Introduction to Networking

CAN201 – Lecture 11

Lecturer: Dr. Wenjun Fan

Revisit – Symmetric vs. Asymmetric Crypto

- Q1: What is the main problem of symmetric cryptography?
- Q2: How would we address this problem?
- Q3: Any issue about asymmetric cryptography?
- Q4: How would we solve this problem?

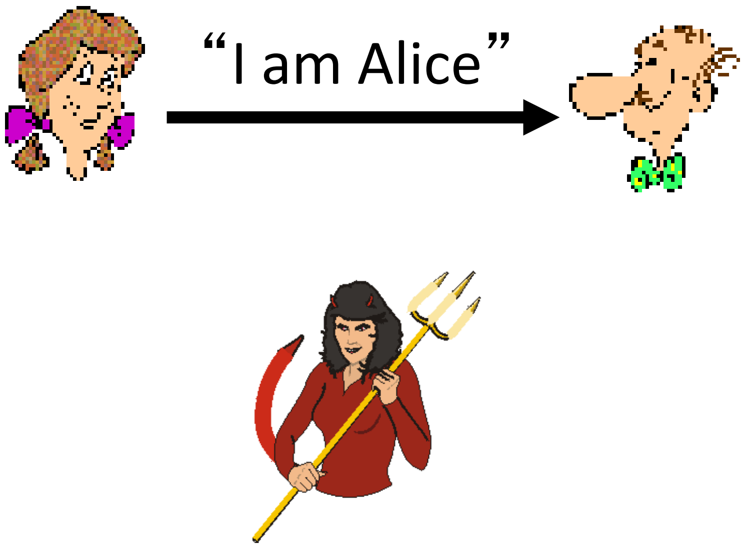
Lecture 11 – Network Security (2)

- **Roadmap**
  1. Authentication and Message integrity,
  2. Securing e-mail
  3. Securing TCP connections: SSL

Authentication

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

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

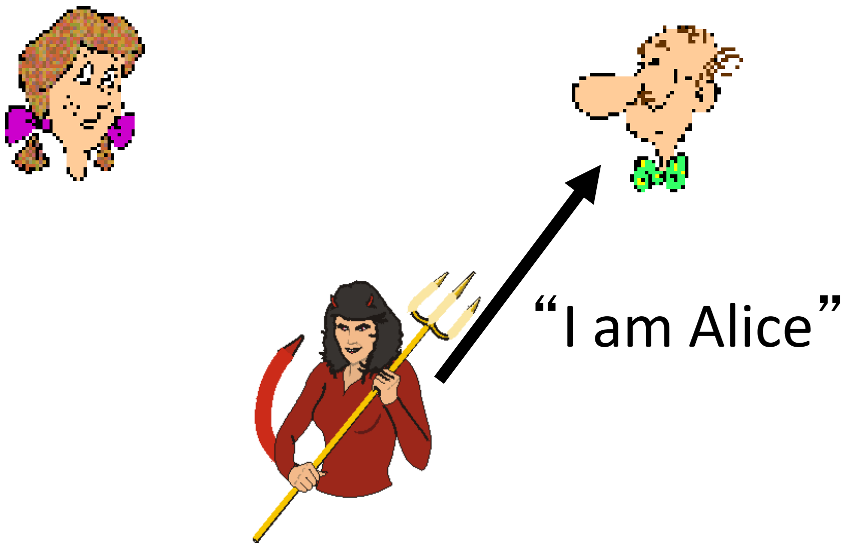


Failure scenario??

Authentication

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

<u>Protocol ap1.0:</u> Alice says "I am Alice"



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

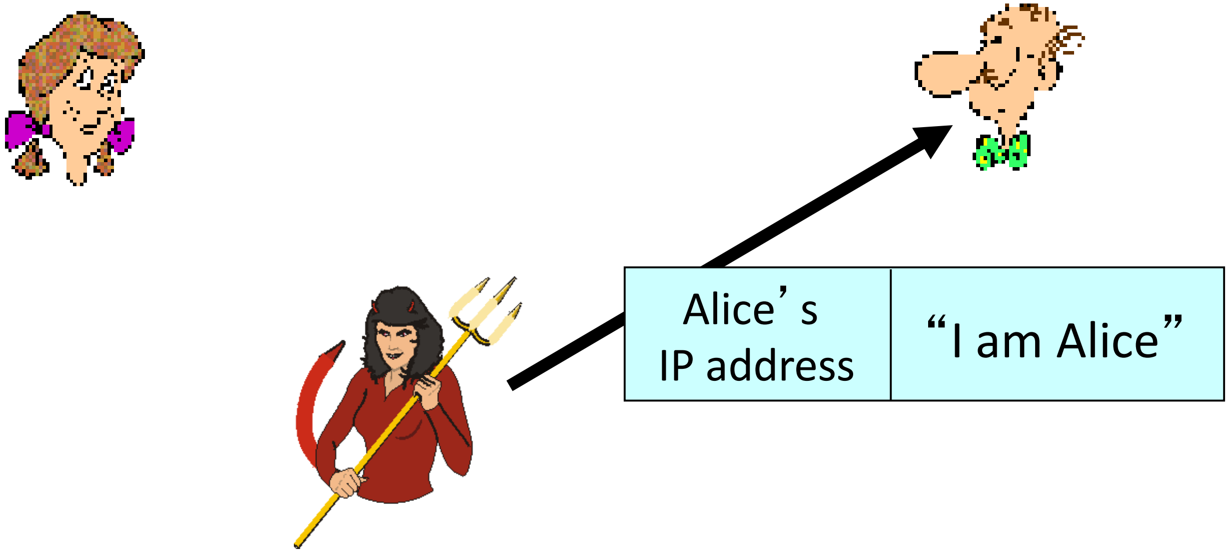
Authentication: another try

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

Failure scenario??

Authentication: another try

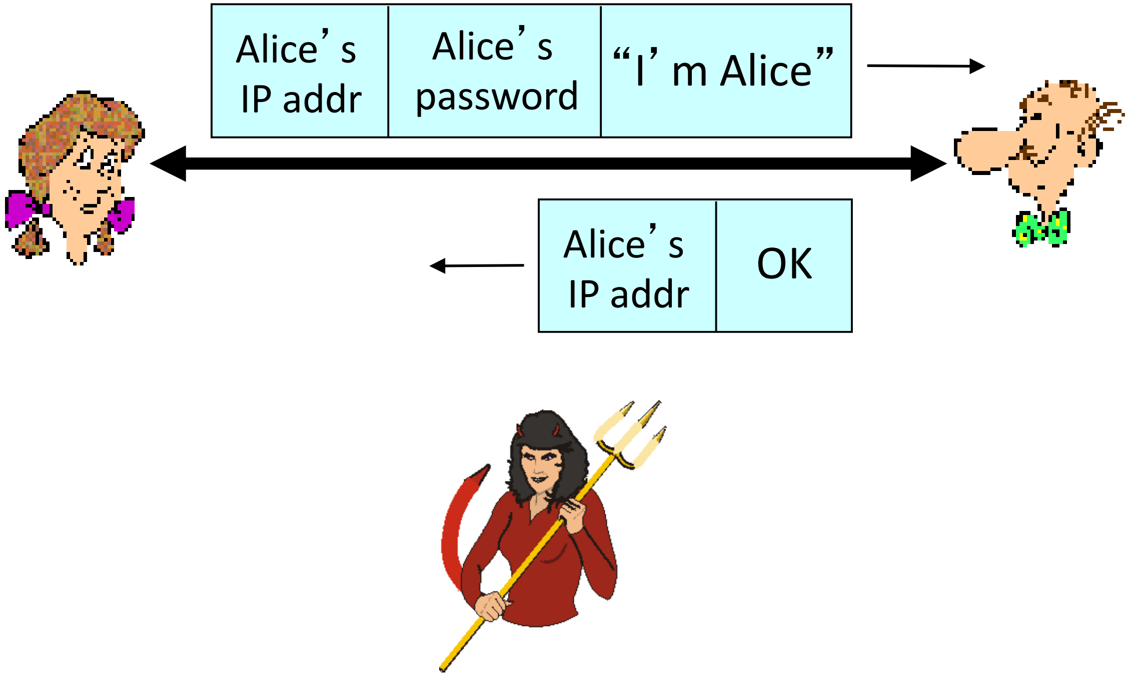
Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address



