# Network Security

Objectives: What security properties are commonly desired? What can bad guys do to jeopardize the properties? Understand crypto techniques for defense: symmetric/asymmetric keys, confidentiality, authentication, integrity; authentication protocol and design issues; securing emails.

NS3: March 19, 2018

Textbook (K&R): Sections 8.1-8.5

# Security properties we care about

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

- sender encrypts message
- receiver decrypts message

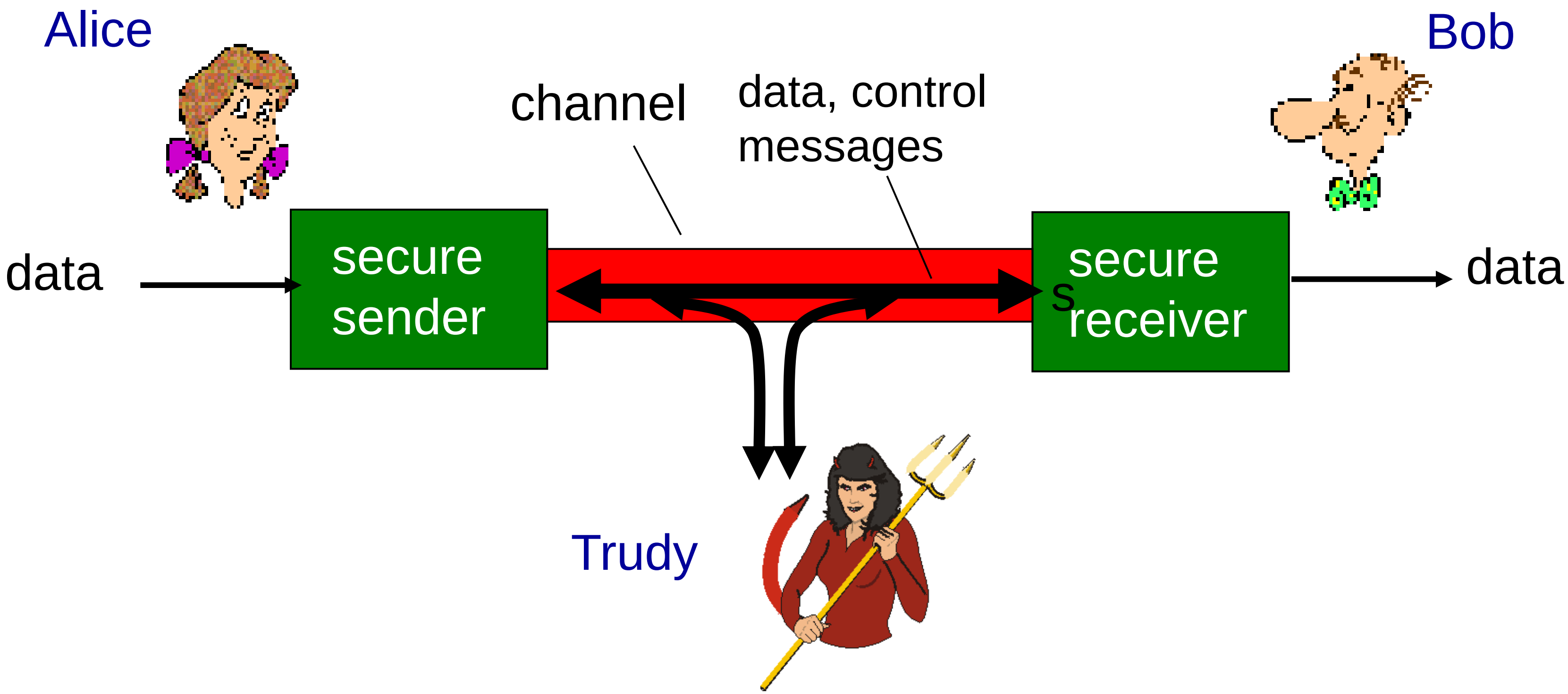
*authentication:* sender, receiver want to confirm identity of each other

*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?

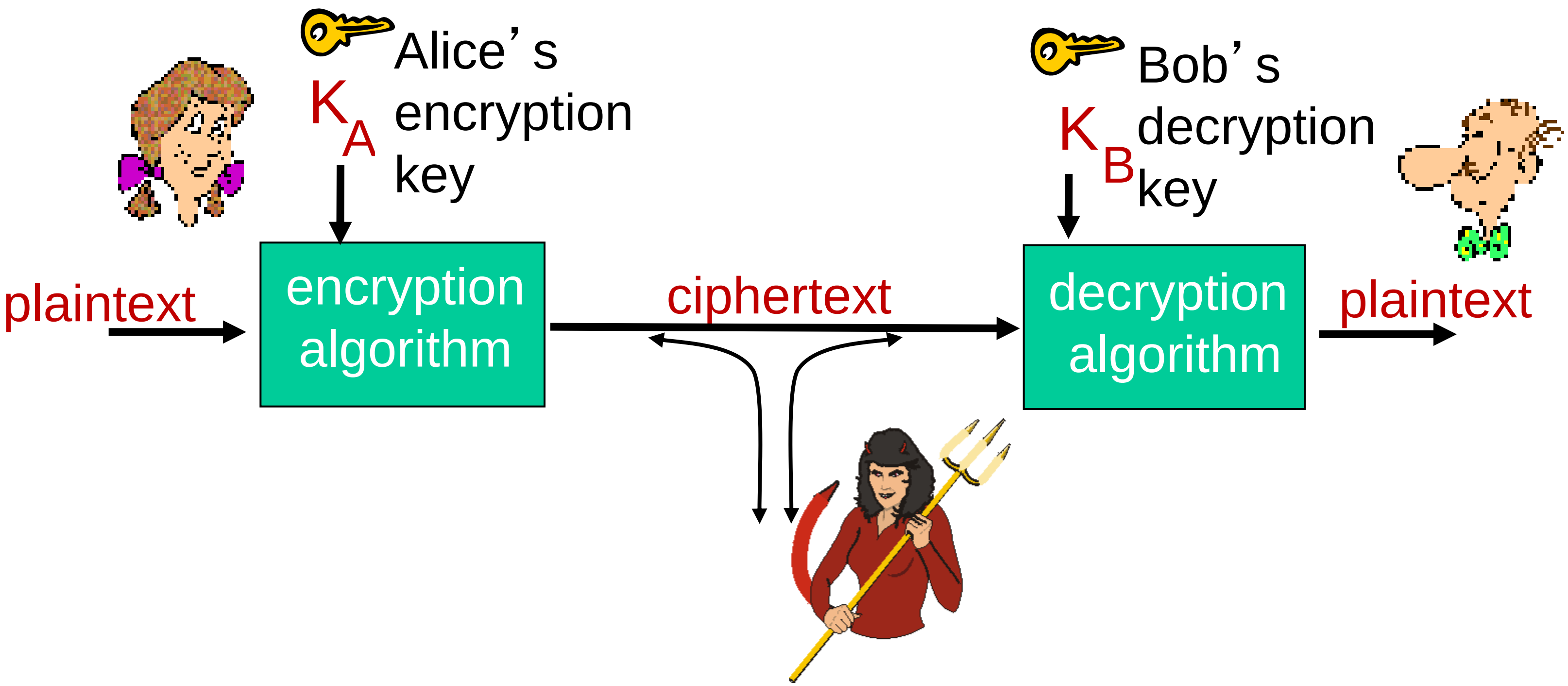
# What can bad guys (and girls) do?

*Q:* What can a "bad guy" do?

*A:* A lot! See section 1.6

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

# The language of cryptography



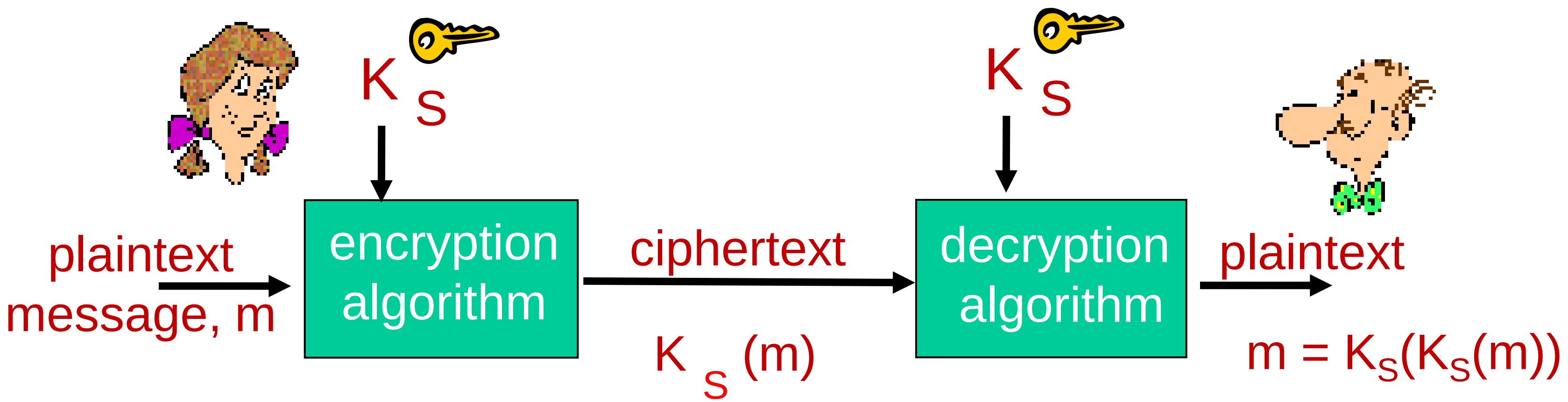

m plaintext message

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

$m = K_B(K_A(m))$

Need *one-way* function: from $K_A(m)$, can't tell what m is.

