# Lec 13: Secure Communication

CSED415: Computer Security
Spring 2025

**Seulbae Kim**

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Administrivia

- This Thursday: 10-minute proposal presentations
  - Submission #1: A two-page proposal document, which must include:
    - Definition: What problem are you trying to solve?
    - Motivation: Why is this problem important?
    - Methodology: How do you plan to solve the problem?
    - Evaluation: How will you evaluate your solution's effectiveness?
    - Plan: What is your tentative timeline?
  - Submission #2: Slides for your in-class presentation
    - It is recommended to explicitly include the items above in your slides
  - Submission deadline: April 3 (Thu) by Noon

# Administrivia

- Midterm exam:
  - Time: Next Tuesday (April 8), 2:00-3:15 PM (75 minutes)
  - Location: Classroom (Science Building II, Room #106)
  - Format: Closed book, closed notes, closed laptop/phone exam
    - Allowed: One-page (US letter- or A4-sized) double-sided handwritten cheat sheet
  - Structure: 7 main questions (each may have sub-questions)
  - Scope: Lectures 1 to 13, Labs 01 and 02

# Study tips for midterm exam

- Review lecture slides and labs
  - Retry labs that were left incomplete
- Study in groups (highly recommended)
  - Ask what-if questions to each other
  - Try to answer together
- Focus on understanding concepts instead of memorizing
  - Utilize cheat sheet for referring to facts and formulas
  - Understand WHY something works or doesn't work
  - Think about potential attacks and defenses (practice threat modeling)

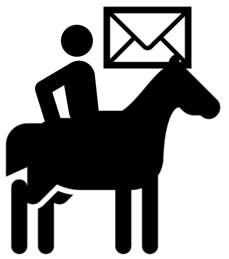
# Recap

- Cryptography:
  - A mechanism for enabling secure communication over insecure, untrusted channels
- Many network-based systems utilize cryptographic schemes for secure communication
  - To guarantee confidentiality, integrity, and authentication
- Today's topic:
  - How various internet services employ cryptographic primitives to ensure a secure connection in practice

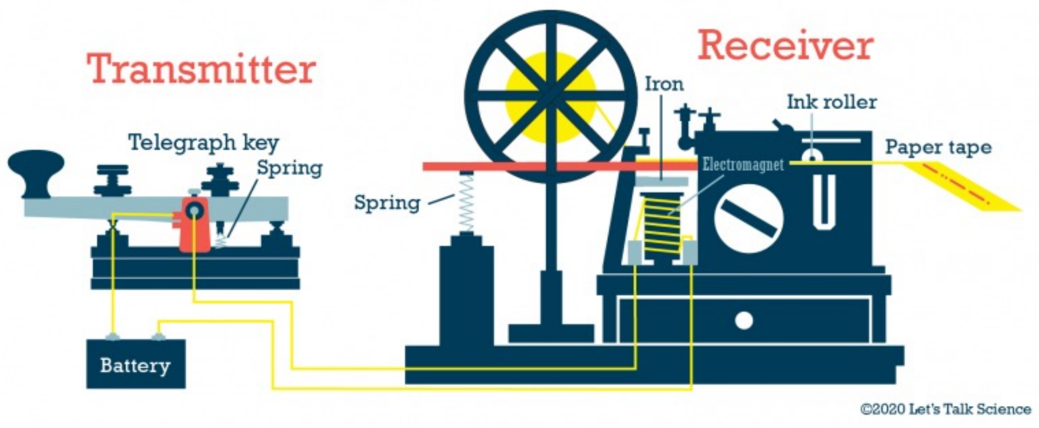
# Secure Emails

# Brief history of email

- Physical transportation
  - Early "remote" messaging was done via physical delivery

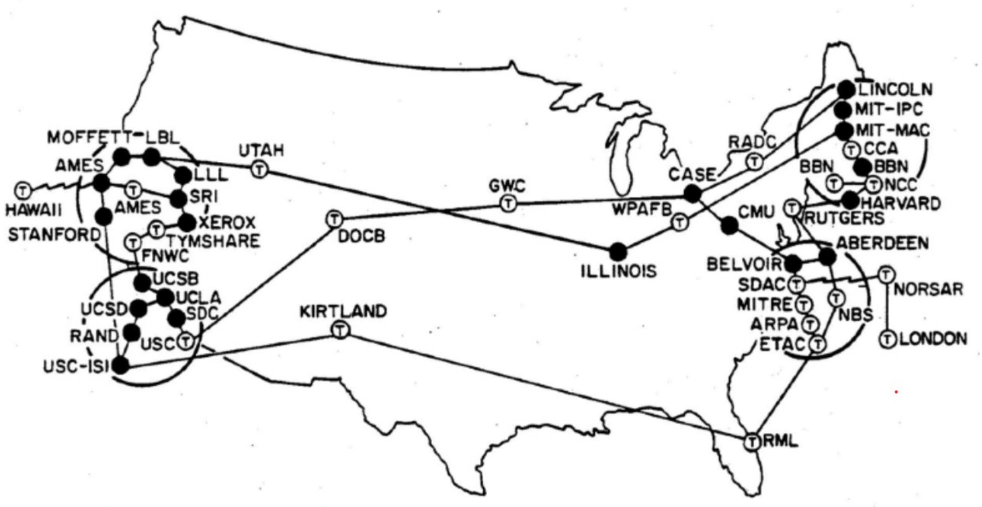


- Electrical telegraphs & Morse code (1800s)
  - Introduced near-instant long-distance text communication over wires



# Brief history of email

- ARPANET (Predecessor of modern internet)
  - Developed by the U.S. Department of Defense (1971)
  - The first email was sent via `SNDMSG` program on ARPANET
  - Introduced the **@** symbol to separate the recipient's username from the host computer's address



# Brief history of email

- SMTP (Simple Mail Transfer Protocol, 1980)
  - A standardized protocol for email transmission
  - Supports sending simple text messages

```
            +----------+                +----------+
+------+    |          |                |          |
| User |<-->|          |      SMTP      |          |
+------+    |  Sender- |Commands/Replies| Receiver-|
+------+    |   SMTP   |<-------------->|    SMTP  |    +------+
| File |<-->|          |    and Mail    |          |<-->| File |
|System|    |          |                |          |    |System|
+------+    +----------+                +----------+    +------+


             Sender-SMTP                Receiver-SMTP

                        Model for SMTP Use
```

Img: RFC 821. Simple Mail Transfer Protocol

# Brief history of email

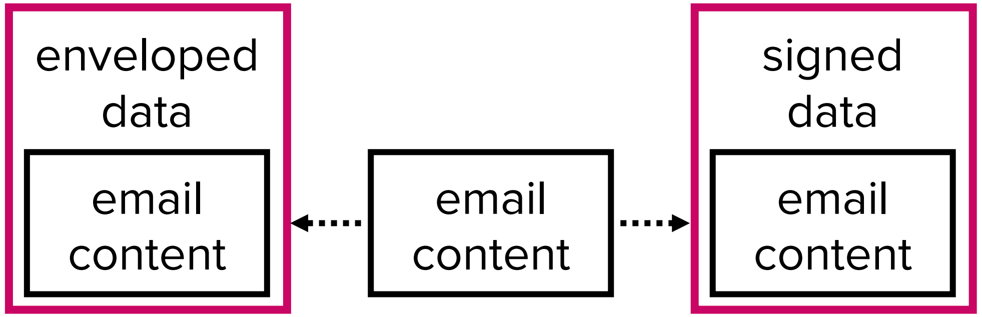
- MIME (Multipurpose Internet Mail Extensions, 1991)
  - Extends email format to handle multimedia content (images, audio, …)
  - Defines extra headers, such as:
    - MIME-Version
    - Content-Type
      - Text/plain, image/jpeg, audio/mp3, …
    - Content-Disposition
      - Inline, attachment
    - Content-Transfer-Encoding
      - base64, ascii, …

```
MIME-Version: 1.0
Content-Type: multipart/mixed;
boundary=frontier
This is a message with multiple parts in MIME format. --frontier

Content-Type: text/plain
This is the body of the message. --frontier

Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+
VGhpcyBpcyB0aGUg Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9
ib2R5Pgo8L2h0bWw+Cg==

--frontier--
```

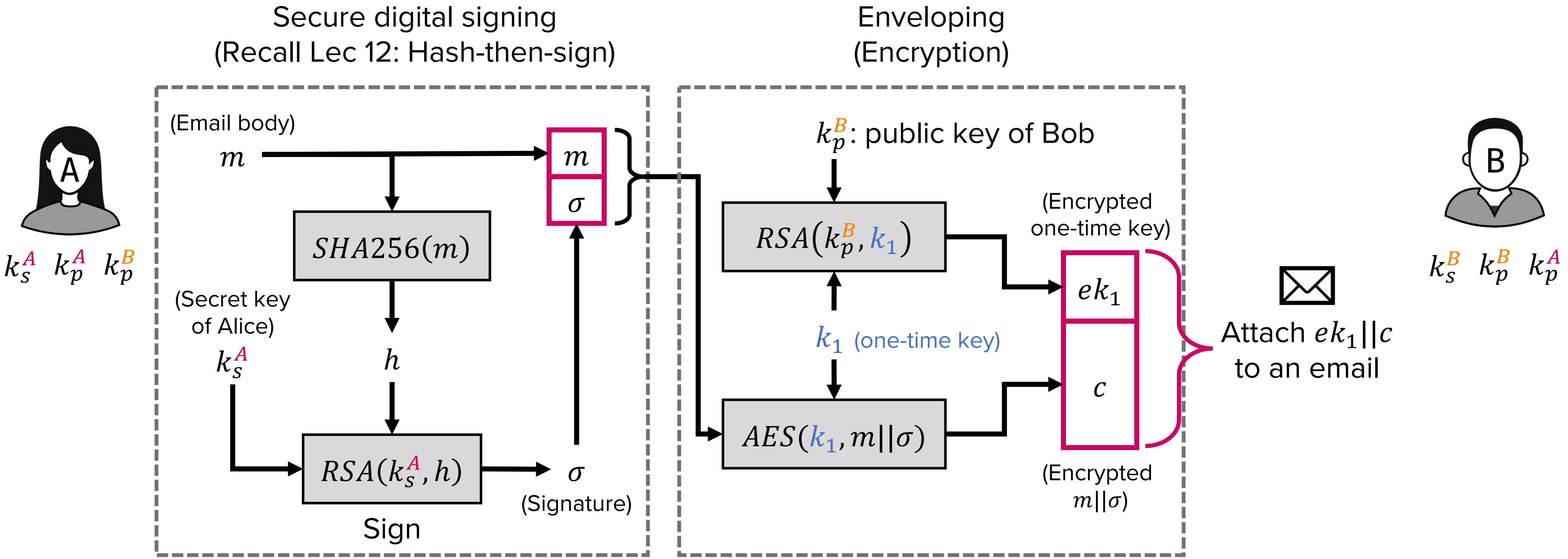
Key observation: None of these early protocols inherently provided security

