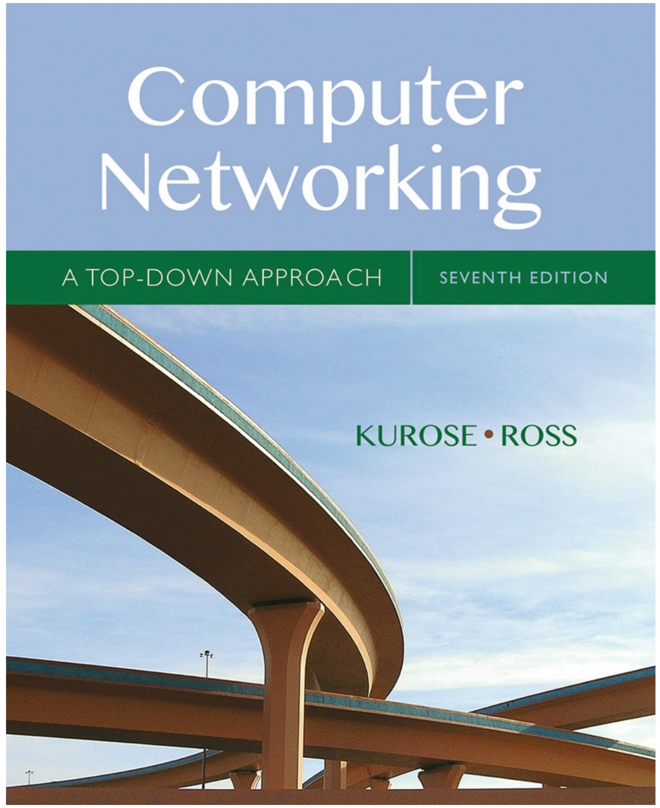
# COMP211
# Chapter 8
# Network Security






*Computer Networking: A Top Down Approach*
7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

# Network Security

## Our Goals:

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

# Outline

- Introduction
  - What is network security?
  - Why is network security important?
  - What are the requirements for a secure network?
  - An introduction to Cryptography
- Symmetric Key Cryptography
- Public Key Cryptography
- Authentication
- Integrity
- Security in Internet protocol stack

# Do we need network security?

- Internet and WWW computing standards (IP, HTTP, etc) are *public*
  - Therefore, intruders know about the types of messages being sent around the Internet
- The Internet is open and pervasive
- The Internet has many connecting components
  - A message sent between two computer will often pass through many others
  - Can we trust the others?

# There are bad guys (and girls) out there!

Q: What can a "bad guy" do?

A: A lot!

- *eavesdrop:* intercept messages (packet sniffing)
  - Traffic analysis
    - Collect (and sell) sensitive information
    - Guess data content by studying traffic patterns

Passive attack
Hard to detect

- *impersonation:* can fake (spoof) source address in packet (or any field in packet)
- man-in-the-middle attacks
  - actively *insert/modify/delete* messages into connection

Active attacks

- *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)

# What is network security?

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

- sender encrypts message
- receiver decrypts message

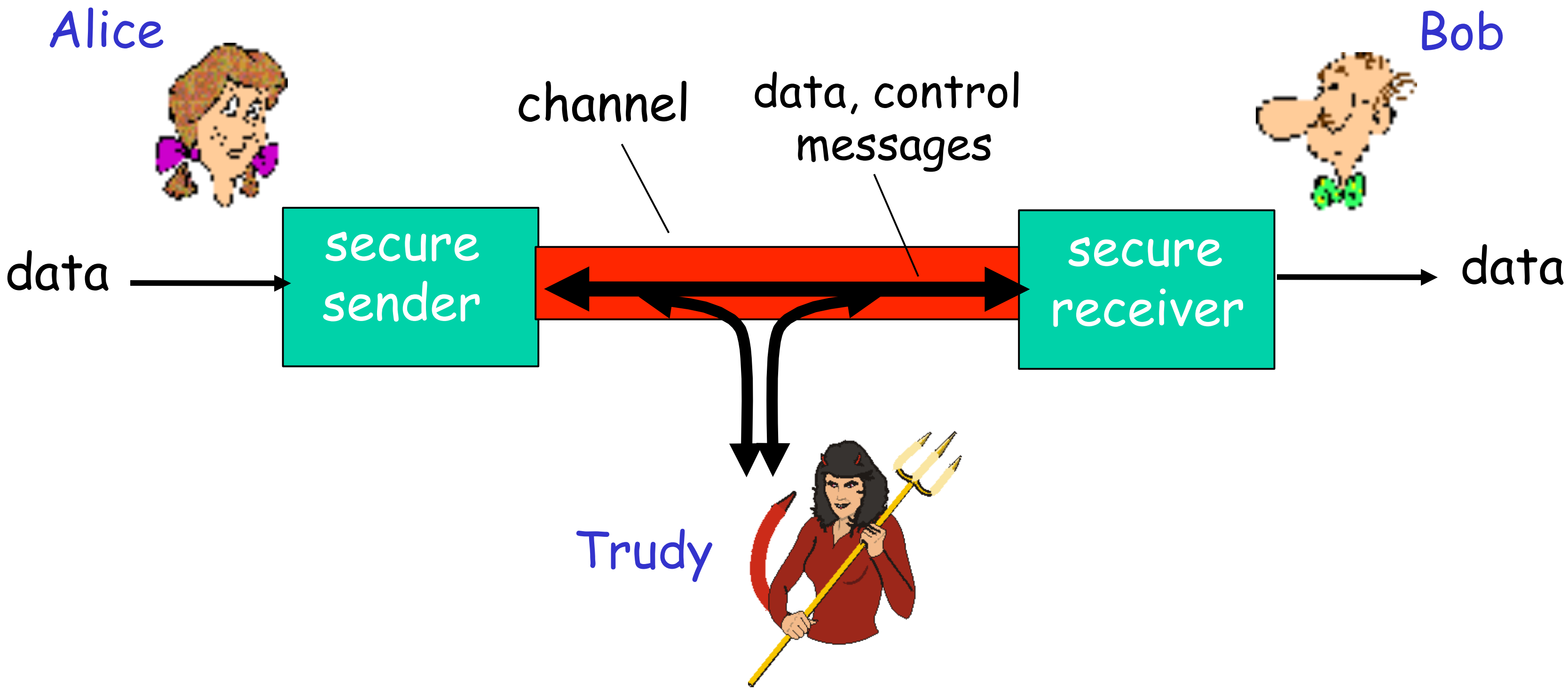
Authentication: sender, receiver want to confirm identity of each other

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 – a jealous spouse?) 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?

# Cryptography

- From the Greek words: ‘Cryptos’ (= secret) and ‘Grafien’ (= writing)
- From ancient times to around 30 years ago: essentially private communications for personal, political and military matters
- Today: study and application of techniques relying on the existence of hard problems
- A lot of historic uses of Cryptography…

# Cryptography in "ancient" times

- The bible codes
  - Atbash, Albam and Atbah
- Spartan Scytale (7th century BC)

- Caesar cipher
- Babington plot
- Enigma

- Some sources
  - The Code Book by Simon Singh
  - The codebreakers: the Story of Secret Writing by David Kahn
  - Google, Wikipedia, etc.

# Caesar cipher (a substitution cipher)

Caesar wants to encrypt the message:

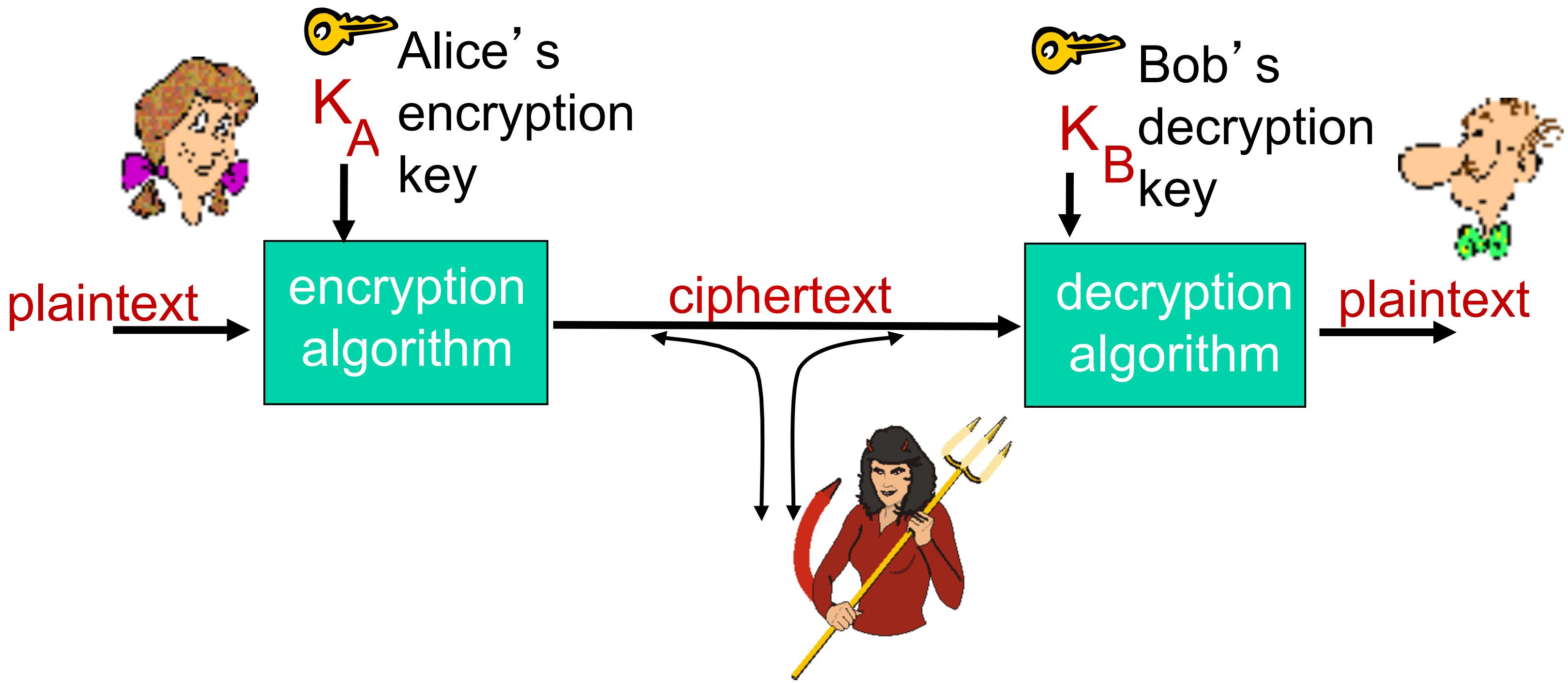
omnia gallia est divisa in partes tres

```
abcdefghijklmnopqrstuvwxyz

defghijklmnopqrstuvwxyzabc
```

rpqld jdoold hvw glylvd lq sduwhv wuhv

How to get the original message back?

# The language of cryptography




m plaintext message

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

$m = K_B(K_A(m))$

# Caesar cipher (a substitution cipher)

Caesar wants to encrypt the message:

plaintext omnia gallia est divisa in partes tres

```
abcdefghijklmnopqrstuvwxyz

defghijklmnopqrstuvwxyzabc
```

ciphertext rpqld jdoold hvw glylvd lq sduwhv wuhv

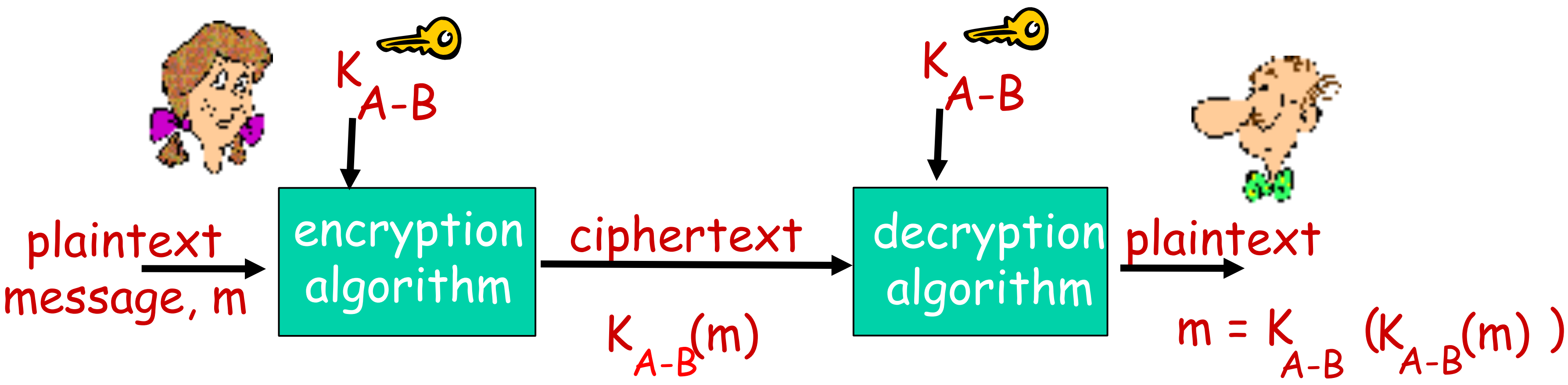
How to get the original message back?

