Network Security

Goals:

- understand principles of network security:
  - cryptography and its *many* uses beyond "confidentiality"
  - authentication
  - message integrity
- security in practice:
  - firewalls and intrusion detection systems
  - security in application, transport, network, link layers

What is network security?

confidentiality: only sender, intended receiver should "understand" message contents

- sender encrypts message
- receiver decrypts message

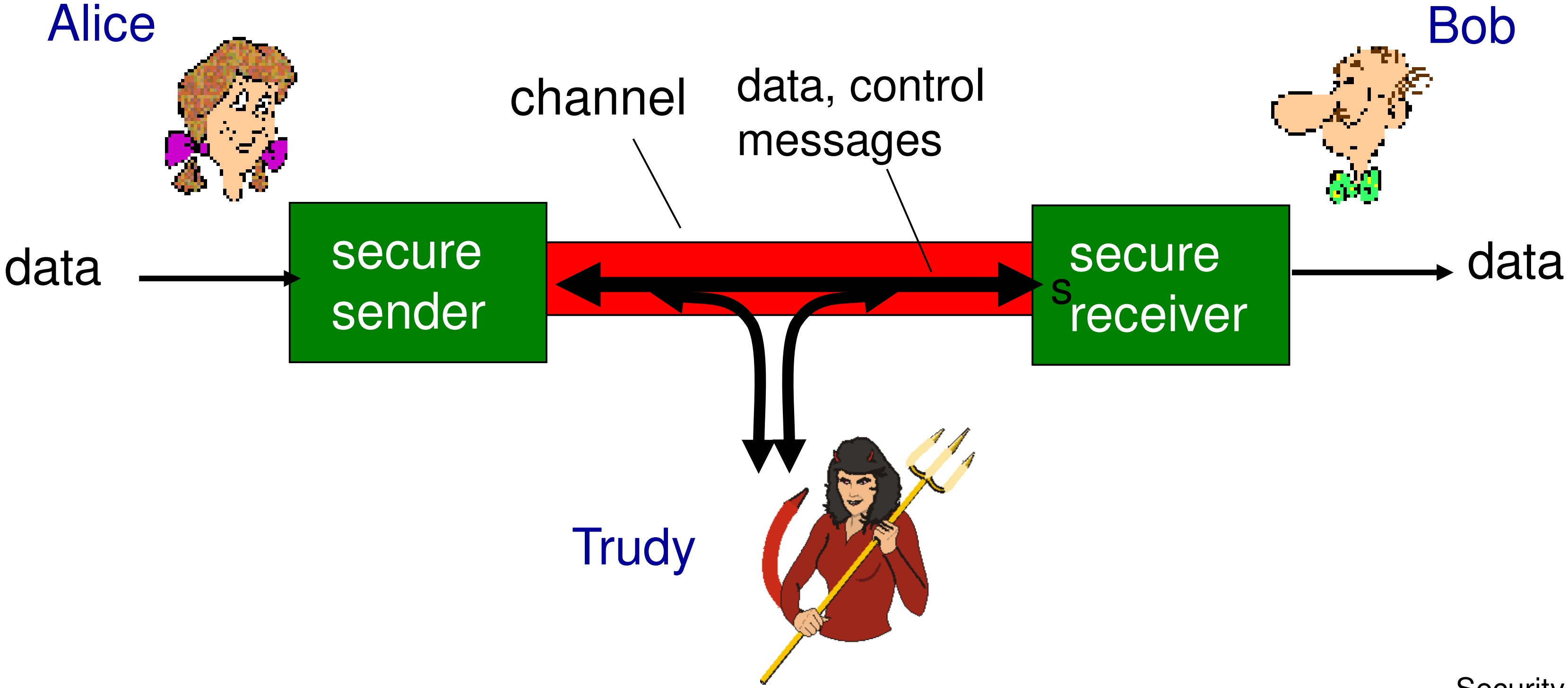
authentication: sender, receiver want to confirm identity of each other

message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

access and availability: services must be accessible and available to users

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages



Alice

Bob

channel

data, control messages

data

secure sender

s secure receiver

data

Trudy

Who might Bob, Alice be?

- … well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- other examples?

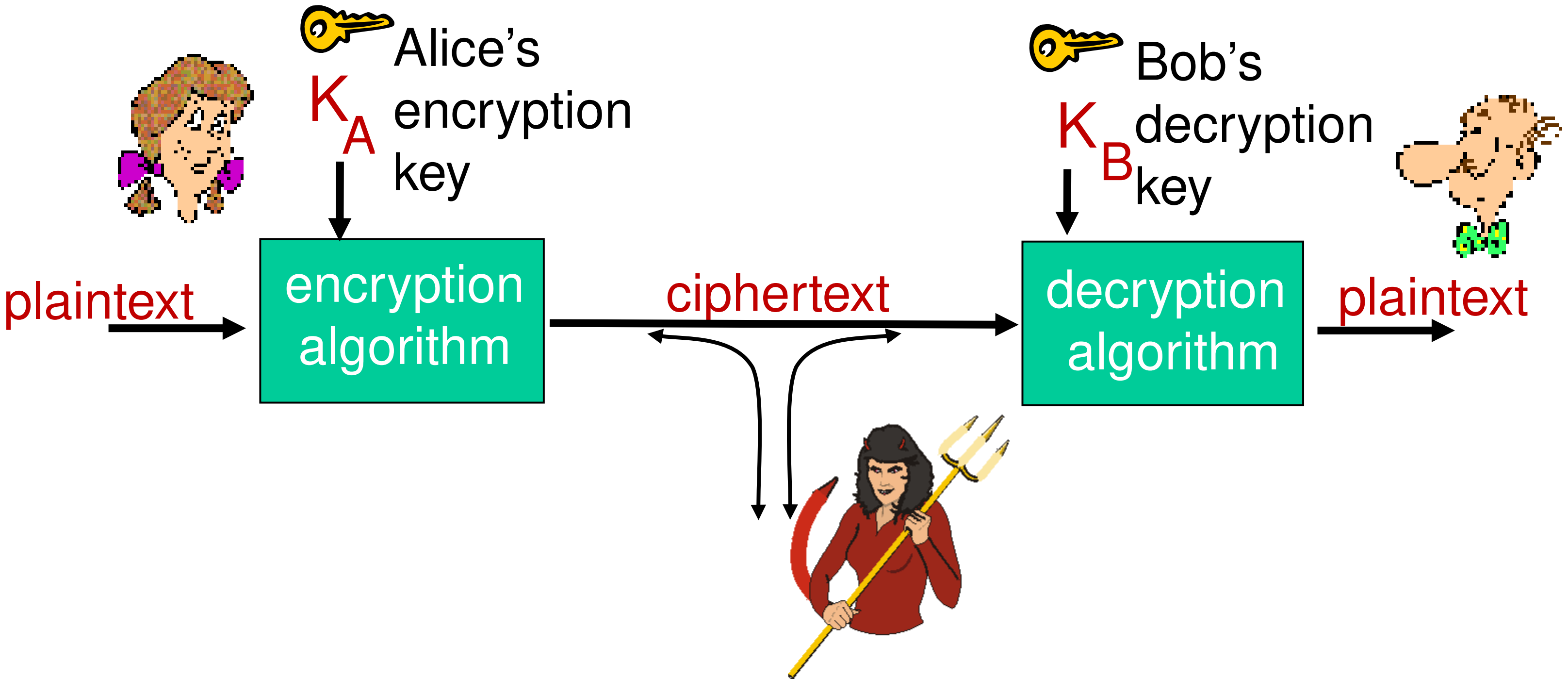
There are malicious agents out there!

Q: What can a "malicious agent" 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)

The language of cryptography



m plaintext message

$K_A(m)$ ciphertext, encrypted with key K_A

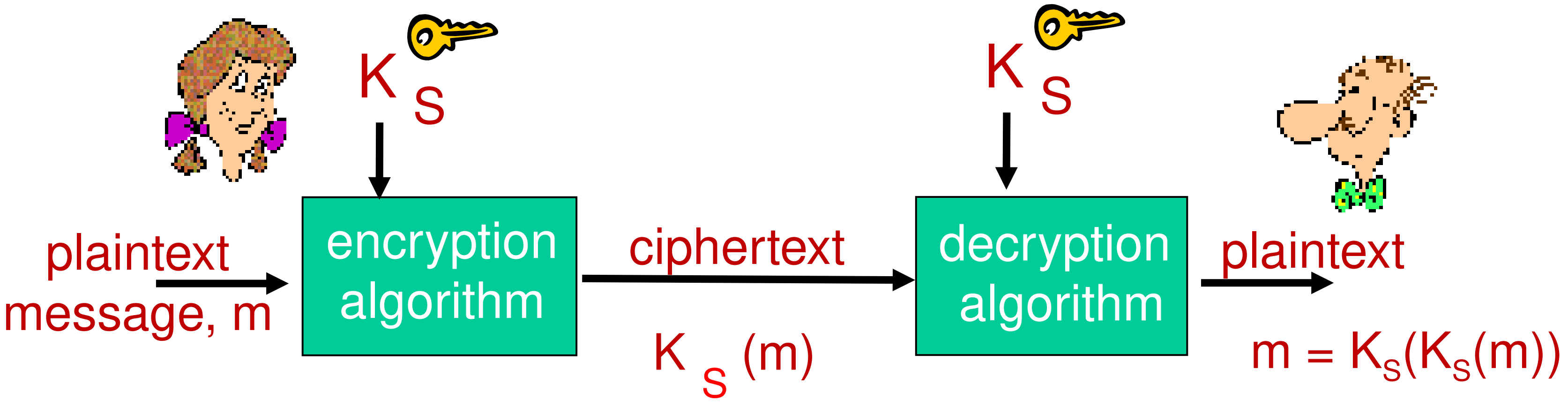
$m = K_B(K_A(m))$

Breaking an encryption scheme

- known-plaintext attack: Trudy has plaintext corresponding to ciphertext
  - e.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,
- chosen-plaintext attack: Trudy can get ciphertext for chosen plaintext

- cipher-text only attack: Trudy has ciphertext she can analyze
- two approaches:
  - brute force: search through all keys
  - statistical analysis

Symmetric key cryptography



symmetric key crypto: Bob and Alice share same (symmetric) key: K_S

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher

Q: how do Bob and Alice agree on key value?

Simple encryption scheme

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

```
 plaintext:   abcdefghijklmnopqrstuvwxyz

ciphertext:   mnbvcxzasdfghjklpoiuytrewq
```

