

# COMP 3331/9331: Computer Networks and Applications

Week 10

Network Security

**Reading Guide: Chapter 8: 8.1 – 8.5**

# Shape our Future

Complete your myExperience and shape  
the future of education at UNSW.

Click the  Experience link in Moodle

or login to [myExperience.unsw.edu.au](https://myExperience.unsw.edu.au)

(use z1234567@ad.unsw.edu.au to login)

The survey is confidential, your identity will never be released

Survey results are not released to teaching staff until after your results are published

# Network Security: Overview

## *Our goals:*

- ❖ understand principles of network security:
  - cryptography and its *many* uses beyond “confidentiality”
  - authentication
  - message integrity

# Network Security: roadmap

8.1 *What is network security?*

8.2 Principles of cryptography

8.3 Message integrity

8.4 Authentication

8.5 Securing email

8.6 - 8.9 SSL, IPSec, Firewall/IDS - **not covered.**

**There are several security electives offered**

# What is network security?

*confidentiality*: only sender, intended receiver should “understand” message contents

- sender encrypts message
- receiver decrypts message

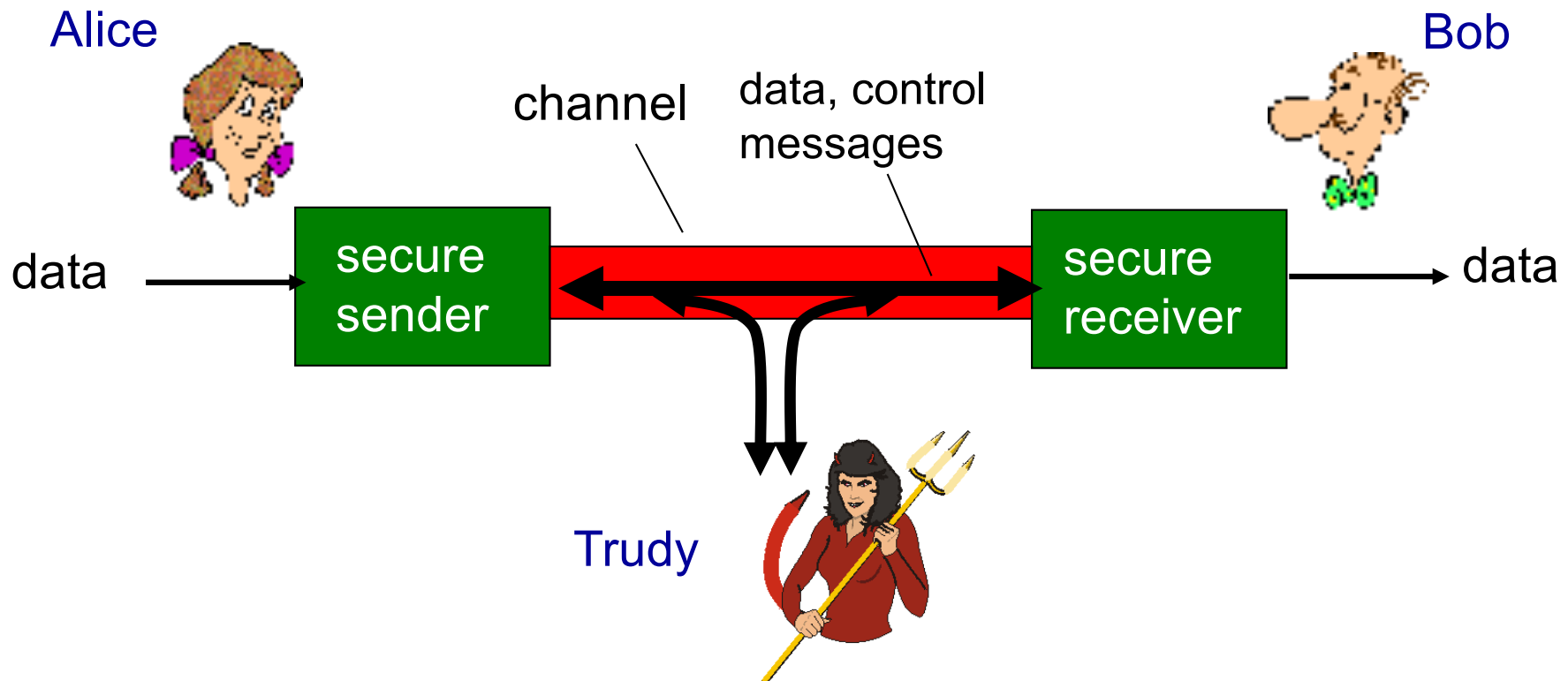
*authentication*: sender, receiver want to confirm identity of each other

*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) 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
- ❖ etc.

# 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)



# Network Security: roadmap

8.1 What is network security?

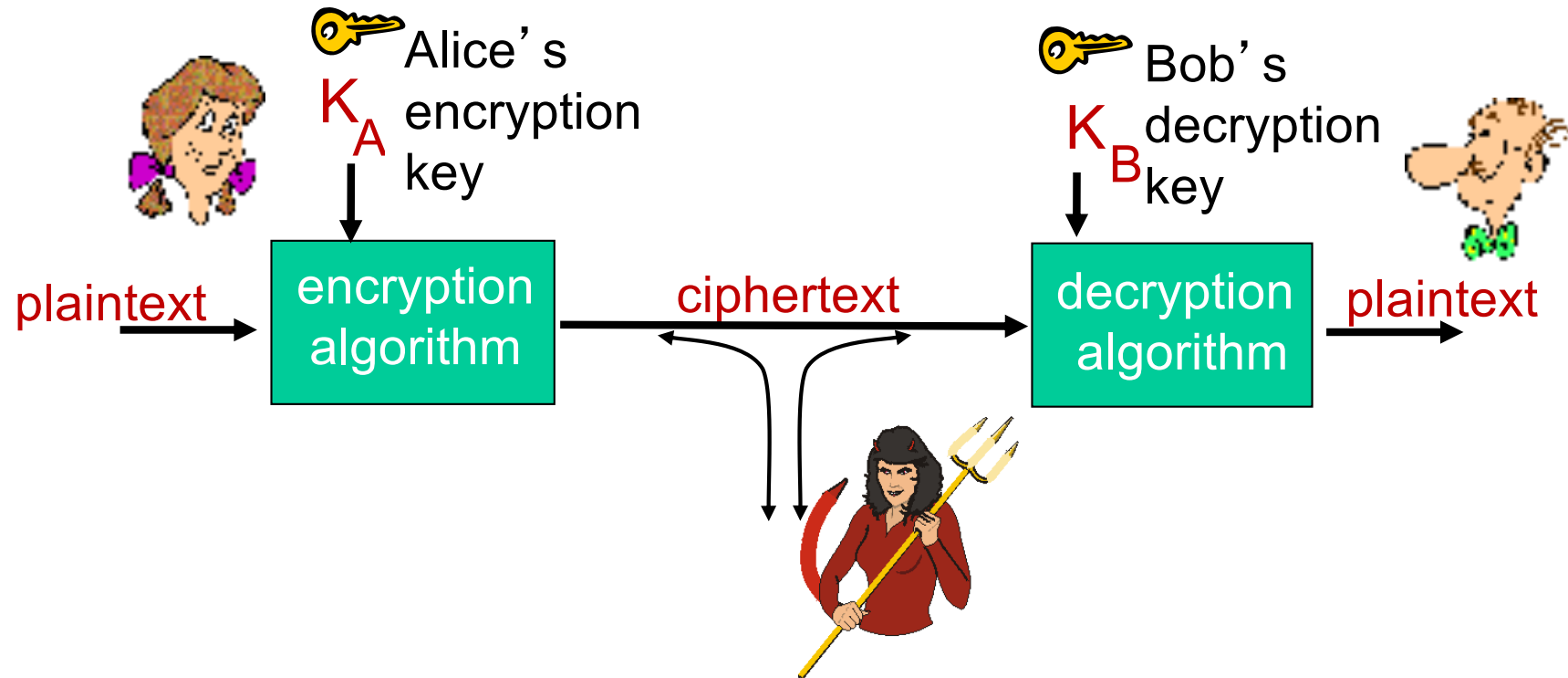
8.2 Principles of cryptography

8.3 Message integrity

8.4 Authentication

8.5 Securing email

# The language of cryptography

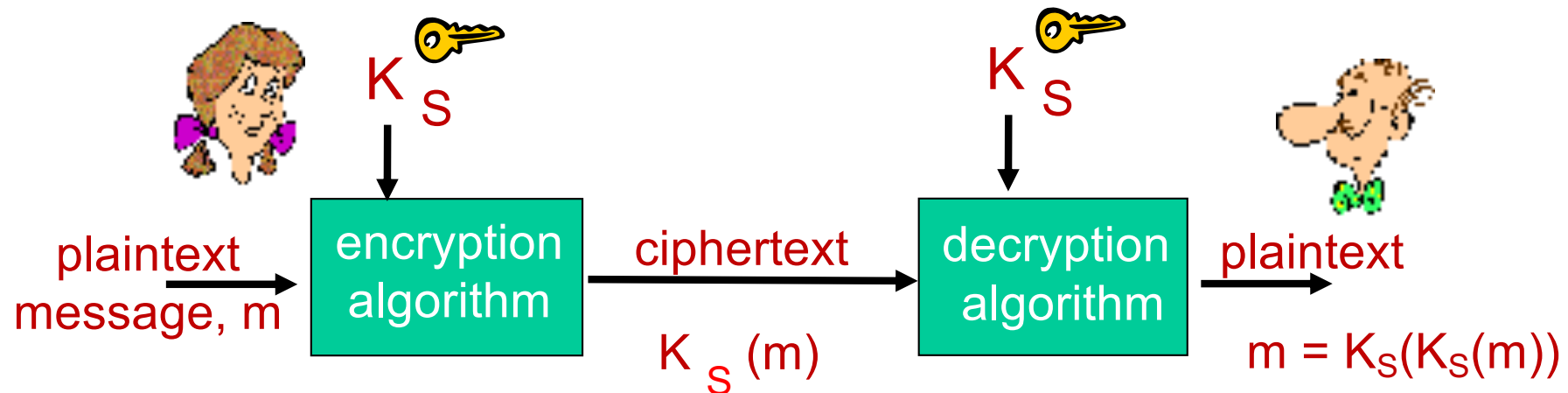


$m$  plaintext message

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

$m = K_B(K_A(m))$

# Symmetric key cryptography



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

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
- ❖ Ceaser Cipher: replace each letter of the alphabet with the letter standing three places further down the alphabet.

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

e.g.:      Plaintext: meet me after the party

            ciphertext: phhw ph diwhu wkh sduwb

 *Encryption key:  $c = (p+3) \bmod 26$*

*Each plaintext letter  $p$  substituted by the ciphertext letter  $c$*

*In general, we have  $c = (p+k) \bmod 26$*

*where  $k$  is in range 1 to 25*

# Simple encryption scheme

- ❖ With only 25 possible keys, the Caesar cipher is vulnerable to brute force cryptanalysis
- ❖ Cipher can be any permutation of the 26 alphabet characters

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

We have  $26!$  ( $> 4 \times 10^{26}$ ) possible keys

# 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:**

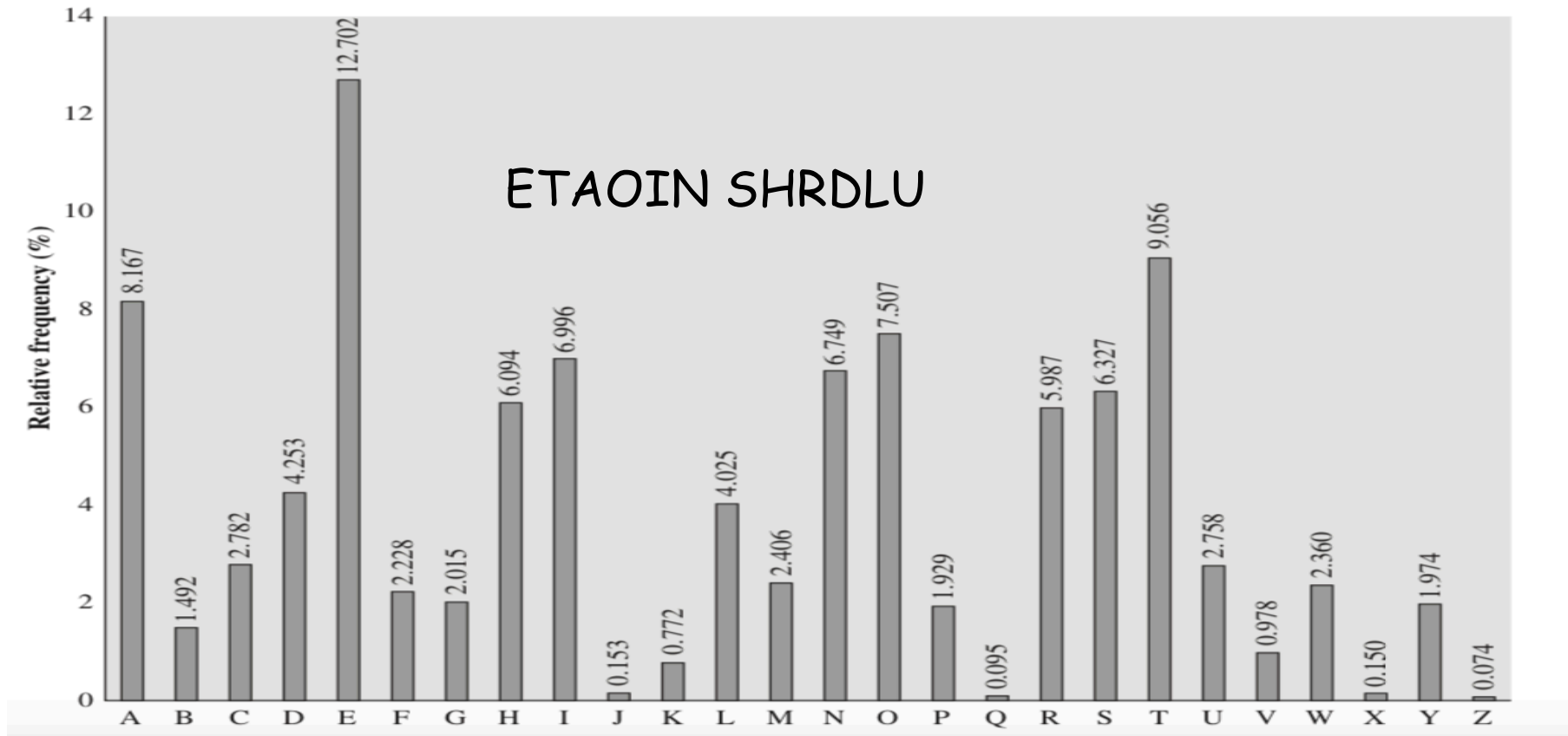
Trudy has (part of) plaintext corresponding to ciphertext

