Operating Systems Design

21. Cryptography: An Introduction

Paul Krzyzanowski

pxk@cs.rutgers.edu

Cryptography ≠ Security

Cryptography may be a component of a secure system

Adding cryptography may not make a system secure

Terms

Plaintext (cleartext), message *M*

encryption, E(*M*)

produces ciphertext, *C*=E(*M*)

decryption: *M*=D(*C*)

Cryptographic algorithm, cipher

Terms: types of ciphers

- Types
  - restricted cipher
  - symmetric algorithm
  - public key algorithm

- Stream vs. Block
  - Stream cipher
    - Encrypt a message a character at a time
  - Block cipher
    - Encrypt a message a chunk at a time

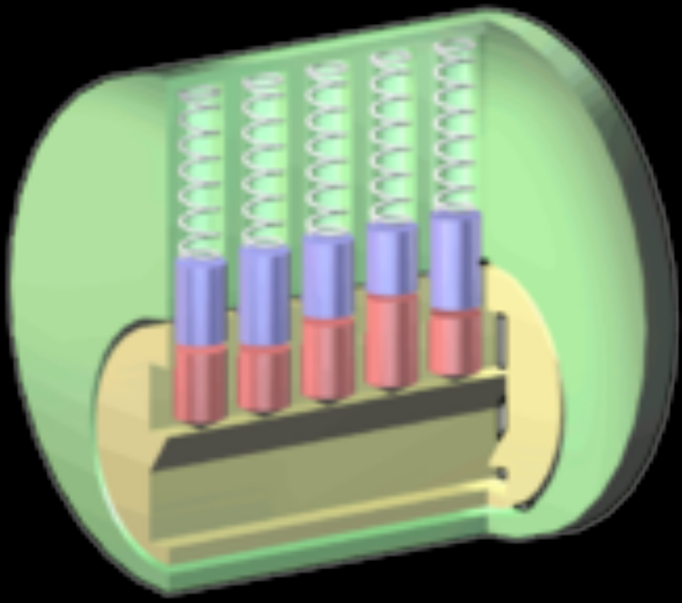
Restricted cipher

Secret algorithm

- Vulnerable to:
  - Leaking
  - Reverse engineering
    - HD DVD (Dec 2006) and Blu-Ray (Jan 2007)
    - RC4
    - All digital cellular encryption algorithms
    - DVD and DIVX video compression
    - Firewire
    - Enigma cipher machine
    - Every NATO and Warsaw Pact algorithm during Cold War
- Not a viable approach!

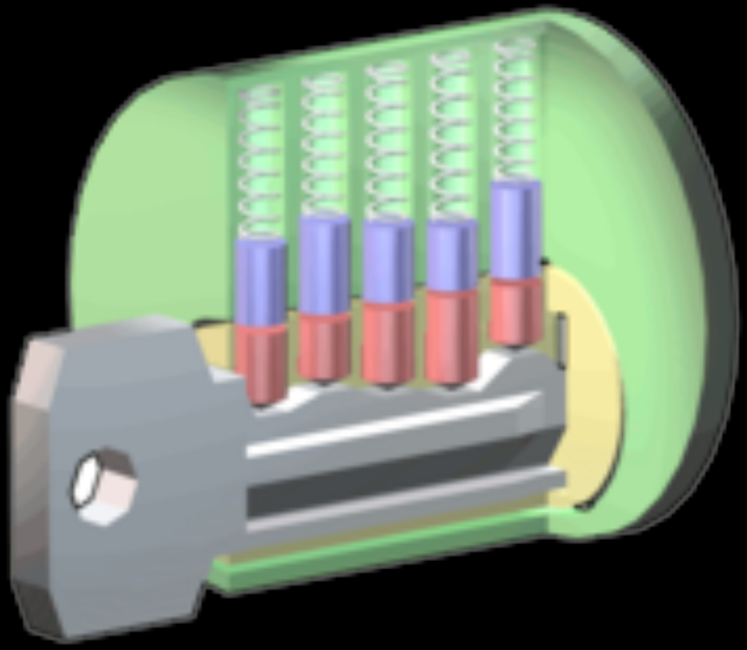
The key

The key



Source: en.wikipedia.org/wiki/Pin_tumbler_lock

The key



Source: en.wikipedia.org/wiki/Pin_tumbler_lock

The key

- We understand how it works:
  - Strengths
  - Weaknesses

- Based on this understanding, we can assess how much to trust the key & lock.

Source: en.wikipedia.org/wiki/Pin_tumbler_lock

Symmetric algorithm

Secret key

$$C = E_K(M)$$

$$M = D_K(C)$$

Public key algorithm

Public and private keys

$$C_1 = E_{\text{public}}(M)$$

$$M = D_{\text{private}}(C_1)$$

also:

$$C_2 = E_{\text{private}}(M)$$

$$M = D_{\text{public}}(C_2)$$

McCarthy’s puzzle (1958)

The setting:

- Two countries are at war
- One country sends spies to the other country
- To return safely, spies must give the border guards a password

- Spies can be trusted
- Guards chat – information given to them may leak

McCarthy’s puzzle

Challenge

How can a guard authenticate a person without knowing the password?

Enemies cannot use the guard’s knowledge to introduce their own spies

Solution to McCarthy's puzzle

Michael Rabin, 1958

Use one-way function, $B = f(A)$

- Guards get *B*
  - Enemy cannot compute A if they know *A*
- Spies give *A*, guards compute *f(A)*
  - If the result is *B,* the password is correct.

Example function:

Middle squares

- Take a 100-digit number (A), and square it
- Let B = middle 100 digits of 200-digit result

One-way functions

- Easy to compute in one direction
- Difficult to compute in the other

Examples:

Factoring:

$pq = N$ EASY

find p,q given N DIFFICULT

Discrete Log:

$a^b \bmod c = N$ EASY

find b given a, c, N DIFFICULT

McCarthy's puzzle example

Example with an 18 digit number

A = 289407349786637777

A^2 = 83756614110525308948445338203501729

Middle square, B = 110525308948445338

Given A, it is easy to compute B

Given B, it is extremely hard to compute A

Hash functions

- one-way function
  - Rabin, 1958: McCarthy's problem
  - middle squares, exponentiation, …
- [one-way] hash function
  - message digest, fingerprint, cryptographic checksum, integrity check
- encrypted hash
  - message authentication code
  - only possessor of key can validate message

Popular hash functions

- SHA-2
  - Designed by the NSA; published by NIST
  - SHA-224, SHA-256, SHA-384, SHA-512
    - e.g., Linux passwords used MD5 and now SHA-512
- SHA-3
  - Under development
- MD5
  - 128 bits (not often used now since weaknesses were found)
- Derivations from ciphers:
  - Blowfish (used for password hashing in OpenBSD)
  - 3DES – used for old Linux password hashes

Cryptography: what is it good for?

- Authentication
  - determine origin of message
- Integrity
  - verify that message has not been modified
- Nonrepudiation
  - sender should not be able to falsely deny that a message was sent
- Confidentiality
  - others cannot read contents of the message

Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- One-way hash functions
- Random number generators

Popular symmetric algorithms

- AES (Advanced Encryption Standard)
  - FIPS standard since 2002
  - 128, 192, or 256-bit keys; operates on 128-bit blocks
- DES, 3DES
  - FIPS standard since 1976
  - 56-bit key; operates on 64-bit (8-byte) blocks
  - Triple DES recommended since 1999 (112 or 168 bits)
- Blowfish
  - Key length from 23-448 bits; 64-bit blocks
- IDEA
  - 128-bit keys; operates on 64-bit blocks
  - More secure than DES but faster algorithms are available

Is DES secure?

56-bit key makes DES relatively weak

- 7.2×10^{16} keys
- Brute-force attack

Late 1990's:

- DES cracker machines built to crack DES keys in a few hours
- DES Deep Crack: 90 billion keys/second
- Distributed.net: test 250 billion keys/second

The power of 2

Adding an extra bit to a key doubles the search space.

Suppose it takes 1 second to attack a 20-bit key:

- 21-bit key: 2 seconds
- 32-bit key: 1 hour
- 40-bit key: 12 days
- 56-bit key: 2,178 years
- 64-bit key: >557,000 years!

AES

From NIST:

> Assuming that one could build a machine that could recover a DES key in a second (i.e., try 2^{56} keys per second), then it would take that machine approximately 149 trillion years to crack a 128-bit AES key. To put that into perspective, the universe is believed to be less than 20 billion years old.

http://csrc.nist.gov/encryption/aes/

Increasing The Key

Can double encryption work for DES?

- Useless if we could find a key K such that:

$$E_K(P) = E_{K2}(E_{K1}(P))$$

- This does not hold for DES (luckily!)

