

# **Lecture 12**

## An Introduction to Security

## 12.1 What is network security?

# What is network security?

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

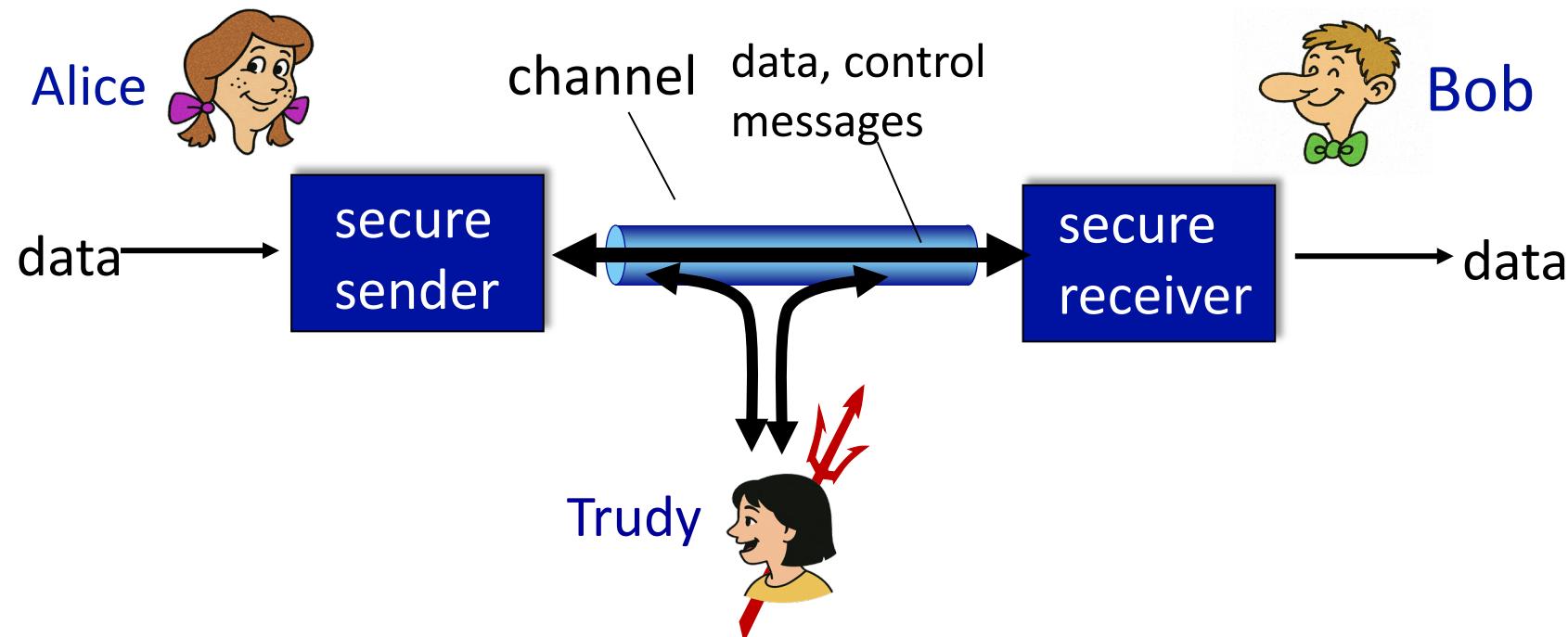
- sender encrypts message
- receiver decrypts message

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

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

# 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



# Friends and enemies: Alice, Bob, Trudy

Who might Bob and 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
- BGP routers exchanging routing table updates

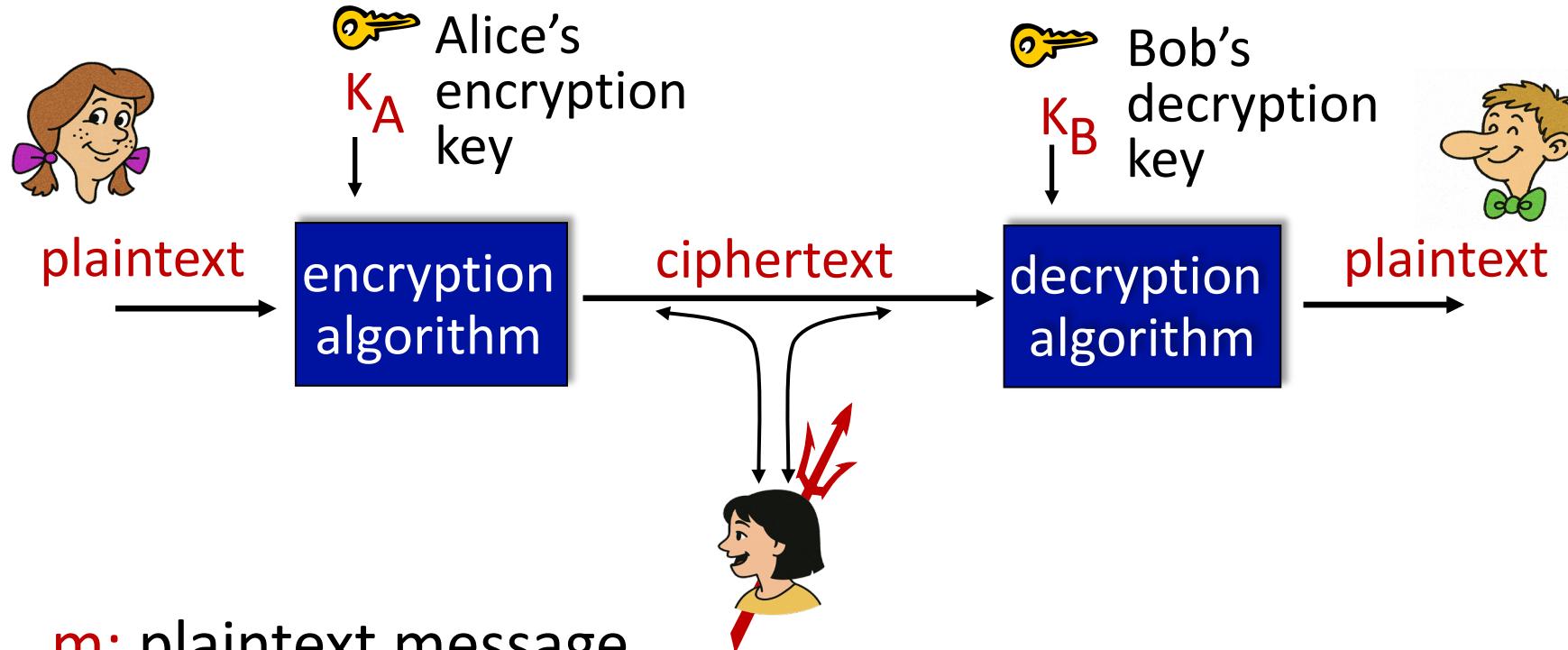
# There are bad guys (and girls) out there!