- e.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,b

- ❖ **chosen-plaintext attack:**

Trudy can get ciphertext for chosen plaintext

# Breaking an encryption scheme



Frequency Histogram Analysis for letters in English language

Monoalphabetic ciphers are easy to break because they reflect the frequency data of the original alphabet

# A more sophisticated encryption approach

- ❖ Polyalphabet ciphers
- ❖  $n$  substitution ciphers,  $M_1, M_2, \dots, M_n$
- ❖ cycling pattern:
  - e.g.,  $n=4$ : and key is  $M_1, M_3, M_4, M_3, M_2; M_1, M_3, M_4, M_3, M_2; \dots$
- ❖ 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



# Two types of symmetric ciphers

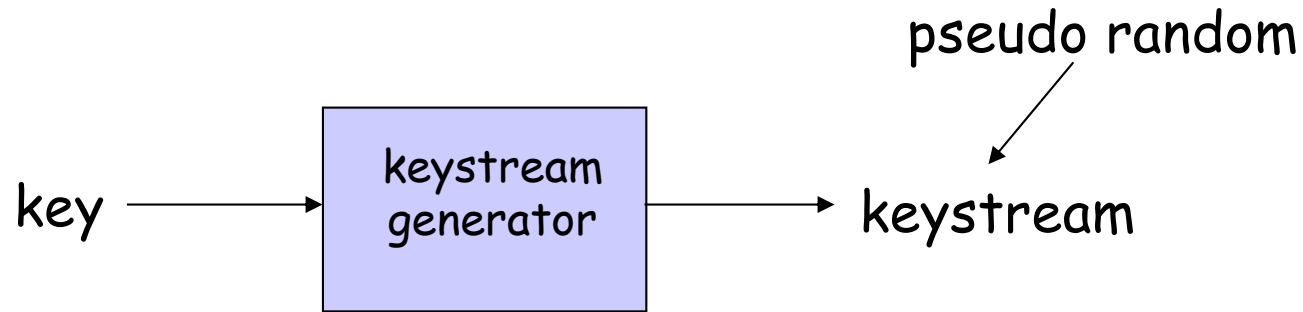
## ❖ **Stream ciphers**

- encrypt one bit at time

## ❖ **Block ciphers**

- Break plaintext message in equal-size blocks
- Encrypt each block as a unit

# Stream Ciphers



- ❖ Combine each bit of keystream with bit of plaintext to get bit of ciphertext
- ❖  $m(i)$  = ith bit of message
- ❖  $ks(i)$  = ith bit of keystream
- ❖  $c(i)$  = ith 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

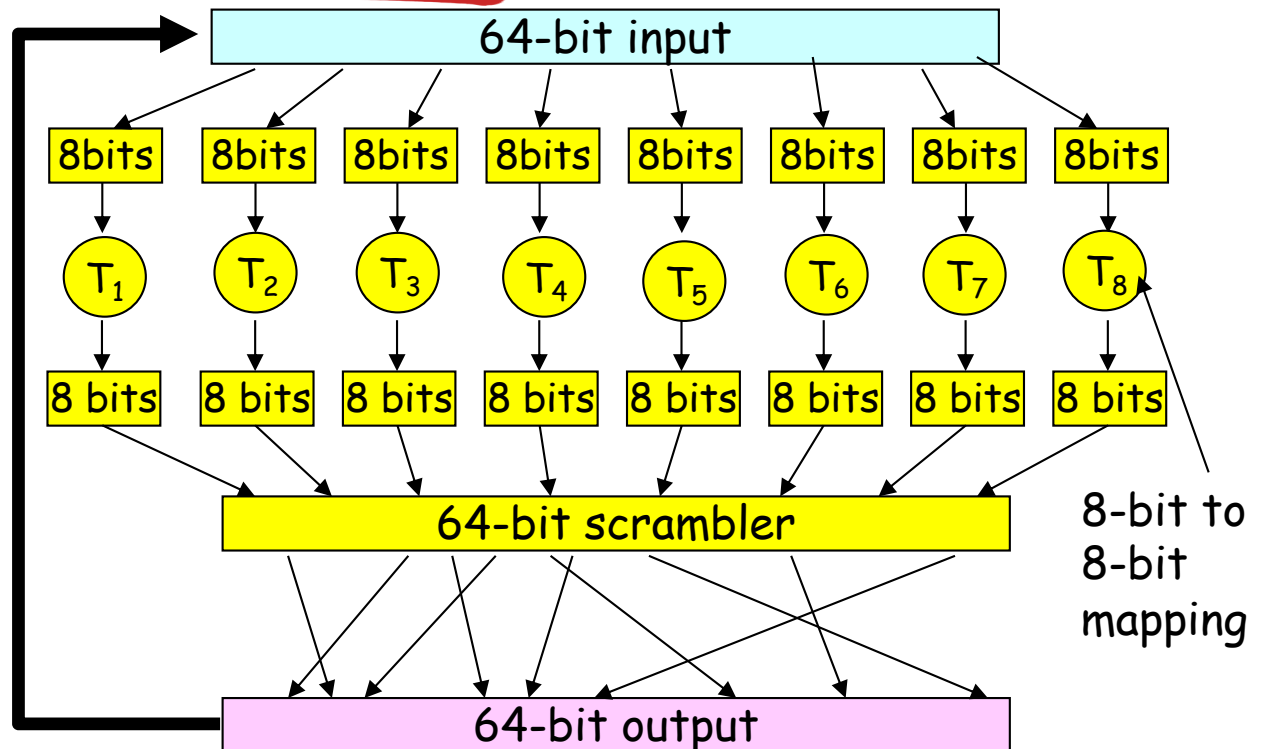
# Block Cipher

- ❖ Ciphertext processed as  $k$  bit blocks
- ❖ 1-to-1 mapping is used to map  $k$ -bit block of plaintext to  $k$ -bit block of ciphertext
- ❖ E.g:  $k=3$  (see table)
  - $010110001111 \Rightarrow 101000111001$
- ❖ Possible permutations =  $8!$  (40,320)
- ❖ To prevent brute force attacks
  - Choose large  $K$  (64, 128, etc)
- ❖ Full-table block ciphers not scalable
  - E.g., for  $k = 64$ , a table with  $2^{64}$  entries required
  - instead use function that simulates a randomly permuted table

Input	Output
000	110
111	001
001	111
010	101
011	100
100	011
101	010
110	000

## Block Cipher (contd.)

- ❖ If only a single round, then one bit of input affects at most 8 bits of output
- ❖ In 2<sup>nd</sup> 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



# 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 using distributed computing
  - no known good analytic attack
- ❖ making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

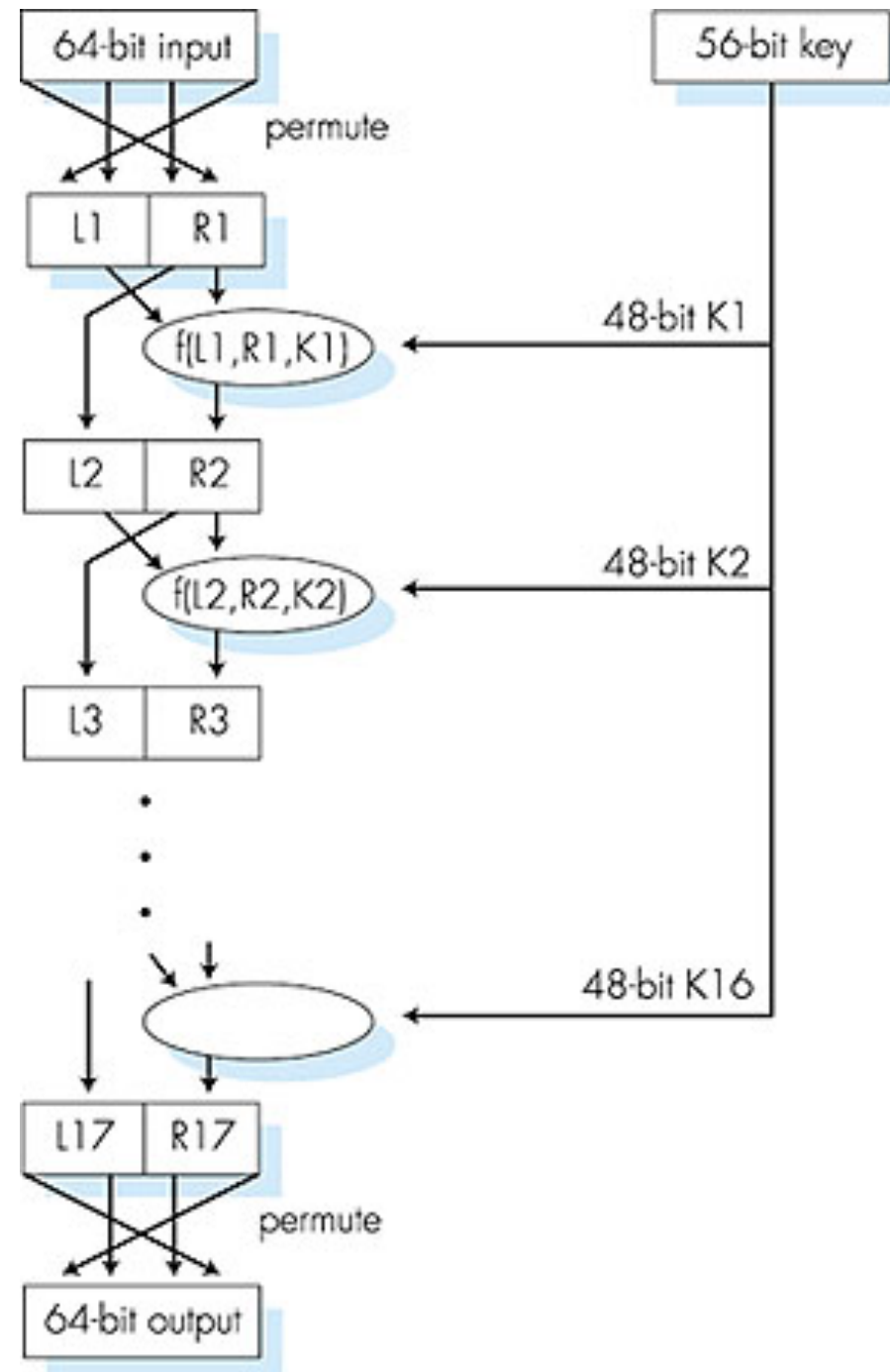
# Symmetric key crypto: DES

## *DES operation*

initial permutation

16 identical “rounds” of  
function application,  
each using different 48  
bits of key

final permutation



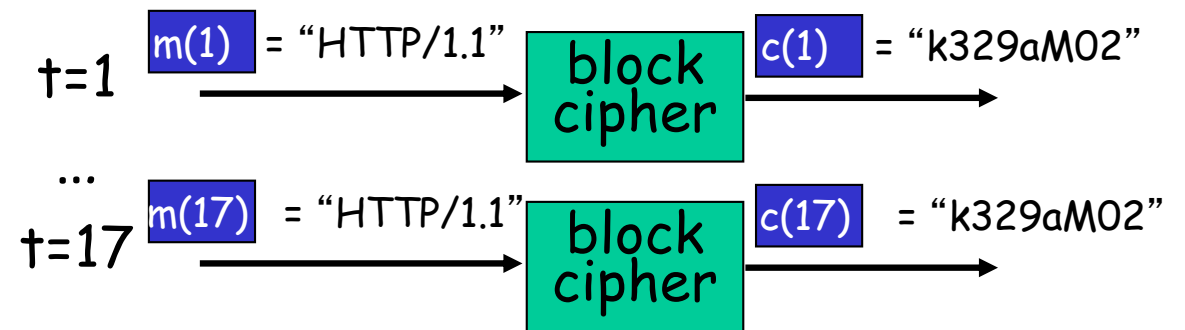
# AES: Advanced Encryption Standard

- ❖ 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



# Cipher Block Chaining

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



- *Use random numbers:* XOR  
ith input block,  $m(i)$  and  
random number  $r(i)$  and  
apply block-cipher  
encryption algorithm
  - $C(i) = K_s(m(i) \oplus r(i))$
  - Send across  $c(i)$  and  $r(i)$

