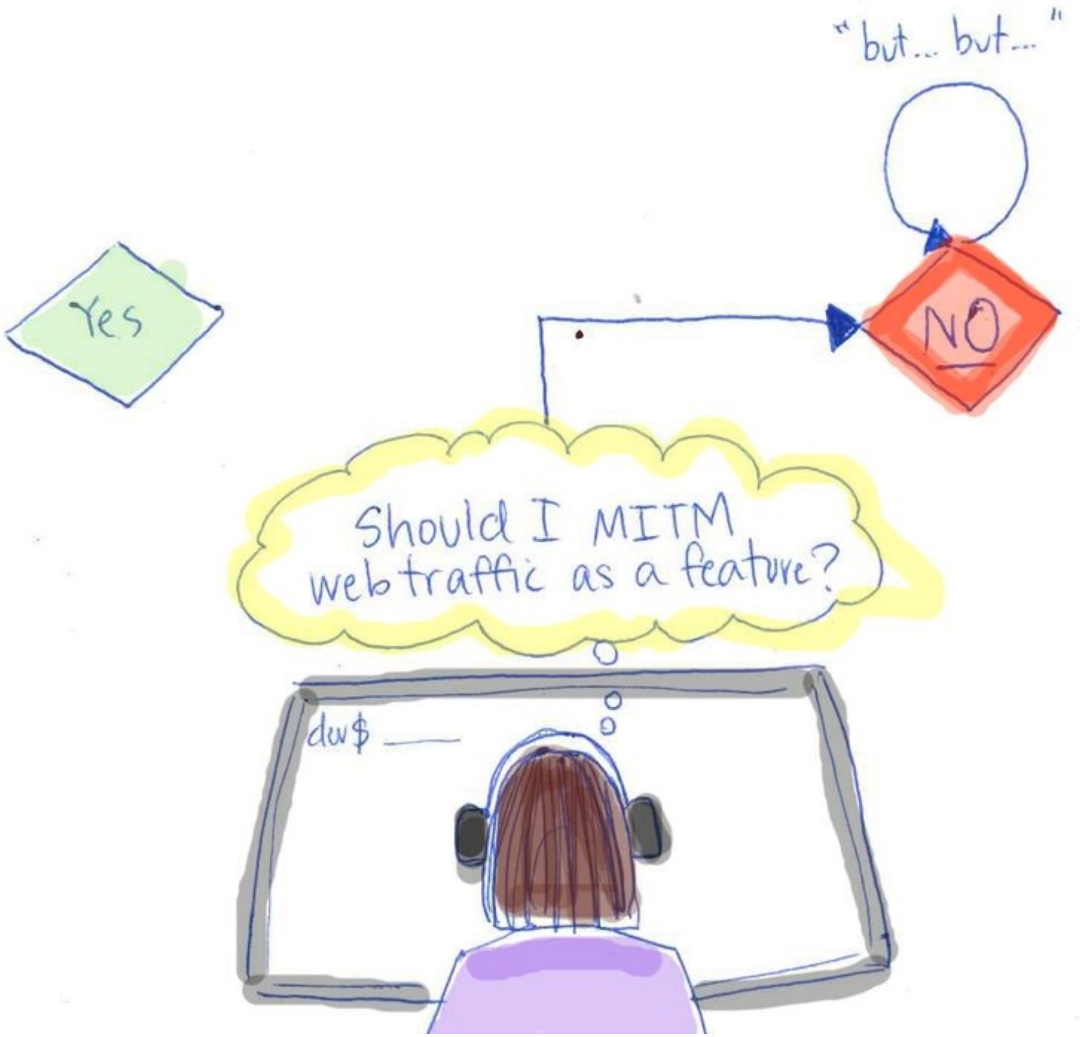
CS 88: Security and Privacy

20: Network Security @ The Transport Layer

18-04-2022

slides adapted from UC Berkeley, Stanford

SWARTHMORE COLLEGE



Credit: Adrienne Porter Felt (Google)

Reading Quiz

The Internet

Global network of networks that ..

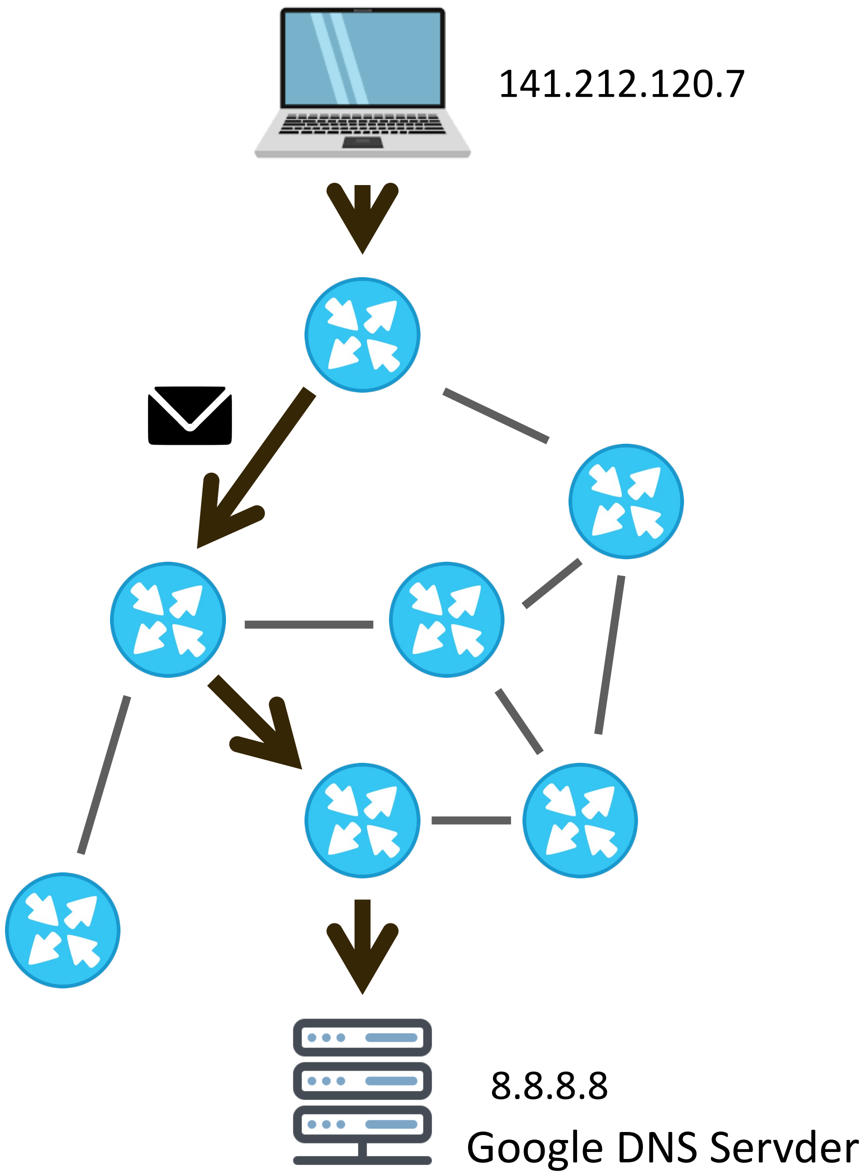
provides **best-effort** delivery of **packets** between connected hosts

Packet: a structured sequence of bytes

- Header: metadata used by network
- Payload: user data to be transported

Every host has a unique identifier — IP address

Series of routers receive packets, look at destination address on the header and send it one hop towards the destination IP address

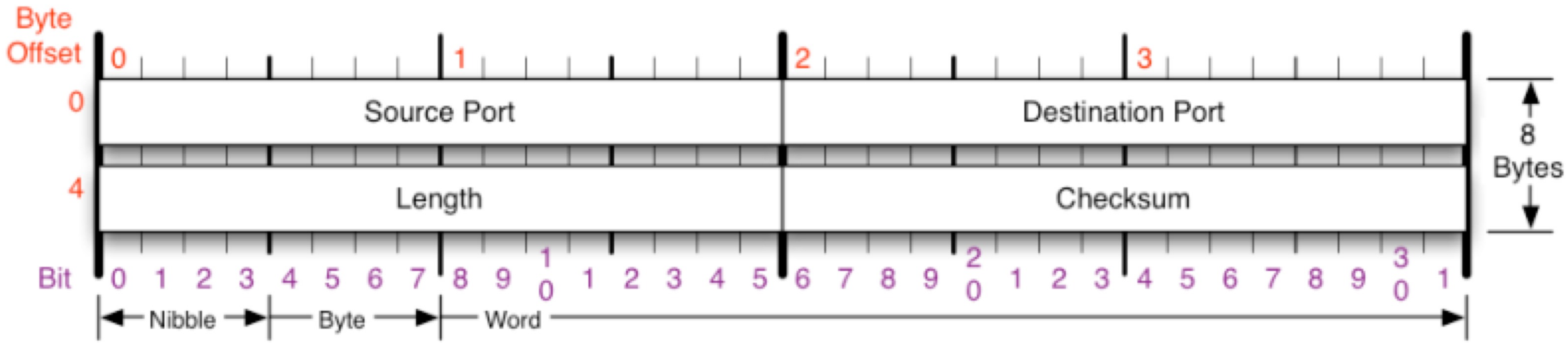


Network Protocols

We define how hosts communicate in published network protocols

Syntax: How communication is structured (e.g., format and order of messages)

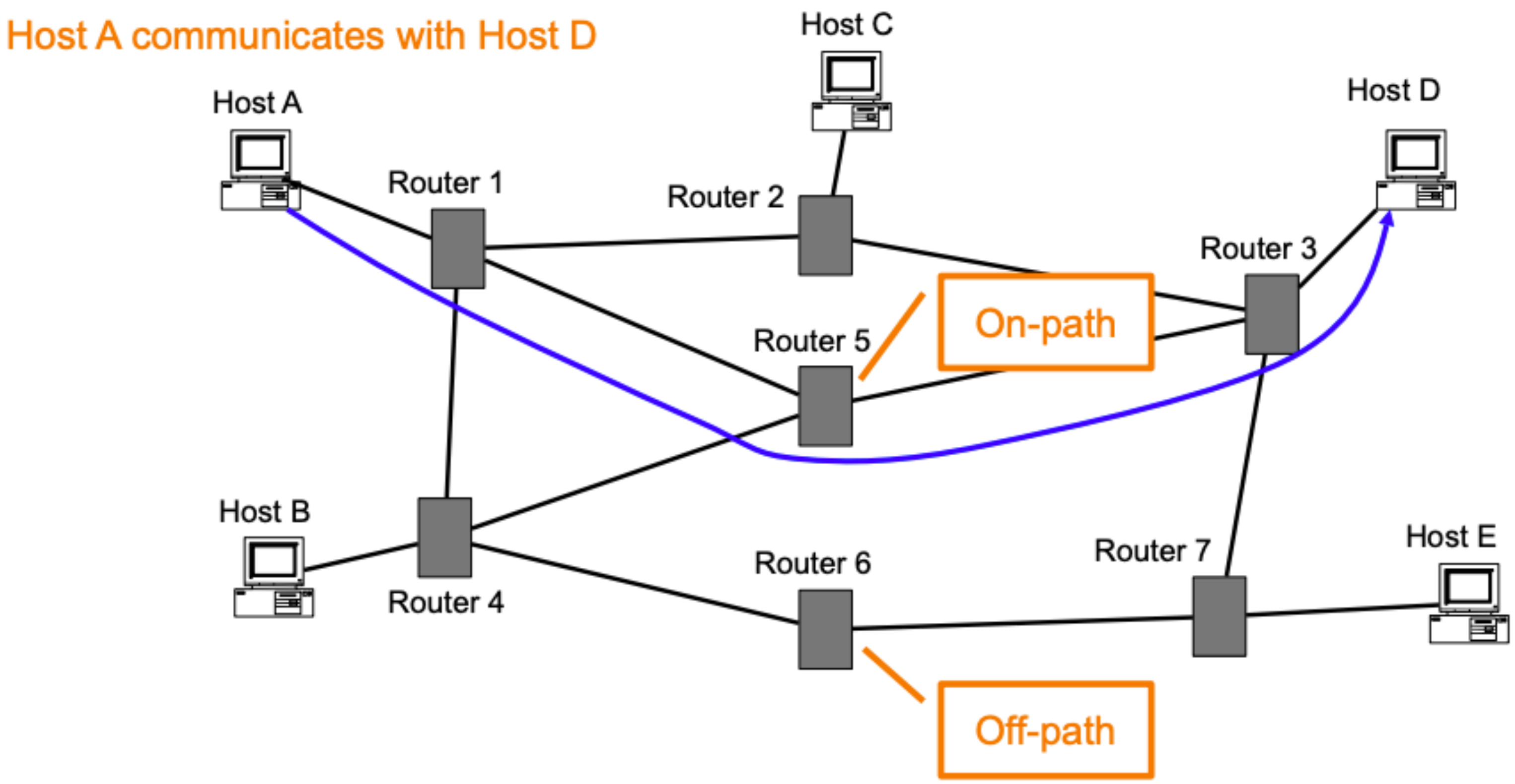
Semantics: What communication means. Actions taken on transmit or receipt of message, or when a timer expires. What assumptions can be made.



Example: What bytes contain each field in a packet header

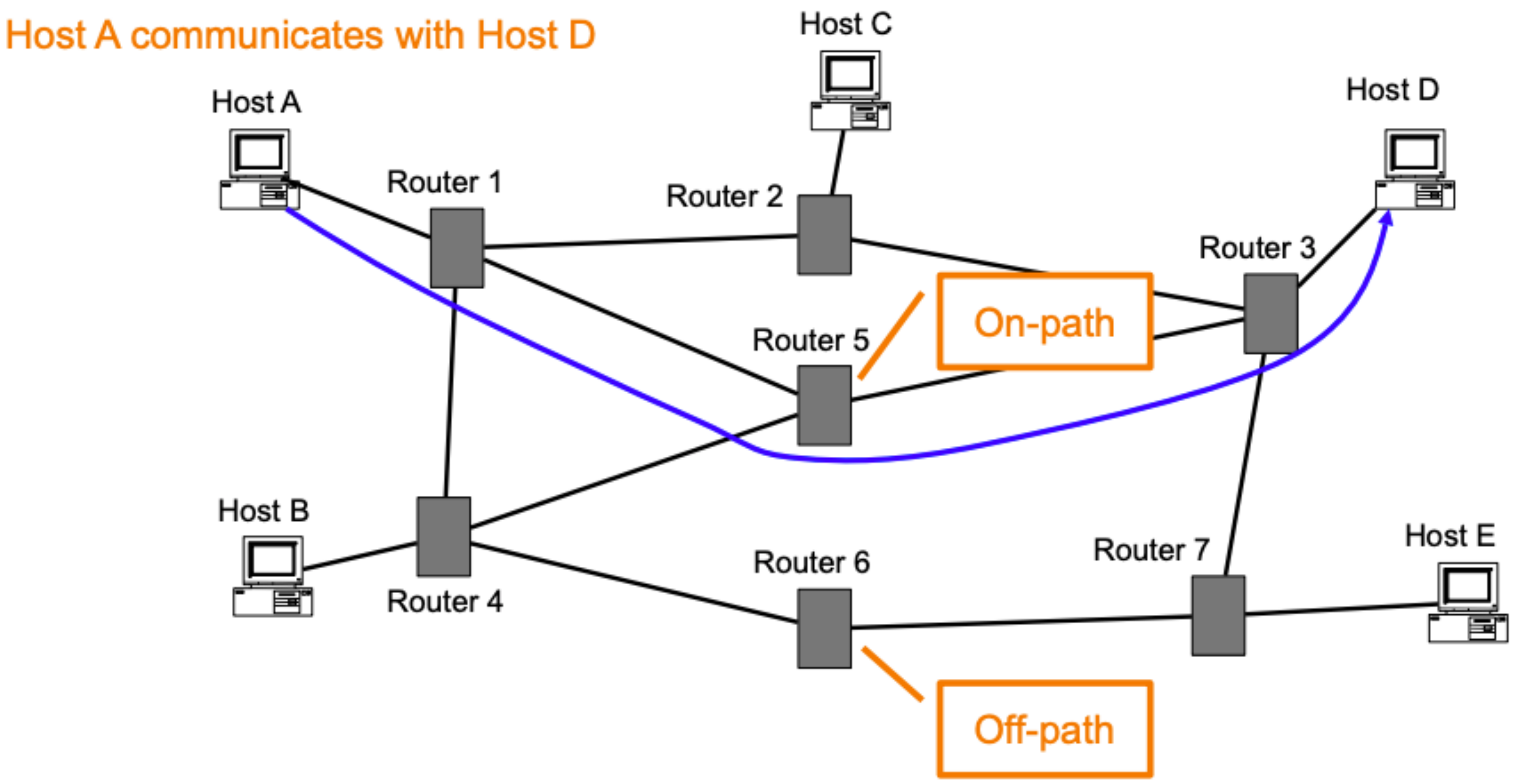
Network Attacks: Classes of Attackers

- MiTM: Can see packets, and can modify and drop packets
- On-path: Can see packets, but can't modify or drop packets
- Off-path: Can't see, modify, or drop packets



Network Attacks: Classes of Attackers

- MiTM: Can see packets, and can modify and drop packets
- On-path: Can see packets, but can’t modify or drop packets
- Off-path: Can’t see, modify, or drop packets



Which type of attacker is more powerful?

