WILFRID LAURIER UNIVERSITY
LAURIER
Inspiring Lives.

CP 460 - Applied Cryptography

Key Establishment

Department of Physics and Computer Science
Faculty of Science, Waterloo

Abbas Yazdinejad, Ph.D.

Fall 2024

Content of this Chapter

- **Introduction**
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - Public-Key Infrastructure

Key Establishment

- **Key Establishment** refers to the process of securely setting up cryptographic keys that are shared between two or more parties to enable secure communication. These keys are essential for encryption, decryption, and ensuring data integrity and authenticity.

Types of Key Establishment

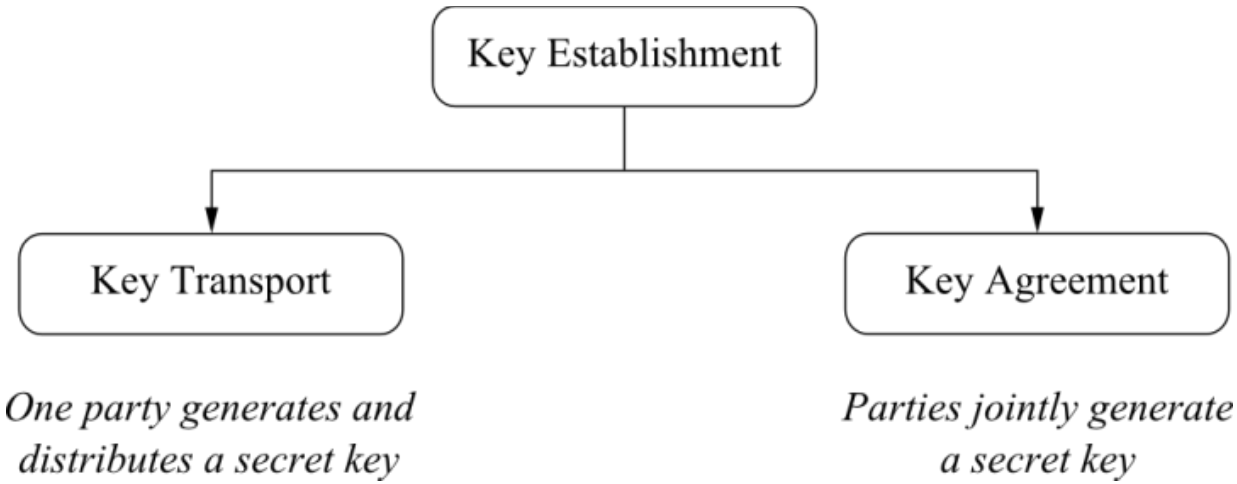
1.Key Transport:

1. One party generates the key and securely transmits it to the other party.
2. Example: A server generates an encryption key and sends it to a client using a secure channel.

2.Key Agreement:

1. Both parties collaborate to generate a shared key without exchanging the actual key.
2. Example: The Diffie-Hellman Key Exchange protocol.

Classification of Key Establishment Methods



In an ideal key agreement protocol, no single party can control what the key value will be.

Why Is Key Establishment Important?

It ensures that the keys used in cryptographic operations (like encryption and MACs) are:

- **Securely shared** between participants.
- Resistant to common attacks like eavesdropping, man-in-the-middle (MITM), or replay attacks.
- Frequently refreshed to maintain security (key freshness).

Key Freshness

Refers to the practice of regularly generating and using new cryptographic keys to minimize the risks associated with long-term key usage. It ensures that the cryptographic security of a system remains robust even if an old key is compromised.

It is often desirable to frequently change the key in a cryptographic system.

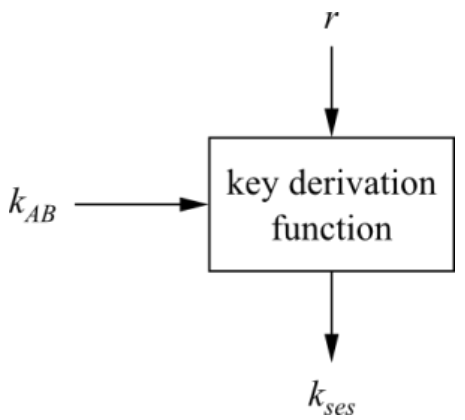
Reasons for key freshness include:

- If a key is exposed (e.g., through hackers), there is limited damage if the key is changed often
- Some cryptographic attacks become more difficult if only a limited amount of ciphertext was generated under one key
- If an attacker wants to recover long pieces of ciphertext, he has to recover several keys which makes attacks harder

Key Derivation

is the process of generating new cryptographic keys, often called **session keys**, from an existing key (called a **master key** or **long-term key**) and some additional input, such as a random value (nonce). This ensures **key freshness** and minimizes the need for frequent full key exchanges, which can be computationally expensive.

