **Servidor de Chaves** (S).

$$A \rightarrow S : \{ B \} \quad (1)$$

O **Servidor de Chaves** cria uma nova chave secreta, a chave de sessão, que será compartilhada por **A** e **B**: $K_{AB}$ .

# 9. Servidores de chaves

O **Servidor de Chaves** envia uma mensagem para **A**, criptografada com a chave secreta de **A**, contendo

- a chave $K_{AB}$
- a chave $K_{AB}$ criptografada com a chave secreta de **B**.

$$S \rightarrow A : \{ K_{AB} , \{ K_{AB} \}_{KB} \}_{KA} \quad (2)$$

# 9. Servidores de chaves

**A** pode desencriptar a mensagem e obter a chave $K_{AB}$

e

enviar a chave $K_{AB}$ encriptada com a chave secreta de **B** para **B**.

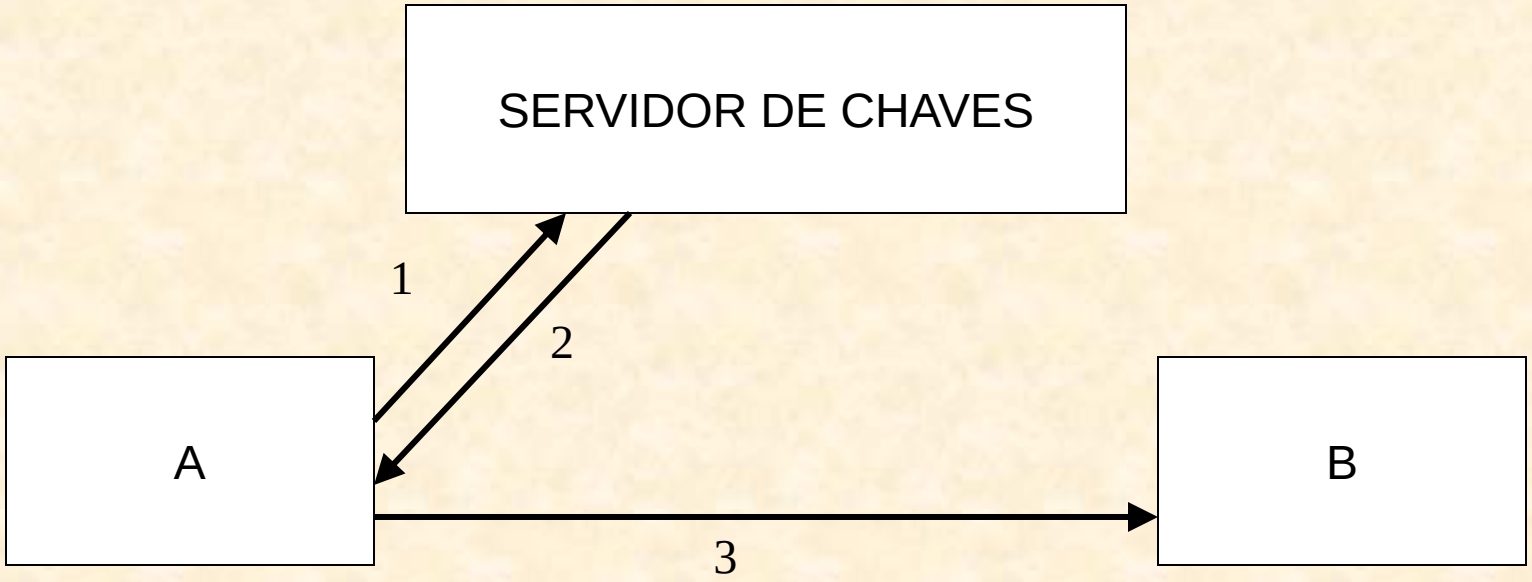
$$A \rightarrow B : \{ \{ K_{AB} \}_{KB} \} \quad (3)$$

# 9. Servidores de chaves

**B** pode desencriptar esta mensagem e obter a nova chave secreta de sessão compartilhada com **A**.

Agora **A** e **B** podem autenticar-se e comunicar-se seguramente com uma chave secreta que só é válida durante uma sessão.

# 9. Servidores de chaves



# 9. Servidores de chaves

Na terminologia do KERBEROS, o servidor é um *trusted third party*, chamado de KDC – *Key Distribution Center*.

