BGP, TCP, UDP, and TLS

CS 161 Spring 2022 - Lecture 18

Next: BGP, TCP, and UDP

- Layer 3 (inter-network)
  - Border Gateway Protocol (BGP): Routing packets
- Layer 4 (transport)
  - Transmission Control Protocol (TCP): Reliably sending packets
  - User Datagram Protocol (UDP): Non-reliably sending packets

Border Gateway Protocol (BGP)

Textbook Chapter 29

Review: Internet Protocol (IP)

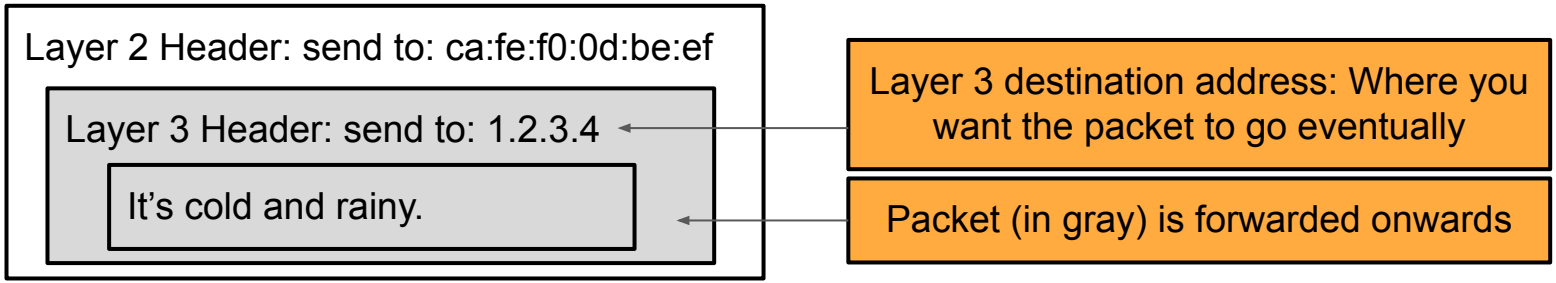
- **Internet Protocol** (**IP**): The universal layer-3 protocol that all devices use to transmit data over the Internet
- **IP address**: An address that identifies a device on the Internet
  - IPv4 is 32 bits (e.g. `35.163.72.93`)
  - IPv6 is 128 bits (e.g. `2607:f140:8801:0000:0000:0000:0001:0023`)
    - Shorthand: omit sets of zeros: `2607:f140:8801::1:23`
  - Globally unique from any single perspective
    - For now, you can think of them as just being globally unique
  - IP addresses help nodes make decisions on where to forward the packet

Subnets

- Recall: Layer 3 routes packets across multiple nodes on different LANs
  - A packet might make many hops across different local networks before it can reach its destination
- IP routes by subnets, which are groups of addresses with a common prefix
  - A subnet is written as a prefix followed by the length of the prefix
    - Example: `128.32.0.0/16` is an IPv4 subnet with all addresses that begin with the prefix of `128.32.*`
    - Since an IPv4 is a 32-bit address and there are 16 bits in the prefix, this subnet represents $2^{32 - 16} = 2^{16}$ addresses

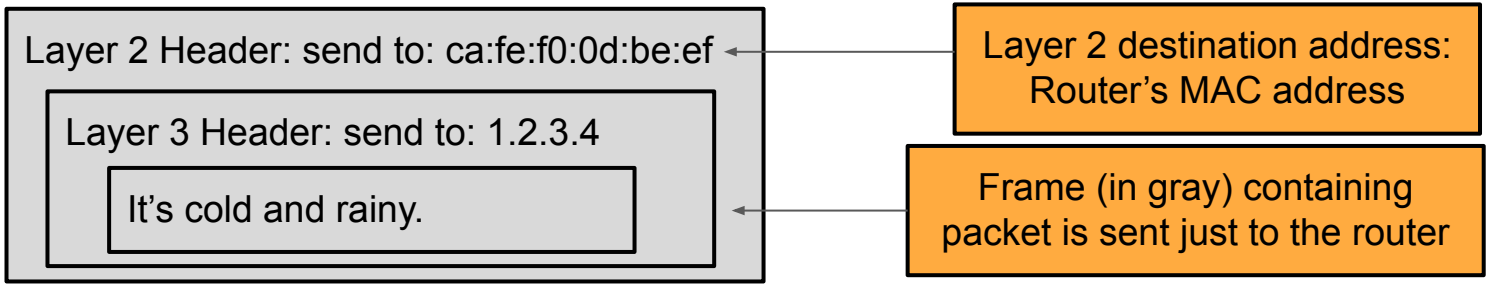
Routing Packets

- To send a packet to a computer within the local network:
  - Verify that the destination IP is in the same subnet
  - Use ARP to get the destination MAC address
  - Send the packet directly to the destination using the destination MAC address
- To send a packet to a computer that is not within the local network:
  - Use ARP to get the gateway's MAC address
  - Send the packet to the gateway
  - Past the gateway, the packet goes to the Internet
  - It's the gateway's job to deliver the packet closer to the destination