A. on-path
B. off-path
C. neither is strictly stronger than the other

Network Attacks: Classes of Attackers

- On-path:
  - Can see packets, but can't modify or drop packet
  - Can see victim's traffic: makes spoofing easy (creating a fake packet)
- Off-path:
  - Can't see, modify, or drop packets? resort to blind spoofing
  - guess/infer header values: sometimes brute-force succeeds!
  - 16 bit header field? only 2^16 possibilities
  - Attacker can spoof translates to attacker has a reasonable chance of success

Protocol Layering

Networks use a stack of protocol layers

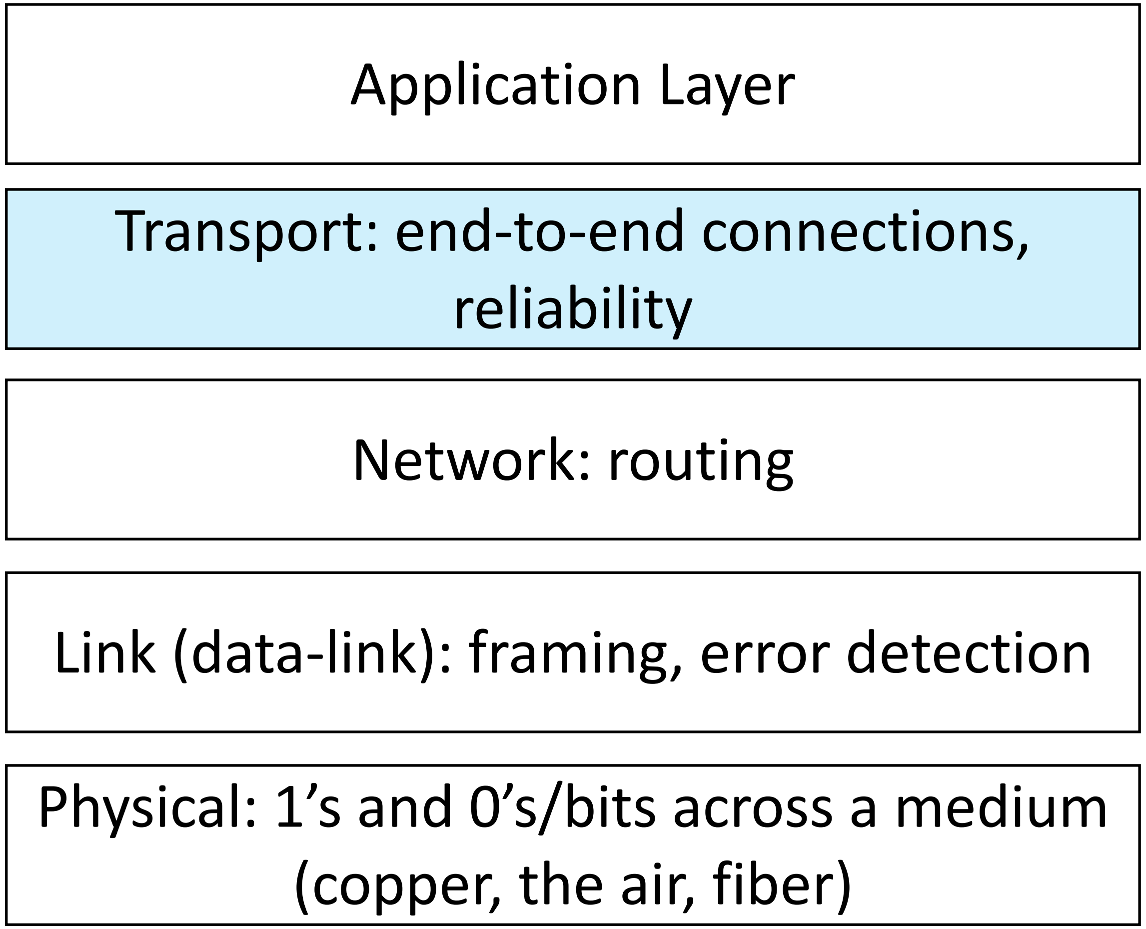
- Each layer has different responsibilities.
- Layers define abstraction boundaries

Lower layers provide services to layers above

- Don't care what higher layers do

Higher layers use services of layers below

- Don't worry about how the layer below works



Transport Layer perspective

Networks use a stack of protocol layers

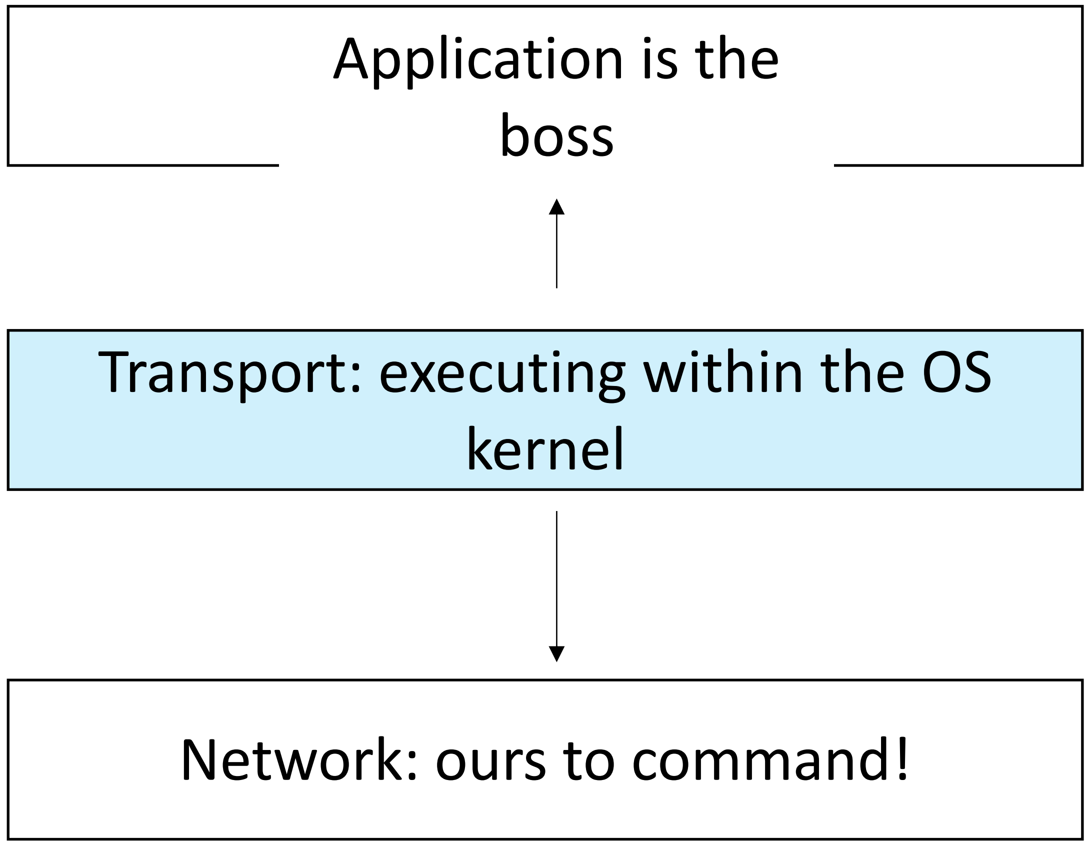
- Each layer has different responsibilities.
- Layers define abstraction boundaries

Lower layers provide services to layers above

- Don't care what higher layers do

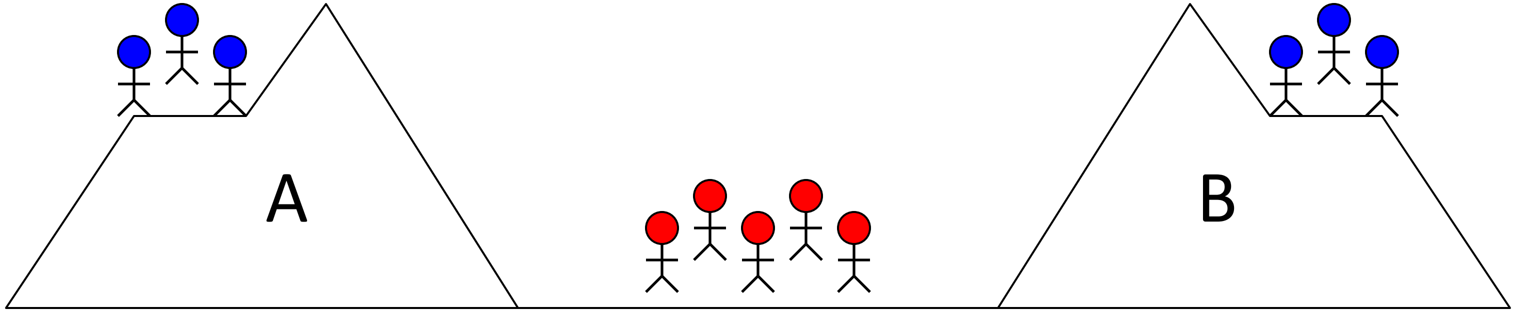
Higher layers use services of layers below

- Don't worry about how the layer below works



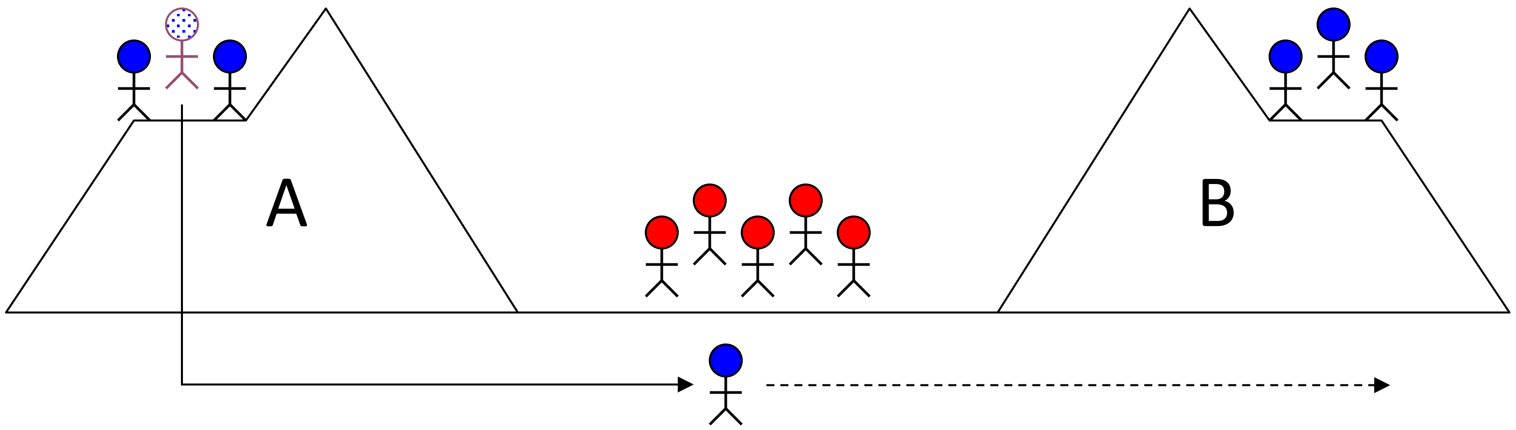
Transmission Control Protocol (TCP)

The Two Generals Problem



- Two army divisions (blue) surround enemy (red)
  - Each division led by a general
  - Both must agree when to simultaneously attack
  - If either side attacks alone, defeat
- Generals can only communicate via messengers
  - Messengers may get captured (unreliable channel)

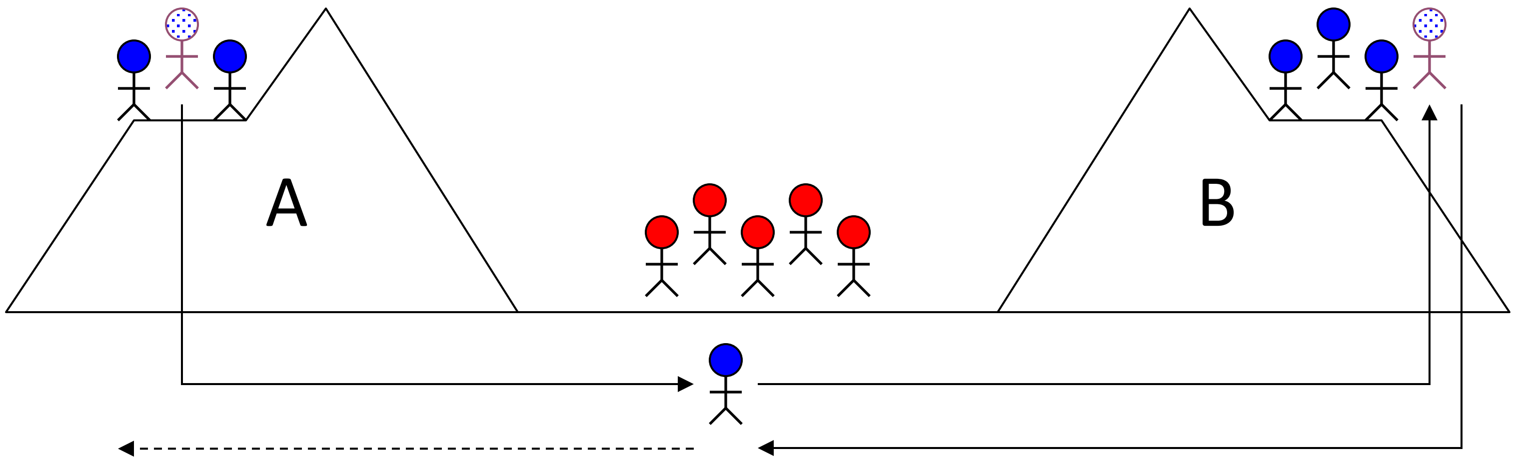
The Two Generals Problem



How do we coordinate?

- Send messenger: “Attack at dawn”
- What if messenger doesn’t make it?

The Two Generals Problem



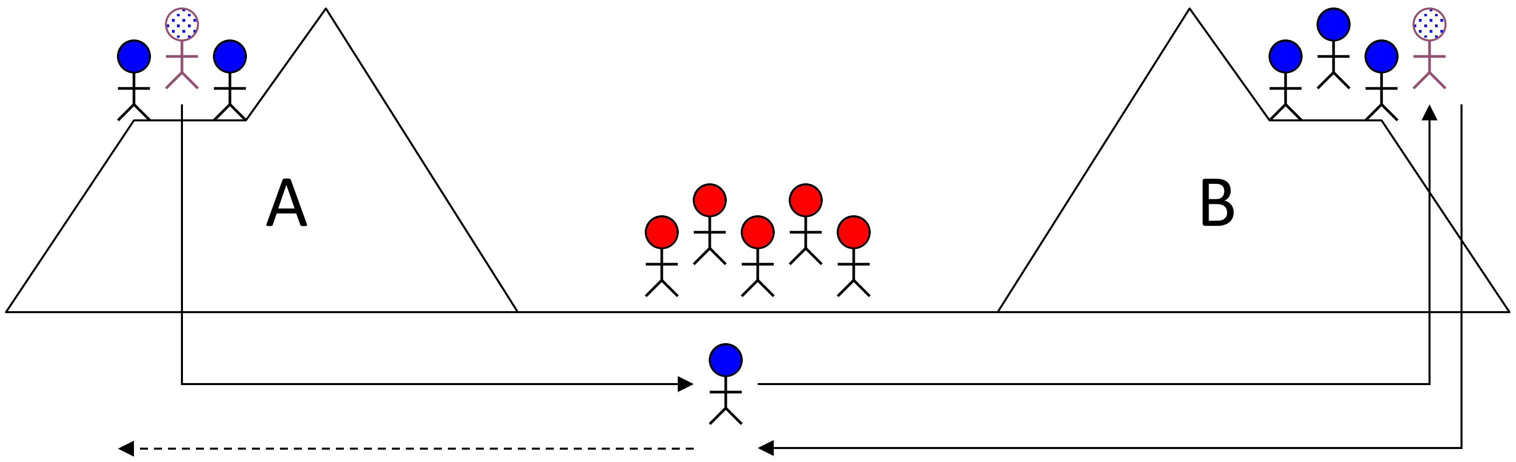
How can we be sure the messenger made it?

- Send acknowledgment: “I delivered message”

In the “two generals problem”, can the two armies reliably coordinate their attack? (using what we just discussed)

- A. Yes (explain how)
- B. No (explain why not)

The Two Generals Problem



Result

- Can't create perfect channel out of faulty one
- Can only increase probability of success

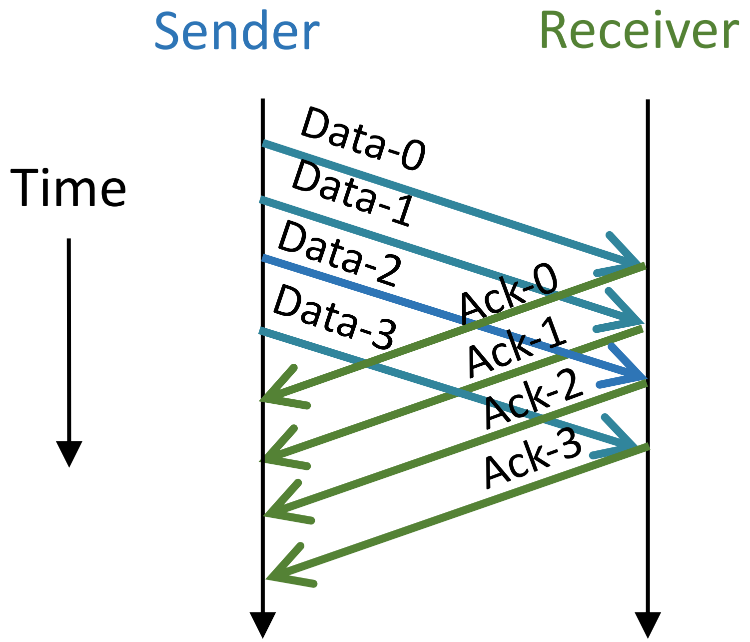
Designing reliability over an unreliable link. What can go wrong?

A. Packets can be dropped
B. Packets can arrive out or order
C. Acknowledgements can arrive out of order
D. All of the above
E. There are more issues….

Designing reliability over an unreliable link. What can go wrong?

- Problem: IP packets have a limited size. To send longer messages, we have to manually break messages into packets
  - When sending packets: TCP will automatically split up messages
  - When receiving packets: TCP will automatically reassemble the packets
  - Now the user doesn't need to manually split up messages!
- Problem: Packets can arrive out of order
  - When sending packets: TCP labels each byte of the message with increasing numbers
  - When receiving packets: TCP can use the numbers to rearrange bytes in the correct order
- Problem: Packets can be dropped
  - When receiving packets: TCP sends an extra message acknowledging that a packet has been received
  - When sending packets: If the acknowledgement doesn't arrive, re-send the packet

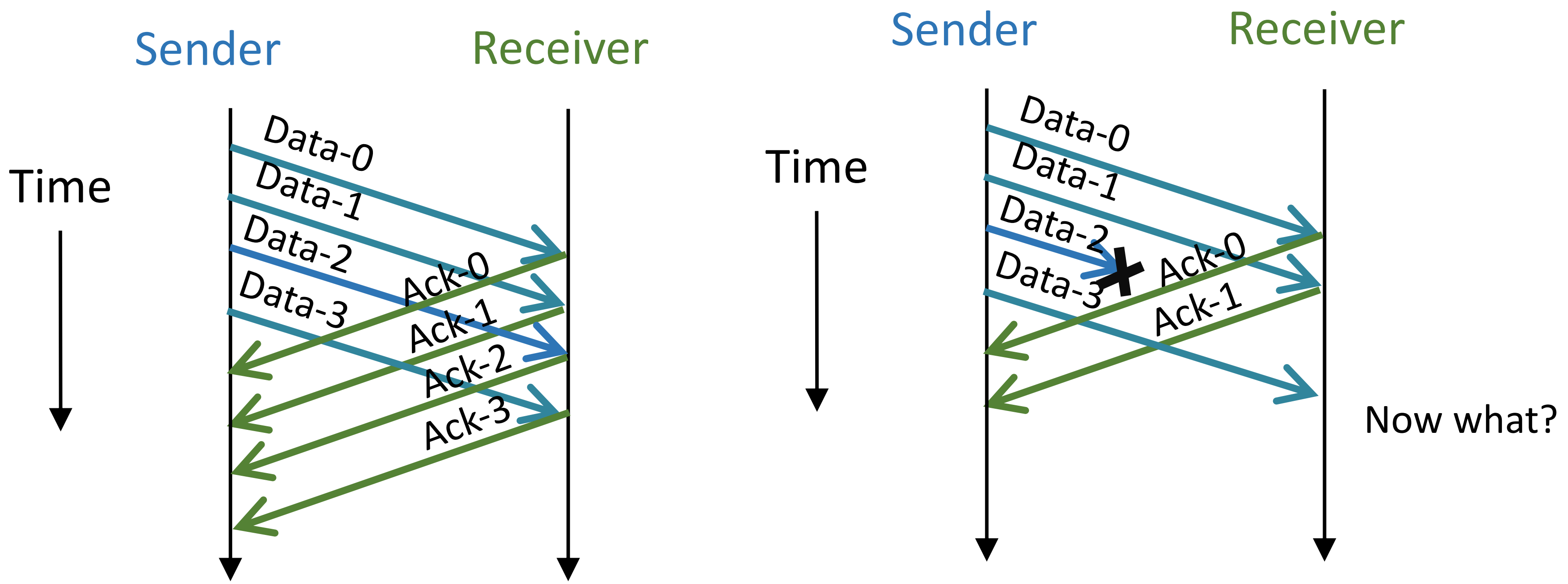
Pipelined Transmission



<u>Keep multiple segments "in flight"</u>

- Allows sender to make efficient use of the link
- Sequence numbers ensure receiver can distinguish segments

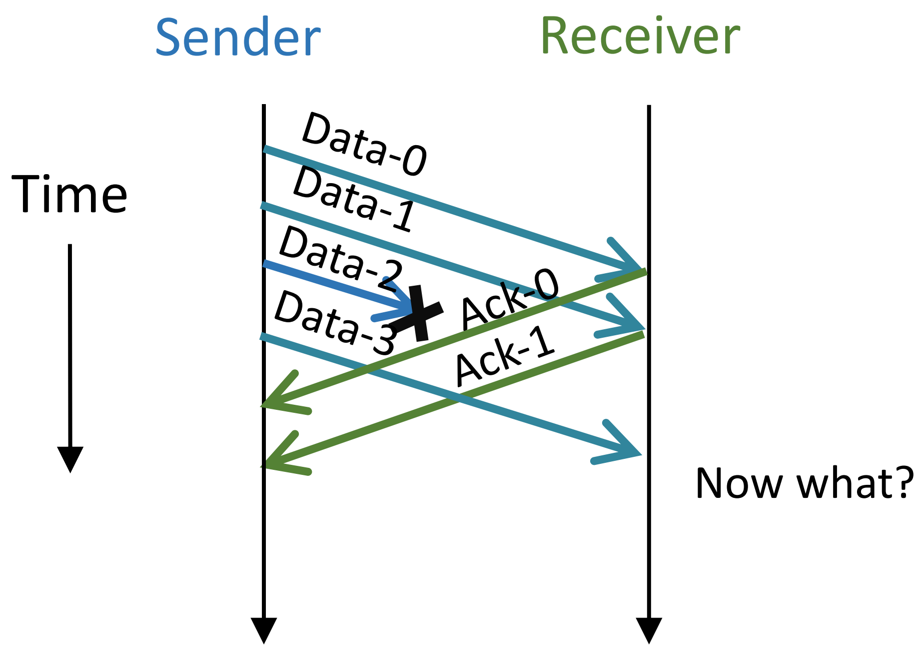
Pipelined Transmission



<u>Keep multiple segments “in flight”</u>

– Allows sender to make efficient use of the link
– Sequence numbers ensure receiver can distinguish segments

What should the sender do here?

