Digital Signatures

Signatures

We use signatures because a signature is:

- Authentic
- Not reusable
- Renders document unalterable
- Unforgeable
- Non repudiatable

Signatures

We use signatures because a signature is

- Authentic
- Unforgeable
- Not reusable
- Non repudiatable
- Renders document unalterable

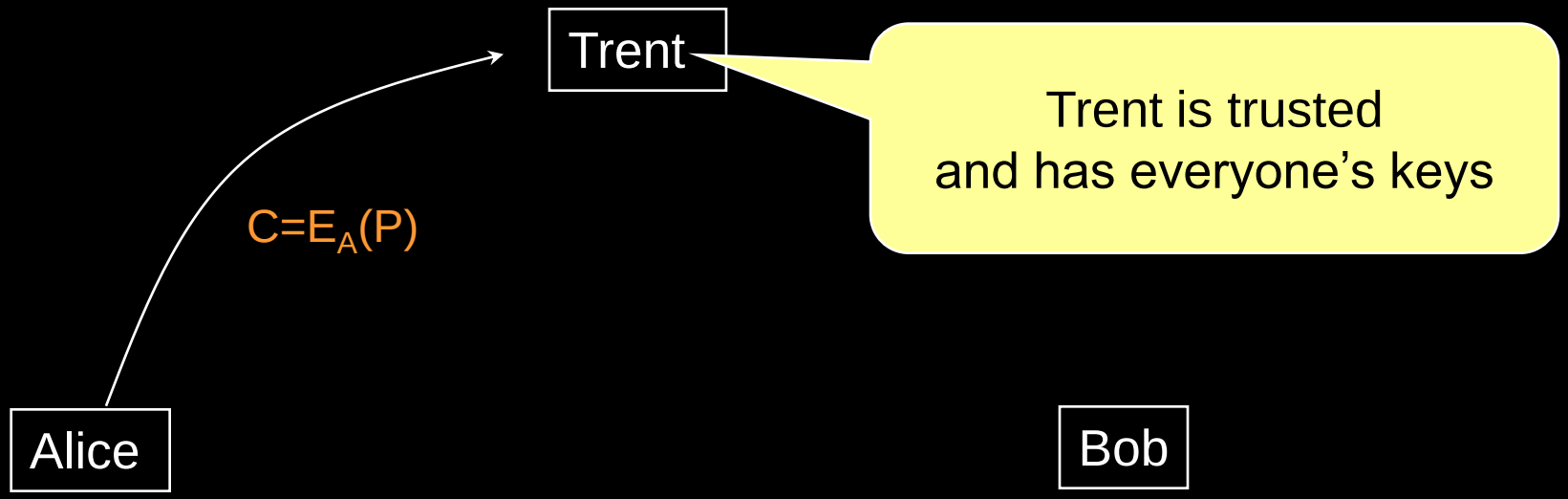
ALL UNTRUE!

Can we do better with **digital signatures**?

Digital signatures - arbitrated protocol

Arbitrated protocol using symmetric encryption

- turn to trusted third party (arbiter) to authenticate messages



Alice encrypts message for *herself* and sends it to Trent

Digital signatures - arbitrated protocol

Trent

$P = D_A(C)$

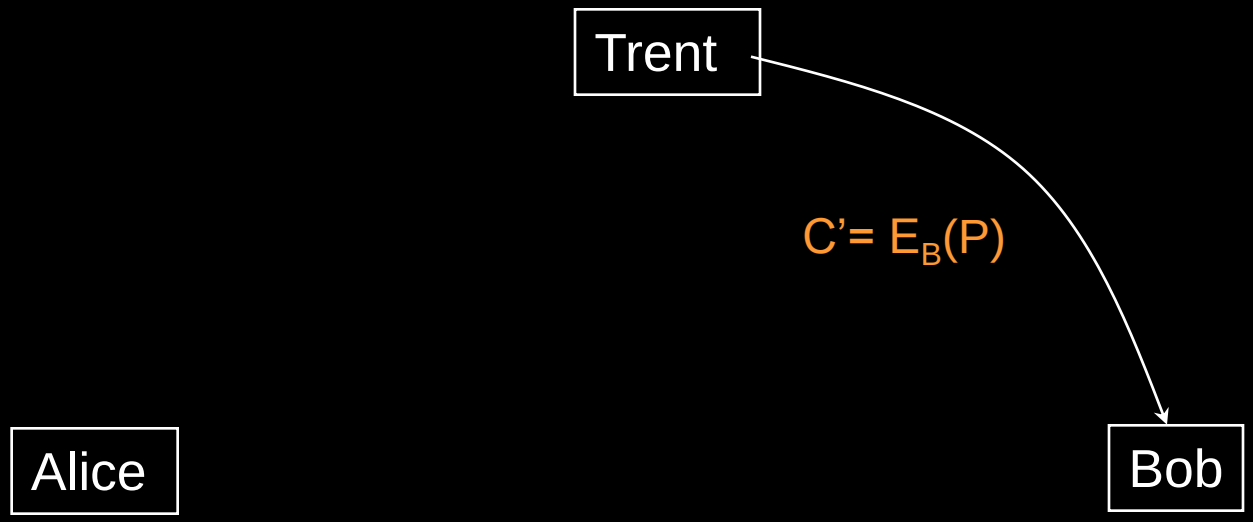
Alice

Bob

Trent receives Alice's message and decrypts it with Alice's key

- this authenticates that it came from Alice
- he may choose to log a hash of the message to create a record of the transmission

Digital signatures - arbitrated protocol



Trent now encrypts the message for Bob and sends it to Bob

Digital signatures - arbitrated protocol

Trent

Alice

Bob

$P' = D_B(C')$

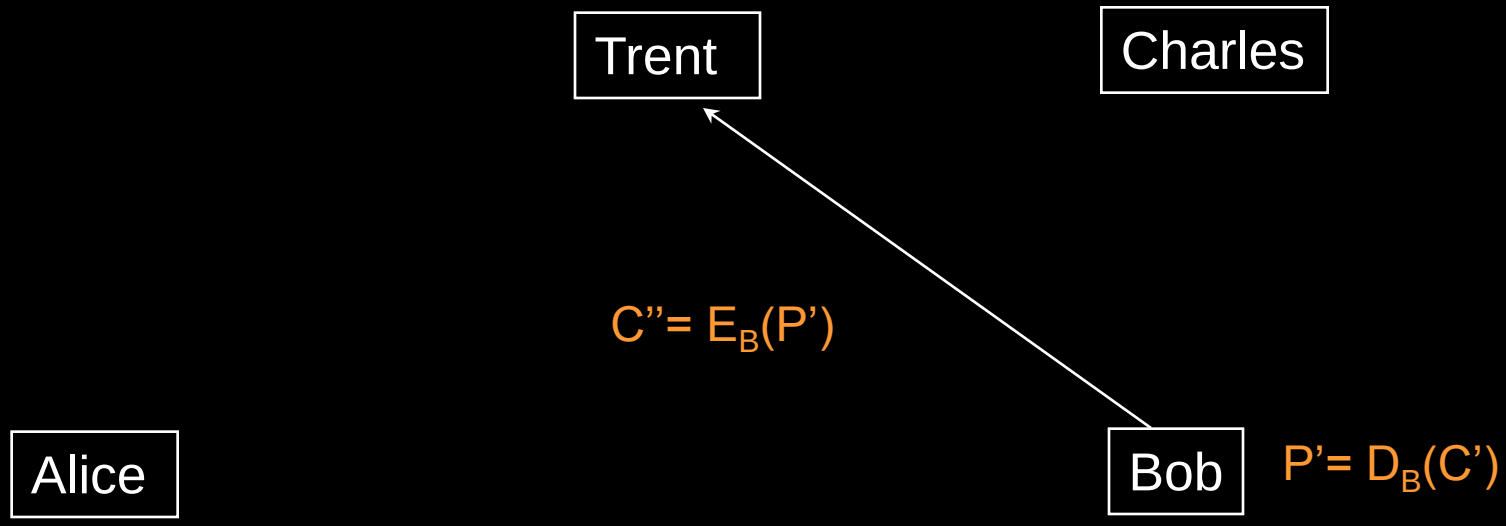
Bob receives the message and decrypts it