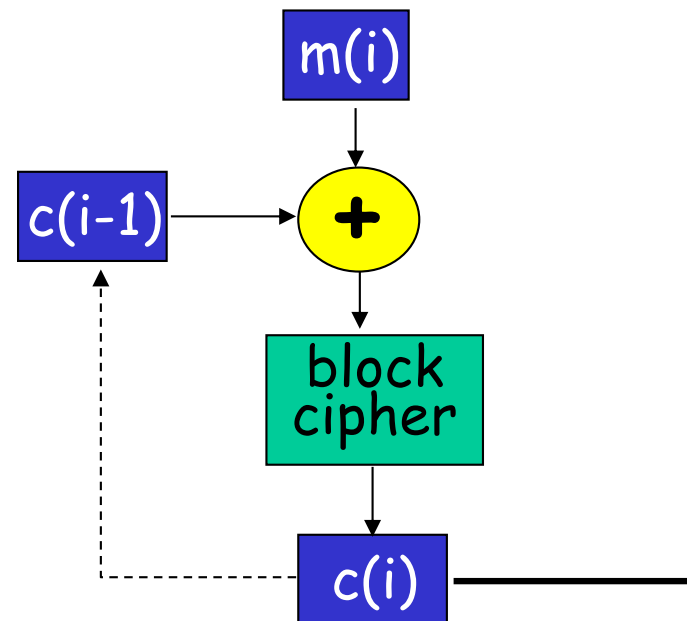
# CBC Example

- ❖ Plaintext: 010 010 010
- ❖ If no CBC, sent txt : 101 101 101
  - 1-to-1 mapping table used
- ❖ Lets use the following random bits
  - r1: 001, r2: 111, r3: 100
  - XoR the plaintext with these random bits
  - $010 \text{ XoR } 001 = 011$
  - Now do table lookup for 011  $\rightarrow$  100
- ❖ We get  $c(1)=100$ ,  $c(2)=010$  and  $c(3)=000$ , although plaintext is the same (010)
- ❖ Need to transmit twice as many bits ( $c(i)$  as well as  $r(i)$ )

Input	Output
000	110
111	001
001	111
010	101
011	100
100	011
101	010
110	000

# Cipher Block Chaining

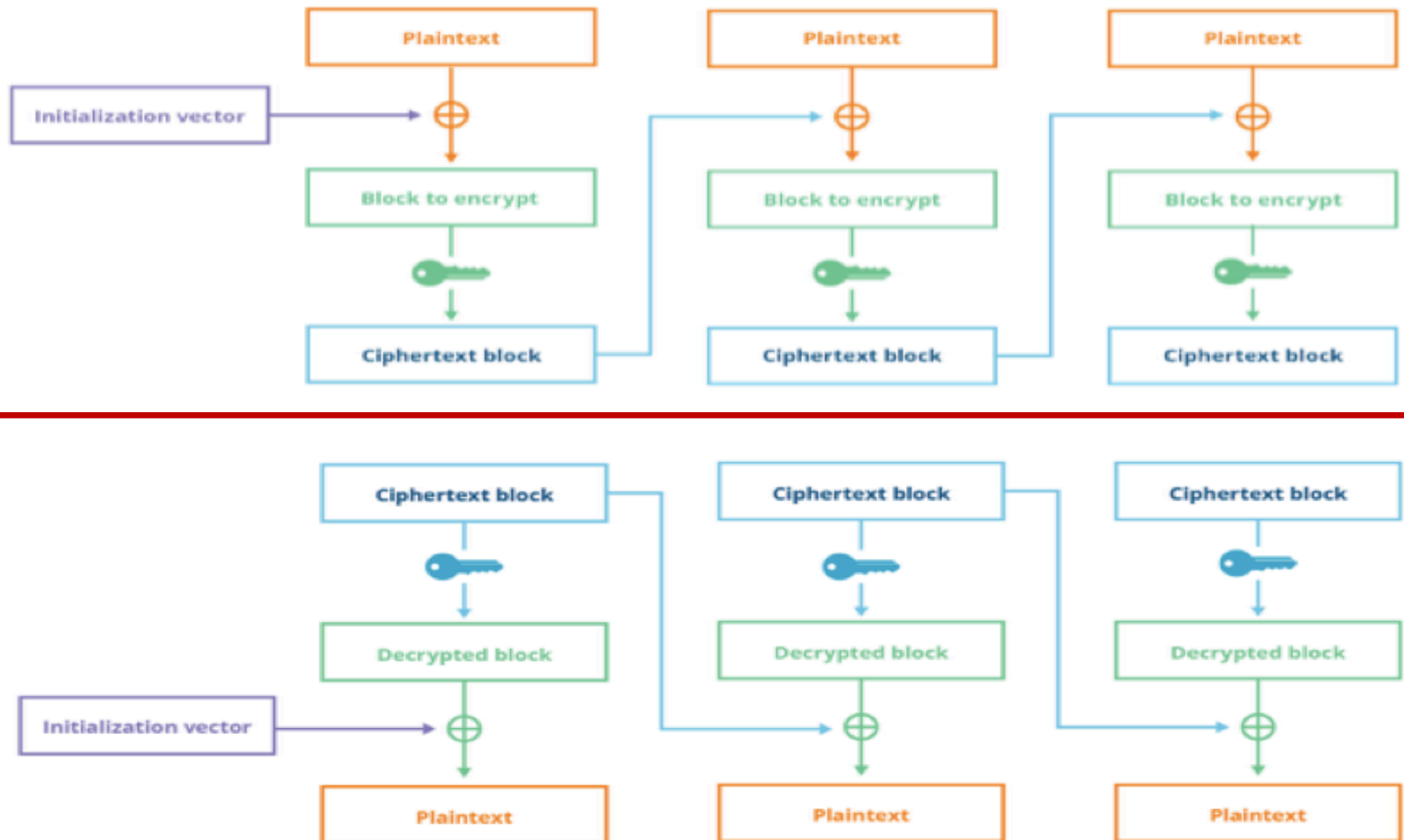
- *cipher block chaining*: send only one random value alongwith the very first message block, and then have the sender and receiver use the computed cipher block in place of the subsequent random number
- XOR ith input block,  $m(i)$ , with previous block of cipher text,  $c(i-1)$ 
  - $c(0)$  is an initialisation vector (random) transmitted to receiver in clear



# Cipher Block Chaining (CBC)

- ❖ CBC generates its own random numbers
  - Have encryption of current block depend on result of previous block
  - $c(i) = K_S( m(i) \oplus c(i-1) )$
  - $m(i) = K_S( c(i) ) \oplus c(i-1)$
- ❖ 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 (CBC)



# Public Key Cryptography



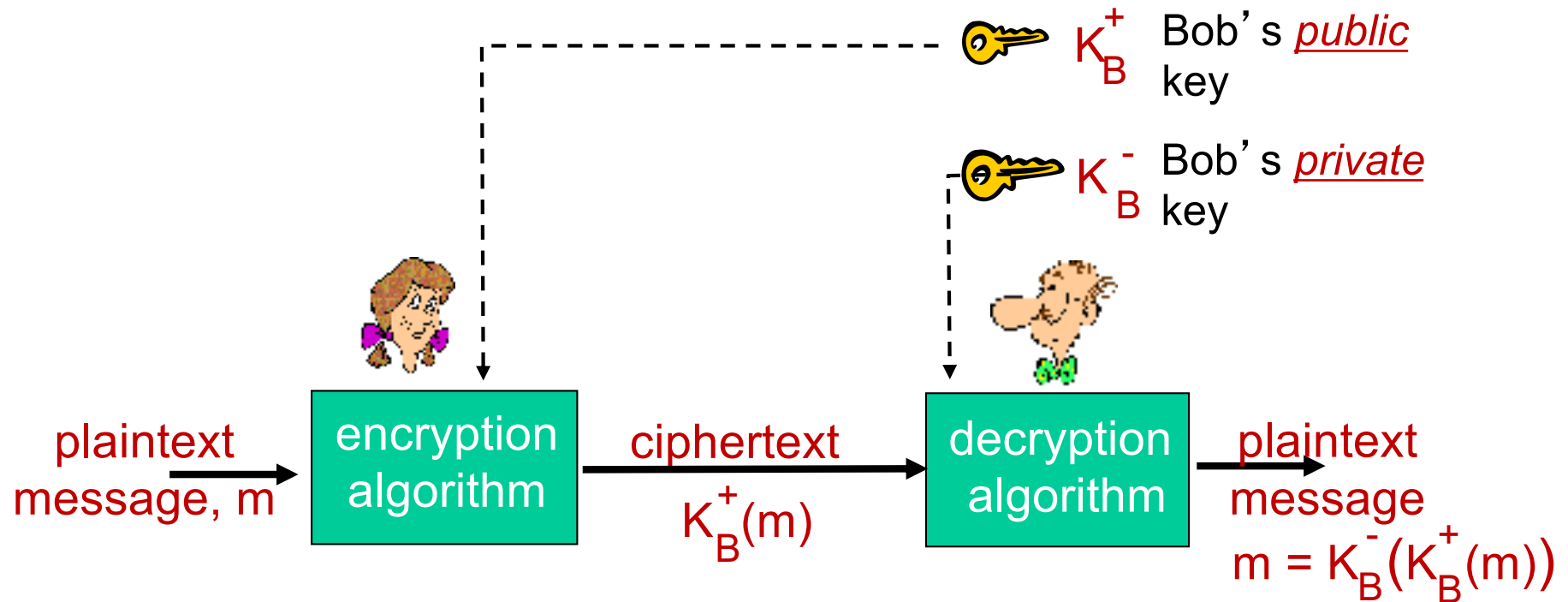
## *symmetric key crypto*

- ❖ 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:  $x=14$ ,  $n=10$ ,  $d=2$ :

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

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

Note: You don't need to know this for the exam

# 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!*

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

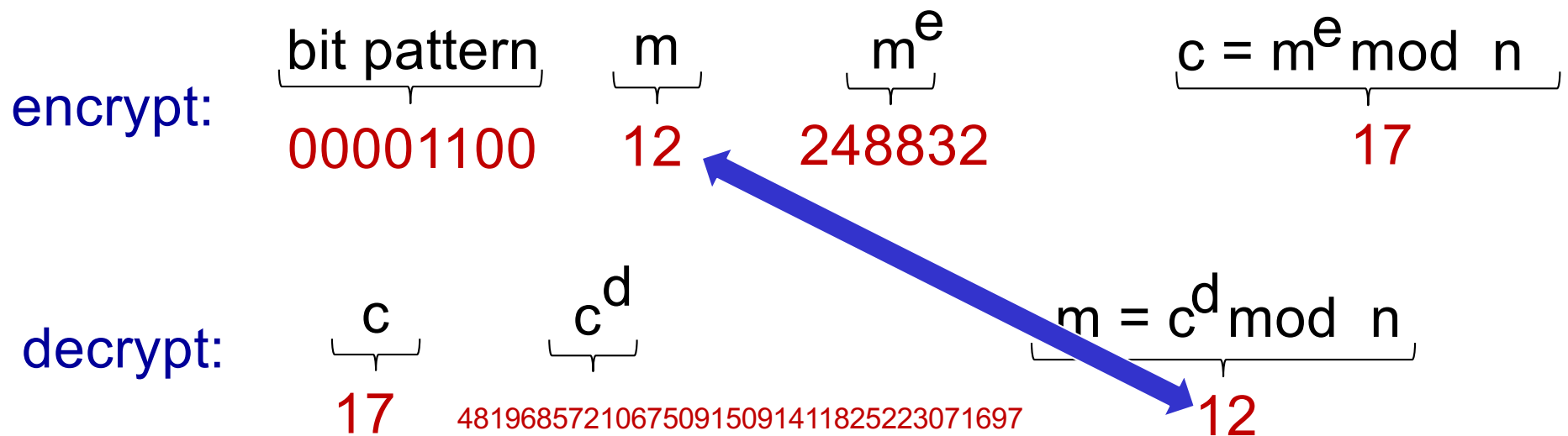
# RSA example:

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.



