Hash, authentication , and TLS (Transport Layer Security)

CE 352, Computer Networks

Salem Al-Agtash

Lecture 25

Slides are adapted from Computer Networking: A Top Down Approach, 7th Edition © J.F Kurose and K.W. Ross

Recap (network security)

- *Security:* well-being of information and infrastructures and rests on:
  - *confidentiality*: only sender, intended receiver should "understand" message contents
    - sender encrypts message and receiver decrypts message
  - *message integrity:* sender, receiver want to ensure message not altered (in transit, or afterwards) without detection
  - *authentication:* sender, receiver want to confirm identity of each other
  - *access and availability*: services must be accessible and available to users

Recap (to cover)

Cryptography

- Secret key algorithms: DES/AES
- Public key algorithms: RSA
- One-way hash functions and message integrity: MD5, SHA2

End-point authentication, access control, public key infrastructure, digital signature

Securing the Internet

- Application layer security: Securing email
- Transport layer security: Securing TCP connection - SSL
- Network layer security: IPsec and VPN
- Data link layer security: Wireless LAN

Recap (Simple encryption scheme)

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

```
plaintext:   abcdefghijklmnopqrstuvwxyz

ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

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

Caesar Cipher: Mathematically give each letter a number

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Then have Caesar cipher as:

$$C = \mathrm{E}(p) = (p + k) \bmod (26)$$

$$p = \mathrm{D}(C) = (C - k) \bmod (26)$$

Total of 26! = 4 x 10^{26} keys , Secure?

Problem is language characteristics

Human languages are **redundant**

Letters are not equally commonly used

More sophisticated encryption: transposition and product

Recap(Symmetric key crypto: DES)

DES operation

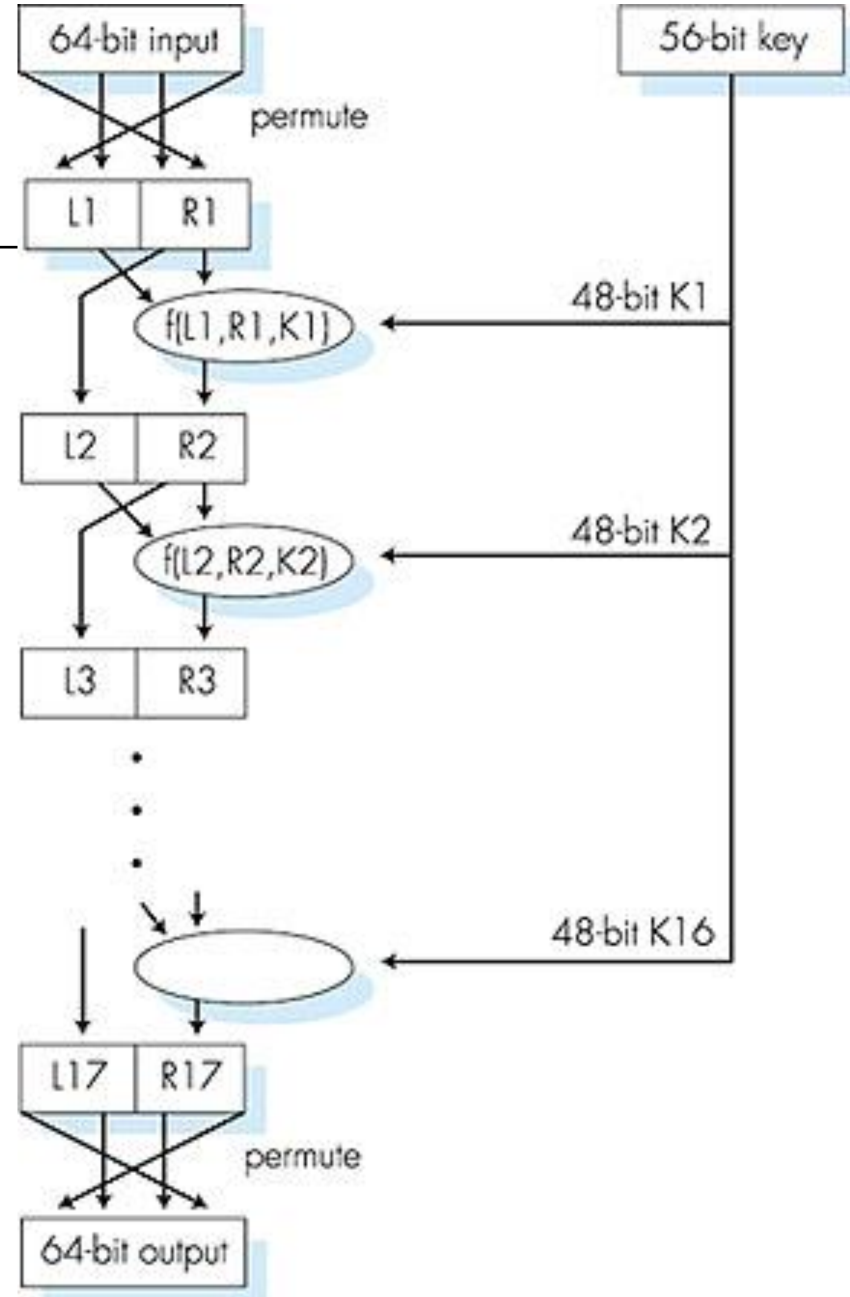
initial permutation

16 identical "rounds" of function (Mangler) application, each using different 48 bits of key final permutation

Brute force search attack possible

Replacement:

- 3DES, but performance issues
- AES:
  - processes data in 128 bit blocks (DES: 64 bits data block)
  - 128, 192, or 256 bit keys (DES: 56 and 3DES: 168 bits)



Recap (Public key encryption algorithms)

requirements:

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

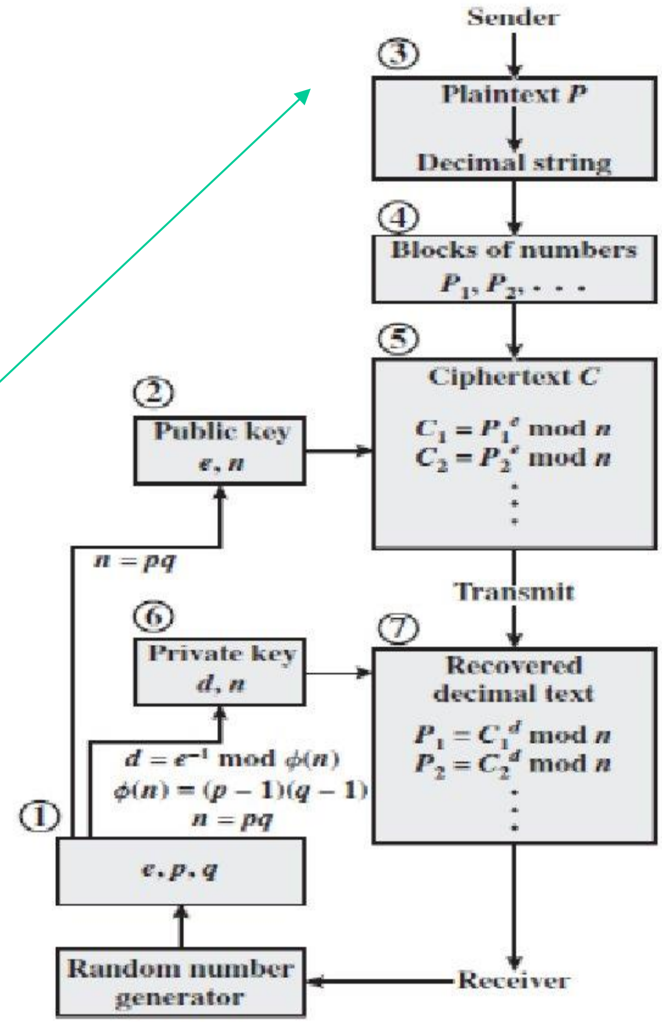
2 given public key K_B^+, it should be impossible to compute private key K_B^-

Important characteristics:

- computationally infeasible to find decryption key knowing only algorithm & encryption key
- computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
- either of the two related keys can be used for encryption, with the other used for decryption (in some schemes)

Recap (Security of Public Key Schemes)

- brute force *exhaustive search* attack is always theoretically possible
- but keys used are too large (>512bits)
- security relies on a *large enough* difference in difficulty between easy (en/decrypt) and *hard* (cryptanalyse) problems
- more generally the *hard* problem is known, but is made hard enough to be impractical to break
- requires the use of *very large numbers*
- hence is *slow* compared to other schemes
- RSA:
  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 $\underbrace{(n,d)}_{K_B^-}$.



Example

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 $\underbrace{(n,d)}_{K_B^-}$.

Consider RSA with p=3 and q=11.

What are n and z?

n = p*q = 33, z = (p-1)(q-1) = 20

Let e be 9.

e = 9 is less than n and has no common factors with z.

Find d such that ed – 1, divisible by z --> (k20+1)/e

with k=4, d = 9

K+ (33, 9)

K- (33, 9)

Encode the word "dog" by encrypting each letter separately.

d = 4, o = 15 and g = 7.

