

Sistemas Operativos II

Segurança e SD (parte 2)

Algoritmos Assimétricos

- ◆ par de chaves pública e privada
 - ◆ a chave **pública** é divulgada e a **privada** mantida em segredo
- ◆ baseados em funções *trap-door*
 - ◆ “função one-way com escapatória”
 - ◆ inversa é muito difícil de calcular, excepto com o conhecimento de um segredo
- ◆ na encriptação usa-se a chave pública do destinatário (para **confidencialidade**)
- ◆ a desencriptação só é possível com a chave privada
- ◆ Vantagem: **não há** necessidade de
 - ◆ confiar uma chave secreta a outro interveniente (que a pode difundir)
 - ◆ mecanismo seguro de distribuição de chaves secretas
- ◆ Computacionalmente mais pesados que os simétricos, devido às operações com n°s primos elevados
- ◆ Exemplo: RSA

Algoritmos Assimétricos: RSA

- ◆ Rivest, Shamir and Adleman (RSA), 1978
- ◆ princípio
 - ◆ encriptação: baseada na multiplicação de n^os primos elevados
 - ◆ é computacionalmente inviável tentar fatorizar o resultado (para tentar descobrir os multiplicandos primos a partir dos quais se geram as chaves)
 - ◆ cada bloco de plaintext é tratado como um inteiro que vai ser alterado com operações potência e módulo
 - ◆ descriptação: *trap door function*
 - ◆ é necessária outra chave do par*
 - ◆ operações de potência e módulo

* - a chave usada depende do propósito (confidencialidade/assinatura – ver adiante)

Algoritmos Assimétricos: RSA

Encontrar um par de chaves (e, d) :

1. Escolher dois n° primos grandes, P e Q (maiores que 10^{100}), e obter:

$$N = P \times Q$$

$$Z = (P-1) \times (Q-1)$$

2. Para d escolher qualquer n° que, juntamente com Z , sejam primos entre si (sem divisores comuns).

Exemplo elucidativo (para valores pequenos P and Q):

$$P = 13, Q = 17 \rightarrow N = 221, Z = 192$$

$$d = 5$$

3. Para obter e resolve-se a equação:

$e \times d = "1 \text{ mod } Z" =$ menor elemento divisível por d na série $Z+1, 2Z+1, 3Z+1, \dots$

$$e \times d = "1 \text{ mod } 192" = 1, 193, 385, \dots$$

385 é divisível por d

$$e = 385/5 = 77$$

Algoritmos Assimétricos: RSA

- ◆ Para encriptar um texto com RSA:
 - ◆ dividir o plaintext em blocos de comprimento k bits, onde $2^k < N$
 - ◆ de tal forma que o nº composto por aqueles bits seja $< N$
 - ◆ usualmente: $512 \leq k \leq 1024$
no exemplo: $k = 7$, uma vez que $2^7 = 128$
 - ◆ Função para encriptar um bloco do plaintext M é
$$E'(e, N, M) = M^e \bmod N$$

para uma mensagem M , o ciphertext é $(M^{77} \bmod 221)$
 - ◆ Desencriptar o bloco cifrado c para obter o bloco original:
 - ◆ $D'(d, N, c) = c^d \bmod N$

Algoritmos Assimétricos: RSA

Rivest, Shamir e Adelman provaram que E' e D' são **inversas** (isto é, $E'(D'(x)) = D'(E'(x)) = x$) para todos os valores de P onde $0 \leq P \leq N$.

Os parâmetros e, N consideram-se a chave da função de encriptação e d, N a chave da função de descriptação

$$K_e = \langle e, N \rangle \text{ e } K_d = \langle d, N \rangle$$

Função de Encriptação:

$$E(K_e, M) = \{M\}_K$$

notação que significa que a mensagem encriptada só pode ser decifrada por quem tiver a chave correspondente do mesmo par K_d)