# S/MIME: Secure MIME

- S/MIME: A set of MIME content types specifically designed to support encryption and/or digital signatures
  - e.g., Content-Type: application/pkcs7-mime (PKCS: public-key cryptography standards)
- Core functionality:
  - Enveloped data: Encrypts the message for confidentiality
  - Signed data: Digitally signs the message for integrity and authenticity
  - Signed and enveloped data: Combines both encryption and signing



# S/MIME workflow



# S/MIME email example

```
MIME-Version: 1.0
Message-Id: <93589100519290l5@postech.ac.kr>
Date: Tue, 02 Apr 2024 00:16:31 +0900 (Korea Standard Time)
From: alice@postech.ac.kr
To: bob@postech.ac.kr
Subject: email example
Content-Type: application/pkcs7-mime; name=smime.p7m; smime-type=enveloped-data
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
```

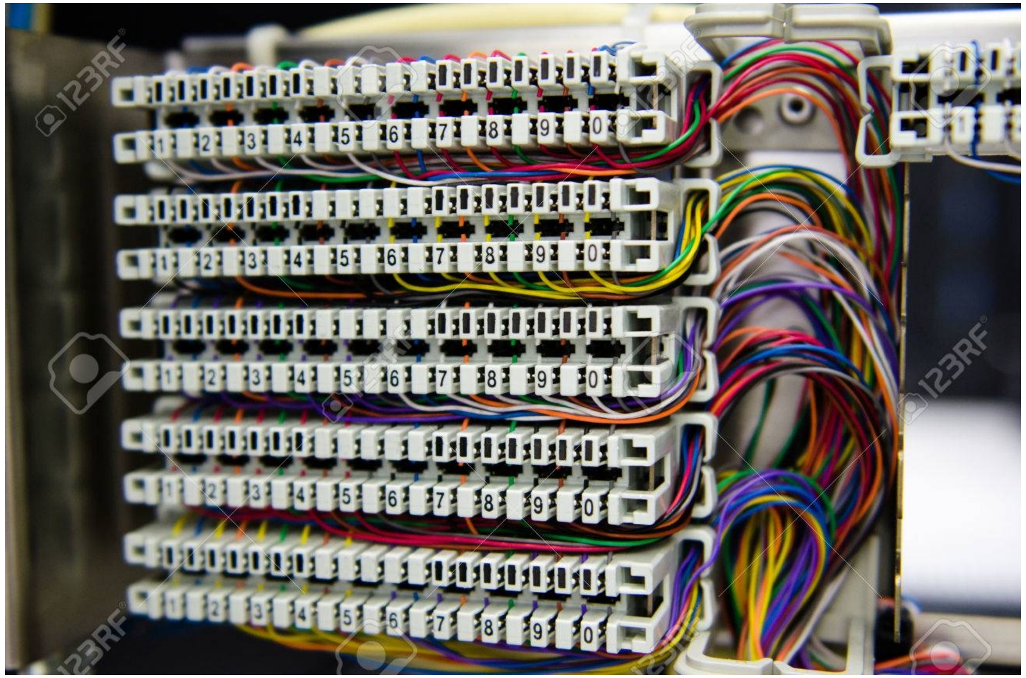

```
MIIBHgYJKoZIhvcNAQcDoIIBDzCCAQsCAQAxgcAwgb0CAQAwJjASMRAwDgYDVQQDEwdDYXJ
sUlNBAhBGNGvHgABWvBHTbi7NXXHQMA0GCSqGSIb3DQEBAQUABIGAC3EN5nGIiJi2lsGPcP
2iJ97a4e8kbKQz36zg6...bGgzoyEd8Ch4H/dd9gdzTd+taTEgS0ipdSJuNnkVY4/M652jK
LFf02hosdR8wQwYJKoZIhvcNAQcBMBQGCCqGSIb3DQMHBAgtaMXpRwZRNYAgDsiSf8Z9P43
LrY4OxUk660cu1lXeCSFOSOpOJ7FuVyU=
```


base64-encoded $ek_1||c$ attachment

# Secure Socket Layer (SSL) / Transport Layer Security (TLS)

# Background: Computer networks

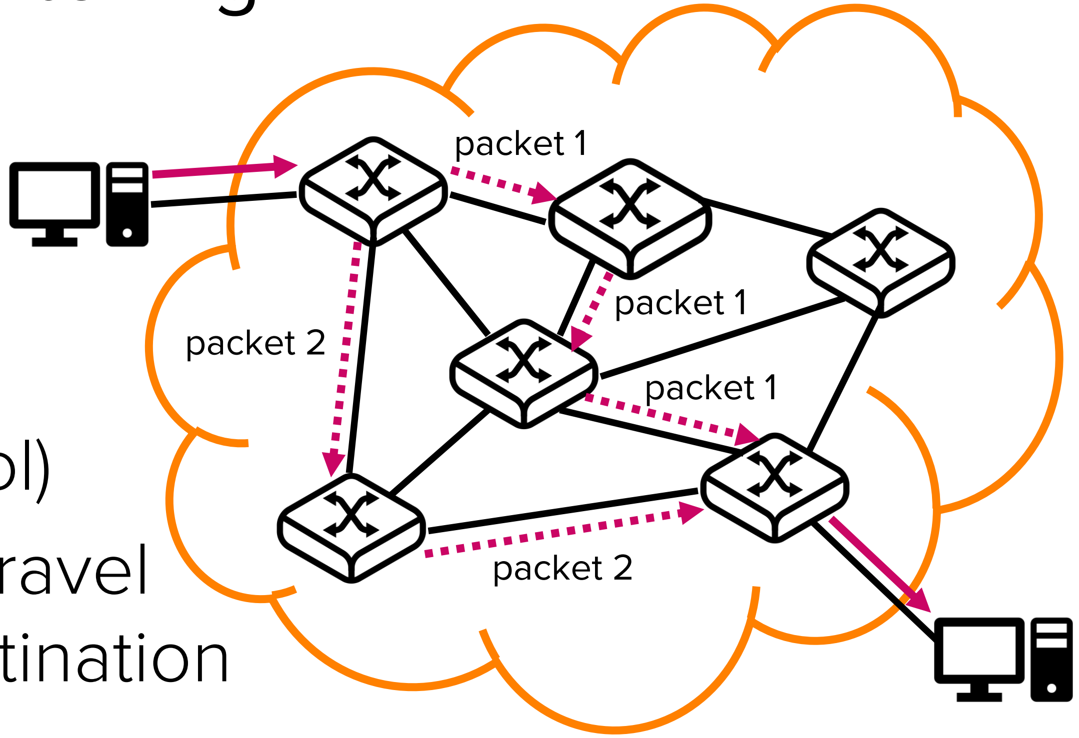
- Remote communication before internet: Circuit switching
  - Legacy phone network
  - Establish a single route through a sequence of hardware devices for two nodes to communicate
    - Route == connected wire
  - Data (electric current) is sent over the route
  - The route is maintained until the communication ends



**Telephone switchboard**

# Background: Computer networks

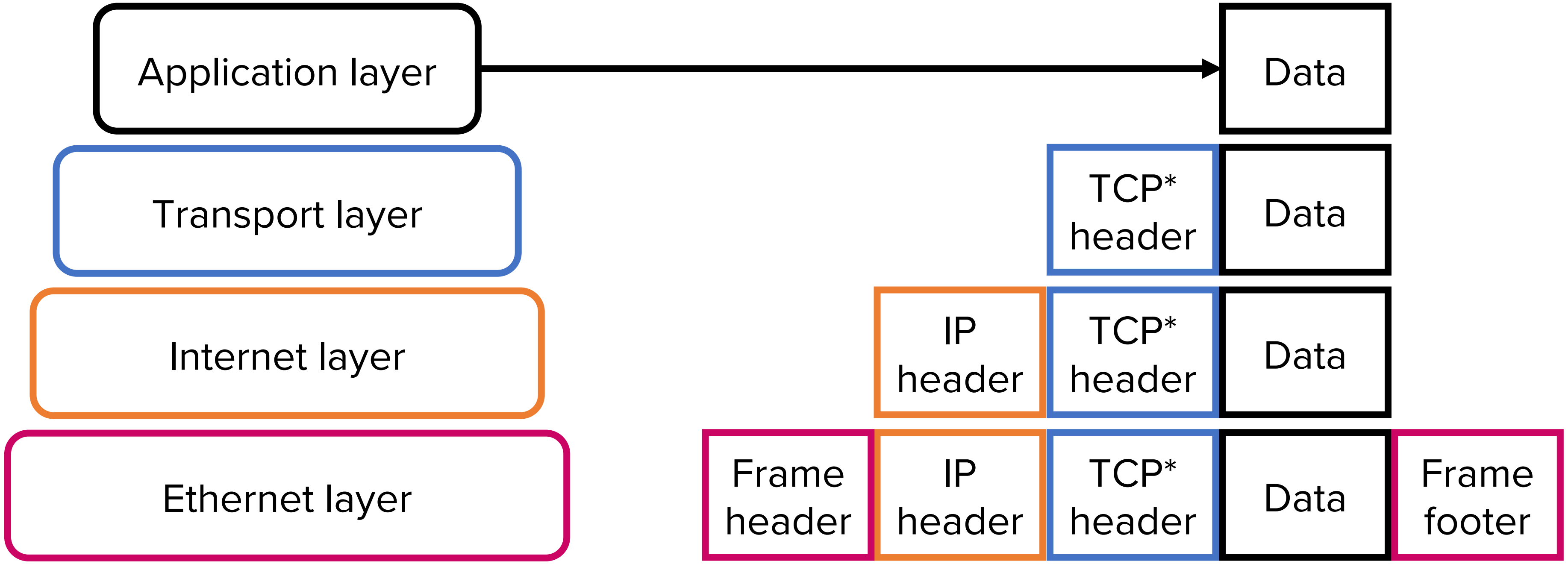
- Internet communication: Packet switching
  - Data is split into smaller packets
  - Packets are transported independently through the network
  - Network switches determine the best route for each packet (routing protocol)
  - Consequently, different packets can travel different paths to reach the same destination




