Cybersecurity Fundamentals

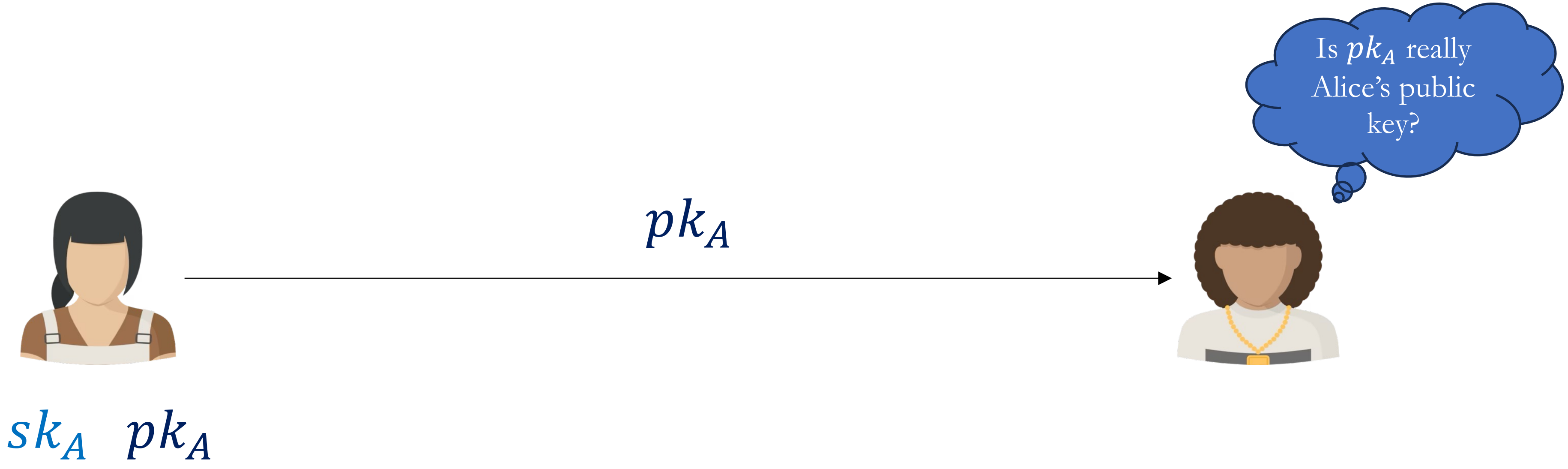
Lecture 8
Public-key Infrastructure and TLS protocol

Thomas Zacharias

University of Glasgow

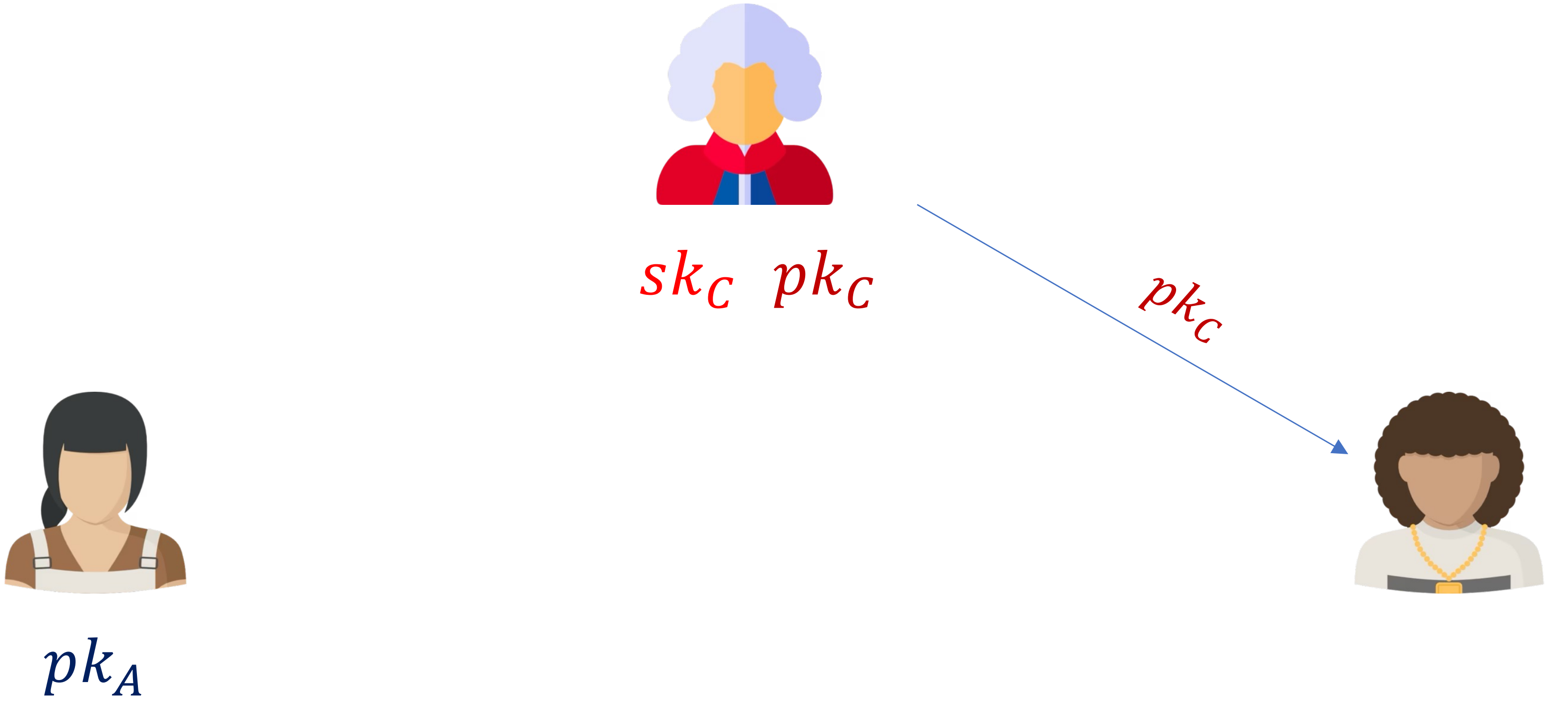
The problem of secure public key distribution (over unauthenticated channels)