Double DES

<u>Vulnerable to meet-in-the-middle attack</u>

If we know some pair (P, C), then:

[1] Encrypt P for all 2^{56} values of K_1

[2] Decrypt C for all 2^{56} values of K_2

For each match where [1] = [2]

- test the two keys against another P, C pair
- if match, you are assured that you have the key

Triple DES

Triple DES with two 56-bit keys:

$$C = E_{K1}(D_{K2}(E_{K1}(P)))$$

Triple DES with three 56-bit keys:

$$C = E_{K3}(D_{K2}(E_{K1}(P)))$$

Decryption used in middle step for compatibility with DES (K_1=K_2=K_3)

$$C = E_K(D_K(E_K(P))) \equiv C = E_{K1}(P)$$

Secure Communication

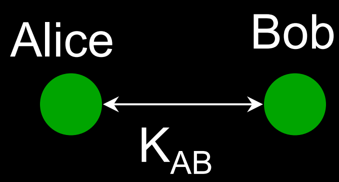
Communicating with symmetric cryptography

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

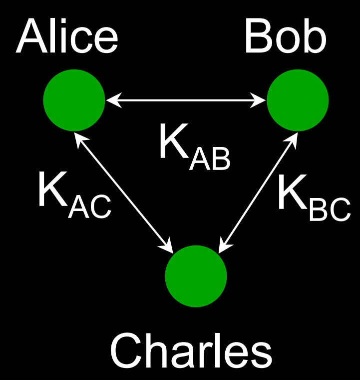
- Key distribution must be secret
  – otherwise messages can be decrypted
  – users can be impersonated

Key explosion

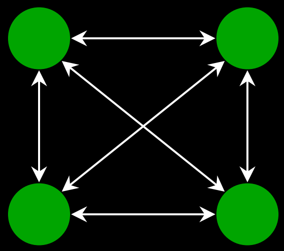
Each pair of users needs a separate key for secure communication



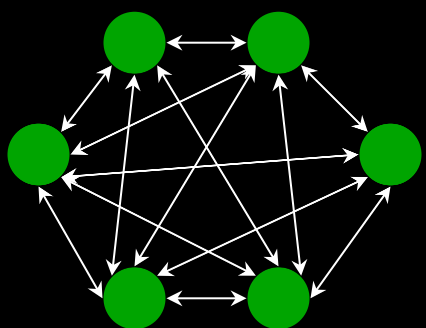
2 users: 1 key



3 users: 3 keys



4 users: 6 keys



6 users: 15 keys

100 users: 4,950 keys

1000 users: 399,500 keys

n users: $\frac{n(n-1)}{2}$ 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 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 two parties to compute a common key without fear from eavesdroppers. Then, they can securely transmit a session key.

Diffie-Hellman Key Exchange

- All arithmetic performed in a 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

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

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

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

Diffie-Hellman exponential key exchange

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

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

- expanding:

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

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

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

- expanding:

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

$K = K'$

K is a *<u>common key</u>*, known *only* to Bob and Alice

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 (can be shared with anyone)
- 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 Encryption

- Key pair: *e, d*
- Agreed-upon modulus: *n*

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

    $c = m^e \bmod n$

- Decrypt:

    $m = c^d \bmod n$

Communication with public key algorithms

Different keys for encrypting and decrypting

- no need to worry about key distribution

Communication with public key algorithms

Alice

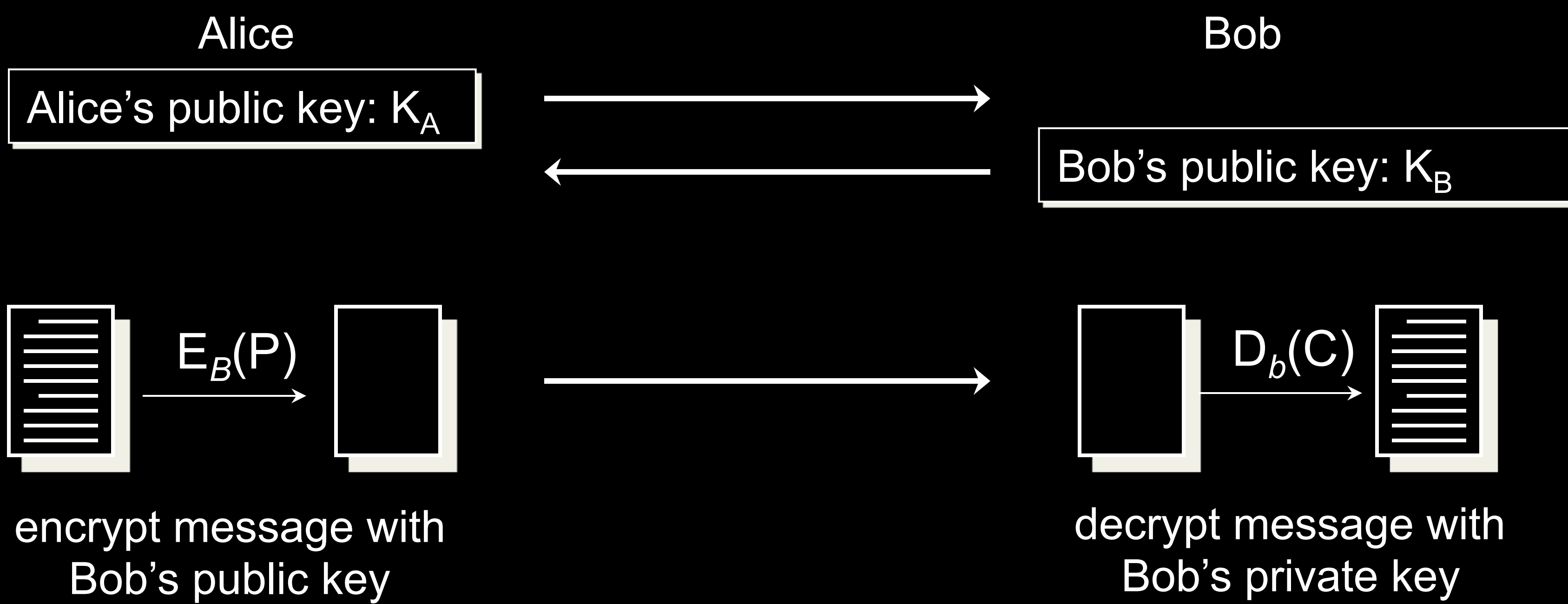
Alice's public key: K_A

Bob

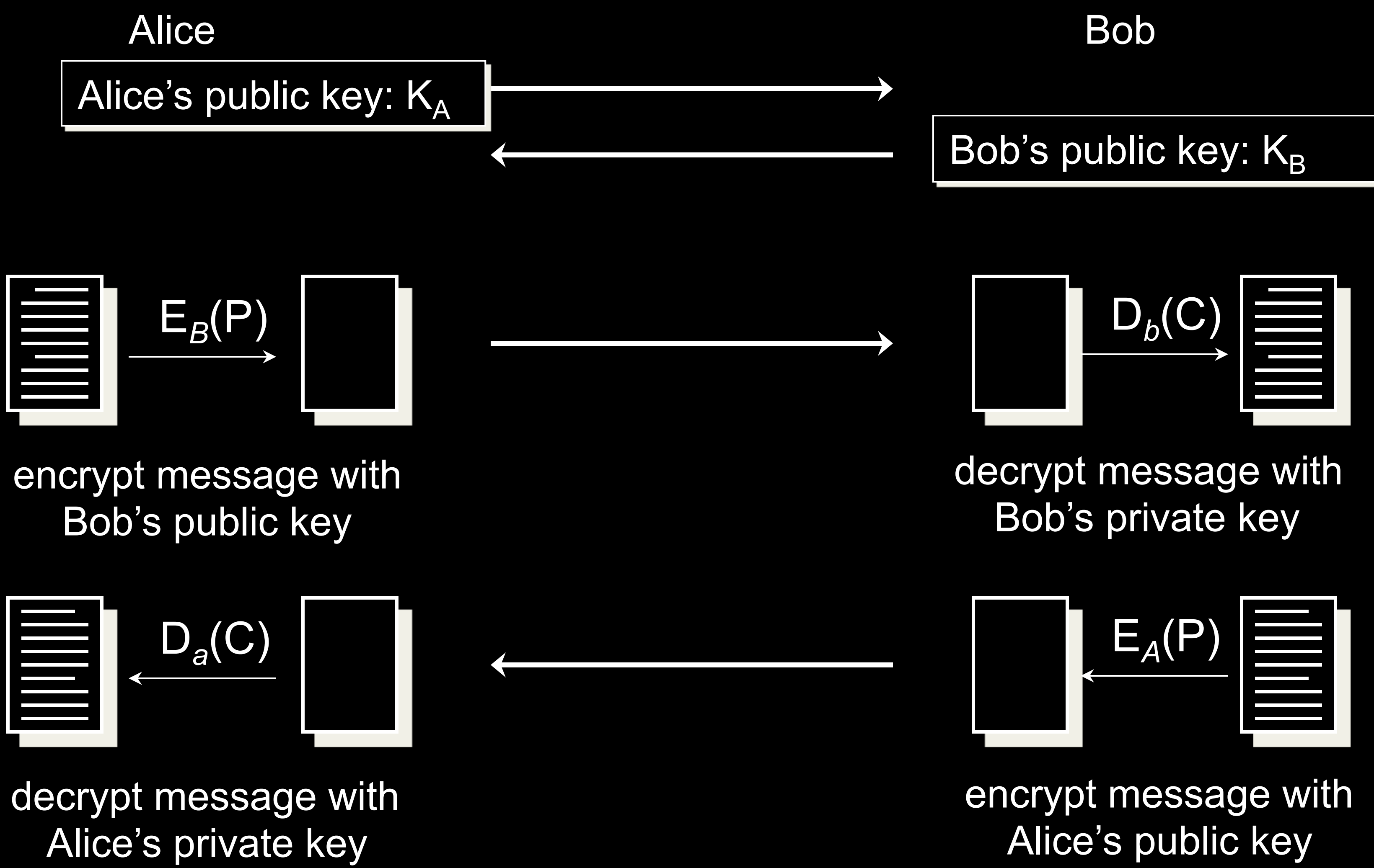
Bob's public key: K_B

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

Communication with public key algorithms



Communication with public key algorithms



Hybrid cryptosystems

Use public key cryptography to encrypt a randomly generated symmetric key

session key

Communication with a hybrid cryptosystem

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*

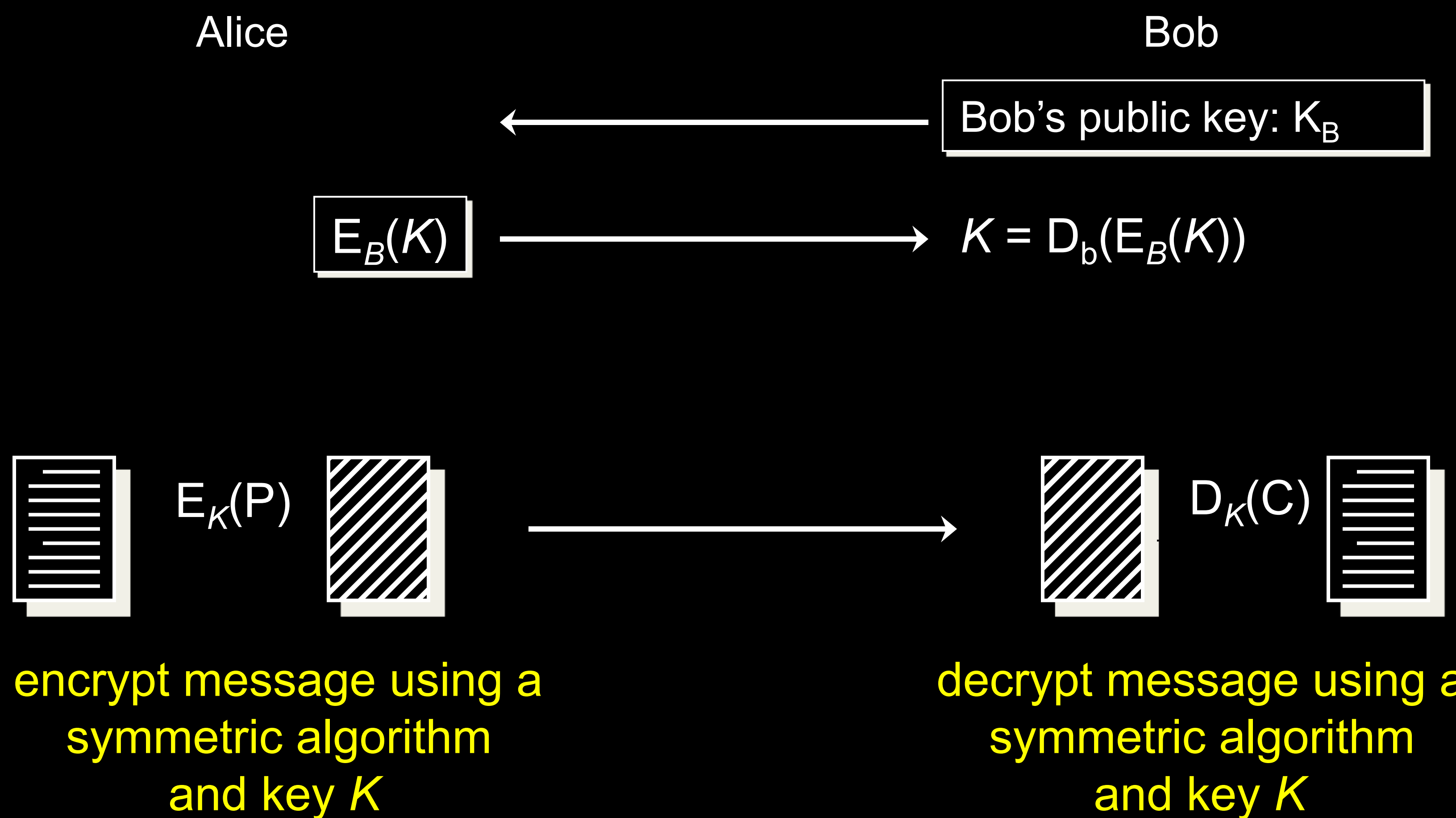
Encrypt session key
with Bob's public key