# 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 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



# Network Security: roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity

8.4 Authentication

8.5 Securing email

# 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”

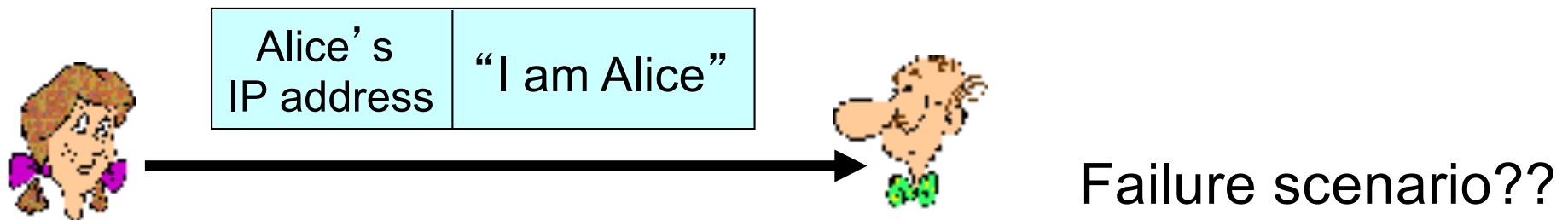


“I am Alice”

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

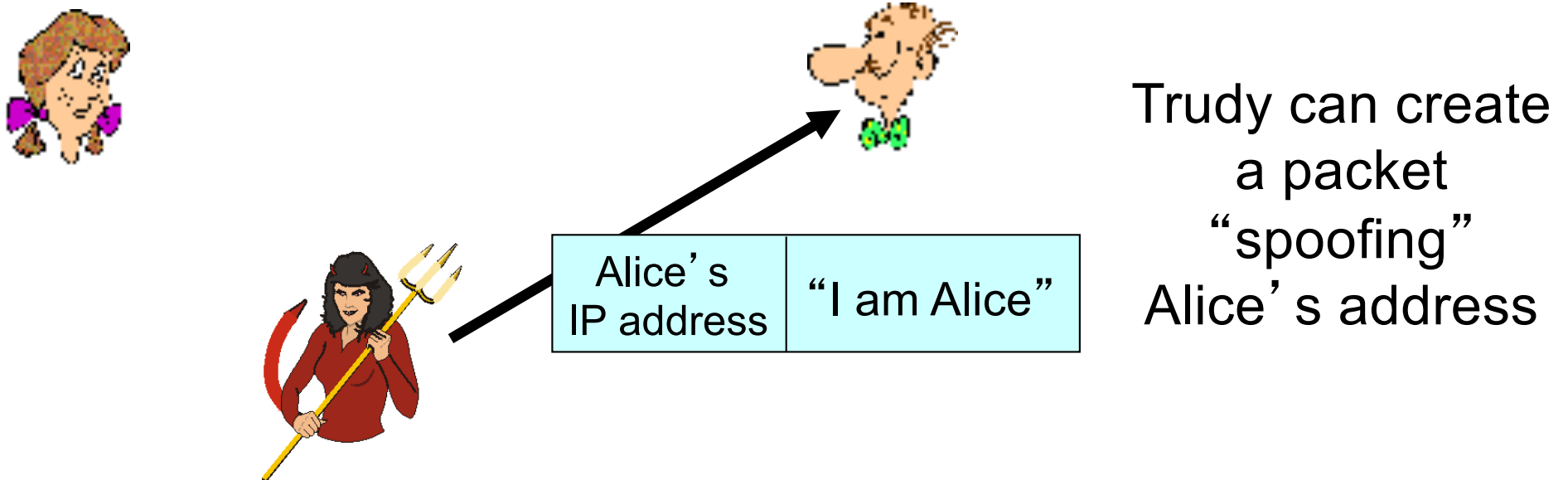
# Authentication: another try

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



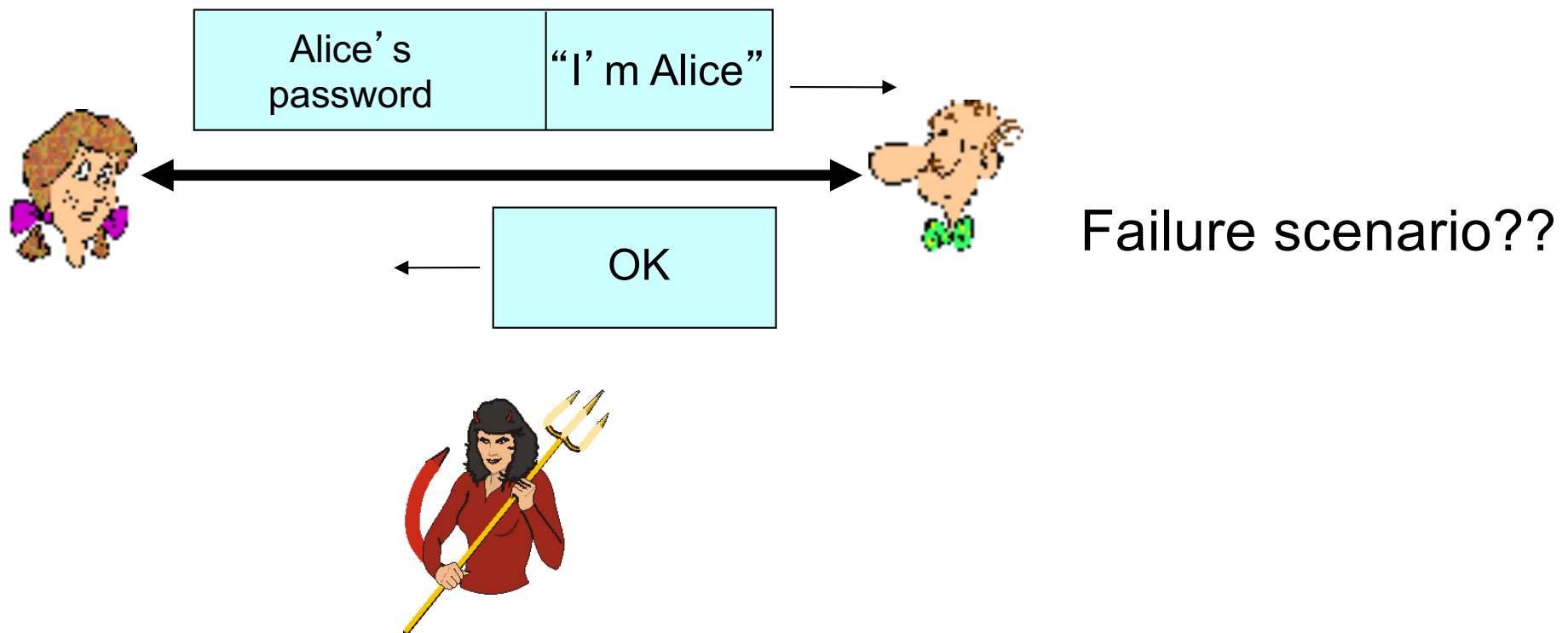
# Authentication: another try

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



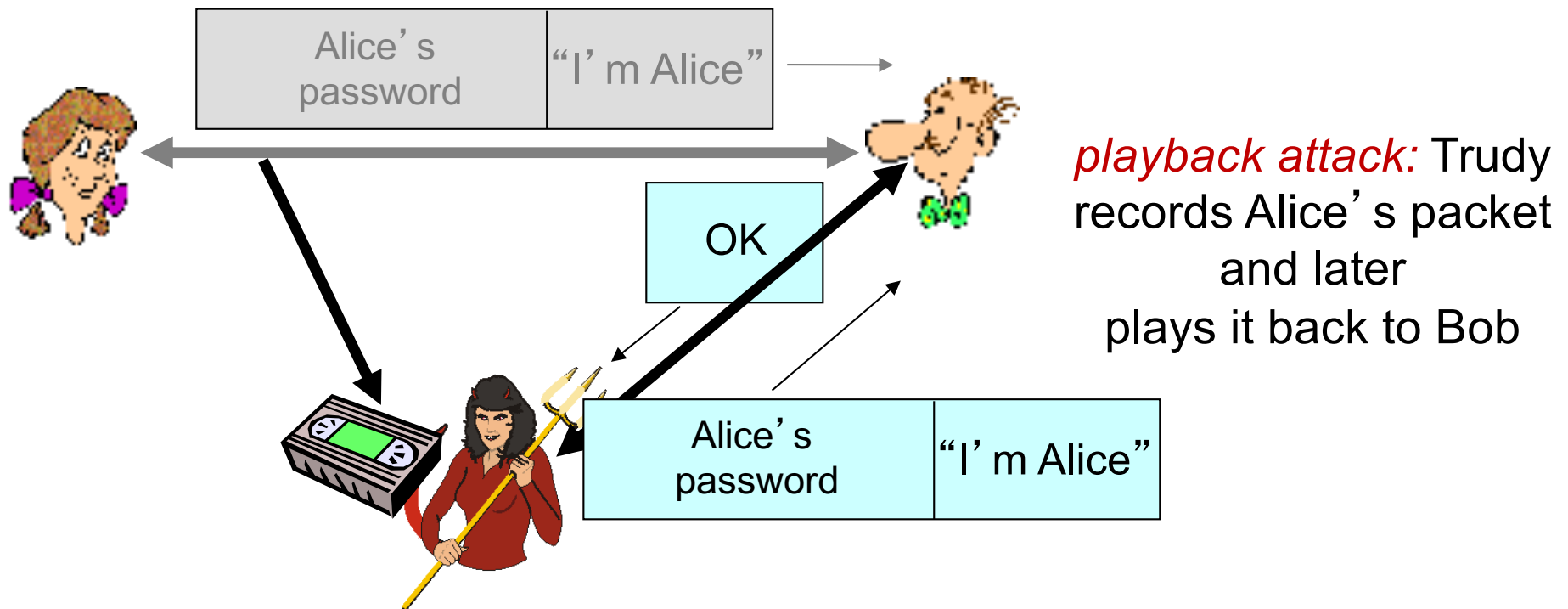
# Authentication: another try

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



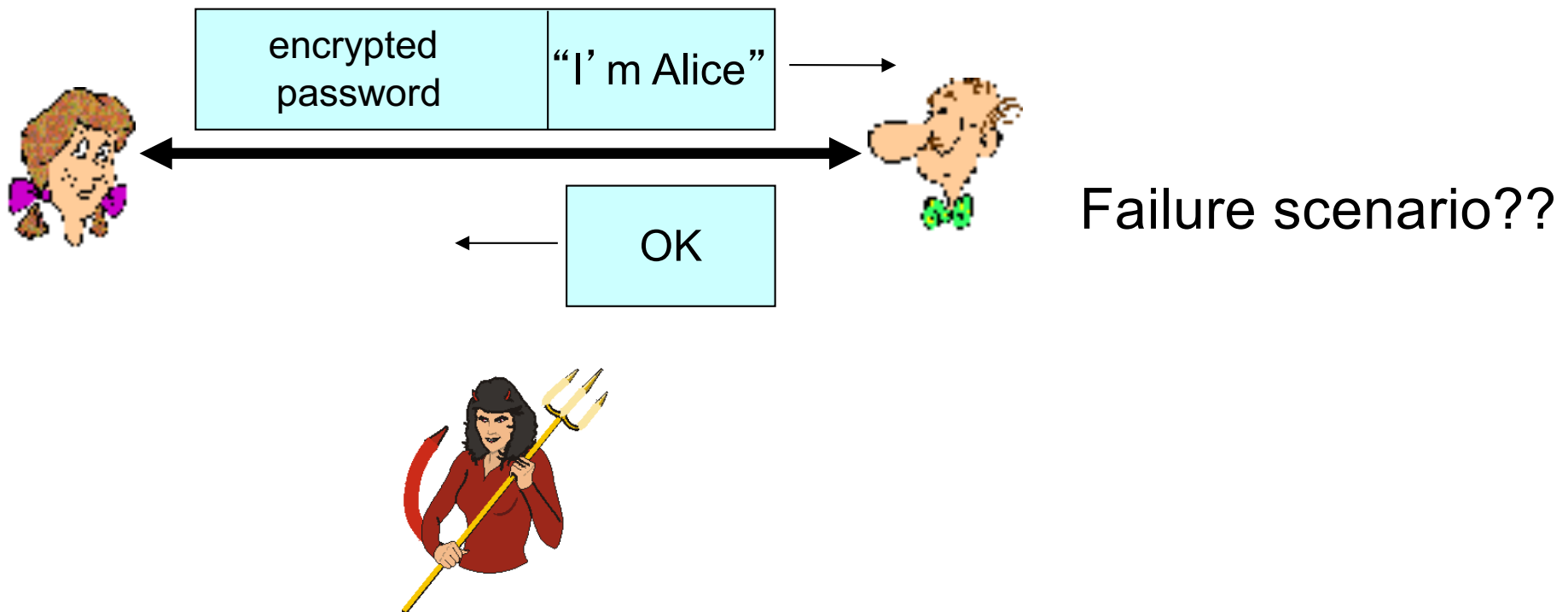
