

Chapter 8

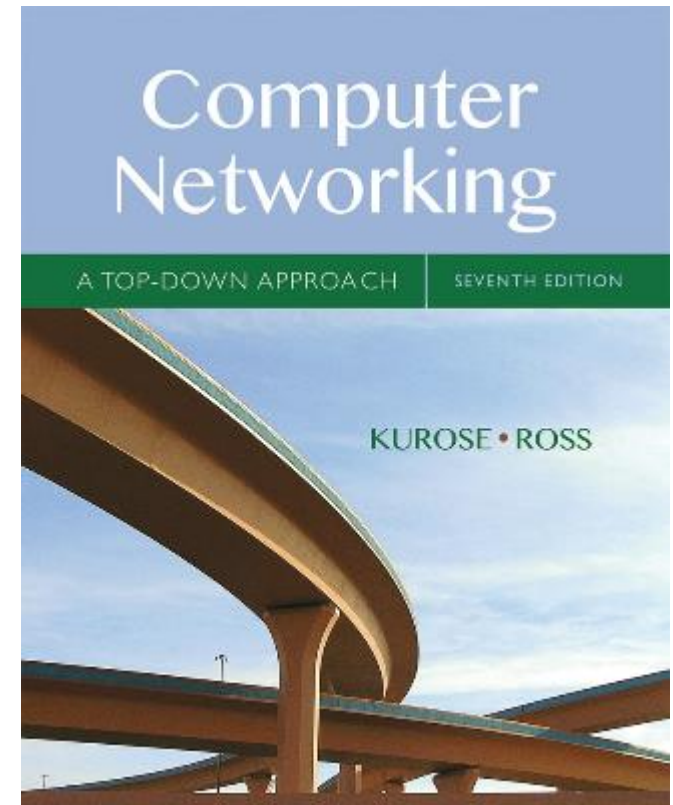
Security

Gabriele Saturni

saturni@di.uniroma1.it

These slides are adapted from the slides provided by Kurose-Ross Book

All material copyright 1996-2016
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer
Networking: A Top
Down Approach*

7th edition

Jim Kurose, Keith Ross

Pearson/Addison Wesley

April 2016

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity, authentication

8.4 Securing TCP connections: SSL

8.5 Network layer security: IPsec

What is network security?



What is network security?



What is network security?



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

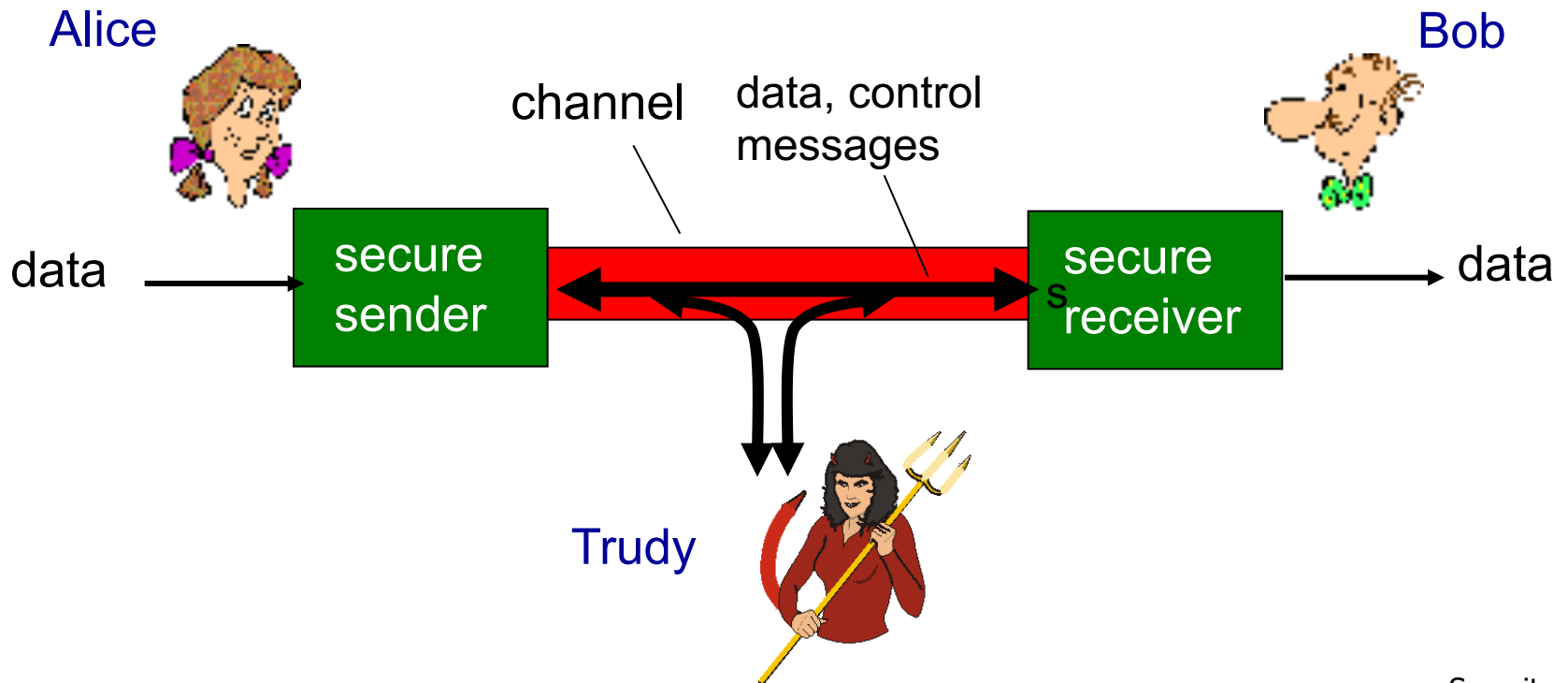
message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

non repudiation: a sender cannot deny having sent a message

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



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 bad guys (and girls) out there!

Q: What can a “bad guy” do?

A: A lot! See section 1.6

- *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)