**Packets sent over routers**

# Background: Computer networks

- Layers in networking
  - Higher layers use the services of lower layers via encapsulation




**TCP/IP network model** **Data encapsulation**

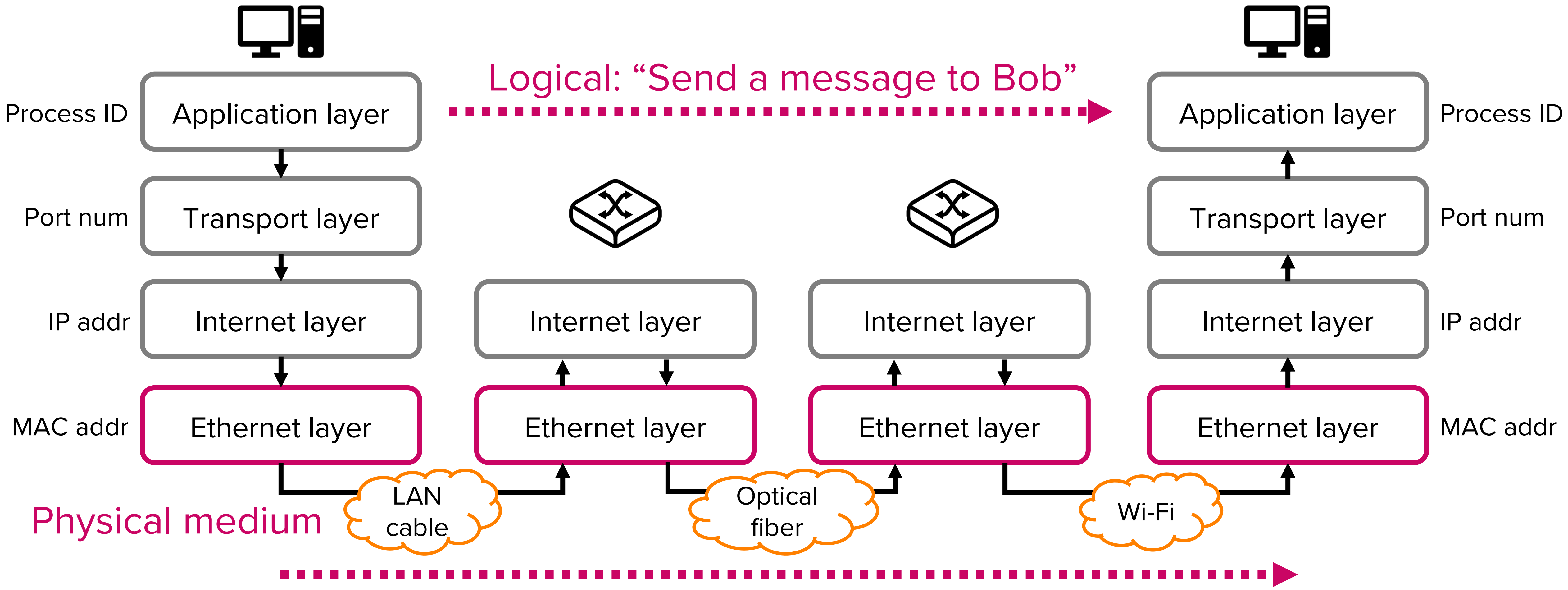
*or UDP

# Background: Computer networks

- Logical and physical data flow

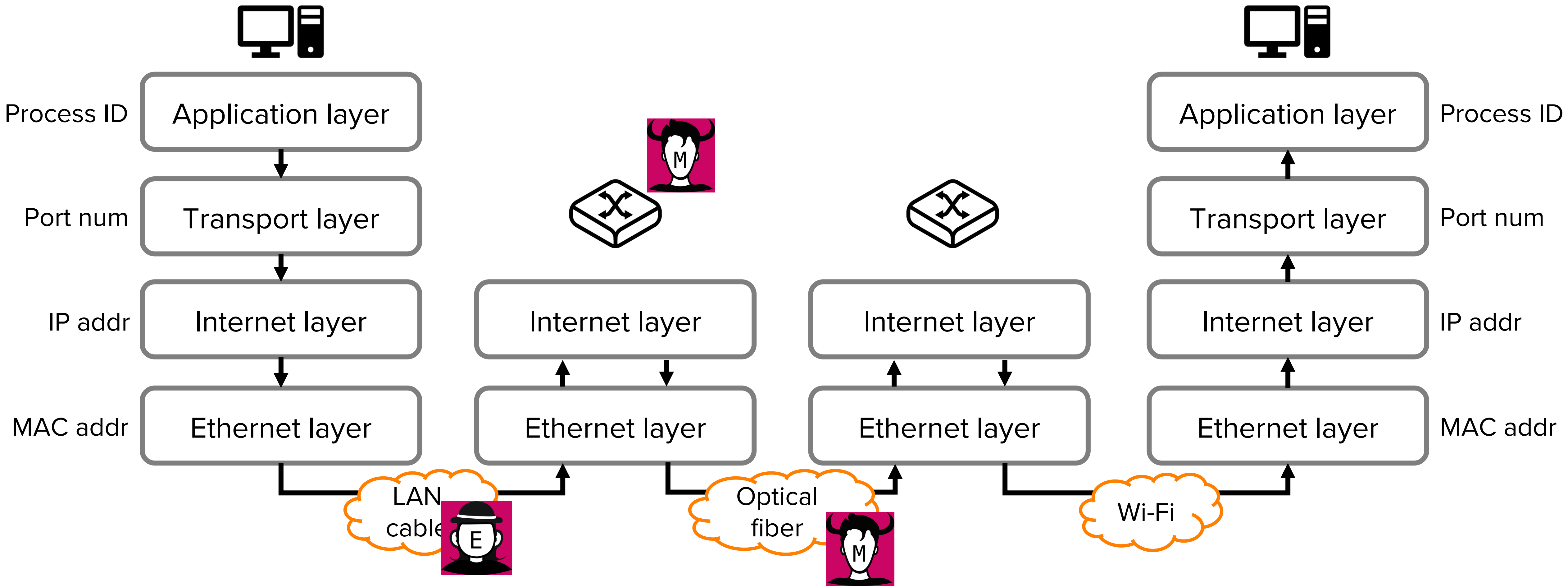
# Background: Computer networks

- Logical and physical data flow



# Security?

## Confidentiality, integrity, authenticity?

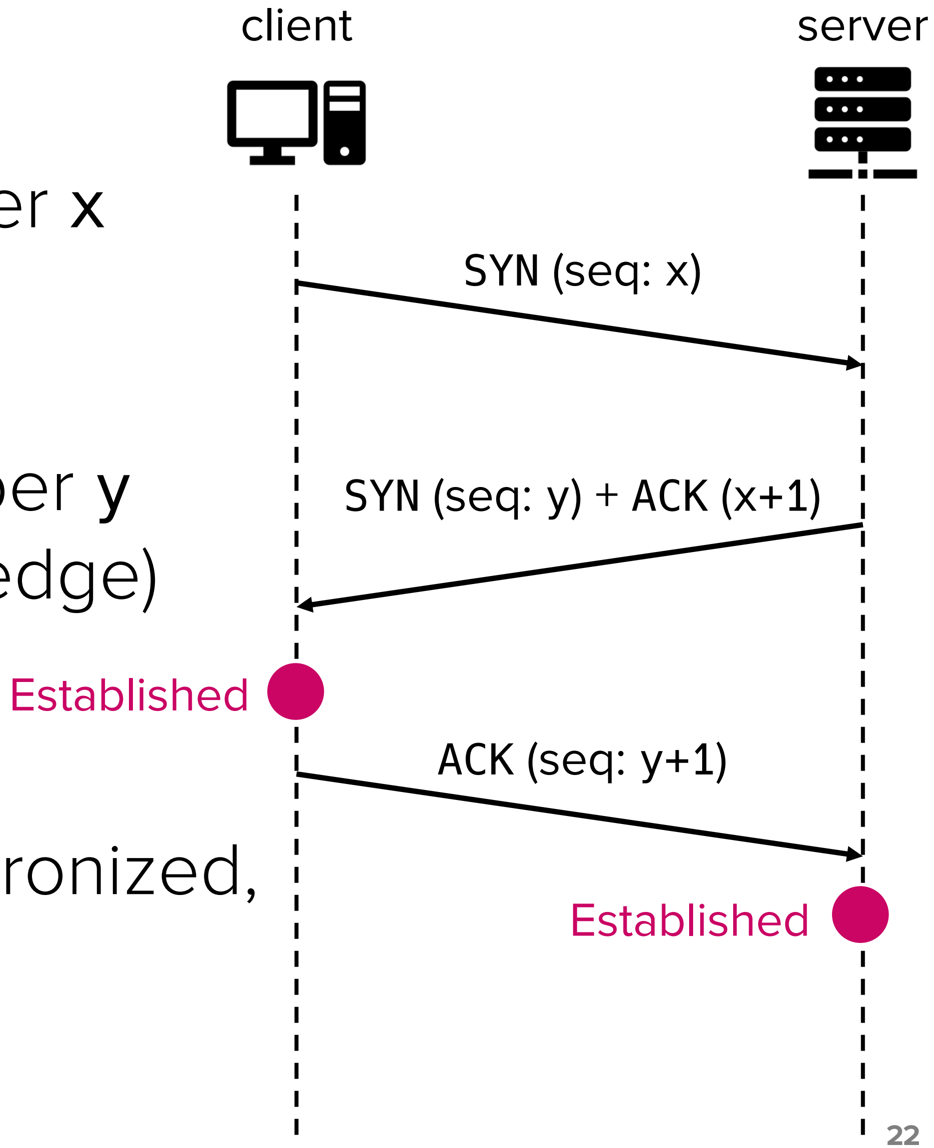


# Background: Transport layer protocols

- TCP and UDP
  - TCP (Transmission Control Protocol): For reliable data transfer
    - Client and server establish connection via the 3-way handshake
      - Client SYN → Server SYN-ACK → Client ACK
  - UDP (User Datagram Protocol): For faster data transfer
    - Connection-less
    - Does not provide reliability nor message ordering

