Data Communication and Computer Networks

11. Computer Networks Security

Dr. Aiman Hanna

Department of Computer Science & Software Engineering
Concordia University, Montreal, Canada

These slides have mainly been extracted, modified and updated from original slides of :
Computer Networking: A Top Down Approach, 6th edition Jim Kurose, Keith Ross
Addison-Wesley, 2013

Additional materials have been extracted, modified and updated from:
Understanding Communications and Networking, 3e by William A. Shay 2005

What is network security?

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

- sender encrypts message
- receiver decrypts message

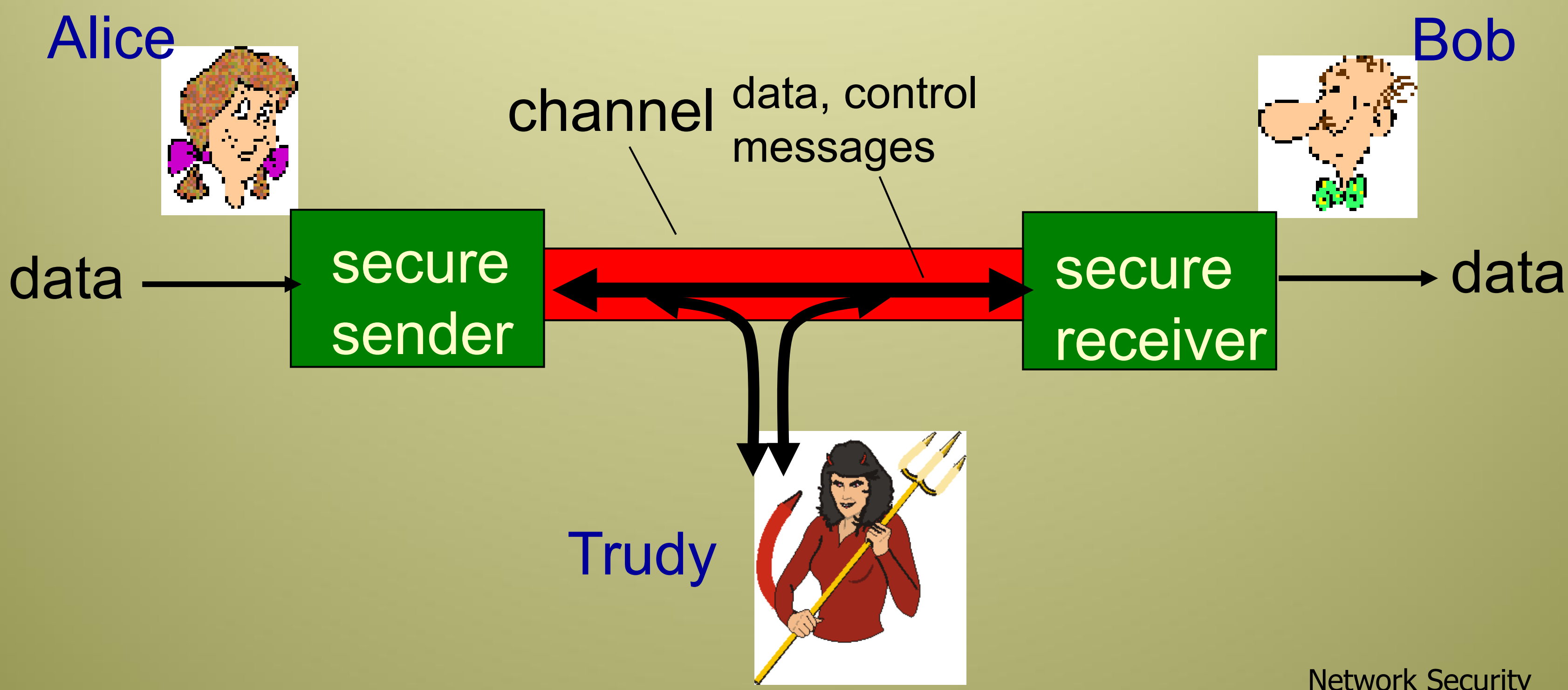
authentication: sender, receiver want to confirm identity of each other

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 (friends) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages



Who might Bob, Alice be?

- … well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- other examples?

There are bad guys (and girls) out there!

Q: What can a "bad guy" do?

A: A lot!

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

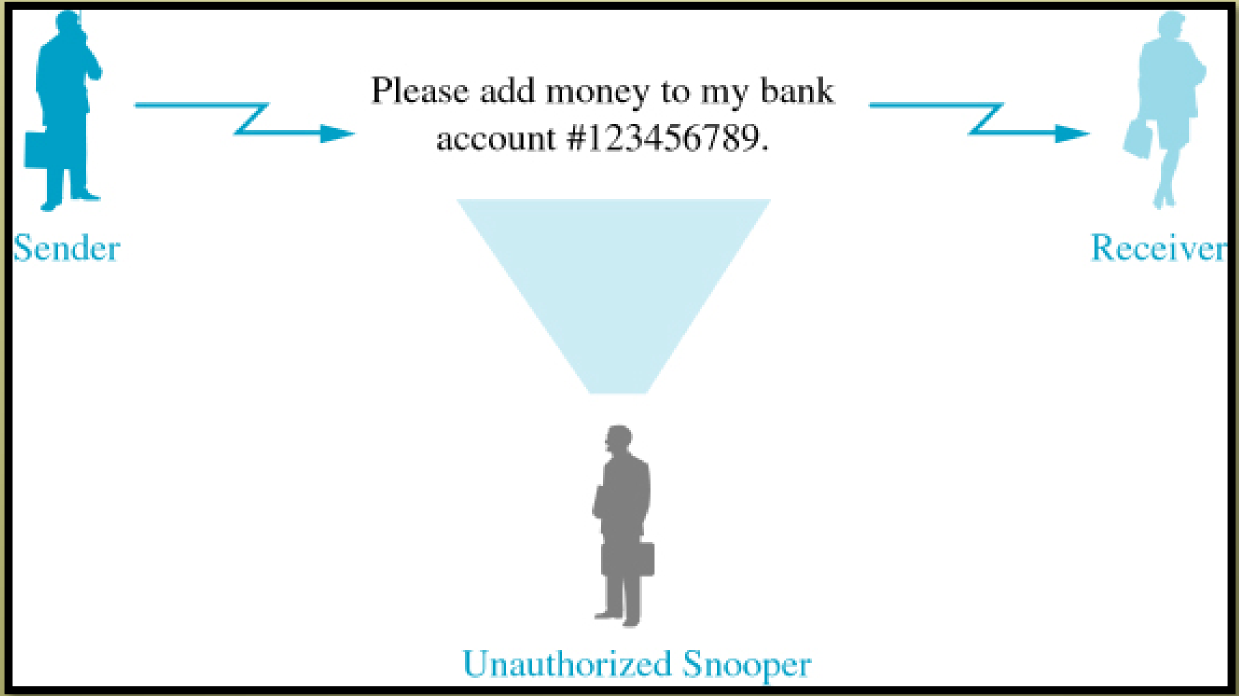
Network security

- ideal scenario: prevent any unauthorized person from intercepting/viewing what is being transferred
- however, this may not be possible
- so, do not secure data; rather prevent unauthorized person from *understanding* them
- ***encryption*** is used to achieve that

The language of cryptography

- **Privacy:** prevent a third party from intercepting the information, and if intercepted from understanding it

- encrypt the information, decryption is then necessary to understand it

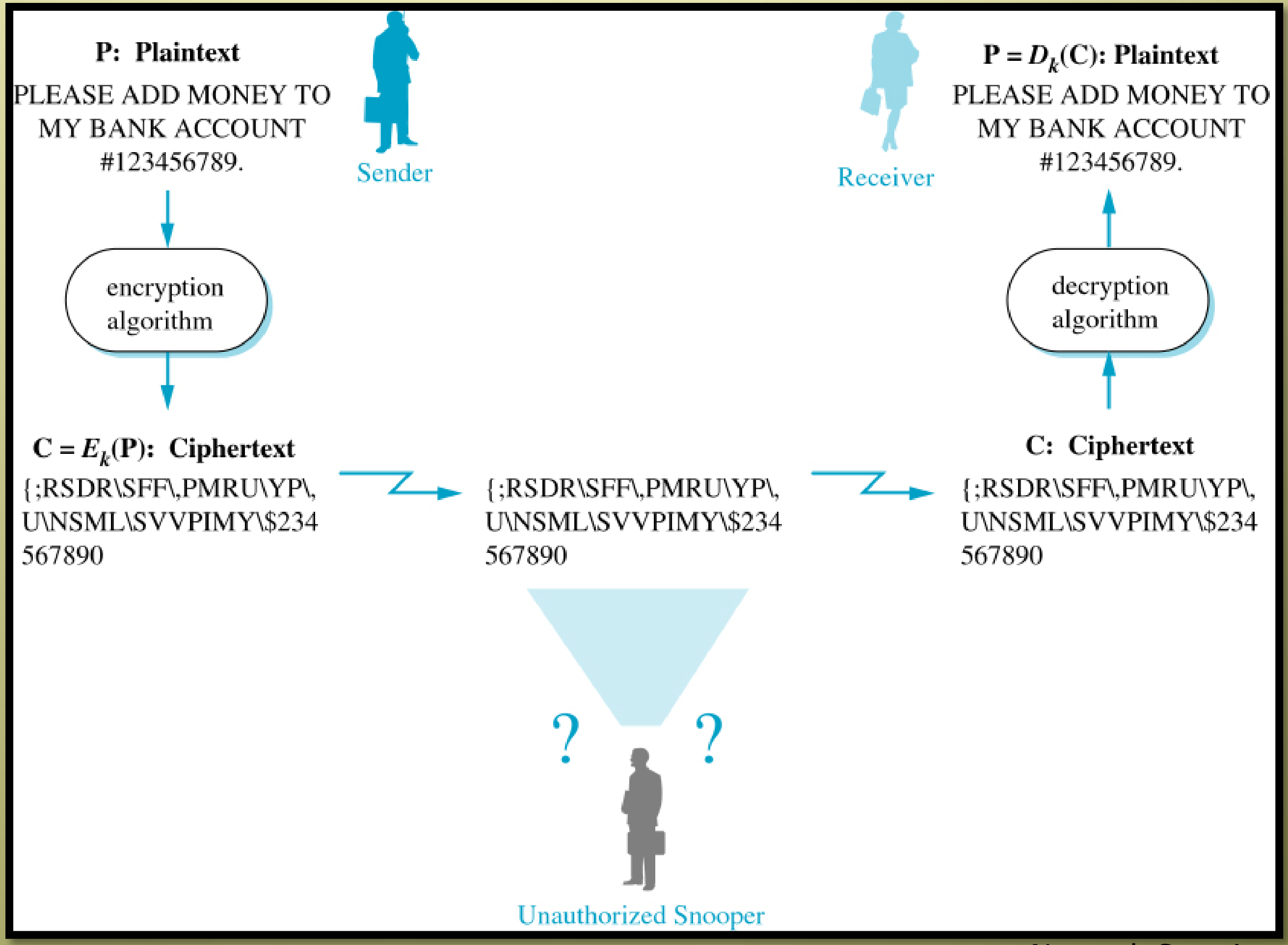


Sending Unsecured Messages

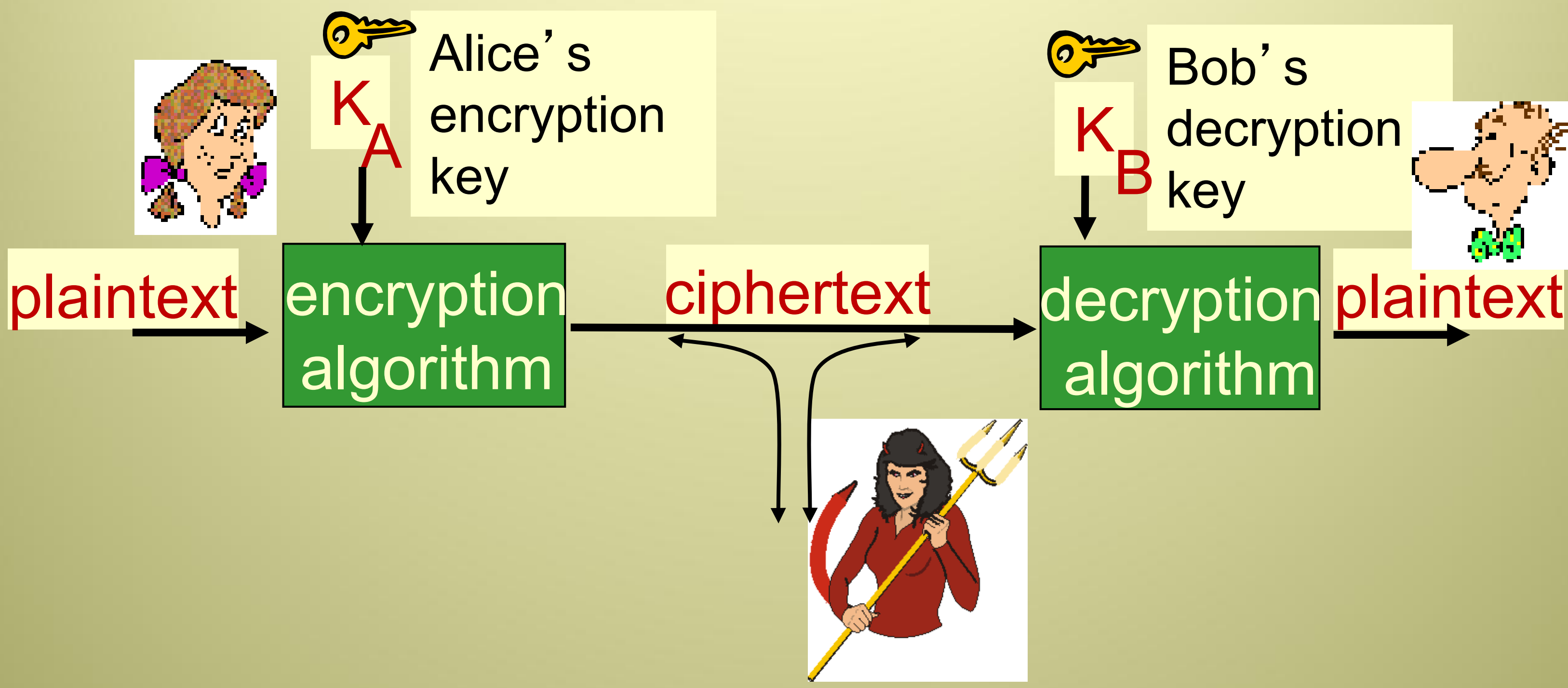
The language of cryptography

- ***Plaintext:*** the message before encryption
- ***Ciphertext***: the encrypted message