# 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? (Not easy question, since key must be agreed on in secret.)

# 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

What is the *size* of the key? What if you know the message contains the word bob somewhere?

# Symmetric key crypto: DES

## DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
  - Padding applied if needed
- block cipher with cipher block chaining
  - You will learn importance of block chaining in NS Lab 2
- how secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
  - no known good analytic attack
- making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

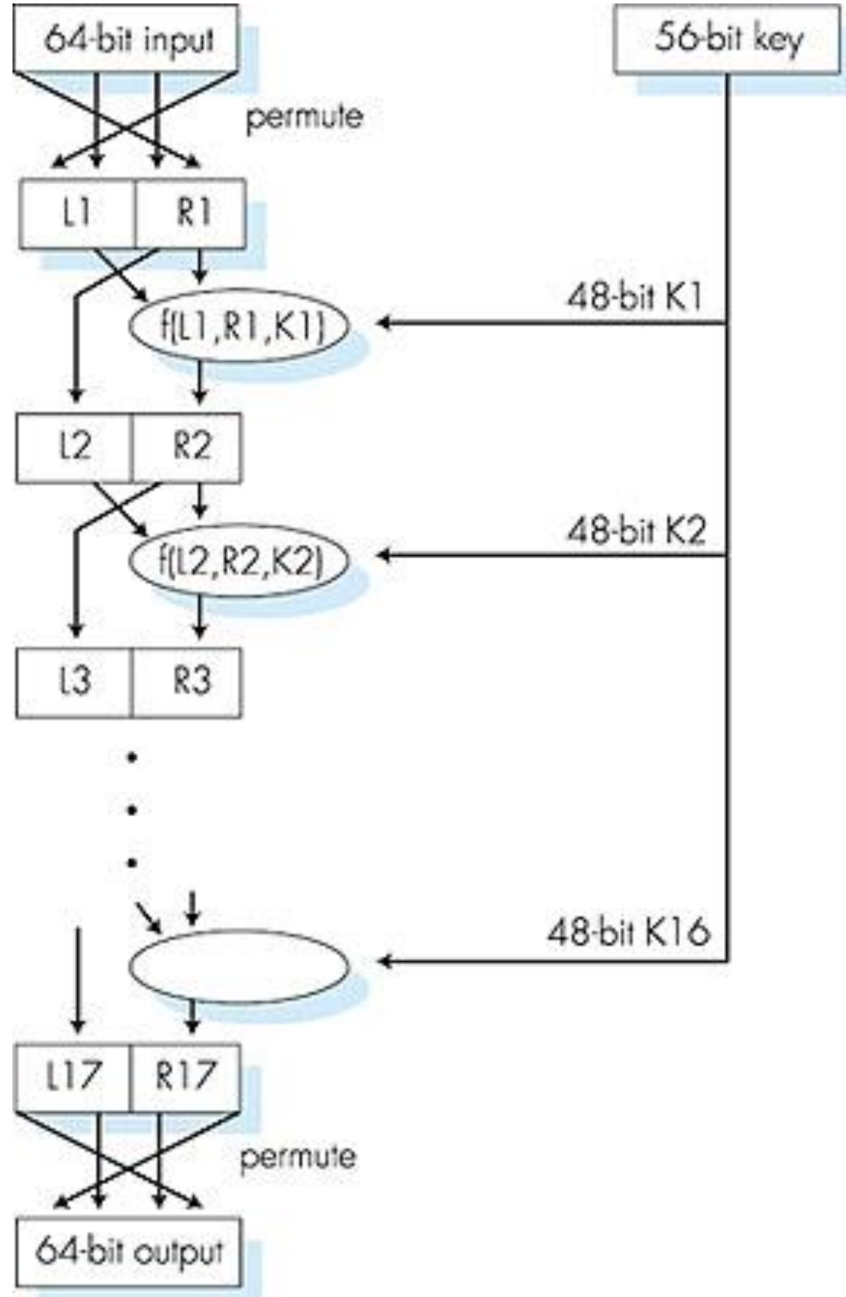
# Symmetric key crypto: DES

## *DES operation*

initial permutation

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

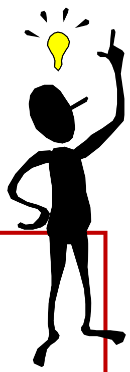
final permutation



# AES: Advanced Encryption Standard

- symmetric-key NIST standard, replaced DES (Nov 2001)
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

# Public Key Cryptography



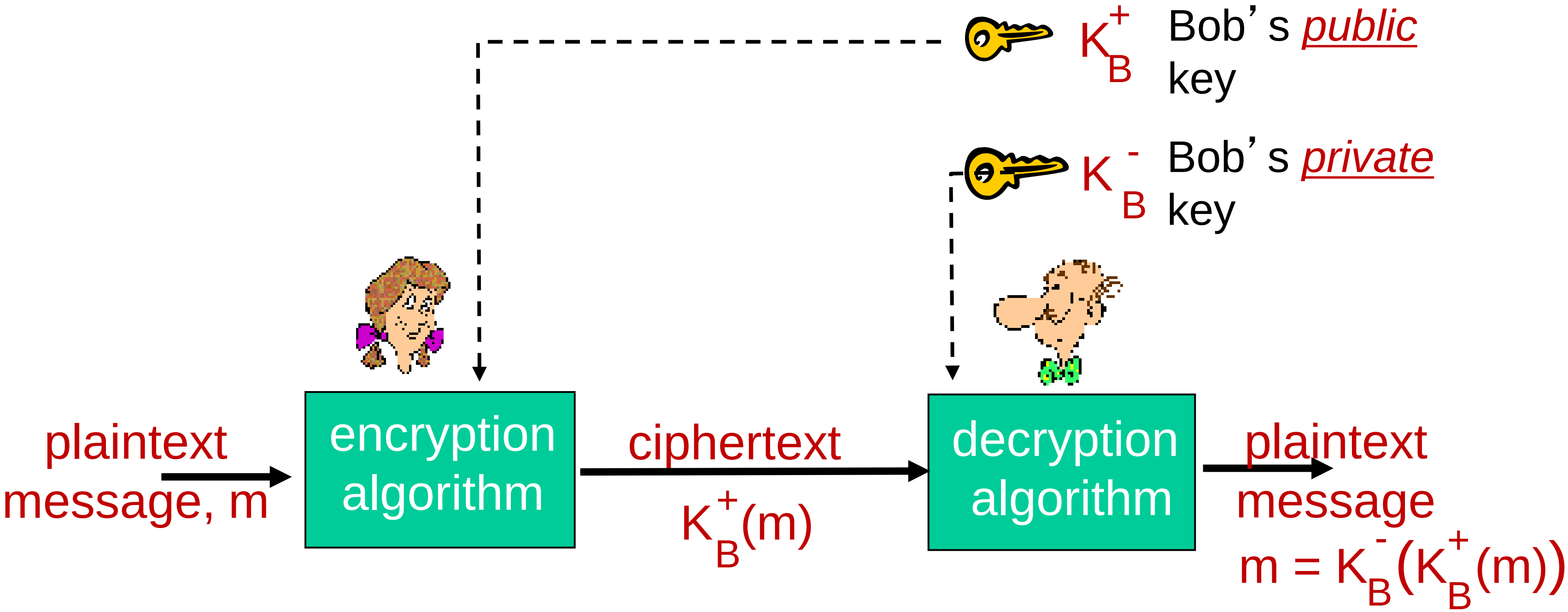
*symmetric key crypto*

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

*public key crypto*

- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver
- also called *asymmetric* key crypto

# 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 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: x=14, n=10, d=2:

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

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

# RSA: getting ready

- message: just a bit pattern
- bit pattern can be uniquely represented by an integer number
- thus, encrypting a message is equivalent to encrypting a number.

*example:*

- m= 10010001 . This message is uniquely represented by the decimal number 145.
- to encrypt m, we encrypt the corresponding number, which gives a new number (the ciphertext).

# RSA: Creating public/private key pair

1. choose two large prime numbers $p, q$. (e.g., 1024 bits each)
2. compute $n = pq$, $z = (p-1)(q-1)$
3. choose e (with e<$n$) that has no common factors with z (e, $z$ are "relatively prime").
4. choose $d$ such that $ed$-1 is exactly divisible by $z$. (in other words: $ed$ mod $z$ = 1).
5. *public* key is $\underbrace{(n,e)}_{K_B^+}$. *private* key is $\underbrace{(n,d)}_{K_B^-}$.

# Activity 3.1

- $p = 3$; $q = 7$
- What is n? What is z?
- Can we then choose $e = 5$ and $d = 5$?
- Even if we can pick the suitable d and e, surely the above p and q don't really work. Why?