- Alice generates a pair of a private key sk_A and a public key pk_A.
- Alice sends pk_A to Bob through a public unauthenticated channel.
- Anyone can forge such a public announcement!
- Bob cannot be sure that pk_{A} indeed belongs to Alice.



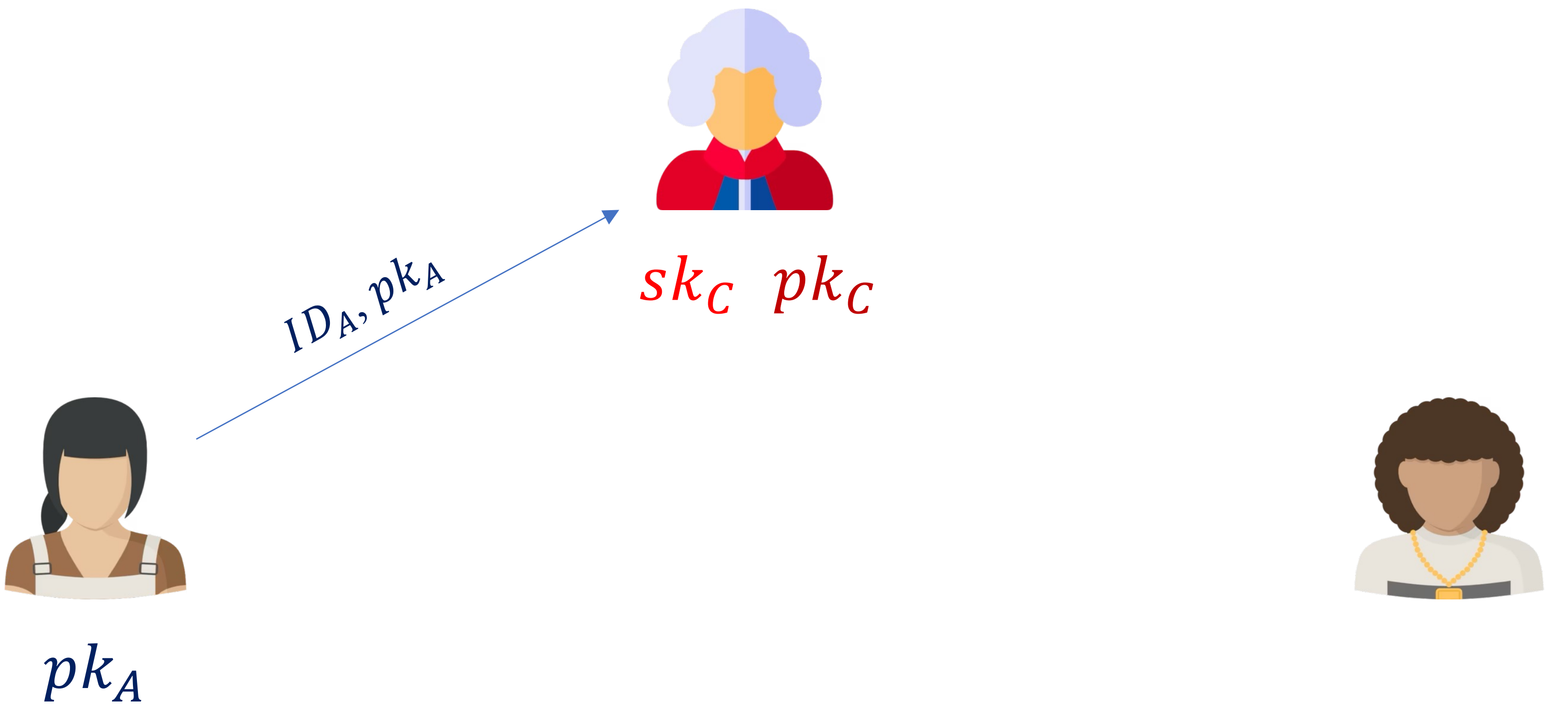
Public-key certificates: the core idea

- Let Claudia be a third party trusted by both Alice and Bob.
- Claudia has generated a pair of a private key sk_C and a public key pk_C.
- Bob can **securely obtain pk_C**.



