

# Chapter 8

# Network Security

## A note on the use of these ppt slides:

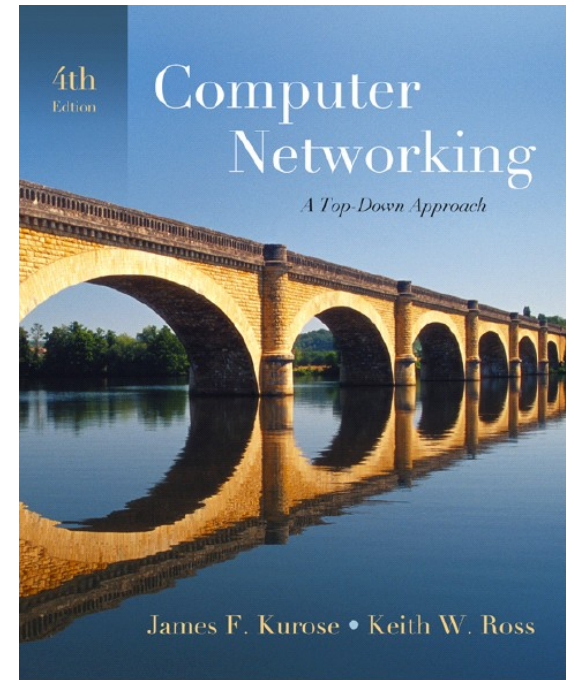
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- ❑ If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- ❑ If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2007

J.F Kurose and K.W. Ross, All Rights Reserved



*Computer  
Networking: A Top  
Down Approach ,  
4<sup>th</sup> edition.  
Jim Kurose, Keith  
Ross  
Addison-Wesley, July  
2007.*

# Network security

## Foundations:

- what is security?
  - cryptography
  - authentication
  - message integrity
  - key distribution and certification

## Security in practice:

- application layer: secure e-mail
- transport layer: Internet commerce, SSL, SET
- network layer: IP security

# What is network security?

**Secrecy (confidentiality):** only sender, intended receiver should understand message contents

- sender encrypts messages
- receiver decrypts messages

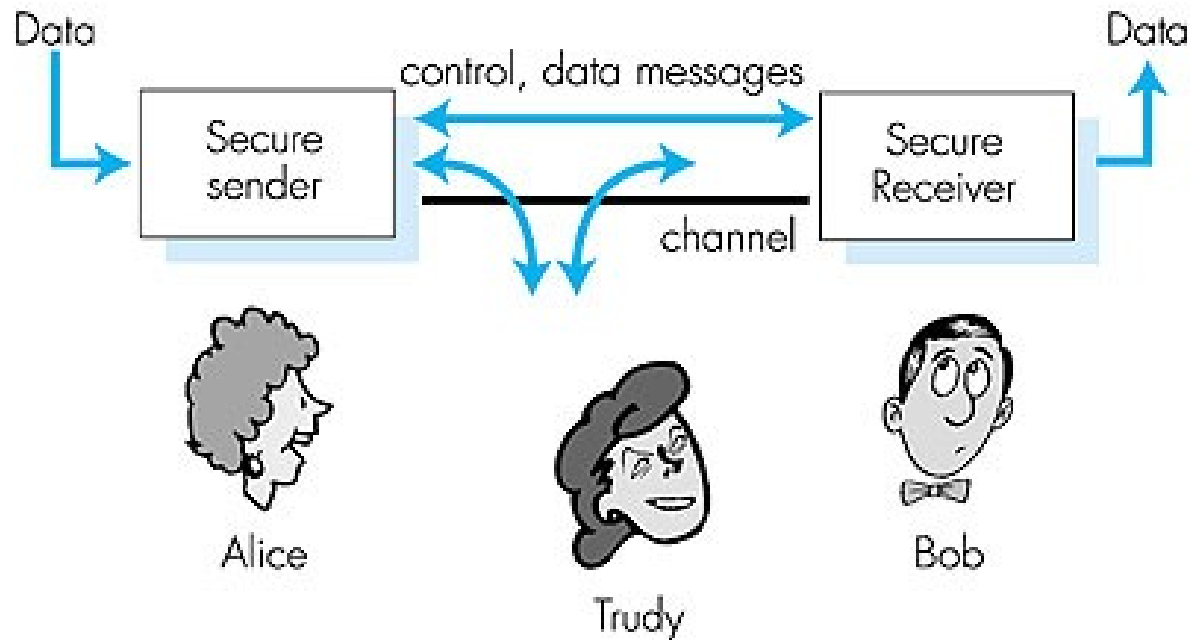
**Authentication:** sender, receiver want to confirm identity of each other

**Message Integrity:** sender, receiver want to ensure message not altered (in transit) without detection

# Confidentiality:

Or “The art of encryption”

# Friends and enemies: Alice, Bob, Trudy



- well-known in network security world
- Bob, Alice want to communicate 'securely'
- Trudy, the intruder may intercept, delete, add messages

# There are bad guys (and girls) out there!

Q: What can a “bad guy” do?

A: a lot!

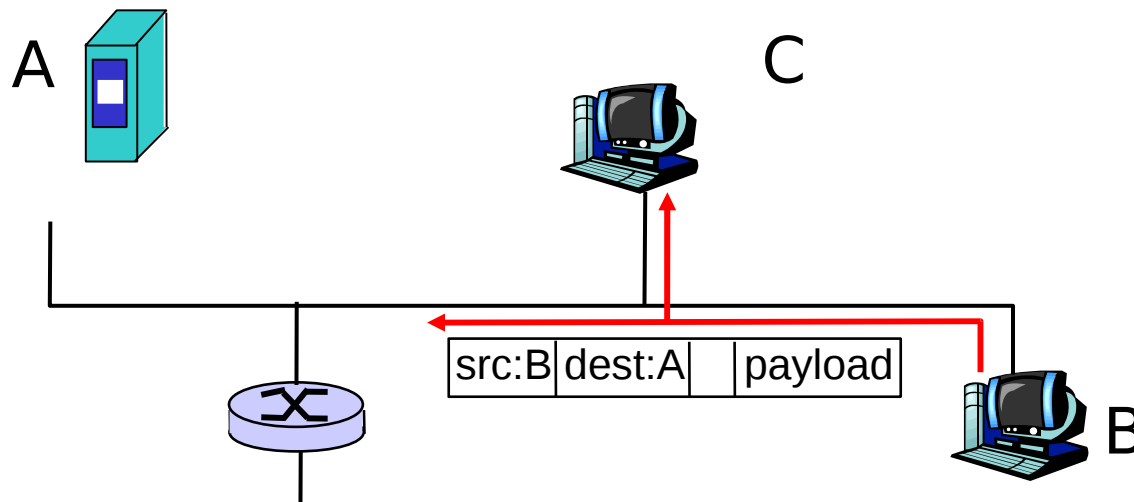
- *eavesdrop*: intercept messages
- actively *insert* messages into connection
- *impersonation*: can fake (spoof) source address in packet (or any field in packet)
- *hijacking*: “take over” ongoing connection by removing sender or receiver, inserting himself in place
- *denial of service*: prevent service from being used by others (e.g., by overloading resources)

*more on this later .....*

# Internet security threats

## Packet sniffing:

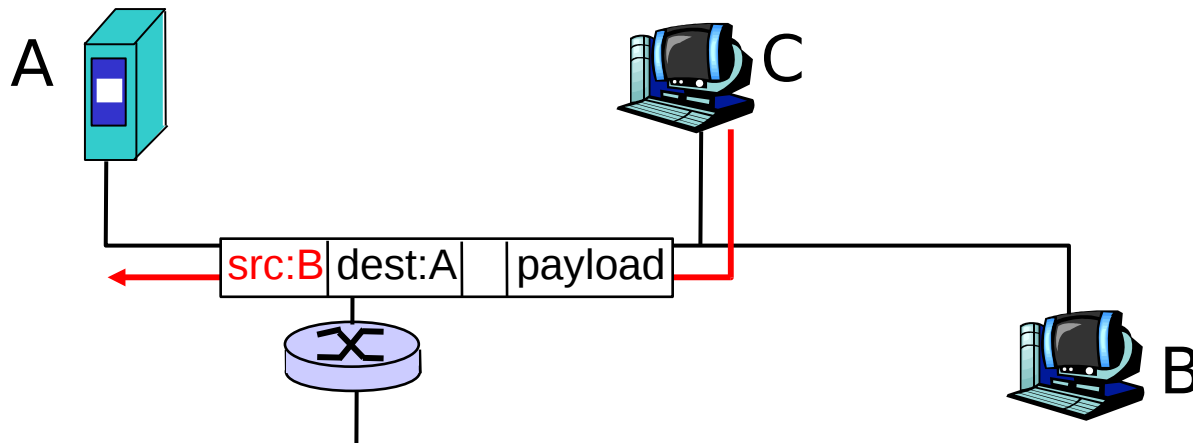
- broadcast media (remember CSMA/CD protocol)
- promiscuous NIC reads all packets passing by
- can read all unencrypted data (e.g. passwords)
- e.g.: C sniffs B's packets



# Internet security threats

## IP Spoofing:

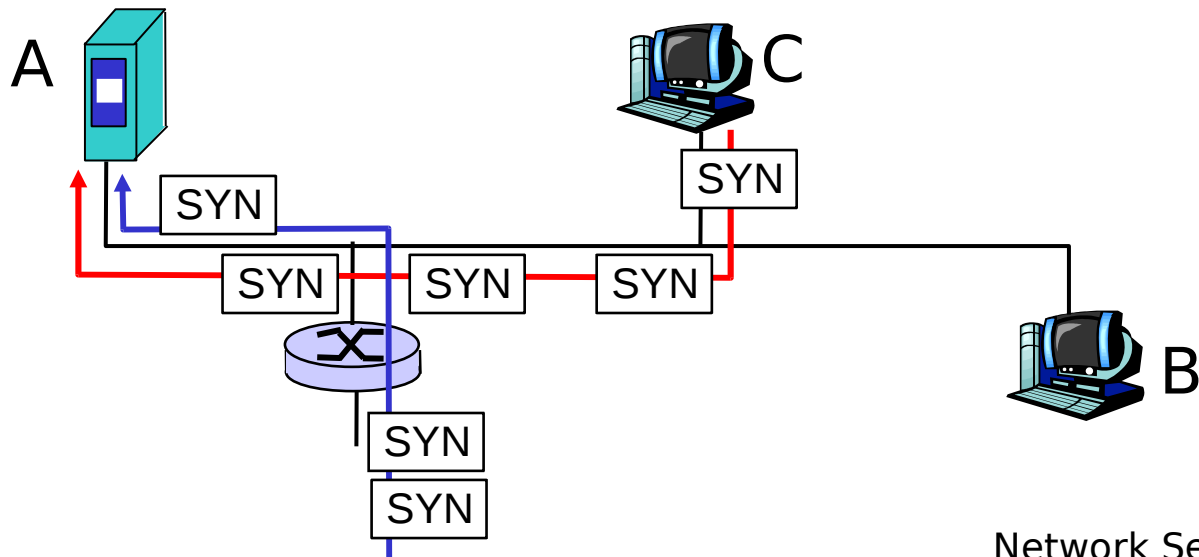
- can generate raw IP packets directly from application, putting any value into IP source address field
- receiver can't tell if source is spoofed
- e.g.: C pretends to be B



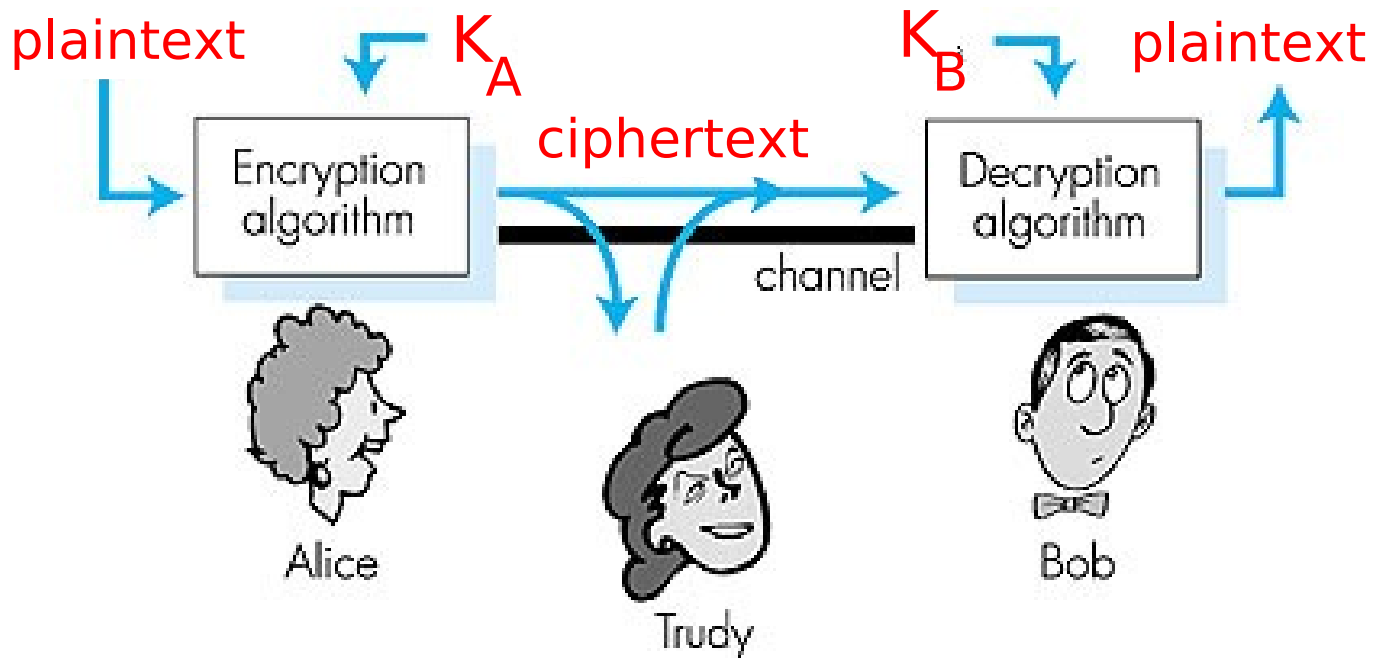
# Internet security threats

## Denial of service (DOS):

- flood of maliciously generated packets swamp receiver
- Distributed DOS (DDOS): multiple coordinated sources swamp receiver
- e.g., C and remote host SYN-attack A



# The language of cryptography

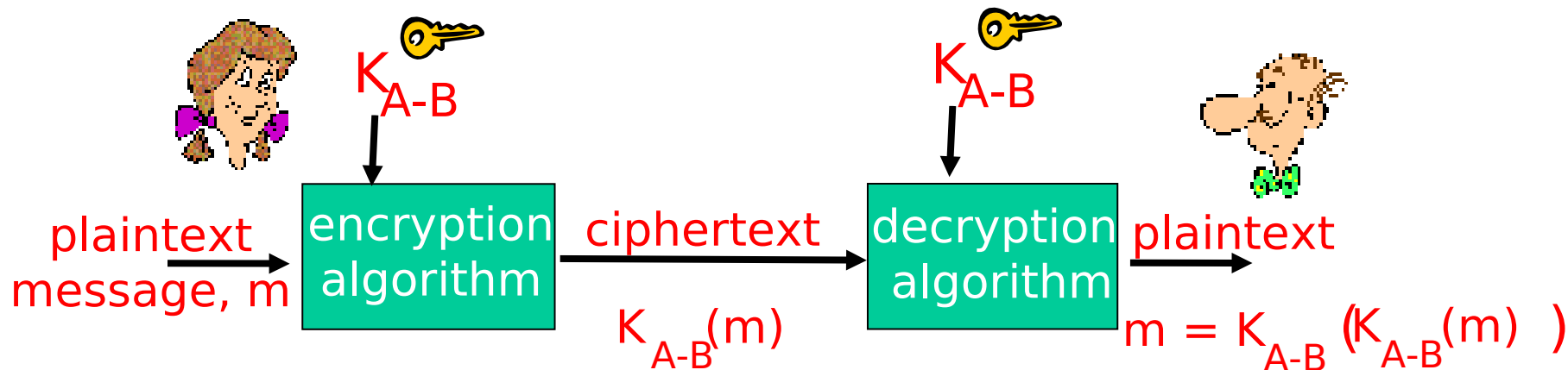


**symmetric key** crypto: sender, receiver keys identical

**public-key** crypto: encrypt key *public*, decrypt key *secret*



# Symmetric key cryptography



- symmetric key** crypto: Bob and Alice share know same (symmetric) key:  $K_{A-B}$
- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher
  - Q: how do Bob and Alice agree on key value?