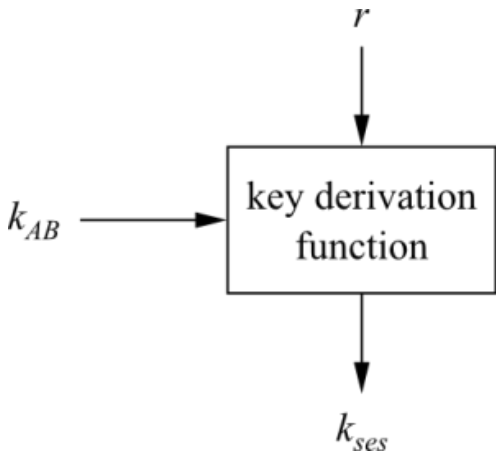
- In order to achieve key freshness, we need to generate new keys frequently.
- Rather than performing a full key establishment every time (which is costly in terms of computation and/or communication), we can **derive multiple session keys k_{ses} from a given key k_{AB}**.



- The key k_{AB} is fed into a key derivation function together with a nonce r („number used only once“).
- Every different value for r yields a different session key

Key Derivation

- The key derivation function is a computationally simple function, e.g., a block cipher or a hash function



- Example for a basic protocol:

Alice		Bob
		generate nonce *r*
	r ←	
derive session key $K_{ses} = e_{kAB}(r)$		derive session key $K_{ses} = e_{kAB}(r)$

- **Why Use Key Derivation?**

1. **Efficient Key Management**: Avoids repeated full key exchanges by deriving multiple session keys from a single shared key.
2. **Enhances Security**:
   - Limits the impact of key compromise.
   - Ensures each session or transaction uses a unique, derived key.
3. **Scalability**: Supports large-scale systems where frequent full key generation is impractical.

Advantages of Key Derivation

1. **Efficiency**: Reduces computational overhead.
2. **Key Freshness**: Ensures each session or transaction uses a unique key.
3. **Security**: Limits the impact of key compromise and protects against replay attacks.

Content of this Chapter

- Introduction
- **The n^2 Key Distribution Problem**
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - Public-Key Infrastructure

The n^2 Key Distribution Problem

- The $n2$ Key Distribution Problem highlights the challenge of managing keys in a network of users who need to securely communicate with one another using symmetric key cryptography.

Challenges

1.Scalability:
1. As n increases, the number of required keys grows quadratically (O(n^2).

2.Key Distribution Overhead:
1. Each key must be securely distributed to both users in the pair.
2. If a new user joins the network, a secure key must be shared with every existing user, increasing the overhead.

3.Key Management Complexity:
1. Storing and maintaining such a large number of keys securely is impractical.
2. Key compromise becomes a major risk because of the high number of keys.