letter	m	m**e	ciphertext = m**e mod 33
d	4	262144	25
o	15	38443359375	3
g	7	40353607	19

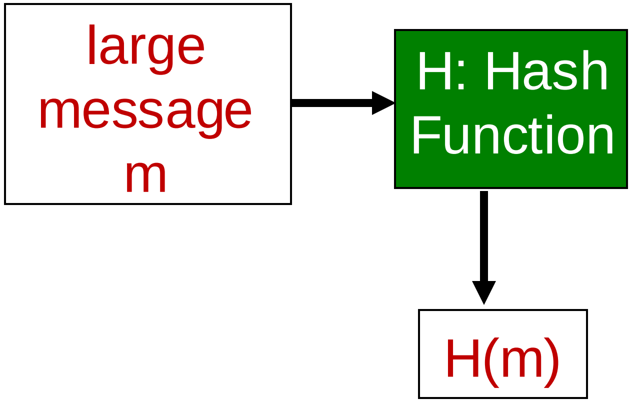
ciphertext	c**d	m = c**d mod n	letter
25	3814697265625	4	d
3	19683	15	o
19	322687697779	7	g

Hashes and 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 is also called message digest



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)

Hash function algorithms

Maps a message M as a "message digest" X = H(M) of constant length, e.g. 128, 160, or 256 bits.

- Well-known examples: MD5, SHA-1, RIPEMD-160, SHA-256.

MD5 hash function widely used (RFC 1321)

- computes 128-bit message digest in 4-step process.

SHA-1 is also used

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

Security requirement

- One-way. Given a message digest X, it is should be "hard" to find a message M satisfying X = H(M).
- Collision resistance. It should be "hard" to find two messages $M1 \neq M2$ such that $H(M1) = H(M2)$.

Properties of a hash function

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. H(m) is relatively easy to compute for any given m, making both hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find m such that H(m) = x → one-way property.
5. For any given block m, it is computationally infeasible to find n NE m such that H(n) = H(m). → weak collision resistance.
6. It is computationally infeasible to find pair m,n such that H(m) = H(n). → strong collision resistance

Digital signatures

cryptographic technique analogous to hand-written signatures:

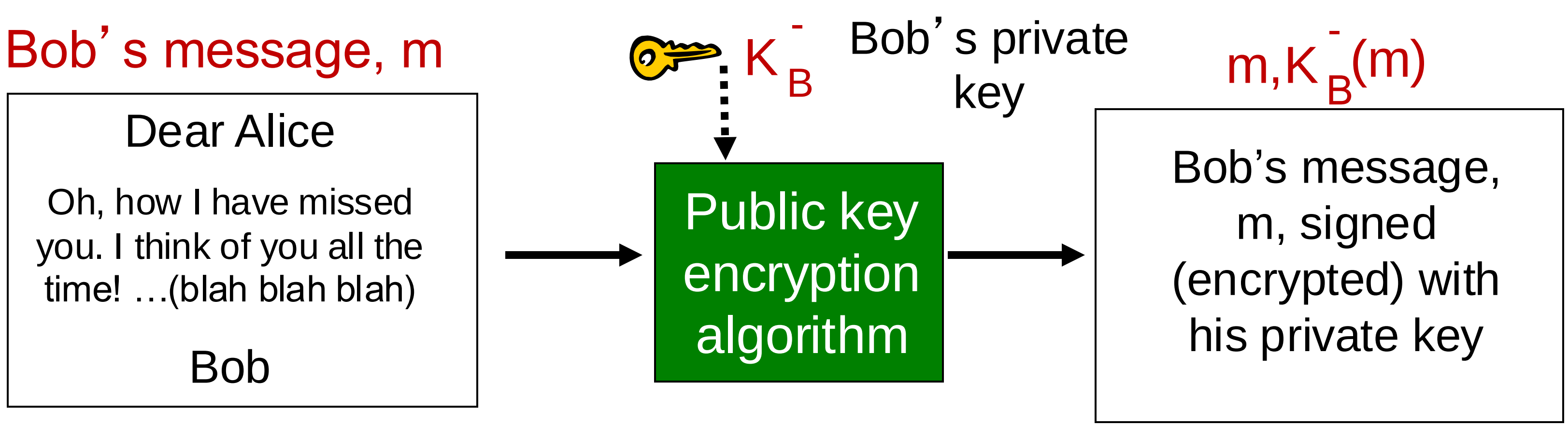
sender (Bob) digitally signs document, establishing he is document owner/creator.

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'

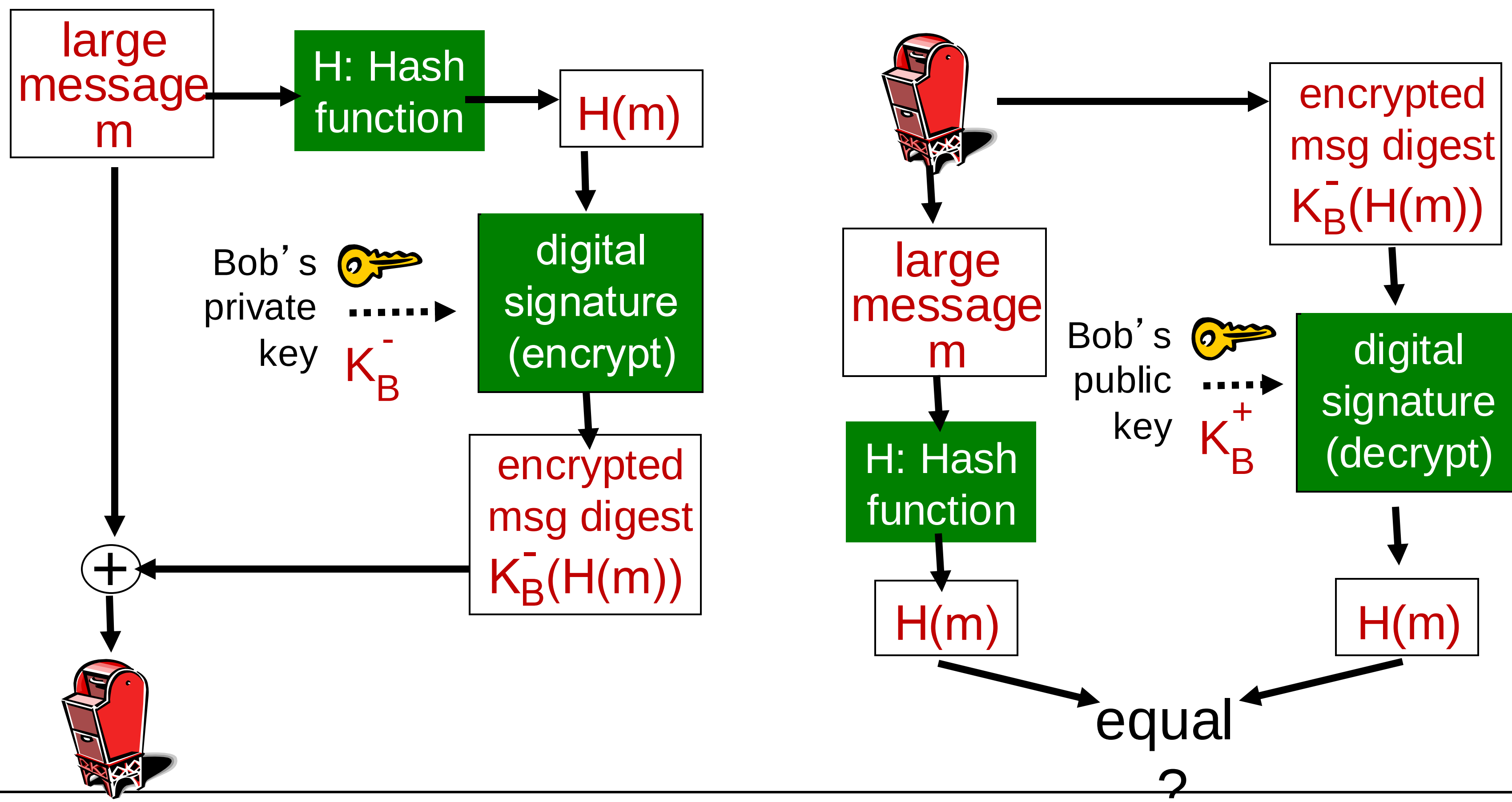
non-repudiation:

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

Digital signature = signed message digest

Bob sends digitally signed message:

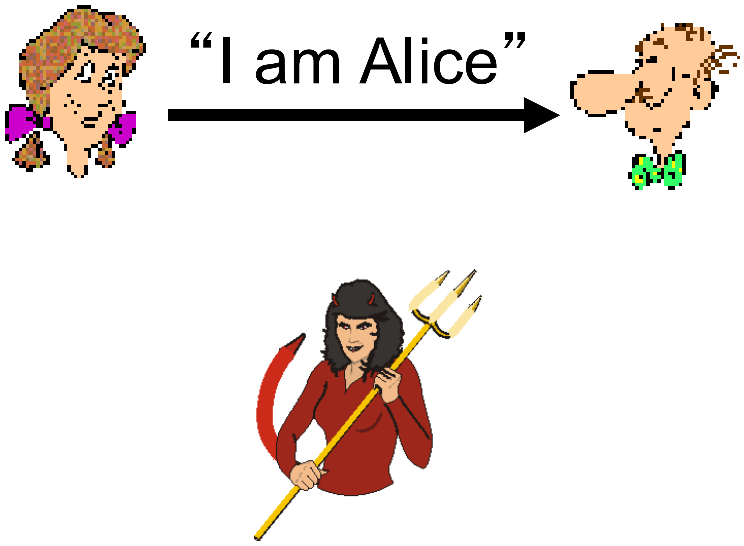
Alice verifies signature, integrity of digitally signed message:



Authentication

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

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

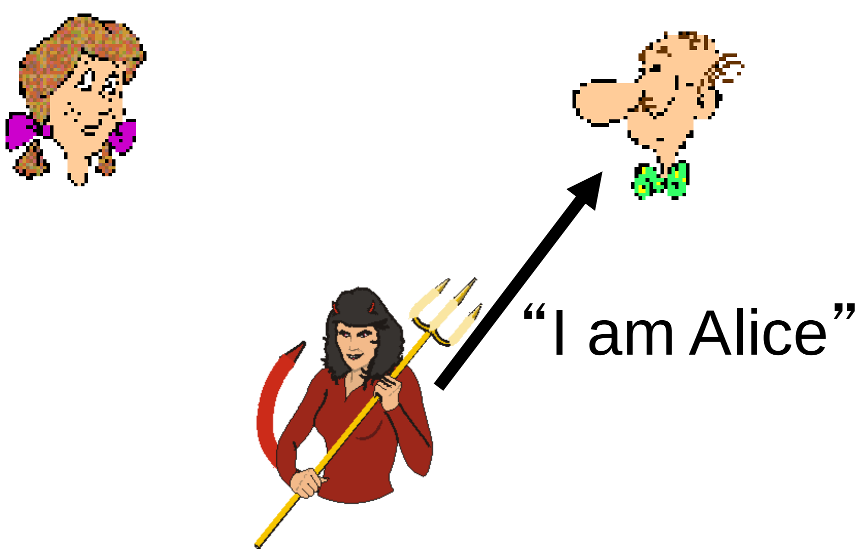


Failure scenario??

