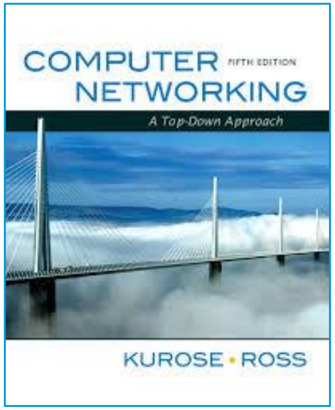
CSC358 *Intro. to Computer Networks*

Lecture 12: *Network Security, Exam Prep*

Amir H. Chinaei, Winter 2016

ahchinaei@cs.toronto.edu
http://www.cs.toronto.edu/~ahchinaei/



Many slides are (inspired/adapted) from the above source

Office Hours: T 17:00–18:00 R 9:00–10:00 BA4222

TA Office Hours: W 16:00-17:00 BA3201 R 10:00-11:00 BA7172
csc358ta@cdf.toronto.edu
http://www.cs.toronto.edu/~ahchinaei/teaching/2016jan/csc358/

Review

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

- sender encrypts message
- receiver decrypts message

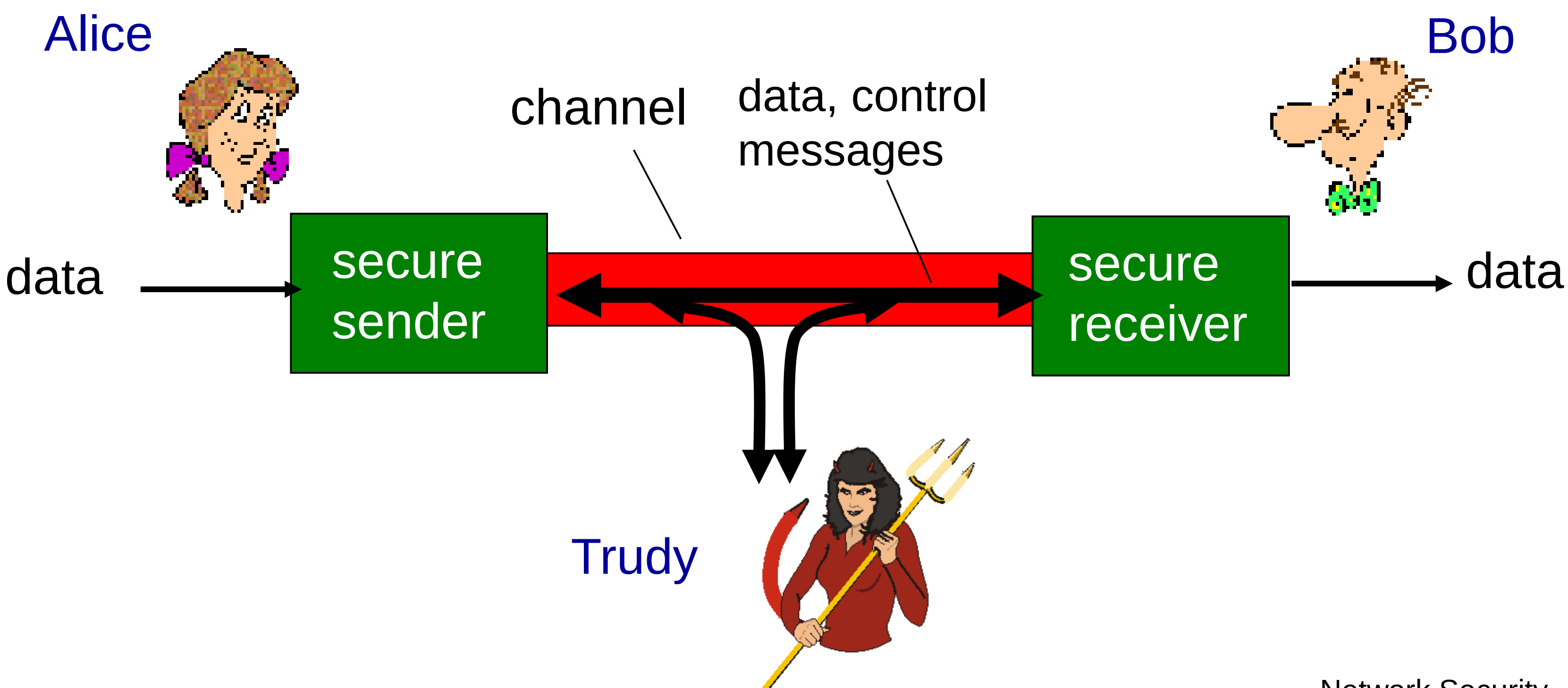
authentication: sender, receiver want to confirm identity of each other

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



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?

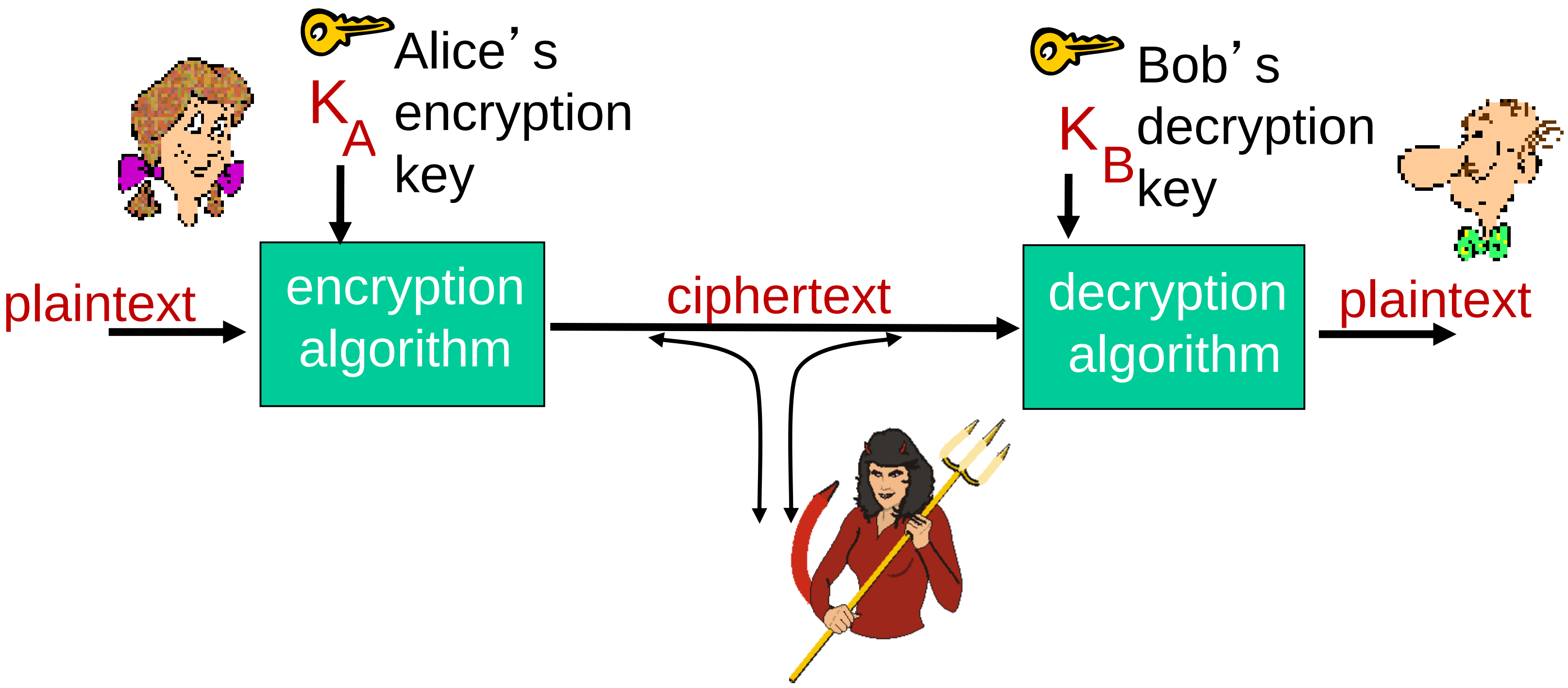
There are bad guys (and girls) out there!

Q: What can a "bad guy" do?

A: A lot!

- *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

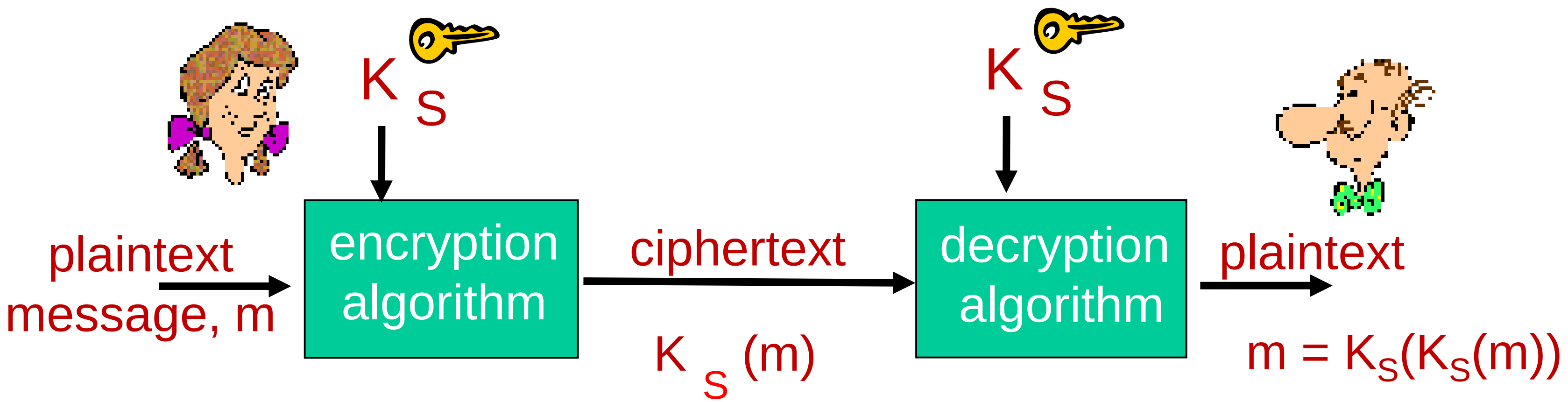
$K_A(m)$ ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

Breaking an encryption scheme

- 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?

