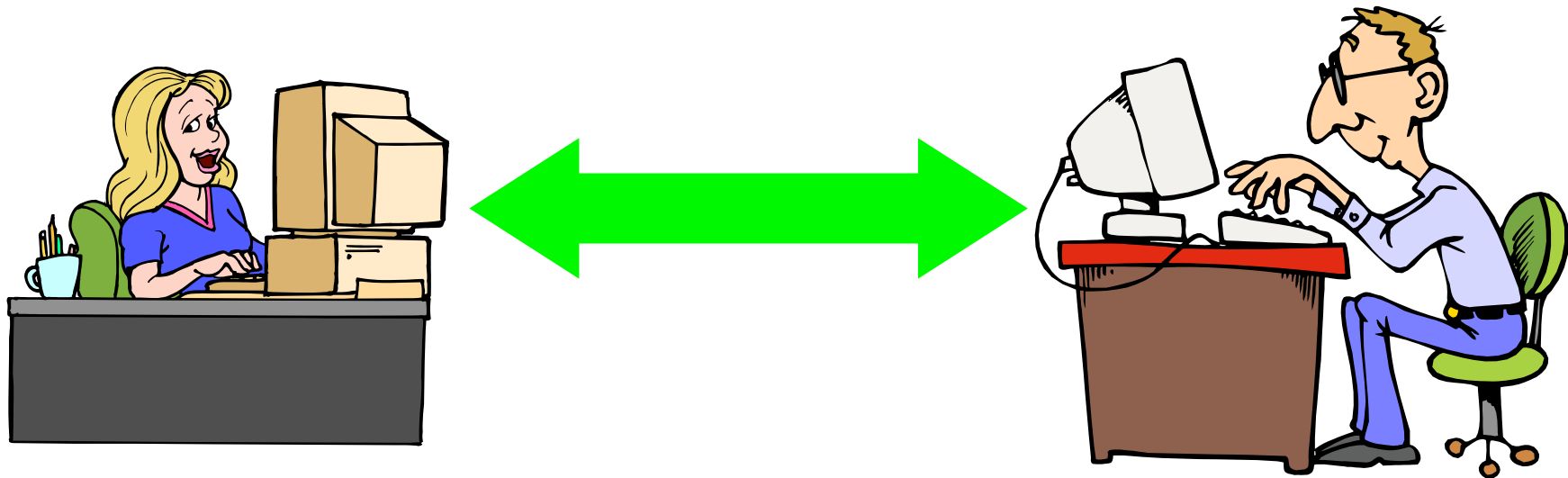


# **Segurança em Sistemas Distribuídos**

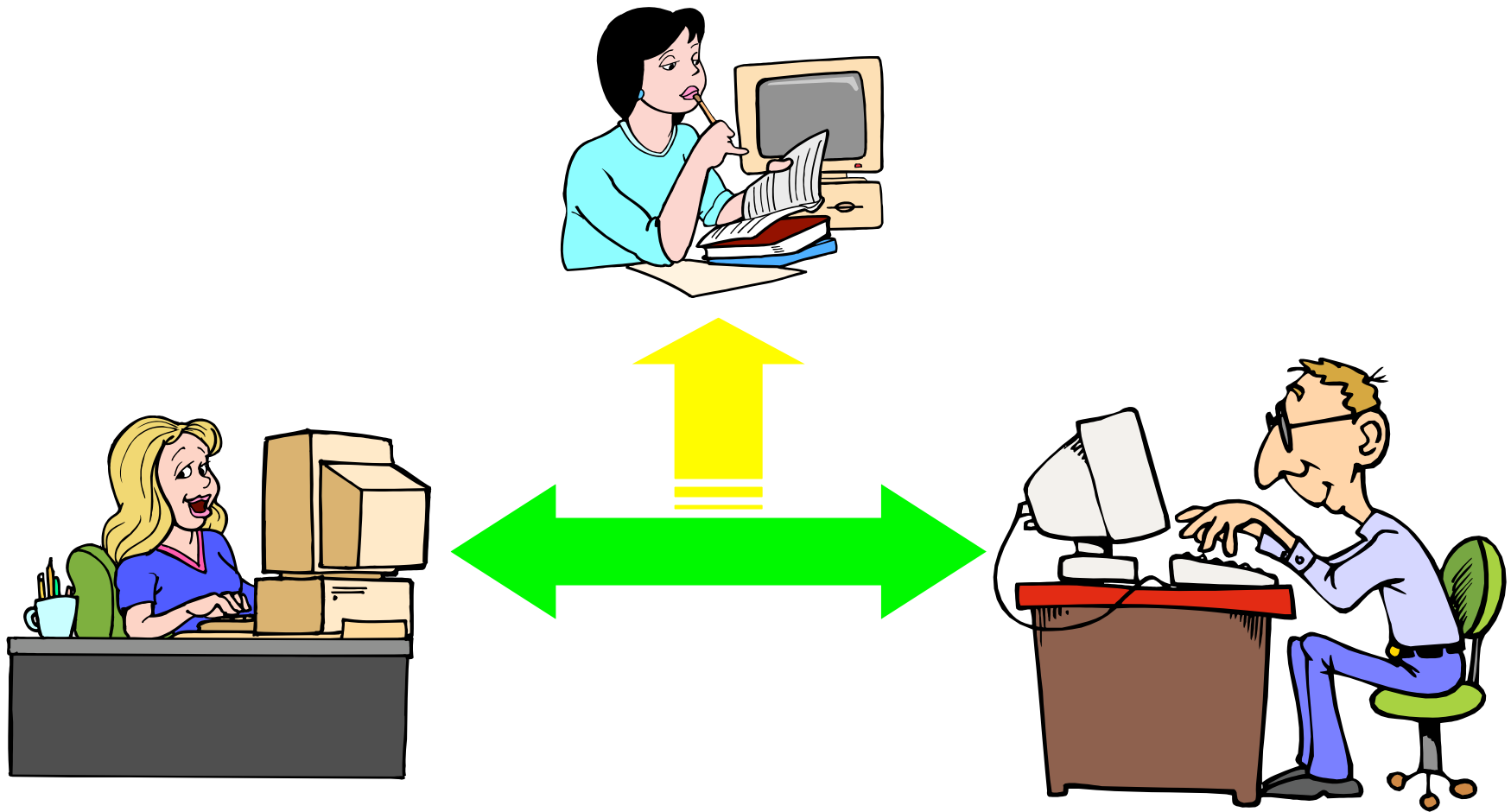
# Security Risks

- **Risks for Client and Server**
  - Eavesdropping (passive attack)
  - Tampering (malicious active attack)
  - Message suppression
  - Message replaying
- **Risks for the Client**
  - Active Content (Illegitimate use)
  - Integrity violation
  - Privacy attack
  - Impersonation/Masquerading
- **Risks for the Server**
  - Webjacking or Masquerading
  - Server Break-in (Backdoors, trojan horses, insider attacks)
  - Integrity violation
  - Impersonation (client spoofing)
  - Denial-of-service attacks