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

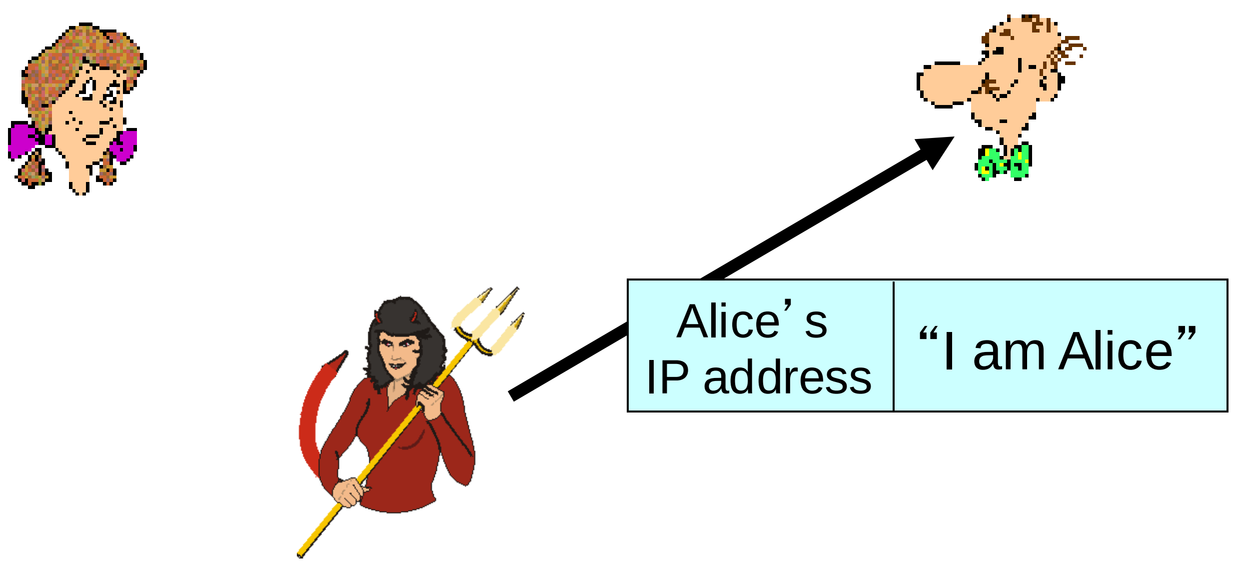
Authentication: another try

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

Failure scenario??

Authentication: another try

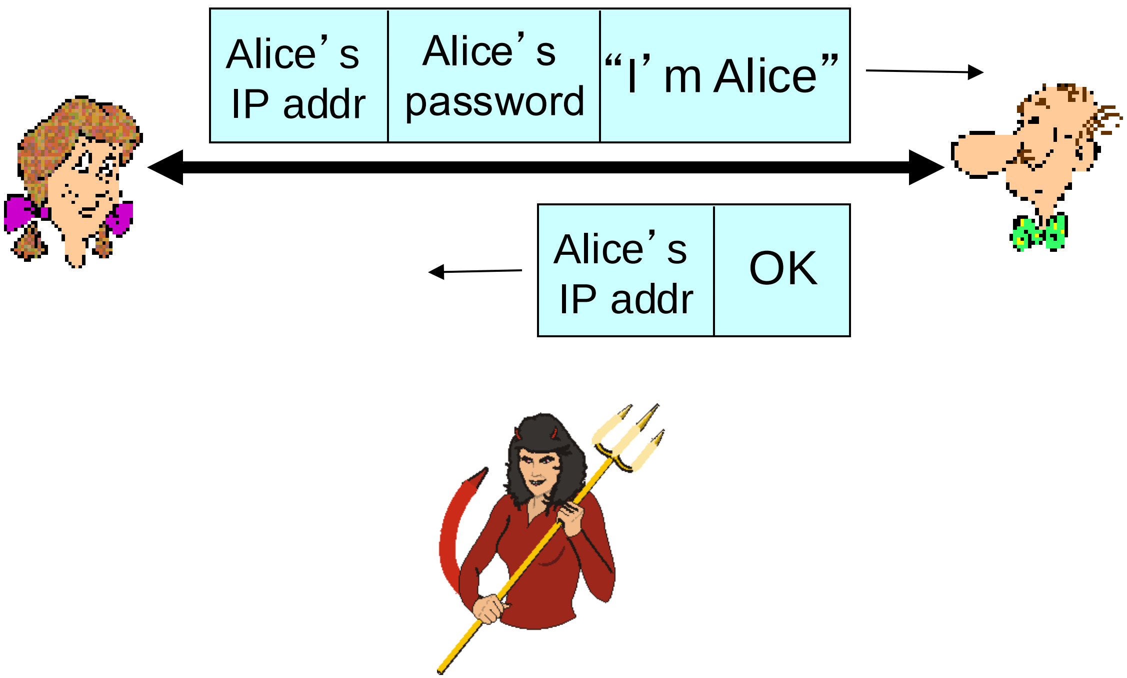
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



Trudy can create a packet "spoofing" Alice's address

Authentication: another try

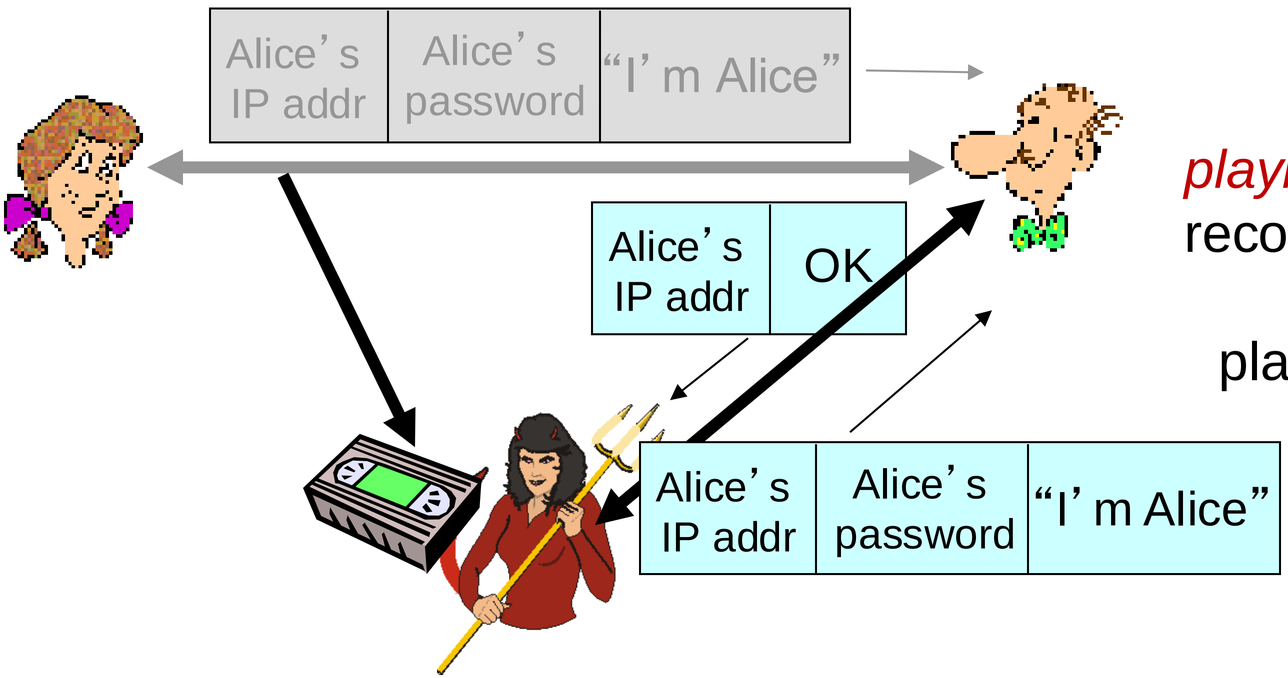
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Failure scenario??