# 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:

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

*result is the same!*

# Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$ ?

follows directly from modular arithmetic:

$$
\begin{aligned}
(m^e \bmod n)^d \bmod n &= m^{ed} \bmod n \\
&= m^{de} \bmod n \\
&= (m^d \bmod n)^e \bmod n
\end{aligned}
$$

# Why is RSA secure?

- suppose you know Bob's public key (n,e). How hard is it to determine d?
  - Private key is (n,d). n is known (from public key). d is related to e (ed - 1 divisible by z).
- essentially need to find factors of n without knowing the two factors p and q
  - If you knew p and q, you could easily compute z. Given e and z, you could easily compute d.
  - Luckily, fact: factoring a big number is hard (no one has figured out how to do it yet)

# RSA in practice: session keys

- exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- use public key cryto 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 $K_S$ at the beginning of a session
- once both have $K_S$, they use symmetric key cryptography for (possibly lots of) subsequent communications throughout the session

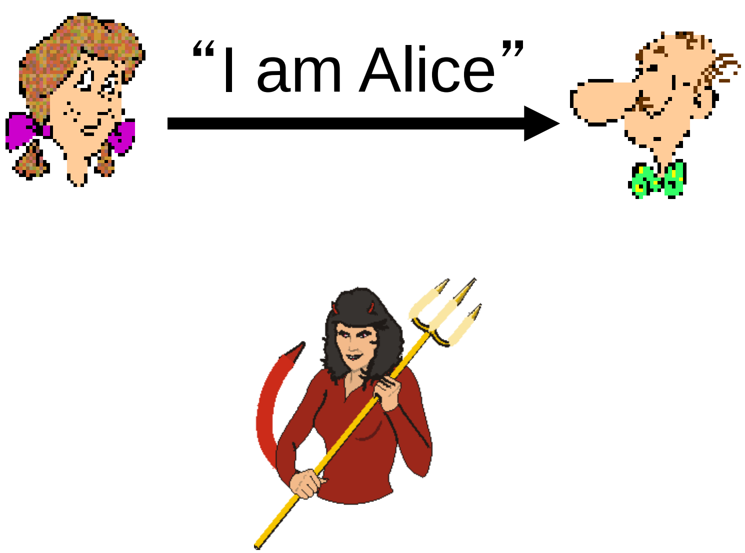
# Activity 3.2

Design a (simple) method for Alice and Bob to use RSA to agree on a secret symmetric key for encrypting communication in a session. (Assume that Alice and Bob know each other's public keys a priori.)

# Authentication

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

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

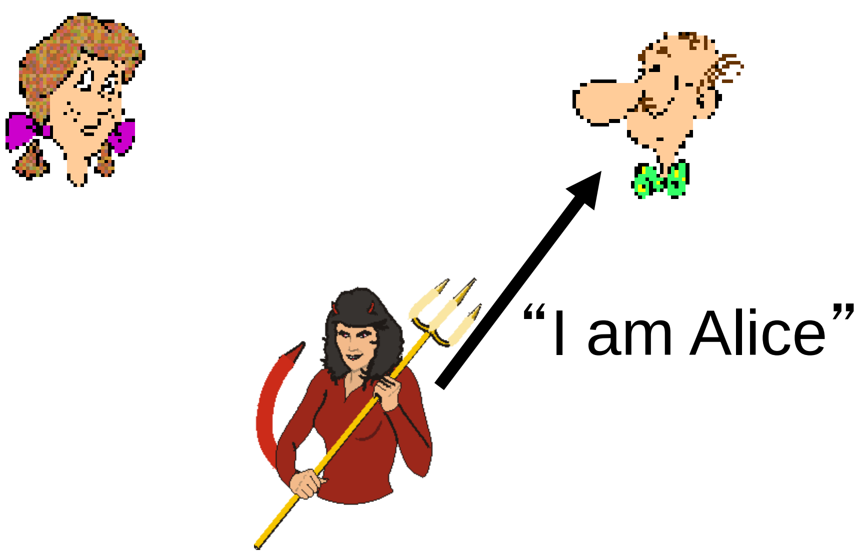



Failure scenario??

# Authentication

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

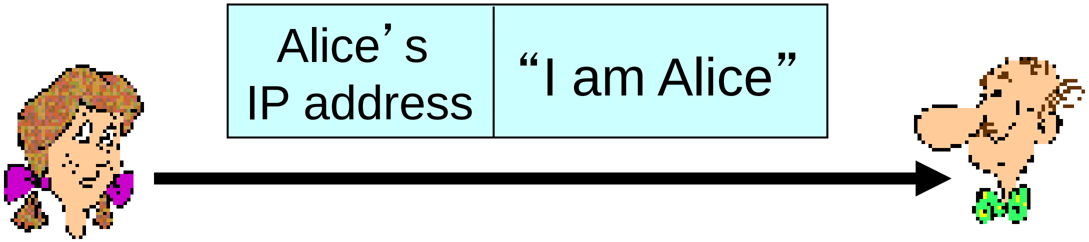
*Protocol ap1.0:* Alice says "I am Alice"




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

# Authentication: another try

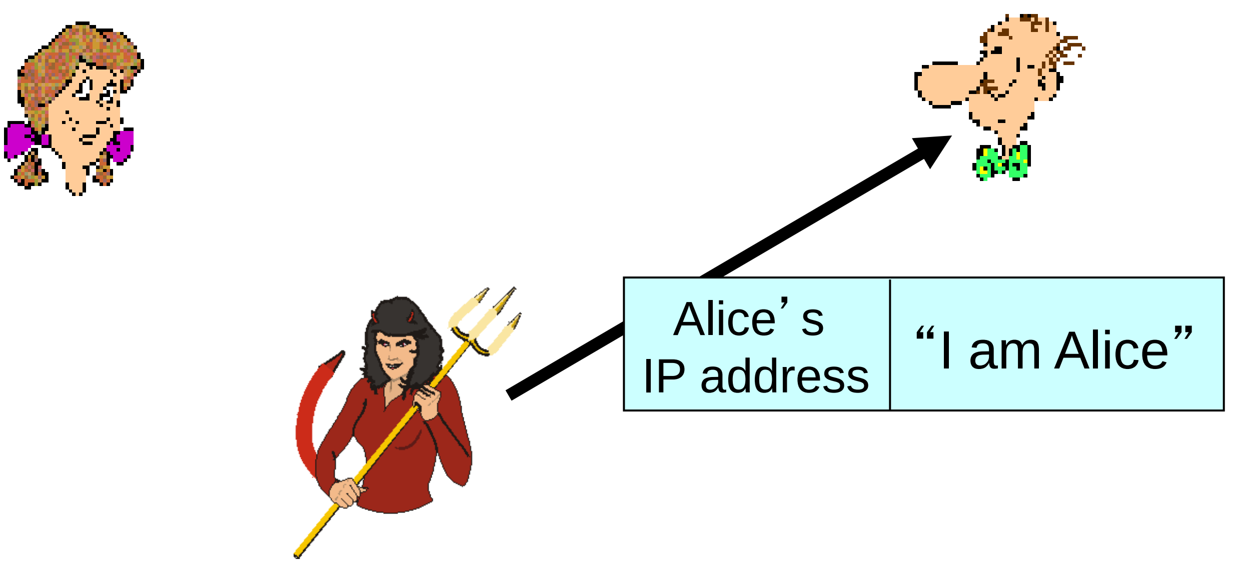
*Protocol ap2.0:* Alice says "I am Alice" in an IP packet containing her source IP address




Failure scenario??

# Authentication: another try

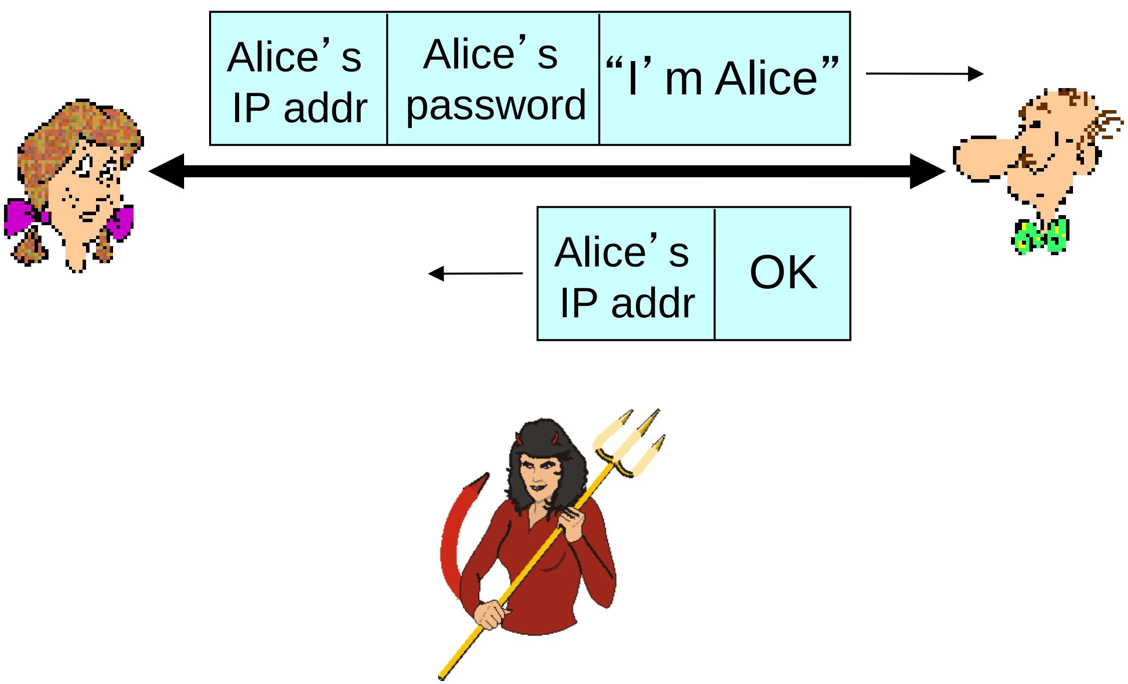
*Protocol ap2.0:* Alice says “I am Alice” in an IP packet containing her source IP address