# Symmetric key crypto: DES

## DES: Data Encryption Standard

- US encryption standard [NIST 1993]
  - 56-bit symmetric key, 64 bit plain text input
  - How secure is DES?
    - DES Challenge: 56-bit-key-encrypted phrase ('Strong cryptography makes the world a safer place') decrypted (brute force) in 4 months
    - no known backdoor decryption approach
  - making DES more secure
    - use three keys sequentially (3-DES) on each datum
    - use cipher-block chaining

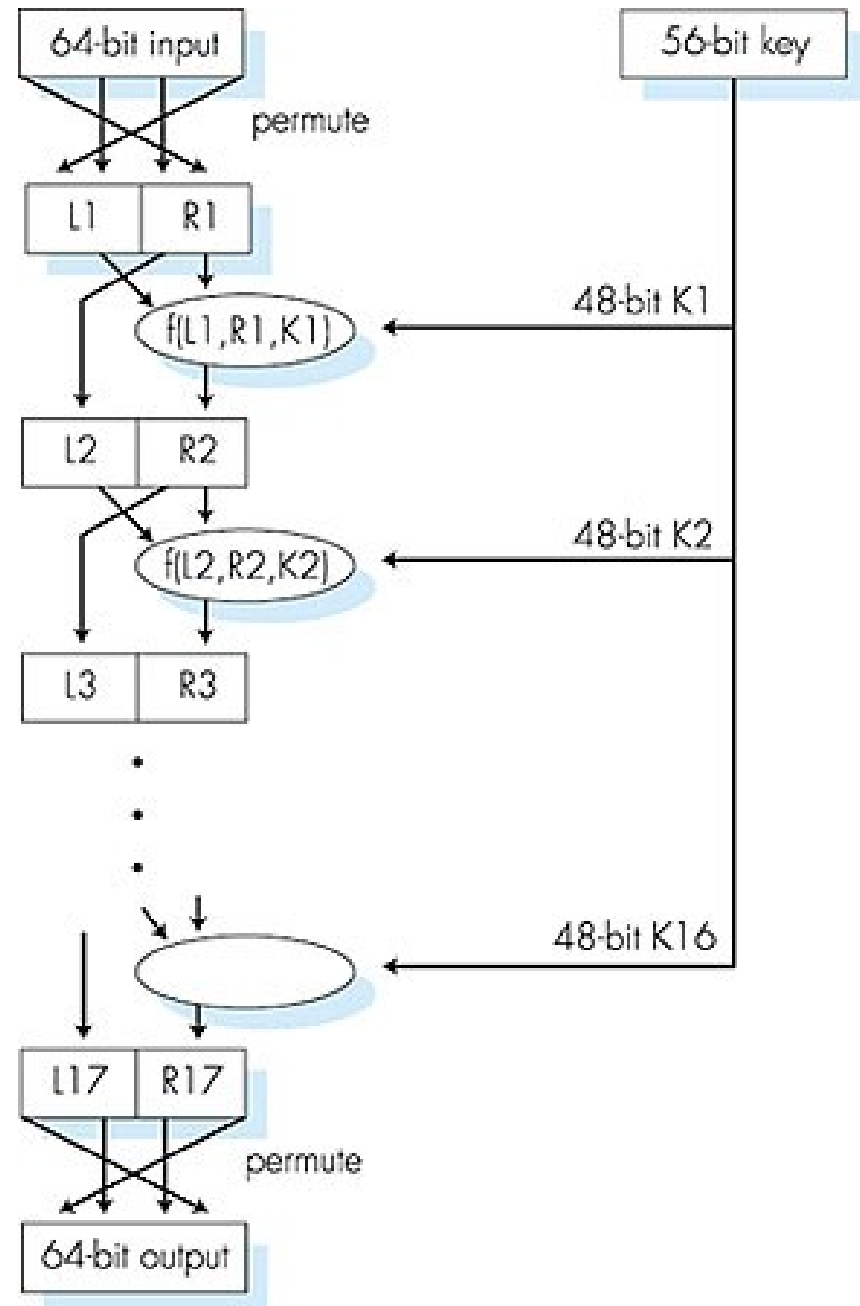
# Symmetric key crypto: DES

## DES operation

initial permutation

16 identical 'rounds'  
of function  
application, each  
using different 48  
bits of key

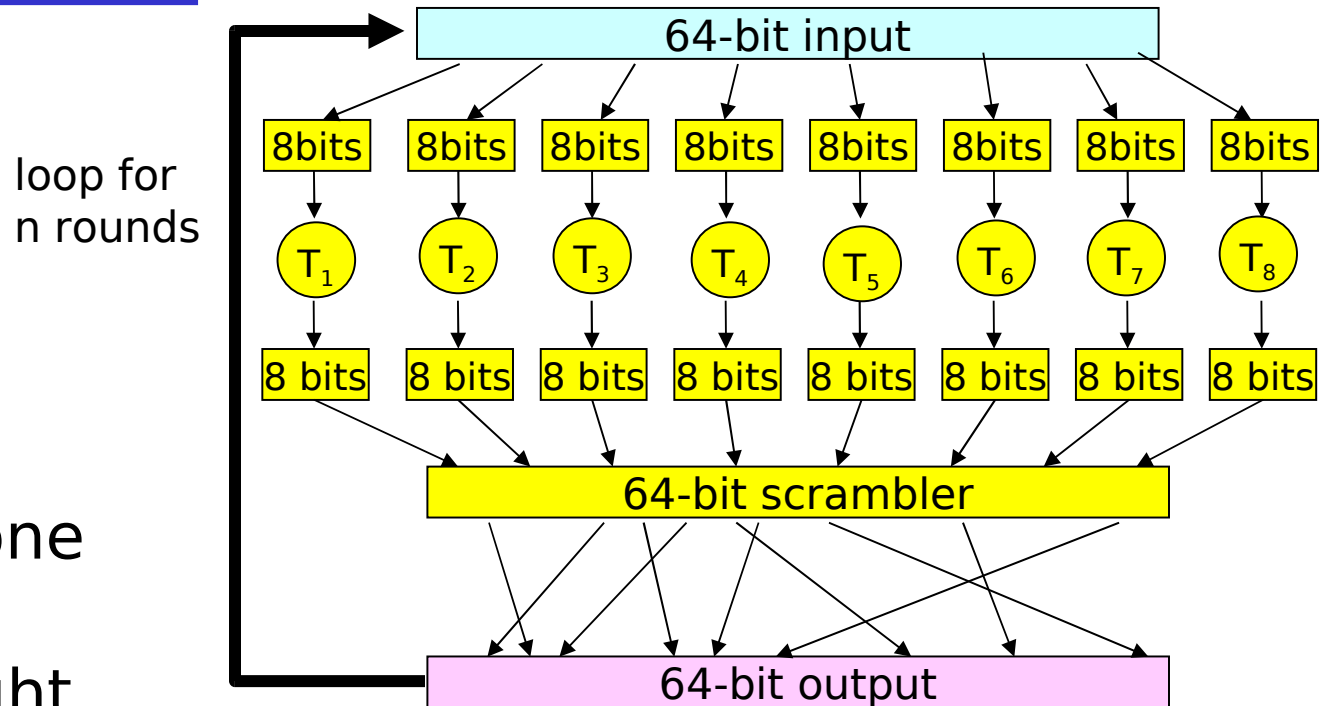
final permutation



# AES: Advanced Encryption Standard

- new (Nov. 2001) symmetric-key NIST standard, replacing DES
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

# Block Cipher



- one pass through: one input bit affects eight output bits
- multiple passes: each input bit affects all output bits
- block ciphers: DES, 3DES, AES

# Public Key Cryptography

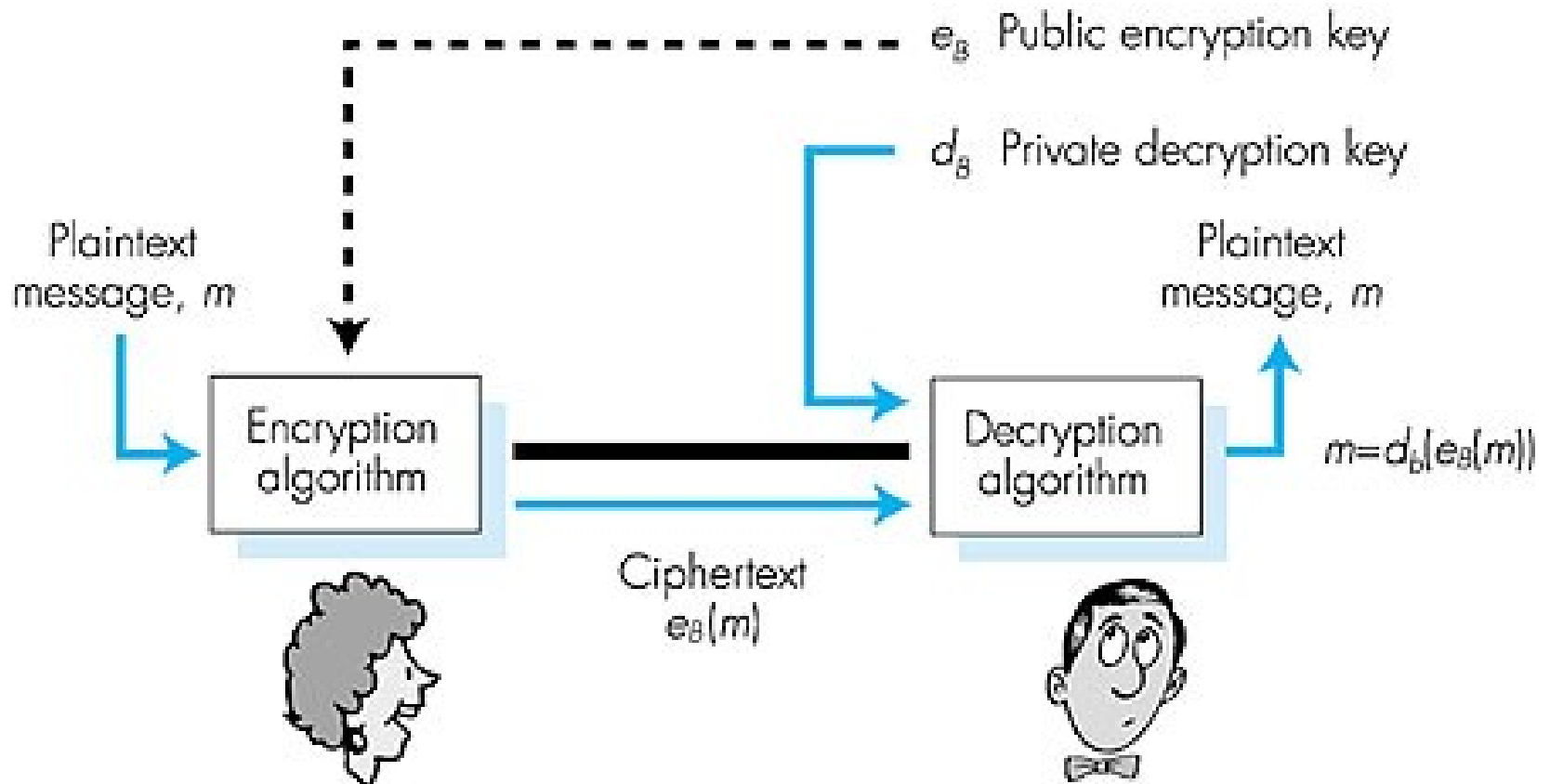
## symmetric key crypto

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never met)?
- Typical problem in the Internet

## public key cryptography

- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- encryption key *public* (known to *all*)
- decryption key private (known only to receiver)

# Public key cryptography



# Public key encryption algorithms

Two inter-related requirements:

① need  $d_B(\cdot)$  and  $e_B(\cdot)$  such that

$$d_B(e_B(m)) = m$$

② need public and private keys  
for  $d_B(\cdot)$  and  $e_B(\cdot)$

**RSA:** Rivest, Shamir, Adelson algorithm

# RSA: Choosing keys

1. Choose two large prime numbers  $p, q$ .  
(e.g., 1024 bits each)
2. Compute  $n = pq$ ,  $z = (p-1)(q-1)$
3. Choose  $e$  (with  $e < n$ ) that has no common factors with  $z$ . ( $e, z$  are 'relatively prime').
4. Choose  $d$  such that  $ed-1$  is exactly divisible by  $z$ .  
(in other words:  $ed \bmod z = 1$  ).
5. *Public* key is  $(n, e)$ . *Private* key is  $(n, d)$ .