Sending Encrypted Messages



The language of cryptography



m plaintext message

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

$m = K_B(K_A(m))$

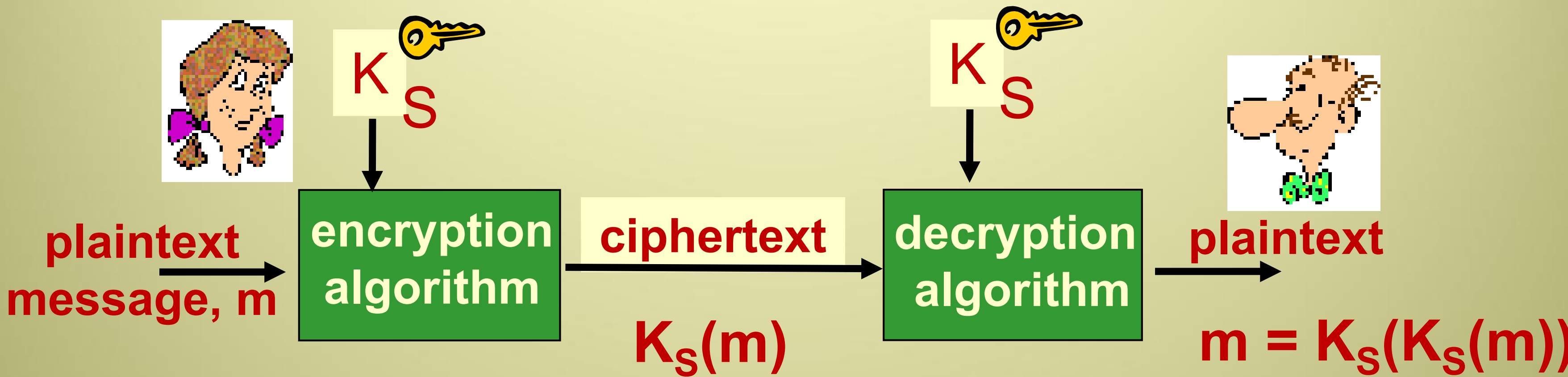
Breaking an encryption scheme!

- encrypted data is decrypted if the encryption key and method are known
- if the encryption and decryption keys are the same, this is called *symmetric key cryptosystem*
- issues:
  - What happens if the key is compromised?
  - What happens if the key and encryption methods became known?
  - What about authentication & authorization?

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,b
- chosen-plaintext attack: Trudy can get ciphertext for chosen plaintext

Symmetric key cryptography



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

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher

Q: how do Bob and Alice agree on key value?

Encryption schemes

Caesar Cipher

- replaces each character by another

- *Example*: What is the plaintext of the following ciphertext:

  Yjq%mpqyu%yjgtg%vjg%tqcfu%yknn%ngcf%wu,%qpna%c%hqqn
  %yqwnf%uca

  this is not so difficult to guess! Is it?

- once some characters are guessed, the rest like the TV show *Wheel of Fortune*

Encryption schemes

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

Encryption schemes

Polyalphabetic Cipher

- plaintext characters are not always replaced with the same ciphertext character
- for example, replace each character depending on character sequence as well as its position in the message

```
for (int i=0; i < length of P; i++)
    C[i] = P[i] + K + (i mod 3)
```

- If K=10, then 10 is added to characters in position 0, 3, 6, …; and 11 is added for those in positions 1, 4, 7; and 12 is added for those in 2, 5, 8

Encryption schemes

Polyalphabetic Cipher

- *Example*: THEMTHENTHEY will be UJHNVKFPWIG
- THE is encrypted into UJH, VKF, and WIG
- repetition is fewer but still there
- a professional thief may still be able to break-in

A more sophisticated encryption approaches

Transposition Cipher

- rearrange the plaintext characters into a 2-D array and sends columns based on a specific permutation

- problem: character frequencies are preserved
- *Example*: FOLLOW THE YELLOW BRICK ROAD
  - if p_1=2, p_2=4, p_3=3, p_4=1, p_m=5 then the encrypted msg is:

    O YWCALHLB LTE KDFW OIOOELRR

COLUMNS

1	2	3	4	5
F	O	L	L	O
W		T	H	E
	Y	E	L	L
O	W		B	R
I	C	K		R
O	A	D		

A more sophisticated encryption approaches

- n substitution ciphers, $M_1,M_2,\ldots,M_n$
- cycling pattern:
  - e.g., n=4: M_1,M_3,M_4,M_3,M_2; M_1,M_3,M_4,M_3,M_2; ..
- for each new plaintext symbol, use subsequent 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

- key need not be just n-bit pattern

A more sophisticated encryption approaches

Bit-Level Ciphering

- not all transmissions are over characters
- creates a key (a bit string) secretly and randomly
- divides the message into substrings of the same length as the key
- XOR all substrings with the key and transmit the result
- decryption in that case is not a reverse operation; rather a repetition of the encryption operation

A more sophisticated encryption approaches

Bit-Level Ciphering

- Example:

1101100101001	Plaintext
1001011001010	Encryption key
0100111100011	Ciphertext = plaintext exclusive OR'd with the encryption key
1001011001010	Decryption key (same as the encryption key)
1101100101001	Plaintext = ciphertext exclusive OR'd with the decryption key

Encryption Using XOR Bit Operation

- key length is sensitive here. Why?

A more sophisticated encryption approaches

Bit-Level Ciphering

- advantages:
  - key is used once, so comparisons to other cipher texts is not possible, so code is unbreakable without trying all possible decryption keys

- such unbreakable ciphers are also called ***one-time pads***

- disadvantages:
  - keys, sometimes large ones, must be communicated to the receiver
  - keys are used only once!

Symmetric key crypto: DES

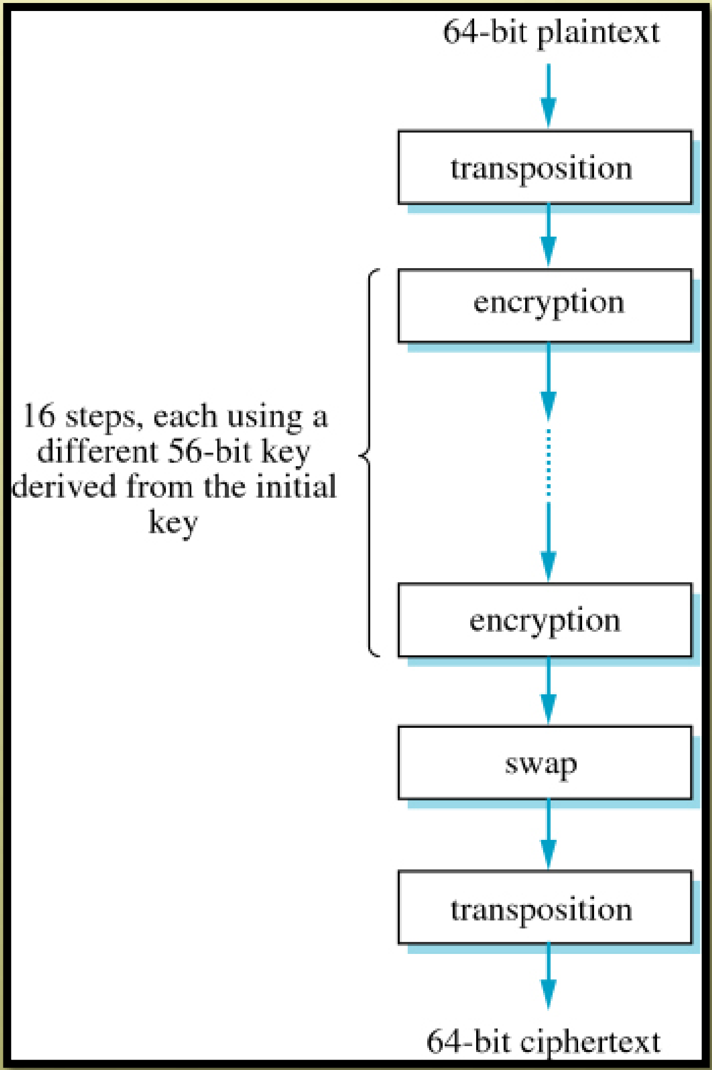
Data Encryption Standards (DES)

- widely used as encryption standard (US encryption standard [NIST 1993])
- divides messages into 64-bit blocks and encrypts each one (using 56-bit symmetric key)
- 8 bits are used for error detection, so the key used is 56-bit
- employs complex steps including transposition, XOR, substitutions, and others
- in general, DES has a total of 19 steps, where the output of each step is the input of the following one