Routing Packets

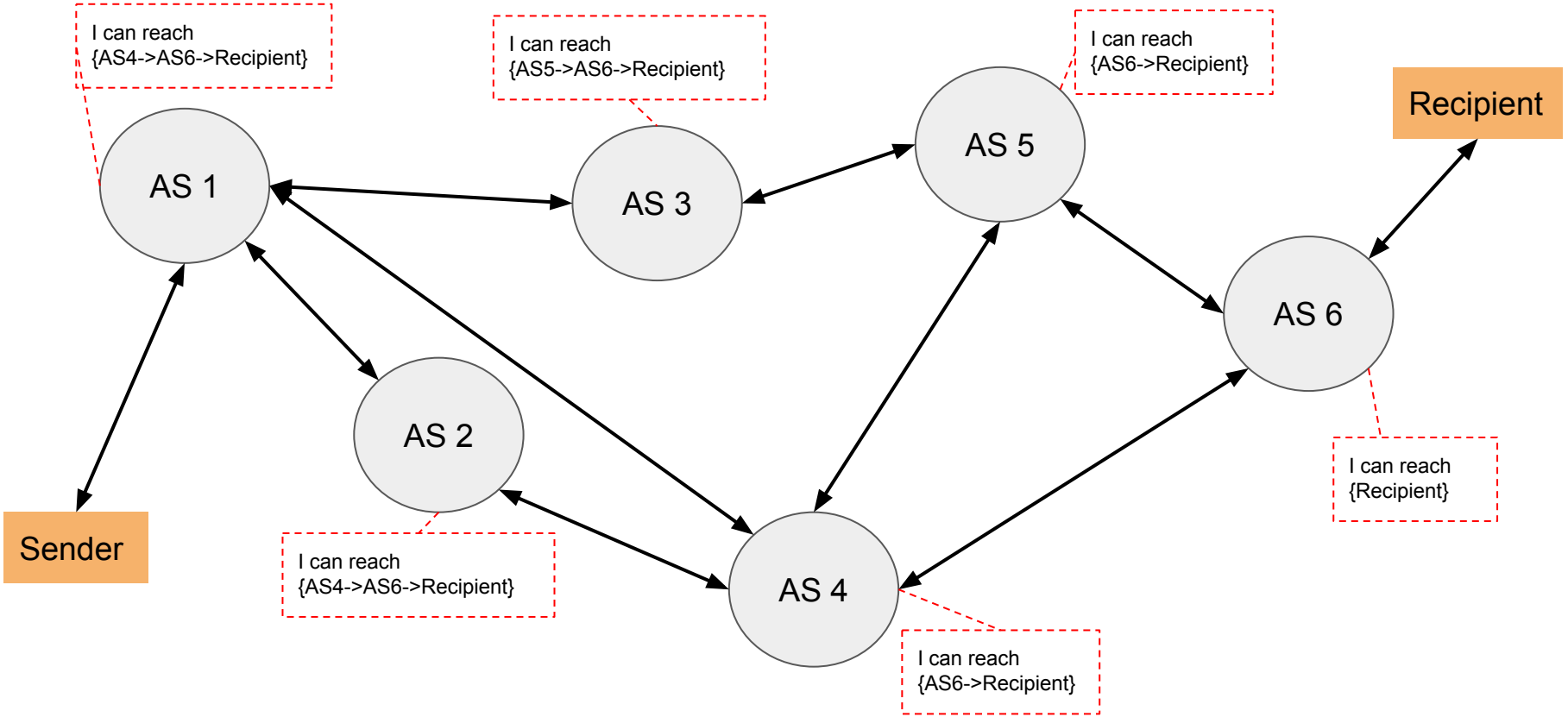
- To send a packet to a computer within the local network:
  - Verify that the destination IP is in the same subnet
  - Use ARP to get the destination MAC address
  - Send the packet directly to the destination using the destination MAC address
- To send a packet to a computer that is not within the local network:
  - Use ARP to get the gateway's MAC address
  - Send the packet to the gateway
  - Past the gateway, the packet goes to the Internet
  - It's the gateway's job to deliver the packet closer to the destination