O KDC é dividido em duas partes:

AS – Authentication Server

TGS – Ticket Granting Server

O protocolo KERBEROS é um pouco mais complexo que o exemplo mostrado.

# 9. Servidores de chaves

O servidor **B** do exemplo é conhecido como

SS – Service Server

O cliente **A** faz login na sua estação usando um nome de usuário e uma senha que também estão armazenados no AS.

A chave secreta de **A** é gerada por Hash a partir da senha, tanto na estação do cliente como no AS.

# 9. Servidores de chaves

O protocolo pode então ser dividido em quatro fases:

- Login (local) do cliente

- Autenticação do cliente no AS

- Autorização do cliente para o serviço pretendido no TGS

- Pedido do serviço pelo cliente no SS

# 9. Servidores de chaves

As autorizações dos serviços no TGS e a verificação no SS são controladas por data-hora (*timestamps*).

Por isto, os relógios das máquinas envolvidas devem estar razoavelmente sincronizados (diferença de menos de 5 minutos).

Uma explicação um pouco mais detalhada e compreensível pode ser vista em:

http://en.wikipedia.org/wiki/Kerberos_(protocol)

# 9. Servidores de chaves

A resposta do TGS ao pedido de autorização é chamada de *ticket*:

*Client-to-server ticket* (inclui o ID do cliente, o endereço IP do cliente, o período de validade e a chave secreta de sessão de *Cliente/Servidor*) criptografado com a chave secreta do servidor SS.

# 9. Servidores de chaves

Um protocolo semelhante que se inicia com uma simples senha (humanamente memorizável) é o

SRP – Secure Remote Password

Referências:

http://srp.stanford.edu/

http://en.wikipedia.org/wiki/Secure_Remote_Password_protocol

# 9. Servidores de chaves

A biblioteca de segurança e criptografia de JAVA possui um conjunto de serviços chamado:

JAAS – *Java Authentication and Authorization Service*

que é baseado na tecnologia do KERBEROS.

# 10. Certificados e PKI

Resumindo o que vimos até agora, temos:

Quatro ferramentas:

- Criptografia simétrica com chaves secretas,
- Funções Hash (funções de resumo),
- Autenticação de mensagens (MAC)
- Criptografia de chave pública

e técnicas para a especificação e implementação de protocolos criptográficos.

# 10. Certificados e PKI

O projeto e o uso destes protocolos criptográficos deve ser feito em:

- um ambiente distribuído,
- com um grande número de usuários
- nos quais só temos uma confiança parcial

Para isto, necessita-se de uma infraestrutura para a distribuição e o gerenciamento de chaves (públicas e privadas), na forma de servidores nos quais se CONFIA.

# 10. Certificados e PKI

A necessidade de se ter uma infraestrutura de grande alcance (pública ou dentro de um grupo ou empresa) levou à idéia de PKI (Public Key Infrastructure).

Uma PKI é um conjunto de hardware, software, pessoas, políticas e procedimentos necessários para criar, gerenciar, distribuir, usar, armazenar e revogar certificados digitais.

# 10. Certificados e PKI

Um certificado digital basicamente liga uma chave pública a uma pessoa, organização ou sistema.

Uma PKI possui uma autoridade central chamada *Certificate Authority* (CA).

Uma CA é um servidor de chaves. Como tal, ela possui uma chave pública própria, que ela publica (e um chave privada mantida em segredo) .

# 10. Certificados e PKI

Os usuários da CA depositam certa confiança no servidor, cujo endereço é conhecido e divulgado.

Um usuário (Alice) da CA que quiser publicar uma chave pública, deposita-a no CA (através a *Registration Authority*) assim como sua identidade.

# 10. Certificados e PKI

O registro da identidade pode ser feito com maior ou menor rigor, correspondendo a uma maior ou menor confiança.

A CA cria um documento assinado (com um MAC) que:

- divulga a chave pública de Alice

e que

- certifica que esta chave pertence a Alice.

Este documento é o Certificado.

# 10. Certificados e PKI

Outros usuários (Bob) podem obter a chave pública de Alice, confiando no Certificado, e na CA, que a chave realmente pertence a Alice.

Um Certificado digital é simplesmente um tipo de dado, mas ele deve ter um formato único, para poder ser facilmente autenticado.

Formatos variáveis, como o XML, só podem ser usados com cuidados próprios.

# 10. Certificados e PKI

Por isto, existe um padrão (ITU-T) para certificados digitais: o X.509

O padrão do X.509 é bastante complexo e muitas organizações criam suas próprias CAs e usam subconjuntos do X.509 ou definem seus próprios formatos para certificados digitais.

