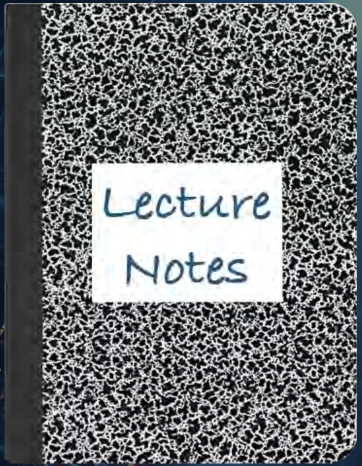
CS 417 – DISTRIBUTED SYSTEMS

Week 12: Security in Distributed Systems

Part 1: Cryptography Intro



Paul Krzyzanowski

© 2023 Paul Krzyzanowski. No part of this content may be reproduced or reposted in whole or in part in any manner without the permission of the copyright owner.

Goals of Security

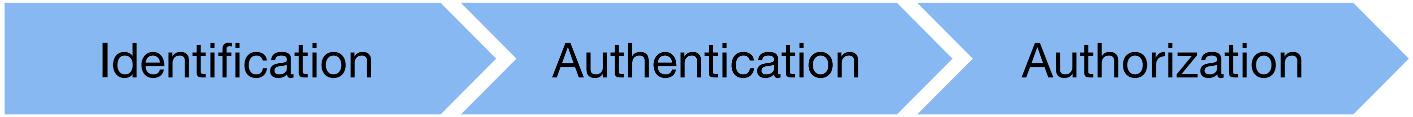
Keep systems, programs, and data secure

The **CIA[*] Triad**:

1. **Confidentiality**

2. **Integrity**

3. **Availability**



[*]No relationship to the Central Intelligence Agency

Confidentiality

- Keep data & resources hidden
  - Data will only be shared with authorized individuals
  - Sometimes – conceal the existence of data or communication
- Traditional focus of computer security

Data confidentiality

“The property that information is not made available or disclosed to unauthorized individuals, entities, or processes [i.e., to any unauthorized system entity].”

– *RFC 4949, Internet Security Glossary*

Integrity

The trustworthiness of the data or resources

- *Preventing unauthorized changes to the data or resources*

- **Data integrity**
  - Data integrity: property that data has not been modified or destroyed in an unauthorized or accidental manner
- **Origin integrity**
  - Authentication
- **System integrity**
  - The ability of a system to perform its intended function, free from deliberate or inadvertent manipulation

Often more important than confidentiality!

Availability

- Being able to use the data or resources
- Property of a system being accessible and capable of working to required performance specifications

Turning off a computer provides confidentiality & integrity but hurts availability

Denial of Service *(DoS) attacks target availability*

Thinking about security

Security is not

just adding encryption

… or using a 512-bit key instead of a 64-bit key

… or changing passwords

… or setting up a firewall

It is a systems issue

= Hardware + firmware + OS + app software + networking + people

= Processes & procedures, policies, detection, forensics

"Security is a chain: it's only as secure as the weakest link"

– Bruce Schneier

The Operating System

The operating system normally handles security

- User authentication – *passwords*
- Access control – *file permissions, system call access*
- Resource management – *memory limits, scheduling*

But it can only control resources it owns

  – Other systems may have different policies

Distributed Control

Distributed systems often use components that belong to different entities

Programs may:

- Call remote services – *are they trustworthy?*
- Receive requests – *are they from a legitimate & authorized user or service?*
- Store data on remote servers – *who manages them*?
- Send data over a network – *what route do the packets take?*

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?