# RSA: Encryption, decryption

0. Given  $(n,e)$  and  $(n,d)$  as computed above

1. To encrypt bit pattern,  $m$ , compute

$$c = m^e \bmod n \quad (\text{i.e., remainder when } m^e \text{ is divided by } n)$$

2. To decrypt received bit pattern,  $c$ , compute

$$m = c^d \bmod n \quad (\text{i.e., remainder when } c^d \text{ is divided by } n)$$

Magic happens!

$$m = (m^e \bmod n)^d \bmod n$$

# RSA example:

Bob chooses  $p=5$ ,  $q=7$ . Then  $n=35$ ,  $z=24$ .

$e=5$  (so  $e$ ,  $z$  relatively prime).

$d=29$  (so  $ed-1$  exactly divisible by  $z$ ).

encrypt:	<u>letter</u>	<u>m</u>	<u><math>m^e</math></u>	<u><math>c = m^e \bmod n</math></u>
	I	12	248832	17
decrypt:	<u>c</u>	<u><math>c^d</math></u>	<u><math>m = c^d \bmod n</math></u>	<u>letter</u>
	17		12	I

$c^d = 481968572106750915091411825223071697$  - too big !! (int type)

# How to solve this problem:

$C^d=17^{29}$ : break it into powers that are multiple of 2.

$$17^{29}=17^{16} \cdot 17^8 \cdot 17^4 \cdot 17$$

$$17^4 \bmod 35 = 11 \text{ hence } 17^4 \bmod 35 = 11 \bmod 35,$$

So 11 can substitute  $17^4$ ,  $17^8$ , and  $17^{16}$  in the expression:

$$17^{29} \bmod 35 = 11^4 \cdot 11^2 \cdot 11 \cdot 17 \bmod 35$$

Also  $11^2 \bmod 35 = 16$

$$17^{29} \bmod 35 = 16^2 \cdot 16 \cdot 11 \cdot 17 \bmod 35$$

This is equivalent to do calculate x, y and z as follows:

$$x = n^4 \bmod 35$$

$$y = x^2 \bmod 35$$

$$z = y^2 \bmod 35$$

$$m = z \cdot y \cdot x \cdot c \bmod 35 \text{ (smaller !!)}$$

*Alternatively:* use a similar method to the one used in CRC.

Remember that  $(c.c.c) \bmod n = (c.c) (c \bmod n) \dots$

# How to solve this problem:

**Repeated Squaring:** calculate  $y = x^e \bmod n$

```
int repeatsquare( int x, int e, int n) {  
  
    y=1;//initialize y to 1, very important  
    while (e > 0) {  
        if (( e % 2 ) == 0) {  
            x = (x*x) % n;  
            e = e/2;  
        }  
        else {  
            y = (x*y) % n;  
            e = e-1;  
        }  
    }  
    return y; //the result is stored in y  
}
```

# RSA: Why:

$$m = (m^e \bmod n)^d \bmod n$$

Number theory result: If  $p, q$  prime,  $n = pq$ ,  
then

$$x^y \bmod n = x^{y \bmod (p-1)(q-1)} \bmod n$$

$$(m^e)^d \bmod n = m^{ed} \bmod n$$

$$= m^{ed \bmod (p-1)(q-1)} \bmod n$$

(using number theory result above)

$$= m^1 \bmod n$$

(since we chose  $ed$  to be divisible by  
 $(p-1)(q-1)$  with remainder 1 )

$$= m$$

# RSA:how strong is it??

## RSA Challenges:

- Historically: a prize was offered to anyone who could break an RSA key of a certain size (See [www.rsa.com/rsalabs/](http://www.rsa.com/rsalabs/) ), **challenge no longer active.**
  - US\$200,000.00 for a 2048 bits factorization problem
  - Last challenge solved (no money for it...):
    - RSA-768 Factored in 2009 by T. Kleinjung et.al.
      - Using a powerful parallel machine and very clever algorithms (4400 1GHZ CPU years, 3 calendar years)
  - Currently RSA-1024 is commonly used in practice
  - RSA key's size matters, see next...

# RSA:how strong is it??

## Common sense calculation:

- Brute force factorization
  - Try all the prime number P that are smaller than the Key
  - When  $\text{Key mod } P = 0$  then found the factors
- How long can it take depends on the RSA key size
- Suppose we have a key of 200 bits and the factors are approximately of the same order of digits
- Each key will have  $\sim 10^{100}$  trial divisions to do
- A 1Gflops machine could do  $10^9$  trials per second
- Say we have  $10^9$  machines in a cluster (massively parallel mach.)
- As we have only  $\sim 10^8$  seconds/year, it would take
  - $10^{74}$  years !!!
  - Remember that the Earth is  $\sim 10^9$  years old

# RSA:how to implement it

## A simplistic approach:

- Translate each byte to a decimal number
- Use the RSA algorithm for each character
  - Problem 1: only a few possible numbers will be used
    - e.g., 256 possible characters may be mapped into non-existing character if the key is large
  - Problem 2: easy to break encryption due to language characteristics
    - e.g., English uses double character such as 'll', 'rr' etc.
    - A 'dictionary' attack could decrypt the message without ever finding the factorization
- In practice the string becomes a large binary number...

# RSA: another important property

The following property will be *very* useful later:

$$\underbrace{K_B^- (K_B^+ (m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+ (K_B^- (m))}_{\text{use private key first, followed by public key}}$$

use public key  
first, followed  
by private key

use private  
key first,  
followed by  
public key

*Result is the  
same!*

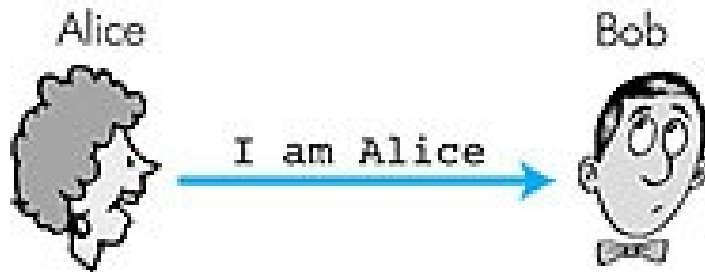
# Authentication:

Ensure that the communication is with the  
“correct” machine

# Authentication

Goal: Bob wants Alice to **prove** her identity to him

Protocol ap1.0: Alice says "I am Alice"

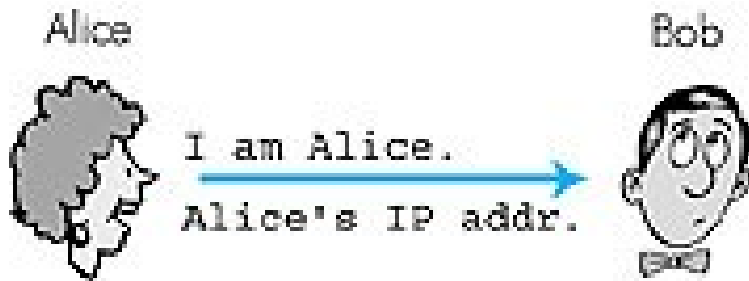


Failure scenario??



# Authentication: another try

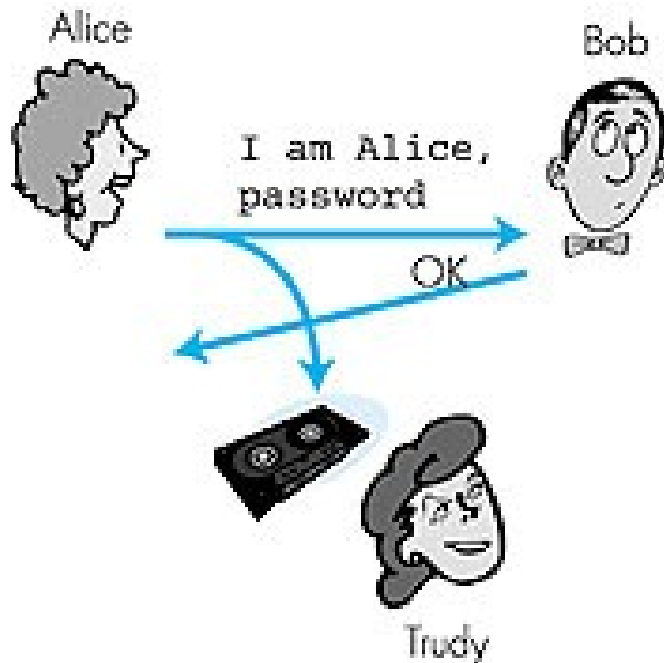
Protocol ap2.0: Alice says "I am Alice" and sends her IP address along to prove it.



Failure scenario??

# Authentication: another try

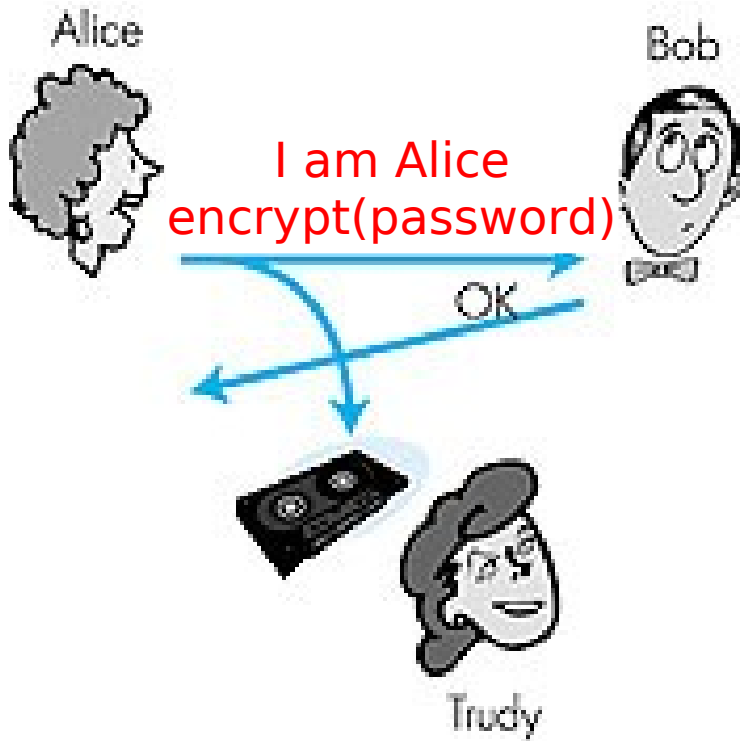
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to prove it.



Failure scenario?

# Authentication: yet another try

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



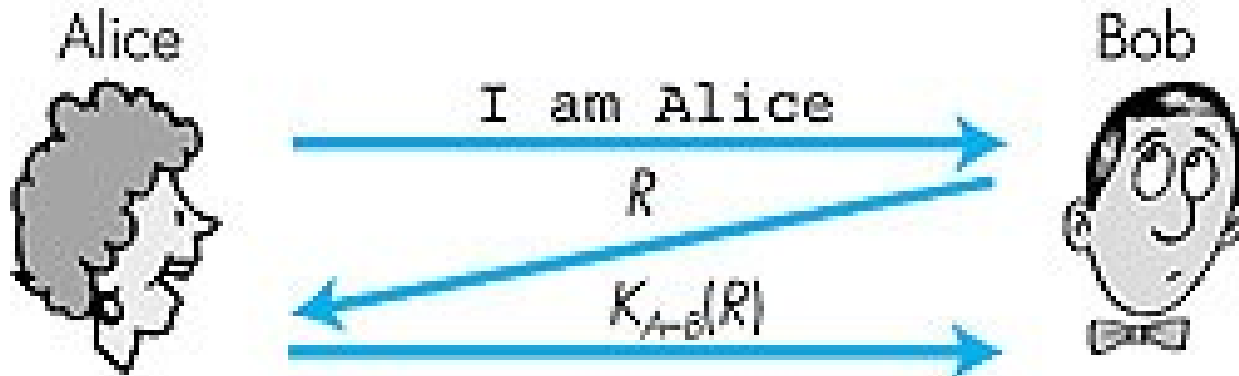
Failure scenario?

# Authentication: yet another try

Goal: avoid playback attack

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

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



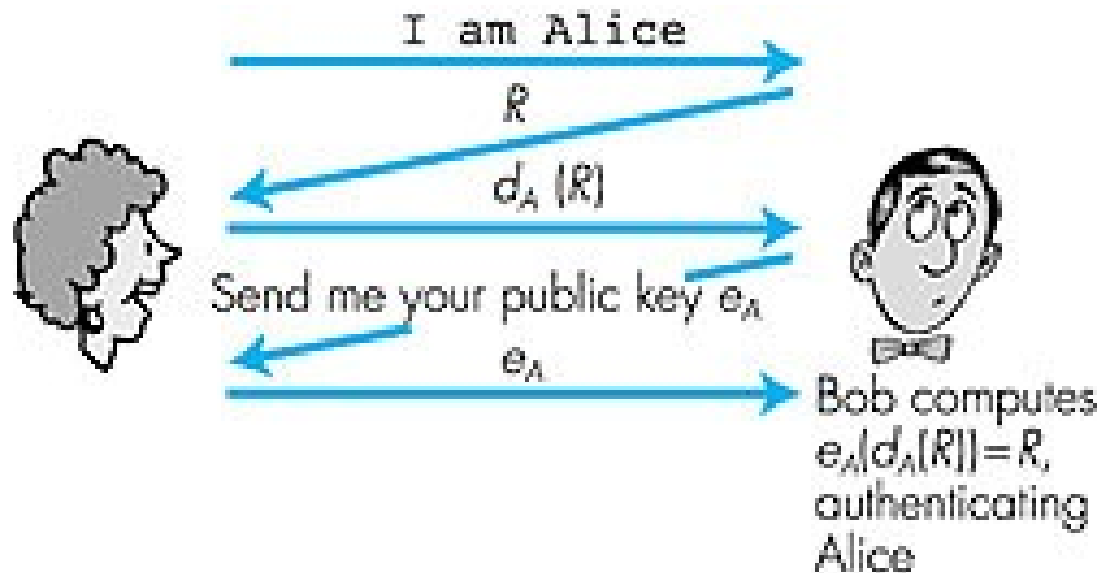
Failures, drawbacks?