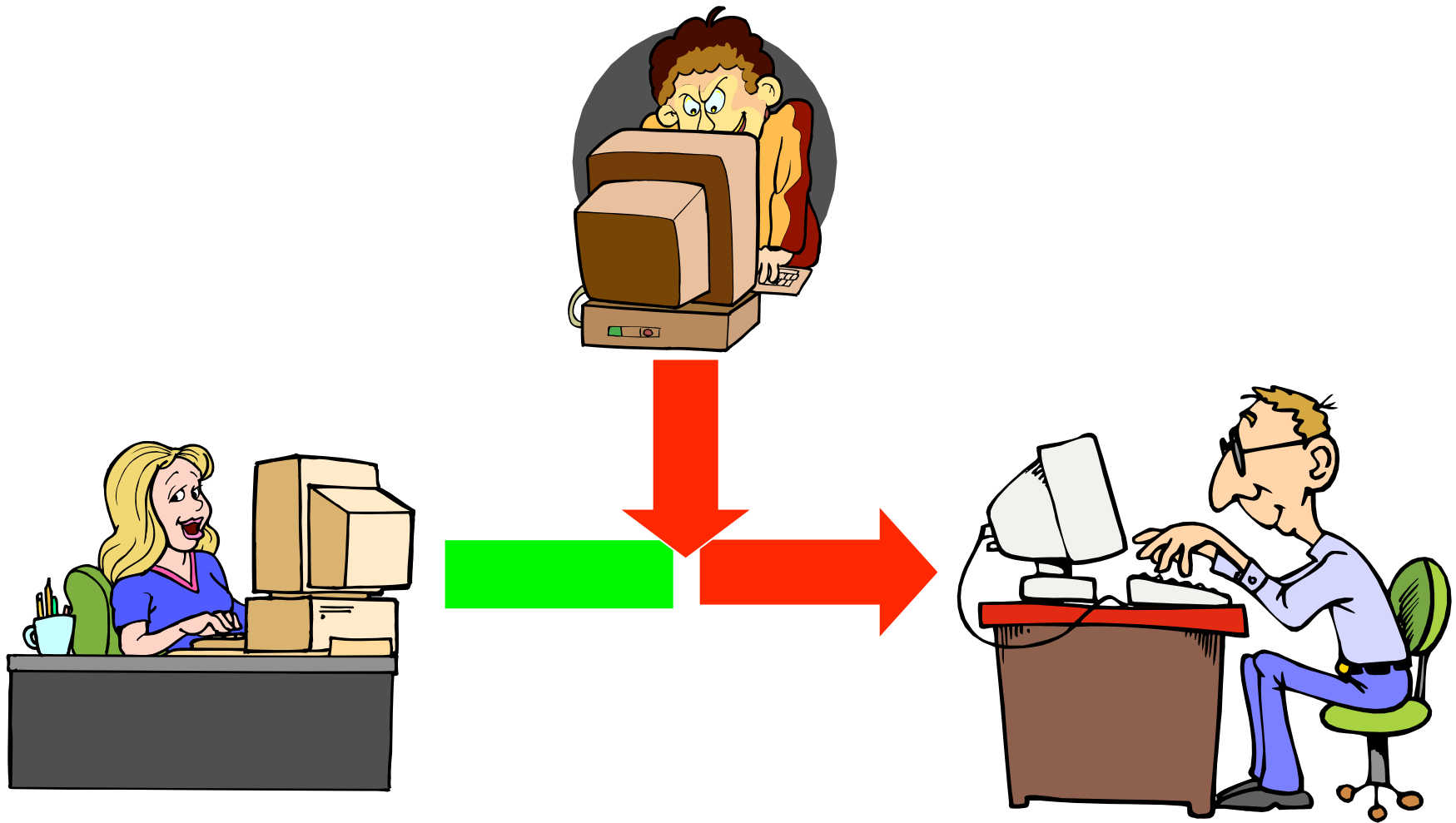
# Bob and Alice



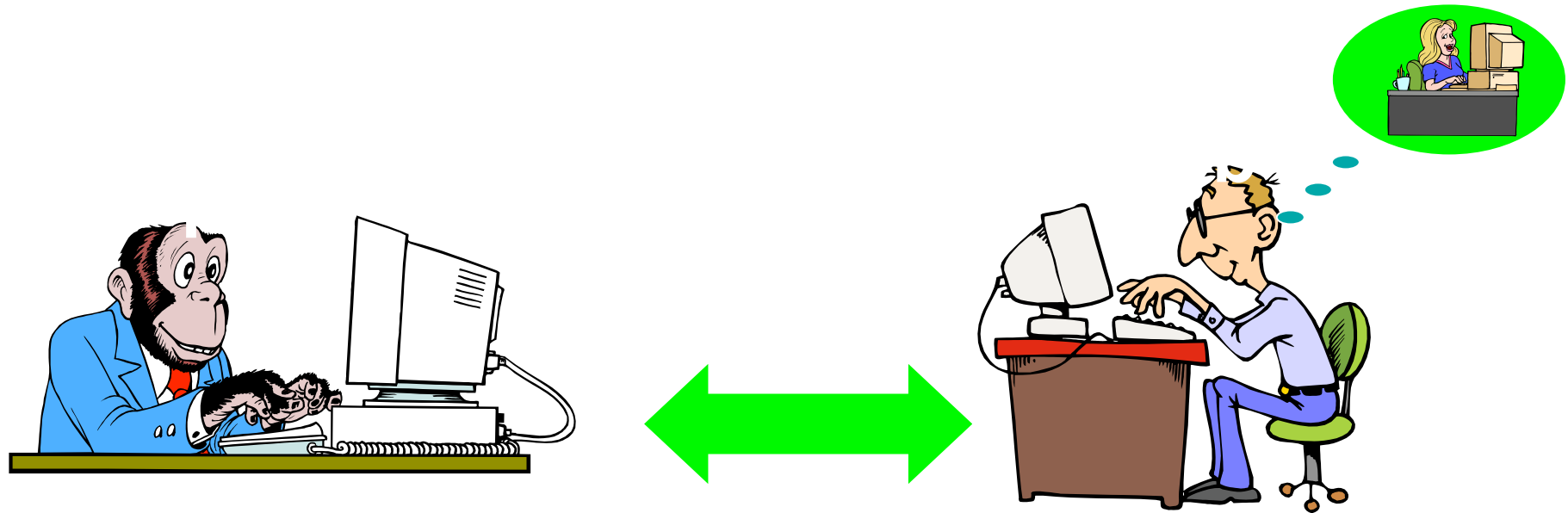
# Eavesdropping



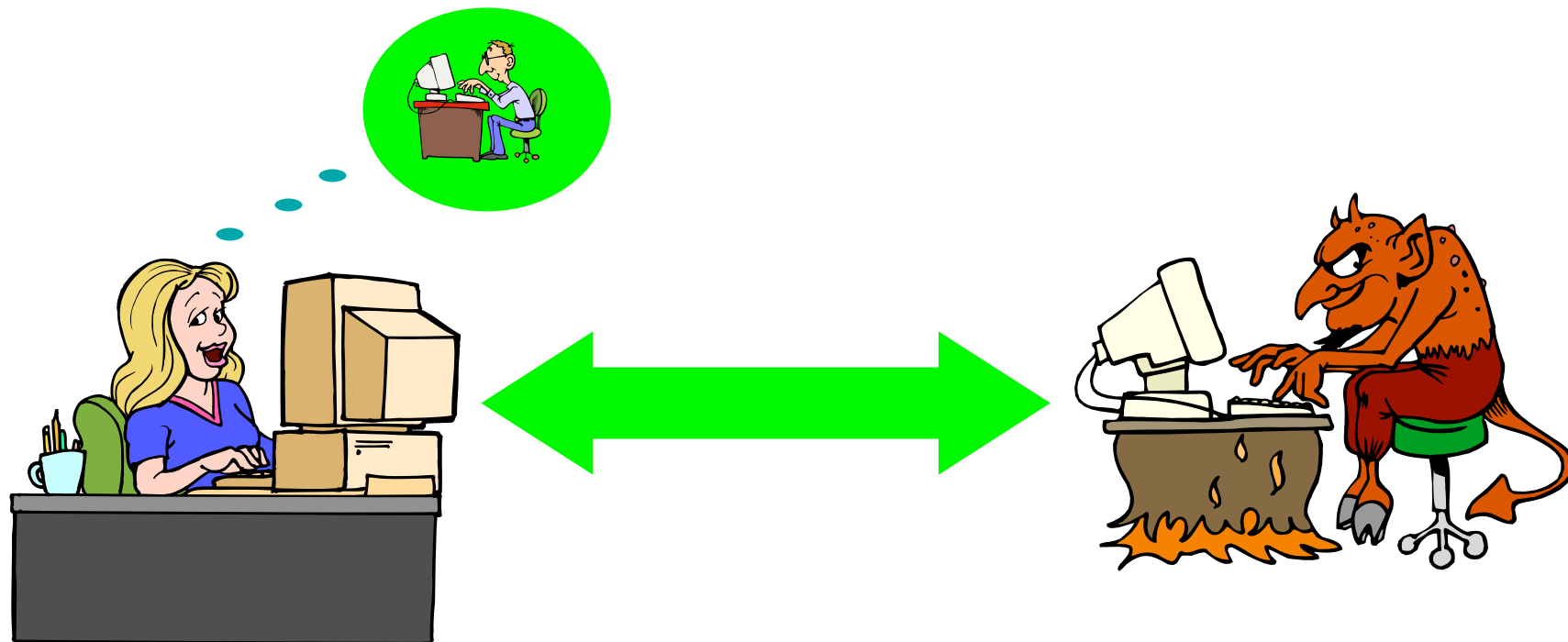
# Malicious Attack



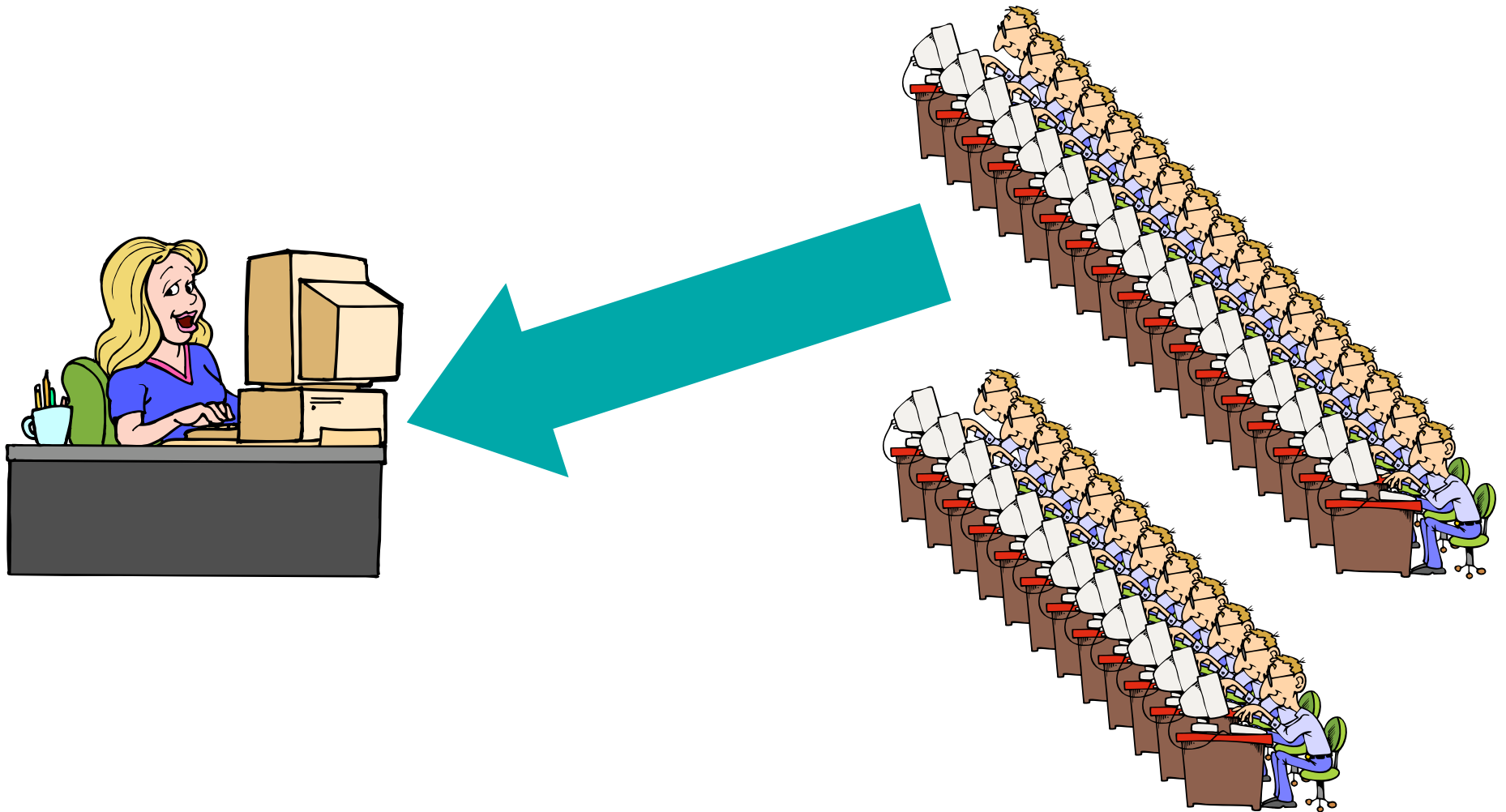
# Spoofting/Masquerading



# Server Masquerading



# Denial-of-Service Attacks





# Possible Attacks

- **Eavesdropping**
  - obtaining private or secret information
- **Masquerading**
  - assuming the identity of another user/principal
- **Message tampering**
  - altering the content of messages in transit
    - man in the middle attack  
(tampers with the secure channel mechanism)
- **Replaying**
  - storing secure messages and sending them at a later date
- **Denial of service**
  - flooding a channel or other resource, denying access to others

# Security Requirements

- **Confidentiality**
  - protection from disclosure from unauthorized people
- **Integrity**
  - data consistency
- **Authentication**
  - assurance of identity of person or data
- **Non-Repudiation**
  - originator of communication cannot deny it later
- **Availability**
  - Legitimate users have access when they need it
- **Access Control**
  - Unauthorized users are kept out

# Authentication of Anonymous Users



On the Internet nobody knows you're a dog...

# Security Mechanisms

- 1- **Encryption** is used to provide confidentiality, can provide authentication and integrity protection.
- 2- **Digital signatures** are used to provide authentication, integrity protection, and non-repudiation.
- 3- **Checksums/hash** algorithms are used to provide integrity protection, can provide authentication.

# Services, Mechanisms and Algorithms

**TLS**

**Signatures**

**DSA**

**RSA**

**Encryption**

**RSA**

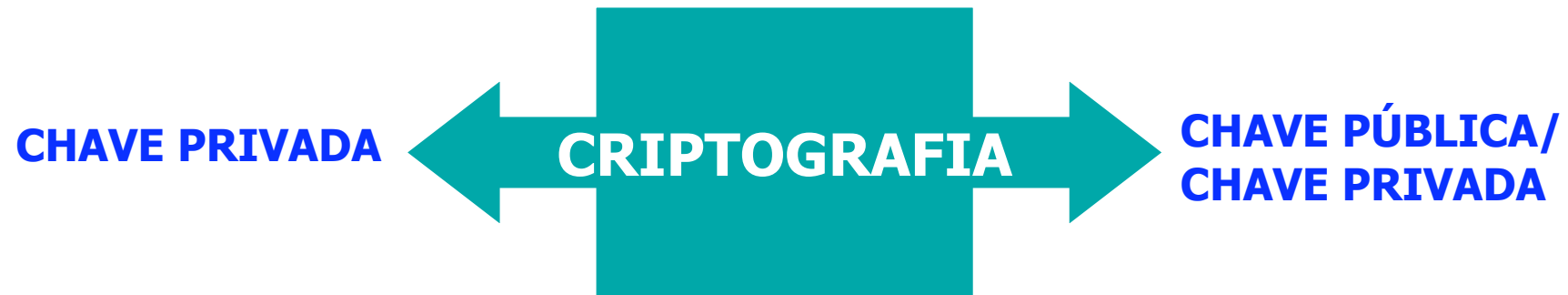
**DES**

**Hashing**

**SHA1**

**MD5**

# Criptografia



**Chave privada:  $K_{ab}$**

**Conhecida por  
ambos os peers.**

**Cada peer tem duas  
chaves:**

**-Chave pública ( $K_{A_{pub}}$ )**

**-Chave privada ( $K_{A_{priv}}$ )**

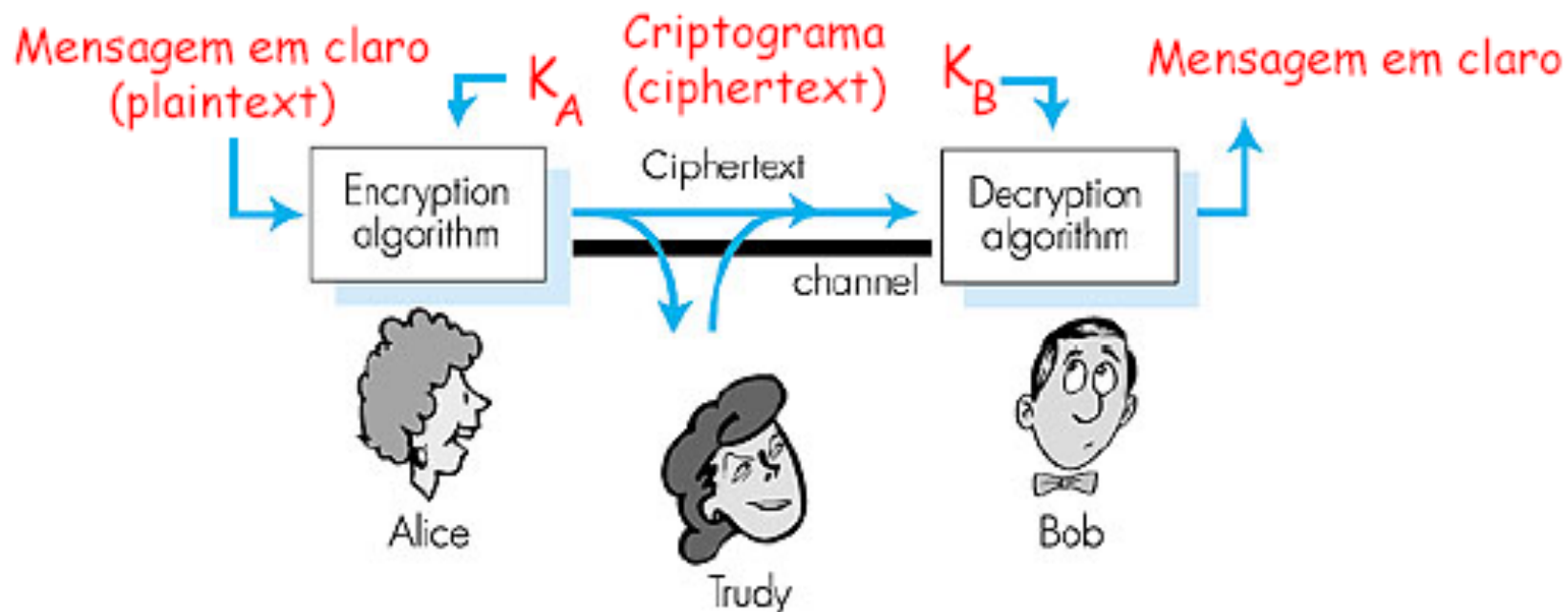
# Notações

---

$K_A$	Alice's secret key
$K_B$	Bob's secret key
$K_{AB}$	Secret key shared between Alice and Bob
$K_{Apriv}$	Alice's private key (known only to Alice)
$K_{Apub}$	Alice's public key (published by Alice for all to read)
$\{M\}^K$	Message $M$ encrypted with key $K$
$[M]_K$	Message $M$ signed with key $K$

---

# Criptografia: Simétrica e Assimétrica

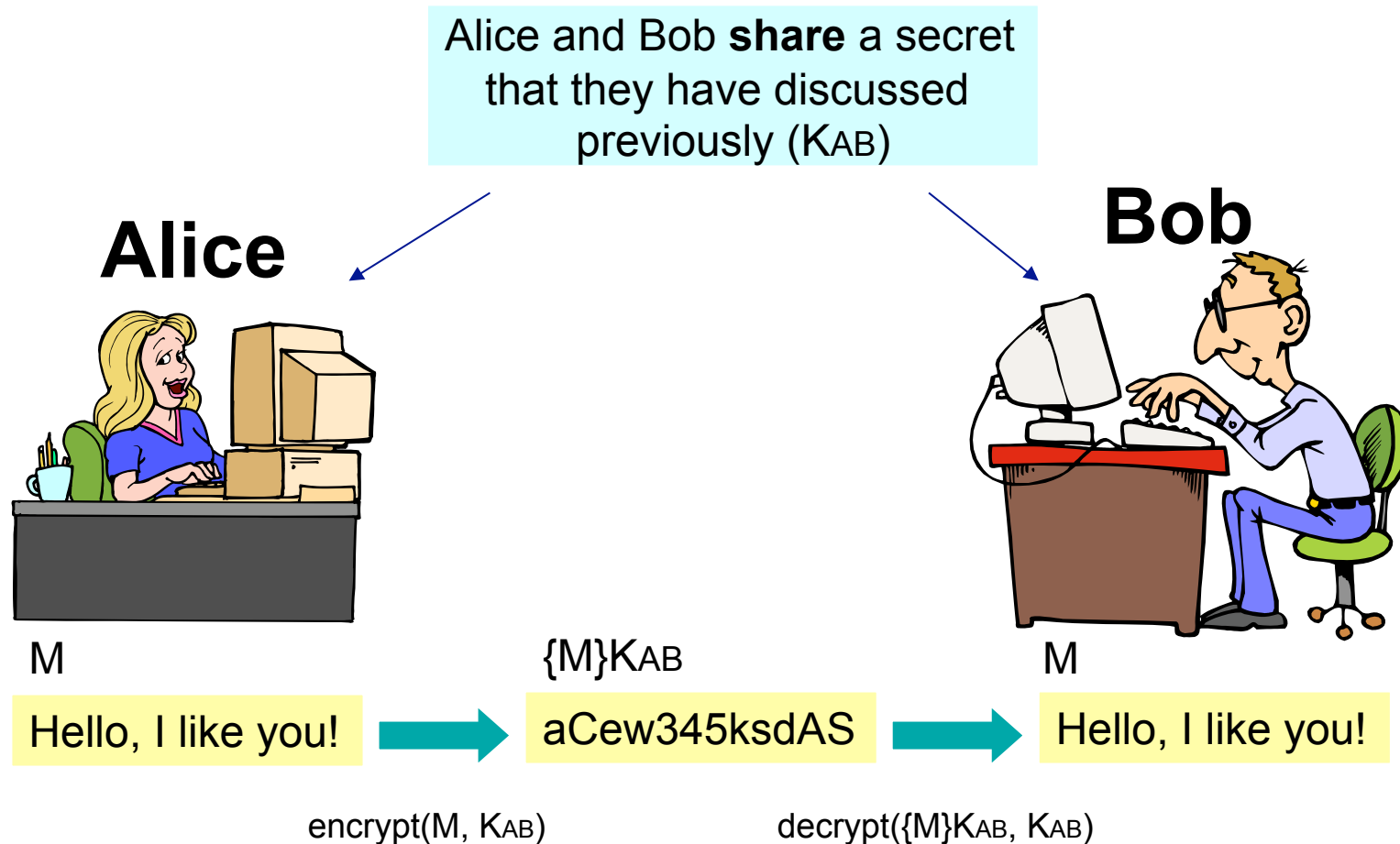


**Criptografia de chave simétrica** : as chaves de cifra e de decifra são idênticas

**Criptografia de chaves assimétrica ou de chave pública** : cifra-se com a chave pública, decifra-se com a chave privada do receptor, ou vice-versa



# Criptografia Simétrica



# Criptografia Assimétrica: Chave Pública/Chave Privada

Bob has a Public Key ( $K_{Bpub}$ )

**Alice**



$M$

Hello, I like you!

$\text{encrypt}(M, K_{Bpub})$

$\{M\}K_{Bpub}$

aCew345ksdAS

$\text{decrypt}(\{M\}K_{Bpub}, K_{Bpriv})$

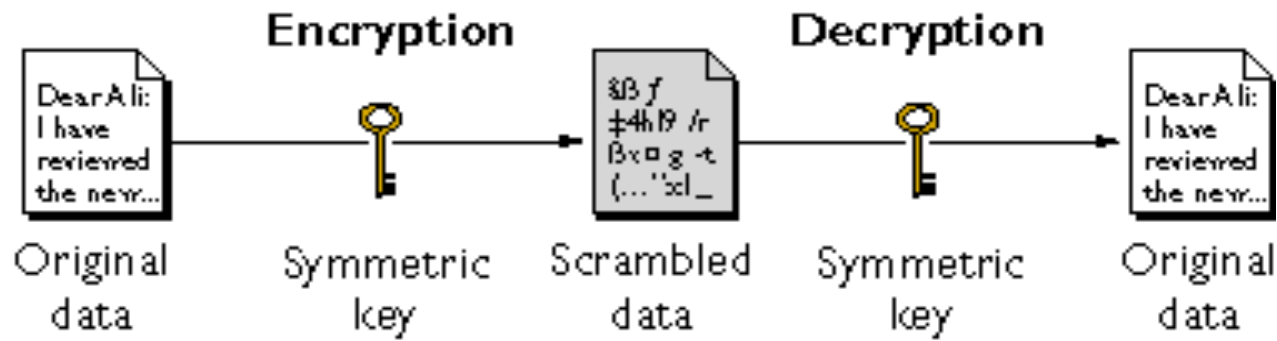
**Bob**



$M$

Hello, I like you!

# Symmetric Encryption



- **PROBLEM:** How to share the symmetric key among two anonymous persons?
- If there is one Web-site and multiple Clients, there cannot be a single encryption key for all the communications.
- Solution: use a [Key Distribution Center](#).

# Algorithms Symmetric Encryption

**TEA:** a simple but effective algorithm developed at Cambridge U (1994) for teaching and explanation. *128-bit key, 700 kbytes/sec*

**DES:** The US Data Encryption Standard (1977). No longer strong in its original form. *56-bit key, 350 kbytes/sec.*

**3DES:** applies DES three times with two different keys. *112-bit key, 120 Kbytes/sec*

**IDEA:** International Data Encryption Algorithm (1990). Resembles TEA. *128-bit key, 700 kbytes/sec*

**AES:** Advanced Encryption Standard (1997). *128/256-bit key.*

*The above speeds are for a Pentium II processor at 330 MHZ.*

# How to Crack an Encrypted Message?

- Criptoanalysis Algorithms
- Dedicated machines
- Brute-Force Attacks
  - Try all the possible keys!
  - Dedicated machines
  - Internet-based cracking
- Find bugs in the encryption software
- Break the system and find the private key
- Break the system and get the plain text

# Size of Private Keys

- 40 bit key  $\rightarrow 2^{40}$  different possibilities
  - On average you only have to try half!
  - Easily cracked by distributed brute force
- 128 bit key:  $2^{128}$  different possibilities
  - This number is larger than the number of molecules in all oceans
  - Safe for today but... You can't tell the future.