Autonomous Systems

- Once your system sends the packet to the gateway, the packet has to be routed through the Internet
- The Internet is a network of networks, comprised of many **autonomous systems** (**AS**)
  - Each AS handles its own internal routing
  - Each AS is uniquely identified by its autonomous system number (ASN)
  - Each AS is comprised of one or more LANs
  - The AS can forward packet to other connected ASes
- The protocol for communicating between different Autonomous Systems is **Border Gateway Protocol** (**BGP**)
  - Each router announces what networks it can provide and the path onward from the router
  - The most precise route with the shortest path and no loops is the preferred route

BGP


I can reach
{AS4->AS6->Recipient}
I can reach
{AS5->AS6->Recipient}
I can reach
{AS6->Recipient}
Recipient
AS 1
AS 3
AS 5
AS 6
AS 2
AS 4
Sender
I can reach
{Recipient}
I can reach
{AS4->AS6->Recipient}
I can reach
{AS6->Recipient}

IP and BGP Attacks

- Each AS implicitly trusts the surrounding ASes and accepts advertised routes
- **IP spoofing**: Malicious clients can send IP packets with source IP values set to a spoofed value
  - Edge ASes should block packets with source IPs set to the wrong value, but some don't
  - Enables packets that look like they're coming from someone else!
- **BGP hijacking**: A malicious autonomous system can lie and claims itself to be responsible for a network which it isn't
  - Example: AS3 broadcasts that it is responsible for 128.32.0.0/16
    - Now, the malicious AS can act as a MITM for traffic to 128.32.0.0!

BGP Hijacking in the Wild

- Bad Guy's Objective:
  - Cause cryptocurrency exchange to load a malicious piece of JavaScript
  - Cryptocurrency exchange uses third-party hosted scripts
- BGP hijack
  - Announce route for the script hosting IP
  - Obtain TLS certificate for script hosting IP
    - We'll see TLS certificates later
    - After all, it IS that computer now!
  - Serve up malicious JavaScript to steal $2M in miscellaneous cryptocurrencies

FREEDOM TO TINKER

Link

Attackers exploit fundamental flaw in the web's security to steal $2 million in cryptocurrency

Henry Birge-Lee, Liang Wang, Grace Cimaszewski, Jennifer Rexford and Prateek Mittal

March 9, 2022

Defending BGP in Practice

- Social contact and monitoring
  - Numerous BGP collectors obtain data and look for anomalous BGP behavior
  - A rogue AS will be cut off from the rest of the Internet:
    Creates a social incentive to be honest
- Resource Public Key Infrastructure (RPKI)
  - Adds cryptographic protections to routing information
  - About 35% of the Internet now works with this
    - Amazon is a big user: If you use Amazon infrastructure you will generally resist BGP attacks

Transmission Control Protocol (TCP)

Textbook Chapter 30

Review: IP Reliability

- **Reliability** ensures that packets are received correctly
  - If the packet has been changed (random error or malicious tampering), it should not be correctly received
  - IP packets include a checksum (unkeyed function, protects against random errors)
  - However, there is no cryptographic MAC, so there are no guarantees if an attacker maliciously modifies packets (and modifies the checksum accordingly)
- IP is unreliable and only provides a **best effort** delivery service, which means:
  - Packets may be lost (“dropped”)
  - Packets may be corrupted
  - Packets may be delivered out of order
- It is up to higher level protocols to ensure that the connection is reliable

Scratchpad: Let's Design It Together

- 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

Transmission Control Protocol (TCP)

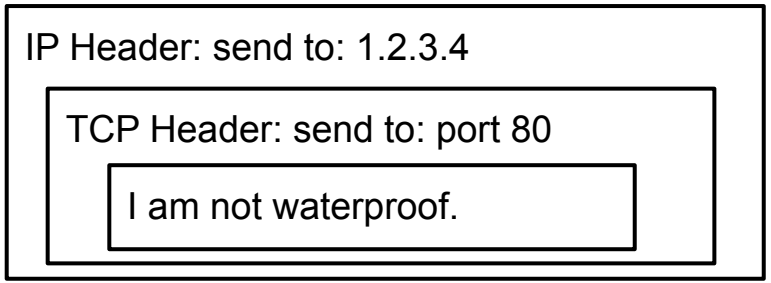
- Provides a byte stream abstraction
  - Bytes go in one end of the stream at the source and come out at the other end at the destination
  - TCP automatically breaks streams into **segments**, which are sent as layer 3 packets
- Provides ordering
  - Segments contain sequence numbers, so the destination 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