Symmetric key crypto: DES

DES: Data Encryption Standard

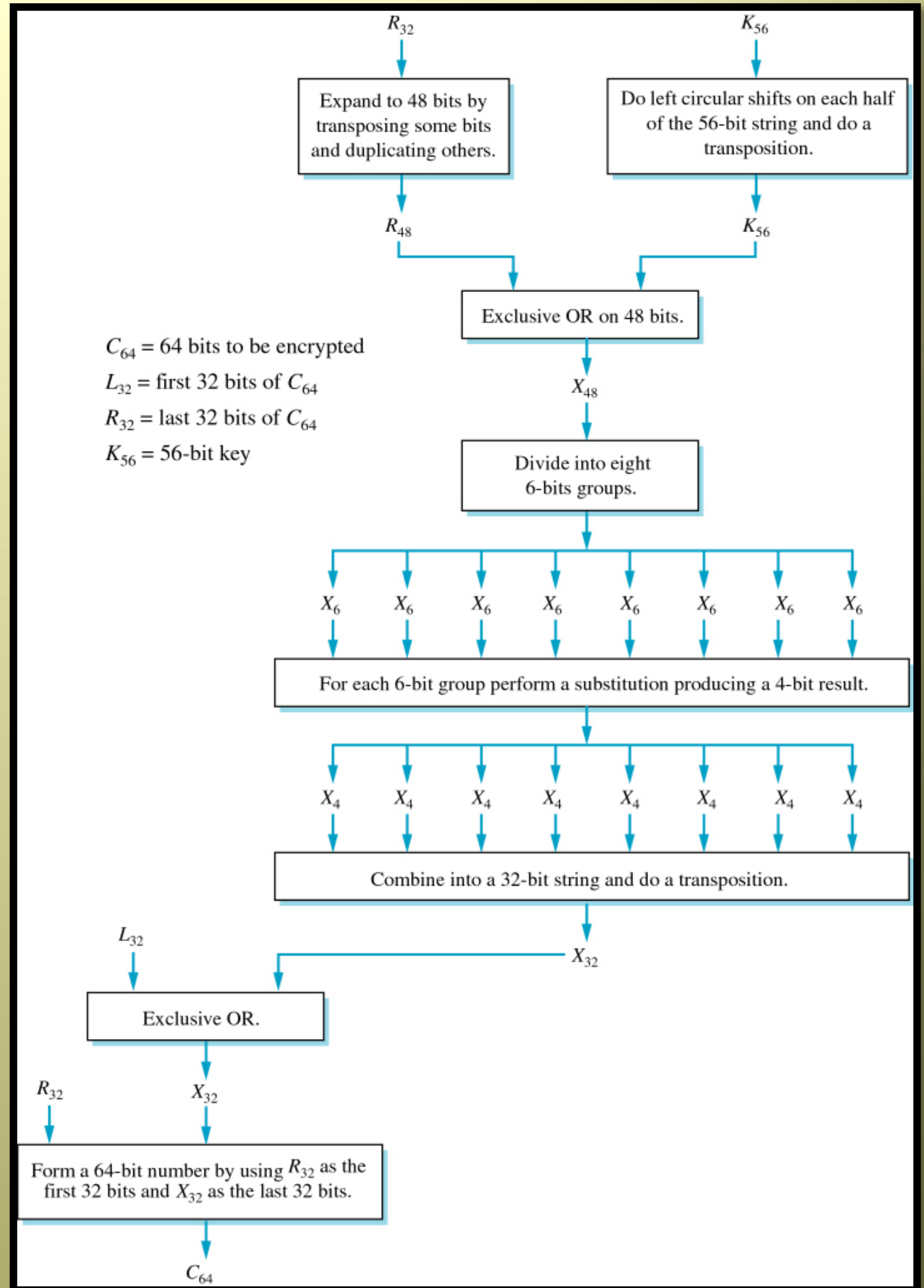
Outline of DES



Symmetric key crypto: DES

DES: Data Encryption Standard

One of the 16 Steps of DES



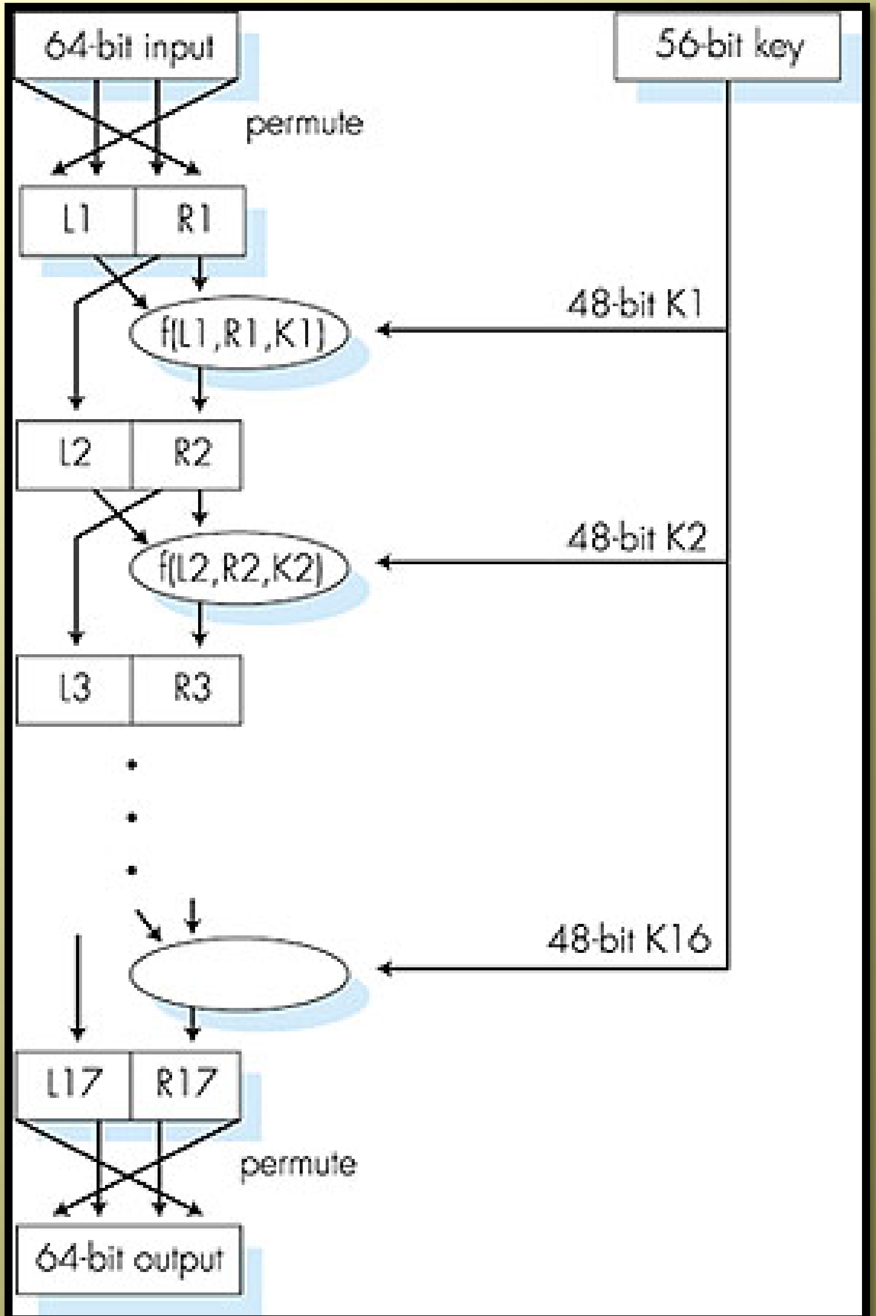
Symmetric key crypto: DES

DES operation

initial permutation

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

final permutation



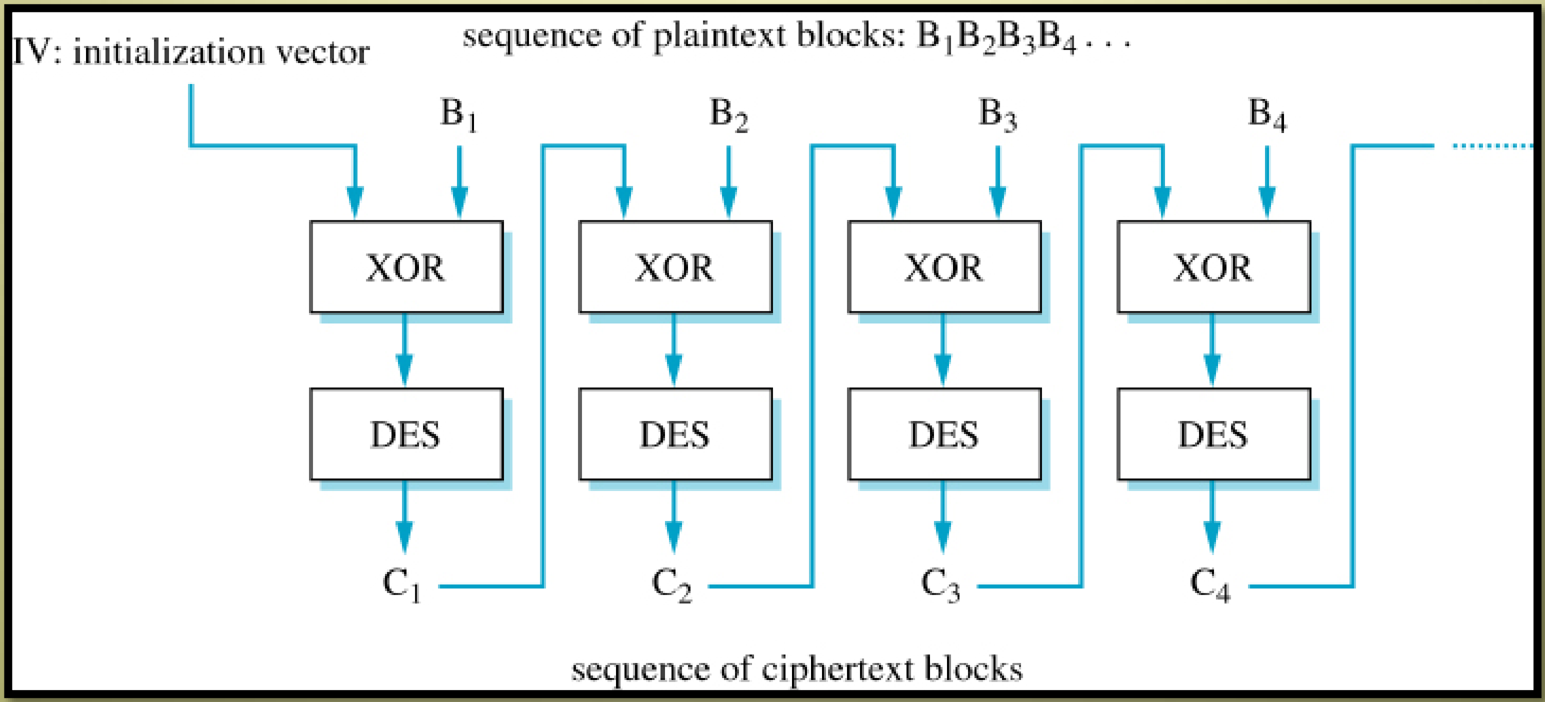
Symmetric key crypto: DES

Data Encryption Standards (DES)

- DES can operate in several modes including:
  - **ECB –** ***Electronic Codebook***
  - **CBC –** ***Cipher Block Chaining***
- with ECB, if the original string has similar 64-bit blocks, then the cipher of these blocks is consequently the same
- this is not good since patterns are possible
- CBC disrupts this pattern by performing an XOR between the block and the previous encrypted block before encrypting the new block
- the 1st XOR is performed with an ***initialization vector***

Symmetric key crypto: DES

Data Encryption Standards (DES) – CBC Mode



Cipher Block Chaining (CBC) Mode of DES Encryption

Symmetric key crypto: DES

- ## How secure is DES?
  - unless the key is knows it is very difficult to break
  - ***Brute Force*** attack is a method where all possible keys are attempted
  - 56-bit key ➔ $2^{56} \approx 7.2 \times 10^{16}$ possible keys
  - brute force these keys would take 4500 years in an Alpha station

- for many years, researchers tried to break DES without success

- in 1998, the Electronic Frontier Foundation built a DES Cracker, a specially designed computer, at a trivial cost of 250,000$
  - EFF_DES_cracker

- that event rendered DES obsolete!

- a key of 128 bits could provide a reasonable solution
  - $2^{128} \approx 3 \times 10^{38}$ possible keys
  - A system that tries 1 billion keys / microsecond would still take about 9.5×10^{15} years to resolve all keys

Triple DES

- provides an alternative to DES
- encrypts data 3 times
- for example, suppose $E_K(M)$ and $D_K(M)$ are DES encryption and decryption using a key K, triple DES is calculated as

$$\mathbf{E_{K3}(D_{K2}(E_{K1}(M)))}$$

- Triple DES uses 168-bit key, and proved to be solid
- relatively slow; 3 times of DES

AES: Advanced Encryption Standard

- another alternative to DES & Triple DES is the Advanced Encryption Standard (***AES***)
- symmetric-key NIST standard, replaced DES (Nov 2001)
- AES uses:
  - Rijndael algorithm, with
  - 128, 192, or 256 bits key
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES! [NIST 2001]

Key distribution and protection

- all the major encryption methods depends on a **secret** key
- if the key is compromised, the algorithm *may* hence become useless
- how do sender and receiver exchange keys securely prior to the session?
  - **Shamir's method**
  - **Diffie-Hellman key exchange**

Key distribution and protection

Shamir's method

- can be used if the information is so sensitive that no single person can be trusted to keep, send or receive it

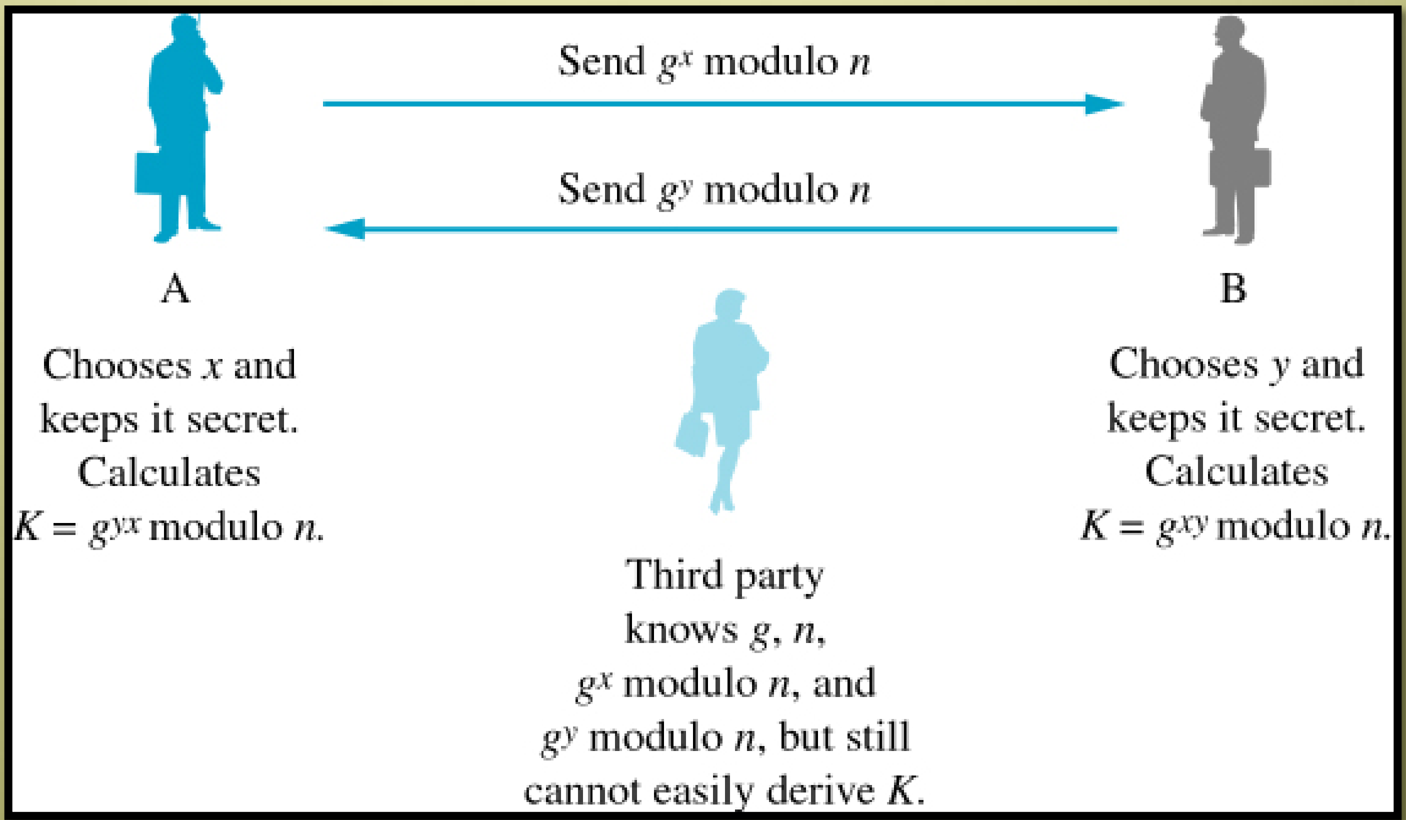
- in such case, instead of keeping the key at one location, break it into different pieces

- Shamir's method does not actually break the key itself; rather
  - uses polynomial $p(x)= a_0+a_1x+a_2x^2+\ldots+a_{k-1}x^{k-1}$
  - each person is given a unique point (part of this polynomial)
  - P(x) can be calculated by communicating only those unique points
  - one of the coefficients 'a_i' is the key

Diffie-Hellman key exchange

- sender chooses a value x and keeps it secret; receiver chooses a value y and keeps it secret
- the sender and receiver can then calculate the key

Diffie-Hellman Key Exchange



Diffie-Hellman key exchange

- Disadvantages:
  - both n and g must be very large (perhaps a thousand bits), in order to make it difficult to determine the key
  - susceptible to the ***Man-in-the-middle attack***
- an intruder may intercept g^x *modulo n* from ***A*** and forward it to ***B*** as $g^{x'}$ *modulo n*
- the intruder then intercepts g^y *modulo n* from ***B*** and forward it to ***A*** as $g^{y'}$ *modulo n*
- as far as A and B are concerned, all is fine. The intruder then uses $g^{xy'}$ *modulo n* to communicate with A and $g^{yx'}$ *modulo n* to communicate with B
- both A and B believe that they are communicating with each other while in reality, each of them is communicating with the intruder which decrypt the messages, then re-encrypting them and send them to the other side

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")?