Se uma chave tiver 128 bits então há cerca de  $2^{128}$  ( $3.4 \times 10^{38}$ ) combinações. Se for possível arranjar um bilião ( $10^3 \times 10^6$ ) de computadores, capazes de testarem um bilião de chaves por segundo cada um, é possível testar  $10^{18}$  chaves por segundo. Então são necessários  $10^{13}$  anos para completar o ataque a uma chave de 128 bits. Ora estima-se que a idade do universo é de cerca de  $10^{10}$  anos.

# A História do DES

- **DES**: Developed by IBM and adopted as the US national standard for government and business applications. It specifies a 56-bit key.
- Jan 1997: RSA Corporation offers a \$10.000 reward to the first person to crack a message encrypted with DES.
- Why? To show that 56-bit keys are too weak.
- Jun 1997: A team cracks the message by brute-force.
  - Resources: between 1.000 and 14.000 persons on the Internet
  - Average power of each computer: Pentium 200MHz processor running the client on the background
  - Effective time to crack the key: 12 weeks after checking 25% of the keys
  - Lucky guy: Michael Sanders (Houston Texas) with a Pentium 90Mhz
  - **"Strong cryptography makes the world a safer place"**.

# Facts about DES

- TLS supports DES, but it's not the only encryption algorithm available.
  - RC2, RC4, RC5, 3DES.
  - 3DES → very slow but it gives an equivalent key size of 112 bits
- DES is considered obsolete. Any document encrypted with DES must be considered vulnerable.



# Symmetric Encryption

- Symmetric encryption address the problem of **eavesdropping** but do not address problems like:
  - **Tampering**
  - **Impersonation**
- **How to provide authentication?**

# **Autenticação**

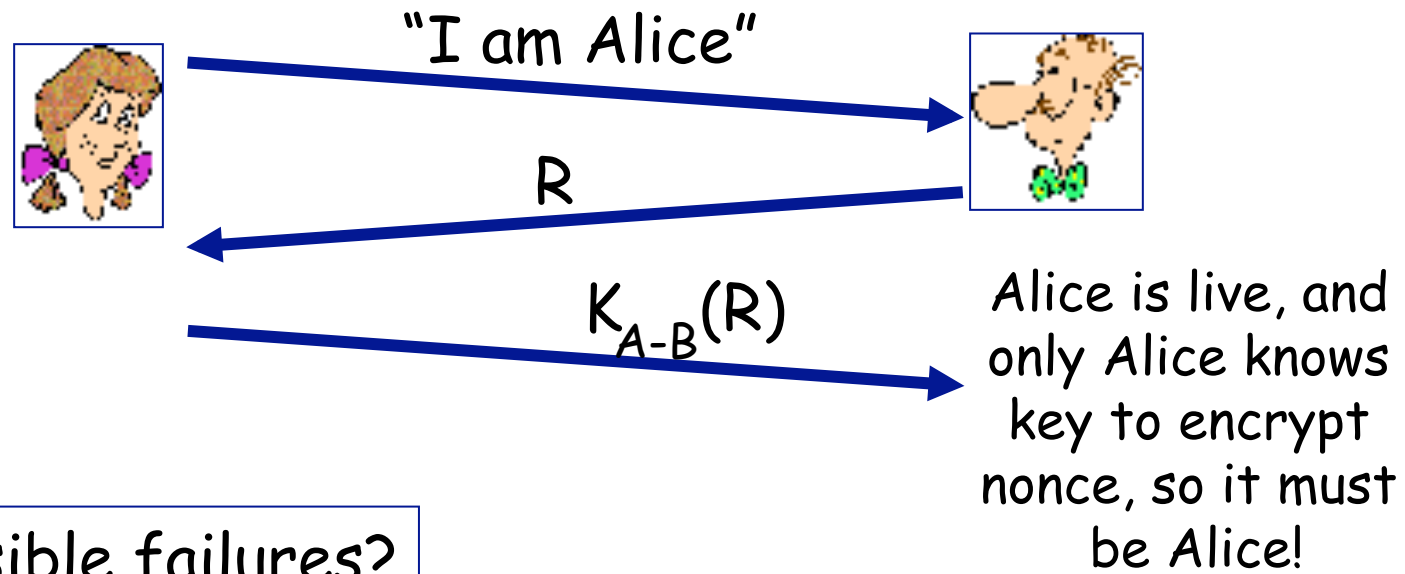
## **O Problema do Man-in-the-Middle**

# Authentication

Goal: avoid playback attack

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

Algorithm: Bob sends to Alice a nonce, R. Alice must return R, encrypted with shared secret key

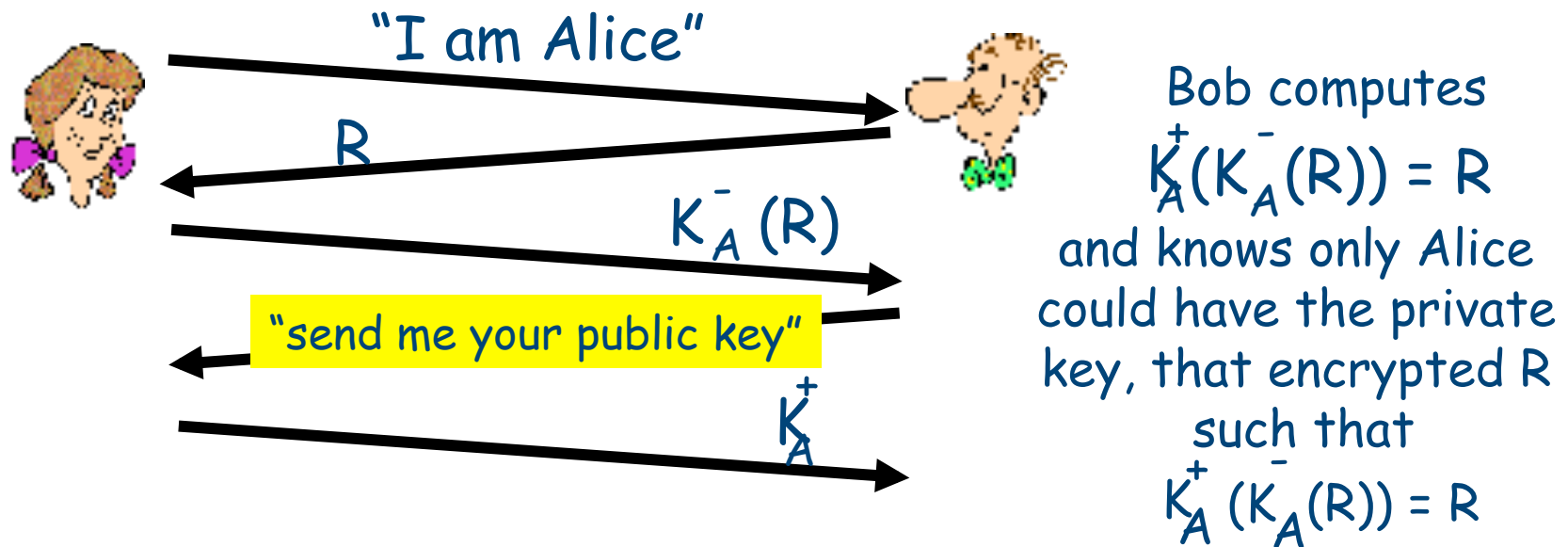


# Authentication

This version requires shared symmetric key

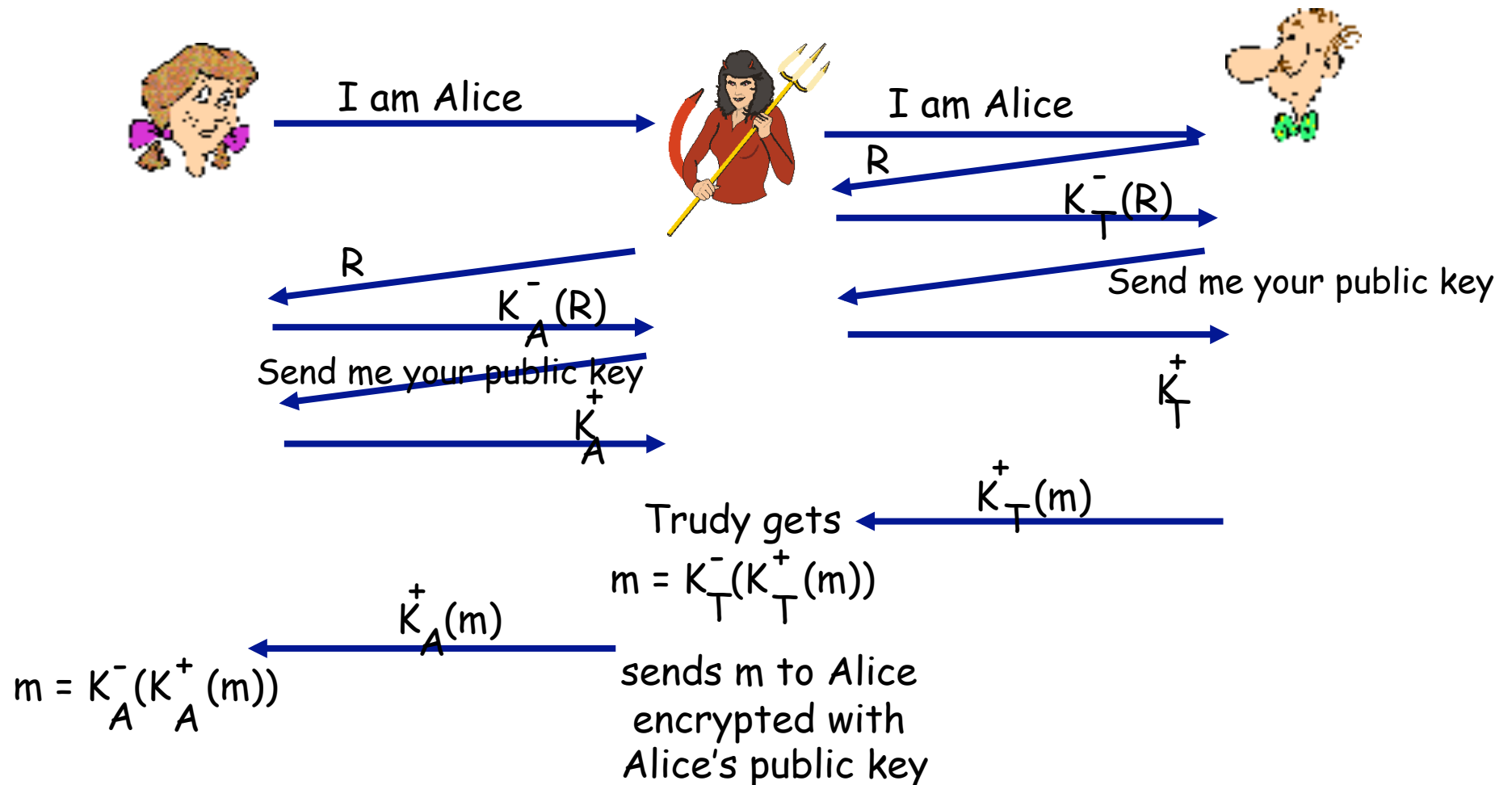
- can we authenticate using public key techniques?

Algorithm: use nonce, public key cryptography

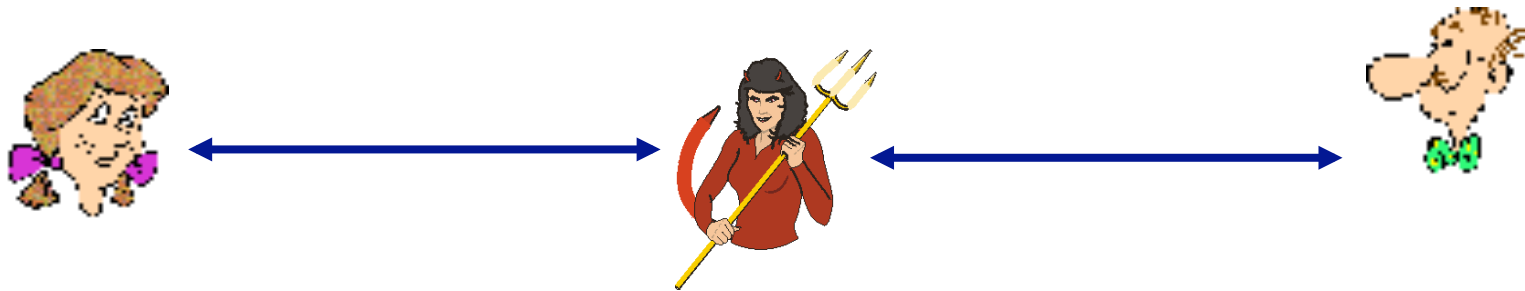


# Man-in-the-middle Attack

- Trudy acts as Alice (to Bob) and as Bob (to Alice)



# Man-in-the-middle Attack



Difficult to detect:

- ❑ Bob receives everything that Alice sends, and vice-versa.
- ❑ But Trudy receives all messages as well...

# **Key Distribution Centers**

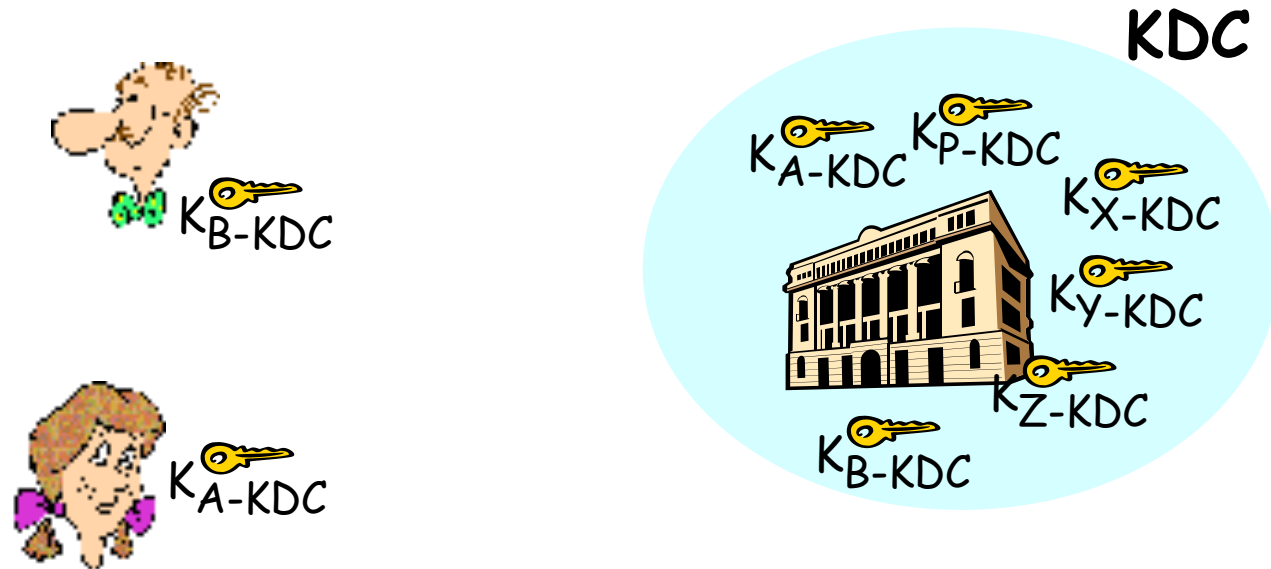
# How to share a secret key?

- Alice e Bob têm de arranjar alguma forma para estar de acordo sobre a chave secreta que vão usar.
- Dado que tal chave não pode ser trocada através da rede, terão de se encontrar ou usar uma terceira pessoa para trocarem a chave.
- Solução: usar uma chave diferente em cada sessão e usar um centro de distribuição de chaves no qual Alice e Bob confiam.