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 subsitution 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

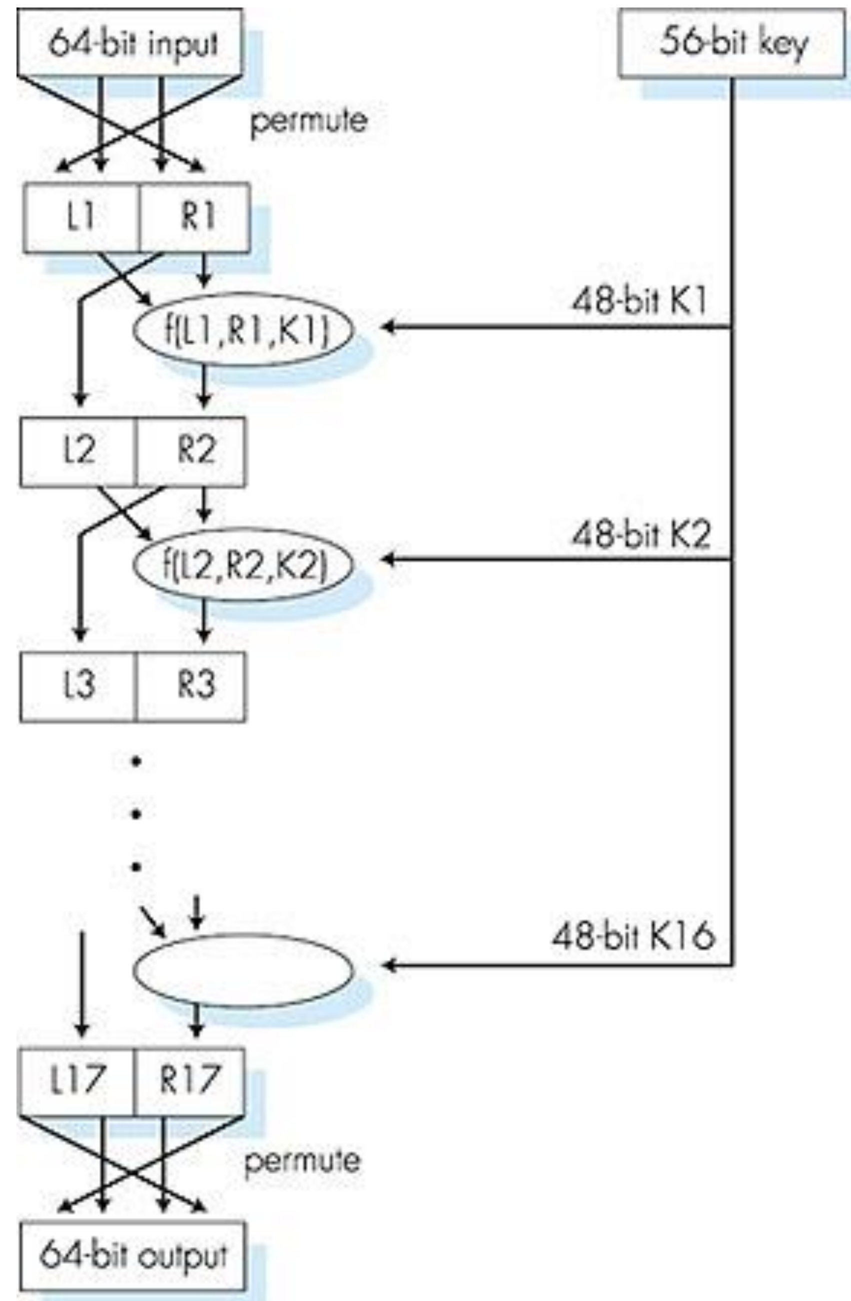
DES: Data Encryption Standard

- 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

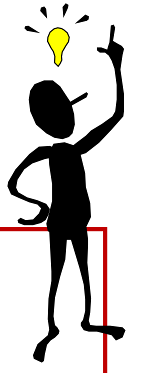
- initial permutation
- 16 identical "rounds" of function application, each using different 48 bits of key
- final permutation



AES: Advanced Encryption Standard

- symmetric-key NIST standard, replacied 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



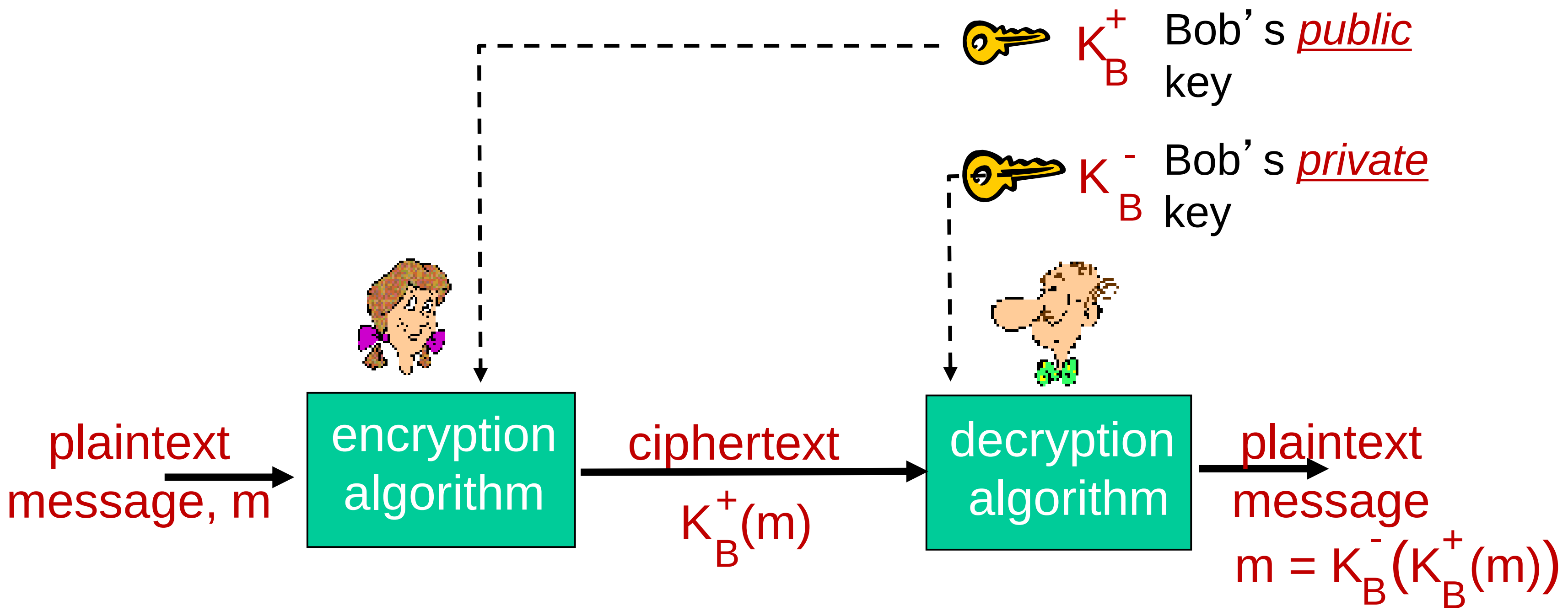
symmetric key crypto

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

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

  a^d mod n = $(a \text{ mod } n)^d$ mod n
- example: x=14, n=10, d=2:

  $(x \text{ mod } n)^d$ mod n = 4^2 mod 10 = 6

  $x^d = 14^2 = 196$ x^d mod 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 (n,e). *private* key is (n,d).

K_B^+ 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.

	bit pattern	m	m^e	$c = m^e \bmod n$
encrypt:	00001100	12	24832	17

	c	c^d	$m = c^d \bmod n$
decrypt:	17	481968572106750915091411825223071697	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,
  $c^d \bmod n = (m^e \bmod n)^d \bmod n$
  $= m^{ed} \bmod n$
  $= m^{(ed \bmod z)} \bmod n$
  $= m^1 \bmod n$
  $= m$

RSA: another important property

The following property will be *very* useful later:

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

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:

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

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

Chapter 8 roadmap

Authentication

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

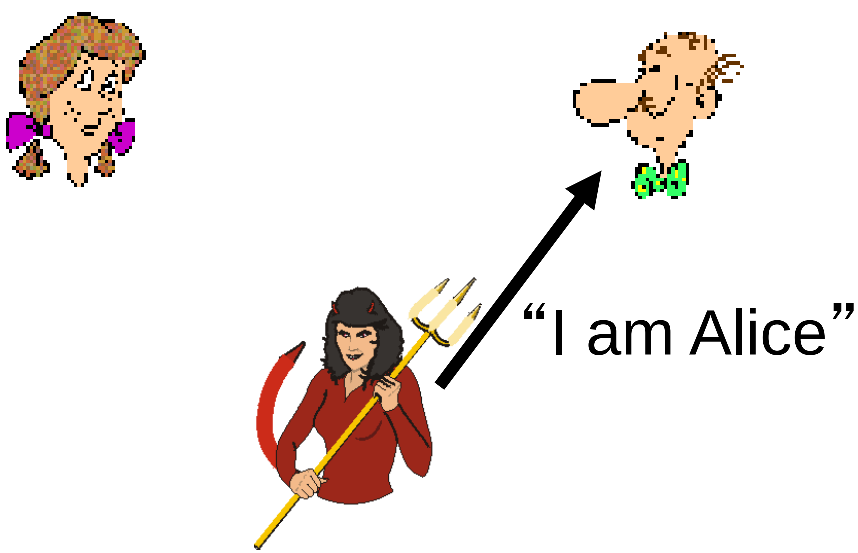
Protocol ap1.0: Alice says "I am Alice"