- Confidentiality
  - Others cannot read contents of the message
- 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

Encryption

Plaintext (cleartext) message P

Cipher = cryptographic algorithm

Encryption *E*(P)

Produces Ciphertext, C = *E*(P)

Decryption, P = *D*(C)

Properties of a good cryptosystem

1. Ciphertext should be indistinguishable from random values

2. Given ciphertext, there should be no way to extract the original plaintext or the key short of enumerating all possible keys (i.e., a *brute force attack*)

3. The keys should be large enough that a brute force attack is not feasible

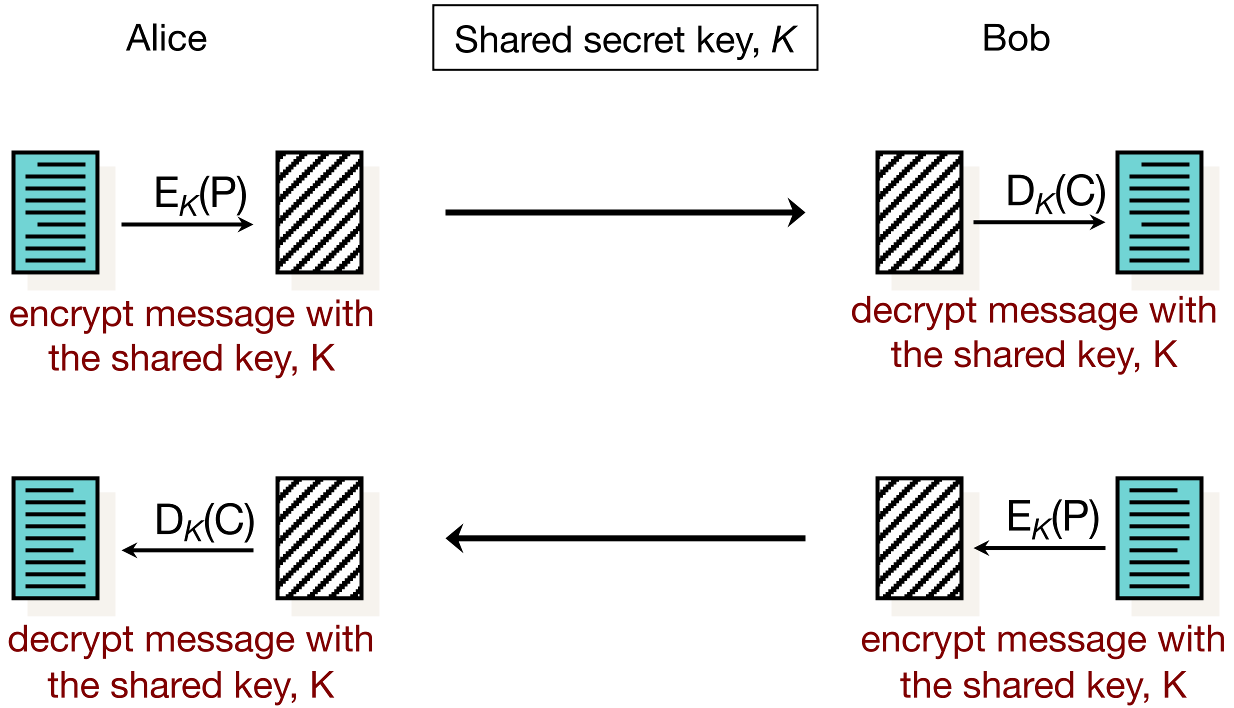
Symmetric key ciphers

Same shared secret key, *K*, for encryption & decryption

$$C = E_K(P)$$

$$P = D_K(C)$$

Communication with a symmetric cipher


Alice
Shared secret key, K
Bob
$E_K(P)$
$D_K(C)$
encrypt message with the shared key, K
decrypt message with the shared key, K
$D_K(C)$
$E_K(P)$
decrypt message with the shared key, K
encrypt message with the shared key, K

Some popular symmetric ciphers

AES (Advanced Encryption Standard)	• FIPS standard since 2002 • 128, 192, or 256-bit keys; operates on 128-bit blocks • By far the most widely used symmetric encryption algorithm
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) • Not actively used anymore; AES is better by any measure
ChaCha20	128 or 256-bit keys – stream cipher – faster than AES on lower-end processors
Twofish	128, 192 or 256 bits – block cipher
IDEA	128-bit keys; operates on 64-bit blocks More secure than DES but faster algorithms are available