$E_B(K)$

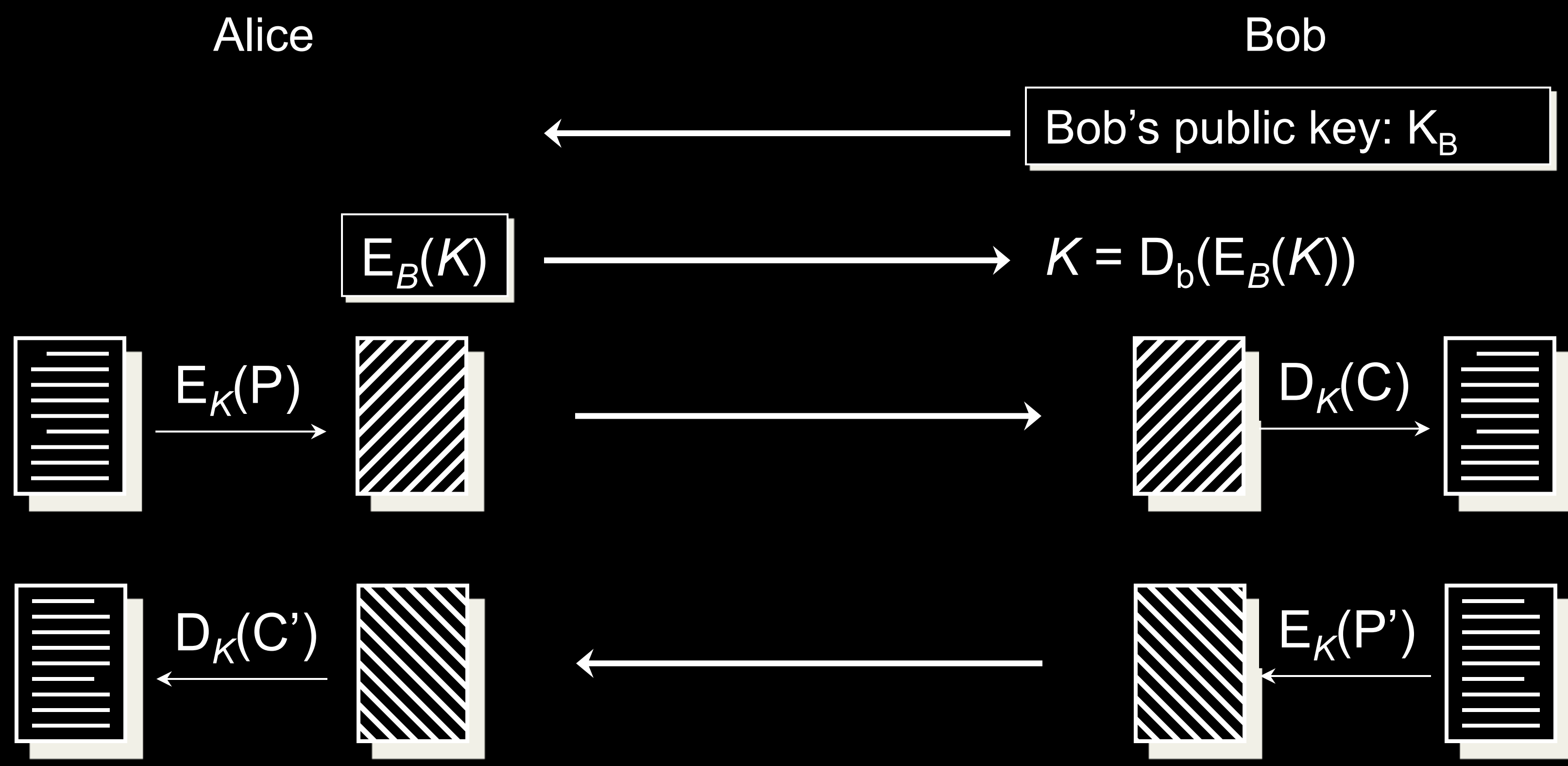
$K = D_b(E_B(K))$

Bob decrypts *K* with
his private key

Communication with a hybrid cryptosystem



Communication with a hybrid cryptosystem



decrypt message using a symmetric algorithm and key *K*

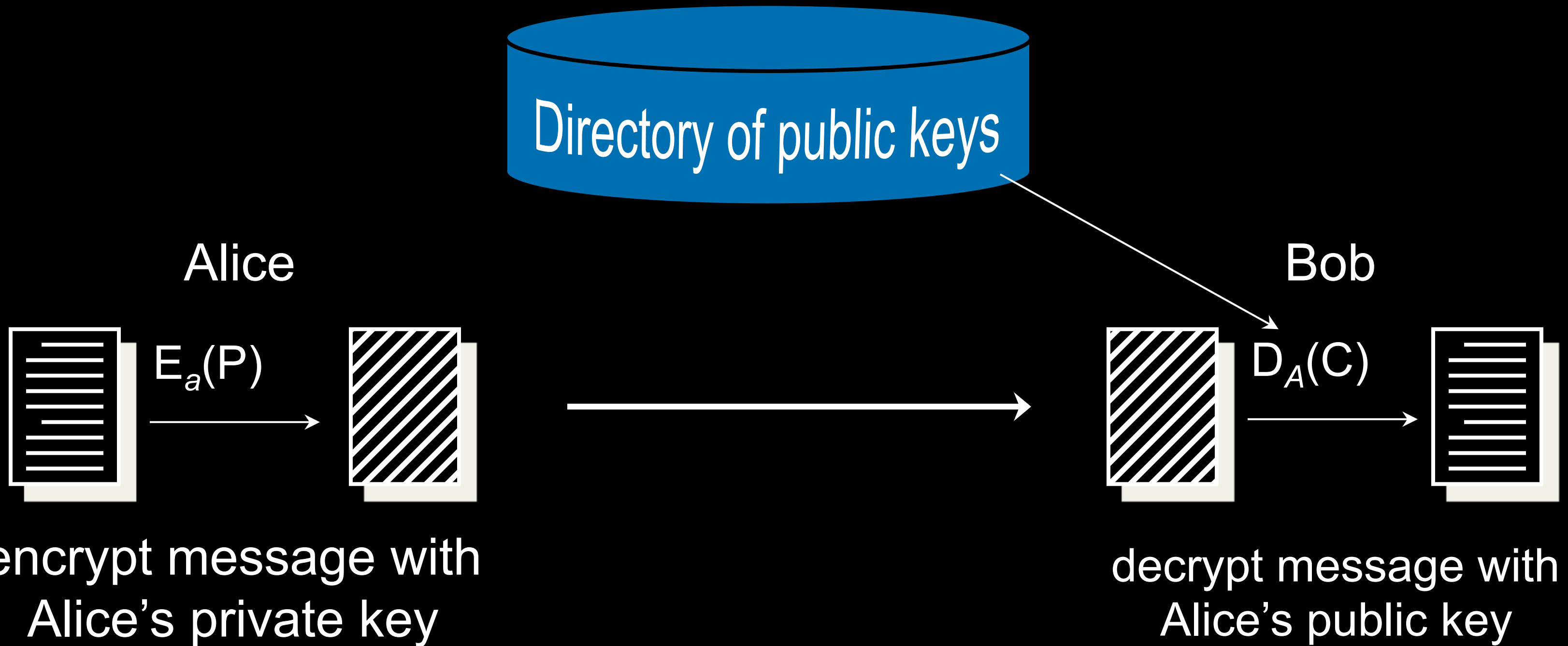
encrypt message using a symmetric algorithm and key *K*

Digital Signatures

- Validate the creator (signer) of the content
- Validate the the content has not been modified since it was signed
- The content does not have to be encrypted

Digital signatures - public key cryptography

Encrypting a message with a private key is the same as signing it!



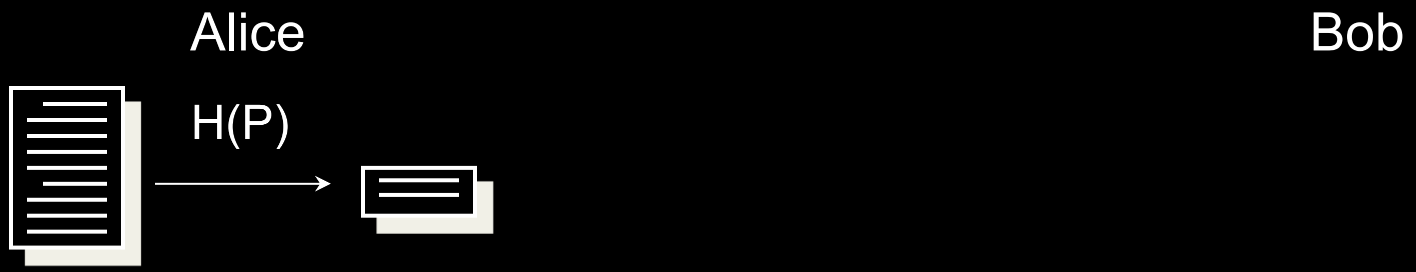
But

- We don’t want to permute/hide the content
  - If Alice was sending binary data to Bob, how would he deduce that it decrypted correctly?
- Public key encryption is considerably slower than symmetric encryption

Signatures: Hashes to the rescue!

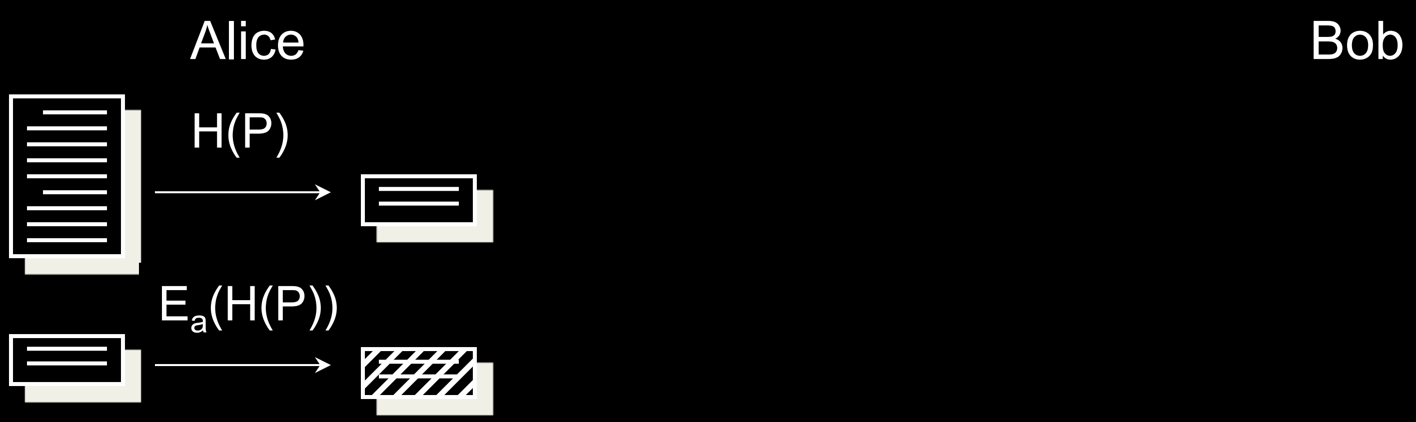
- Create a hash of the message
- Encrypt the hash with your public key and send it with the message
- Recipient validates the hash by decrypting it with your public key and comparing it with the hash of the received message
  - If the hashes don't match, that means either
    (a) the message was modified
    or (b) the encrypted hash was modified
    or (c) the hash was not encrypted by the correct party