Função de Descriptação:

$$D(K_d, \{M\}_K) = M$$

Algoritmos Assimétricos: RSA

- ♦ algoritmo dispendioso do ponto de vista computacional, mesmo para pequenas mensagens M
- ♦ utilizado com chaves de comprimento maior (2048 bits...)
- ♦ *chosen plaintext attack*
 - ♦ como dispõe da chave pública, o atacante pode gerar mensagens para encriptar e comparar com o ciphertext que pretende decifrar. Vai gerando sucessivas mensagens plaintext até acertar no ciphertext.
 - ♦ (Não precisa da chave privada) Defesa: usar mensagens de comprimento superior à chave **privada**. Assim o ataque é mais difícil que um ataque à chave.
- ♦ fatorizar 10^{200} com 1 milhão de instruções por segundo e com o melhor algoritmo de 1978 demoraria 4 mil milhões de anos

Comunicação com chaves públicas

Bob has a public/private key pair $\langle K_{Bpub}, K_{Bpriv} \rangle$

1. Alice obtains Bob's public key K_{Bpub}
2. Alice creates a new shared key K_{AB} , encrypts it using K_{Bpub} using a public-key algorithm and sends the result to Bob.
3. Bob uses the corresponding private key K_{Bpriv} to decrypt it.

- ◆ Mallory pode interceptar o pedido inicial de Alice
 - ◆ Interfere enviando a sua chave pública no lugar da chave pública de Bob
 - ◆ Pode enganar ambos os interlocutores (Alice,Bob) ficando no meio das mensagens de ambos, dissimulado
 - ◆ *Man In The Middle Attack*

Algoritmos Híbridos

- ◆ resolvem o problema de exigência computacional dos algoritmos assimétricos
- ◆ robustos
- ◆ combinam técnicas de encriptação simétrica e assimétrica
 - ◆ criptografia de chave pública para autenticar os intervenientes e para transmissão de chaves secretas
 - ◆ algoritmos simétricos de chave secreta para restante encriptação
- ◆ ex: PGP, SSL

Algoritmos Híbridos: PGP

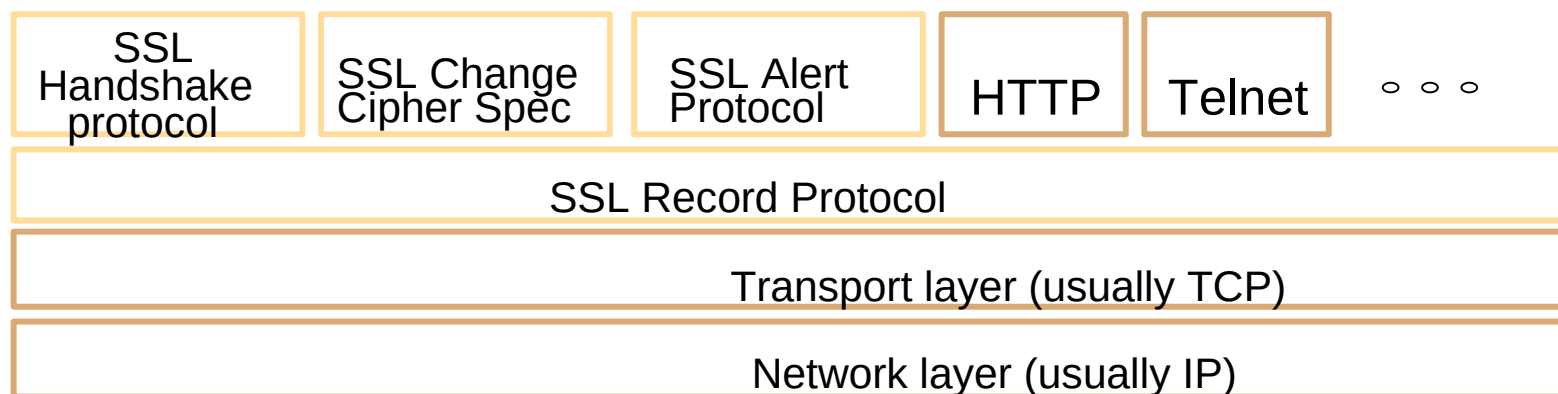
- ◆ Pretty Good Privacy (PGP)
 - ◆ RSA (criptografia de chave pública) para autenticação e transmissão de chave secreta
 - ◆ IDEA, 3DES (criptografia de chave secreta) para encriptar documentos
- ◆ Atualmente há várias ferramentas PGP com vários tipos de funcionalidades/algoritmos disponíveis
 - ◆ E-mail...