- **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)
- Remember: TCP is built on top of IP, so the IP header (and therefore the IP address) is still present



Establishing Sequence Numbers

- Each TCP connection requires two sets of sequence numbers
  - One sequence number for messages from the client to the server
  - One sequence number for messages from the server to the client
- Before starting a TCP connection, the client and server must agree on two **initial sequence numbers** (ISNs)
  - The ISNs are different and random for every connection (for security reasons, as we'll see soon)

H	e	l	l	o		s	e	r	v	e	r
50	51	52	53	54	55	56	57	58	59	60	61

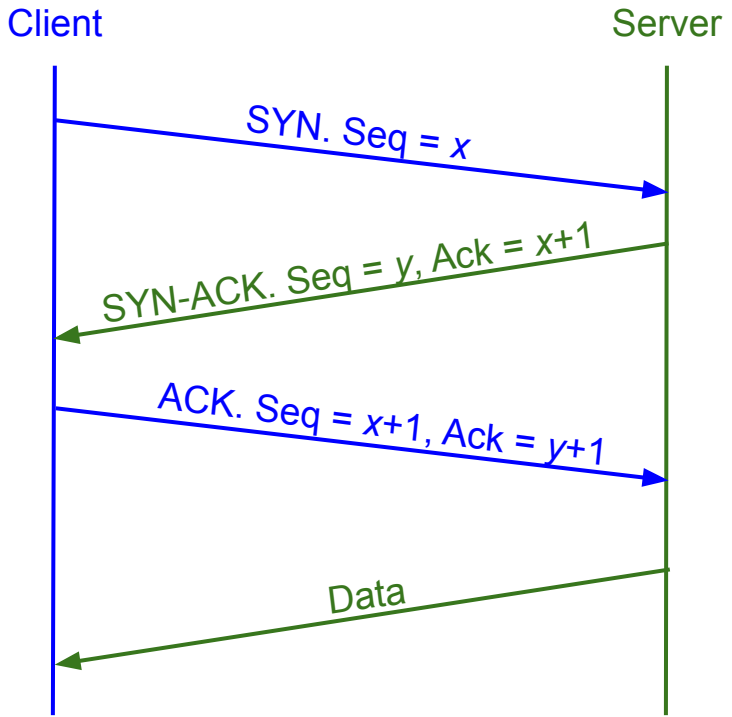
In this example, messages from the client are numbered starting at 50.

H	e	l	l	o		c	l	i	e	n	t
25	26	27	28	29	30	31	32	33	34	35	36

In this example, messages from the server are numbered starting at 25.

TCP: 3-Way Handshake

1. Client chooses an initial sequence number *x* its bytes and sends a SYN (synchronize) packet to the server
2. Server chooses an initial sequence number *y* for its bytes and responds with a SYN-ACK packet
3. Client then returns with an ACK packet
4. Once both hosts have synchronized sequence numbers, the connection is "established"