Public Key Cryptography

Public-key algorithm

Two related keys (*A*, *a*)

$$C = E_A(P) \qquad P = D_a(C)$$
$$C' = E_a(P) \qquad P = D_A(C')$$

A is a **public** key

a is a **private** key

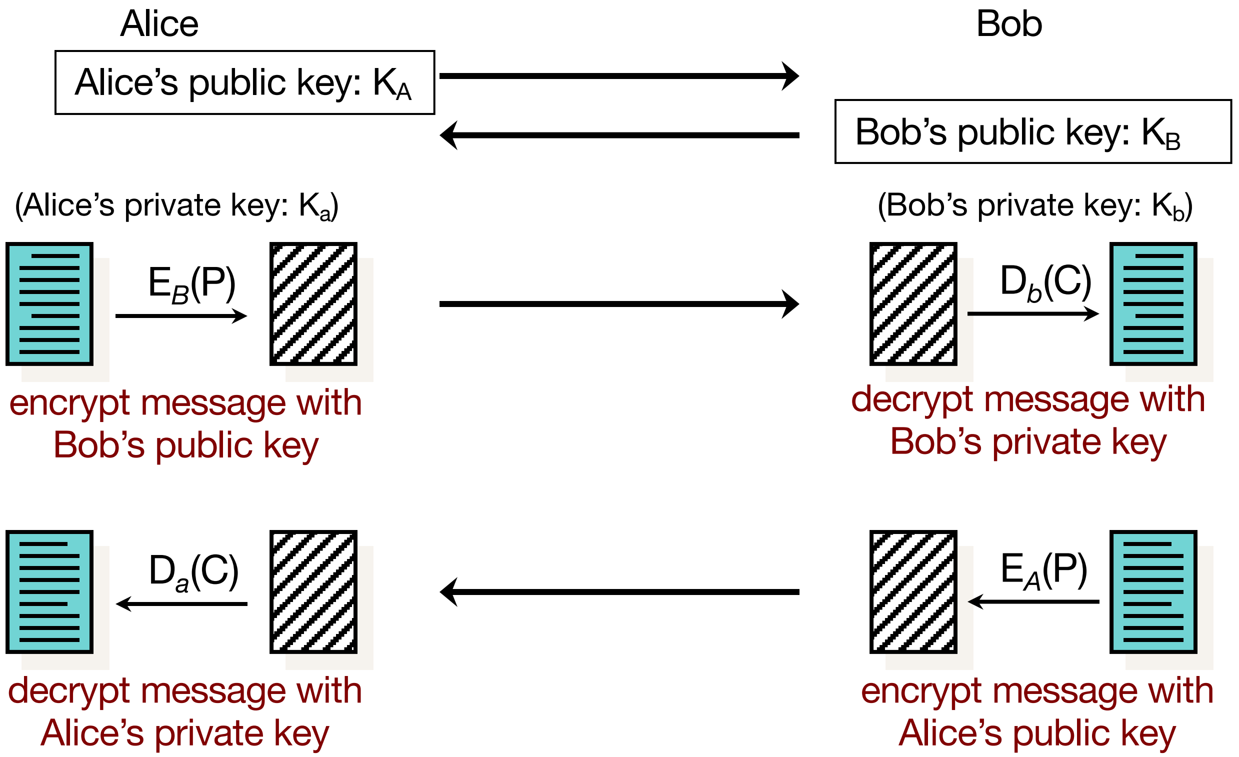
- Examples:
  - RSA
  - Elliptic Curve Cryptography (ECC)
- Key length
  - Unlike symmetric cryptography, not every number is a valid key
  - 3072-bit RSA ≈ 256-bit elliptic curve ≈ 128-bit symmetric cipher
  - 15360-bit RSA ≈ 521-bit elliptic curve ≈ 256-bit symmetric cipher