Algoritmos Híbridos: SSL

- ◆ Secure Sockets Layer (SSL) - Netscape Corporation, 1996
 - ◆ mecanismo híbrido: autenticação e troca de chaves secretas via criptografia de chave pública
 - ◆ TLS: uma norma que resulta da extensão do SSL
 - ◆ requer apenas certificados de chave pública atribuídos por uma CA reconhecida por ambas as partes
 - ◆ APIs do protocolo disponíveis em Java (e outras linguagens e ferramentas)
- ◆ Permite
 - ◆ **negociação dos algoritmos de autenticação e encriptação** (facilita a comunicação entre plataformas distintas num sistema heterogéneo – *porque podem negociar algoritmos suportados por ambas as partes*)
 - ◆ estabelecimento de canal seguro sem contacto prévio ou ajuda de terceiros
 - ◆ os certificados usados devem ser emitidos por entidades reconhecidas por ambas as partes
 - ◆ o nível de segurança é acordado, em cada sentido. Pode haver apenas autenticação de uma das partes, por exemplo (ou então das duas).

Algoritmos Híbridos: SSL

- ◆ Duas Camadas (*ao nível da camada de sessão do modelo OSI*)
 - ◆ Handshake Layer
 - ◆ negociação de algoritmos, geração e envio de chaves
 - ◆ SSL Record Protocol Layer
 - ◆ encriptação de mensagens e autenticação através de um protocolo com conexão (ex: TCP)
 - ◆ **pode garantir: integridade, confidencialidade e autenticação da fonte**
mas depende da configuração usada