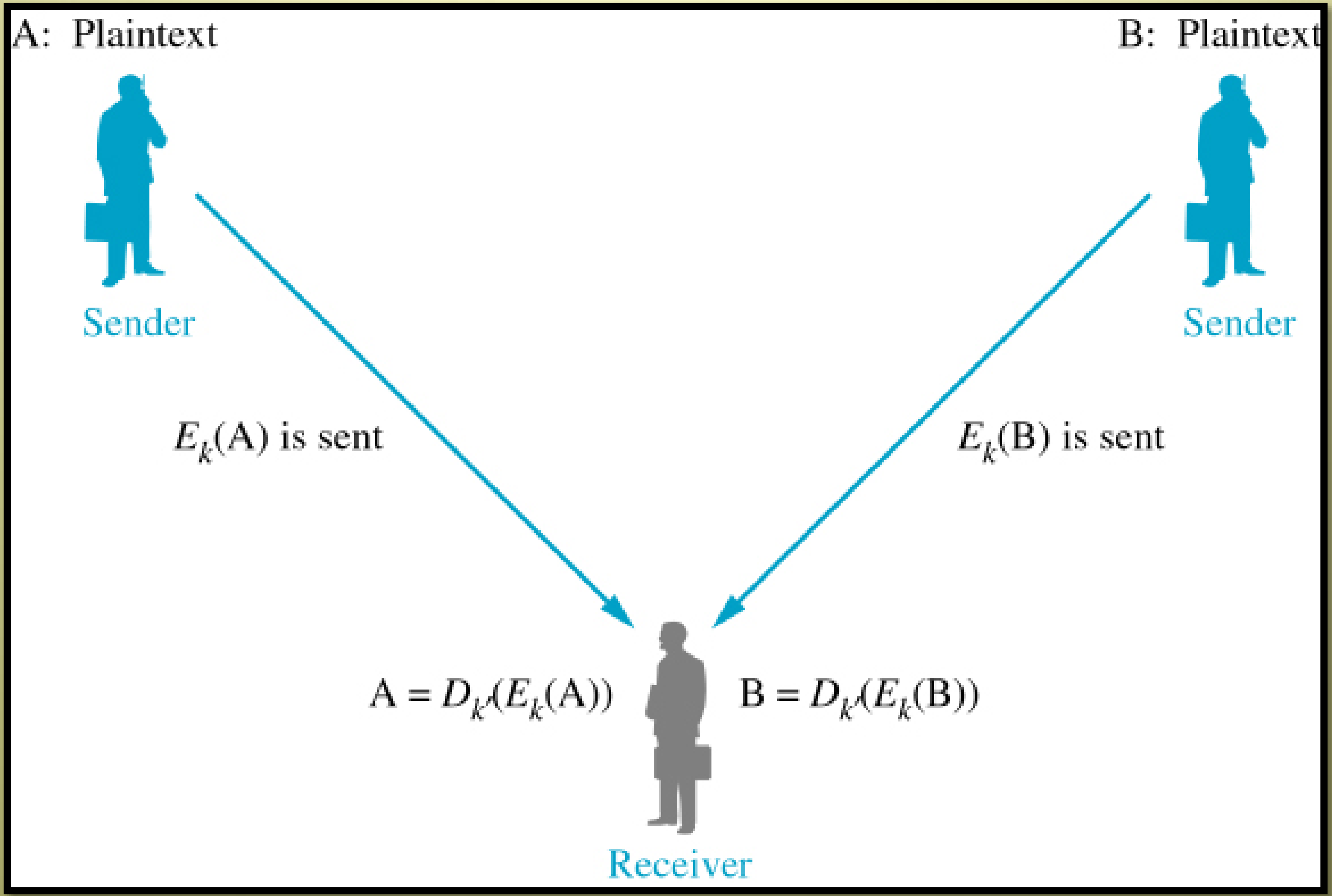
public key crypto

- 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

- ***reasonable assumption:*** If you know the encryption algorithm and the key then you can decrypt
- a fact: In real life, not every reasonable thing holds true
- the idea here is to have the encryption algorithm known, and have the key *public* (known to the entire world)!
- yet, have only the receiver capable of decrypting the message
- each receiver has some secret knowledge, for example a private key, that is necessary to decrypt the message
- such systems are called ***Public Key Cryptosystems***

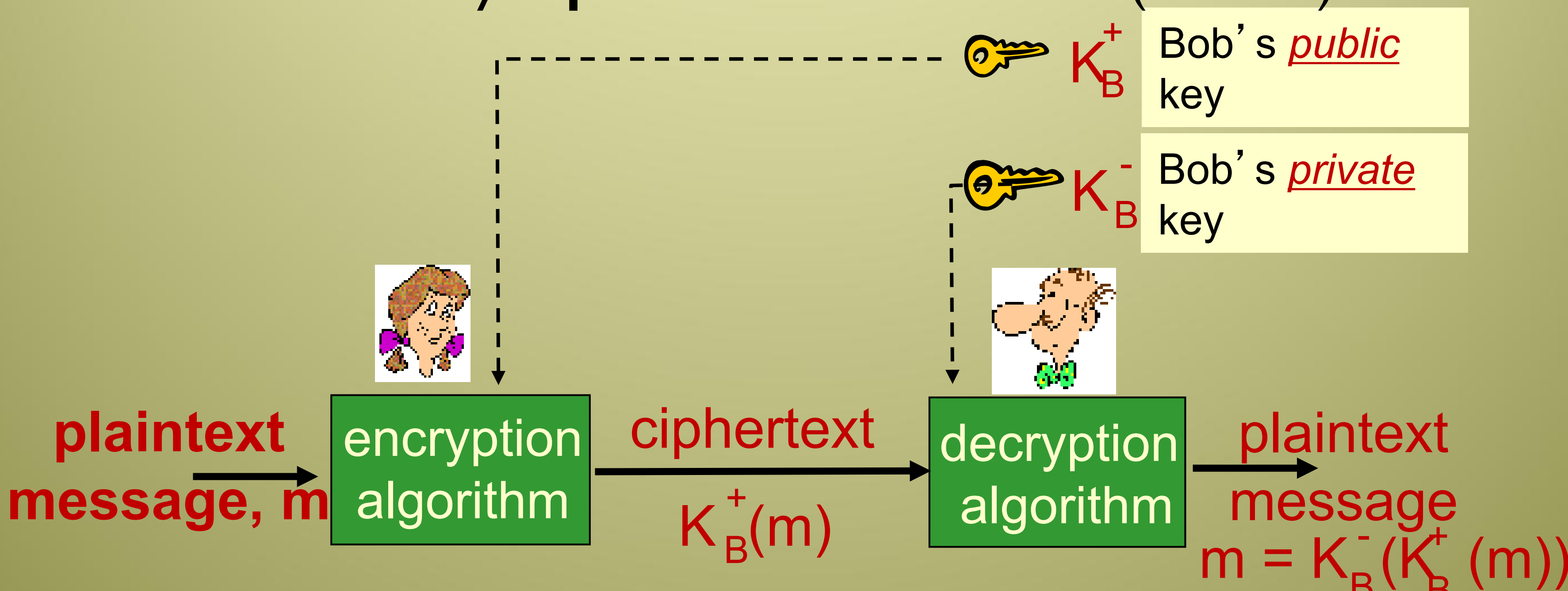
Public key cryptography



Multiple Senders Using The Same Encryption Method

Public key cryptography

- assume the existence of an algorithm such that $K_a(K_b(m)) = m$ for encryption/decryption keys K_b and K_a
- instead of having Alice and Bob sharing the same key (as in symmetric key crypto), let them have two keys, where:
  - one the keys is a **public** key that is available to *everyone* in the world!
  - the other key is **private** to the receiver (i.e. Bob)



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 algorithm

- designed by **R**ivest, **S**hamir and **A**dleman
- based on mathematical operations over very large numbers
- ciphertext is surprisingly easy to calculate and very difficult to break even if the key is known
- the idea here is to have the encryption algorithm known, and the key is known

Prerequisite: modular arithmetic

- x mod 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:
  $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$
  $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$

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 algorithm

RSA Algorithm – Encryption

- Illustrative example: Assume messages have only uppercase characters

1. Assign simple code to each character, for example, 1 to 26

2. Choose *p* & *q* prime numbers ➔ *n* = p * q
   Both p & q are secrets and known to the receiver
   For example n = 11 * 7

3. Find a number *k* that is relatively prime to (p -1) * (q -1), in this example 60
   This k is the encryption key. In this example, k can be 7

4. Divide the message into components; each with many characters to avoid repetition. In this example however assume each component has one character.
   For example, if the message is "HELLO" ➔ Component are H, E, L, L & O

5. Concatenate the binary codes of each character in a component and find the integer value of the result.
   In our example, the integers of the components will be: 8, 5, 12, 12 & 15

RSA algorithm

- Illustrative example (Continues …)

6. Encrypt the message by raising each number to the power of *k* then modulo *n*.

   in our example, that is:

   8^7 modulo 77; 5^7 modulo 77; 12^7 modulo 77; 12^7 modulo 77; 15^7 modulo 77

   The results compose the encrypted message

   in our example, the encrypted message 57, 47, 12, 12, 71

➔ Now when the receiver gets this encrypted message, how can it decrypt it?

RSA algorithm

RSA Algorithm – *Decryption*

- Illustrative example (Continues …)
  - How can the receiver decrypt the message?

1. Find a value k’ such that

   [(k * k’) – 1] modulo [(p – 1) * (q – 1)] = 0

   in other words, (k * k’) – 1 is evenly divisible by (p – 1) * (q – 1)

➔ The value of *k’* is the decryption key

In our example, k’ can be 43 since (43 * 7) – 1 = 300 divides 60

2. Raise each number of the encrypted message by k’ then do modulo n

   In our example, that will be:

   57^{43} modulo 77; 47^{43} modulo 77; 12^{43} modulo 77;
   12^{43} modulo 77; 71^{43} modulo 77

   ➔ that results in 8, 5, 12, 12 and 15, which are the original numbers

RSA: Summery

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 k (with $k<n$) that has no common factors with z (k, z are "relatively prime").
4. choose k' such that $k.k'-1$ is exactly divisible by z. (in other words: $k.k' \bmod z = 1$).
5. *public* key is $\underbrace{(n,k)}_{K_B^+}$. *private* key is $\underbrace{(n,k')}_{K_B^-}$.

RSA: Summery

encryption, decryption

0. given (n,k) and (nk') as computed above

1. to encrypt message m ($<n$), compute

$$c = m^k \bmod n$$

2. to decrypt received bit pattern, c, compute

$$m = c^{K'} \bmod n$$

magic happens!

$$m = (\underbrace{m^k \bmod n}_{c})^{k'} \bmod n$$

RSA - example2:

Bob chooses *p=5, q=7*. Then *n=35, z=24*.

k=5 (so *e, z* relatively prime).

k'=29 (so *k.k'-1* exactly divisible by z).

encrypting 8-bit messages.

encrypt: $\underbrace{\text{bit pattern}}_{\text{0000l000}}$ $\underbrace{m}_{12}$ $\underbrace{m^k}_{24832}$ $\underbrace{c = m^k \bmod n}_{17}$

decrypt: $\underbrace{c}_{17}$ $\underbrace{c^{k'}}_{481968572106750915091411825223071697}$ $\underbrace{m = c^{K'} \bmod n}_{12}$

Why does RSA work?

- must show that $c^{k'} \bmod n = m$
  where $c = m^k \bmod n$
- 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,
  $$c^{k'} \bmod n = (m^k \bmod n)^{k'} \bmod n$$
  $$= m^{k.k'} \bmod n$$
  $$= m^{(k.k' \bmod z)} \bmod n$$
  $$= m^1 \bmod n$$
  $$= m$$

How secure is RSA?

- Encryption algorithm requires k & n
- Decryption requires k’ & n
- Interception of a message would reveal both k & n

➔ So, the question is how easy can *k’* be calculated/obtained?
K’ is chosen based on:
$[(k * k') - 1] \text{ modulo } [(p - 1) * (q - 1)] = 0$

➔ If p & q are guessed then k’ is obtained

➔ $n = p * q$, and n is known!

➔ It does not look so difficult then; does it?