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

Authentication: another try

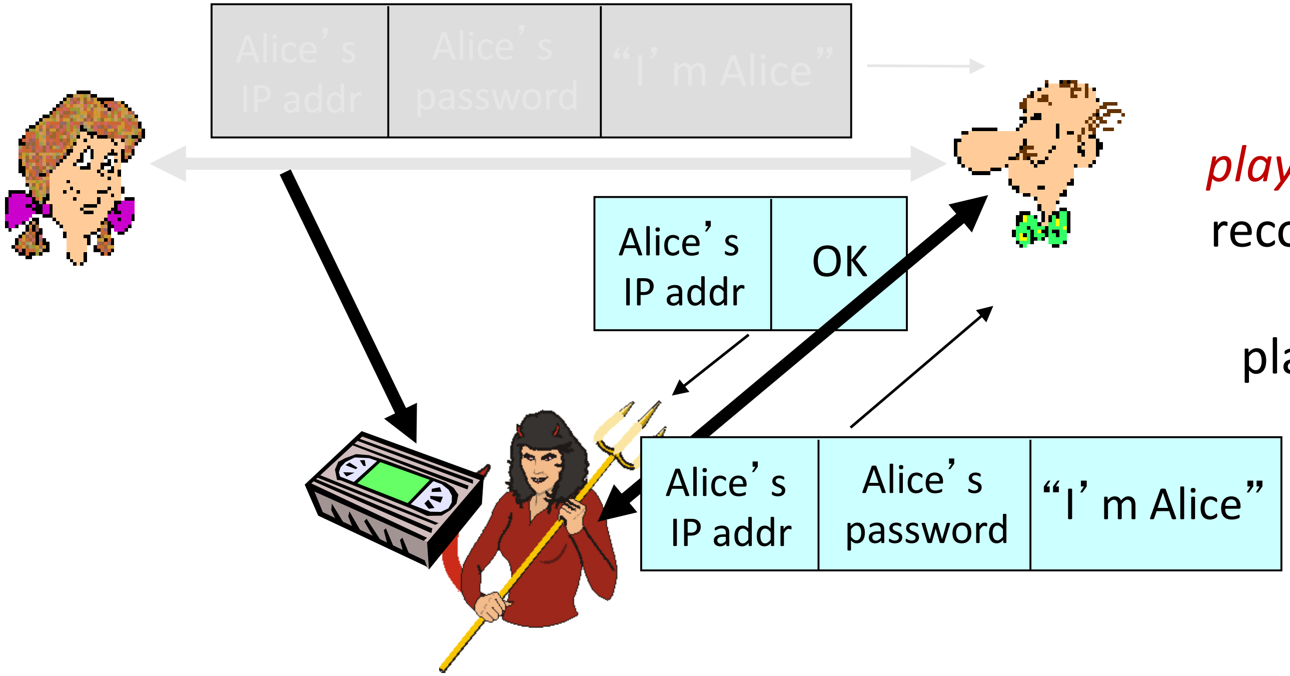
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Failure scenario??

Authentication: another try

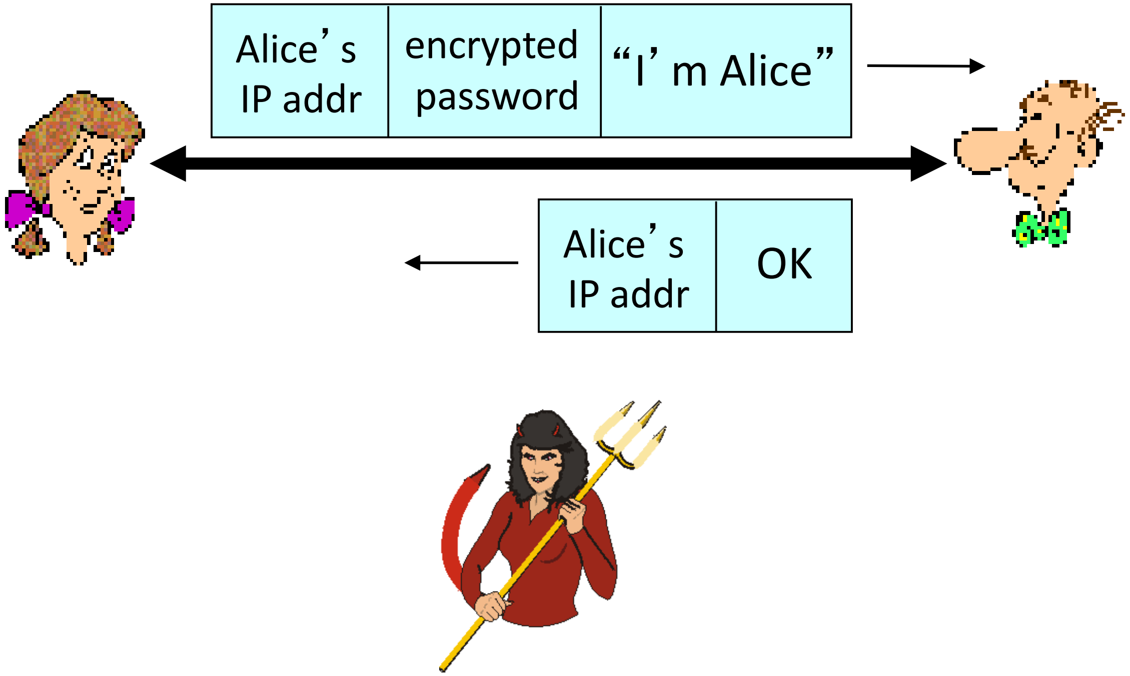
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



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

Authentication: yet another try

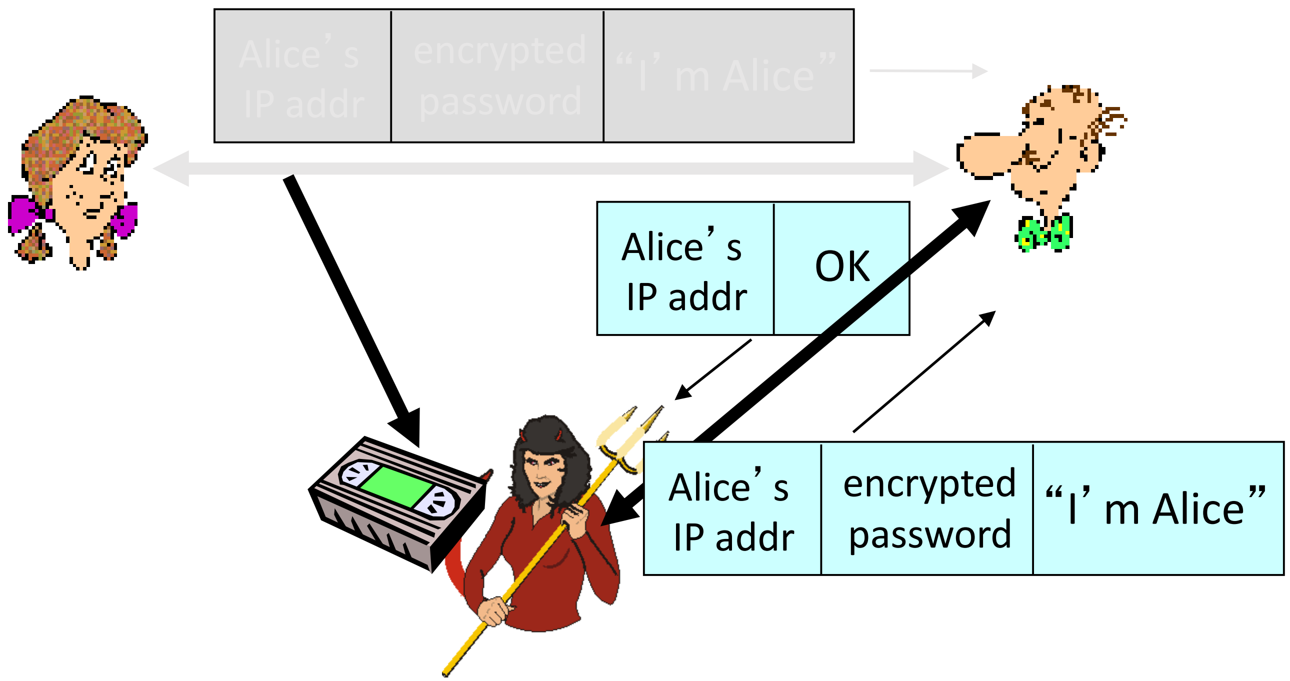
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: yet another try

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



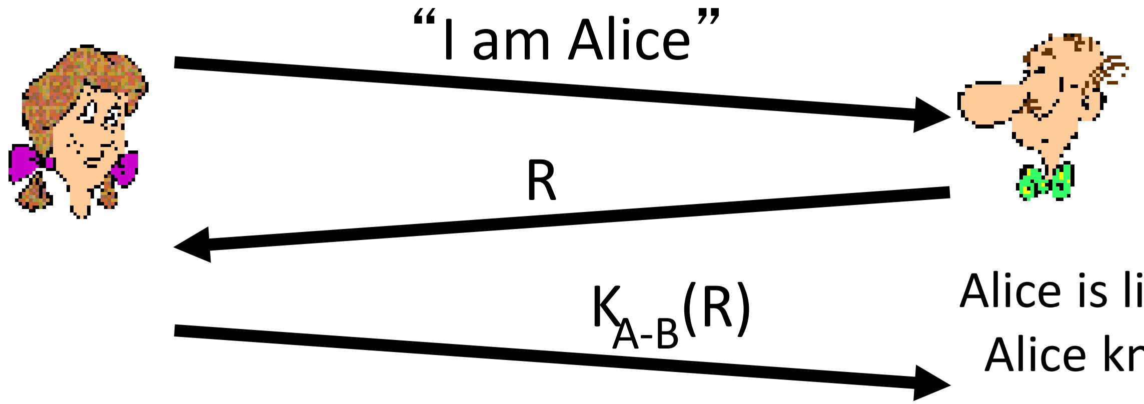
record
and
playback
still works!