e.g.:
```
 Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Encryption key: mapping from set of 26 letters to set of 26 letters

A more sophisticated encryption approach

- n substitution ciphers, $M_1, M_2, \ldots, M_n$
- cycling pattern:
  - e.g., n=4: M_1, M_3, M_4, M_3, M_2; M_1, M_3, M_4, M_3, M_2; ..
- for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
  - dog: d from M_1, o from M_3, g from M_4

Encryption key: n substitution ciphers, and cyclic pattern

- key need not be just n-bit pattern

Symmetric key crypto: DES

DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- block cipher with cipher block chaining
- how secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
  - no known good analytic attack
- making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

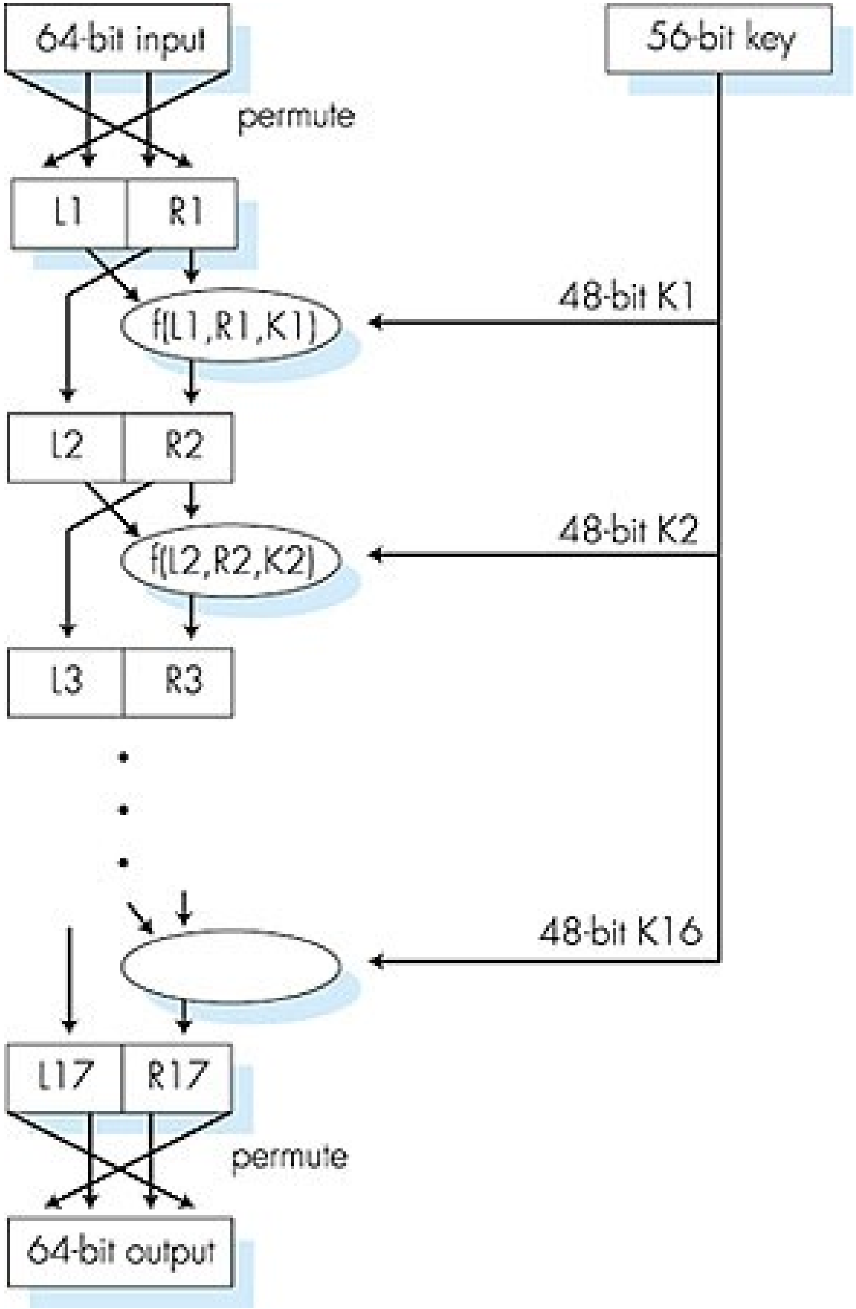
Symmetric key crypto: DES

DES operation

initial permutation

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

final permutation



AES: Advanced Encryption Standard

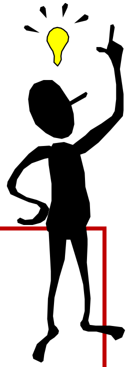
- symmetric-key NIST standard, replaced DES (Nov 2001)
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

Public Key Cryptography

symmetric key crypto

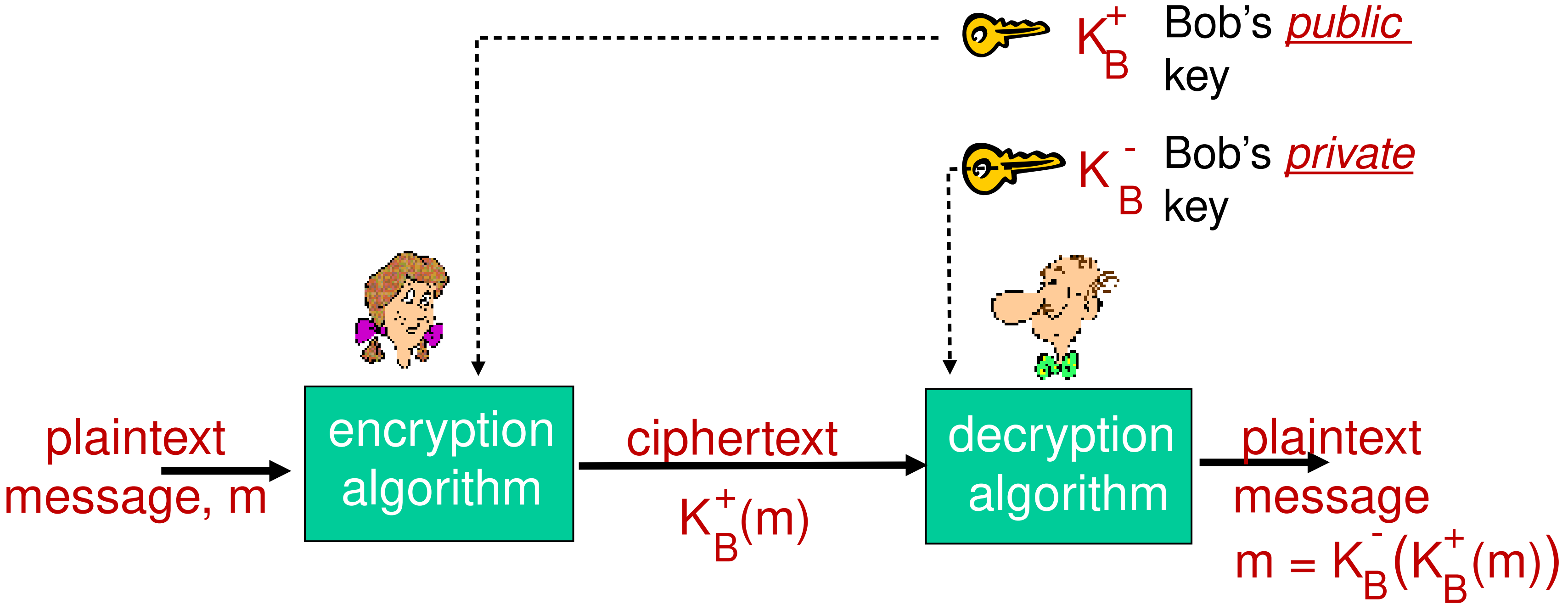
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

public key crypto



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

Public key cryptography



K_B^+ Bob's *public* key

K_B^- Bob's *private* key

plaintext message, m

encryption algorithm

ciphertext

$K_B^+(m)$

decryption algorithm

plaintext message

$m = K_B^-(K_B^+(m))$

Public key encryption algorithms

requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

Prerequisite: modular arithmetic

- x mod n = remainder of x when divide by n
- facts:

  [(a mod n) + (b mod n)] mod n = (a+b) mod n

  [(a mod n) - (b mod n)] mod n = (a-b) mod n

  [(a mod n) * (b mod n)] mod n = (a*b) mod n
- thus

  $(a \bmod n)^d \bmod n = a^d \bmod n$
- example: x=14, n=10, d=2:

  $(x \bmod n)^d \bmod n = 14^2 \bmod 10 = 6$

  $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$

RSA: getting ready

- message: just a bit pattern
- bit pattern can be uniquely represented by an integer number
- thus, encrypting a message is equivalent to encrypting a number

example:

- m= 10010001 . This message is uniquely represented by the decimal number 145.
- to encrypt m, we encrypt the corresponding number, which gives a new number (the ciphertext).

RSA: Creating public/private key pair

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 $\underbrace{(n,e)}_{K_B^+}$. *private* key is $\underbrace{(n,d)}_{K_B^-}$.

RSA: encryption, decryption

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

1. to encrypt message m $(<n)$, compute

$$c = m^e \bmod n$$

2. to decrypt received bit pattern, c, compute

$$m = c^d \bmod n$$

