Distributed Systems

24. Cryptographic Systems: An Brief Introduction

Paul Krzyzanowski

Rutgers University

Fall 2016

Cryptography ≠ Security

Cryptography may be a component of a secure system

Adding cryptography may not make a system secure

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

Terms

Plaintext (cleartext) message P

Encryption *E(*P*)*

Produces Ciphertext, C = *E(*P*)*

Decryption, P = *D(*C*)*

Cipher = cryptographic algorithm

Terms: types of ciphers

- Restricted cipher
- Symmetric algorithm
- Public key algorithm

Restricted cipher

Secret algorithm

- If you know the algorithm, you can encrypt & decrypt
- Vulnerable to:
  - Leaking
  - Reverse engineering
- Hard to validate its effectiveness (who will test it?)
- Not a viable approach!

Symmetric-key algorithm

- Known algorithm but we introduce a secret parameter – the key
- Same secret key, *K*, for encryption & decryption

$$C = E_K(P)$$

$$P = D_K(C)$$

- Examples: AES, 3DES, IDEA, RC5
- Key length
  - Determines number of possible keys
    - DES: 56-bit key: $2^{56} = 7.2 \times 10^{16}$ keys
    - AES-256: 256-bit key: $2^{256} = 1.1 \times 10^{77}$ keys
  - *Brute force attack*: try all keys

The power of 2

Adding one extra bit to a key doubles the search space

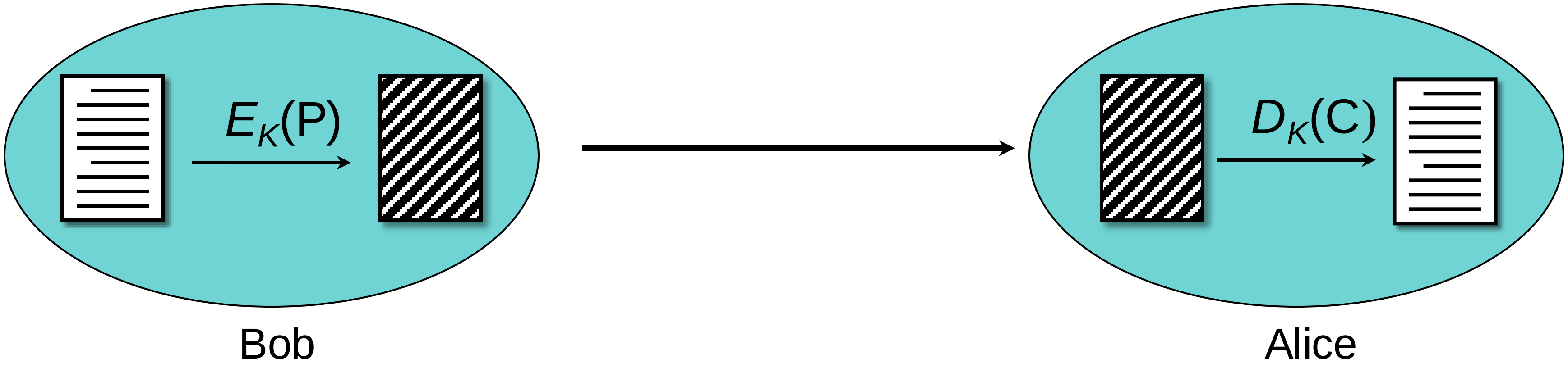
Suppose it takes 1 second to search through all keys with a 20-bit key

key length	number of keys	search time
20 bits	1,048,576	1 second
21 bits	2,097,152	2 seconds
32 bits	4.3×10^{9}	~ 1 hour
56 bits	7.2×10^{16}	2,178 years
64 bits	1.8×10^{19}	> 557,000 years
256 bits	1.2×10^{77}	3.5×10^{63} years

Distributed & custom hardware efforts typically allow us to search between 1 and >100 billion 64-bit (e.g., RC5) keys per second

Communicating with symmetric cryptography

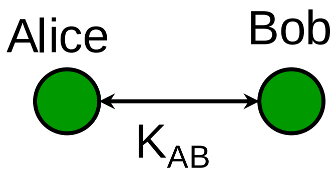
- Both parties must agree on a secret key, *K*
- Message is encrypted, sent, decrypted at other side



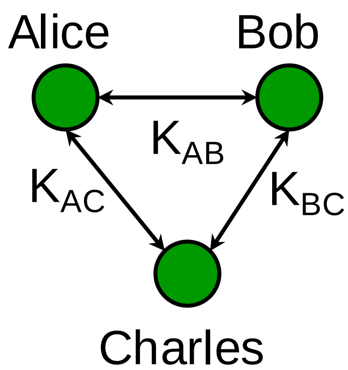
- Key distribution must be secret
  - otherwise messages can be decrypted
  - users can be impersonated

Key explosion

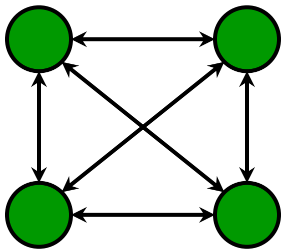
Each pair of users needs a separate key for secure communication