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



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

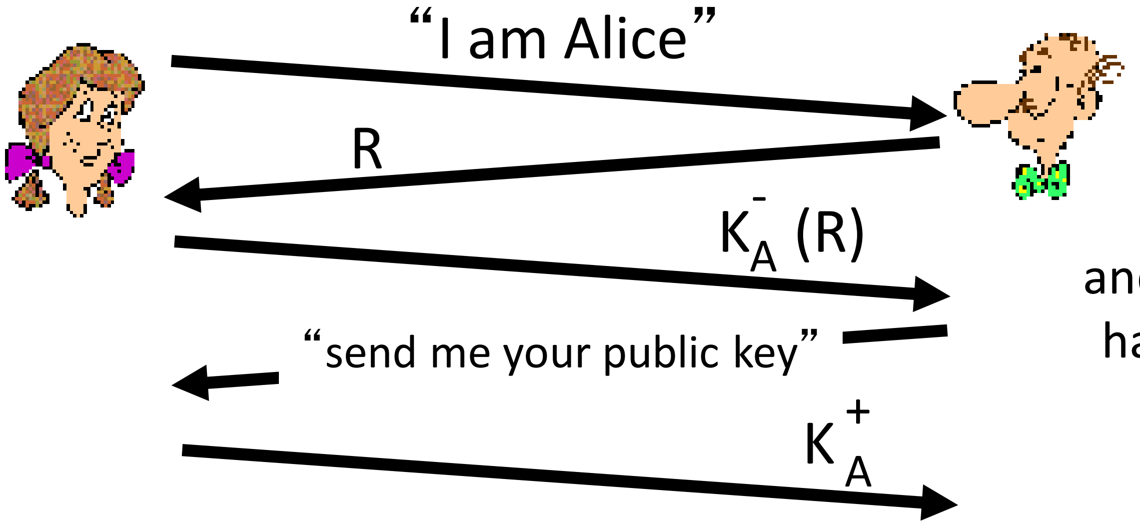
Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



Bob computes

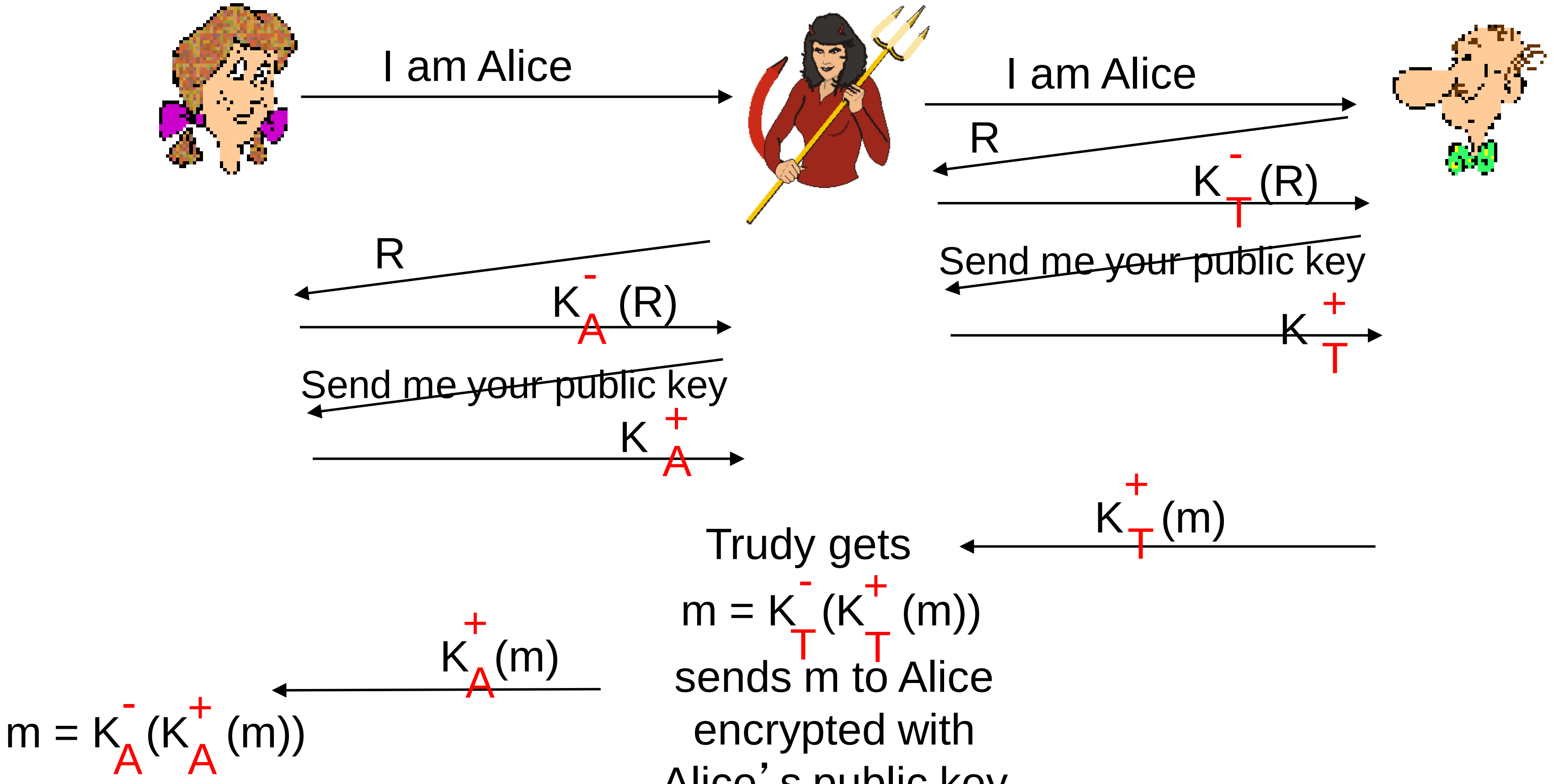
$K_A^+(K_A^-(R)) = R$

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

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

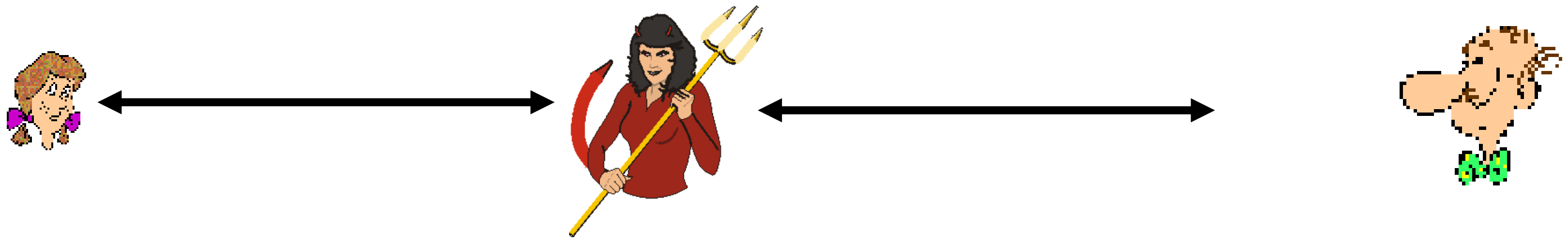
ap5.0: security hole

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



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!