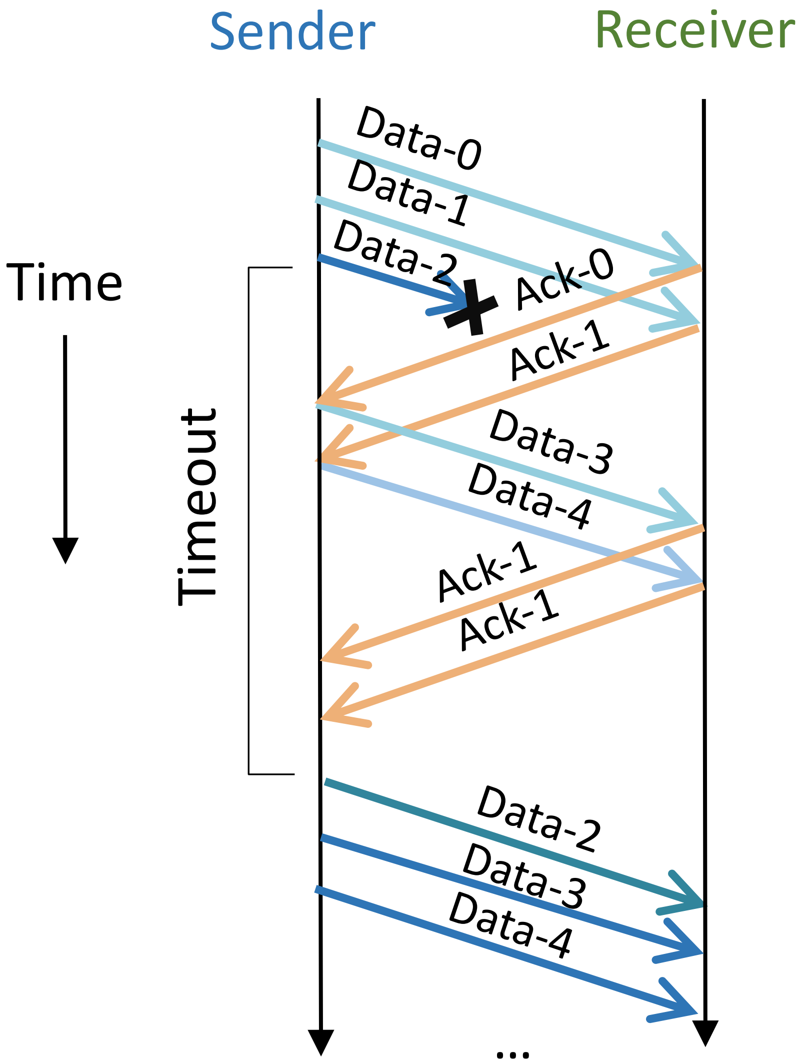


What information does the sender need to make that decision?

What is required by either party to keep track?

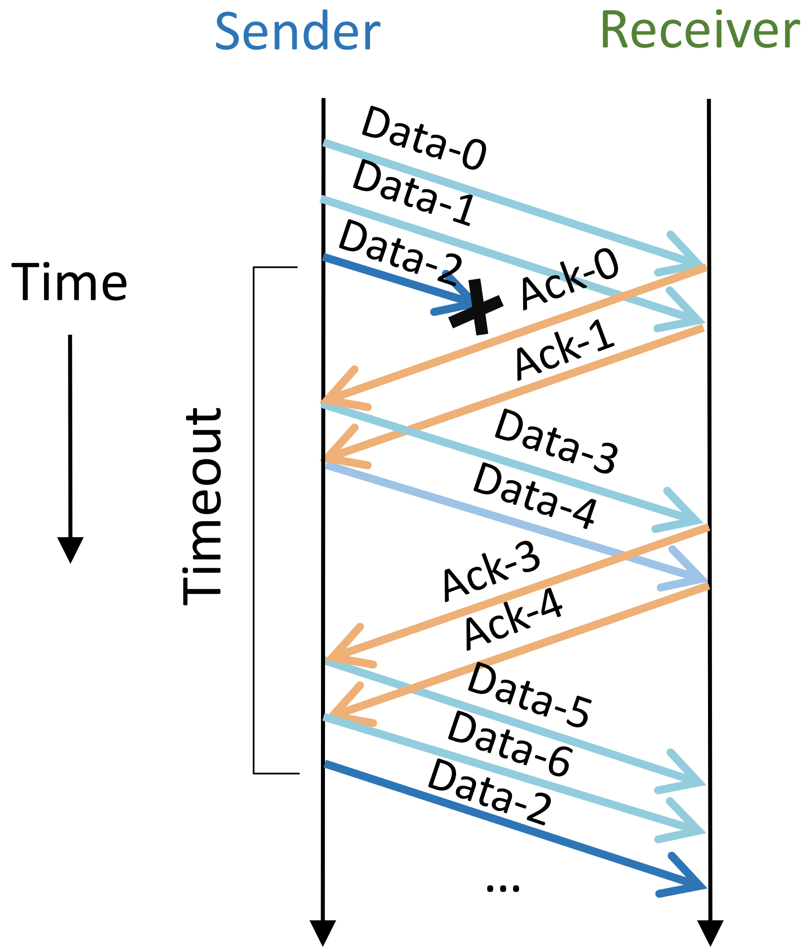
- Start sending all data again from 0.
- Start sending all data again from 2.
- Resend just 2, then continue with 4 afterwards.

Go-Back-N



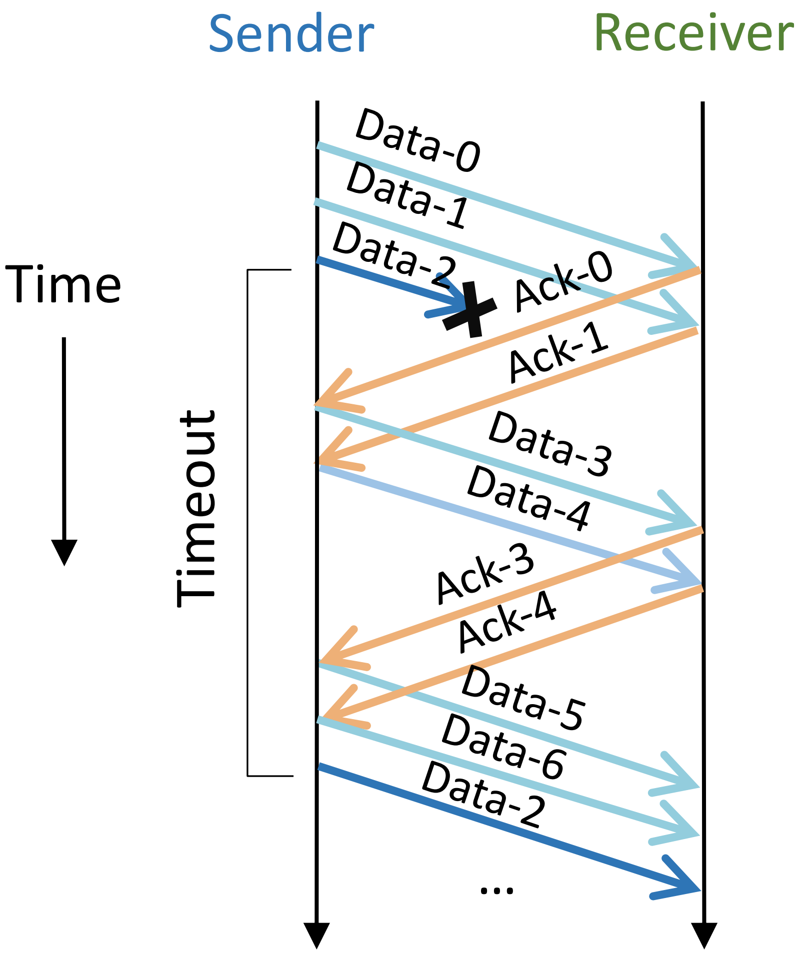
- Retransmit from point of loss
    - Segments between loss event and retransmission are ignored
    - “Go-back-N” if a timeout event occurs

Selective Repeat

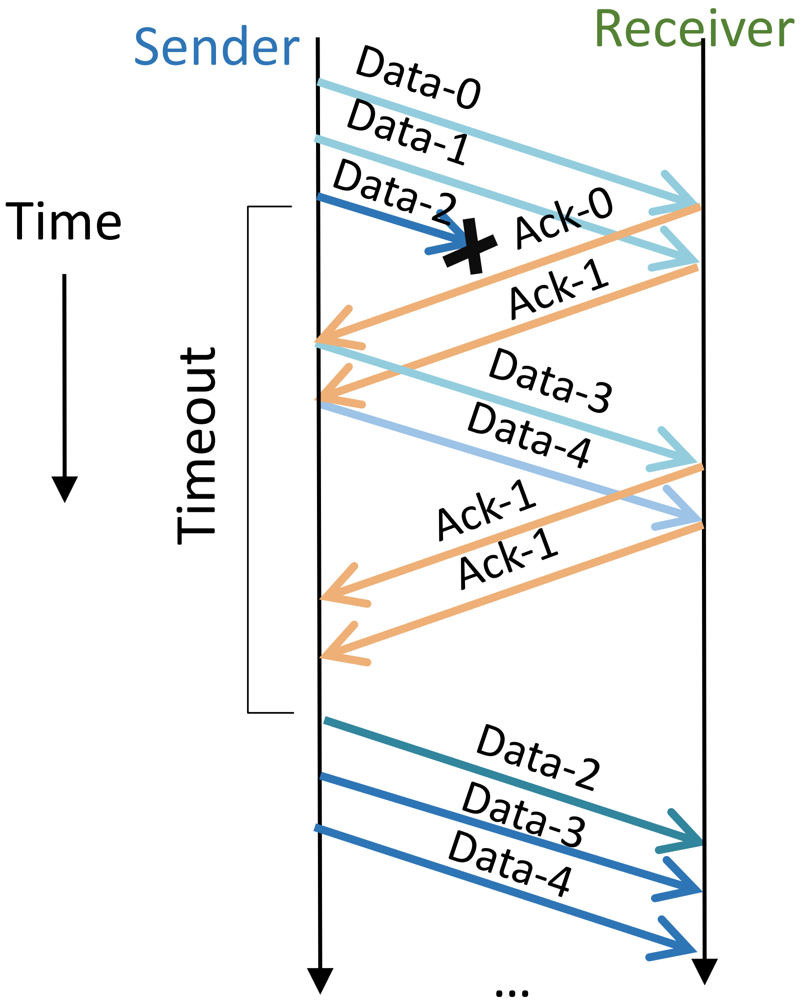


- Receiver ACKs each segment individually (not cumulative)
- Sender only resends those not ACKed

What should the sender do here?



Selective Repeat: Sender only resends those packets not ACKed



Go-Back-N: Retransmit from point of loss

What information does the sender need to make that decision?

What is required by either party to keep track?

A. Go-Back-N less work for the receiver

B. Selective Repeat less work for the network.

C. Some other combination, both are horrible.

Transmission Control Protocol (TCP)

- Provides a byte stream abstraction
  - Bytes go in one end of the stream at the source and come out at the other end at the destination
  - TCP automatically breaks streams into **segments**,
- Provides ordering
  - Segments contain sequence numbers, so the destination can reassemble the stream in order
- Provides reliability
  - The destination sends acknowledgements (ACKs) for each sequence number received
  - If the source doesn’t receive the ACK, the source sends the packet again
- Provides ports
  - Multiple services can share the same IP address by using different ports

Ports: An Analogy

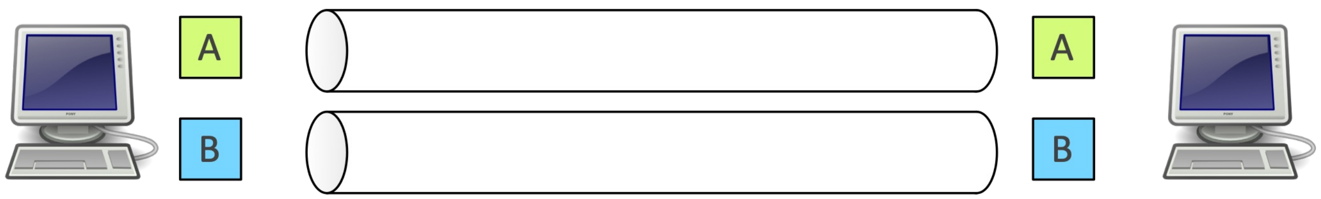
- Alice is pen pals with Bob. Alice's roommate, Carol, is also pen pals with Bob
- Bob's replies are addressed to the same global (IP) address
  - How can we tell which letters are for Alice and which are for Bob?
- Solution: Add a room number (port number) inside the letter
  - In private homes, usually a port number is meaningless
  - But, in public offices (servers), like Cory Hall, the port numbers are constant and known

Ports

Each application on a host is identified by a *port number*

TCP connection established between port *A* on host *X* to port *B* on host *Y* Ports are 1–65535 (16 bits)

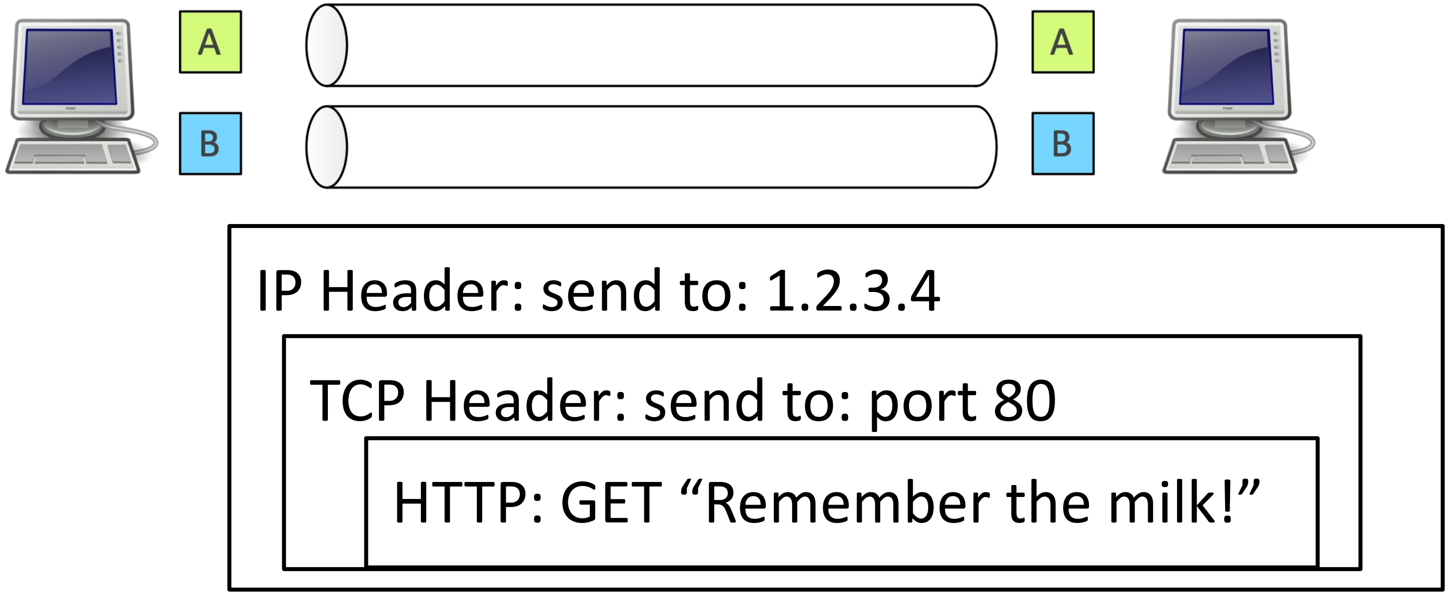
Some destination port numbers used for specific applications by convention



Ports

Ports help us distinguish between different applications on the same computer or server

- On private computers, port numbers can be random
- On public servers, port numbers should be constant and well-known (so users can access the right port)



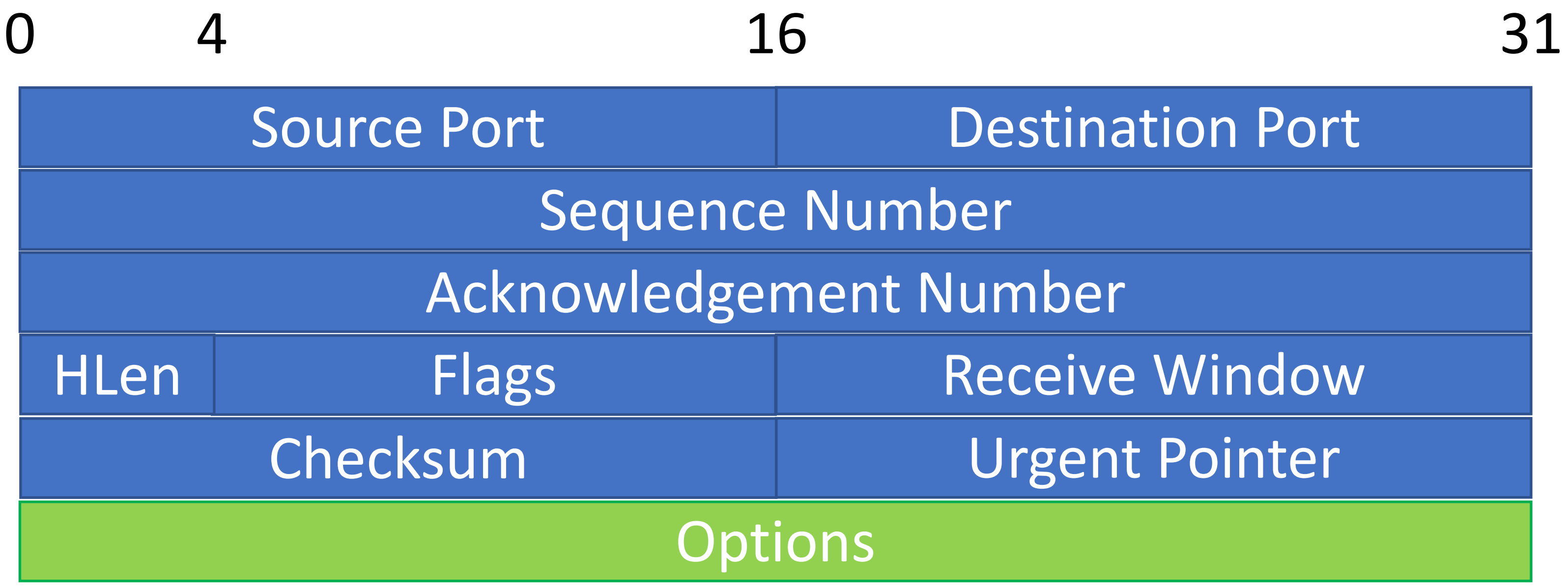
Common Ports

Port	Application
80	HTTP (Web)
443	HTTPS (E2E encrypted Web)
25	SMTP
22	SSH
23	Telnet
53	DNS