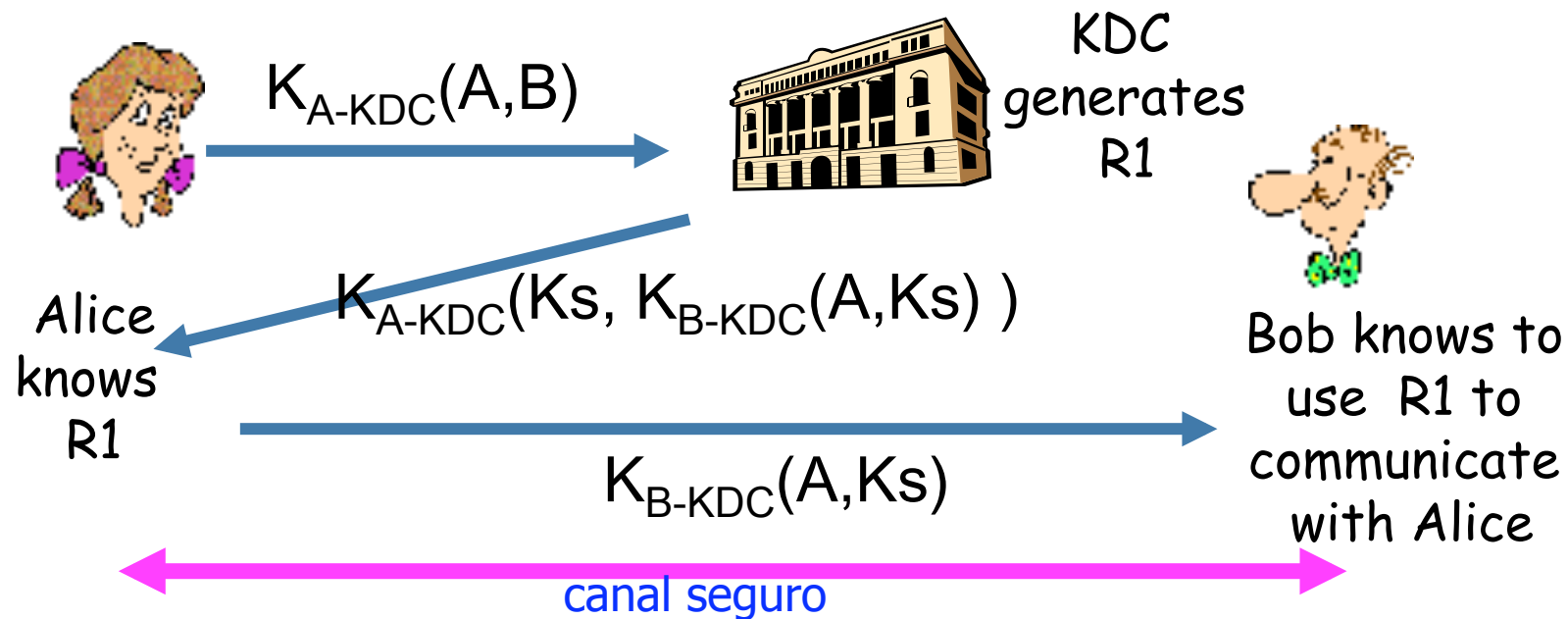
# Key Distribution Center (KDC)

- Alice, Bob need to share a key.
- **KDC:** server shares different secret key with *each* registered user
- Alice, Bob know own symmetric keys,  $K_{A-KDC}$   $K_{B-KDC}$ , for communicating with KDC.



# Key Distribution Center (KDC): short summary

- Use a KDC to generate a shared key for Bob and Alice.



Alice and Bob communicate: using **Ks** as *session key* for shared symmetric encryption

# Protocolo Needham-Schroeder

- Este protocolo permite a dois *principals* A e B estabelecerem um canal seguro e autenticarem-se mutuamente.
- Baseia-se na presença de um KDC (*Key Distribution Center*) que conhece as chaves secretas de A e B ( $K_a$  e  $K_b$ ).
- O KDC é capaz de gerar chaves de sessão ( $K_s$ ) que vão ser usadas por A e B para comunicarem de forma segura.
- O protocolo resiste aos ataques *eavesdropping*, *masquerading*, *message tampering* e *replaying*.

# O Protocolo Needham-Schroeder

## 1) A -> KDC: A, B, Na

Eu sou A e quero falar com B, dá-me um "ticket" para eu me autenticar perante ele.

Na é um número aleatório gerado por A que garante a "unicidade" da transacção, isto é, garante que o valor de Na é único e não passou ainda em mensagens.

## 2) KDC -> A: { Na, B, Ks, { Ks, A }Kb }Ka

Aqui estão Na, Ks e o ticket cifrados com a tua chave. Ks não passa em claro e Na permite a A reconhecer o KDC pois só ele conhece Ka (para além de A). A não pode modificar o ticket pois este está cifrado com a chave de B.

## 3) A -> B: {Ks, A}Kb, {Na'}Ks

A diz a B que quer falar com ele e manda-lhe o ticket.

## 4) B -> A: {Na'-1}Ks

B prova que é B pois foi capaz de decifrar {Na}Ks o que pressupõe que decifrou {Ks,A}Kb

# Possível ataque e solução

A mensagem 3) anterior pode ser *replayed* mais tarde. Isso em princípio não tem problema mas se por acaso  $K_s$  for um dia comprometida, então Trudy pode conseguir autenticar-se perante Bob como se fosse Alice. A solução consiste em acrescentar um *timestamp* ao ticket que deve ser testada por Bob após ter recebido a mensagem 3). Tal exige que os relógios de A, B e do KDC estejam sincronizados a menos de um valor considerado suficiente para tornar impossível a quebra da chave  $K_s$ .

O protocolo fica então assim:

## 1) A -> KDC: A, B, $N_a$

Eu sou A e quero falar com B, dá-me um *ticket* para eu me autenticar perante ele.  
 $N_a$  é um número aleatório gerado por A.  $t$  impede a utilização por replaying do *ticket*.

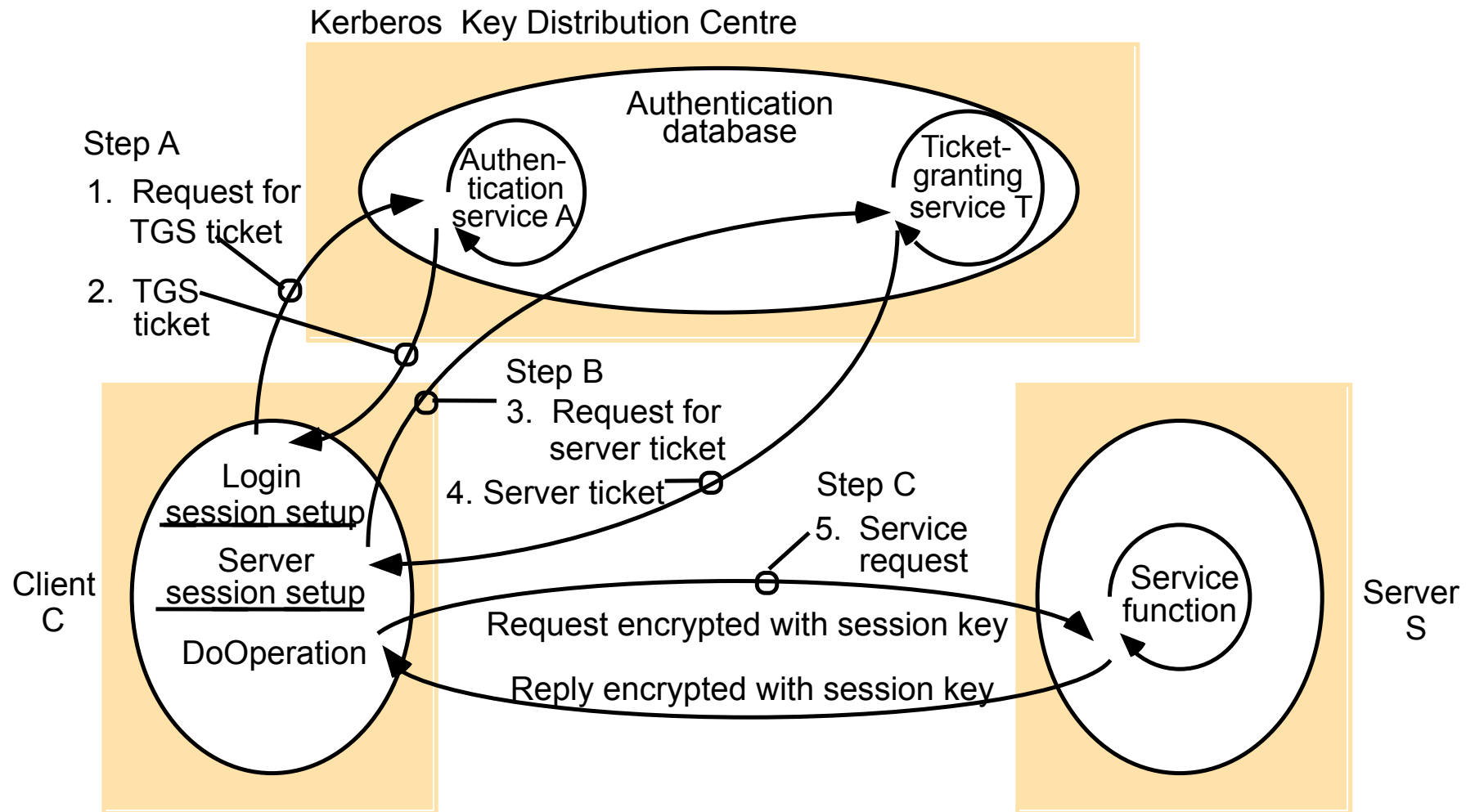
## 2) KDC -> A: { $N_a$ , B, $K_s$ , { $K_s$ , A, $t$ } $K_b$ } $K_a$

Aqui estão  $N_a$ ,  $K_s$  e o ticket cifrados com a tua chave.  $K_s$  não passa em claro e  $N_a$  permite a A reconhecer o KDC. A não pode modificar o ticket pois este está cifrado com a chave de B.

## 3) A -> B: { $K_s$ , A, $t$ } $K_b$ , { $N_a'$ } $K_s$

## 4) B -> A: { $N_a'-1$ } $K_s$

# Arquitetura do Kerberos



# Protocolo Kerberos (v5.0)

## PHASE 1: OBTAIN KERBEROS SESSION KEY AND TGS TICKET. ONCE PER LOGIN SESSION

**A -> AS: A, TGS, n1**

Olá AS, eu sou a *Alice* e quero fazer “login” no sistema.

**AS -> A: { Ks, n1}Ka, {A,TGS, t1, t2, Ks}Ktgs**

Toma lá uma chave (Ks) cifrada com a tua chave, que te permite obter tickets e um ticket que te permite falar com o TGS, válido de t1 até t2.

**AS= Authentication Service**

**TGS: Ticket Granting Service**

## PHASE II: OBTAIN A TICKET FOR A SERVER B. ONCE PER CLIENT-SERVER SESSION.

**A -> TGS:  $\{A, TGS, t1, t2, Ks\}_{K_{tgs}}, \{A, t3\}_{Ks}, B, n2$**

Olá TGS. Eu sou A e quero falar com B;  $\{A, t3\}_{Ks}$  autentica A perante o TGS visto que demonstra que A conhece  $Ks$ . Repare-se que  **$Ka$**  foi usada para se obter  $Ks$  mas **a partir daí vai deixar de ser usada** pois o ticket  $\{A, TGS, t1, t2, Ks\}_{K_{tgs}}$  permite a A dialogar com o TGS de forma segura sem usar  $Ka$ , o que lhe permite apagá-la da memória.

$\{A, t3\}_{Ks}$  diz-se um “autenticador” com uma time-stamp ( $t3$ ) que impede o replaying desta mensagem.

**TGS -> A:  $\{B, n2, K_{ab}\}_{Ks}, \{A, B, t1', t2', K_{ab}\}_{Kb}$**

Aqui tens uma chave ( $K_{ab}$ ) e um ticket para falares com B.



### PHASE III: ISSUE A SERVER REQUEST WITH A TICKET.

**A -> B:**  $\{A, t3'\}_{Kab}, \{A, B, t1', t2', Kab\}_{Kb}, n3$

### PHASE IV: AUTHENTICATE SERVER.

**B -> A:**  $\{n3\}_{Kab}$

A pede a B que se autentique mandando-lhe um nonce ( $n3$ ) que serve para A autenticar B por "challenge-response" e para evitar replaying.

Os valores de  $t1, t1', t2, t2', t3, t3'$  são timestamps, para evitar que A, B, TGS, ... memorizem esses valores que têm usado em autenticação; tal permite detectar *replaying* dado que se pressupõe que os relógios das diferentes máquinas estão sincronizados.

$n1, n2$  e  $n3$  são nonces para confirmar que o parceiro é quem se julga ser; o facto de serem nonces impede também replaying.

Caso A necessite de falar com outro servidor, volta a pedir ao TGS uma chave de sessão e um ticket.

# Uso do Protocolo Kerberos

- NFS
  - Email
  - Login
  - Printing
- 
- (Os serviços têm chaves secretas, apenas conhecidas pelos serviços e pelo Servidor de Kerberos).

# **Public-Key Encryption**

# Public Key Encryption



Bob publishes his public key to the world. Everyone knows it.  $K_{\text{Bob\_Public}}$

Bob keeps his private key safe from the world.  $K_{\text{Bob\_Private}}$

**Alice**



M

Hello, I like you!

$\{M\}_{K_{\text{Public\_Bob}}}$

aCew345ksdAS

M

Hello, I like you!

**Bob**



$\text{encrypt}(M, K_{\text{Public\_Bob}})$

$\text{decrypt}(\{M\}_{K_{\text{Public\_Bob}}}, K_{\text{Private\_Bob}})$

# Algorithms

- **RSA** (512 – 1024 bits)
  - **ElGamal**
  - **Diffie-Hellman/DSS** (1024-4096)
  - Elliptic curve algorithms
- 
- Main limitation: **performance....**
  - Public-key encryption is 1000 x slower than symmetric encryption.
  - **Makes it unpractical to encrypt long messages.**

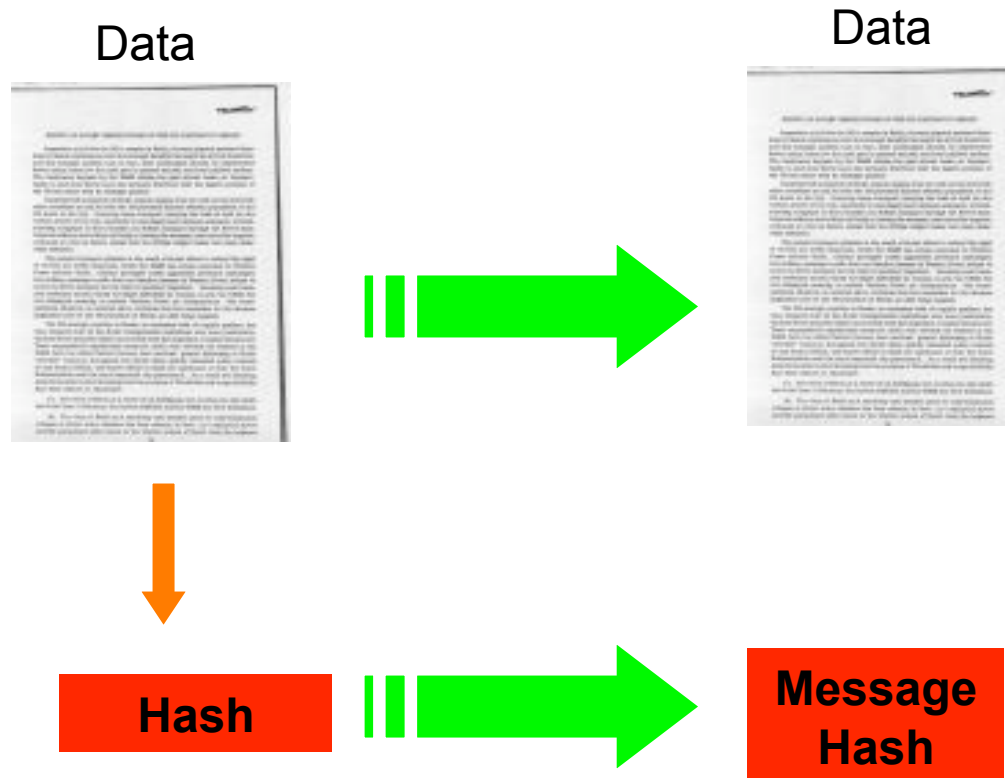
# Performance

	Key size/hash size (bits)	PRB optimized 90 MHz Pentium I (Mbytes/s)	Crypto++ 2.1 GHz Pentium 4 (Mbytes/s)
TEA	128	-	23,081
DES	56	2,113	21,34
Triple-DES	112	0,775	9,848
IDEA	128	1,219	18,963
AES	128	-	61,010
AES	192	-	53,145
AES	256	-	48,229
MD5	128	17,025	216,674
SHA-1	160	-	67,977

# How to avoid Tampering?

- Tamper detection rely on a mathematical function called a **one-way hash** (or **message digest**).
- If there is some tampering on the message data this will be detected by the “hash” code.
- **Integrity Checking.**