Key: the shift of the alphabet (3 in the example)

# Outline

- Introduction
- Symmetric Key Cryptography
- Public Key Cryptography
- Authentication
- Integrity
- Security in Internet protocol stack

# Symmetric key cryptography




Symmetric key crypto: Bob and Alice share same (symmetric) key: $K_{A-B}$

- e.g., key is knowing alphabet shift in Caeser cipher
- Q: how do Bob and Alice agree on key value?

# How secure is Caesar cipher ?

Caesar wants to encrypt the message:

plaintext omnia gallia est divisa in partes tres

symmetric key

```
abcdefghijklmnopqrstuvwxyz
defghijklmnopqrstuvwxyzabc
```

ciphertext rpqld jdoold hvw glylvd lq sduwhv wuhv

There are only 25 possible keys! Given a ciphertext it is easy to compute the corresponding plaintext.

# Monoalphabetic cipher

- Substitute one letter for another
  - Similar to Caesar's, except no fixed pattern of substitution
  - The key is a one-to-one mapping between letters

```
plaintext:   abcdefghijklmnopqrstuvwxyz

ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

E.g.:

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

# How secure are monoalphabetic ciphers?

- Key is a mapping from the set of 26 letters to the set of 26 letters
- 26 factorial (26!) different pairings
  - 26! = 26 x 25 … x 2 x 1
  - = 403291461126605635584000000
- Use statistical analysis, e.g. 'e' and 't' account for 13% and 9% of letter occurrences respectively

# How secure are monoalphabetic ciphers?

- If Trudi knows that the words ‘alice’ and ‘bob’ are in the plaintext, then given the ciphertext she can determine the mapping of 7 letters
  - Less possibilities to be checked!
- Trudi can also notice that some certain letters appear often together (‘in’, ‘it’, ‘the’, ‘ing’, ...)
- What kind of information does Trudy have when breaking a cipher?

# Breaking Encryption

- *Cipher-text only attack*
  - Intruder analyses encrypted message
  - Statistical methods: e.g., knowing the frequency of letters or combinations in plaintext language
  - Brute-force attack: try every possible key (infeasible for long keys)
- *Known-plaintext attack*
  - Intruder knows some of the

    (plaintext, ciphertext) pairings
- *Chosen-plaintext attack*
  - Intruder can get ciphertext for some chosen plaintext
  - Monoalphabetic ciphers can be easily broken in this case
    - Simply ask to encrypt:
      "The quick brown fox jumps over the lazy dog"

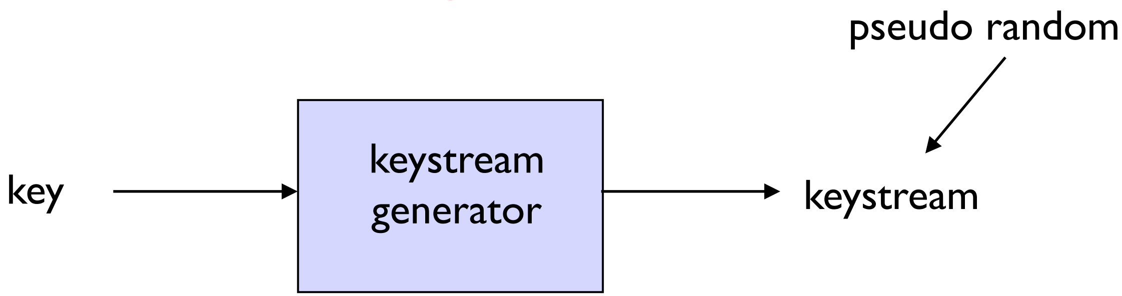
# Polyalphabetic encryption

- n monoalphabetic cyphers, $M_1, M_2, \ldots, M_n$
- Cycling pattern:
  - e.g., for 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 monoalphabetic pattern in cyclic pattern
