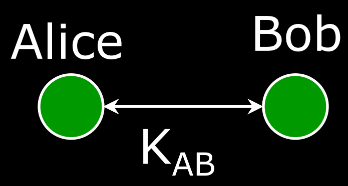
Symmetric cryptography

- Both parties must agree on a secret key, *K*
- message is encrypted, sent, decrypted at other side

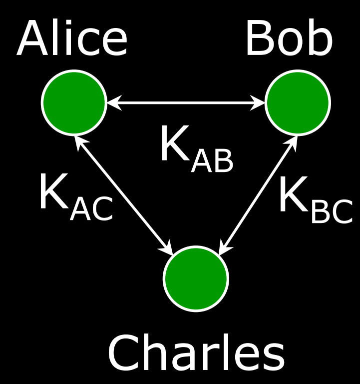
- Key distribution must be secret
  - otherwise messages can be decrypted
  - users can be impersonated

Key explosion

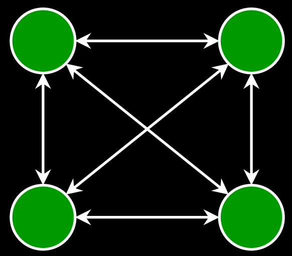
Each pair of users needs a separate key for secure communication



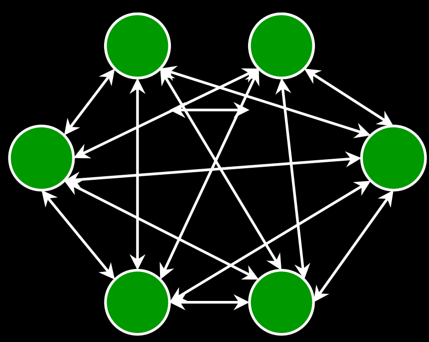
2 users: 1 key



3 users: 3 keys



4 users: 6 keys



6 users: 15 keys

100 users: 4950 keys

1000 users: 399500 keys

n users: $\dfrac{n(n-1)}{2}$ keys

Key distribution

Secure key distribution is the biggest problem with symmetric cryptography

Key exchange

How can you communicate securely with someone you've never met?

Whit Diffie: idea for a *public key* algorithm

Challenge: can this be done securely?

- Knowledge of public key should not allow derivation of private key

Diffie-Hellman exponential key exchange

Key distribution algorithm

- first algorithm to use public/private keys
- *not* public key encryption
- based on difficulty of computing discrete logarithms in a finite field compared with ease of calculating exponentiation

allows us to negotiate a secret **session key** without fear of eavesdroppers

Diffie-Hellman exponential key exchange

- All arithmetic performed in field of integers modulo some large number
- Both parties agree on
  - a **large prime number *p***
  - and a number $\alpha < p$
- Each party generates a public/private key pair

private key for user *i*: X_i

public key for user *i*: $Y_i = \alpha^{X_i} \bmod p$

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes

- Bob has secret key X_B
- Bob has public key Y_B

$$K = Y_B^{X_A} \bmod p$$

K = (Bob's public key)$^{\text{(Alice's private key)}}$ mod p

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes

$$K = Y_B^{X_A} \bmod p$$

- Bob has secret key X_B
- Bob has public key Y_B
- Bob computes

$$K' = Y_A^{X_B} \bmod p$$

K′ = (Alice's public key) $^{\textbf{(Bob's private key)}}$ ***mod p***

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes

- expanding:

$$K = Y_B^{X_A} \bmod p$$

$$\begin{aligned} K &= Y_B^{X_A} \bmod p \\ &= (\alpha^{X_B} \bmod p)^{X_A} \bmod p \\ &= \alpha^{X_B X_A} \bmod p \end{aligned}$$

- Bob has secret key X_B
- Bob has public key Y_B
- Bob computes

- expanding:

$$K' = Y_A^{X_B} \bmod p$$

$$\begin{aligned} K &= Y_B^{X_A} \bmod p \\ &= (\alpha^{X_B} \bmod p)^{X_A} \bmod p \\ &= \alpha^{X_B X_A} \bmod p \end{aligned}$$

$$\boldsymbol{K = K'}$$

K is a **common key**, known *only* to Bob and Alice

Diffie-Hellman example

Suppose $p = 31667$, $\alpha = 7$

Alice picks

$X_A = 18$

Alice's public key is:

$Y_A = 7^{18} \bmod 31667 = 6780$

Bob picks

$X_B = 27$

Bob's public key is:

$Y_B = 7^{27} \bmod 31667 = 22184$

$K = 22184^{18} \bmod 31667$

K = 14265

$K = 6780^{27} \bmod 31667$

K = 14265

Key distribution problem is solved!