Lecture 11 – Network Security (2)

- **Roadmap**
  1. Authentication and Message integrity
  2. Securing e-mail
  3. Securing TCP connections: SSL

Digital signatures

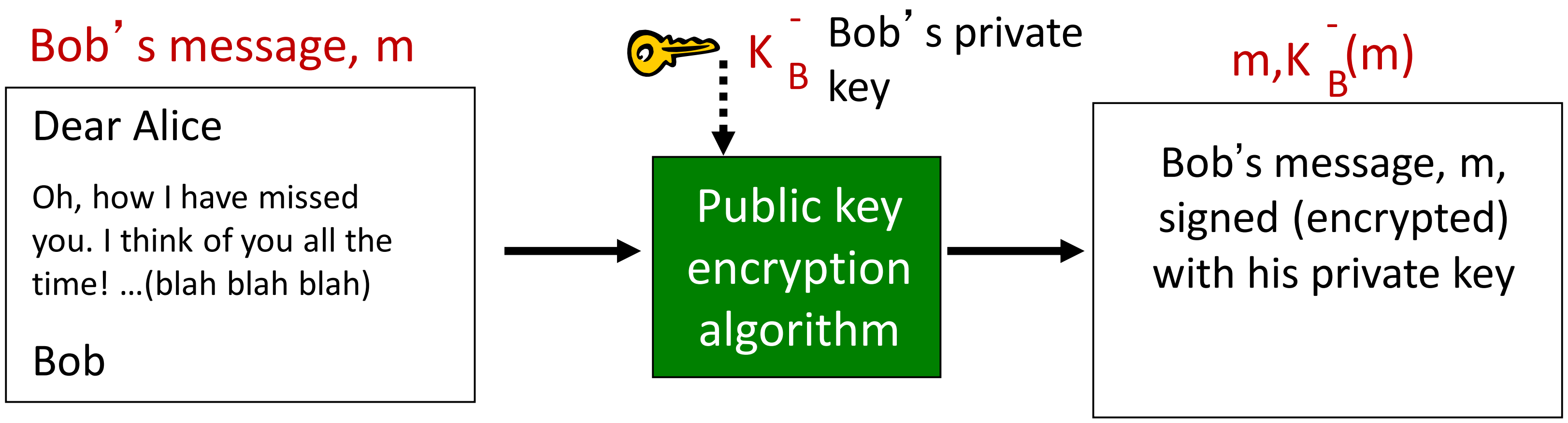
cryptographic technique analogous to hand-written signatures:

- 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, $K_B^-(m)$



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:

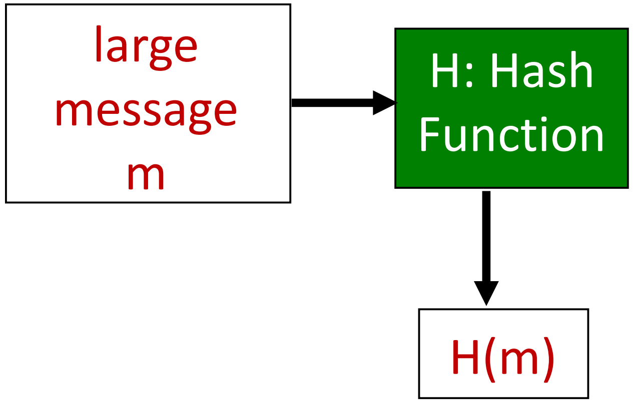
✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

Message digests

computationally expensive to public-key-encrypt long messages

goal: fixed-length, easy-to-compute digital "fingerprint"

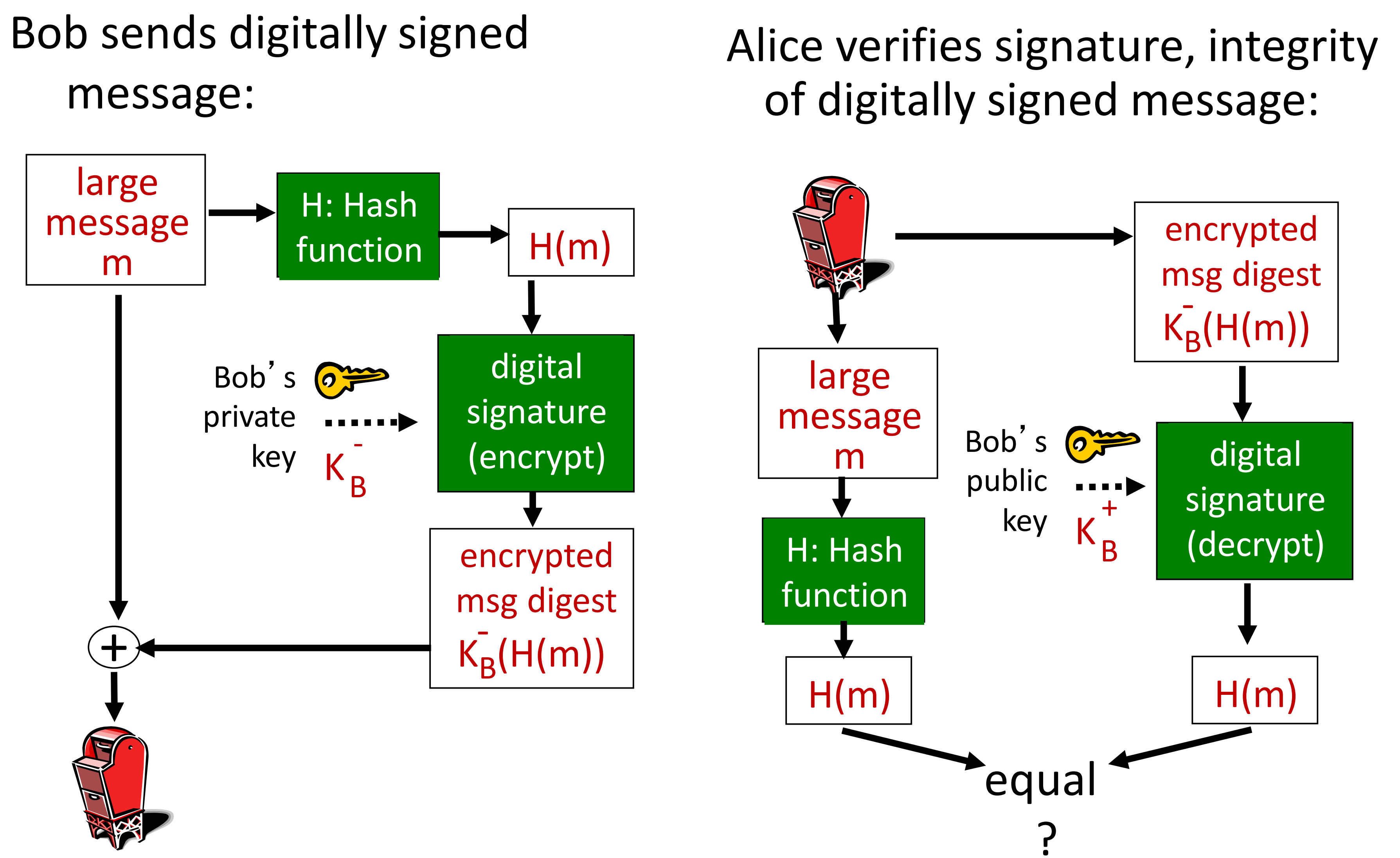
- apply hash function H to *m*, get fixed size message digest, *H(m).*



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



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:

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

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

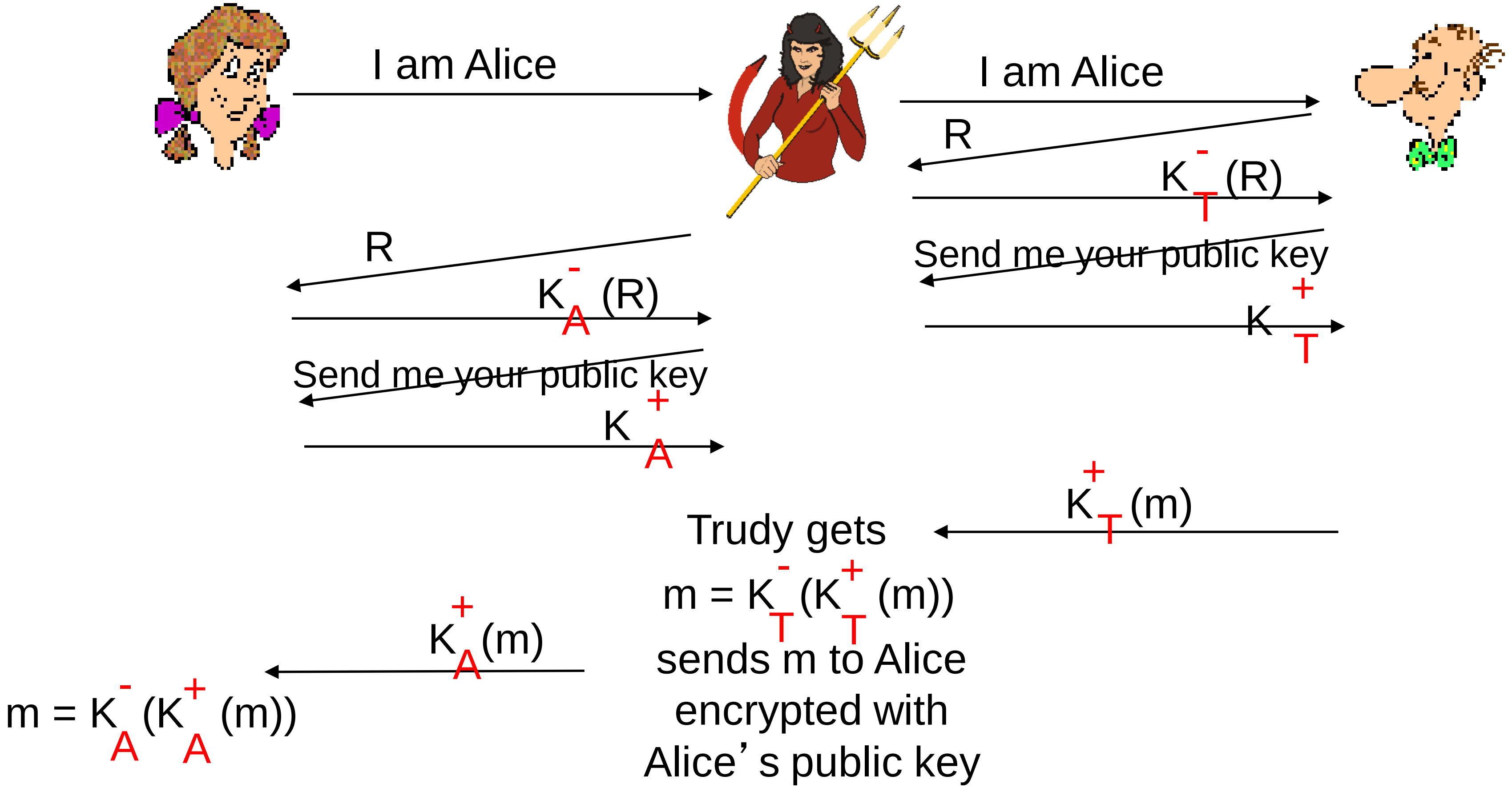
different messages
but identical checksums!

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 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

Recall: ap5.0 security hole

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

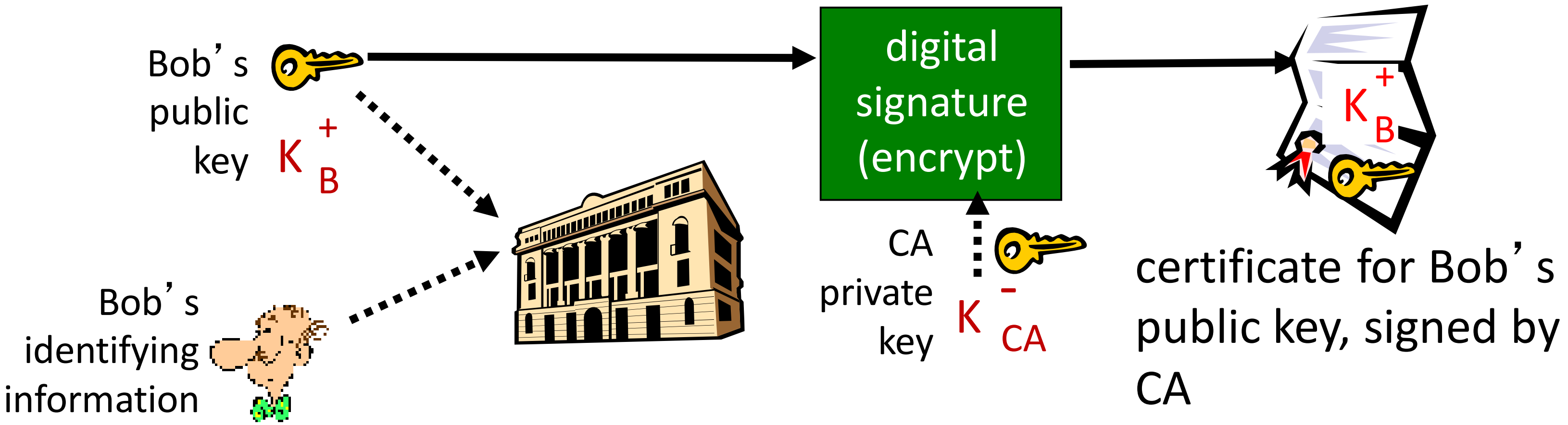


Public-key certification

- **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

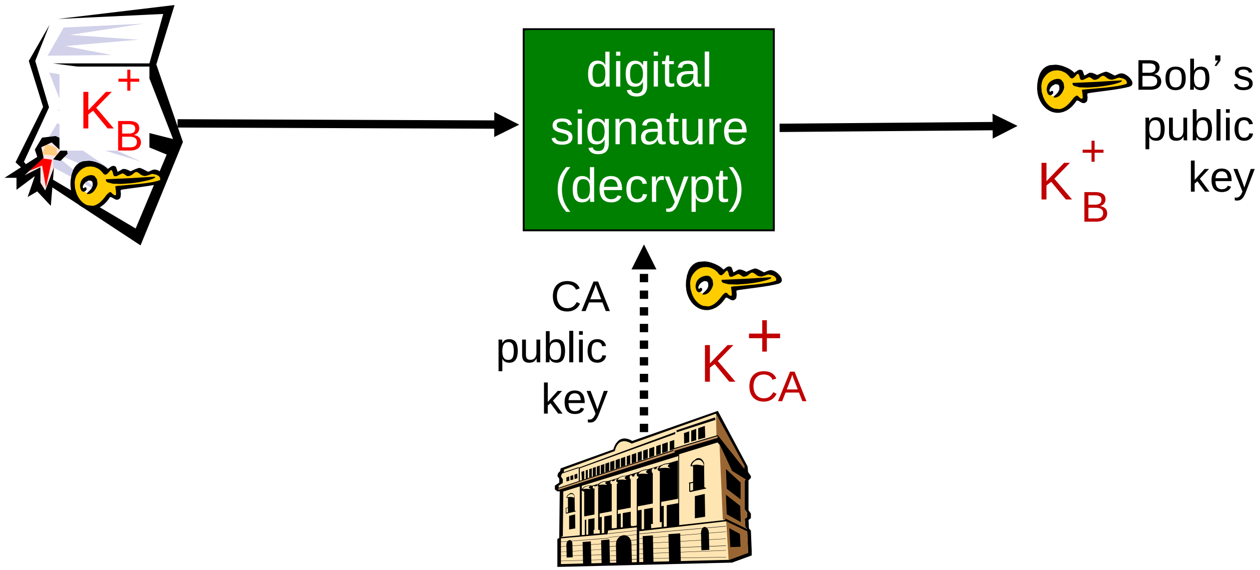
Certification authorities

- *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"