- it *must* have come from Trent
  since only Trent and Bob have Bob's key
- if the message says it's from Alice, it must be - we trust Trent

Digital signatures with multiple parties

Bob can forward the message to Charles in the same manner.

Trent can validate stored hash to ensure that Bob did not alter the message



Bob encrypts message with his key and sends it to Trent

Digital signatures with multiple parties

Trent

Charles

$P'' = D_B(C'')$

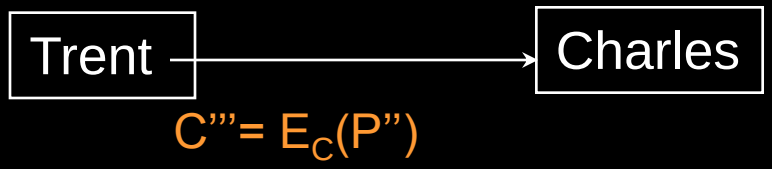
Alice

Bob

Trent decrypts the message

- knows it must be from Bob
- looks up ID to match original hash from Alice's message
- validates that the message has not been modified
- adds a "signed by Bob" indicator to the message

Digital signatures with multiple parties



Alice

Bob

Trent encrypts the new message for Charles

Digital signatures with multiple parties

Trent

Charles

$P''' = D_C(C''')$

Alice

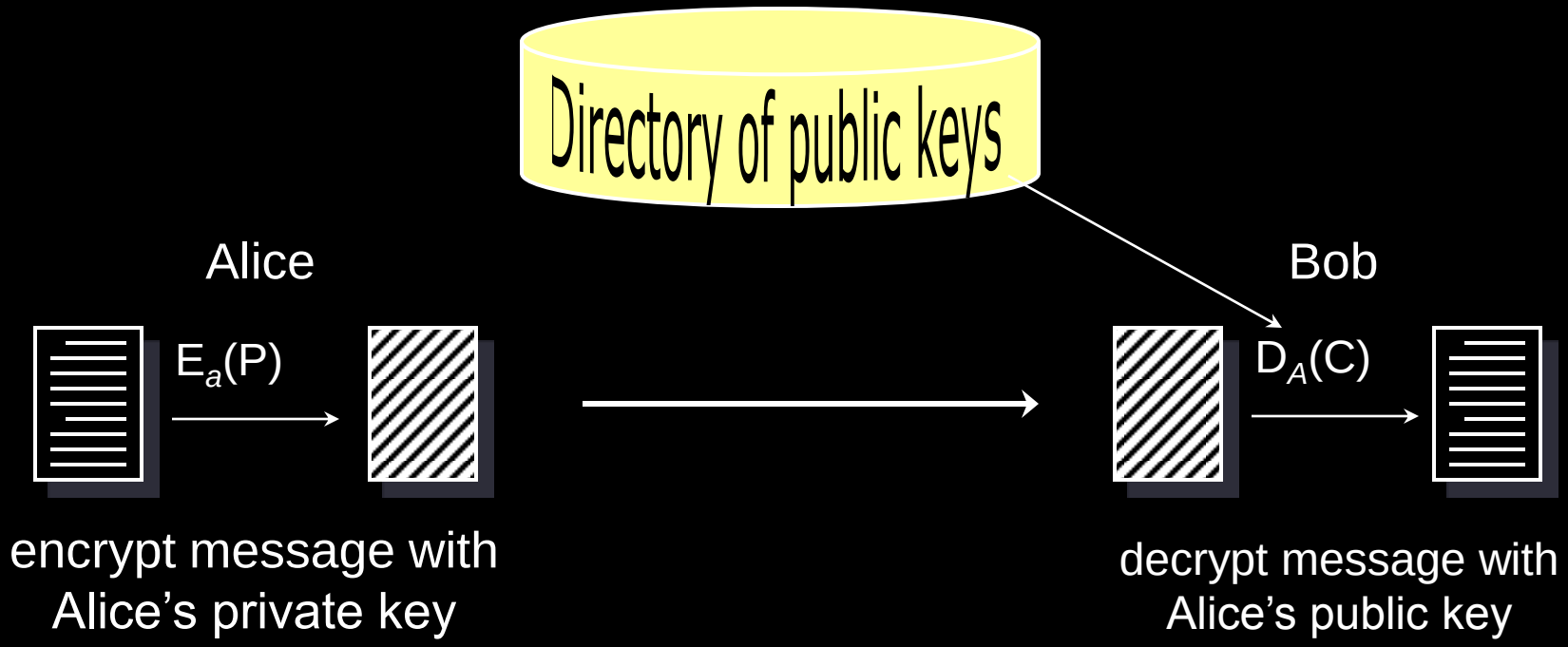
Bob

Charles decrypts the message

- knows the message must have come from Trent
- trusts Trent's assertion that the message originated with Alice and was forwarded through Bob

Digital signatures - public key cryptography

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



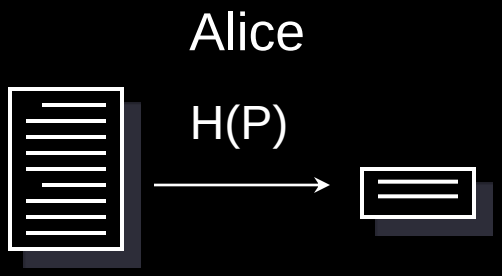
Digital signatures - public key cryptography

- What if Alice was sending Bob binary data?
  - Bob might have a hard time knowing whether the decryption was successful or not
- Public key encryption is considerably slower than symmetric encryption
  - what if the message is very large?
- What if we don't want to hide the message, yet want a valid signature?