Q: What can a “bad guy” do?

A: A lot! (recall section 1.6)

- **eavesdrop**: intercept messages
- actively **insert** messages into connection
- **impersonation**: can fake (spoof) source address in packet (or any field in packet)
- **hijacking**: “take over” ongoing connection by removing sender or receiver, inserting himself in place
- **denial of service**: prevent service from being used by others (e.g., by overloading resources)

# The language of cryptography



$m$ : plaintext message

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

$m = K_B(K_A(m))$

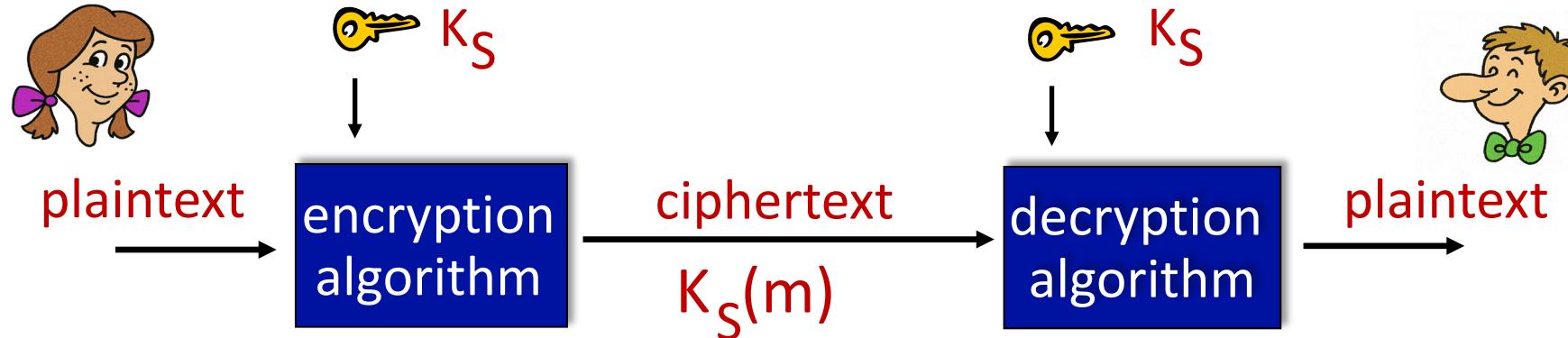
# Breaking an encryption scheme

- **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

## 12.2 Principles of cryptography

# Symmetric key cryptography



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

- 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

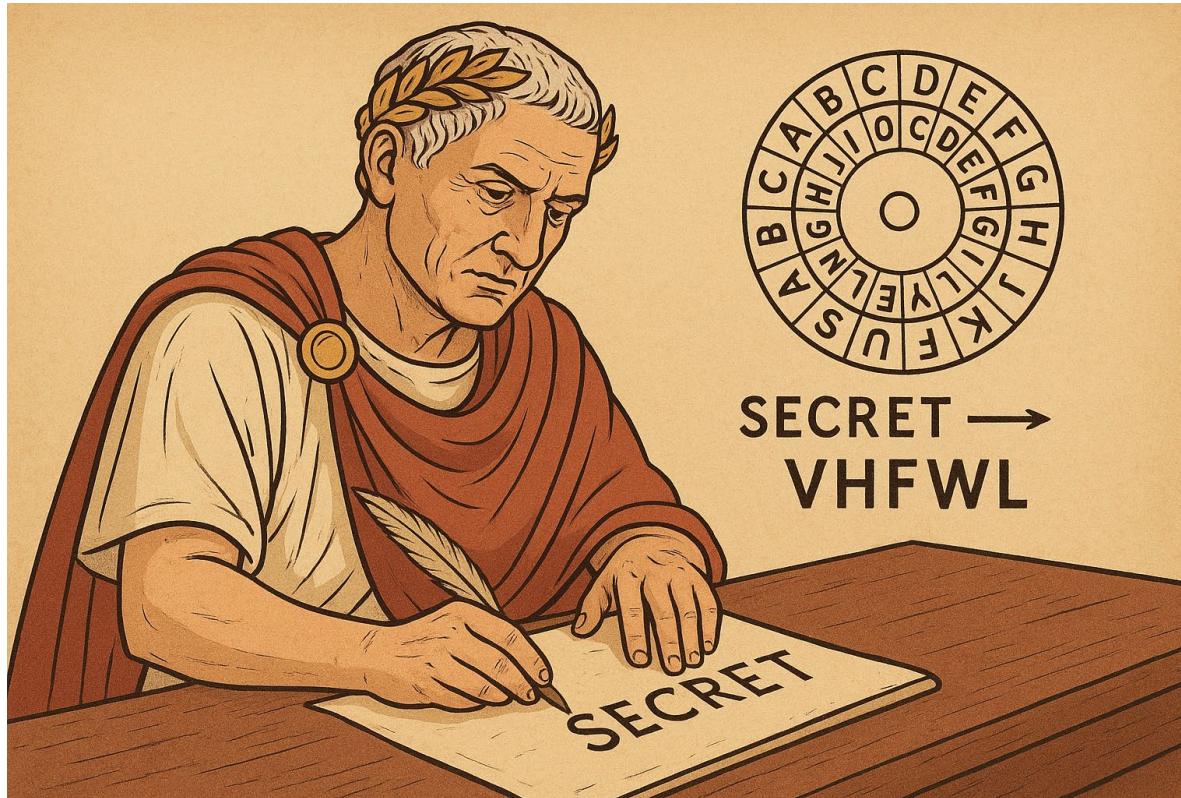
e.g.: Plaintext: bob. i love you. alice

ciphertext: nkn. s gktc wky. mgsbc



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

# Simple encryption scheme



Caesar has  
lost it!