# Authentication: ap5.0

ap4.0 requires shared symmetric key

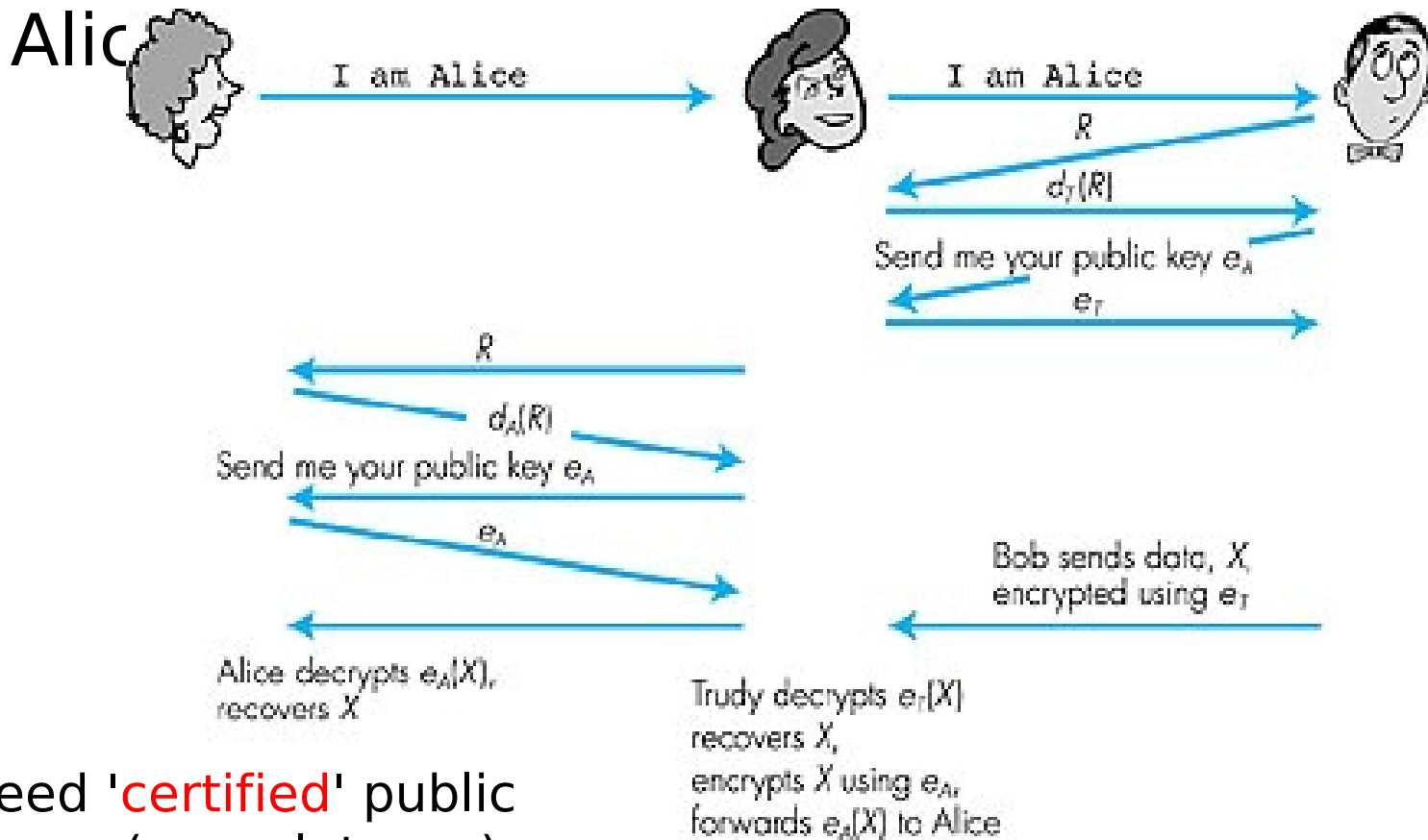
- ▢ problem: how do Bob, Alice agree on key
- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



# ap5.0: security hole

**Man (woman) in the middle attack:** Trudy poses as Alice (to Bob) and as Bob (to Alice)



Need '**certified**' public keys (more later ...)

# Message Integrity:

Ensure that the message is not tampered with.

# Message Integrity

Bob receives msg from Alice, wants to ensure:

- message originally came from Alice
  - message not changed since sent by Alice

## Cryptographic Hash:

- takes input  $m$ , produces fixed length value,  $H(m)$ 
  - e.g., as in Internet checksum
- computationally infeasible to find two different messages,  $x$ ,  $y$  such that  $H(x) = H(y)$ 
  - equivalently: given  $m = H(x)$ , ( $x$  unknown), can not determine  $x$ .
  - note: Internet checksum *fails* this requirement!

# Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of message
- is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

<u>message</u>	<u>ASCII format</u>	<u>message</u>	<u>ASCII format</u>
I O U 1	49 4F 55 31	I O U <u>9</u>	49 4F 55 <u>39</u>
0 0 . 9	30 30 2E 39	0 0 . <u>1</u>	30 30 2E <u>31</u>
9 B 0 B	39 42 4F 42	9 B 0 B	39 42 4F 42
	<u>B2 C1 D2 AC</u>		<u>B2 C1 D2 AC</u>

different messages  
but identical checksums!

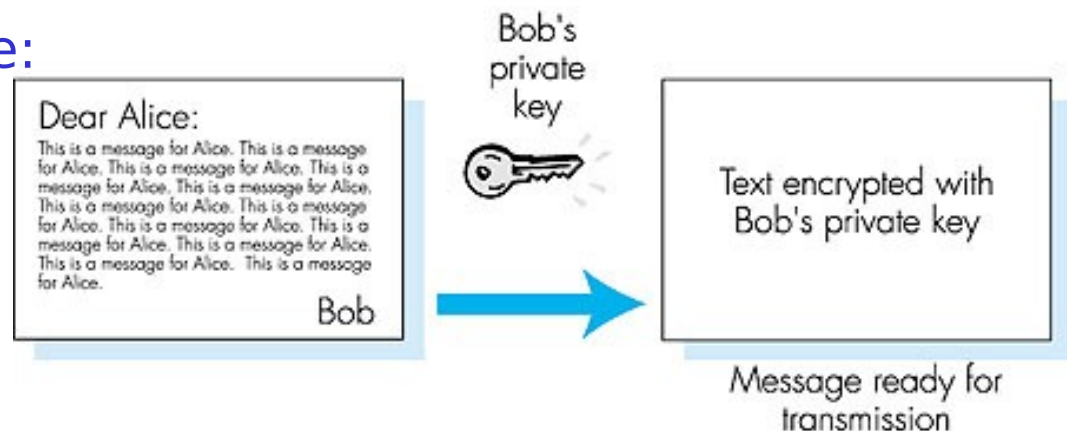
# Digital Signatures

Cryptographic technique analogous to hand-written signatures.

- Sender (Bob) digitally signs document, establishing he is document owner/creator.
- **Verifiable, non-forgable:** recipient (Alice) can verify that Bob, and no one else, signed document.

Simple digital signature for message  $m$ :

- Bob encrypts  $m$  with his public key  $d_B$ , creating signed message,  $d_B(m)$ .
- Bob sends  $m$  and  $d_B(m)$  to Alice.



# Digital Signatures (more)

- Suppose Alice receives msg  $m$ , and digital signature  $d_B(m)$
- Alice verifies  $m$  signed by Bob by applying Bob's public key  $e_B$  to  $d_B(m)$  then checks  $e_B(d_B(m)) = m$ .
- If  $e_B(d_B(m)) = m$ , whoever signed  $m$  must have used Bob's private key.

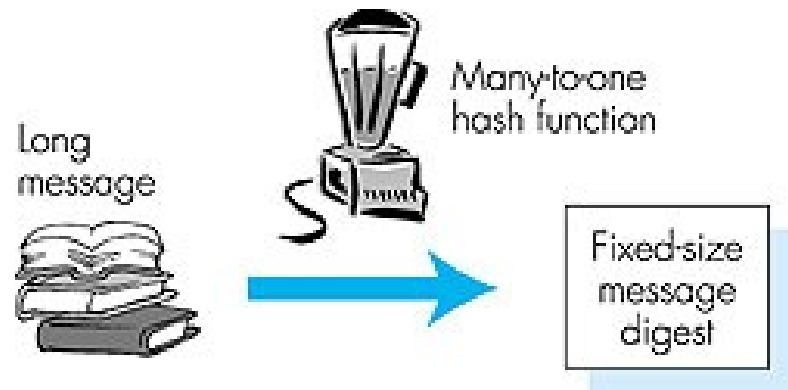
## Alice thus verifies that:

- Bob signed  $m$ .
- No one else signed  $m$ .
- Bob signed  $m$  and not  $m'$ .

## Non-repudiation:

- Alice can take  $m$ , and signature  $d_B(m)$  to court and prove that Bob signed  $m$ .

# Message Digests



Computationally expensive to public-key-encrypt long messages

**Goal:** fixed-length, easy to compute digital signature, 'fingerprint'

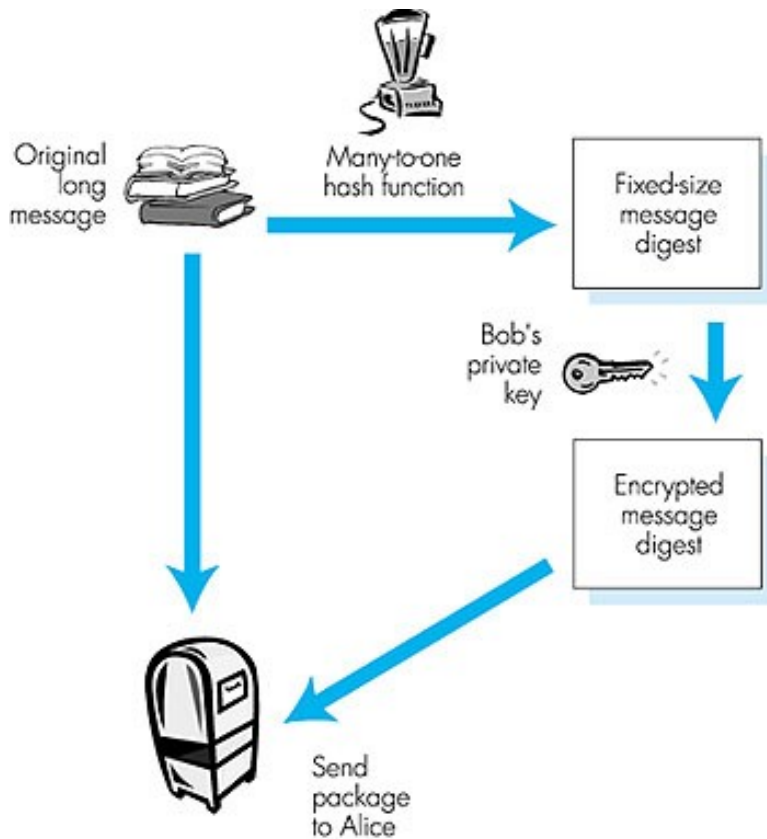
- apply hash function  $H$  to  $m$ , get fixed size message digest,  $H(m)$ .

## Hash function properties:

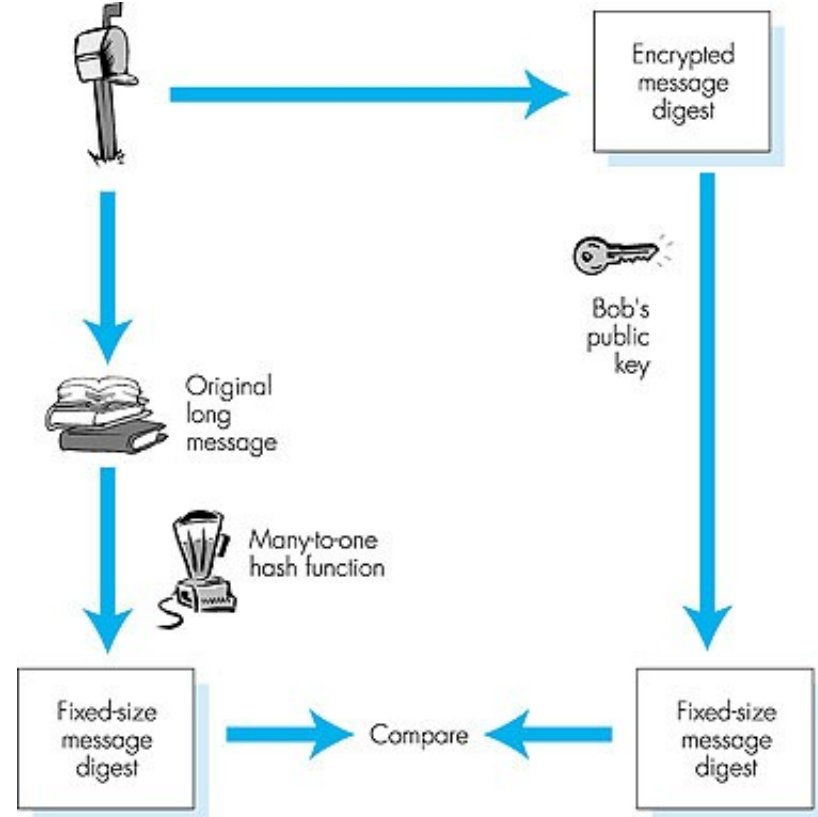
- Many-to-1
- Produces fixed-size msg digest (fingerprint)
- Given message digest  $x$ , computationally infeasible to find  $m$  such that  $x = H(m)$
- computationally infeasible to find any two messages  $m$  and  $m'$  such that  $H(m) = H(m')$ .

# Digital signature = Signed message digest

Bob sends digitally signed message:



Alice verifies signature and integrity of digitally signed message:



# Hash Function Algorithms

- Internet checksum would make a poor message digest.
  - Too easy to find two messages with same checksum.
  - Even using a 128-bit CRC it would be easy to find a second message to fit to the CRC
- MD5 hash function widely used.
  - Computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string  $x$ , appears difficult to construct msg  $m$  whose MD5 hash is equal to  $x$ .
- SHA-1 is also used.
  - US standard
  - 160-bit message digest

# Hash Function Algorithms

- **MD5**
  - Try that using md5sum command in Linux ...
  - MD5 is a very reliable way to fingerprint a file
  - From rfc1321: ...*"The MD5 algorithm] takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest"...*

# Trusted Intermediaries

## Problem:

- How do two entities establish shared secret key over network?