SSL protocols: ☐

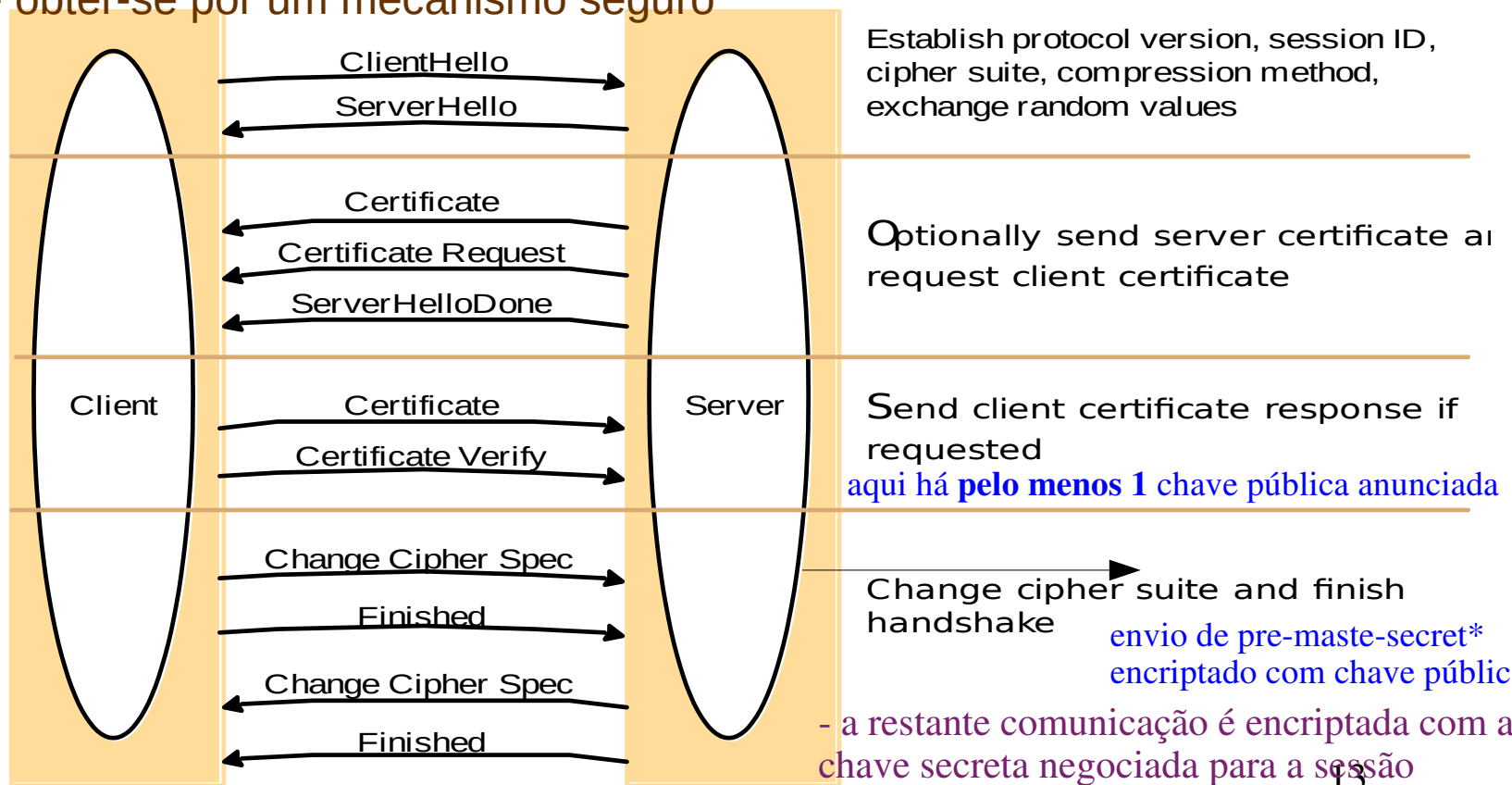
Other protocols: ☐

Algoritmos Híbridos: SSL

◆ Handshake

◆ vulnerável a *man-in-the-middle*

- ◆ Para evitar isso: a chave pública para validar o certificado do interlocutor deve obter-se por um mecanismo seguro