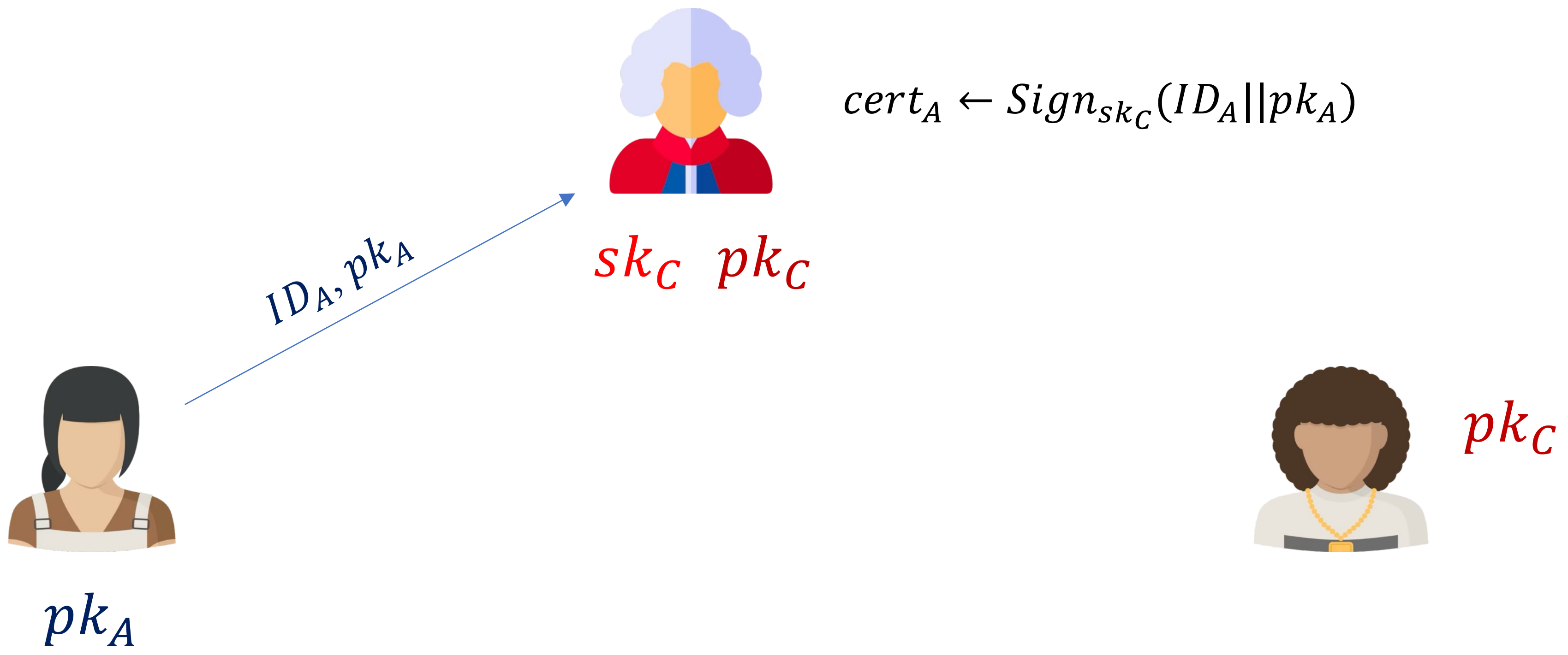
Public-key certificates: the core idea

- Alice provides Claudia with her ID and public key in **some secure manner**.