➔ p & q are very big numbers that n is usually more than 200 digits

➔ **It is very difficult to guess (factor in fact) p & q from n**

➔ **Factoring an RSA 2048-bit number has been worth 200,000$ prize** **(as of Sept 12, 2007)**

Click here to see RSA Challeng Numbers

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:

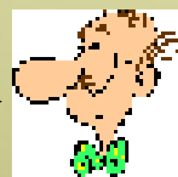
$$
\begin{aligned}
(m^e \bmod n)^d \bmod n &= m^{ed} \bmod n \\
&= m^{de} \bmod n \\
&= (m^d \bmod n)^e \bmod n
\end{aligned}
$$

Authentication

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

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

“I am Alice” →

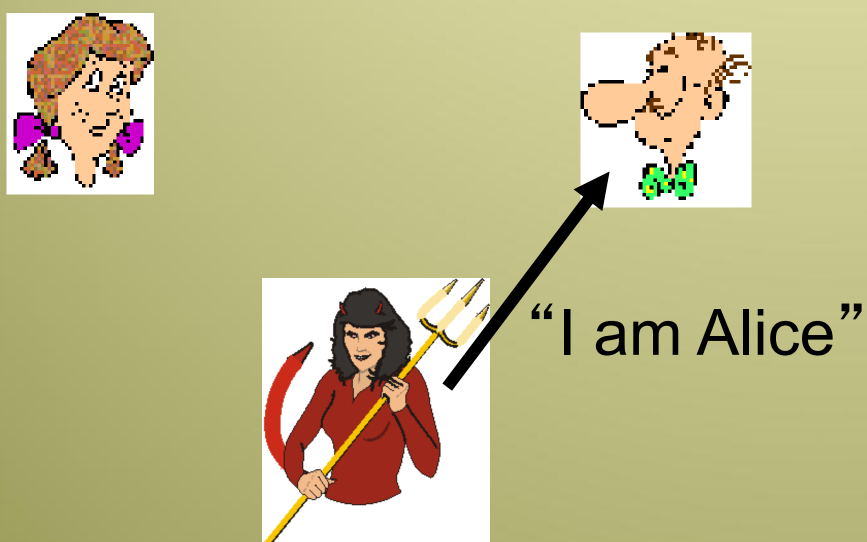


Failure scenario??

Authentication

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

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

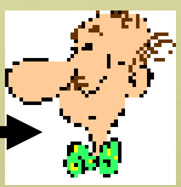


in a network, Bob can not “see” Alice, so Trudy simply declares herself to be Alice

Authentication: another try

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

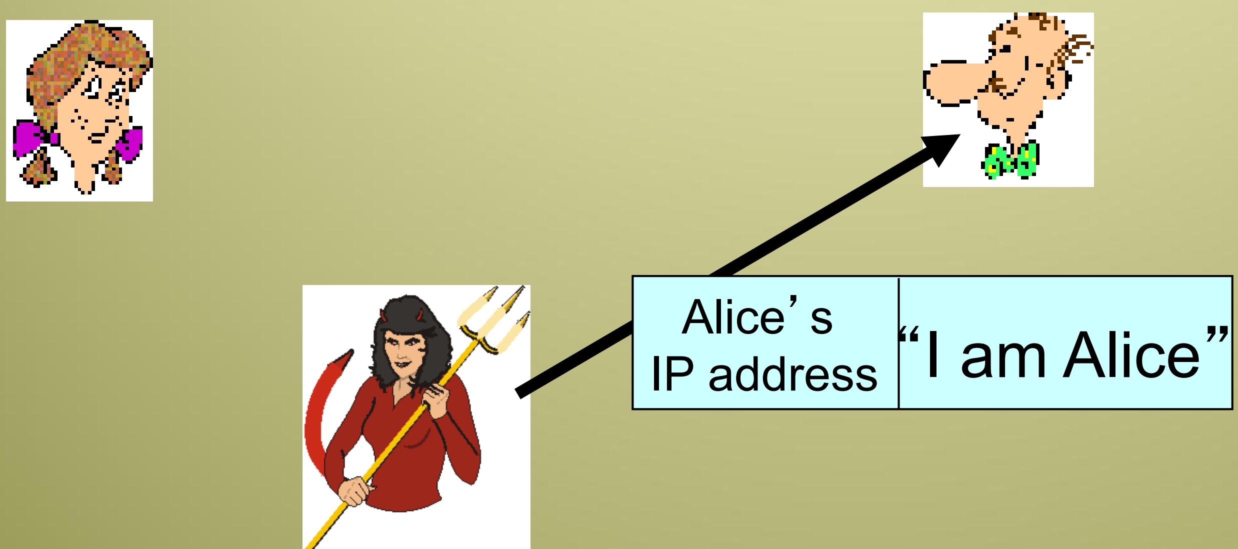
Alice's IP address	"I am Alice"



Failure scenario??

Authentication: another try

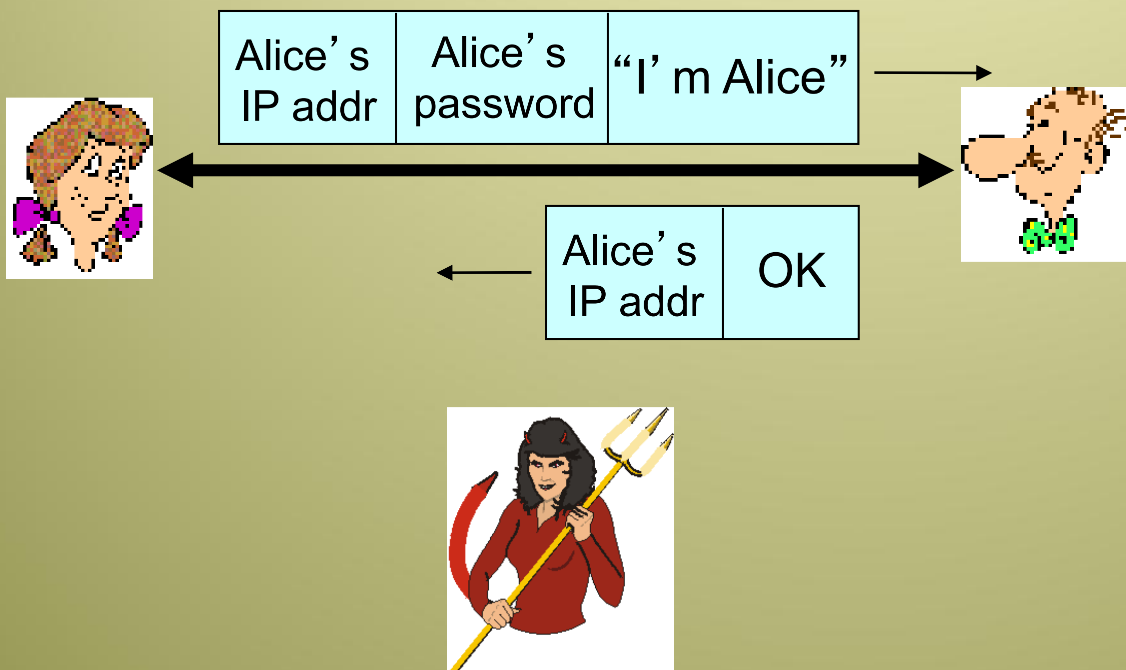
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



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

Authentication: another try

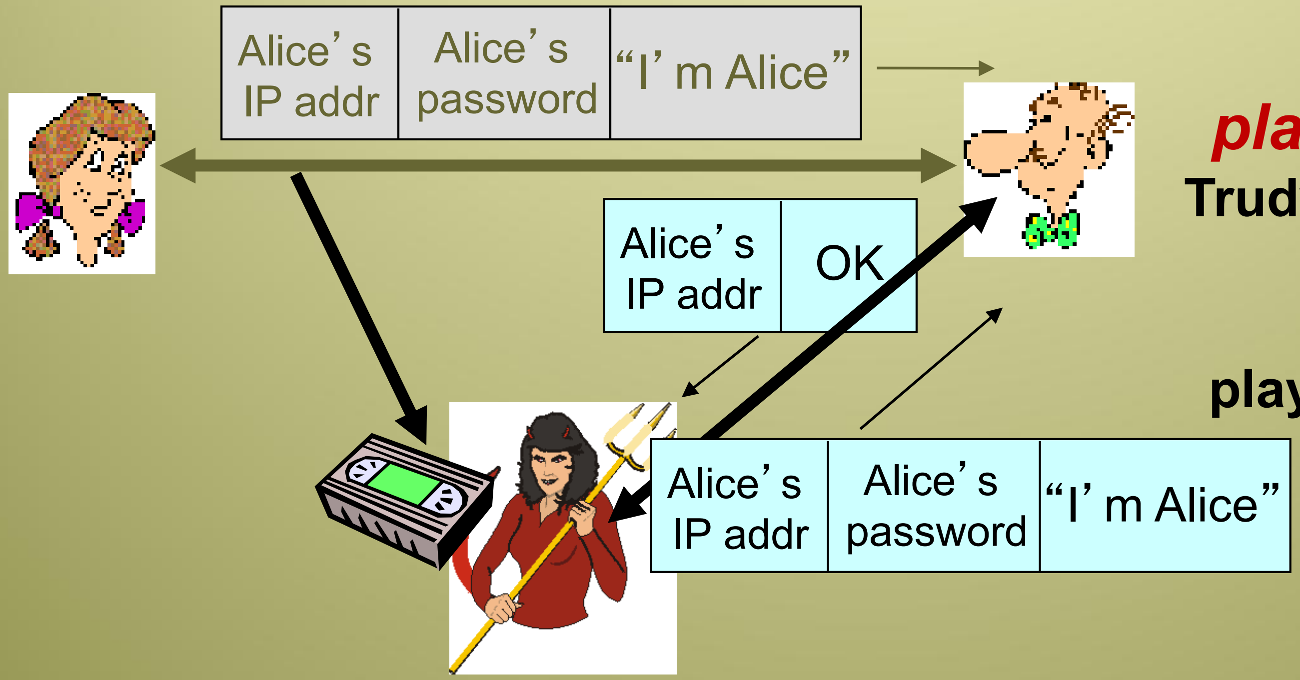
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Failure scenario??

Authentication: another try

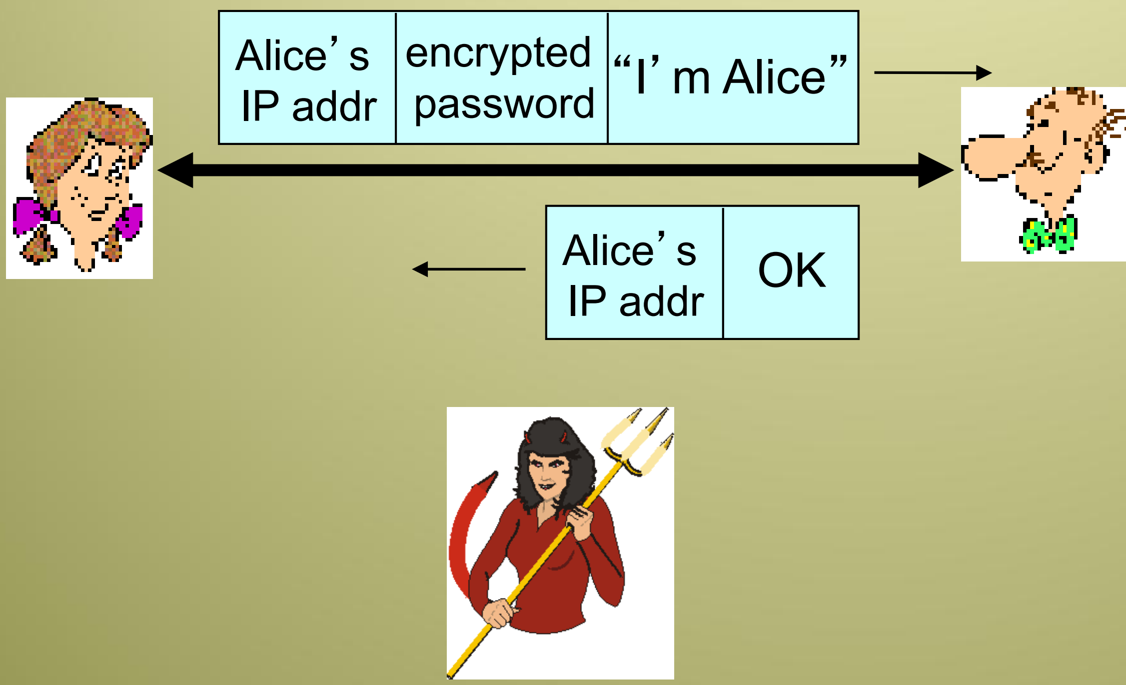
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



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

Authentication: yet another try

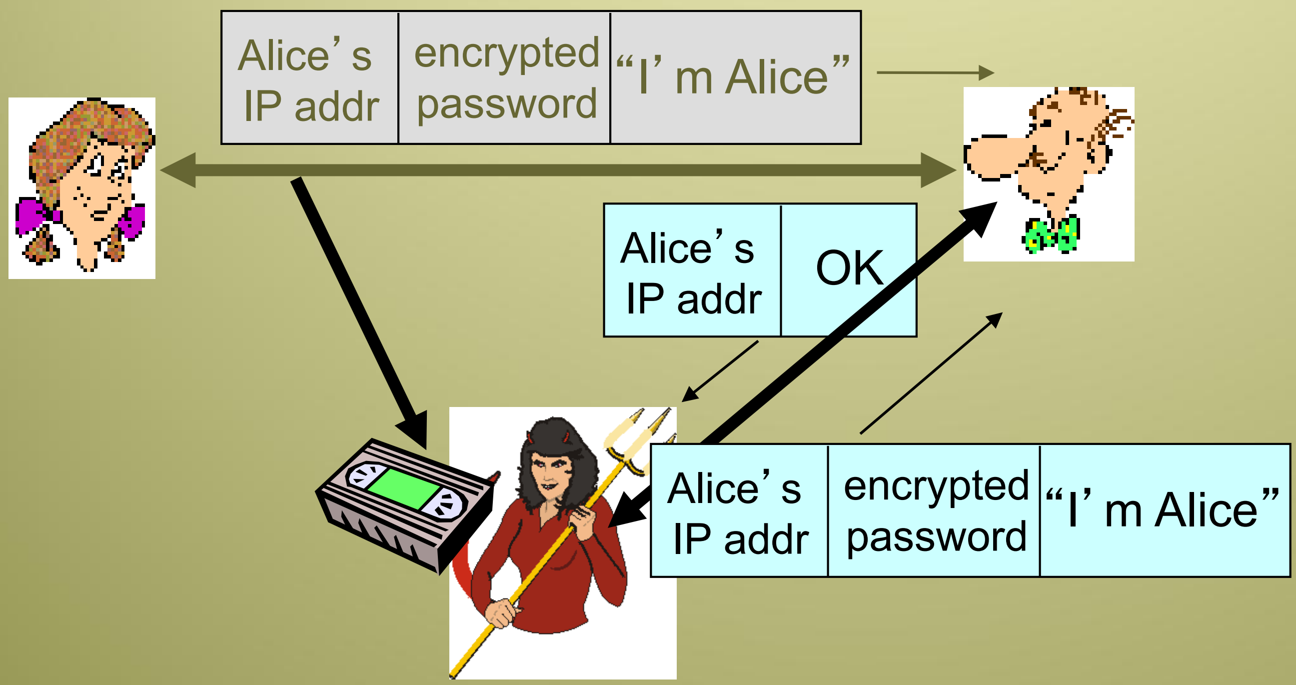
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: yet another try