Digital signatures - public key cryptography

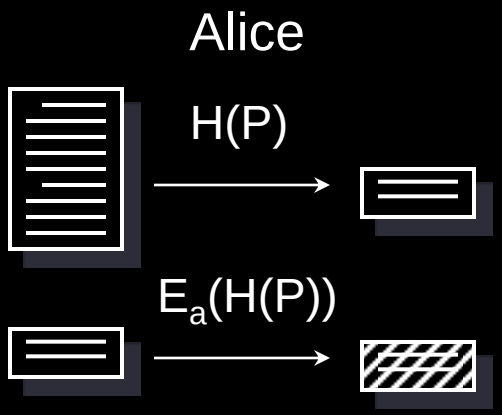
- Create a **hash** of the message
- **Encrypt the hash** and send it with the message
- Validate the hash by decrypting it and comparing it with the hash of the received message

Digital signatures - public key cryptography



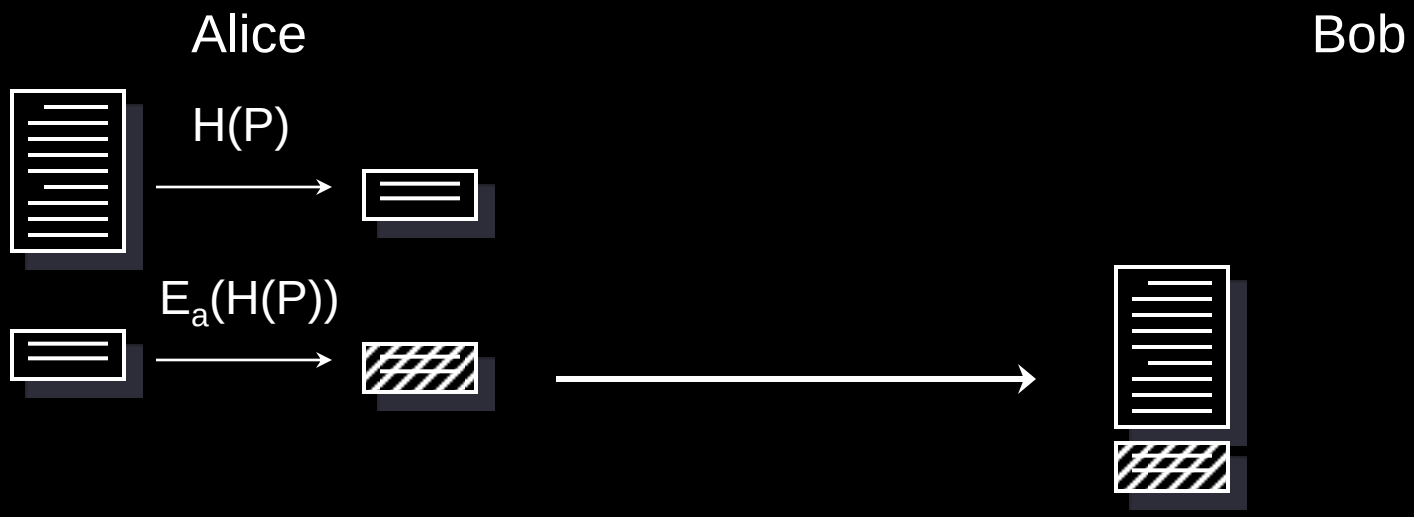
Alice generates a hash of the message

Digital signatures - public key cryptography



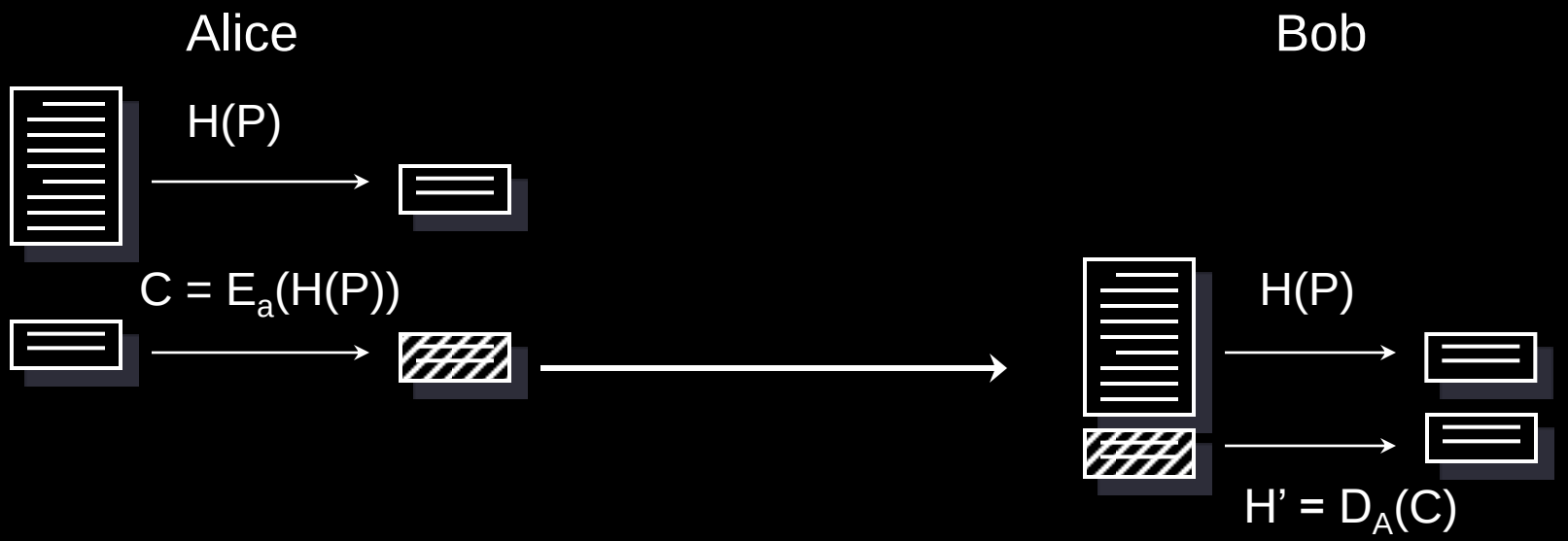
Alice encrypts the hash with her private key

