Sécurité

Computer Networking: A Top Down Approach , 5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.

Chapter 8: Network Security

Chapter goals:

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

Chapter 8 roadmap

What is network security?

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

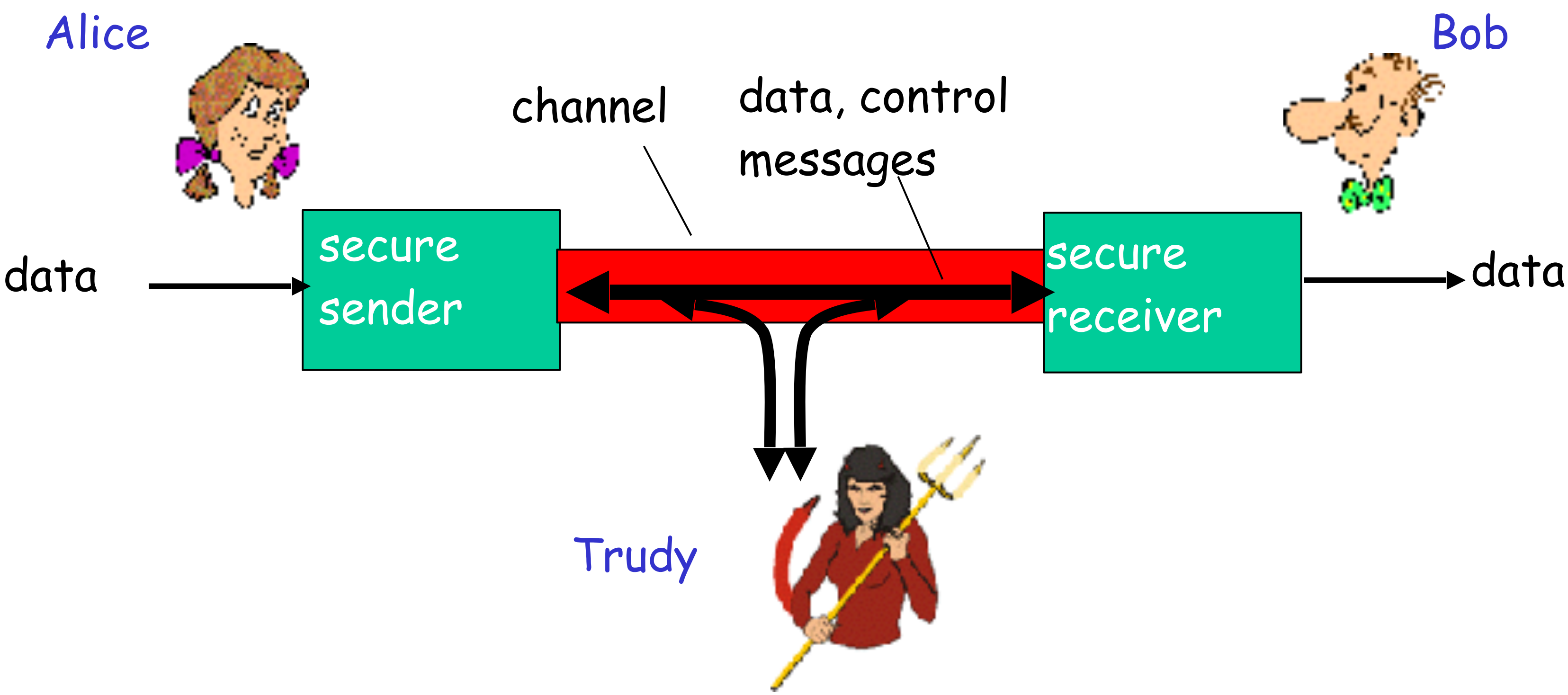
Authentication: sender, receiver want to confirm identity of each other

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
- other examples?

There are bad guys (and girls) out there!

Q: What can a "bad guy" do?