  - 'dog': d from $M_1$, o from $M_3$, g from $M_4$
- Key: the n ciphers and the cyclic pattern

# Two types of symmetric ciphers

- Block ciphers
  - Break plaintext message in equal-size blocks
  - Encrypt each block as a unit
- Stream ciphers
  - encrypt one bit at time

# Stream Ciphers




- Combine each bit of keystream with bit of plaintext to get bit of ciphertext
- m(i) = i' th bit of message
- ks(i) = i' th bit of keystream
- c(i) = i' th bit of ciphertext
- $c(i) = ks(i) \oplus m(i)$ ($\oplus$ = exclusive or)
- $m(i) = ks(i) \oplus c(i)$

# RC4 Stream Cipher

- RC4 is a popular stream cipher
  - Extensively analyzed and considered good
  - Key can be from 1 to 256 bytes
  - Used in WEP for 802.11
  - Can be used in SSL

# Block ciphers

- Message to be encrypted is processed in blocks of k bits (e.g., 64-bit blocks).
- 1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext

Example with k=3:

| input | output |
|---|---|
| 000 | 110 |
| 001 | 111 |
| 010 | 101 |
| 011 | 100 |

| input | output |
|---|---|
| 100 | 011 |
| 101 | 010 |
| 110 | 000 |
| 111 | 001 |

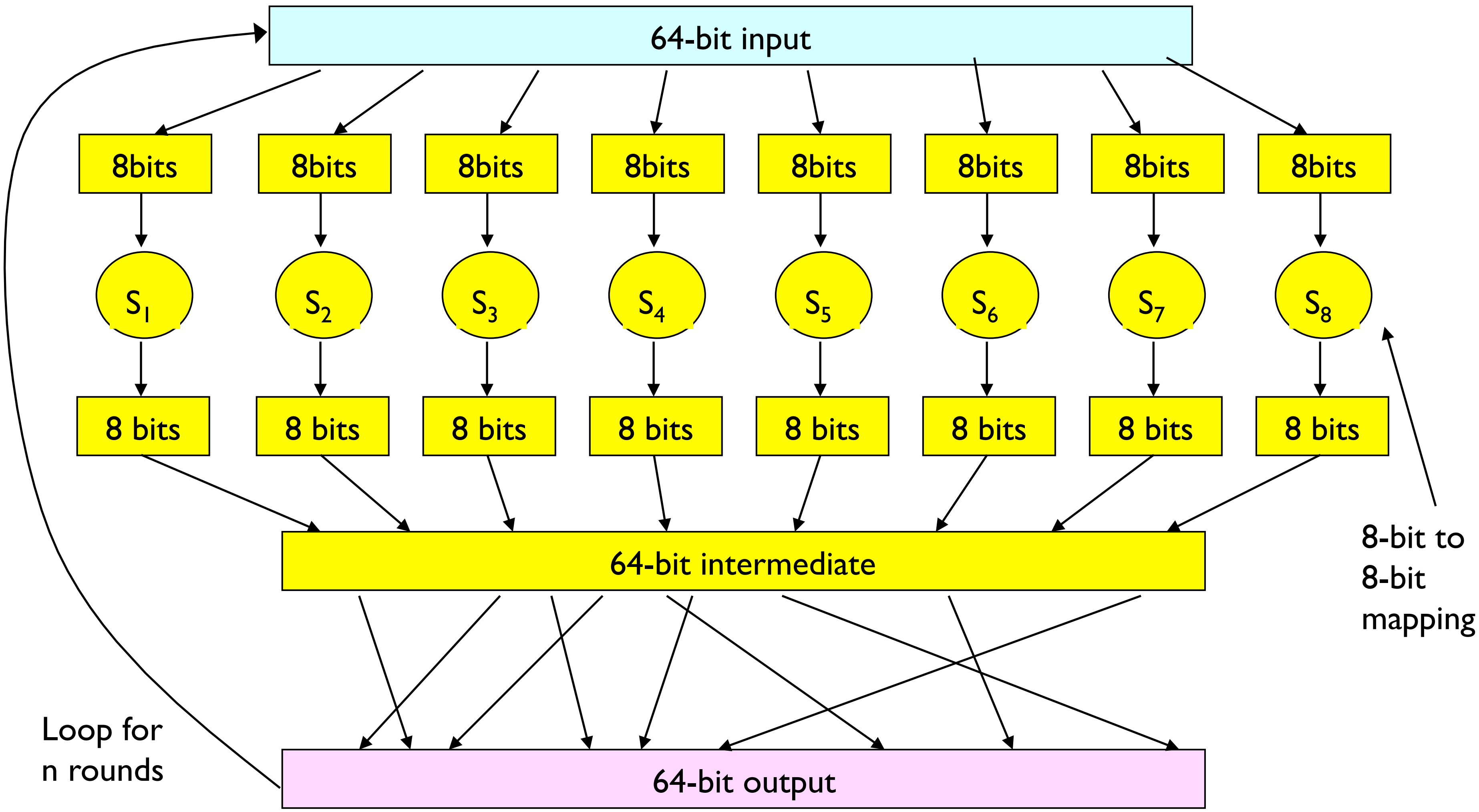
What is the ciphertext for 010110001111 ?

# Block ciphers

- How many possible mappings are there for k=3?
  - How many 3-bit inputs?
  - How many permutations of the 3-bit inputs?
  - Answer: 8! = 40,320;  not very many!
- In general, $2^k!$ mappings;   huge for k=64
- Problem:
  - Table approach requires table with $2^{64}$ entries, each entry with 64 bits
- Table too big: instead use function that simulates a randomly permuted table

# Prototype function

*From Kaufman et al*



# Why rounds in prototpe?

- If only a single round, then one bit of input affects at most 8 bits of output.
- In 2nd round, the 8 affected bits get scattered and inputted into multiple substitution boxes.
- How many rounds?
  - How many times do you need to shuffle cards
  - Becomes less efficient as n increases

# Encrypting a large message

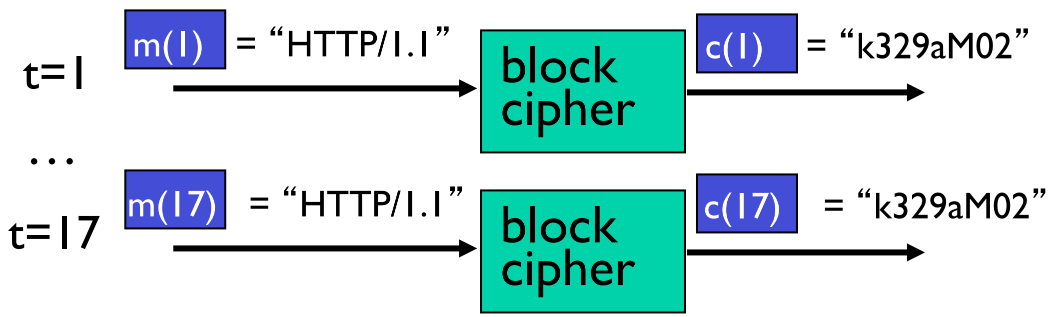
- Why not just break message in 64-bit blocks, encrypt each block separately?
  - If same block of plaintext appears twice, will give same cyphertext.
- How about:
  - Generate random 64-bit number r(i) for each plaintext block m(i)
  - Calculate $c(i) = K_S( m(i) \oplus r(i) )$
  - Transmit c(i), r(i), i=1,2,…
  - At receiver: $m(i) = K_S(c(i)) \oplus r(i)$
  - Problem: inefficient, need to send c(i) and r(i)

# Cipher Block Chaining (CBC)

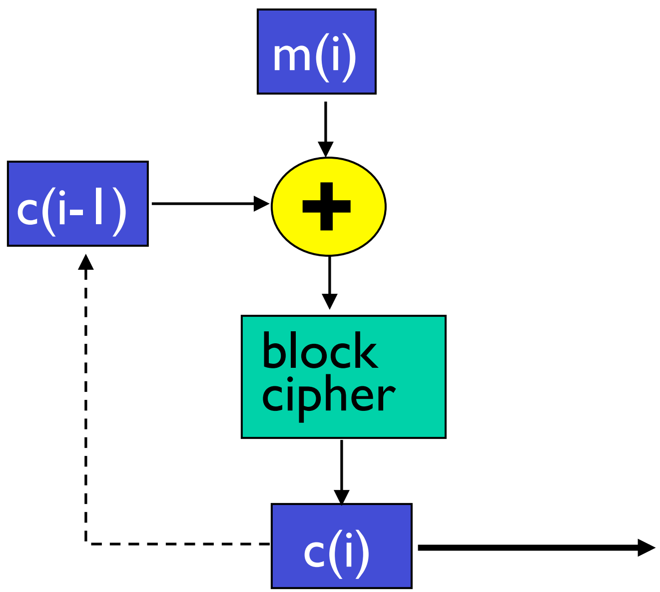
- CBC generates its own random numbers
  - Have encryption of current block depend on result of previous block