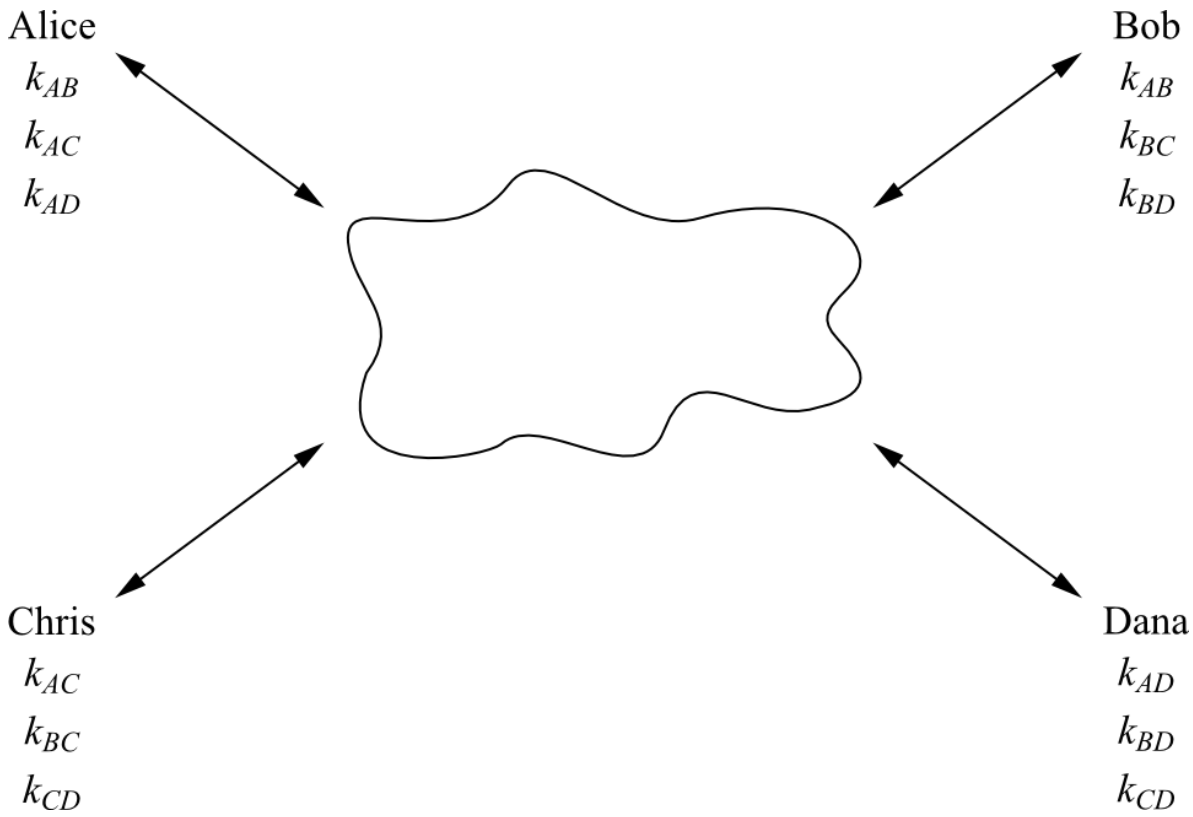
The n^2 Key Distribution Problem

- **Solutions**

1) **Key Distribution Centers (KDC)**:
   - A trusted central authority manages all keys.
   - Each user has only one long-term key shared with the KDC.
   - The KDC generates and distributes session keys for each pair of users, significantly reducing the number of required keys.
2) **Public-Key Cryptography**:
   - Instead of symmetric keys, users generate public-private key pairs.
   - Public keys can be shared openly, eliminating the need for n2n^2n2 symmetric keys.
3) **Key Agreement Protocols**:
   - Protocols like **Diffie-Hellman** allow users to establish shared keys dynamically without pre-distribution.

The n^2 Key Distribution Problem

- Simple situation: Network with n users. Every user wants to communicate securely with every of the other n-1 users.
- Naïve approach: Every pair of users obtains an individual key pair