Authentication: another try

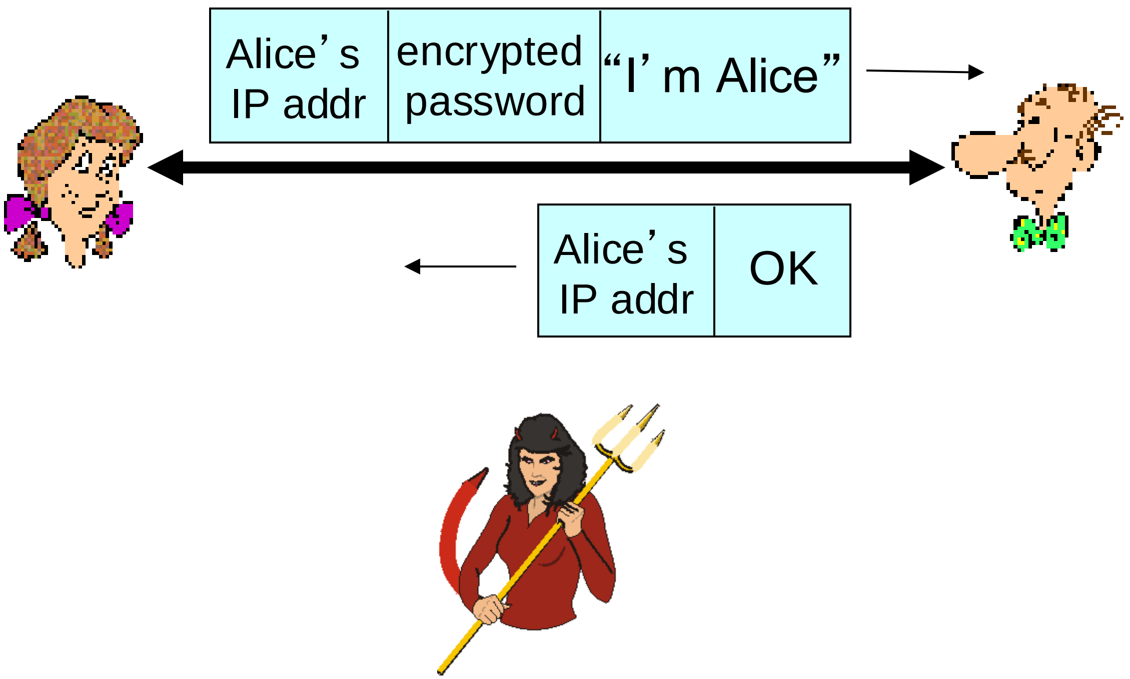
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



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

Authentication: yet another try

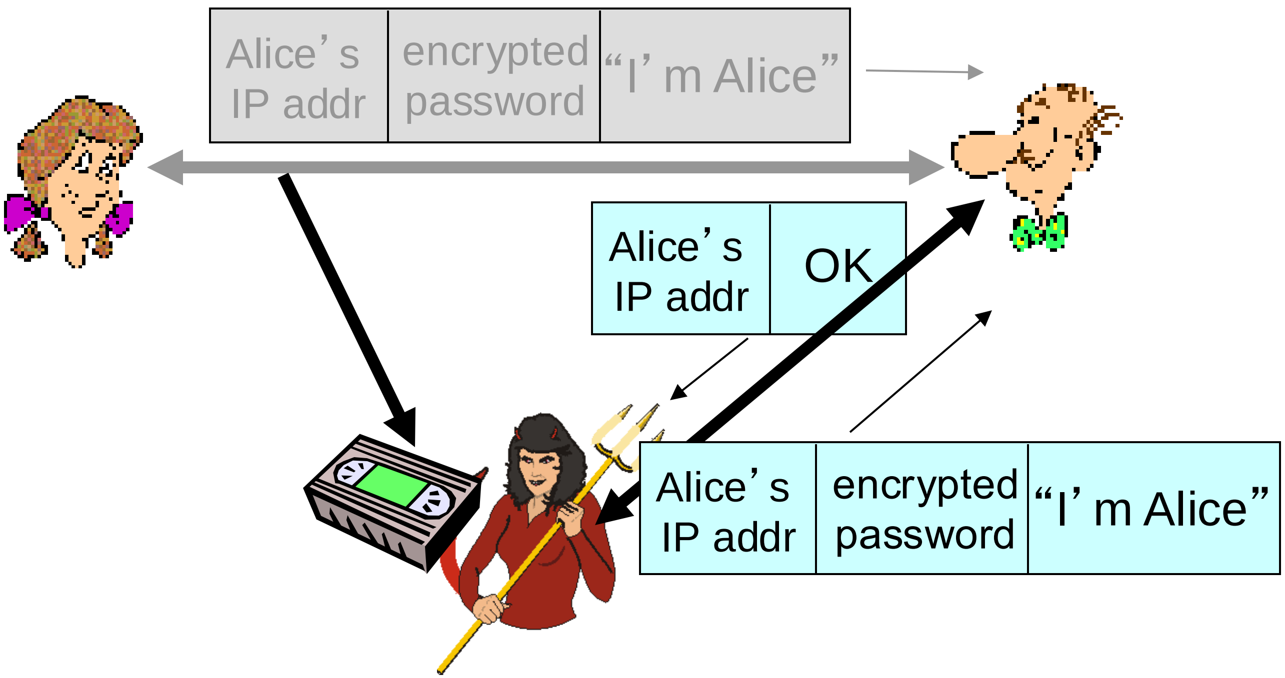
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: yet another try

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



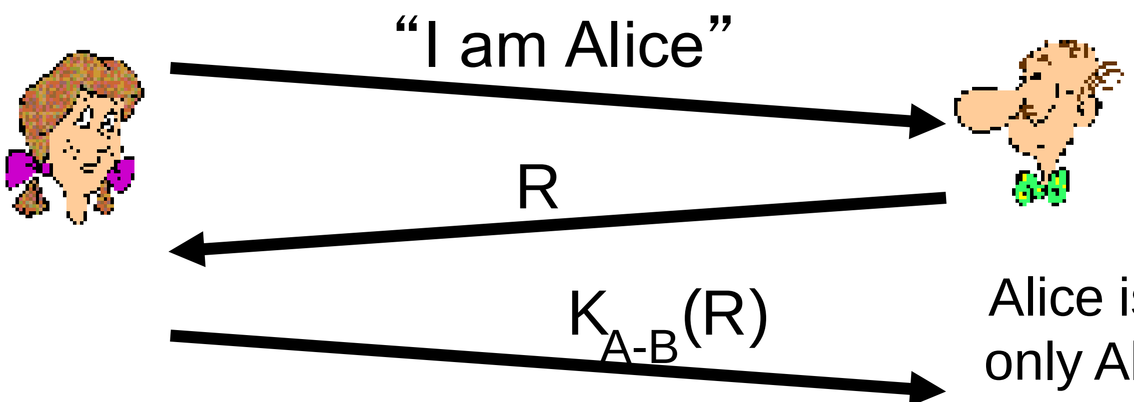
record and playback *still* works!

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!