magic happens! $$m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$$

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).

encrypting 8-bit messages.

encrypt: $\underbrace{\text{bit pattern}}_{00001100}$ $\underbrace{m}_{12}$ $\underbrace{m^e}_{248832}$ $\underbrace{c = m^e \bmod n}_{17}$

decrypt: $\underbrace{c}_{17}$ $\underbrace{c^d}_{481968572106750915091411825223071697}$ $\underbrace{m = c^d \bmod n}_{12}$

Why does RSA work?

- must show that $c^d \bmod n = m$ where $c = m^e \bmod n$
- fact: for any x and y: $x^y \bmod n = x^{(y \bmod z)} \bmod n$
  - where $n= pq$ and $z = (p-1)(q-1)$
- thus,

$$
\begin{aligned}
c^d \bmod n &= (m^e \bmod n)^d \bmod n \\
&= m^{ed} \bmod n \\
&= m^{(ed \bmod z)} \bmod n \\
&= m^1 \bmod n \\
&= m
\end{aligned}
$$

RSA: another important property

The following property will be *very* useful later:

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

result is the same!

Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

follows directly from modular arithmetic:

$$
\begin{aligned}
(m^e \bmod n)^d \bmod n &= m^{ed} \bmod n \\
&= m^{de} \bmod n \\
&= (m^d \bmod n)^e \bmod n
\end{aligned}
$$

Why is RSA secure?

- suppose you know Bob's public key (n,e). How hard is it to determine d?
- essentially need to find factors of n without knowing the two factors p and q
    - fact: factoring a big number is hard

RSA in practice: session keys

- exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

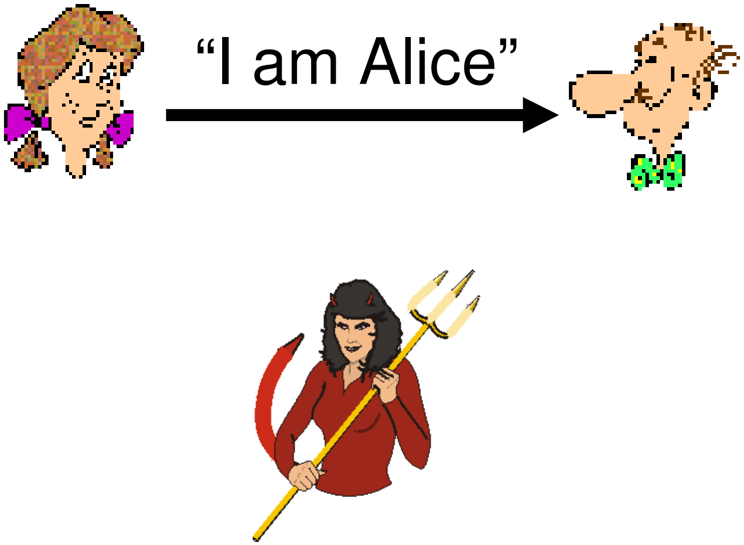
session key, K_S

- Bob and Alice use RSA to exchange a symmetric key K_S
- once both have K_S, they use symmetric key cryptography

Authentication

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

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

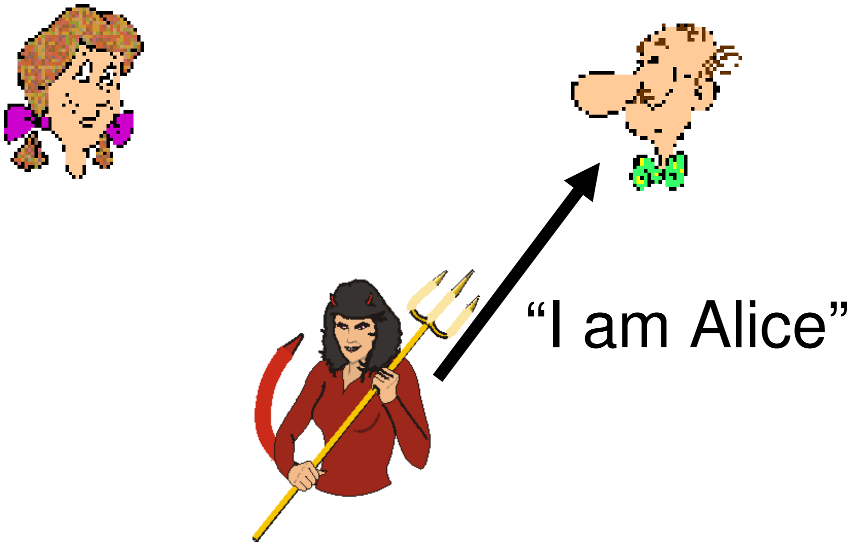


Failure scenario??

Authentication

Goal: Bob wants Alice to “prove” her identity to him

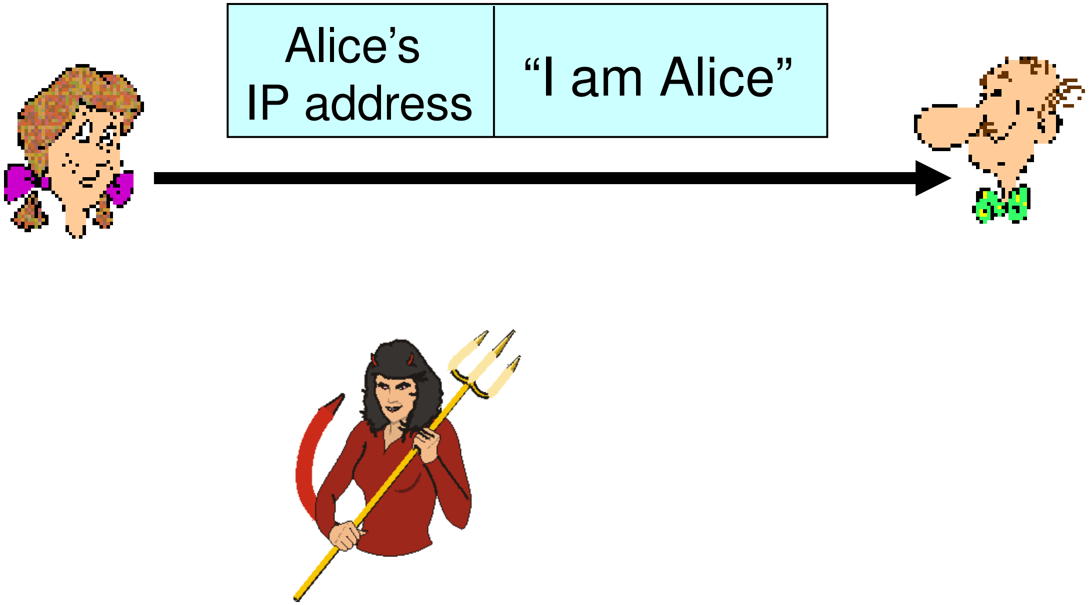
Protocol ap1.0: Alice says “I am Alice”



in a network,
Bob can not “see” Alice,
so Trudy simply declares
herself to be Alice

Authentication: another try

Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address

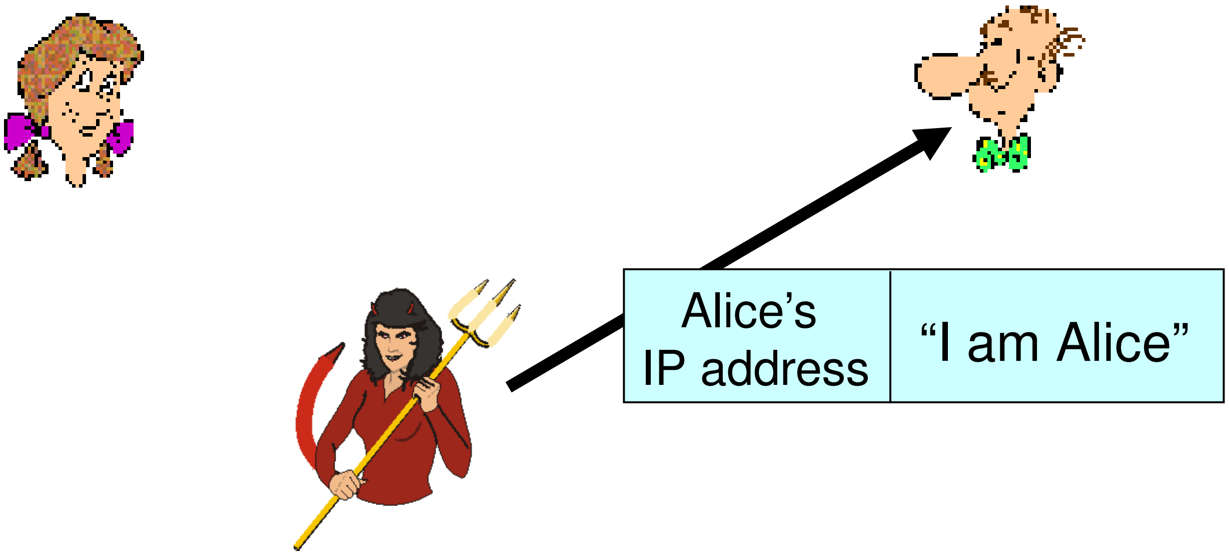


Alice’s IP address	“I am Alice”

Failure scenario??

Authentication: another try

Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address

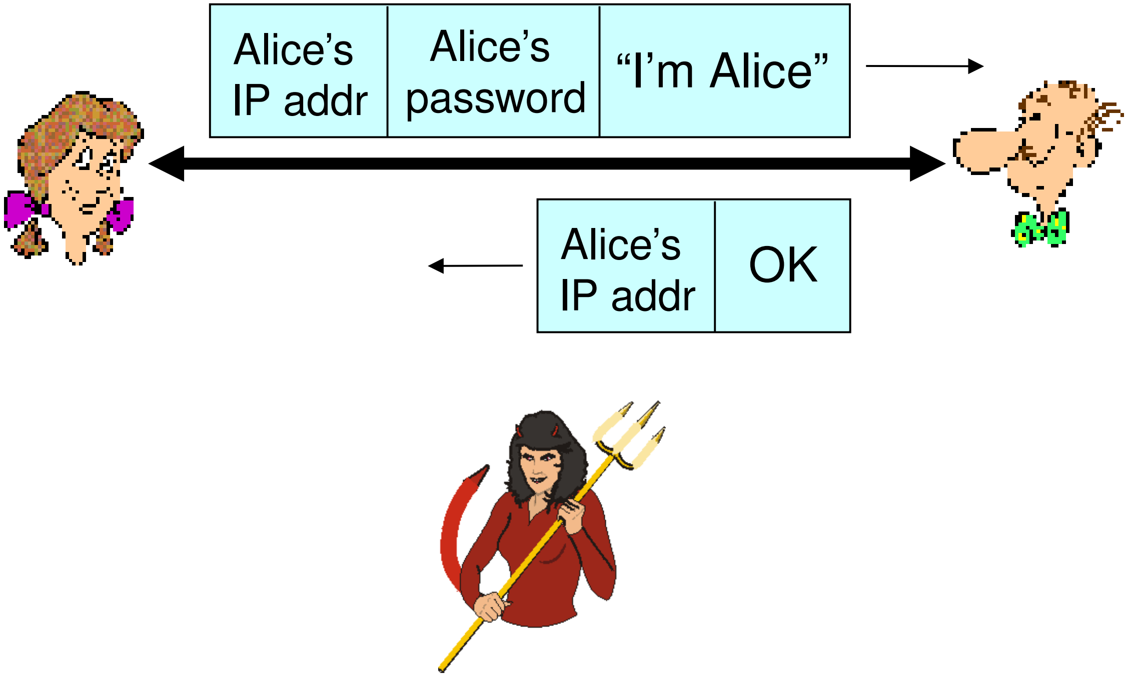


Alice’s IP address | “I am Alice”

Trudy can create
a packet “spoofing”
Alice’s address

Authentication: another try

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



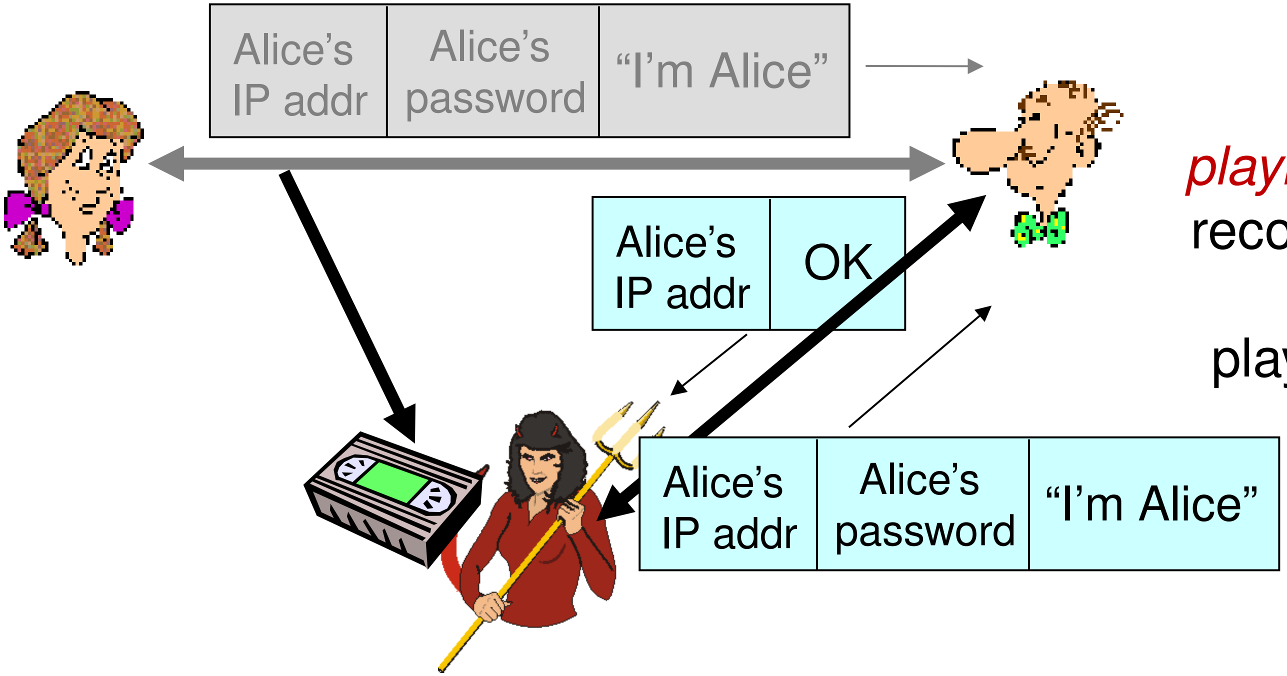
Alice's IP addr | Alice's password | "I'm Alice"

Alice's IP addr | OK

Failure scenario??

Authentication: another try

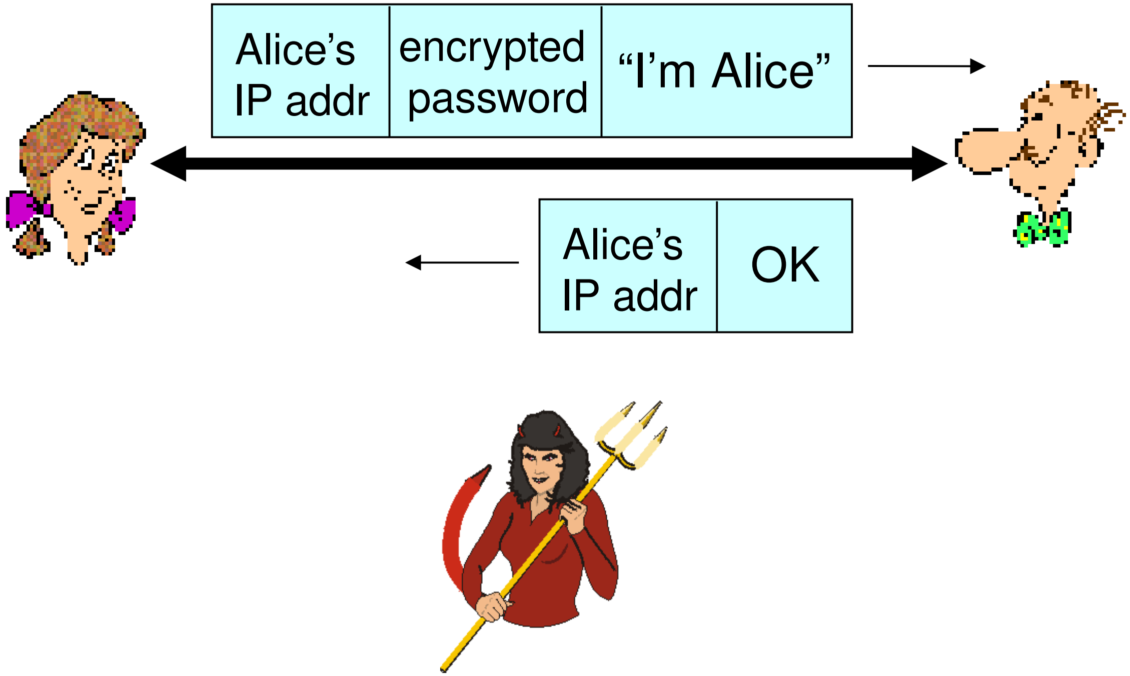
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



playback attack: Trudy records Alice's packet and later plays it back to Bob