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



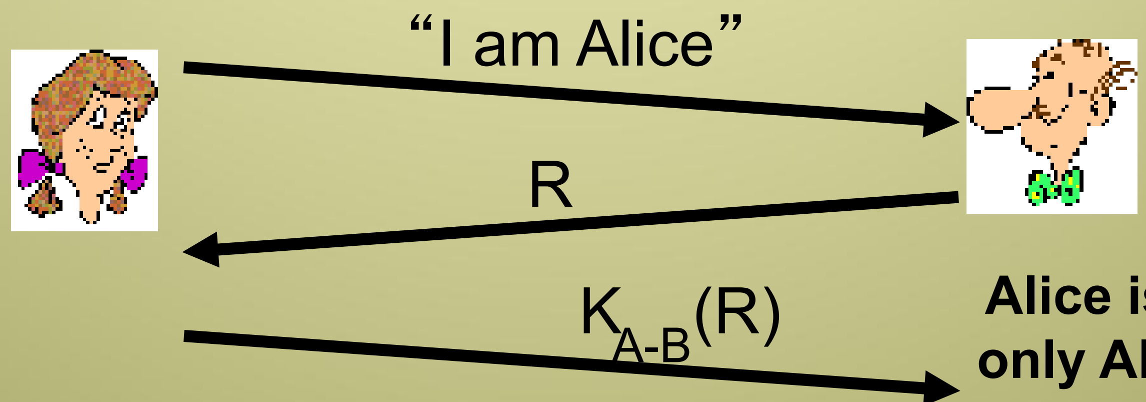
record and playback *still* works!

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 (symmetric) secret key



Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

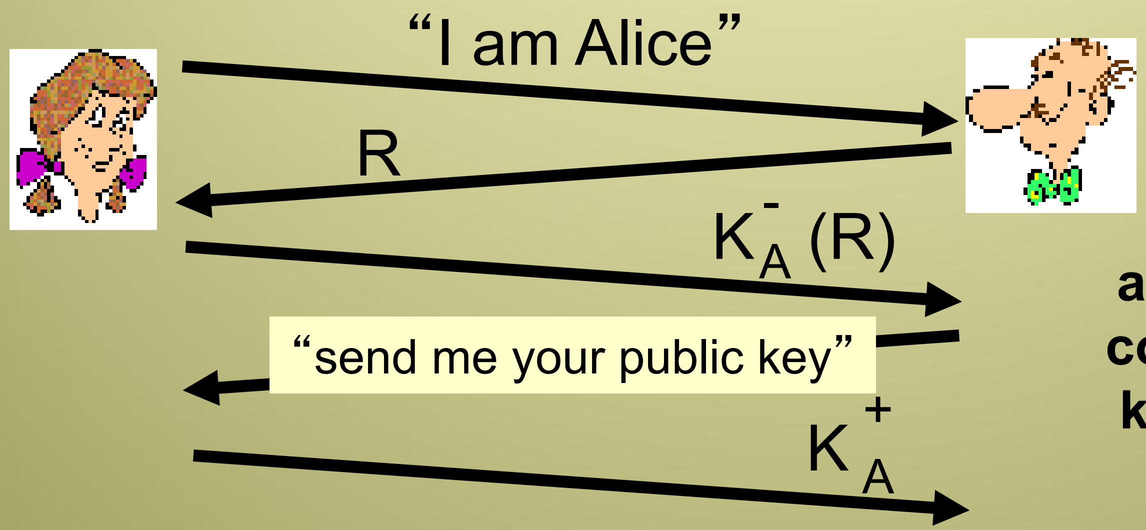
Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



Bob computes

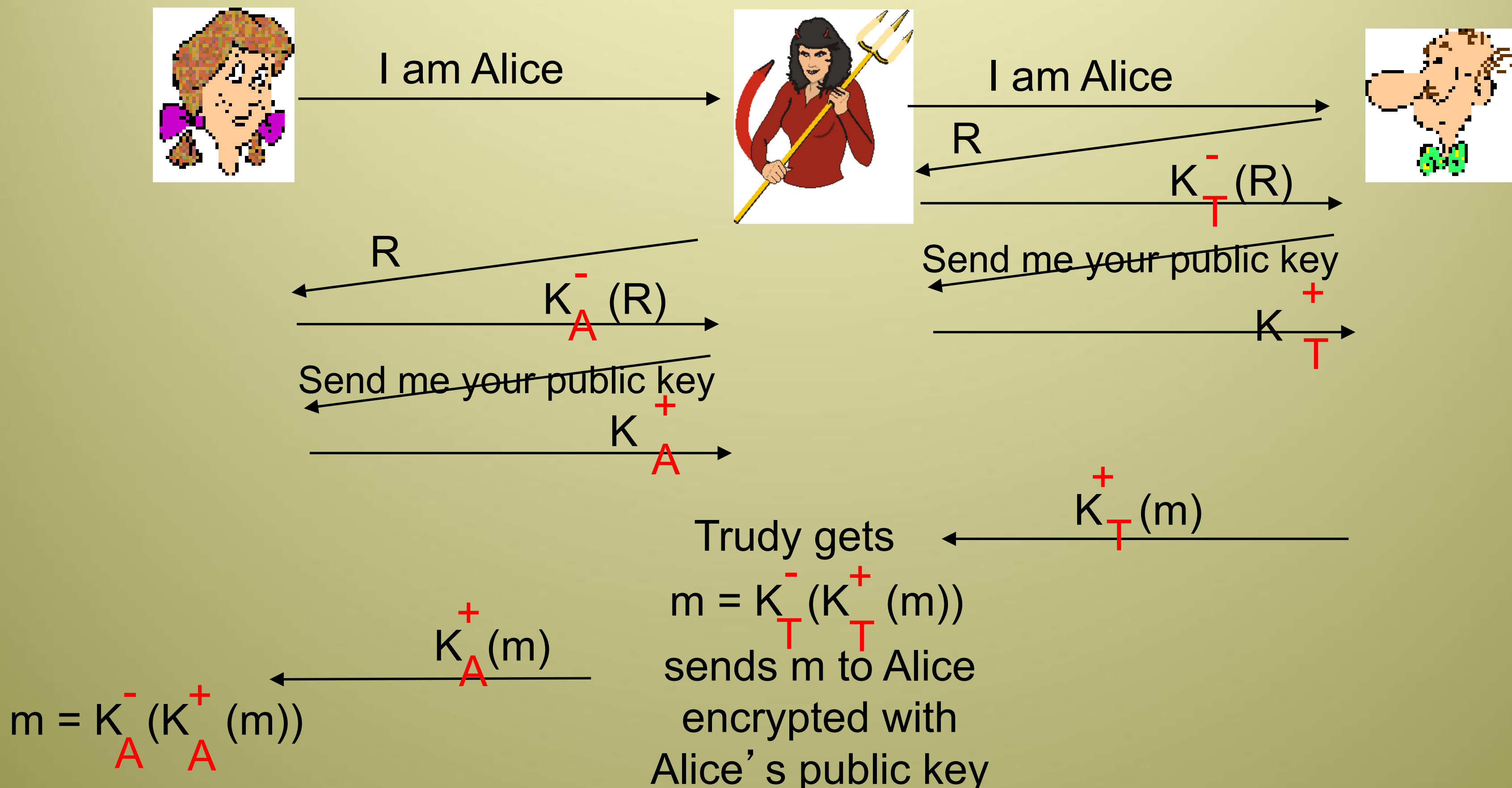
$$K_A^+(K_A^-(R)) = R$$

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

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

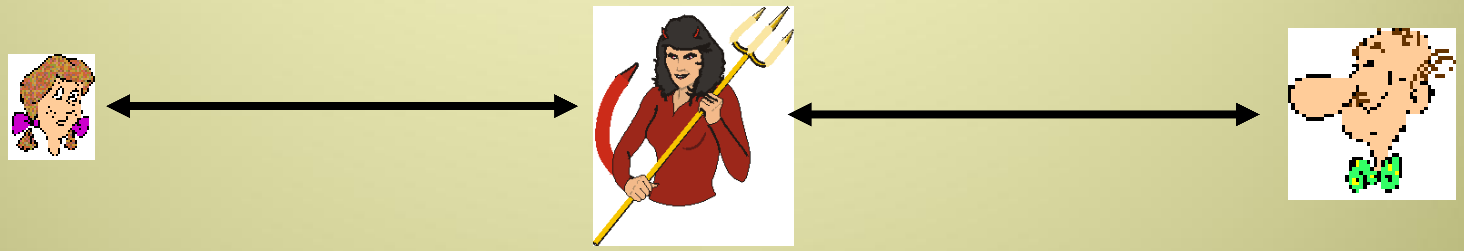
ap5.0: security hole

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



ap5.0: security hole

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

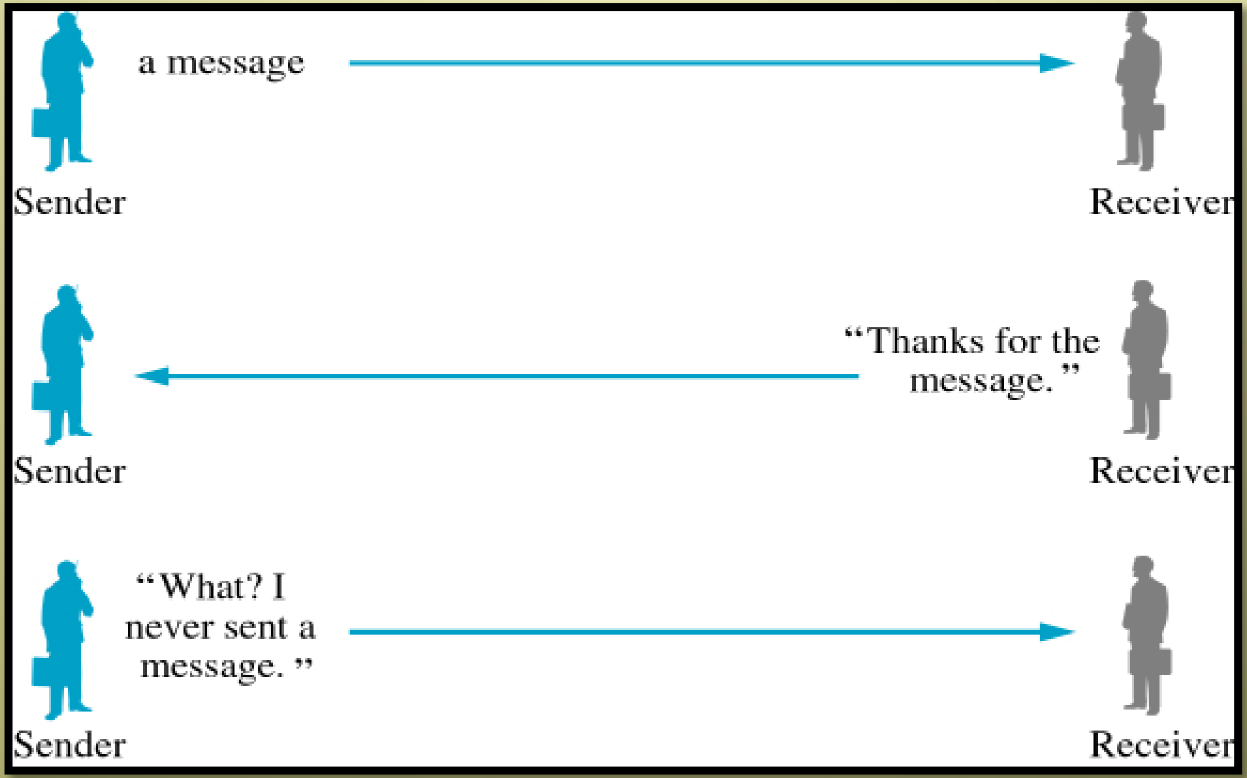


difficult to detect:

- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation!)
- problem is that Trudy receives all messages as well!