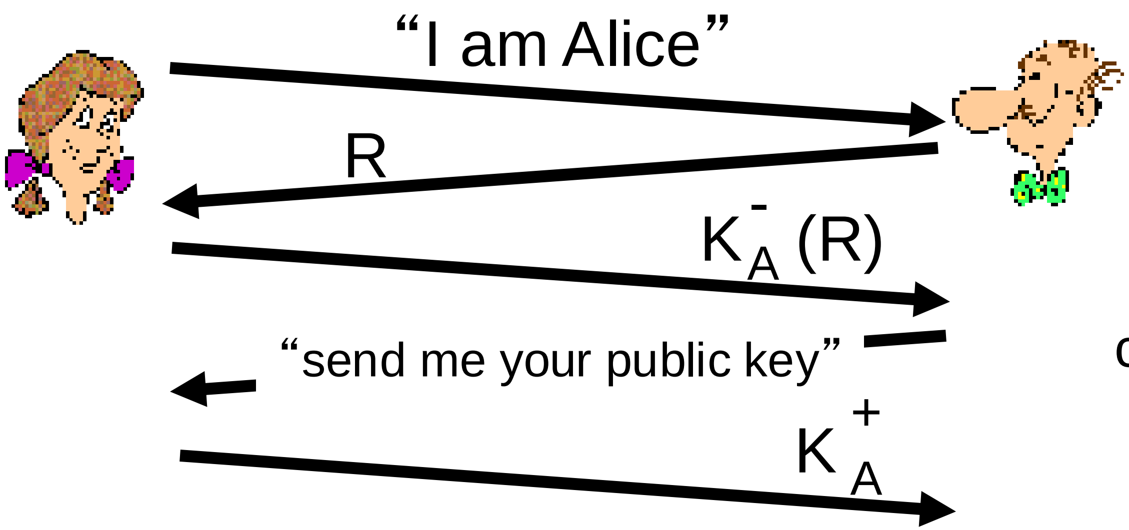
Drawbacks: requires shared symmetric key

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

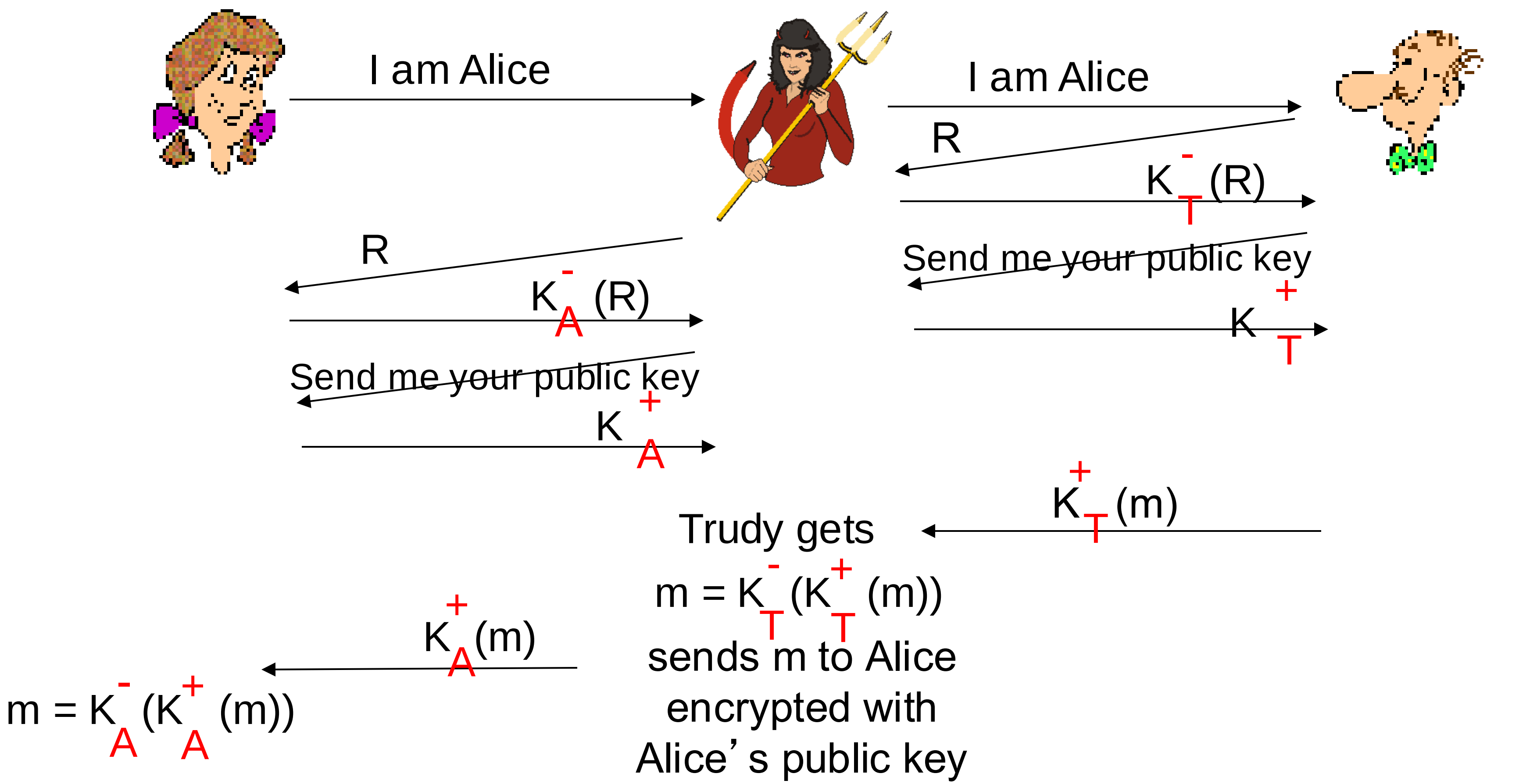
$$K_A^+(K_A^-(R)) = R$$

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

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

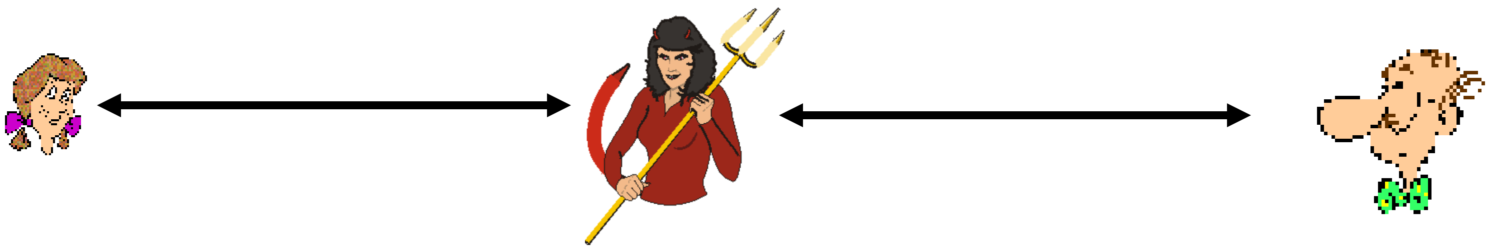
ap5.0: security hole

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:

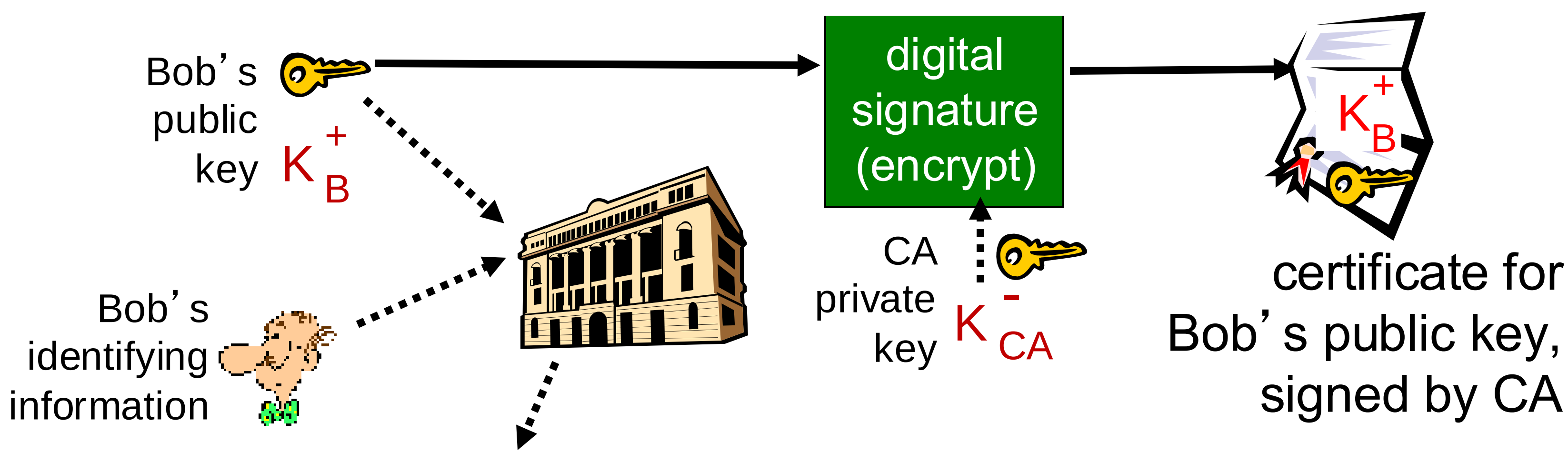
- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation!)
- problem is that Trudy receives all messages as well!

Public key certification → 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"

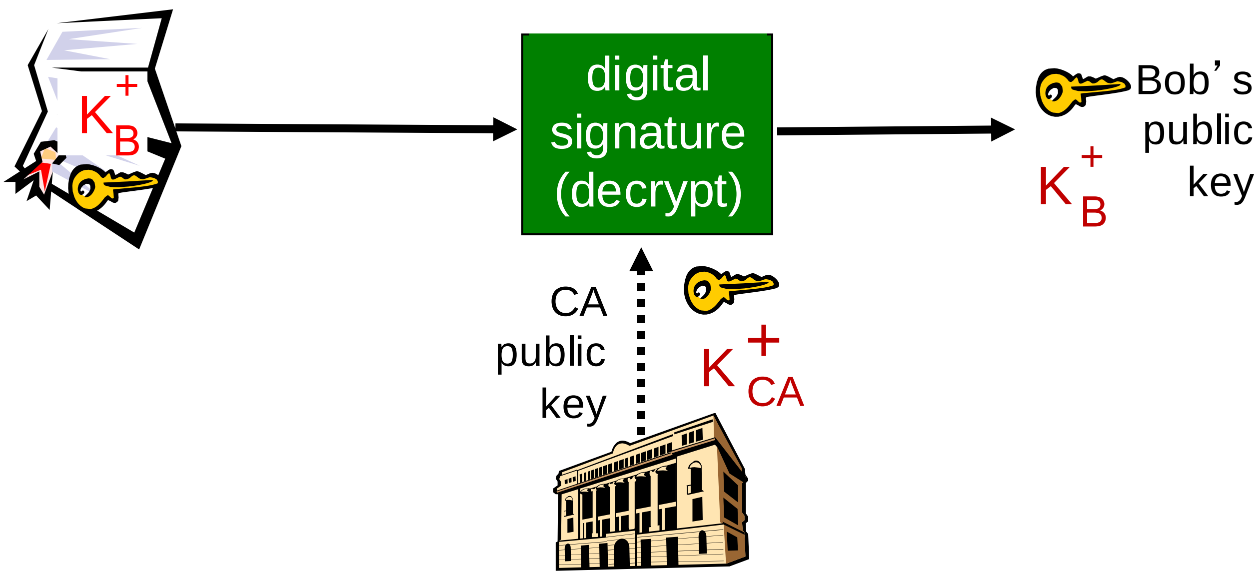


e.g. Comodo, Digicert, GlobalSign, Symantec, GoDaddy, etc..