Authentication: yet another try

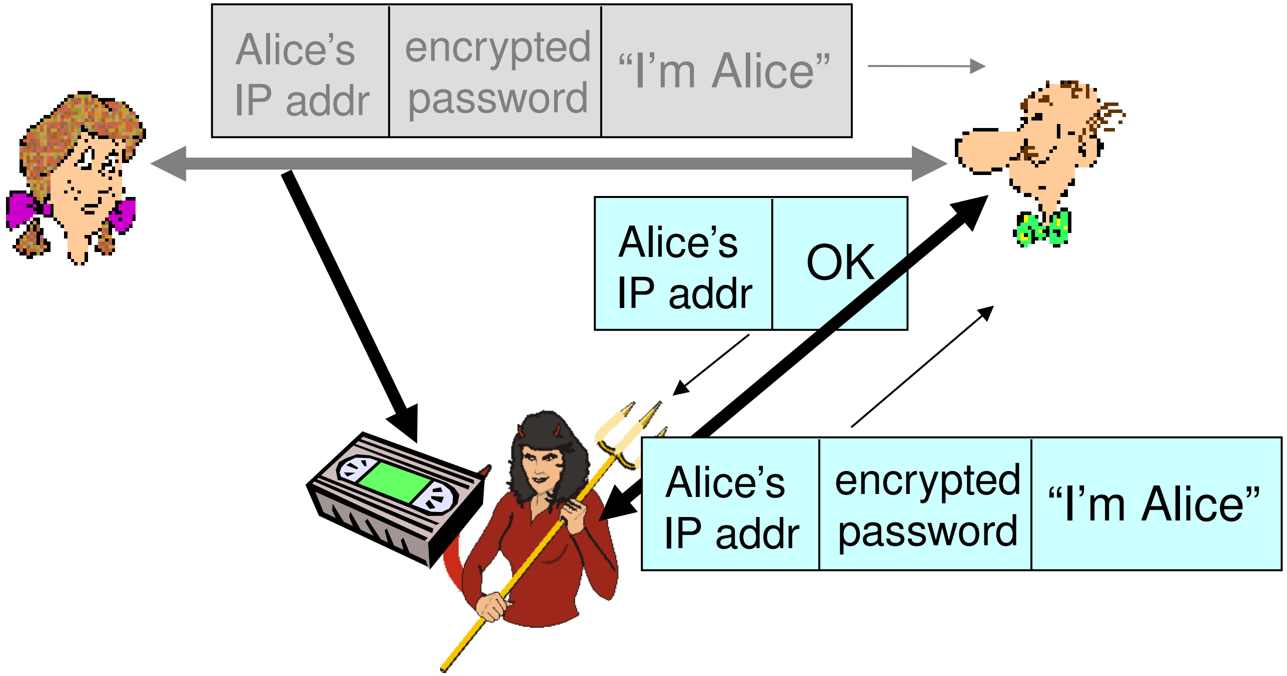
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: yet another try

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



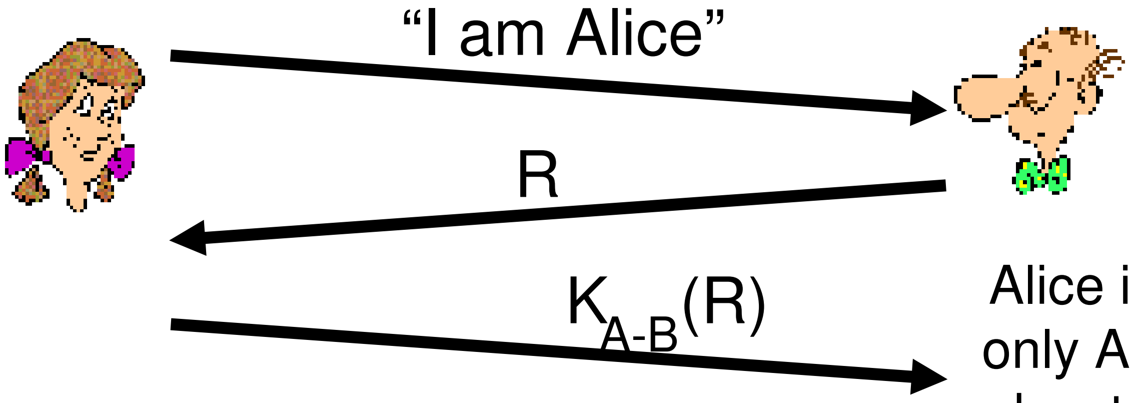
record and playback *still* works!

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 a *nonce*, R. Alice must return R, encrypted with shared secret key



"I am Alice"

R

$K_{A-B}(R)$

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

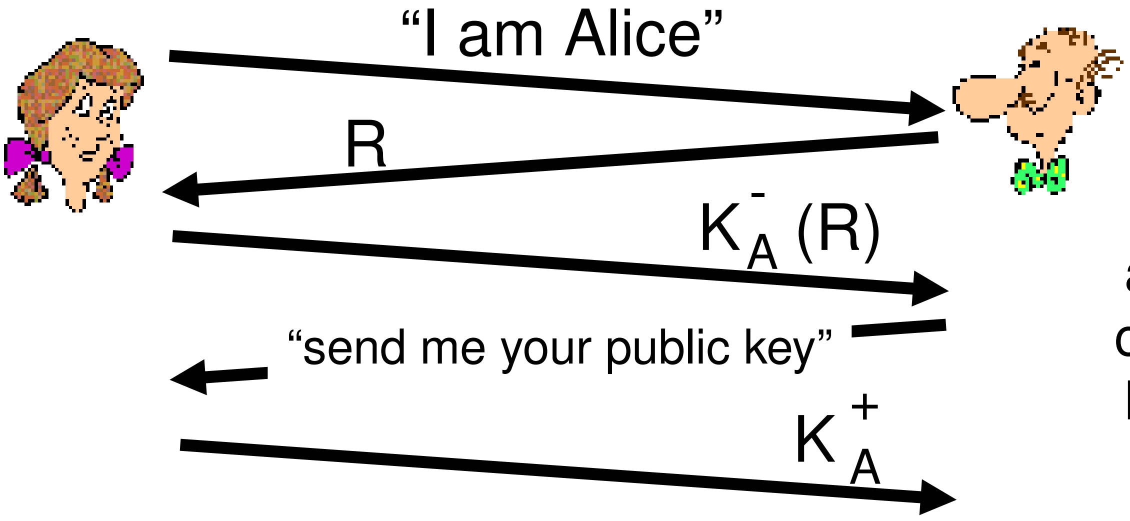
Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



"I am Alice"

R

$K_A^-(R)$

"send me your public key"

K_A^+

Bob computes

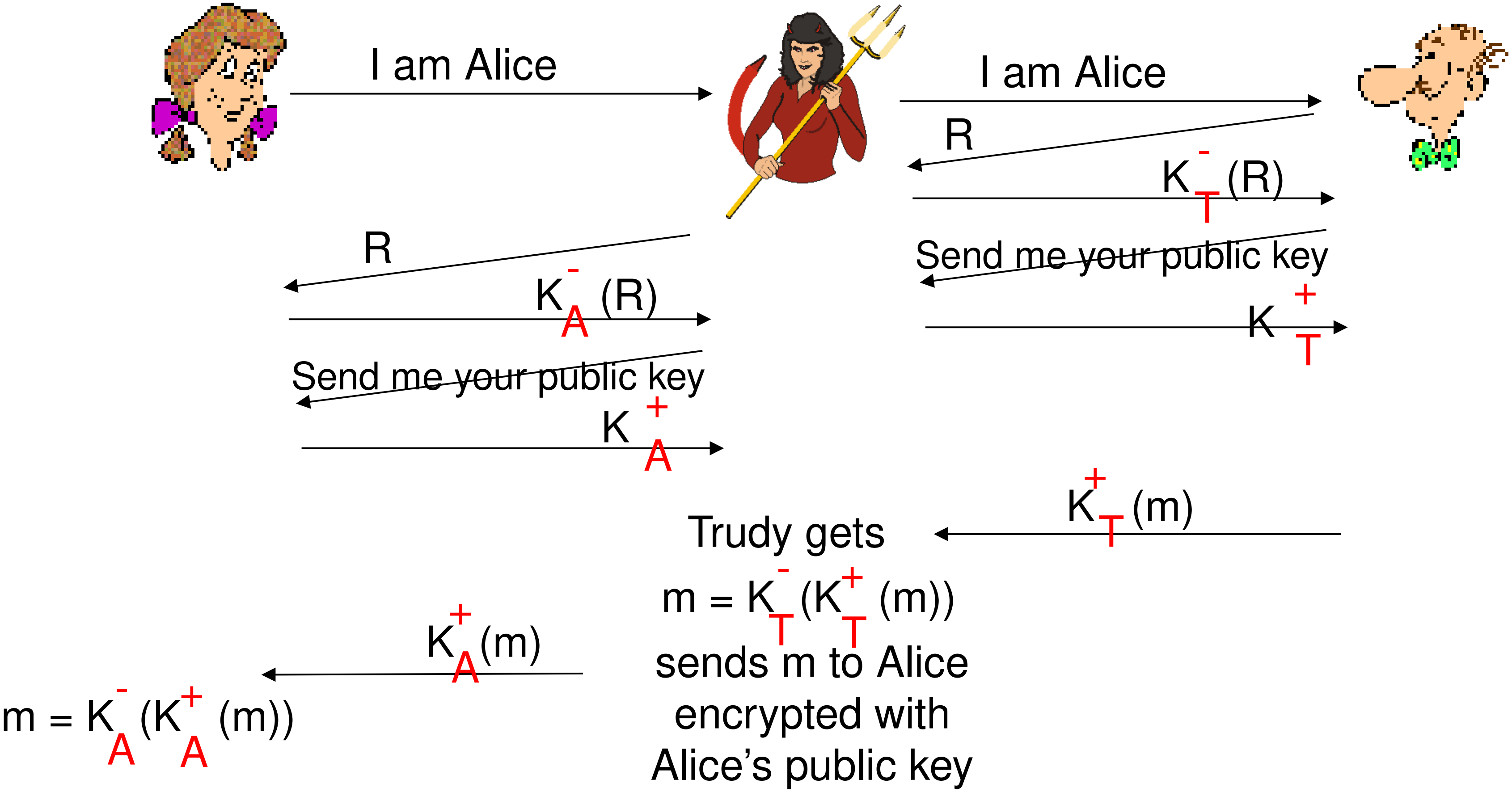
$$K_A^+(K_A^-(R)) = R$$

and knows only Alice could have the private key, that encrypted R such that

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

ap5.0: security hole

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



I am Alice

I am Alice

R

$K_T^-(R)$

Send me your public key

K_T^+

R

$K_A^-(R)$

Send me your public key

K_A^+

$K_T^+(m)$

Trudy gets

$m = K_T^-(K_T^+(m))$

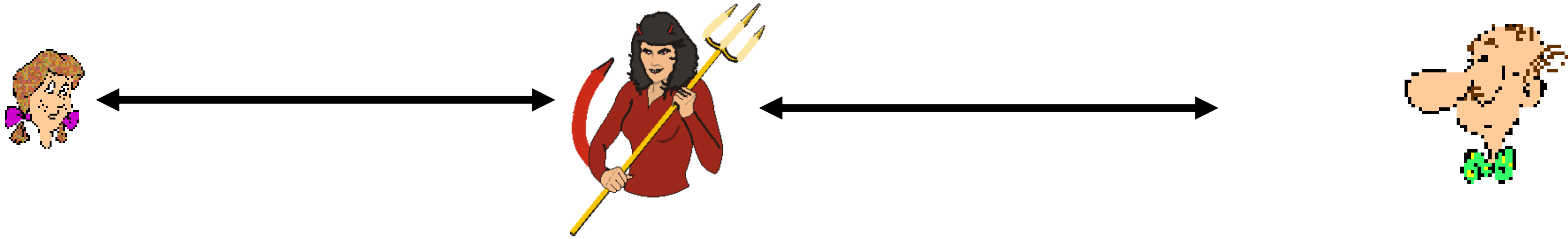
sends m to Alice encrypted with Alice’s public key

$K_A^+(m)$

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

ap5.0: security hole

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



difficult to detect:

- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation!)
- problem is that Trudy receives all messages as well!

Digital signatures

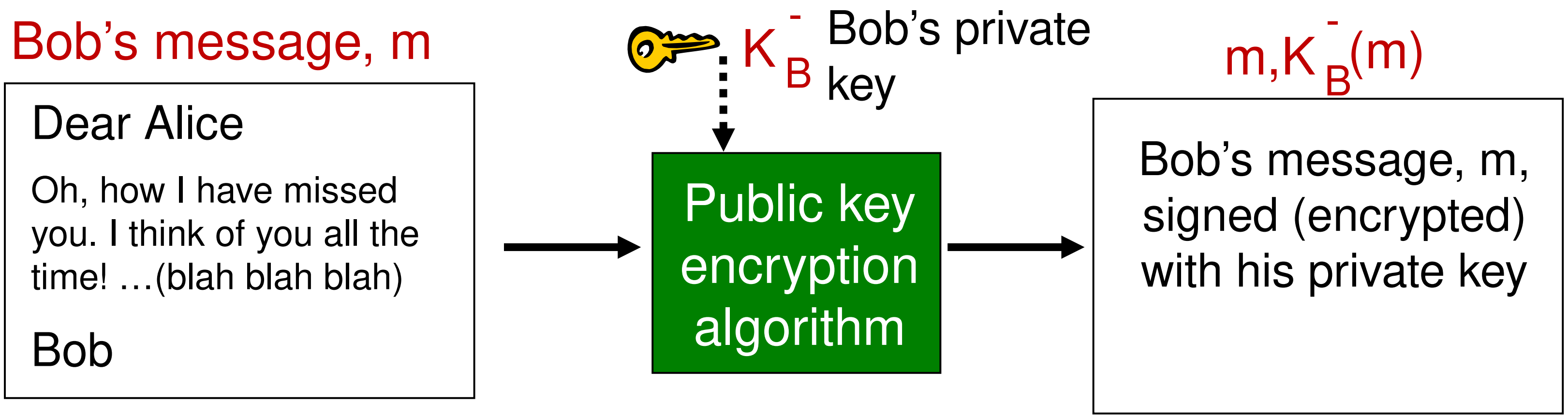
cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

Digital signatures

simple digital signature for message m:

- Bob signs m by encrypting with his private key K_B, creating "signed" message; $K_B(m)$



Bob's message, m

Dear Alice

Oh, how I have missed you. I think of you all the time! …(blah blah blah)

Bob

K_B^-
Bob's private key
Public key encryption algorithm
$m,K_B^-(m)$
Bob's message, m, signed (encrypted) with his private key

Digital signatures

- suppose Alice receives msg m, with signature: m, $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

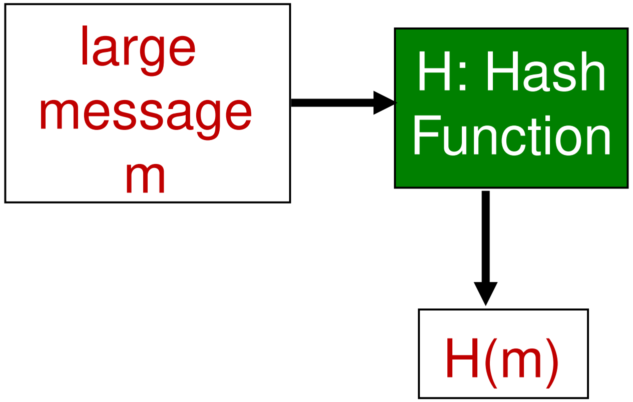
Alice thus verifies that:

- Bob signed m
- no one else signed m
- Bob signed m and not m'

non-repudiation:

- ✓ Alice can take m, and signature $K_B^-(m)$ 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 "fingerprint"

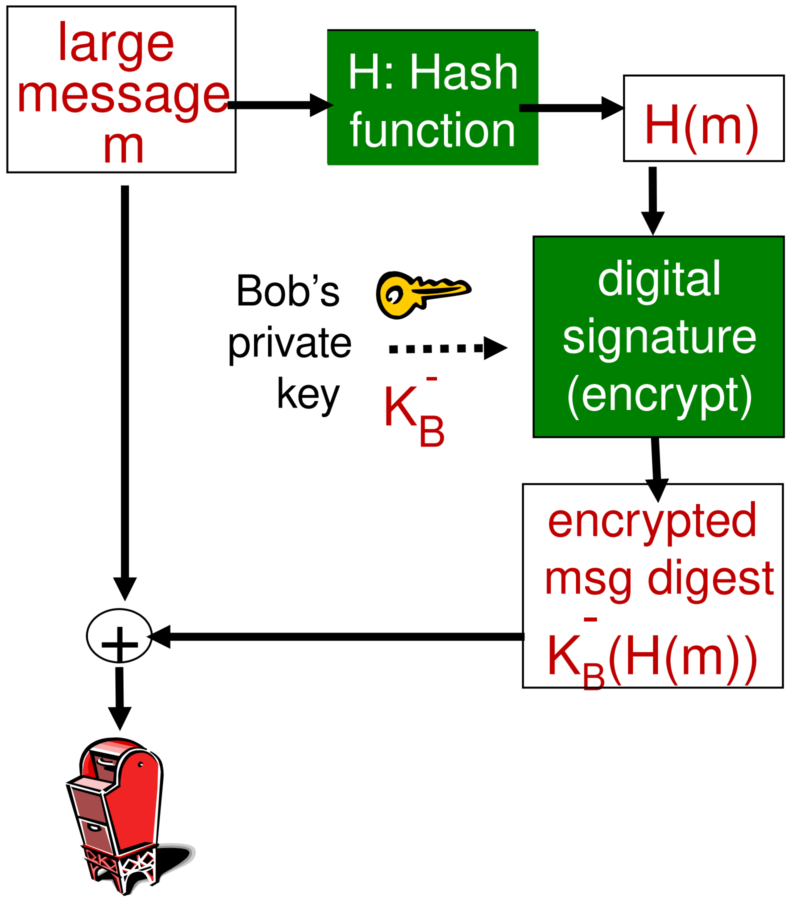
- apply hash function H to *m*, get fixed size message digest, *H(m).*

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)

Digital signature = signed message digest

Bob sends digitally signed message:



large message m

H: Hash function

H(m)

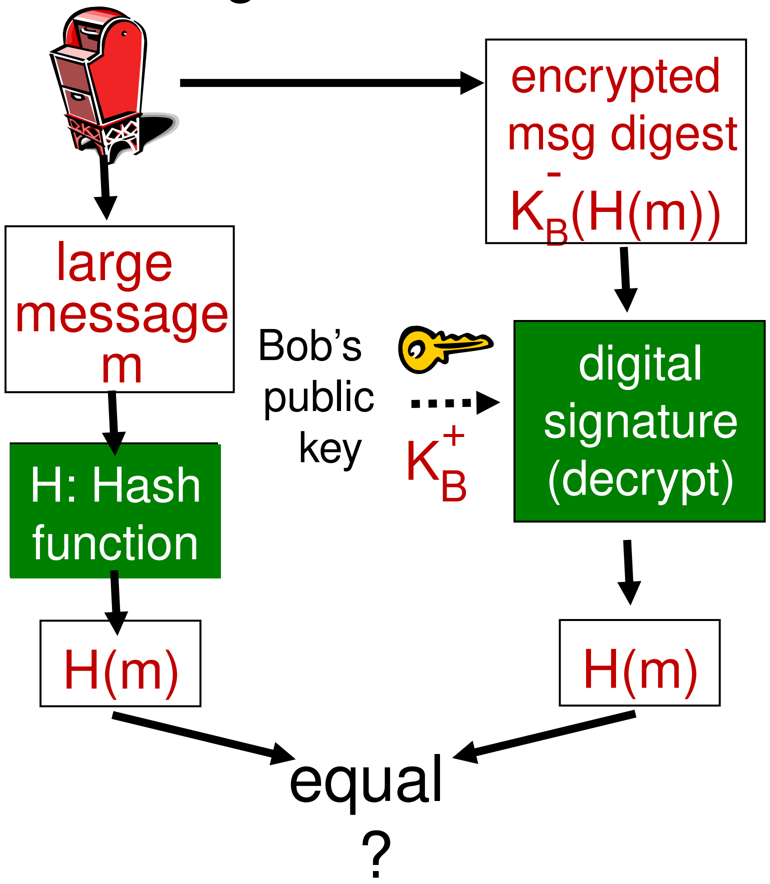
Bob's private key K_B^-

digital signature (encrypt)

encrypted msg digest $K_B^-(H(m))$

+

Alice verifies signature, integrity of digitally signed message:



encrypted msg digest $K_B^-(H(m))$

large message m

H: Hash function

H(m)

Bob's public key K_B^+

digital signature (decrypt)

H(m)

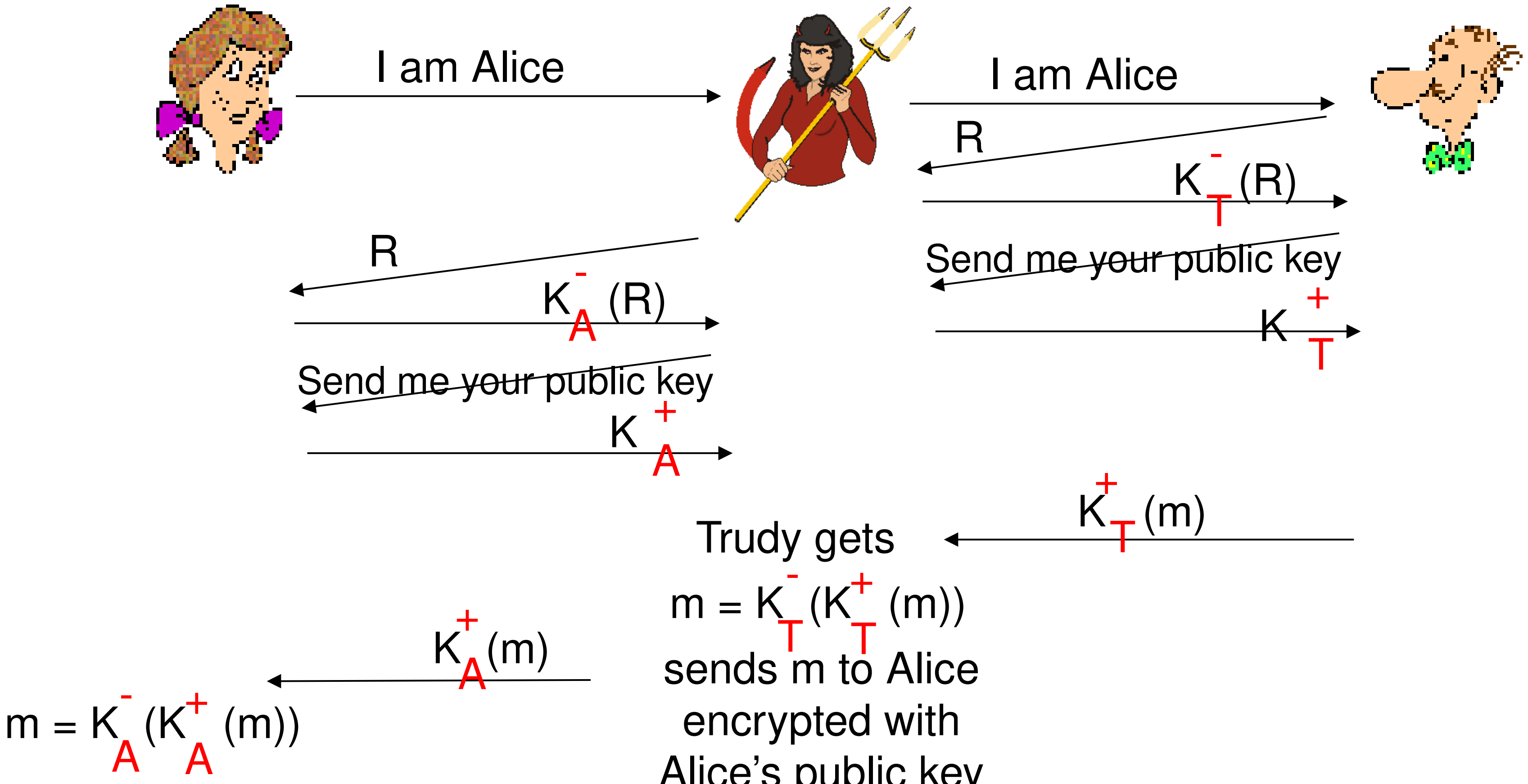
equal ?

Hash function algorithms

- MD5 hash function widely used (RFC 1321)
  - 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
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest
- SHA-2 A family of two similar hash functions, with different block sizes, known as SHA-256 and SHA-512. They differ in the word size; SHA-256 uses 32-bit words where SHA-512 uses 64-bit words.

Recall: ap5.0 security hole

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



I am Alice

I am Alice

R

$K_T^-(R)$

Send me your public key

K_T^+

R

$K_A^-(R)$

Send me your public key

K_A^+

$K_T^+(m)$

Trudy gets

$m = K_T^-(K_T^+(m))$

sends m to Alice encrypted with Alice's public key

$K_A^+(m)$

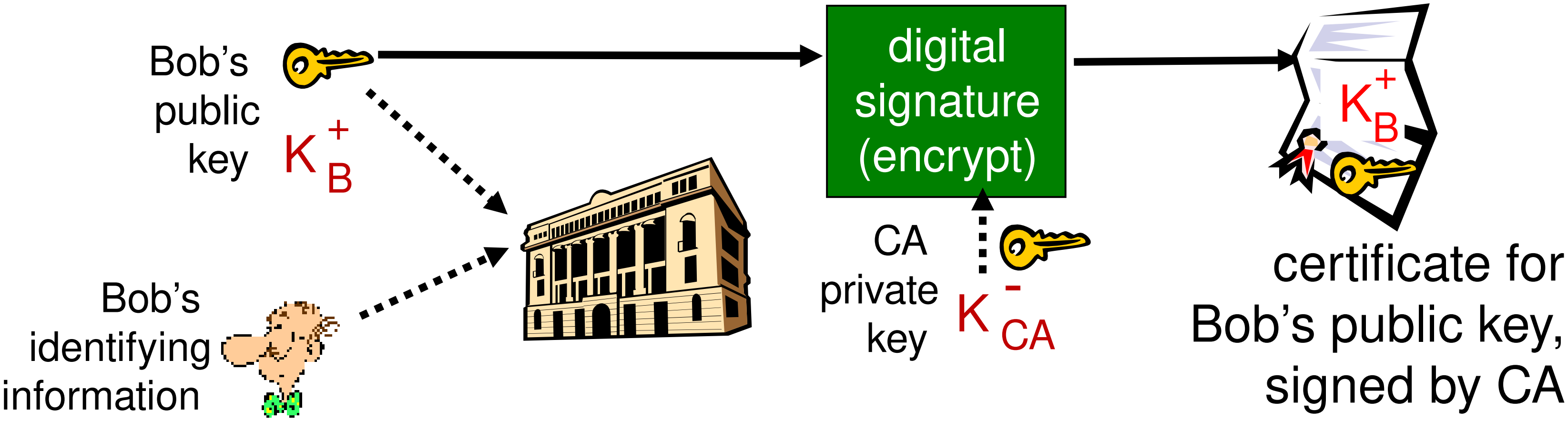
$m = K_A^-(K_A^+(m))$

Public-key certification

- motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key
  - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
  - Bob doesn't even like pepperoni

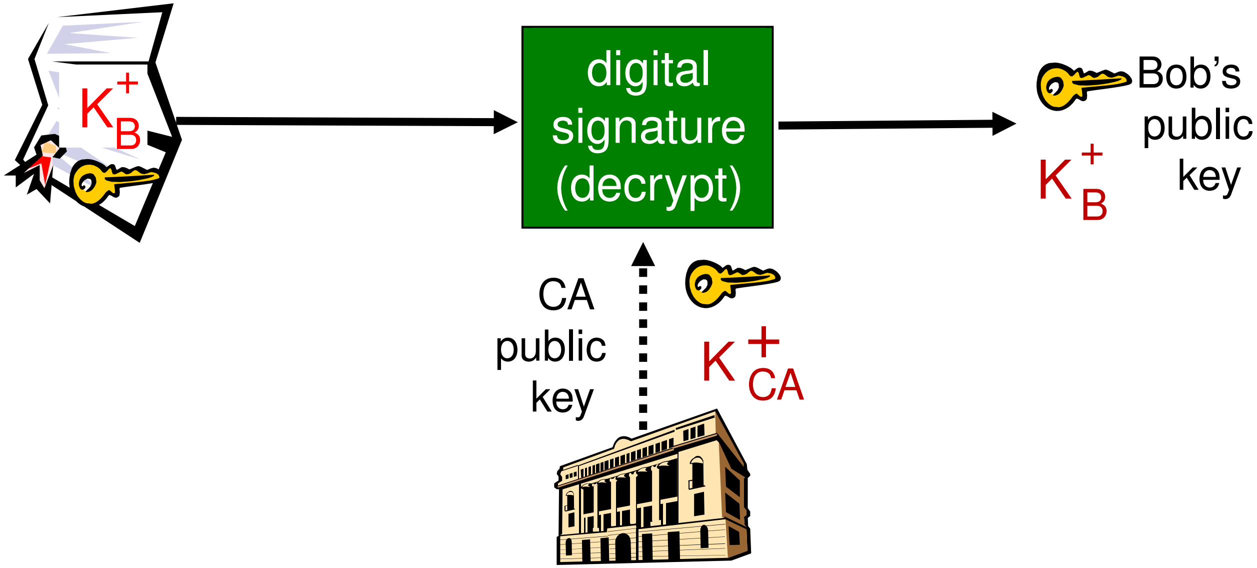
Certification authorities

- *certification authority (CA):* binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"


Bob's public key
K_B^+
Bob's identifying information
digital signature (encrypt)
CA private key
K_{CA}^-
K_B^+
certificate for Bob's public key, signed by CA

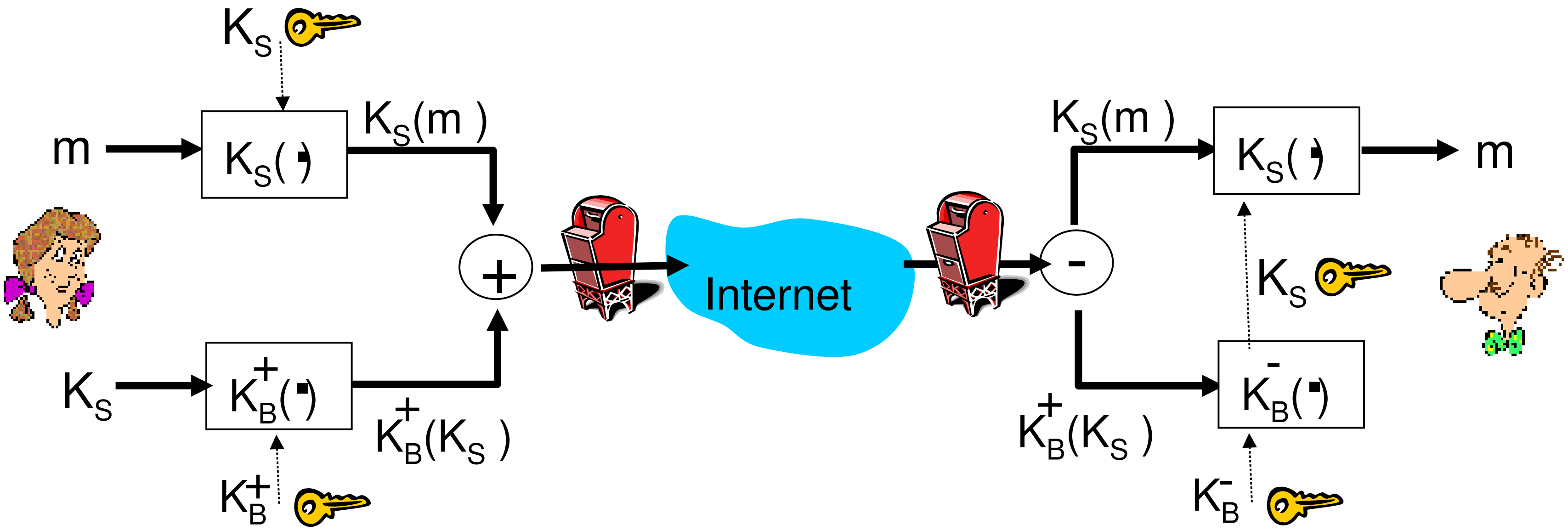
Certification authorities

- 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



Secure e-mail

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

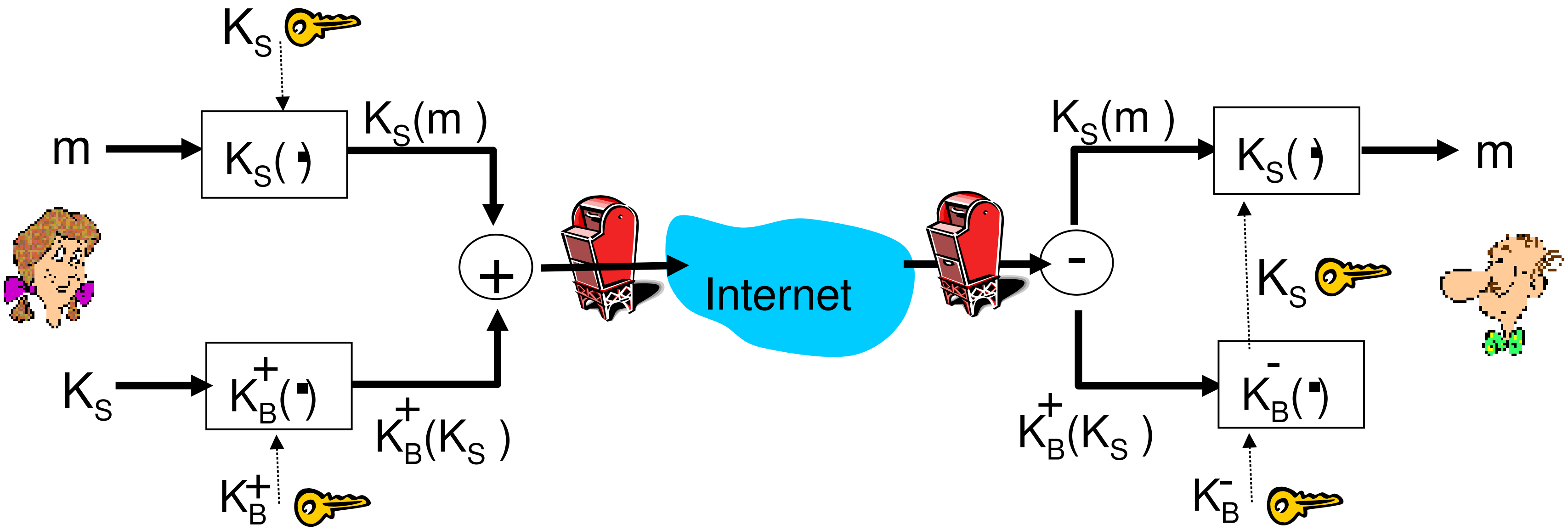


Alice:

- 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 e-mail

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

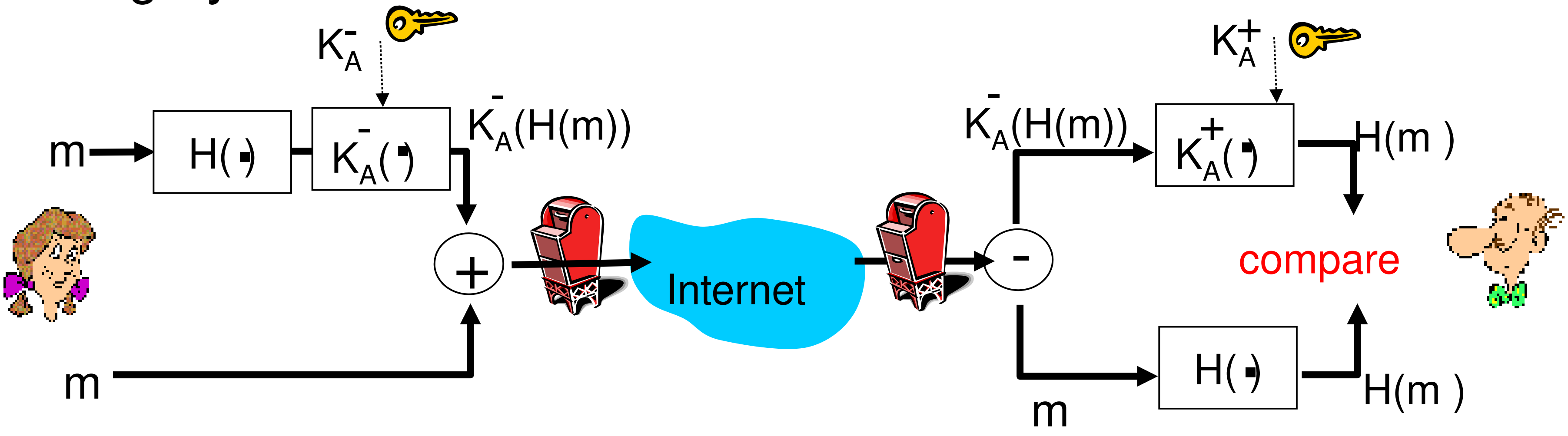


Bob:

- uses his private key to decrypt and recover K_S
- uses K_S to decrypt $K_S(m)$ to recover m

Secure e-mail (continued)

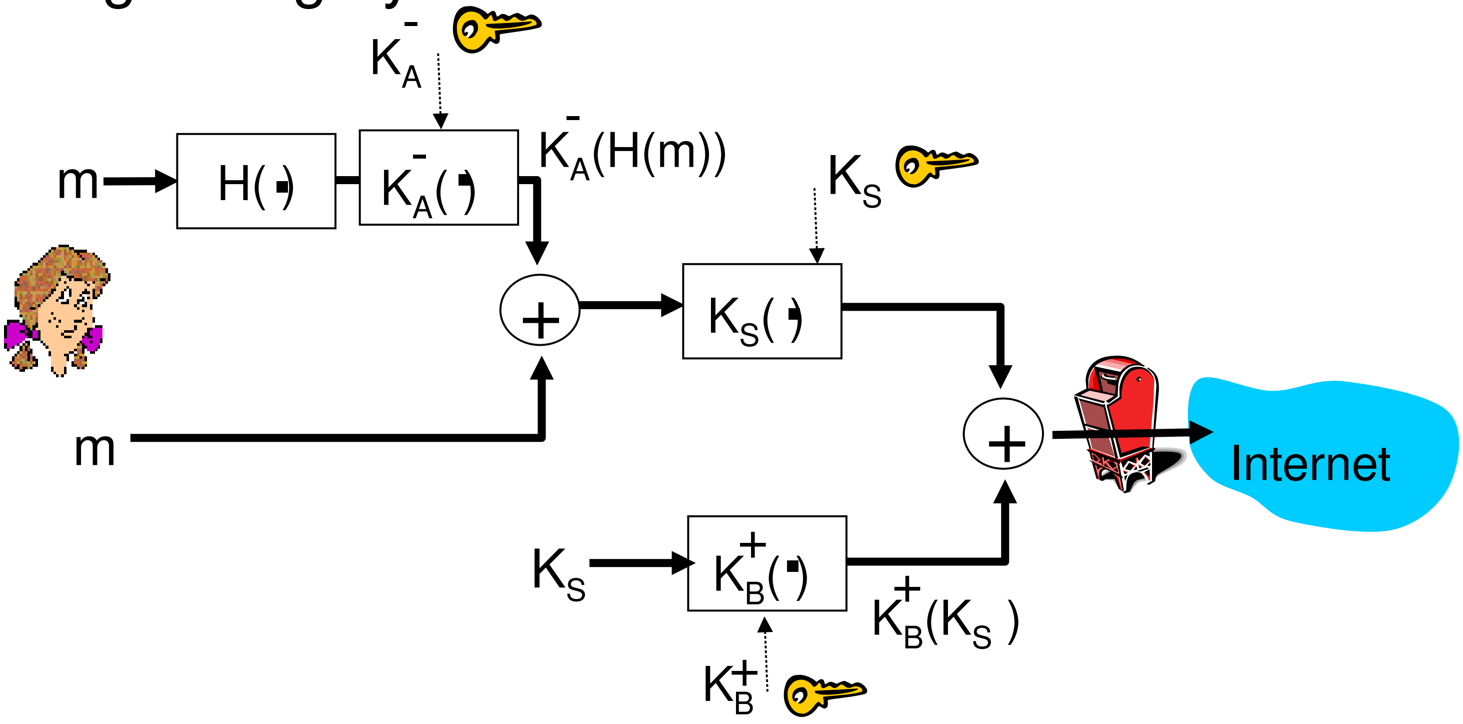
Alice wants to provide sender authentication message integrity



- 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.



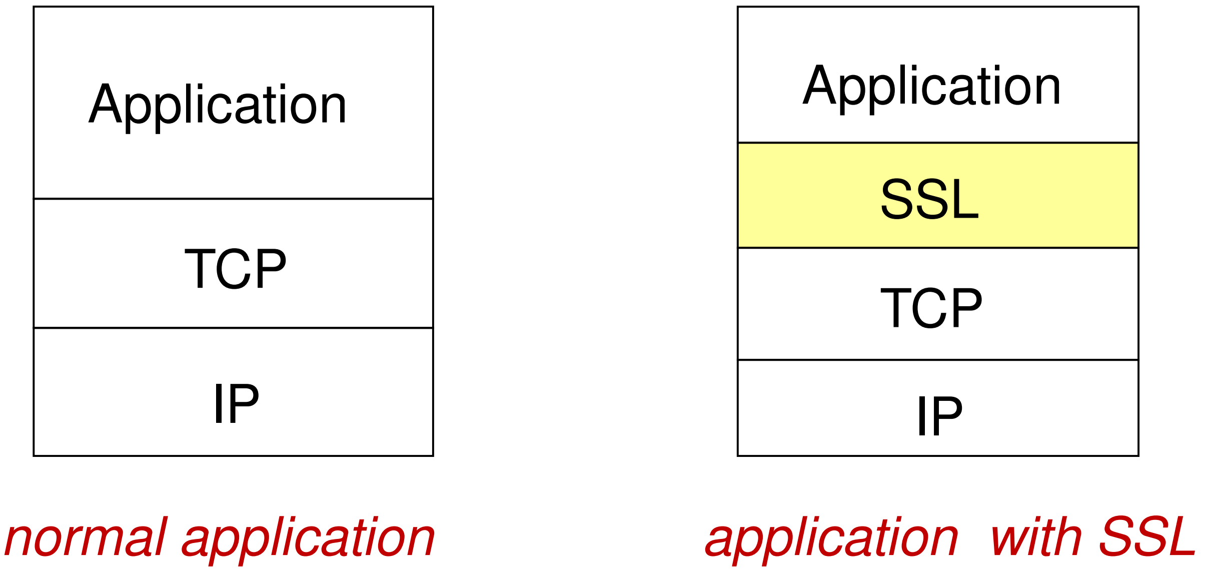
Alice uses three keys: her private key, Bob's public key, newly created symmetric key

SSL: Secure Sockets Layer

- widely deployed security protocol
  - supported by almost all browsers, web servers
  - https
  - billions $/year over SSL
- mechanisms: [Woo 1994], implementation: Netscape
- variation -TLS: transport layer security, RFC 2246
- provides
  - *confidentiality*
  - *integrity*
  - *authentication*

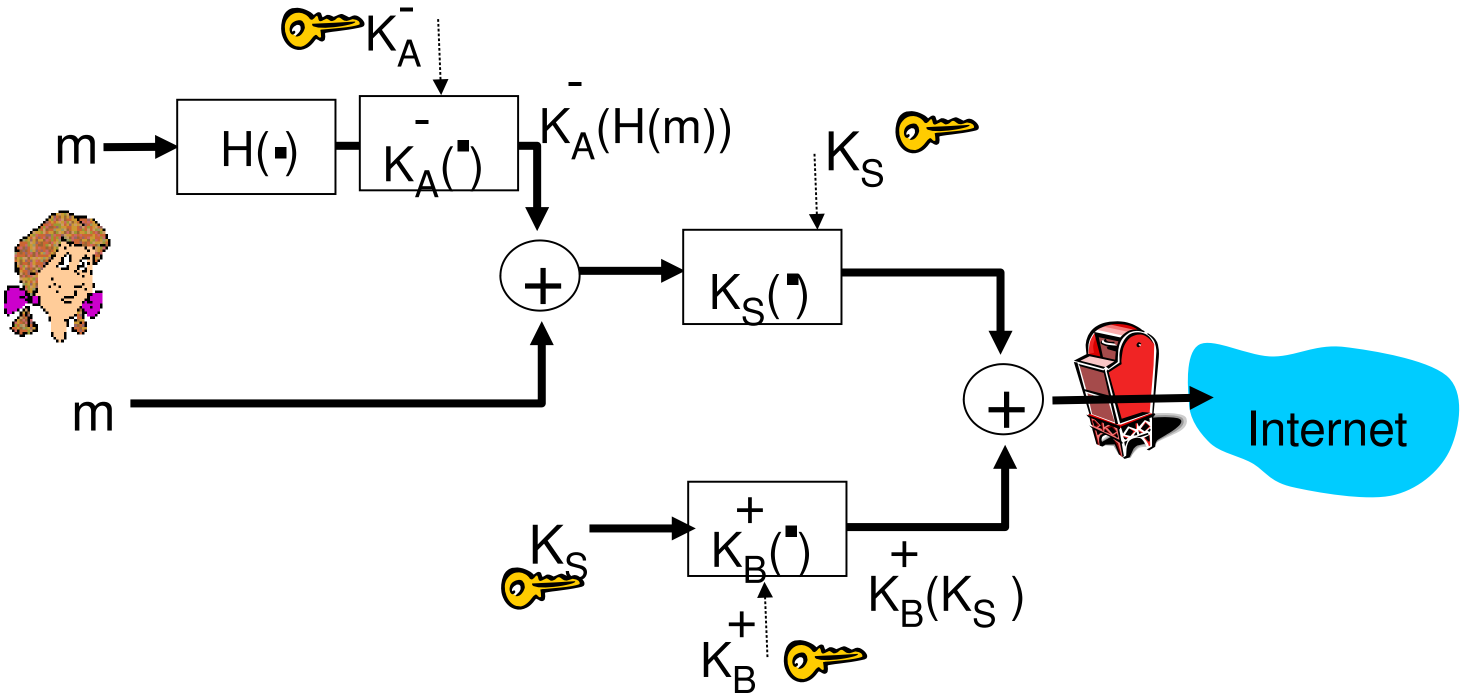
- original goals:
  - Web e-commerce transactions
  - encryption (especially credit-card numbers)
  - Web-server authentication
  - optional client authentication
  - minimum hassle in doing business with new merchant