Communication with public key algorithms

Different keys for encrypting and decrypting

- No need to worry about secure key distribution

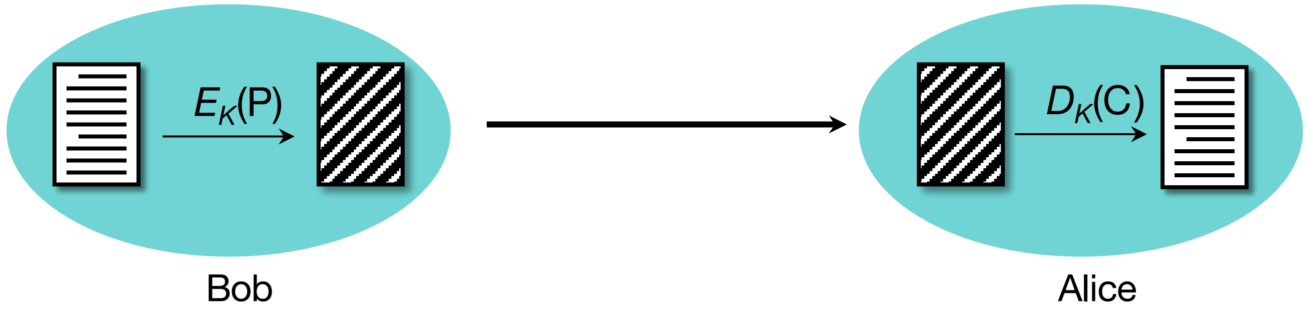
Communication with public key algorithms



Alice
Bob
Alice's public key: K_A
Bob's public key: K_B
(Alice's private key: K_a)
(Bob's private key: K_b)
$E_B(P)$
$D_b(C)$
encrypt message with Bob's public key
decrypt message with Bob's private key
$D_a(C)$
$E_A(P)$
decrypt message with Alice's private key
encrypt message with Alice's public key

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 distribution

Secure key distribution is the biggest problem with symmetric cryptography

Distributing Keys

- **Pre-shared keys**
  - Initial configuration, out of band (send via USB key, recite, …)
- **Trusted third party**
  - Knows all keys
  - Alice creates a temporary key (**session key**)
  - Encrypts it with her key – sends to Trent
  - Trent decrypts it and sends it to Bob
  - Alternatively: Trent creates a session key – encrypts it for Alice & for Bob
- **Public key cryptography**
  - Alice encrypts a message with Bob's public key
  - Only Bob can decrypt
- **Diffie-Hellman**

Permanent vs. Ephemeral Keys

Permanent keys

- Keys you use over and over again – e.g., your password

Ephemeral keys

- Keys that are created spontaneously for one use, such as a communication session, and then never used again

  = **session keys**

Why use ephemeral keys?