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

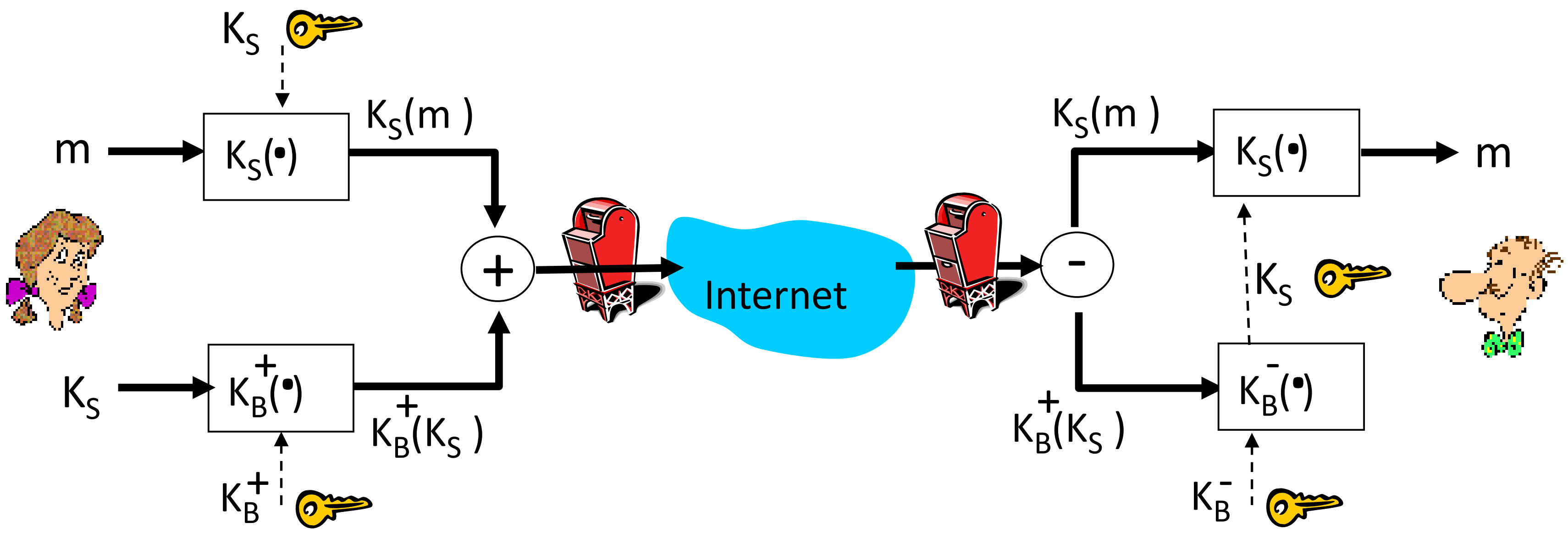


Lecture 11 – Network Security (2)

- **Roadmap**
  1. Message integrity, authentication
  2. Securing e-mail
  3. Securing TCP connections: SSL

Secure e-mail

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

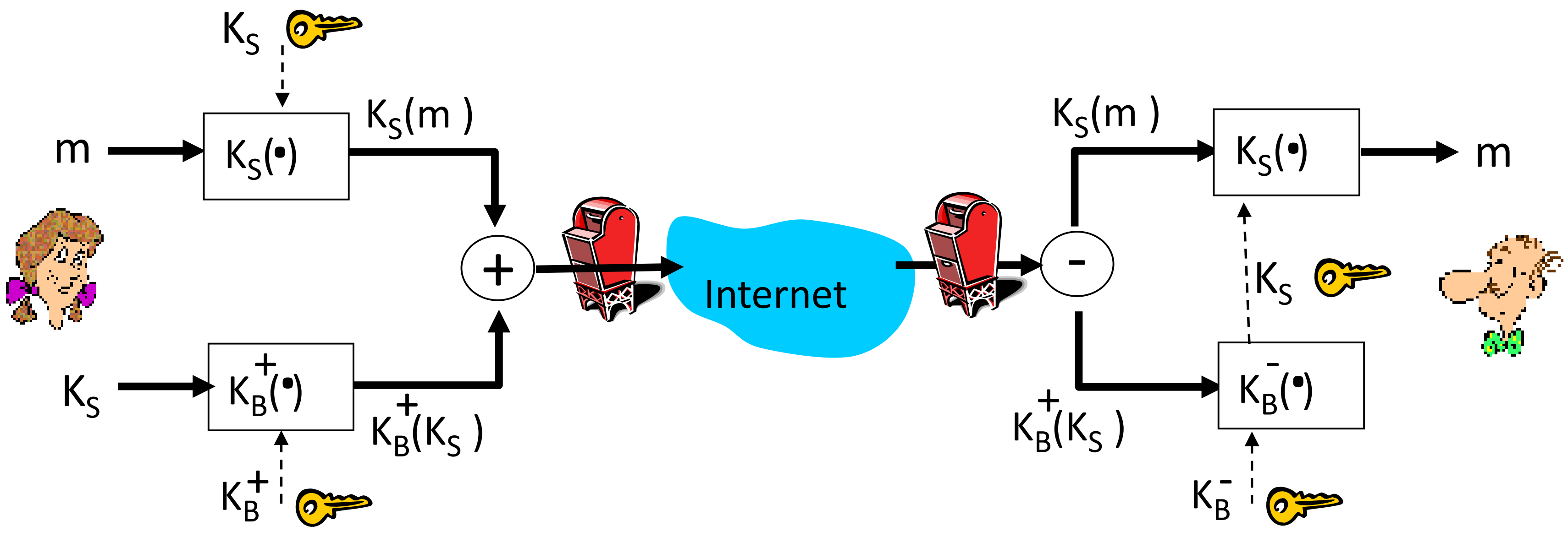


Alice:

- 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.

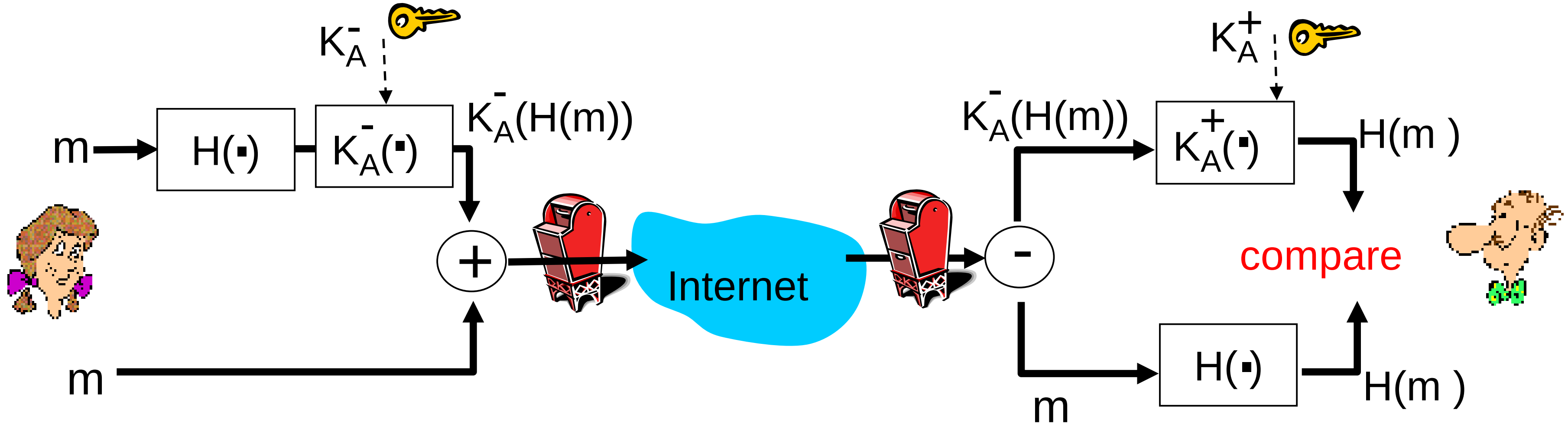


Bob:

- 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)

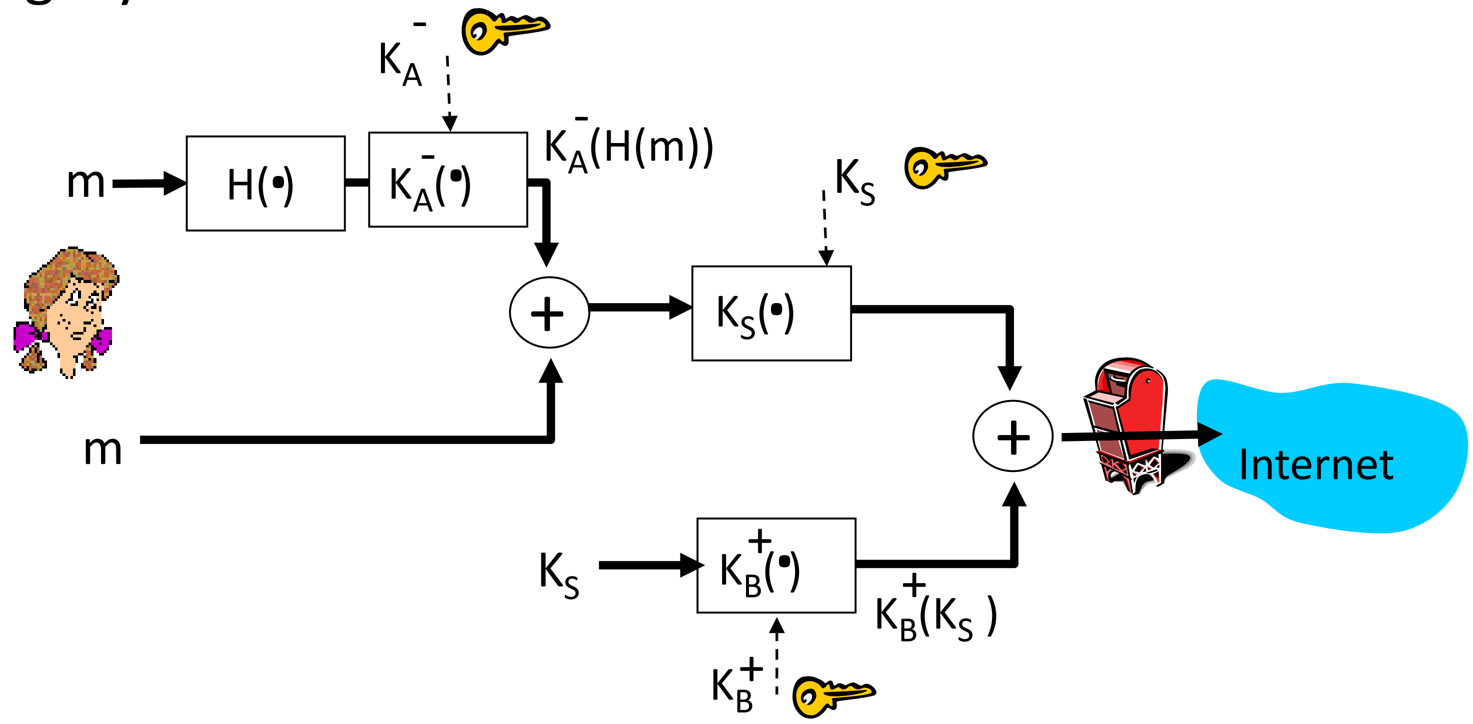
Alice wants to provide sender authentication message integrity