- available to all TCP applications
  - secure socket interface

SSL and TCP/IP



- SSL provides application programming interface (API) to applications
- C and Java SSL libraries/classes readily available

Could do something like PGP:

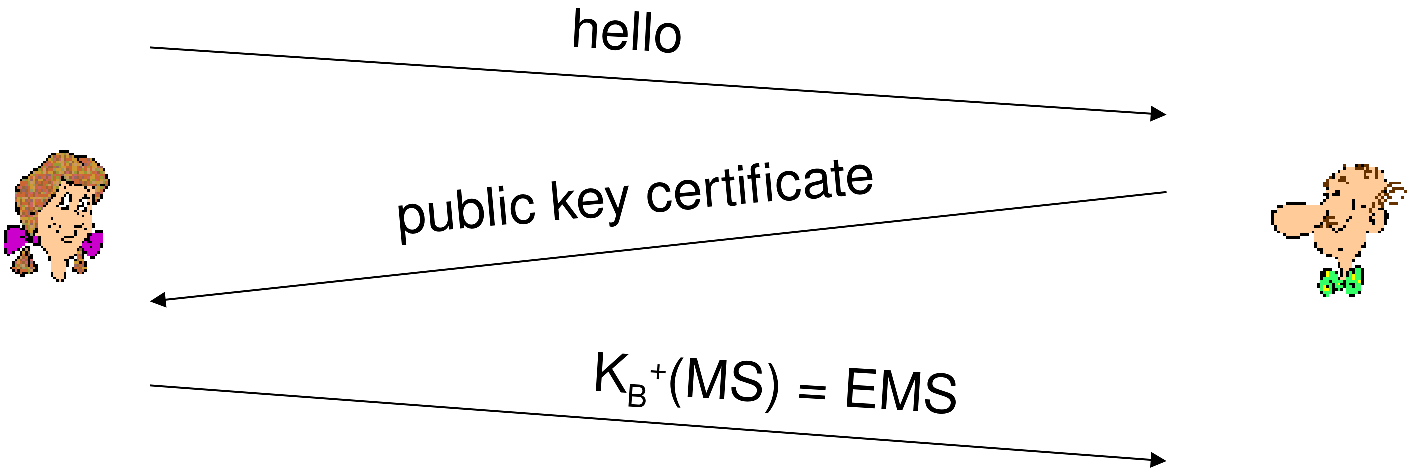


- but want to send byte streams & interactive data
- want set of secret keys for entire connection
- want certificate exchange as part of protocol: handshake phase

Toy SSL: a simple secure channel

- *handshake:* Alice and Bob use their certificates, private keys to authenticate each other and exchange shared secret
- *key derivation:* Alice and Bob use shared secret to derive set of keys
- *data transfer:* data to be transferred is broken up into series of records
- *connection closure:* special messages to securely close connection

Toy: a simple handshake



hello

public key certificate

$K_B^+(MS) = EMS$

MS: master secret

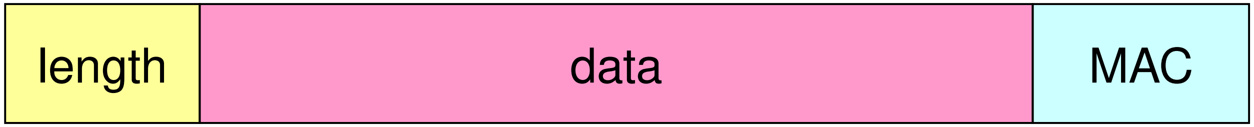
EMS: encrypted master secret

Toy: key derivation

- considered bad to use same key for more than one cryptographic operation
  - use different keys for message authentication code (MAC) and encryption
- four keys:
  - K_c = encryption key for data sent from client to server
  - M_c = MAC key for data sent from client to server
  - K_s = encryption key for data sent from server to client
  - M_s = MAC key for data sent from server to client
- keys derived from key derivation function (KDF)
  - takes master secret and (possibly) some additional random data and creates the keys

Toy: data records

- why not encrypt data in constant stream as we write it to TCP?
  - where would we put the MAC? If at end, no message integrity until all data processed.
  - e.g., with instant messaging, how can we do integrity check over all bytes sent before displaying?
- instead, break stream in series of records
  - each record carries a MAC
  - receiver can act on each record as it arrives
- issue: in record, receiver needs to distinguish MAC from data
  - want to use variable-length records



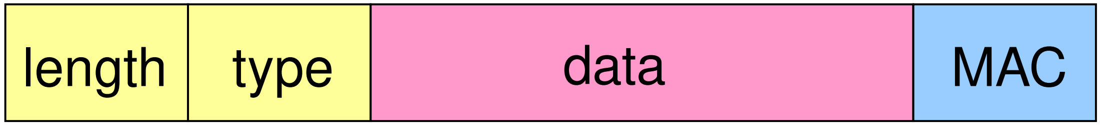
Toy: sequence numbers

- *problem:* attacker can capture and replay record or re-order records
- *solution:* put sequence number into MAC:
  - MAC = MAC(M_x, sequence||data)
  - note: no sequence number field

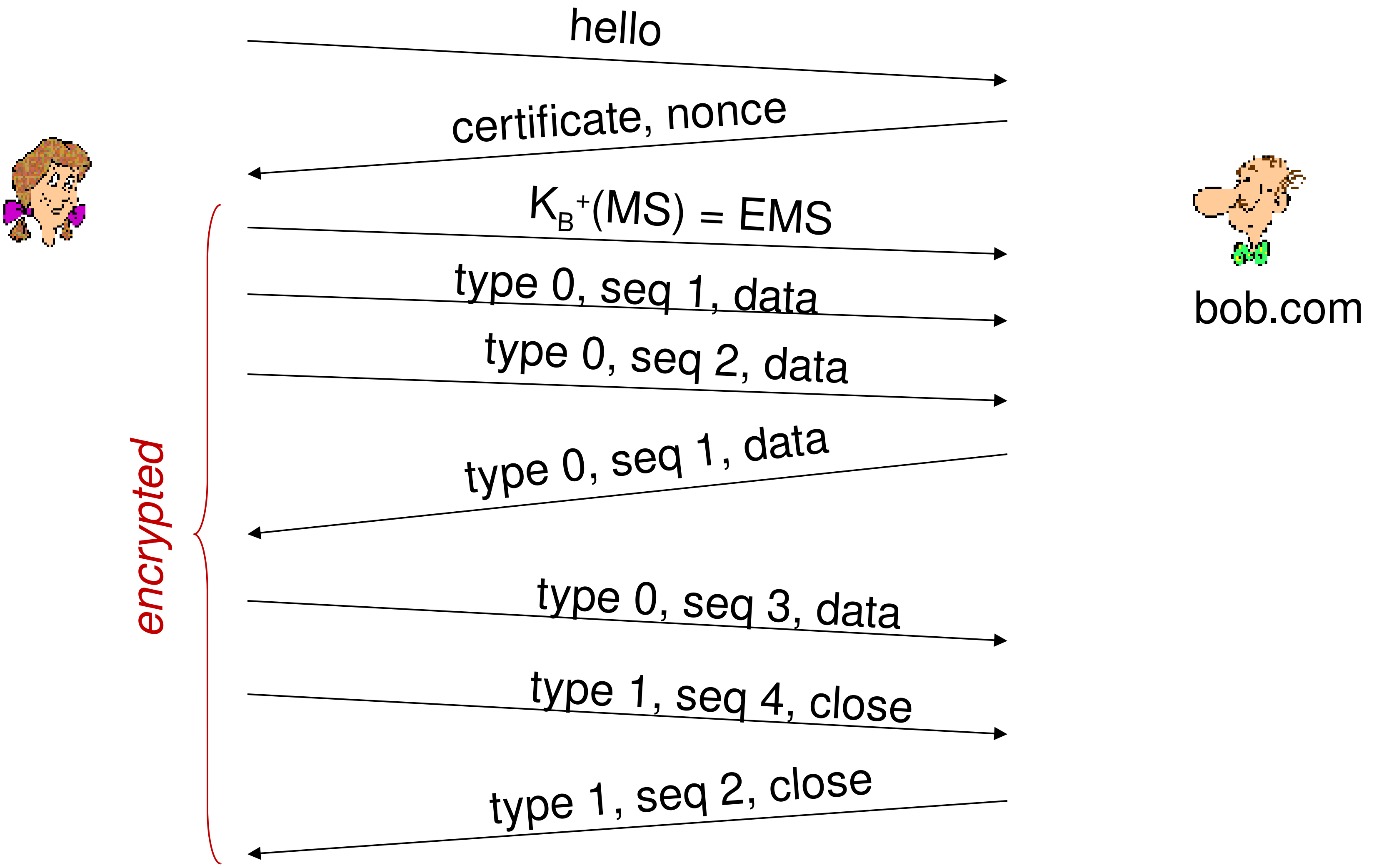
- *problem:* attacker could replay all records
- *solution:* use nonce

Toy: control information

- *problem:* truncation attack:
  - attacker forges TCP connection close segment
  - one or both sides thinks there is less data than there actually is.
- *solution:* record types, with one type for closure
  - type 0 for data; type 1 for closure
- MAC = MAC(M_x, sequence||type||data)



Toy SSL: summary



Toy SSL isn't complete

- how long are fields?
- which encryption protocols?
- want negotiation?
  - allow client and server to support different encryption algorithms