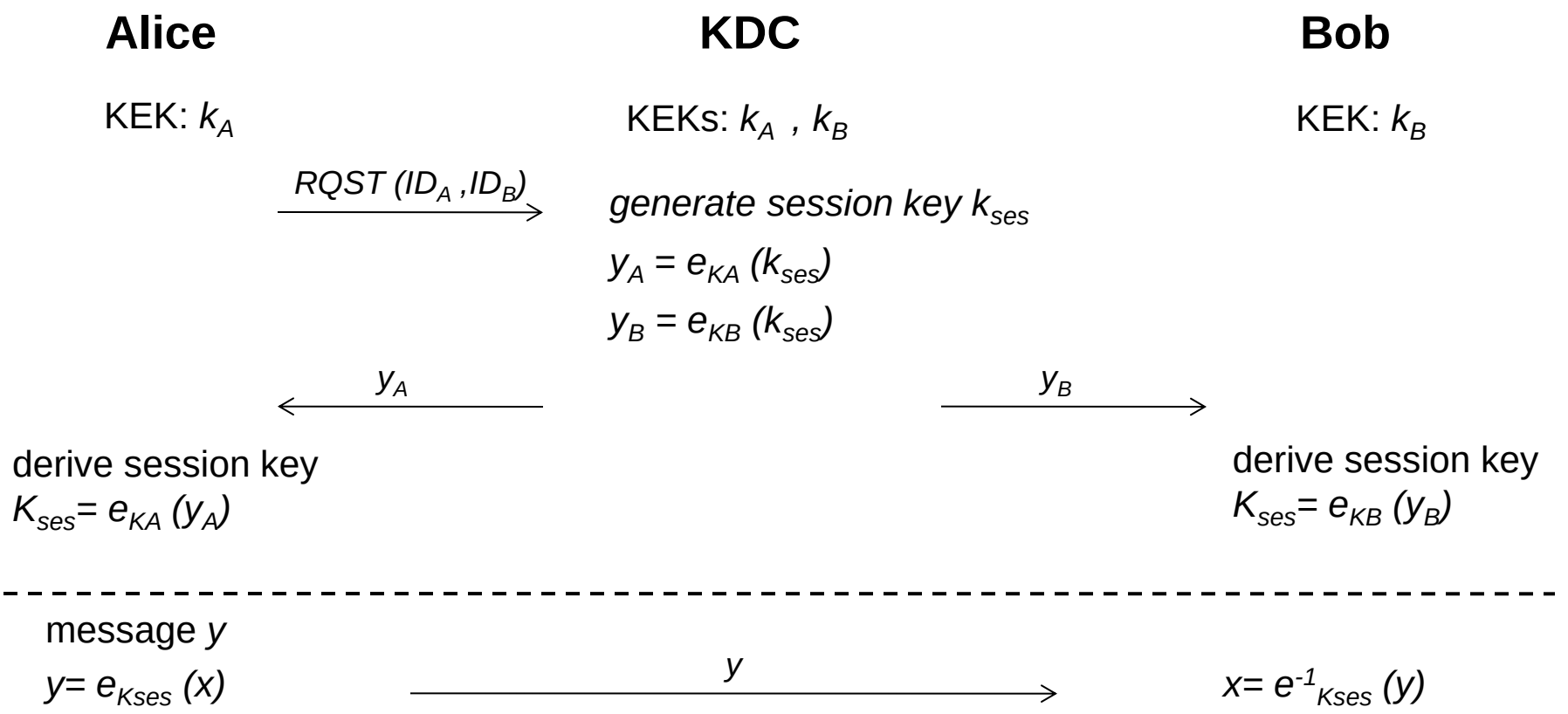


Content of this Chapter

- Introduction
- The n^2 Key Distribution Problem
- **Symmetric Key Distribution**
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - Public-Key Infrastructure

Key Establishment with Key Distribution Center

- Key Distribution Center (KDC) = Central party, trusted by all users
- KDC shares a **key encryption key (KEK)** with each user
- Principle: KDC sends session keys to users which are encrypted with KEKs



Key Establishment with Key Distribution Center

The **Key Distribution Center (KDC)** is a trusted central authority used to facilitate secure key establishment in symmetric-key systems. It helps overcome the scalability challenges of the n^2Key Distribution Problem by reducing the number of long-term keys required in the system.

- Advantages over previous approach:
  - Only *n* long-term key pairs are in the system
  - If a new user is added, a secure key is only needed between the user and the KDC (the other users are not affected)
  - Scales well to moderately sized networks

- *Kerberos* (a popular authentication and key distribution protocol) is based on KDCs
- More information on KDCs and Kerberos: Section 13.2 of *Understanding Cryptography*

Key Establishment with Key Distribution Center

Advantages of Using a KDC

1.Scalability:

1. Only n long-term keys are needed for nnn users, as opposed to n(n−1)/2 symmetric keys for all user pairs.
2. Adding a new user requires only one additional long-term key shared with the KDC.

2.Session Key Freshness:

1. The KDC generates new session keys for each communication session, ensuring key freshness.

3.Centralized Key Management:

1. The KDC manages all keys, simplifying key generation and distribution.

Key Establishment with Key Distribution Center

Remaining problems:

- **No *Perfect Forward Secrecy***: If the KEKs are compromised, an attacker can decrypt past messages if he stored the corresponding ciphertext
- **Single point of failure**: The KDC stores all KEKs. If an attacker gets access to this database, all past traffic can be decrypted.
- **Communication bottleneck**: The KDC is involved in every communication in the entire network (can be countered by giving the session keys a long life time)
- For more advanced attacks (e.g., key confirmation attack): Cf. Section 13.2 of *Understanding Cryptography*

Content of this Chapter

- Introduction
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- **Asymmetric Key Distribution**
  - **Man-in-the-Middle Attack**
  - Certificates
  - Public-Key Infrastructure

Man-in-the-middle attack

- A **Man-in-the-Middle (MITM) Attack** occurs when an attacker secretly intercepts and manipulates communication between two parties (e.g., Alice and Bob) without their knowledge. The attacker (often called **Oscar**) impersonates each party to the other, allowing them to intercept, modify, or steal information.