## Solution:

- trusted key distribution centre (KDC) acting as intermediary between entities

## Problem:

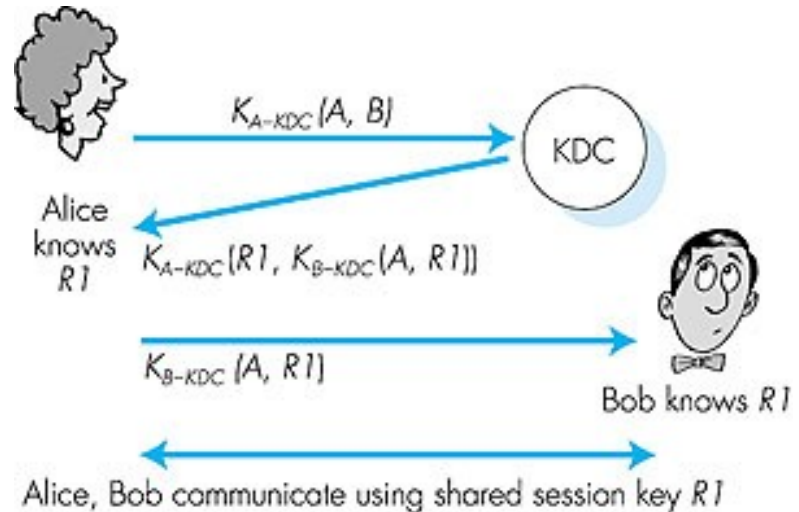
- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she know it is Bob's public key, not Trudy's?

## Solution:

- trusted certification authority (CA)

# Key Distribution Center (KDC)

- Alice, Bob need shared symmetric 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.



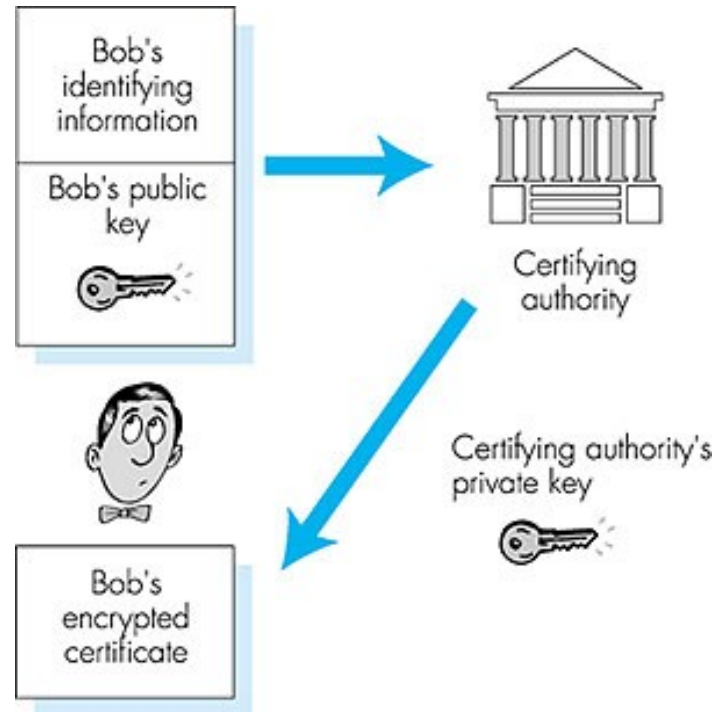
- Alice communicates with KDC, gets session key  $R1$ , and  $K_{B-KDC}(A, R1)$
- Alice sends Bob  $K_{B-KDC}(A, R1)$ , Bob extracts  $R1$
- Alice, Bob now share the symmetric key  $R1$ .

# Key Distribution Center (KDC)

- Example: Kerberos
- See <http://web.mit.edu/kerberos/>
- "Kerberos is a network authentication protocol"
- "...firewalls assume that the bad guys are on the outside, which is often a very bad assumption..."
- Kerberos is freely available from MIT!
- Limitations of such approach:
  - Assumes users uses 'good' passwords
  - Assumes that only the network connections are vulnerable to attacks
- ==> importance of OS security cannot be underestimated

# Certification Authorities

- Certification authority (CA) binds public key to particular entity.
- Entity (person, router, etc.) can register its public key with CA.
  - Entity provides proof of identity to CA.
  - CA creates certificate binding entity to public key.
  - Certificate digitally signed by CA.



- When Alice wants Bob's public key:
- gets Bob's certificate (Bob or elsewhere).
- Apply CA's public key to Bob's certificate, get Bob's public key

# Certification Authorities

- Take a look at the CA in the browser..

Your Certificates | Other People's | Web Sites | Authorities

You have certificates on file that identify these certificate authorities:

Certificate Name	Security Device
GP Root 2	Built-in Object Token
Visa eCommerce Root	Built-in Object Token
VeriCert, Inc.	
http://www.valicert.com/	Built-in Object Token
http://www.valicert.com/	Built-in Object Token
http://www.valicert.com/	Built-in Object Token
VeriSign, Inc.	
Verisign Class 1 Public Primary Certification Authority	Built-in Object Token
Verisign Class 2 Public Primary Certification Authority	Built-in Object Token
Verisign Class 3 Public Primary Certification Authority	Built-in Object Token
Verisign Class 1 Public Primary Certification Authority - G2	Built-in Object Token
Verisign Class 2 Public Primary Certification Authority - G2	Built-in Object Token
Verisign Class 3 Public Primary Certification Authority - G2	Built-in Object Token
Verisign Class 4 Public Primary Certification Authority - G2	Built-in Object Token
VeriSign Class 1 Public Primary Certification Authority - G3	Built-in Object Token
VeriSign Class 2 Public Primary Certification Authority - G3	Built-in Object Token
VeriSign Class 3 Public Primary Certification Authority - G3	Built-in Object Token
VeriSign Class 4 Public Primary Certification Authority - G3	Built-in Object Token
Class 1 Public Primary OCSP Responder	Built-in Object Token
Class 2 Public Primary OCSP Responder	Built-in Object Token
Class 3 Public Primary OCSP Responder	Built-in Object Token
VeriSign Time Stamping Authority CA	Built-in Object Token

View Edit Import Delete

Help

General Details

**Certificate Hierarchy**  
VeriSign Class 4 Public Primary Certification Authority - G3

**Certificate Fields**

- Issuer
- Validity
  - Not Before
  - Not After
- Subject
- Subject Public Key Info
  - Subject Public Key Algorithm
  - Subject's Public Key
- Certificate Signature Algorithm
- Certificate Signature Value

**Field Value**

```

8f fa 25 0b 4f 5b e4 a4 4e 27 55 ab 22 15 59 3c
ca b5 0a d4 4a db ab dd a1 5f 53 c5 a0 57 39 c2
ce 47 2b be 3a c8 56 bf c2 d9 27 10 3a b1 05 3c
c0 77 31 bb 3a d3 05 7b 6d 9a 1c 30 8c 80 cb 93
93 2a 83 ab 05 51 82 02 00 11 67 6b f3 88 61 47
5f 03 93 a5 5b 0d e0 f1 d4 a1 32 35 85 b2 3a db
b0 82 ab d1 cb 0a bc 4f 8c 5b c5 4b 00 3b 1f 2a
82 a6 7e 36 85 dc 7e 3c 67 00 b5 e4 3b 52 e0 a8
eb 5d 15 f9 c6 6d f0 ad 1d 0e 85 b7 a9 9a 73 14
  
```

General Details

**This certificate has been verified as follows:**

- SSL Server Certificate
- Email Signer Certificate
- Email Recipient Certificate
- Status Responder Certificate

**Issued To**

Common Name (CN)	VeriSign Class 4 Public Primary Certification Authority - G3
Organization (O)	VeriSign, Inc.
Organizational Unit (OU)	VeriSign Trust Network
Serial Number	00:EC:A0:A7:8B:6E:75:6A:01:CF:C4:7C:CC:2F:94:5E:D7

**Issued By**

Common Name (CN)	VeriSign Class 4 Public Primary Certification Authority - G3
Organization (O)	VeriSign, Inc.
Organizational Unit (OU)	VeriSign Trust Network

**Validity**

Issued On	01/10/99
Expires On	17/07/36

**Fingerprints**

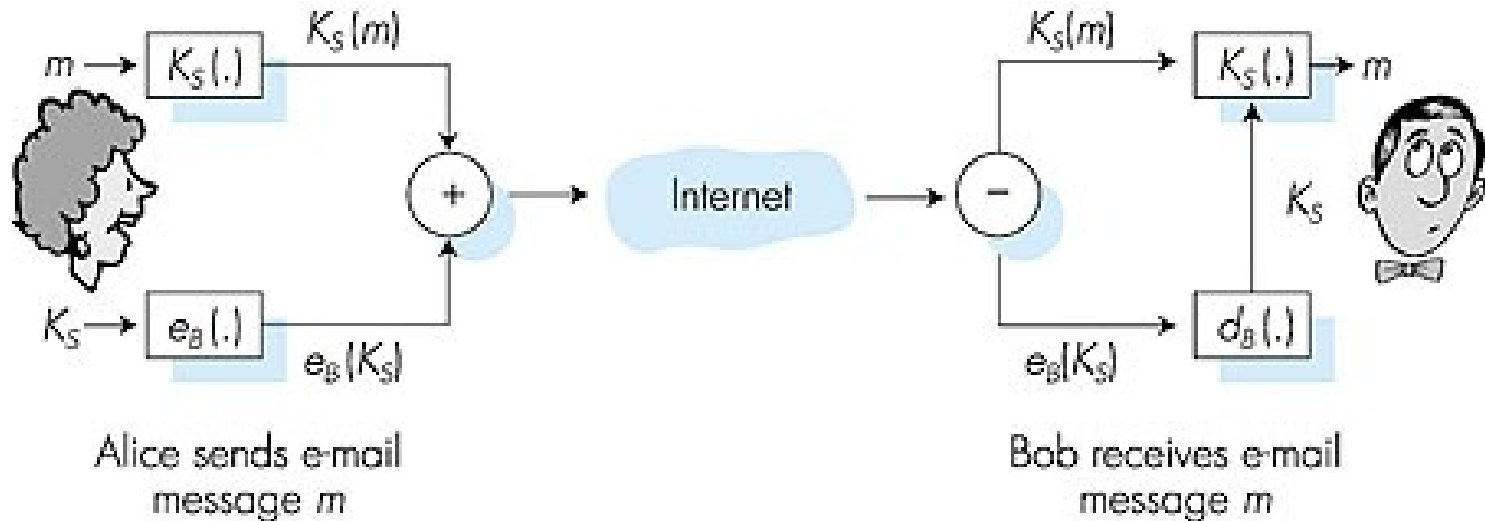
SHA1 Fingerprint	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
MD5 Fingerprint	DB:C8:F2:27:2E:B1:EA:6A:29:23:5D:FE:56:3E:33:DF

# Secure email:

Putting it all together

# Secure e-mail

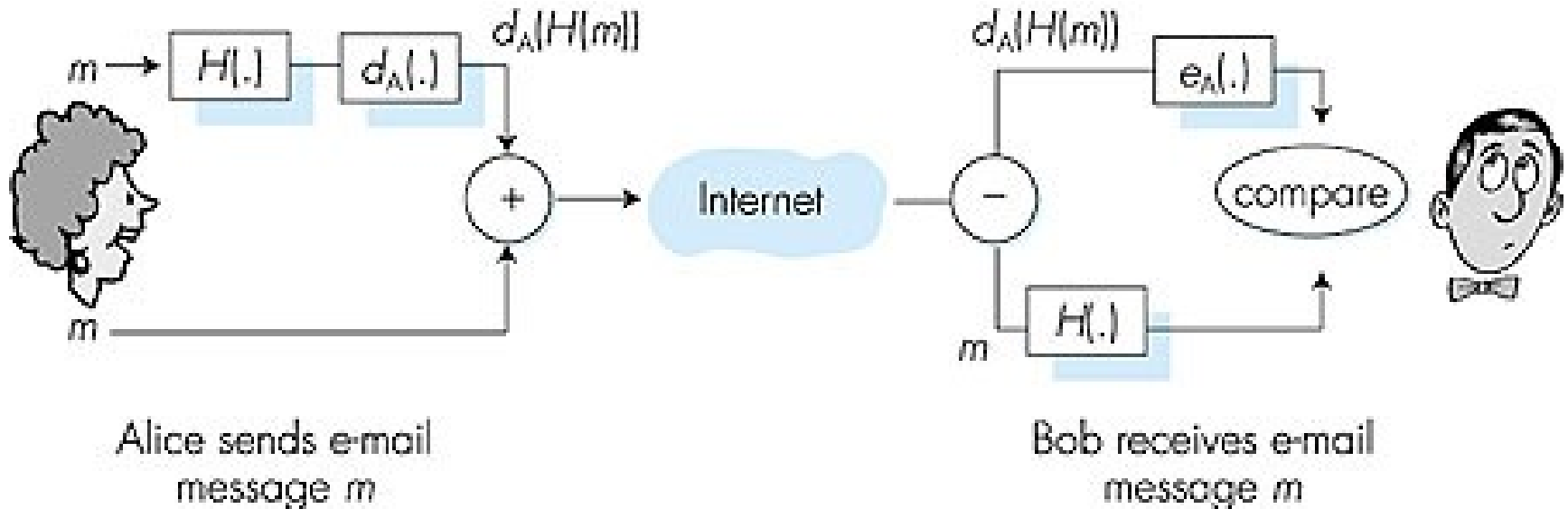
- Alice wants to send secret e-mail message,  $m$ , to Bob.



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

# Secure e-mail (continued)

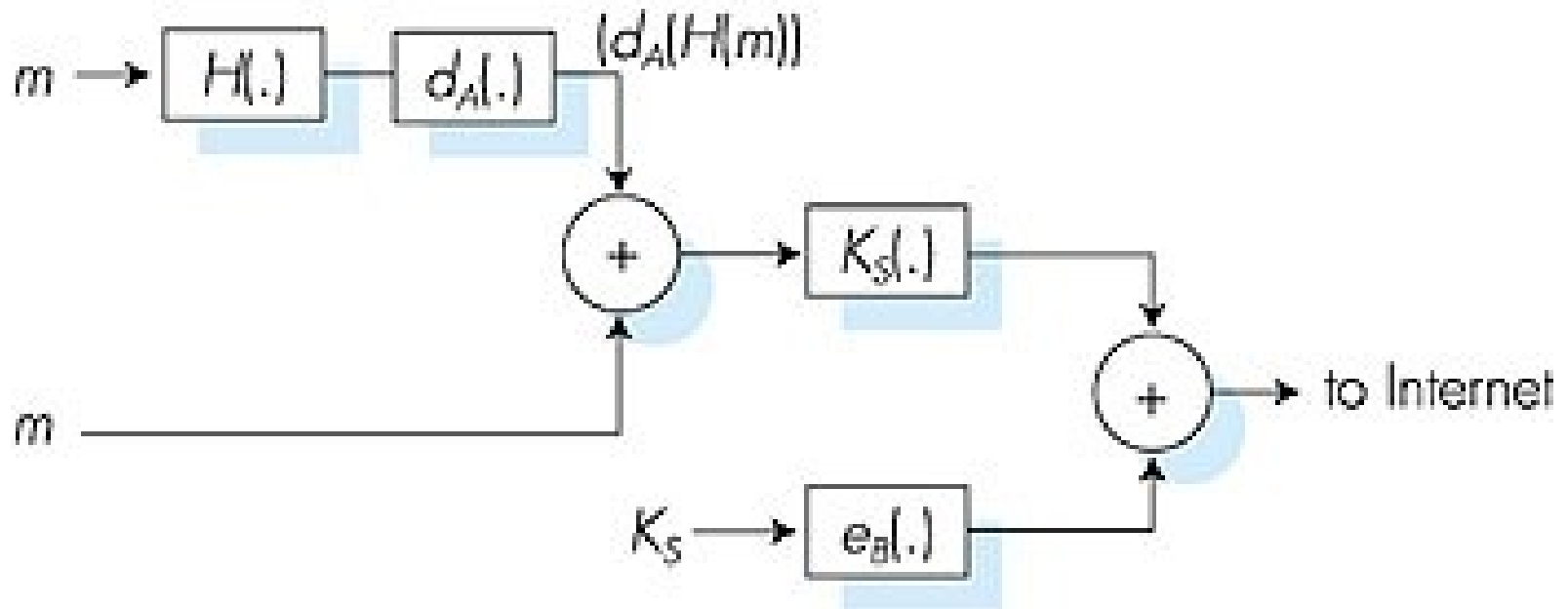
- Alice wants to provide sender authentication  
message integrity.