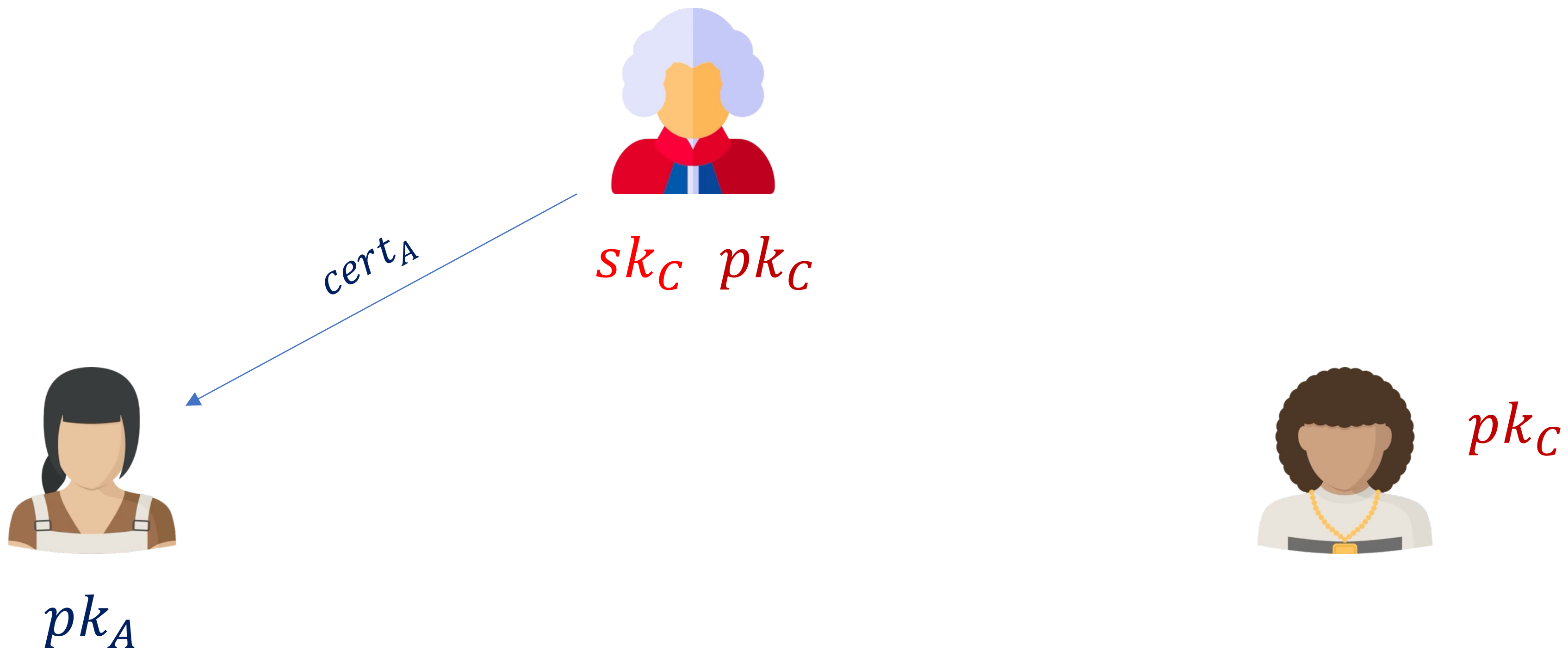
Public-key certificates: the core idea

- Claudia creates **a public-key certificate** for Alice by producing a **signature on Alice's ID and public key**.



Public-key certificates: the core idea

- Claudia provides Alice with the public-key certificate.



Public-key certificates: the core idea

- Alice sends her ID, public key, and certificate to Bob through the public unauthenticated channel.

Public-key certificates: the core idea

- Using pk_C, **Bob verifies** that $cert_A$ is a valid signature on ID_A, pk_A.
- If verification is successful, **then Bob accepts** pk_A as Alice's legitimate public key.

Missing parts (to be resolved)

1. How does Bob securely learn pk_C in the first place?
2. How can Claudia be sure that pk_A is Alice's public key?

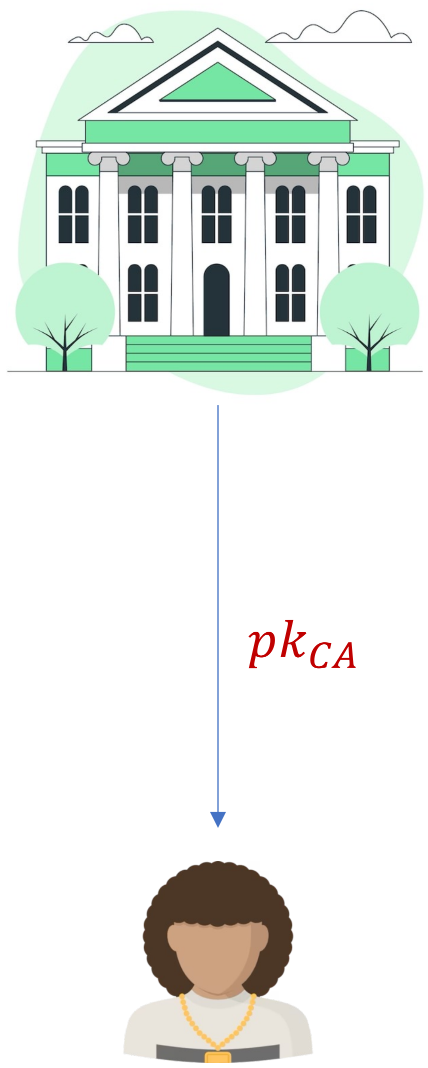
Fully specifying such details (and others) defines a Public-key Infrastructure (PKI)

A simple PKI

- A single **certificate authority (CA)** is trusted by everybody and issues certificates for everyone's public key.
- Typically, a CA would be
  - a company that certifies public keys,
  - a government agency,
  - or a department within an organisation (to be used by people within the organisation).