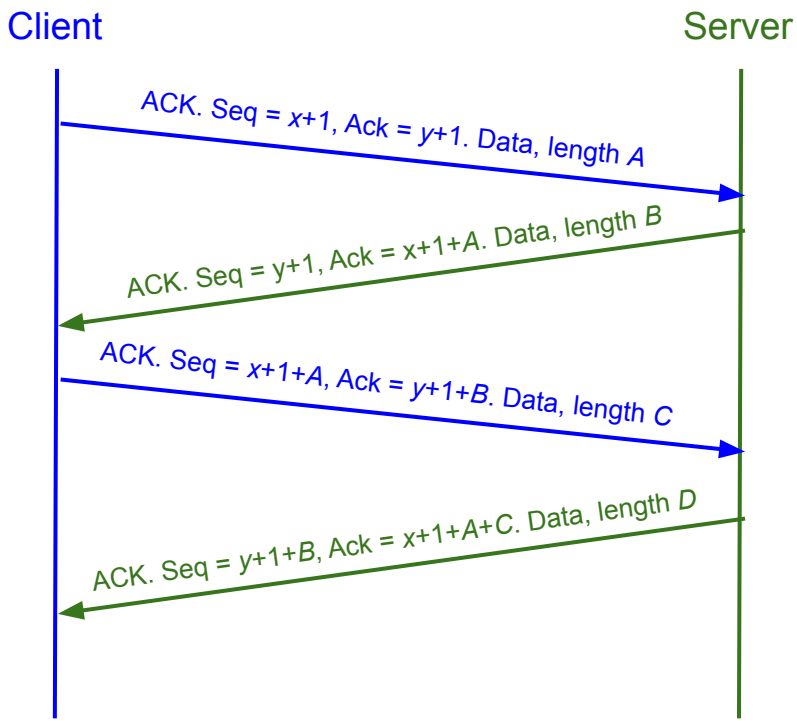
TCP: Sending and Receiving Data

- The TCP handlers on each side track which TCP segments have been received for each connection
  - A connection is identified by these 5 values (sometimes called a 5-tuple)
    - Source IP
    - Destination IP
    - Source Port
    - Destination Port
    - Protocol
- Data from the bytestream can be presented to the application when all data before has been received and presented
  - Recall: TCP presents data to the application as a bytestream, so the order must be preserved from one end to the other, even if packets are received out of order

TCP: Sending and Receiving Data

- Byte *i* of the bytestream is represented by sequence number *x* + *i*
  - The first byte is byte *i* = 1, since sequence number *x* was used for the SYN packet and *y* for the SYN-ACK packet
- A packet's sequence number is the number of the first byte of its data
  - This number is from the sender's set of sequence numbers
- A packet's ACK number, if the ACK flag is set, is the number of the byte immediately after the last received byte
  - This number is from the receiver's set of sequence numbers
  - This would be (sequence number) + (length of data) for the last received packet

TCP: Sending and Receiving Data



TCP: Retransmission

- If a packet is dropped (lost in transit):
  - The recipient will not send an ACK, so the sender will not receive the ACK
  - The sender repeatedly tries to send the packet again until it receives the ACK
- If a packet is received, but the ACK is dropped:
  - The sender tries to send the packet again since it didn't receive the ACK
  - The recipient ignores the duplicate data and sends the ACK again
- When packets are dropped in TCP, TCP assumes that there is congestion and sends the data at a slower rate