Chapter 8 roadmap

8.1 What is network security?

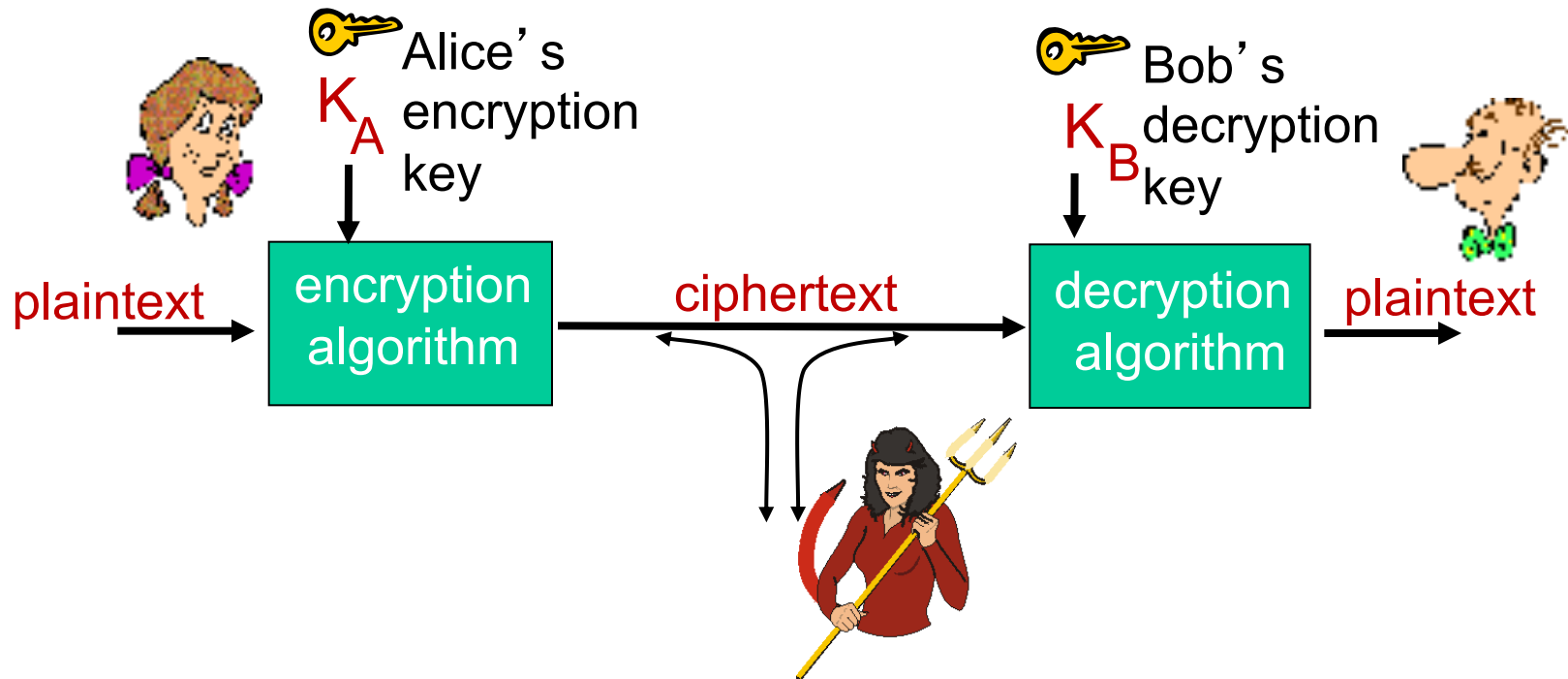
8.2 *Principles of cryptography*

8.3 Message integrity, authentication

8.4 Securing TCP connections: SSL

8.5 Network layer security: IPsec

The language of cryptography



m plaintext message

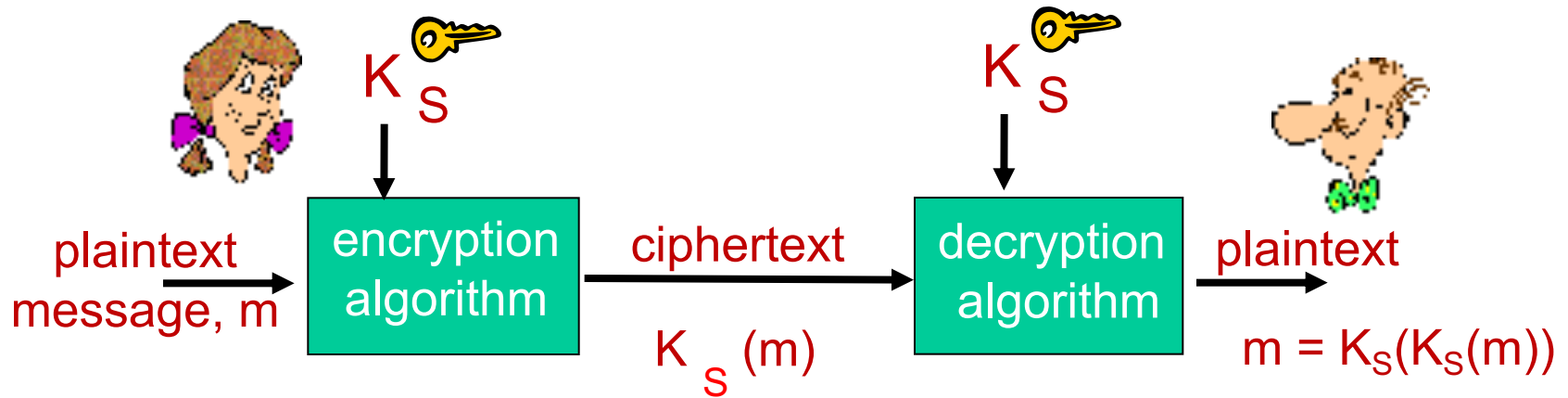
$K_A(m)$ ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

Breaking an encryption scheme

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

Symmetric key cryptography

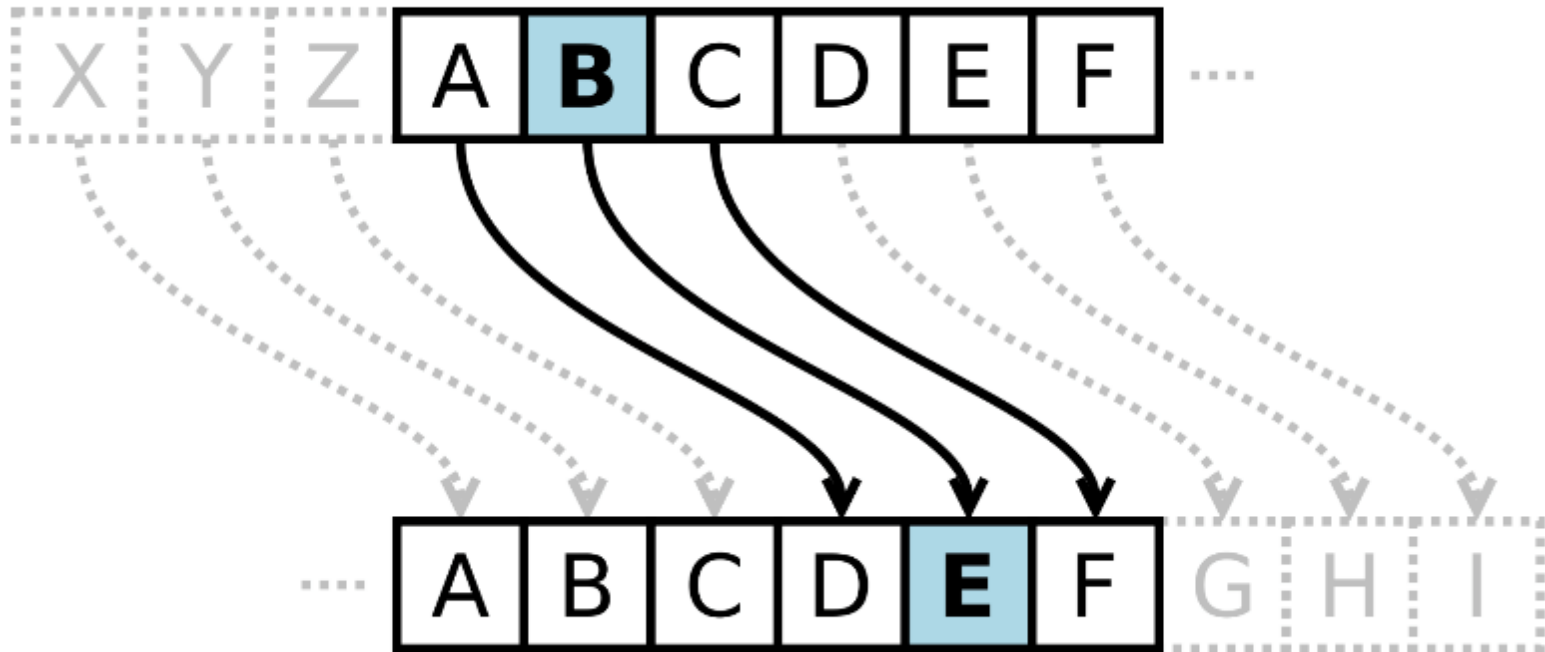


symmetric key crypto: Bob and Alice share same (symmetric) key: K_S

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

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

Caesar cipher scheme



Simple encryption scheme

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

plaintext:	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
ciphertext:	m	n	b	v	c	x	z	a	s	d	f	g	h	j	k	l	p	o	i	u	y	t	r	e	w	q

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

Simple encryption scheme

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

plaintext:	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
		↓																								↓
ciphertext:	m	n	b	v	c	x	z	a	s	d	f	g	h	j	k	l	p	o	i	u	y	t	r	e	w	q

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

Easy to break! These cipher does not change the properties of the plaintext. Repeated letters in the plaintext will correspond to repeated letters in the ciphertext.

A more sophisticated encryption approach

- n substitution ciphers, M_1, M_2, \dots, 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