# Authentication: another try

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



# Authentication: yet another try

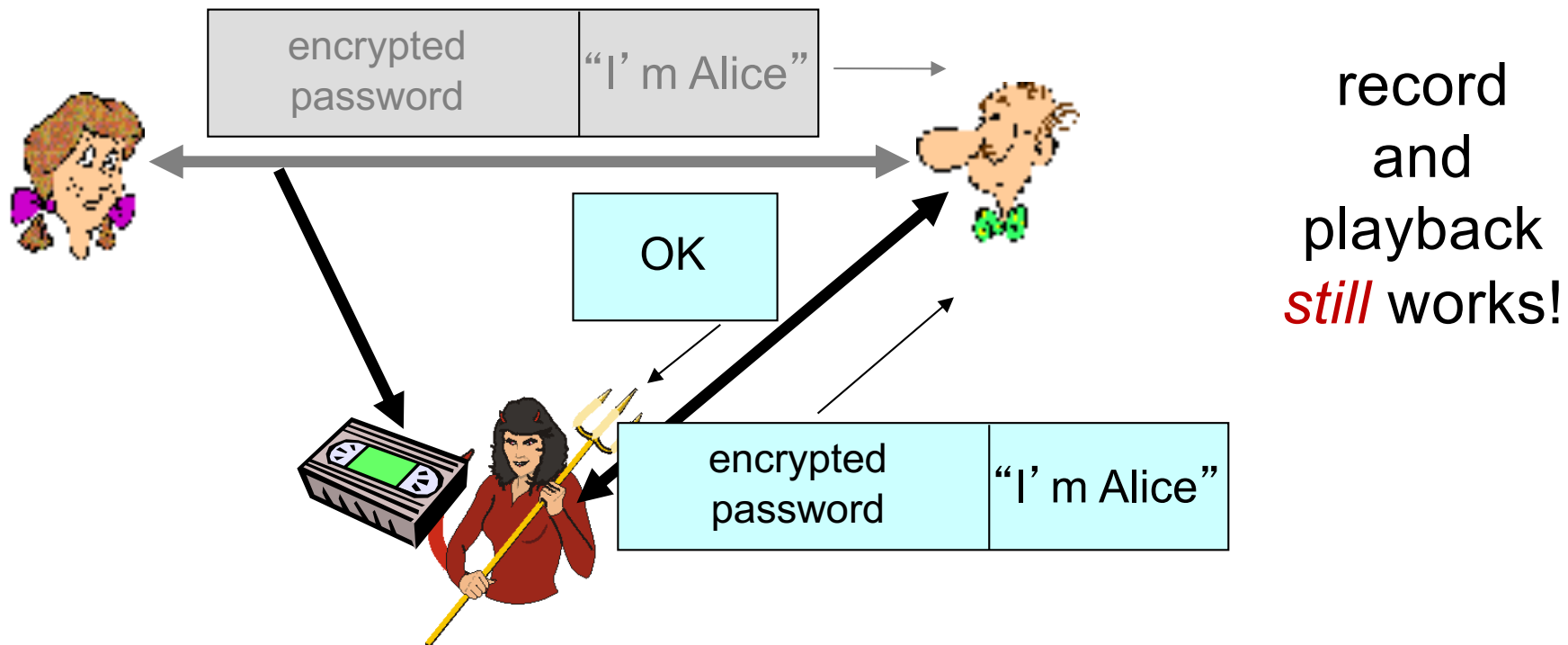
*Protocol ap3.1:* Alice says “I am Alice” and sends her *encrypted* secret password to “prove” it.





# Authentication: yet another try

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

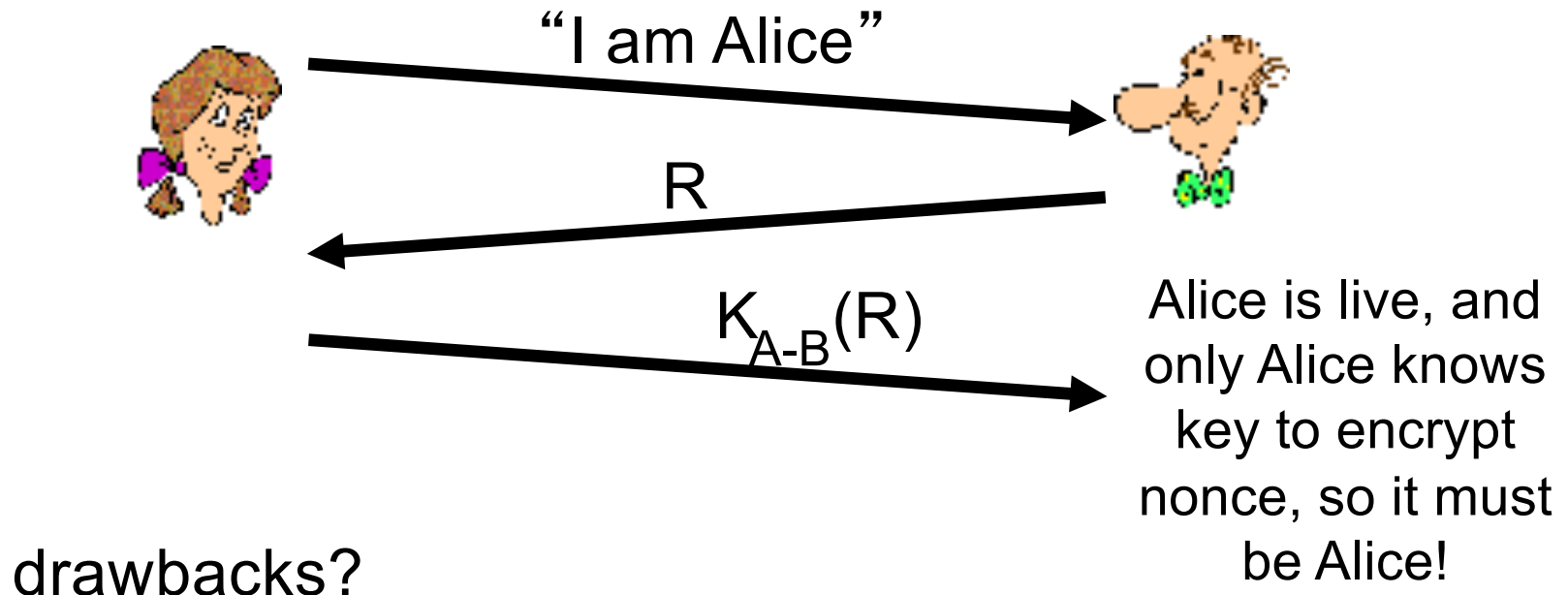


# Authentication: yet another try

*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

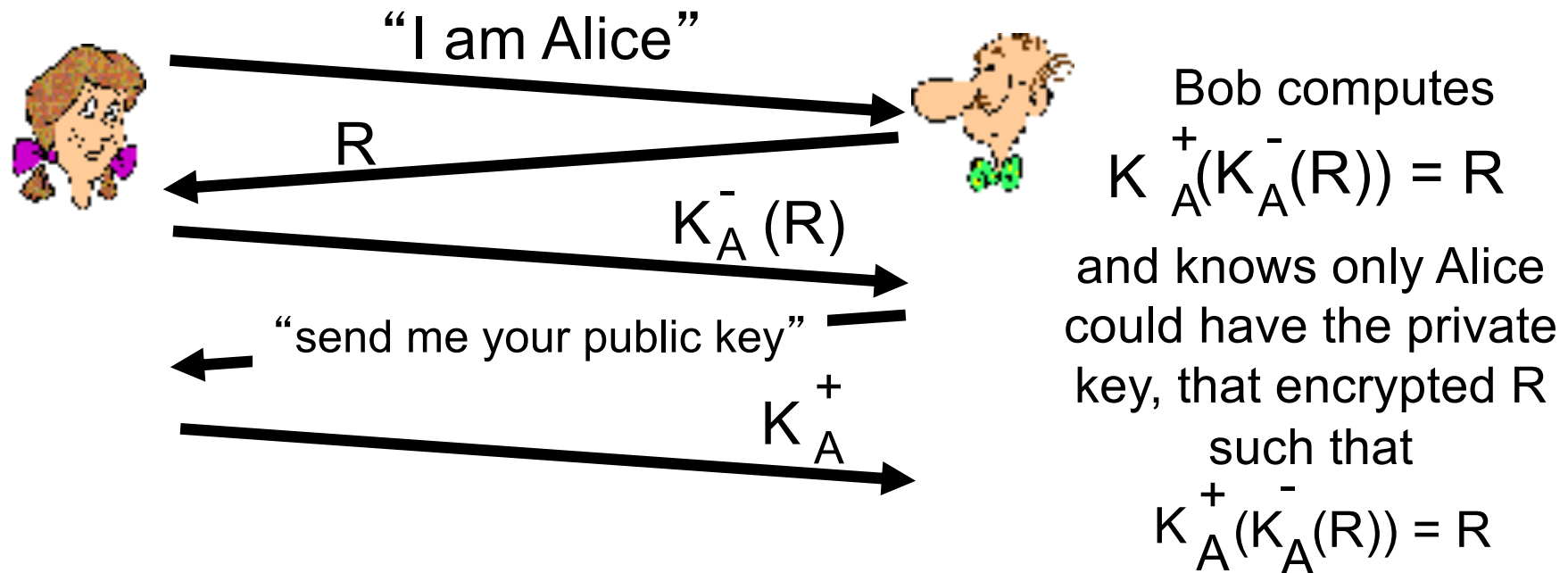


# Authentication: ap5.0

ap4.0 requires shared symmetric key

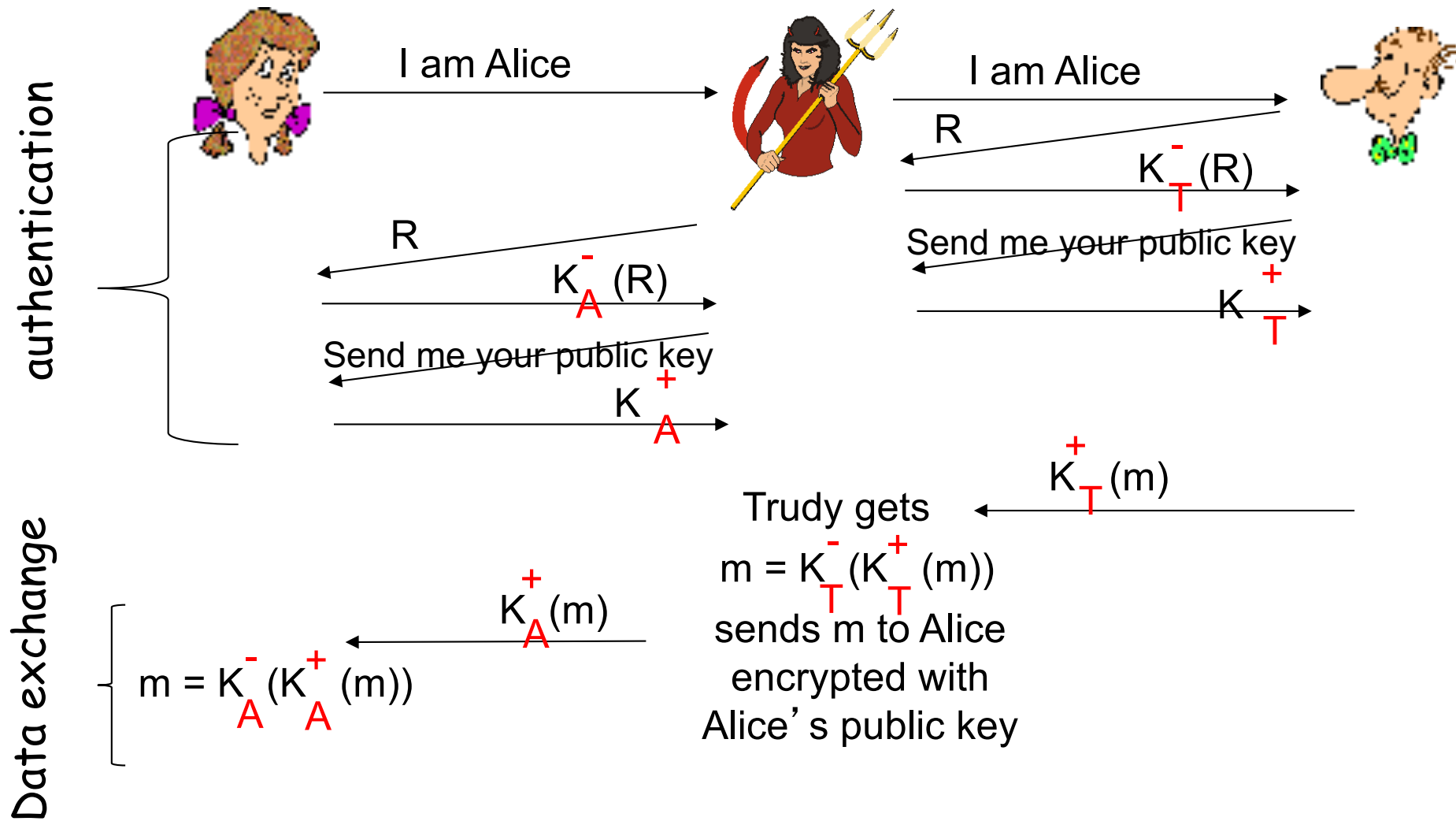
❖ can we authenticate using public key techniques?

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



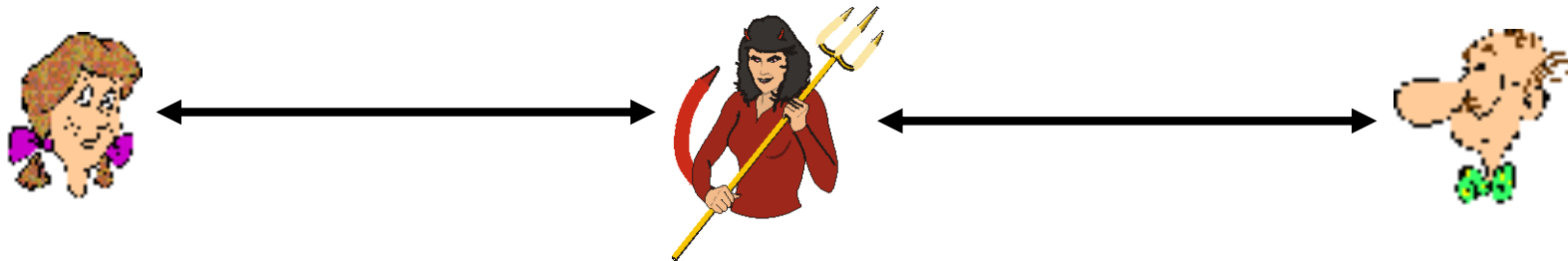
# ap5.0: security hole

*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!