  - $c(i) = K_S(\ m(i) \oplus c(i\text{-}1)\ )$
  - $m(i) = K_S(\ c(i)) \oplus c(i\text{-}1)$
- How do we encrypt first block?
  - Initialization vector (IV): random block = c(0)
  - IV does not have to be secret
- Change IV for each message (or session)
  - Guarantees that even if the same message is sent repeatedly, the ciphertext will be completely different each time

# Cipher Block Chaining

- cipher block: if input block repeated, will produce same cipher text:




- *cipher block chaining:* XOR ith input block, m(i), with previous block of cipher text, c(i-1)
  - c(0) transmitted to receiver in clear
  - what happens in "HTTP/1.1" scenario from above?



# Symmetric key in the real world: DES

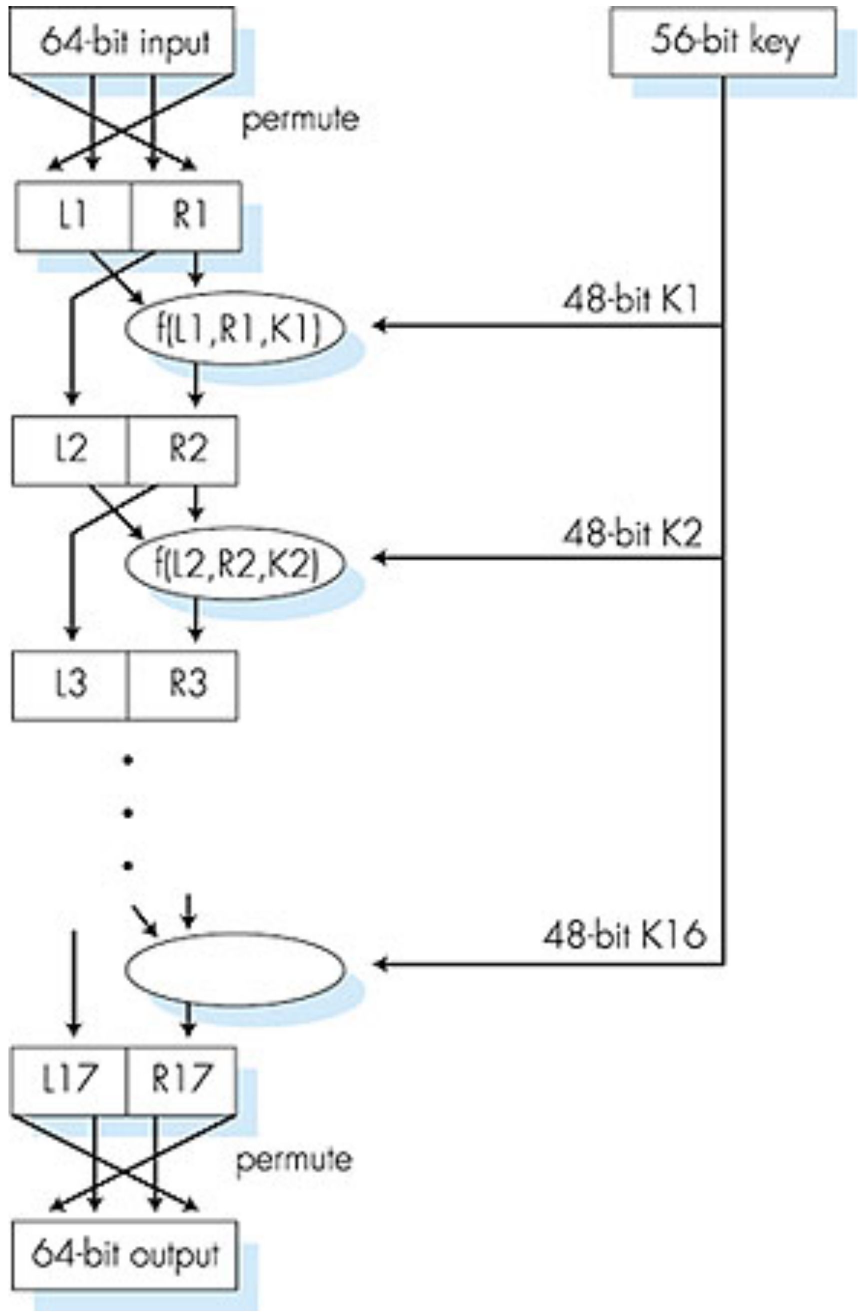
## DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key
  - $2^{56} = 72057594037927936$
- 64-bit plaintext input
- How secure is DES?
  - no known good analytic attack
  - DES Challenge III (1999): 56-bit-key-encrypted phrase decrypted (brute force) in 22h 15m
  - 1 supercomputer ‘Deep Crack’ and 100,000 distributed PCs on the internet testing 245 billion keys per second!
- Making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys (actually encrypt, decrypt, encrypt) - using cipher-block chaining

# Symmetric key crypto: DES

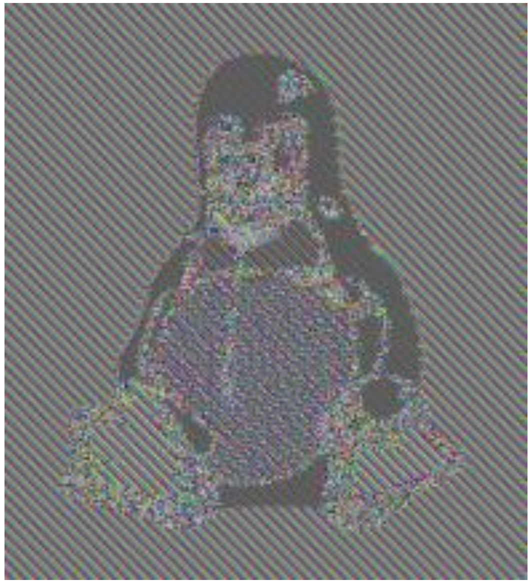
DES operation

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

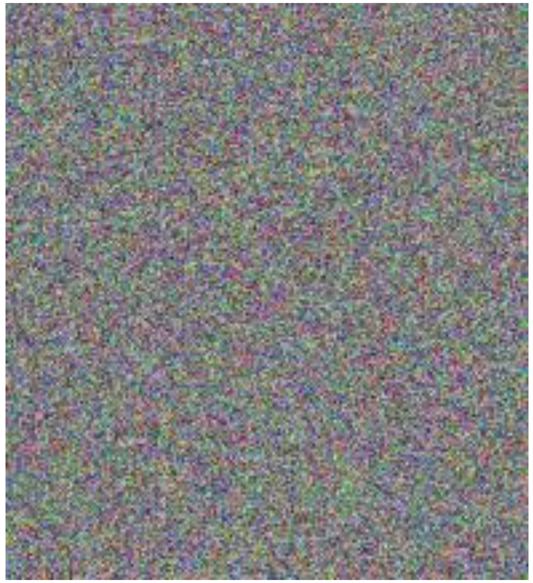


# Symmetric key crypto: DES

Original



Without cipher-block chaining



With cipher-block chaining

# AES: Advanced Encryption Standard

- New (Nov. 2001) symmetric-key NIST standard, replacing DES
- Processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- $2^{256}$ = 115,792,089,237,316,195,423,570,985,008,687,907,853,269, 984,665,640,564,039,457,584,007,913,129,639,936 (that's 78 digits)
- Brute force decryption (try each key) taking 1sec on DES, takes 149 trillion years for AES

# So AES is unbreakable then?

- Not at all!
- The key could be found on the first guess (a probability of $1/2^{256}$)!
- The trick is to have a key space so large that it is not worth anyone trying a brute-force attack

# Outline

- Introduction
- Symmetric Key Cryptography
- Public Key Cryptography
- Authentication
- Integrity
- Security in Internet protocol stack

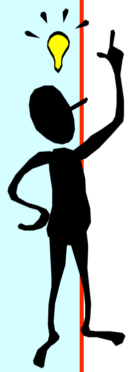
# Public Key Cryptography

*symmetric key crypto*

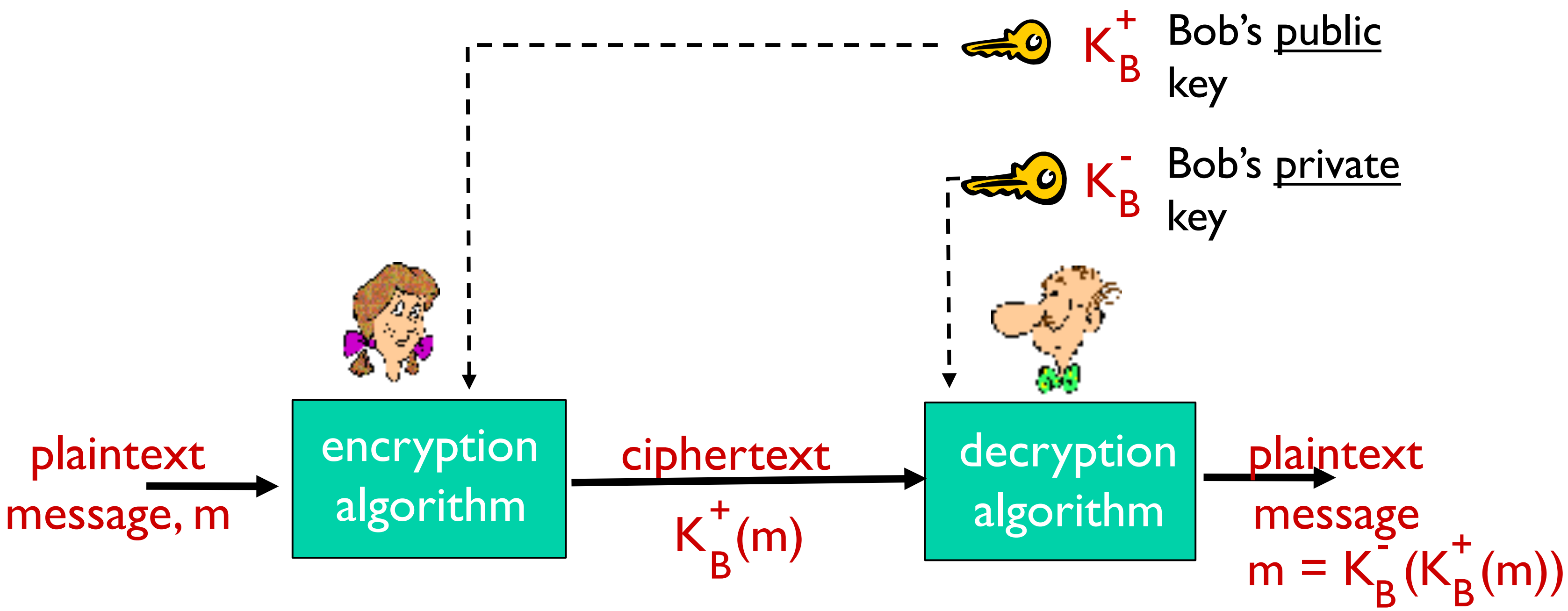
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

*public key cryptography*

- 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