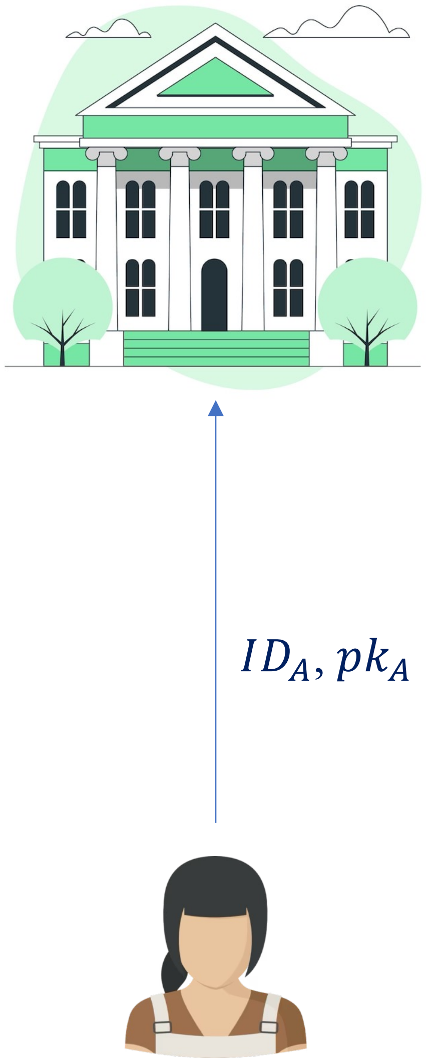
A simple PKI

- Bob securely learns CA's public key pk_{CA}:
  - by providing pk_{CA} together with some software (e.g., a web browser).
  - via physical means (e.g., if CA is within an organisation, then any employee can obtain an authentic copy of pk_{CA} directly from the CA).



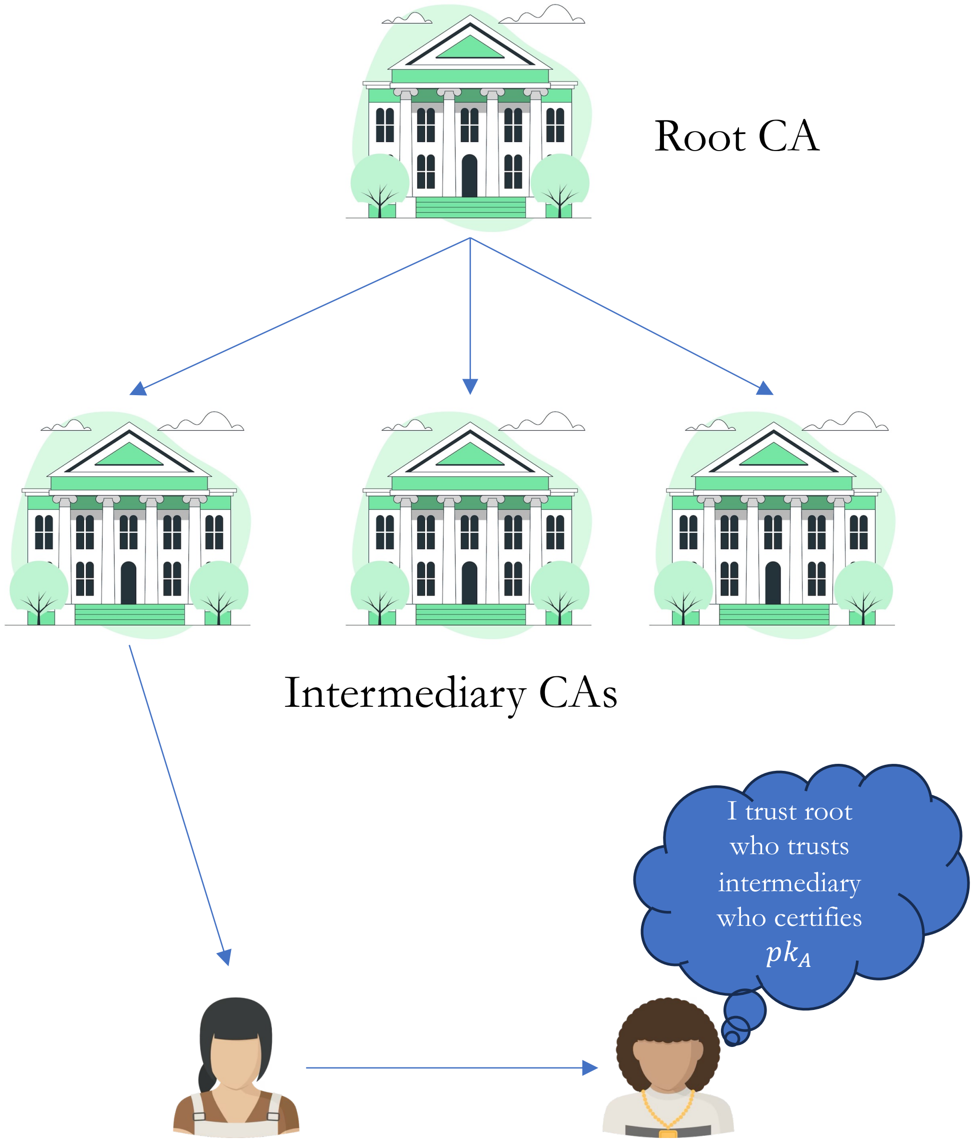
A simple PKI

- CA is sure that pk_A is Alice's public key:
  - by receiving a **certificate signing request** (CSR) from Alice that contains ID_A, pk_A and necessary identifying information (e.g., email address, company domain name).
  - by requesting that Alice shows up in person with a copy of pk_A and some identifying information.