# Network Security: roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity

8.4 Authentication

8.5 Securing email

# Confidentiality vs Integrity

- ❖ Confidentiality: message private and secret
- ❖ Integrity: protection against message tampering
- ❖ Encryption alone may not guarantee integrity
  - Attacker can modify message under encryption without learning what it is
- ❖ Public Key Crypto Standard (PKCS)
  - “RSA encryption is intended primarily to provide confidentiality .... It is not intended to provide integrity”
- ❖ Both confidentiality and integrity are needed for security

# Digital signatures

cryptographic technique analogous to hand-written signatures:

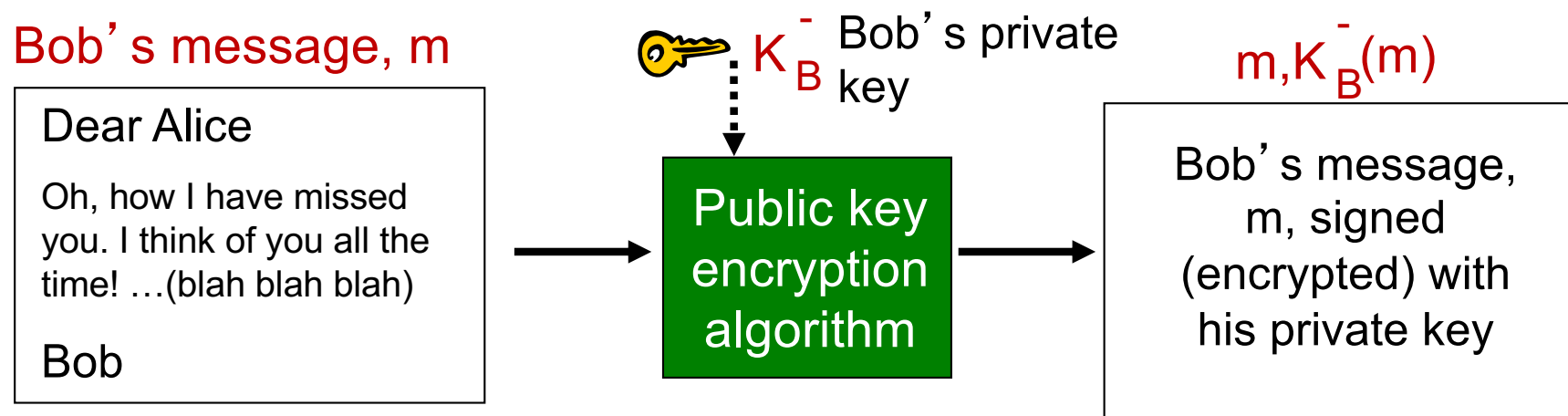
- ❖ sender (Bob) digitally signs document, establishing he is document owner/creator.
- ❖ *verifiable, nonforgeable*: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document



# 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:

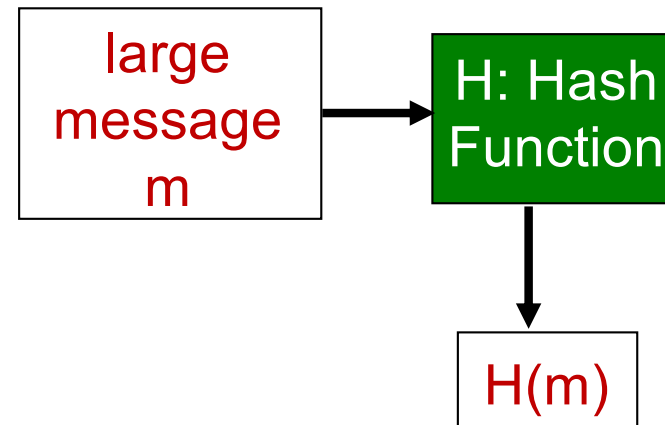
- ✓ Alice can take  $m$ , and signature  $K_B^-(m)$  to court and prove that Bob signed  $m$

# 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:

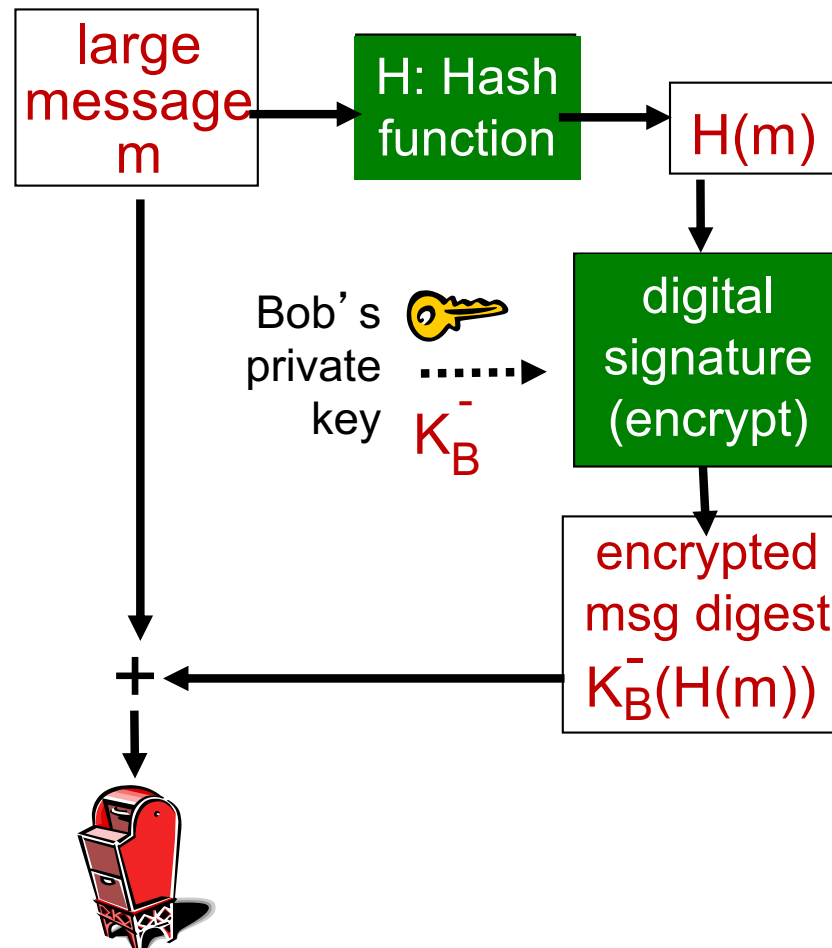
<u>message</u>	<u>ASCII format</u>		<u>message</u>	<u>ASCII format</u>
I O U 1	49 4F 55 31		I O U <u>9</u>	49 4F 55 <u>39</u>
0 0 . 9	30 30 2E 39		0 0 . <u>1</u>	30 30 2E <u>31</u>
9 B O B	39 42 D2 42		9 B O B	39 42 D2 42
<hr/>			<hr/>	
	B2 C1 D2 AC	different messages		B2 C1 D2 AC
but identical checksums!				

# Hash function algorithms

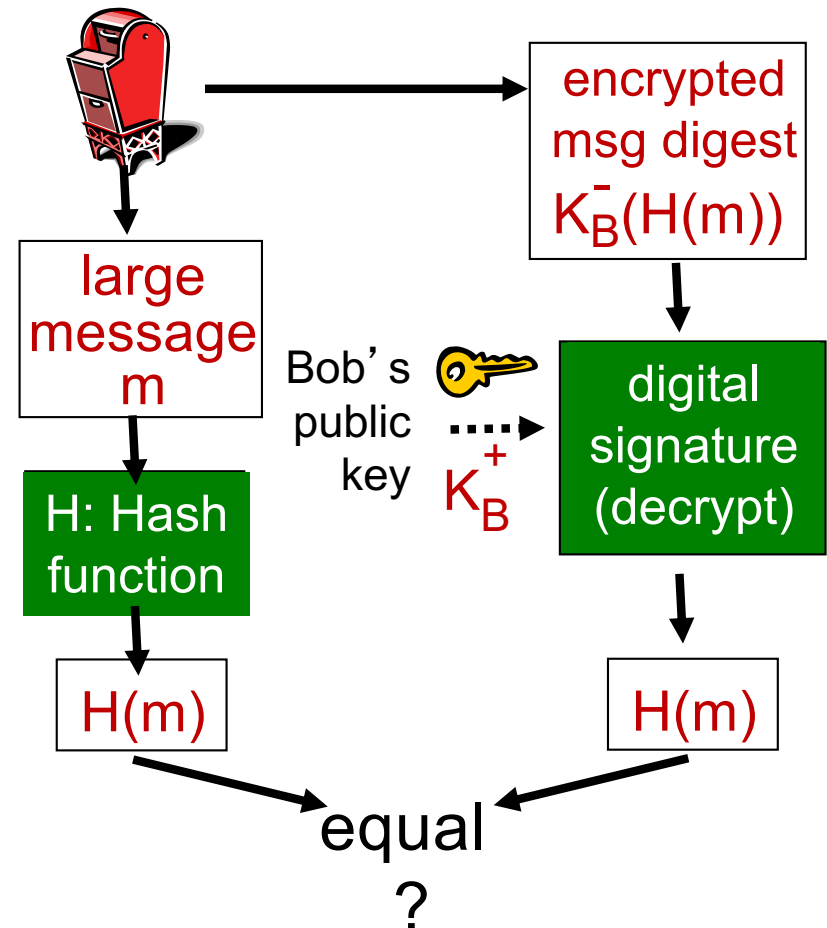
- ❖ MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string  $x$ , appears difficult to construct msg  $m$  whose MD5 hash is equal to  $x$
- ❖ SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest
- ❖ SHA-2 and SHA-3 (recent standard) are better - security

# Digital signature = signed message digest

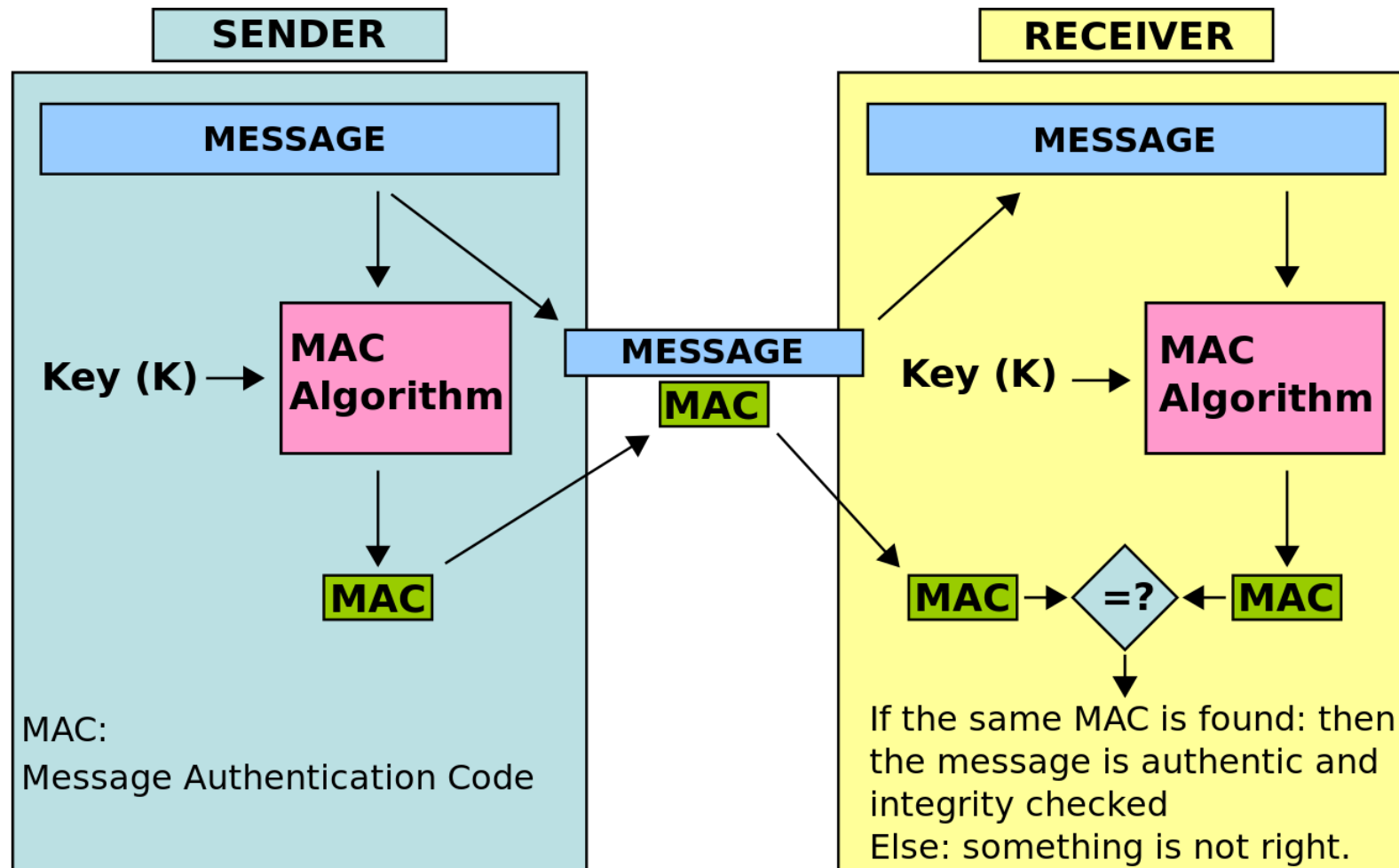
Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:



# Message Authentication Code (MAC)



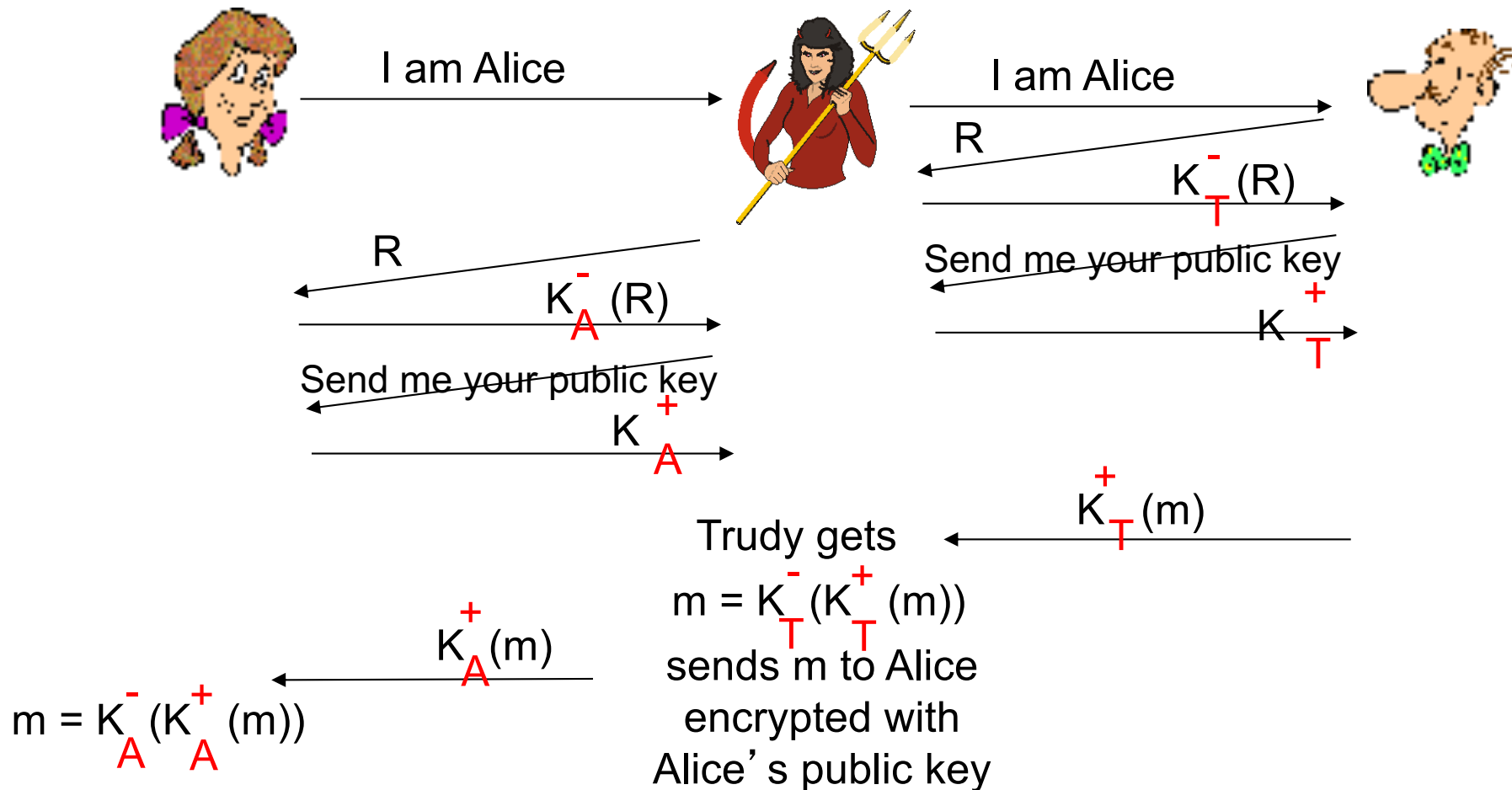
**Digital signatures use asymmetric key crypto**

**MAC allows a way to sign a message but using symmetric key**

**Requires a shared secret key**

# Recall: ap5.0 security hole

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





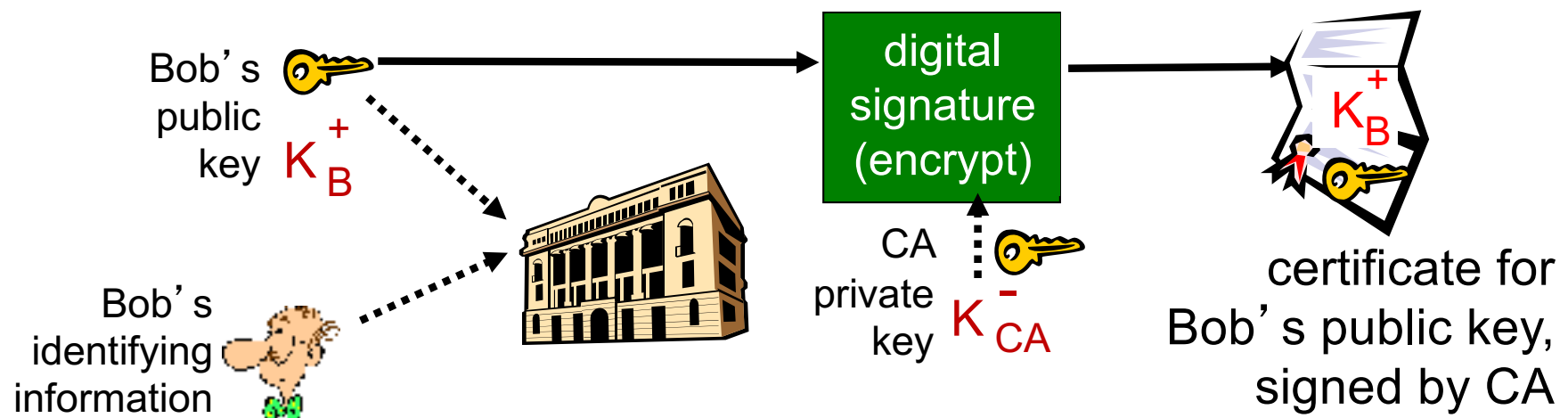
# Public-key certification

- ❖ 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



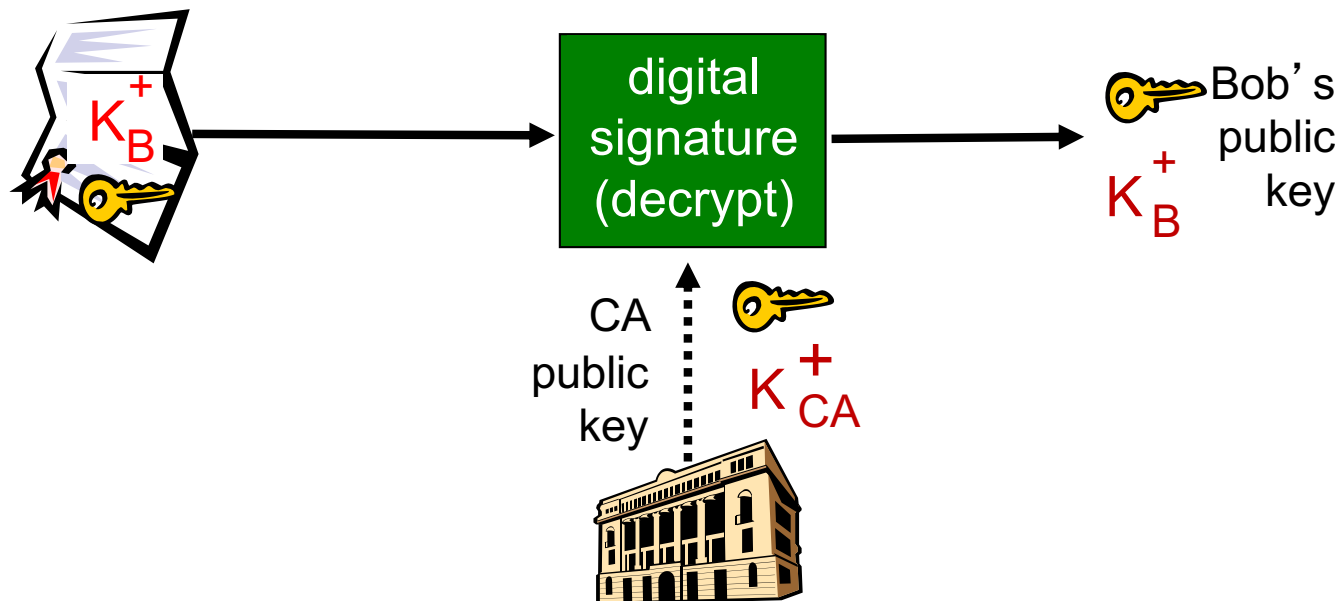
# Certification authorities

- ❖ *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”



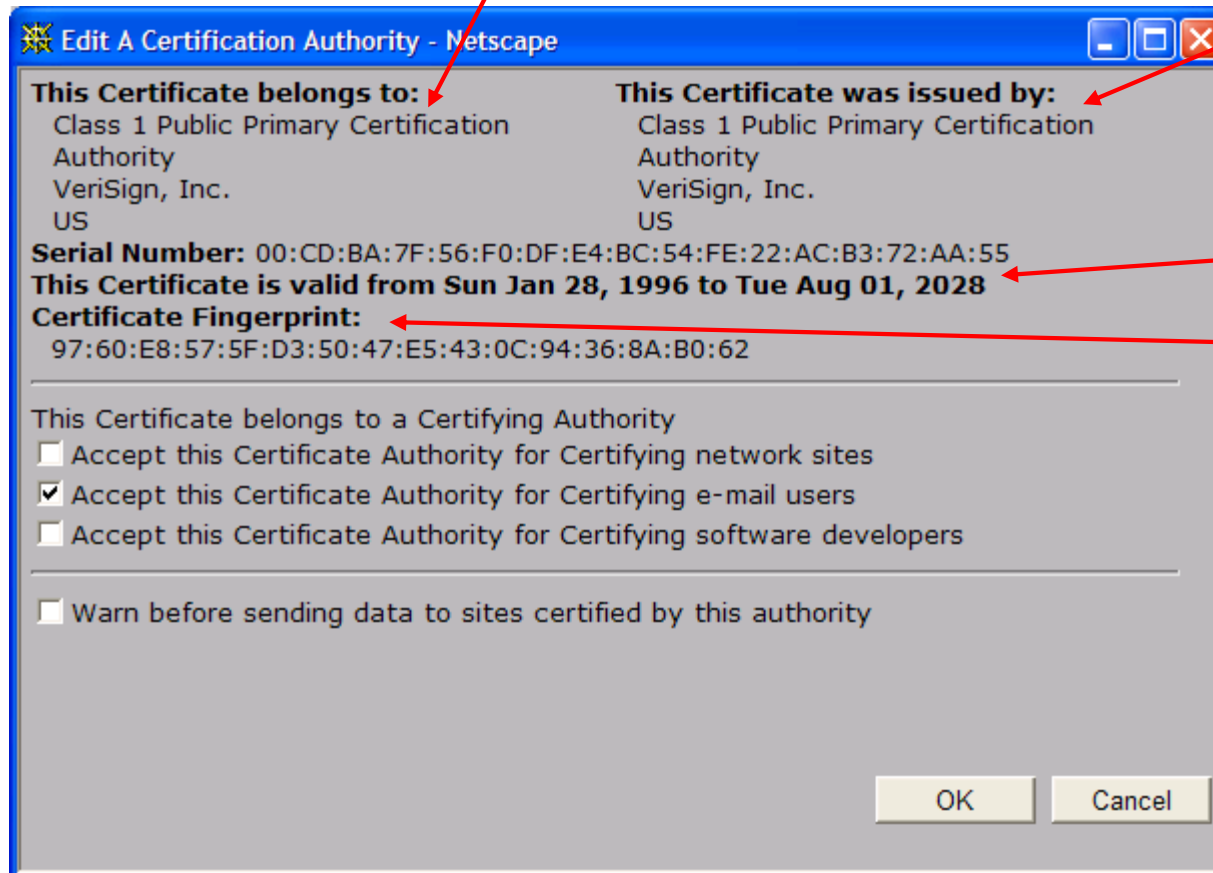
# Certification authorities

- ❖ 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

# Certificates: summary

- ❖ Primary standard X.509 (RFC 2459)
- ❖ Certificate contains:
  - Issuer name
  - Entity name, address, domain name, etc.
  - Entity's public key
  - Digital signature (signed with issuer's private key)
- ❖ Public-Key Infrastructure (PKI)
  - Certificates and certification authorities
  - Often considered “heavy”