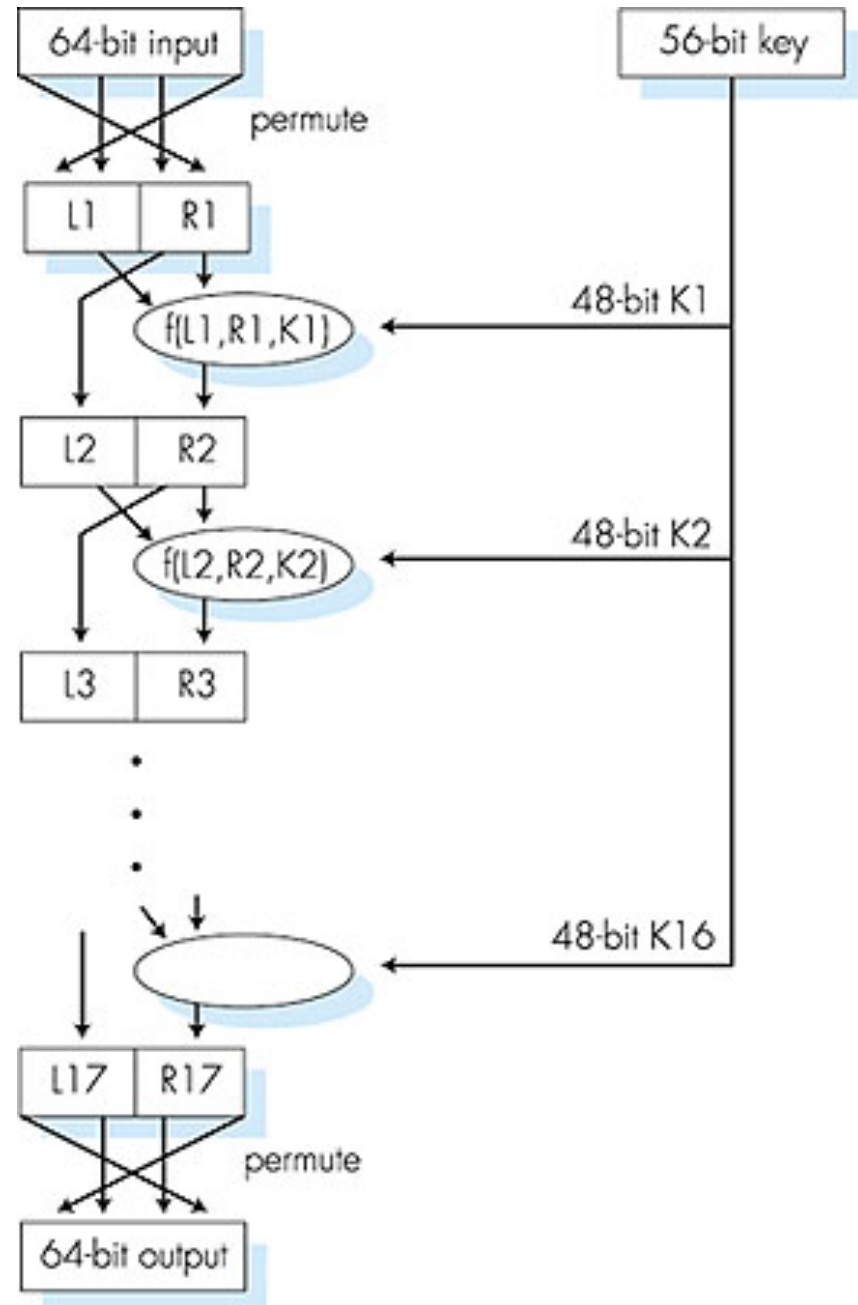
Symmetric key crypto: DES

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



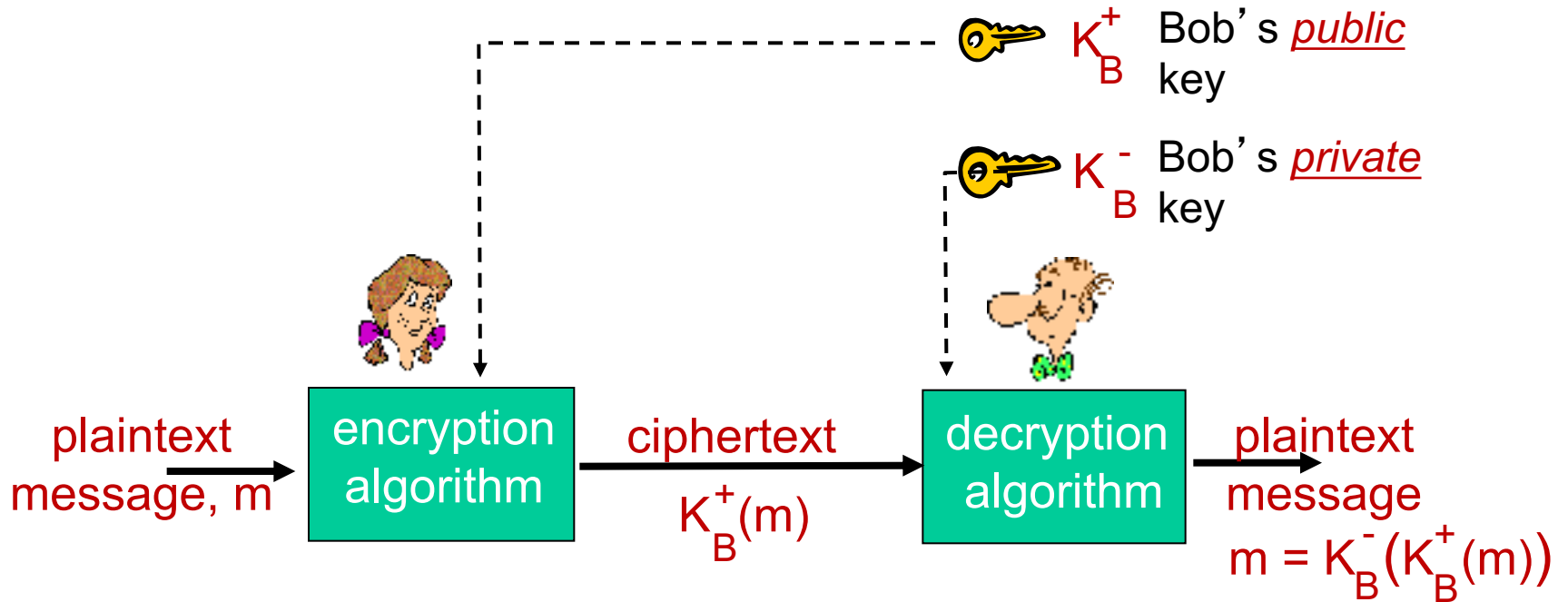
symmetric key crypto

- 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



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

- $x \bmod n$ = remainder of x when divide by n

- facts:

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a+b) \bmod n$$

$$[(a \bmod n) - (b \bmod n)] \bmod n = (a-b) \bmod n$$

$$[(a \bmod n) * (b \bmod n)] \bmod n = (a*b) \bmod n$$

- thus

$$(a \bmod n)^d \bmod n = a^d \bmod n$$

- example: $x=14$, $n=10$, $d=2$:

$$(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$$

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

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!

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

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:

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

use public key first,
followed by
private key

use private key
first, followed by
public key

result is the same!

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

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity, *authentication*

8.4 Securing TCP connections: SSL

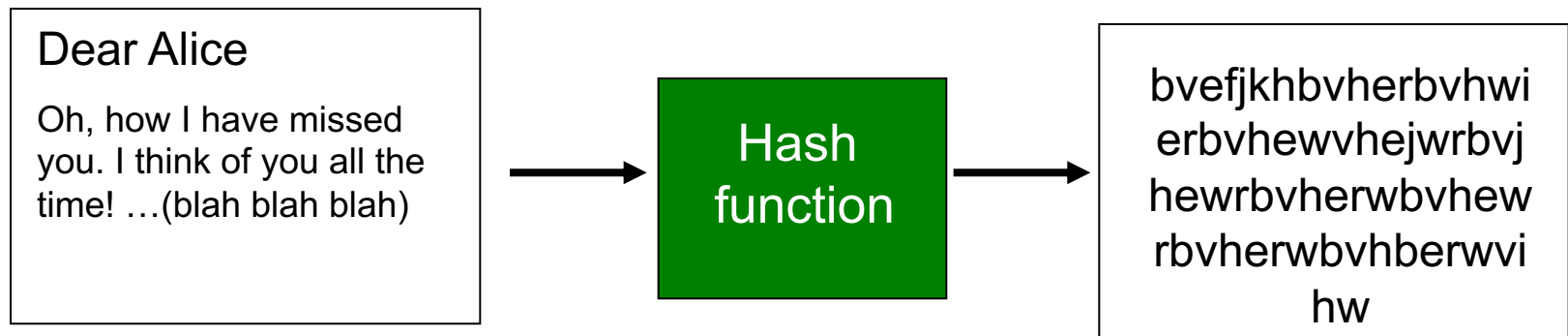
8.5 Network layer security: IPsec

Message Integrity

- In the previous slides we saw how encryption can be used to provide confidentiality.
- Now, we turn to the equally important cryptography topic of providing **message authentication** (or integrity).
- *Recall:* message integrity means that a message m was not compromised.

Cryptography Hash functions

- A cryptography hash function H is required to have the following property:
 - It is computationally infeasible to find any two different message x and y such that $H(x) = H(y)$

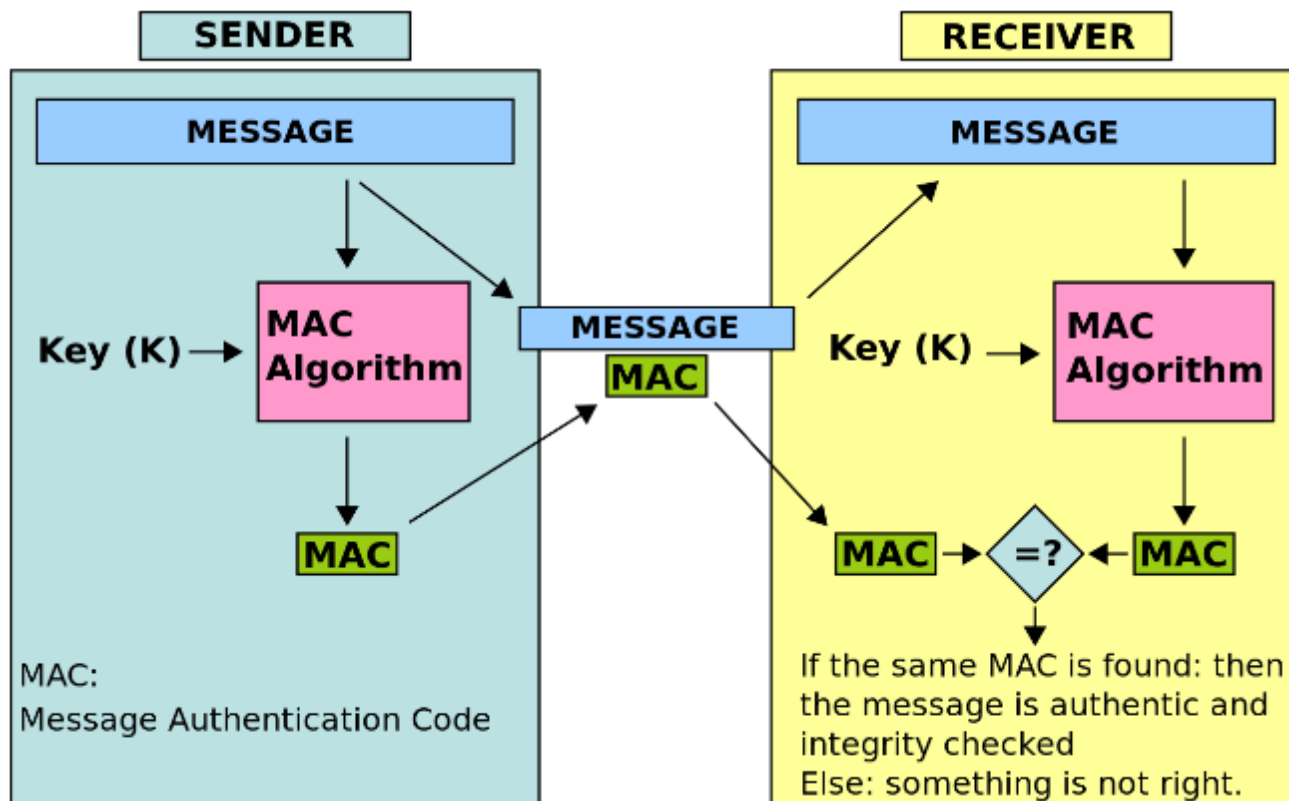


Hash function algorithms

- **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
- **SHA-2 (better than SHA-1)**
 - US standard [NIST, FIPS PUB 180-2]
 - stronger than SHA-1
 - 256-bit message digest
- **SHA-3**
 - the stronger version of SHA algorithms
 - US future standard [NIST, FIPS PUP 202]
 - 384-bit message digest

Message Authentication Code (MAC)

- Based on hash function for guarantee **message integrity**.



Digital signatures

- In the previous slides we saw how encryption can be used to provide confidentiality and message integrity.
- Now, we turn to the equally important cryptography topic of providing **non-repudiation**. The property that ensure that a sender can not deny having sent a particular message.
- Digital Signature ensures this property.