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

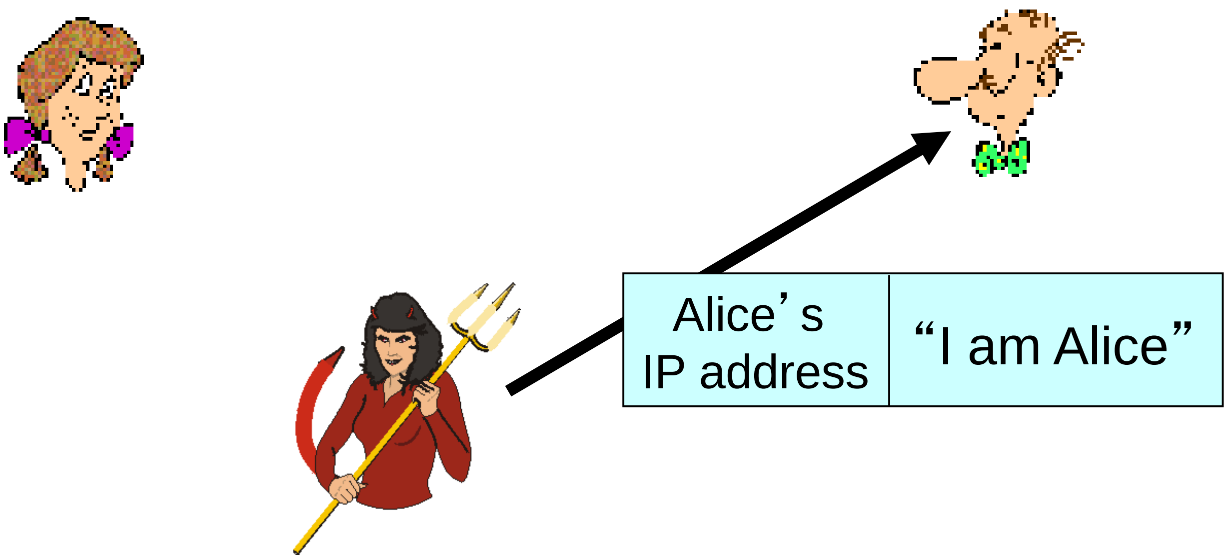
Authentication: another try

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

Failure scenario??

Authentication: another try

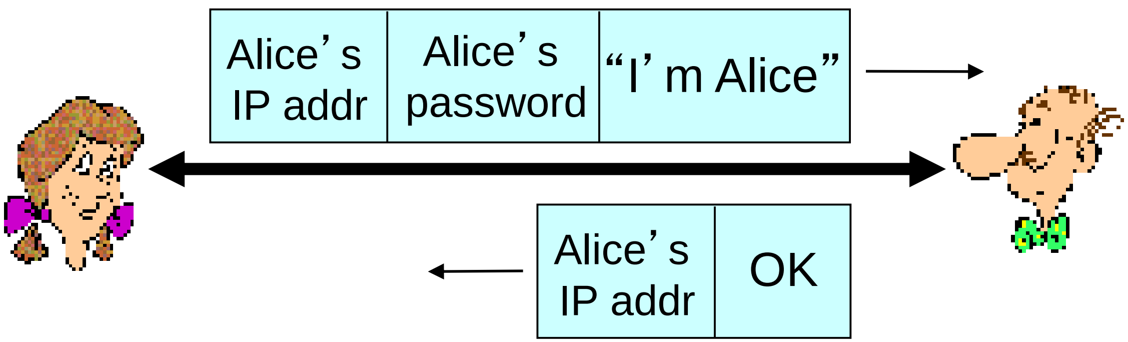
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



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

Authentication: another try

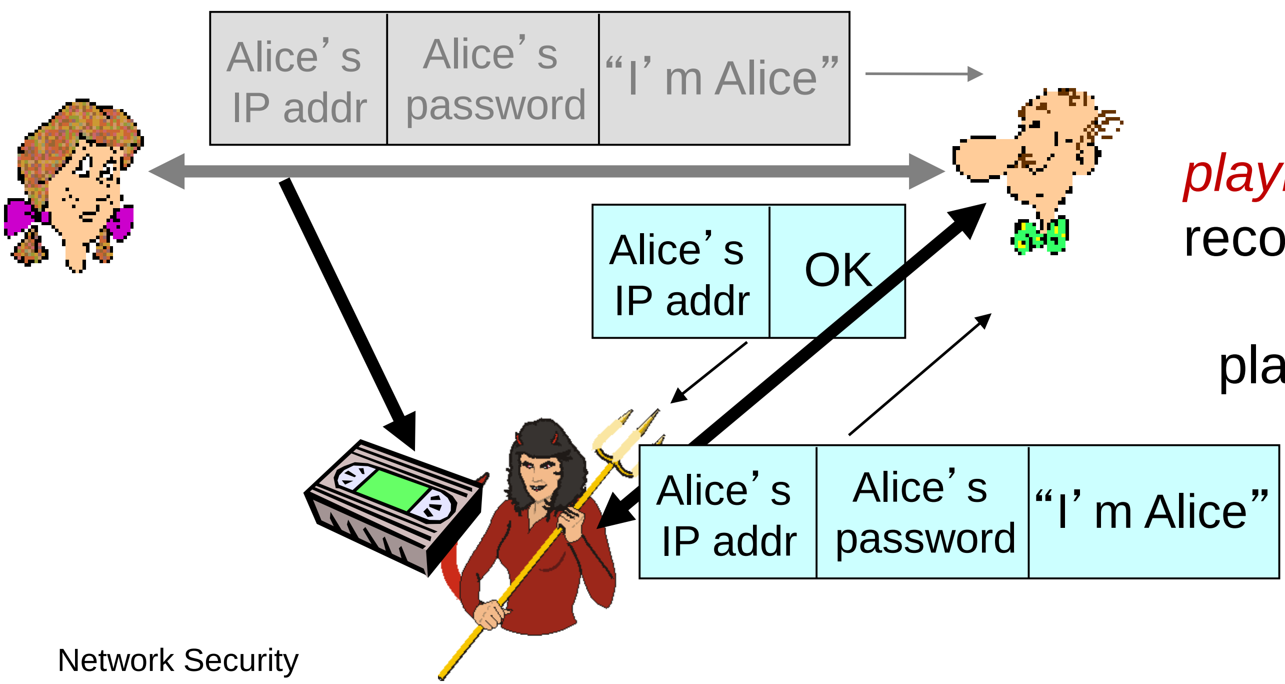
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Failure scenario??

Authentication: another try

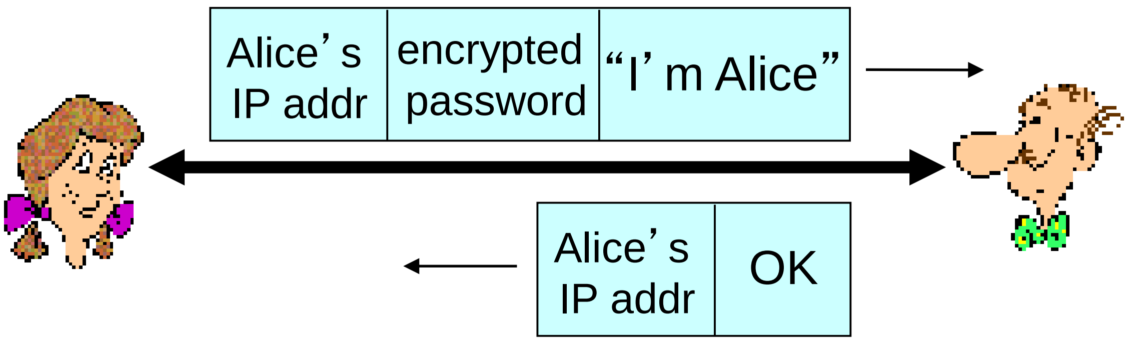
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



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

Authentication: yet another try

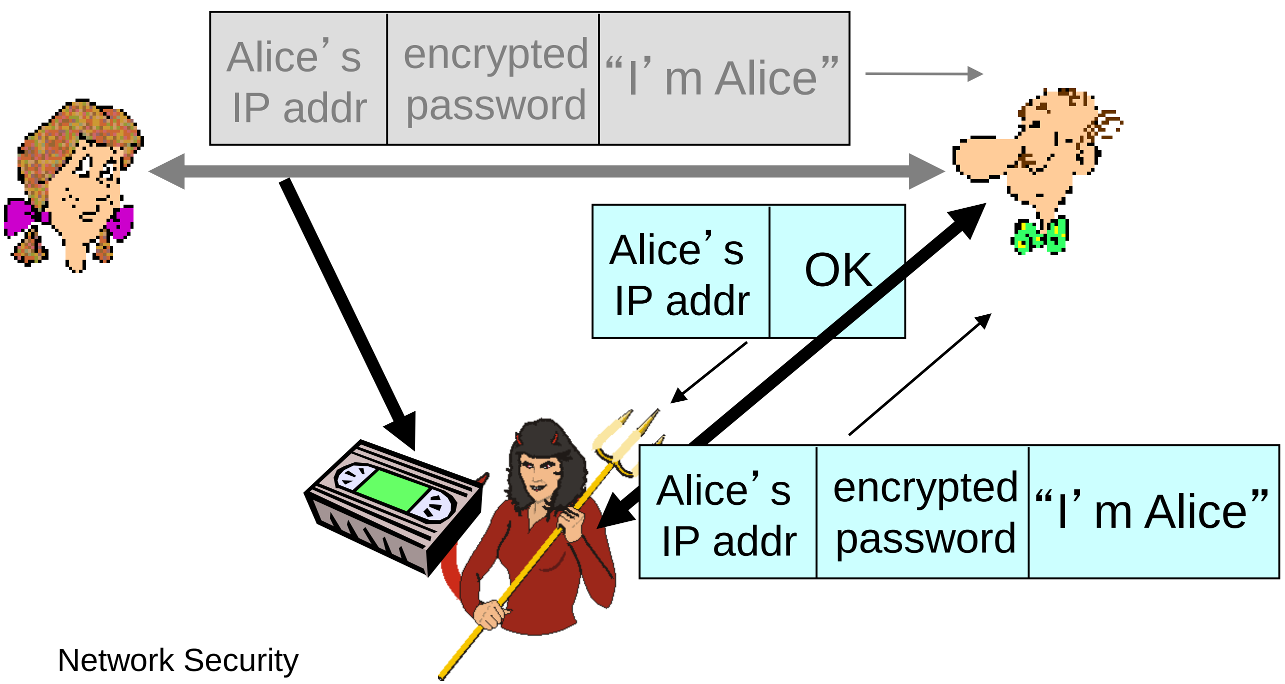
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: yet another try