Multiple CA hierarchy

- In reality, PKI implementations come with a large list of CAs and their public keys. These CAs
  - directly sign end-user certificates or
  - sign a small number of Intermediate CAs that in turn sign end-user certificates.
- Generally, the CAs are structured into single-tier, two-tier, and three-tier hierarchies.
- The CA hierarchy approach provides
  - an increased level of security for the root CA that can be kept offline (protecting the private key of the root CA from a compromise).
  - granularity, by allowing administrators to deploy CAs in different locations and with customised security levels for each CA.



Example: a two-tier CA hierarchy

Invalidating certificates

- Public-key **certificates should not be valid indefinitely**. E.g.,
  - A user's private key may be stolen, so the user must remove the corresponding public key from circulation.
  - An employee may leave a company and should no longer receive encrypted communication.
- According to **X.509 standard** for the format of public-key certificates, the CA includes in the certificate (signature):
  - A **validity period** of the certificate.
  - A **serial number** that is unique for this CA. It can be used to verify if the certificate has been revoked (by checking a certificate revocation list maintained by the CA).

The TLS protocol

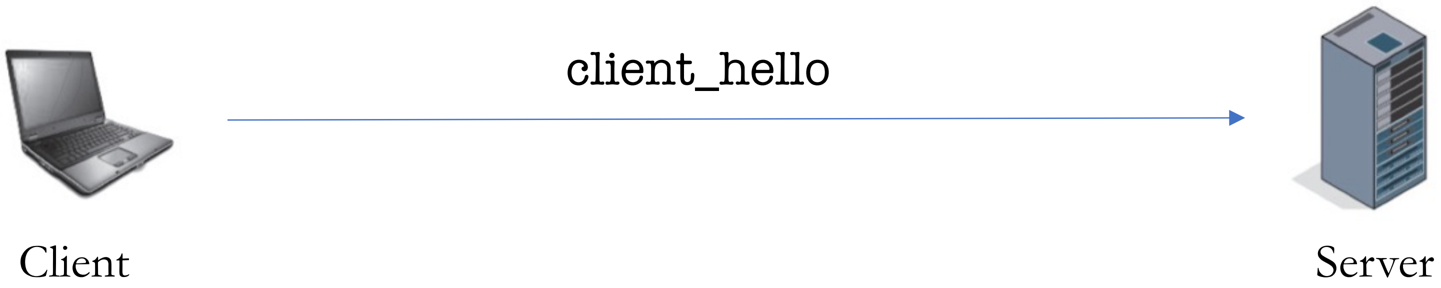
- **Transport Layer Security (TLS)** is an extensively used protocol that enables secure communication over the web.
  - Used by the browsers any time we connect to a website using `https`.
  - Most recent version: TLS 1.3 (2018).
- **Security guarantees** when a client and a server use TLS:
  - Data **confidentiality** (using encryption)
  - Data **integrity** (using MACs)
  - Server **authentication** and (optionally) client authentication (using public-key cryptography)

The TLS protocol

- TLS comprises the following protocols:
  - **Handshake protocol**: allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys.
  - **Change Cipher Spec protocol**: client and server send a message notifying the receiving party that subsequent messages will be protected by the negotiated cryptographic parameters and keys.
  - **Record protocol**: using the negotiated cryptographic parameters and keys, it protects the confidentiality and the integrity of subsequent exchanged messages.
  - **Alert protocol**: used to convey TLS-related alerts to the peer entity.
    - An alert level can be fatal (session closes immediately) or warning (indicating a problem).
    - Examples of alert descriptions: "decryption failed", "handshake failure", "certificate revoked", etc.
  - **Heartbeat protocol**: a request and response protocol that
    - assures the sender of the request that the recipient is still alive, if there has been no activity for a while.
    - generates activity during idle periods avoiding closure by a firewall that does not tolerate idle connections.

Overview of the TLS Handshake protocol

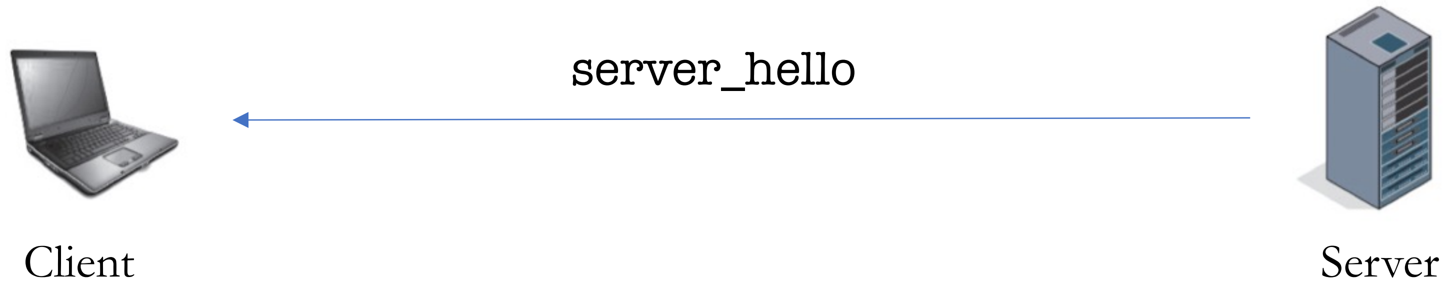
1. The client begins by sending a `client_hello` message that specifies:
   - The highest TLS version understood by the client.
   - A random nonce n_C.
   - A session identifier.
   - The ciphersuite (i.e., the combinations of cryptographic algorithms) supported by the client.
   - The data compression methods supported by the client.