How It Works

1.Setup:

1. Alice and Bob want to establish a secure communication channel.
2. Both parties exchange cryptographic information (e.g., public keys) to set up encryption.

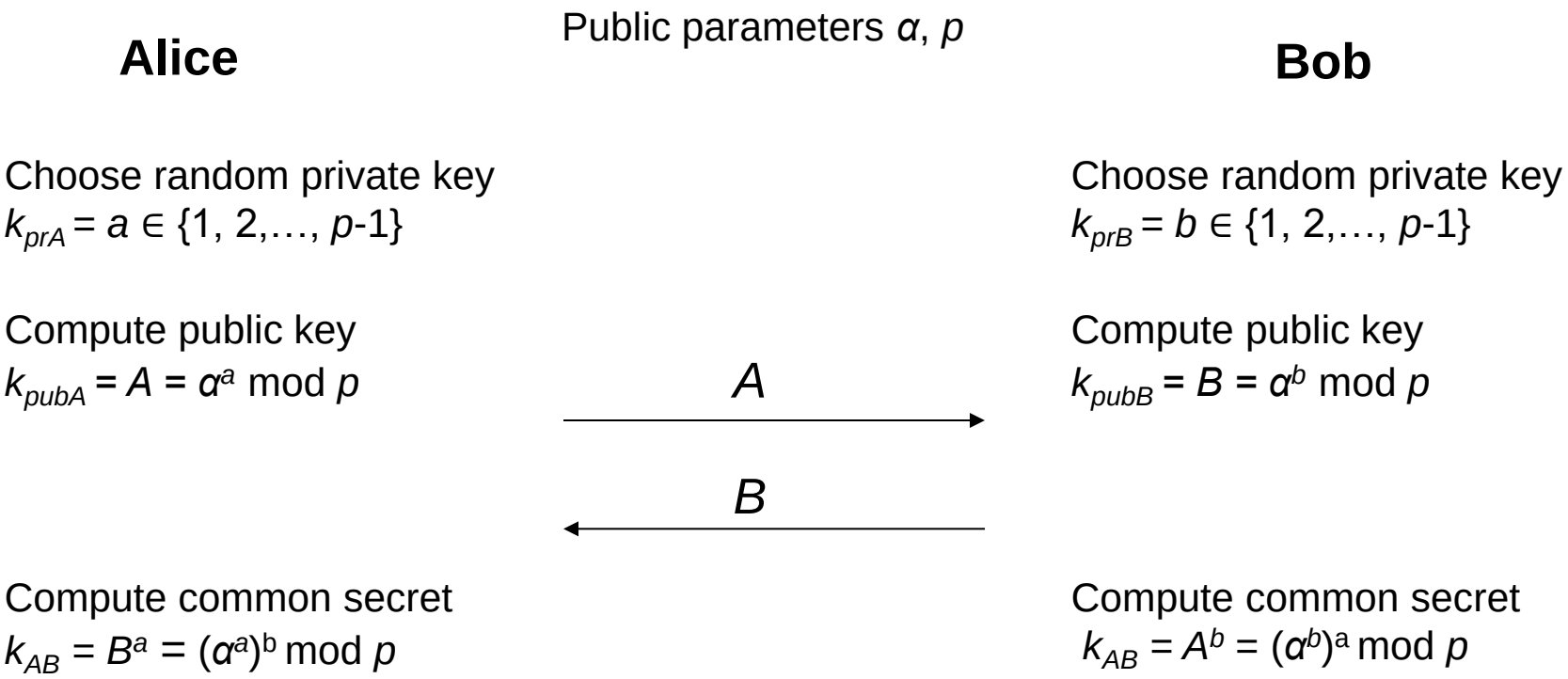
2.Interception:

1. Oscar intercepts the communication and substitutes his public key for both parties.
2. Alice thinks she is communicating with Bob, but she is actually talking to Oscar.
3. Similarly, Bob thinks he is communicating with Alice, but he is also talking to Oscar.