# Requirements for public key encryption algorithms

- Need $K_B^+(\ )$ and $K_B^-(\ )$ such that

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

- It is computationally easy to
  - Generate a pair of keys
  - Encrypt and decrypt messages using these keys
- It is computationally infeasible
  - Determine the private key from the public key
  - Recover the plaintext from the public key and the ciphertext

# 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: a=14, n=10, d=2:
  $(a \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$
  $a^d \bmod 10 = 14^2 \bmod 10 = 196 \bmod 10 = 6$

# RSA: getting ready

- A message is a bit pattern.
- A 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 cyphertext).

# RSA: Choosing keys

RSA: Rivest, Shamir, Adleman algorithm

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 bit pattern, $m$, compute

   $c = m^e \bmod n$ (i.e., remainder when $m^e$ is divided by $n$)

2. To decrypt received bit pattern, $c$, compute

   $m = c^d \bmod n$ (i.e., remainder when $c^d$ is divided by $n$)

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 | 248832 | 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 $c = m^e \bmod n$
- Result from number theory: 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,
  $c^d \bmod n = (m^e \bmod n)^d \bmod n$
  $= m^{ed} \bmod n$
  $= m^{(ed \bmod z)} \bmod n$ (by the result above)
  $= m^1 \bmod n$ (since ed is divisible by (p-1)(q-1) with remainder 1)
  $= m$

# RSA: another important property

The following property will be *very* useful later:

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

*Result is the same!*

Why is it true for RSA?

# 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.

# Generating RSA keys

- Have to find big primes p and q
- Approach: make good guess then apply testing rules

# Session keys

- Exponentiation is computationally intensive
- DES is at least 100 times faster than RSA
- Combination of public and symmetric key cryptography using 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

# Outline

- Introduction
- Symmetric Key Cryptography
- Public Key Cryptography
- Authentication
- Integrity
- Security in Internet protocol stack

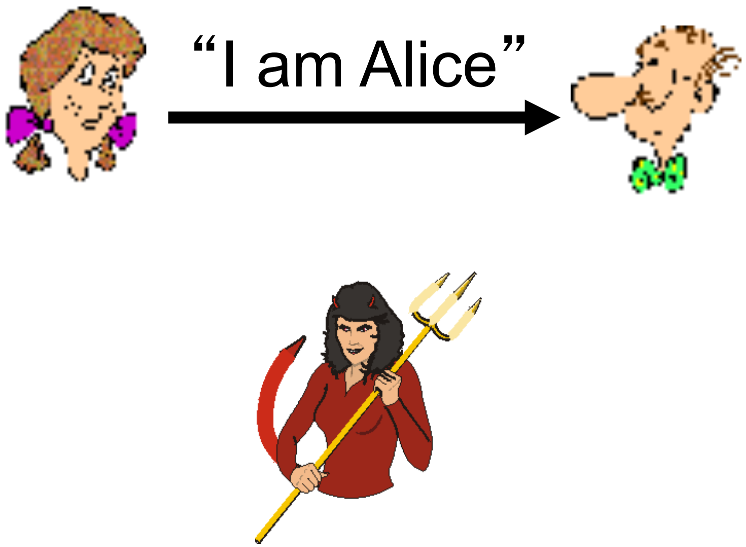
# What is authentication?

- Process of proving one's identity to someone else
- As humans, we authenticate each other using personal traits, e.g. faces, voices
- For electronic systems, use *authentication protocols*
  - Typically run *before* some other protocol

# Authentication

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

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