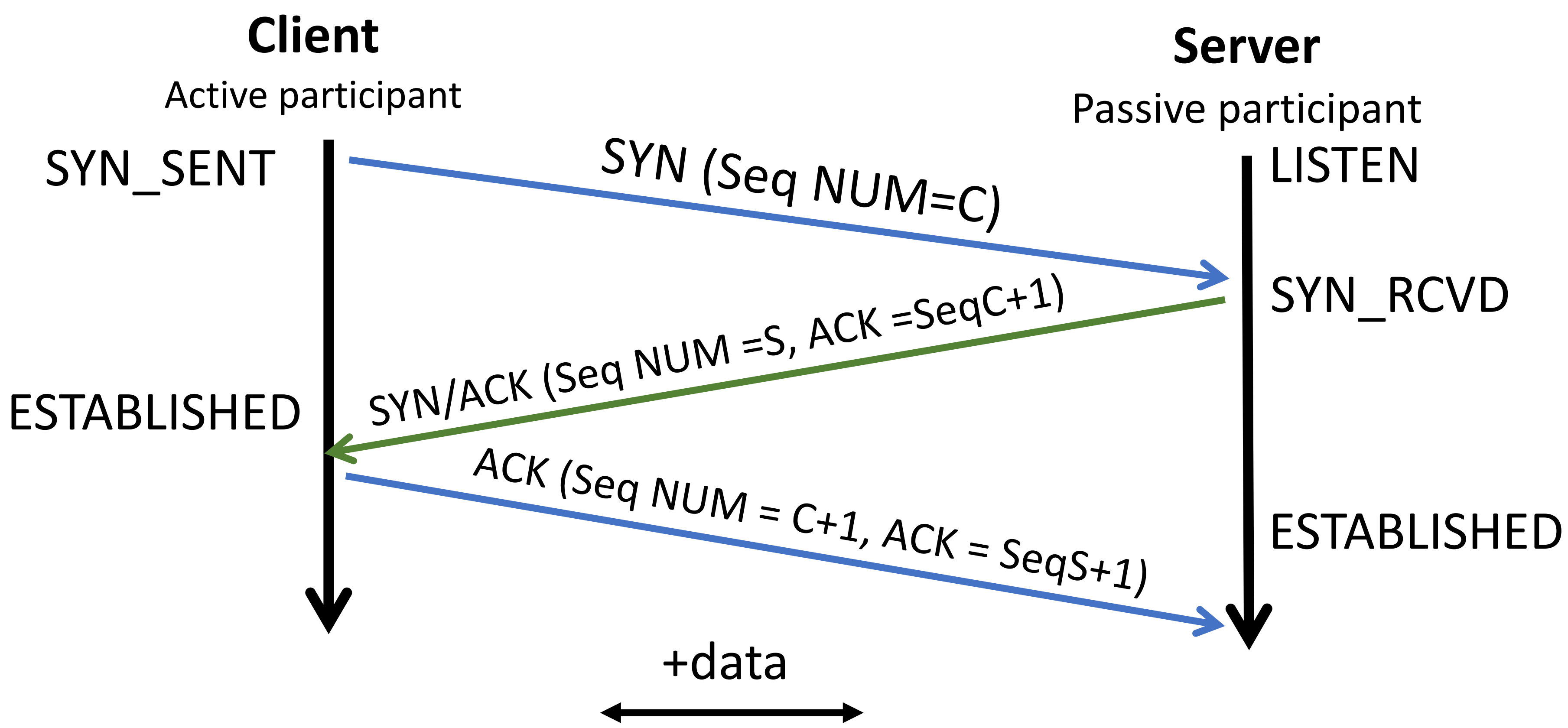
Transmission Control Protocol

Reliable, in-order, bi-directional byte streams

- Port numbers for demultiplexing
- Flow control
- Congestion control, approximate fairness

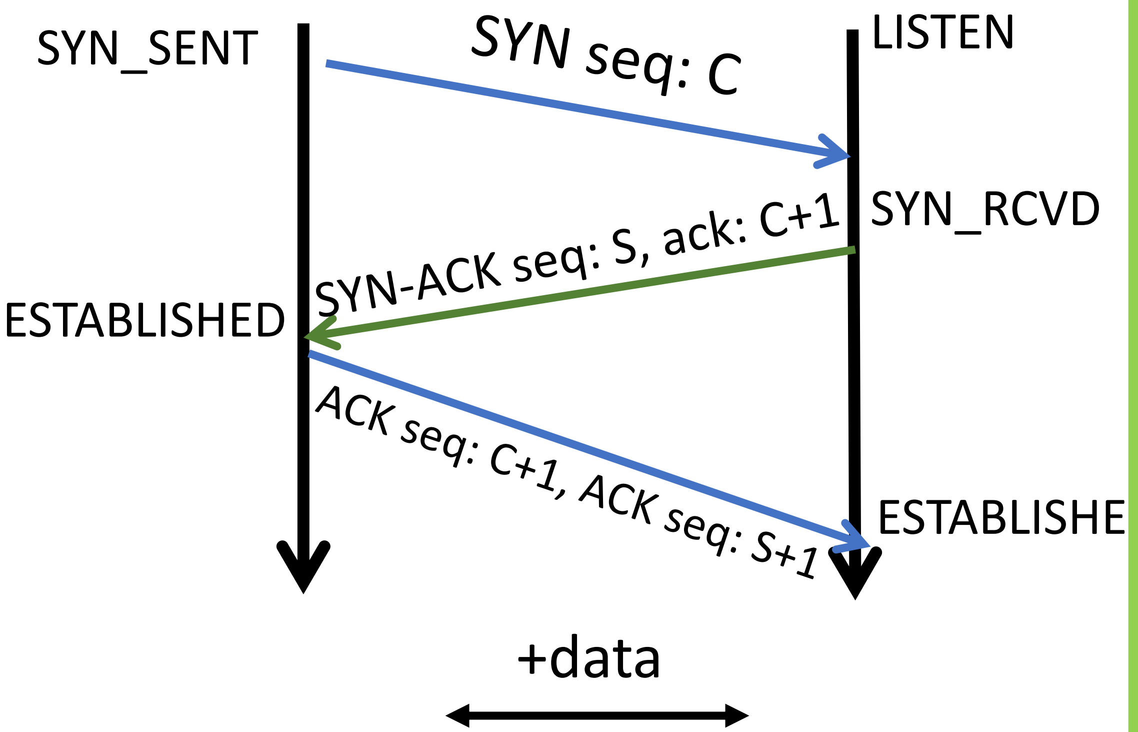


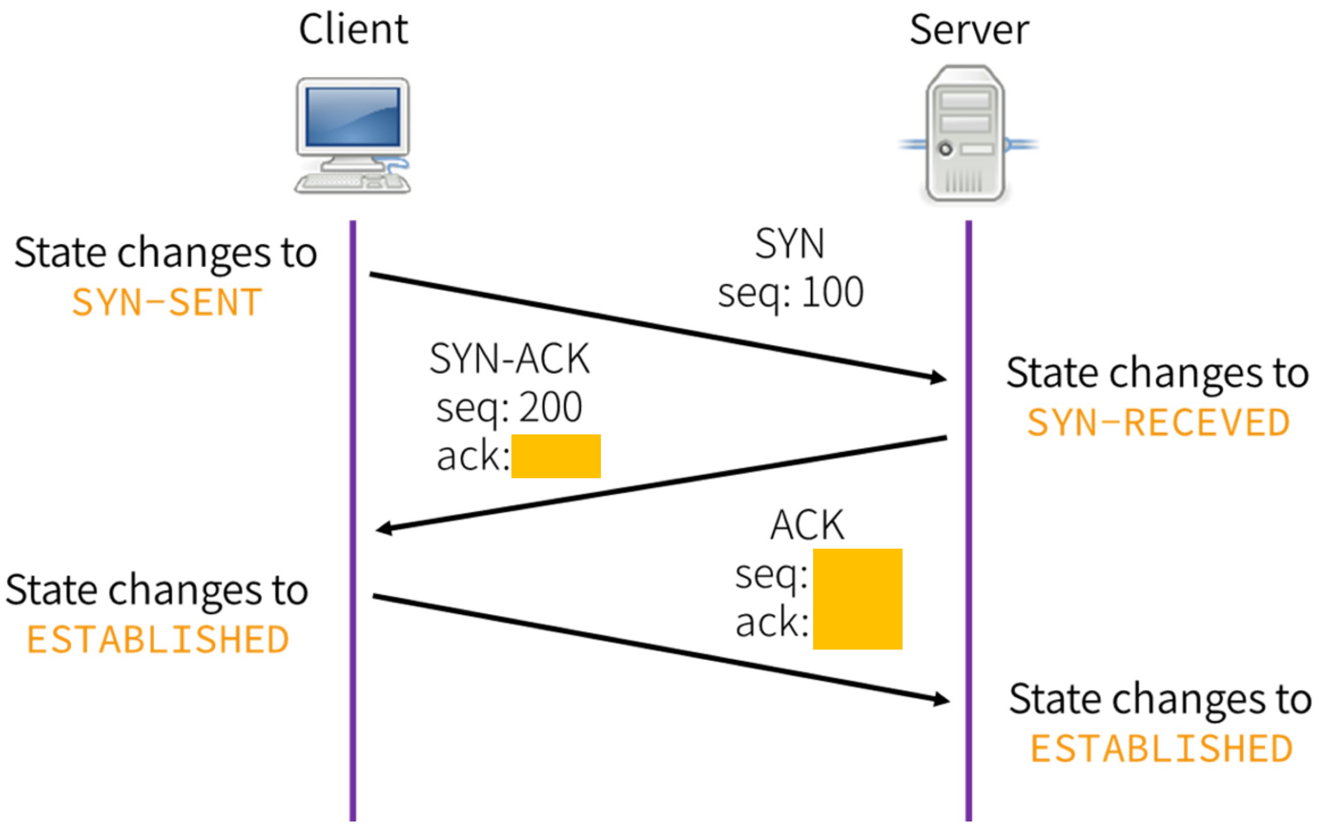
Three Way Handshake



- Each side:
  - Notifies the other of starting sequence number
  - ACKs the other side's starting sequence number

TCP Three Way Handshake





A. SYN-ACK: ack:200, ACK: seq: 300, ack: 400

B. SYN-ACK: ack:201, ACK: seq: 301, ack: 401

C. SYN-ACK: ack:101, ACK: seq: 101, ack: 201

D. SYN-ACK: ack:101, ACK: seq: 201, ack: 101

How should we choose the initial sequence number?

A. Start from zero

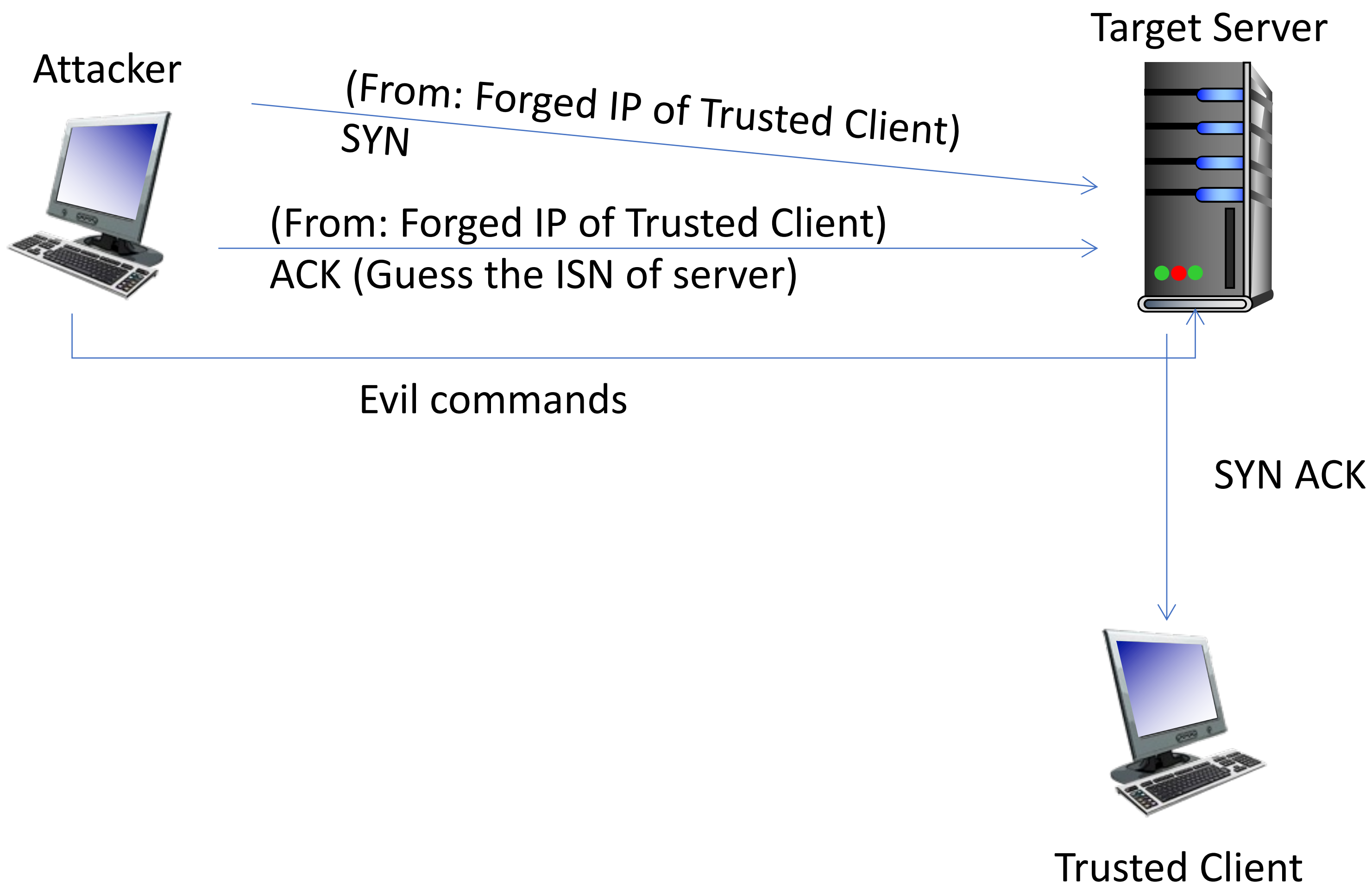
B. Start from one

C. Start from a random number

D. Start from some other value (such as…?)

What can go wrong with sequence numbers?
-How they're chosen?
-In the course of using them?

TCP Connection Spoofing: Sequence Prediction Attack

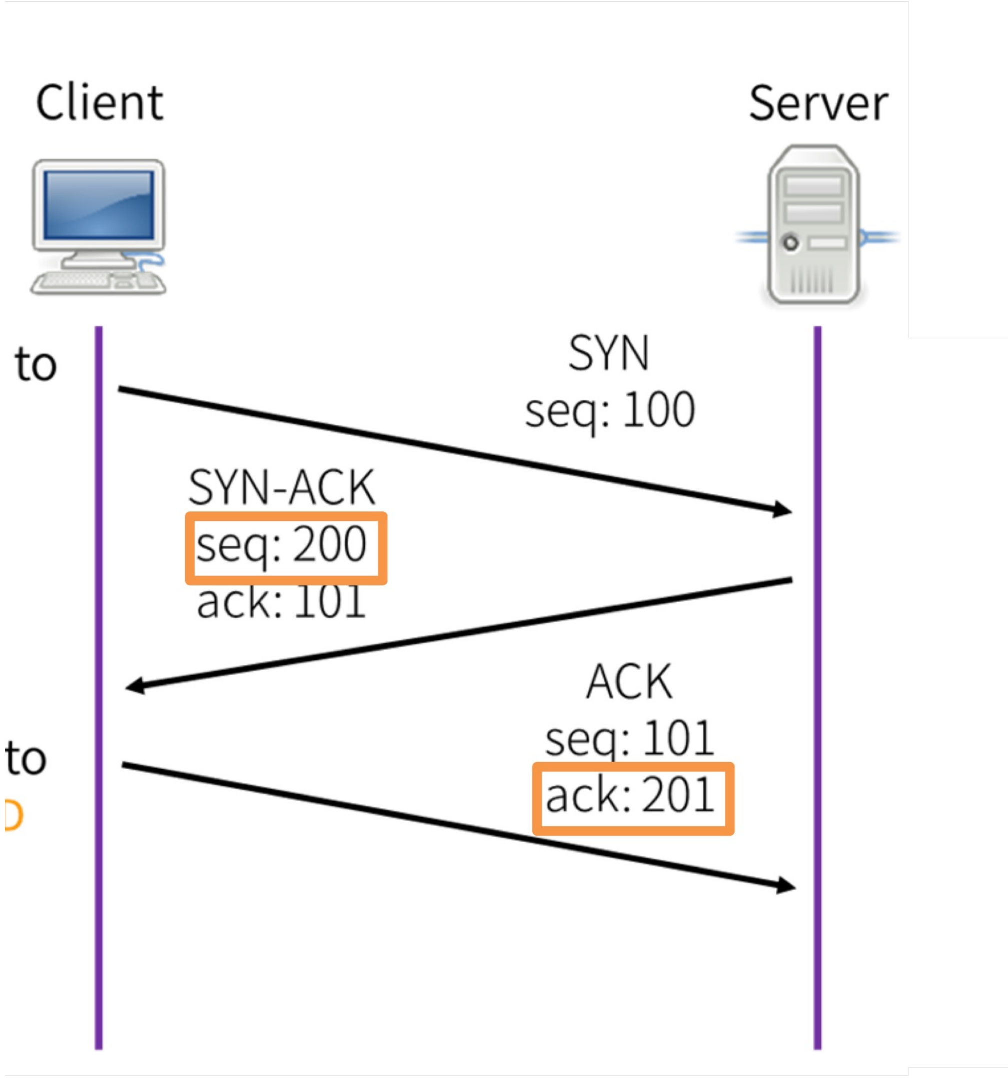


TCP Connection Spoofing

Can we impersonate another host when *initiating* a connection?

Off-path attacker can send initial SYN to server …
… but cannot complete three-way handshake without seeing the server's sequence number

1 in 2^{32} chance to guess right if initial sequence number chosen uniformly at random



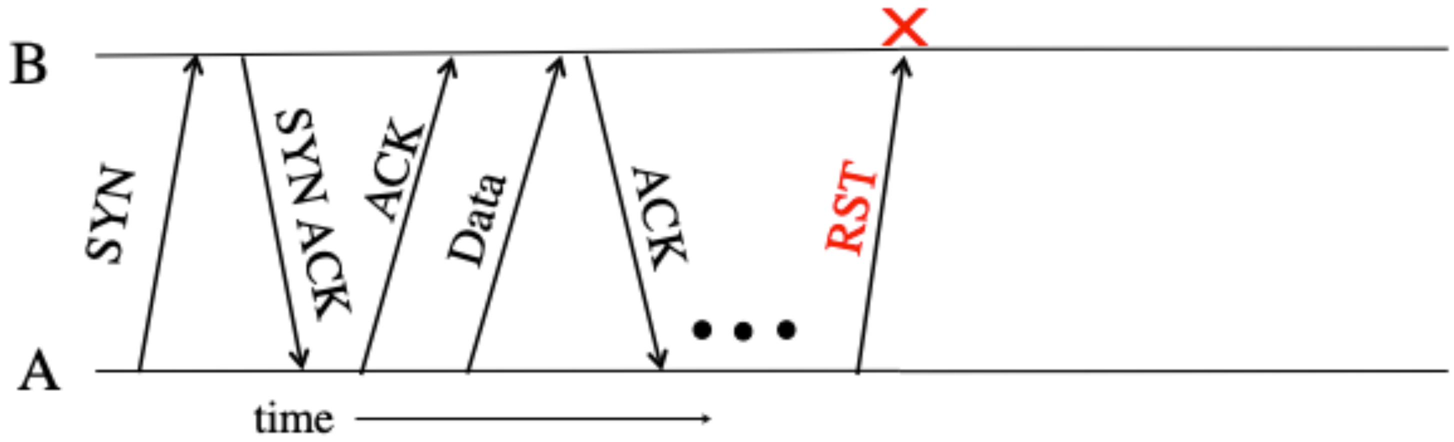
TCP Flags: Ending/Aborting a Connection

- ACK
  - Indicator that the user is acknowledging the receipt of something (in the ack number)
  - Pretty much always set except the very first packet
- SYN
  - Indicator of the beginning of the connection
- FIN
  - One way to end the connection
  - Requires an acknowledgement
  - No longer sending packets, but will continue to receive
- RST
  - One way to end a connection
  - Does not require an acknowledgement
  - No longer sending or receiving packets

TCP: Ending/Aborting a Connection

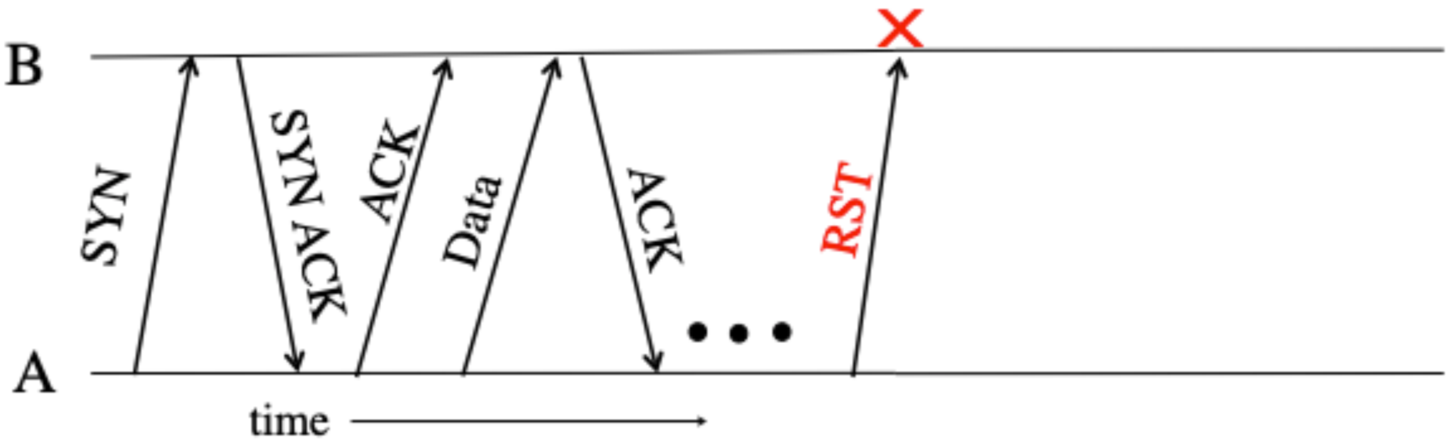
- To **end** a connection, one side sends a packet with the FIN (finish) flag set, which should then be acknowledged
  - This means “I will no longer be sending any more packets, but I will continue to receive packets”
  - Once the other side is no longer sending packets, it sends a packet with the FIN flag set
- To **abort** a connection, one side sends a packet with the RST (reset) flag set
  - This means “I will no longer be sending nor receiving packets on this connection”
  - RST packets are not acknowledged since they usually mean that something went wrong

TCP RST Injection



- If A sends a TCP packet with RST flag to B and sequence number fits, connection is terminated
  - Unilateral, and takes effect immediately

TCP RST Injection Attack



Who can do RST injection?