# Quiz



- ❖ Suppose Bob wants to send Alice a digital signature for the message  $m$ . To create the digital signature
  - a) Bob applies a hash function to  $m$  and encrypts the result with his private key
  - b) Bob applies a hash function to  $m$  and encrypts the result with Alice's public key
  - c) Bob encrypts  $m$  with his private key and then applies a hash function to the result
  - d) Bob applies a hash function to  $m$  and encrypts the result with his public key

ANSWER: A (see Slide 62)

# Quiz



❖ Suppose a CA creates Bob's certificate, which binds Bob's public key to Bob. This certificate is signed with

- a) Bob's private key
- b) Bob's public key
- c) The CA's private key
- d) The CA's public key
- e) Donald Trump's key

ANSWER: C (see Slide 66)



# Network Security: roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity

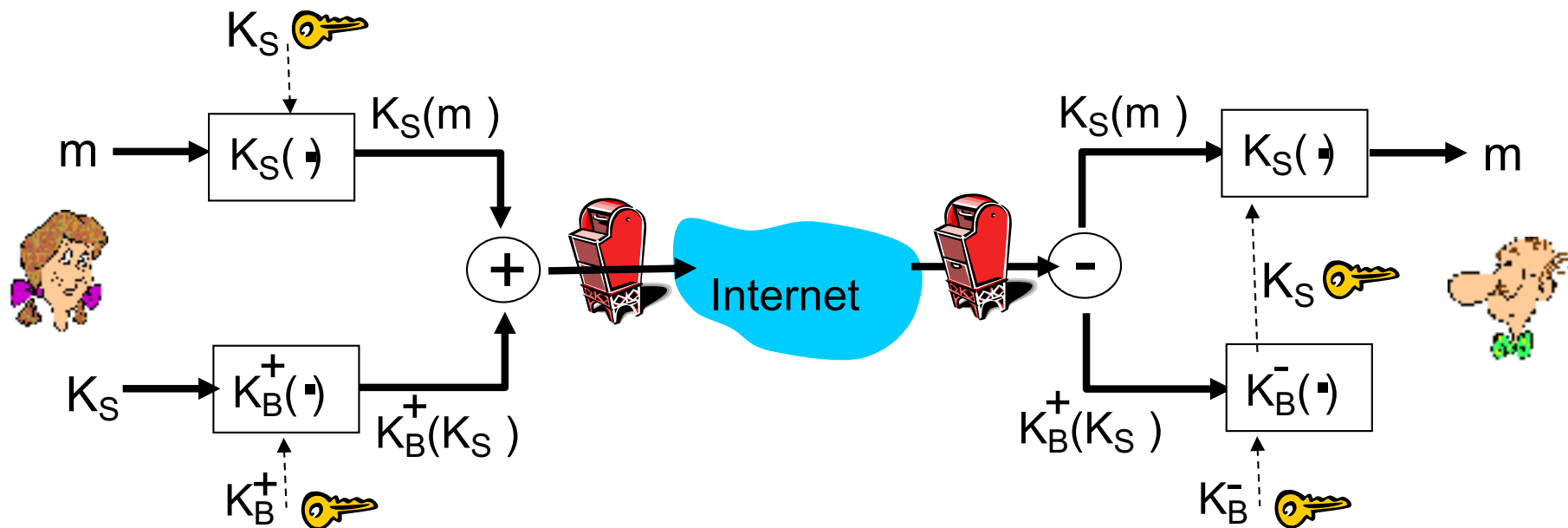
8.4 Authentication

8.5 Securing e-mail



# Secure e-mail

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

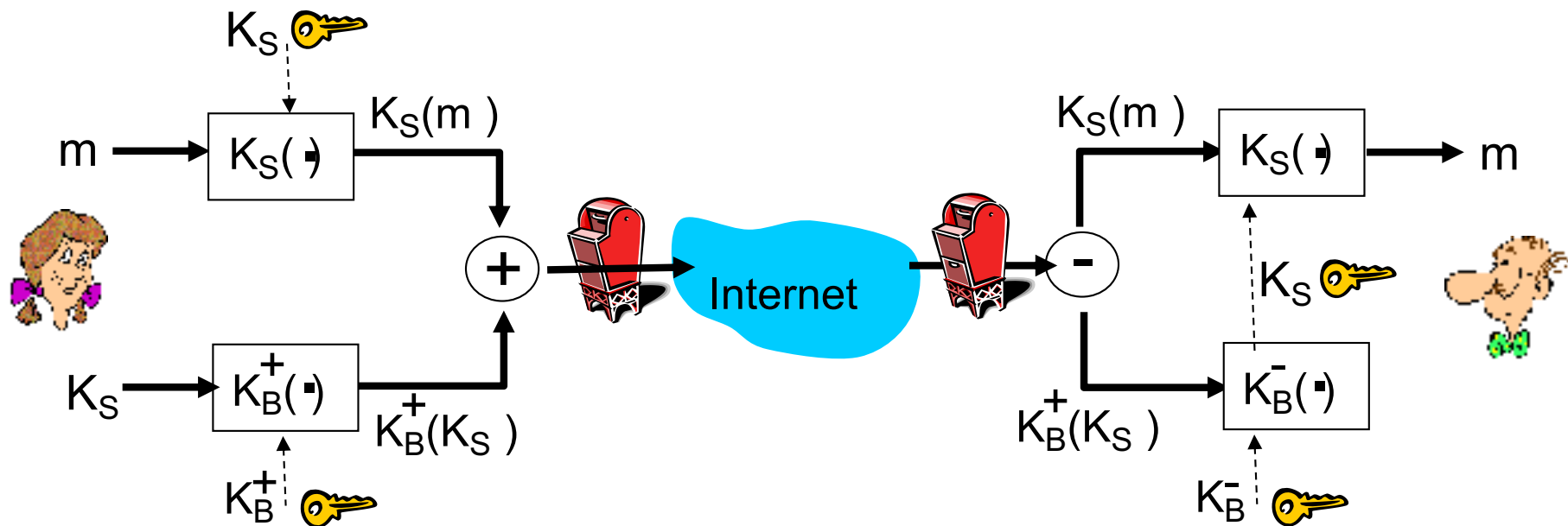


*Alice:*

- ❖ generates random *symmetric* session 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.

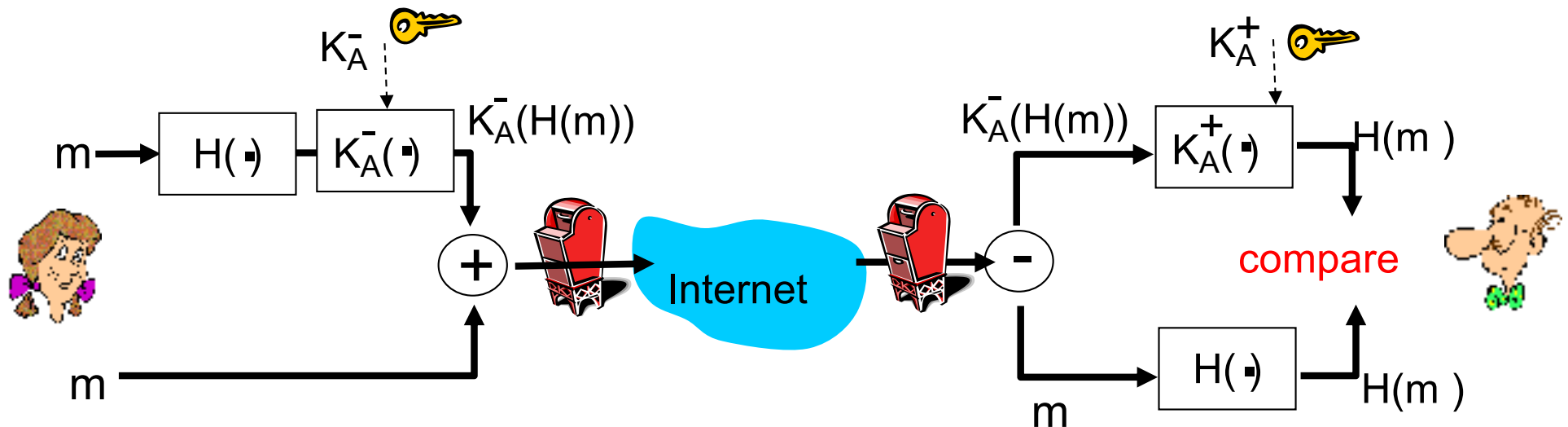


*Bob:*

- ❖ 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)

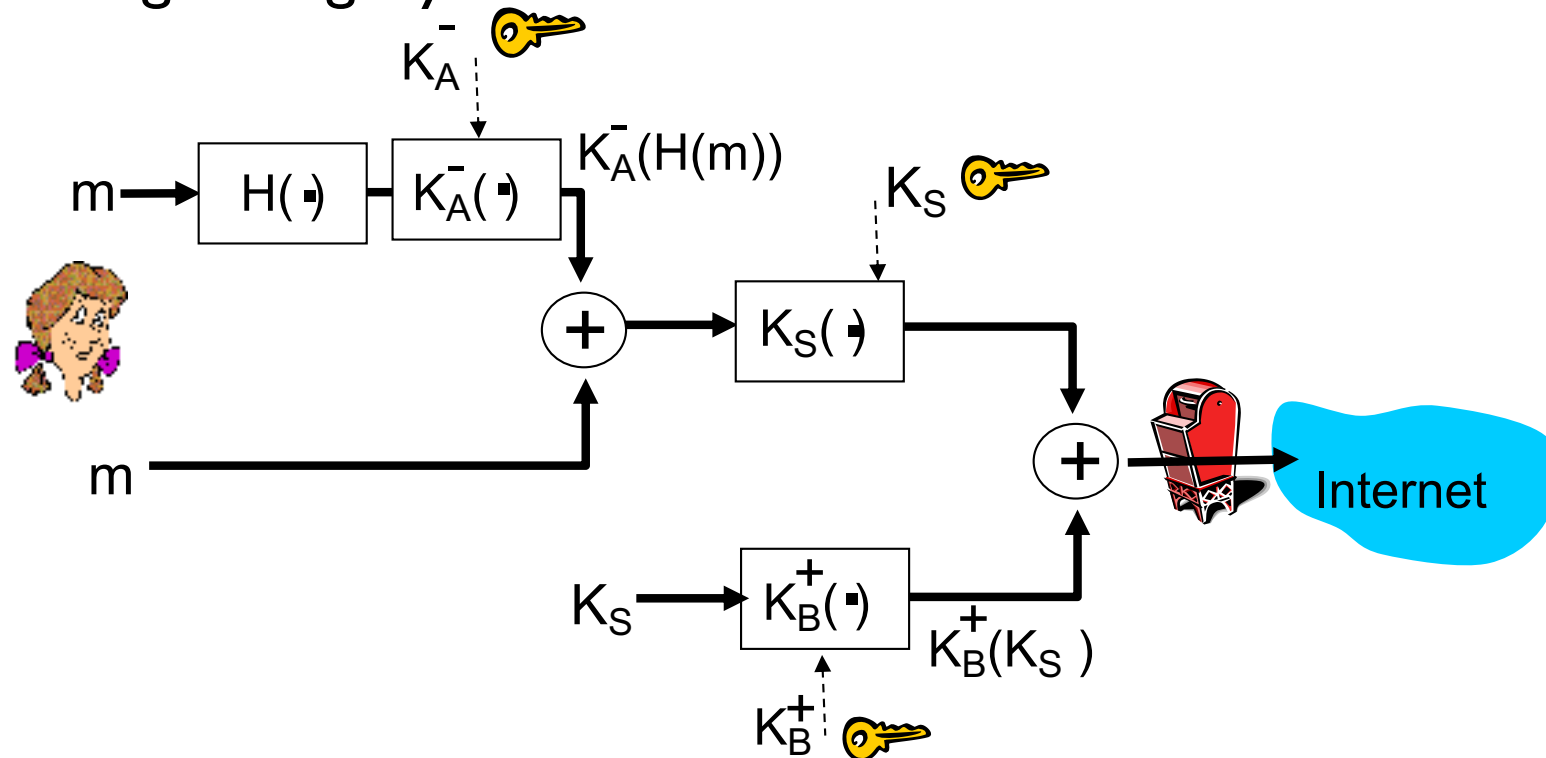
- ❖ Alice wants to provide sender authentication, message integrity



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

# Secure e-mail (continued)

- ❖ Alice wants to provide confidentiality, sender authentication, message integrity.



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

# Secure E-mail: PGP

- ❖ De-factor standard for email encryption
- ❖ On installation PGP creates public, private key pair
  - Public key posted on user's webpage or placed in a public key server
  - Private key protected by password
- ❖ Option to digitally sign the message, encrypt the message or both
- ❖ MD5 or SHA for message digest
- ❖ CAST, triple-DES or DEA for symmetric key encryption
- ❖ RSA for public key encryption

# Secure E-mail: PGP

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash:  SHA1
Bob:
Can I see you tonight?
Passionately yours, Alice
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset:  noconv
yhHJRhhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----
```

**Figure 8.22** ♦ A PGP signed message

```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=1KhnMikLo0+1/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp
-----END PGP MESSAGE
```

**Figure 8.23** ♦ A secret PGP message