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

## Secure e-mail (continued)

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



Note: Alice uses both her private key, Bob's public key.

# Pretty good privacy (PGP)

- Internet e-mail encryption scheme, a 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, was target of 3-year federal investigation.

## A PGP signed message:

```
---BEGIN PGP SIGNED MESSAGE---  
Hash: SHA1  
  
Bob:(secret message)  
  
---BEGIN PGP SIGNATURE---  
Version: PGP 5.0  
Charset: noconv  
yhHJRHhGJGhgg/12EpJ+1o8gE4vB3mqJh  
FEvZP9t6n7G6m5Gw2  
---END PGP SIGNATURE---
```

# Pretty good privacy (PGP)

- Freely available on <http://web.mit.edu/network/pgp.html>
- Look also [www.pgp.com](http://www.pgp.com)
- Zimmermann has received technical awards
  - 2001: he was inducted into the CRN Industry Hall of Fame
  - 2000: InfoWorld named him one of the Top 10 Innovators in E-Business
  - 1999: Louis Brandeis Award from Privacy International
  - 1998: Lifetime Achievement Award from Secure Computing Magazine
  - 1996: the Norbert Wiener Award from Computer Professionals for promoting the responsible use of technology.

# Secure Protocols

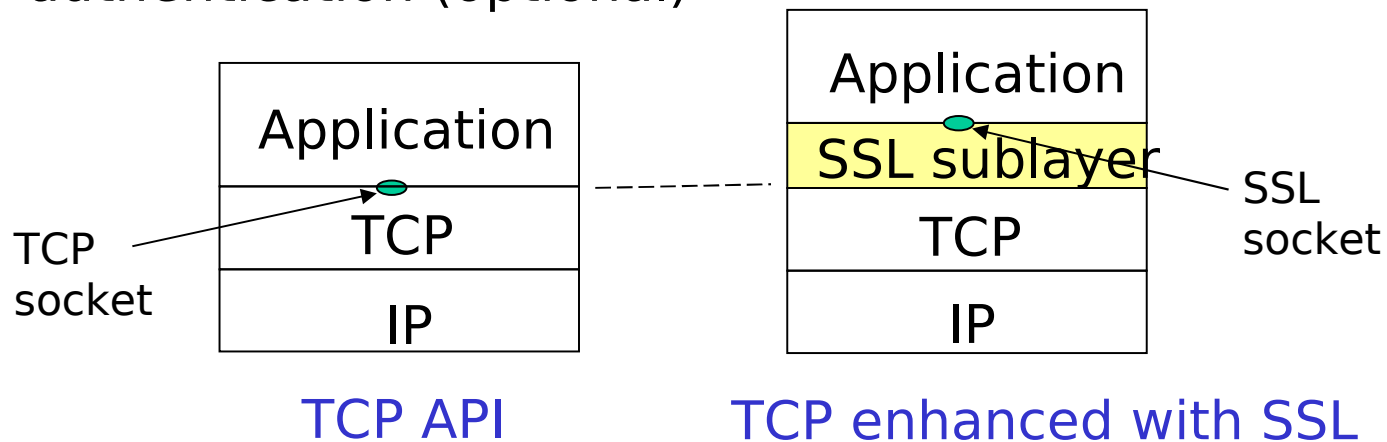
Protocols that do not use encryption are inherently insecure!

# Secure sockets layer (SSL)

- PGP provides security for a specific network app.
- SSL works at transport layer. Provides security to any TCP-based app using SSL services.
- SSL: used between WWW browsers, servers for I-commerce (shttp).
- SSL security services:
  - ▣ server authentication
    - data encryption
    - client authentication (optional)
- Server authentication:
  - SSL-enabled browser includes public keys for trusted CAs.
  - Browser requests server certificate, issued by trusted CA.
  - Browser uses CA's public key to extract server's public key from certificate.
- Visit your browser's security menu to see its trusted CAs.

# Secure sockets layer (SSL)

- provides transport layer security to any TCP-based application using SSL services.
  - e.g., between Web browsers, servers for e-commerce (shttp)
- security services:
  - server authentication, data encryption, client authentication (optional)



## SSL (continued)

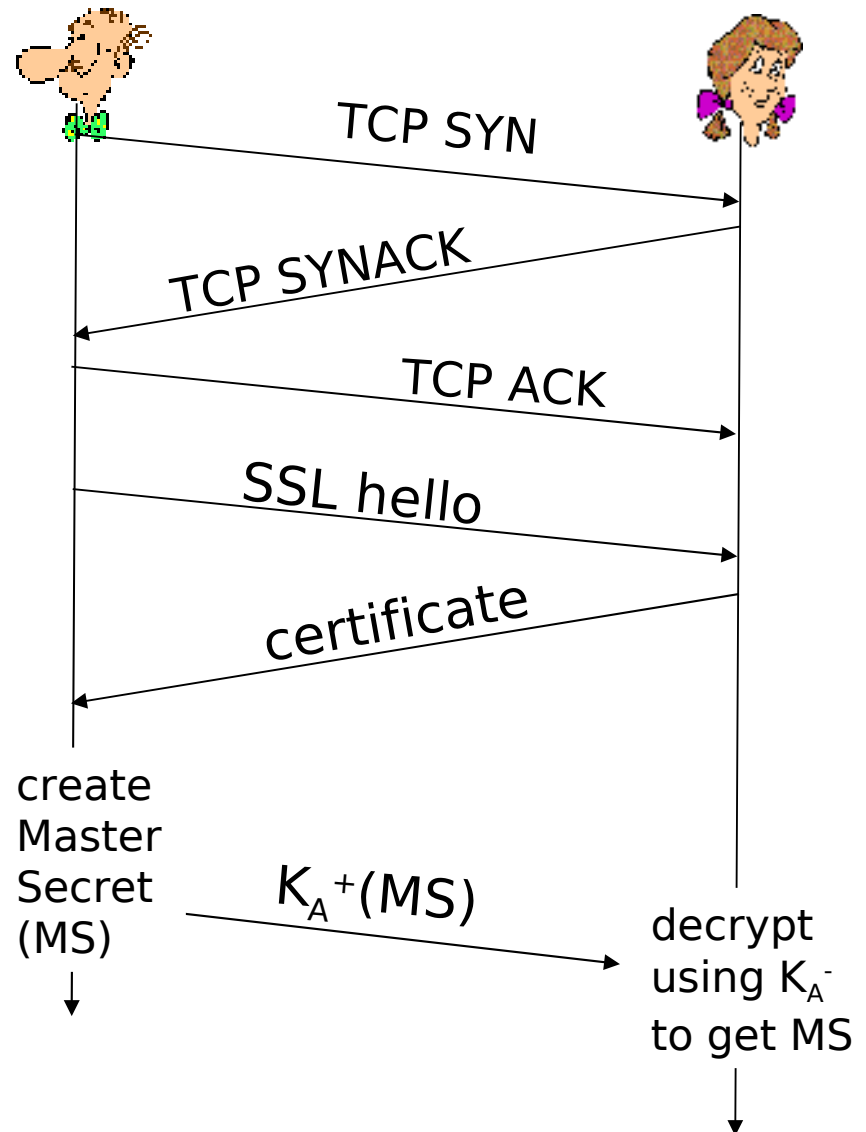
### Encrypted SSL session:

- Browser generates symmetric session key, encrypts it with server's public key, sends encrypted key to server.
  - Using its private key, server decrypts session key.
  - Browser, server agree that future msgs will be encrypted.
  - All data sent into TCP socket (by client or server) encrypted with session key.
- SSL: basis of IETF Transport Layer Security (TLS).
  - SSL can be used for non-Web applications, e.g., IMAP.
  - Client authentication can be done with client certificates.

# SSL: three phases

## 1. Handshake:

- Bob establishes TCP connection to Alice
- authenticates Alice via CA signed certificate
- creates, encrypts (using Alice's public key), sends master secret key to Alice
  - nonce exchange not shown



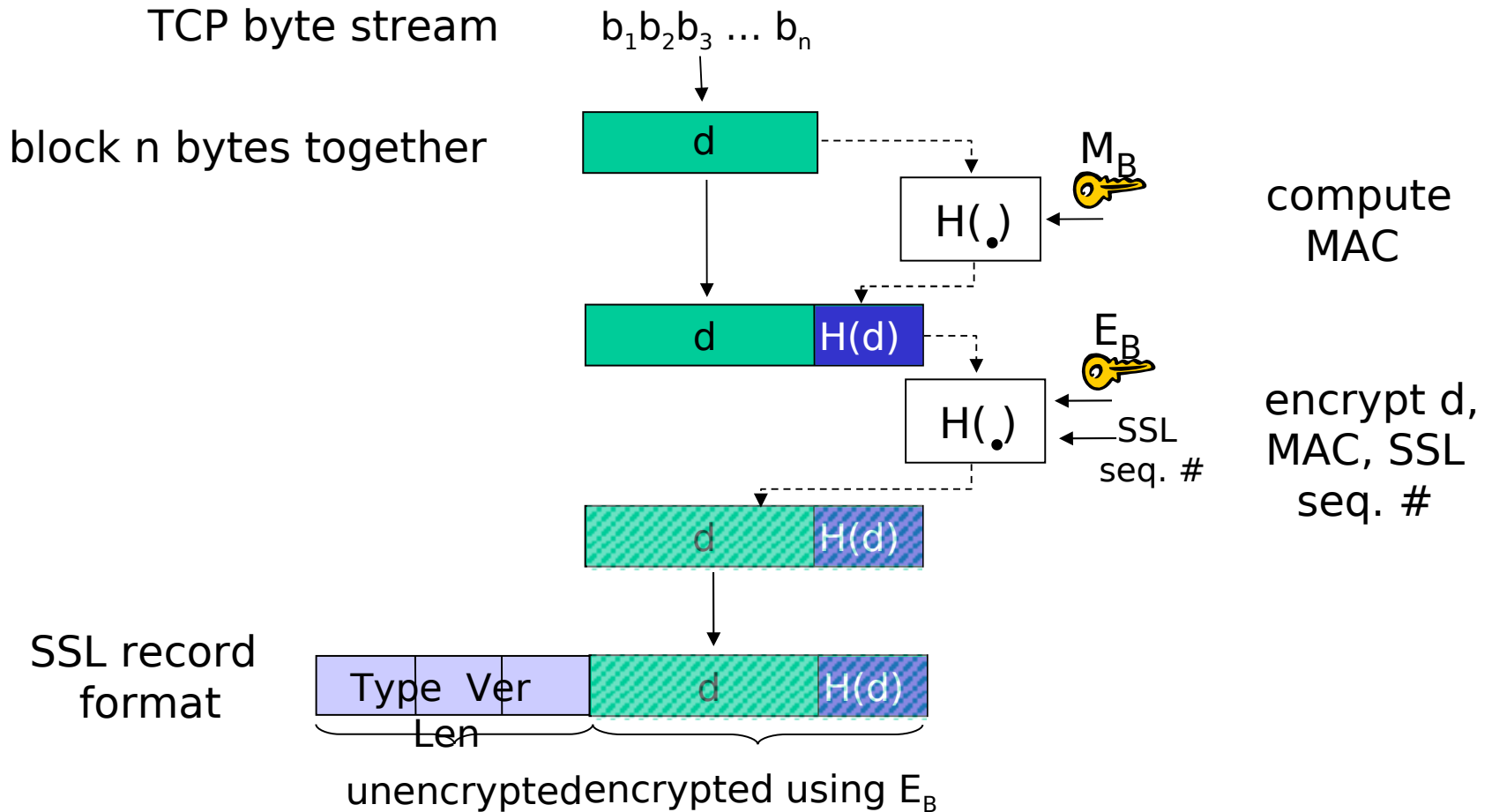
# SSL: three phases

## *2. Key Derivation:*

- Alice, Bob use shared secret (MS) to generate 4 keys:
  - $E_B$ : Bob->Alice data encryption key
  - $E_A$ : Alice->Bob data encryption key
  - $M_B$ : Bob->Alice MAC (Message Authentication Code) key
  - $M_A$ : Alice->Bob MAC key
- encryption and MAC algorithms negotiable between Bob, Alice
- why 4 keys?

# SSL: three phases

## 3. Data transfer



# Secure electronic transactions (SET)

- designed for payment-card transactions over Internet.
- provides security services among 3 players:
  - customer
  - merchant
  - Merchant's bankAll must have certificates.
- SET specifies legal meanings of certificates.
  - apportionment of liabilities for transactions
- Customer's card number passed to merchant's bank without merchant ever seeing number in plain text.
  - Prevents merchants from stealing, leaking payment card numbers.
- Three software components:
  - Browser wallet
  - Merchant server
  - Acquisition gateway
- See text for description of SET transaction.