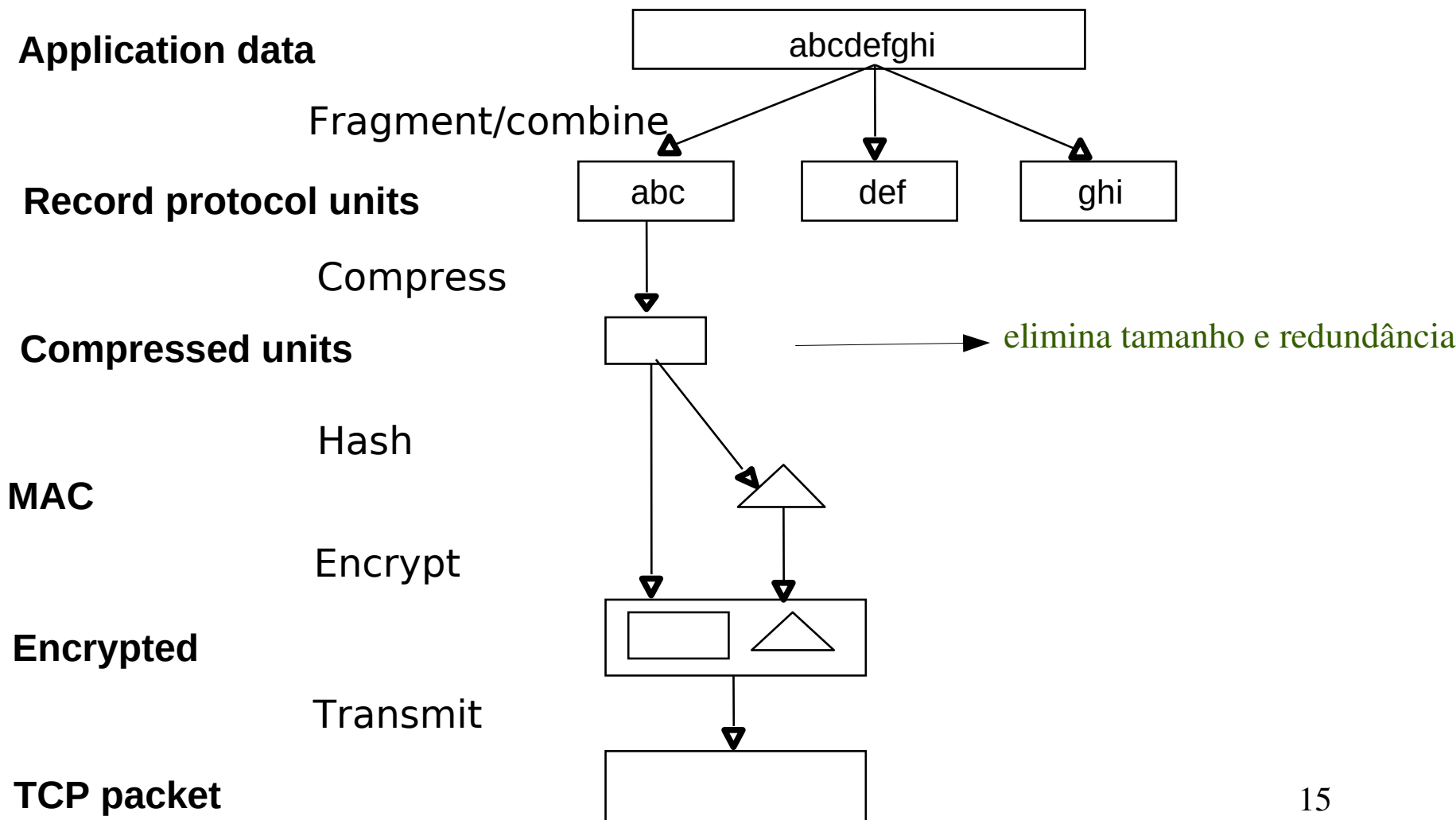
Algoritmos Híbridos: SSL

◆ opções configuradas no handshake

| <i>Component</i> | <i>Description</i> | <i>Example</i> |
|--------------------------|---|----------------------------------|
| Key exchange method | the method to be used for exchange of a session key | RSA with public-key certificates |
| Cipher for data transfer | the block or stream cipher to be used for data | IDEA |
| Message digest function | for creating message authentication codes (MACs) | SHA |

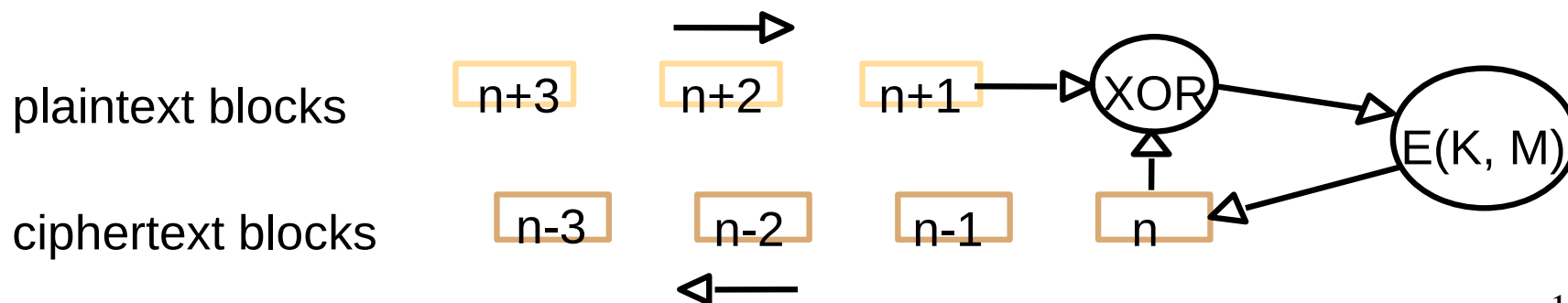
Algoritmos Híbridos: SSL

◆ camada SSL record: a transmissão dos dados



Encriptação de Blocos: **Block Cipher**

- ◆ encriptar uma mensagem em blocos independentes
 - ◆ integridade da mensagem não é garantida sem um hash ou checksum
 - ◆ o atacante pode reconhecer padrões nos blocos de ciphertext e relacionar com o plaintext
- ◆ cipher block chaining (CBC)
 - ◆ cada bloco é combinado com o ciphertext precedente (xor) antes de ser encriptado.
 - ◆ Para decifrar, o bloco descriptado é XOR-ed com o ciphertext do bloco anterior, resultando o formato inicial do bloco.