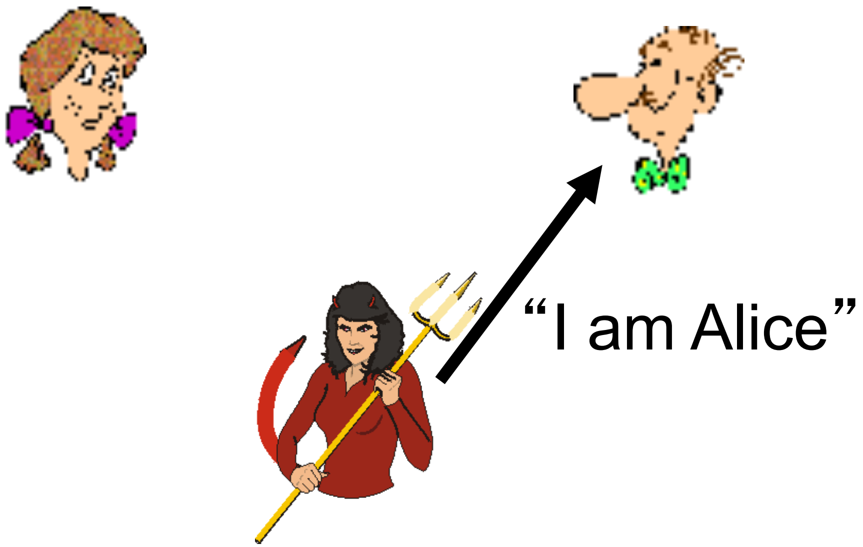



Failure scenario??

# 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

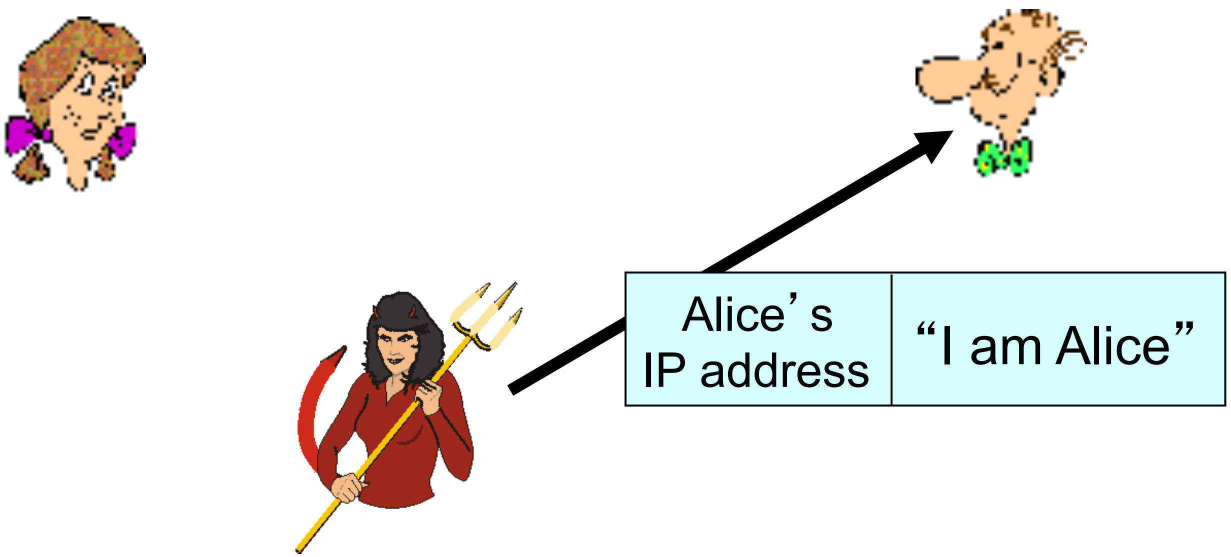
# Authentication: another try

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


Failure scenario??

# Authentication: another try

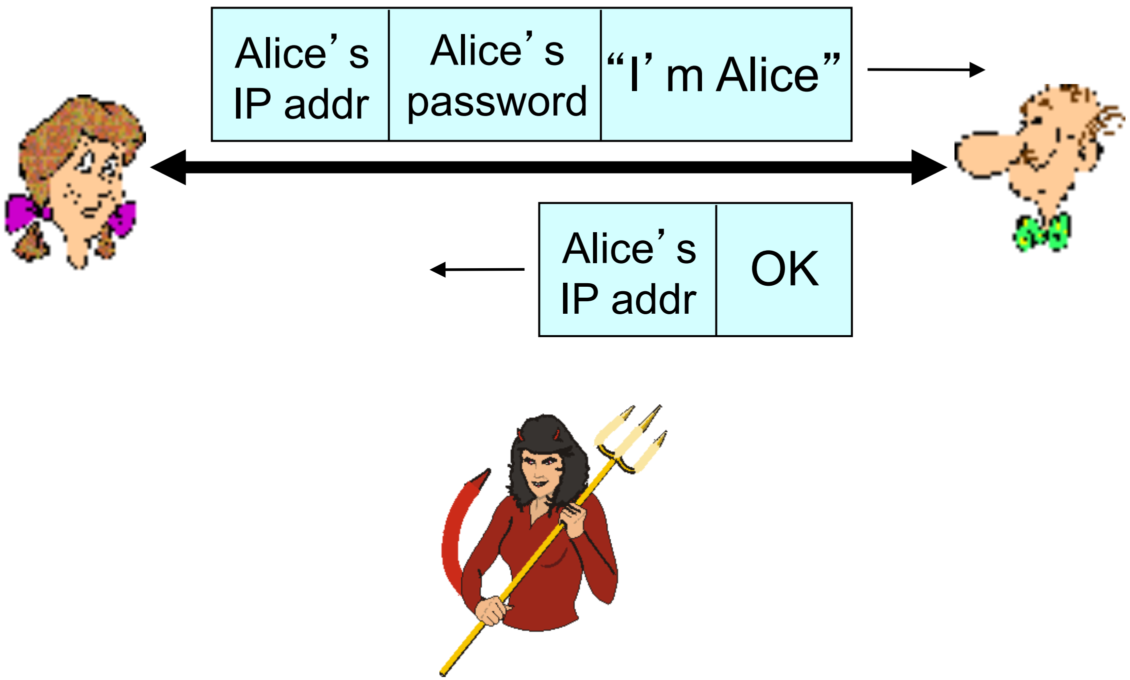
*Protocol ap2.0:* Alice says "I am Alice" in an IP packet containing her source IP address




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

# Authentication: another try

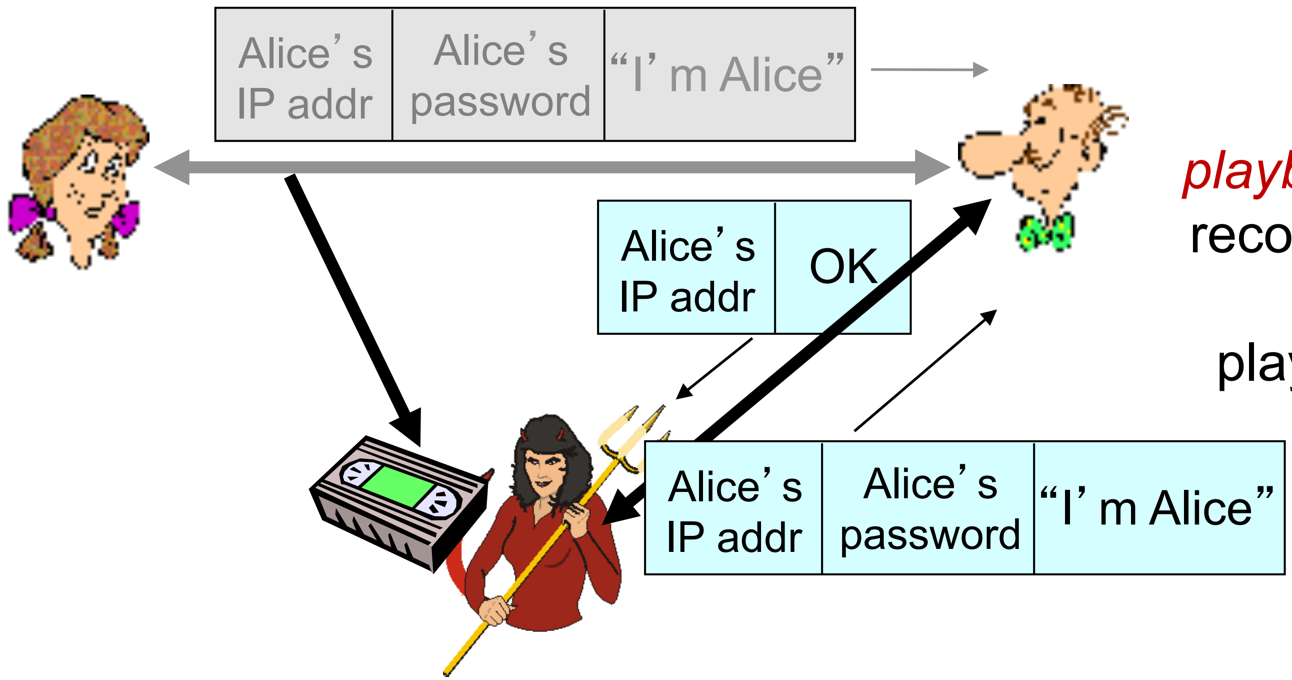
*Protocol ap3.0:* Alice says "I am Alice" and sends her secret password to "prove" it.




Failure scenario??

# Authentication: another try

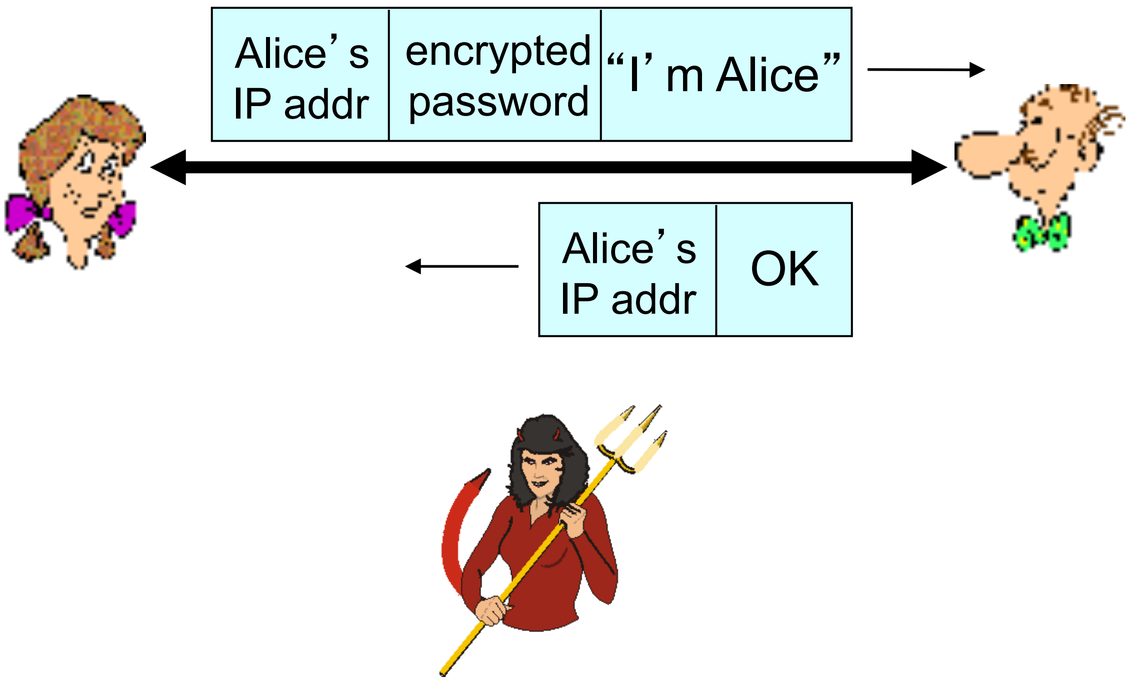
*Protocol ap3.0:* Alice says "I am Alice" and sends her secret password to "prove" it.




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

# Authentication: yet another try

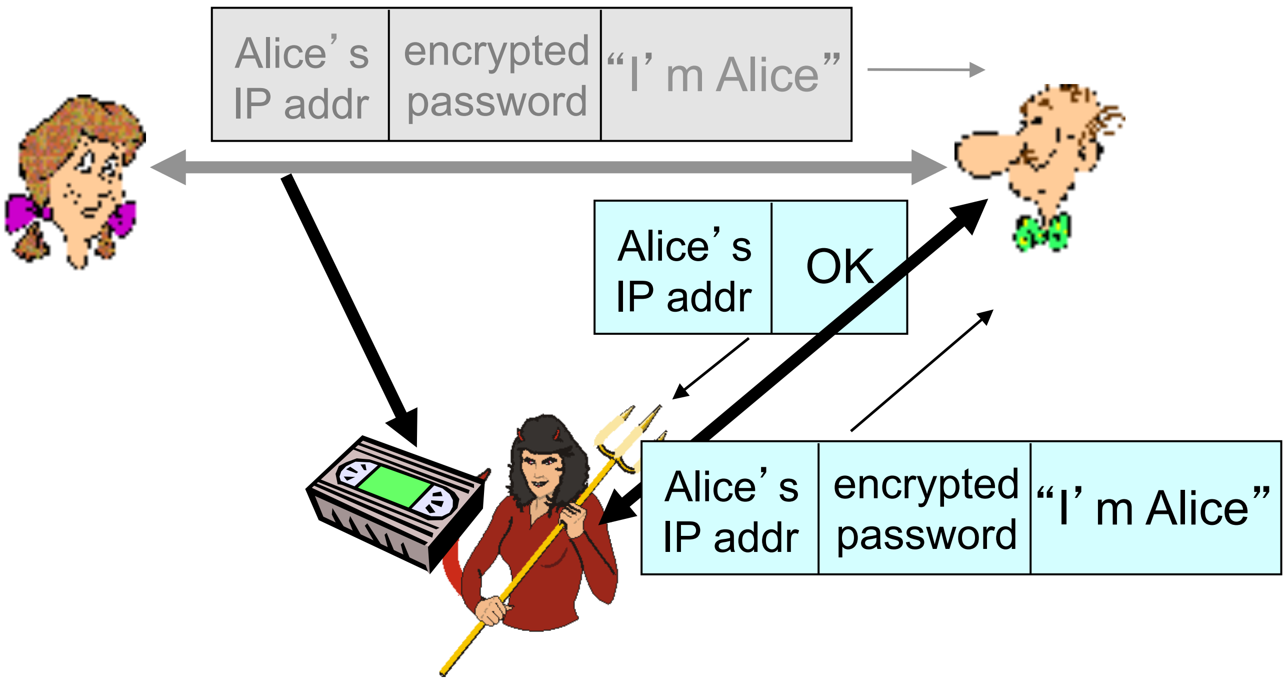
*Protocol ap3.1:* Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.




Failure scenario??

# Authentication: yet another try

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



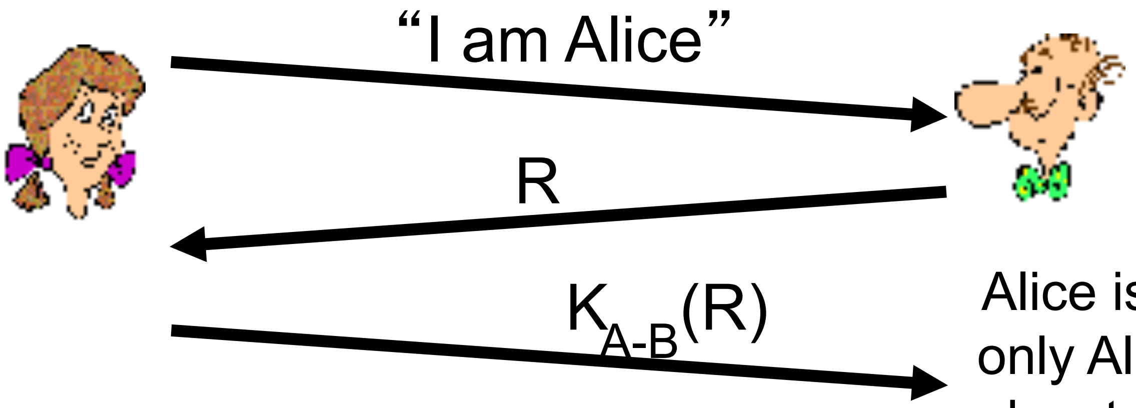

record and playback *still* works!

# 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

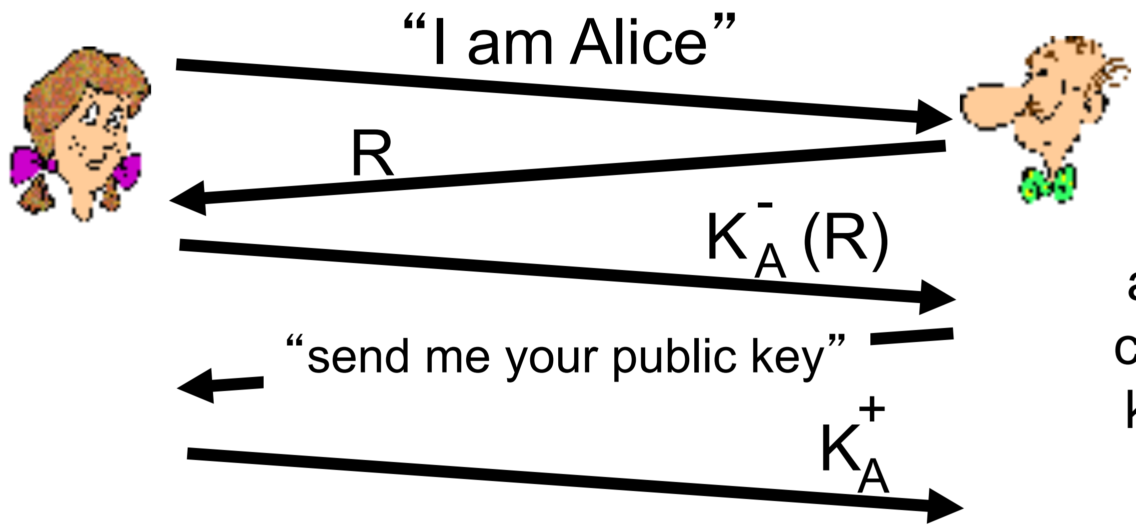
- ap4.0 requires shared symmetric key
- Can we authenticate using public key techniques?
- Recall the following property:

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

*Result is the same!*

# Authentication: ap5.0

*ap5.0:* use nonce, public key cryptography




Bob computes

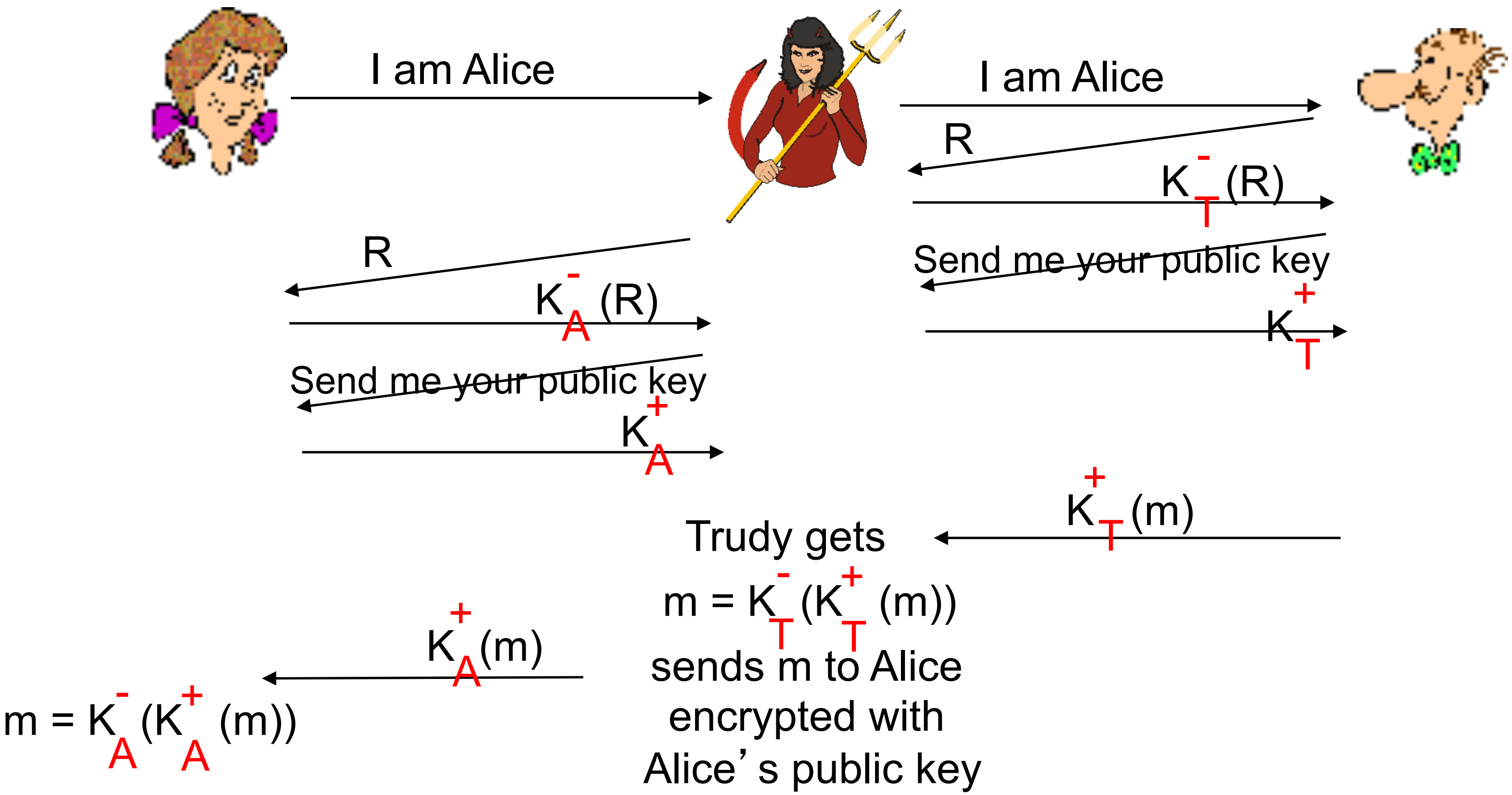
$$K_A^+(K_A^-(R)) = R$$

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

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

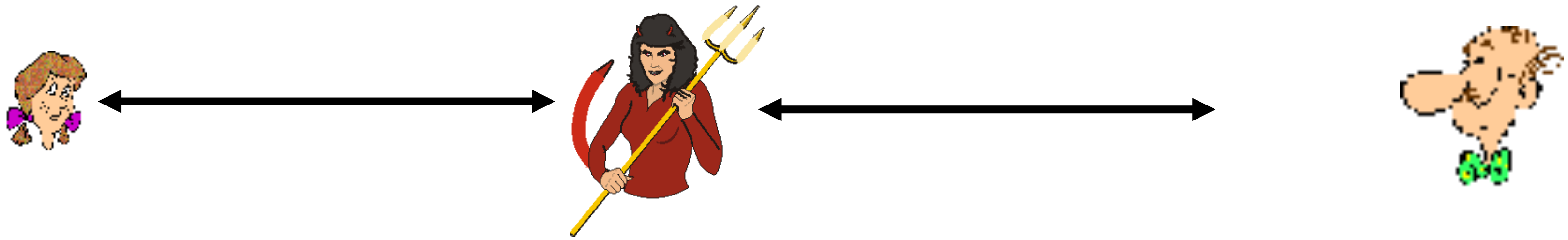
# ap5.0: security hole

*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!

# Outline

- Introduction
- Symmetric Key Cryptography
- Public Key Cryptography
- Authentication
- Integrity
  - Digital Signatures
  - Public Key Infrastructure
  - Hash Functions
- Security in Internet protocol stack

# What is message integrity?

- Allows communicating parties to verify that received messages are authentic.
  - Content of message has not been altered
  - Source of message is who/what you think it is
  - Message has not been replayed
  - Sequence of messages is maintained
- Example: proving that an email came from a specific person

# 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

# Public key encryption property

❖ Recall the following property:

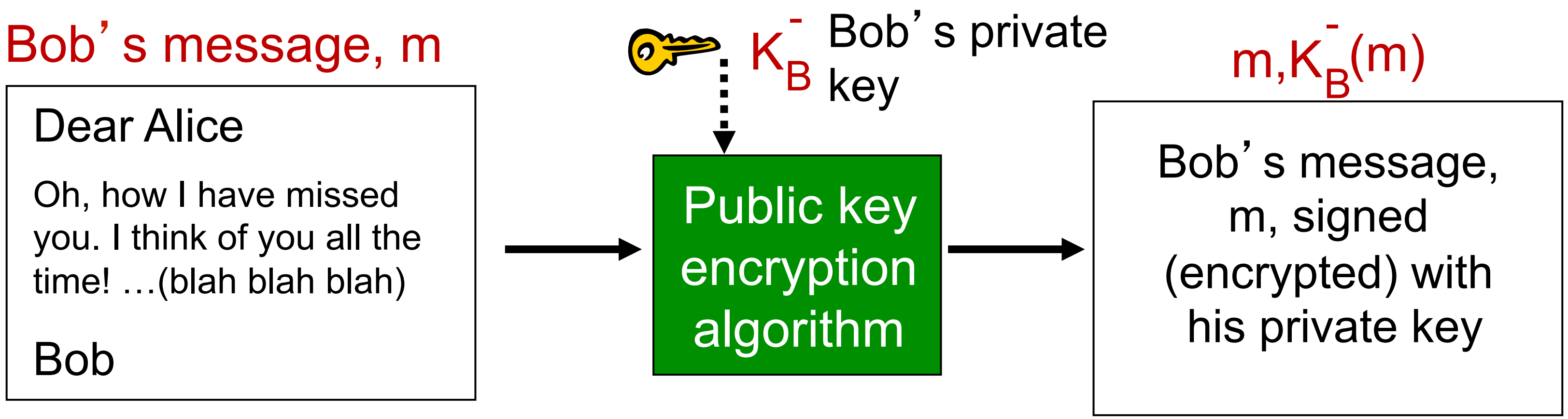
$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

*Result is the same!*

# 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.

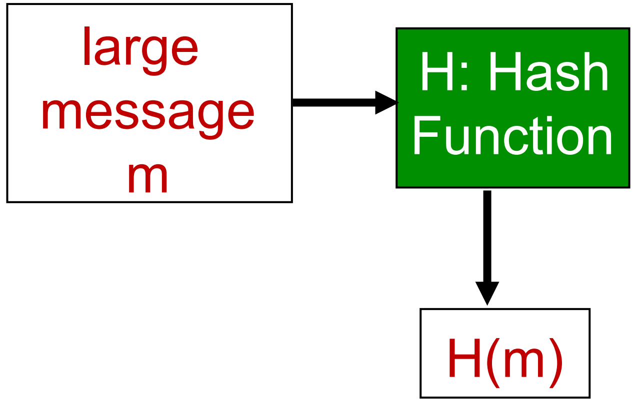
Alice thus verifies that:

- 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)

Sign only small 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

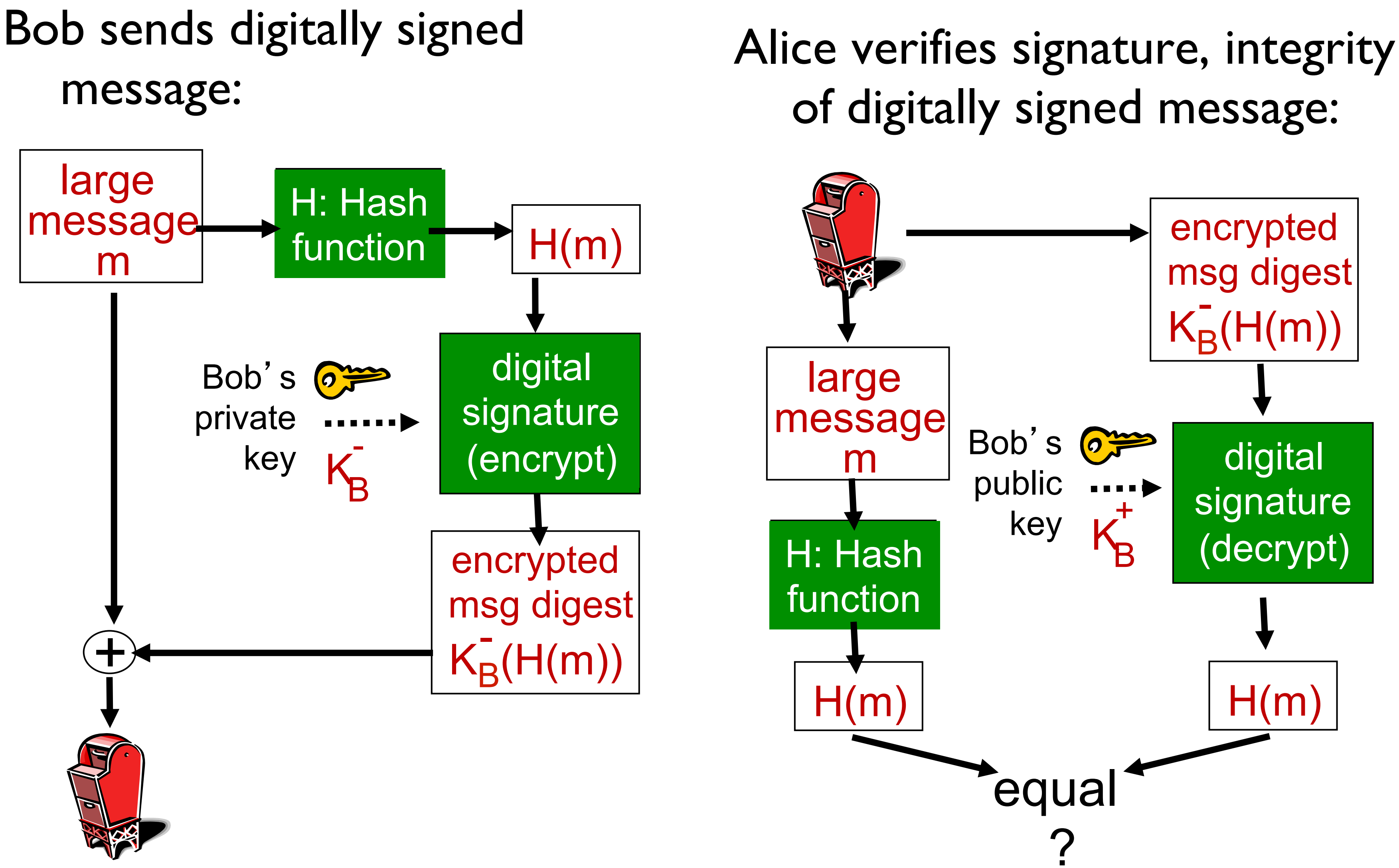
But given message with given hash value, it is easy to find another message with same hash value:

| 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!

# Digital signature = signed message digest

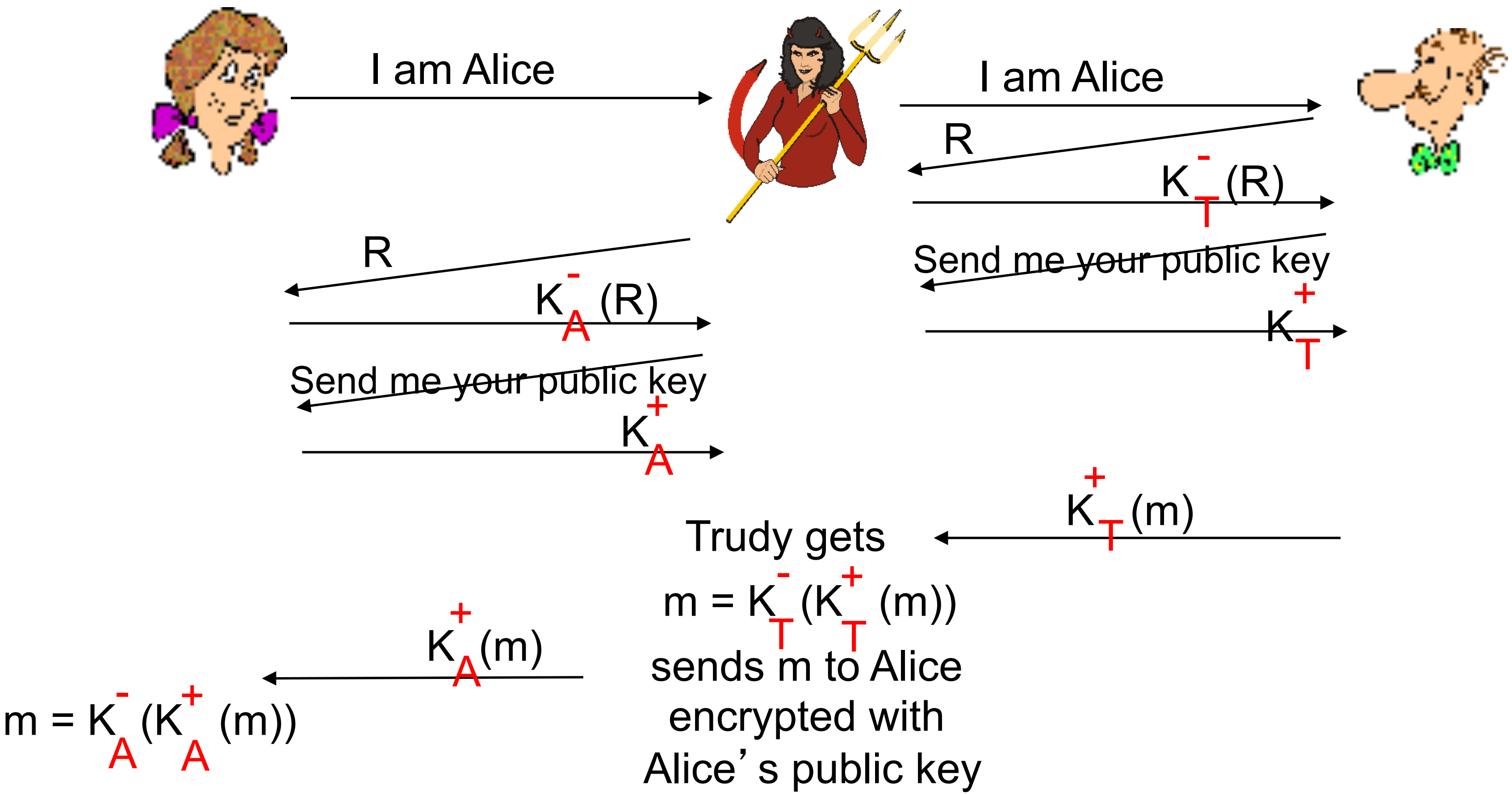


# Hash function algorithms

- 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)

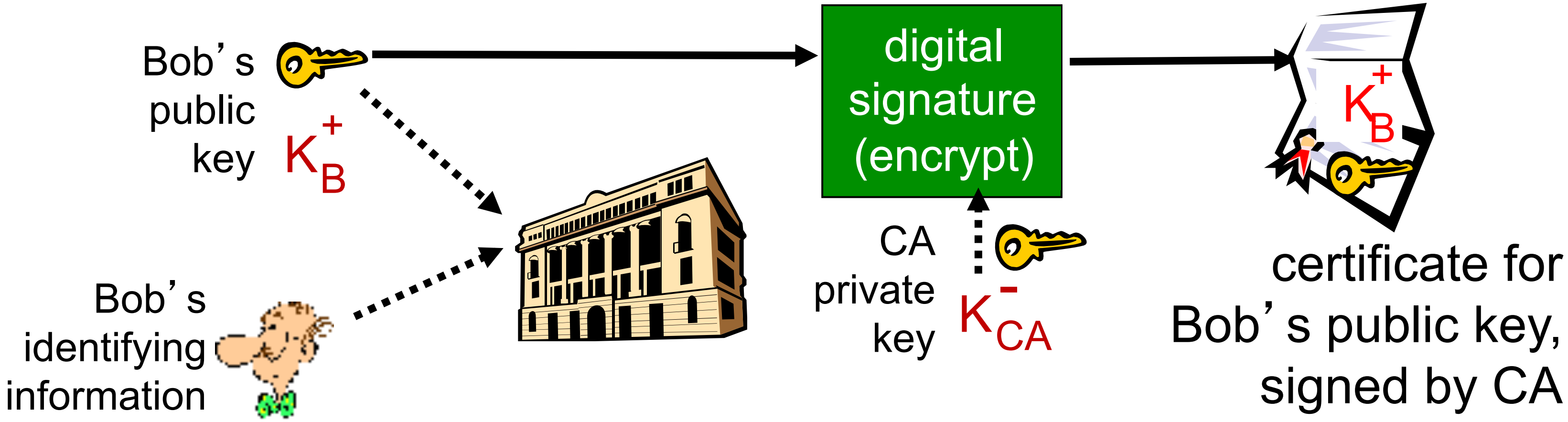


# Public-key certification

- 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

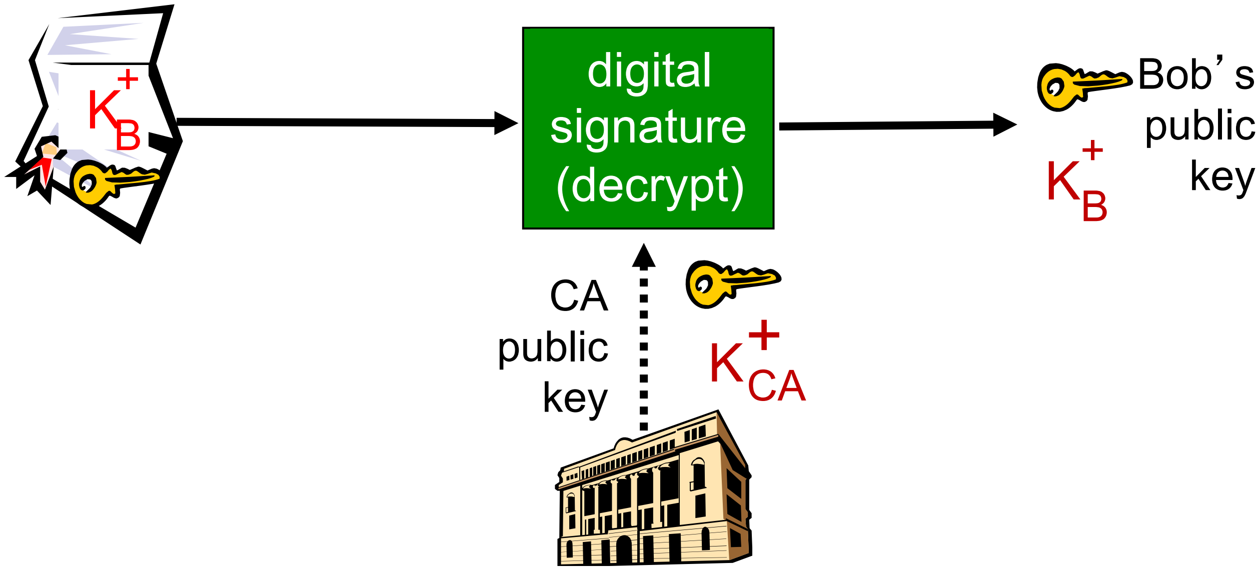
# Certification authorities

- *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"



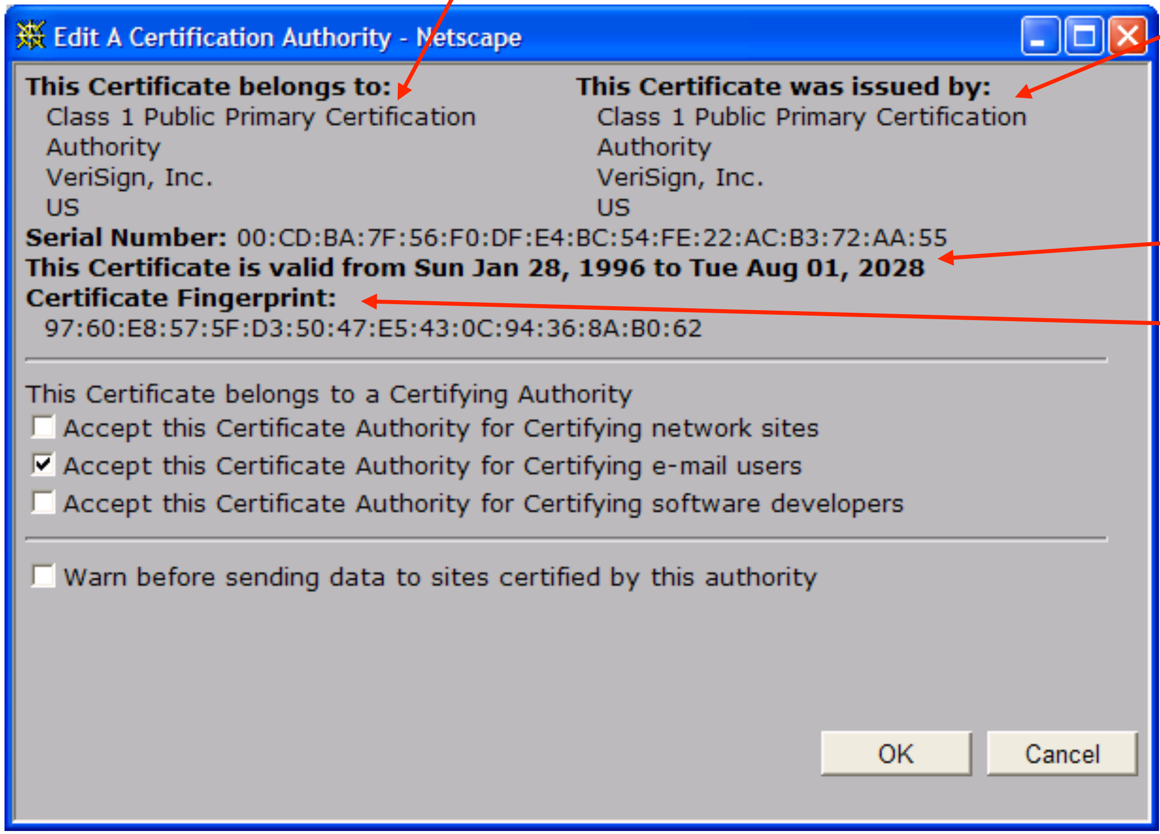
# Certification authorities

- 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



# A certificate contains:

- Serial number (unique to issuer)
- info about certificate owner, including algorithm and key value itself (not shown)



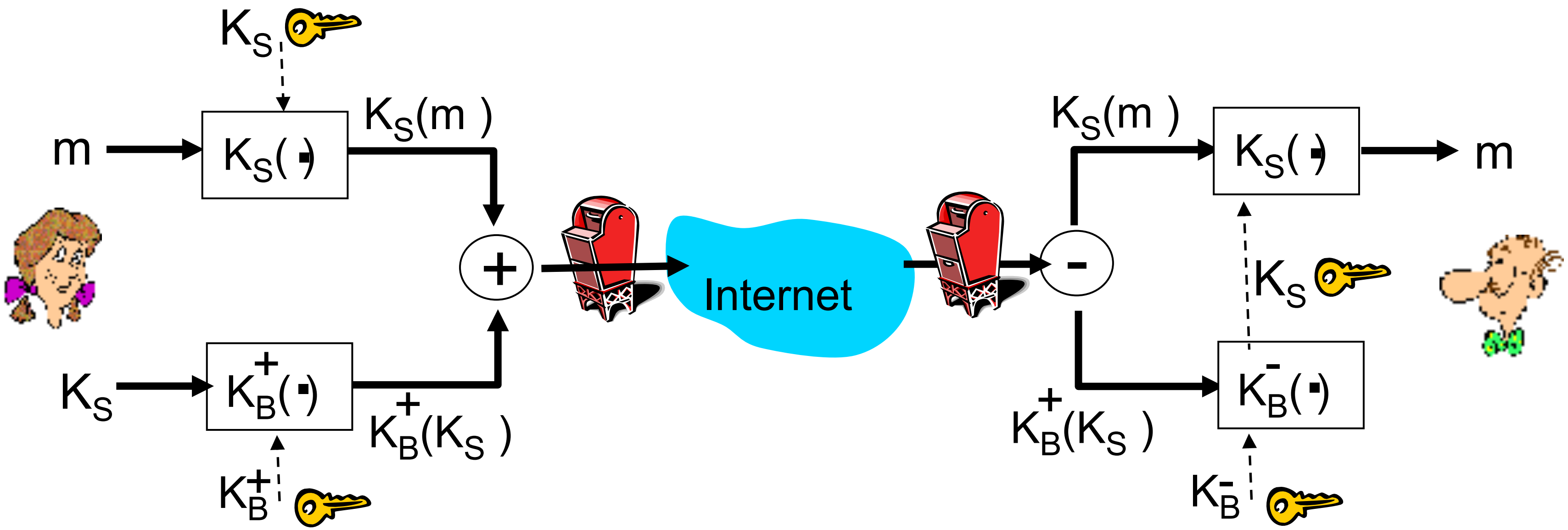

- info about certificate issuer
- valid dates
- digital signature by issuer

# Outline

- Introduction
- Symmetric Key Cryptography
- Public Key Cryptography
- Authentication
- Integrity
- Security in Internet protocol stack
  - secure e-mail
  - secure sockets
  - wireless security: 802.11 WEP

# Secure e-mail

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

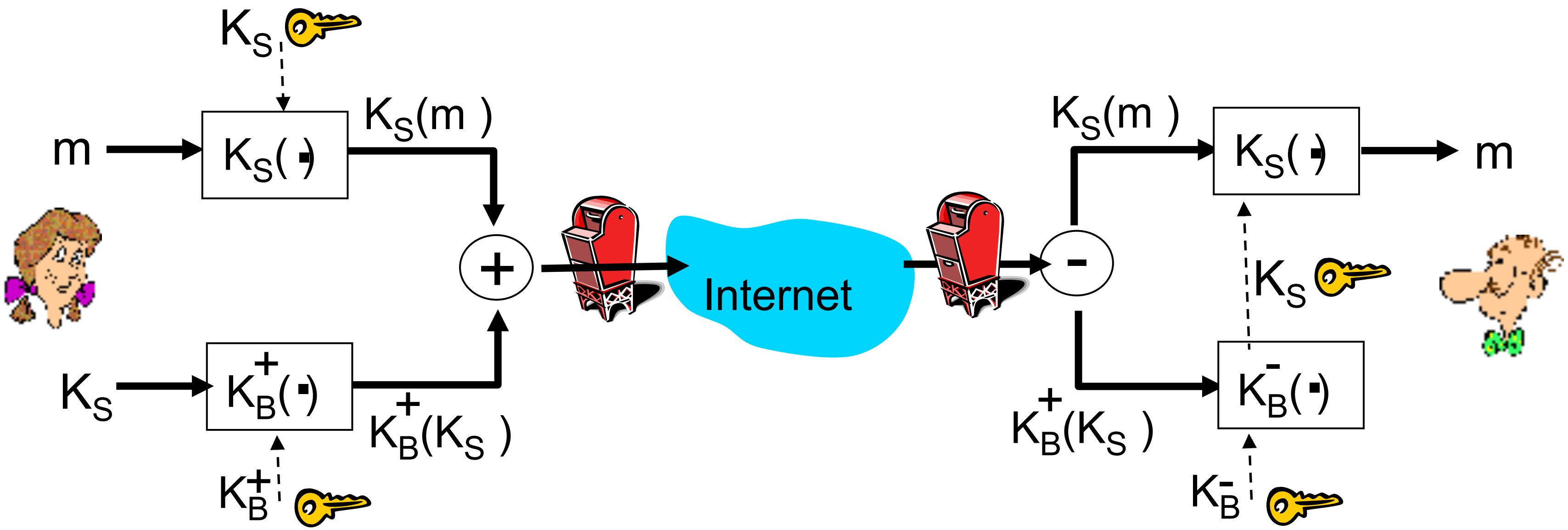



*Alice:*

- 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.

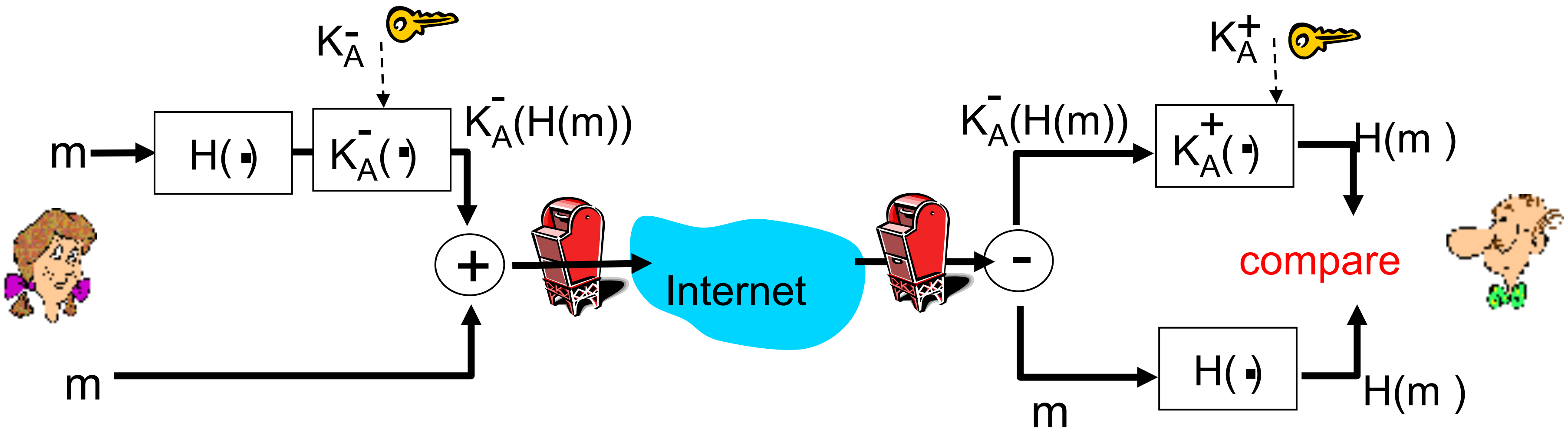



*Bob:*

- 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)

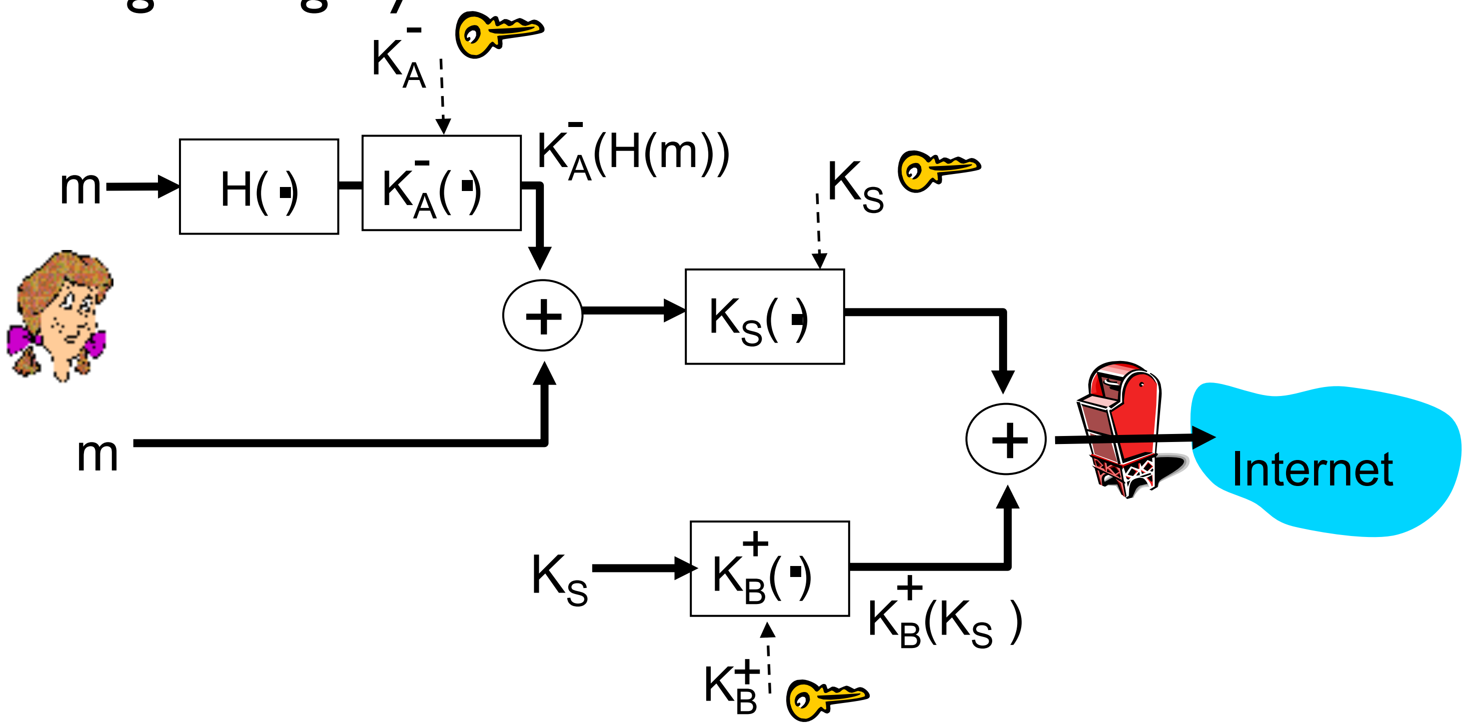
- Alice wants to provide sender authentication message integrity




- 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

# Pretty good privacy (PGP)

- used for signing, encrypting and decrypting e-mails
- de-facto standard
- Design (in essence) the same as on previous slide.
  - Uses symmetric key cryptography, public key cryptography, hash function, and digital signature as described.
- Provides secrecy, sender authentication, integrity.
- Inventor, Phil Zimmerman, was target of 3-year U.S. federal investigation (crypto programs considered munitions under U.S. law)

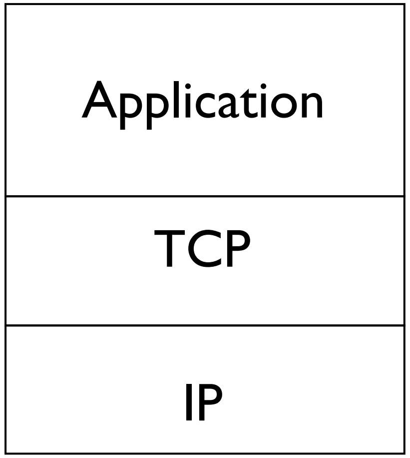
*A PGP signed message:*