# Background: TCP handshake

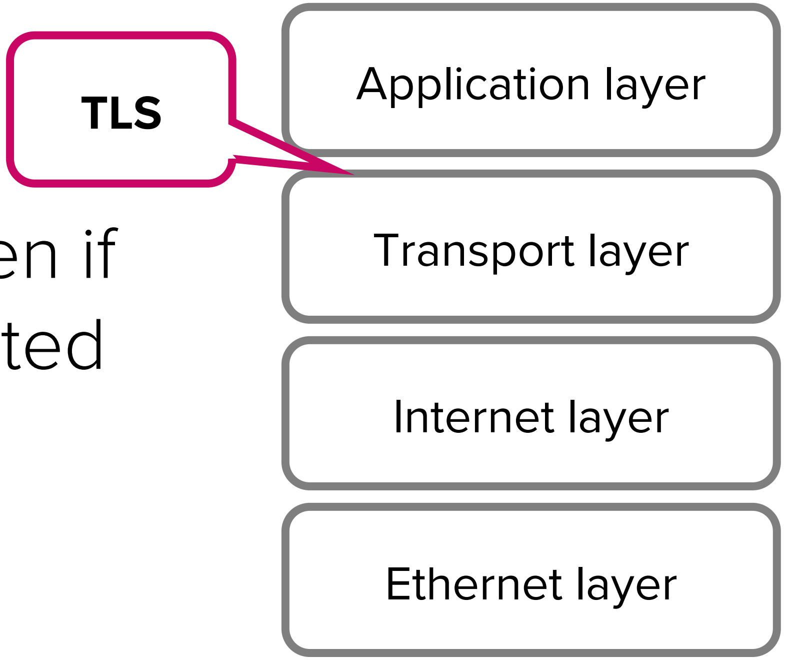
- 3-way handshake
  1. Client selects an initial sequence number x and sends a **`SYN`** (synchronize) packet to the server
  2. Server selects an initial sequence number y and responds with a **`SYN+ACK`** (acknowledge) packet
  3. Client responds with an **`ACK`** packet
  4. Once the sequence numbers are synchronized, connection is established



# SSL/TLS protocol

- Secure Sockets Layers protocol (SSL)
  - Outdated and replaced by TLS. “SSL” refers to TLS now
- Transport Layer Security protocol (TLS)
  - Built on top of TCP
  - Goal: End-to-end encryption and integrity, even if every intermediate node/connection is untrusted

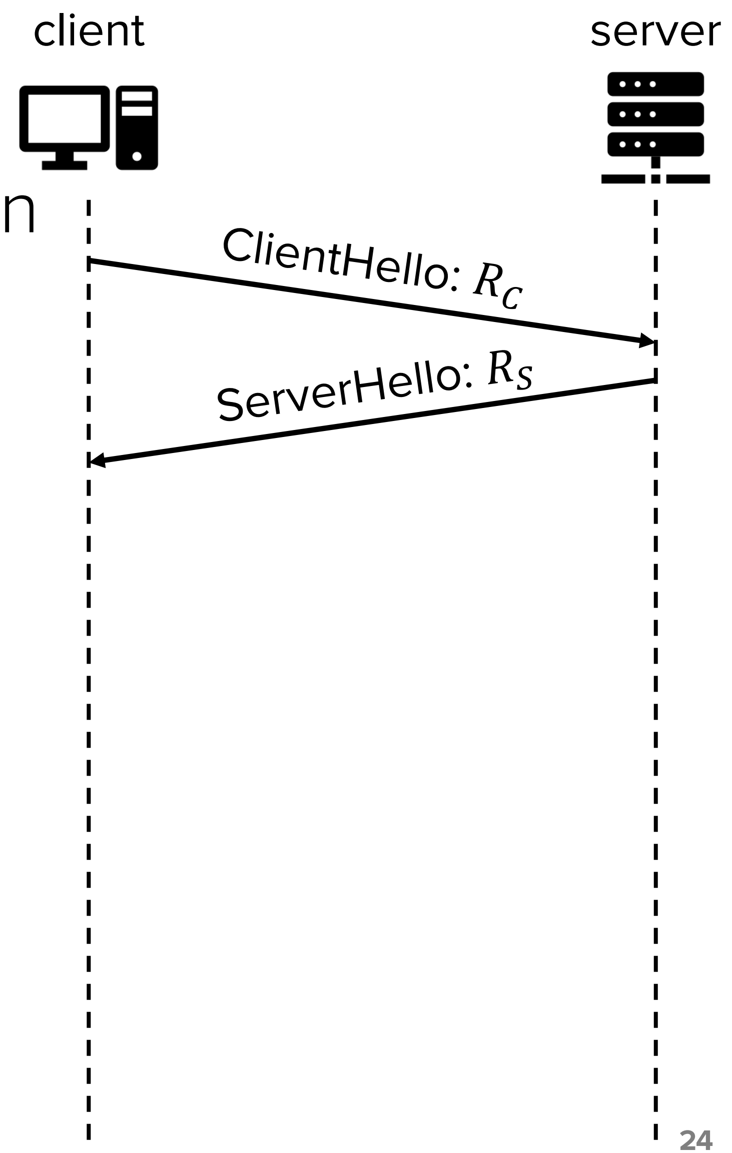

**Transport layer encapsulation** / **Transport layer encapsulation with TLS**




**TCP/IP network model**

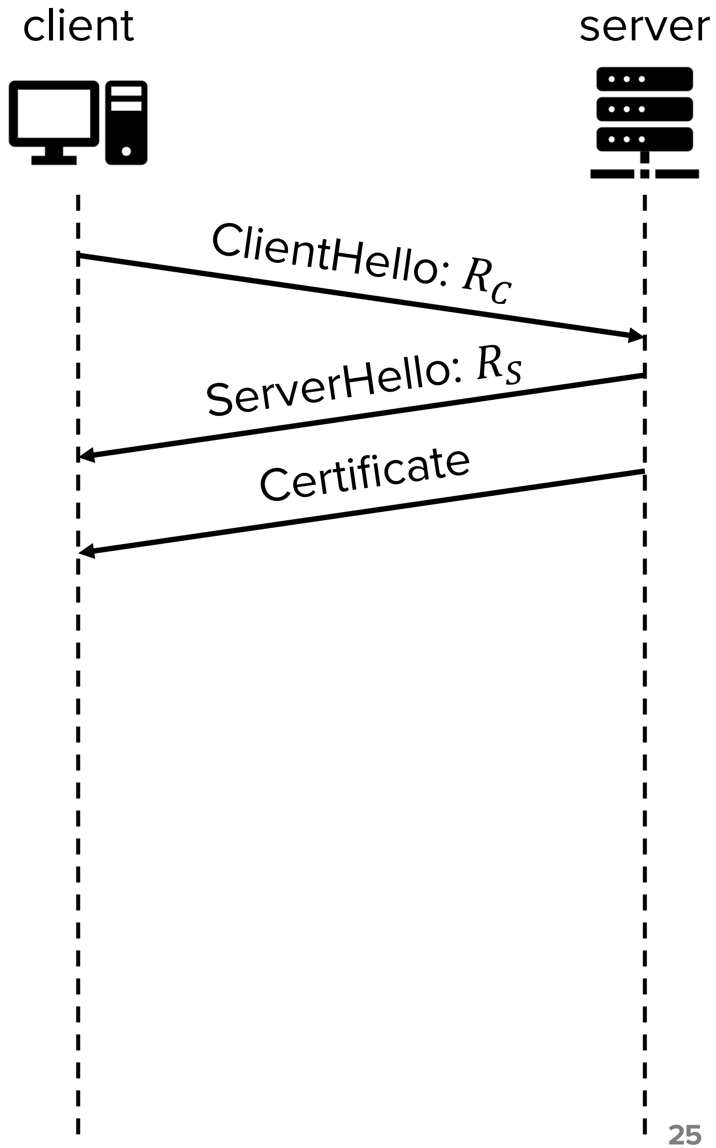
# TLS handshake

- Step 1: Exchange hellos
  - Assumption: A TCP connection has already been established via 3-way handshake
  - Client sends **ClientHello**
    - A 256-bit random number $R_C$
    - A list of supported cryptographic algorithms
  - Server sends **ServerHello**
    - A 256-bit random number $R_S$
    - The algorithm to use (chosen from the client's list)
  - $R_c$ and $R_s$ prevent replay attacks



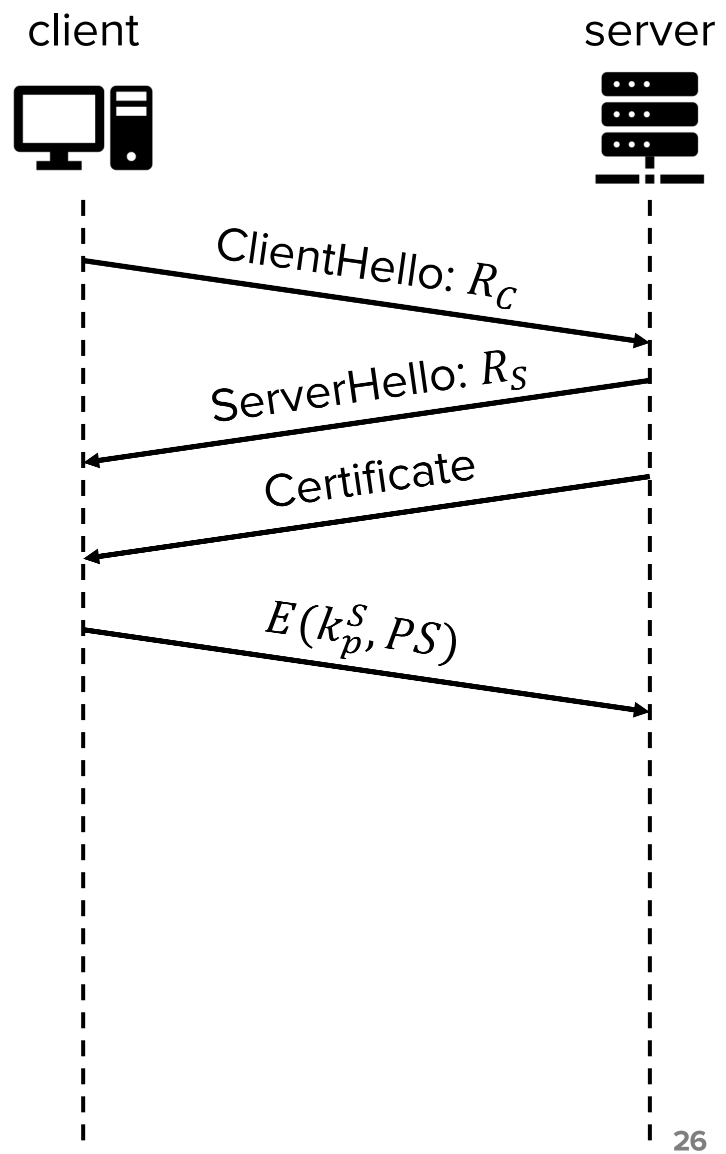
# TLS handshake

- Step 2: Server sends its certificate
  - Recall: Certificate includes the server's identity and public key, signed by a trusted CA
  - Client verifies the server's certificate
    - Using the CA's public key
  - The client now knows the server's public key
    - Server's public key: $k_p^S$
    - The client is not yet sure if it is talking to the legitimate server (not an impersonator)
      - Since certificates are public, anyone can present anyone else's certificate