- 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)

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity

8.4 Securing e-mail

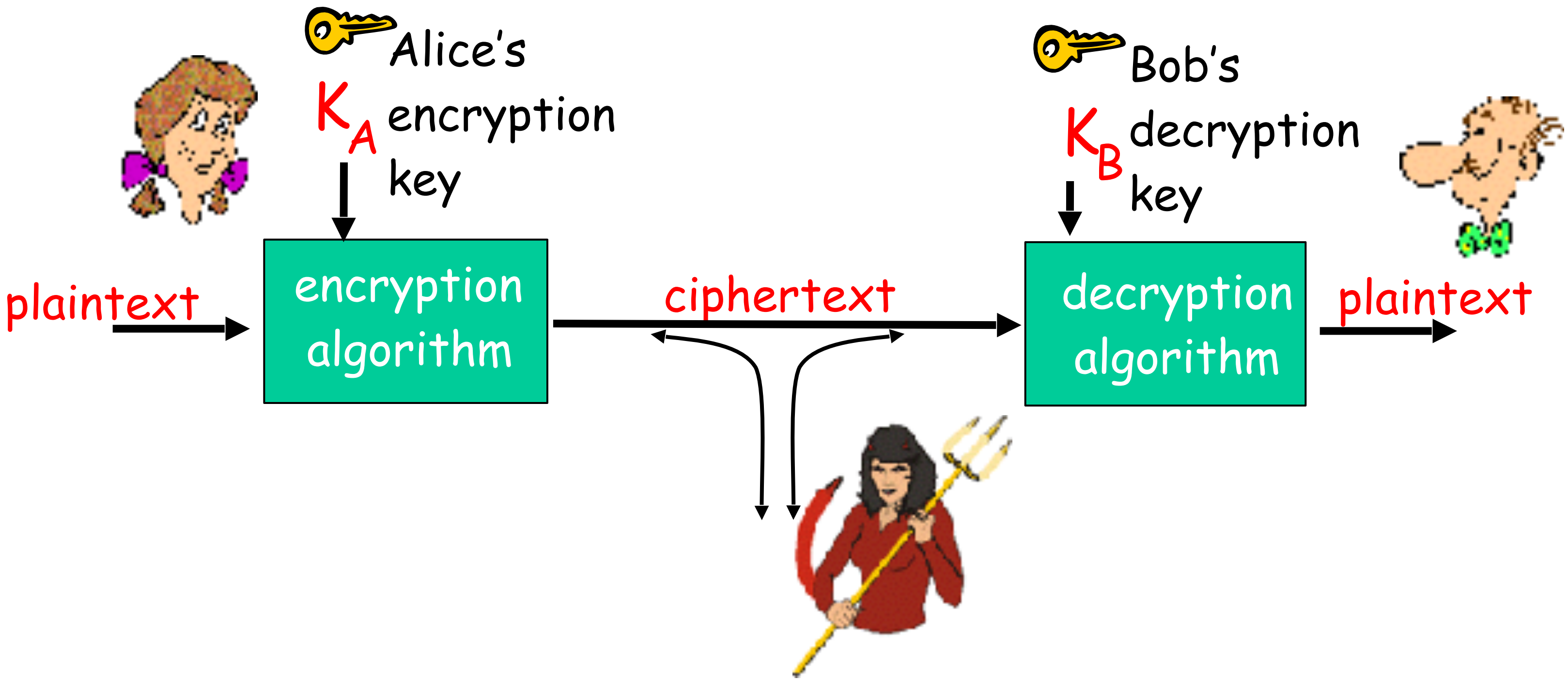
8.5 Securing TCP connections: SSL

8.6 Network layer security: IPsec

8.7 Securing wireless LANs

8.8 Operational security: firewalls and IDS

The language of cryptography



m plaintext message

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

$m = K_B(K_A(m))$

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
```

Key: the mapping from the set of 26 letters to the set of 26 letters

Polyalphabetic encryption

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

Breaking an encryption scheme

- Cipher-text only attack: Trudy has ciphertext that she can analyze
- Two approaches:

must be able to differentiate resulting plaintext from gibberish

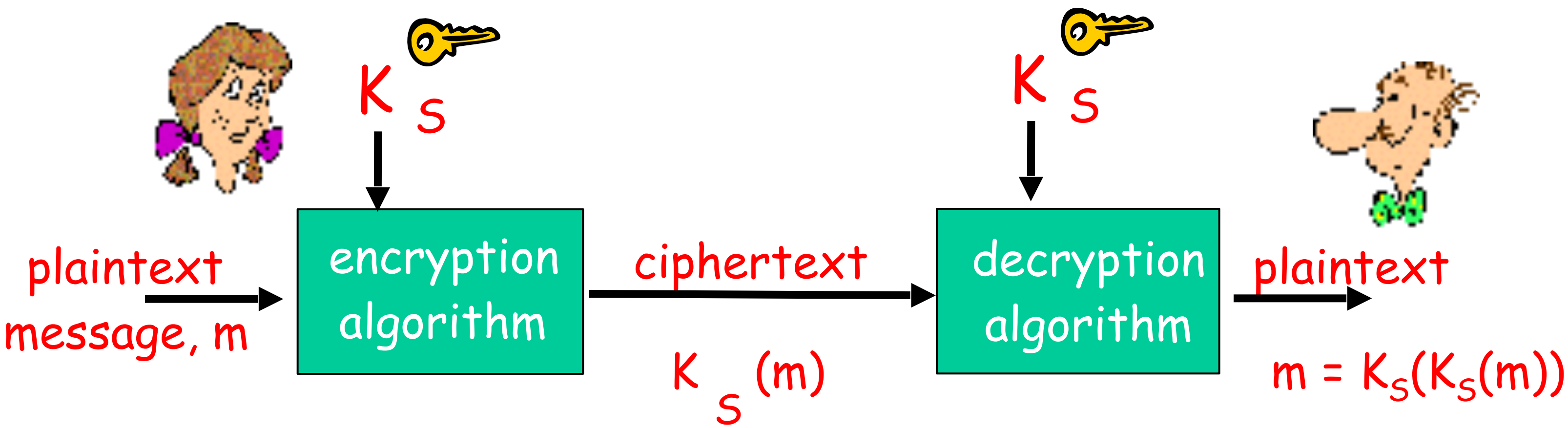
- Search through all keys
- Statistical analysis