🔑 *This method is commonly called the ceasar cipher!*

# A more sophisticated encryption approach

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

# Block Ciphers

- Message is broken into n-bit blocks
- Translation table created
  - All permutations of that sequence length

- Both knows the table
  - 3-bit table size = 8
  - Key space, 3-bit = 40,320
  - Key space, 64-bit =  $1,2688693219 \times 10^{89}$
  - 64-bit table size = 18,446,744,073,709,551,616

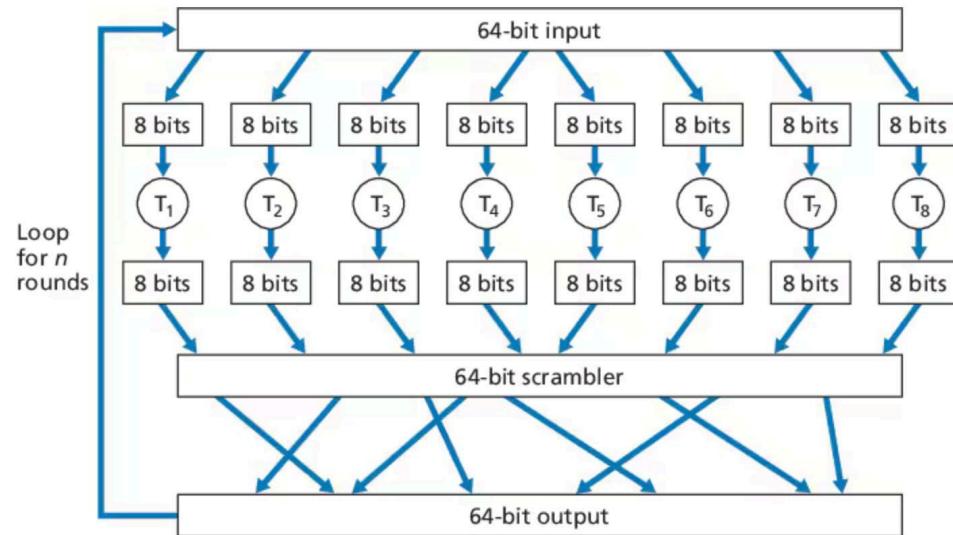
**Table 8.1** ♦ A specific 3-bit block cipher

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

# Block Ciphers

- Split into chunks of 8-bit
- 8-bit to 8-bit table (256 mappings)
- Scrambled
- Loops to affect all bits

Figure 8.5 ♦ An example of a block cipher



# Cipher-Block Chaining

- **Problem 1:** For identical blocks, a block cipher would, of course, produce the same ciphertext

- E.g. HTTP/1.1

- **Solution 1:** Introduce randomness

- XOR to randomize and (un)randomize (fast)

- **Problem 2:** Large transfer overhead!

- **Solution:** Initialization Vector and using the previous cipher as “random”

$$c(i) = K_S(m(i) \oplus r(i))$$

XOR

$c(1), r(1), c(2), r(2), c(3), r(3)$

$$c(i) = K_S(m(i) \oplus c(i - 1))$$

Previous  
cipher

# 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

# 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

## 12.3 Public Key Cryptography

# Public Key Cryptography

## symmetric key crypto:

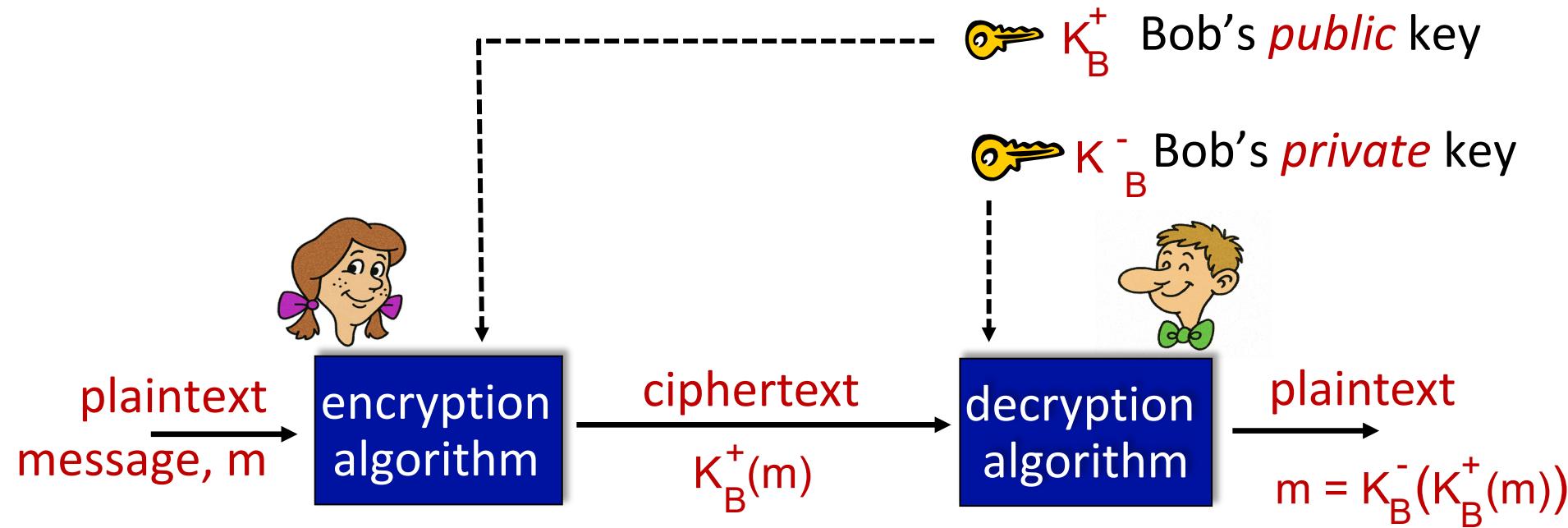
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never “met”)?

## 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 cryptography revolutionized 2000-year-old (previously only symmetric 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

# 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  $p, q$ .
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  $(n, d)$
1. to encrypt message  $m (< n)$ , compute

$$c = m^e \text{ mod } n$$

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

$$m = c^d \text{ mod } n$$

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

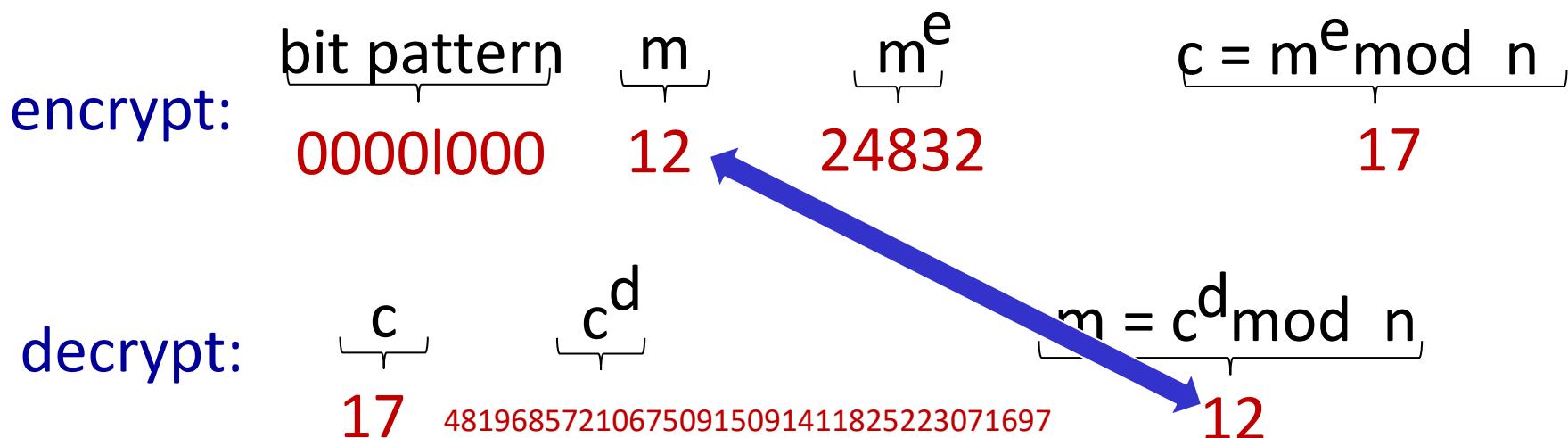
# RSA example:

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.