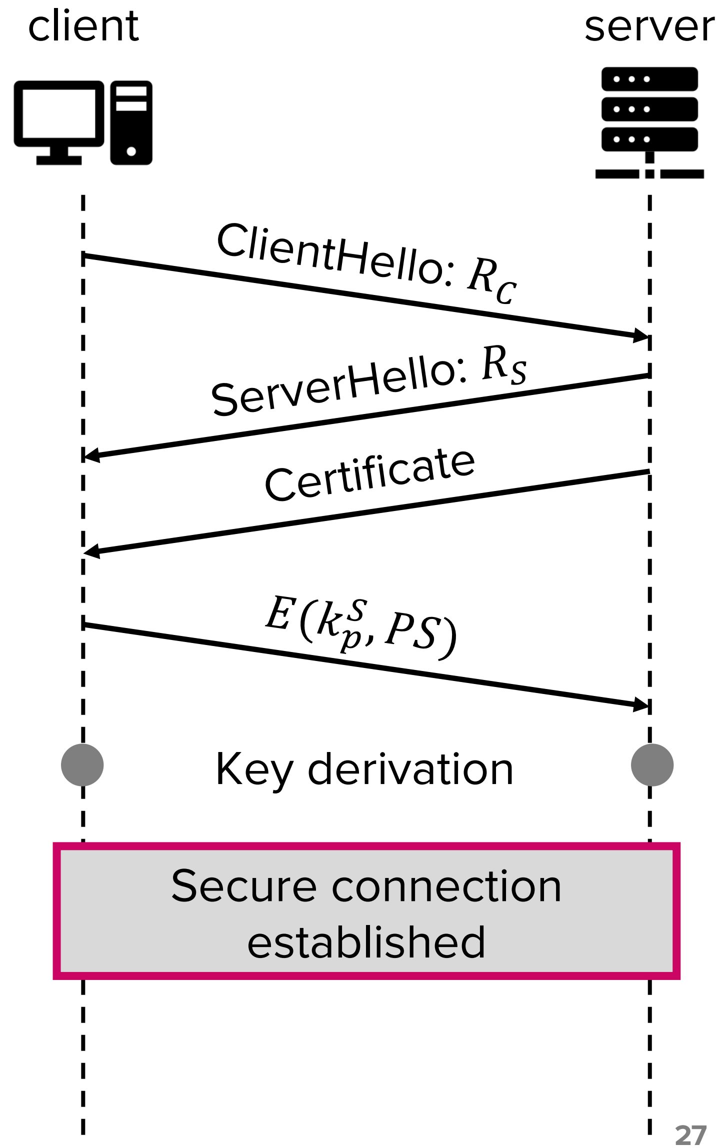
# TLS handshake

- Step 3: Share premaster secret
  - The client randomly generates a premaster secret ($PS$)
  - The client encrypts $PS$ with the server's public key ($k_p^S$) and sends it to the server
  - The server decrypts $PS$ using its secret key ($k_s^S$)
    - No one else can decrypt $E\left(k_p^S, PS\right)$
    - Therefore, if the server presents a valid $PS$ later, the client can be assured that the server is not an impersonator



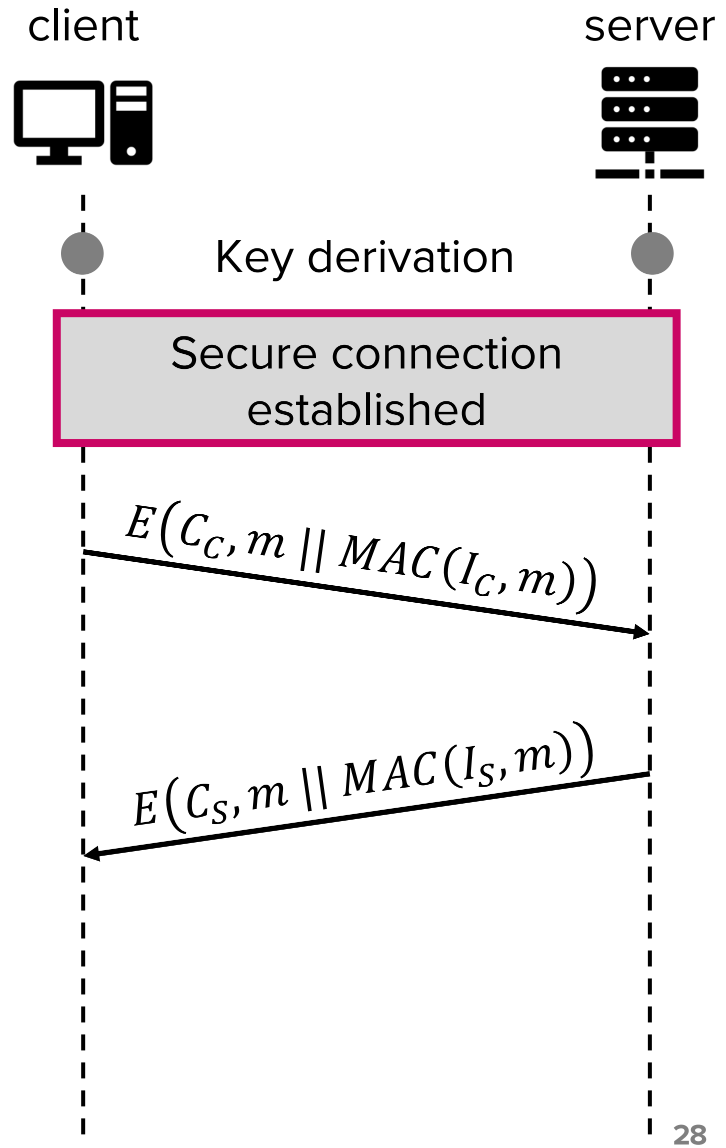
# TLS handshake

- Step 4: Derive symmetric session keys
  - Both sides derive session keys from the same $R_C$, $R_S$, and $PS$
    - Usually by seeding a PRNG with $R_C$, $R_S$, and $PS$
    - Any difference would result in different session keys
  - Four symmetric session keys are derived
    - $C_C$: Encryption key for client → server msgs
    - $C_S$: Encryption key for server → client msgs
    - $I_C$: For generating MAC of client → server msgs
    - $I_S$: For generating MAC of server → client msgs



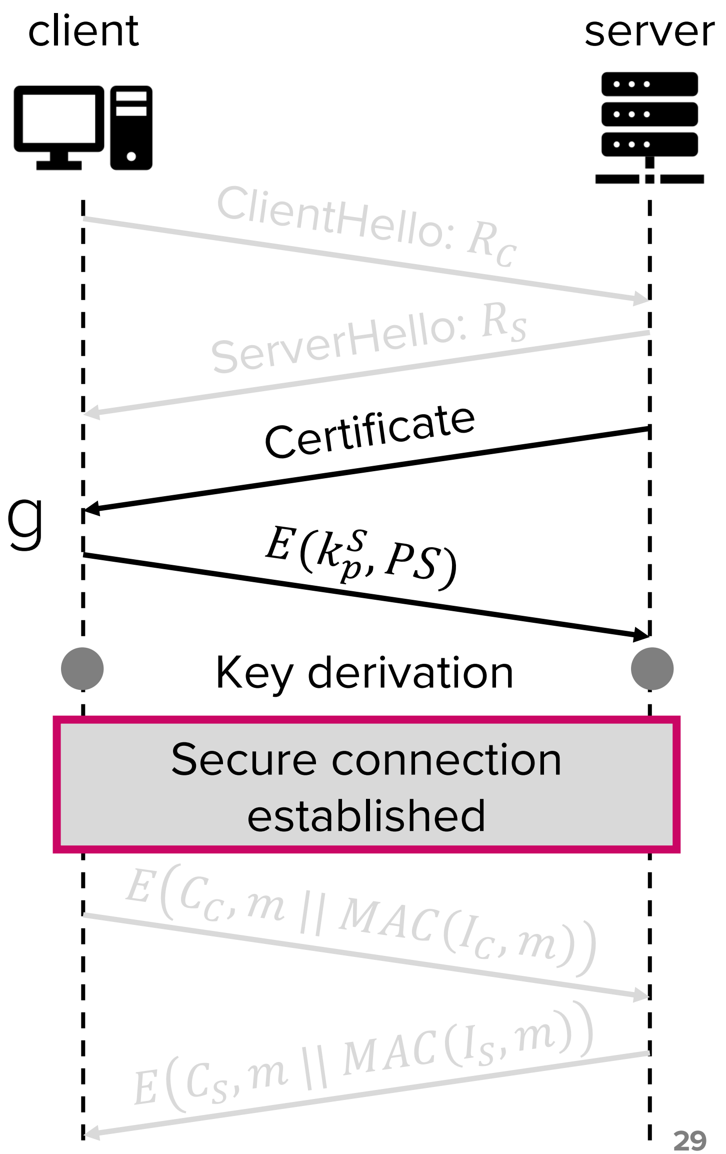
# TLS message exchange

- Messages can now be sent securely
  - Utilize Authenticated Encryption (AE)
    - With the derived session keys, generate MAC of $m$, append the MAC to $m$, and then encrypt
    - Note:
      - Even though Encrypt-then-MAC is considered safer (recall: Lecture 11), TLS uses MAC-then-Encrypt for backward compatibility with legacy applications