A. off-path attacker

B. on-path attacker

C. man-in-the-middle

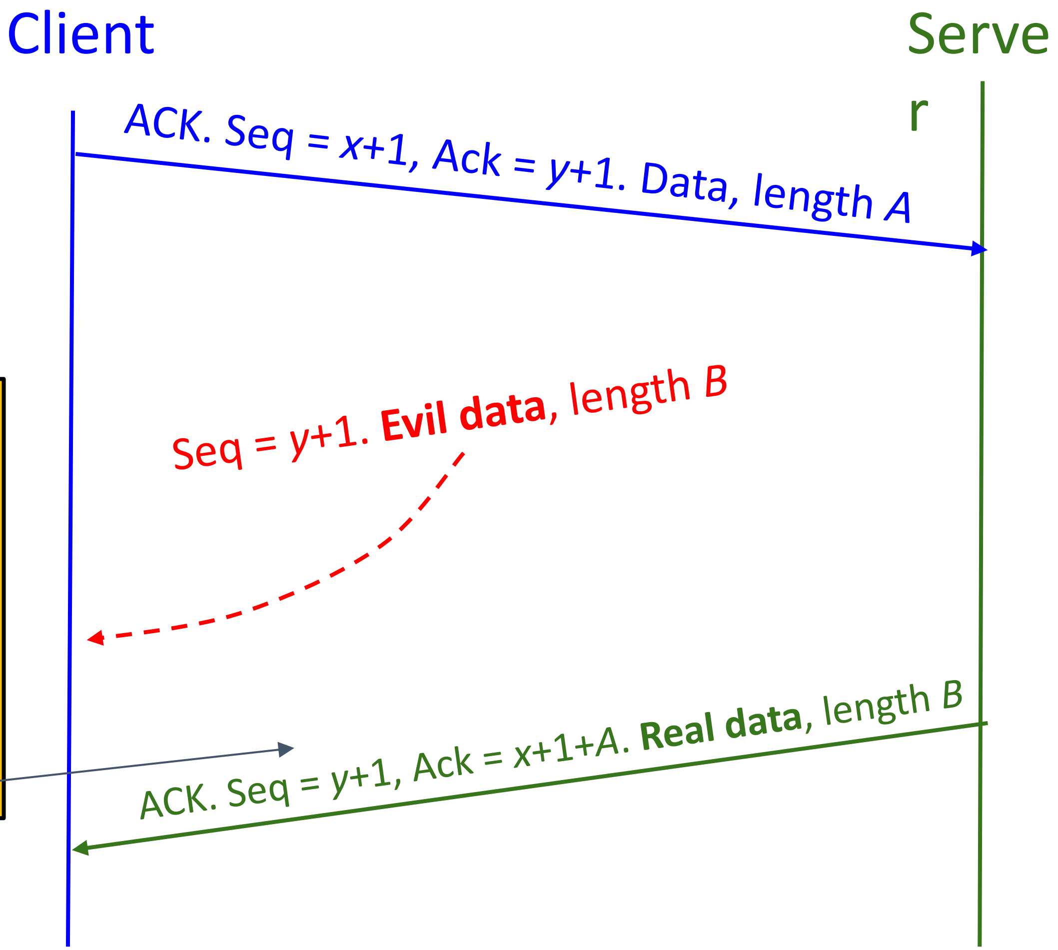
The attacker can inject RST packets and block connection

TCP clients must respect RST packets and stop all communication

Who uses this? Historically..

- China: The Great Firewall does this to TCP requests
- A long time ago: Comcast, to block BitTorrent uploads
- Some intrusion detection systems: To hopefully mitigate an attack in progress

TCP Data Injection: Tampering with an existing session to modify or inject data into a connection

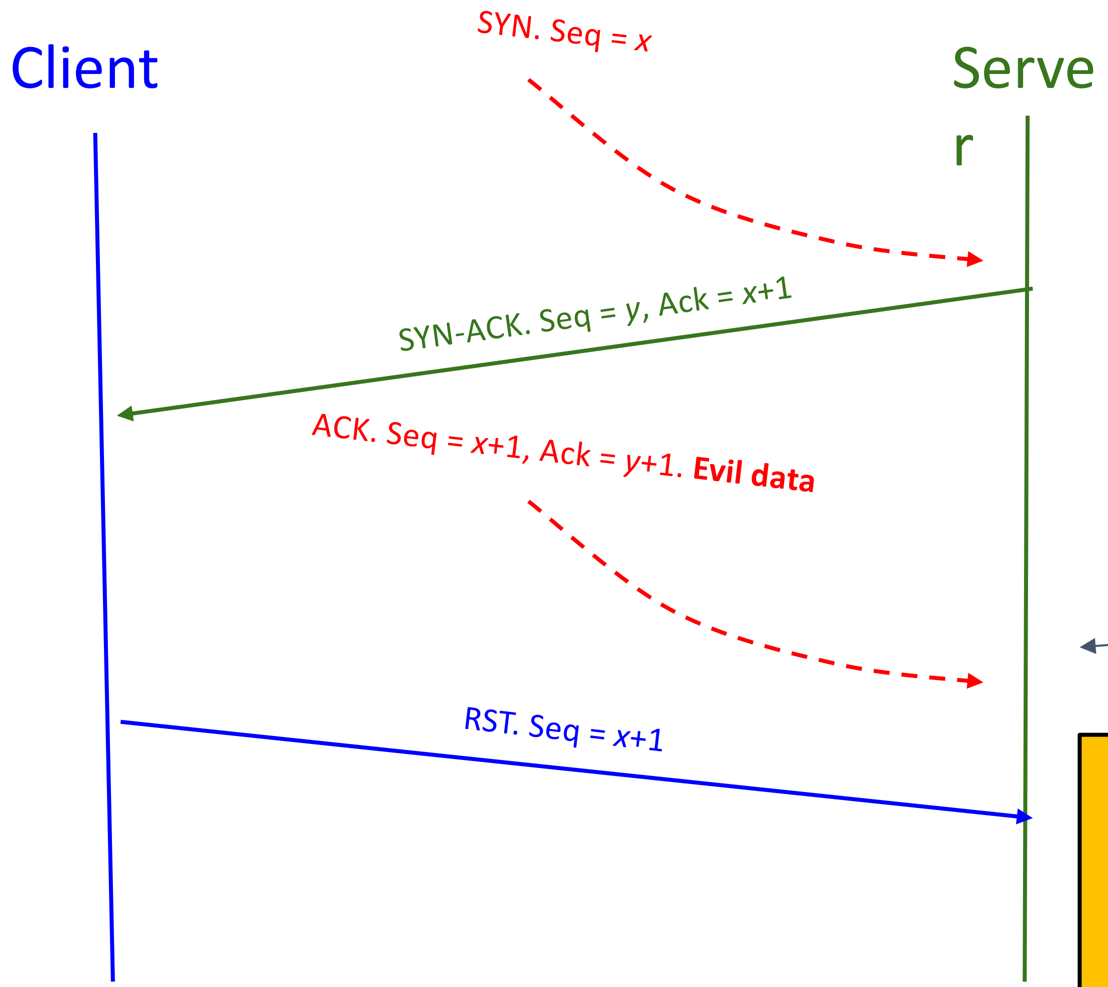


This packet will be ignored by the client since the client already processed the malicious packet!

TCP Attacks

- **TCP hijacking**: Tampering with an existing session to modify or inject data into a connection
  - **Data injection**: Spoofing packets to inject malicious data into a connection
    - Need to know: The sender's sequence number
  - Easy for MITM and on-path attackers, but off-path attackers must guess 32-bit sequence number (called **blind injection/hijacking**, considered difficult)
  - For on-path attackers, this becomes a race condition since they must beat the server's legitimate response

TCP Spoofing



An on-path attacker must send the evil data before the server receives the

A MITM attack could just drop the client's packets, however

TCP Provides..

A. Confidentiality
B. Availability
C. Integrity
D. None of the above

TCP Provides..

- TCP provides no confidentiality or integrity
  - Instead, we rely on higher layers (like TLS, more on this next time) to prevent those kind of attacks
- Defense against off-path attackers rely on choosing random sequence numbers
  - Bad randomness can lead to trivial off-path attacks: TCP sequence numbers used to be based on the system clock!

TLS: transport layer security

SSL/TLS

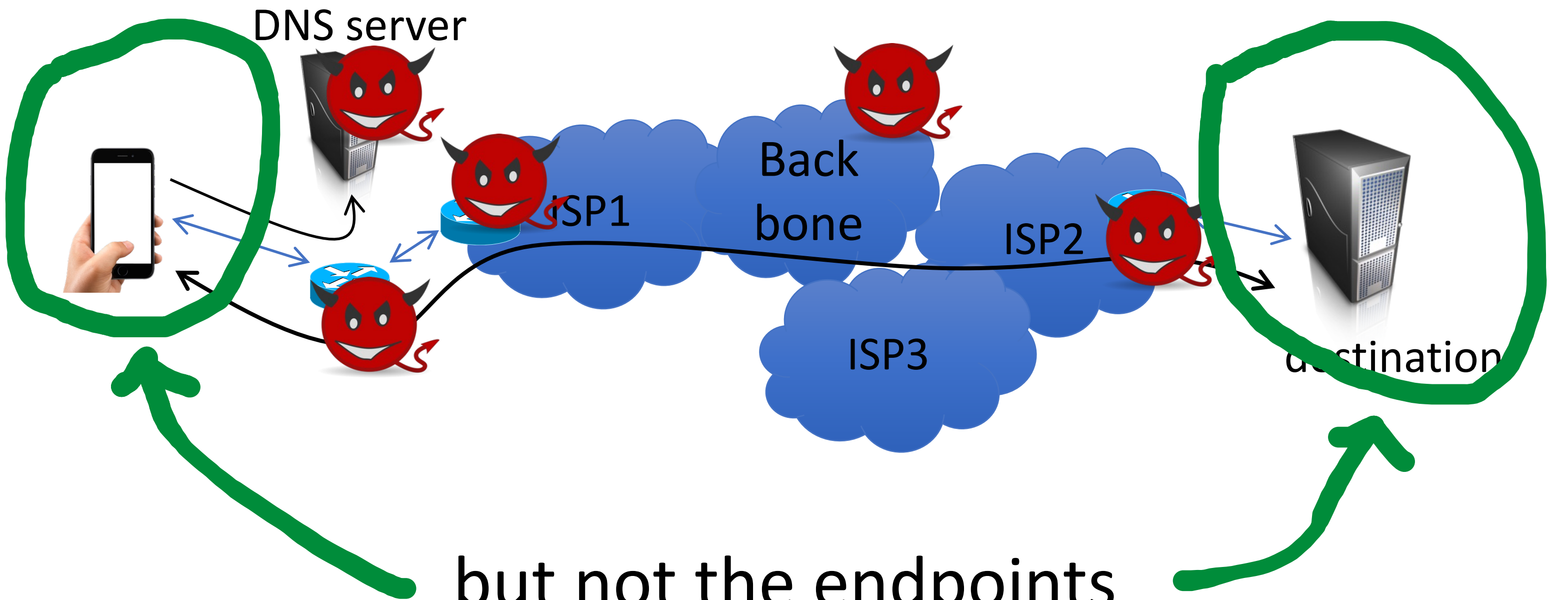
- Secure Sockets Layer and Transport Layer Security protocols
    - Same protocol design, different cryptographic algorithms
- The de facto standard for Internet security
    - “The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications”
- Deployed in every Web browser (HTTPS); also mobile applications, payment systems, VoIP, many distributed systems, etc.

SSL / TLS Guarantees

- End-to-end secure communications in the presence of a network attacker
  - Attacker completely 0wns the network: controls Wi-Fi, DNS, routers, his own websites, can listen to any packet, modify packets in transit, inject his own packets into the network
- Scenario: you are reading your email from an Internet café connected via a r00ted Wi-Fi access point to a dodgy ISP in a hostile authoritarian country

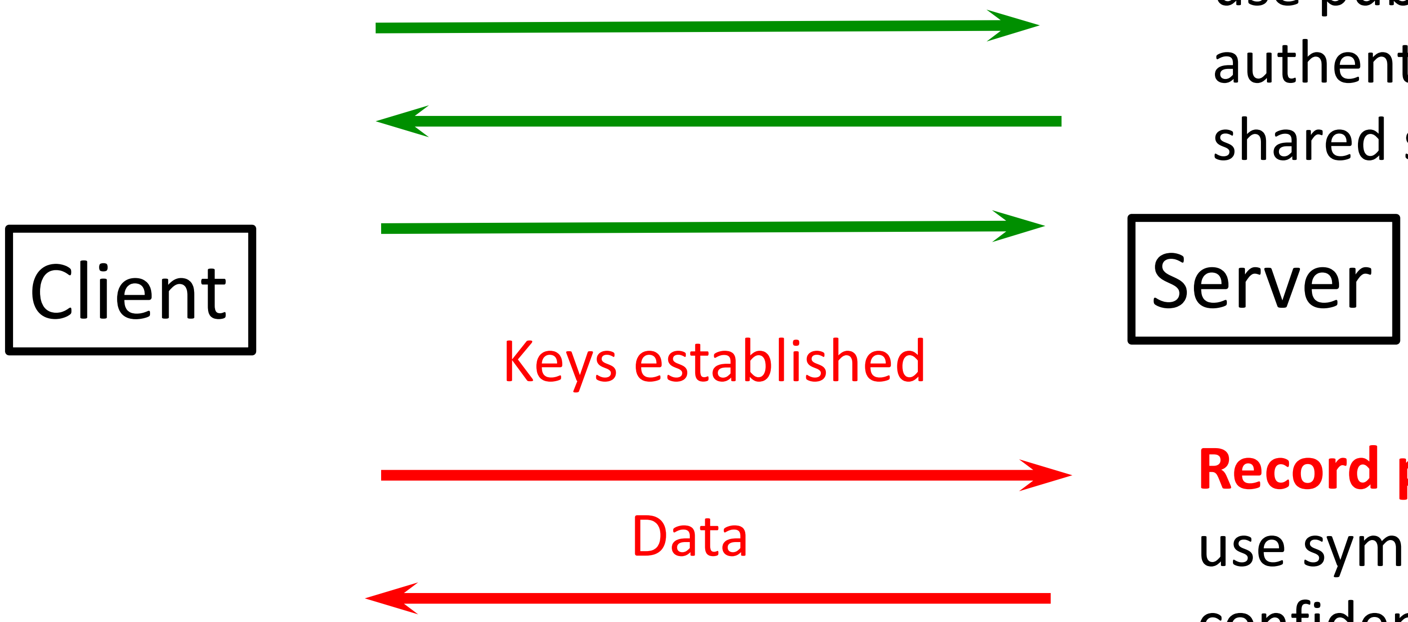
TLS Threat Model

Remember TCP/IP, DNS attacks?
TLS is all that stands between us and oblivion…



but not the endpoints

Establishing a Secure Channel

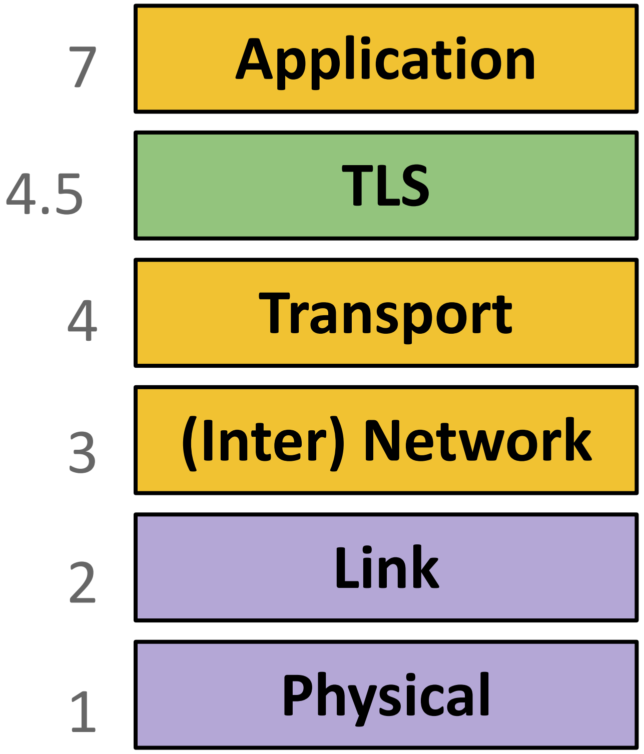


Handshake protocol:
use public-key cryptography to authenticate each other, establish shared symmetric keys

Record protocol:
use symmetric keys to protect confidentiality, integrity, authenticity of exchanged data

Transport Layer Security

- TLS (Transport Layer Security): A protocol for creating a secure communication channel over the Internet
  - Replaces SSL (Secure Sockets Layer), which is an older version of the protocol
- TLS is built on top of TCP
  - Relies upon: Byte stream abstraction between the client and the server
  - Provides: Byte stream abstraction between the client and the server
    - The abstraction appears the same to the end client, but TLS provides confidentiality and integrity!