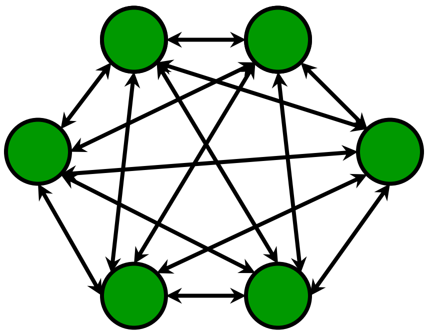
2 users: 1 key



3 users: 3 keys



4 users: 6 keys



6 users: 15 keys

100 users: 4,950 keys

1000 users: 399,500 keys

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

Key distribution

Secure key distribution is the biggest problem with symmetric cryptography

Diffie-Hellman Key Exchange

Key distribution algorithm

- First algorithm to use public/private "keys"
- *Not* public key encryption
- Uses a **one-way function**
  Based on difficulty of computing discrete logarithms in a finite field compared with ease of calculating exponentiation

Allows us to negotiate a secret **common key** without fear of eavesdroppers

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) $^{\textit{\textbf{(Alice's private key)}}}$ ***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

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

- 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:

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

K = K'

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

RSA Public Key Cryptography

- Ron Rivest, Adi Shamir, Leonard Adleman created a public key encryption algorithm in 1977
- Each user generates two keys:
  - Private key (kept secret)
  - Public key (can be shared with anyone)
- Algorithm based on the difficulty of factoring large numbers
  - keys are functions of a pair of large (~300 digits) prime numbers

Public-key algorithm

Two related keys:

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

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

K_1 is a public key

K_2 is a private key

Examples:

- RSA and Elliptic curve algorithms
- DSS (digital signature standard)

Key length

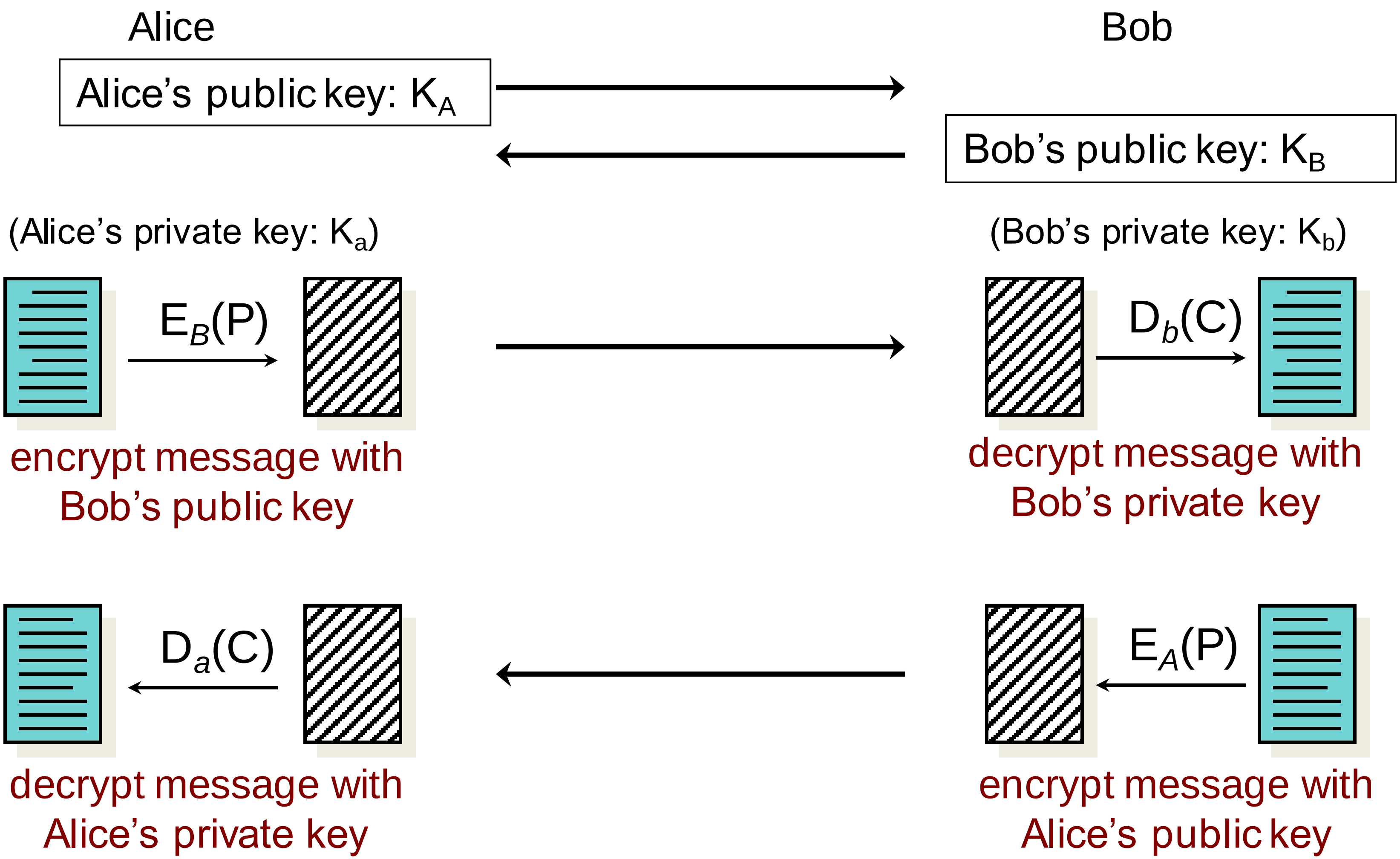
- Unlike symmetric cryptography, not every number is a valid key
- 3072-bit RSA = 256-bit elliptic curve = 128-bit symmetric cipher
- 15360-bit RSA = 521-bit elliptic curve = 256-bit symmetric cipher