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

# Authentication: another try

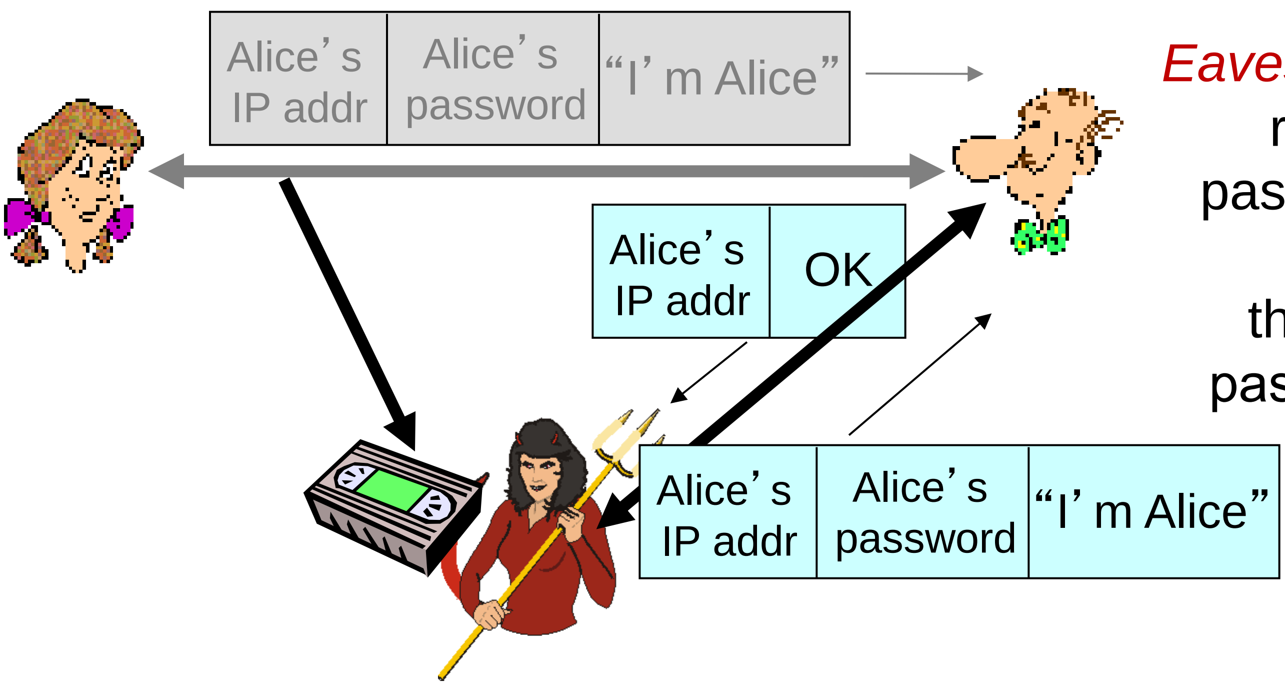
*Protocol ap3.0:* Alice says "I am Alice" and sends her secret password to "prove" it.




Failure scenario??

# Authentication: another try

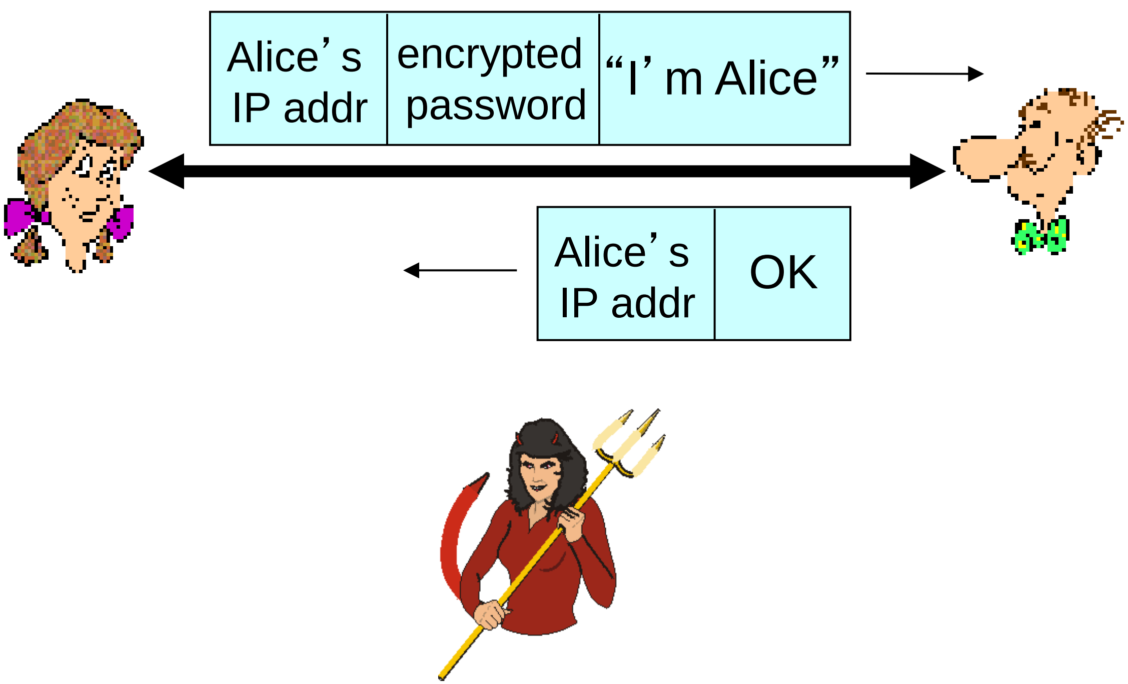
*Protocol ap3.0:* Alice says "I am Alice" and sends her secret password to "prove" it.




*Eavesdrop attack:* Trudy records Alice's password (or packet) and sends the same to Bob: password isn't really secret!

# Authentication: yet another try

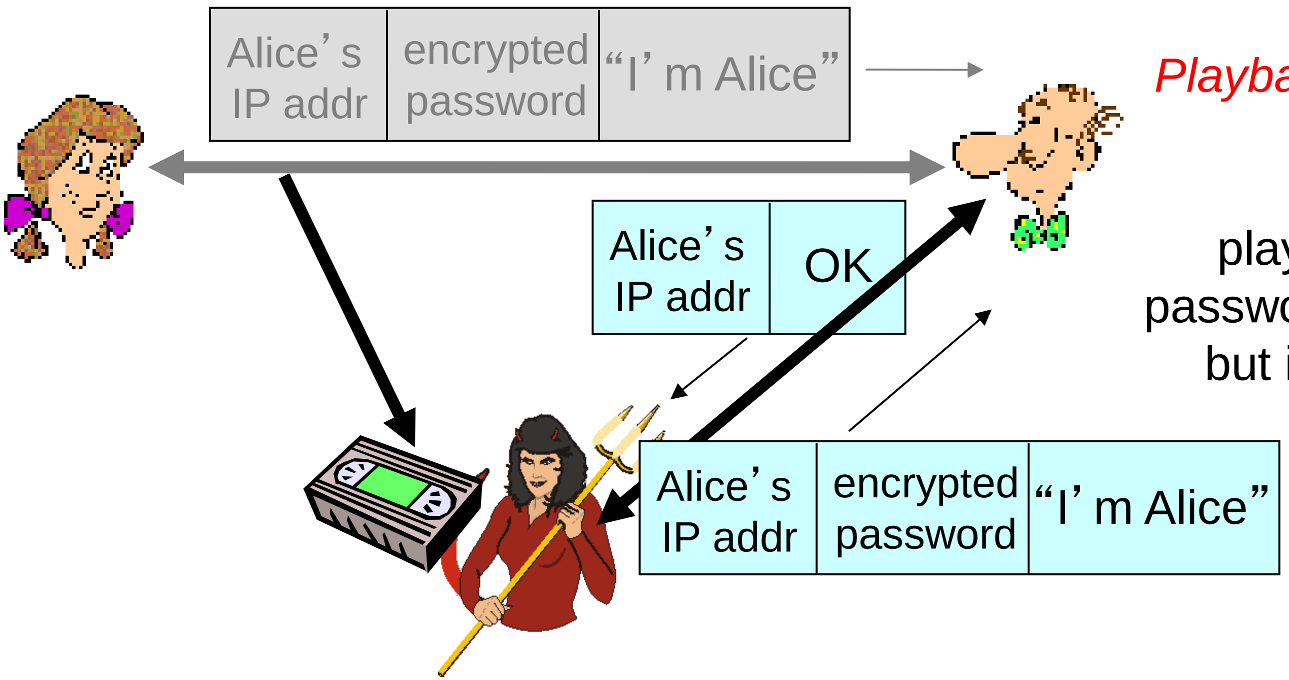
*Protocol ap3.1:* Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.




Failure scenario??

# Authentication: yet another try

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




*Playback (or replay) attack*: Trudy records packet and plays it back to Bob – password is now truly secret but it still doesn't work!