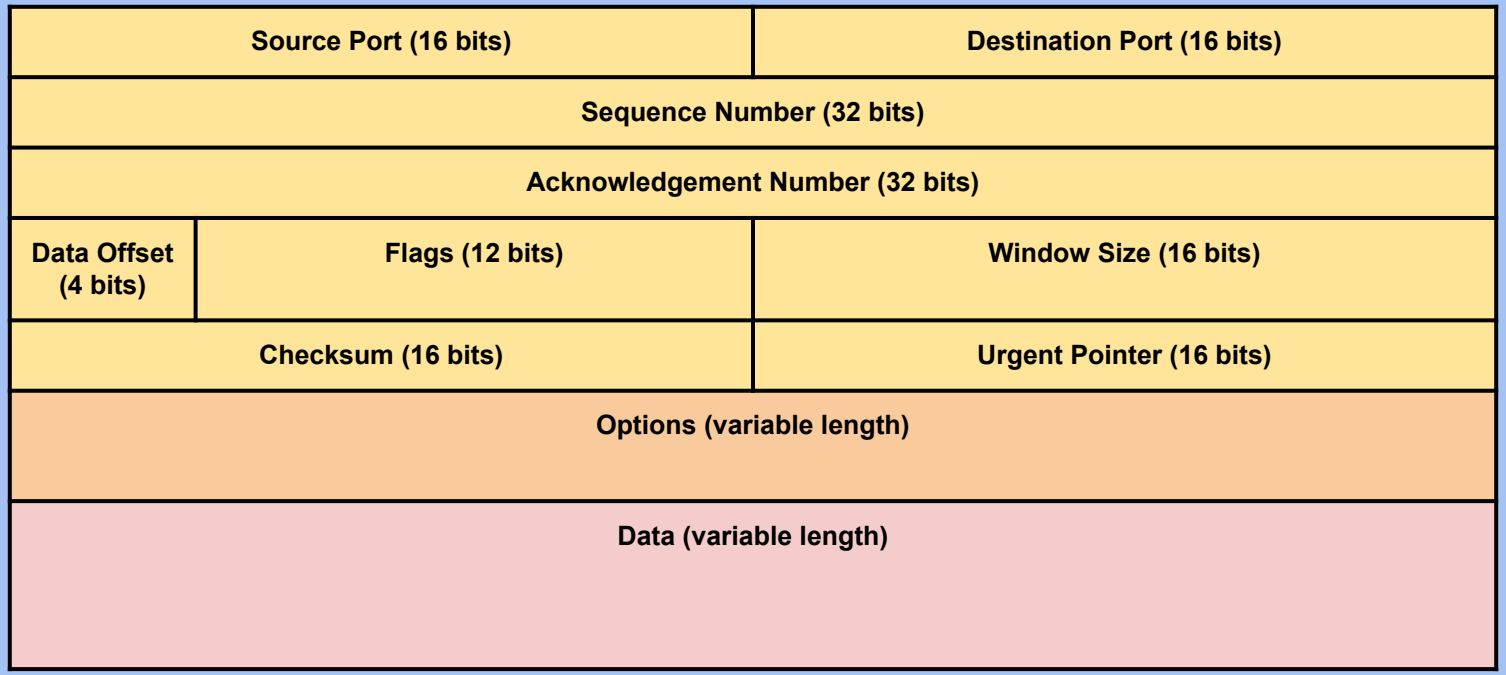
TCP: Ending/Aborting a Connection

- 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 Flags

- 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

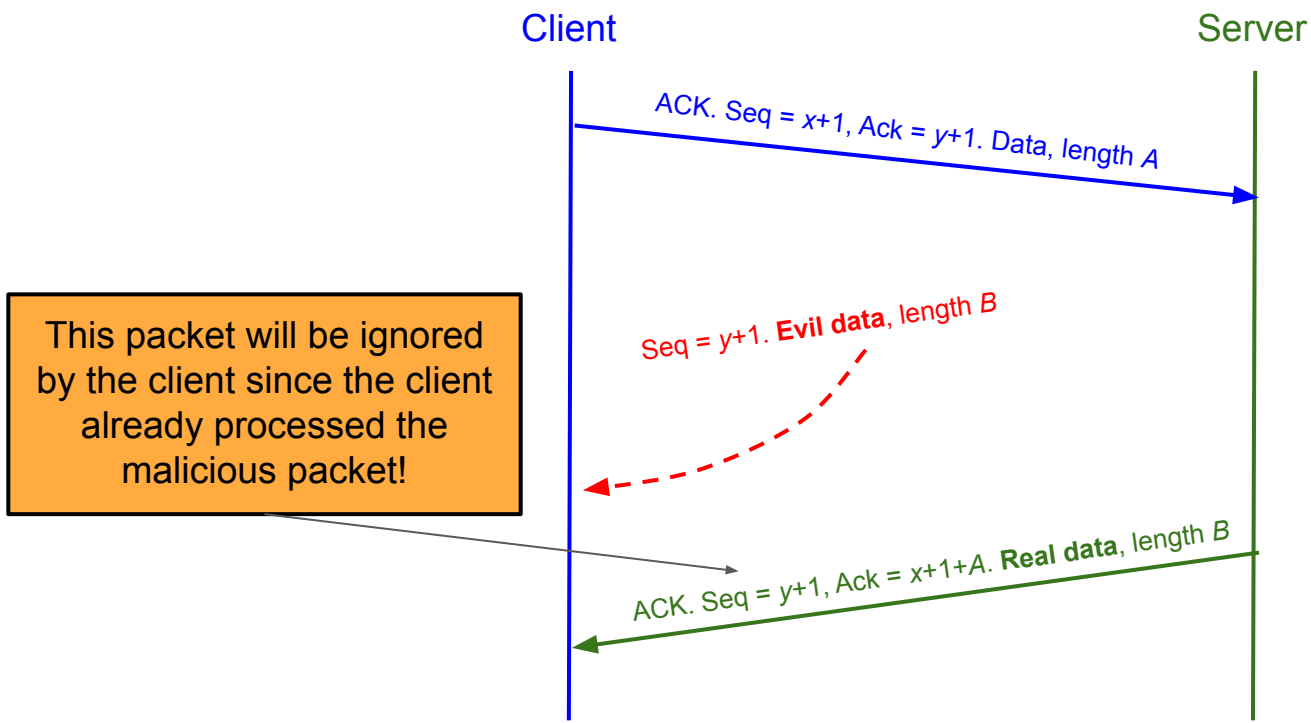
TCP Packet Structure



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
  - **RST injection**: Spoofing a RST packet to forcibly terminate a connection
    - Same requirements as packet injection, so easy for on-path and MITM attackers, but hard for off-path attackers
    - Often used in censorship scenarios to block access to sites