Overview of the TLS Handshake protocol

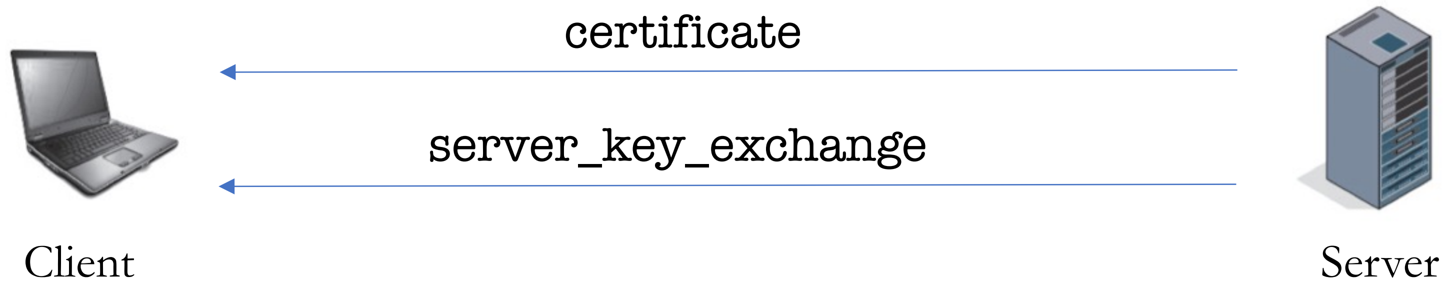
2. The server selects:
   - The highest TLS version that both the client and server support.
   - An appropriate ciphersuite and compression method from the choices offered by the client.

   Then, it responds by sending a `server_hello` message that contains:
   - The selected version, ciphersuite, and compression method.
   - A random nonce n_S.



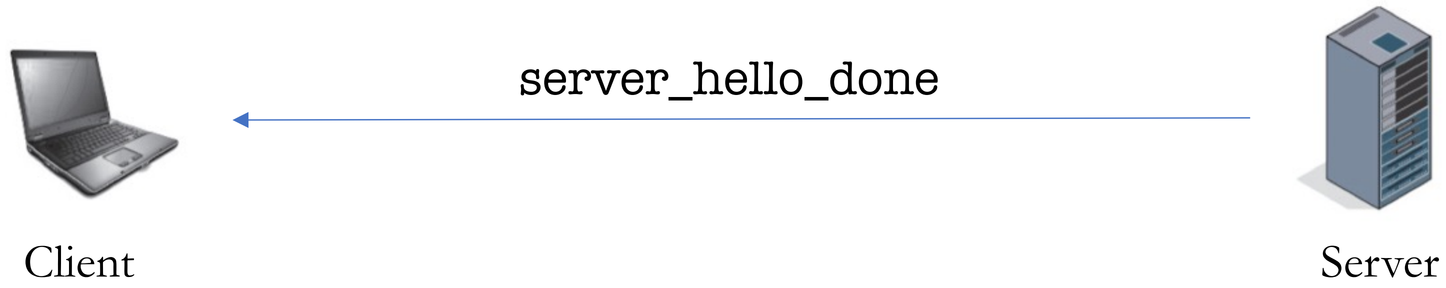
Overview of the TLS Handshake protocol

3. The server sends:
    - Its public key pk_S and a corresponding public-key certificate (normally, issued by a CA that the client knows).
    - If the above information is not sufficient for key exchange, a `server_key_exchange` message (e.g., a public Diffie-Hellman value g^{x_S})



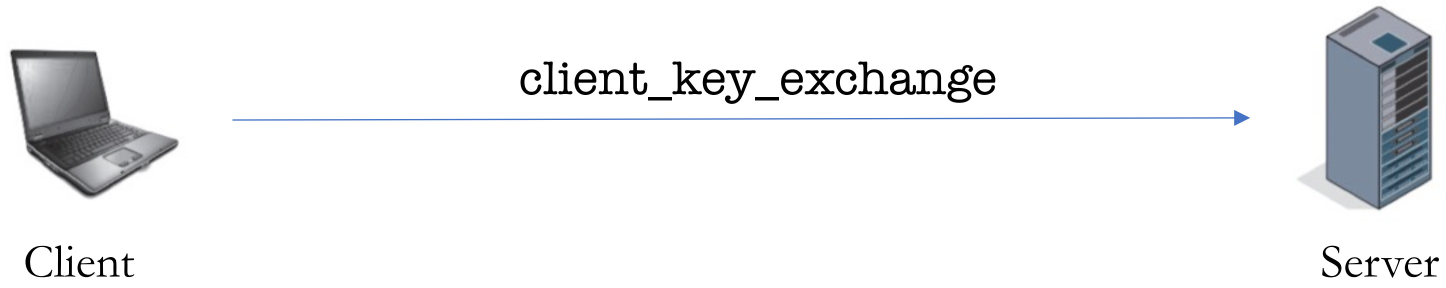
Overview of the TLS Handshake protocol

4. The server completes the negotiation part by sending a `server_hello_done` message and waits for client's response.



Overview of the TLS Handshake protocol

5. The client
   - verifies the validity of the public-key certificate.
   - sends a `client_key_exchange` message that contains information for both parties computing a value called **Premaster Secret**. This message can be:
     - an encryption of a random string under the server's public key pk_S.
     - the client's public Diffie-Hellman value g^{x_C}.