Digital signatures

- used for authentication purposes

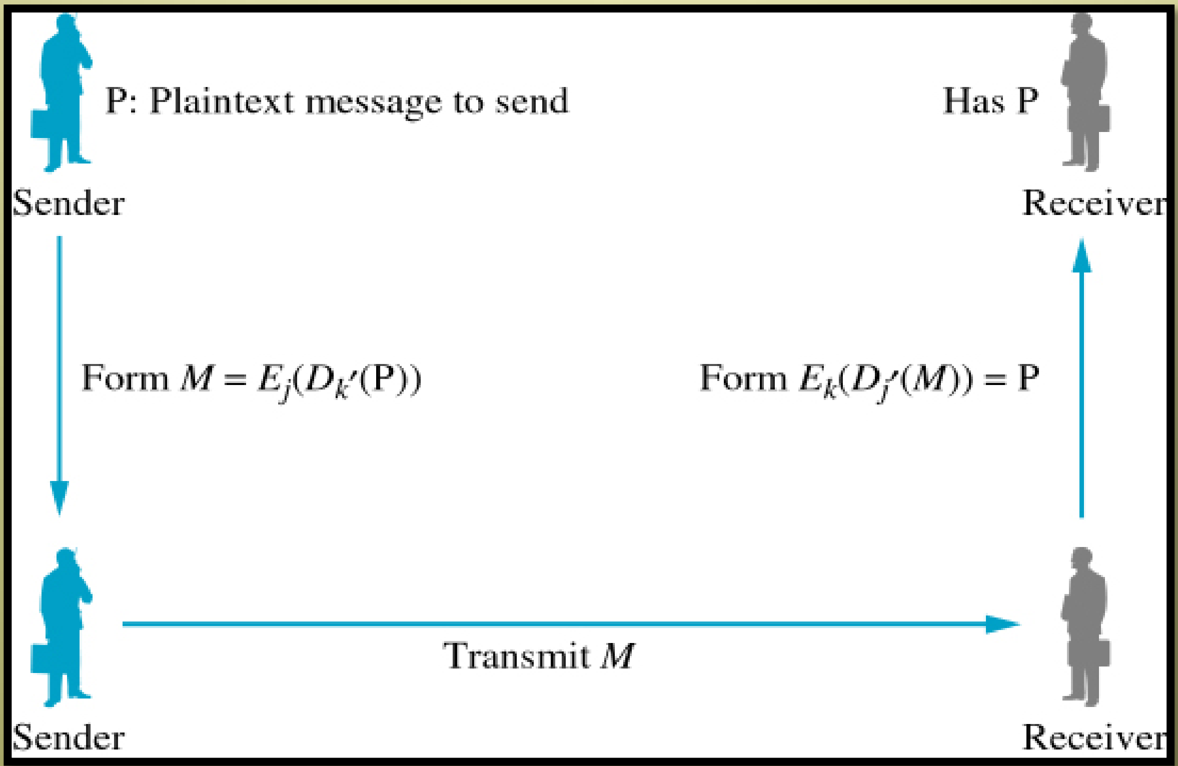


Sender Denying Sending a Message

Digital signatures

- the sender has an encryption key *k'* and the receiver has a decryption key *j'*; both k' and j' are private keys while k & j are public keys

- the sender is hence the only one that can sends an authenticated message



Sending a Message Using a Digital Signature

Digital signatures

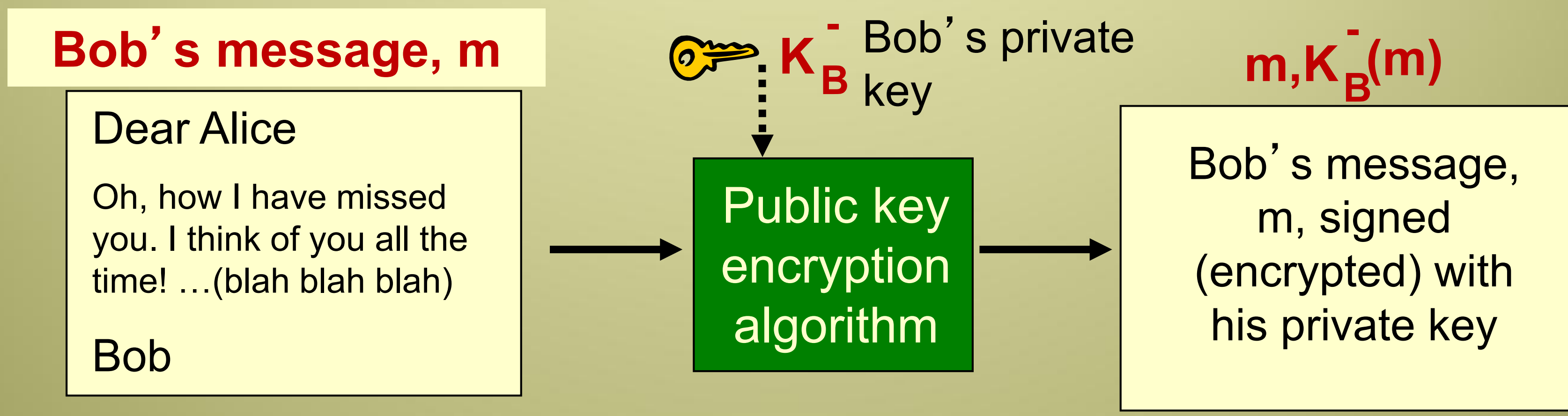
cryptographic technique analogous to hand-written signatures:

- 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.

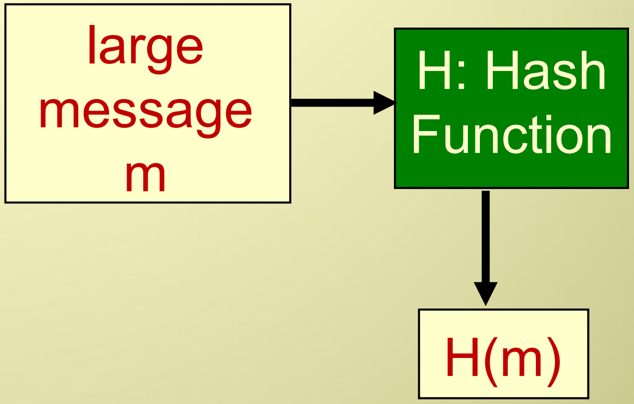
Alice thus verifies that:

- ✓ 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"

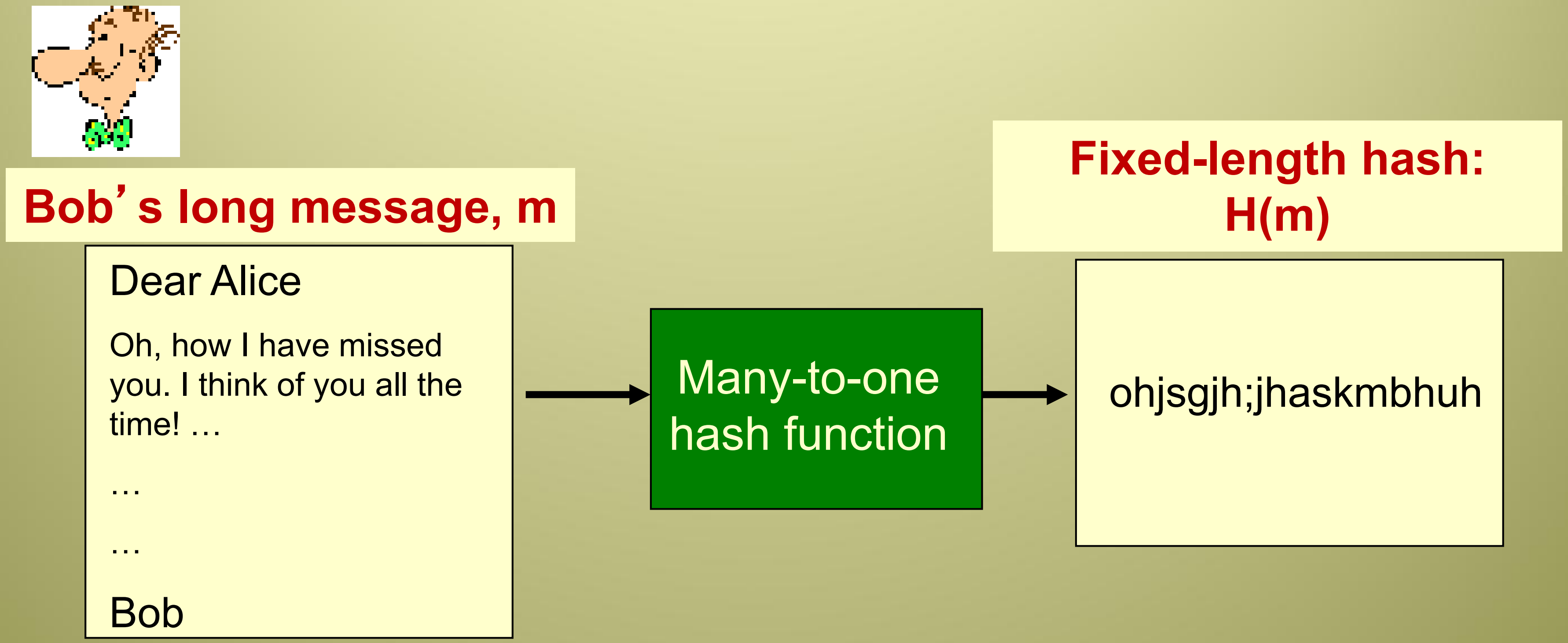
- apply hash function H to *m*, get fixed size message digest, *H(m)*.

Hash function properties:

- many-to-1
- produces fixed-size msg digest (fingerprint)
- given two different messages x and y, it is computationally infeasible that H(x) = H(y)

Digital signatures – Hash functions

- the hash function used, also referred to as **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!

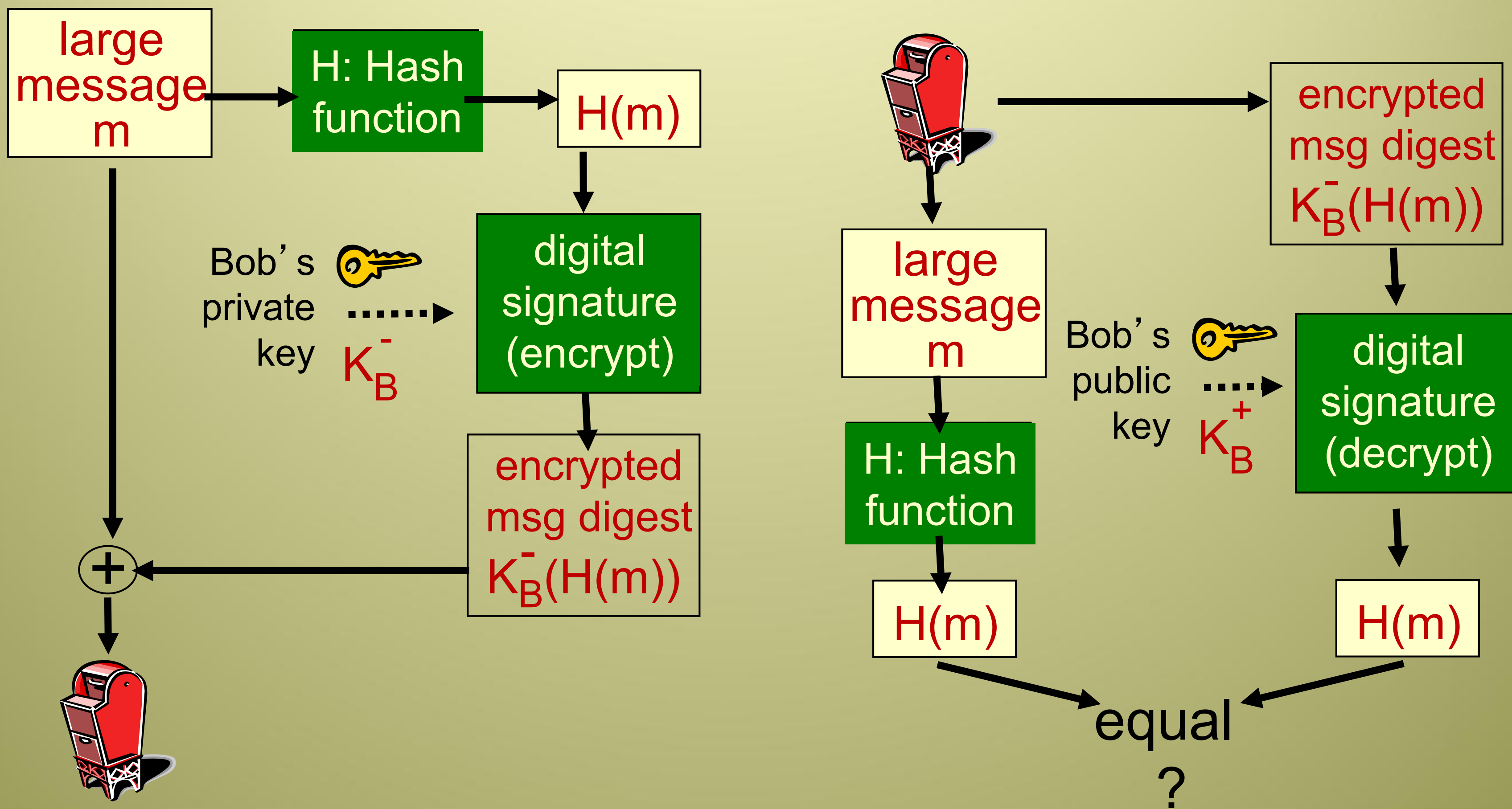
Hash function algorithms

- major existing hash functions include:
  - **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** ***- Secure Hash Algorithm***
    - US standard [NIST, FIPS PUB 180-1]
    - 160-bit message digest
    - the longer output length makes SHA-1 more secure

Digital signature = signed message digest

Bob sends digitally signed message:

Alice verifies signature, integrity of digitally signed message:

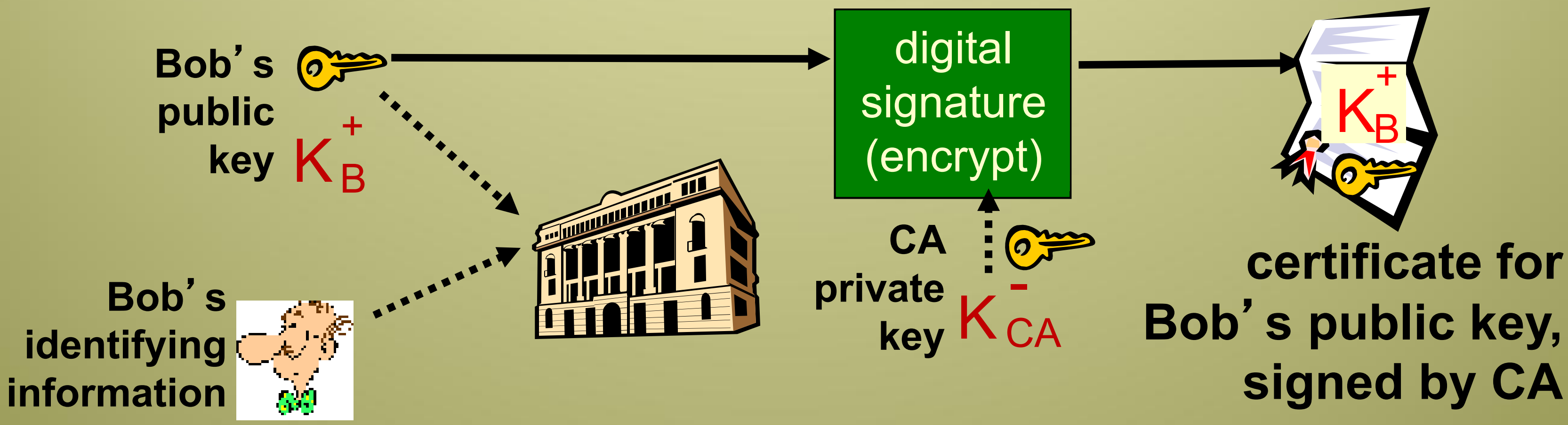


Public-key certification

- an important application to digital signature is public-key certification:
  - certifies that a public-key indeed belongs to a specific entity
- 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!