# SSH: an example of secure connection

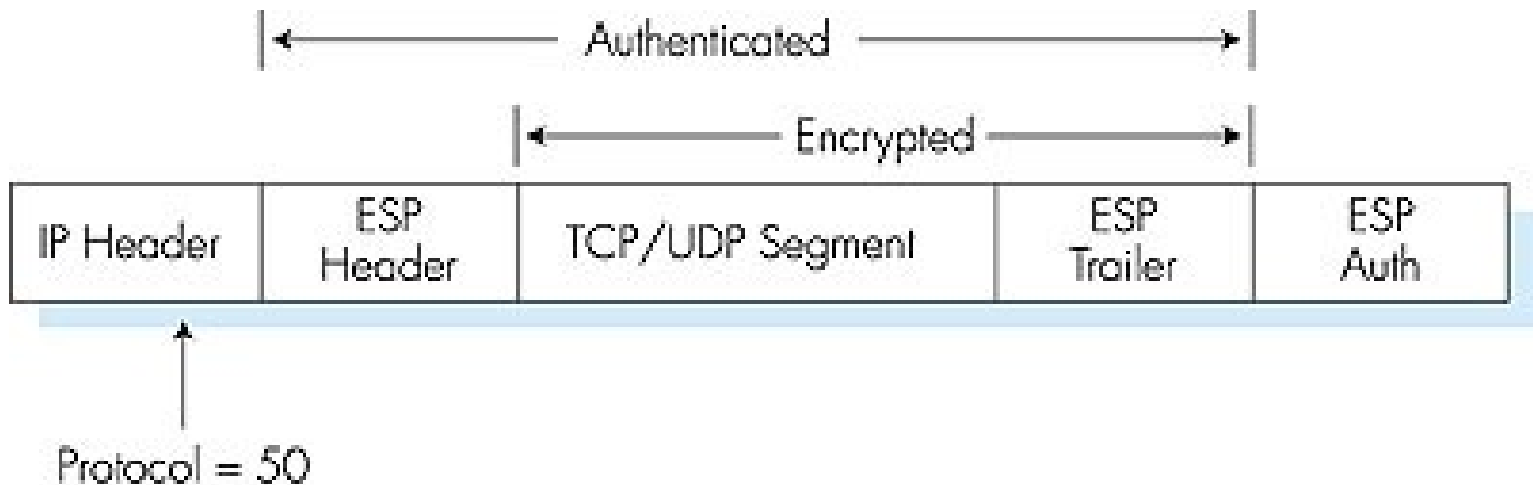
- Telnet or rsh are not secure
- They transmit login/passwords over the network
- SSH is safer because it encrypts the login/password
- Authenticates the hosts
- Keeps keys on the local user's directory
- Example of know\_hosts file:
  - `hostname1,130.113.118.147 ssh-rsa  
AAAAB3NzaC1yc2EAAAABIwAAAIEA smnfyxDMN7o1UrXuvj  
chDDFGRVdwRLVC+/pVoXvrVl5Byxp/GQSdWJeYzMyEyKaN  
Q+IgFpiBGqnsqfk8uQJCzyJnB3nkYSAhVlz2emjuC6kuJ8  
yFgoIx0N4v9NVEeSgSEIua6aVBi4a4t fy2sSj15aYzWPS0  
mJoG+hnt6lEaDY0`

# Ipsec: Network Layer Security

- **Network-layer secrecy:**
  - sending host encrypts the data in IP datagram
  - TCP and UDP segments; ICMP and SNMP messages.
- **Network-layer authentication**
  - destination host can authenticate source IP address
- **Two principle protocols:**
  - authentication header (AH) protocol
  - encapsulation security payload (ESP) protocol
- **For both AH and ESP, source, destination handshake:**
  - create network-layer logical channel called a service agreement (SA)
- **Each SA unidirectional.**
- **Uniquely determined by:**
  - security protocol (AH or ESP)
  - source IP address
  - 32-bit connection ID

# ESP Protocol

- Provides secrecy, host authentication, data integrity.
- Data, ESP trailer encrypted.
- Next header field is in ESP trailer.
- ESP authentication field is similar to AH authentication field.
- Protocol = 50.

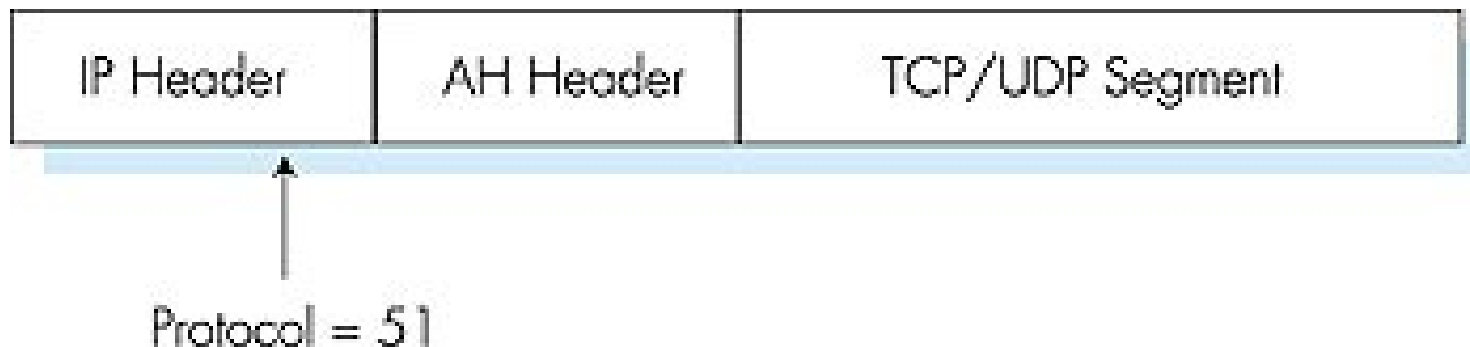


# Authentication Header (AH) Protocol

- Provides source host authentication, data integrity, but not secrecy.
- AH header inserted between IP header and IP data field.
- Protocol field = 51.
- Intermediate routers process datagrams as usual.

## AH header includes:

- connection identifier
- authentication data: signed message digest, calculated over original IP datagram, providing source authentication, data integrity.
- Next header field: specifies type of data (TCP, UDP, ICMP, etc.)



# Wireless Security

Wireless networks without encryption??

# IEEE 802.11 security

- *war-driving*: drive around Bay area, see what 802.11 networks available?
  - ▣ More than 9000 accessible from public roadways
    - 85% use no encryption/authentication
    - packet-sniffing and various attacks easy!
- *securing 802.11*
  - encryption, authentication
  - first attempt at 802.11 security: Wired Equivalent Privacy (WEP): a failure
  - current attempt: 802.11i

## Wired Equivalent Privacy (WEP):

- authentication as in protocol *ap4.0*
  - host requests authentication from access point
  - access point sends 128 bit nonce
  - host encrypts nonce using shared symmetric key
  - access point decrypts nonce, authenticates host
- no key distribution mechanism
- authentication: knowing the shared key is enough

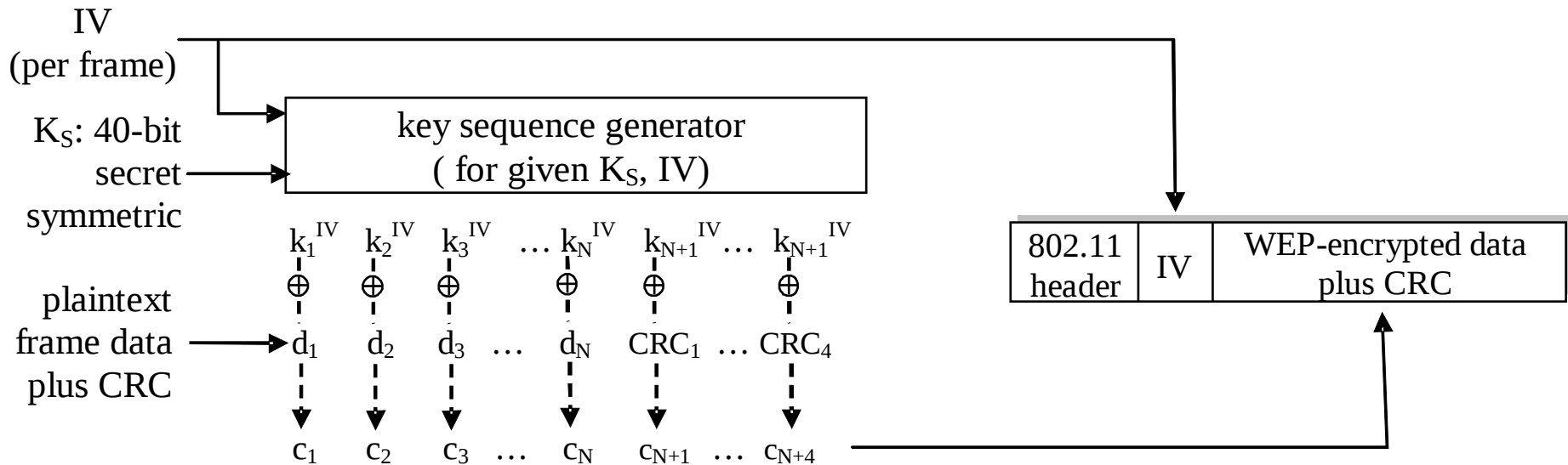
# WEP data encryption

- host/AP share 40 bit symmetric key (semi-permanent)
- host appends 24-bit initialization vector (IV) to create 64-bit key
- 64 bit key used to generate stream of keys,  $k_i^{\text{IV}}$
- $k_i^{\text{IV}}$  used to encrypt  $i$ th byte,  $d_i$ , in frame:

$$c_i = d_i \text{ XOR } k_i^{\text{IV}}$$

- IV and encrypted bytes,  $c_i$  sent in frame

# 802.11 WEP encryption



Sender-side WEP encryption

# Breaking 802.11 WEP encryption

## security hole:

- 24-bit IV, one IV per frame, -> IV's eventually reused
- IV transmitted in plaintext -> IV reuse detected

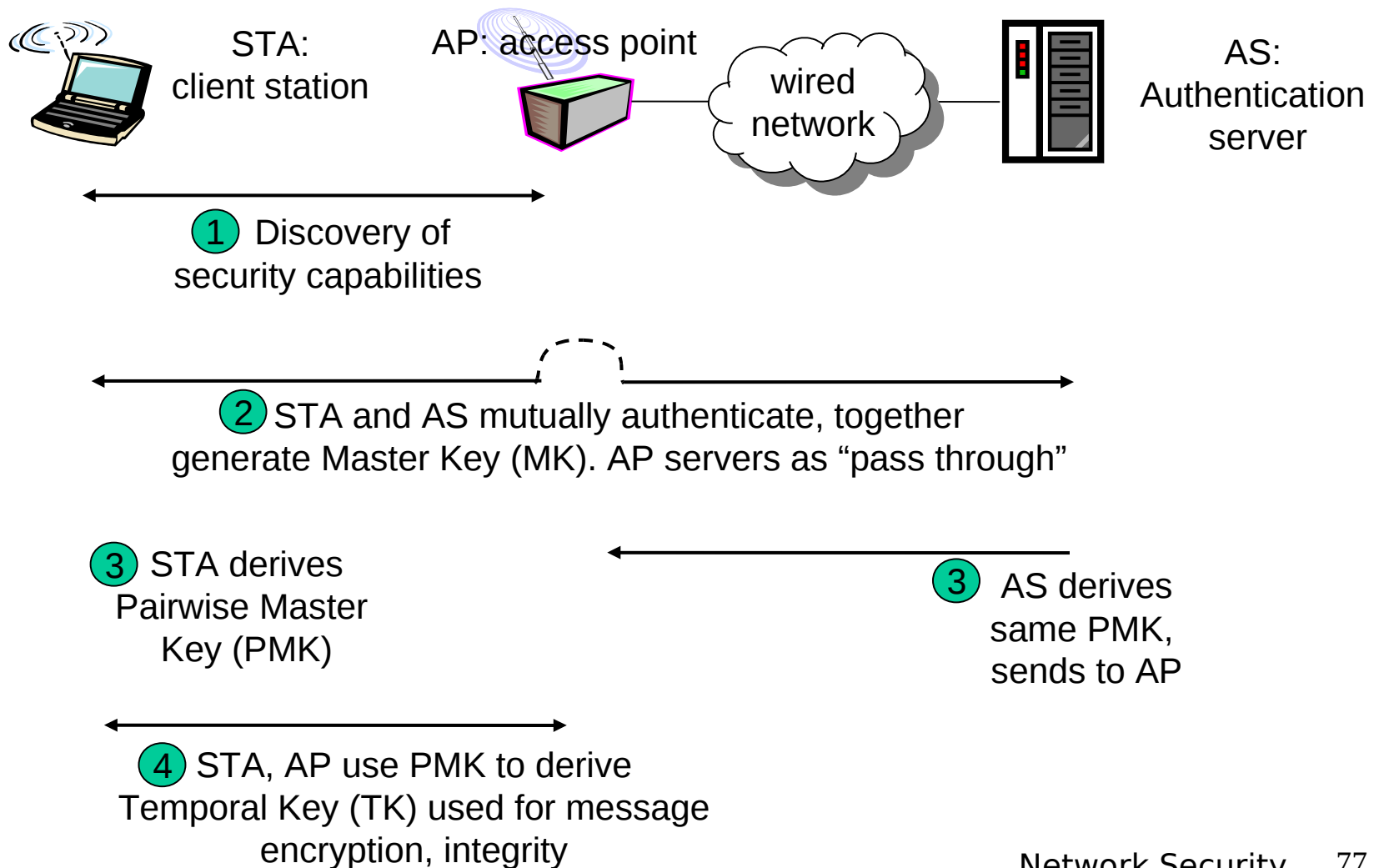
## □ attack:

- Trudy causes Alice to encrypt known plaintext  $d_1$   
 $d_2$   $d_3$   $d_4$  ...
- Trudy sees:  $c_i = d_i \text{ XOR } k_i^{\text{IV}}$
- Trudy knows  $c_i$   $d_i$ , so can compute  $k_i^{\text{IV}}$
- Trudy knows encrypting key sequence  $k_1^{\text{IV}}$   $k_2^{\text{IV}}$   $k_3^{\text{IV}}$   
...
- Next time IV is used, Trudy can decrypt!