Digital signatures

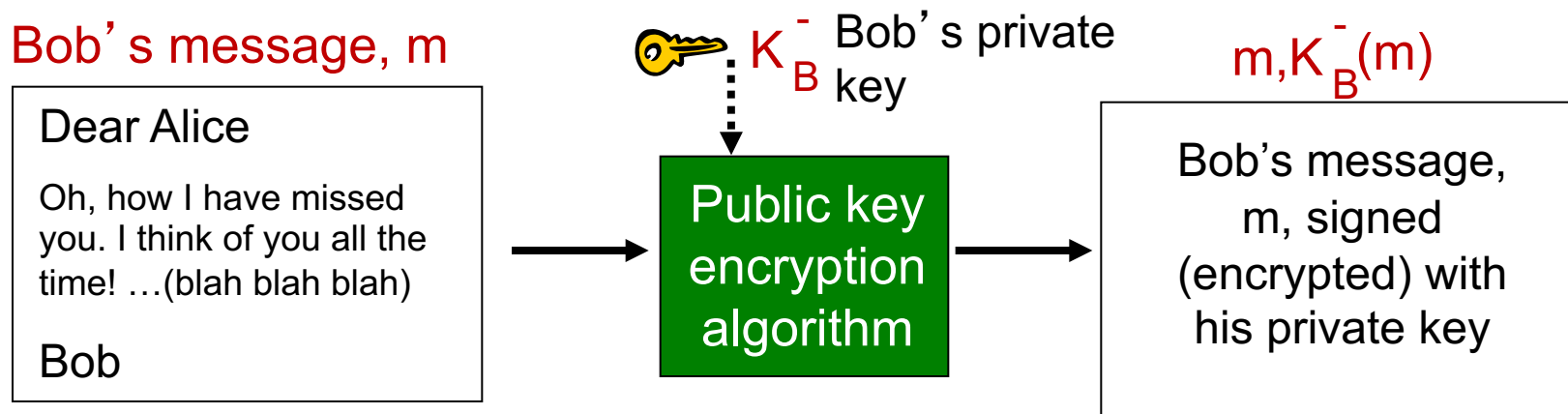
cryptographic technique analogous to hand-written signatures:

- 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

Entity authentication

- What we had showed:
 - ✓ how guarantee the confidentiality.
 - ✓ how guarantee the integrity.
 - ✓ An entity that sent a message can not deny it.
- But... still... what can be done for authenticate the entity?



Messages

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”

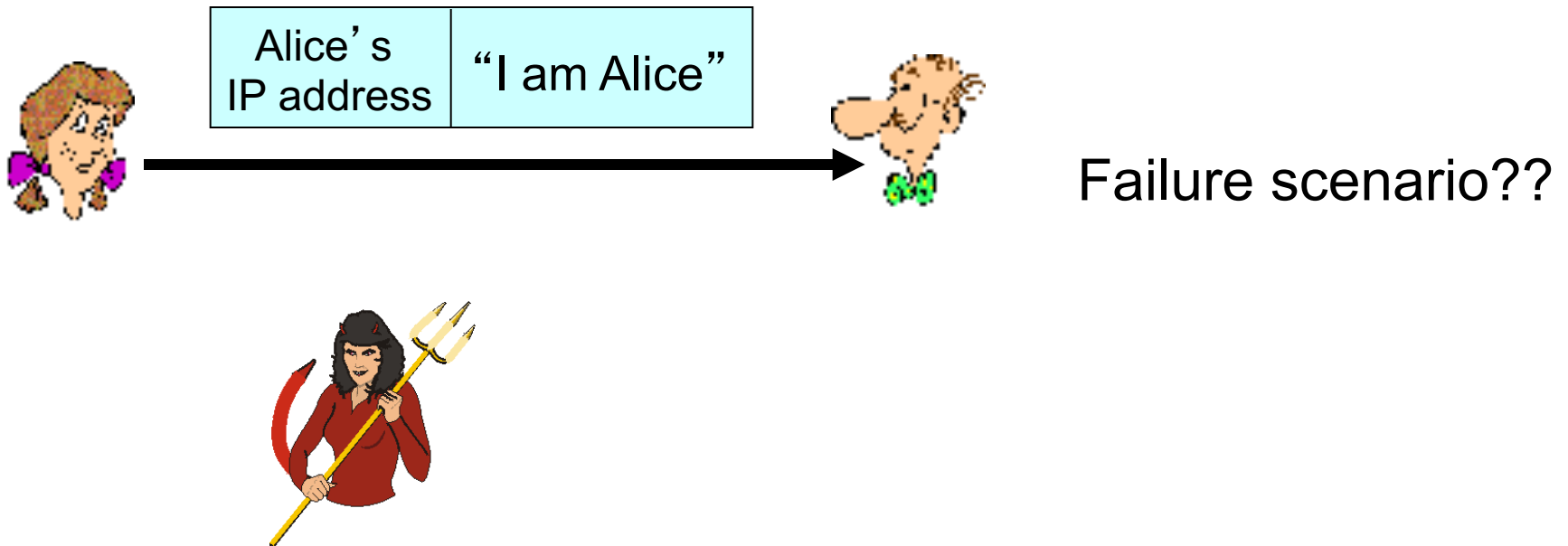


“I am Alice”

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

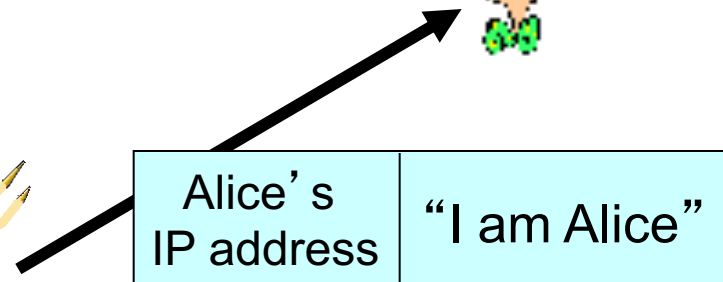
Authentication: another try

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



Authentication: another try

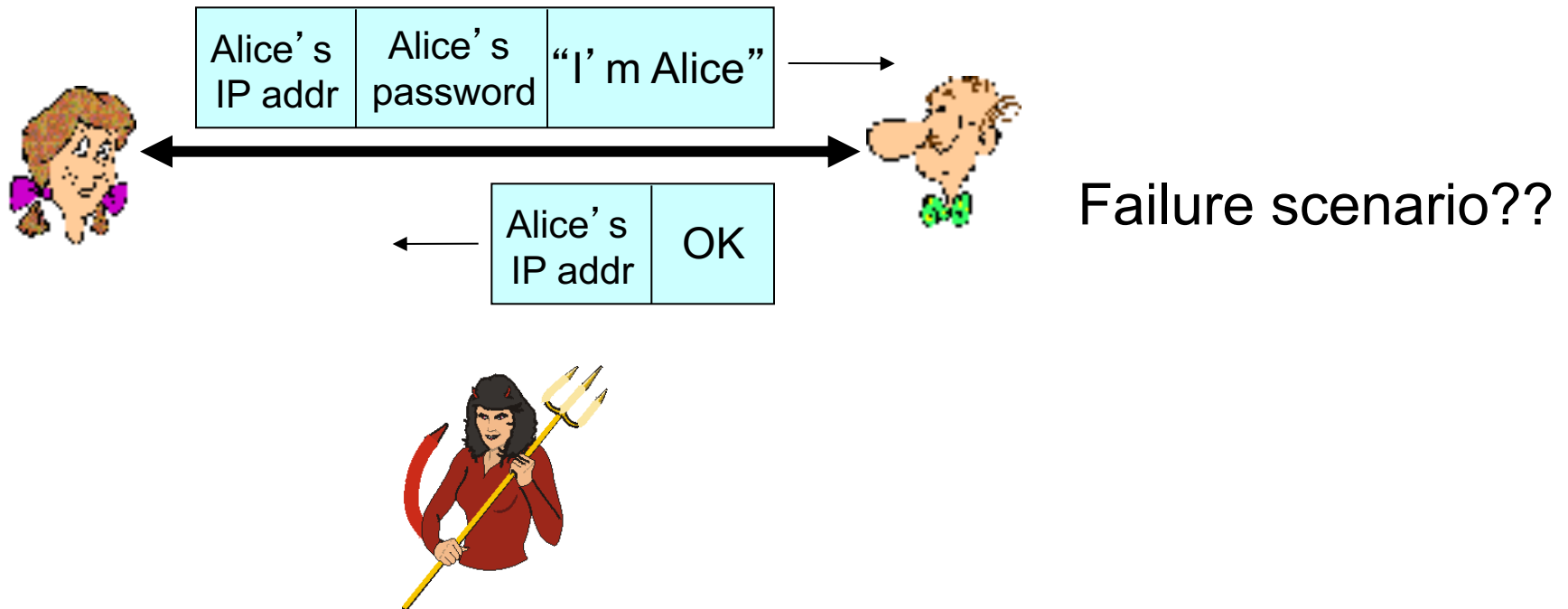
Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address