Digital signatures - public key cryptography



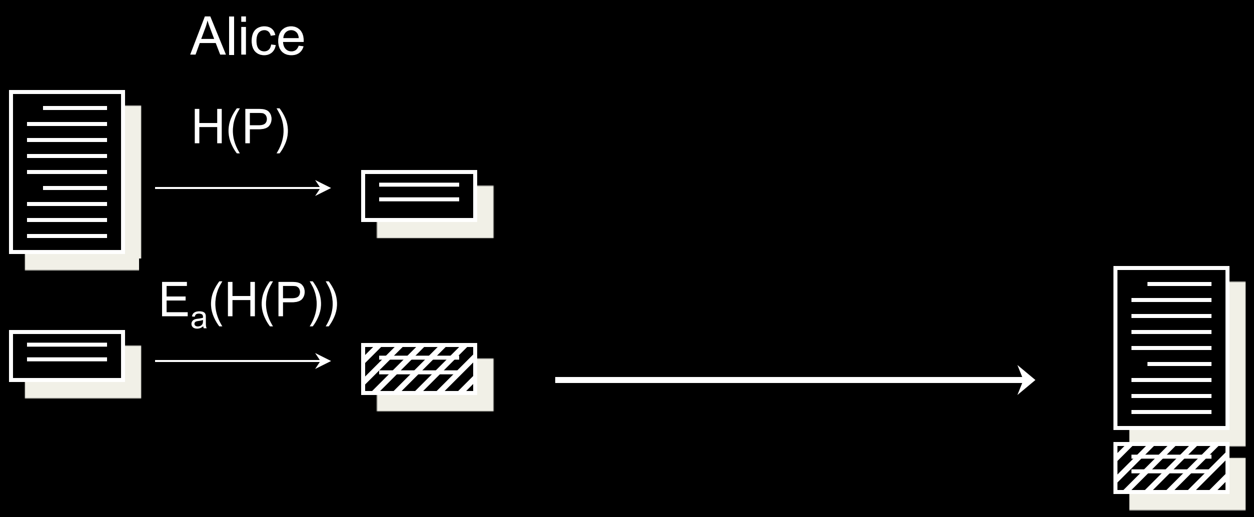
Alice generates a hash of the message

Digital signatures - public key cryptography



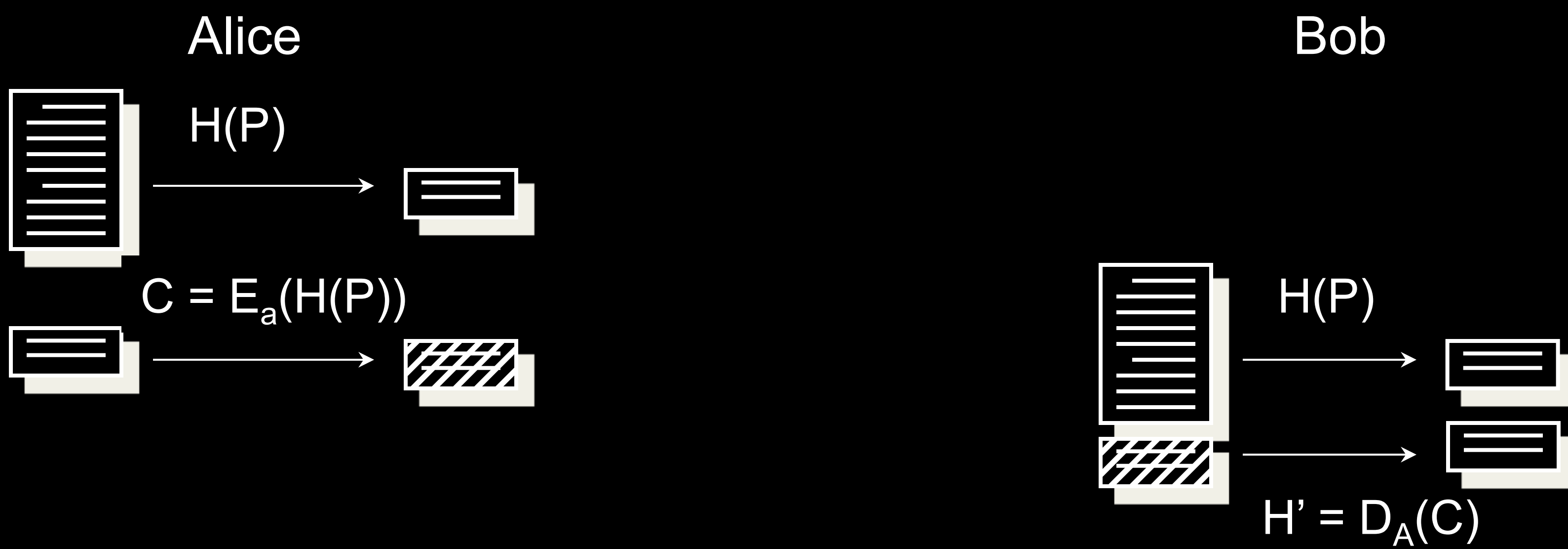
Alice encrypts the hash with her private key

Digital signatures - public key cryptography



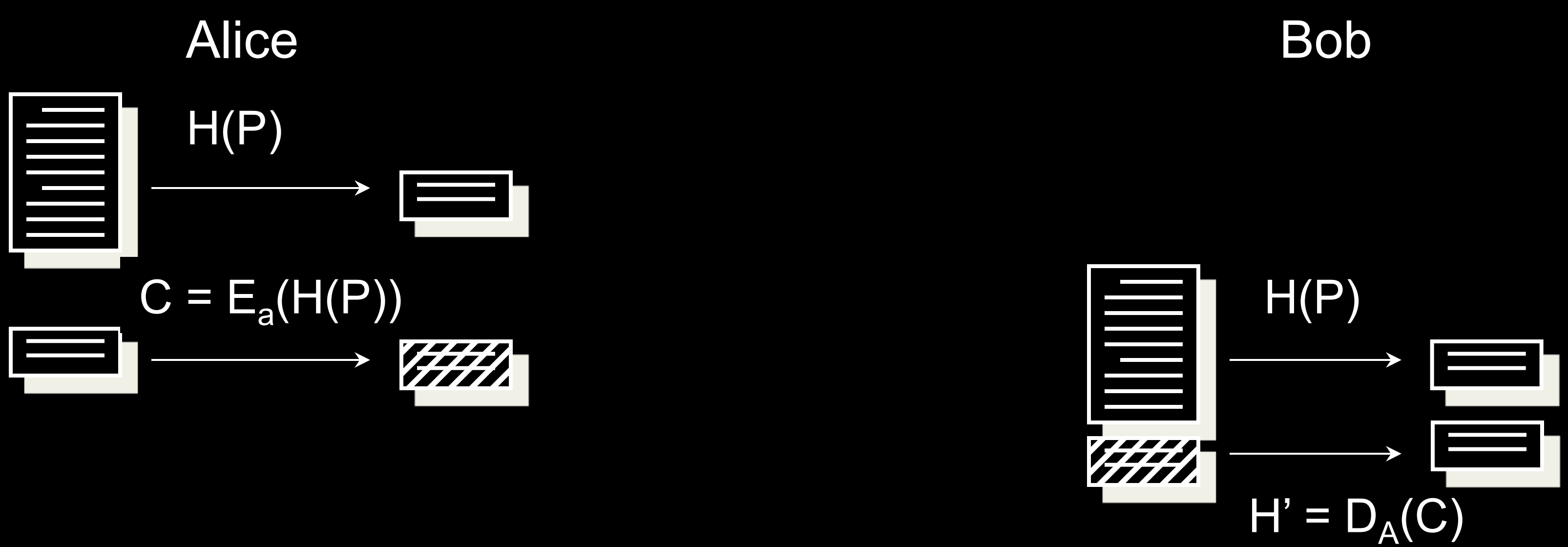
Alice sends Bob the message and the encrypted hash

Digital signatures - public key cryptography



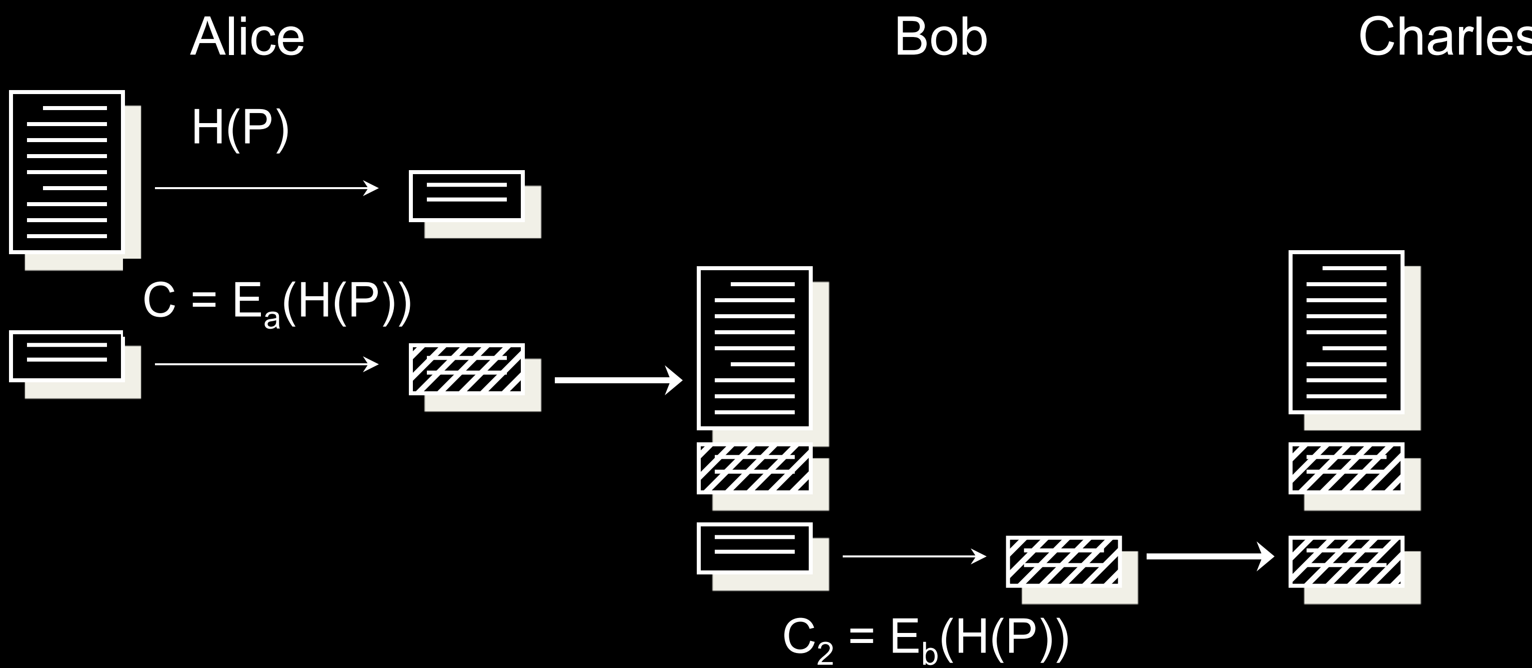
1. Bob decrypts the has using Alice's public key
2. Bob computes the hash of the message sent by Alice