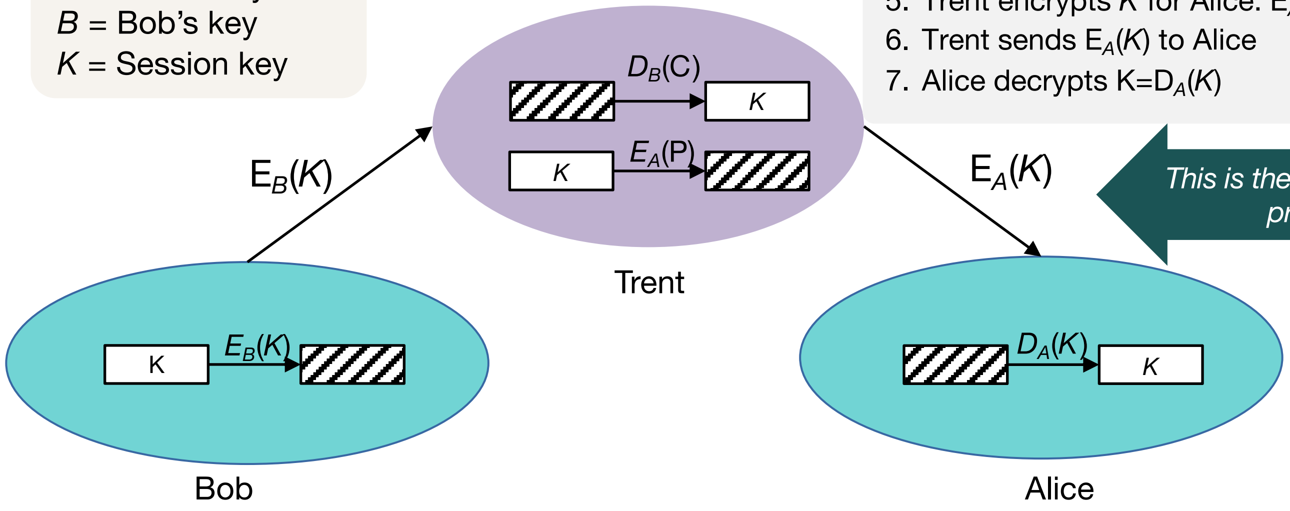
- The more data is encrypted with the same key, the easier it is for cryptanalysts to try to mount attacks
- **Perfect forward secrecy** = encrypt data with ephemeral keys
  - If an attacker gets hold of a key, it will not enable them to decrypt other sessions
- We may have key exchange protocols that need to create a key for two parties to communicate

Key exchange with a trusted third party

- Trusted third party, Trent, knows all the keys
- Everyone else only knows their own keys

1. Bob creates a random session key, *K*
2. Bob encrypts it with his secret key: $E_B(K)$
3. Bob sends $E_B(K)$ to Trent
4. Trent decrypts using Bob's key
5. Trent encrypts *K* for Alice: $E_A(K)$
6. Trent sends $E_A(K)$ to Alice
7. Alice decrypts $K=D_A(K)$

A = Alice's key
B = Bob's key
K = Session key



This is the key exchange process

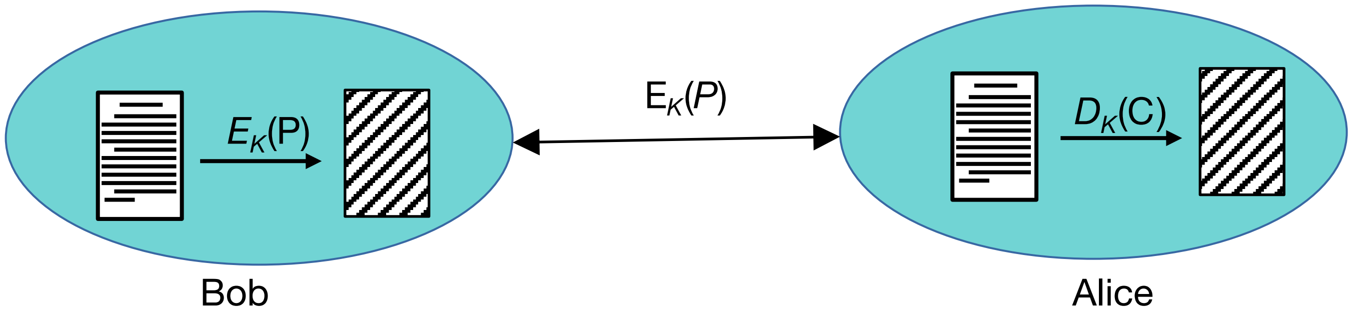
We'd need to enhance this to avoid *replay attacks: time stamps, sequence numbers*

Key exchange with a trusted third party

… continued

A =	Alice's key
B =	Bob's key
K =	Session key

1. Bob creates a random session key, K
2. Bob encrypts it with his secret key: $E_B(K)$
3. Bob sends $E_B(K)$ to Trent
4. Trent decrypts using Bob's key
5. Trent encrypts K for Alice: $E_A(K)$
6. Trent sends $E_A(K)$ to Alice
7. Alice decrypts $K=D_A(K)$
8. **Alice & Bob communicate, encrypting messages with the session key, *K***