  - allow client and server to choose together specific algorithm before data transfer

SSL cipher suite

- cipher suite
  - public-key algorithm
  - symmetric encryption algorithm
  - MAC algorithm
- SSL supports several cipher suites
- negotiation: client, server agree on cipher suite
  - client offers choice
  - server picks one

common SSL symmetric ciphers

- DES – Data Encryption Standard: block
- 3DES – Triple strength: block
- RC2 – Rivest Cipher 2: block
- RC4 – Rivest Cipher 4: stream

SSL Public key encryption

- RSA

Real SSL: handshake (1)

Purpose

1. server authentication
2. negotiation: agree on crypto algorithms
3. establish keys
4. client authentication (optional)

Real SSL: handshake (2)

1. client sends list of algorithms it supports, along with client nonce
2. server chooses algorithms from list; sends back: choice + certificate + server nonce
3. client verifies certificate, extracts server's public key, generates pre_master_secret, encrypts with server's public key, sends to server
4. client and server independently compute encryption and MAC keys from pre_master_secret and nonces
5. client sends a MAC of all the handshake messages
6. server sends a MAC of all the handshake messages

Real SSL: handshaking (3)

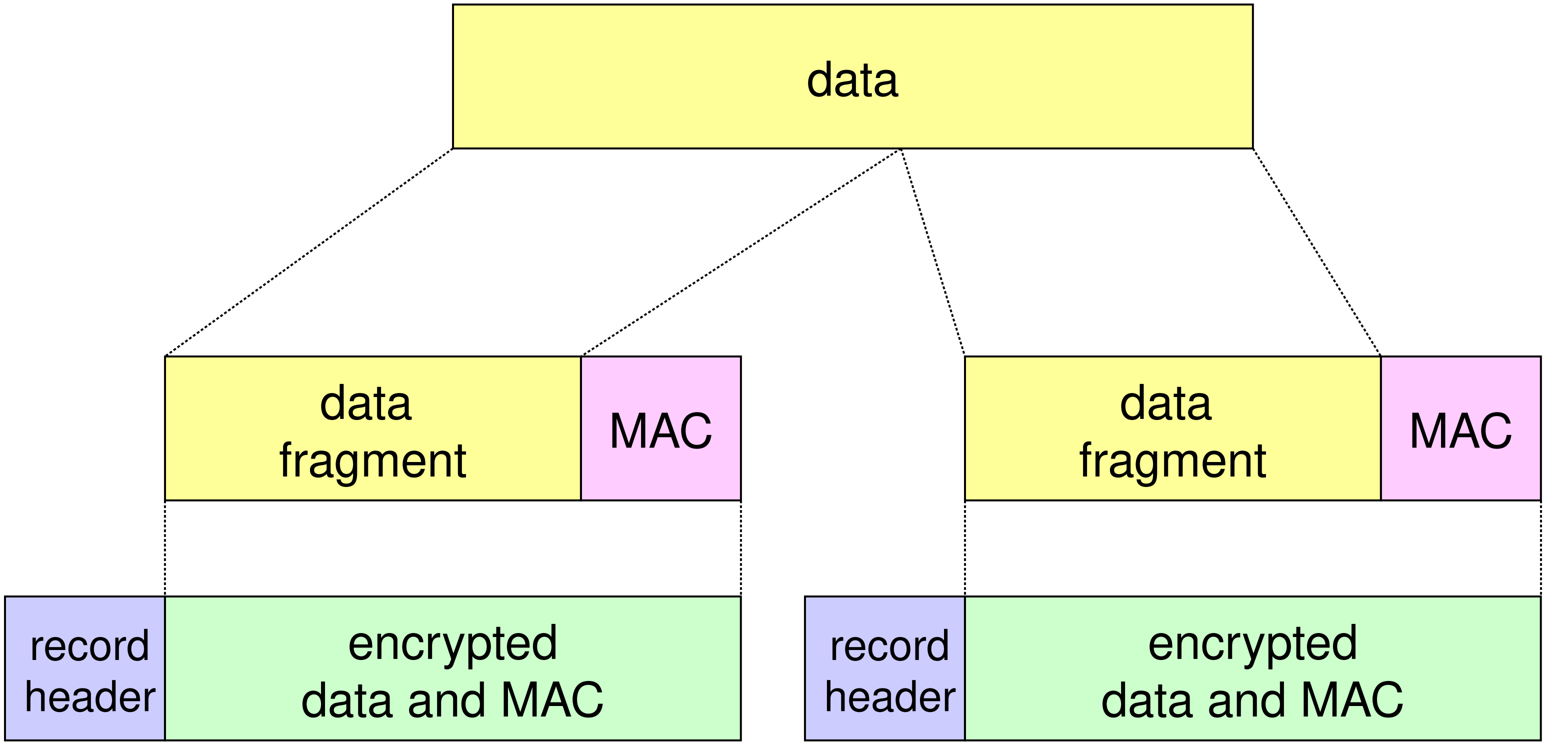
last 2 steps protect handshake from tampering

- client typically offers range of algorithms, some strong, some weak
- man-in-the middle could delete stronger algorithms from list
- last 2 steps prevent this
  - last two messages are encrypted

Real SSL: handshaking (4)

- why two random nonces?
- suppose Trudy sniffs all messages between Alice & Bob
- next day, Trudy sets up TCP connection with Bob, sends exact same sequence of records
  - Bob (Amazon) thinks Alice made two separate orders for the same thing
  - solution: Bob sends different random nonce for each connection. This causes encryption keys to be different on the two days
  - Trudy's messages will fail Bob's integrity check

SSL record protocol

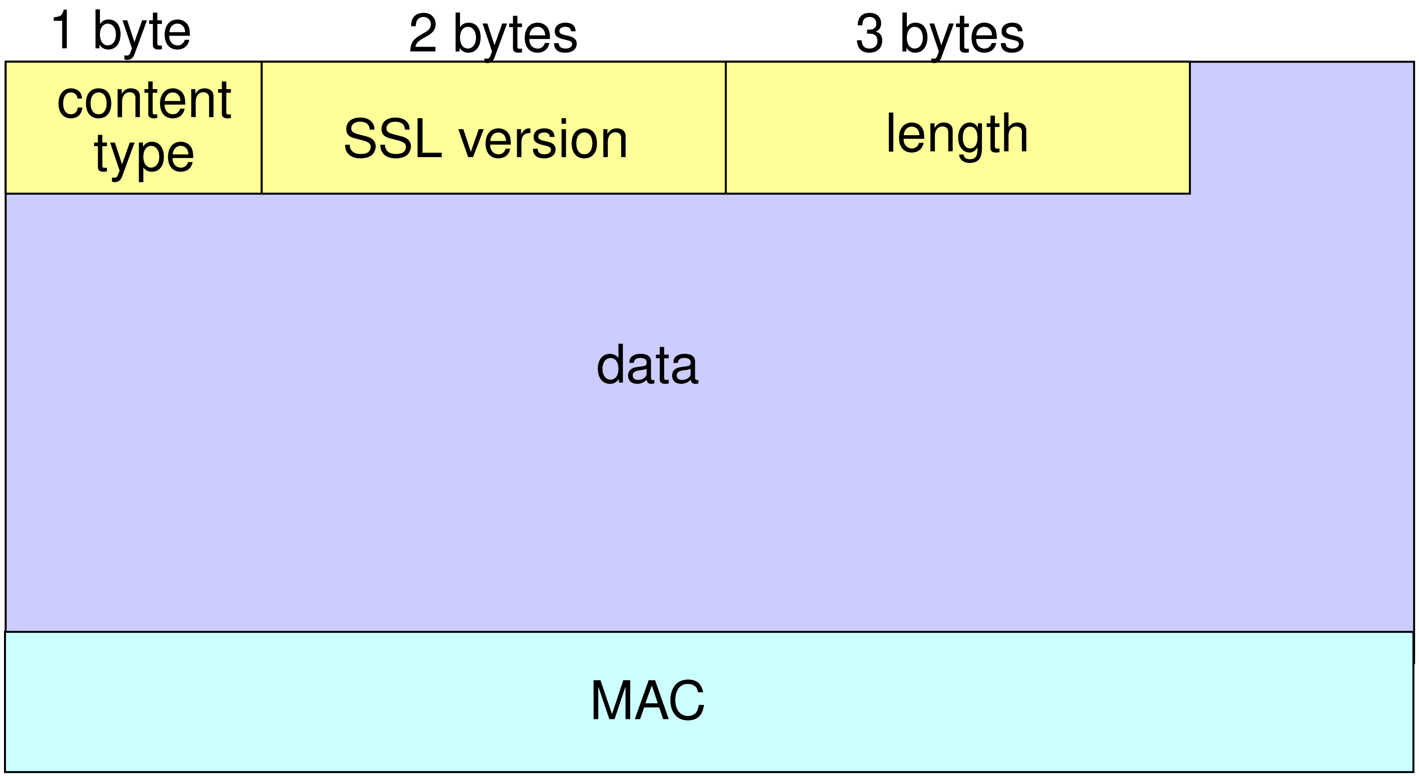


record header: content type; version; length

MAC: includes sequence number, MAC key M_x

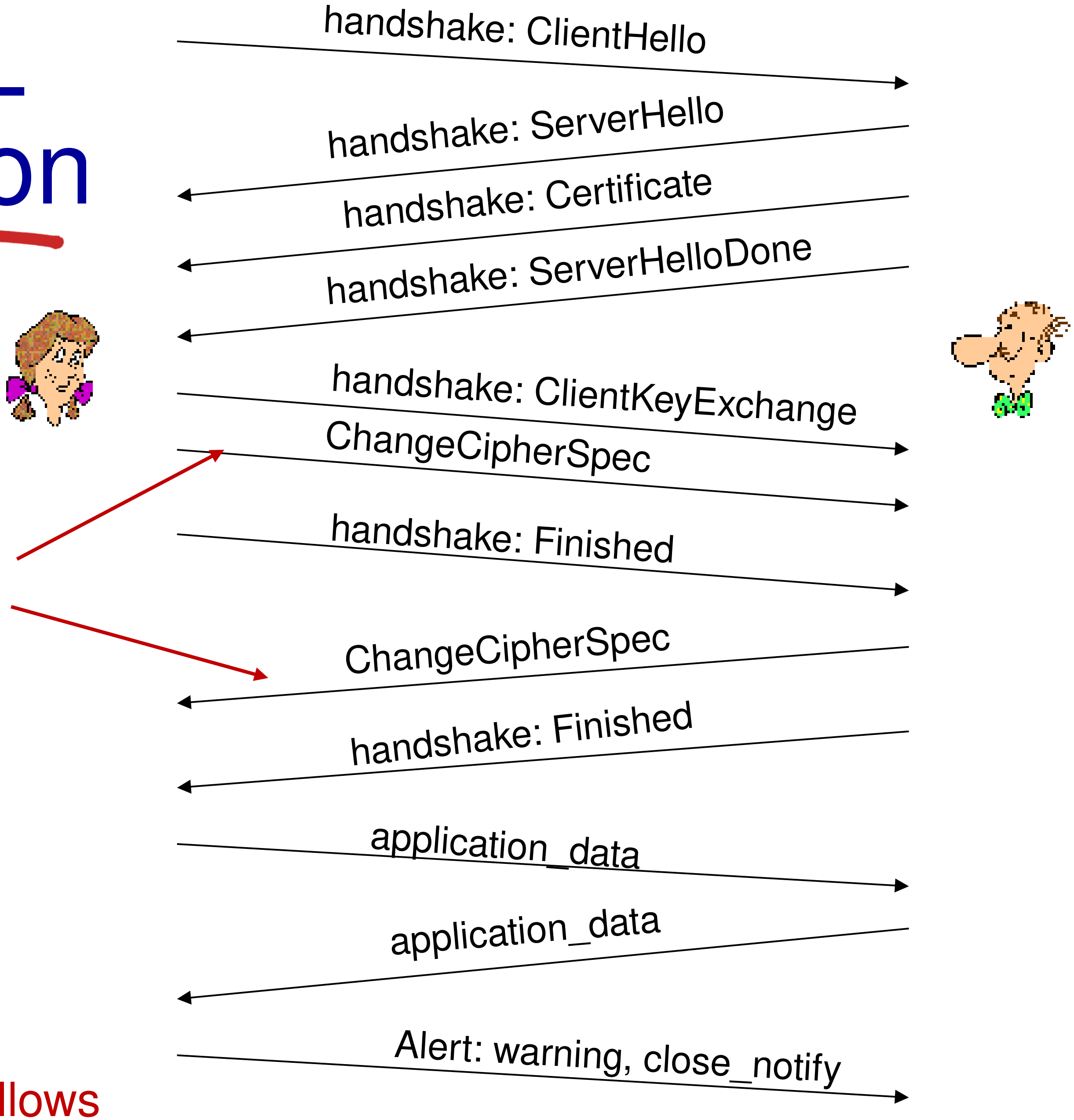
fragment: each SSL fragment 2^{14} bytes (~16 Kbytes)

SSL record format



data and MAC encrypted (symmetric algorithm)

Real SSL connection



handshake: ClientHello

handshake: ServerHello

handshake: Certificate

handshake: ServerHelloDone

handshake: ClientKeyExchange

ChangeCipherSpec

handshake: Finished

ChangeCipherSpec

handshake: Finished

application_data

application_data

Alert: warning, close_notify

everything henceforth is encrypted

TCP FIN follows

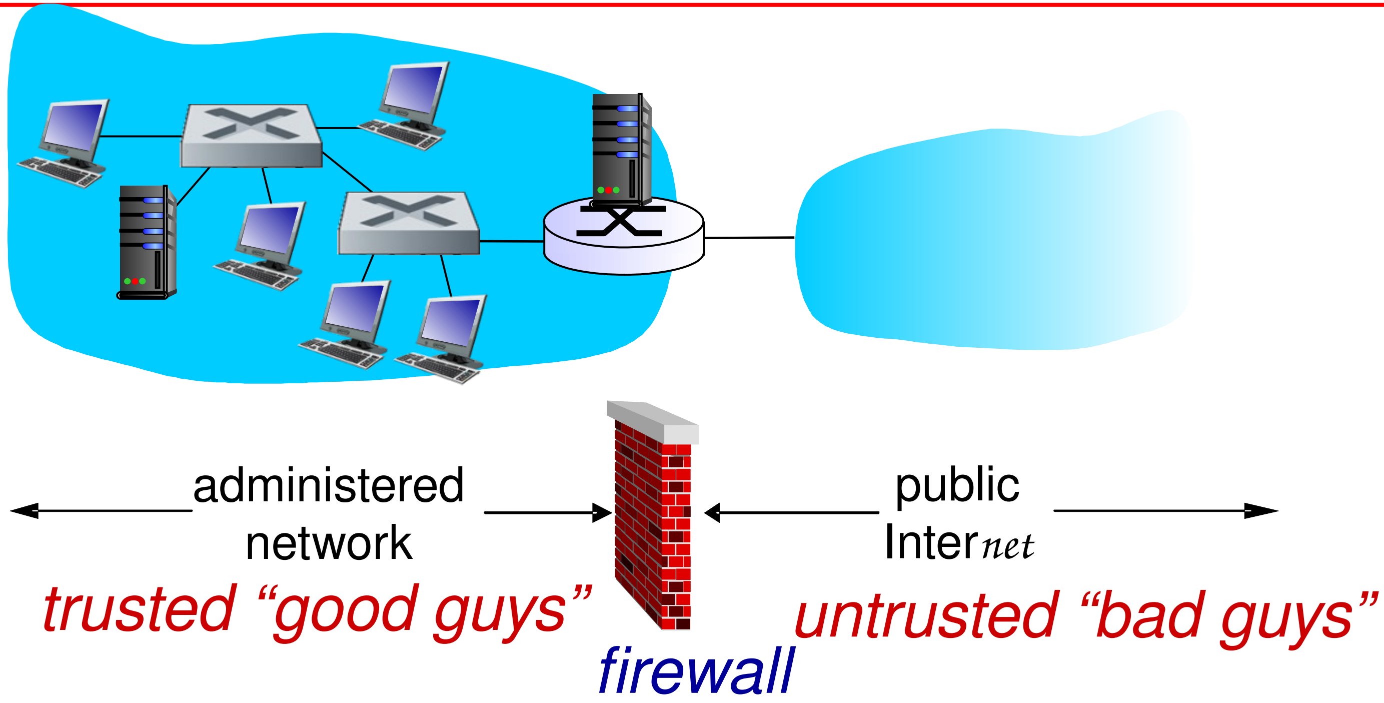
Key derivation

- client nonce, server nonce, and pre-master secret input into pseudo random-number generator.
  - produces master secret