Certification authorities

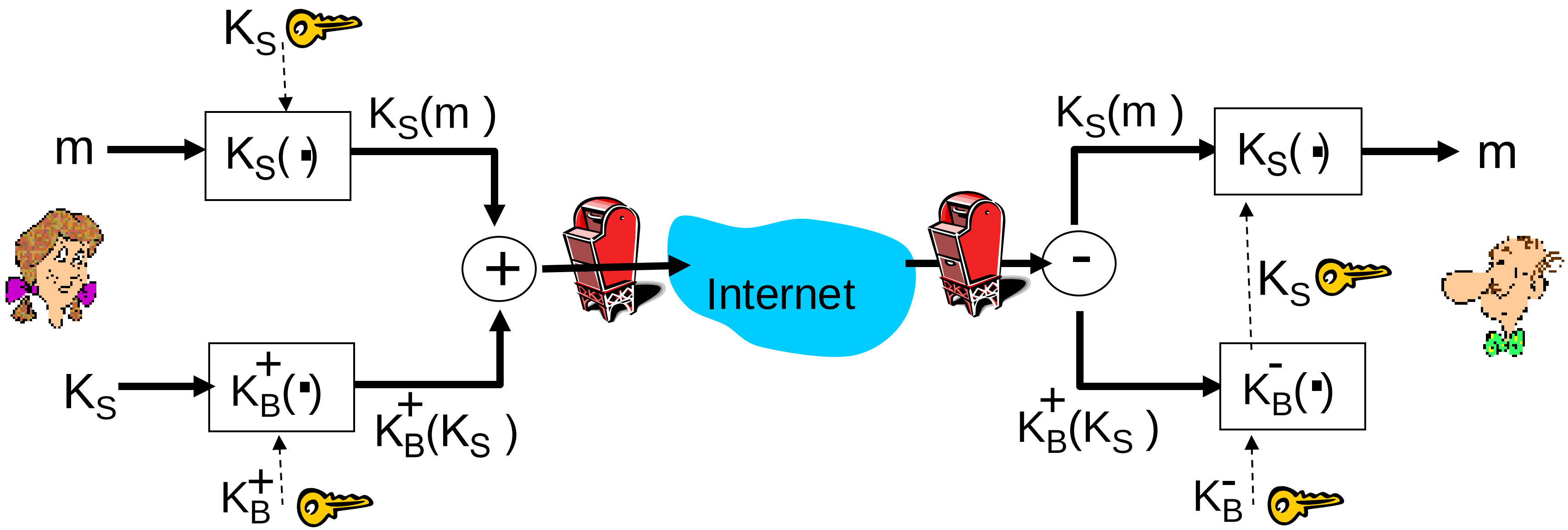
when Alice wants Bob's public key:

- 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.

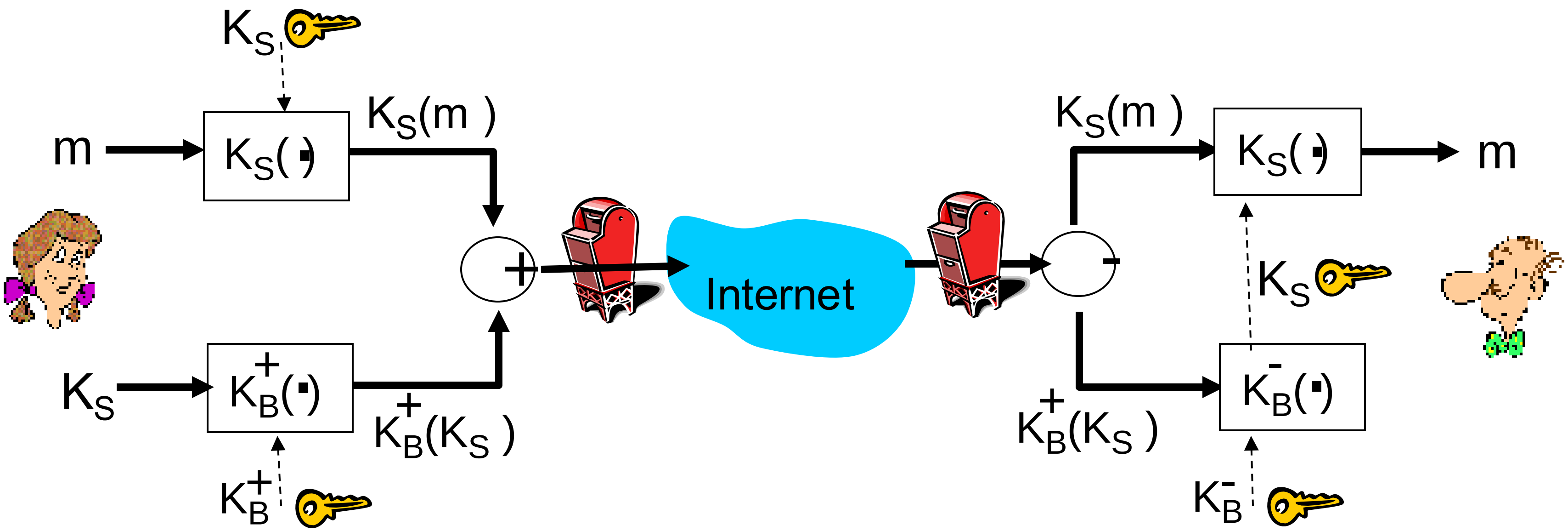


Alice:

- 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.



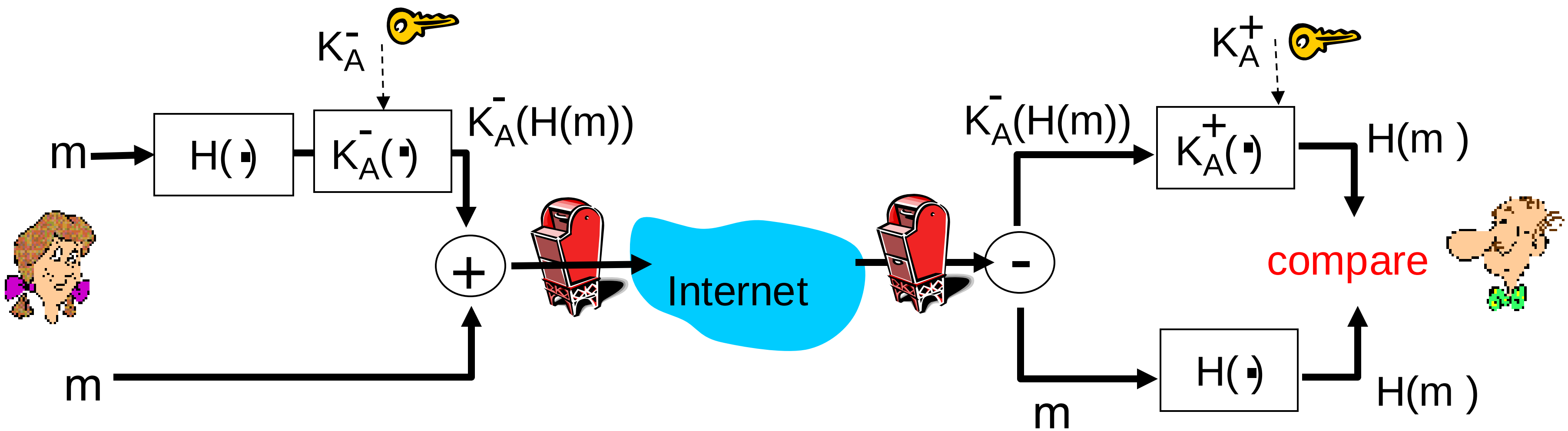
Bob:

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

public key encryption is relatively inefficient, particularly for long messages.