Communication with public key algorithms

Different keys for encrypting and decrypting

- No need to worry about key distribution
- Share public keys
- Keep private keys secret

Communication with public key algorithms



Hybrid Cryptosystems

- Session key: randomly-generated key for one communication session
- Use a public key algorithm to send the session key
- Use a symmetric algorithm to encrypt data with the session key

Public key algorithms are almost never used to encrypt messages

– MUCH slower; vulnerable to *chosen-plaintext attacks*

– RSA-2048 approximately 55x slower to encrypt and 2,000x slower to decrypt than AES-256

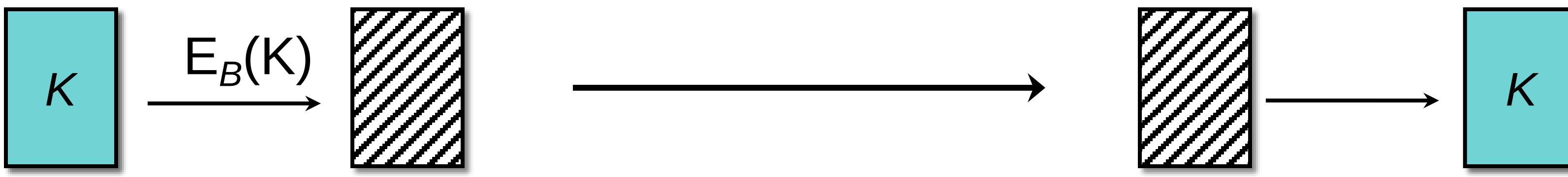
Communication with a hybrid cryptosystem

Alice

Bob

Bob's public key: K_B

Pick a random **session key**, *K*



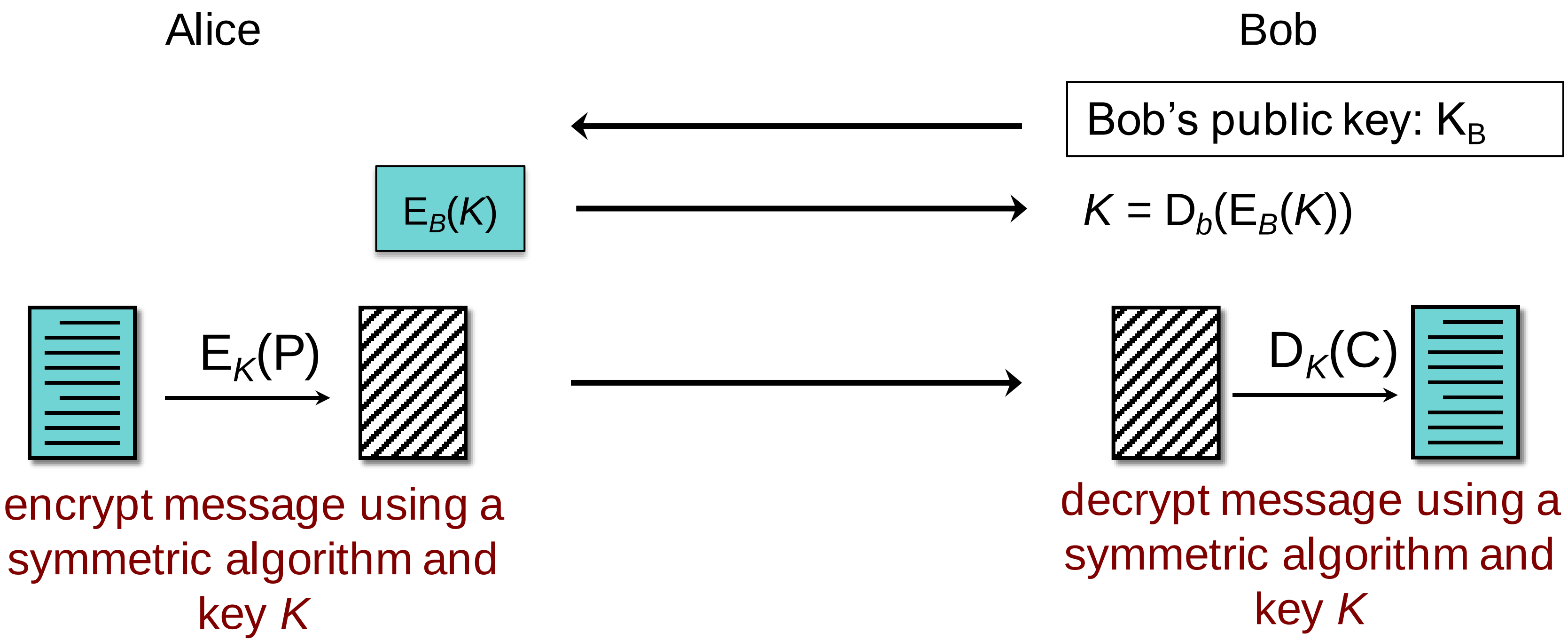
encrypt session key with Bob's public key