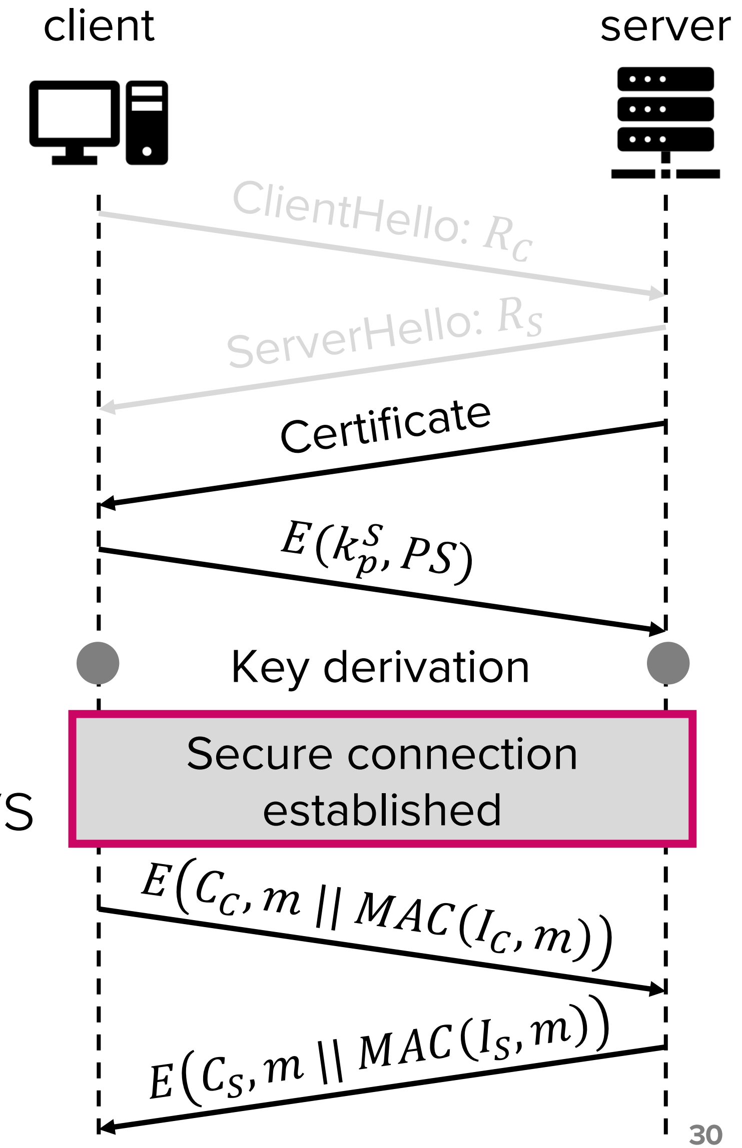
# Security of TLS

- Authenticity: Can client make sure that it is talking to the legitimate server?
  - The server sends its certificate, so the client can verify it and obtain server's public key $k_p^S$
  - The server proves that it owns the corresponding secret key $k_s^S$ by decrypting the encrypted $PS$
  - An impersonator cannot derive the same set of session keys as he/she does not own the secret key to decrypt the encrypted $PS$



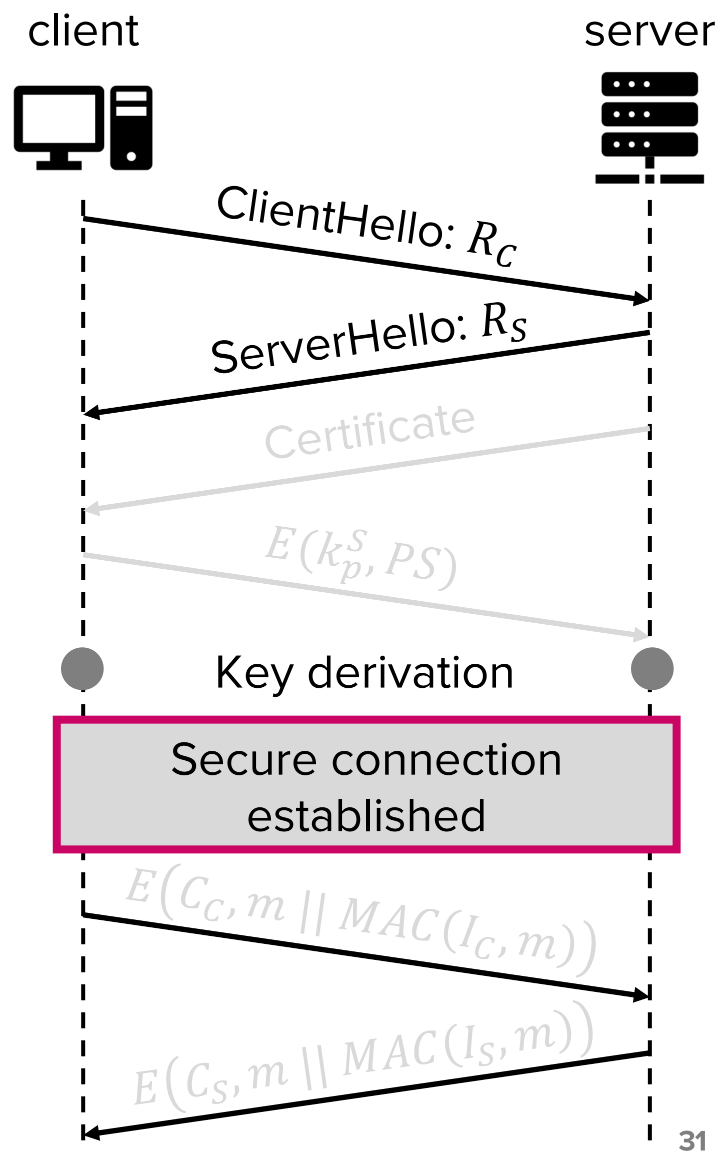
# Security of TLS

- Confidentiality and Integrity: How can both parties ensure that attackers cannot read or tamper with their messages?
  - The attacker does not know $PS$
    - Cannot decrypt $E\left(k_p^S, PS\right)$ without $k_s^S$
  - The session keys are derived from $PS$
    - $C_C$, $C_S$, $I_C$, and $I_S$
  - Authenticated encryption using the session keys provide confidentiality and integrity



# Security of TLS

- Robustness to replay attacks:
  How can both parties ensure that
  an attacker is not replaying old messages
  from a past TLS connection?
  - Every TLS handshake uses a different random values ($R_C$ and $R_S$) that are exchanged during via ClientHello and ServerHello messages
  - The session keys are derived from $R_C$ and $R_S$
    - These keys are different for every TLS connection



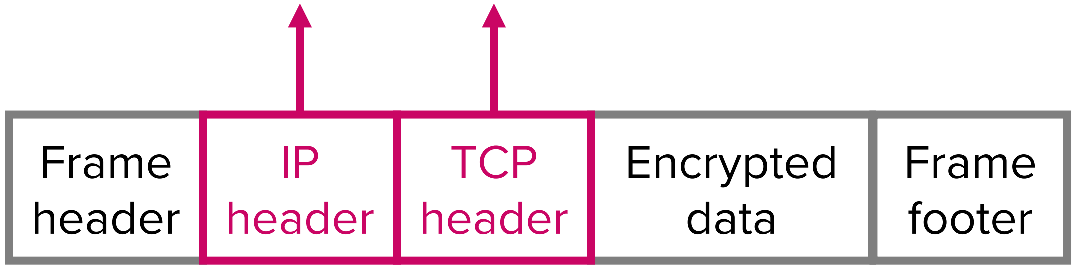
# What TLS does and doesn't

- TLS guarantees end-to-end security
  - Even if every entity between the client and the server is malicious, TLS provides a secure communication channel
  - Examples
    - A local attacker captures all Wi-Fi communications
      - The attacker cannot decipher or manipulate TLS messages w/o the session keys
    - A MitM tries to inject TCP packets
      - These packets will be rejected (cannot generate valid MACs w/o session keys)
  - Caveat: TLS does not guarantee end-to-end security if one end is malicious (e.g., communicating with a malicious server)
    - TLS only protects data in transit

# What TLS does and doesn't

- TLS does not guarantee anonymity
  - Anonymity: Hiding the client's and server's identities from attackers
  - Attackers can still figure out who is communicating with TLS
    - Server's certificate, containing server's identity, is sent during the handshake
    - Attacker can still observe IP addresses and ports from the headers of underlying IP and TCP layers

Required for routing (i.e., locating src/destination), so cannot be encrypted




**Encapsulation after TLS**

# What TLS does and doesn't

- TLS does not guarantee availability
  - Availability: Keeping the connection open in the face of attackers
  - Attackers can block or drop TLS packets to stop TLS connections
  - In other words, TLS connections can still be censored
    - South Korean government blocks access to porn and gambling websites