Secure e-mail (continued)

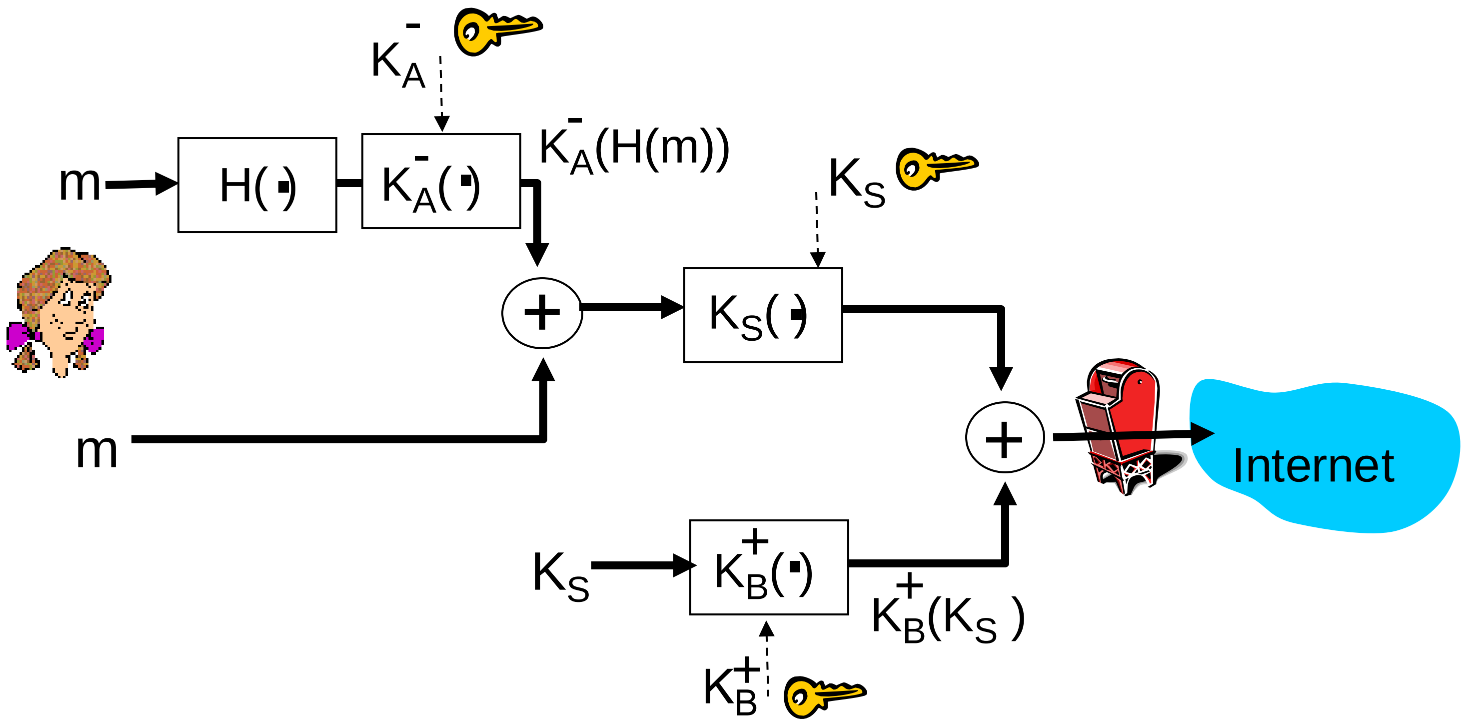
Alice wants to provide sender authentication message integrity



- Alice digitally signs message
- sends both message (in the clear) and digital signature
- Bob obtains Alice's public key via the certificate and uses to get H(m)

Secure e-mail (continued)

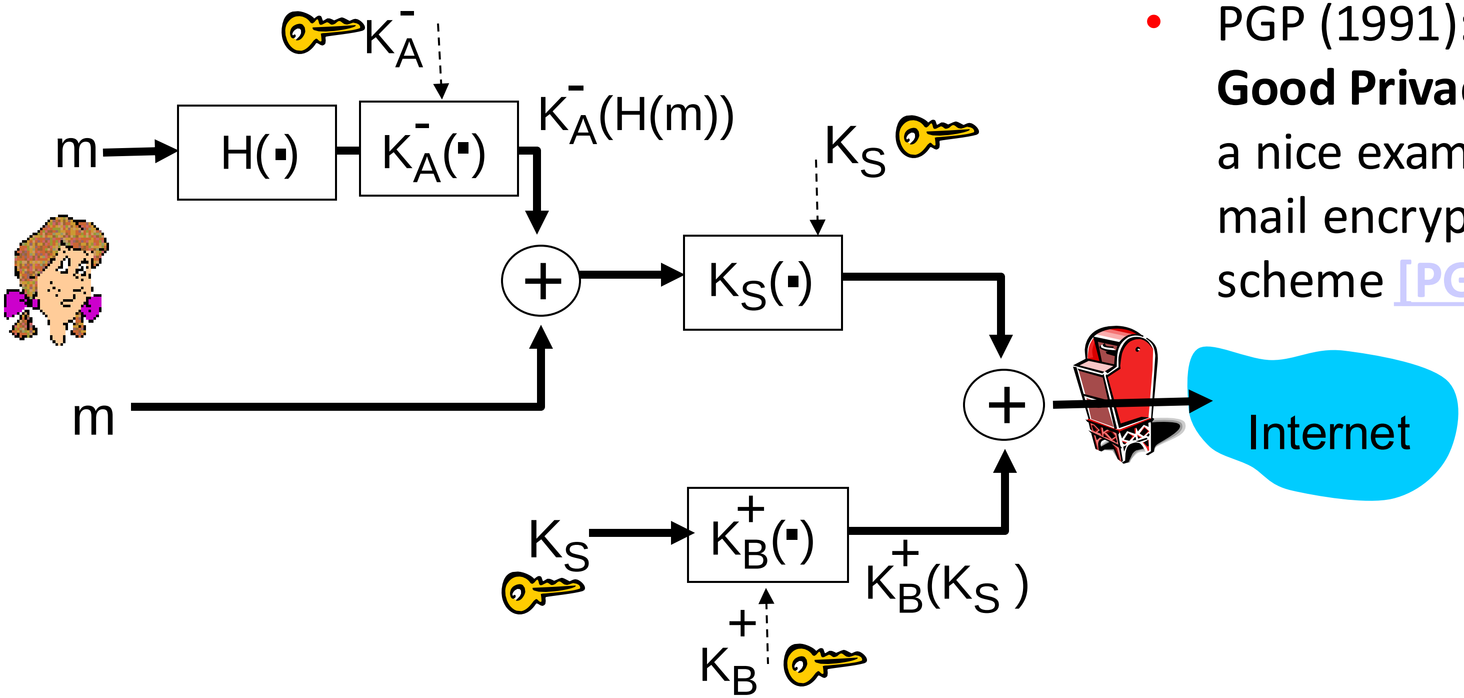
Alice wants to provide secrecy, sender authentication, message integrity.



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

And so she used "symmetric key cryptography, public key cryptography, a hash function, and a digital signature"

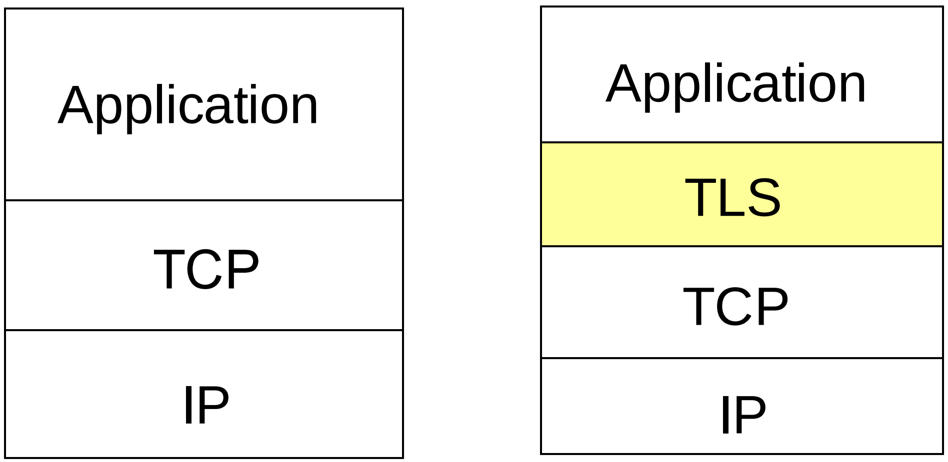
PGP email encryption



- PGP (1991): **Pretty Good Privacy (PGP)** is a nice example of an e-mail encryption scheme [PGPI 2016]

- 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

TLS and TCP/IP



normal application *application with TLS*

- SSL: Secure Socket Layer → TLS (Transport Layer Security)
- TLS provides application programming interface (API) to applications
- C and Java TLS libraries/classes readily available

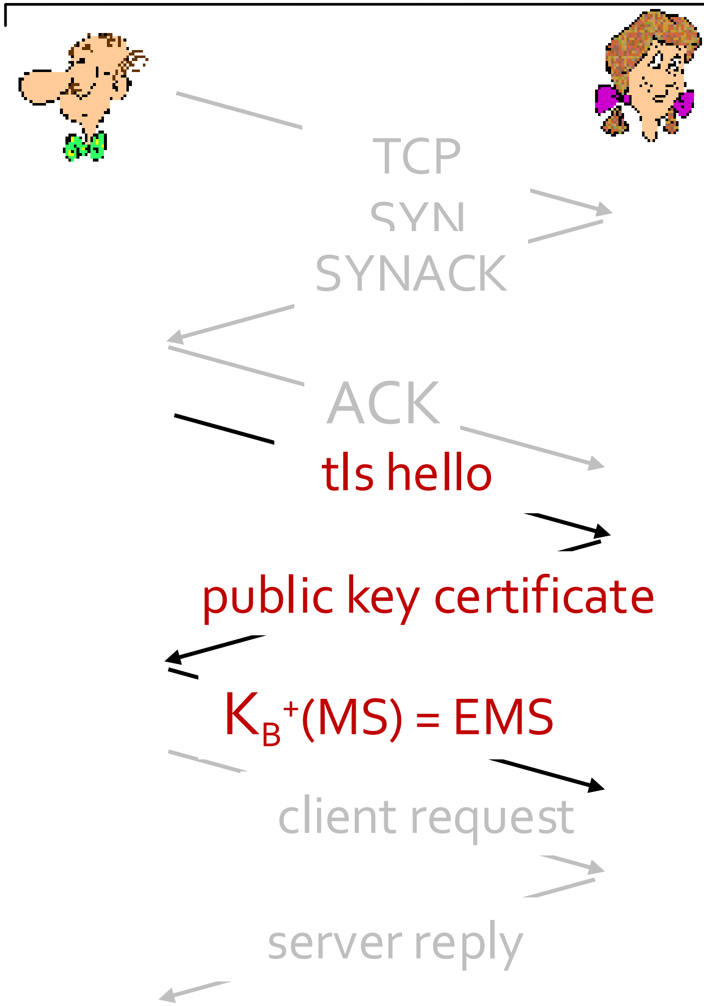
Transport-layer security (TLS)

- widely deployed security protocol above the transport layer
  - supported by almost all browsers, web servers: https (port 443)
- history:
  - early research, implementation: secure network programming, secure sockets
  - secure socket layer (SSL) deprecated [2015]
  - TLS 1.3: RFC 8846 [2018]
- provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing*
  - authentication: via *public key cryptography*

all techniques we have studied!