- 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

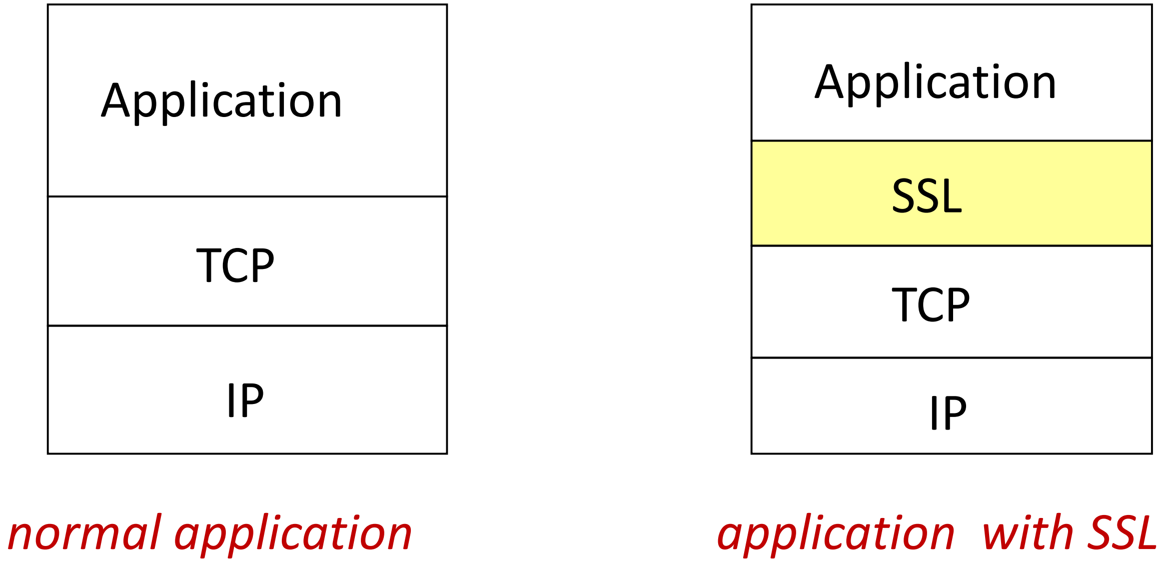
Lecture 11 – Network Security (2)

- **Roadmap**
  1. Message integrity, authentication
  2. Securing e-mail
  3. Securing TCP connections: SSL

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

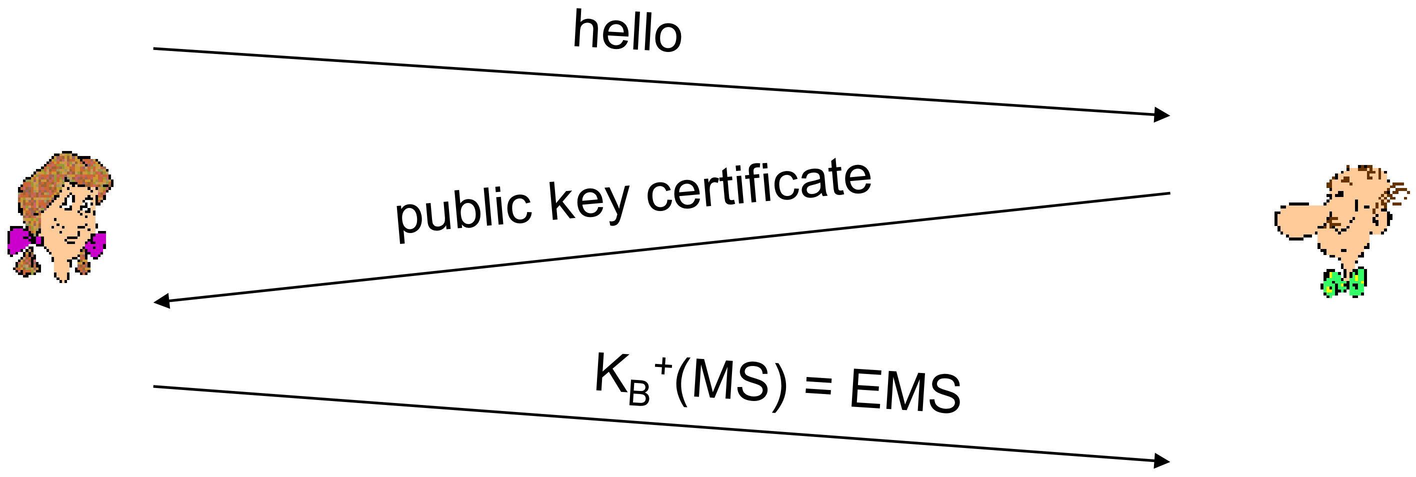
Toy SSL: a simple secure channel

Four phases:

- ***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: handshake

Alice needs to
(a) establish a TCP connection with Bob,
(b) verify that Bob is *really* Bob,
(c) send Bob a master secret key



MS: master secret

EMS: encrypted master secret

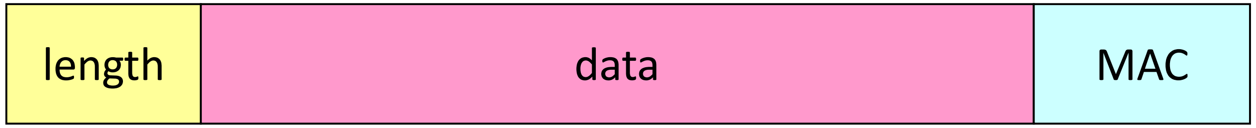
Toy: key derivation

Readers should note that the MAC here (standing for “message authentication code”) is not the same MAC used in link-layer protocols (standing for “medium access control”)!

- **considered bad to use same key (the master secret Key) for more than one cryptographic operation**
  - use different keys for message authentication code (MAC) and encryption
  - MAC = H(m+s), m:= message; s:= MAC key
- **four keys generated from the MS:**
  - 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 (MS) and (possibly) some additional random data and creates the keys