- master secret and new nonces input into another random-number generator: “key block”
  - because of resumption: TBD
- key block sliced and diced:
  - client MAC key
  - server MAC key
  - client encryption key
  - server encryption key
  - client initialization vector (IV)
  - server initialization vector (IV)

Firewalls

firewall

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



administered network

trusted "good guys"

firewall

public Inter*net*

untrusted "bad guys"

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

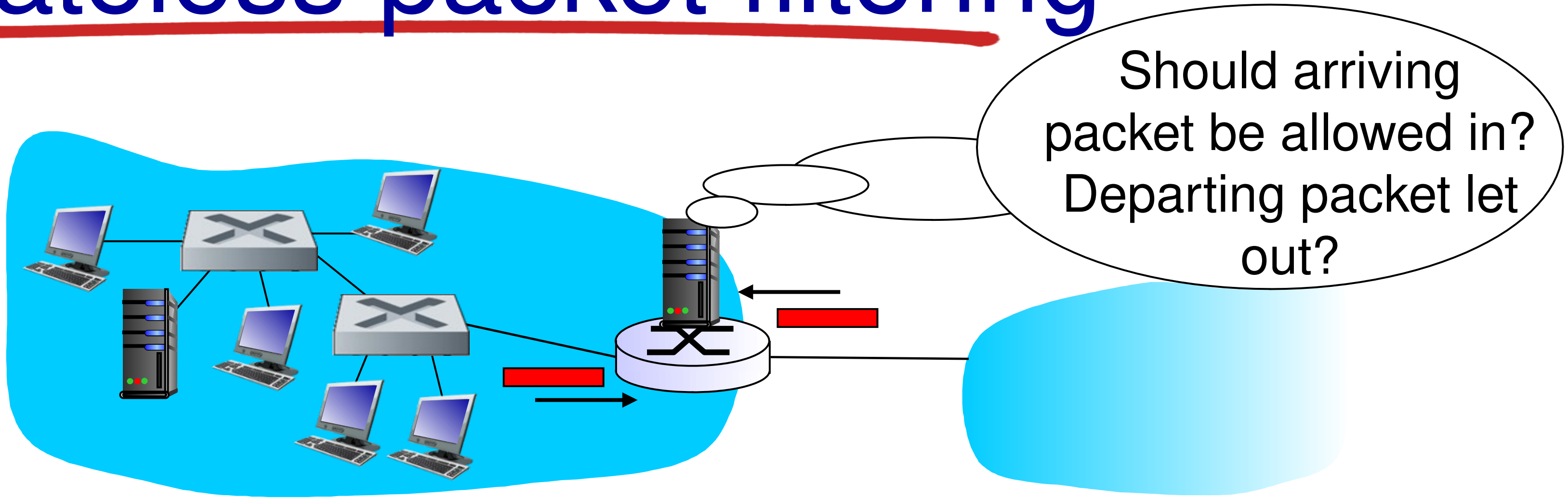
allow only authorized access to inside network

- set of authenticated users/hosts

three types of firewalls:

- stateless packet filters
- stateful packet filters
- application gateways

Stateless packet filtering


Should arriving packet be allowed in? Departing packet let out?

- 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
  - *result:* all incoming, outgoing UDP flows and telnet connections are blocked
- *example 2:* block inbound TCP segments with ACK=0.
  - *result:* prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside.

Stateless packet filtering: more examples

Policy	*Firewall Setting*
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 (e.g. 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	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	outside 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): determine whether incoming, outgoing packets "makes sense"
  - timeout inactive connections at firewall: no longer admit packets

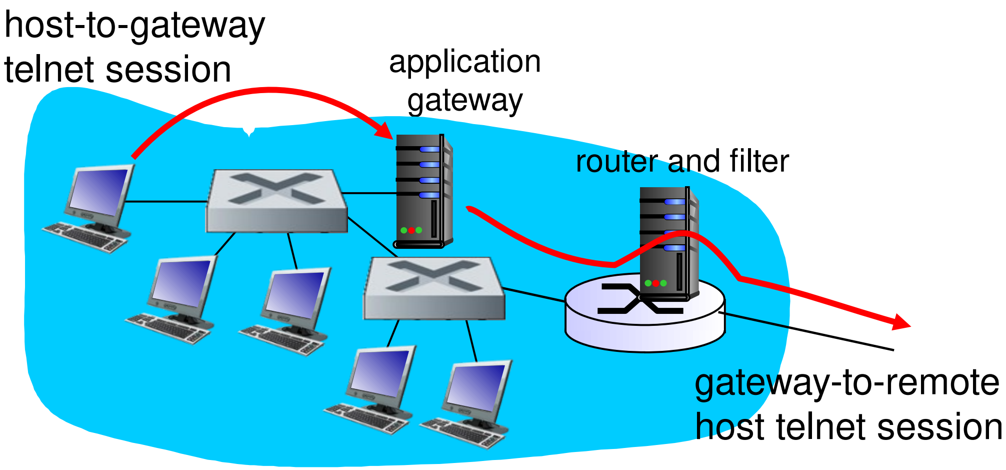
Stateful packet filtering

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

- filter packets on application data as well as on IP/TCP/UDP fields.
- *example:* allow select internal users to telnet outside


host-to-gateway telnet session
application gateway
router and filter
gateway-to-remote host telnet session

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, gateways

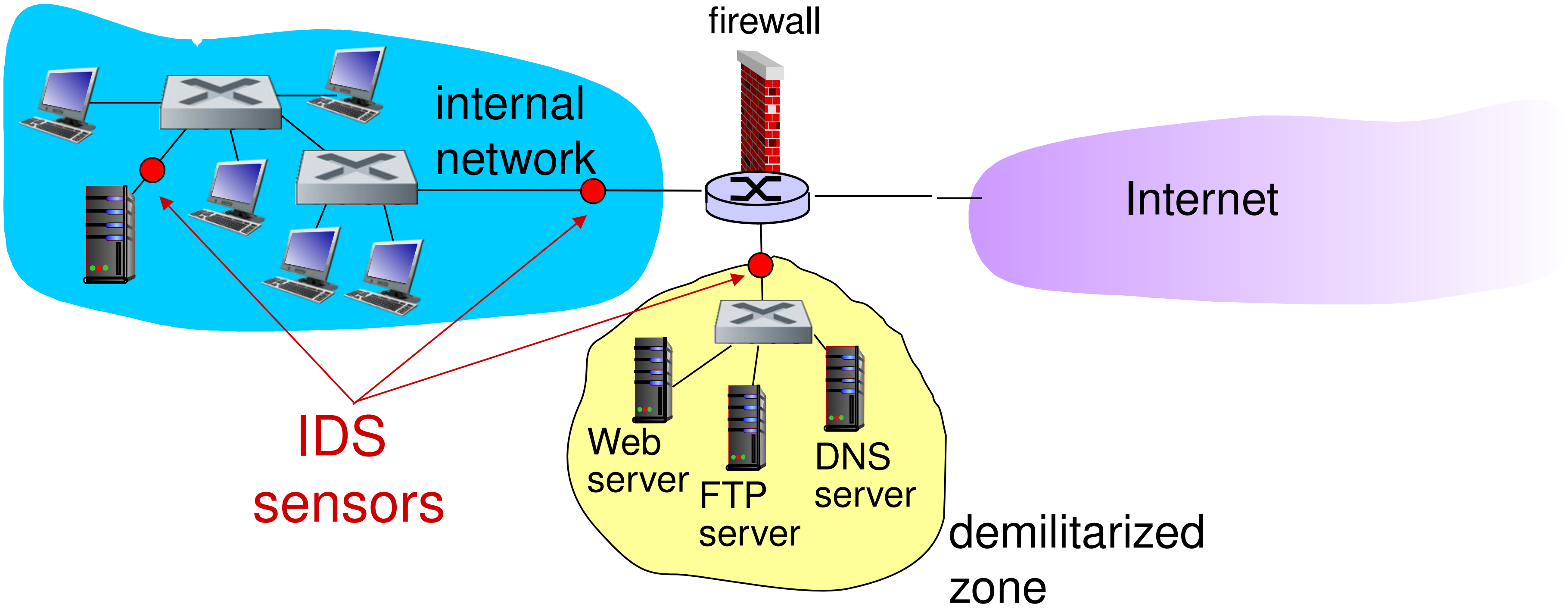
- *IP spoofing:* router can't know if data "really" comes from claimed source
- if multiple app's. need special treatment, each has own app. gateway
- client software must know how to contact gateway.
  - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for 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)
- message integrity
- end-point authentication

…. used in many different security scenarios

- secure email
- secure transport (SSL)
- IP sec
- 802.11

operational security: firewalls and IDS

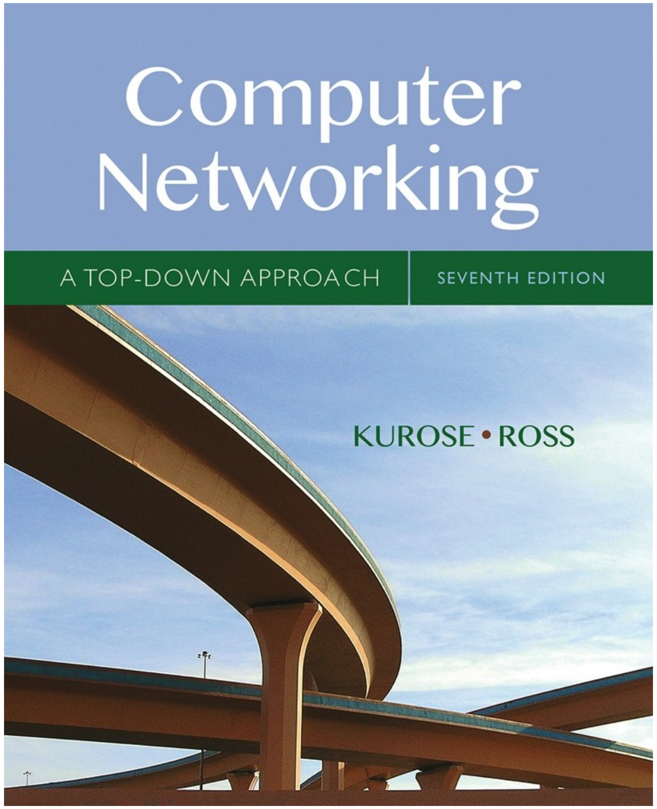
Chapter 8
Security

A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and 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) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides 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



Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016