# 10. Certificados e PKI

**Exemplos de PKI**

A PKI universal: seria gerenciada por uma organização mundial, acreditada por todos, como a ONU, que publicaria e certificaria as chaves públicas de todos.

Tal idéia nunca foi inplementada ou tentada e por todas as razões nunca existirá.

# 10. Certificados e PKI

## Acesso a uma VPN:

Uma companhia que disponibiliza o uso de uma VPN para seus funcionários trabalhando fora da empresa.

O departamento de TI da empresa mantem uma CA para permitir que as máquinas dos funcionários e os pontos de acesso da empresa se autentiquem e iniciem uma troca de mensagens criptografadas.

# 10. Certificados e PKI

## Acesso eletrônico a bancos (*Electronic Banking*):

O próprio banco mantem uma CA para garantir:

• por um lado, a confiança de que o cliente está realmente se comunicando com o banco,

• por outro lado, a autenticidade do cliente e para produzir uma prova que evite a repudiação.

# 10. Certificados e PKI

## Uma empresa de cartões de crédito:

É uma empresa que se baseia na cooperação entre muitos bancos espalhados pelo mundo todo.

Um pagamento de um cliente que usa o banco A deve ser feito para o comerciante que usa o banco B.

Os bancos A e B devem realizar transações seguras de pagamento e a empresa do cartão pode manter sua própria CA para certificar as chaves dos bancos A e B.

# 10. Certificados e PKI

## Certificados multinível

Quando o número de usuários de um CA cresce, surge a idéia de criar uma estrutura hierárquica de CAs.

O certificado correspondente terá então várias certificações, uma para cada nível da hierarquia.

# 10. Certificados e PKI

Um certificado possui então uma cadeia de certificações (*certification chain*) onde cada certificação certifica a certificação de nível abaixo.

Embora a idéia de uma hierarquia de CAs ajude a gerenciar uma grande quantidade de usuários, ela introduz mais complexidade, o que nunca é bom quando se trata de segurança.

# 10. Certificados e PKI

## Expiração

Toda chave criptográfica deve ser trocada regularmente.

Um certificado, que define uma chave pública, também deve ter datas de expiração e eventualmente uma data de início de validade.

Por isto, os protocolos que  utilizam os certificados devem ter acesso a um relógio seguro.

# 10. Certificados e PKI

## Comparação entre um Servidor de Chaves e uma PKI.

- Um Servidor de Chaves necessita que todos os usuários estejam online para usar as chaves. Uma PKI não.
- Uma PKI é mais facilmente distribuída.
- Uma PKI provê não-repudiação (mais facilmente).
- Uma estrutura hierárquica de CAs não precisa que o CA raiz esteja sempre online.

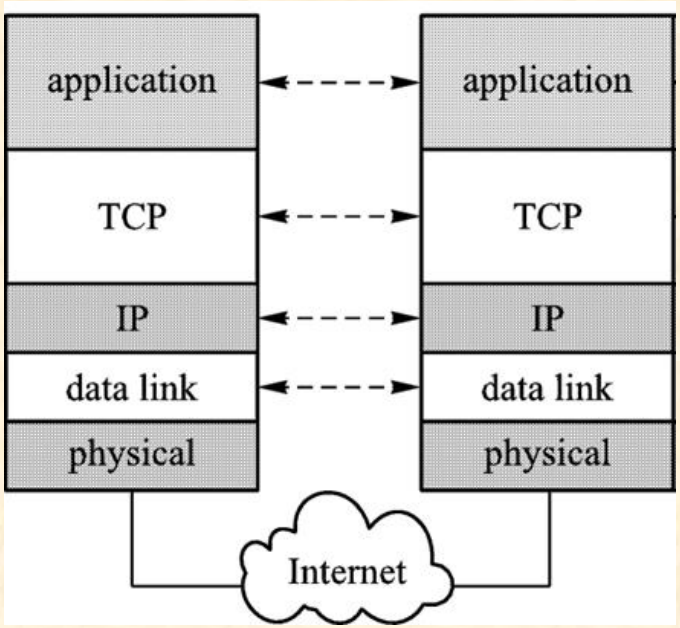
# 10. Certificados e PKI

• Um servidor de chaves é mais simples e por isto, possivelmente mais seguro.