# Why is RSA secure?

- You know Bob's public key  $(n,e)$ . How hard is it to determine  $d$ ?
- Find factors of  $n$  without knowing the two factors  $p$  and  $q$ 
  - 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 session key  $K_s$
- once both have  $K_s$ , they use symmetric key cryptography

# Using Public-Key Cryptography

There are *lots* of non-confidentiality uses of public-key cryptography!

- symmetric-key agreement
- authentication
- digital signatures
- message integrity

## 12.4 Brief look into Security for the Internet

# Authentication

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap1.0:** Alice says “I am Alice”



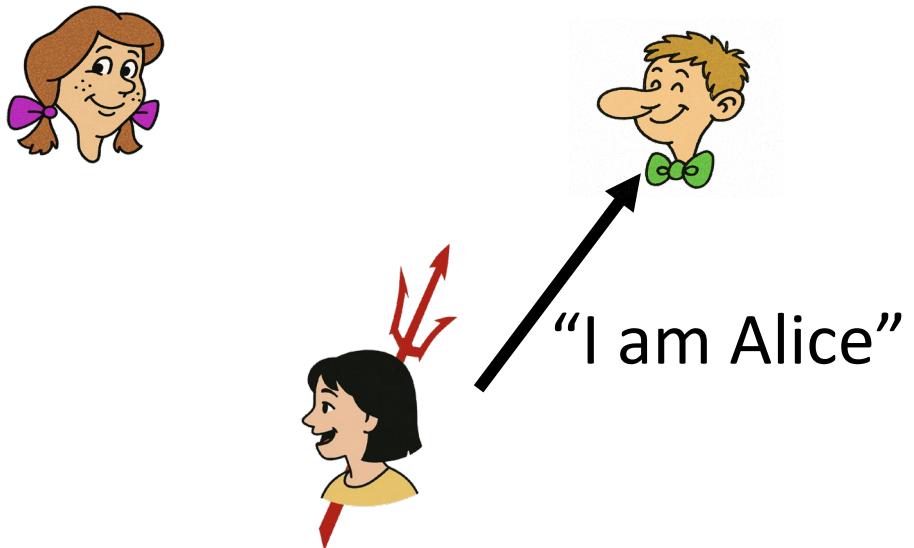
*failure scenario??*



# Authentication

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap1.0:** Alice says “I am Alice”



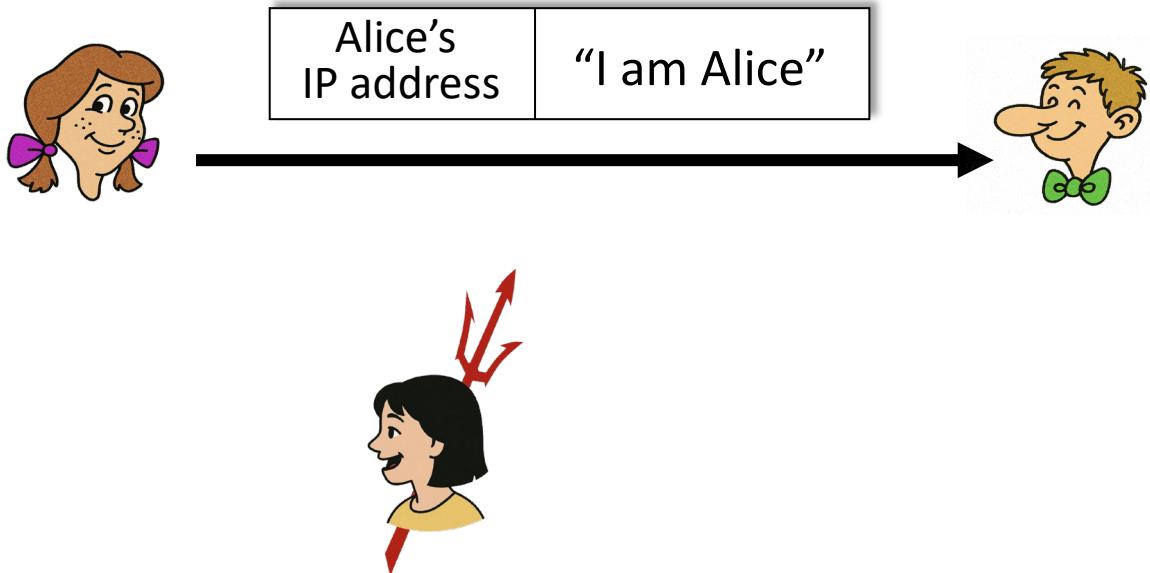
*in a network, Bob can not “see” Alice, so Trudy simply declares herself to be Alice*