- Known-plaintext attack: trudy has some plaintext corresponding to some ciphertext
  - eg, in monoalphabetic cipher, trudy determines pairings for a,l,i,c,e,b,o,b.
- Chosen-plaintext attack: trudy can get the cyphertext for some chosen plaintext

Types of Cryptography

- Crypto often uses keys:
  - Algorithm is known to everyone
  - Only “keys” are secret
- Public key cryptography
  - Involves the use of two keys
- Symmetric key cryptography
  - Involves the use one key
- Hash functions
  - Involves the use of no keys
  - Nothing secret: How can this be useful?

Symmetric key cryptography



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

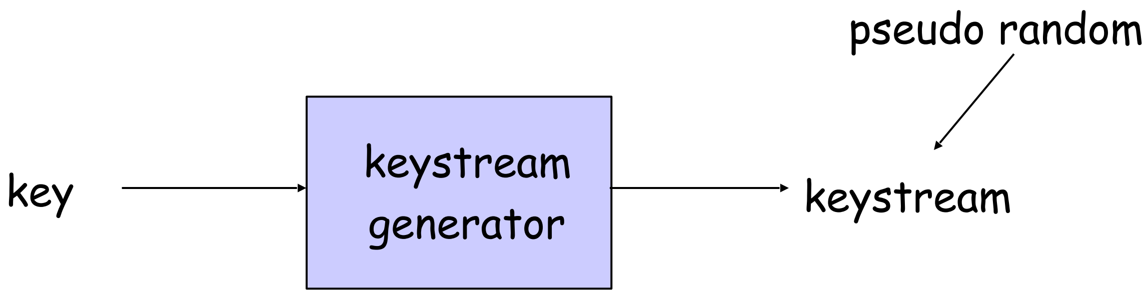
- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher

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

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

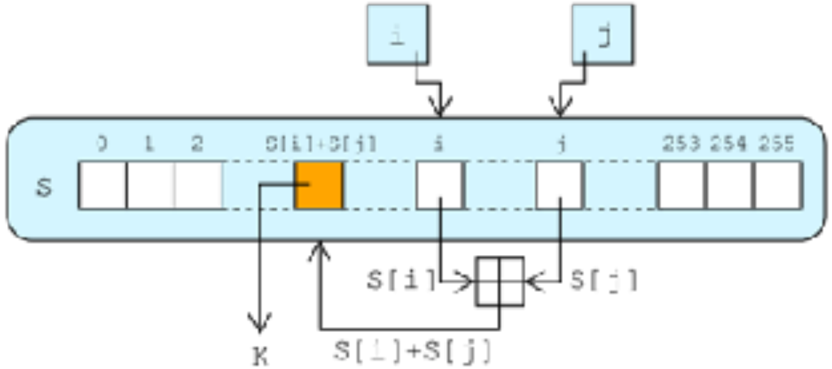


- Combine each bit of keystream with bit of plaintext to get bit of ciphertext