Conclui-se que a escolha depende da especificação do sistema.

# 11. Protocolos de rede

Considerando o modelo em camadas para as redes de computadores, podemos analisar o uso de criptografia em cada um deles.



# 11. Protocolos de rede

• Criptografia na camada de aplicação: proteção fim-a-fim. A aplicação faz todo o trabalho de criptografia. As outras camadas não são alteradas.

• Criptografia na camada de transporte: o pacote inteiro ou só o corpo. O roteamento IP não é alterado. A aplicação também não é alterada.

# 11. Protocolos de rede

• Criptografia na camada de rede: o pacote inteiro (modo túnel) ou só o corpo (modo de transporte). Proteção enlace-a-enlace.

• Criptografia na subcamada enlace: usada por exemplo em redes wireless.

# 11. Protocolos de rede

Os protocolos criptográficos de rede mais difundidos são:

- VPN
- IPsec
- SSL/TLS
- HTTPS
- PGP
- S/MIME
- SSH
- WEP
- WPA e WPA2
- BLUETOOTH

# 11. Protocolos de rede

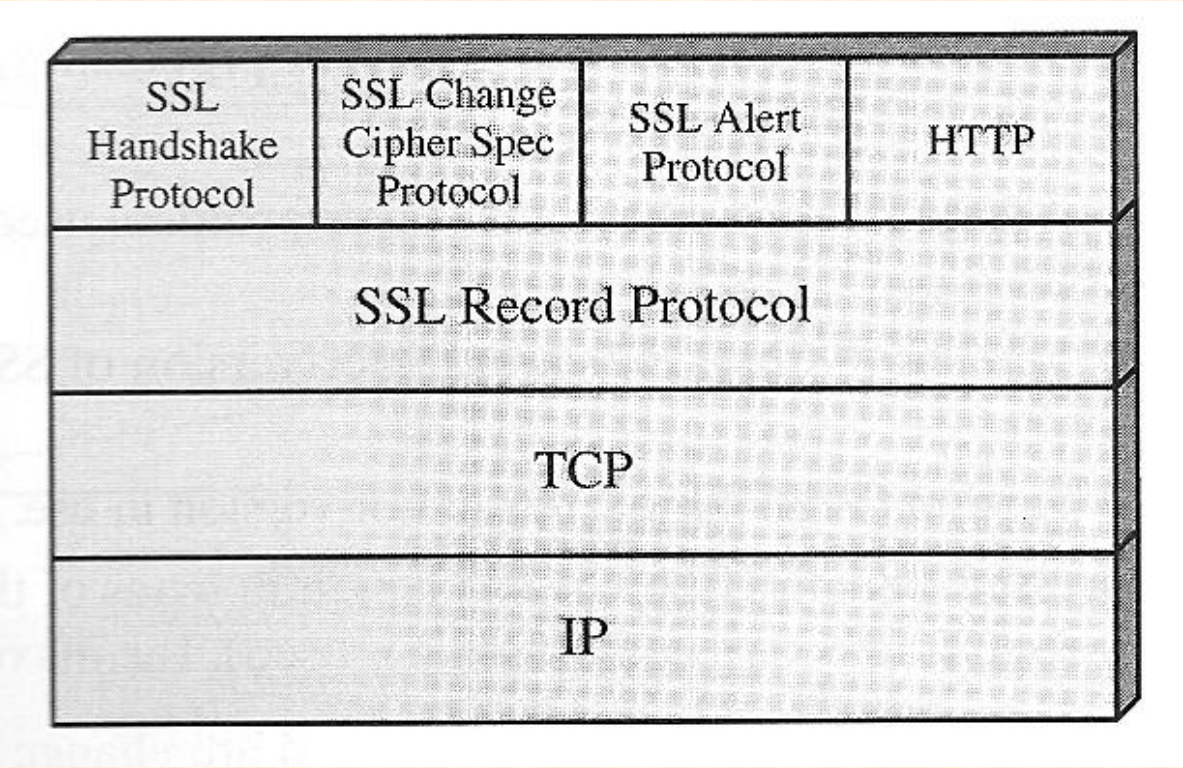
## SSL/TLS - *Secure Sockets Layer* e *Transport Layer Security*

SSL/TLS é um protocolo criptográfico com duas camadas que fica entre a camada de transporte (TCP) e a camada de aplicação.

Na verdade ele pode ser visto como um protocolo de aplicação.