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



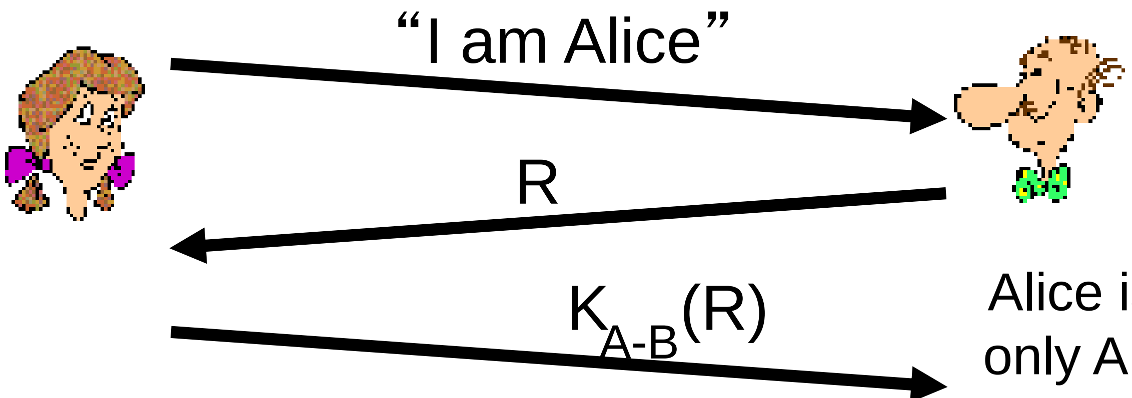
record and playback *still* works!

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!

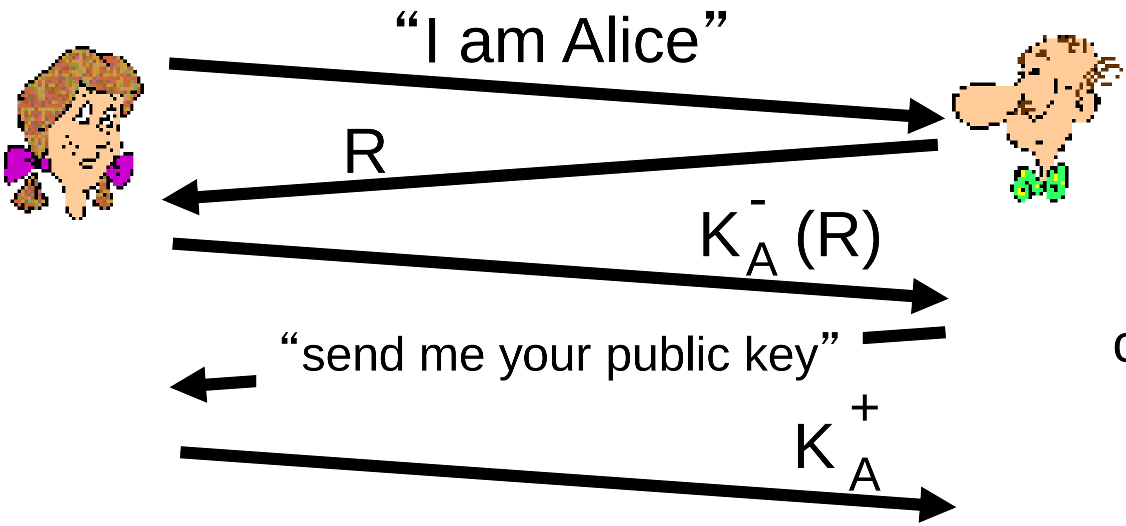
Failures, drawbacks?

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

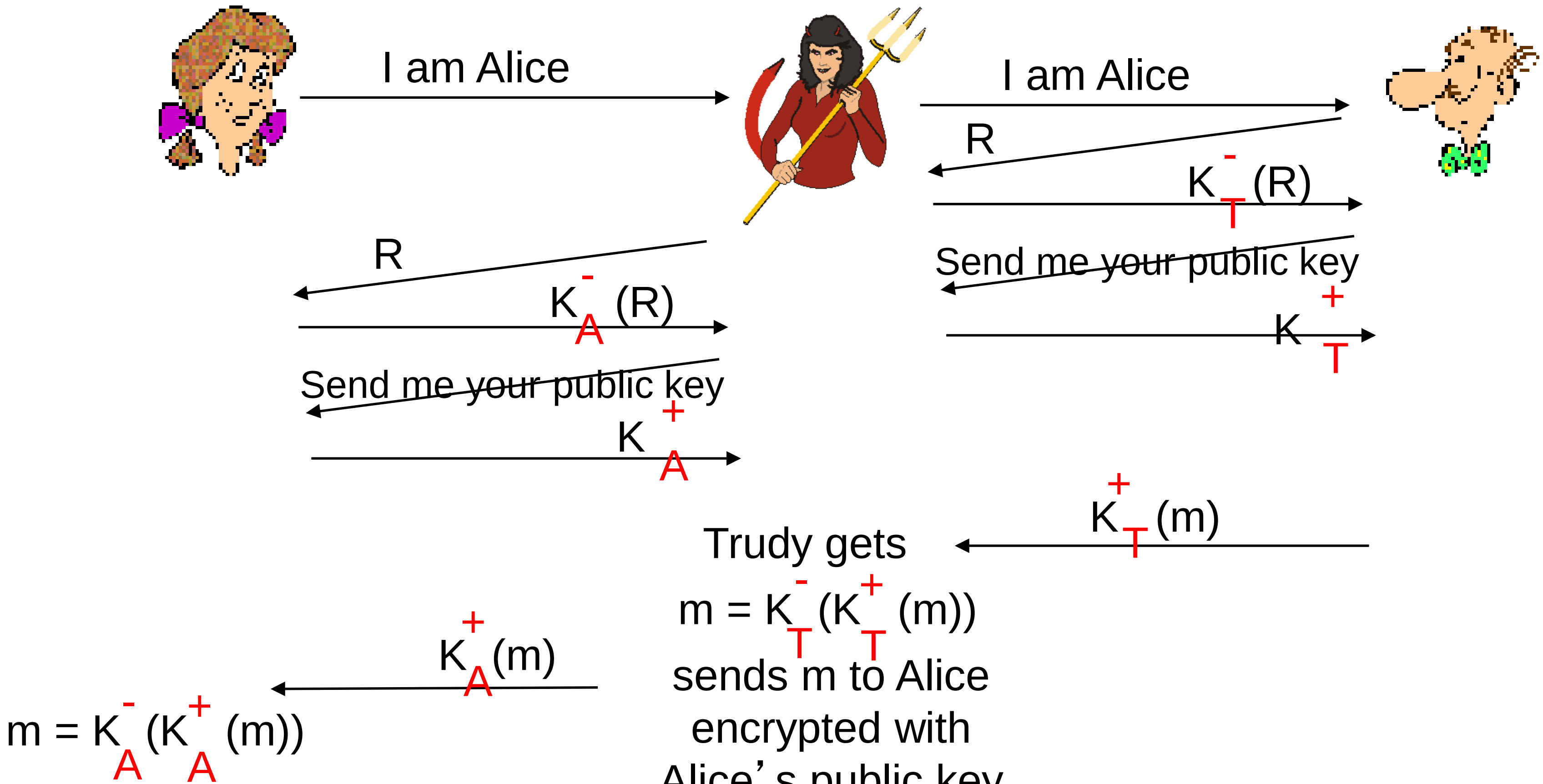
$$K_A^+(K_A^-(R)) = R$$

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

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

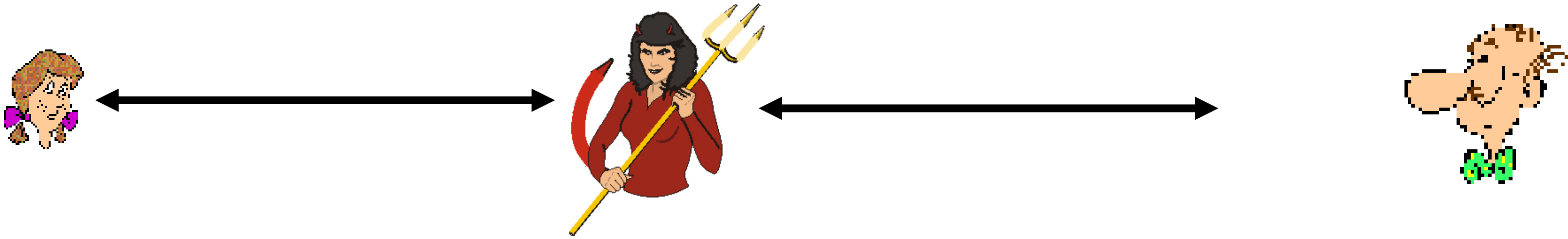
ap5.0: security hole

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!