Encriptação de Blocos - CBC

- ◆ Desvantagem

- ◆ pode ser usado apenas em **canais fiáveis**
 - ◆ Se um bloco se perder não será possível decifrar os restantes

- ◆ Vantagem

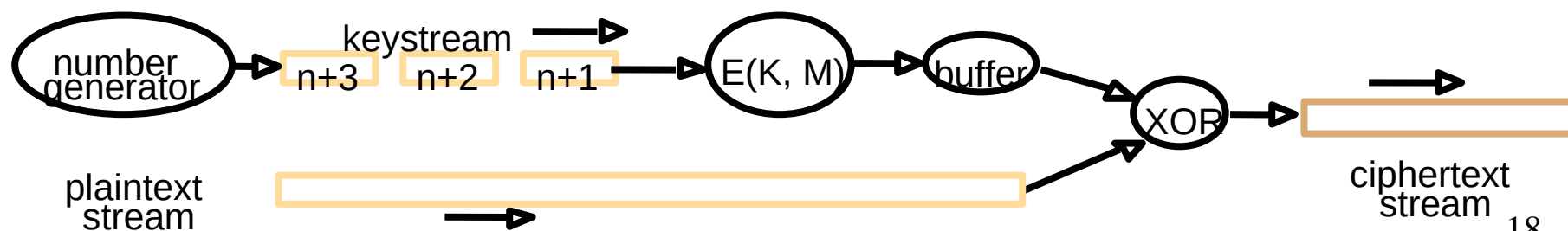
- ◆ Dois blocos iguais de *plaintext* **não** terão o mesmo *ciphertext*

- ◆ Mesma mensagem, 2 destinos

- ◆ A transmissão será a mesma (o que poderia dar pistas a um adversário)
- ◆ Excepto se se adicionar um valor inicial antes dos dados, distinto para cada destino
 - ◆ *Initialization Vector*

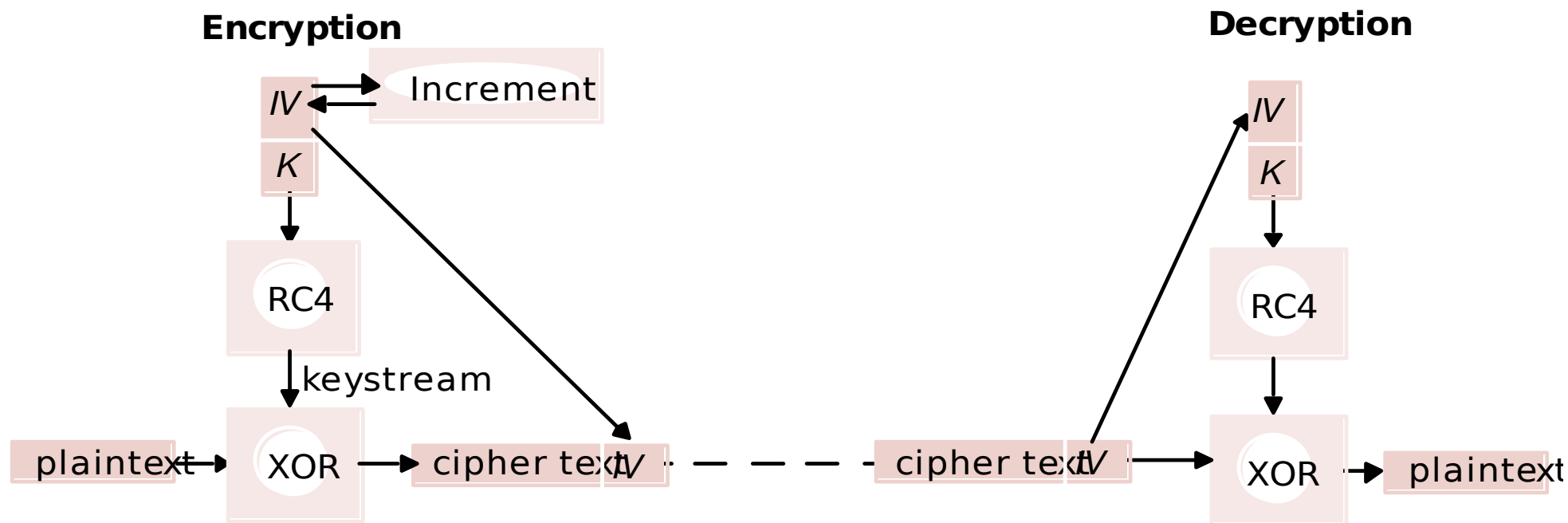
Encriptação de uma **Stream**: *Stream Cipher*