# Authentication: another try

**Goal:** Bob wants Alice to “prove” her identity to him

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

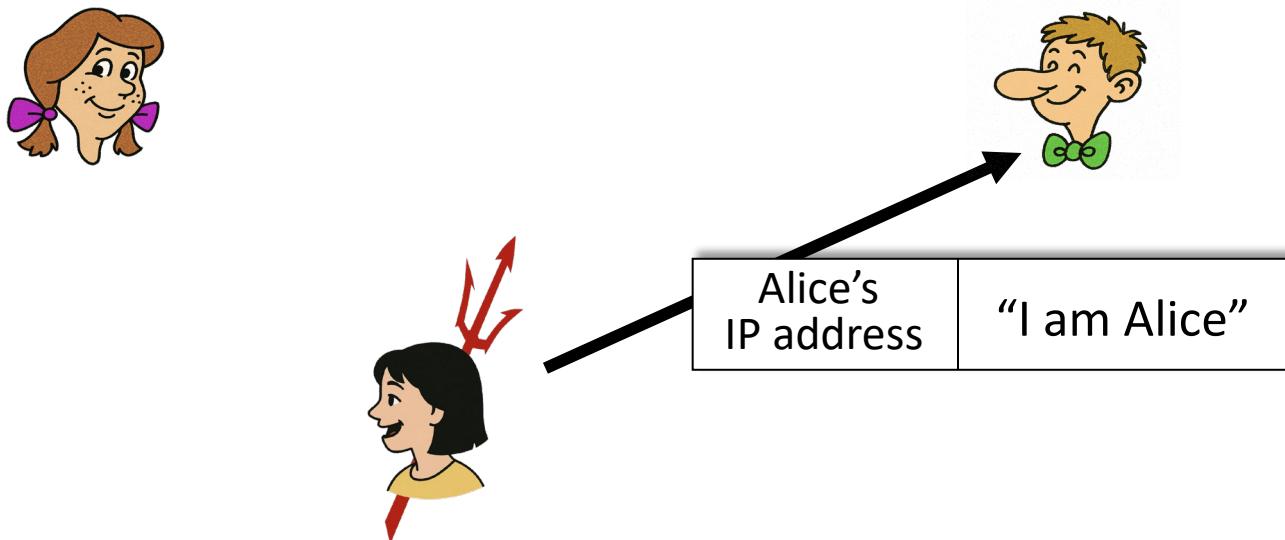


*failure scenario??*

# Authentication: another try

**Goal:** Bob wants Alice to “prove” her identity to him

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

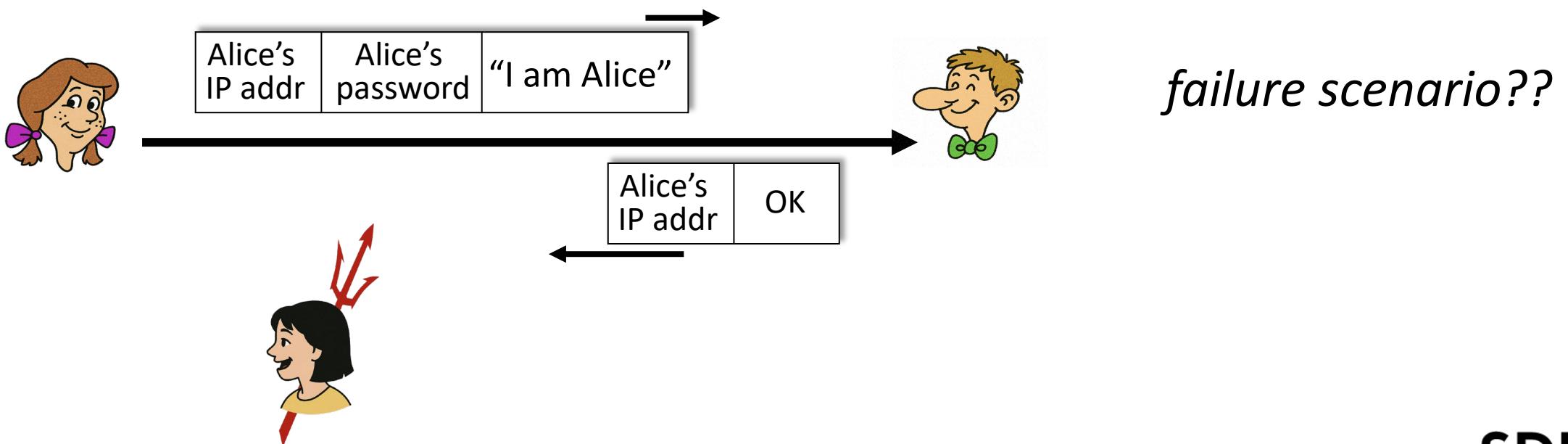


*Trudy can create  
a packet “spoofing”  
Alice’s address*

# Authentication: a third try

**Goal:** Bob wants Alice to “prove” her identity to him

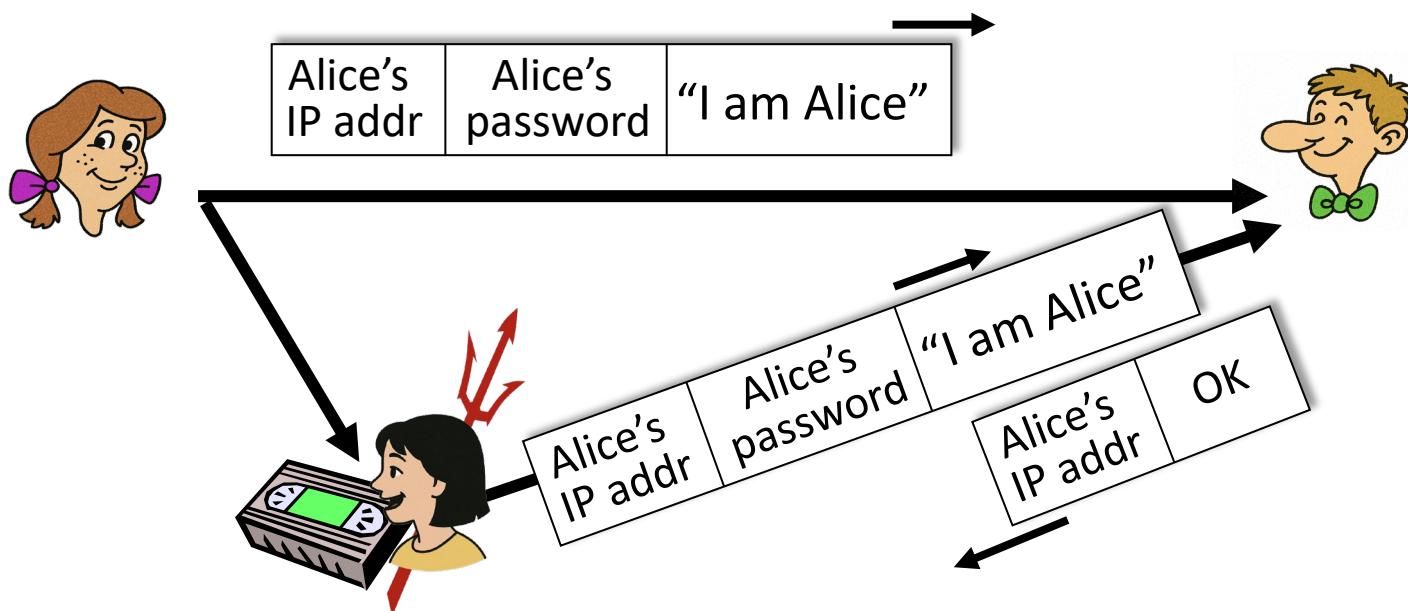
**Protocol ap3.0:** Alice says “I am Alice” and sends her secret password to “prove” it.