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

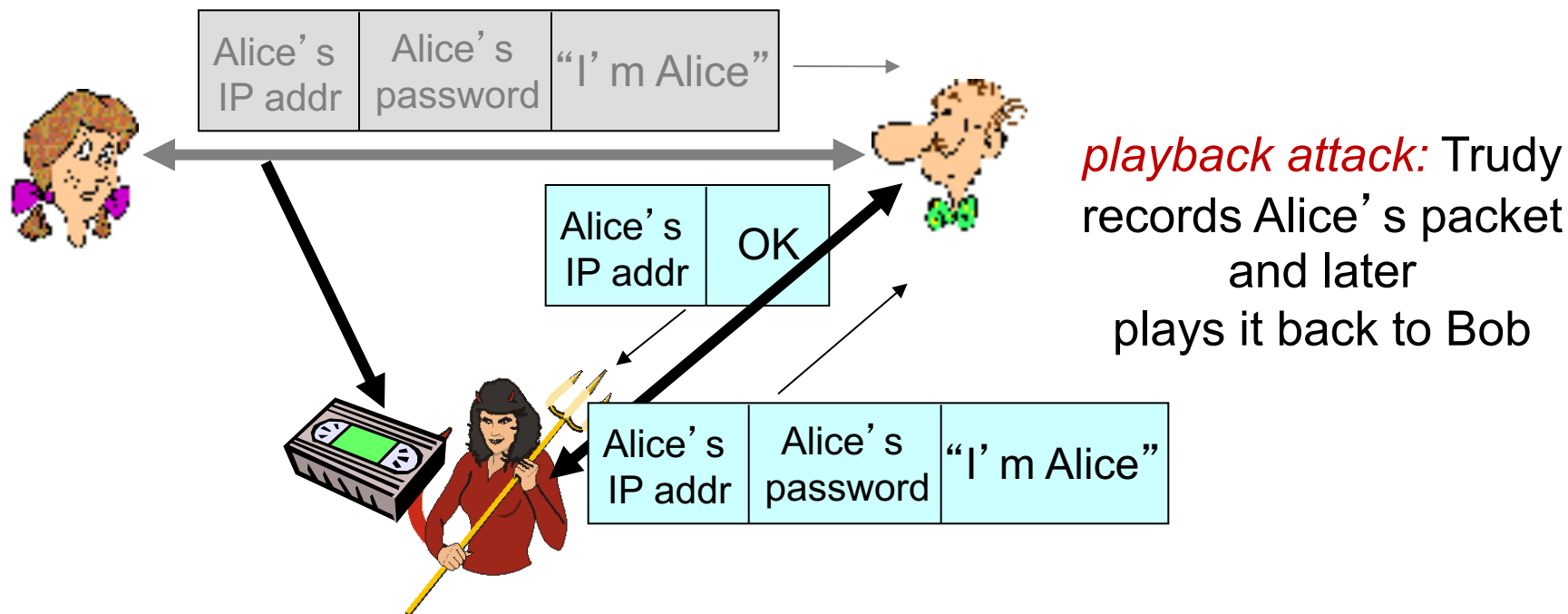
Authentication: another try

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



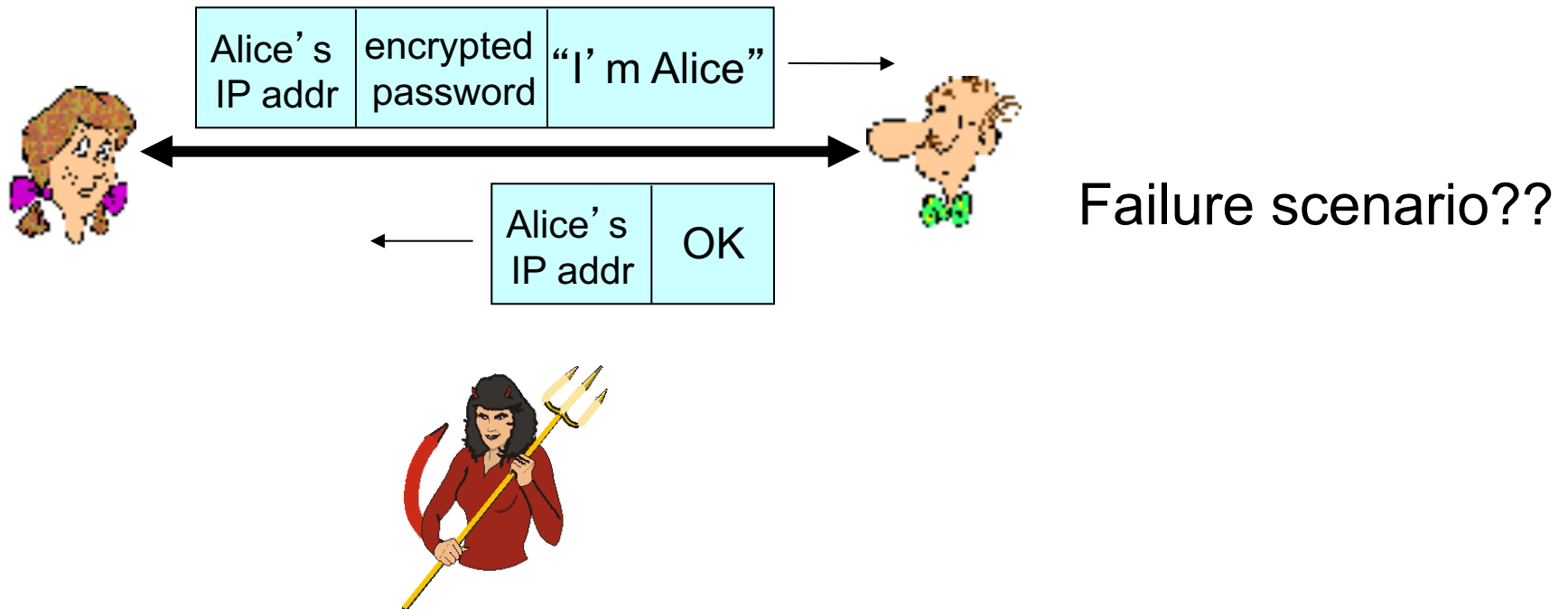
Authentication: another try

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



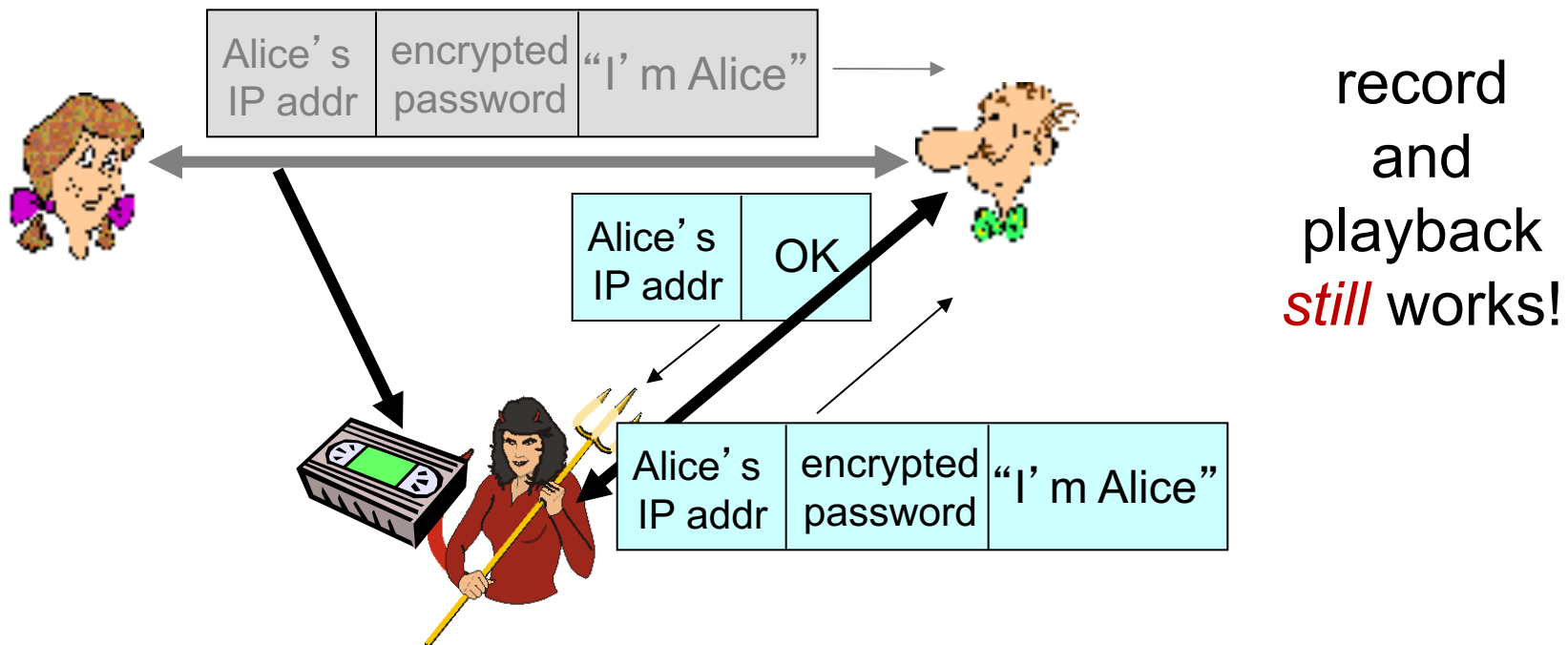
Authentication: yet another try

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



Authentication: yet another try

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

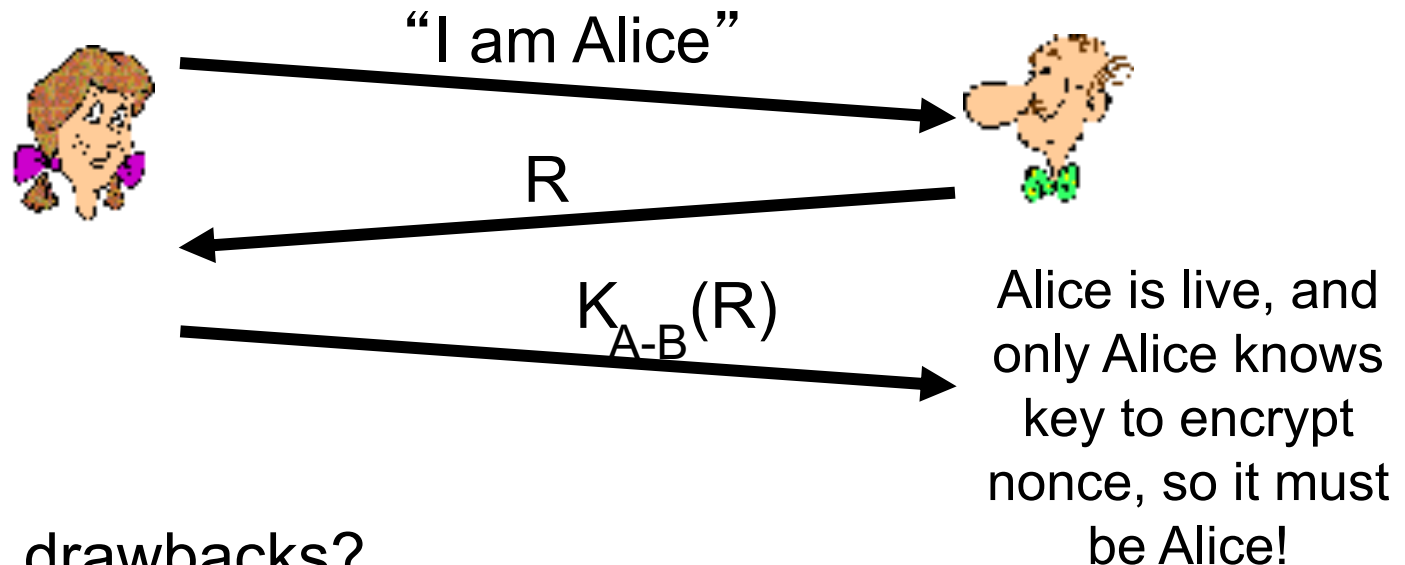


Authentication: yet another try