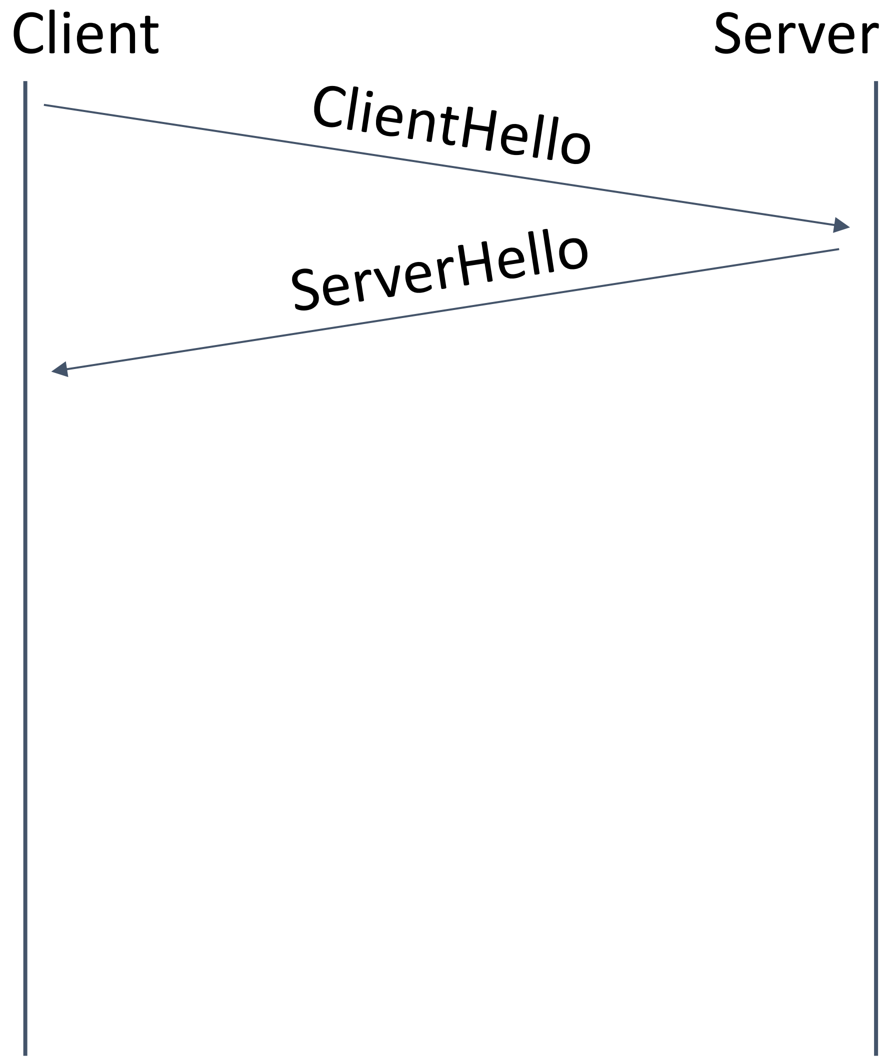


Today: Secure Internet Communication with TLS

- Goals of TLS
  - Confidentiality: Ensure that attackers cannot read your traffic
  - Integrity: Ensure that attackers cannot tamper with your traffic
    - Prevent replay attacks
      - The attacker records encrypted traffic and then replays it to the server
      - Example: Replaying a packet that sends “Pay $10 to Mallory”
  - Authenticity: Make sure you’re talking to the legitimate server
    - Defend against an attacker impersonating the server

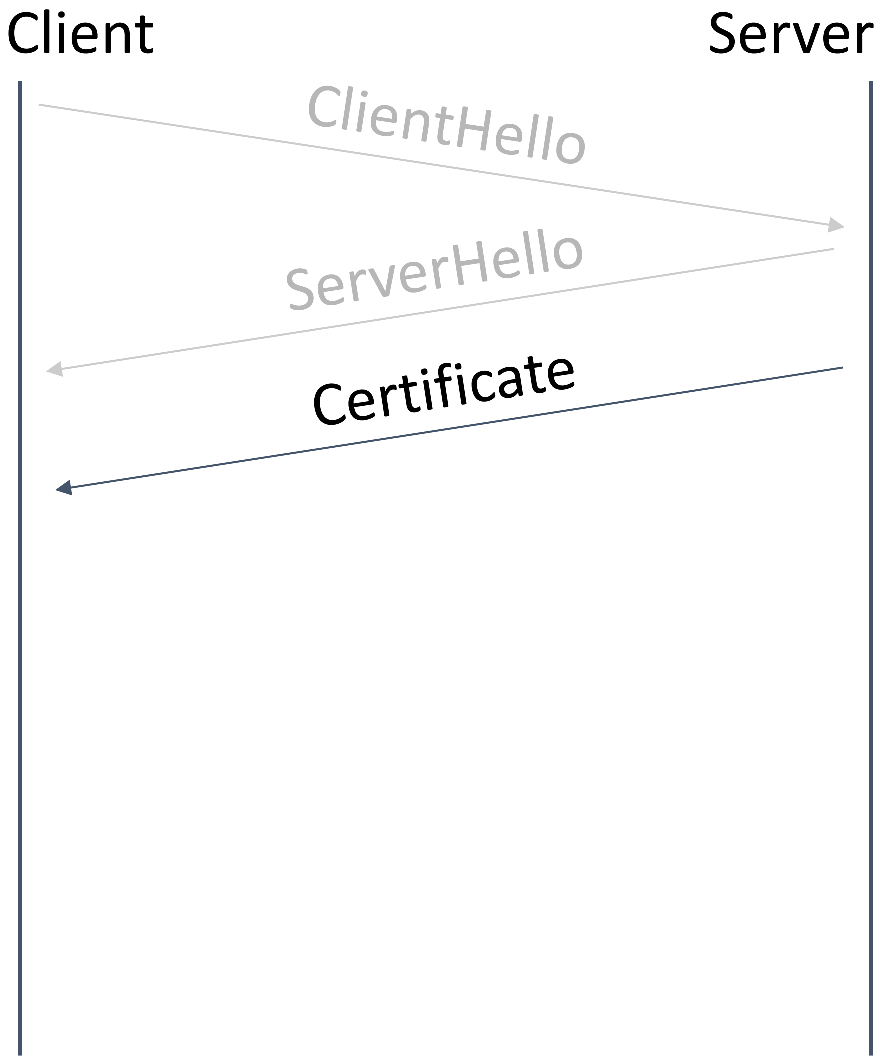
TLS Handshake Step 1: Exchange Hellos

- Assume an underlying TCP connection has already been formed
- The client sends ClientHello with
  - A 256-bit random number RB ("client random")
  - A list of supported cryptographic algorithms
- The server sends ServerHello with
  - A 256-bit random number RS ("server random")
  - The algorithms to use (chosen from the client's list)
- RB and RS prevent replay attacks
  - RB and RS are randomly chosen for every handshake
  - This guarantees that two handshakes will never be exactly identical



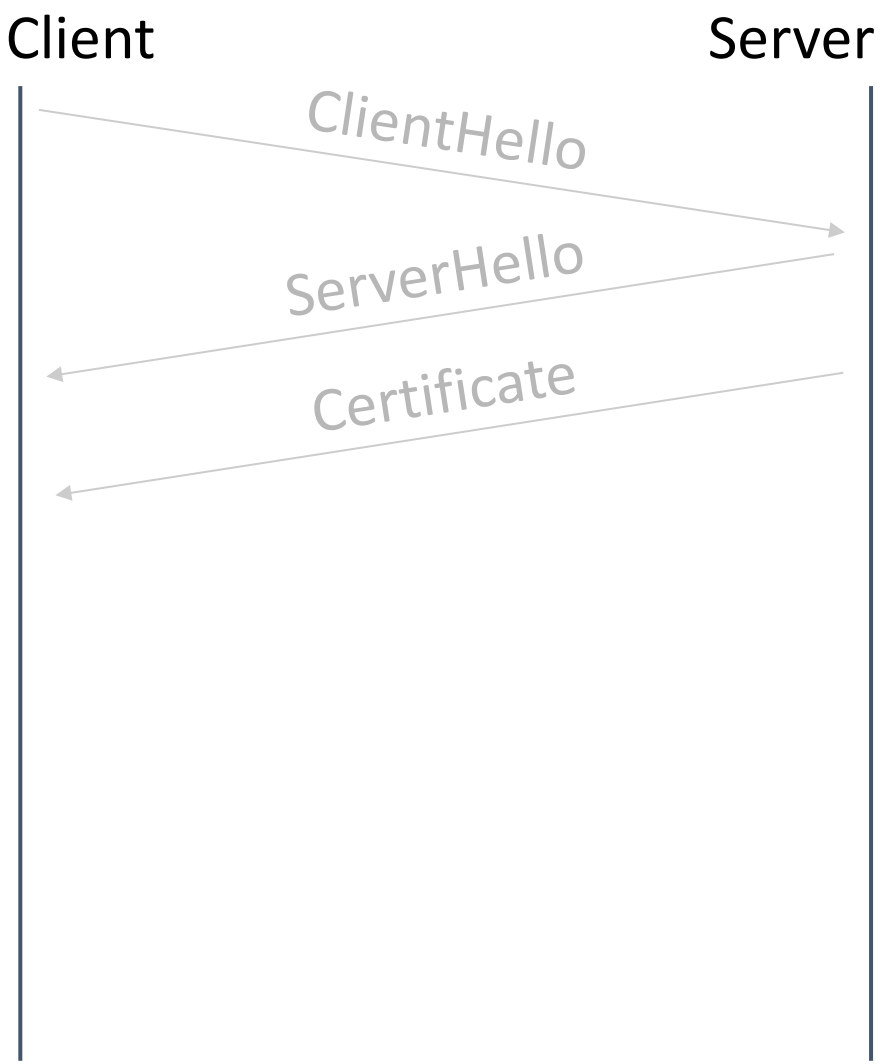
TLS Handshake Step 2: Certificate

- The server sends its certificate
  - Recall certificates: The server’s identity and public key, signed by a trusted certificate authority
- The client validates the certificate
  - Verify the signature in the certificate
- The client now knows the server’s public key
  - The client is not yet sure that they are talking to the legitimate server (not an impersonator)
  - Recall: Certificates are public. Anyone can provide a certificate for anybody



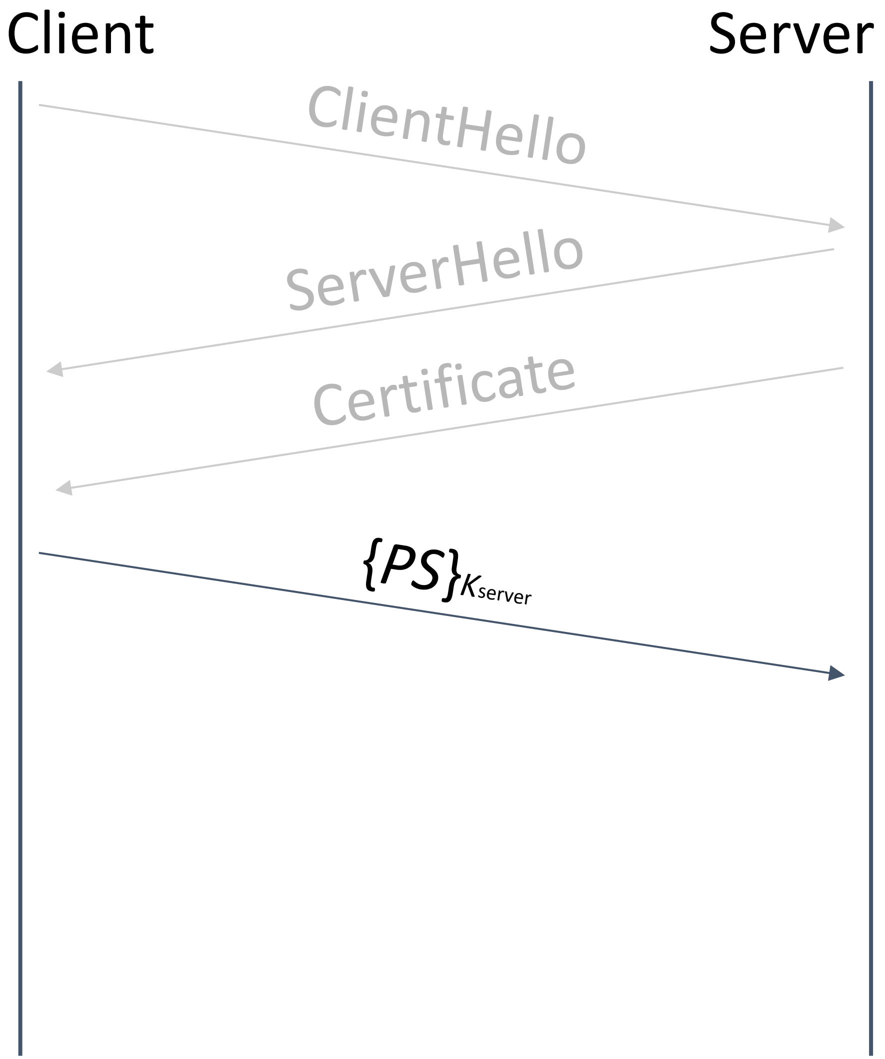
TLS Handshake Step 3: Premaster Secret

- This step has two main purposes
  - Make sure the client is talking to the legitimate server (not an impersonator)
    - The server must prove that it owns the private key corresponding to the public key in the certificate
  - Give the client and server a shared secret
    - An attacker should not be able to learn the secret
    - This will help the client and the server secure messages later
- Two approaches to sharing a premaster secret: RSA or Diffie-Hellman (DHE)



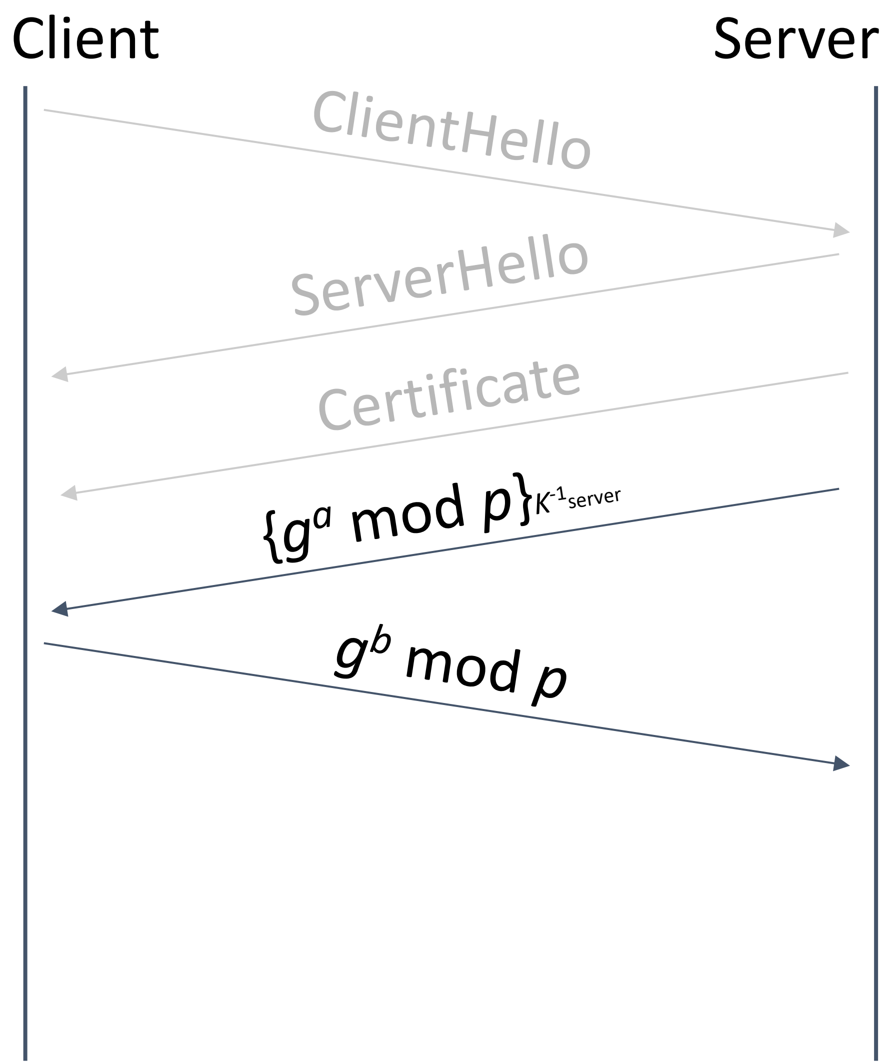
TLS Handshake Step 3: Premaster Secret (RSA)

- The client randomly generates a premaster secret (PS)
- The client encrypts PS with the server's public key and sends it to the server
  - The client knows the server's public key from the certificate
- The server decrypts the premaster secret
- The client and server now share a secret
  - Recall RSA encryption: Nobody except the legitimate server can decrypt the premaster secret
  - Proves that the server owns the private key (otherwise, it could not decrypt PS)



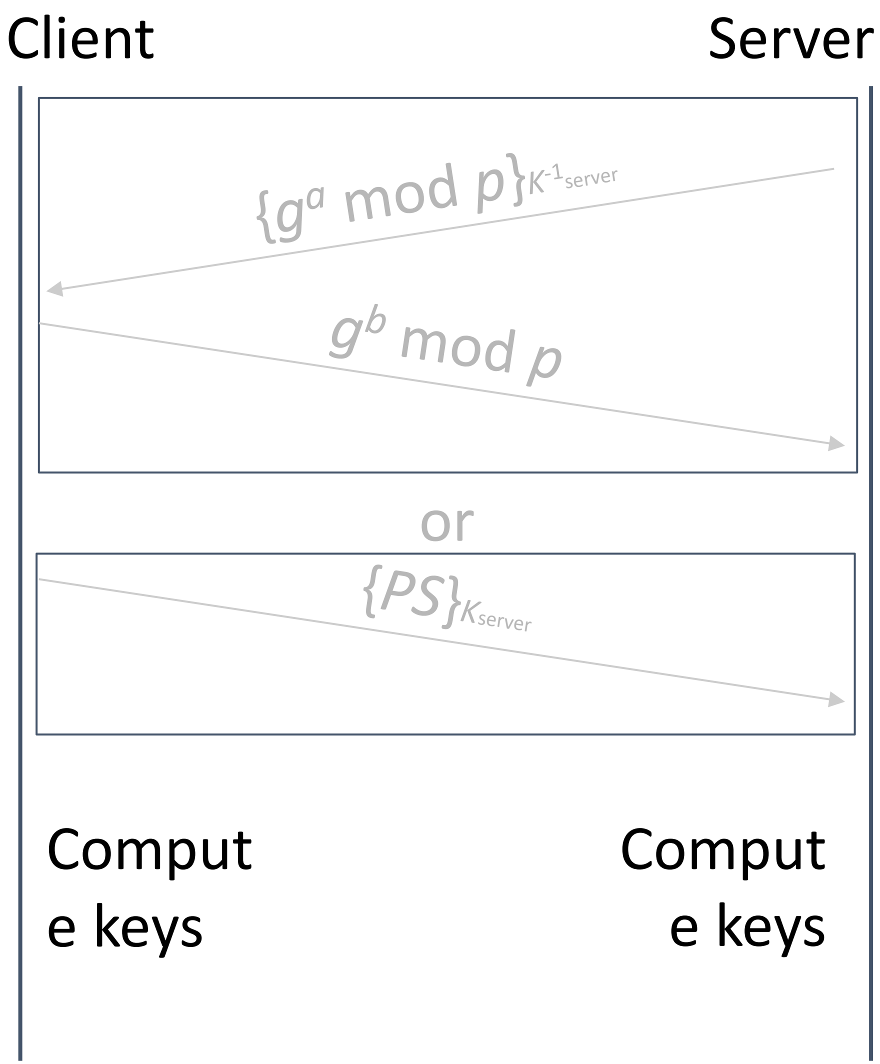
TLS Handshake Step 3: Premaster Secret (DHE)

- The server generates a secret a and computes $g^a \bmod p$
- The server signs $g^a \bmod p$ with its private key and sends the message and signature
- The client verifies the signature
  - Proves that the server owns the private key
- The client generates a secret b and computes $g^b \bmod p$
- The client and server now share a premaster secret: $g^{ab} \bmod p$
  - Recall Diffie-Hellman: an attacker cannot compute gab mod p