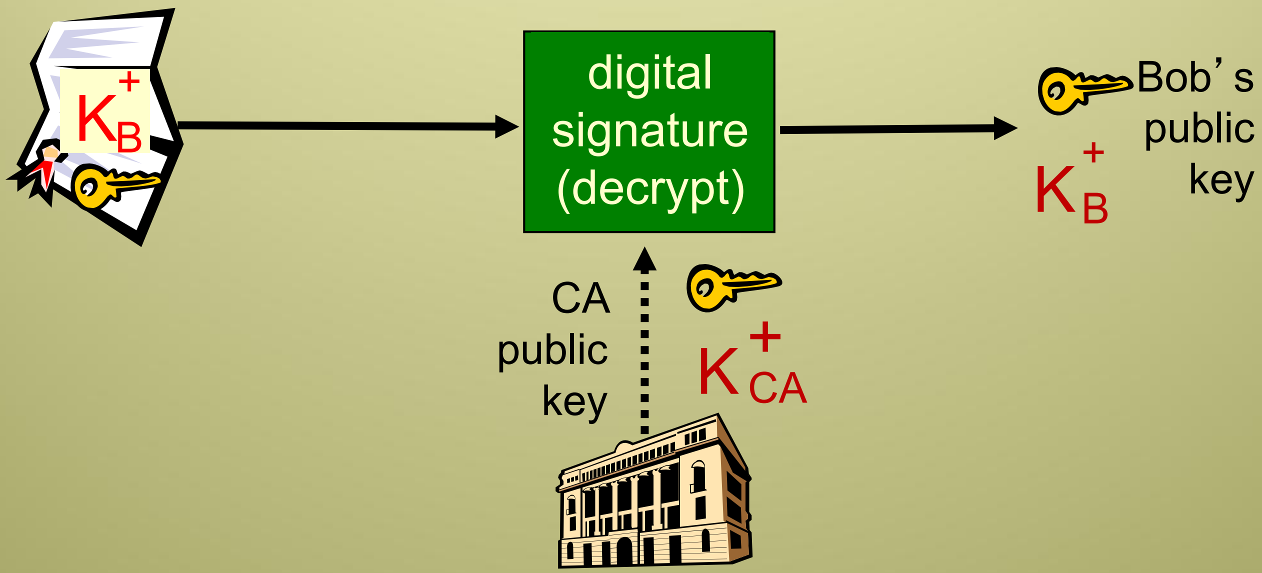
Certification authorities

- *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"
  - the certificate itself is digitally signed by the CA



Certification authorities

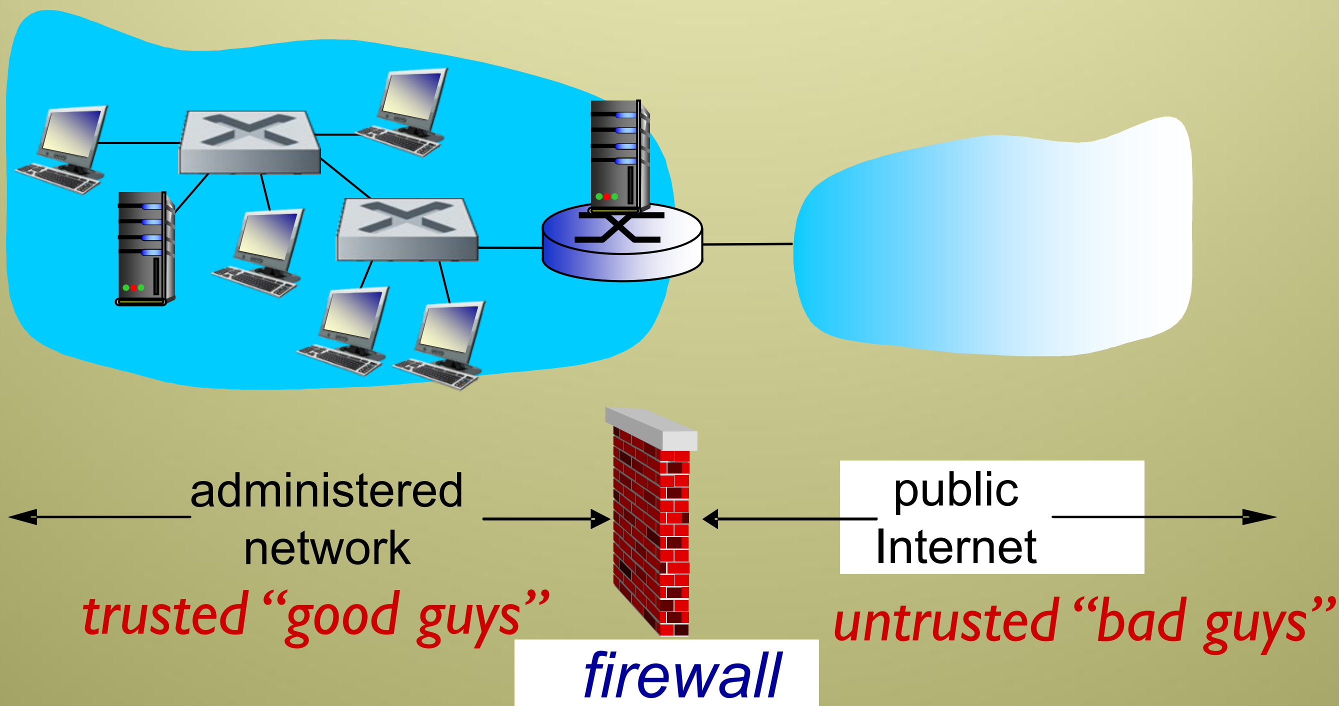
- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key



Firewalls

firewall

isolates organization's internal net from larger Internet, allowing some packets to pass, blocking others



Firewalls: why

prevent denial of service attacks:

- SYN flooding: attacker establishes many bogus TCP connections, no resources left for "real" connections

prevent illegal modification/access of internal data

- e.g., attacker replaces CIA's homepage with something else

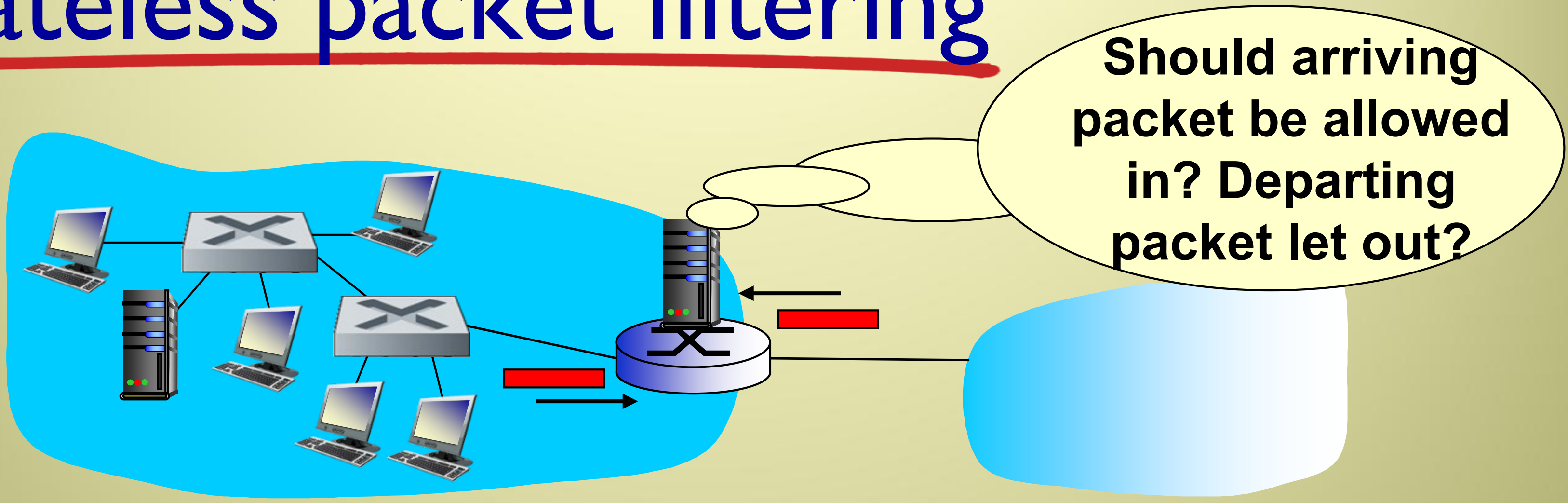
allow only authorized traffic to inside network

- set of authenticated users/hosts

three types of firewalls:

- stateless packet filters
- stateful packet filters
- application gateways

Stateless packet filtering



- internal network connected to the ISP (and hence the Internet) via a gateway router *(router firewall)*
- router *filters packet-by-packet,* decision to forward/drop packet based on:
  - source IP address, destination IP address
  - TCP/UDP source and destination port numbers
  - TCP flag bits: SYN, ACK, …
  - other rules for datagram leaving/entering
  - …

Stateless packet filtering: example

- *example 1:* block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23
  - *result:* all incoming, outgoing UDP flows and telnet connections are blocked

- *example 2:* block inbound TCP segments with ACK=0.
  - *result:* prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside.

Stateless packet filtering: more examples

Policy	*Firewall Setting*
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for institution's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets - except DNS and router broadcasts.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP packets going to a "broadcast" address (e.g. 130.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

Access Control Lists

- *ACL:* table of rules, applied top to bottom to incoming packets: (action, condition) pairs

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----
deny	all	all	all	all	all	all

Stateful packet filtering

- *stateless packet filter:* heavy handed tool
  - admits packets that "make no sense," e.g., dest port = 80, ACK bit set, even though no TCP connection established:

action	source address	dest address	protocol	source port	dest port	flag bit
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK

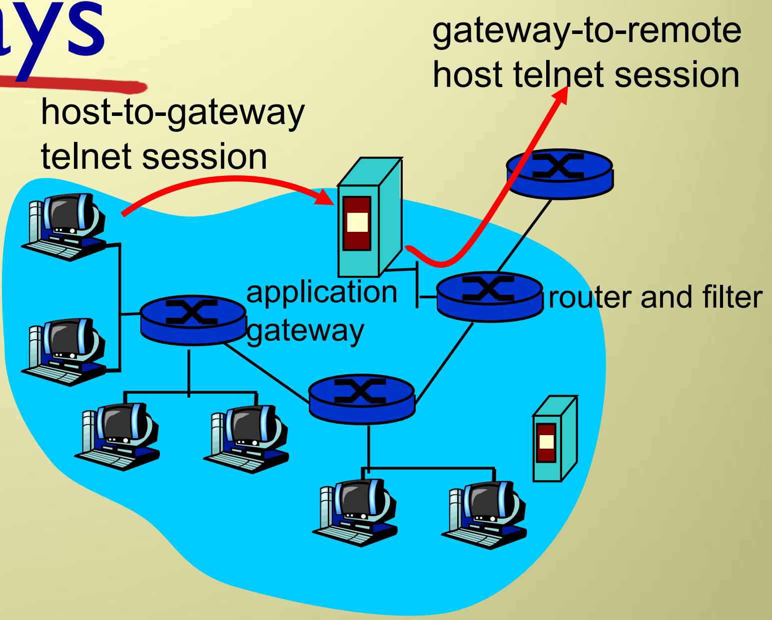
- *stateful packet filter:* track status of every TCP connection
  - track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets "makes sense"
  - timeout inactive connections at firewall: no longer admit packets

Stateful packet filtering

- ACL augmented to indicate need to check connection state table before admitting packet

action	source address	dest address	proto	source port	dest port	flag bit	check conxion
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----	X
deny	all	all	all	all	all	all	

Application gateways



- filters packets on application data as well as on IP/TCP/UDP fields.
- *example:* allow select internal users to telnet outside.

1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host. Gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway.

Limitations of firewalls, gateways

- *IP spoofing:* router can't know if data "really" comes from claimed source
- if multiple app's. need special treatment, each has own app. Gateway
- client software must know how to contact gateway.
  - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for UDP
- *tradeoff:* degree of communication with outside world, level of security
- many highly protected sites still suffer from attacks

Intrusion detection systems

- packet filtering:
  - operates on TCP/IP headers only
  - no correlation check among sessions
- *IDS: intrusion detection system*
  - *deep packet inspection:* look at packet contents (e.g., check character strings in packet against database of known virus, attack strings)
  - examine correlation among multiple packets
    - port scanning, network mapping, DoS attack. …
- IDS systems can broadly be classified as:
  - **Signature-based** systems
  - **Anomaly-based** systems

Intrusion detection systems

- multiple IDSs: different types of checking at different locations for performance reasons
- allows filtering at a further downstream location, hence targeting only part of the traffic