- User maintains private key
- Publishes public key in database ("phonebook")

- Communication begins with key exchange to establish a common key
- Common key can be used to encrypt a **session key**
  - increase difficulty of breaking common key by reducing the amount of data we encrypt with it
  - session key is valid *only* for one communication session

RSA: Public Key Cryptography

- Ron Rivest, Adi Shamir, Leonard Adleman created a true public key encryption algorithm in 1977
- Each user generates two keys
  - private key (kept secret)
  - public key
- difficulty of algorithm based on the difficulty of factoring large numbers
  - keys are functions of a pair of large (~200 digits) prime numbers

RSA algorithm

Generate keys

- choose two random large prime numbers p, q
- Compute the product $n = pq$
- randomly choose the encryption key, e, such that:

  e and $(p - 1)(q - 1)$ are relatively prime
- use the extended Euclidean algorithm to compute the decryption key, d:

  $$ed = 1 \bmod ((p - 1)(q - 1))$$
  $$d = e^{-1} \bmod ((p - 1)(q - 1))$$
- discard p, q

RSA algorithm

- encrypt
  - divide data into numerical blocks < n
  - encrypt each block:

    $c = m^e \bmod n$

- decrypt:

  $m = c^d \bmod n$

Asymmetric Encryption: RSA

- P=5 & q=7
- n=5*7=35 and z=(4)*(6) = 24
- e = 5
- d = 29 , (29x5 = 1 mod 24)
- Keys generated are
  - Public key: (5, 35)
  - Private key is (29, 35)
- Encrypt the word love using (c = m^e mod n)
  - Assume that the alphabets are between 1 & 26

Plain Text	Numeric Representation	m^e	Cipher Text (c = m^e mod n)
l	12	248832	17
o	15	759375	15
v	22	5153632	22
e	5	3125	10

Asymmetric Encryption: RSA

- Decrypt the word love using ($m = c^d \bmod n$)
  - d=29, n=35

Cipher Text	c^d	($m = c^d \bmod n$)	Plain Text
17	481968572106750915091411825223072000	12	l
15	12783403948858939111232757568359400	15	o
22	852643319086537701956194499721110000000	22	v
10	100000000000000000000000000000	5	e