Digital signatures

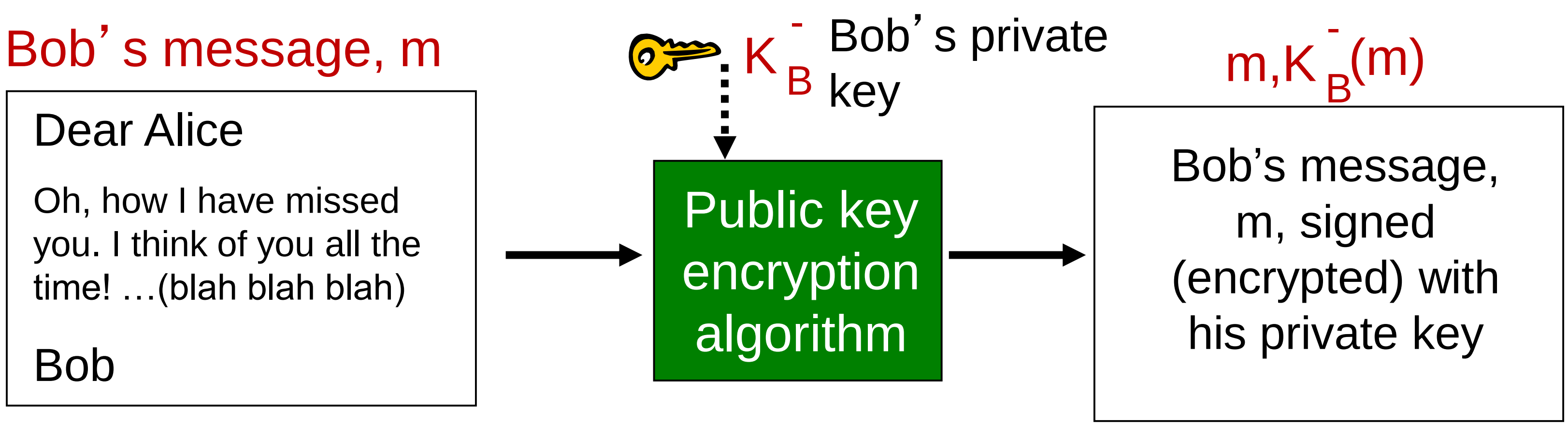
cryptographic technique analogous to hand-written signatures:

- 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.

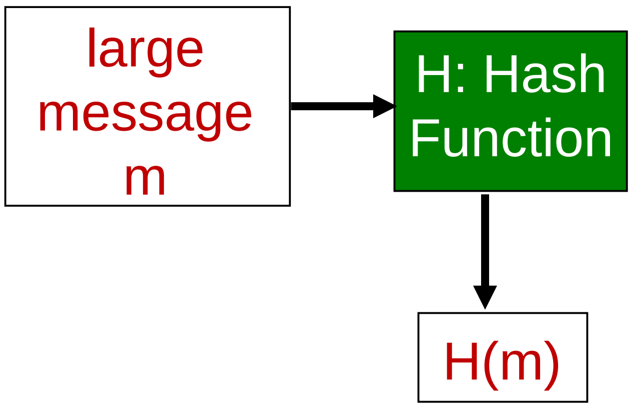
Alice thus verifies that:

- ✓ 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)

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

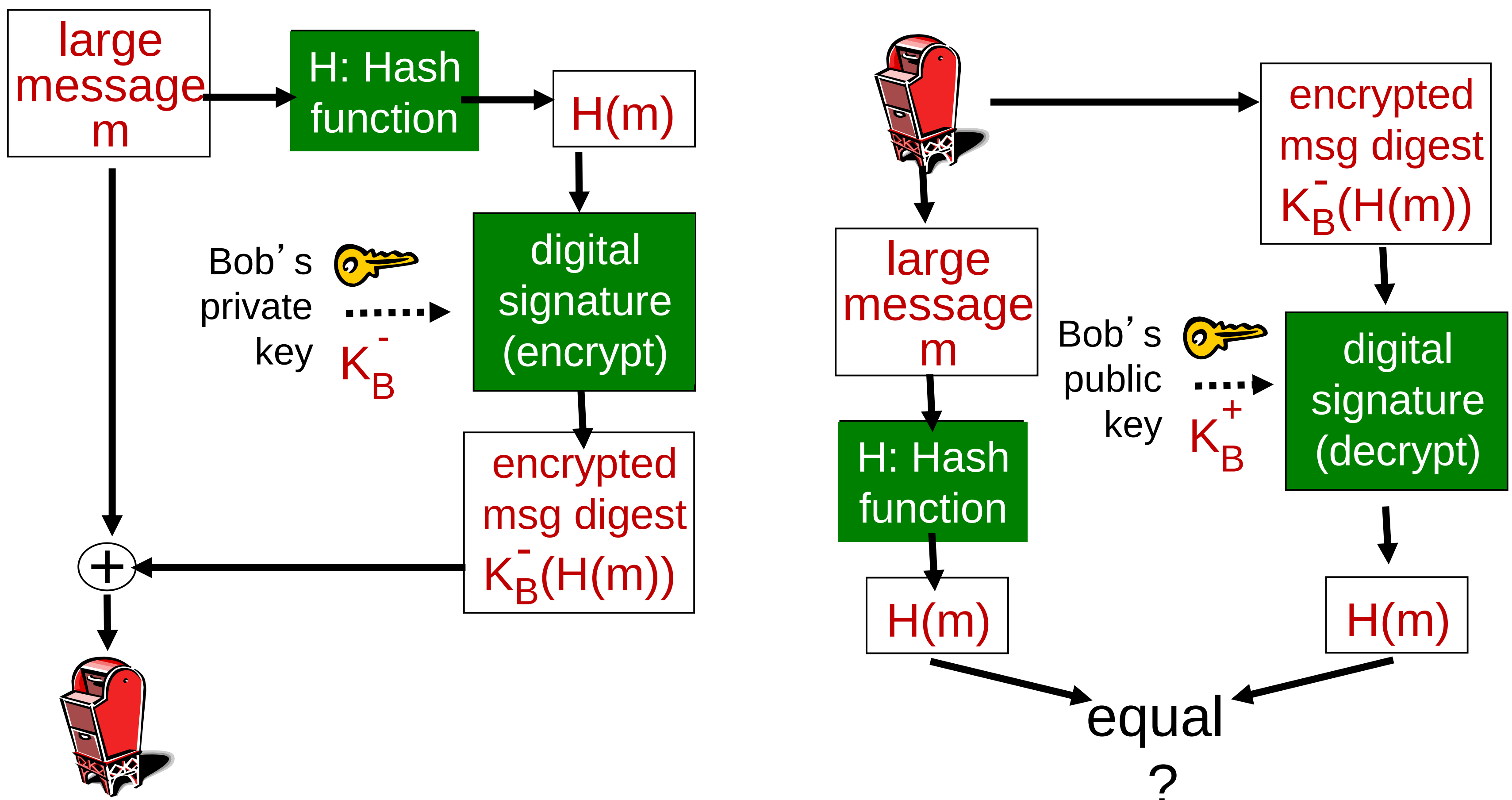
message	ASCII format
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 D2 42
	B2 C1 D2 AC

different messages but identical checksums!

Digital signature = signed message digest

Bob sends digitally signed message:

Alice verifies signature, integrity of digitally signed message:

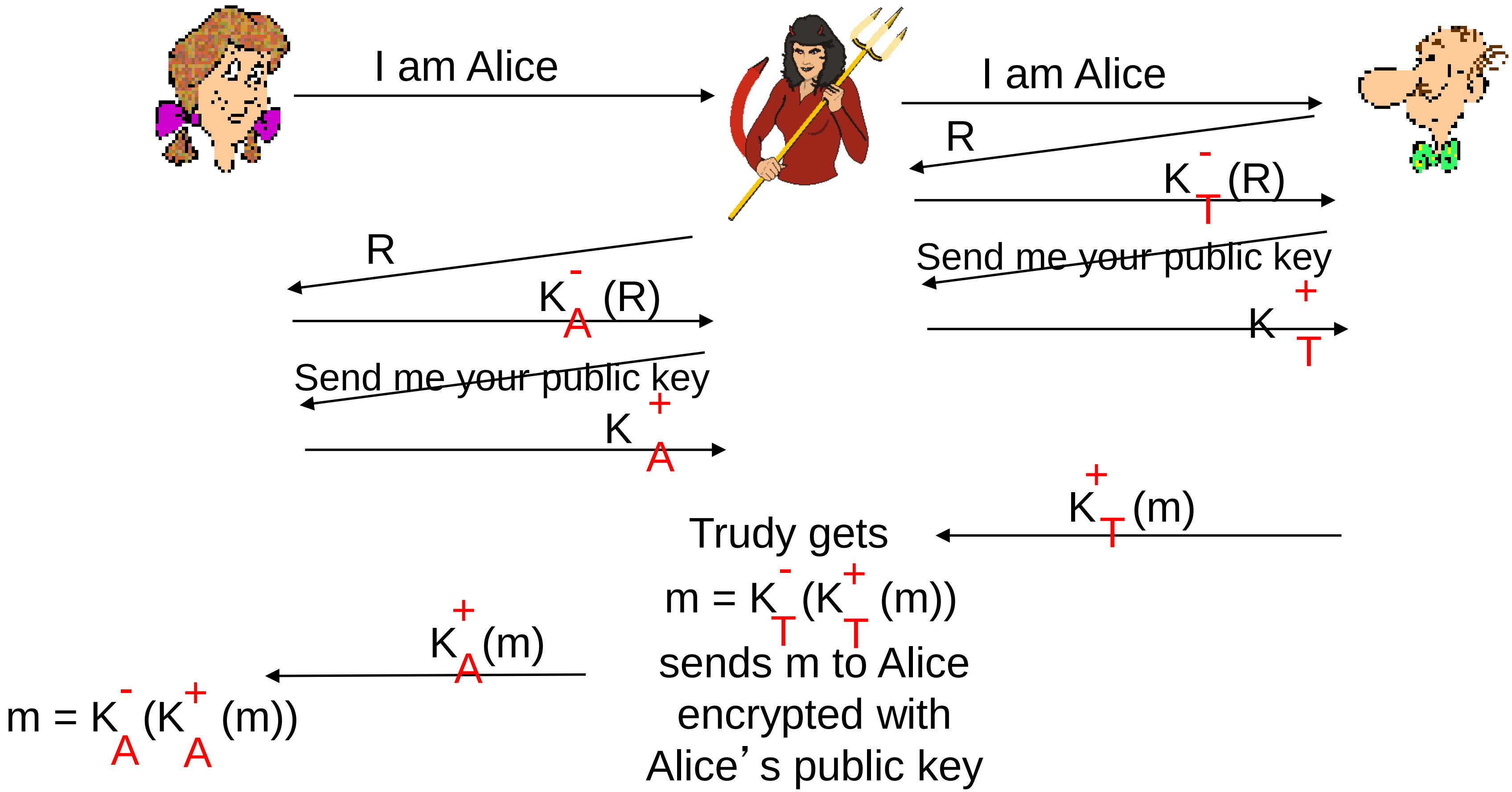


Hash function algorithms

- 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)