# 802.11i: improved security

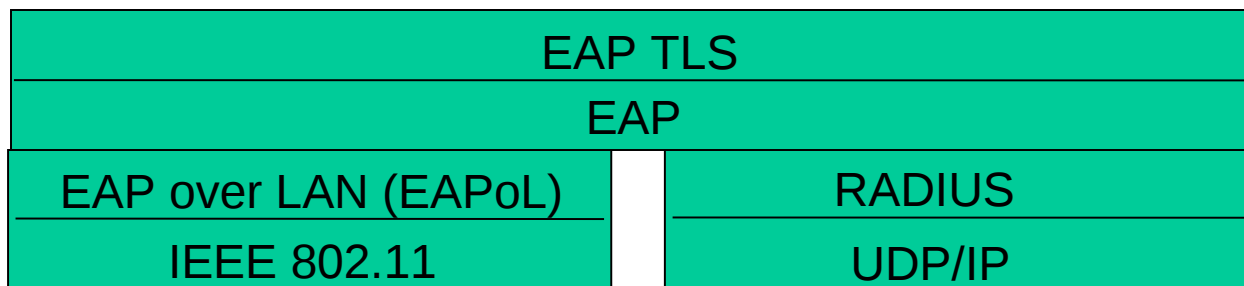
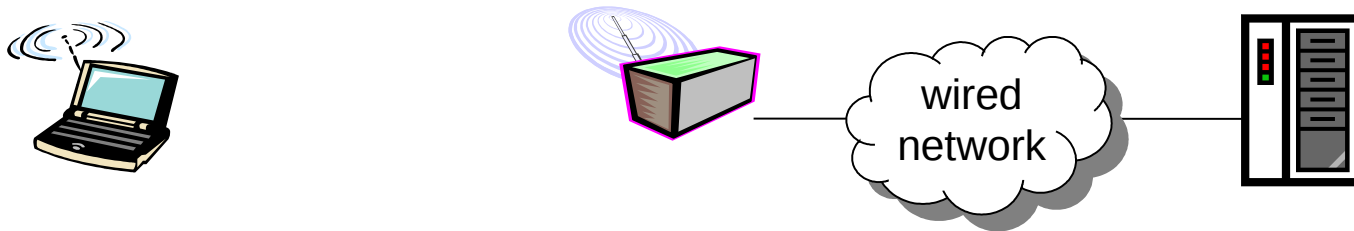
- numerous (stronger) forms of encryption possible
- provides key distribution
- uses authentication server separate from access point

# 802.11i: four phases of operation



# EAP: extensible authentication protocol

- EAP: end-end client (mobile) to authentication server protocol
- EAP sent over separate “links”
  - mobile-to-AP (EAP over LAN)
  - AP to authentication server (RADIUS over UDP)



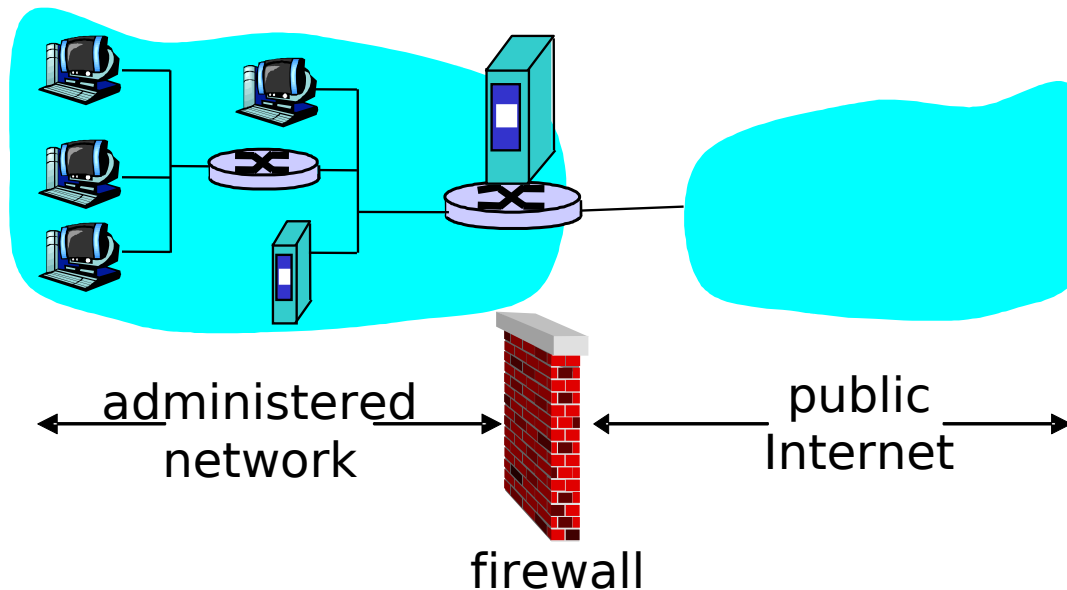
# Firewalls

... and the doors in the back of the networks...

# Firewalls

## firewall

isolates organization's internal net from larger Internet, allowing some packets to pass, blocking others.



# Firewalls: Why

## prevent denial of service attacks:

- SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections

## prevent illegal modification/access of internal data.

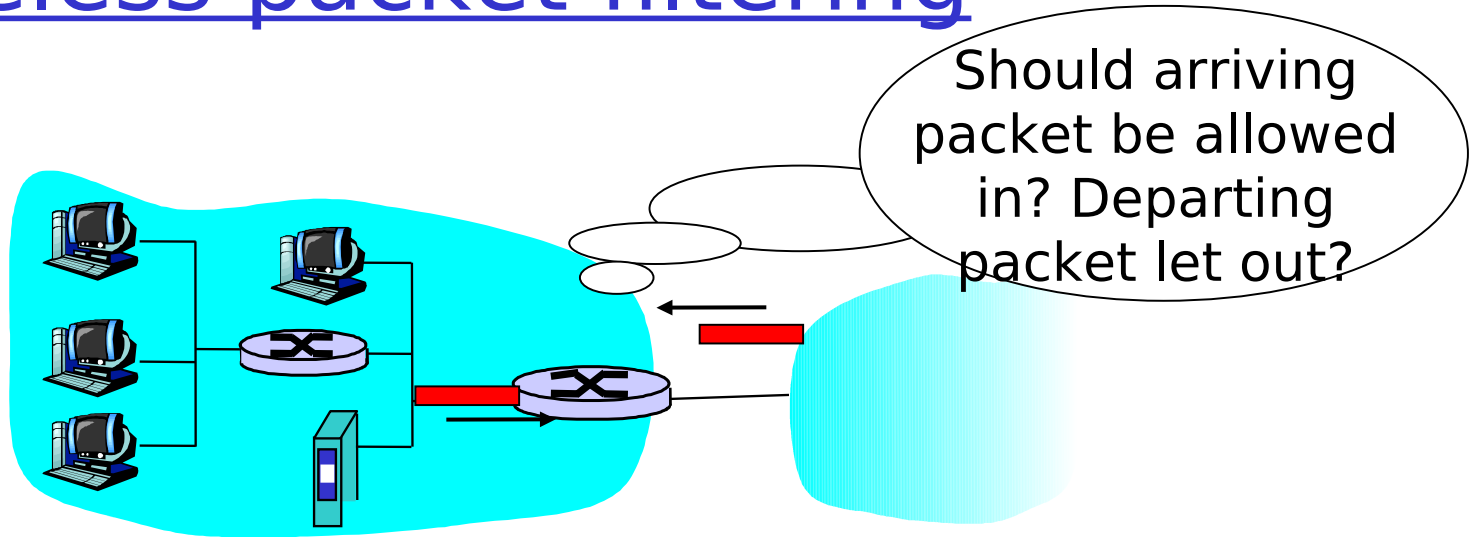
- e.g., attacker replaces CIA’s homepage with something else

## allow only authorized access to inside network (set of authenticated users/hosts)

## three types of firewalls:

- stateless packet filters
- stateful packet filters
- application gateways

# Stateless packet filtering



- internal network connected to Internet via **router firewall**
- router **filters packet-by-packet**, decision to forward/drop packet based on:
  - ▣ source IP address, destination IP address
  - ▣ TCP/UDP source and destination port numbers
    - ICMP message type
    - TCP SYN and ACK bits

# Stateless packet filtering: example

- example 1: block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23.
  - all incoming, outgoing UDP flows and telnet connections are blocked.
- example 2: Block inbound TCP segments with ACK=0.
  - prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside.

# Stateless packet filtering: more examples

<u>Policy</u>	<u>Firewall Setting</u>
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for institution's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets - except DNS and router broadcasts.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP packets going to a "broadcast" address (eg 130.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

# Access Control Lists

- **ACL:** table of rules, applied top to bottom to incoming packets: (action, condition) pairs

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16 outside of 222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	222.22/16 of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16 outside of 222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	222.22/16 of 222.22/16	222.22/16	UDP	53	> 1023	----
deny	all	all	all	all	all	all

# Stateful packet filtering

- stateless packet filter: heavy handed tool
  - admits packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established:

action	source address	dest address	protocol	source port	dest port	flag bit
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK

- *stateful packet filter*: track status of every TCP connection
  - track connection setup (SYN), teardown (FIN): can determine whether incoming, outgoing packets “makes sense”
  - timeout inactive connections at firewall: no longer admit packets

# Stateful packet filtering

- ACL augmented to indicate need to check connection state table before admitting packet

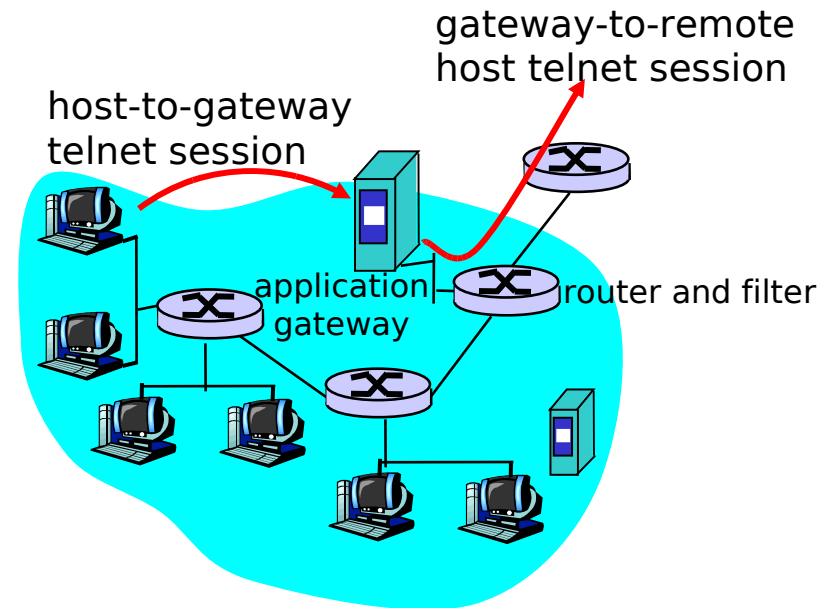
action	source address	dest address	proto	source port	dest port	flag bit	check conxion
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----	X
deny	all	all	all	all	all	all	

# Application gateways

- filters packets on application data as well as on IP/TCP/UDP fields.

- example: allow select internal users to telnet outside.

1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host. Gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway.



## Limitations of firewalls and gateways

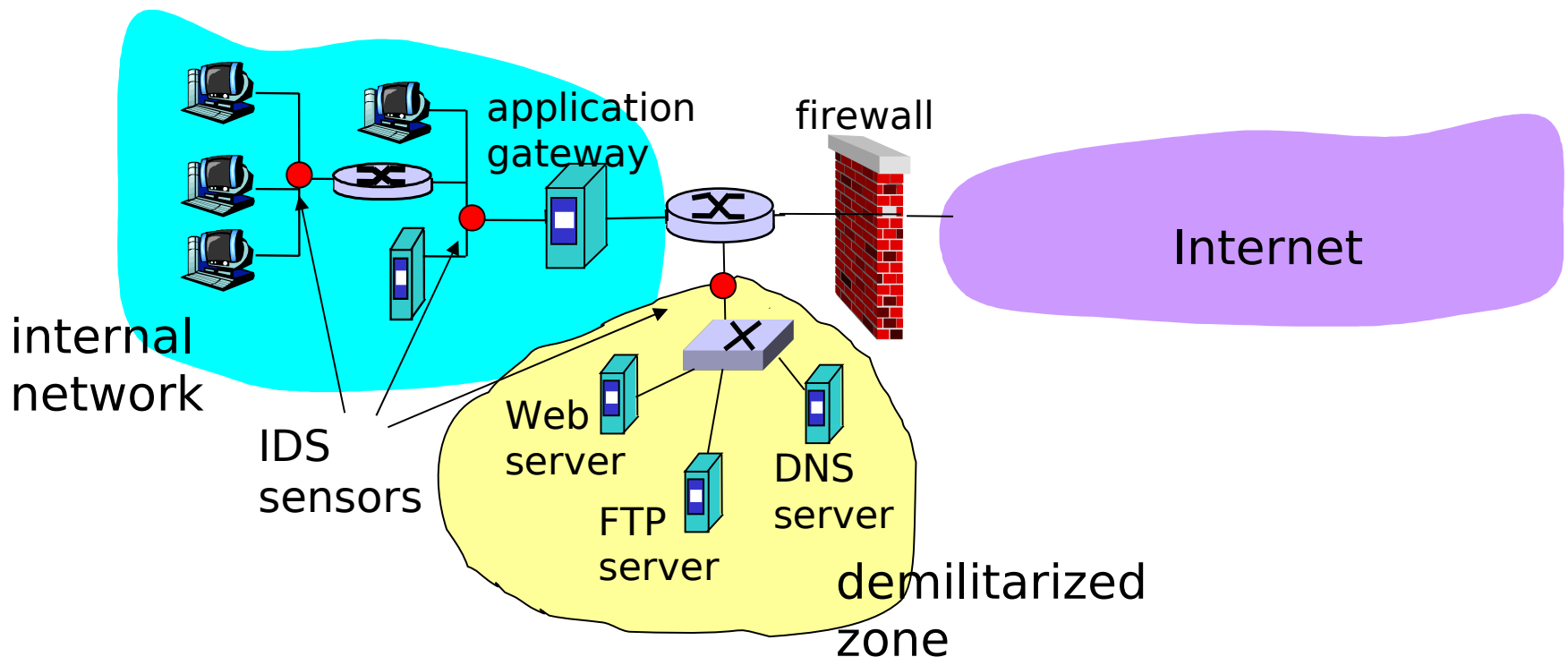
- IP spoofing: router can't know if data "really" comes from claimed source
- if multiple app's. need special treatment, each has own app. gateway.
- client software must know how to contact gateway.
  - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for UDP.
- tradeoff: **degree of communication with outside world, level of security**
- many highly protected sites still suffer from attacks.

# Intrusion detection systems

- packet filtering:
  - operates on TCP/IP headers only
  - no correlation check among sessions
- *IDS: intrusion detection system*
  - *deep packet inspection*: look at packet contents (e.g., check character strings in packet against database of known virus, attack strings)
  - *examine correlation* among multiple packets
    - port scanning
    - network mapping
    - DoS attack

# Intrusion detection systems

- multiple IDSs: different types of checking at different locations



# Network Security (summary)

## Basic techniques.....

- cryptography (symmetric and public)
- authentication
- message integrity

## .... used in many different security scenarios

- secure email
- secure transport (SSL)
- IP sec