- ♦ usado para transmissões em tempo real, quando não se pode esperar para completar um bloco
- ♦ é gerada uma **sequência** de n°s, para gerar blocos que são encriptados com chave secreta e que depois são **XOR-ed** com o **plaintext** disponível
 - ♦ os blocos de *keystream* depois de encriptados podem ainda ser encadeados como CBC
- ♦ receptor: conhece a sequência e sabe como desencriptar a keystream. Usa o XOR para recuperar o plaintext
 - ♦ NOTA: $((a \text{ XOR } b) \text{ XOR } b) == a$
- ♦ nº inicial combinado entre sender e receiver
- ♦ Suporta variação no volume de dados ao longo do tempo e faz um tratamento rápido dos dados (XOR ; a keystream pode ser preparada antes)



RC4 stream cipher in IEEE 802.11 WEP

- ◆ IEEE 802.11 – rede sem fios, WiFi
 - ◆ Dados em trânsito vulneráveis a qualquer dispositivo no alcance da transmissão
- ◆ WEP: wired equivalent privacy



802.11 WEP

- ◆ WEP: wired equivalent privacy (*versão base*)
 - ◆ Problemas
 - ◆ Partilha da chave é um ponto de risco
 - ◆ Solução: usar criptografia de chave pública para negociar e transmitir chaves individuais, como acontece em TLS/SSL
 - ◆ O ponto de acesso não era autenticado
 - ◆ Atacante com K podia fazer spoof & masquerading, controlando os dados em tráfego...
 - ◆ Solução: autenticação do ponto de acesso com C. de chave pública
 - ◆ Problemas com o stream cipher keystream reset
 - ◆ Se há perda de pacotes, o reset/sincronização pode dar pistas ao atacante
 - ◆ Chaves de 40 e de 64 bits vulneráveis a ataques de força bruta
 - ◆ Solução: chaves de 128 bits
 - ◆ RC4 stream cipher tem características que comprometem o secretismo da chave (mesmo que de 128 bits)
 - ◆ Solução: permitir a negociação das especificações da cifra, como em TLS

Algoritmos de Autenticação

- ◆ **autenticação** de um ou dois interlocutores/peers/participantes

Algoritmos

- ◆ Needham-Schroeder
- ◆ Kerberos
- ◆ Baseados em *Tickets* e *challenges* (desafios)
 - ◆ ***Ticket***: mensagem encriptada pelo Servidor de Autenticação com uma chave do *principal*. Contém a identidade do interlocutor e a chave secreta gerada para usar na sessão.
 - ◆ ***Challenge***: transmissão de informação (ticket) de forma a que só o verdadeiro destinatário possa ler. O processo é encarado como um desafio, porque:
 - ◆ Vencer o desafio é conseguir decifrar a informação (e continuar o processo)
 - ◆ os atacantes são afastados/eliminados e não conseguem avançar

Algoritmos de Autenticação: Needham–Schroeder

- ◆ 1978, com o surgimento dos *network file services*

há um servidor de autenticação, S, que conhece a identificação e a **chave** secreta de cada *principal* no sistema