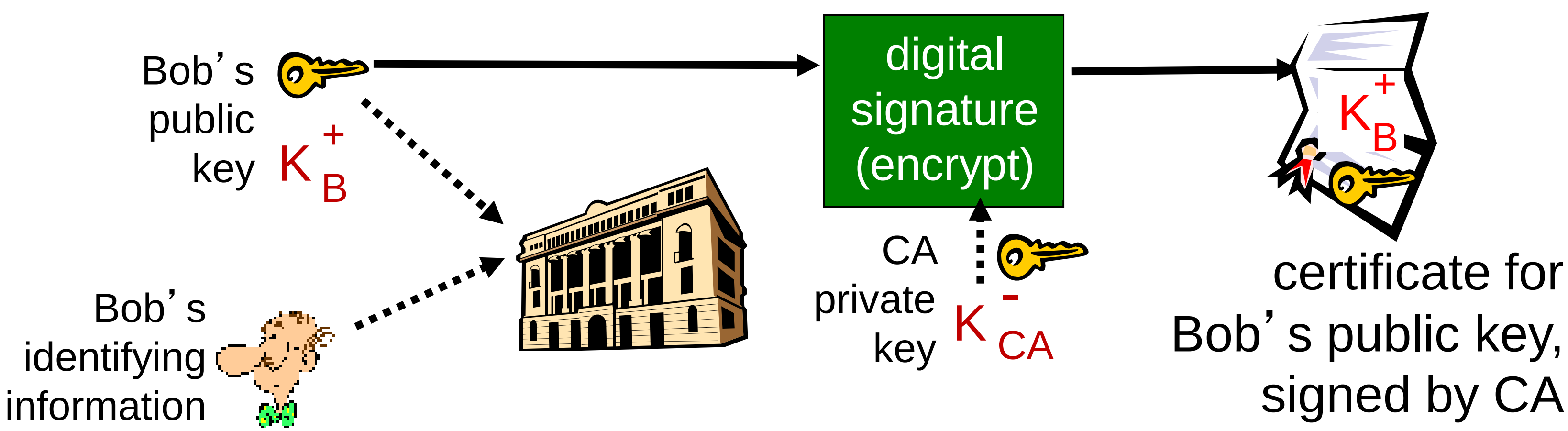


Public-key certification

- 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"

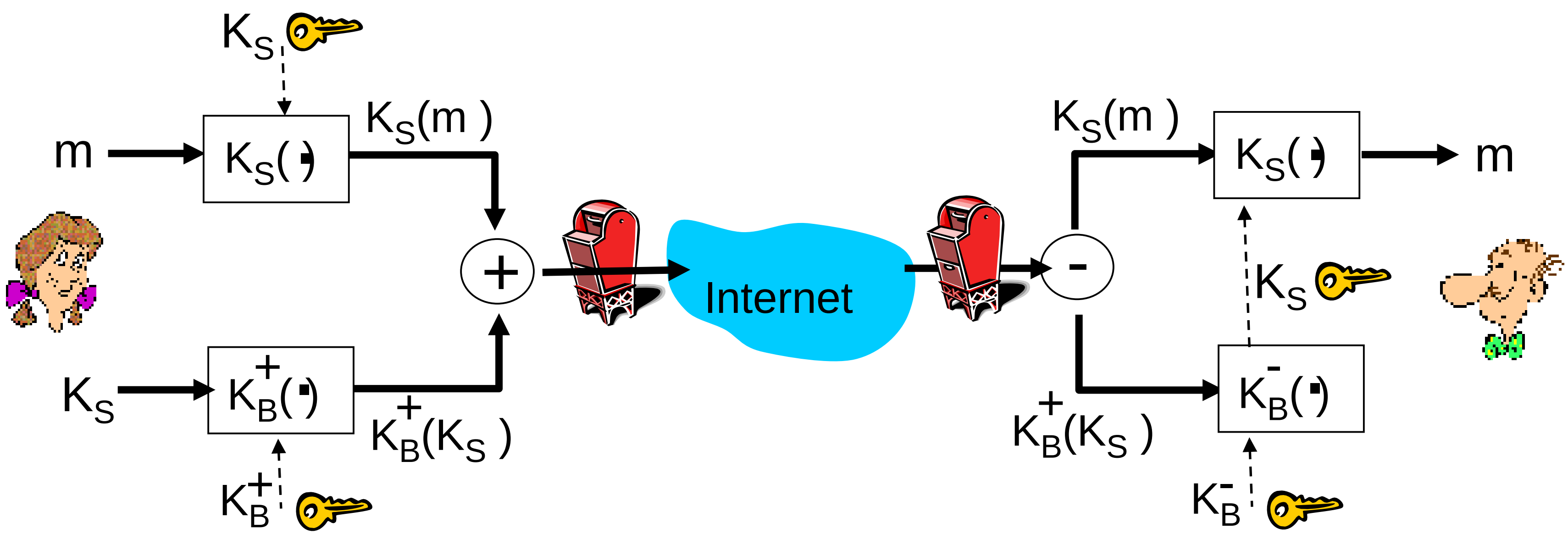


Chapter 8 roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity, authentication
8.4 Securing e-mail
8.5 Securing TCP connections: SSL
8.6 Network layer security: IPsec
8.7 Securing wireless LANs
8.8 Operational security: firewalls and IDS

Secure e-mail

- Alice wants to send confidential e-mail (**secrecy**), m, to Bob.

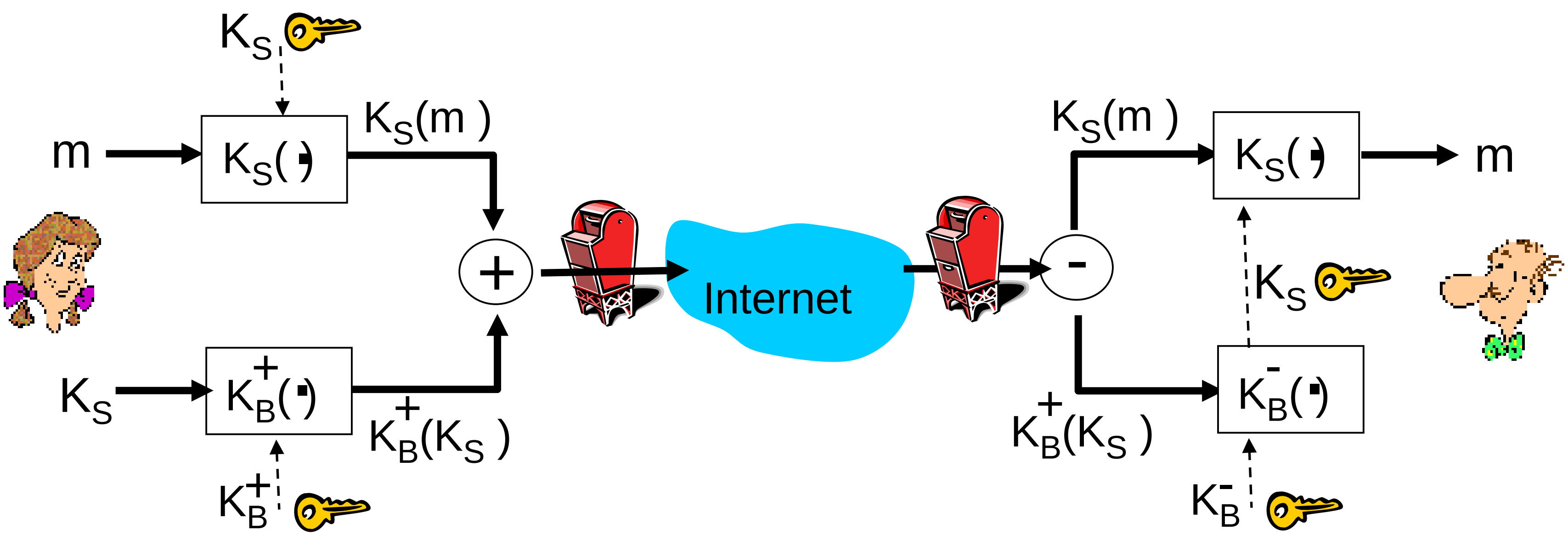


Alice:

- 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 (**secrecy**), m, to Bob.

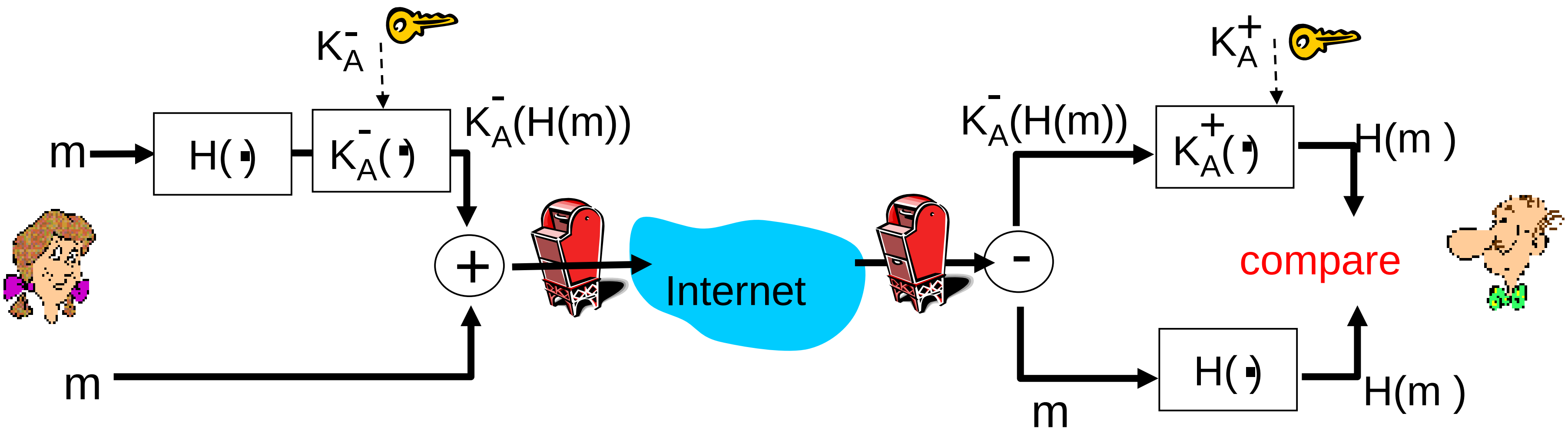


Bob:

- 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)

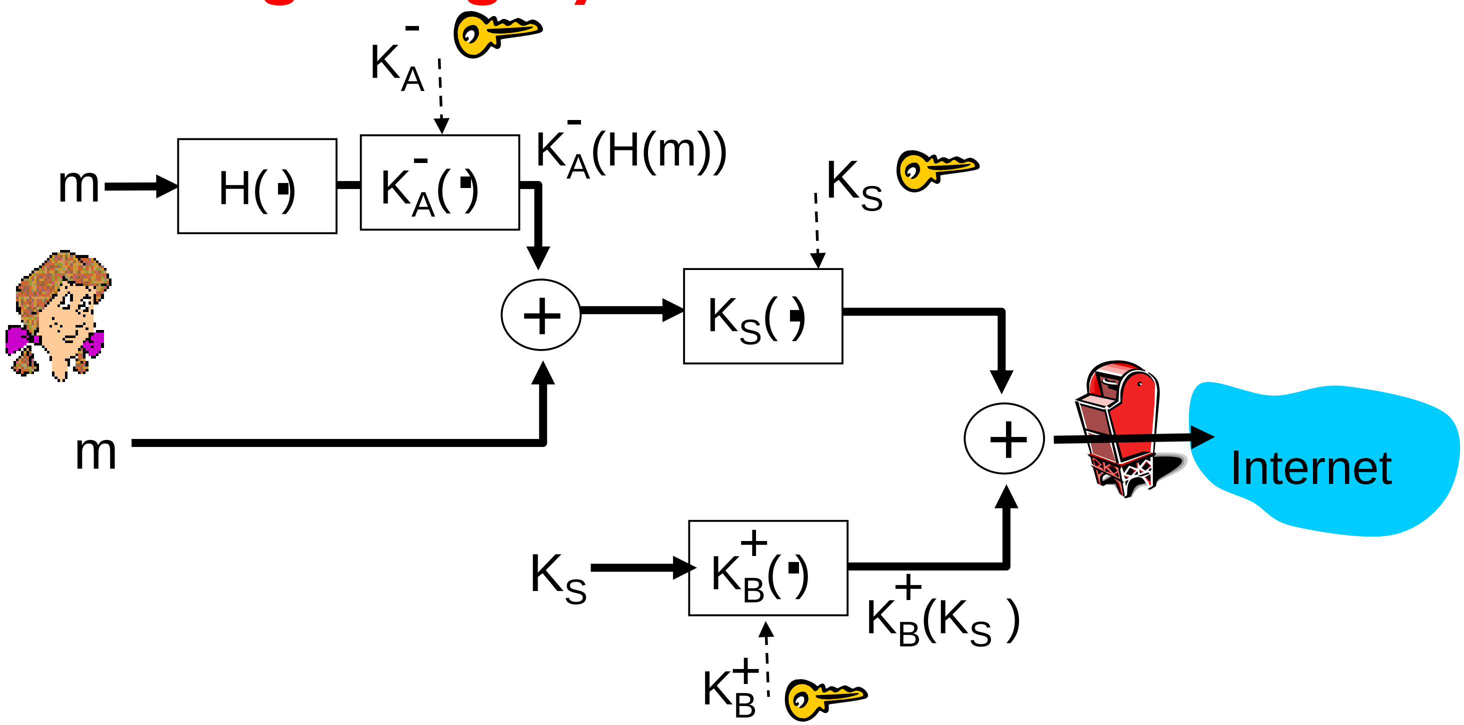
- Alice wants to provide **sender authentication** and **message integrity**



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

Secure e-mail (continued)

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



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

Chapter 8 roadmap

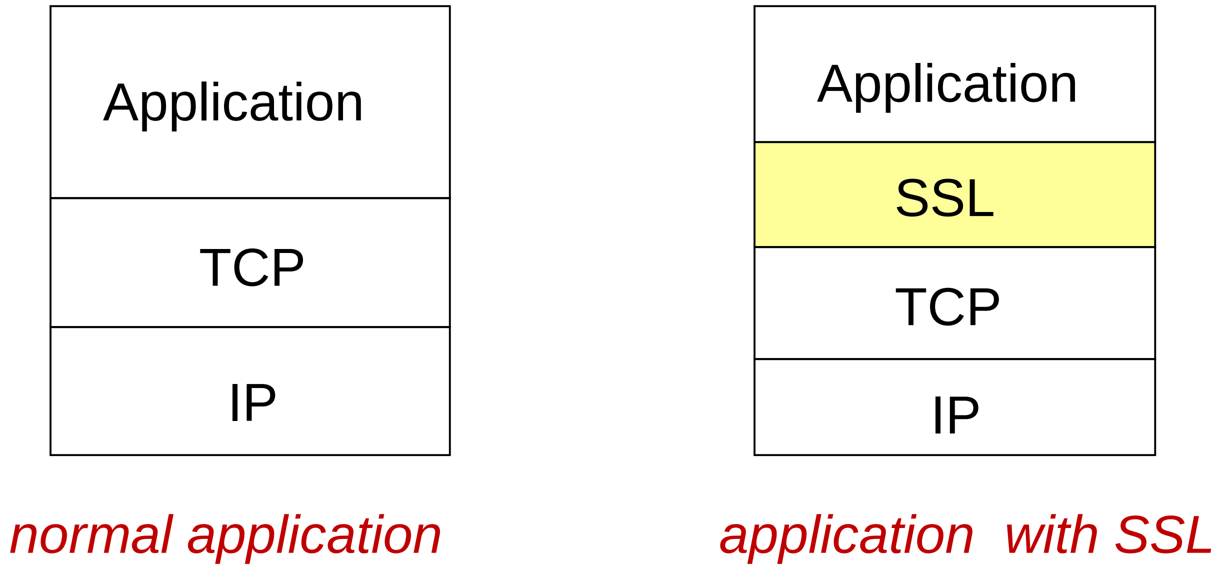
8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity
8.4 Securing e-mail
8.5 Securing TCP connections: SSL
8.6 Network layer security: IPsec
8.7 Securing wireless LANs
8.8 Operational security: firewalls and IDS

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*

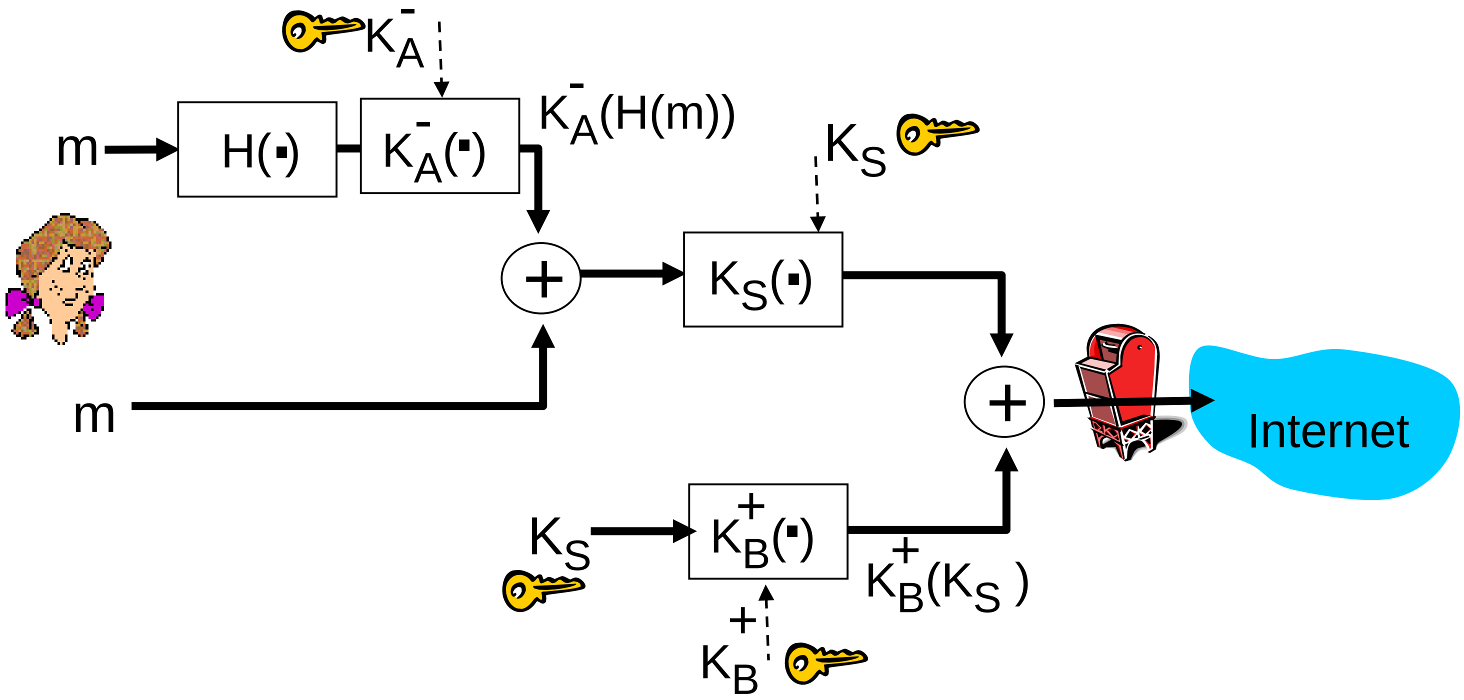
- original goals:
  - Web e-commerce transactions
  - encryption (especially credit-card numbers)
  - Web-server authentication
  - optional client authentication
  - minimum hassle in doing business with new merchant
- available to all TCP applications
  - secure socket interface