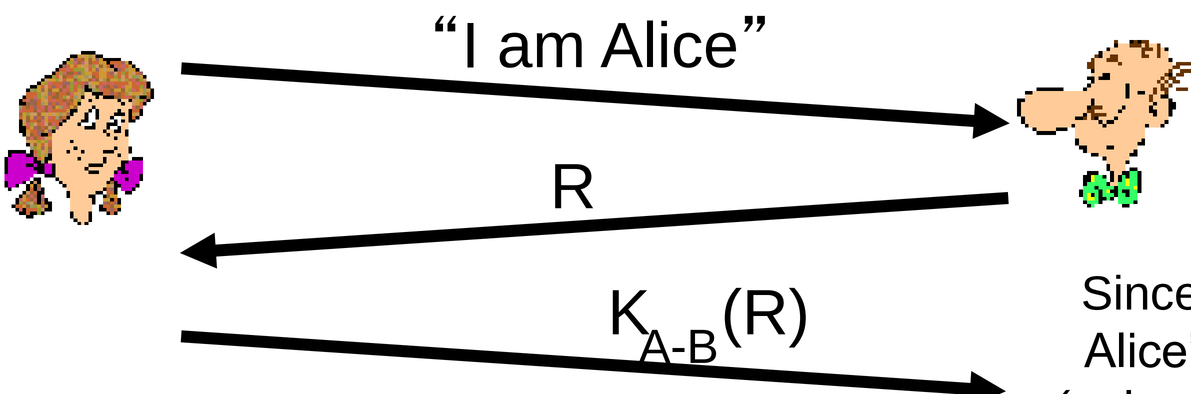
# Authentication: yet another try

*Goal:* avoid playback attack

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

(in practice, nonce can be a large random number)

*ap4.0:* to prove Alice "live", Bob sends Alice *nonce*, R. Alice must return R, encrypted with shared secret key




Since nonce is fresh, Alice's response to it (only she knows how to generate response) must be fresh, so we're talking to a live Alice (not say yesterday's version)!

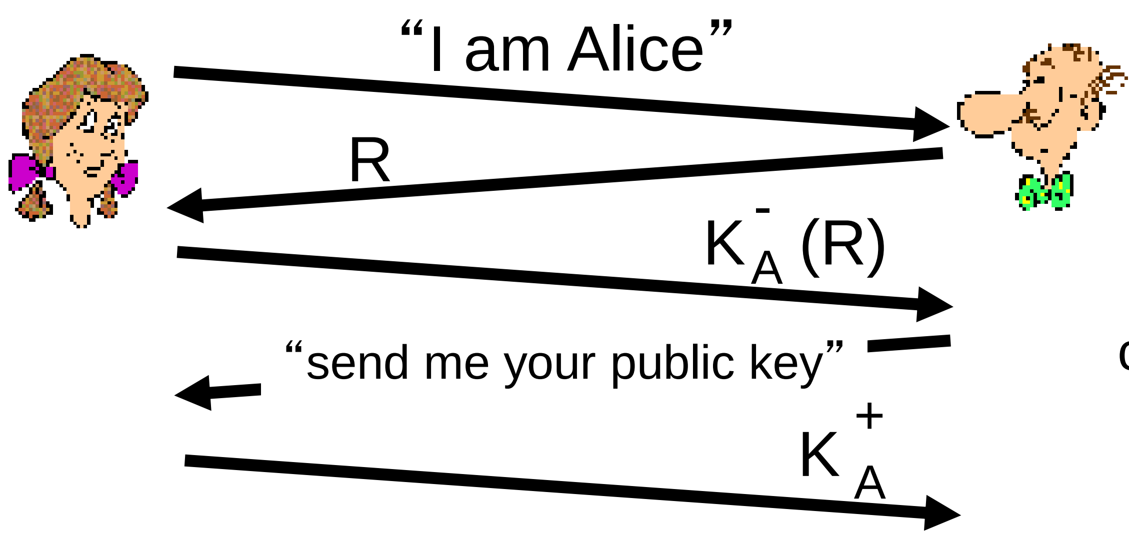
Failures, drawbacks?

# 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

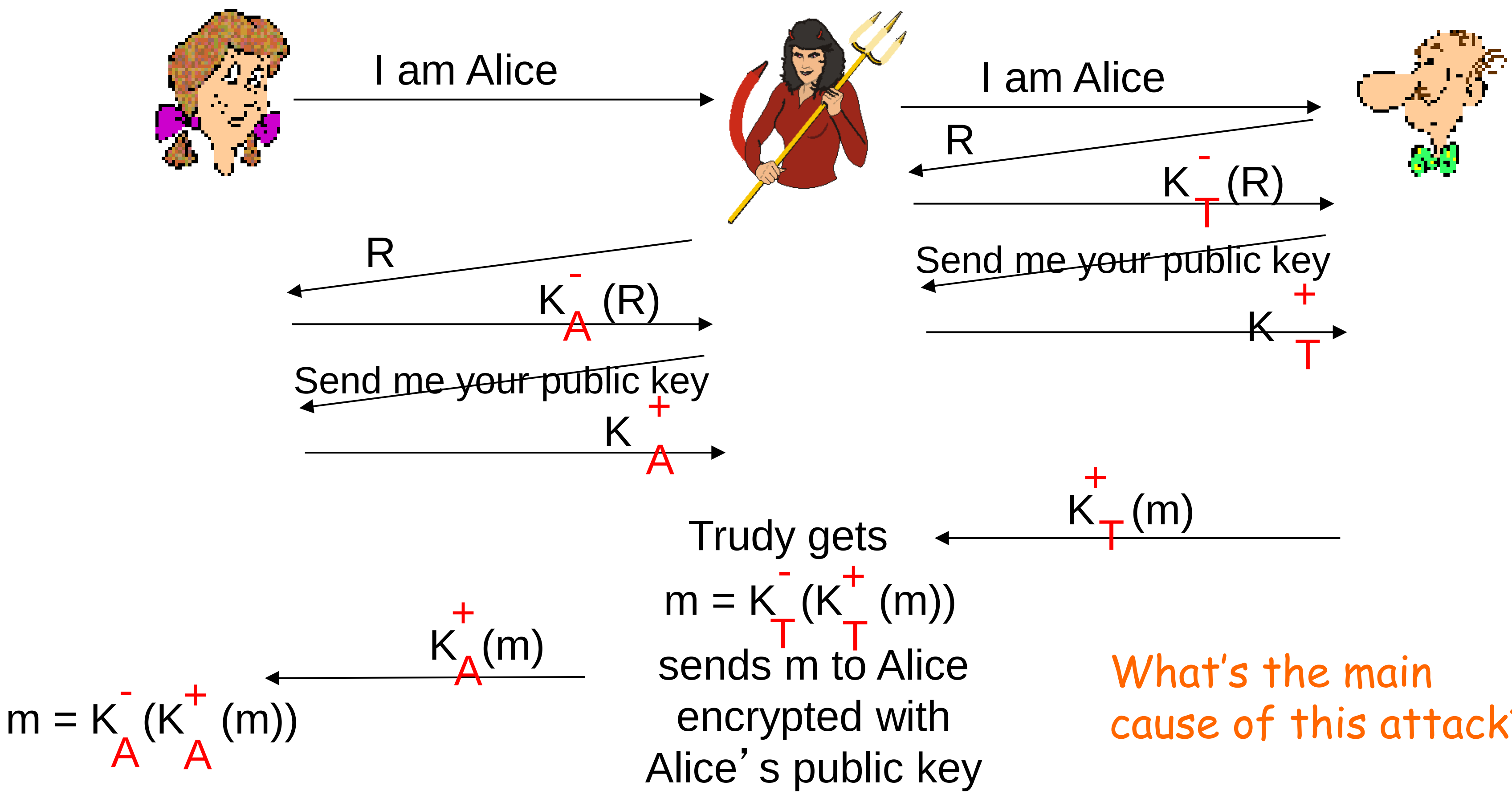
$$K_A^+(K_A^-(R)) = R$$

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

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

# ap5.0: security hole

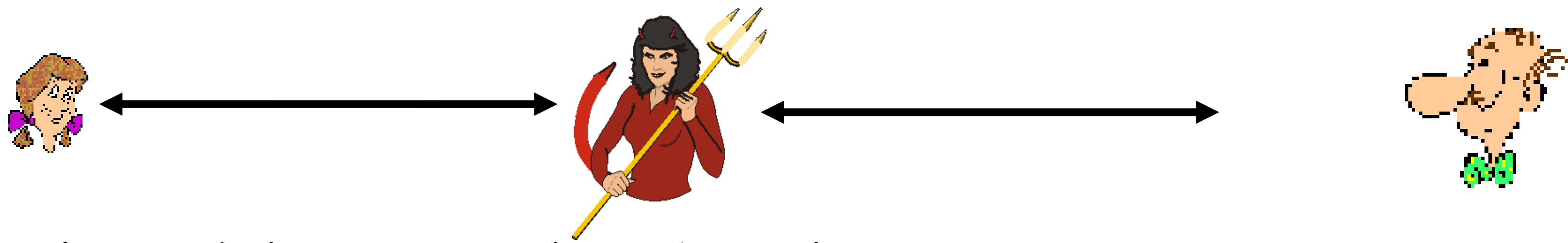
*man (or woman) in the middle attack:* Trudy poses as Alice (to Bob) and as Bob (to Alice)




What's the main cause of this attack?