# 11. Protocolos de rede

Sua organização é a mostrada na figura abaixo:



# 11. Protocolos de rede

Ele é usado em várias aplicações, entre as quais o HTTP, para formar o HTTPS, e para criar VPNs (Virtual Private Network), na OpenVPN.

Há dois conceitos importantes no SSL/TLS:

• Conexão: presta um serviço de transporte para a entidade acima.Toda conexão é associada a uma sessão.

• Sessão: são criadas pelo protocolo SSL Handshake, que define os parâmetros criptográficos. Estes parâmetros serão usados pelas conexões desta sessão.

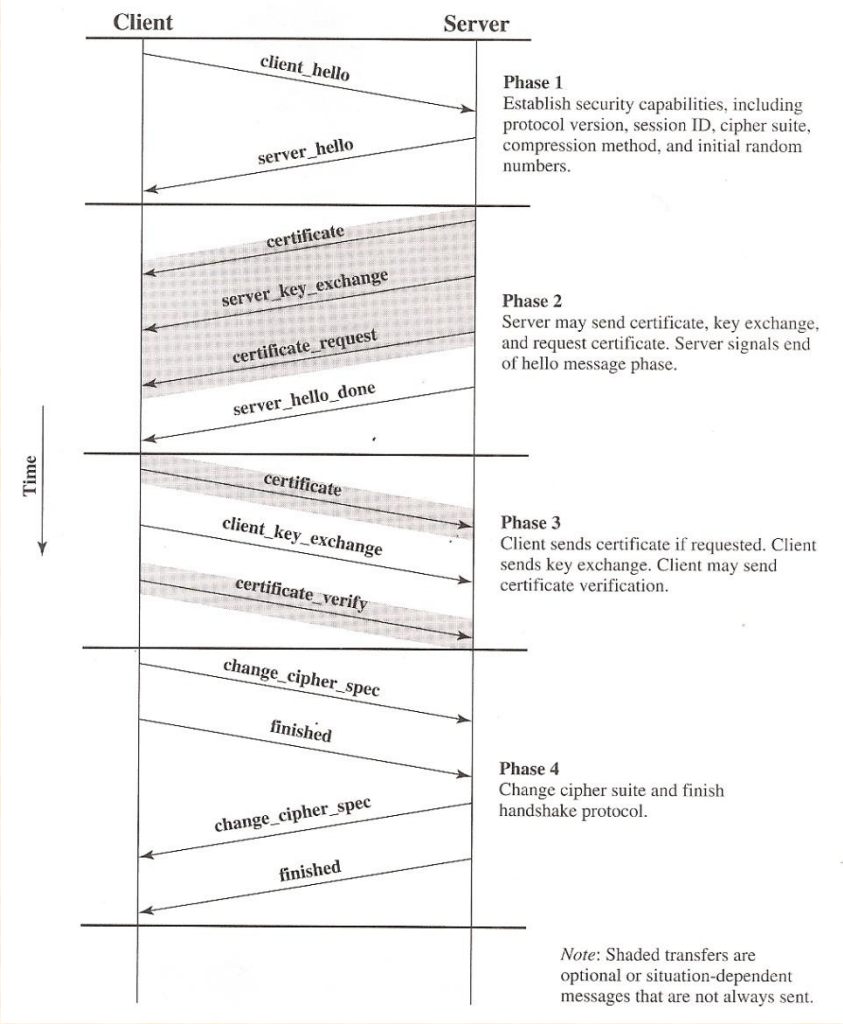
# 11. Protocolos de rede

## Handshake Protocol

Permite que o cliente e o servidor se autentiquem, negociem os parâmetros criptográficos e permitam o início da troca de mensagens criptografadas.

O protocolo tem 4 fases, mostradas na figura a seguir.

# 11. Protocolos de rede



# 11. Protocolos de rede

## Change Cipher Spec Protocol

É um protocolo usado na fase 4 do Handshake Protocol, para atualizar os dados criptográficos que serão utilizados na sessão.

## Alert Protocol

É usado para enviar alertas relacionados com o SSL/TLS à entidade par.

Exemplos são erro no MAC e fim de conexão (close_notify).

# 11. Protocolos de rede

## SSL Record Protocol

Provê dois serviços para uma conexão:

Confidencialidade, através criptografia com encriptação  simétrica.

Integridade, através a autenticação (MAC).

A operação deste protocolo pode ser vista na figura a seguir.

# 11. Protocolos de rede