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:

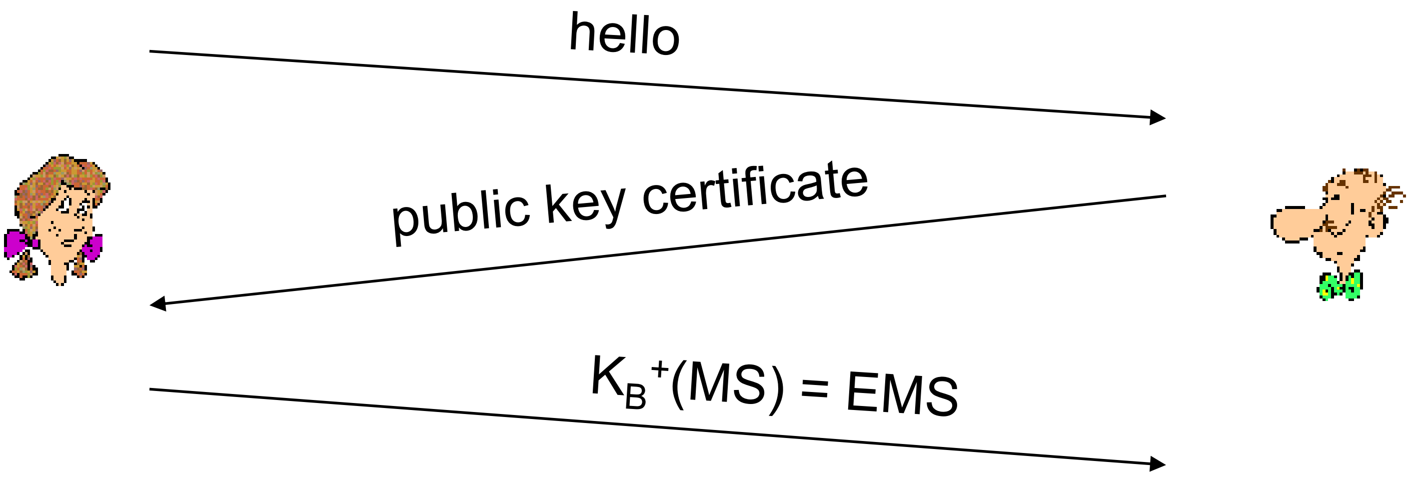


- but want to send byte streams & interactive data
- want set of secret keys for entire connection
- want certificate exchange as part of protocol: handshake phase

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

Big Picture: a simple handshake



MS: master secret

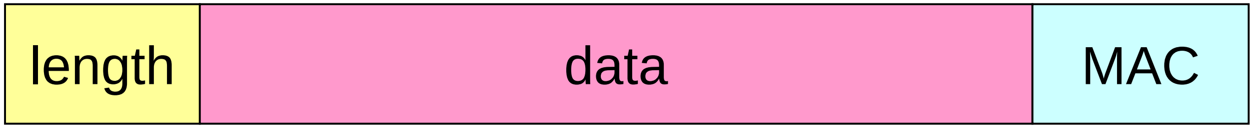
EMS: encrypted master secret

Big Picture: 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

Big Picture: 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



Big Picture: sequence numbers

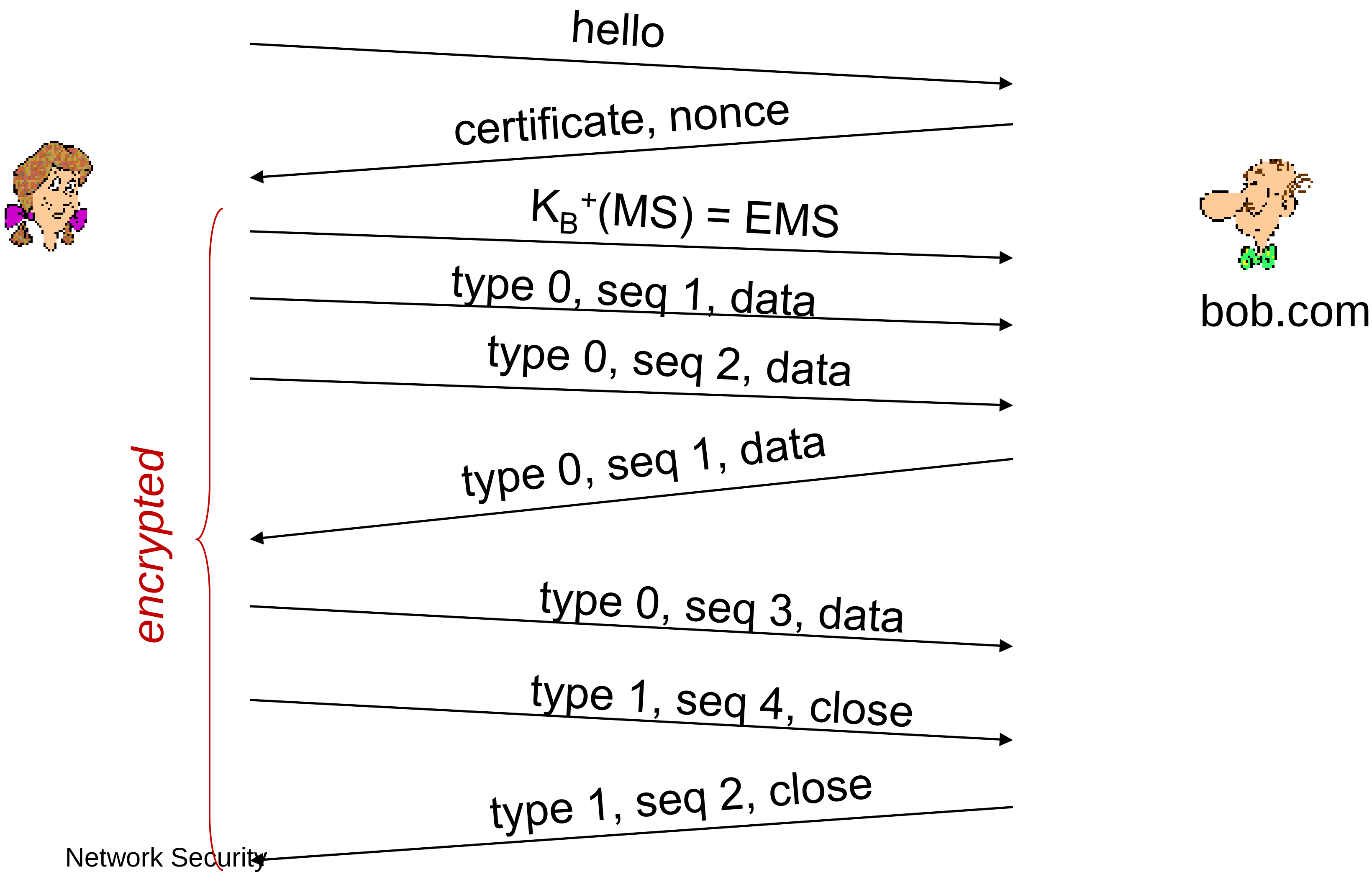
- *problem:* attacker can capture and replay record or re-order records
- *solution:* put sequence number into MAC:
  - MAC = MAC(M_x, sequence||data)
  - note: no sequence number field

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

Big Picture: 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)

SSL: Big Picture summary



Final Exam Prep

Final exam: cover page

UNIVERSITY OF TORONTO
Faculty of Arts and Science

April 2016 Examinations

CSC358H1S
Introduction to Computer Networks

Duration—2 hours

No Aids Allowed

You must achieve at least 50% of this exam or 50% of the weighted average of this exam and the midterm, to pass this course.

Student # ---

First Name ---

Last Name ---

This exam consists of 7 questions for a total of 50 points.

Write neatly and concisely. If we cannot read it, we cannot grade it.

You will earn 20% for any question you leave blank or write "I cannot answer this question".

# 1	# 2	# 3	# 4	# 5	# 6	# 7	Total
/12	/5	/5	/8	/8	/8	/4	/50

Good Luck!

Final exam: questions distribution

- The structure is similar to that of the midterm.
- 7 questions for a total of 50 points
- #1 (12 points, 24% of the exam)
  - *Mostly concepts from Chapters 1, 2, 3, 4, 5, and 8*
- #2, #3 (5 points, 10% of the exam, each)
  - *Detailed questions on Chapters 1 and 2 (pre-midterm)*
- #4, #5, #6 (8 points, 16% of the exam, each)
  - *Detailed questions on Chapters 3, 4, and 5*
- #7 (4 points, 8% of the exam)
  - *Detailed questions on Chapter 8 (8.1-8.5)*

Final exam: approach/final answer

- Most questions require to calculate the final answer.
  - This is, in fact, good!
  - Relatively simple numbers and calculations are required.
  - If you end up in complicated calculations, you can conclude that you are probably in a wrong track.

- Also, a final answer with a missing or wrong approach/justification does not receive points.

- Write neatly and concisely, such that you do not lose points unnecessarily.

Final exam: 50% rule, difficulty

- **Remember: you are required to earn 50% of the final exam or 50% of the weighted average of the midterm and final exam to pass the course.**
  - Example: if a student receives perfect points in all assignments and have collected several bonus points, but has not earned at least 50% of the above, he/she will receive an F in the course.

- The exam is **long & difficult** for students who are not prepared; and, it's fair & doable in ~ an hour for others.

Final exam: preparation

- Similar to the midterm;
- In addition to preparation for pre-midterm part (refer to Lecture 5);
- Make sure you understand details/concepts of Assignments 3 to 5, Tutorials 5 to 11, reading from the book, and the following problems:
  - Ch3: even questions from P2-P40, as well as 41, 45, and 53
  - Ch4: even questions from P2-P40, as well as 43 and 49
  - Ch5: P2, P4, P10, P14, P18, P20, P26, P28, P32, P34 and P36
  - Ch8: P1-P12, P15-P18, P20-P22
  - Reference is the 5th edition

Last but not the least!

- If you want to do me a favour:
  - ??

- **Thanks and good luck!**