- m(i) = ith bit of message
- ks(i) = ith bit of keystream
- c(i) = ith bit of ciphertext
- c(i) = ks(i) ⊕ m(i) (⊕ = exclusive or)
- m(i) = ks(i) ⊕ c(i)

RC4 Stream Cipher

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

RC4



- ❐ Chiffrement RC4
  - ❍ Générateur de bit pseudo-aléatoires : le résultat est combiné avec le texte en claire
    - État interne (secret) = permutation sur 256 octets + pointeur i et j (8bits) indices dans un tableau
  - ❍ Le tableau (permutation) est construit à partir de la clé
  - ❍ Pour toujours:
    - i=i+1 mod 256
    - j=j+s[i] mod 256
    - Echanger s[i] et s[j]
    - octet codé=[(s[i]+s[j] mod 256) XOR octet

Block ciphers

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

Example with k=3:

input	output
000	110
001	111
010	101
011	100

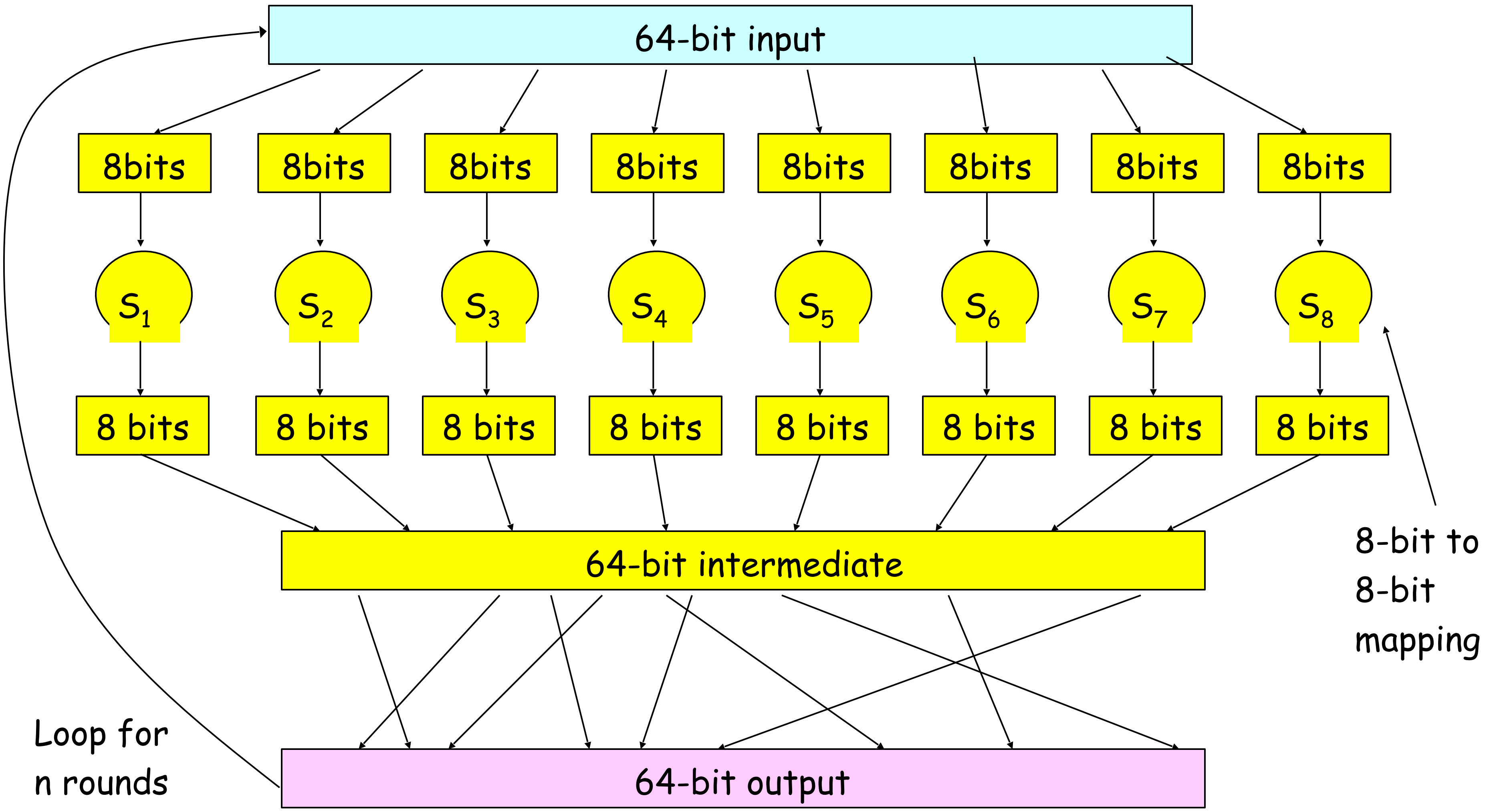
input	output
100	011
101	010
110	000
111	001

What is the ciphertext for 010110001111 ?

Block ciphers

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