$K = D_b(E_B(K))$

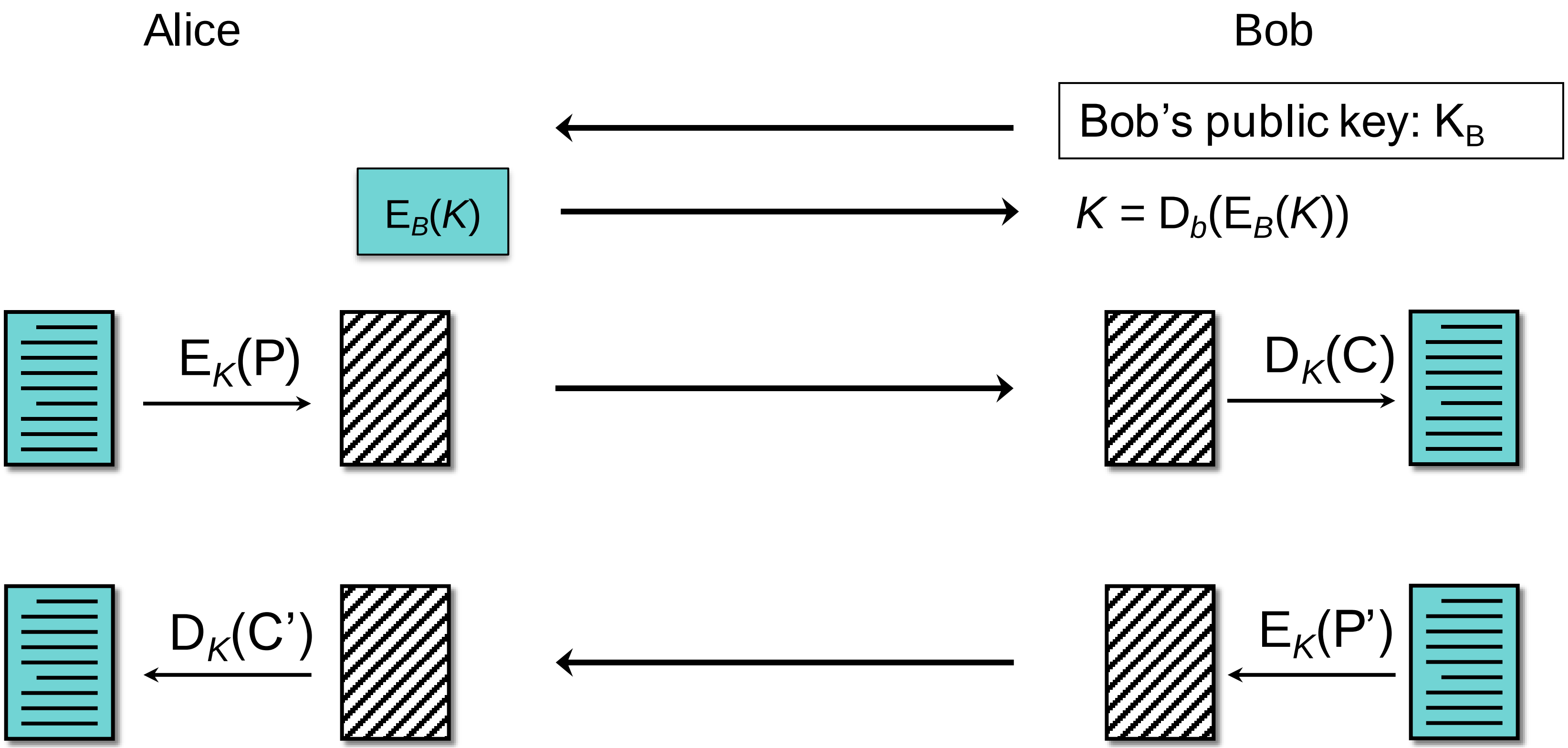
Bob decrypts *K* with his private key

Now Bob knows the secret session key, K

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*

Message Authentication

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

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 difficult to compute A

"Difficult" = no known short-cuts; requires an exhaustive search

Message Integrity: Digital Signatures

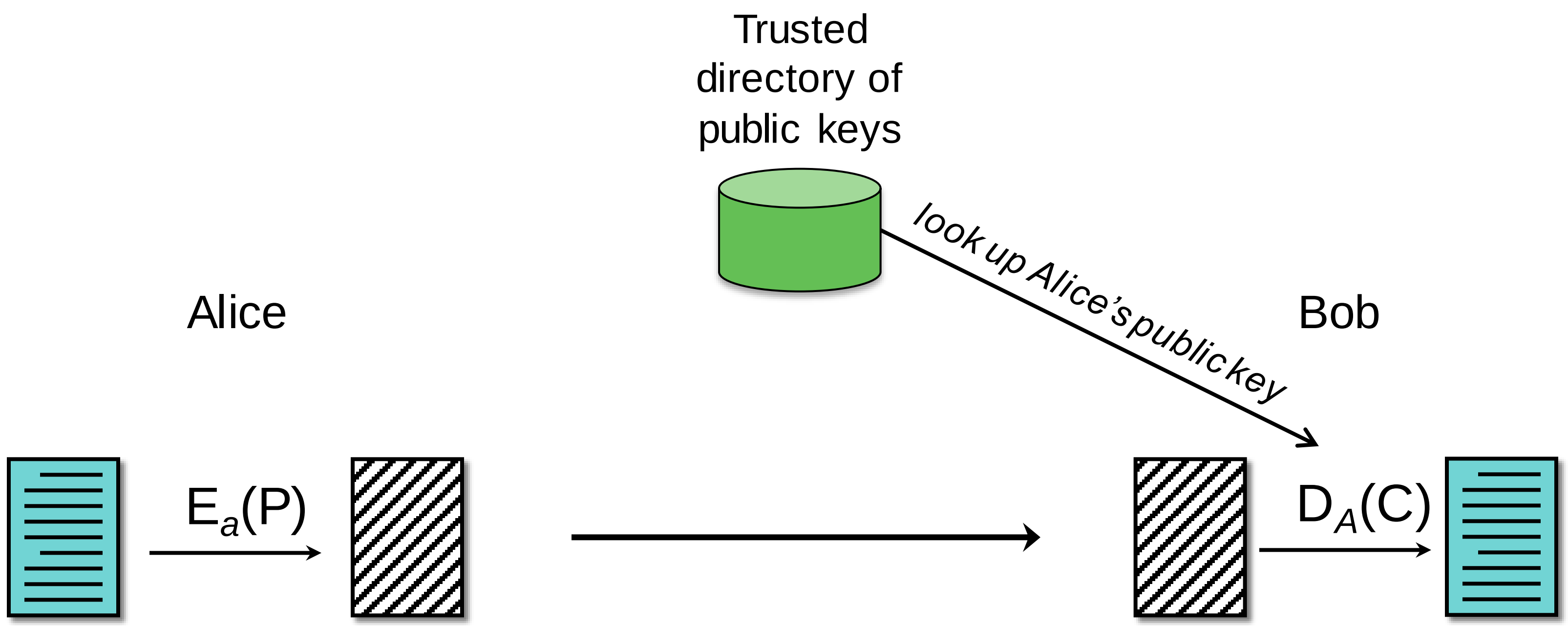
Validate:

1. The creator (signer) of the content
2. The content has not been modified since it was signed

The content itself does not have to be encrypted

Digital Signatures: Public Key Cryptography

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



Encrypt message with Alice's private key

Decrypt message with Bob's public key

But…

- Not quite what we want
  - We don't want to permute or hide the content
  - We just want Bob to verify that the content came from Alice
- Moreover...
  - Public key cryptography is much slower than symmetric encryption
  - What if Alice sent Bob a multi-GB movie?

Hash functions

- Cryptographic hash function (also known as a digest)
  - Input: arbitrary data
  - Output: fixed-length bit string
- Properties
  - **One-way function**
    - Given *H=hash(M)*, it should be difficult to compute *M*, given *H*
  - **Collision resistant**
    - Given *H=hash(M)*, it should be difficult to find *M',* such that *H=hash(M')*
    - For a hash of length L, a perfect hash would take $2^{(L/2)}$ attempts
  - **Efficient**
    - Computing a hash function should be computationally efficient

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
  - NIST standard as of 2015
- MD5
  - 128 bits (not often used now since weaknesses were found)
- Hash functions deriverd from ciphers:
  - Blowfish (used for password hashing in OpenBSD)
  - 3DES – used for old Linux password hashes

Digital signatures using hash functions

- You:
  - Create a hash of the message
  - Encrypt the hash with your private key & send it with the message
- Recipient:
  - Decrypts the encrypted hash using your public key
  - Computes the hash of the received message
  - Compares the decrypted hash with the message hash
  - If they’re the same then the message has not been modified

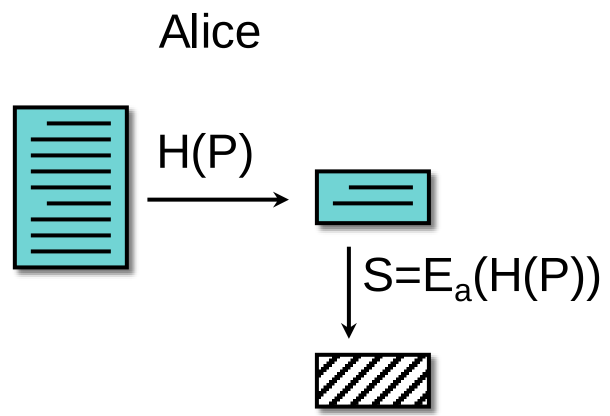
Message Authentication Codes vs. Signatures

- **Message Authentication Code (MAC)**
  - Hash of message encrypted with a symmetric key:
    An intruder will not be able to replace the hash value
- **Digital Signature**
  - Hash of message encrypted with the owner's private key
    - Alice encrypts the hash with her private key
    - Bob validates it by decrypting it with her public key & comparing with *hash(M)*
  - Provides non-repudiation: recipient cannot change the encrypted hash

Digital signatures: public key cryptography

Alice generates a hash of the message

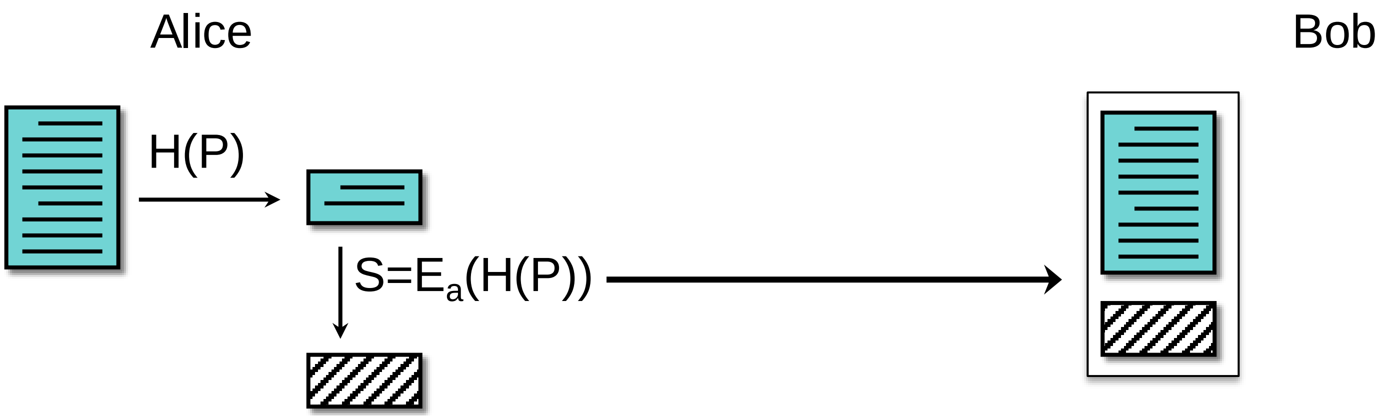
Digital signatures: public key cryptography