# TLS in Action: HTTPS

# HTTPS: HTTP over TLS

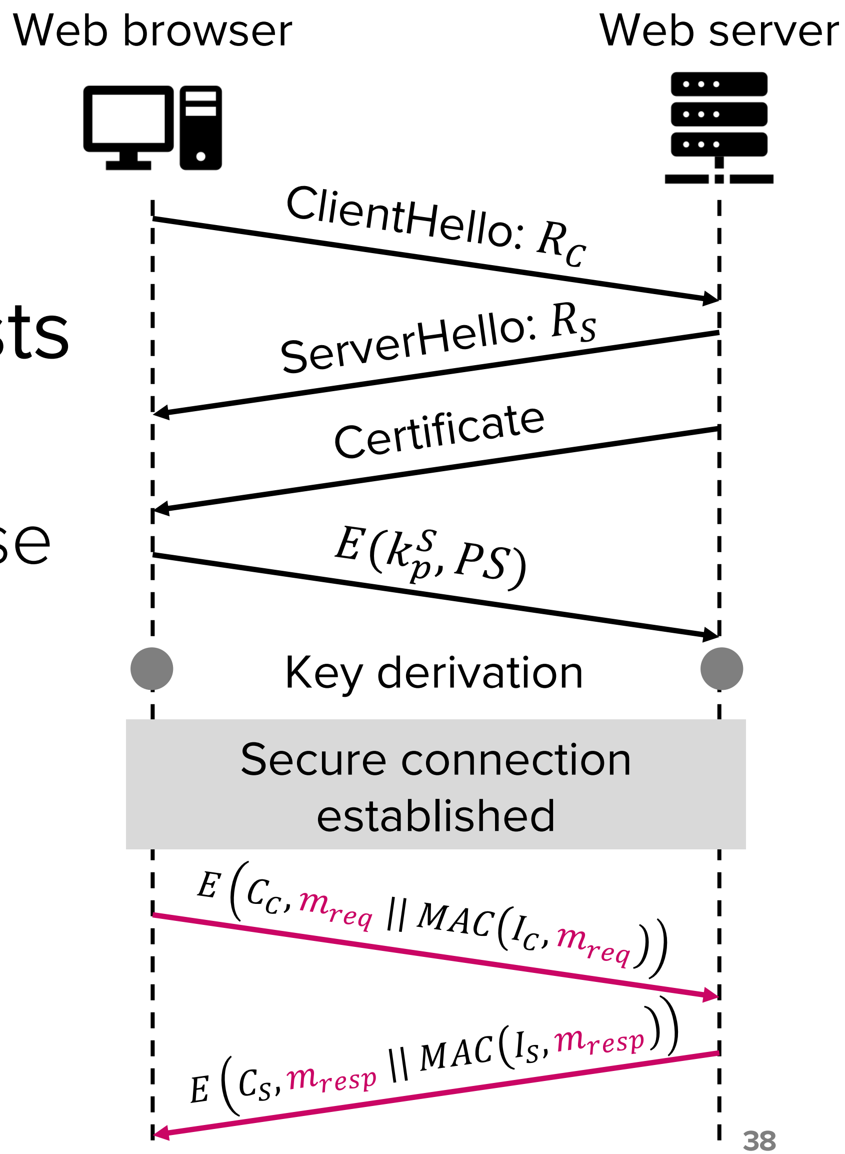
- Combination of HTTP (Hypertext Transfer Protocol) and TLS
  - TLS applied specifically for securing the communication between a web browser and a web server
- All modern browsers support HTTPS protocol
  - If not, avoid at all costs!
- URLs of the servers providing HTTPS connection start with https://

# HTTPS: HTTP over TLS

- Connection
  - HTTP connection uses port 80
  - HTTPS connection uses port 443, which invokes TLS protocol
- HTTPS encrypts:
  - URL path of the requested document (web page)
  - HTTP headers
  - Contents of the document
  - Form data (e.g., username and password)
  - Cookies between the server and the browser

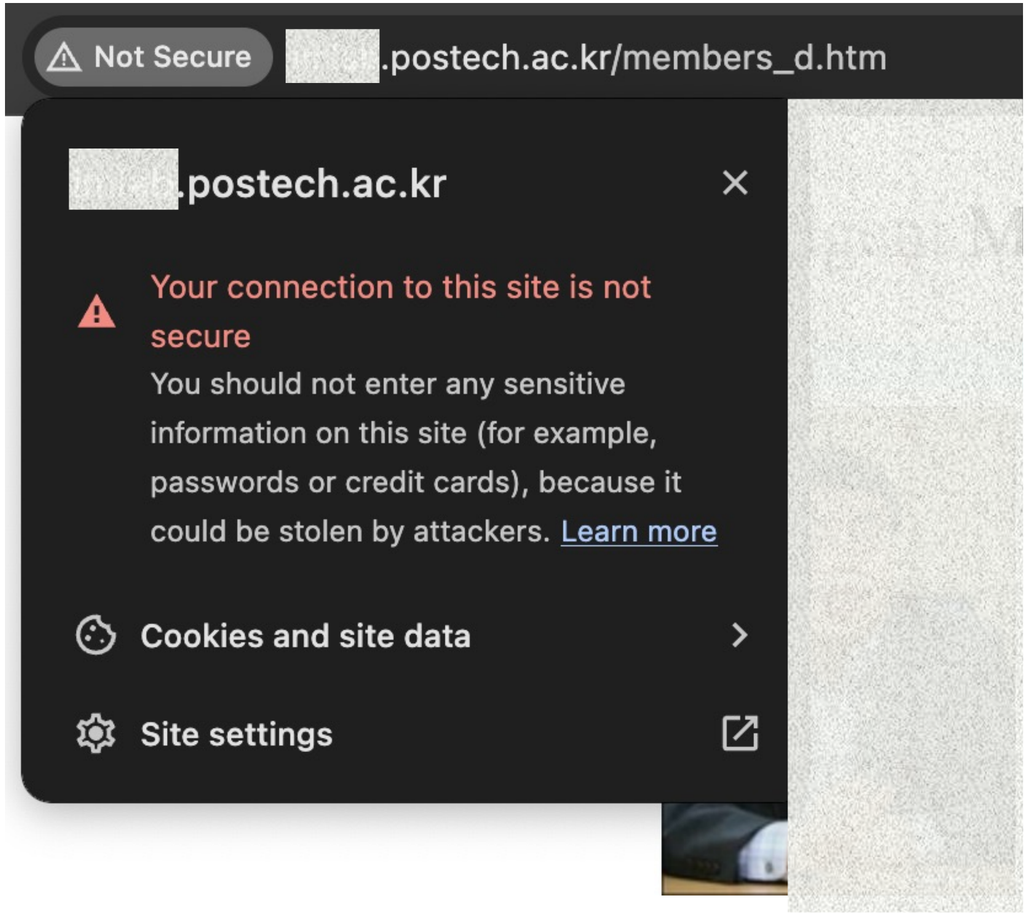
# HTTPS connection

- The web browser initiates a TLS connection to the web server
- After the TLS handshake, all HTTP requests and responses are encrypted
  - i.e., $m_{req}$: HTTP request, $m_{resp}$: HTTP response