Essa chave secreta é conhecida apenas pelo *principal* e pelo servidor **S**, servindo para autenticação do *principal* junto do servidor e para cifrar mensagens entre os mesmos

Nonce: valor inteiro que se adiciona a uma mensagem para demonstrar que é (ou que não é) recente

Algoritmos de Autenticação: Needham–Schroeder

| Header | Message | Notes |
|---------|---|---|
| 1. A→S: | A, B, N_A | A requests S to supply a key for communication with B. |
| 2. S→A: | $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_B}\}_{K_A}$ | S returns a message encrypted in A's secret key, containing a newly generated key K_{AB} (<i>session key</i>) and a 'ticket' encrypted in B's secret key. The nonce N_A demonstrates that the message was sent in response to the preceding one. A believes that S sent the message because only S knows A's secret key. |
| 3. A→B: | $\{K_{AB}, A\}_{K_B}$ | A sends the 'ticket' to B. |
| 4. B→A: | $\{N_B\}_{K_{AB}}$ | B decrypts the ticket and uses the new key K_{AB} to encrypt another nonce N_B . |
| 5. A→B: | $\{N_B - 1\}_{K_{AB}}$ | A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of N_B . |

dificuldade: S ter conhecimento prévio das chaves de A e B

vulnerabilidade: B não sabe se (3) é recente ou um *replay*

- ◆ **solução:** usar um timestamp t à mensagem $\{K_{AB}, A, t\}_{K_B}$; assim B pode verificar se a mensagem é atual (kerberos)

Funções Seguras de Hash ou Digest

- ♦ uma **função de digest** $h=H(M)$ é **segura** se:
 - ♦ dado M é fácil calcular h
 - ♦ dado h é inviável/difícil calcular M
 - ♦ dado M , é muito difícil encontrar $M' \neq M$ tal que $H(M)=H(M')$
- ♦ tais funções têm a propriedade *one-way*
- ♦ Como o *hash* tem um tamanho fixo, é possível encontrar mensagens naquelas condições... a probabilidade disso acontecer deve ser baixa)
- ♦ Se o *signer* conhecer duas mensagens M e M' com o mesmo digest poderá posteriormente alegar que enviou M' e não M , tendo ocorrido um erro de transmissão de M'

Funções Seguras de Hash ou Digest

◆ MD5 (1992)

- ◆ em quatro ciclos. Cada aplica uma função não linear a um dos 16 segmentos de 32 bits um bloco de 512 bits da mensagem original
- ◆ digest de 128 bits
- ◆ um dos algoritmos mais eficientes em uso atualmente

◆ SHA (1995)

- ◆ baseado no algoritmo MD5, introduzindo operações adicionais
 - ◆ digest de 160 bits
 - ◆ relativamente mais lento que o MD5 mas o tamanho do digest oferece mais garantias contra ataques de força bruta
- ◆ é possível usar um algoritmo simétrico de encriptação para gerar digests, mas nesse caso a chave usada será necessária para a validação
- ◆ *ver CBC*


Funções Seguras de Hash ou Digest

◆ Ferramentas

- ◆ md5sum
- ◆ Openssl (md5, sha... mais de 10 variantes de digest)
 - ◆ Por exemplo, para validar a integridade de um .iso obtido num download...

www.kubuntu.org/getkubuntu/download#download-block

MD5 Sums



Before burning a CD, it is highly recommended that you verify the MD5 sum (hash) of the ISO file. For instructions, please see [HowToMD5SUM](#).

Below is a list of MD5 sums to check with your downloaded file:

| | |
|---|-----------------------------------|
| <u>b0e1c4e2c6f996fa21f21afbf4224bdc</u> | kubuntu-11.04-alternate-amd64.iso |
| 05906ea0810a54f7245f75da4fb48bcb | kubuntu-11.04-alternate-i386.iso |
| 1559d08255c6d2a4a8868f0ab3485f43 | kubuntu-11.04-desktop-amd64.iso |
| 6226d0ae7ab35df955f1c07df285232f | kubuntu-11.04-desktop-i386.iso |