Digital signatures - public key cryptography



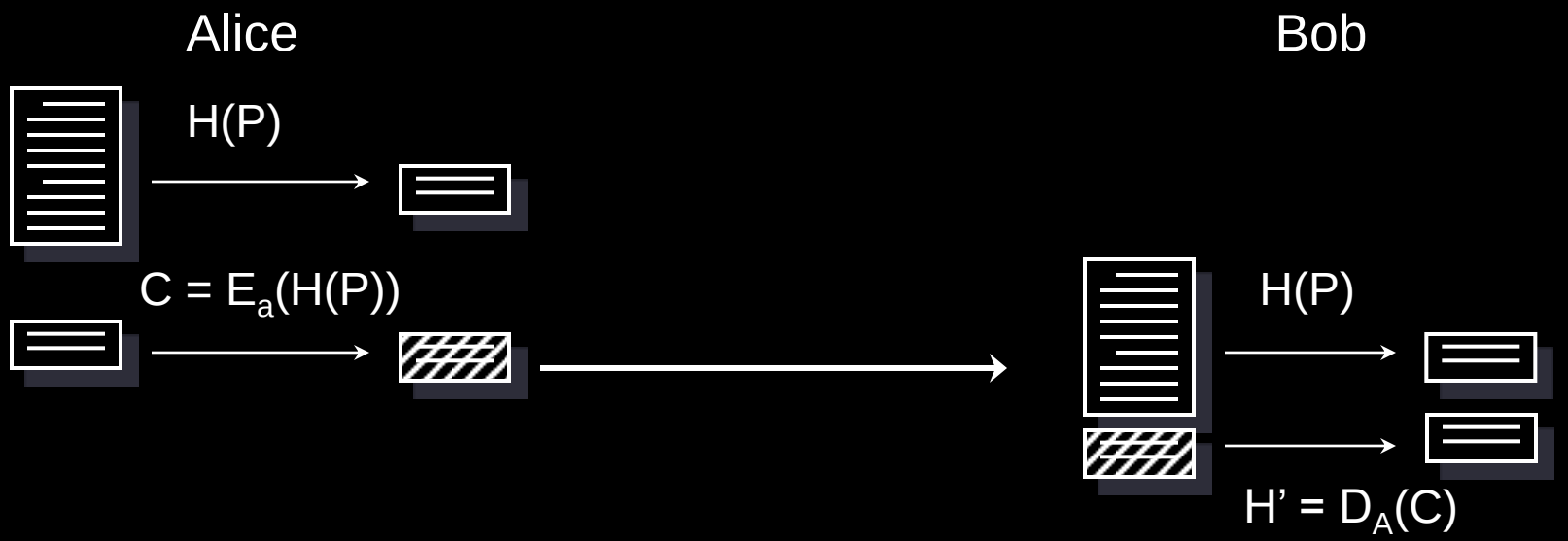
Alice sends Bob the message and the encrypted hash

Digital signatures - public key cryptography



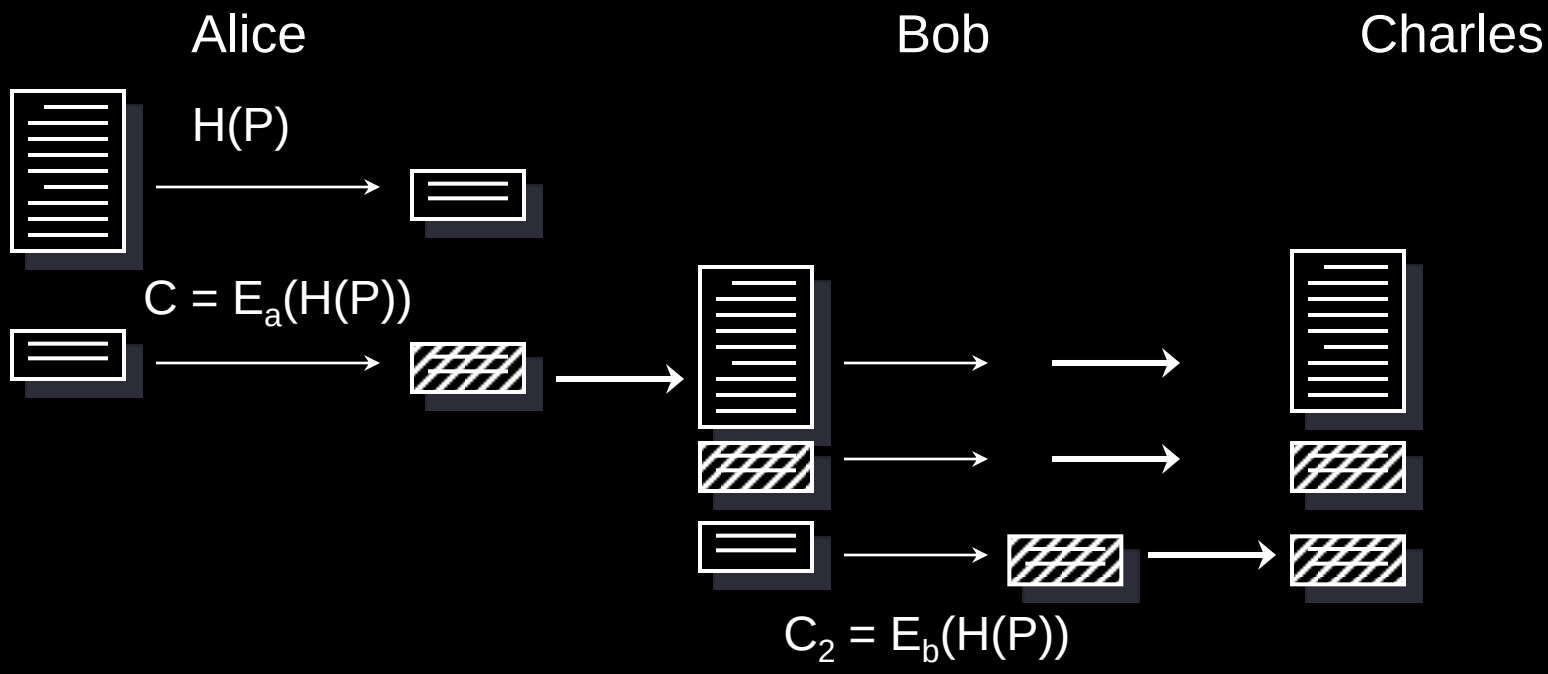
1. Bob decrypts the has using Alice's public key
2. Bob computes the hash of the message sent by Alice