Toy: data transfer – in 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



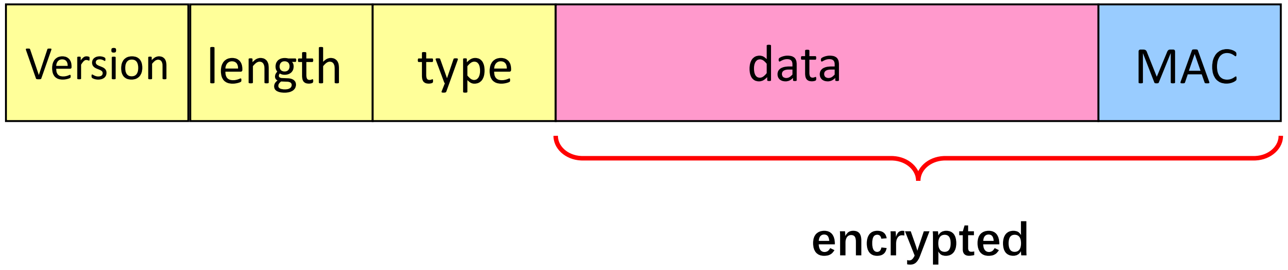
Toy: data transfer - sequence numbers

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

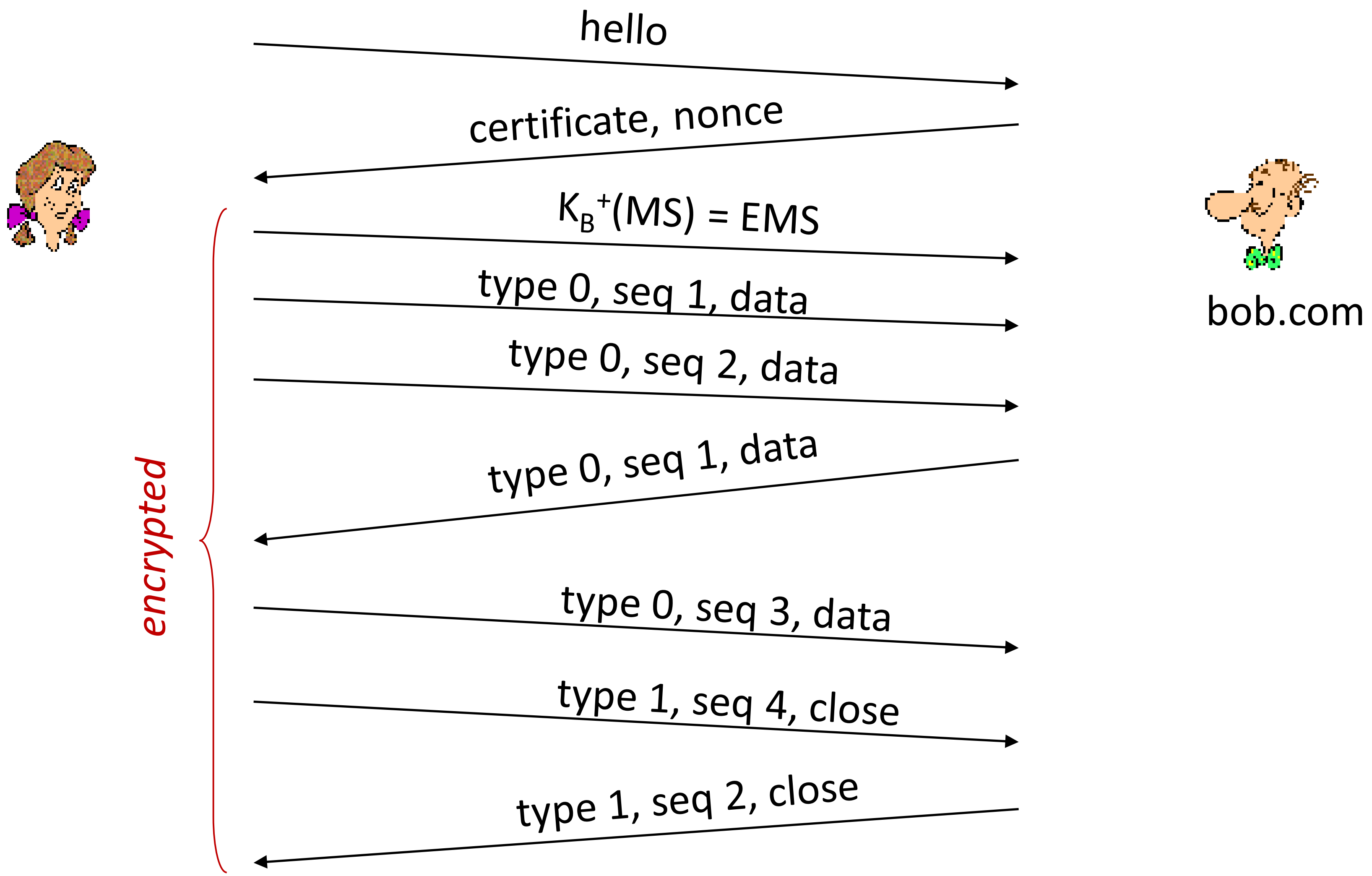
- ***problem:*** **attacker could replay all records**
- ***solution:*** **use nonce**

Toy: connection closure

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



Toy SSL: summary



Toy SSL isn't complete

- **how long are the fields?**
- **which encryption protocols?**
- **client and server 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: handshake (3)

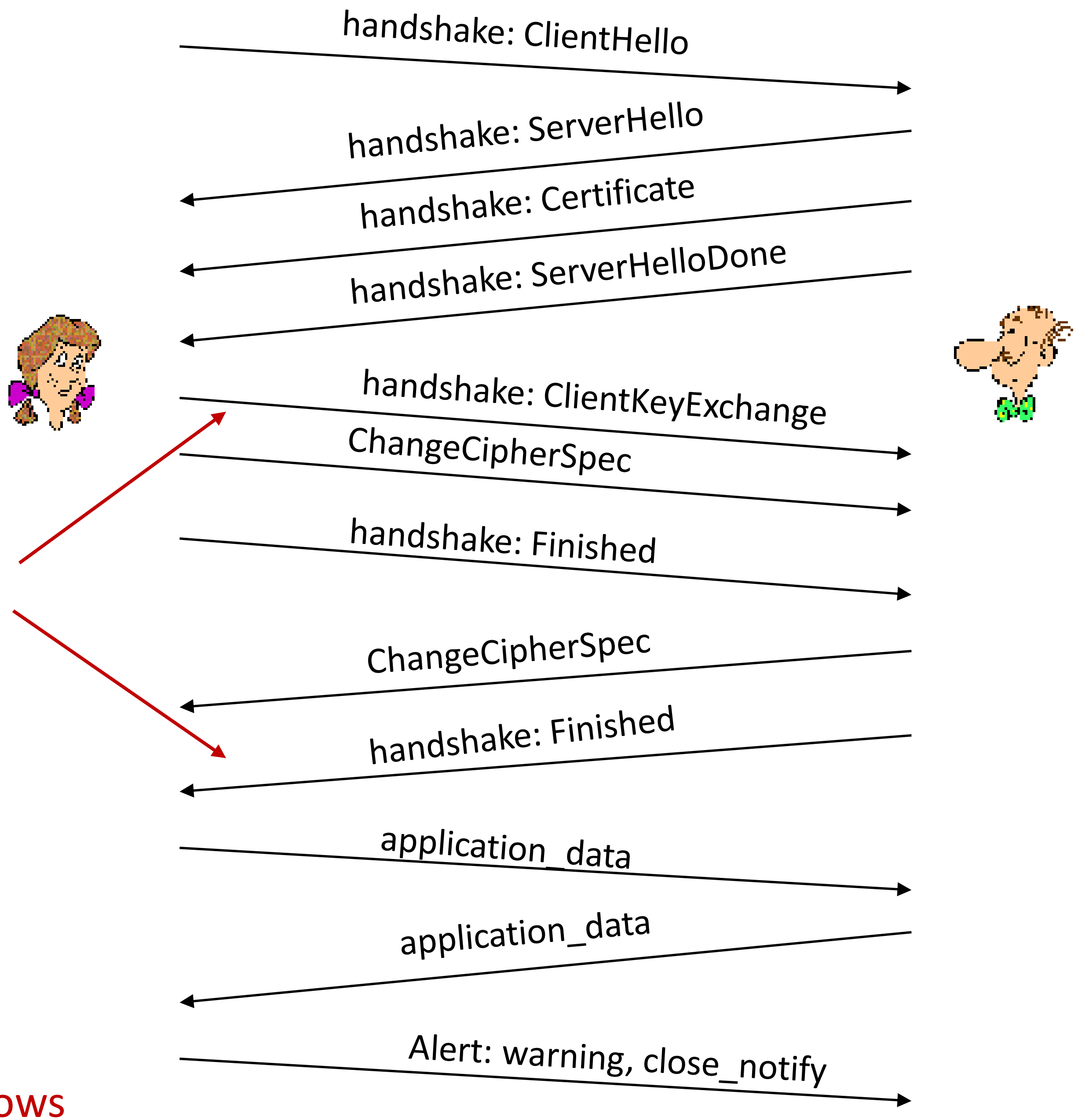
last 2 steps protect handshake from tampering

- **In step 1, client typically offers range of algorithms, in plain-text, some strong, some weak**
- **man-in-the middle could delete stronger algorithms from list**
- **last 2 steps (step 5 and 6) prevent this**
  - last two messages are encrypted

Real SSL: handshake (4)

- **why two random nonces, in step 1 and 2 respectively?**
- **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

Real SSL connection



everything henceforth is encrypted

TCP FIN follows

Thanks.

- **Addresses**
  - Email: Wenjun.Fan@xjtlu.edu.cn
  - Office: EE 214
- **Office hours**
  - Monday: 12:00 – 13:00
  - Tuesday: 12:00 – 13:00