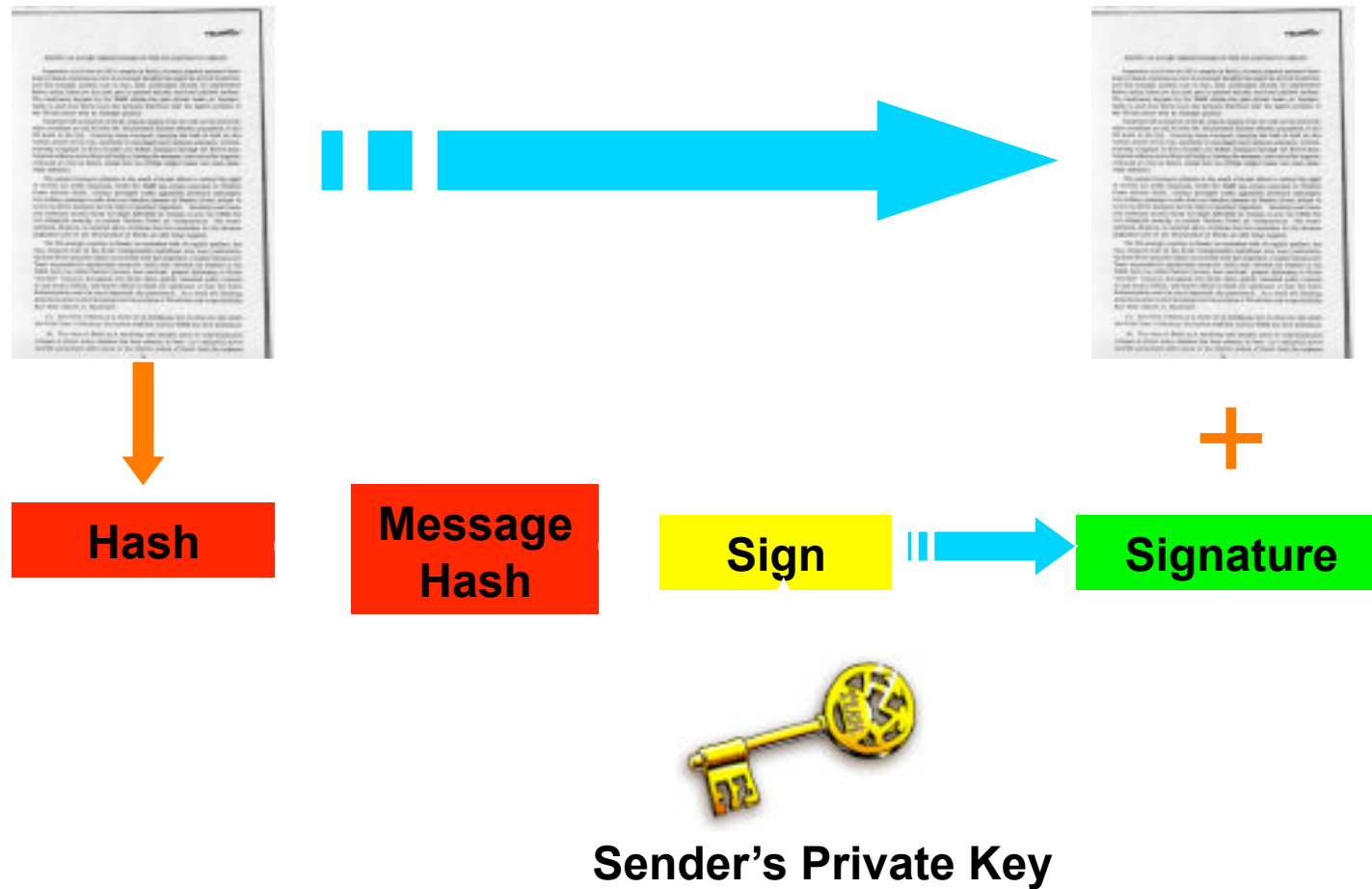
# Hash Functions



- Creates a unique “fingerprint” for a message
- Anyone can alter the data and calculate a new hash value
- **Hash has to be protected in some way**



# Digital Signatures



- Combines a hash with a digital signature algorithm.

# Digital Signatures (cont)

Data



+

Signature



Hash



Verify



?

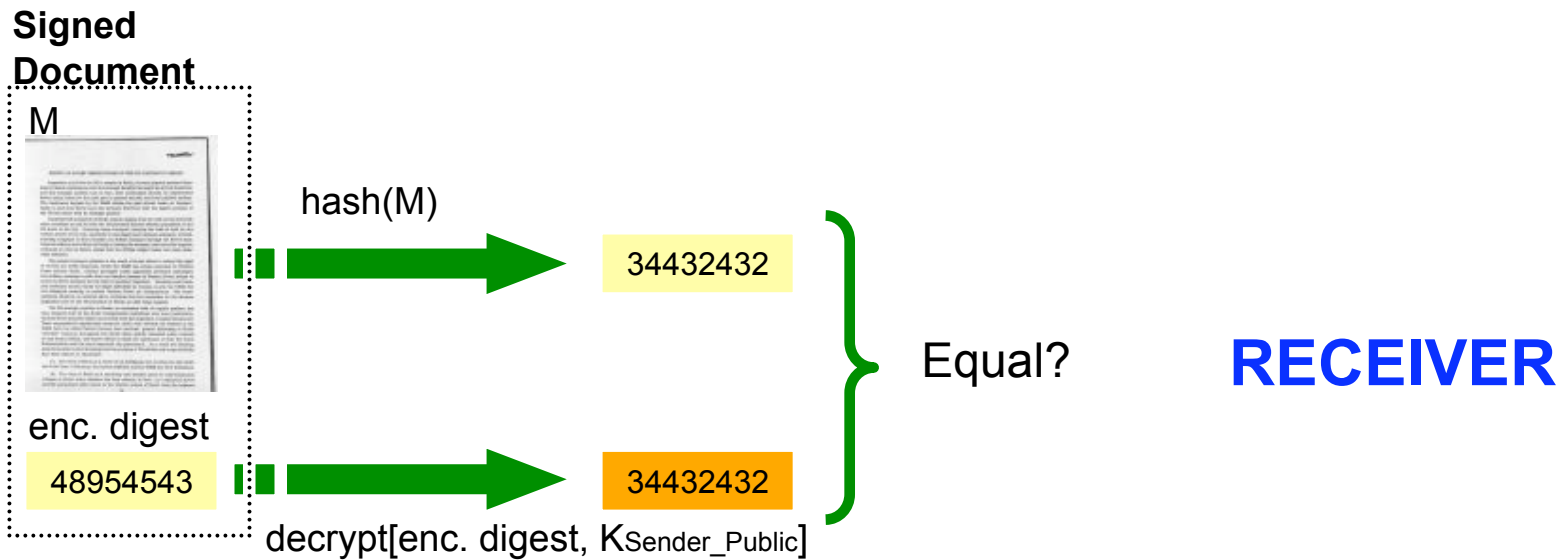
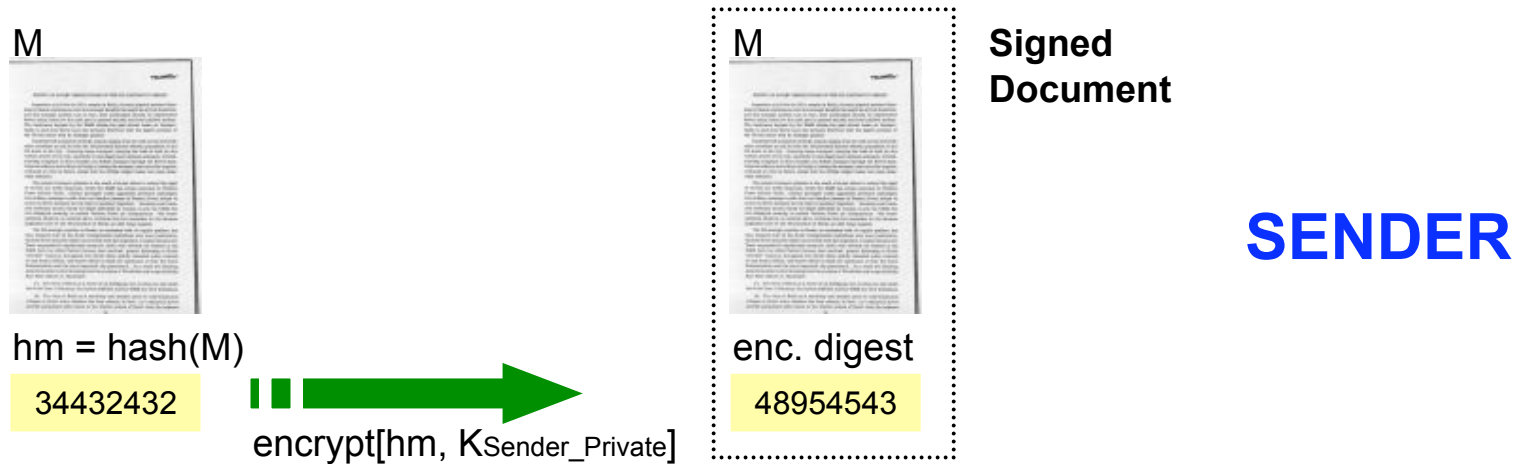
- Signature checking.



These digital signatures  
validate data integrity.

Sender's Public Key

# Exemplo: Digital signatures



# Message Digest Functions

- **MD4** (128 bits)
- **MD5** (128 bits)
- **SHA-1** (160 bits)

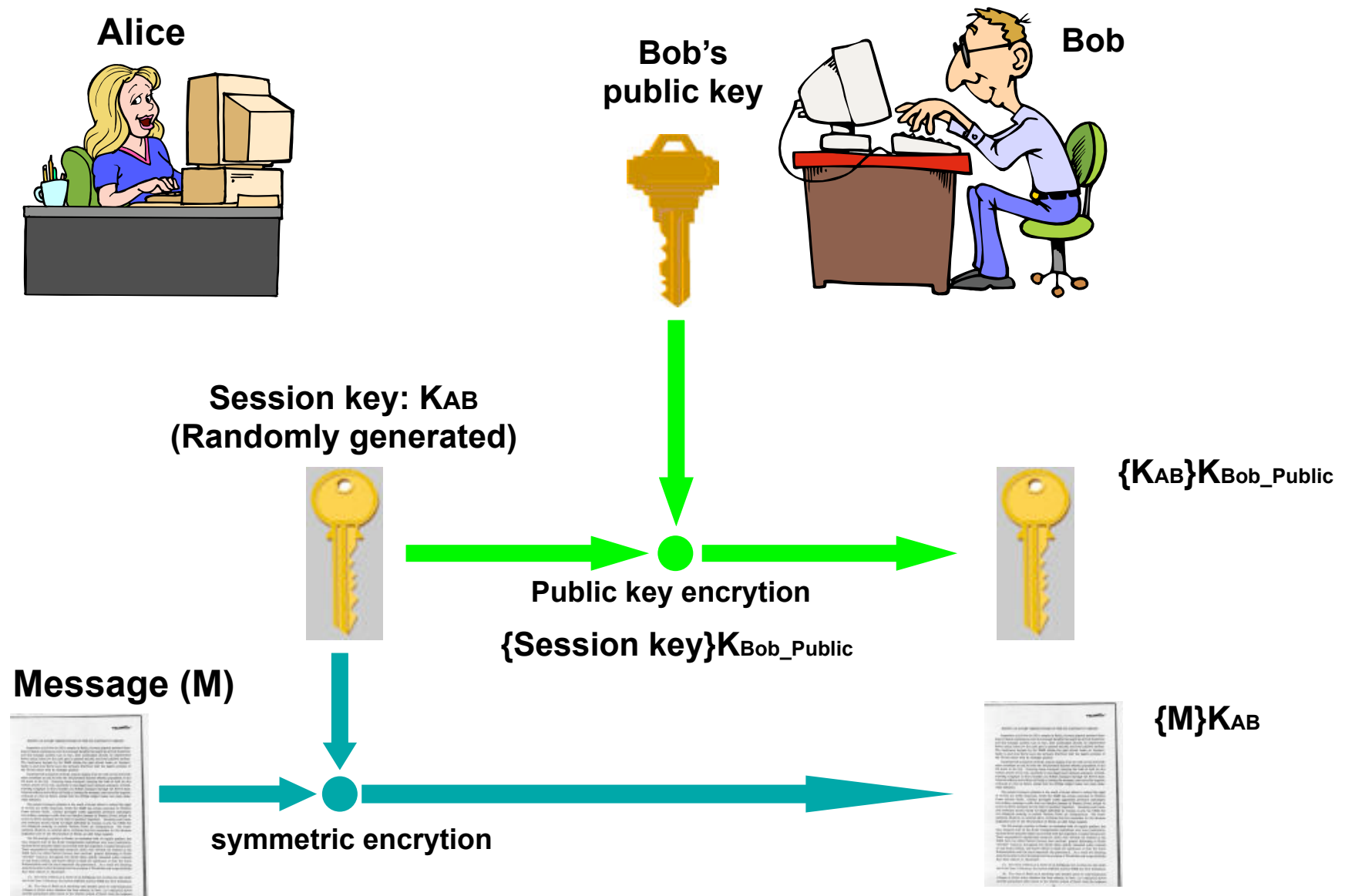
# Digital Signatures

- Avoids message tampering.
- Supports some degree of non-repudiation.

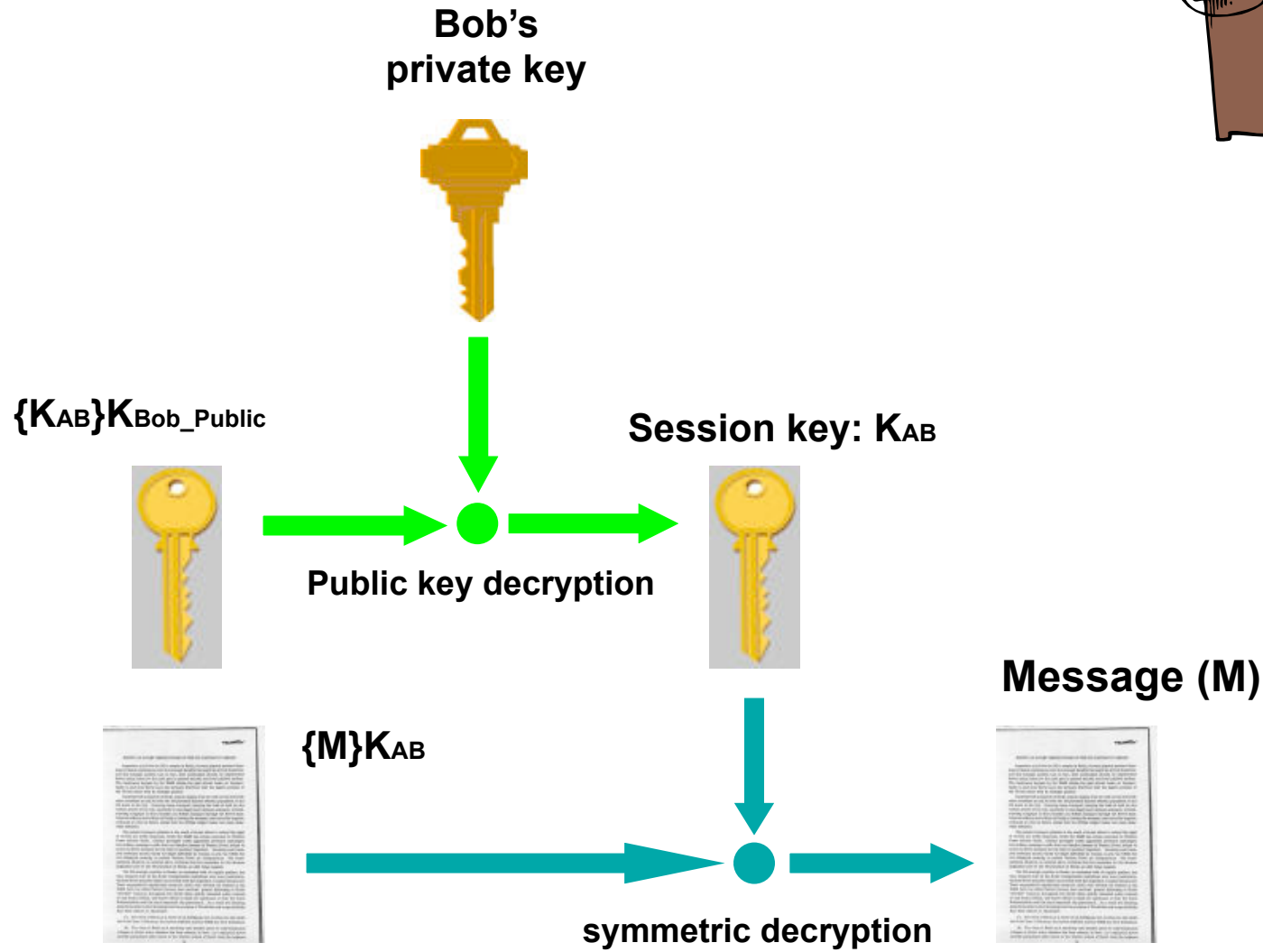
# Hybrid Approach !