# Authentication: a third try

**Goal:** Bob wants Alice to “prove” her identity to him

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

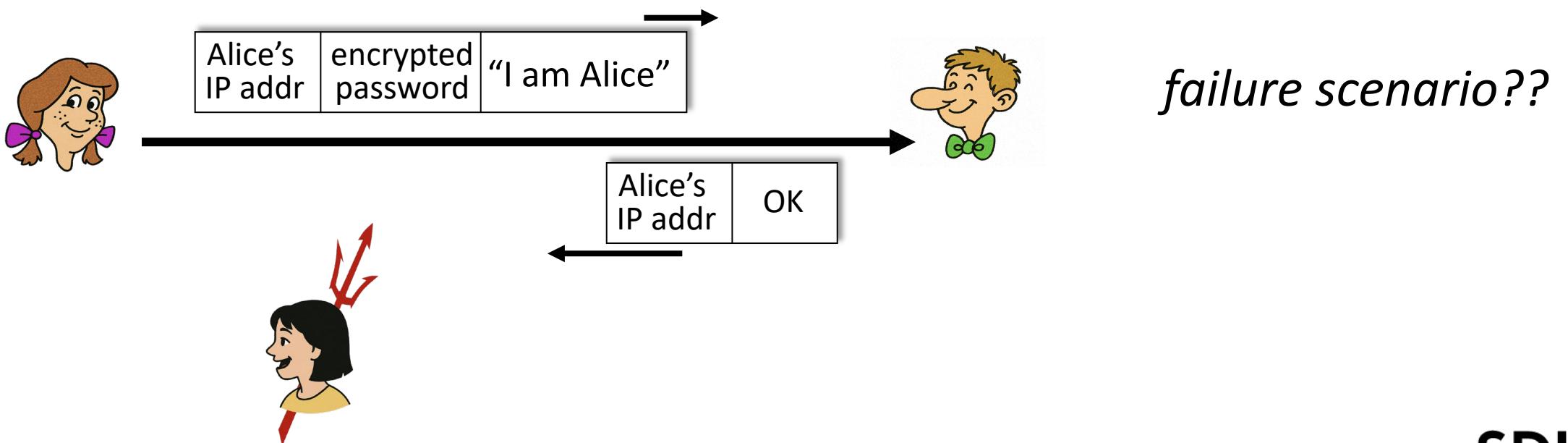


*playback attack:  
Trudy records  
Alice's packet  
and later  
plays it back to Bob*

# Authentication: a modified third try

**Goal:** Bob wants Alice to “prove” her identity to him

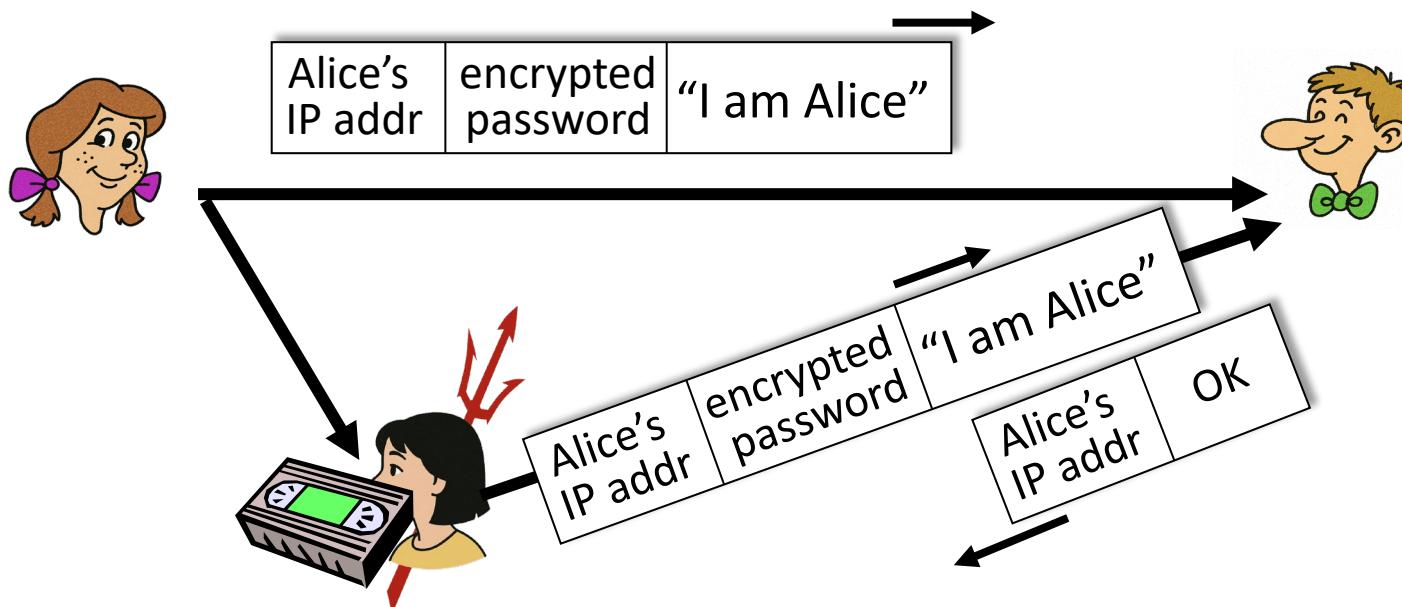
**Protocol ap3.0:** Alice says “I am Alice” and sends her encrypted secret password to “prove” it.