3.Communication:

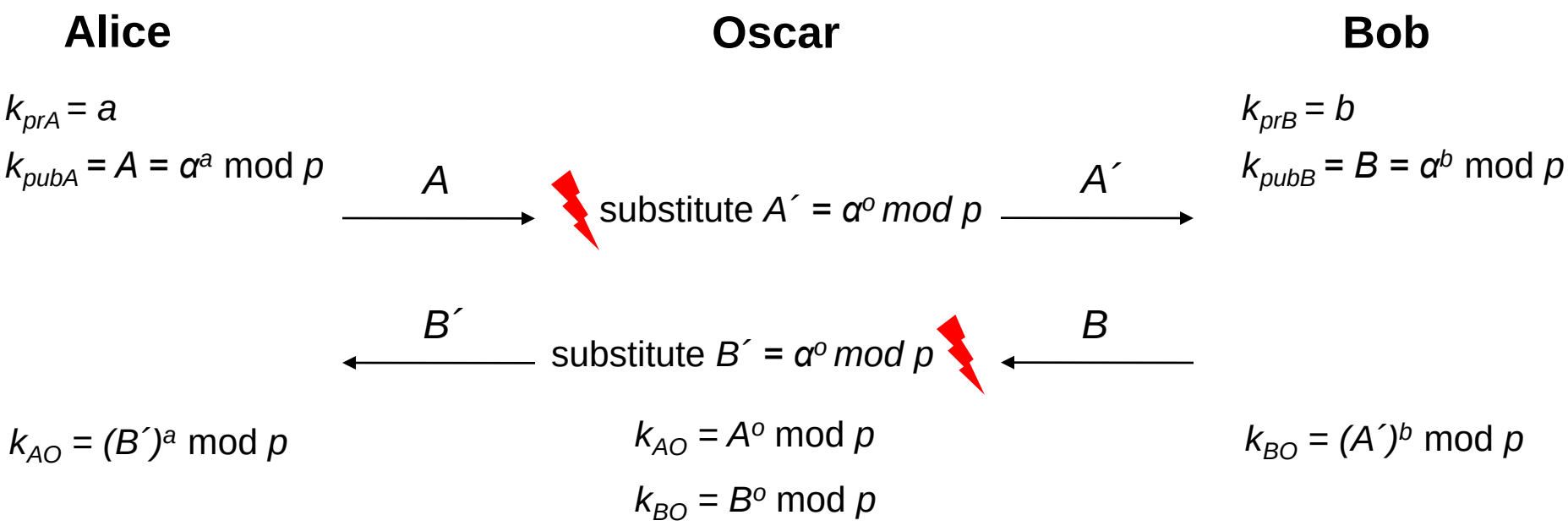
1. Oscar establishes separate encrypted channels with Alice and Bob.
2. All messages pass through Oscar, allowing him to read, modify, or inject new messages before forwarding them to the intended recipient.

Recall: Diffie–Hellman Key Exchange (DHKE)



- Widely used in practice
- If the parameters are chosen carefully (especially a prime $p > 2^{1024}$), the DHKE is secure against *passive* (i.e., listen-only) attacks
- However: If the attacker can *actively* intervene in the communciation, the man-in-the-middle attack becomes possible

Man-in-the-Middle Attack



- Oscar computes a session key k_{AO} with Alice, and k_{BO} with Bob
- However, Alice and Bob think they are communicationg with each other !
- The attack efficiently performs 2 DH key-exchanges: Oscar-Alice and Oscar-Bob
- Here is why the attack works:

Alice computes: $k_{AO} = (B')^a = (\alpha^o)^a$

Oscar computes: $k_{AO} = A^o = (\alpha^a)^o$

Bob computes: $k_{BO} = (A')^b = (\alpha^o)^b$

Oscar computes: $k_{BO} = B^o = (\alpha^a)^o$

Very, very important facts about the Man-in-the-Middle Attack

- **The man-in-the-middle-attack is not restricted to DHKE; it is applicable to any public-key scheme, e.g. RSA encryption. ECDSA digital signature, etc. etc.**
- The attack works always by the same pattern: Oscar replaces the public key from one of the parties by his own key.
- The attack is also known as MIM attack or Janus attack

- Q: What is the underlying problem that makes the MIM attack possible?
- A: The public keys are not authenticated: When Alice receives a public key which is allegedly from Bob, she has no way of knowing whether it is in fact his. (After all, a key consists of innocent bits; it does not smell like Bob‘s perfume or anything like that)

Even though public keys can be sent over unsecure channels, they require authenticated channels.