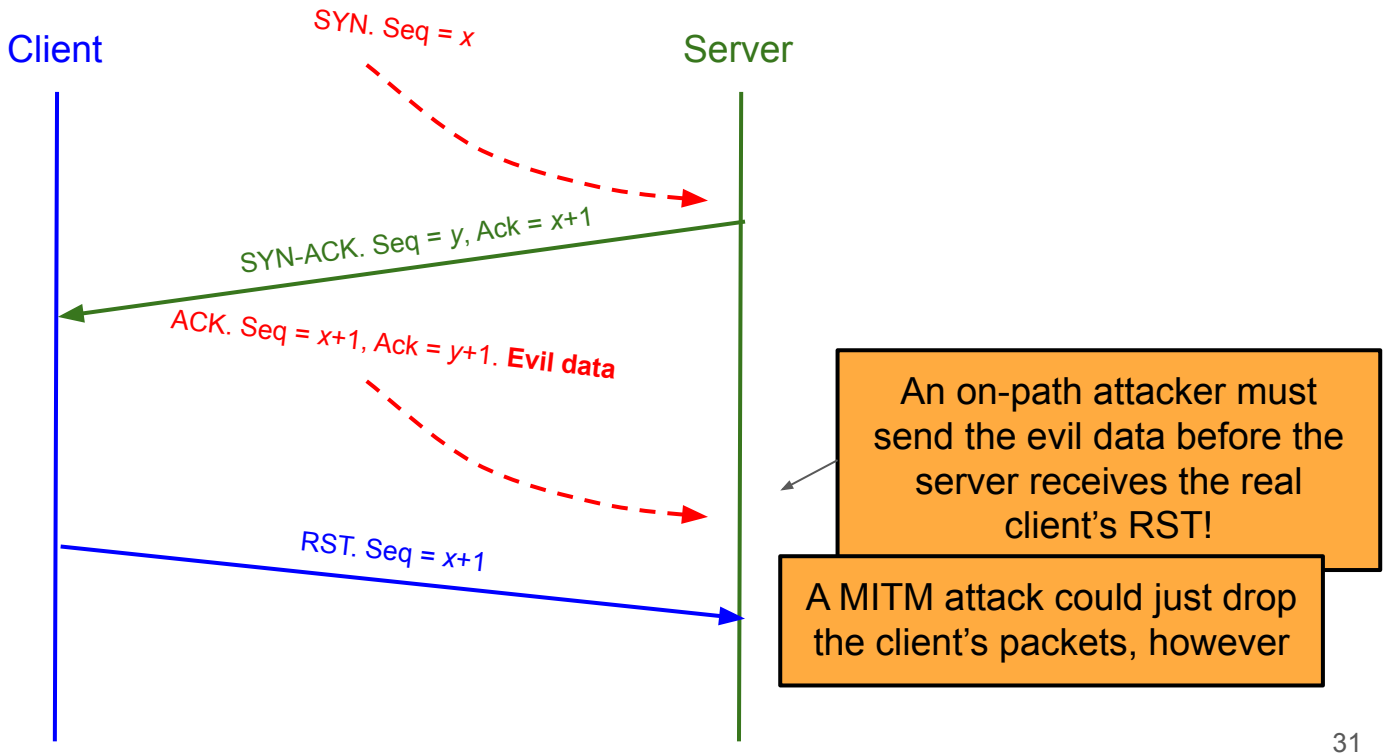
TCP Data Injection



TCP Attacks

- **TCP spoofing**: Spoofing a TCP connection to appear to come from another source IP address
  - Recall: IP packets can often be spoofed if the AS doesn't block source addresses
  - Need to know: Sequence number in the server's response SYN-ACK packet
  - Easy for MITM and on-path attackers, but off-path attackers must guess 32-bit sequence number (called **blind spoofing**, also considered difficult)
  - For on-path attackers, this is a race condition, since the real client will send a RST upon receiving the server's SYN-ACK!

TCP Spoofing



TCP Attacks

- 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!