Prototype function



Why rounds in prototype?

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

Encrypting a large message

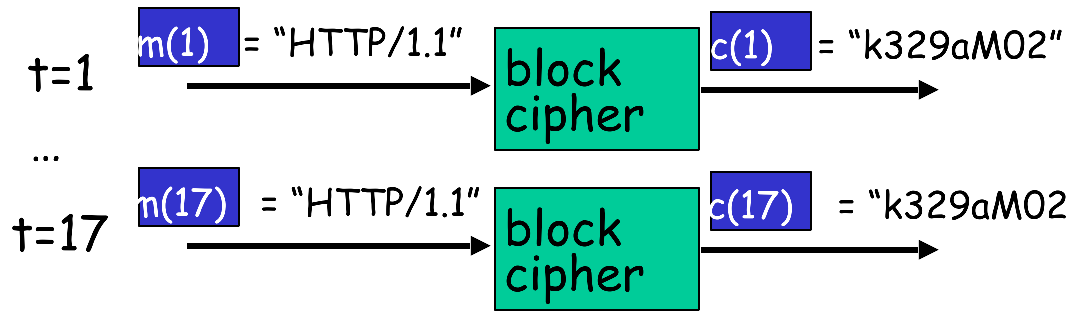
- Why not just break message in 64-bit blocks, encrypt each block separately?
  - If same block of plaintext appears twice, will give same cyphertext.
- How about:
  - Generate random 64-bit number r(i) for each plaintext block m(i)
  - Calculate $c(i) = K_S(m(i) \oplus r(i))$
  - Transmit c(i), r(i),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

Cipher Block Chaining (CBC)

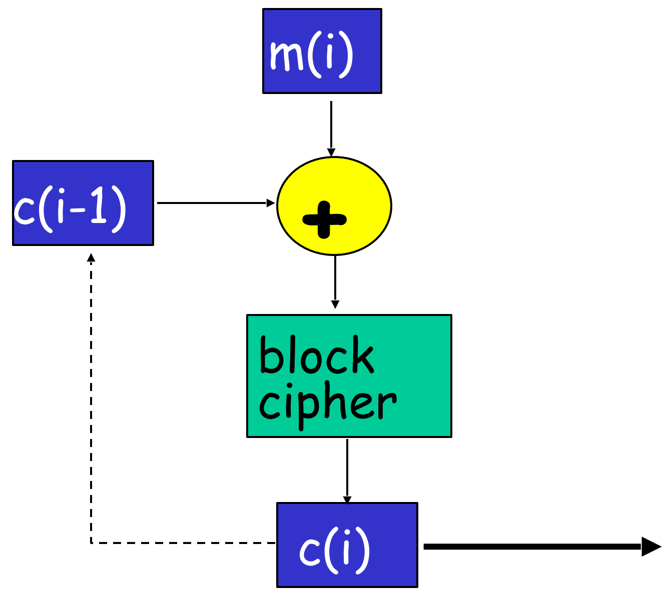
- CBC generates its own random numbers
  - Have encryption of current block depend on result of previous block
  - $c(i) = K_S(\ m(i) \oplus c(i-1)\ )$
  - $m(i) = K_S(\ c(i)) \oplus c(i-1)$
- How do we encrypt first block?
  - Initialization vector (IV): random block = c(0)
  - IV does not have to be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