Digital signatures - public key cryptography



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

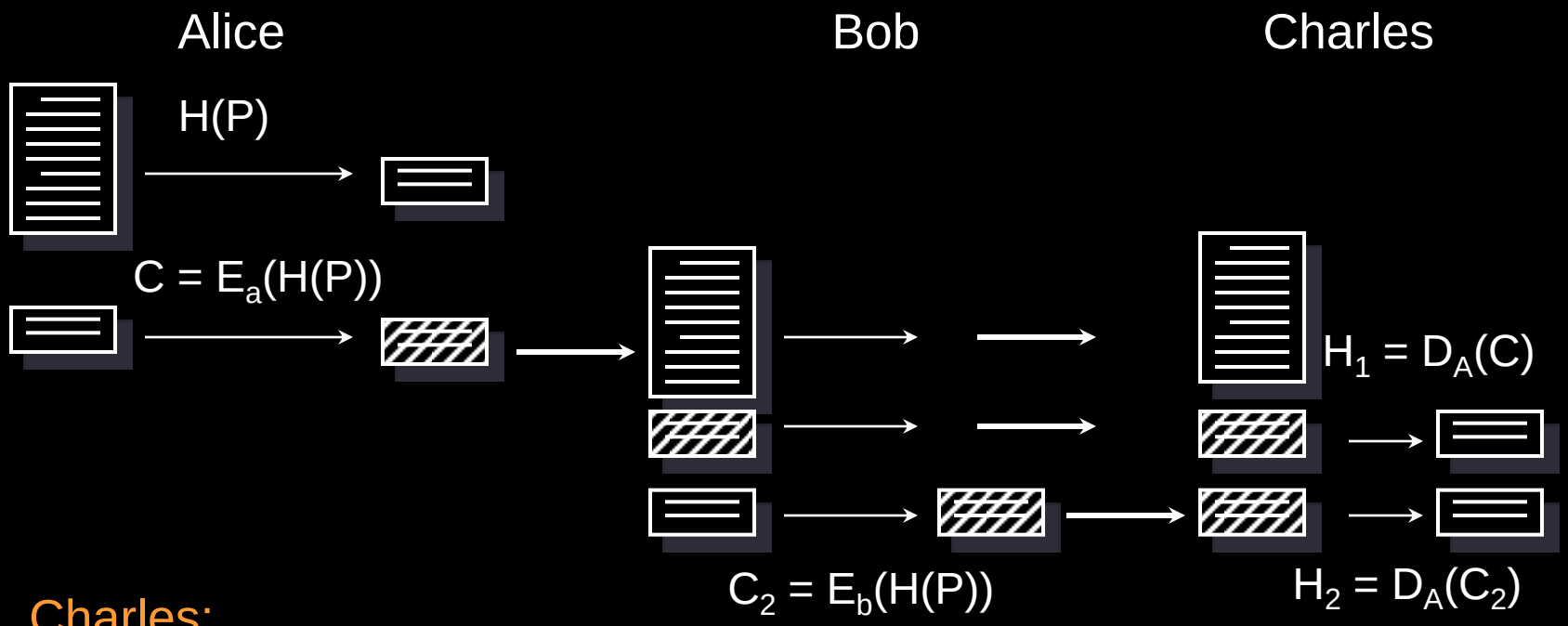
Digital signatures - multiple signers



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:

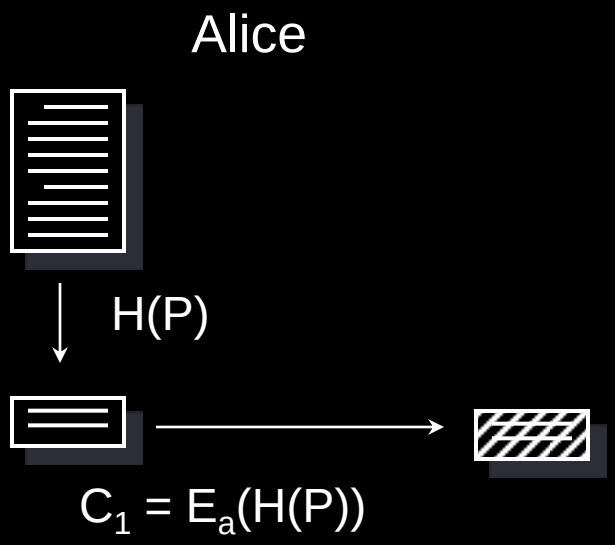
- generates a hash of the message: H(P)
- decrypts Alice's encrypted hash with Alice's public key
  - validates Alice's signature
- decrypts Bob's encrypted hash with Bob's public key
  - validates Bob's signature

Secure and authenticated messaging

If we want secrecy of the message

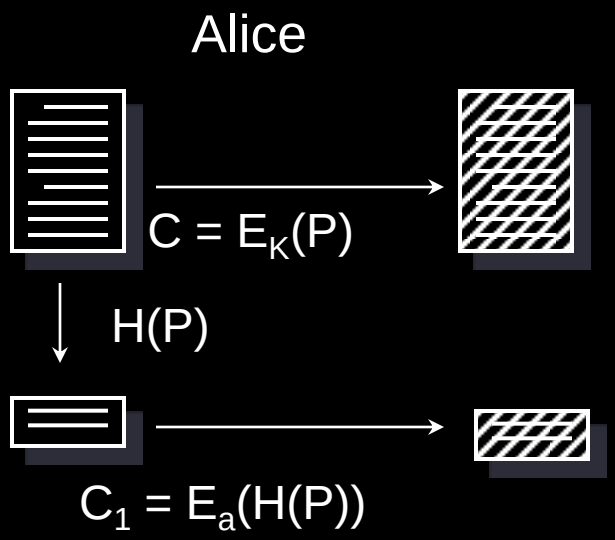
- combine encryption with a digital signature
- use a session key:
  pick a random key, *K*, to encrypt the message with a symmetric algorithm
- encrypt *K* with the public key of each recipient
- for signing, encrypt the hash of the message with sender's private key

Secure and authenticated messaging



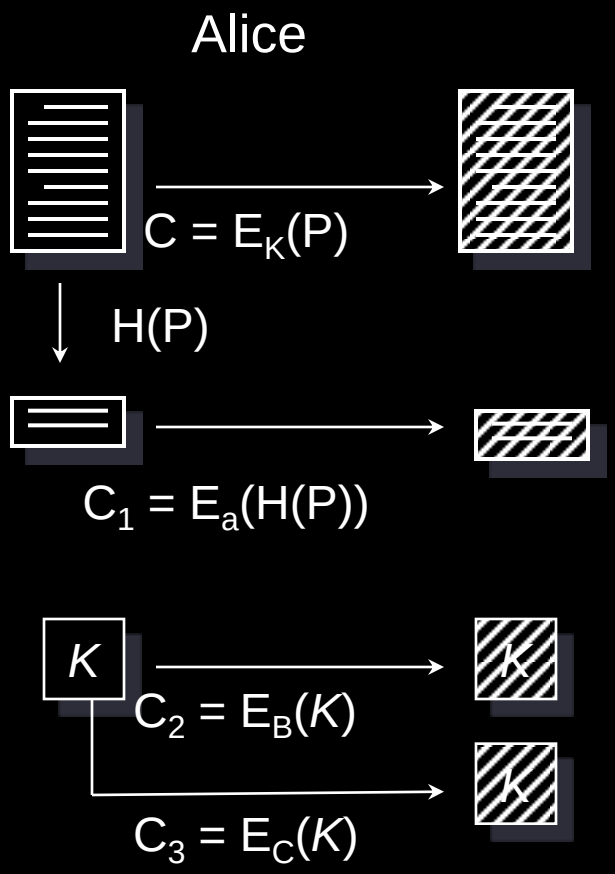
Alice generates a digital signature by encrypting the message digest with her private key.