# Authentication: a modified third try

**Goal:** Bob wants Alice to “prove” her identity to him

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



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

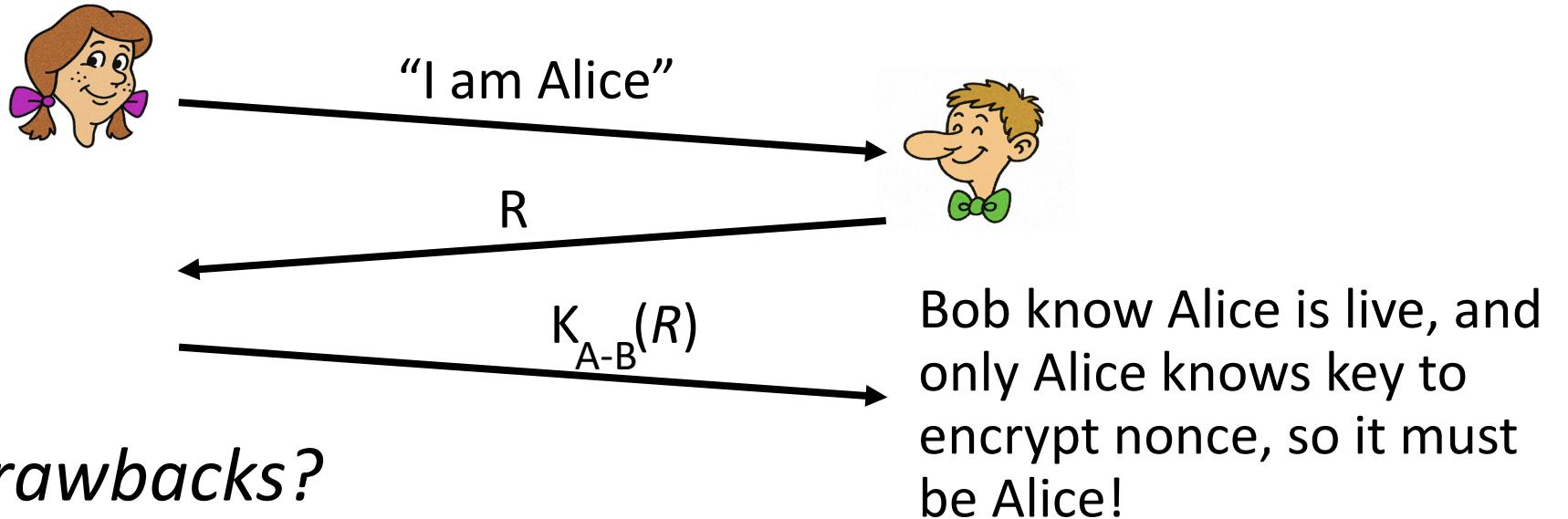
# Authentication: a fourth try

**Goal:** avoid playback attack

**nonce:** number ( $R$ ) used only **once-in-a-lifetime**

**protocol ap4.0:** to prove Alice “live”, Bob sends Alice *nonce*,  $R$

- Alice must return  $R$ , encrypted with shared secret key

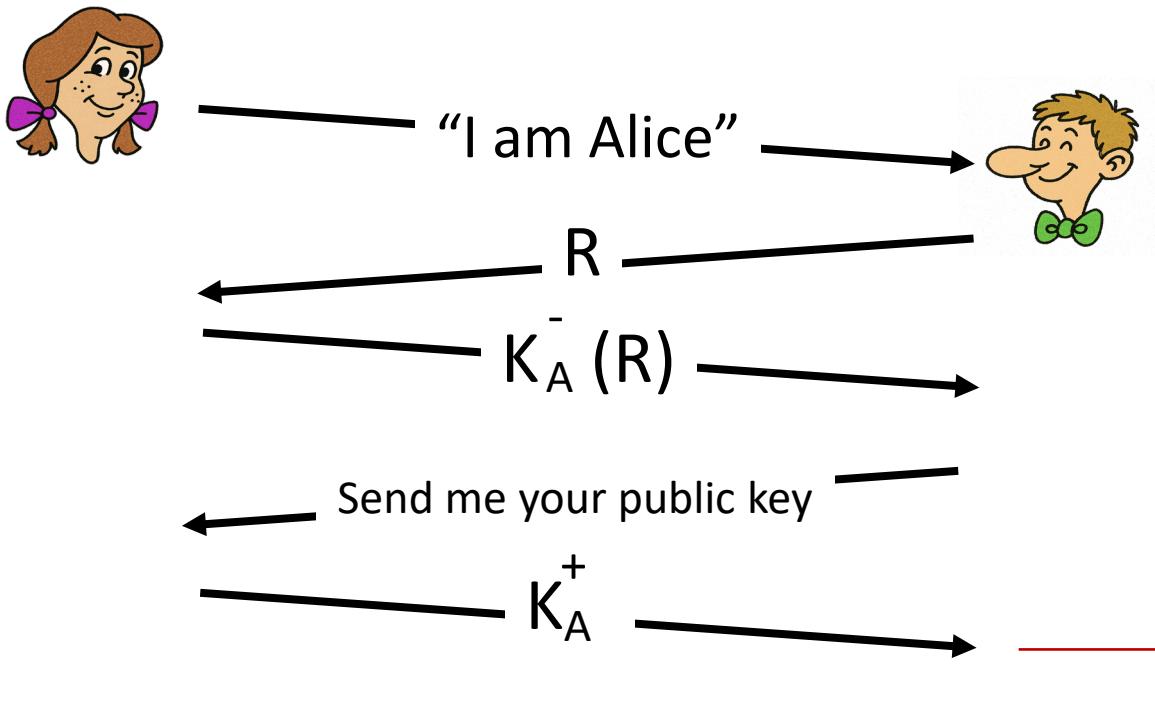


*Failures, drawbacks?*

# Authentication: ap5.0

ap4.0 requires shared symmetric key - can we authenticate using public key techniques?

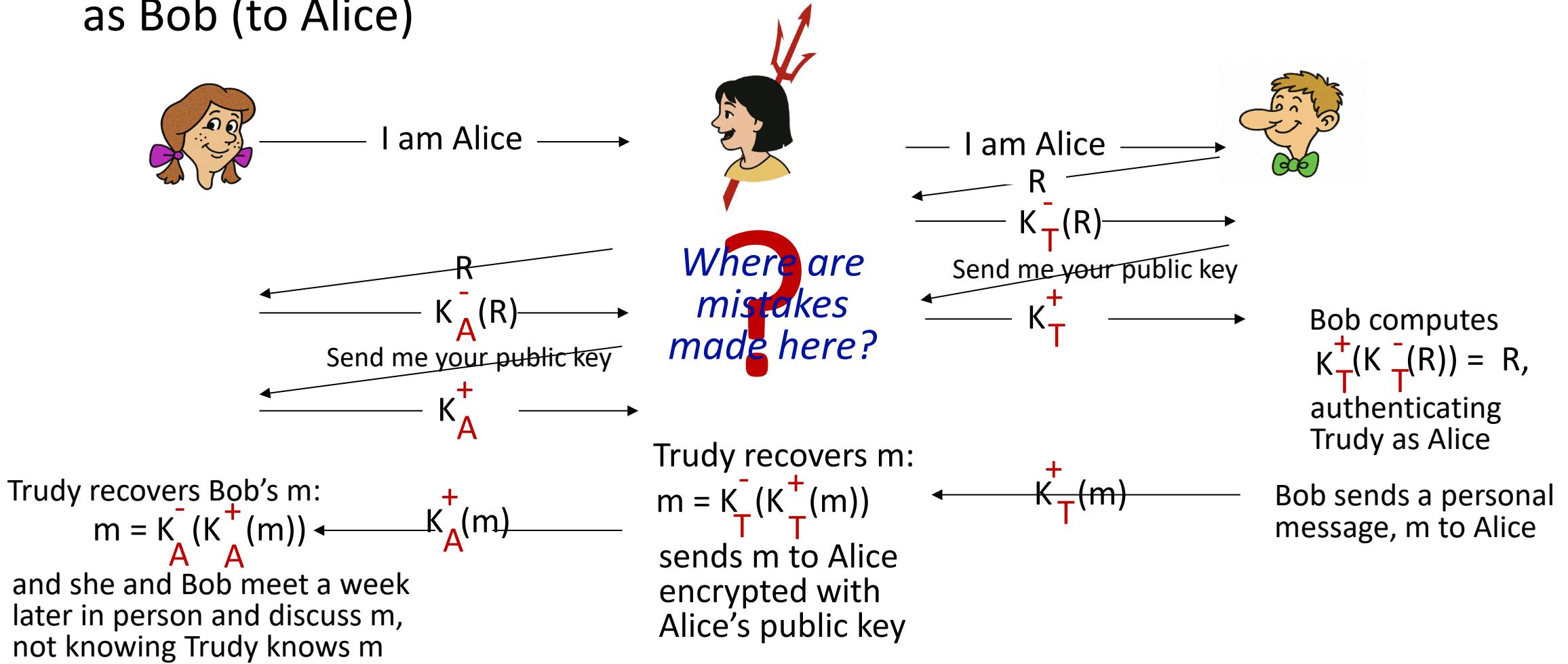
ap5.0: use nonce, public key cryptography