Cipher Block Chaining

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



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



Symmetric key crypto: DES

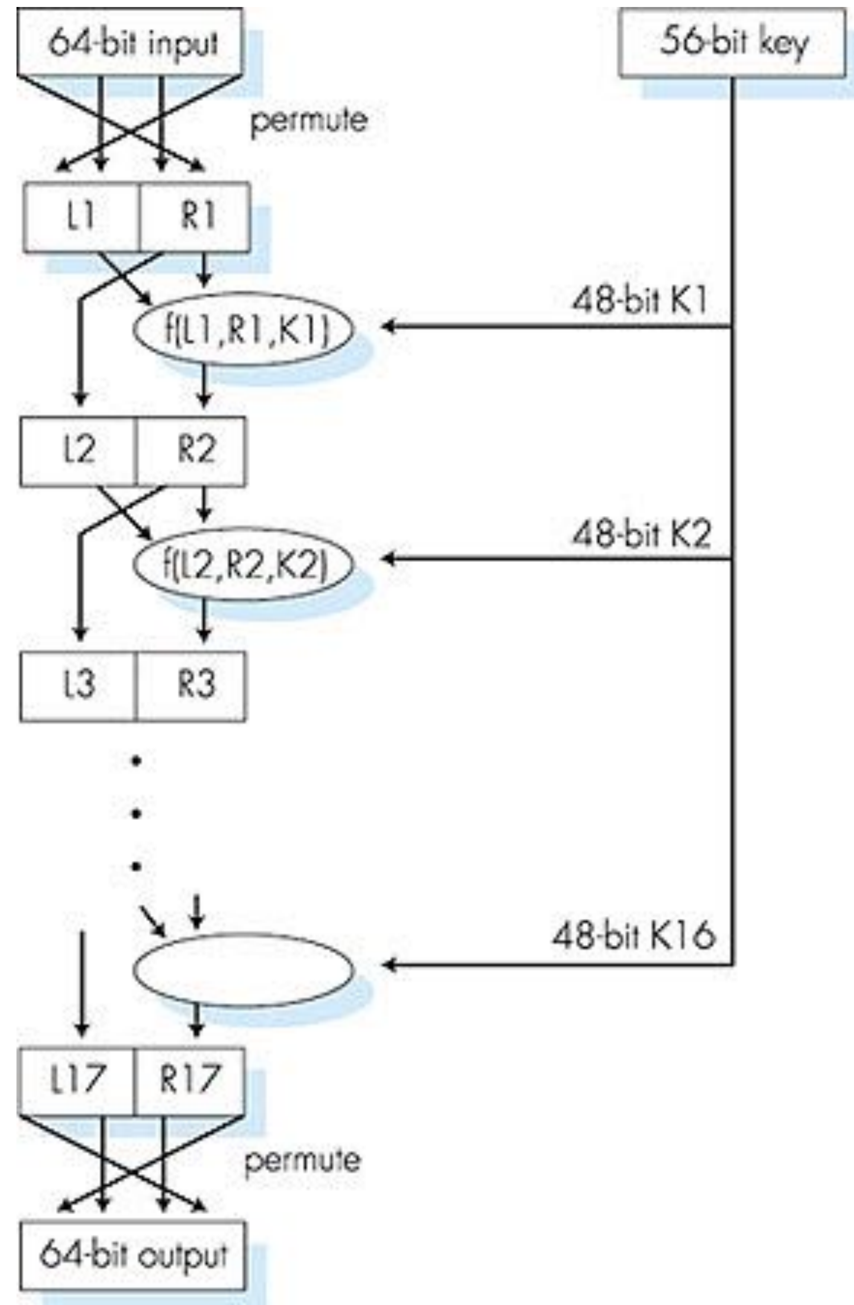
DES: Data Encryption Standard

- 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
  - No known good analytic attack
- making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys
    (actually encrypt, decrypt, encrypt)

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

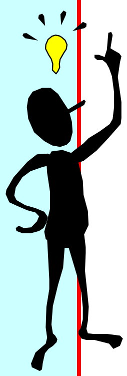
- new (Nov. 2001) symmetric-key NIST standard, replacing DES
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

Public Key Cryptography

symmetric key crypto

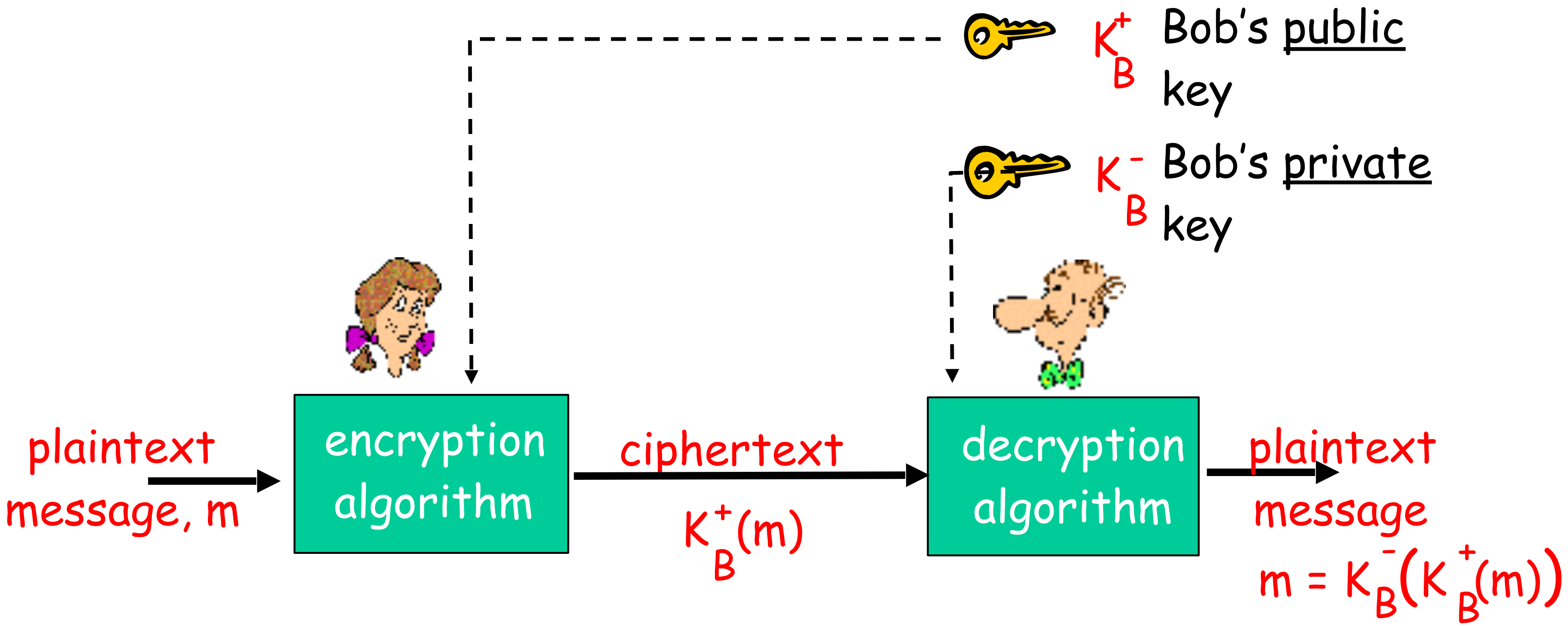
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

public key cryptography



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

Public key cryptography



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