- Public-key encryption is unsuitable for transferring large documents. Too slow...
- Public-key and symmetric cryptography are combined in a way to take advantage of their best features.
- Use Public-key encryption to send a **session key**, and do **symmetric encryption** of the message using this session key.
- Approach used by the TLS protocol.

# Establishing Secure Communication - The Principle



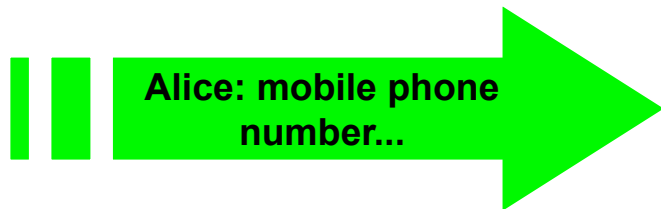
# Establishing Secure Communication





# Sender Authentication...

Incoming Message...



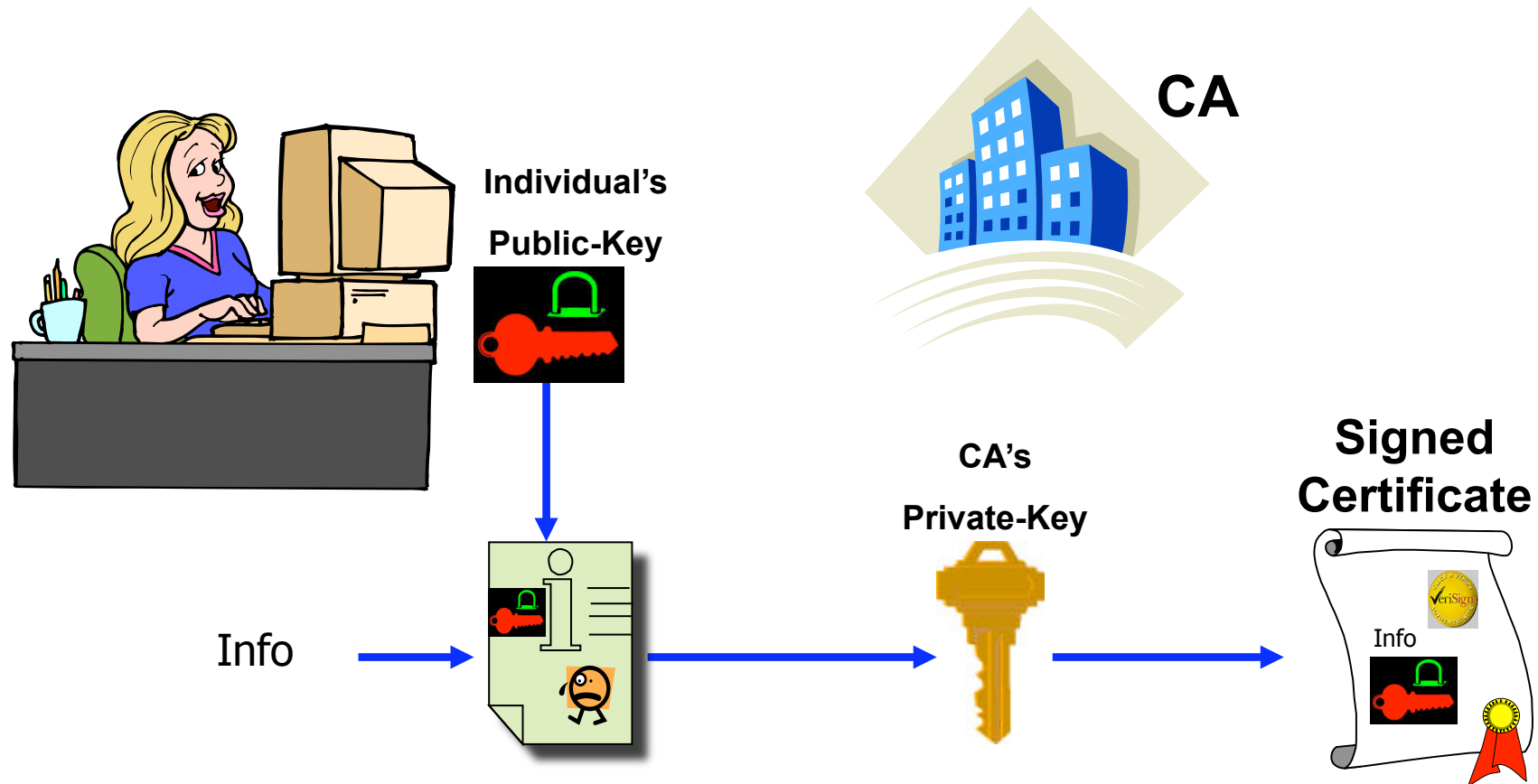
# Question...

- Alice sends a message to Bob ...
- The message is encrypted with Bob's Public Key.
- The message is decrypted by Bob, using his Private Key.
  
- Can Bob be sure that the message is coming from Alice?
- Is it possible to happen some spoofing from Malek?
- How to prevent that to occur?

# Certificates and Authentication

- A **certificate** is an electronic document used to identify an individual, a server or a company, and to associate that identity with a **public key**.
- Public-key cryptography uses certificates to address the problem of **impersonation**.

# Certifying Authorities (CAs)



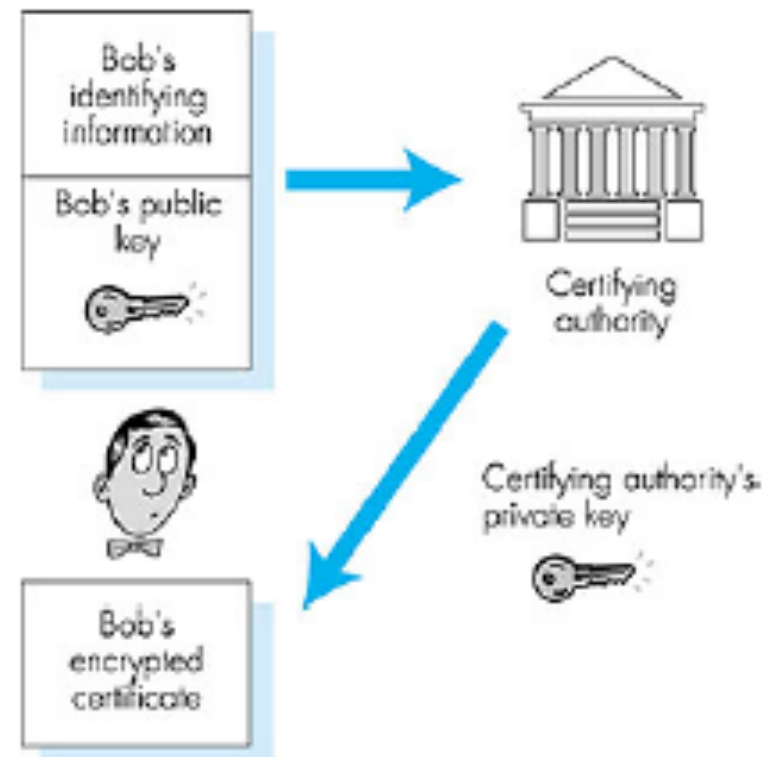
- Certificate Authorities (CAs) are entities that validate identities and issue certificates.

# Certifying Authorities

Uma certifying authority (CA) associa chaves públicas a principals.

Quando a Alice quer verificar qual a chave pública do Bob:

- 1) arranja um certificado dessa chave (Bob pode dá-lo !).
- 2) Verifica a assinatura da CA através da respectiva chave pública.

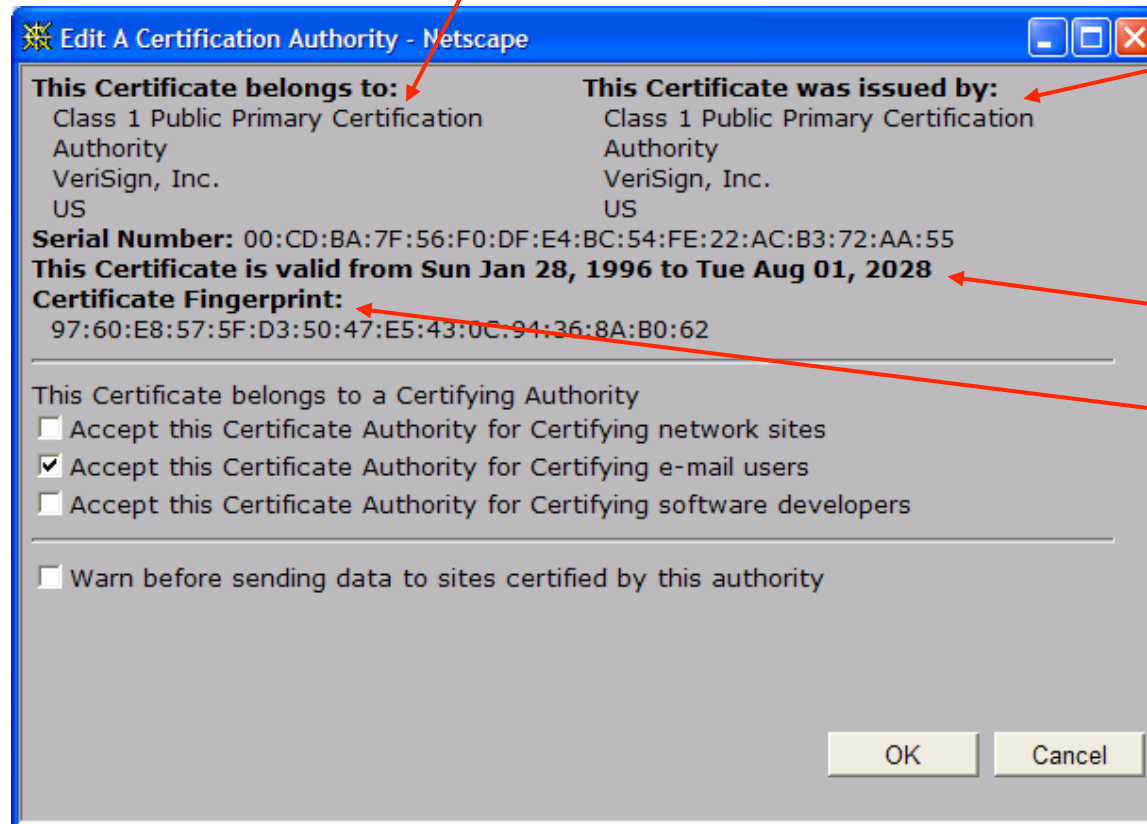


# Ordering a TLS Certificate

<b>Country Name:</b>	<b>US</b>
<b>State or Province:</b>	<b>Utah (don't use illegal characters: -'&amp;)</b>
<b>City or Locality:</b>	<b>Orem</b>
<b>Organization Name:</b>	<b>Acme Widgets Inc.</b>
<b>Dpt. or Unit Name:</b>	<b>Web Operations</b>
<b>Common Name:</b>	<b>www.acme.com (domain listed on certificate)</b>
<b>Webmaster E-mail:</b>	<b>john@acme.com</b>
<b>Login Name:</b>	<b>acme</b>
<b>Domain Name:</b>	<b>www.acme.com</b>
<b>IP Number</b>	<b>192.41.11.51</b>
<b>Webmaster Name</b>	<b>John Doe</b>
<b>e-mail contact</b>	<b>Reseller@bigbucks.com</b>

# A certificate contains:

- Serial number (unique to issuer)
- info about certificate owner, including algorithm and key value itself (not shown)



■ info about certificate issuer

■ valid dates

■ digital signature by issuer

# Certificado de um Banco

---

1. <i>Certificate type</i>	Public key
2. <i>Name:</i>	Bob's Bank
3. <i>Public key:</i>	$K_{Bpub}$
4. <i>Certifying authority</i>	Fred – The Bankers Federation
5. <i>Signature</i>	$\{Digest(field\ 2 + field\ 3)\}_{K_{Fpriv}}$

---



# Types of Certificates

- Personal Certificates
- Server TLS Certificates
- S/MIME Certificates
- Software Publisher Certificates
- CA Certificates

# Certifying Authorities

- Verisign



- Twathe



- CREN

- Multicert

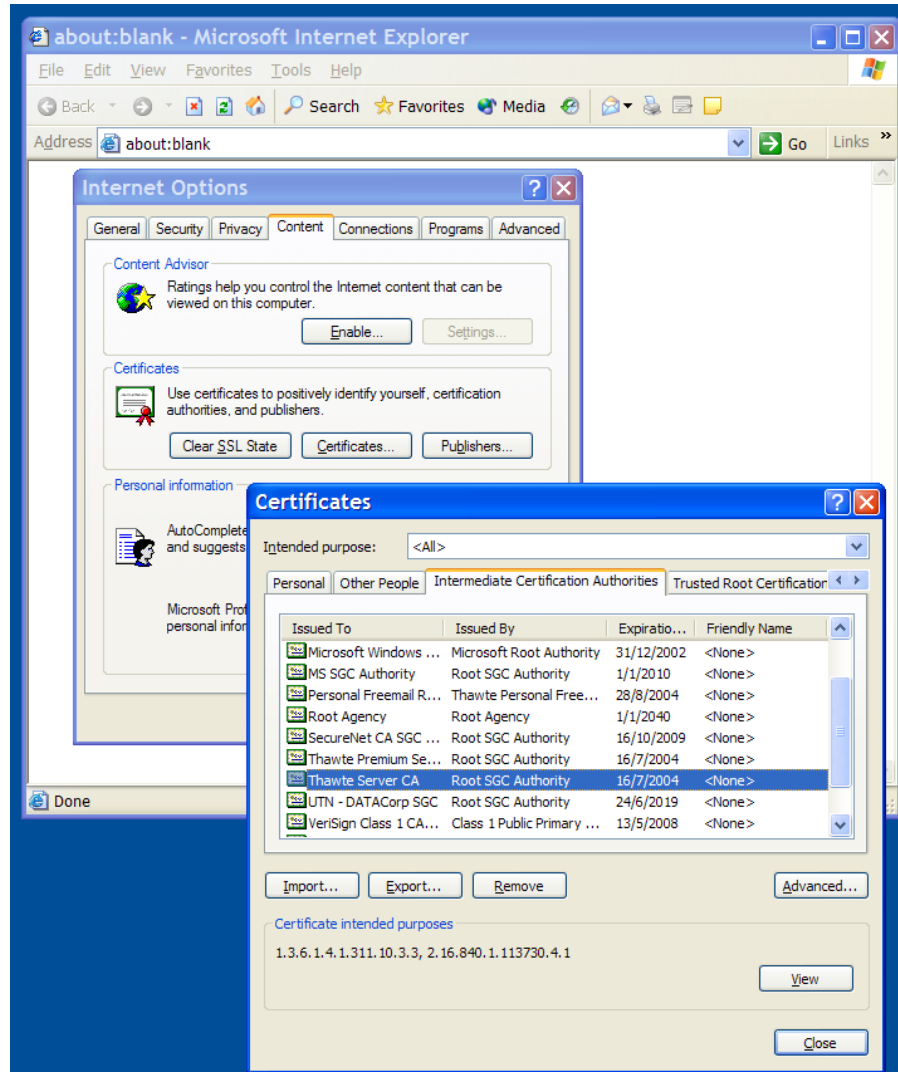
- Banks

- Governments

# Trusting CAs

- In order for the certificate to be trusted, it must be signed using the private key of the CA.
  - But for you to verify it, you need the CA's public key!
- But, how can you trust the CA? How do you get the CA's public key securely???

# Trusting CAs



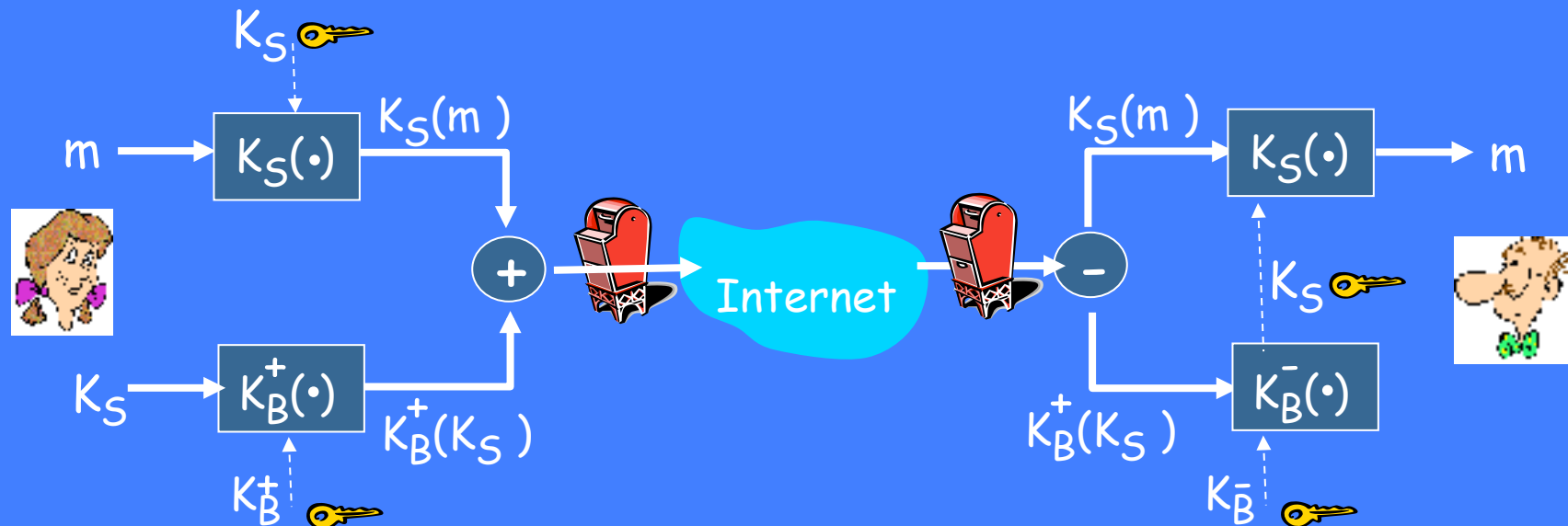
Public keys from CAs must be delivered through a secure channel.