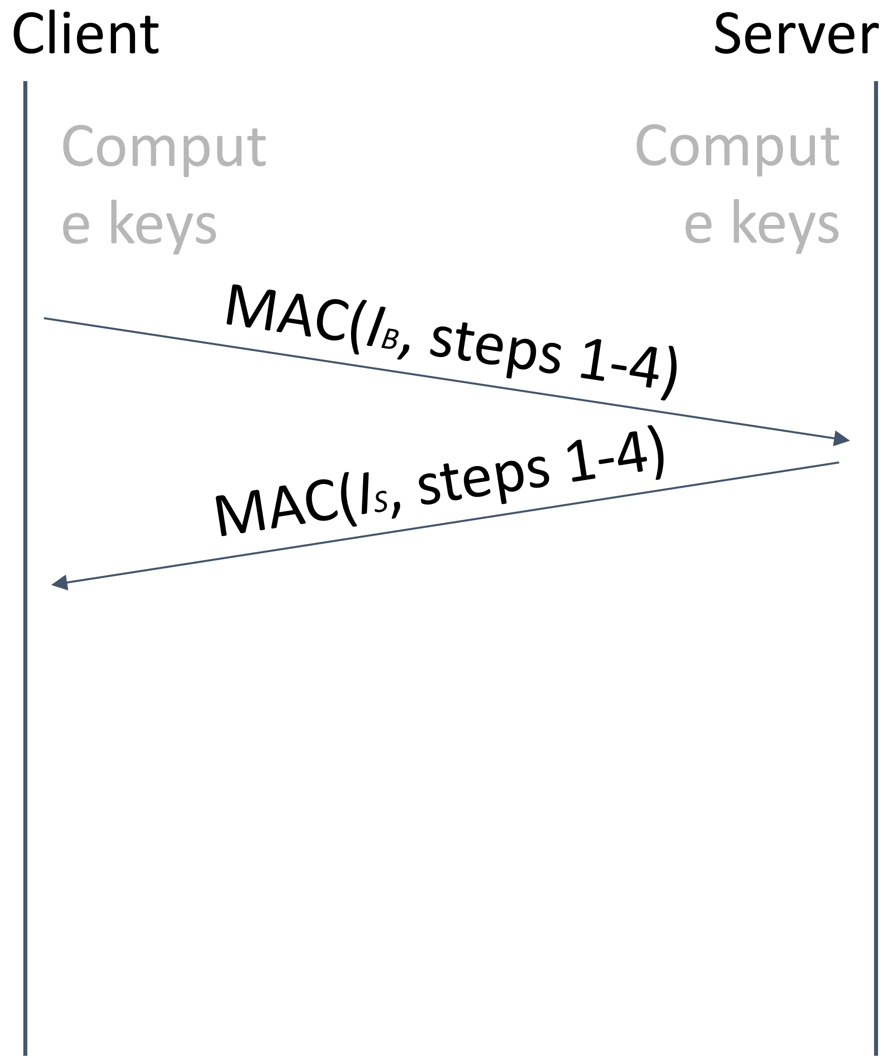
TLS Handshake Step 4: Derive Symmetric Keys

- The server and client each derive symmetric keys from RB, RS, and PS
  - Usually derived by seeding a PRNG with the three values
  - Changing any of the values results in different symmetric keys
- Four symmetric keys are derived
  - CB: For encrypting client-to-server messages
  - CS: For encrypting server-to-client messages
  - IB: For MACing client-to-server messages
  - IS: For MACing server-to-client messages
  - Note: Both client and server know all four keys



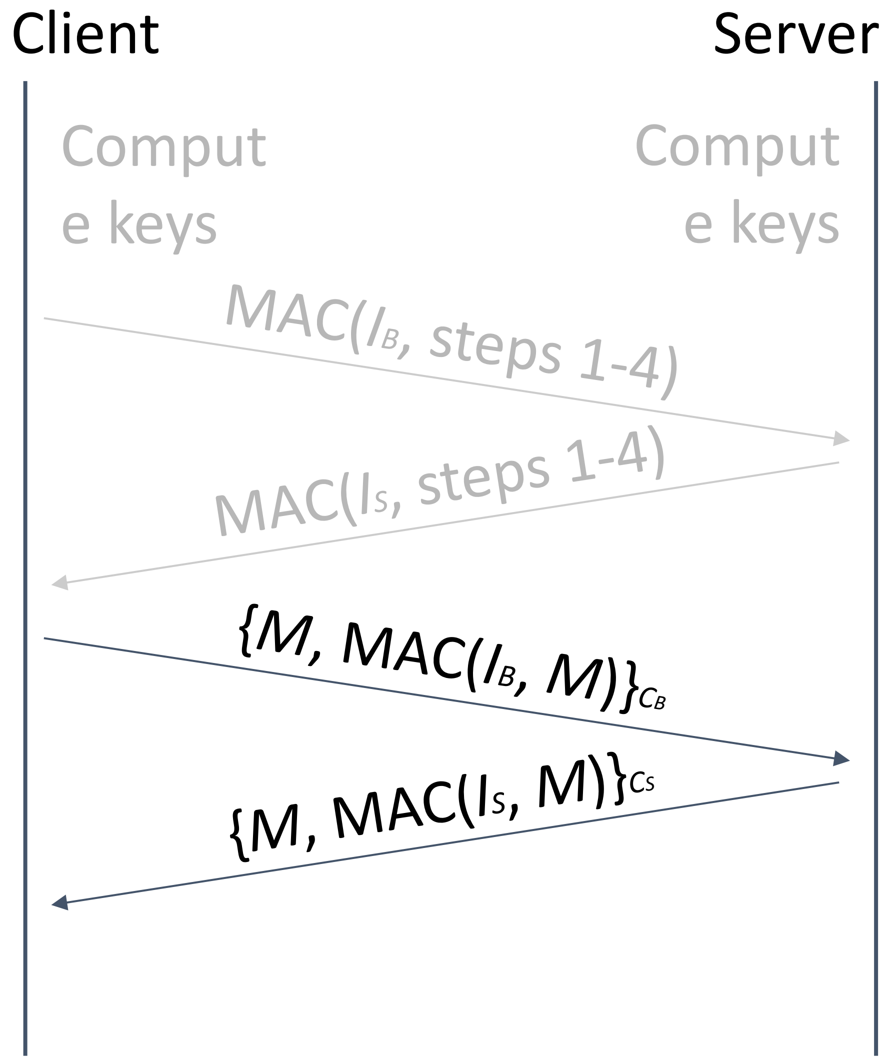
TLS Handshake Step 5: Exchange MACs

- The server and client exchange MACs on all the messages of the handshake so far
  - Recall MACs: Any tampering on the handshake will be detected



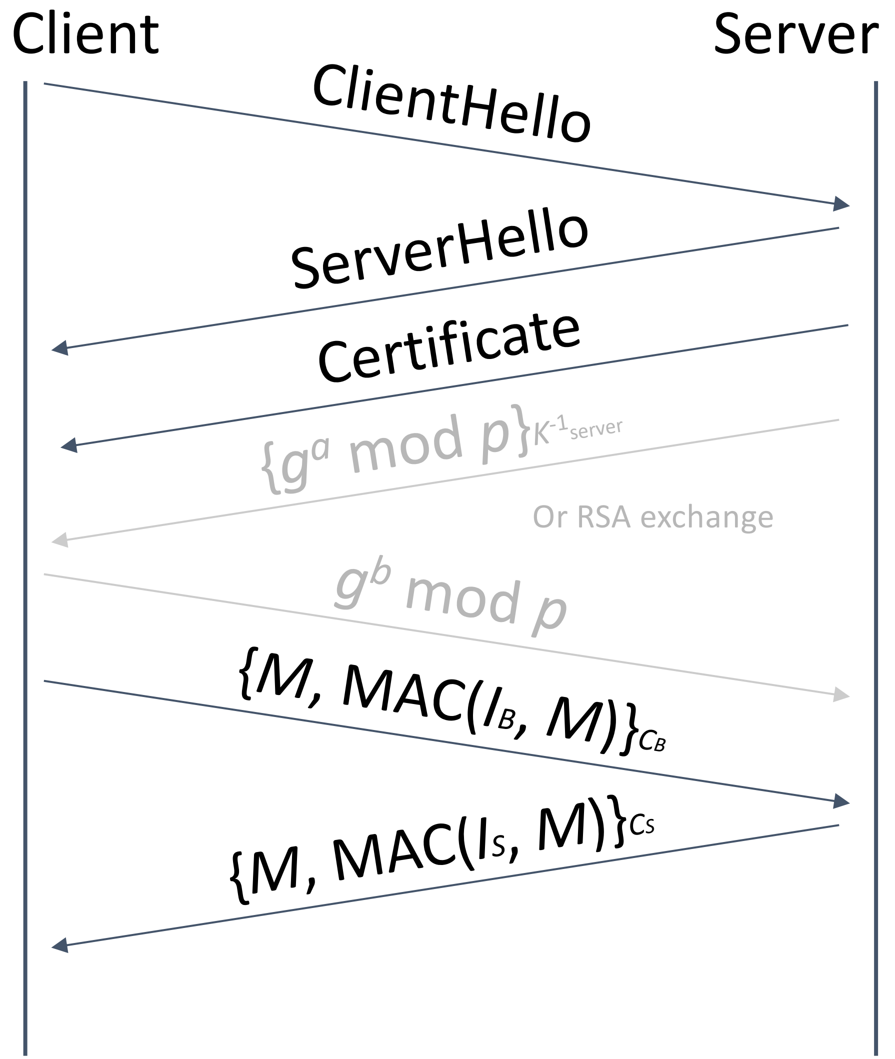
TLS Handshake Step 6: Send Messages

- Messages can now be sent securely
  - Encrypted then MAC'd
  - Note: TLS uses Authenticate-then-encrypt, even though encrypt-then-Authenticate is generally considered better.



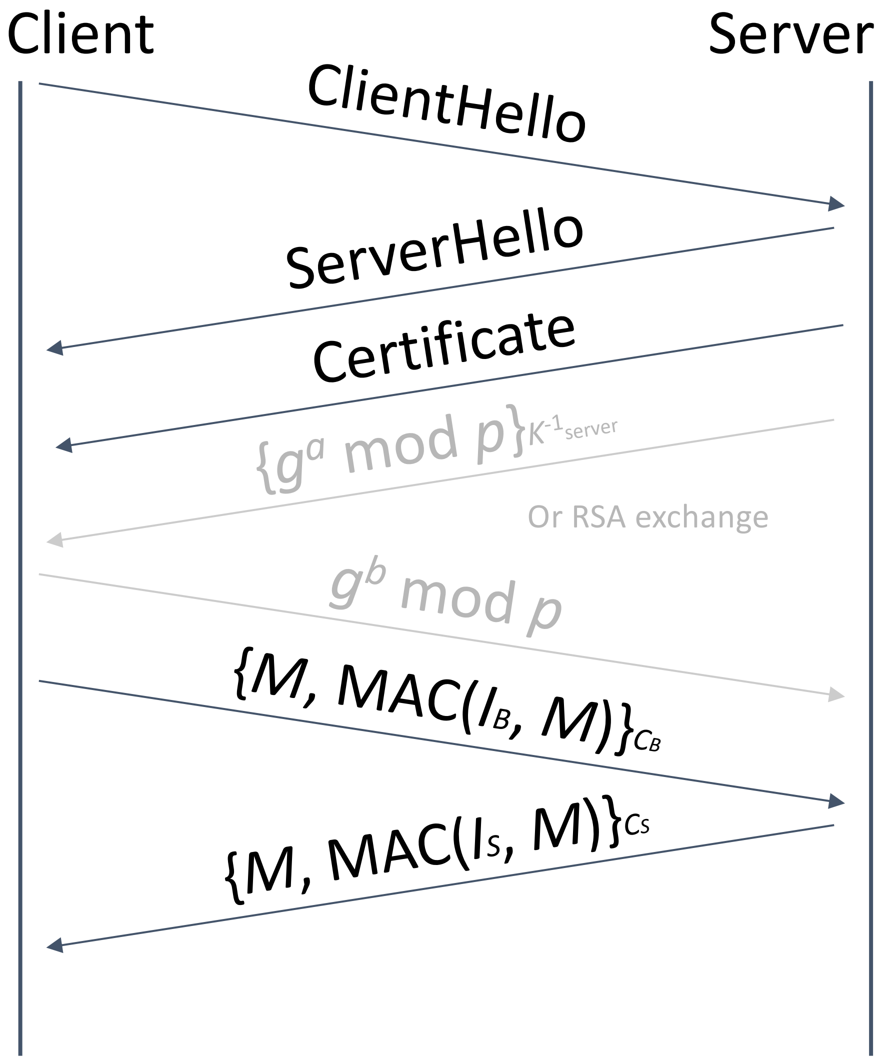
TLS: Talking to the Legitimate Server

- How can we be sure we are talking to the legitimate server?
  - The server sent its certificate, so we know the server's public key
  - The server proved that it owns the corresponding private key
    - RSA: The server decrypted the PS
    - DHE: The server signed its half of the exchange
- An attacker impersonating the server would not have the server's private key (assuming they have not compromised the server)



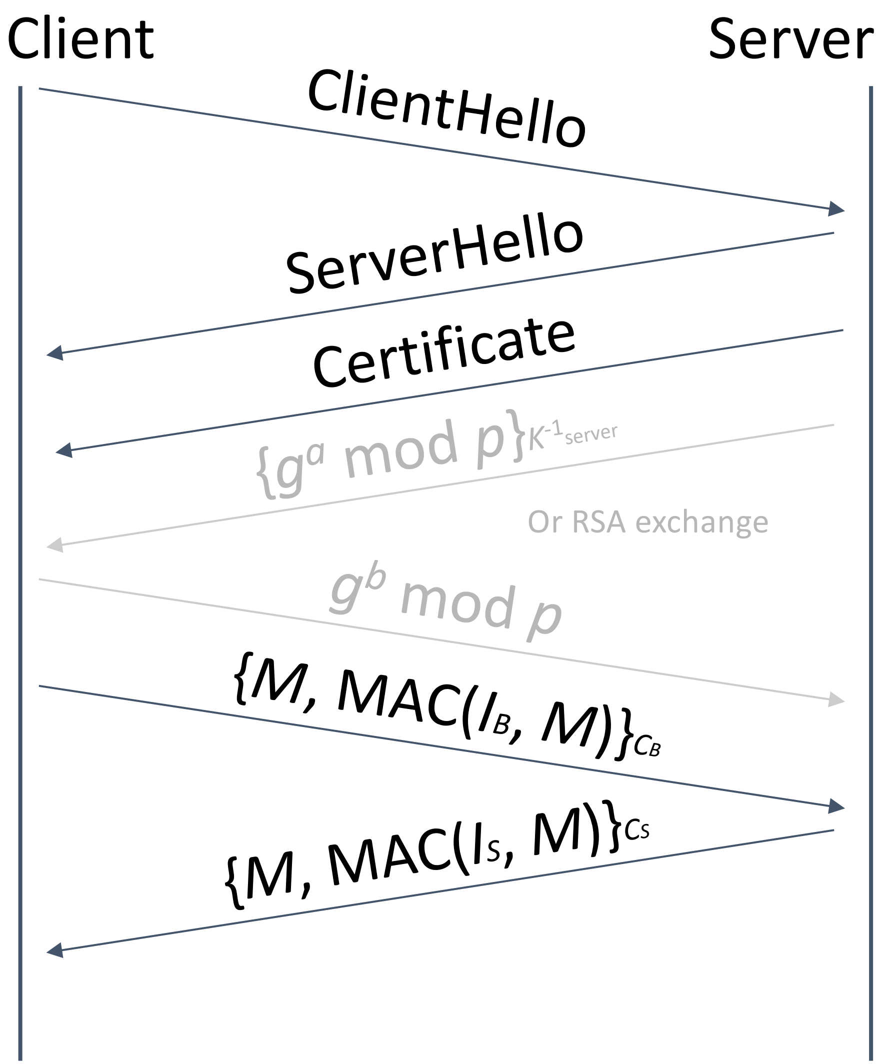
TLS: Securing Messages

- How can we be sure that network attackers can't read or tamper with our messages?
- The attacker doesn't know PS
  - RSA: PS was encrypted with the server's public key
  - DHE: An attacker cannot learn the Diffie-Hellman secret
- The symmetric keys are derived from PS
  - The attacker doesn't know the symmetric keys used to encrypt and MAC messages
- Encryption and MACs provide confidentiality and integrity



TLS: Replay Attacks

- How can we be sure that the attacker hasn't replayed old messages from the current TLS connection?
- Add record numbers in the encrypted TLS message
    - Every message uses a unique record number
    - If the attacker replays a message, the record number will be repeated
- TLS record numbers are not TCP sequence numbers
    - Record numbers are encrypted and used for security
    - Sequence numbers are unencrypted and used for correctness, in the layer below



Forward Secrecy

Forward Secrecy

- Forward secrecy: If an attacker records a connection now and compromises secret values later, they cannot compromise the recorded connection
- RSA TLS: No forward secrecy is guaranteed
  - The adversary can record RB, RS, and the encrypted PS
  - If the adversary later compromises the server's private key, they can decrypt PS and derive the keys!
- DHE TLS: Guaranteed forward secrecy
  - Diffie-Hellman provides forward secrecy: PS is deleted after the TLS session is over, so the adversary can't learn the keys, even if they later compromise the server's private key
  - Note: Because the server's Diffie-Hellman component is signed, the adversary can't MITM the Diffie-Hellman exchange without the server's private key

TLS 1.3 Changes

- TLS 1.3: The latest version of the TLS protocol (2018)
- RSA no longer supported (only DHE)
  - Guarantees forward secrecy
- Performance optimization: The client sends g^b mod p in ClientHello
  - If the server agrees to use DHE, the server sends g^a mod p (with signature) in ServerHello
  - Potentially saves two messages later in the handshake
- Eliminates attacks associated with the insecure MAC-then-encrypt pattern.

TLS in Practice

TLS: Efficiency

- Public-key cryptography: Minor costs
  - Client and server must perform Diffie-Hellman key exchange or RSA encryption/decryption
- Symmetric-key cryptography: Effectively free
  - Modern hardware has dedicated support for symmetric-key cryptography
  - Performance impact is negligible
- Latency: Extra waiting time before the first message
  - Must perform the entire TLS handshake before sending the first message

TLS Provides End-to-End Security

- TLS provides end-to-end security: Secure communication between the two endpoints, with no need to trust intermediaries
  - Even if everybody between the client and the server is malicious, TLS provides a secure communication channel
  - End-to-end security does not help if one of the endpoints is malicious (e.g. communicating with a malicious server)
  - Example: An local network attacker (on-path) tries to read our Wi-Fi session, but can’t read TLS messages
  - Example: A man-in-the-middle tries to inject TCP packets, but packets will be rejected because the MAC won’t be correct
- Using TLS defends against most lower-level network attacks

TLS Does Not Provide Anonymity

- Anonymity: Hiding the client’s and server’s identities from attackers
- An attacker can figure out who is communicating with TLS
    - The certificate is sent during the TLS handshake, containing the server’s name
    - The client may also indicate the name of the server in the ClientHello (called Server Name Indication, or SNI)
    - An attacker can see IP addresses and ports of the underlying IP and TCP protocols

TLS Does Not Provide Availability

- Availability: Keeping the connection open in the face of attackers
- An attacker can stop a TLS connection
  - MITM can drop encrypted TLS packets
  - On-path attacker can still do RST injection to abort the underlying TCP connection
- Result: A TLS connection can still be censored
  - The censor can block TLS connections

TLS for Applications

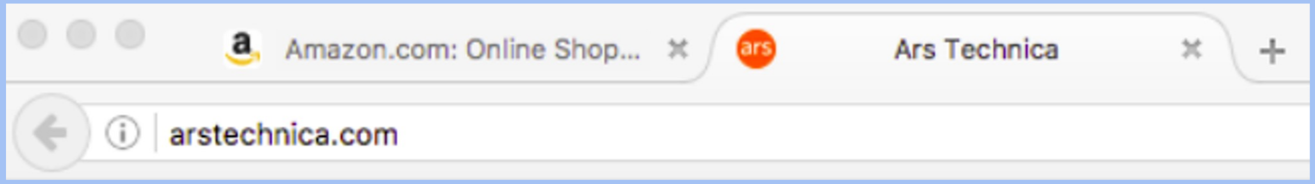
- Internet layering: TLS provides services to higher layers (the application layer)
- HTTPS: The HTTP protocol run over TLS
  - In contrast, HTTP runs over plain TCP, with no TLS added
- Other secure application-layer protocols besides HTTPS exist
  - Pretty much anything that runs over TCP can also run over TLS, since the bytestream abstraction is maintained
  - Example: Email protocol can use the STARTTLS command to uses TLS to secure communications
- TLS does not defend against application-layer vulnerabilities
  - Example: SQL injection, XSS, CSRF, and buffer overflow vulnerabilities in the application are still exploitable over TLS