```
---BEGIN PGP SIGNED MESSAGE---
Hash: SHA1

Bob:My husband is out of town
   tonight.Passionately yours,
   Alice

---BEGIN PGP SIGNATURE---
Version: PGP 5.0
Charset: noconv
yhHJRHhGJGhgg/
   12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G
   6m5Gw2
---END PGP SIGNATURE---
```

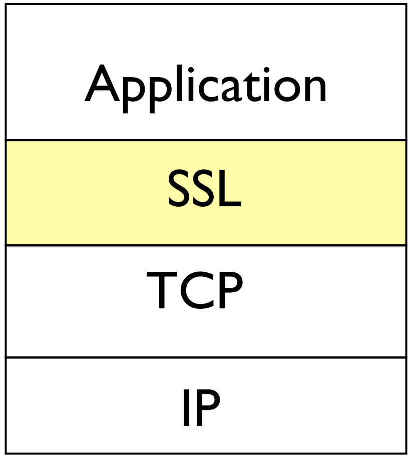
# SSL: Secure Sockets Layer

- Widely deployed security protocol
  - Supported by almost all browsers and web servers
  - https
  - Crucial for E-commerce applications
- Originally designed by Netscape in 1993
- Provides
  - Confidentiality
  - Integrity
  - Authentication
- Original goals:
  - Had Web e-commerce transactions in mind
  - 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




Normal Application




Application with SSL

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

# SSL (continued)

- Security services:
  - server authentication
  - data encryption
  - client authentication (optional)

- Server authentication:
  - SSL-enabled browser includes public keys for trusted CAs.
  - Browser requests server certificate, issued by trusted CA.
  - Browser uses CA's public key to extract server's public key from certificate.

- Check your browser's security menu to see its trusted CAs

# SSL (continued)

Encrypted SSL session:

- Browser generates *symmetric session key*, encrypts it with server's public key, sends encrypted key to server.
- Using private key, server decrypts session key.
- Browser, server know session key
  - All data sent into TCP socket (by client or server) encrypted with session key.

- SSL: basis of IETF Transport Layer Security (TLS).
- SSL can be used for non-Web applications, e.g., IMAP.
- Client authentication can be done with client certificates.

# IEEE 802.11 (Wi-Fi) security

- *War-driving:* drive around San Francisco Bay area, see what 802.11 networks available
  - More than 9000 accessible from public roadways
  - 85% use no encryption/authentication
  - packet-sniffing and various attacks easy!
- *Wired Equivalent Privacy (WEP):* authentication as in protocol *ap4.0*
  - host requests authentication from access point
  - access point sends 128 bit nonce
  - host encrypts nonce using shared symmetric key
  - access point decrypts nonce, authenticates host

# IEEE 802.11 (Wi-Fi) security

- *Wired Equivalent Privacy (WEP)*: data encryption
  - Stream cipher (RC4) used: *message XOR key*

| L | R | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

E.g.:

| | |
|---|---|
| Plaintext | 1100 |
| XOR | |
| Key | 0101 |
| = | |
| Ciphertext | 1001 |

# IEEE 802.11 (Wi-Fi) security

- Easily cracked if the same key is used every time
- Example:
  - Messages $a$ and $b$ encrypted with key $k$
  - $E_k(a) = a \text{ XOR } k$ and $E_k(b) = b \text{ XOR } k$
- However, XOR is commutative
  - $(a \text{ XOR } b) \text{ XOR } c = a \text{ XOR } (b \text{ XOR } c)$
- And for any $a$, the inverse w.r.t XOR is $a$
  - $a \text{ XOR } a = 000\ldots$ and $j \text{ XOR } 000\ldots = j$
- Intercept $E_k(a)$ and $E_k(b)$ , *then*

$$
\begin{aligned}
& E_k(a) \text{ XOR } E_k(b) & \\
=\ & (a \text{ XOR } k) \text{ XOR } (b \text{ XOR } k) & \text{(definition of } E_k\text{)} \\
=\ & a \text{ XOR } b \text{ XOR } (k \text{ XOR } k) & \text{(commutative law)} \\
=\ & a \text{ XOR } b & \text{(self-inverse law)}
\end{aligned}
$$

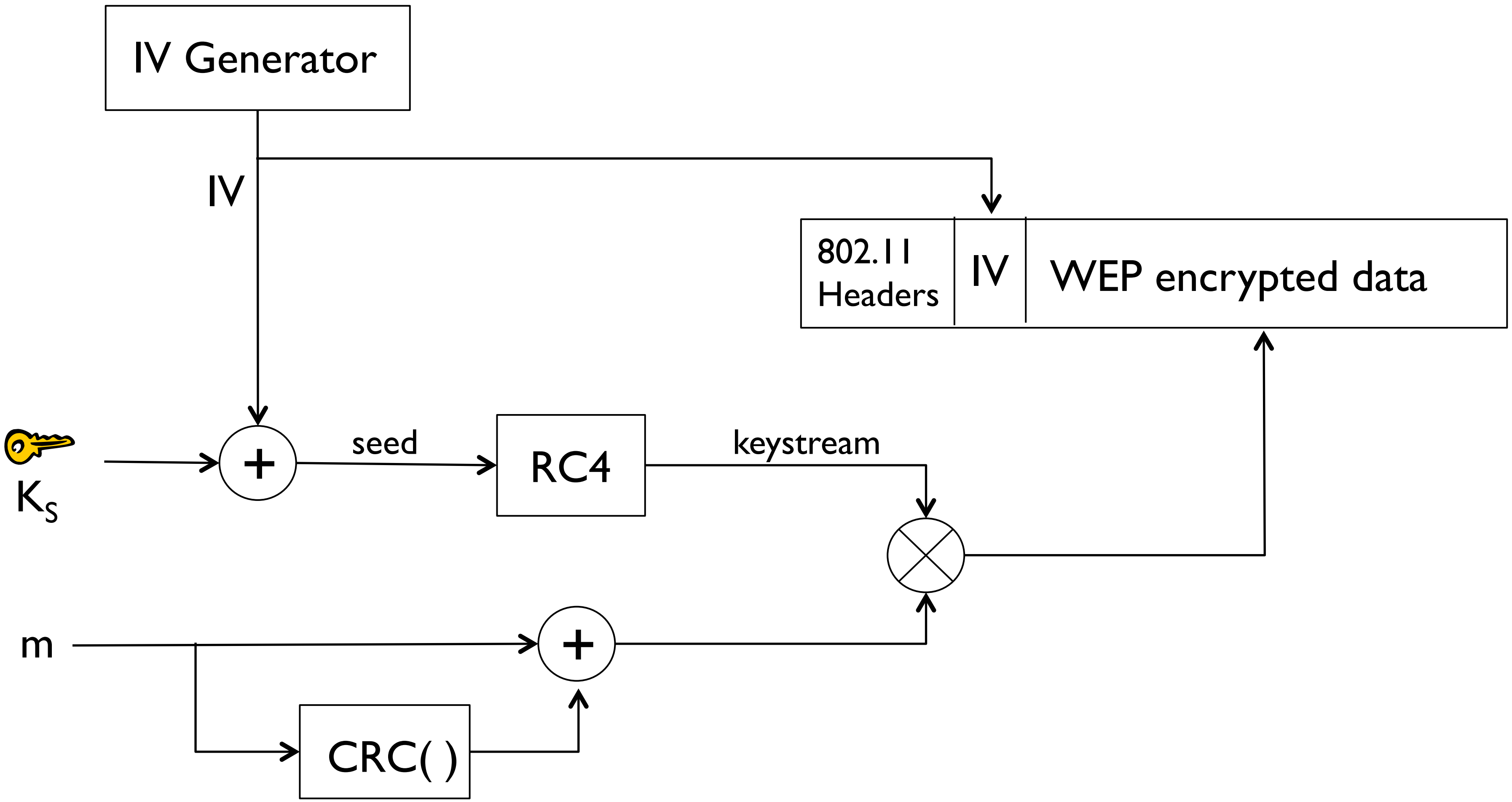
# IEEE 802.11 (Wi-Fi) security

- *Wired Equivalent Privacy (WEP):* data encryption
  - Host/AP share 40 bit symmetric key (semi-permanent)
  - Host appends 24-bit initialization vector (IV) to every message to create 64-bit key
  - 64 bit key used to generate stream of keys, $k_i^{IV}$
  - $k_i^{IV}$ used to encrypt ith byte, $d_i$, in frame:

$$c_i = d_i \text{ XOR } k_i^{IV}$$

  - IV and encrypted bytes, $c_i$ sent in frame
    - IV sent as plaintext

# 802.11 WEP encryption




Sender-side WEP encryption

# Breaking 802.11 WEP encryption

## Security hole:

- 24-bit IV, one IV per frame, -> IV's eventually reused
  - 99% probability the same IV reused after just 12000 frames (birthday paradox)
- IV transmitted in plaintext -> IV reuse detected

## Attack:

- Trudy causes Alice to encrypt plaintext $d_1\ d_2\ d_3\ d_4 \ldots$
- Trudy sees: $c_i = d_i$ XOR $k_i^{IV}$
- Trudy knows $c_i\ d_i$, so can compute $k_i^{IV}$
- Trudy knows encrypting key sequence $k_1^{IV}\ k_2^{IV}\ k_3^{IV} \ldots$
- Next time IV is used, Trudy can decrypt!

# IEEE 802.11i (Wifi Protected Access - WPA)

- IEEE 802.11 superceded by IEEE 802.11i
- 802.11i uses
  - Shared private key to establish a session key
  - Four-way handshake for authentication
  - Two nonces to prevent playback attacks
  - GTK (Group Temporal Key) to decrypt multicast and broadcast traffic
- Lightweight (pre-shared key) version for small business and home users

# 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