- Installation CD, etc.

**Secure Email**

# Secure Email

□ Alice wants to send confidential e-mail,  $m$ , to Bob.

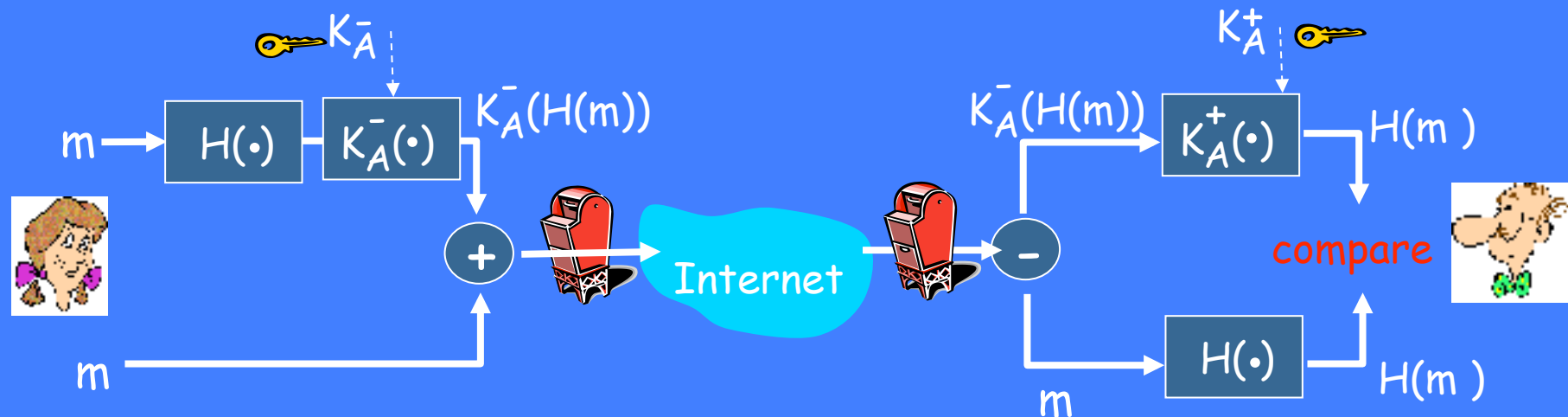


## Alice:

- generates random *symmetric* private key,  $K_S$ .
- encrypts message with  $K_S$  (for efficiency)
- also encrypts  $K_S$  with Bob's public key.
- sends both  $K_S(m)$  and  $K_B(K_S)$  to Bob.

# Secure Email (continued)

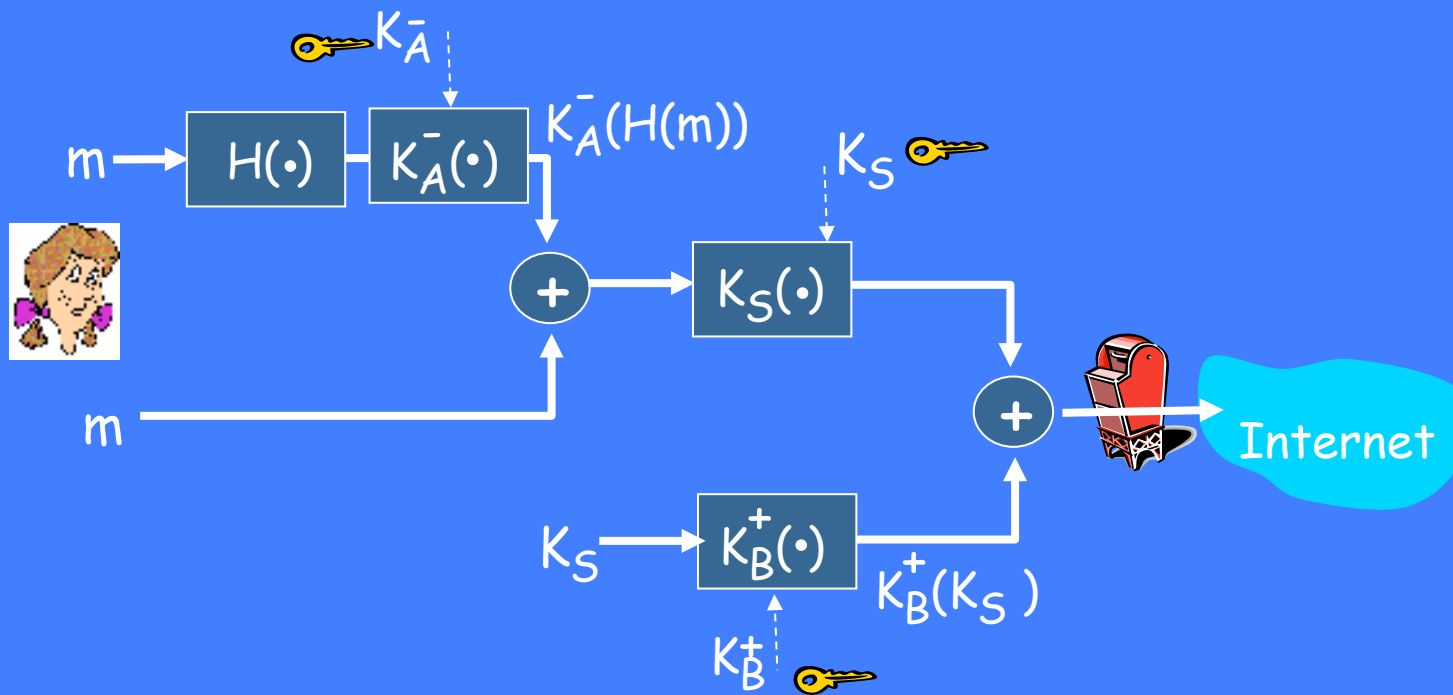
- Alice wants to provide sender authentication and message integrity.



- Alice digitally signs message.
- sends both message (in the clear) and digital signature.

# Secure Email (continued)

- Alice wants to provide secrecy, sender authentication, message integrity.



**Alice uses three keys:** her private key, Bob's public key, newly created symmetric key.



# Pretty Good Privacy (PGP)

- Internet e-mail encryption scheme, de-facto standard.
- Uses symmetric key cryptography, public key cryptography, hash function, and digital signature as described.
- Provides secrecy, sender authentication, integrity.
- Inventor, Phil Zimmerman.

```
---BEGIN PGP SIGNED  
MESSAGE---  
Hash: SHA1
```

```
Bob:My husband is out of town  
tonight.Passionately  
yours, Alice
```

```
---BEGIN PGP SIGNATURE---  
Version: PGP 5.0  
Charset: noconv  
yhHJRHhGJGhgg/12EpJ  
+lo8gE4vB3mqJhFEvZP9t6n7G6  
m5Gw2  
---END PGP SIGNATURE---
```

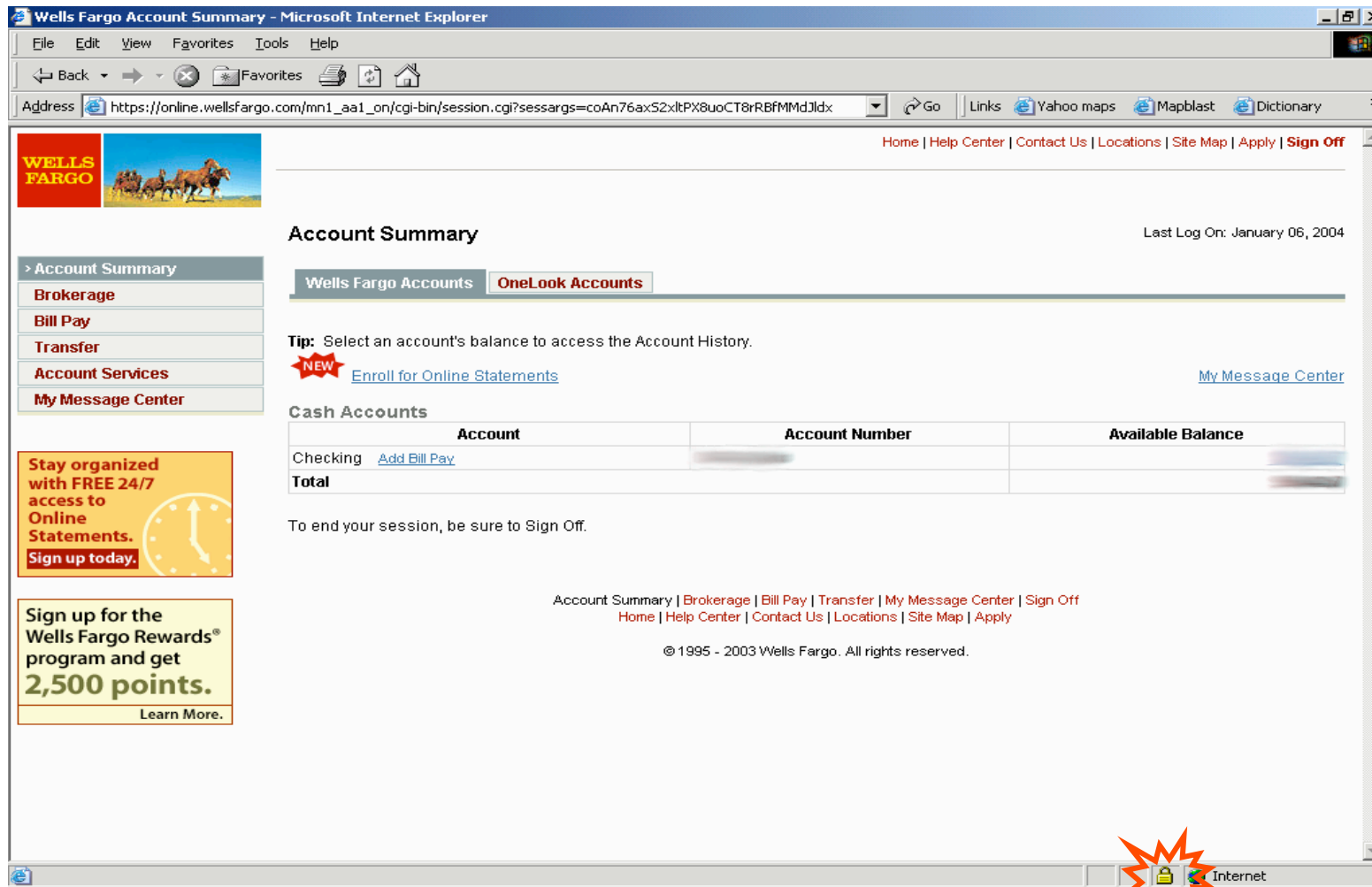
# **TLS Protocol**

## **Transport Layer Security**

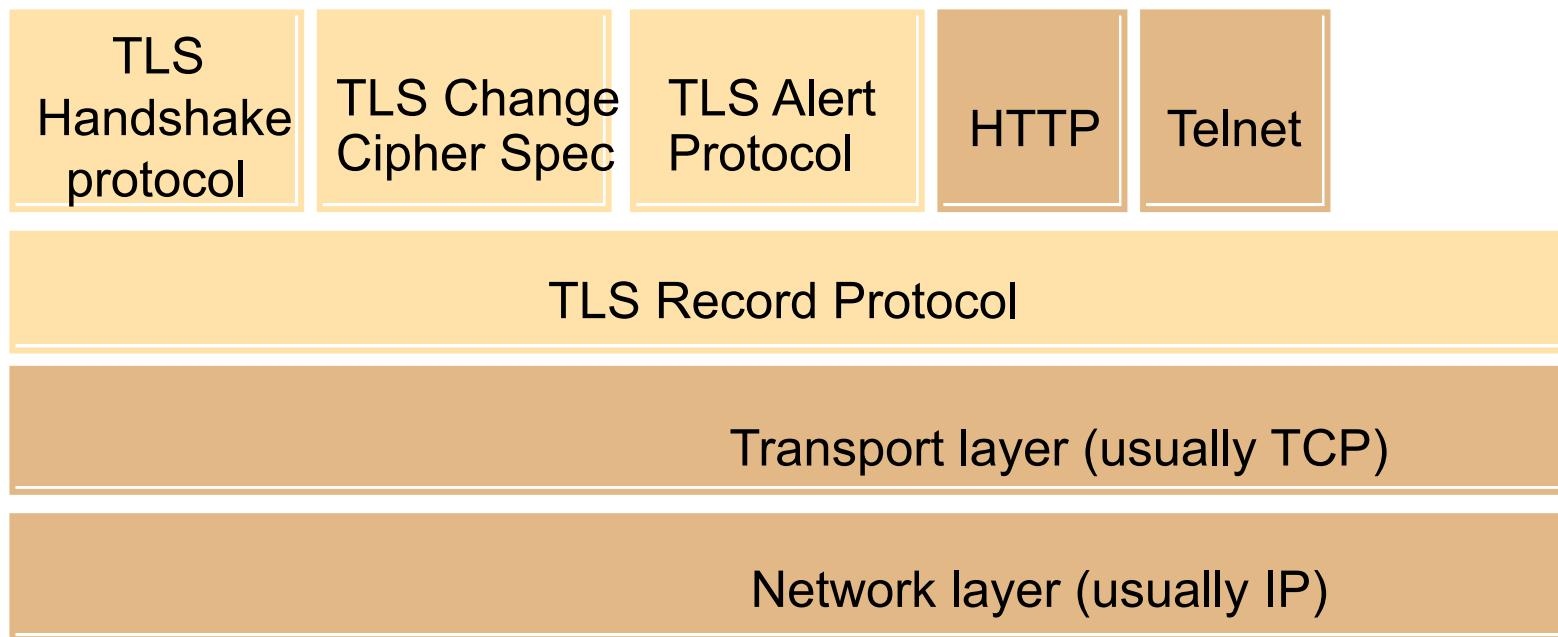
# TLS Protocol

- TLS: Transport Layer Security
  - A descendant of Secure Sockets Layer (SSL).
- Authenticates server using digital signatures.
- Can authenticate client (optional).
- Confidentiality protection via encryption.
- Integrity protection.
- Provides end-to-end protection within a session.

# HTTPS (HTTP sobre TLS)



# TLS Stack



TLS protocols:



Other protocols:



# TLS: in one slide

- TLS requires a server TLS certificate, at a minimum.
- As part of the initial "handshake" process, the server presents its certificate to the client to authenticate the server's identity.
- The authentication process uses **Public-Key Encryption** and **Digital Signatures** to confirm that the server is in fact the server it claims to be.
- Once the server has been authenticated, the client and server use techniques of **Symmetric-Key Encryption**, which is very fast, to encrypt all the information they exchange for the remainder of the session and to detect any tampering that may have occurred.