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



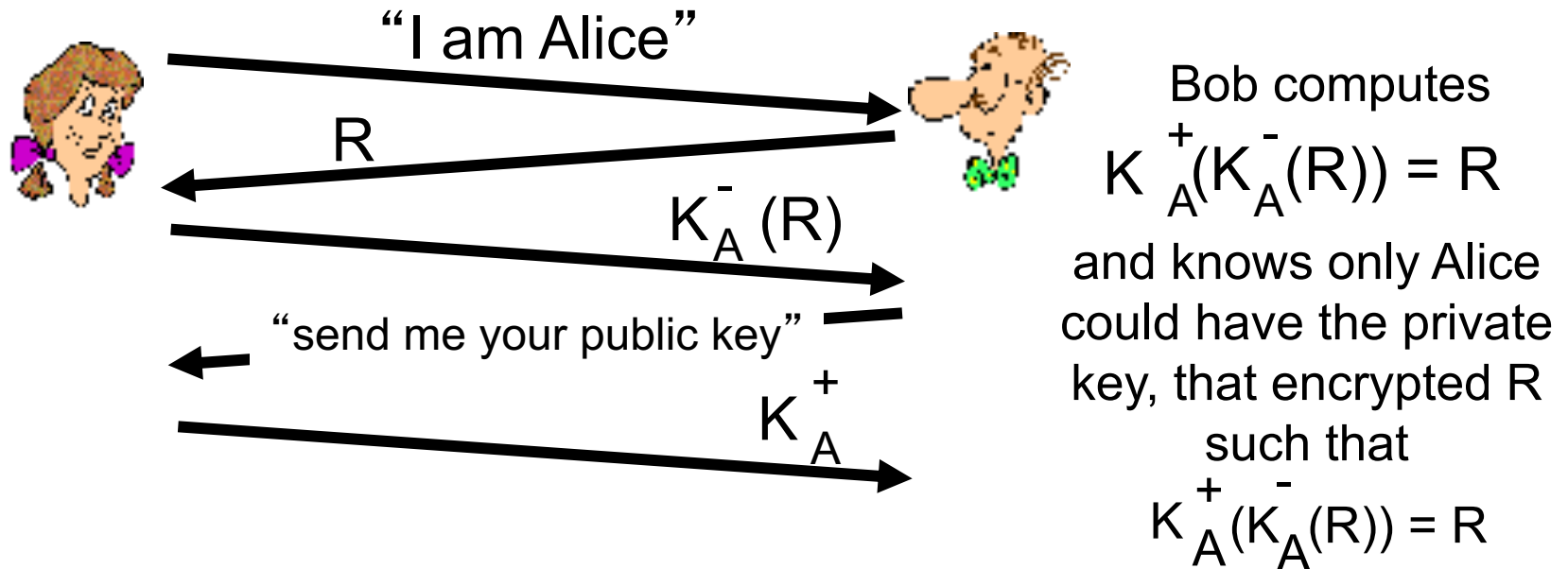
Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key and ... **how they agree on that key?**

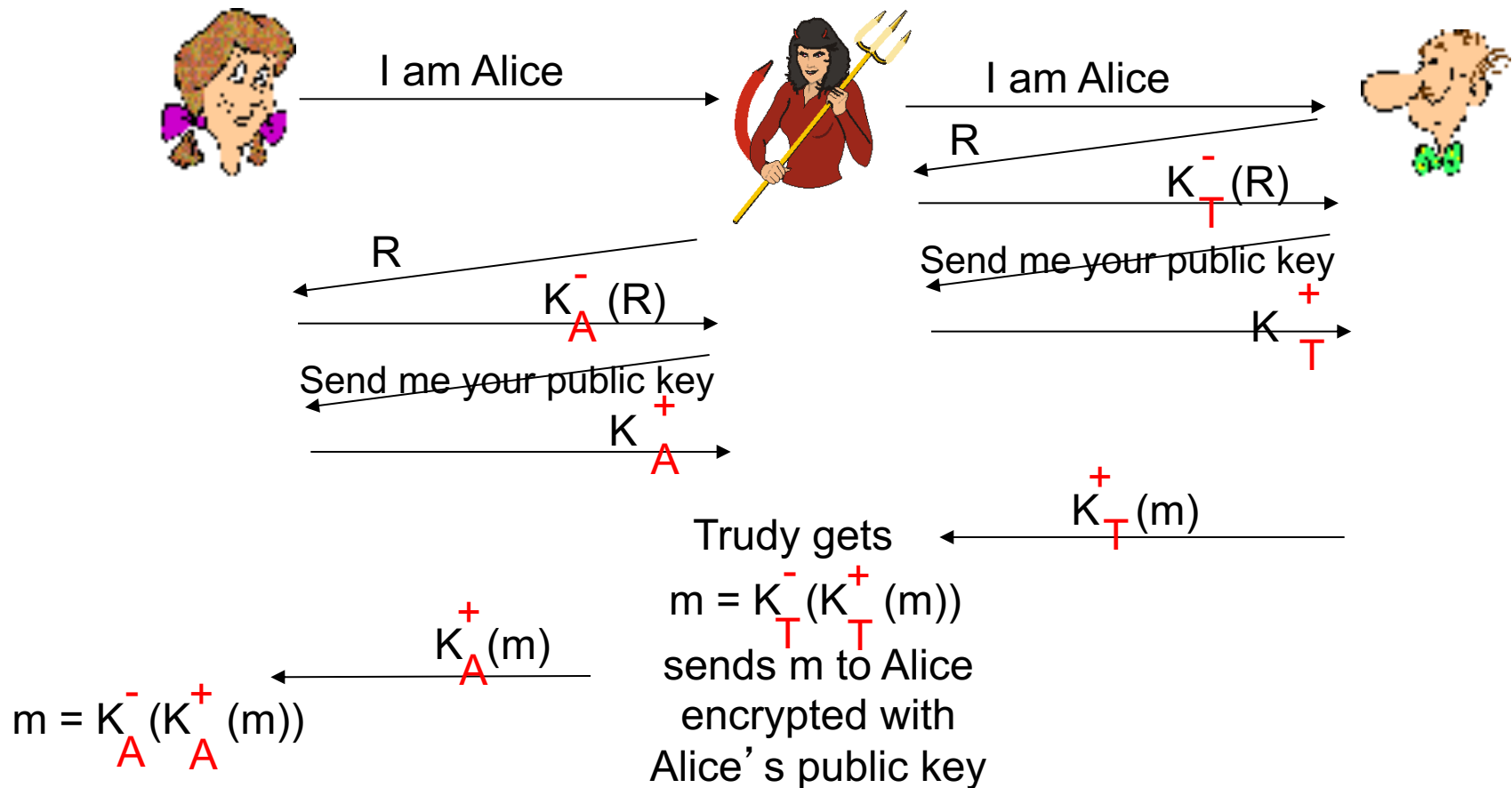
- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



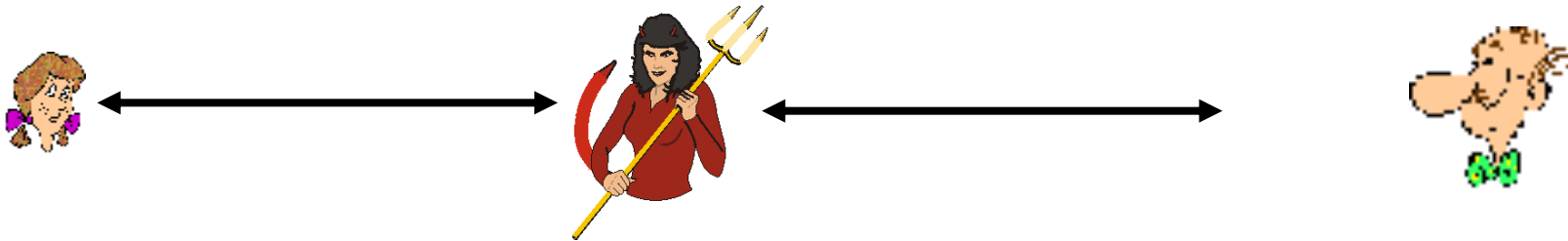
ap5.0: security hole

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:

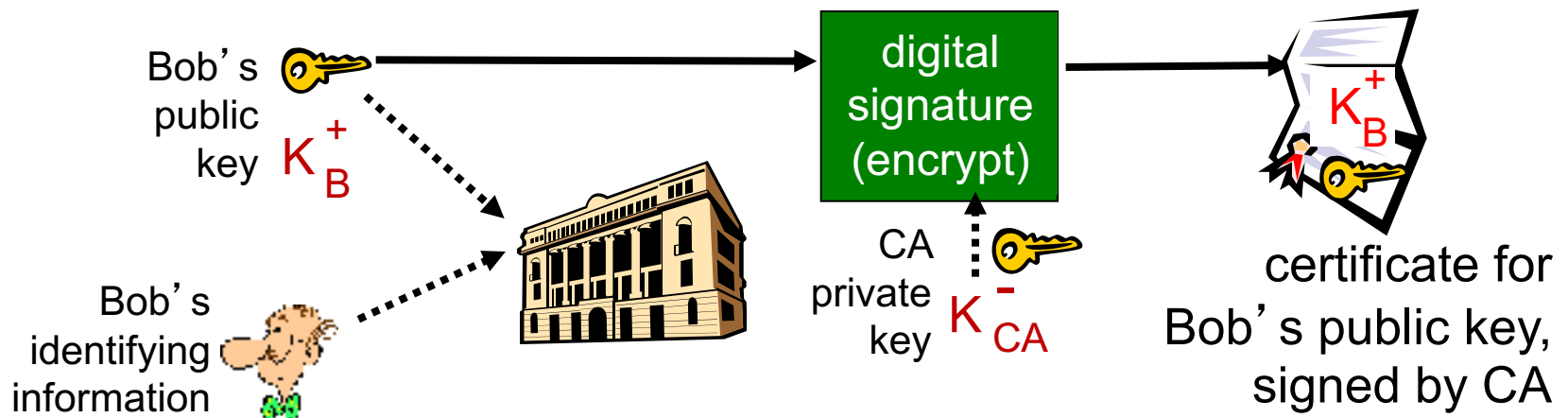
- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation!)
- problem is that Trudy receives all messages as well!