Secure and authenticated messaging



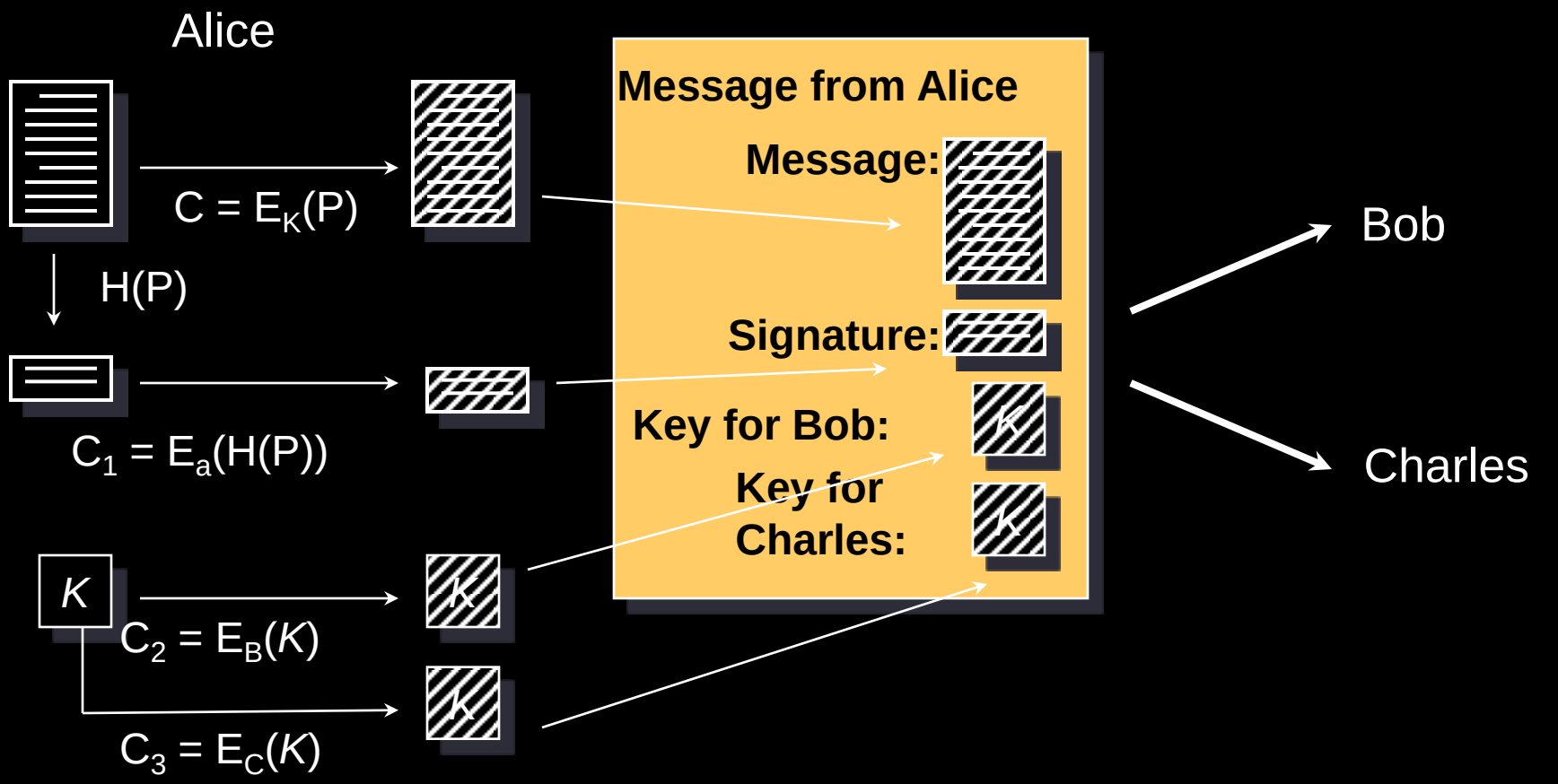
Alice picks a random key, *K*, and encrypts the message (P) with it using a symmetric algorithm.

Secure and authenticated messaging



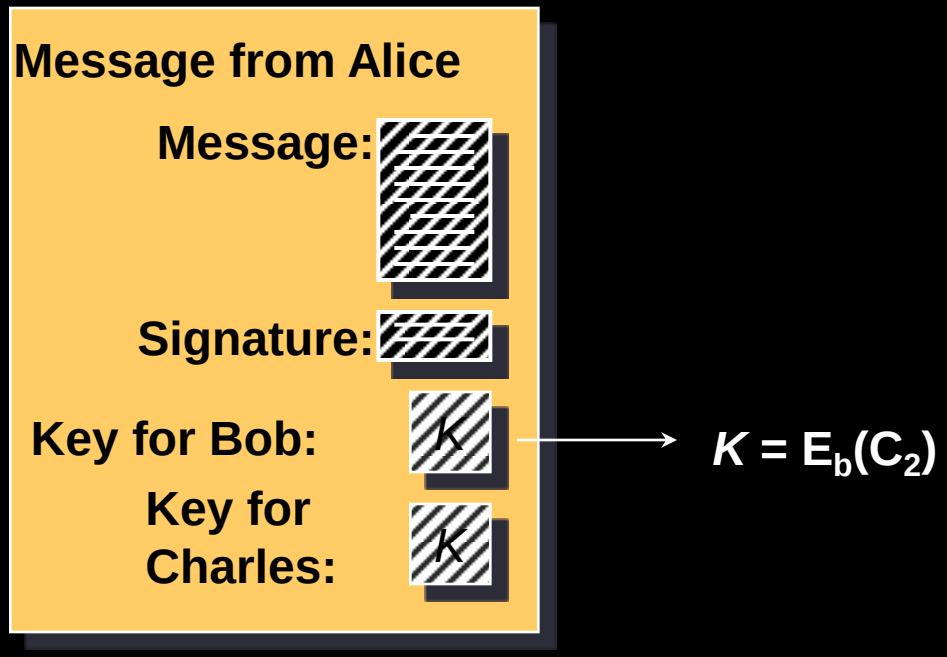
Alice encrypts the session key for each recipient of this message: Bob and Charles using their public keys.

Secure and authenticated messaging



The aggregate message is sent to Bob and Charles

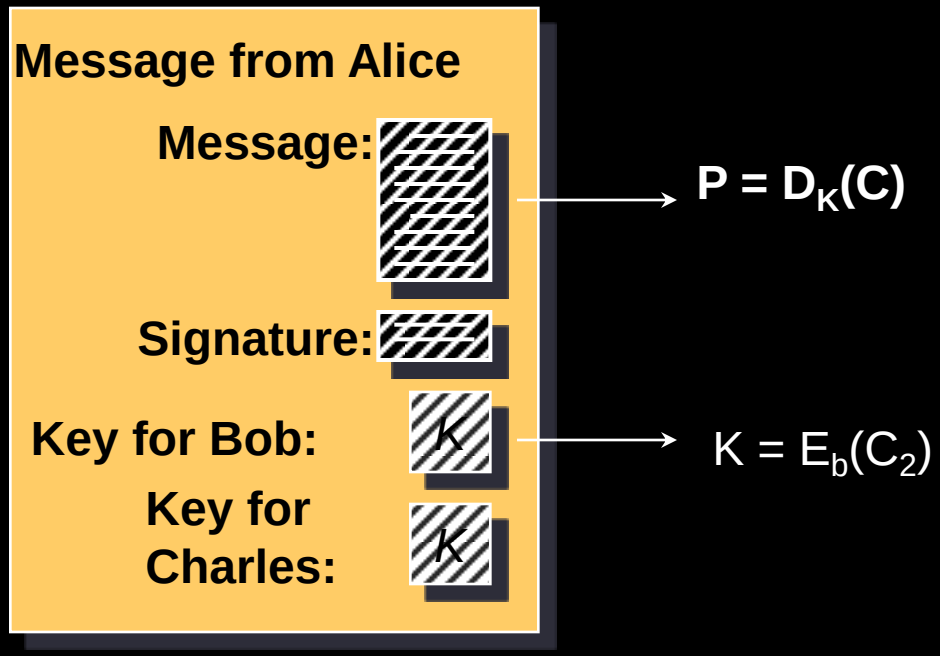
Secure and authenticated messaging



Bob receives the message:

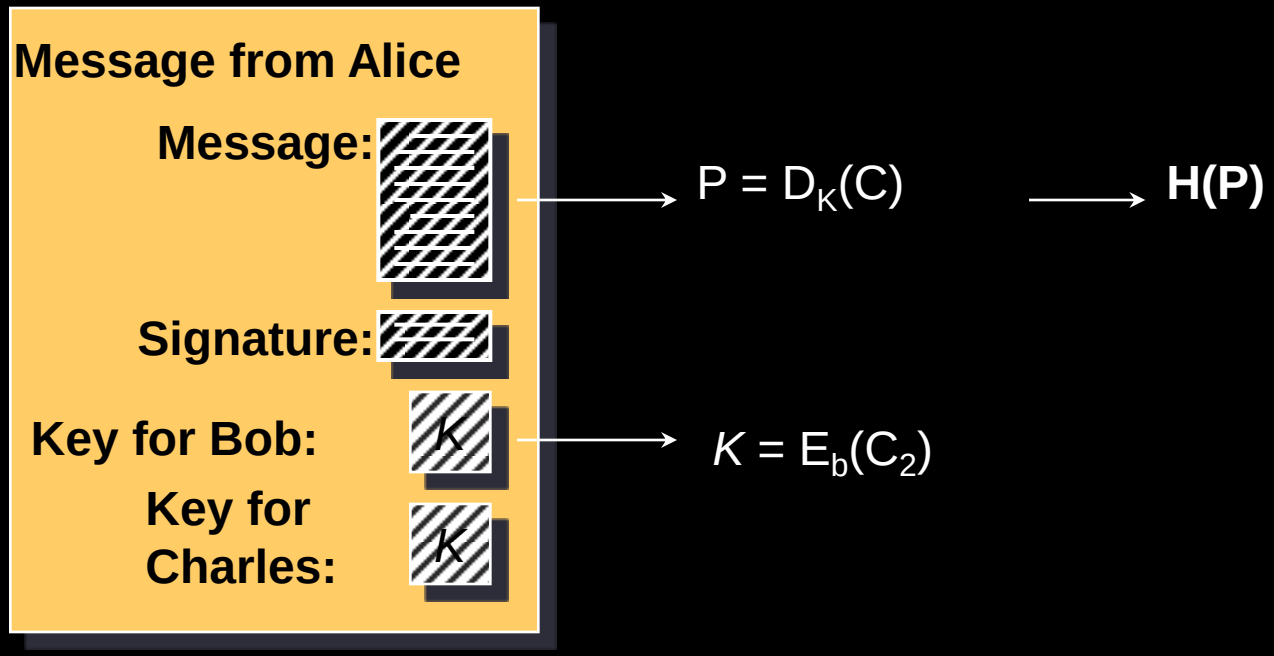
- extracts key by decrypting it with his private key