# ap5.0: security hole

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



Result: Trudy knows everything about the conversation – confidentiality breached

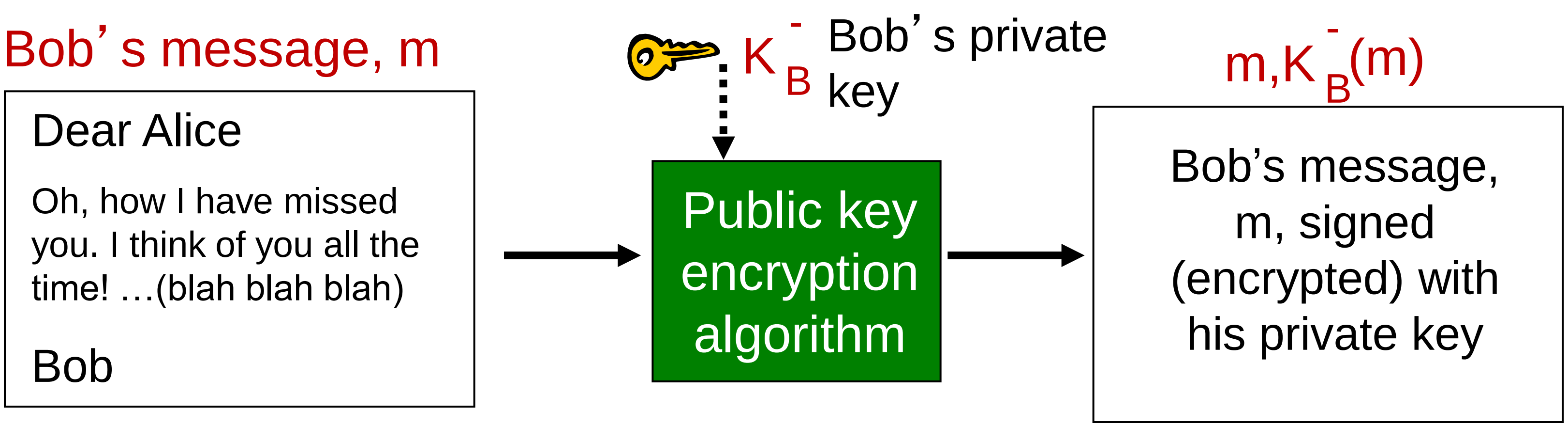
difficult to detect:

- 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!
- This attack is totally transparent to Alice and Bob

# 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

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 the document (*non-repudiation*)
- Can we use symmetric key for non-repudiation?

# 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.

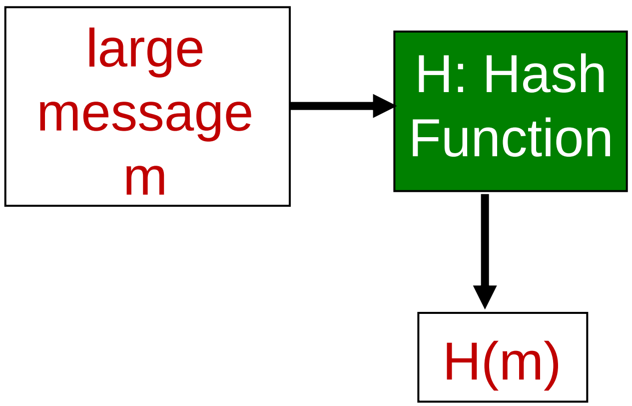
Alice thus verifies that:

- ü Bob signed m
- ü no one else signed m
- ü Bob signed m and not m'

Now we really achieve 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:

- produces fixed-size *message digest* (fingerprint), generally much smaller than message
- many-to-one (collisions possible, but hopefully rare)
- 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!

Will birthdays work better as message digests?

# Homework 3.1: Birthday Attack (due: Apr 4, midnight)

- Can I use birthday of John as John's "digest"? - Do two of us have the same birthday? (Assume 365 possible birthdays.)
- Given k people, what is the probability that none of them share the same birthday? (Write down formula.)
  - When first person announces her birthday, what's probability it won't collide with a previously announced birthday?
  - After second person announces her birthday, what's the probability of no collisions so far?
  - After third person?
  - Can you now write down a general formula of the probability of no collisions after all k persons have announced?
  - Now, write each factor in your formula in the form (1 – x), so we can answer the next question.

# Birthday Attack (cont'd)

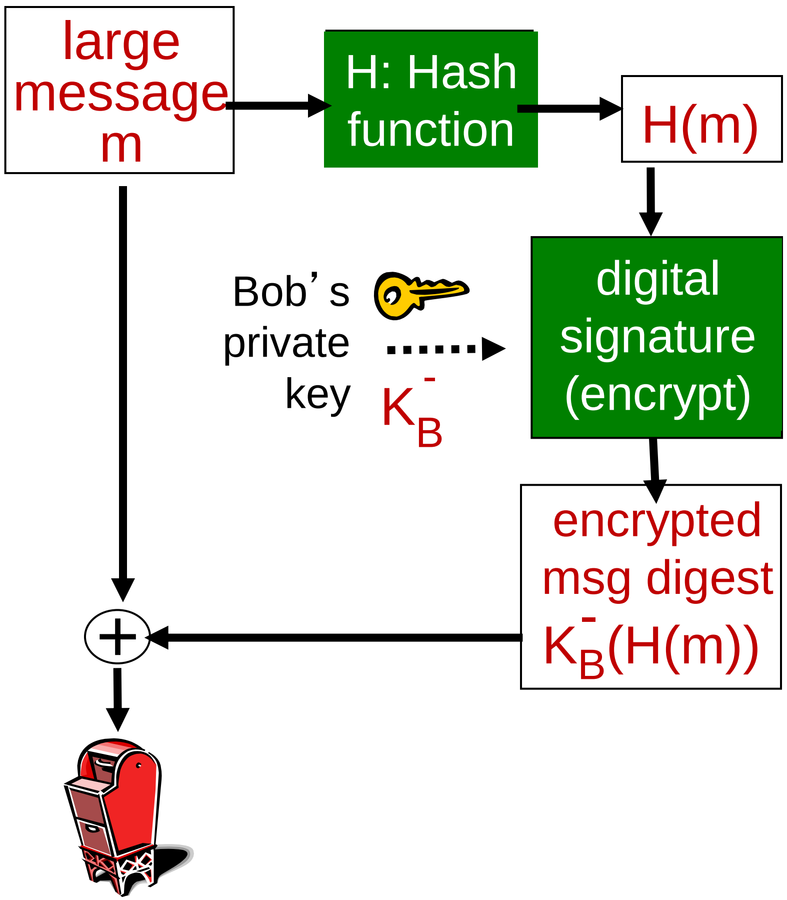
- Using the approximation $1 - x \approx e^{-x}$ (for x small), write down the previous formula of no collisions in the form $e^{y}$.
- Show that if $k > 23$, then the probability of no collisions is $< 0.5$, i.e., if we have more than 23 people, it's more likely than not for two of them to share the same birthday.
- So if people are messages, and their birthdays are message digests, it's too easy for these digests to collide. Describe a security attack based on easy collisions of message digests.

# Hash functions that do work (when birthdays don't)

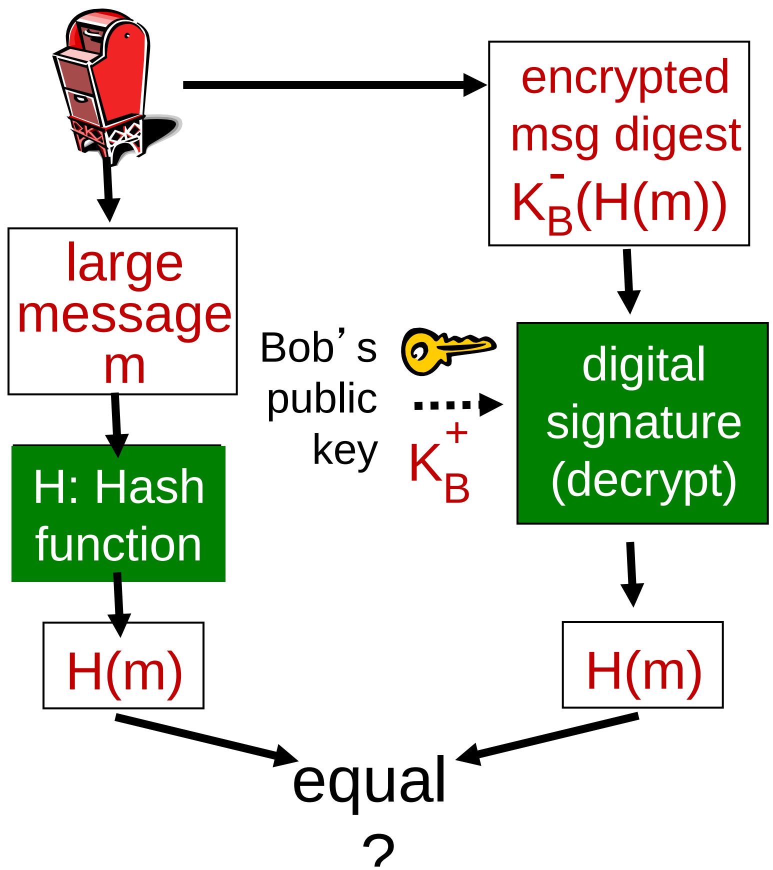
- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string x, appears difficult to construct msg m whose MD5 hash is equal to x
- SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

# Digital signature = signed message digest

Bob sends digitally signed message:



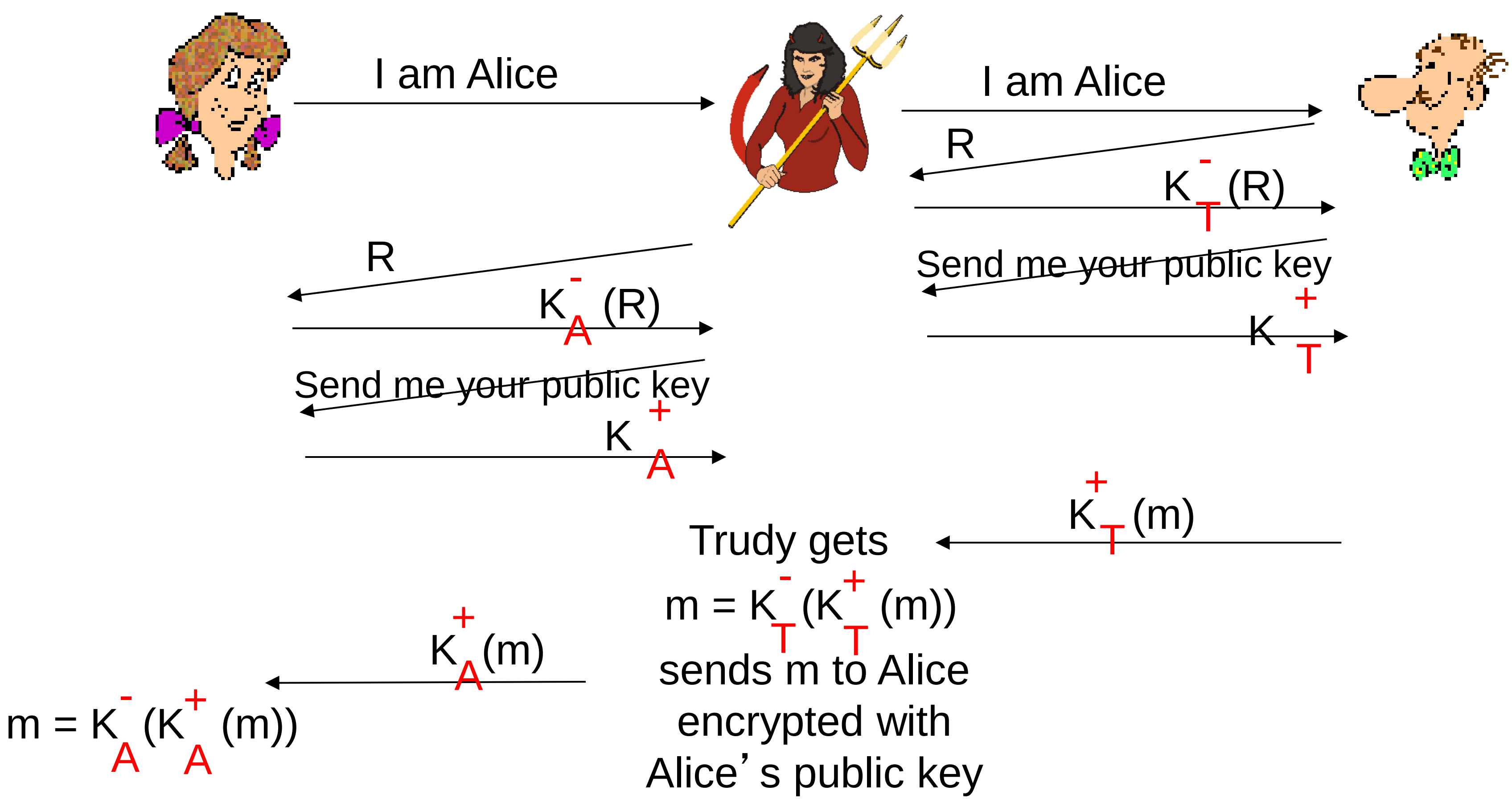

Alice verifies signature, integrity of digitally signed message:




If Alice's check goes through, what does she know?

# Recall: ap5.0 security hole

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



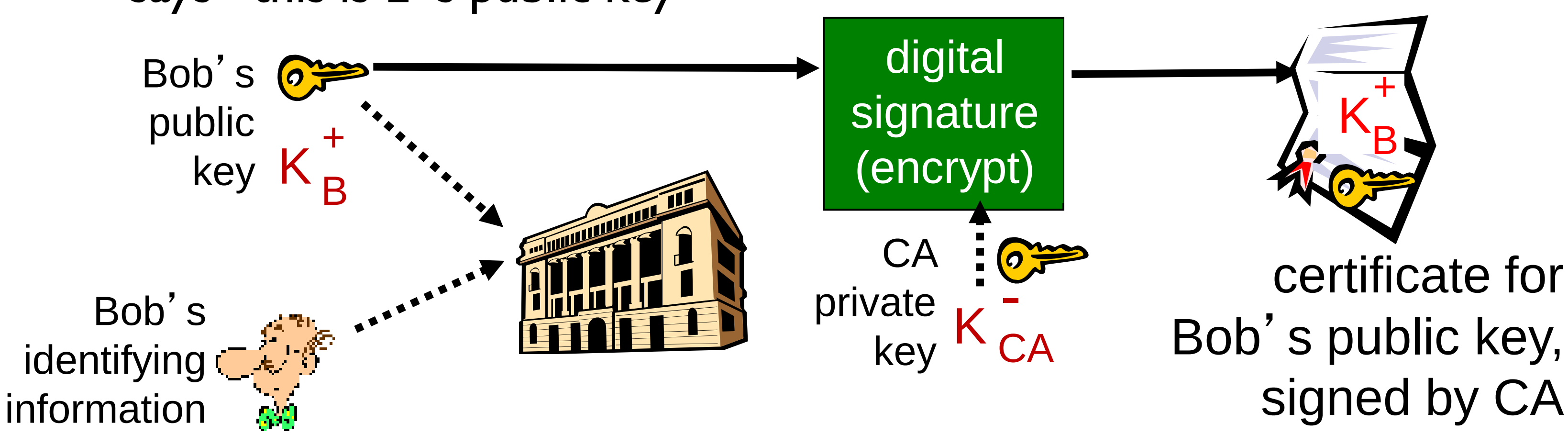
# Public-key certification

- 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

How's this attack different from the one on Slide 8.41 in terms of impact?

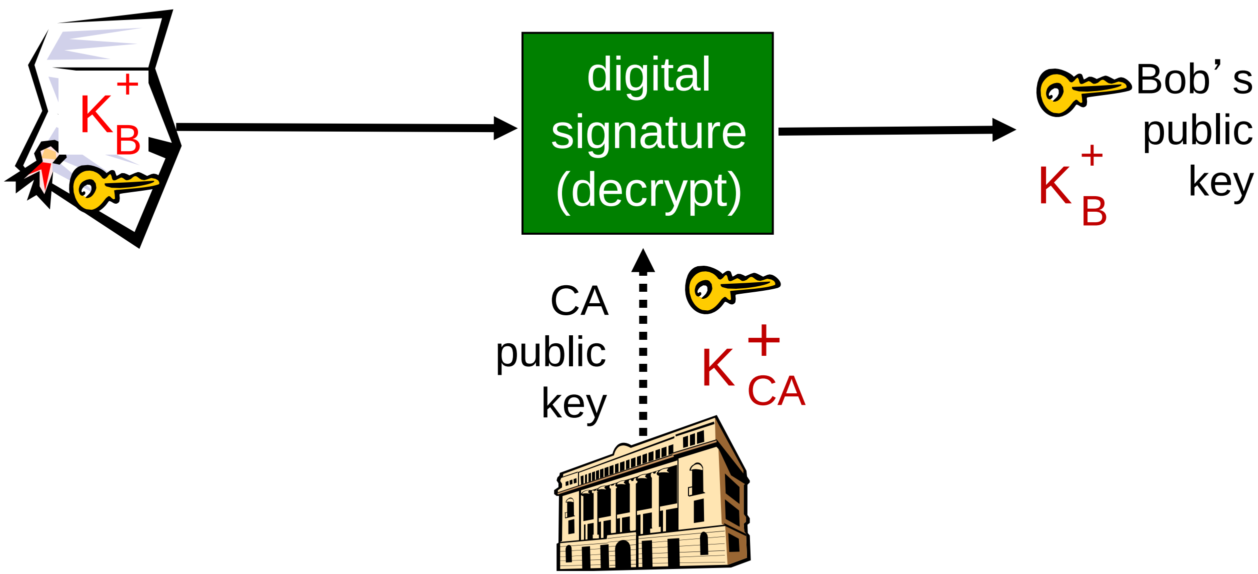
# Certification authorities

- *certification authority (CA):* binds public key to particular entity, E. Can be government (e.g., IDA) or well known provider (e.g., VeriSign)
- 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

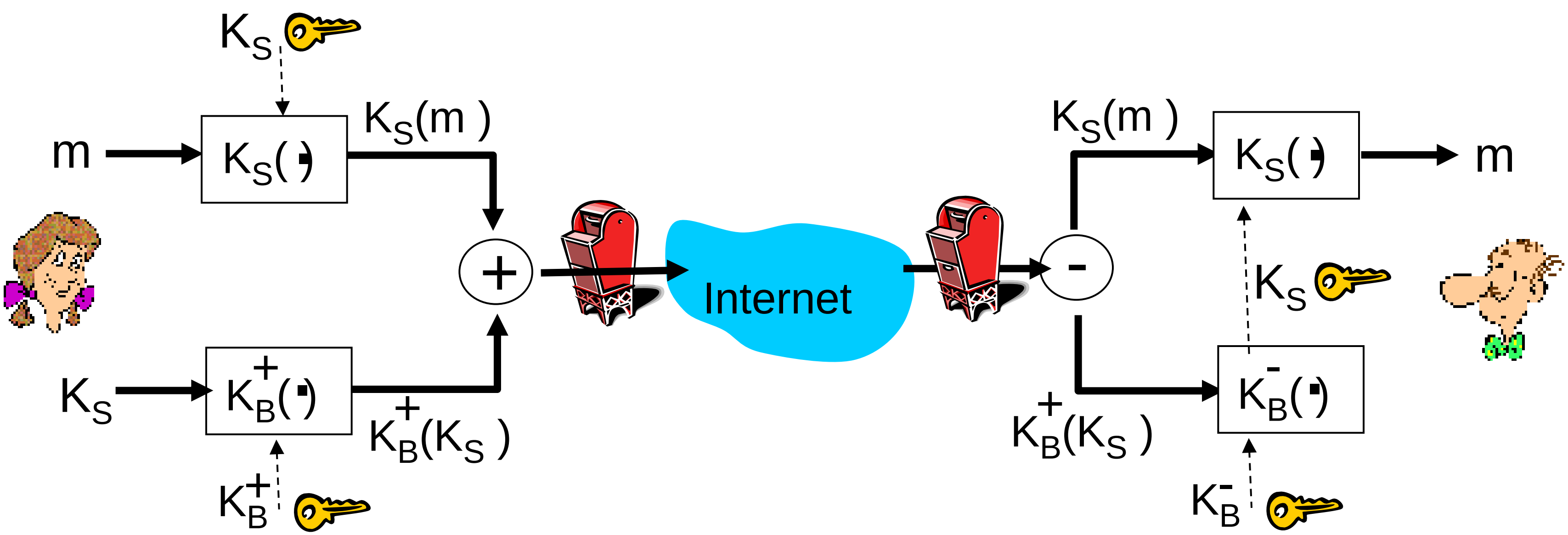
- when Alice wants Bob's public key:
  - gets Bob's certificate (from Bob *or elsewhere*).
  - apply CA's public key to Bob's certificate, get Bob's public key




Instead of trusting Bob's public key, we trust CA's public key. So we use trust for CA to *bootstrap* trust for Bob. – Why is this practical?