- activities
  - handshake: Alice, Bob use their certificates, private keys to authenticate each other, exchange or create shared secret
  - key derivation: Alice, Bob use shared secret to derive set of keys
  - data transfer: stream data transfer: data as a series of records
  - connection closure: special messages to securely close connection

TLS: Big picture



MS: master secret

EMS: encrypted master secret

- Bob establishes TCP connection with Alice
- Bob verifies that Alice is really Alice
- Bob sends Alice a master secret key (MS), used to generate all other keys for TLS session
- potential issues:
  - 3 RTT before client can start receiving data (including TCP handshake)

SSL cipher suite

cipher suite

- public-key algorithm
- symmetric encryption 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

Purpose

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

SSL services

Fragmentation

- Divides the data into blocks of 2^{14} or less

Compression

- Each fragment of data is compressed using a negotiated method (optional)

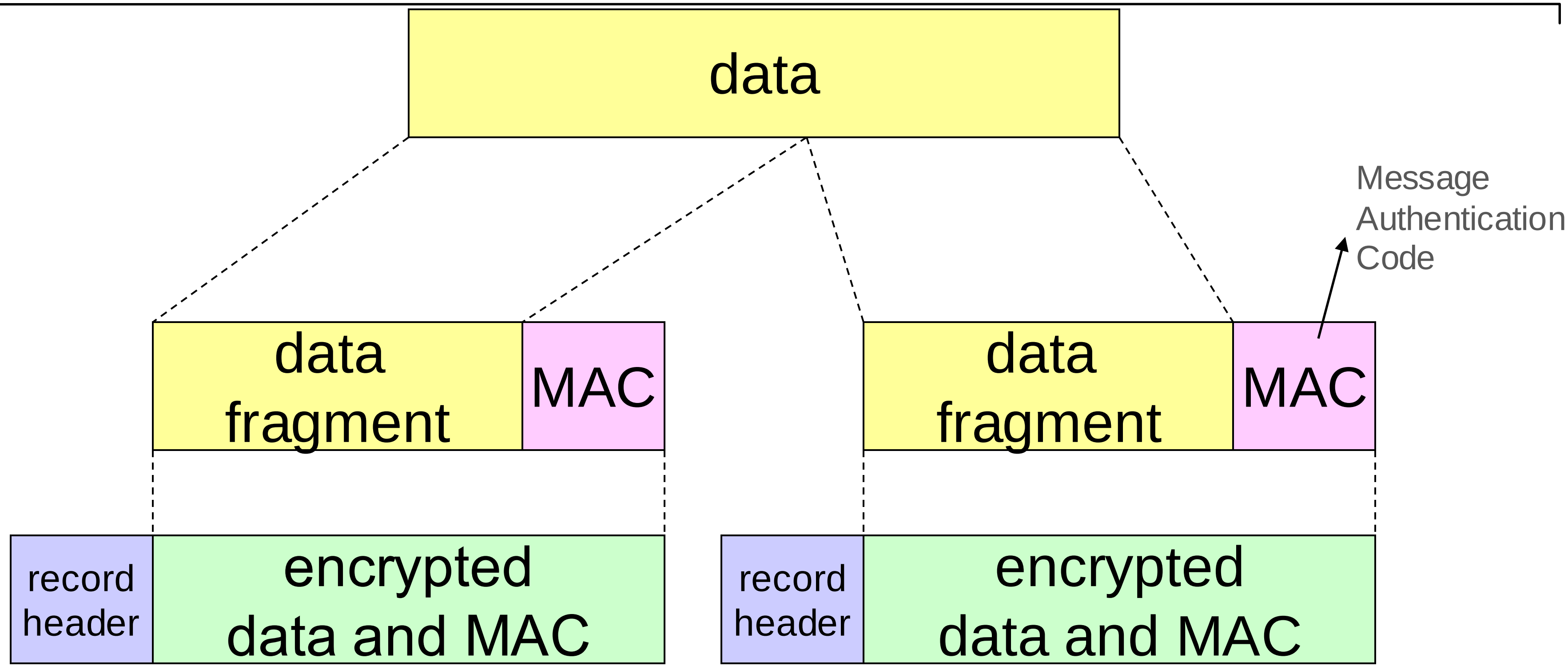
Message Integrity

- Uses key-hashed function to create MAC (Message Authenticated code)

Confidentiality

- Original data and the MAC are encrypted using symmetric key cryptography

SSL record protocol

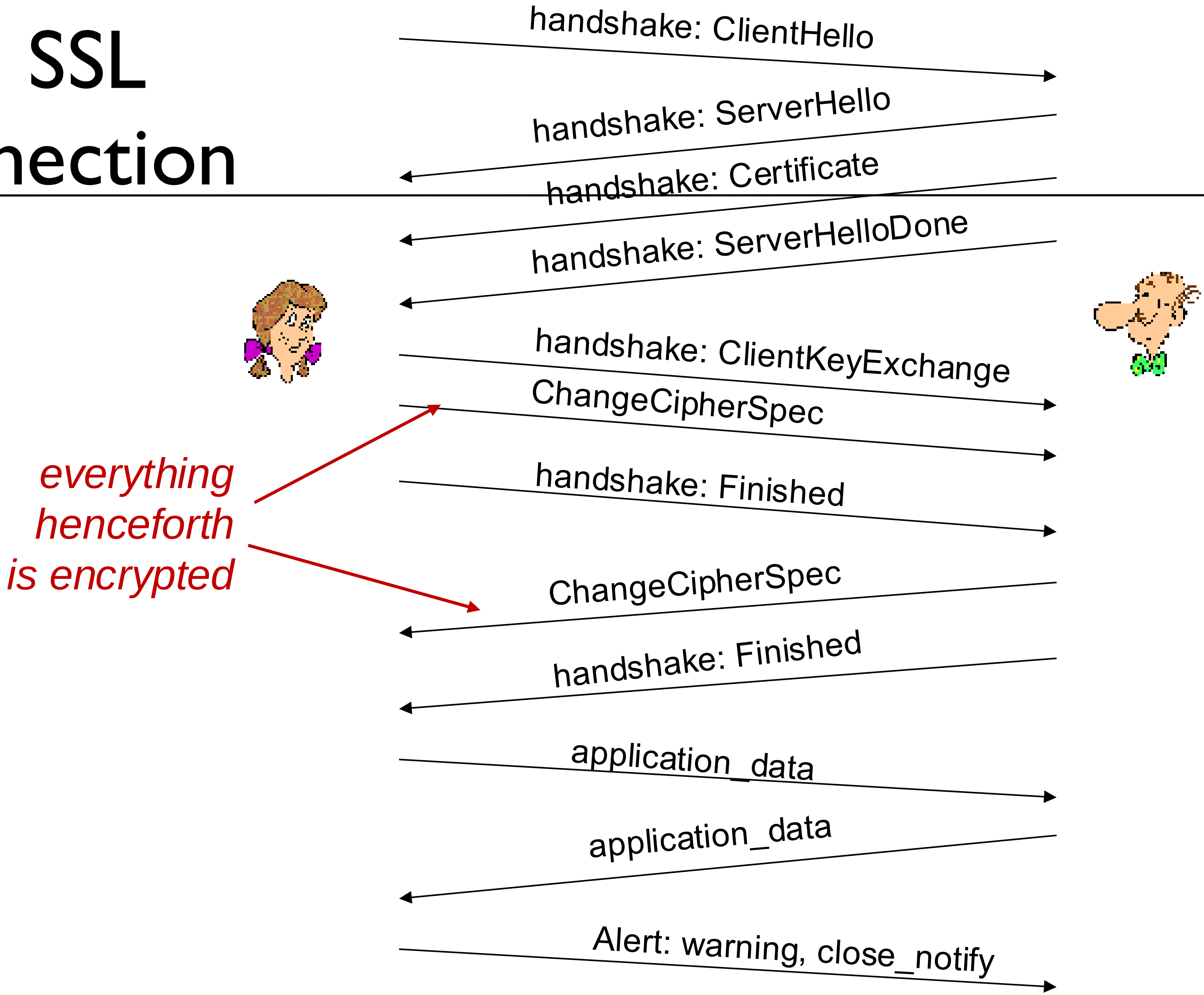


record header: content type; version; length

MAC: includes sequence number, MAC key M_x

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

Real SSL connection



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”
- 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)

Summary

Today:

- Hashes and message digests
- Hash function algorithms
- Digital signatures
- Authentication
- Public key certification authorities
- Secure email and TLS

Next time:

- read 8.7, 8.8 and 8.9 of K&R (Ipsec, Secure wireless LAN, IDS,..)
- follow on Canvas! material and announcements

Any questions?