Alice encrypts the hash with her private key

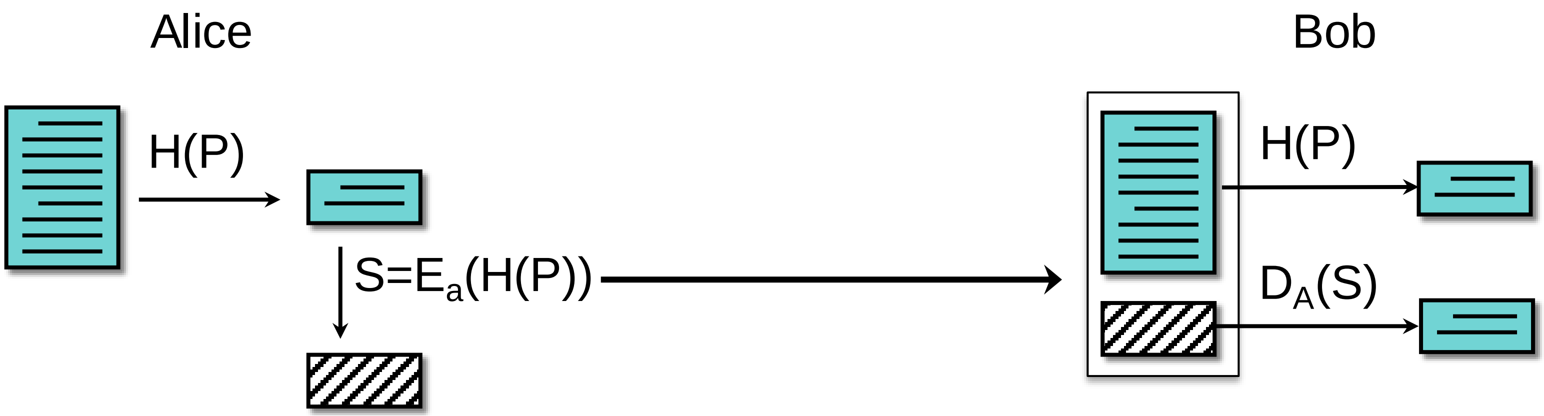
This is her **signature**.

Digital signatures: public key cryptography



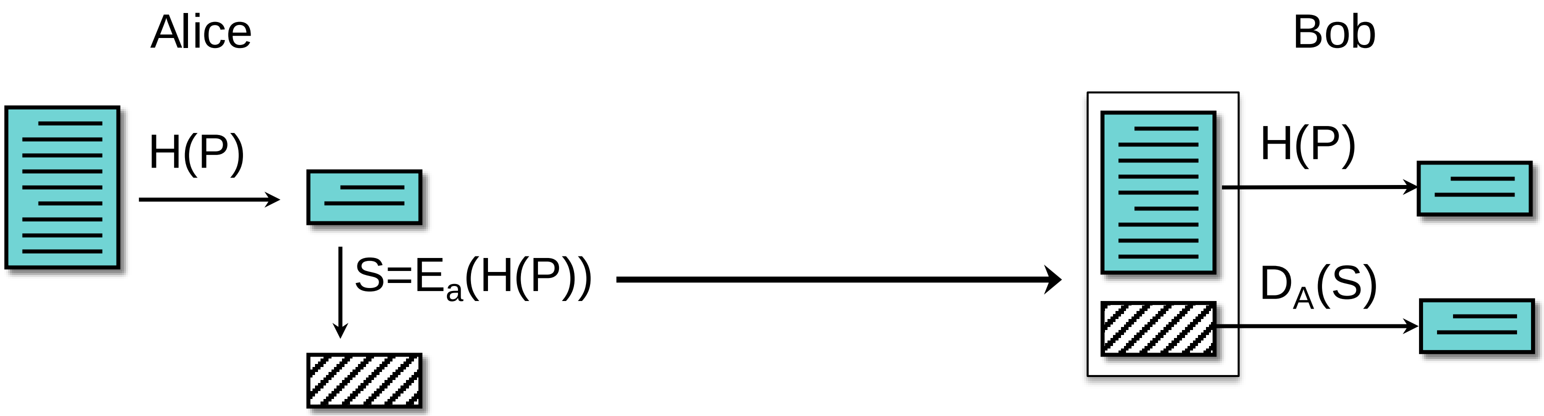
Alice sends Bob the message & the encrypted hash