Digital signatures - public key cryptography



If the hashes match
- the encrypted hash *must* have been generated by Alice
- the signature is valid

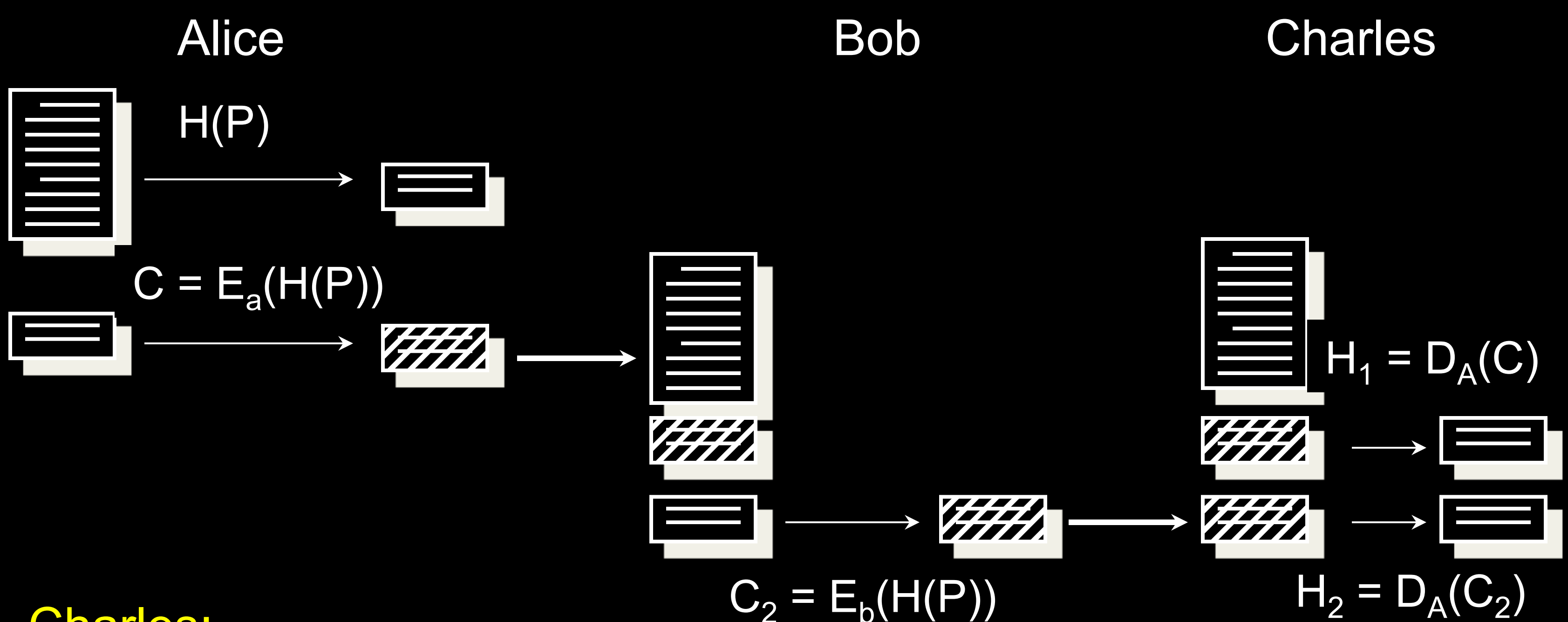
Digital signatures - multiple signers



Bob generates a hash (same as Alice's) and encrypts it with his private key

- sends Charles:
  {message, Alice's encrypted hash, Bob's encrypted hash}

Digital signatures - multiple signers



Charles:

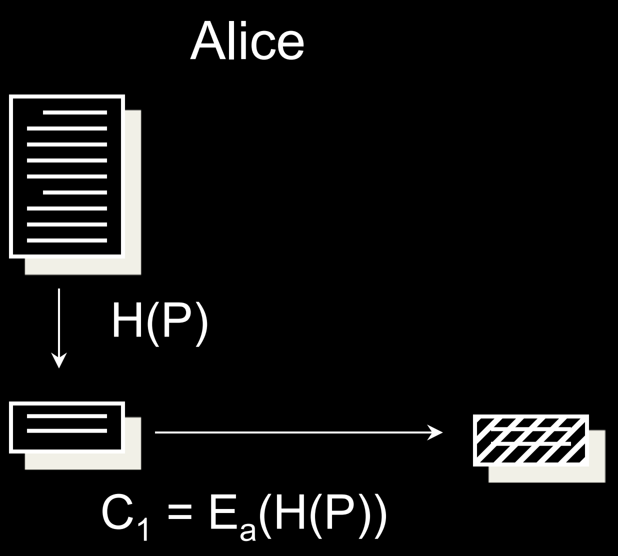
- generates a hash of the message: H(P)
- decrypts Alice’s encrypted hash with Alice’s public key
  - validates Alice’s signature
- decrypts Bob’s encrypted hash with Bob’s public key
  - validates Bob’s signature

Covert AND authenticated messaging

If we want to keep the message secret

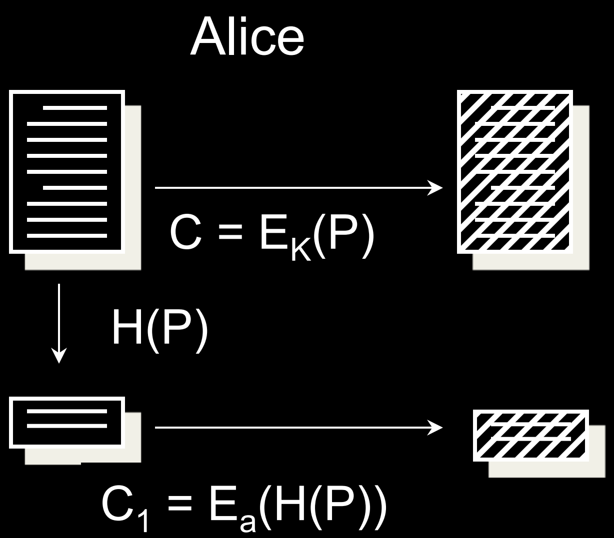
- combine encryption with a digital signature
- use a session key:
  pick a random key, *K*, to encrypt the message with a symmetric algorithm
- encrypt *K* with the public key of each recipient
- for signing, encrypt the hash of the message with sender's private key

Secure and authenticated messaging



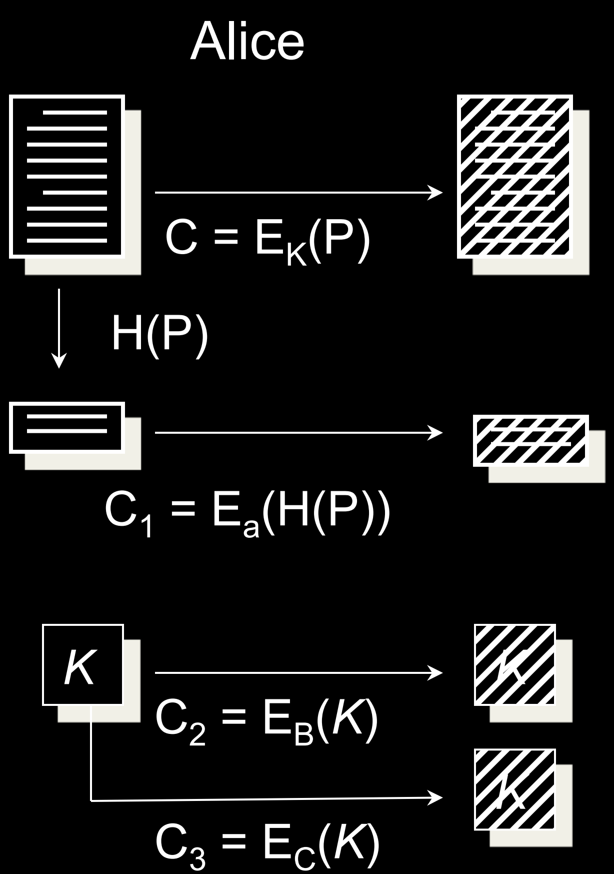
Alice generates a digital signature by encrypting the message digest with her private key.