SSL Stripping Attacks

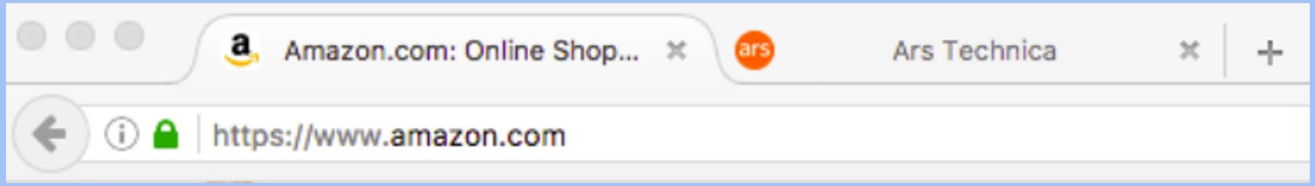
- Browsers often default to using unencrypted HTTP
  - If a user types google.com into the browser, the browser opens http://www.google.com
  - To mitigate this, websites will often redirect from the HTTP to the HTTPS version of its site
  - This requires the client to first receive the unprotected HTTP redirect response
- SSL stripping: Forcing a user to use unencrypted HTTP instead of HTTPS
  - A MITM attacker intercepts the first HTTP request and creates their own HTTPS connection to the server
  - The user never receives a redirect to HTTPS, so it believes the site wants them to use HTTP
  - Defense: HTTP Strict-Transport-Security (HSTS) header tells browsers to only access the server with HTTPS

TLS in Browsers

- Original design:
  - When your browser communicates with a server over TLS, your browser displays a lock icon
  - If TLS is not used, there is no lock icon
- What the lock icon means
  - Communication is encrypted (TLS guarantee)
  - You are talking to the legitimate server (TLS guarantee)
  - Any external images or scripts are also fetched over TLS



This website uses HTTP: no lock icon



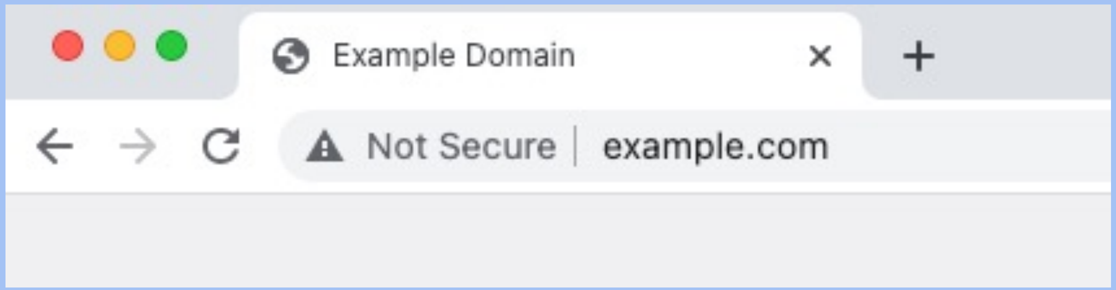
This website uses HTTPS: lock icon

TLS in Browsers

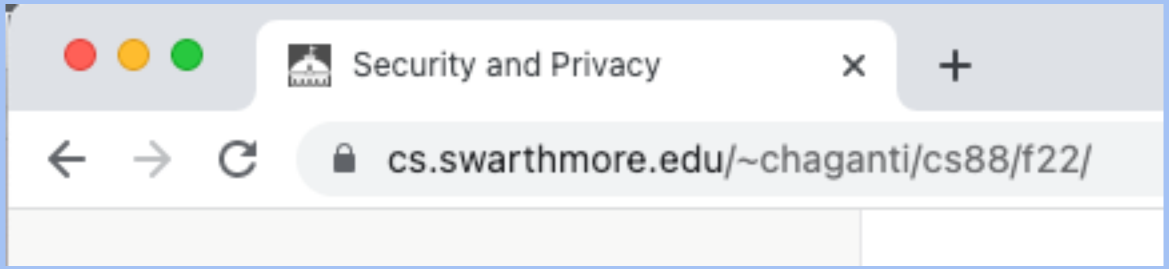
- What users think the lock icon means
  - This website is trustworthy, no matter where the lock icon actually appears
- Attack: The attacker adds their own lock icon somewhere on the page
  - The user thinks they're using TLS, but actually is not using TLS
- Attack: The user might be communicating with an attacker's website over TLS
  - The lock icon appears, but the user is actually vulnerable!

TLS in Browsers

- Modern design: Add a “not secure” icon to connections that don’t use TLS
  - Adds a signal on unencrypted sites
  - Encourages websites to stop supporting all unencrypted, HTTP traffic and redirect to HTTPS



This website uses HTTP: insecure icon



This website uses HTTPS: lock icon

TLS Attack: PRNG Sabotage

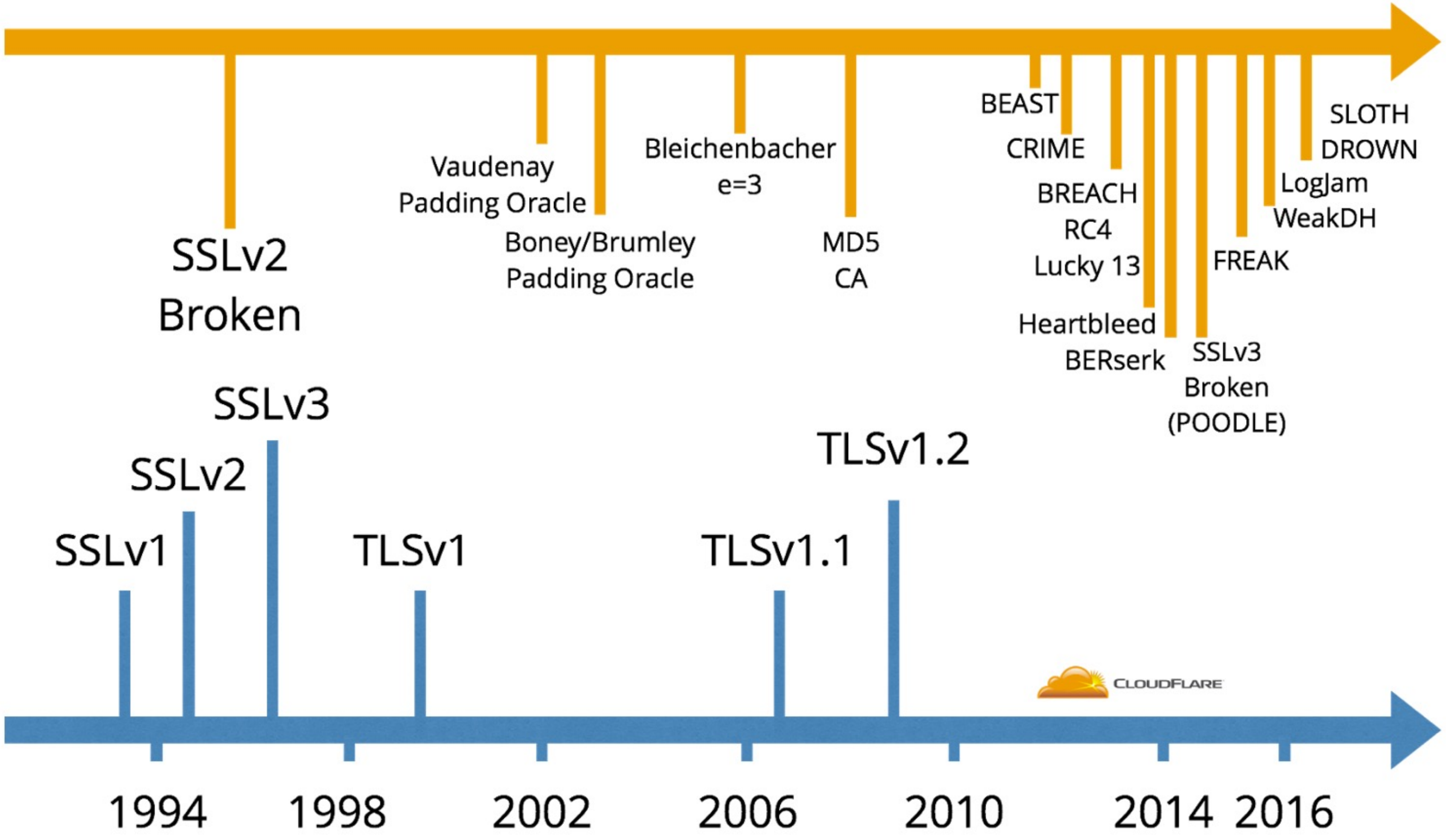
- TLS with Diffie-Hellman
  - An attacker who learns the DHE secret a can derive the PS g^{ab} mod p (recall g^{b} mod p is sent over the channel)
  - An attacker who knows the PS can derive the symmetric keys (recall RC and RS are sent over the channel)
- Use a PRNG to generate all random values
  - Includes the server DHE secret a and the client DHE secret b
- Attack: PRNG is sabotaged + no rollback resistance?
  - Threat: Attacker has compromised internal state of PRNG and can learn the next bit.
  - Rollback-Resistance: any previously-generated output of the pRNG should still be computationally indistinguishable from random, even if the attacker knows the current internal state of the PRNG
- Attack: See subsequent PRNG output and work backwards to learn the DHE secret

TLS 1.3: the new standard

- Several years of collaboration between industry and academia
  - Standardized by IETF in
- Major differences:
  - RSA key exchange removed: no passive decryption attacks
  - Only secure DFH parameters allowed: no bad choices in parameters
  - Handshake encrypted immediately after key exchange: limits metadata available to eavesdropper
  - Protocol downgrade protection: protects against being downgraded to prior insecure versions

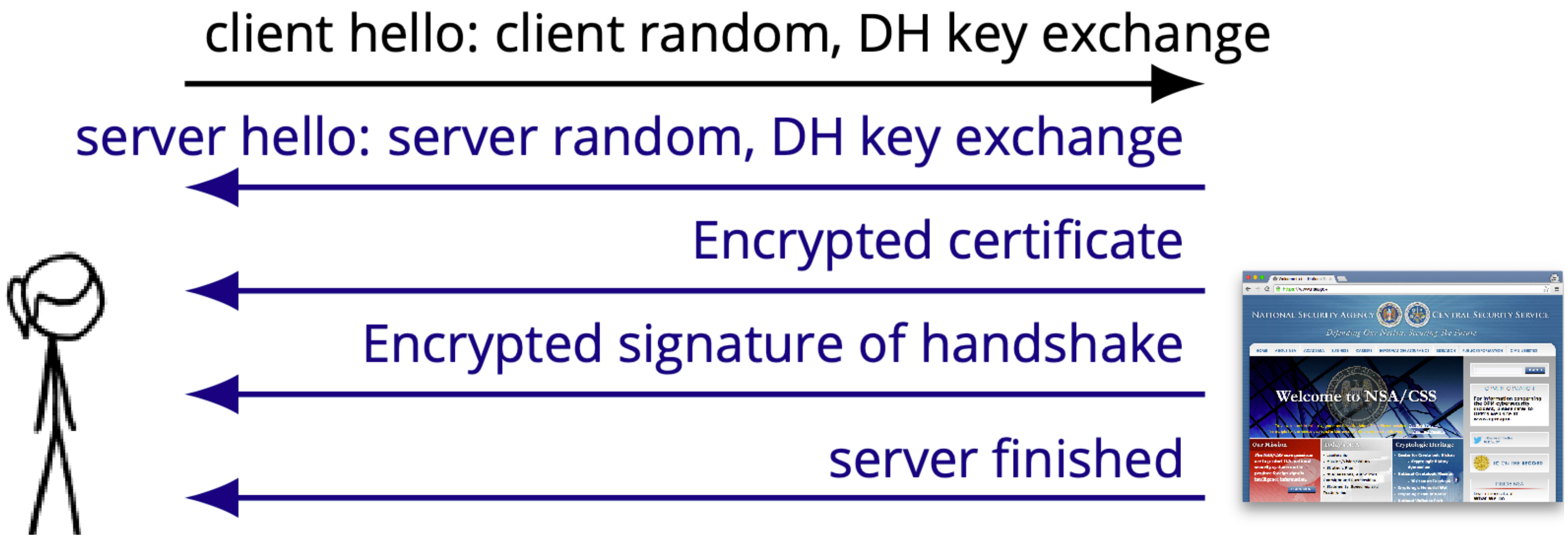
TLS v. 1.2 and below have had a lot of vulnerabilities

- Early versions of SSL developed before cryptographic protocol design was fully understood.
- Later protocol versions retained insecure options for backwards compatibility.



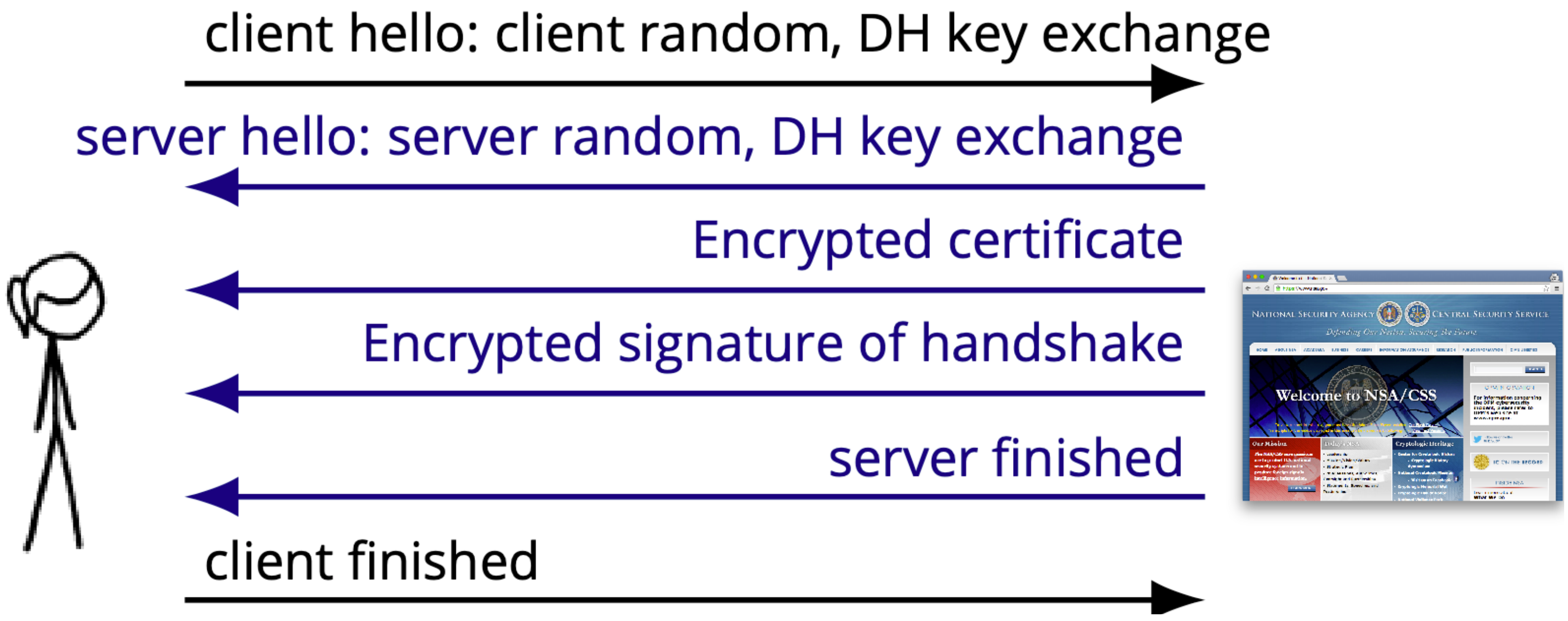
TLS 1.3

TLS 1.3 encrypts the handshake immediately after doing a Diffie-Hellman key exchange.



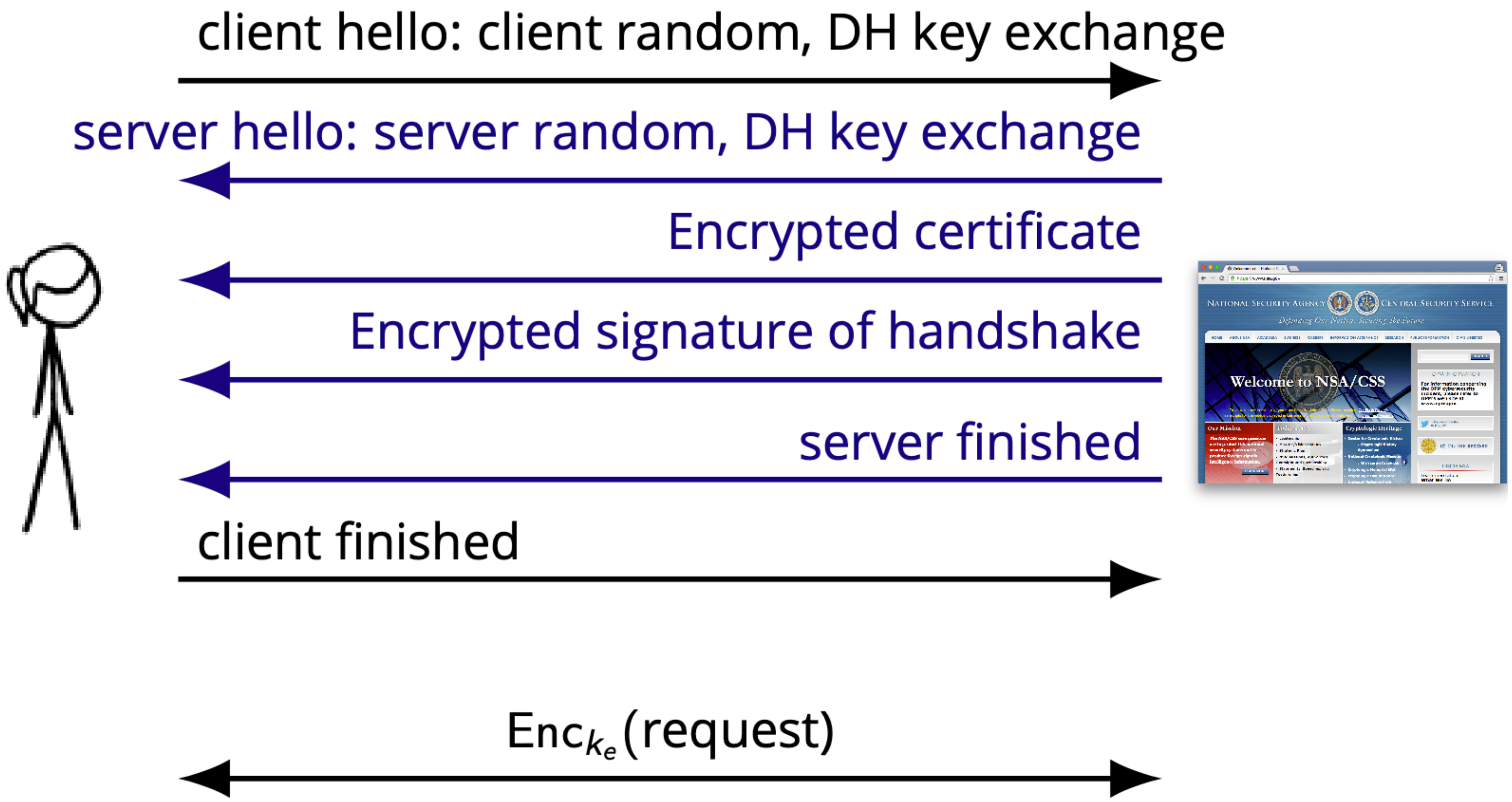
TLS 1.3

TLS 1.3 encrypts the handshake immediately after doing a Diffie-Hellman key exchange.



TLS 1.3

TLS 1.3 encrypts the handshake immediately after doing a Diffie-Hellman key exchange.



TLS 1.3: deployment difficulties

- Adoption slower than it should be.

“Despite widespread TLS 1.3 adoption, old and vulnerable protocols are being left enabled. RSA handshakes are allowed by 52 percent of web servers, SSL v3 is enabled on 2 percent of sites, and 2.5 percent of certificates had expired."

-f5.com

Major reasons

- HTTPS proxies: Reliance on RSA key exchange to make passive decryption and traffic analysis easier. Removing RSA key exchange breaks these boxes
- MiTM hardware
- Bad implementations with hardcoded TLS versions. No way to update these ☹.