Bob computes  
 $K_A^+ (K_A^-(R)) = R$   
and knows only Alice could have the private key, that encrypted  $R$  such that  
 $K_A^+ (K_A^-(R)) = R$

# Authentication: ap5.0 – there's still a flaw!

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

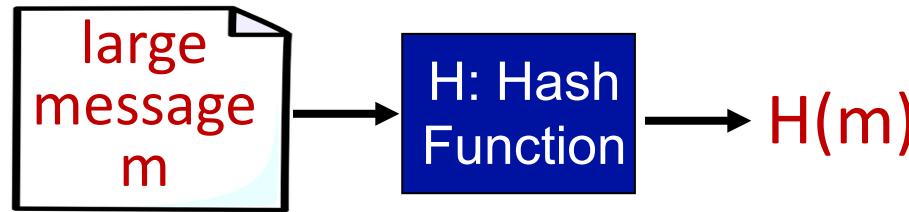


# Message digests

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 and produces fixed-size msg digest (fingerprint) that is appended to the message
- **With a shared secret key “as part of H” it can be used on the receiver side to hash and be compared**
- given message digest  $x$ , it must be 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>
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42

<u>message</u>	<u>ASCII format</u>
I O U 9	49 4F 55 <u>39</u>
0 0 . 1	30 30 2E <u>31</u>
9 B O B	39 42 D2 42

*different messages*

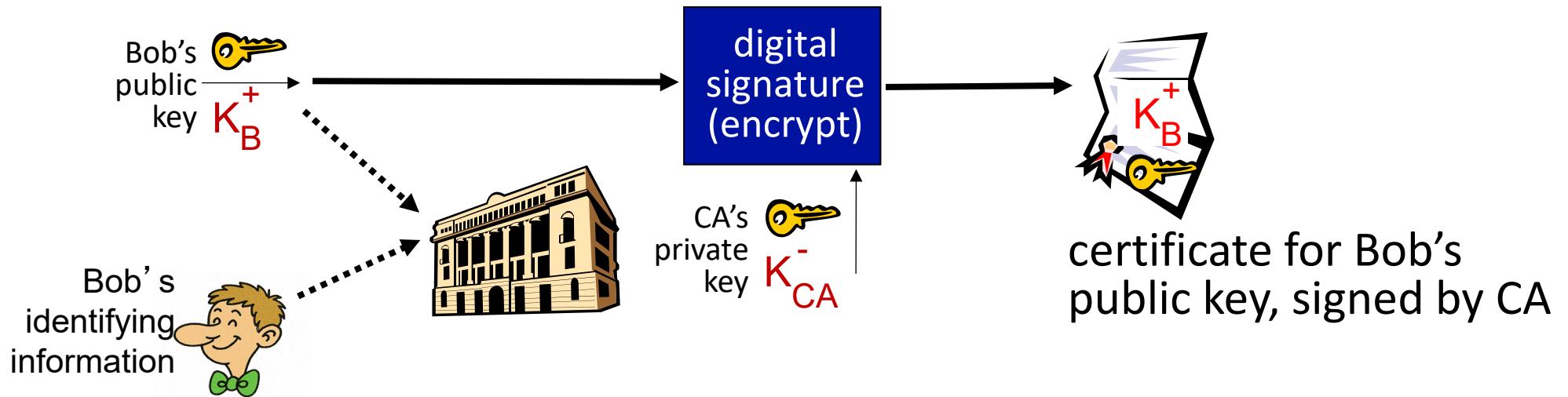
*but identical checksums!*

# Hash function algorithms

- In Use
  - SHA-2 (SHA-256), SHA-3
- Outdated
  - MD5, SHA-1
  
- **Hash functions provide:**
  - digital signature
  - message integrity

# Public key Certification Authorities (CA)

- certification authority (CA): binds public key to particular entity, E
- entity (person, website, router) registers its public key with CE provides “proof of identity” to CA
  - CA creates certificate binding identity E to E’s public key
  - certificate containing E’s public key digitally signed by CA: CA says “this is E’s public key”



# Transport-layer security (TLS)

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

} *all techniques we have studied!*

# That's it!

## Suggested exercise:

- Write a program that implements the Ceasar cipher or variations thereof.
- Write a 250 word essay on your favorite roman historical character and encrypt it using the cipher
- Send the encrypted essay to a classmate over a socket connection
- Force a classmate to break your cipher and read the essay
- Discuss why your roman character is better than theirs

- Fill out the course evaluation
- Discussion on suggestions for improvements
- Exam details
- Remember the mandatory hand-in!

Today we covered:

8.1-8.2 in detail

8.3-8.4 in brief

8.5 on TLS was just touched

# Exam Details

- Language: English or Danish
- Structure
  - Pick a random topic on arrival
  - Presentation on that topic, 5-7 min
    - Visual aid recommended (e.g. Power Point)
  - Discussion, 10-12 min
    - Questions to the entire curriculum
  - Evaluation and grade, 3min
- Besides the presentation material no notes are allowed at the exam
- Remember to bring your study card for identification
- Final exam schedule will be reported before 24/12.
- Topics:
  - Application Layer
  - Transport Layer
  - Network Layer
  - Data Link Layer
- Expected presentation content: Provide an overall description of the layer's role and the important technologies/protocols used in the layer.