# TLS: Handshake Protocol

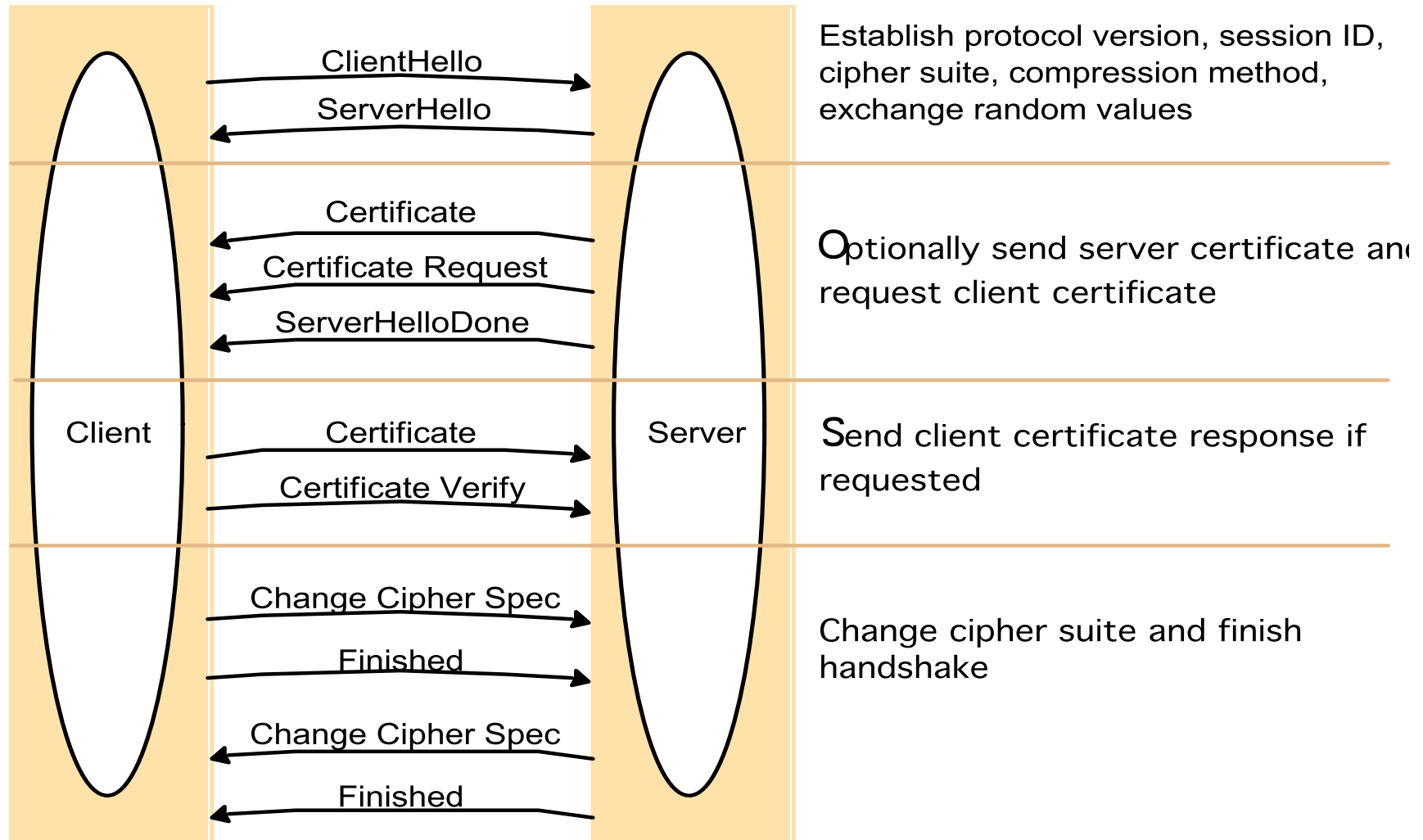
- Negotiate the cipher suite.
- Establish a shared session key.
- Authenticate the server to the client.
- Optionally authenticate the client to the server.
- Select the cryptographic algorithms, or ciphers, that they both support.
- Use public-key encryption techniques to generate shared secrets.
- Establish an encrypted TLS connection.

# Server Authentication

- Um browser preparado para TLS inclui as chaves públicas de várias CAs
- O browser solicita ao servidor um certificado emitido por uma CA em que ele confie.
- O browser usa a chave pública da CA para extrair e verificar a chave pública do servidor.



# TLS: Handshake Protocol



# TLS Handshake Configuration Options

<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

# Ciphers used with TLS

- **DES**. Data Encryption Standard.
- **DSA**. Digital Signature Algorithm.
- **KEA**. Key Exchange Algorithm.
- **MD5**. Message Digest algorithm developed by Rivest.
- **RC2 and RC4**. Rivest encryption ciphers (RSA Data Security).
- **RSA**. A public-key algorithm for both encryption and authentication. Developed by Rivest, Shamir, and Adleman.
- **RSA key exchange**. A key-exchange algorithm for TLS based on the RSA algorithm.
- **SHA-1**. Secure Hash Algorithm.
- **SKIPJACK**. A classified symmetric-key algorithm implemented in FORTEZZA-compliant hardware used by the U.S. Government.
- **Triple-DES**. DES applied three times.

# TLS Characteristics

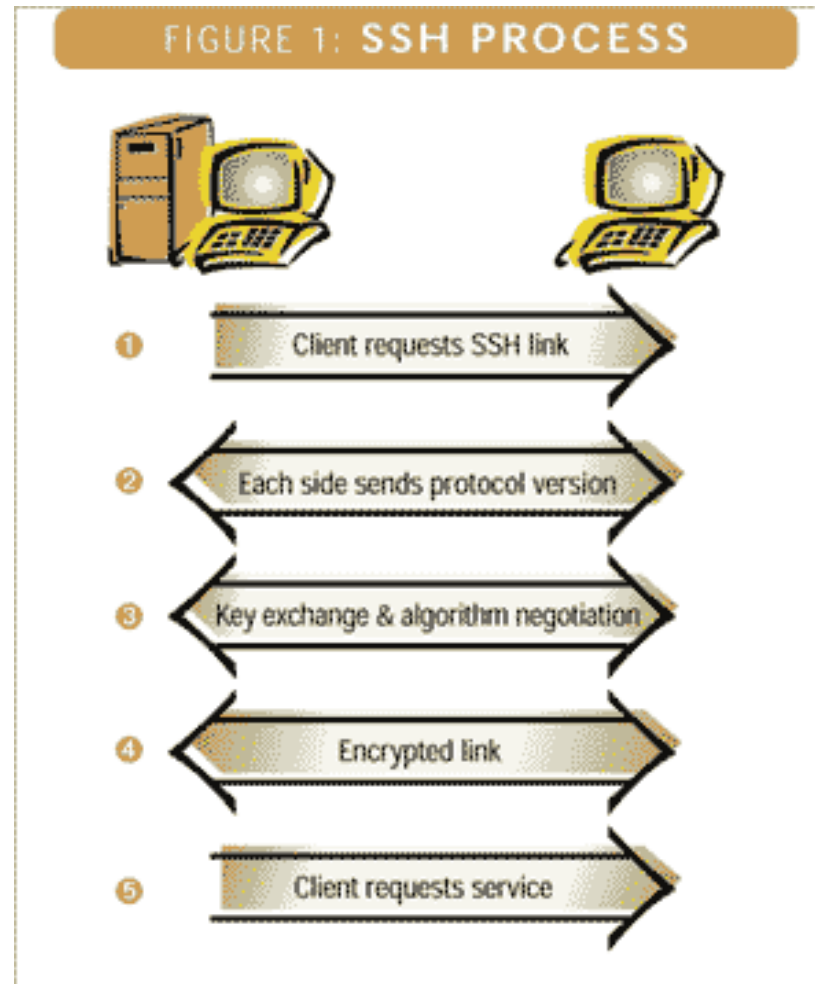
- Designed for multiple protocols (HTTP, SMTP, NNTP, POP3, FTP).
- Mainly used in HTTP.
- Compute-intensive...

# SSL Challenge...

- SSL challenge – broken in 1995
- The 40-bit secret part of the key was broken by a brute force search on a network of about 120 workstations and a few parallel computers at INRIA, France.
- The key was found after scanning a little more than half the key space in 8 days.
  - Moore's law: the computational power of a machine doubles every 18 months
  - Today's computers are about 128x faster than in 1995.
  - $8\text{days}/128 = 1.5 \text{ hours}$

# **SSH: Secure Shell**

# SSH Protocol



# SSH Protocol

- Client → Server: request a new session.
- Server → Client: asks for the version SSH-1 or SSH-2?
- Client → Server: sends the version.
- Server → Client: sends Public\_Key and a Temporary\_Key
- Client → Server: creates a Session\_Key that is encrypted with the Temporary\_Key and the Public\_Key of the Server.
- Server: uses Private\_Key to decrypt and gets the Session\_Key.
- The server authenticates the client with one of these options: DSA, RSA, OpenPGP or password.

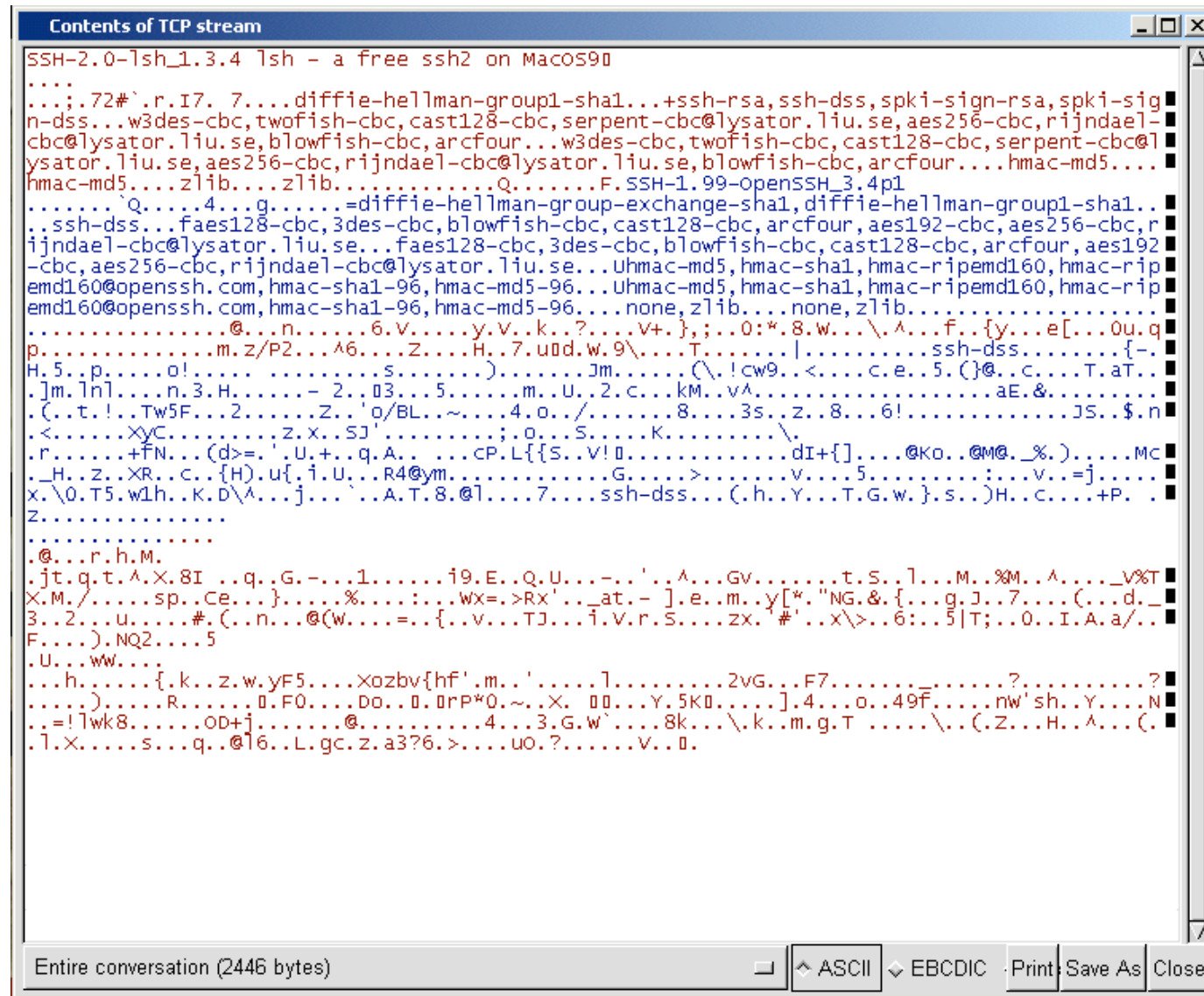


# Sniffing a Telnet Session

## Contents of TCP stream

```
.....#.....#..'..$......#.....P.....#..'..$......ansi..
.....
This system is for the use of authorized account holders only.
Individuals using this computer system without authority, or in excess
of their authority, are subject to having all of their activities on
this system monitored and recorded by system personnel.
In the course of monitoring individuals improperly using this system,
or in the course of routine system maintenance, the activities of
authorized account holders may also be monitored.
Anyone using this system expressly consents to such monitoring and is
advised that if such monitoring reveals possible evidence of criminal
activity, system personnel may provide the evidence gathered to law
enforcement officials.
login: sswaannbbeerrg
Password: some-dumb-password
Last login: Thu Nov 21 18:57:44 from 
University of Minnesota Interactive UNIX Service
All users of University-provided accounts are responsible for reading and
complying with the University's Acceptable Use Policy, available online at:
  http://www.fpd.finop.umn.edu/groups/ppd/documents/policy/Acceptable_Use.cfm
  http://www.fpd.finop.umn.edu/groups/ppd/documents/appendix/useguidelines.cfm
Need help? Call (612) 301-4357 or send e-mail to help@tc.umn.edu.
Help line hours are Mon-Thu 8am-11pm; Fri 8am-5pm; Sat 12-5pm; Sun 5pm-11pm.
NOTE 8/26/2002: Please check all scripts that use /export/home as the
home directory, proper home directory is now /home
You have mail.
```

# Sniffing a SSH Session



The screenshot shows a network sniffer window titled "Contents of TCP stream". The main area displays the raw data of an SSH session, which is a mix of ASCII text and binary data represented by hex escapes. The text is color-coded: red for ASCII and blue for binary. The session starts with a banner: "SSH-2.0-lsh\_1.3.4 lsh - a free ssh2 on MacOS9". This is followed by a list of supported algorithms, a key exchange process, and a series of authentication attempts and responses. The bottom status bar indicates "Entire conversation (2446 bytes)" and provides buttons for "ASCII", "EBCDIC", "Print", "Save As", and "Close".

```
SSH-2.0-lsh_1.3.4 lsh - a free ssh2 on MacOS9
...72#`r.I7. 7....diffie-hellman-group1-sha1...+ssh-rsa,ssh-dss,spki-sign-rsa,spki-sig
n-dss...w3des-cbc,twofish-cbc,cast128-cbc,serpent-cbc@lysator.liu.se,aes256-cbc,rijndael-
cbc@lysator.liu.se,blowfish-cbc,arcfour...w3des-cbc,twofish-cbc,cast128-cbc,serpent-cbc@l
ysator.liu.se,aes256-cbc,rijndael-cbc@lysator.liu.se,blowfish-cbc,arcfour....hmac-md5....
hmac-md5....zlib....zlib....Q.....F.SSH-1.99-OpenSSH_3.4p1
.....`Q.....4...g.....=diffie-hellman-group-exchange-sha1,diffie-hellman-group1-sha1..
..ssh-dss...faes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc,r
ijndael-cbc@lysator.liu.se...faes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192
-cbc,aes256-cbc,rijndael-cbc@lysator.liu.se...Uhmach-md5,hmac-sha1,hmac-ripemd160,hmac-rip
emd160@openssh.com,hmac-sha1-96,hmac-md5-96...Uhmach-md5,hmac-sha1,hmac-ripemd160,hmac-rip
emd160@openssh.com,hmac-sha1-96,hmac-md5-96....none,zlib....none,zlib.....
.....@...n.....6.V.....y.V..k..?...v+.,;.:0*:8.w...\.^...f...{y...e[...Ou.q
p.....m.z/P2...^6....Z...H..7.u0d.w.9\...T.....|.....ssh-dss.....{-
H.5..p.....o!.....s.....).....Jm.....(\..!cw9..<....c.e..5..()@..c...T.aT..
.]m.ln]....n.3.H.....- 2..03...5.....m..U..2.C...kM..v^.....aE.&.....
.(.t.!.Tw5F...2.....Z.. 'o/BL..~....4.o../.....8....3s..z..8...6!.....JS..$.n
.<.....XyC.....z.x..SJ'.....;..o...S.....K.....\
.r.....+FN... (d>=. 'U.+..q.A.. ...cP.L[{S..V!0.....dI+{]....@ko..@M@._%).....Mc
._H..z..XR..c..{H).u{.i.U...R4@ym.....G.....>.....v....5.....:..V...=j.....
x.\0.T5.w1h..K.D\^...j...`..A.T.8.@l....7....ssh-dss... (h..Y...T.G.w..}.s..)H..c....+P..
Z.....
.....
.@...r.h.M.
.jt.q.t.^X.8I ..q..G.-...1.....i9.E..Q.U...-..'.^...Gv.....t.S..l...M..%M..^..._v%T
X.M./.....sp..Ce...}....%.....wx=>Rx'...at.- ]..e..m..y[*."NG.&{...g.J..7....(..d._
3..2...u.....#.(..n...@ (w.....={..v...TJ...i.V.r.S...zx.#'..x\>..6:..5|T;..0..I.A.a/..
F....).NQ2....5
.U...ww...
...h.....{.k..z.w.yF5...Xozbv{hf'.m..'.....l.....2vG...F7.....?.....?
.....).....R.....0.F0....Do..0..0rP*0..~..X..00...Y.5K0.....].4...o..49f.....nw'sh..Y...N
..=!lwK8.....0D+]......@.....4...3.G.w'.....8k...\.k..m.g.T .....\. (Z...H..^..(
.l.X.....s...q..@16..L.gc.z.a3?6.>....uo.?.....V..0.
```