Diffie-Hellman Key Exchange

Key distribution algorithm

- Allows two parties to share a secret key over a non-secure channel
- *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 **common key** without fear of eavesdroppers:

common key = *f(your_private_key, their_public_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 sends Bob public key Y_A
- Alice computes

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

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

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

Diffie-Hellman exponential key exchange

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

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

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

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

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

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice sends Bob 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 sends Alice 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

Hybrid Cryptosystems

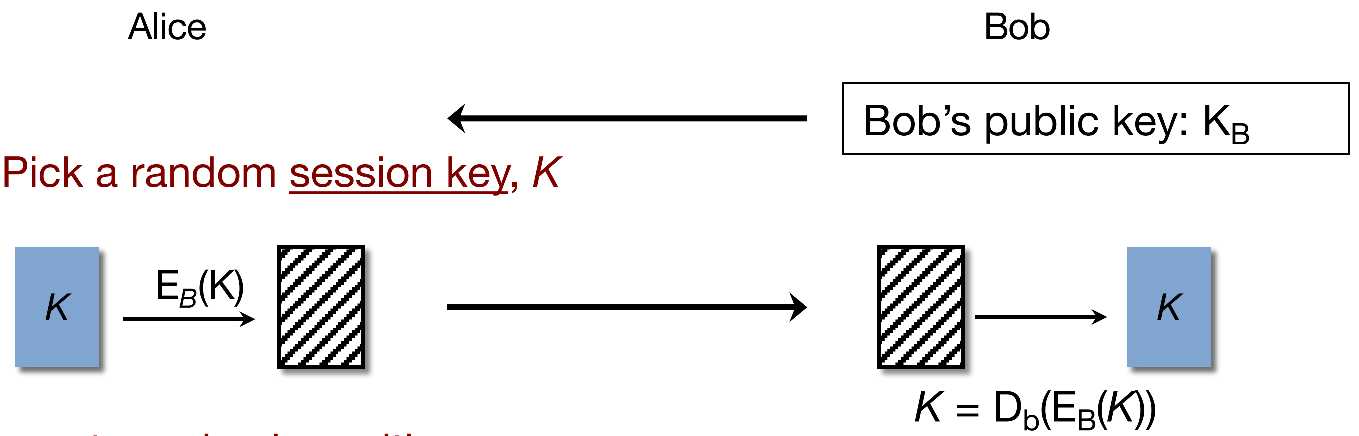
Hybrid Cryptosystems