Communication with public key algorithms

Different keys for encrypting and decrypting

- no need to worry about key distribution

Communication with public key algorithms

Alice

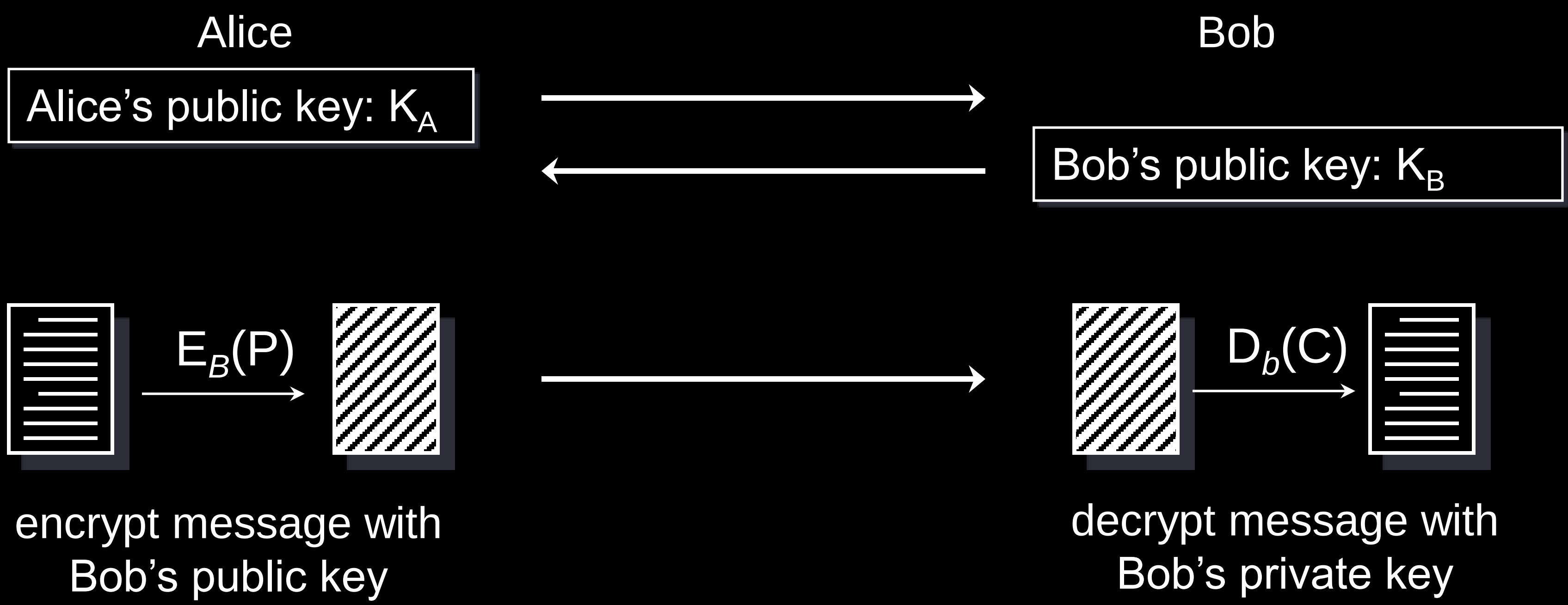
Alice's public key: K_A

Bob

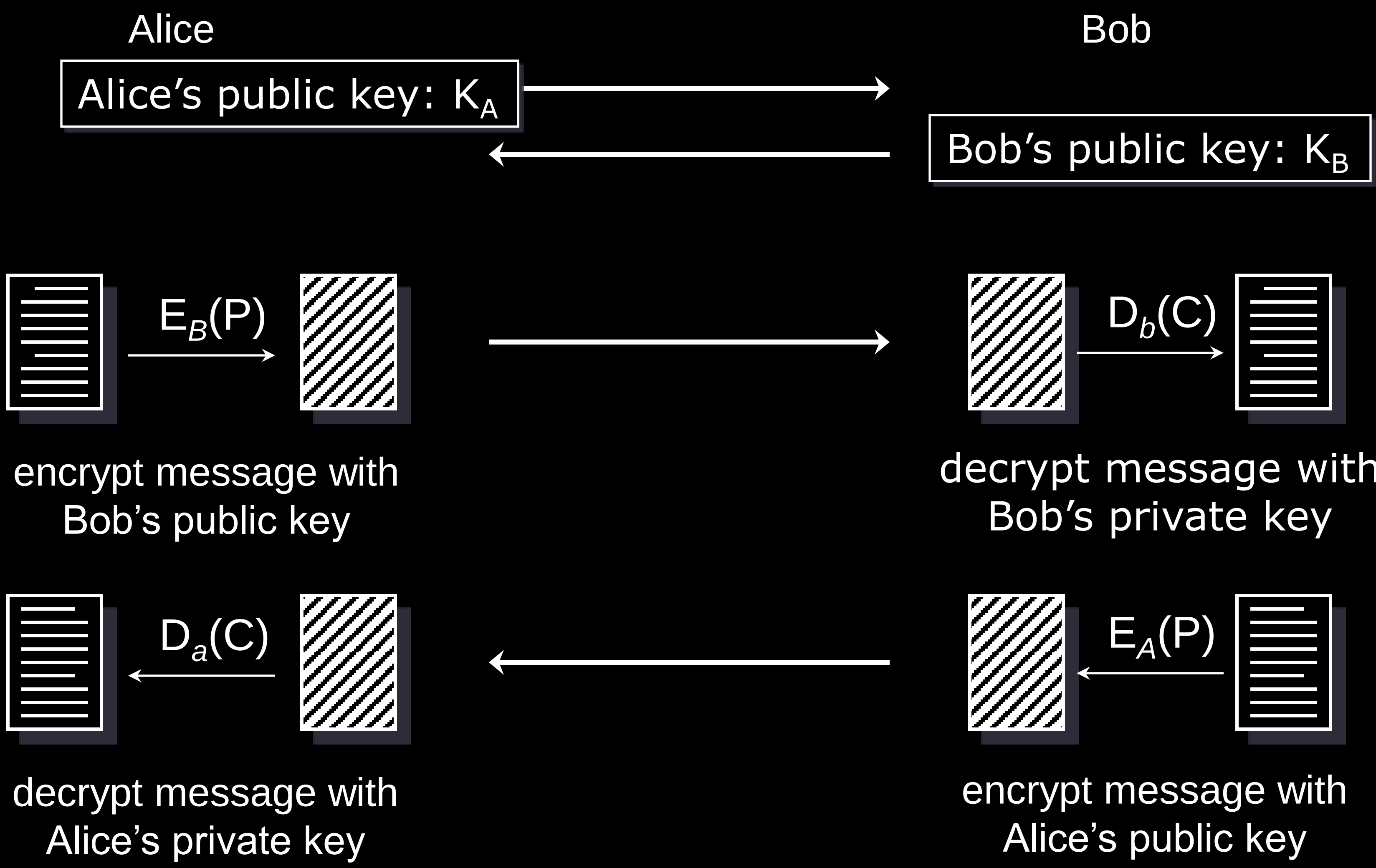
Bob's public key: K_B

exchange public keys
(or look up in a directory/DB)