Secure and authenticated messaging



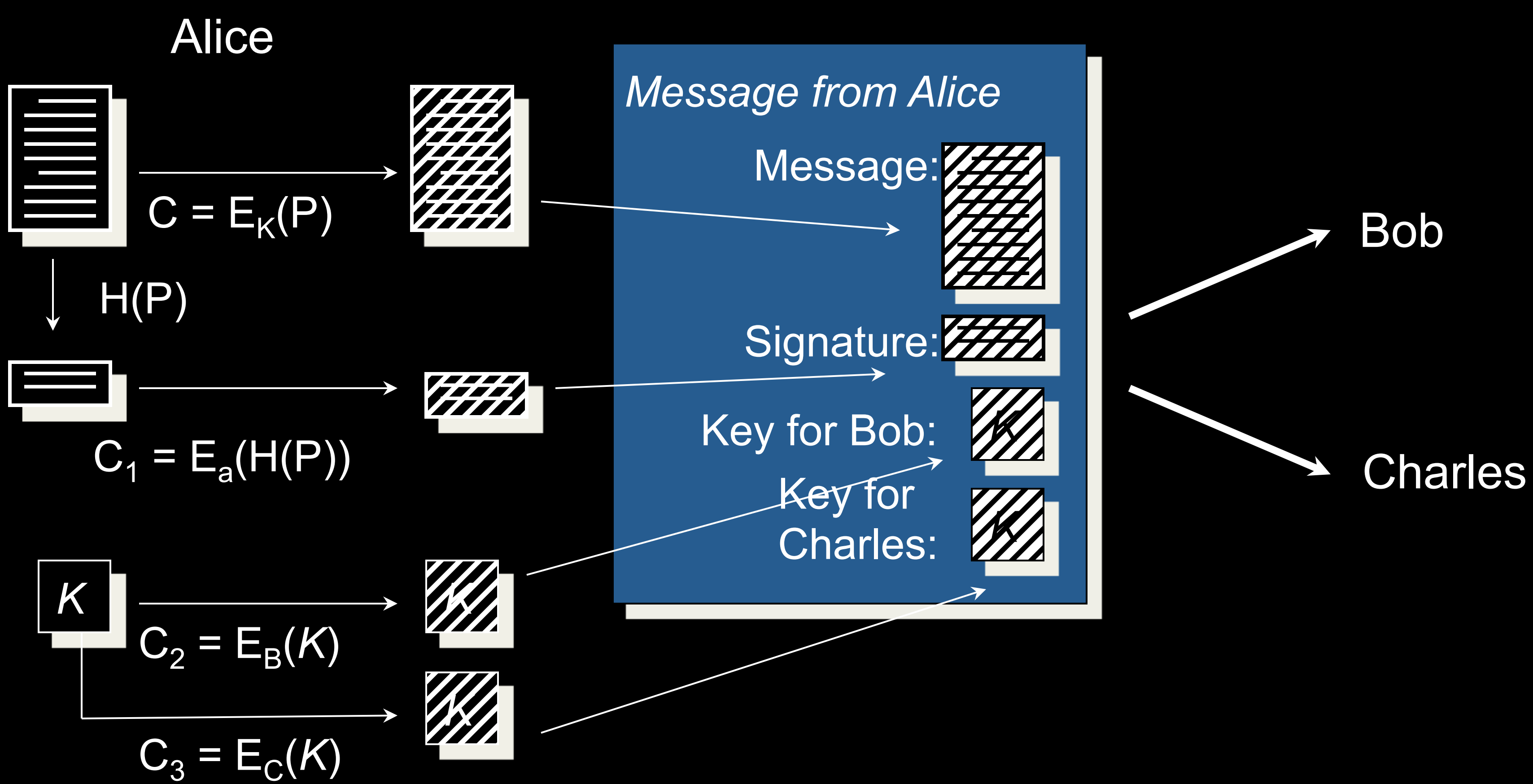
Alice picks a random key, *K*, and encrypts the message (P) with it using a symmetric algorithm.

Secure and authenticated messaging



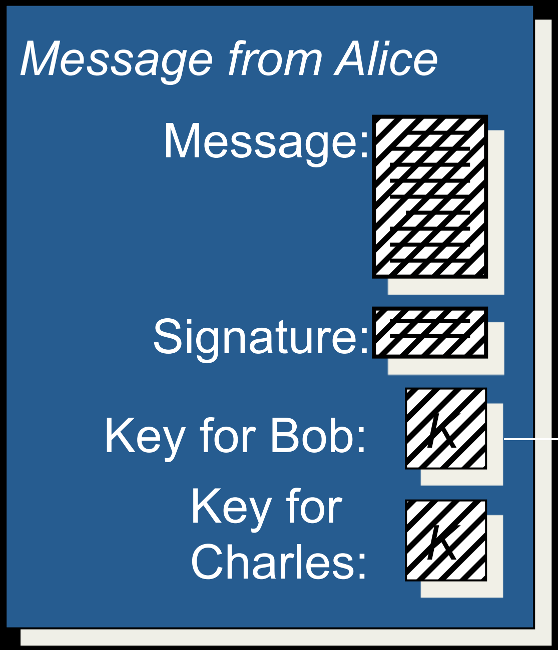
Alice encrypts the session key for each recipient of this message: Bob and Charles using their public keys.

Secure and authenticated messaging



The aggregate message is sent to Bob and Charles

Secure and authenticated messaging

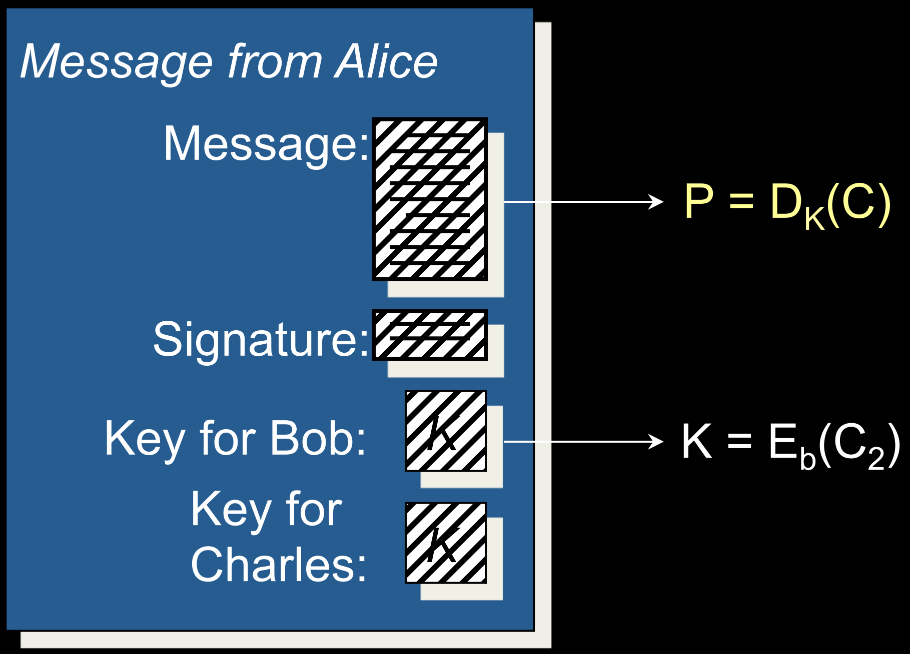


$K = E_b(C_2)$

Bob receives the message:

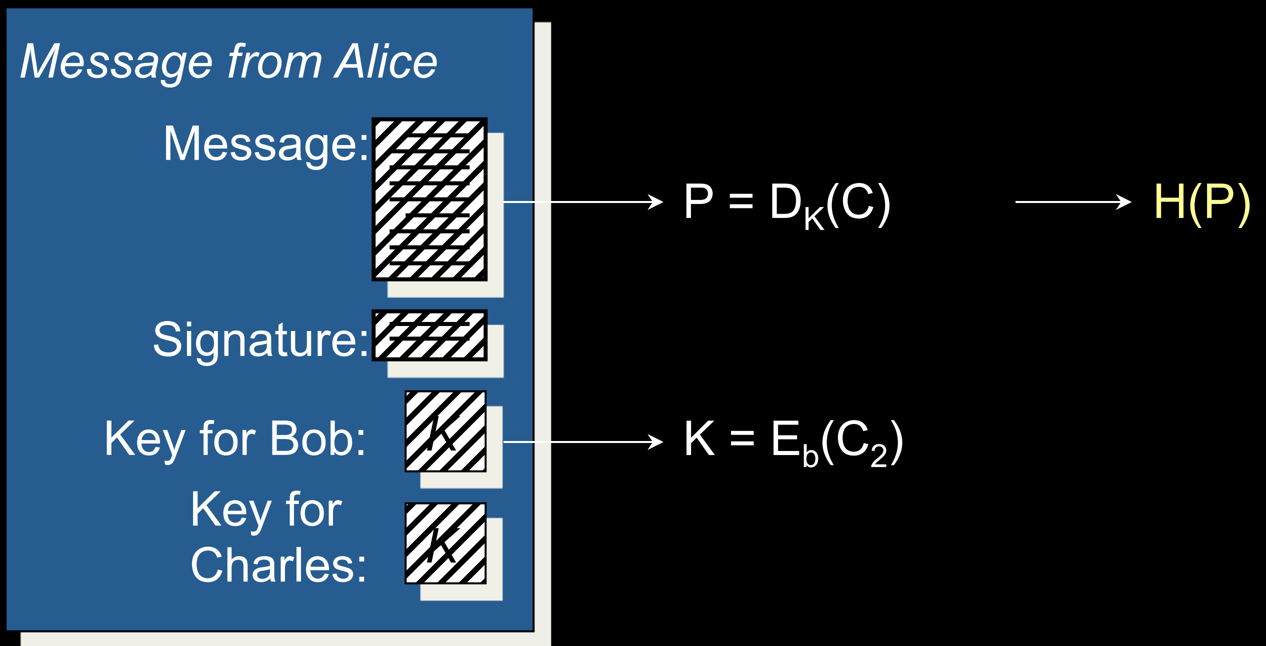
- extracts key by decrypting it with his private key