Content of this Chapter

- Introduction
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - **Certificates**
  - Public-Key Infrastructure

Certificates

A **certificate** is a digital document issued by a trusted authority that binds a user or entity's **public key** to their **identity**. Certificates are used to prevent **Man-in-the-Middle (MITM) attacks** by ensuring that public keys are authentic and belong to the claimed parties.

- In order to authenticate public keys (and thus, prevent the MIM attack) , all public keys are digitally signed by a central trusted authority.
- Such a construction is called *certificate*

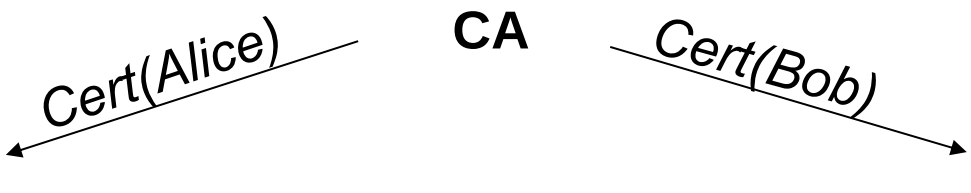
certificate = public key + ID(user) + digital signature over public key and ID

- In its most basic form, a certificate for the key k_{pub} of user Alice is:

$$\textbf{Cert(Alice)} = (k_{pub}, \textit{ID(Alice)}, sig_{K_{CA}}(k_{pub}, \text{ID(Alice)}))$$

- Certificates bind the identity of user to her public key
- The trusted authority that issues the certificate is referred to as ***certifying authority (CA)***
- „Issuing certificates“ means in particular that the CA computes the signature $sig_{K_{CA}}(k_{pub})$ using its (super secret!) private key k_{CA}
- The party who receives a certificate, e.g., Bob, verifies Alice‘s public key using the public key of the CA

Diffie–Hellman Key Exchange (DHKE) with Certificates



$k_{prA} = a$

$k_{pubA} = A$

$Cert(Alice) = ((A, ID_A), sig_{K_{CA}}(A, ID_A))$

$k_{prB} = b$

$k_{pubB} = B = \alpha^b \bmod p$

$Cert(Bob) = ((B, ID_B), sig_{K_{CA}}(B, ID_B))$

$Cert(Alice)$ →

← $Cert(Bob)$

verify certificate
$ver_{Kpub,CA}(Cert(Bob))$

if verification is correct:
Compute common secret
$k_{AB} = B^a = (\alpha^a)^b \bmod p$

verify certificate
$ver_{Kpub,CA}(Cert(Alice))$

if verification is correct:
Compute common secret
$k_{AB} = A^b = (\alpha^b)^a \bmod p$

Content of this Chapter

- Introduction
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - **Public-Key Infrastructure**

Certificates in the Real World

- **Structure of Real-World Certificates**
- Real-world certificates are based on widely used standards like **X.509**, which define the format and fields of certificates. A typical certificate includes:

1. **Serial Number**: A unique identifier for the certificate.
2. **Issuer**: The Certificate Authority (CA) that issued the certificate.
3. **Validity Period**: Defines the "Not Before" and "Not After" dates, specifying when the certificate is valid.
4. **Subject**: The entity to which the certificate is issued (e.g., a website or individual).
5. **Subject's Public Key**:
    1. Includes the algorithm (e.g., RSA, ECC) and the public key value.
6. **Signature Algorithm**: Specifies the algorithm used by the CA to sign the certificate (e.g., RSA-SHA256).
7. **Digital Signature**: The CA's signature on the certificate, binding the public key to the subject.

Applications of Certificates in the Real World

1. **Website Security (TLS/SSL Certificates)**:
    1. Certificates are used to establish secure HTTPS connections. Protects against **Man-in-the-Middle (MITM)** attacks and encrypts data between the browser and the server.
2. **Email Encryption and Signing**:
    1. Encrypt emails so only the intended recipient can read them.
    2. Digitally sign emails to verify the sender's identity.
3. **Code Signing**:
    - Certificates ensure that software or applications are authentic and have not been tampered with.
    - Example: Microsoft or Apple uses certificates to verify software developers.
4. **Device Authentication**:
    - IoT devices and hardware components use certificates to securely identify themselves and establish trust with other systems.
5. **Secure Communication**:
    - Virtual Private Networks (VPNs): Certificates are used to authenticate endpoints and establish secure communication.
6. **Government and Enterprise Use**:
    - Digital IDs: Governments issue certificates to citizens for secure online services.
    - Enterprises use certificates for employee authentication and secure internal communications.


Thank
You