Secure and authenticated messaging



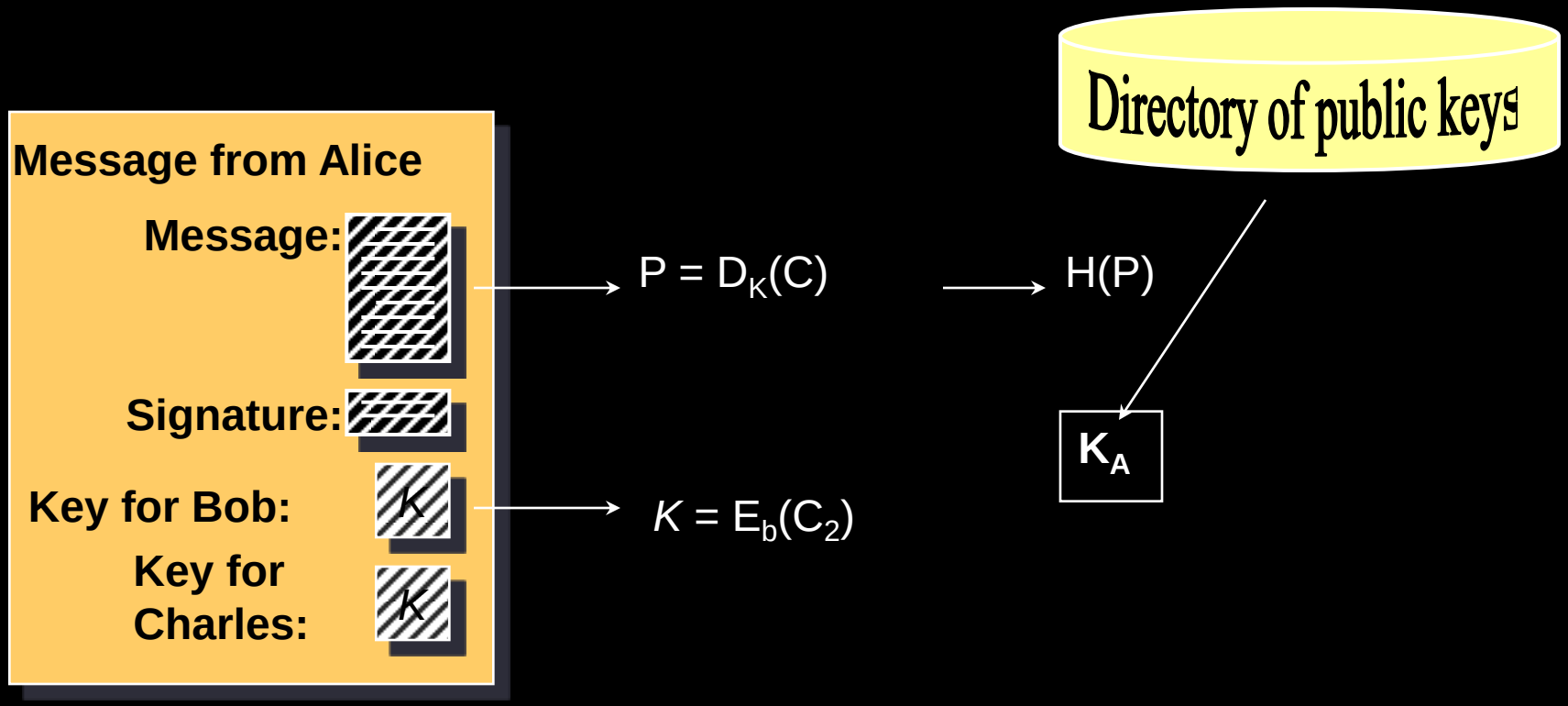
Bob decrypts the message using *K*

Secure and authenticated messaging



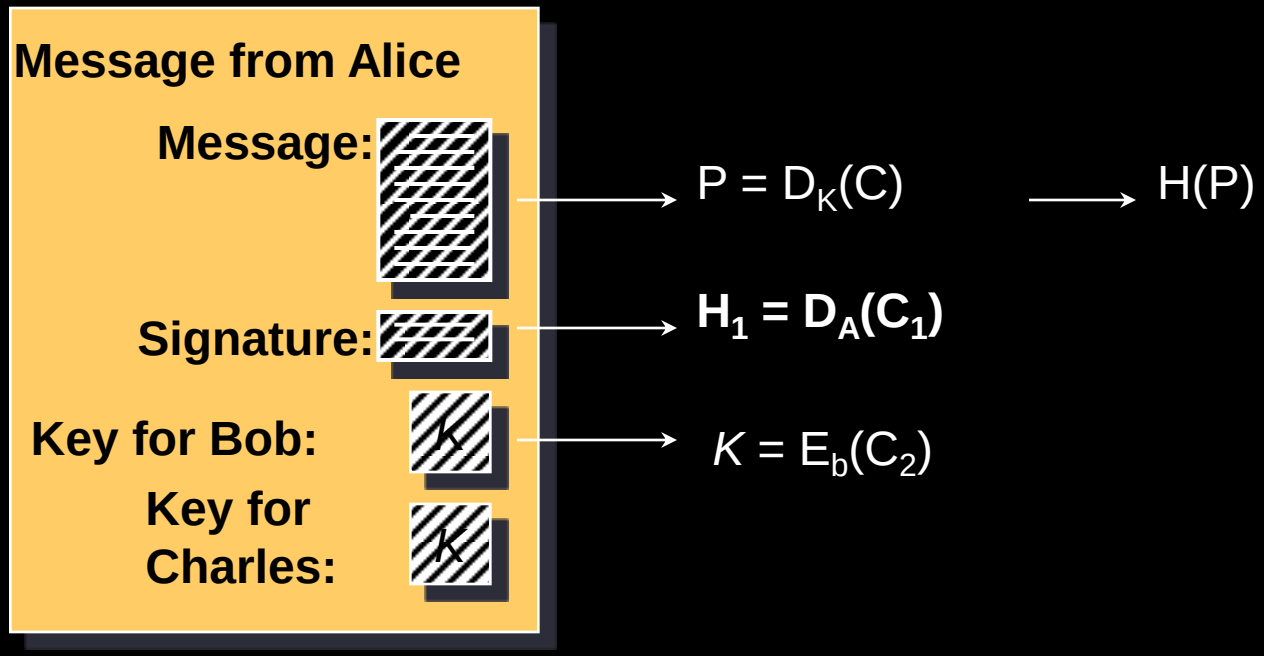
Bob computes the hash of the message

Secure and authenticated messaging



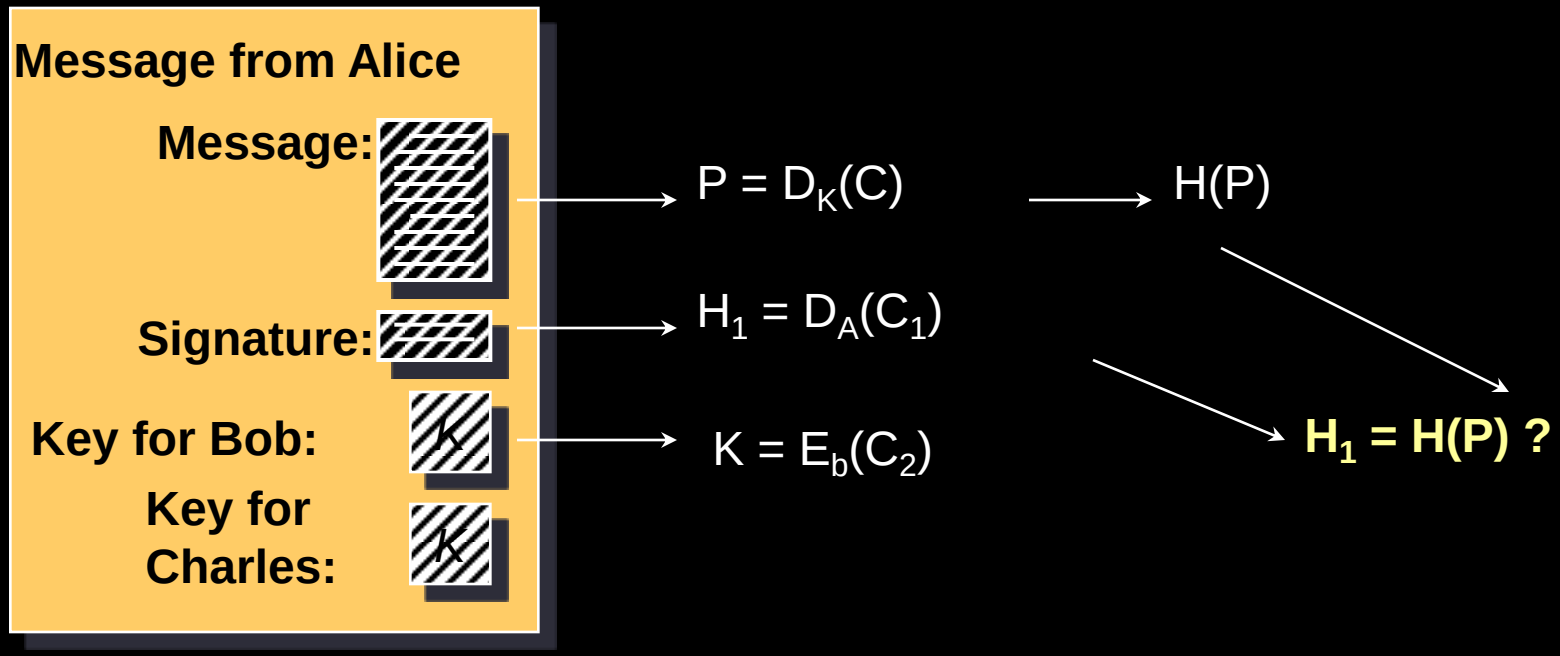
Bob looks up Alice's public key

Secure and authenticated messaging



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

Secure and authenticated messaging



Bob validates Alice's signature

Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- One-way hash functions
- Random number generators
  - Nonces, session keys

Examples

- Key exchange
  - Public key cryptography
- Key exchange + secure communication
  - Public key + symmetric cryptography
- Authentication
  - Nonce + encryption
- Message authentication codes
  - Hashes
- Digital signature
  - Hash + encryption

The end