Secure and authenticated messaging



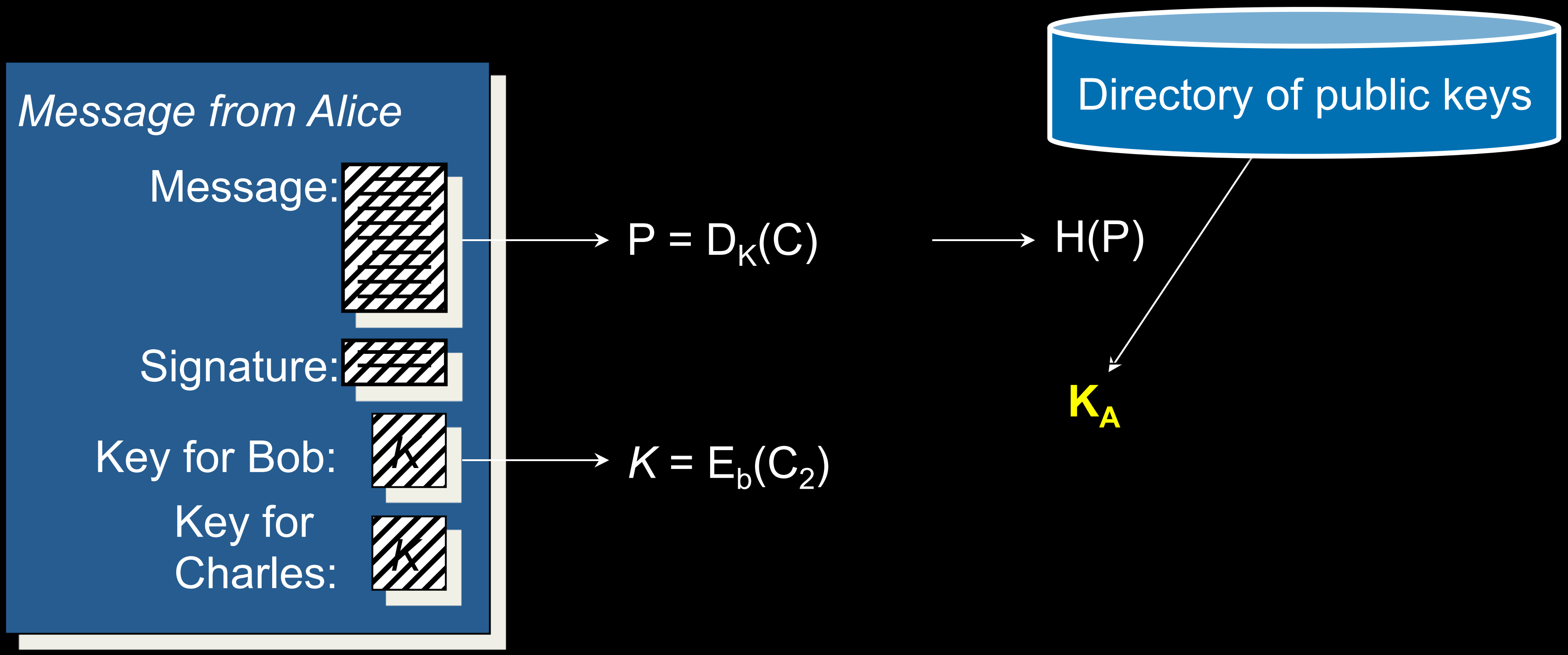
Bob decrypts the message using *K*

Secure and authenticated messaging



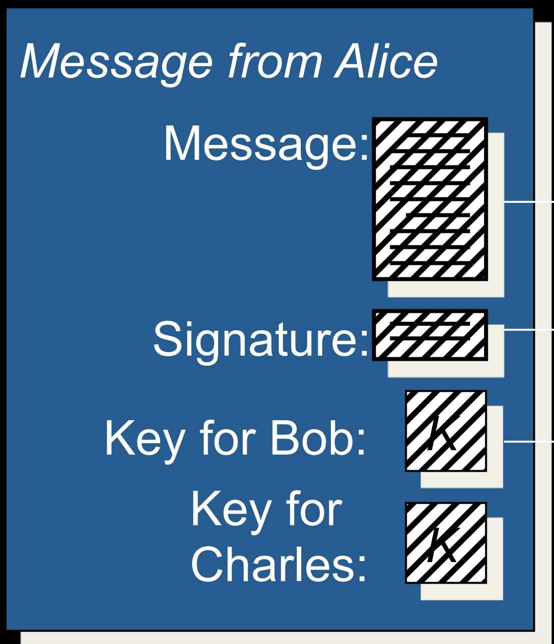
Bob computes the hash of the message

Secure and authenticated messaging



Bob looks up Alice's public key

Secure and authenticated messaging



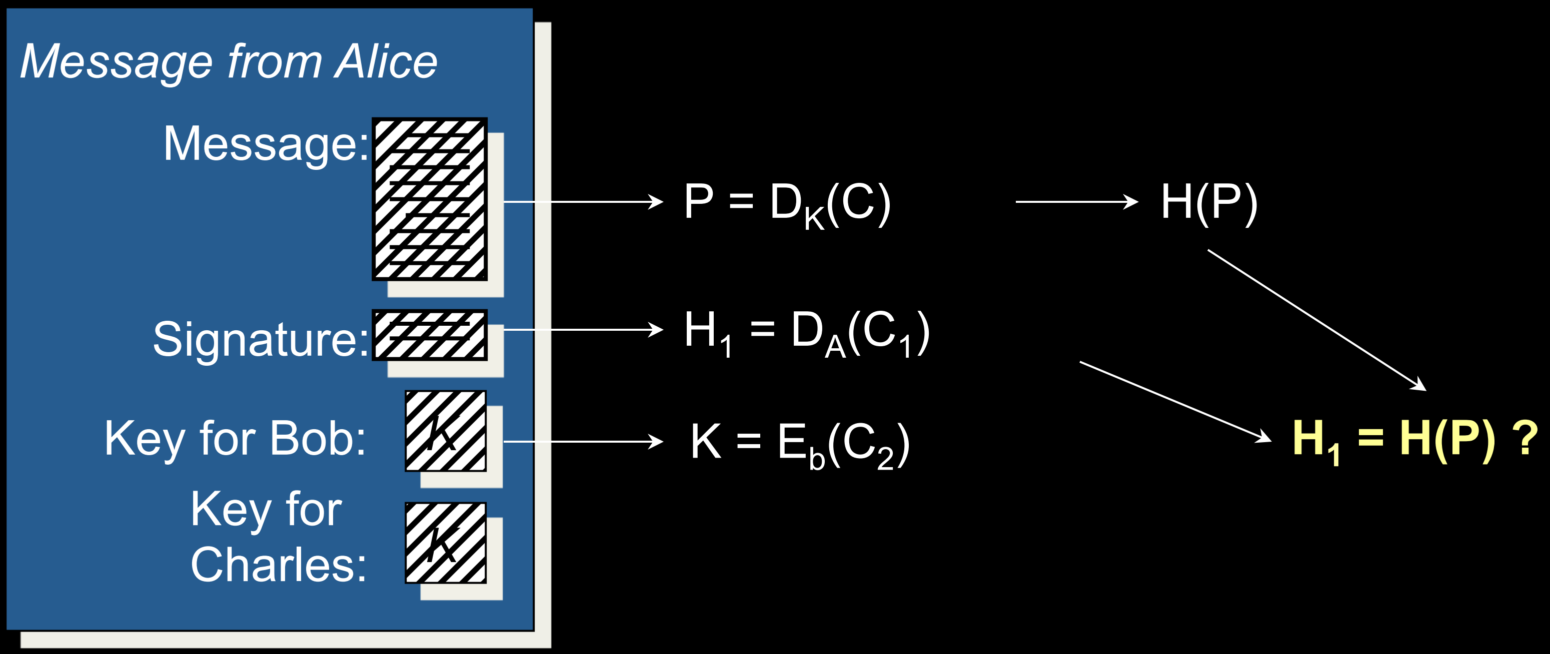
$P = D_K(C)$ → $H(P)$

$H_1 = D_A(C_1)$

$K = E_b(C_2)$

Bob decrypts Alice's signature using Alice's public key

Secure and authenticated messaging



Bob validates Alice's signature

The End