Public-key certification

- 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

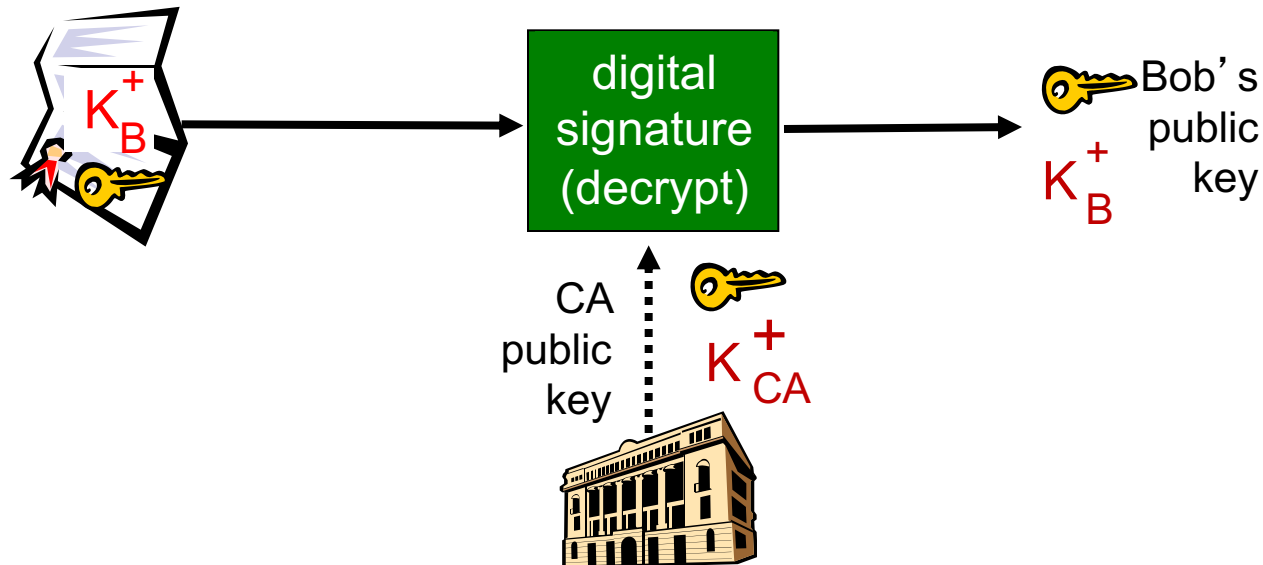
Certification authorities

- *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



Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity

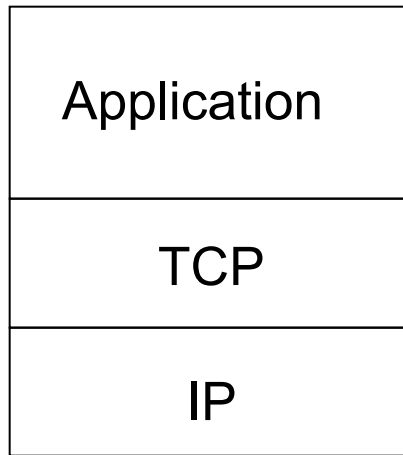
8.4 Securing TCP connections: SSL

8.5 Network layer security: IPsec

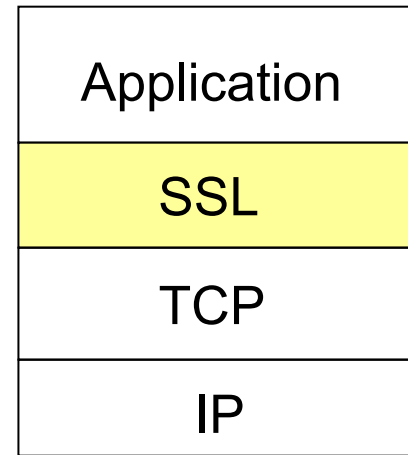
SSL: Secure Sockets Layer

- 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 problems in doing business with new merchant
- available to all TCP applications
 - secure socket interface

SSL and TCP/IP



normal application



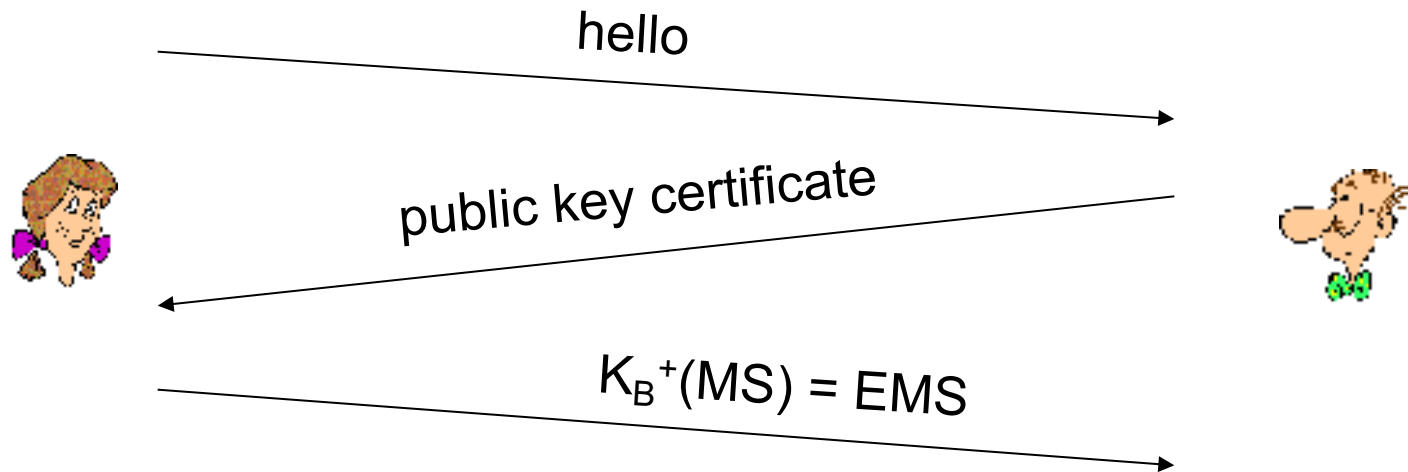
application with SSL

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

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



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:
 - $MAC = MAC(M_x, \text{sequence}||\text{data})$
 - 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
- $MAC = MAC(M_x, \text{sequence} || \text{type} || \text{data})$

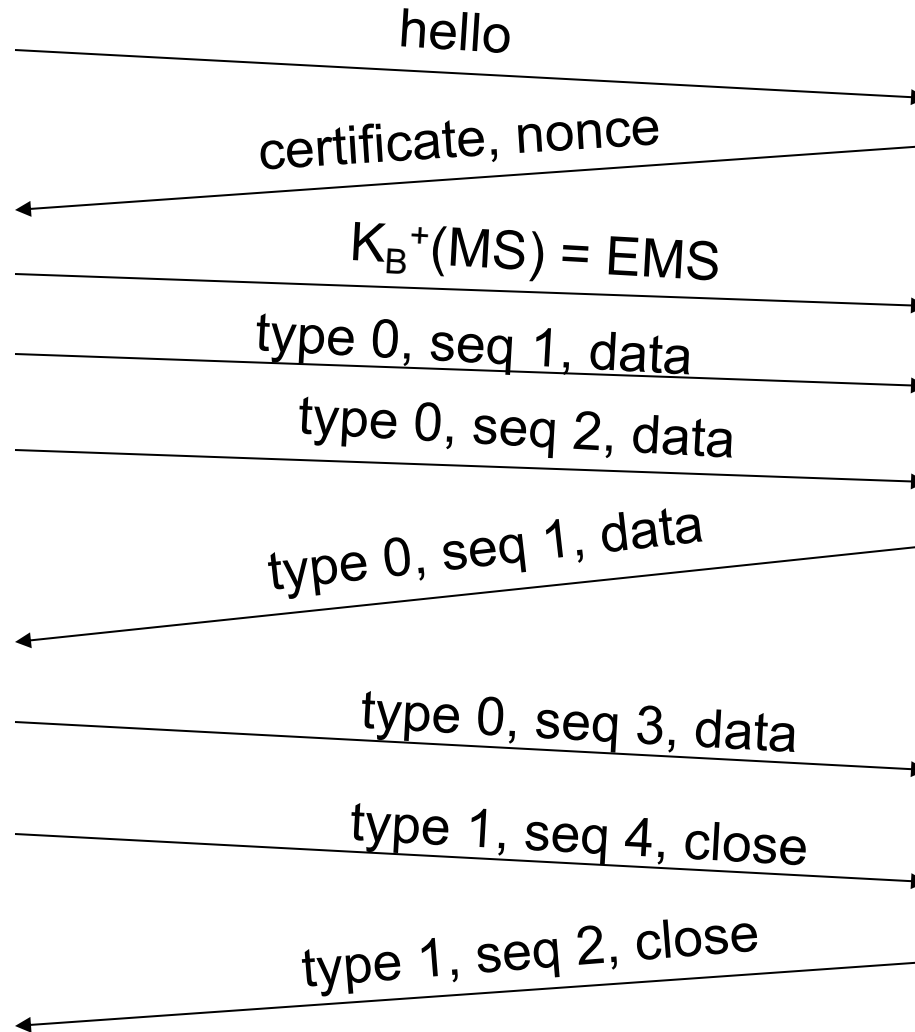


Toy SSL: summary



bob.com

encrypted



Real SSL: handshake (I)

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)