Digital signatures: public key cryptography



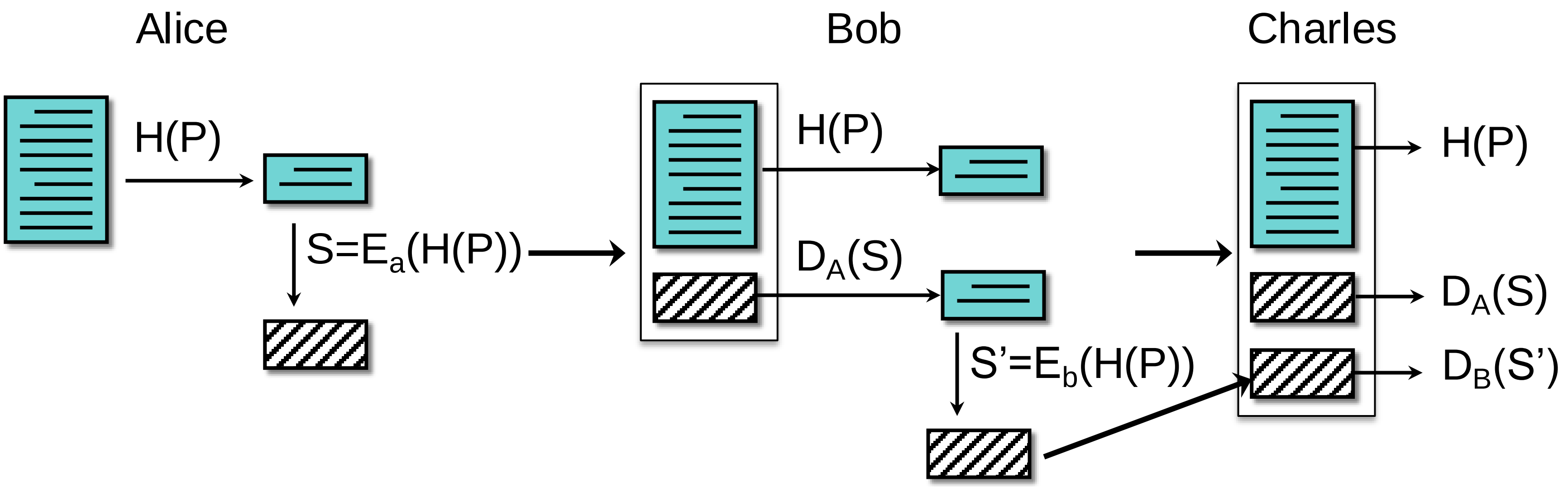
1. Bob decrypts the hash using Alice's public key
2. Bob computes the hash of the message sent by Alice

Digital signatures: public key cryptography



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

Digital signatures: multiple signers



Charles:

- Generates a hash of the message, H(P)
- Decrypts Alice's signature with Alice's public key
  - Validates the signature: $D_A(S) \stackrel{?}{=} H(P)$
- Decrypts Bob's signature with Bob's public key
  - Validates the signature: $D_B(S) \stackrel{?}{=} H(P)$

Covert AND authenticated messaging

If we want to keep the message secret

– combine encryption with a digital signature

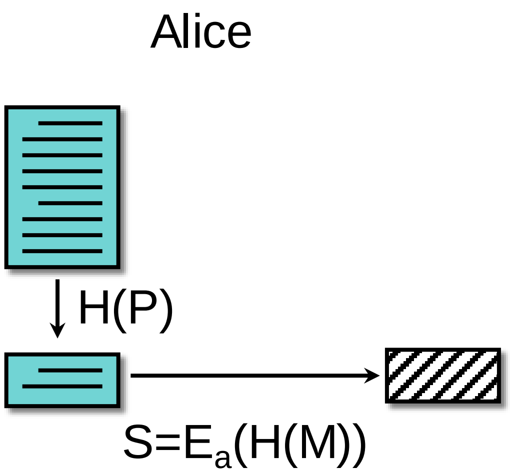
Use a session key:

– Pick a random key, *K*, to encrypt the message with a symmetric algorithm

– encrypt *K* with the public key of each recipient

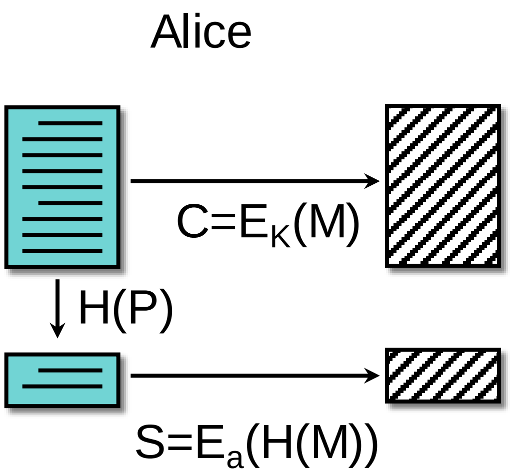
– for signing, encrypt the hash of the message with sender's private key