# Secure e-mail

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

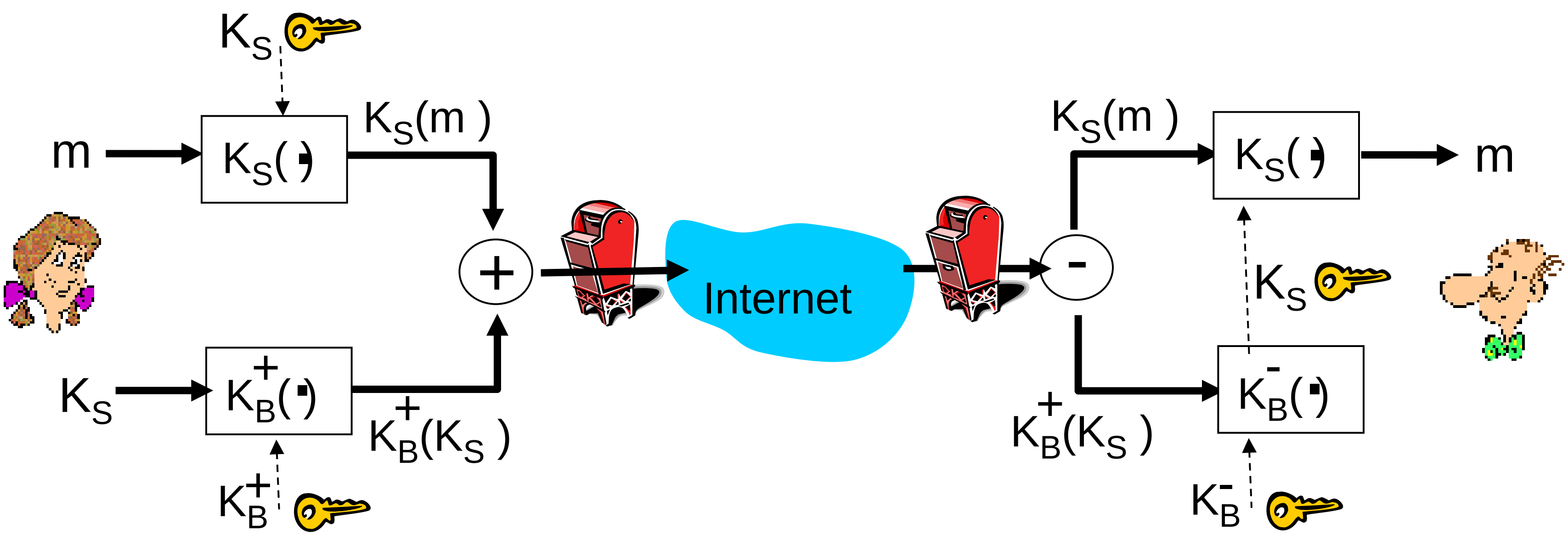



*Alice:*

- 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.

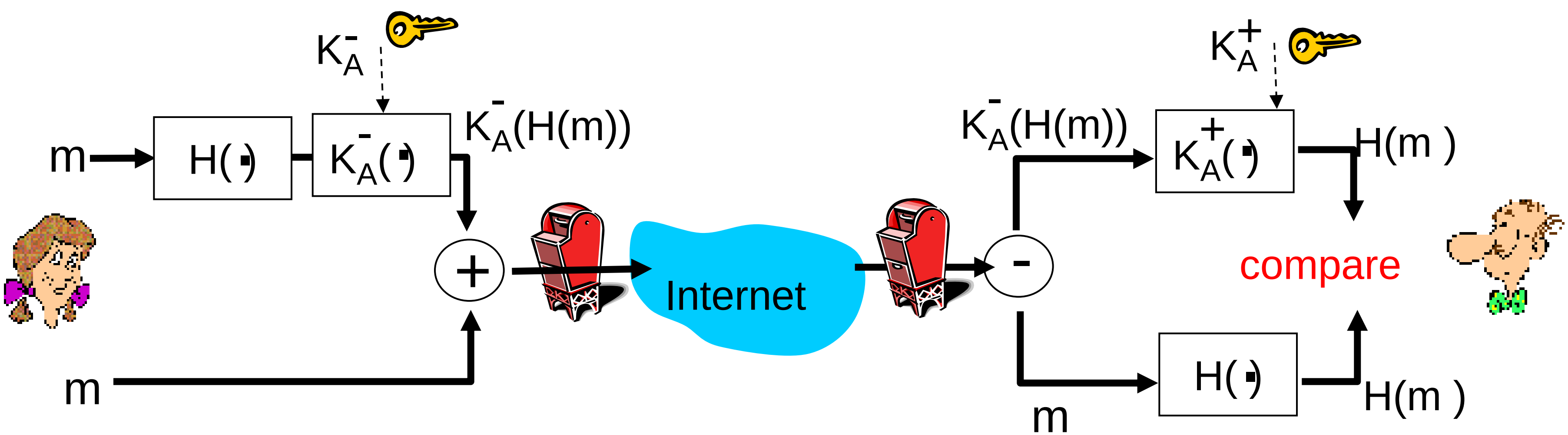



*Bob:*

- 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)

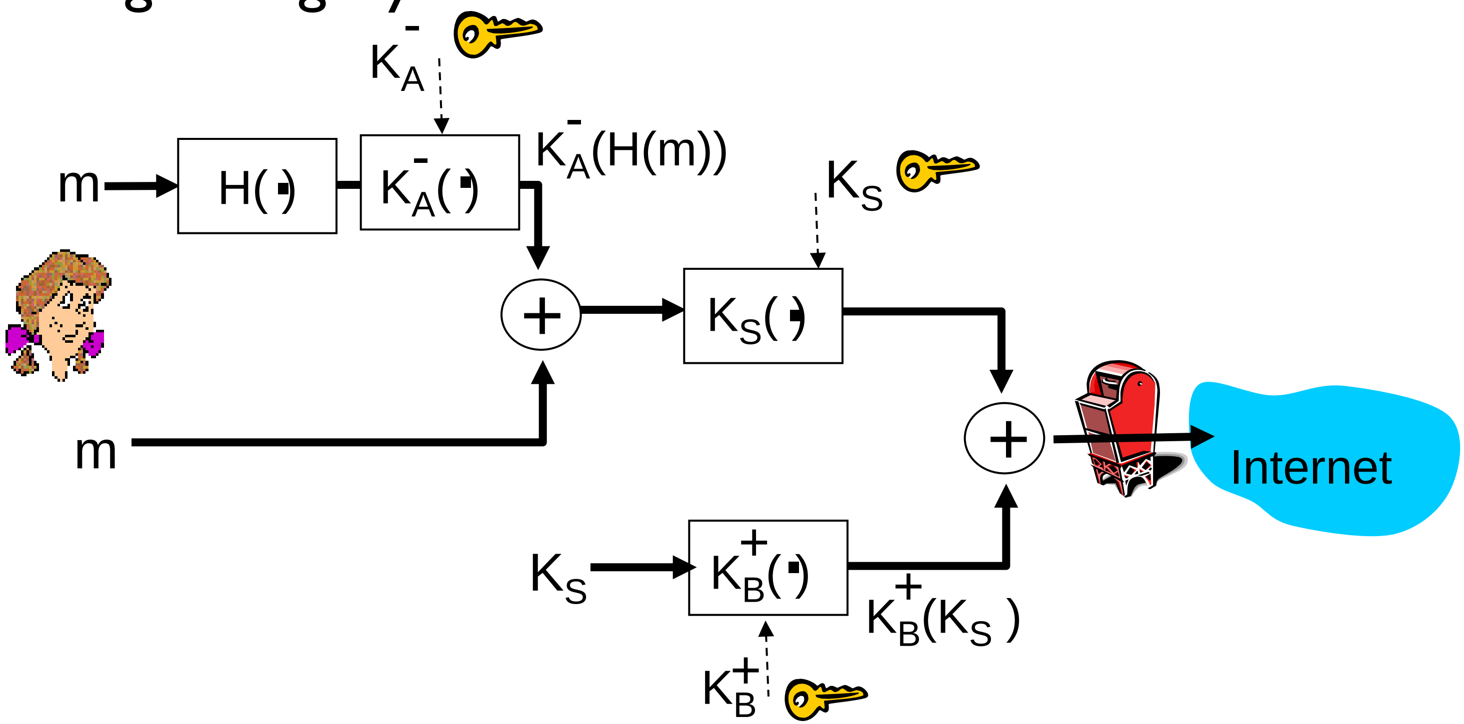
- Alice wants to provide sender authentication message integrity




- 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