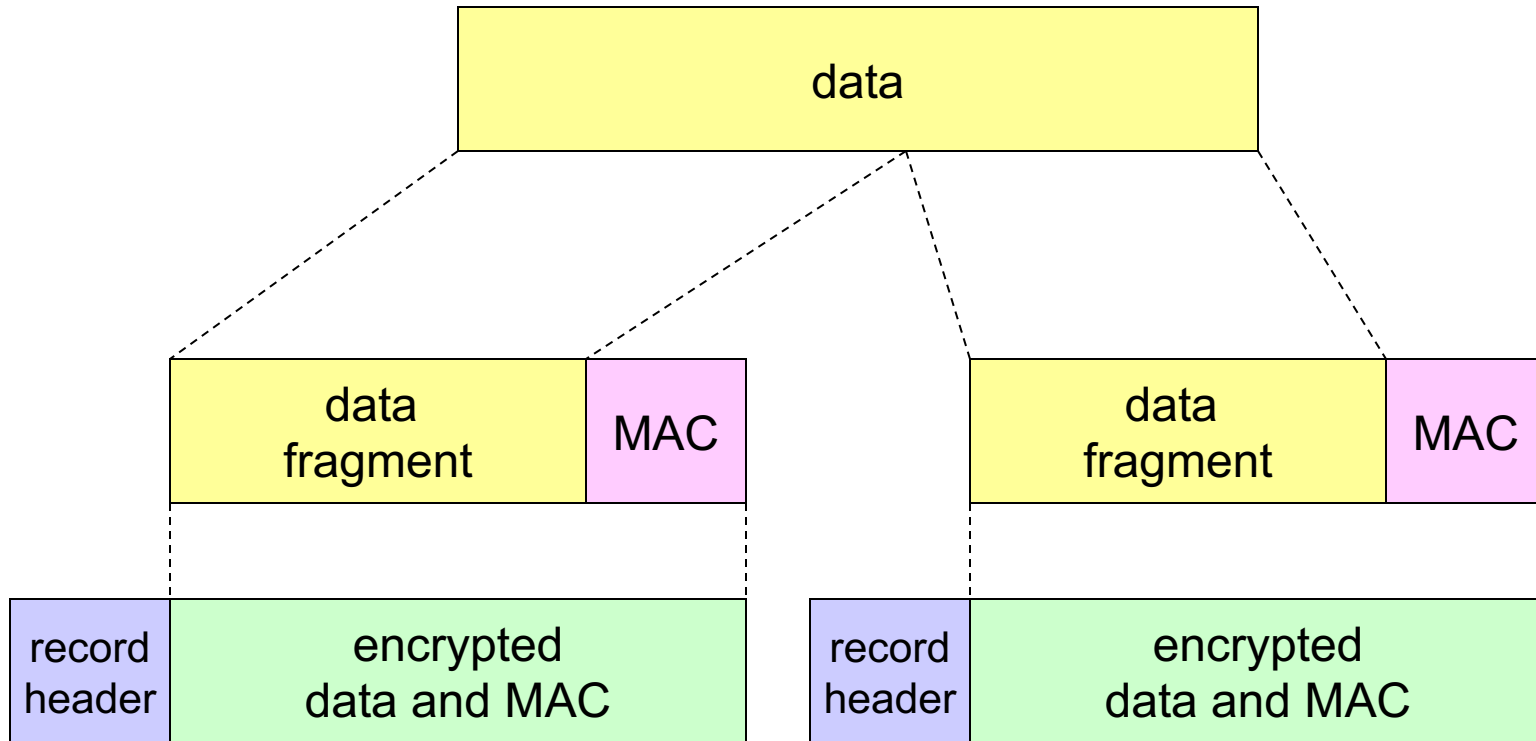
last 2 steps protect handshake from tampering

- 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

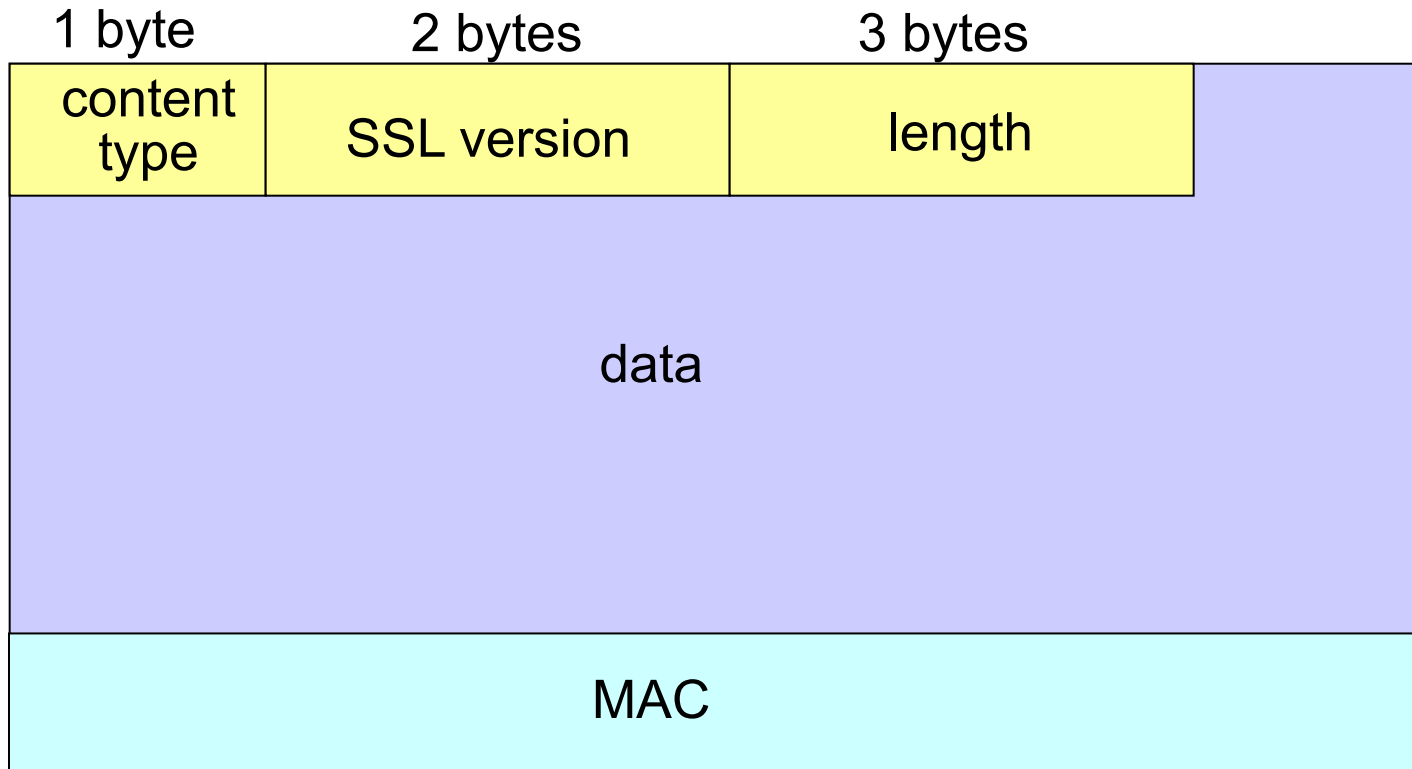


record header: content type; version; length

MAC: includes sequence number, MAC key M_x

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

SSL record format

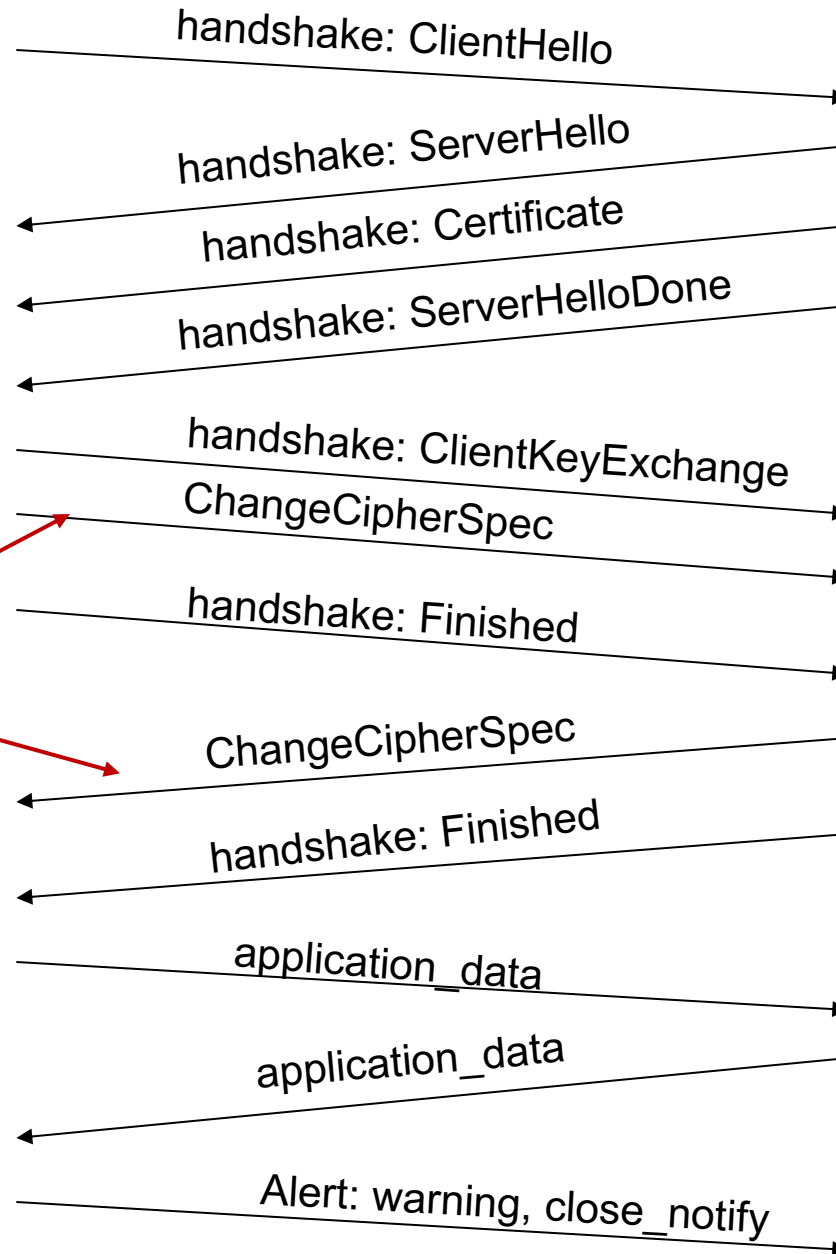


data and MAC encrypted (symmetric algorithm)

Real SSL connection

*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”
- key block contains:
 - client MAC key
 - server MAC key
 - client encryption key
 - server encryption key
 - client initialization vector (IV) (used by the encryption schema initialization)
 - server initialization vector (IV) (used by the encryption schema initialization)