- A message is a bit pattern.
- A bit pattern can be uniquely represented by an integer number.
- Thus encrypting a message is equivalent to encrypting a number.

Example

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

RSA: 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.

encrypt:	bit pattern	m	m^e	$c = m^e \bmod n$
	00001000	12	24832	17

decrypt:	c	c^d	$m = c^d \bmod n$
	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,

$$
\begin{aligned}
c^d \bmod n &= (m^e \bmod n)^d \bmod n \\
&= m^{ed} \bmod n \\
&= m^{(ed \bmod z)} \bmod n \\
&= m^1 \bmod n \\
&= m
\end{aligned}
$$

RSA: another important property

The following property will be very useful later:

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

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?
- Essentially need to find factors of n without knowing the two factors p and q.
- Fact: factoring a big number is hard.

Generating RSA keys

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

Session keys

- Exponentiation is computationally intensive
- DES is at least 100 times faster than RSA

Session key, K_S

- Bob and Alice use RSA to exchange a symmetric key K_S (or use Diffie Hellman)
- Once both have K_S, they use symmetric key cryptography

Diffie-Hellman

- p (un nombre premier) et g (inférieur à p aleatoire) sont publics
- Alice choisit un secret S_A (et Bob choisit un secret S_B)
- Alice rend public $T_A=g^{S_A}$ mod p (Bob rend public $T_B=g^{S_B}$ mod p)
- La clef symétrique que peut calculer Alice est $K_S=T_B^{S_A}$ mod p. Bob peut aussi la calculer $K_S=T_A^{S_B}$ mod p
  - $T_B^{S_A}$ mod p=$g^{S_A}g^{S_B}$ mod p =$T_A^{S_B}$ mod p

Diffie-Hellman

- Mais si un attaquant connait g, $T_A=g^{S_A}$ mod p et $T_B=g^{S_B}$ mod p pour calculer $K_S=g^{S_A}g^{S_B}$ mod p, il faut qu'il calcule S_A ou S_b
- Or S_A = log discret(T_A) dans le groupe cyclique d'ordre p et de générateur g
- un tel calcul est « difficile »