Communication with public key algorithms



Communication with public key algorithms



Public key woes

Public key cryptography is great but:

- RSA about 100 times slower than DES in software, 1000 times slower in HW
- Vulnerable to chosen plaintext attack
  - if you know the data is one of *n* messages, just encrypt each message with the recipient's public key and compare
- It's a good idea to reduce the amount of data encrypted with any given key
  - but generating RSA keys is computationally very time consuming

Hybrid cryptosystems

Use public key cryptography to encrypt a randomly generated symmetric key

session key

Communication with a hybrid cryptosystem

Alice

Bob

Bob’s public key: K_B

Get recipient’s public key
(or fetch from directory/database)

Communication with a hybrid cryptosystem

Alice

Bob

Bob's public key: K_B

Pick random session key, *K*

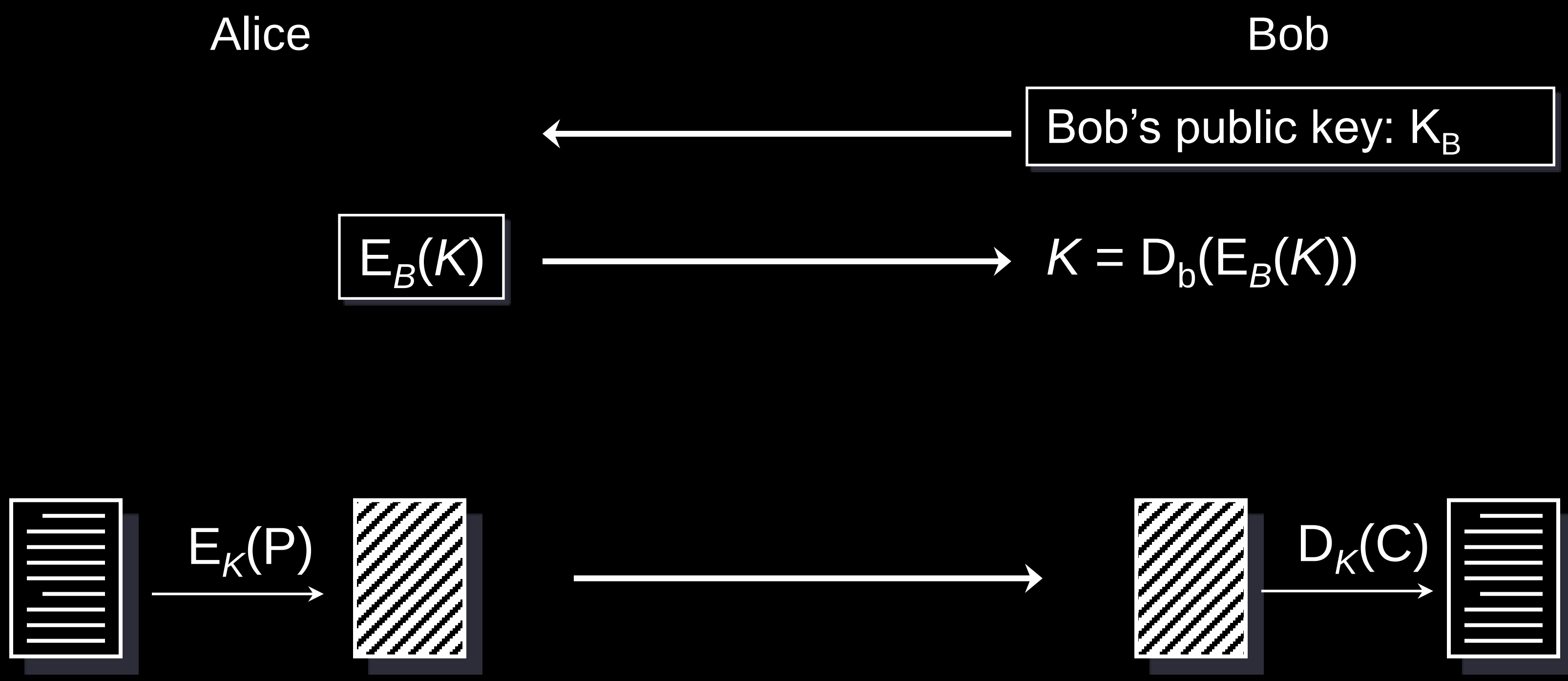
Encrypt session key
with Bob's public key