Covert and authenticated messaging



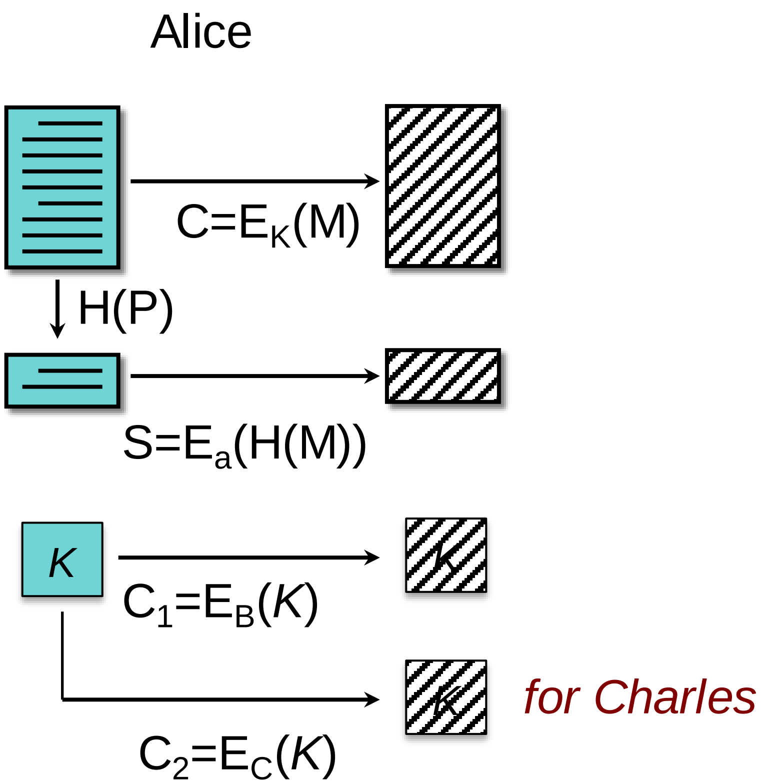
Alice generates a digital signature by
encrypting the message with her private key

Covert and authenticated messaging



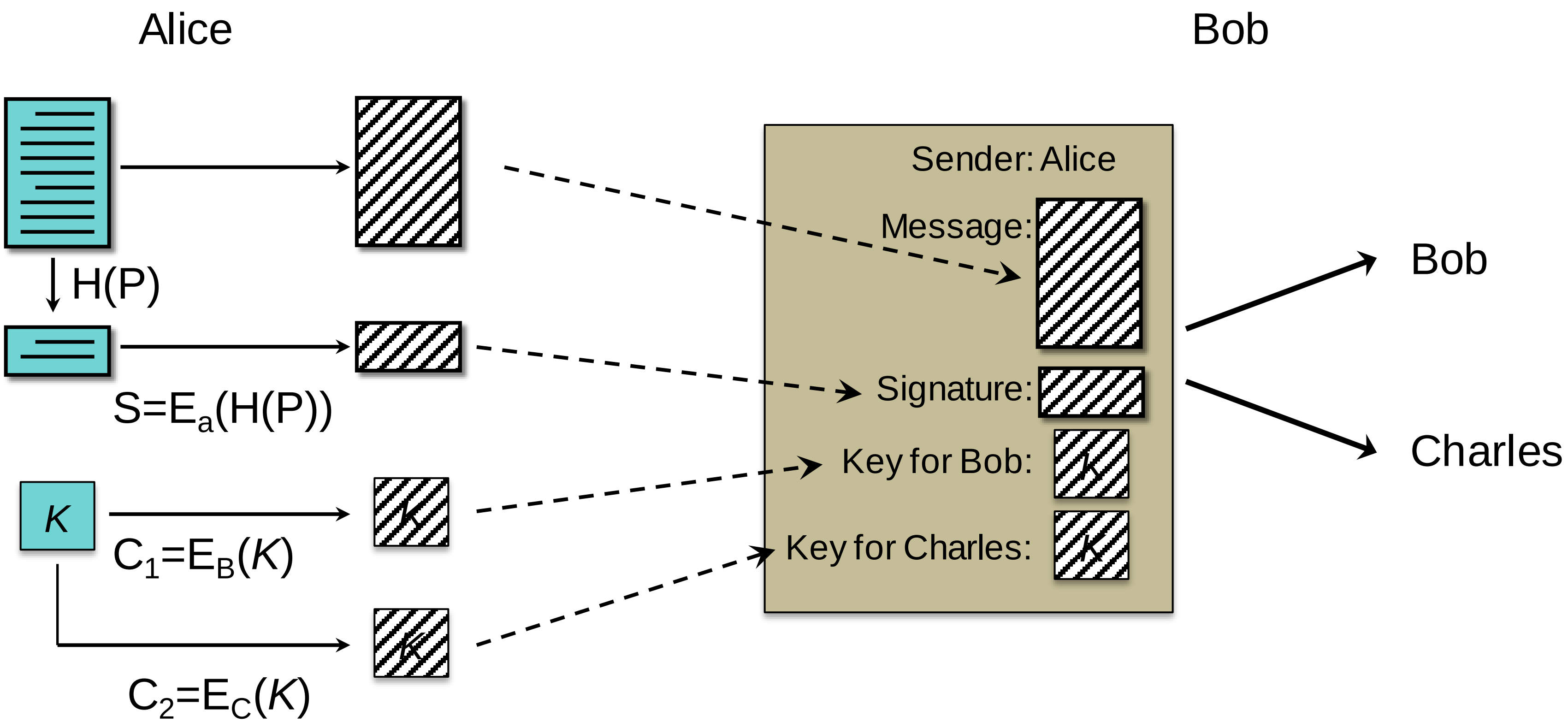
Alice picks a random key, *K*, and encrypts the message *P* with it using a symmetric cipher

Covert and authenticated messaging



Alice encrypts the session key for each recipient of this message using their public keys

Covert and authenticated messaging



The aggregate message is sent to Bob & Charles

Cryptographic toolbox

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

The End