- 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

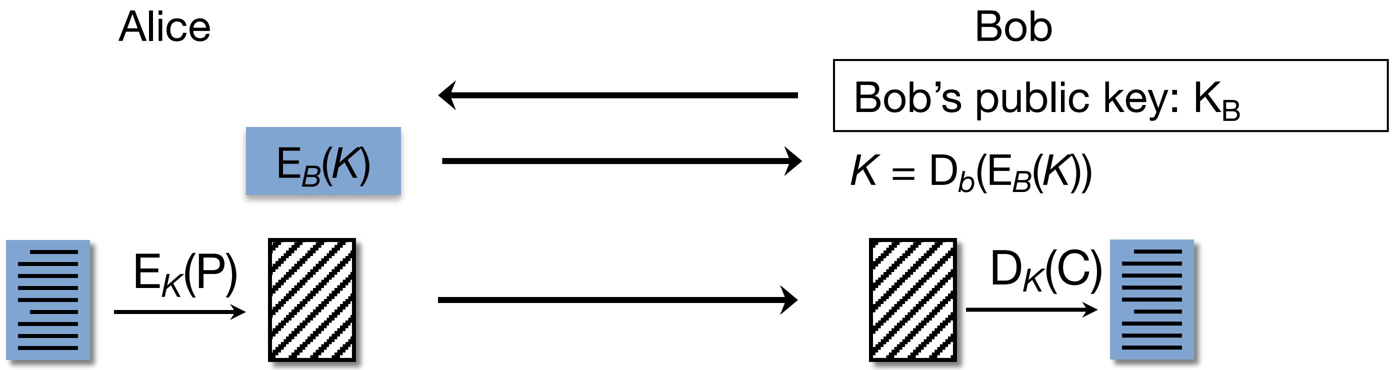


encrypt session key with Bob's public key

Bob decrypts K with his private key

Now Bob knows the secret session key, K

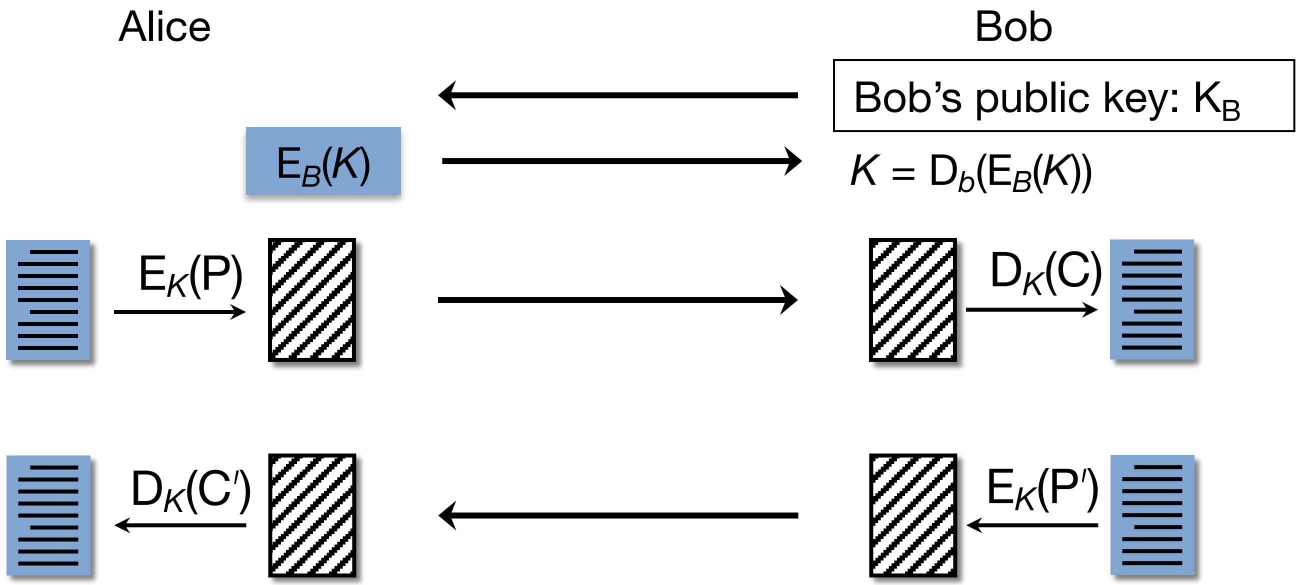
Communication with a hybrid cryptosystem



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*

Cryptographic systems: summary

- Symmetric ciphers
  - Shared secret key
- Asymmetric ciphers – public key cryptosystems
  - Non-shared private key & publicly-shared public key
- Hybrid cryptosystem
  - Use a **public key algorithm** (including Diffie-Hellman) to send a randomly-chosen **session key**
    - Session key is **ephemeral**
  - Use a **symmetric key** algorithm with the session key to encrypt traffic back & forth
  - Diffie-Hellman usually used because it's quick to generate keys
- Key exchange algorithms
  - Trusted Third Party
  - Diffie Hellman
  - Public key

$\left.\begin{array}{l}\text{Diffie Hellman}\\\text{Public key}\end{array}\right\}$ *Enables secure communication without using a 3rd party or knowledge of a shared secret*

The End