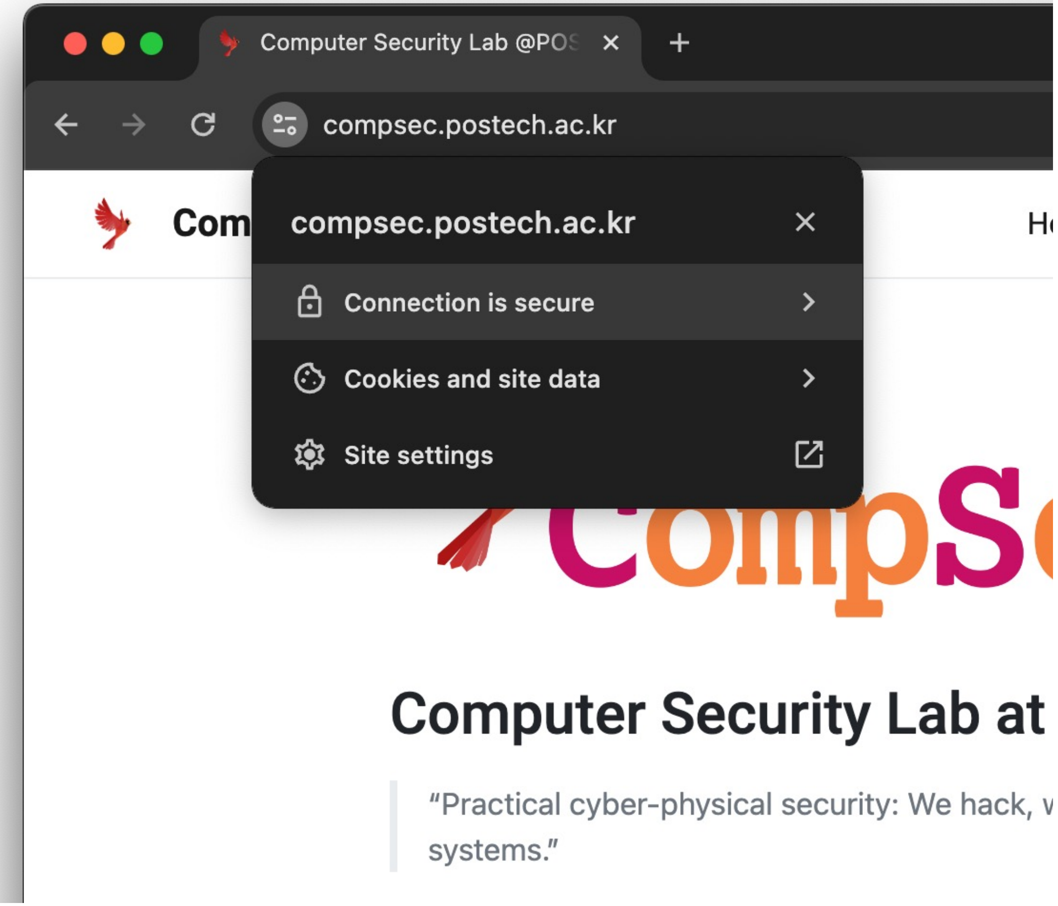


# HTTP vs HTTPS

http://[redacted].postech.ac.kr

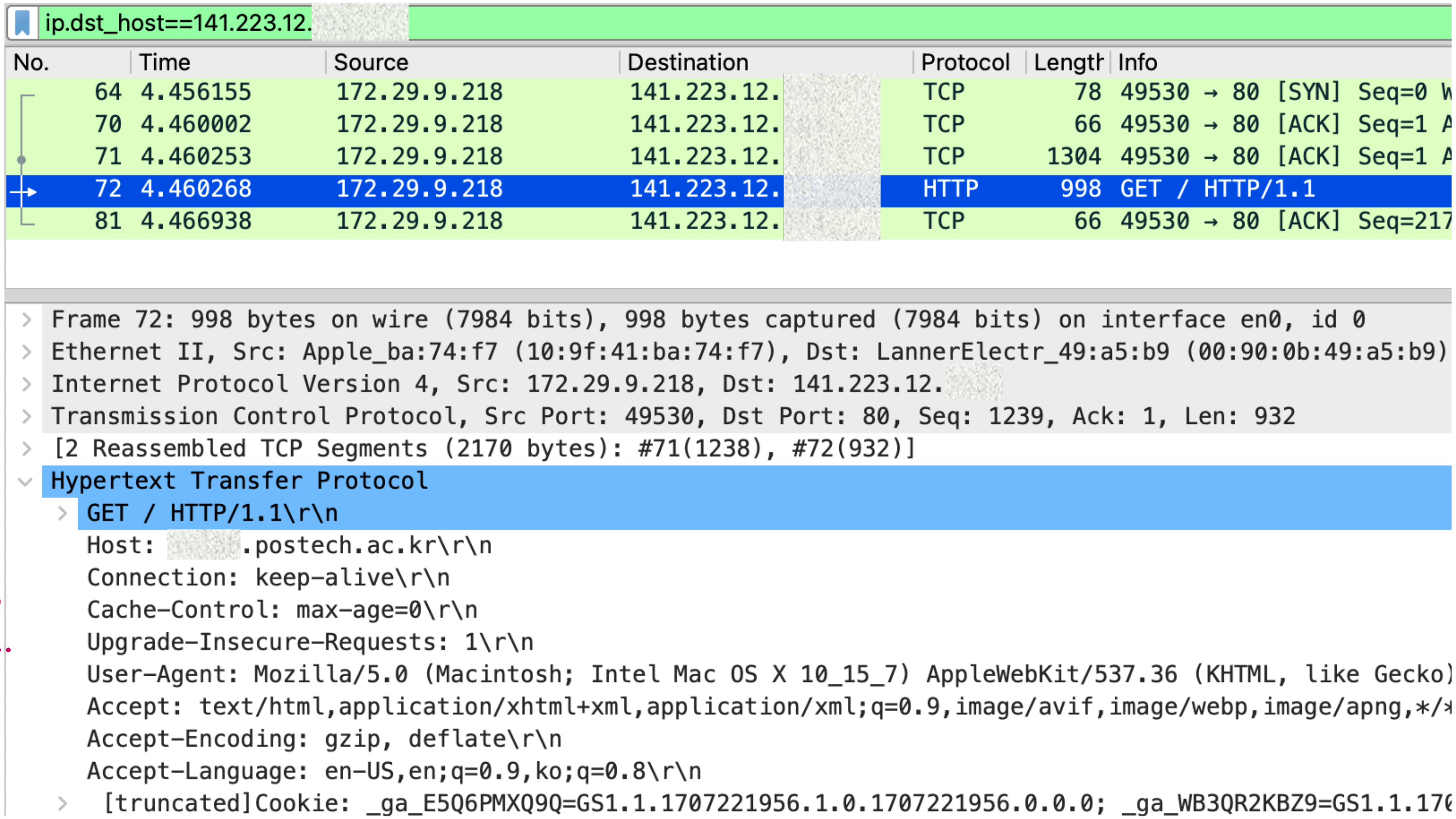



https://compsec.postech.ac.kr



# HTTP vs HTTPS

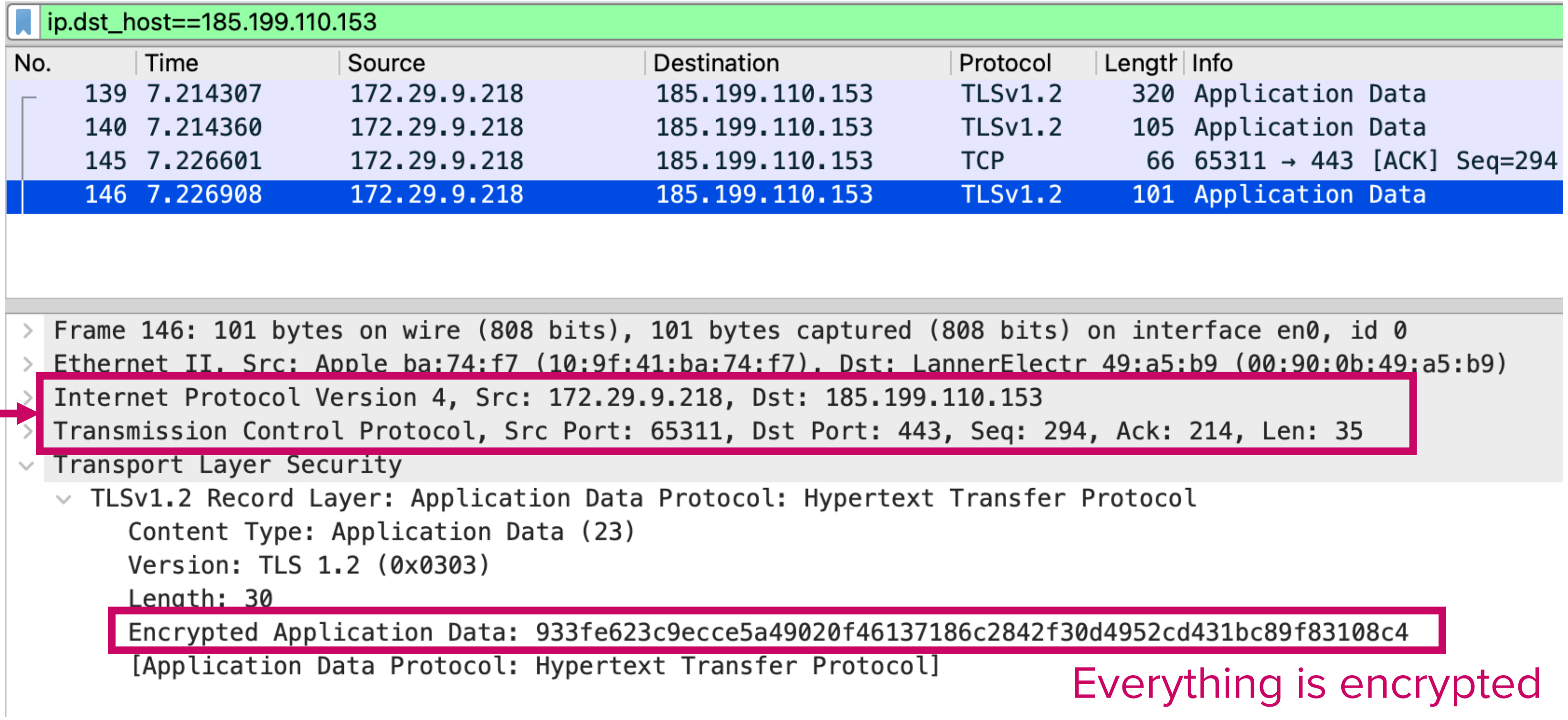
- HTTP packet captured with Wireshark




URL, HTTP header, contents, cookies, … are publicly visible

# HTTP vs HTTPS

- HTTPS packet captured with Wireshark




Everything is encrypted

Caution: HTTPS does not hide your identity

# Remaining Challenge: Can we trust CAs?

# Recall: Certificates are the key in TLS protocol

- Server sends its certificate
  - Server's identity (domain name) and its public key signed by CA's secret key
- The browser verifies the server's certificate
  - CA's public key is embedded in the browser
    - Recall: Lecture 12, trust anchor
  - The browser uses the CA's public key to verify the certificate
- Once verified, the browser trusts the server's public key

# Issues: Unknown CA

- What if the browser does not have the CA's public key?
  - Not all CA information is embedded
  - Typical behavior: Warn the user that the website is not verified
    - Connection can still be established, but the server's legitimacy is not assured
- Potential problems
  - The server indeed is a malicious server
  - End-to-end security is broken

# Issues: Revocation

- What if an attacker steals a server's private key?
  - The certificate with the corresponding public key is no longer valid
  - TLS certificates have an expiry date, but it takes time to expire
- Solution: Certificate revocation lists (CRL)
  - Recall: Lecture 12
  - The CA occasionally sends out lists of certificates that should be revoked
  - Browsers must regularly update the revocation lists

# Issues: Too many trusted CAs

- Recall: We designate multiple trust anchors to solve the single-point-of-failure problem
- A CA might be compromised or malicious, and issue fraudulent certificates
  - A CA gets hacked
  - An attacker bribes the CA to issue a fraudulent certificate
- Problem: Too many trust anchors
  - Modern browsers trust 100~200 CAs
  - One compromised CA is enough for attackers to launch large-scale attacks (the weakest link matters!)

# Issues: Too many trusted CAs

- Real-world incidents: Comodo SSL certificate breach (2011)
  - Comodo was a major CA
    - Not anymore
  - Comodo's account was compromised
  - Iranian hacker issues nine fraudulent SSL certificates for popular websites, including Gmail, Hotmail, Skype, …
  - (although not known) the hacker could impersonate these websites and intercept all TLS-encrypted traffic
    - Again, end-to-end security is broken if one end is malicious

# Issues: Too many trusted CAs

- Real-world incidents: DigiNotar server hack (2011)
  - DigiNotar was another major CA
  - Attacker (allegedly backed by Iranian government) compromised all eight certificate-issuing servers of DigiNotar
  - The attacker issued more than 500 fake certificates, including Google's

# Issues: Trust anchors

- Real-world incidents: Symantec mis-issuance (2017)
  - One of the largest providers of TLS certificates (not anymore)
  - Symantec issued certificates without sufficiently validating the identity of the entities requesting them
    - They issued certificates for domains without verifying ownership
      - "Hey Symantec, I own google.com. Can you issue a certificate?" → "Absolutely"
    - They issued certificates for non-existent domains
      - "Hey Symantec, I own xcvbmnzgirsjcxv.com. Can you issue a certificate?" → "Absolutely"
  - 30K problematic certificates were issued, according to Google
  - Chrome and Firefox revoked all Symantec-issued certificates

Still an unsolved problem!

# Modern CA example: Let’s Encrypt

- To use TLS, every web server needs to obtain and maintain certificates
  - Most certificate providers charge money for issuing TLS certificates
- Let’s Encrypt (LE) issues certificates for free
  - Web server requests a certificate
  - LE sends the server a file to be uploaded
  - The server uploads the file to the website
  - LE verifies that the file has appeared on the website
  - Identity verified (domain & ownership) → LE issues a certificate

# Summary

- SSL/TLS is a fundamental protocol for secure communication on the internet
- S/MIME secures email content end-to-end, providing encryption and/or digital signatures
- Certificates and CAs remain a critical piece of the trust model

# Questions?