$E_B(K)$

$K = D_b(E_B(K))$

Bob decrypts *K* with
his private key

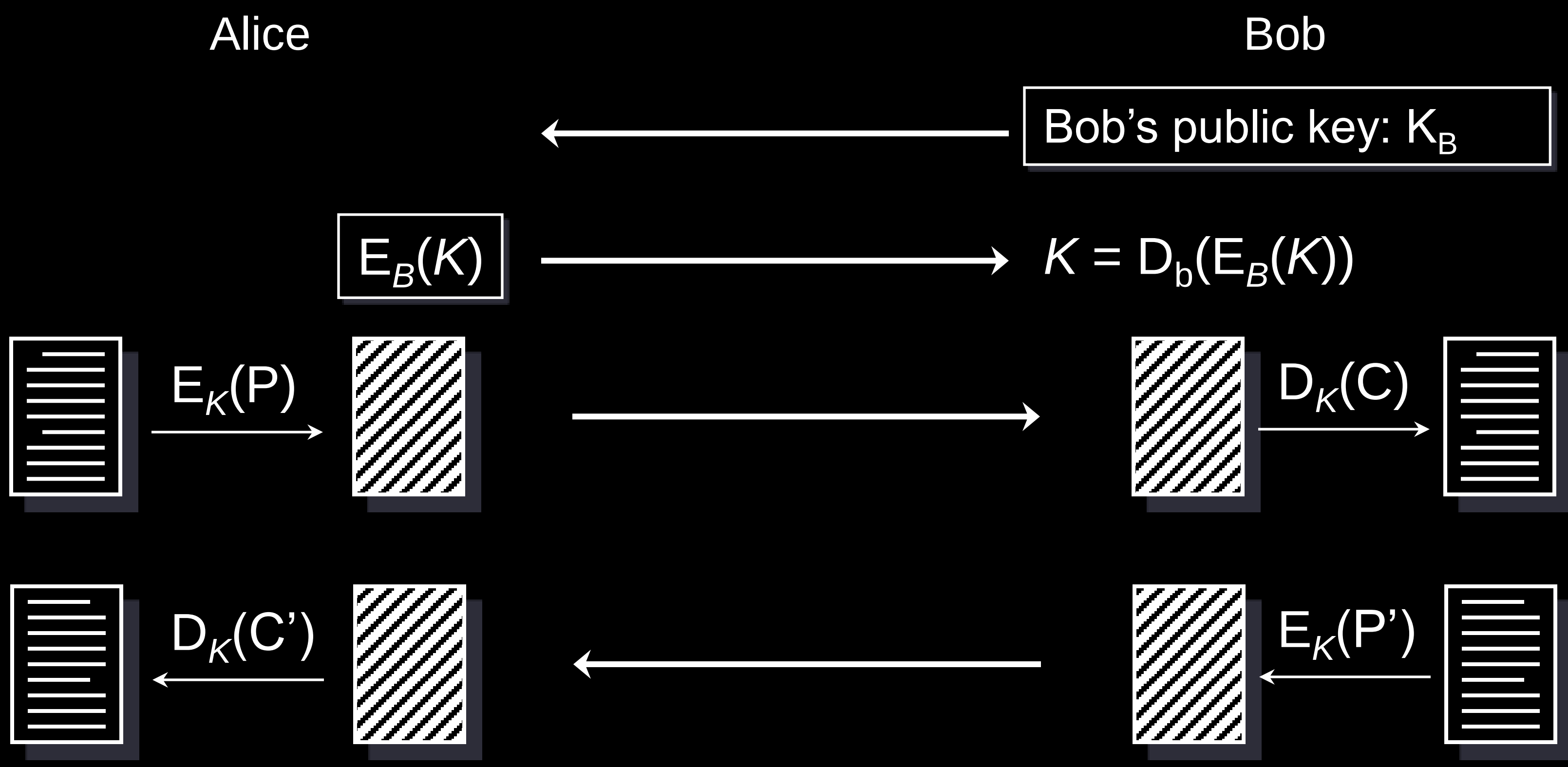
Communication with a hybrid cryptosystem



encrypt message using a symmetric algorithm and key *K*

decrypt message using a symmetric algorithm and key *K*

Communication with a hybrid cryptosystem



decrypt message using a symmetric algorithm and key *K*

encrypt message using a symmetric algorithm and key *K*