Overview of the TLS Handshake protocol

6. The client and server
   a) compute the same Premaster Secret
      - either by the server decrypting and learning the client's random string, or
      - by both parties completing the Diffie-Hellman key exchange (i.e., computing $(g^{x_C})^{x_S} = (g^{x_S})^{x_C}$).

   b) use the Premaster Secret and the random nonces n_C, n_S to derive the shared cryptographic keys for symmetric encryption and MAC.

$k_{SYM}, k_{MAC} \leftarrow$ 

Client

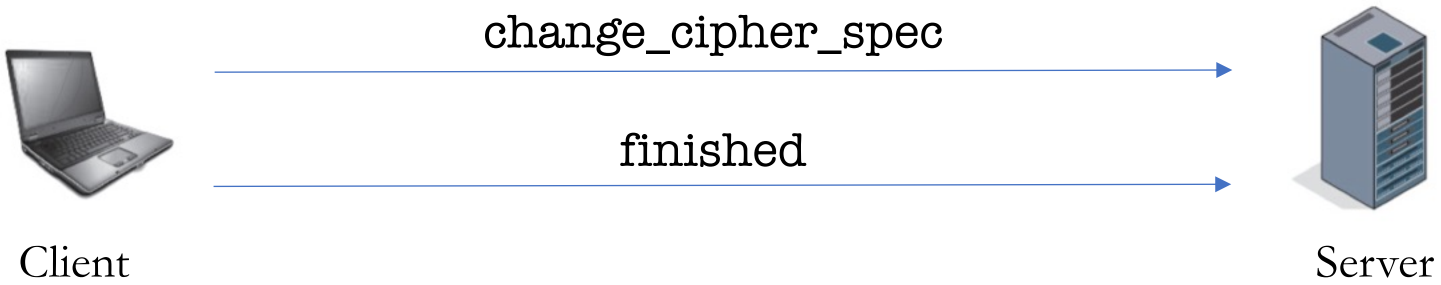
 $\rightarrow k_{SYM}, k_{MAC}$

Server

Overview of the TLS Handshake protocol

7. The client sends
   - A `change_cipher_spec` message notifying the server that it is switching to encrypted mode.
   - A `finished` message that contains a hash of the previously exchanged messages. This message is authenticated and encrypted using the derived keys.

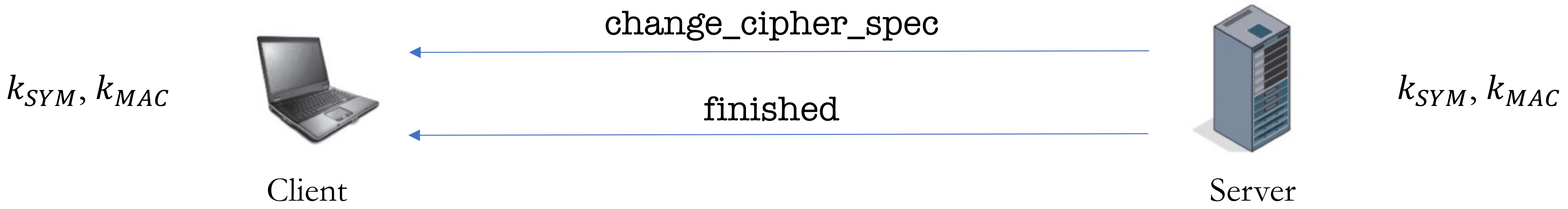
k_{SYM}, k_{MAC}



k_{SYM}, k_{MAC}

Overview of the TLS Handshake protocol

8. The server decrypts the **finished** message and verifies the MAC. If verification is successful, the server sends
   - A **change_cipher_spec** message notifying the client that it is switching to encrypted mode.
   - A **finished** message that contains a hash of the previously exchanged messages. This message is authenticated and encrypted using the derived keys.



Overview of the TLS Handshake protocol

9. The client decrypts the `finished` message and verifies the MAC. If verification is successful, the handshake is completed.

k_{SYM}, k_{MAC}

Client

Server

k_{SYM}, k_{MAC}

Overview of the TLS Handshake protocol

The client and server may begin to securely exchange application layer data using the negotiated cryptographic algorithms under the derived keys.

k_{SYM}, k_{MAC}

k_{SYM}, k_{MAC}

Client

Server

End of Lecture 8

The slides content is related to Sections 2.4, 23.2, 23.3, and 22.3 of "Computer Security Principles and Practice (3rd Edition)" by Stallings and Brown