User Datagram Protocol (UDP)

Textbook Chapter 30

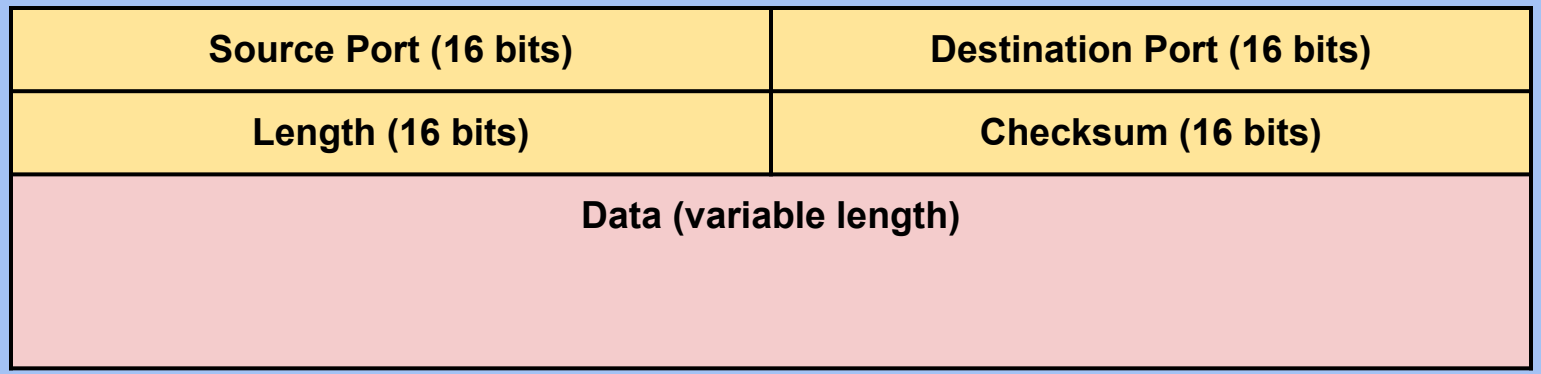
User Datagram Protocol (UDP)

- Provides a datagram abstraction
  - A message, sent in a single layer 3 packet (though layer 3 could fragment the packet)
  - Max size can be very large, but best to keep it as the maximum size of an unfragmented packet because of the unreliability of fragmentation
  - Applications break their data into datagrams, which are sent and received as a single unit
    - Contrast with TCP, where the application can use a bytestream abstraction
- No reliability or ordering guarantees, but adds ports
  - It still has *best effort* delivery, but a datagram should be received or not in the face of random corruption
- Much faster than TCP, since there is no 3-way handshake
  - Usually used by low-latency, high-speed applications where errors are okay (e.g. video streaming, games)

UDP Attacks

- No sequence numbers, so relatively easy to inject data into a connection or spoof connections
  - Higher layers must provide their own defenses against these attacks!

UDP Packet Structure



Summary: BGP, TCP, and UDP

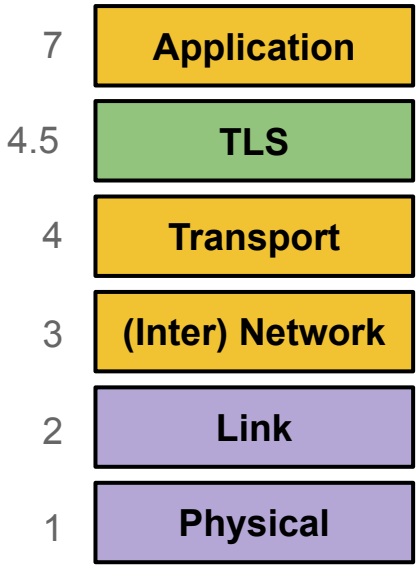
- Border Gateway Protocol (BGP): Routing packets
  - The Internet is made of smaller **autonomous systems** (**AS**)
  - Each AS broadcasts the shortest routes it knows of (dependent on the shortest routes of its neighbors and distance to neighbors)
- Transmission Control Protocol (TCP): Reliably sending packets
  - 3-way handshake: Client sends SYN, server sends SYN-ACK, client sends ACK
  - Provides reliability, ordering, and ports
  - Attack: TCP hijacking through data injection or RST injection
    - Blind attacks must guess the client's or server's sequence numbers
  - Attack: TCP spoofing by sending a spoofed SYN packet
    - Blind attacks must guess the server's sequence number
- User Datagram Protocol (UDP): Non-reliably sending packets
  - No reliability or ordering, only ports
  - Same injection and spoofing attacks as TCP, but easier

TLS

Textbook Chapter 31

TLS

- 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!
- Alternate version, Datagram TLS (dTLS) available for UDP but not commonly used



Today: Secure Internet Communication with TLS

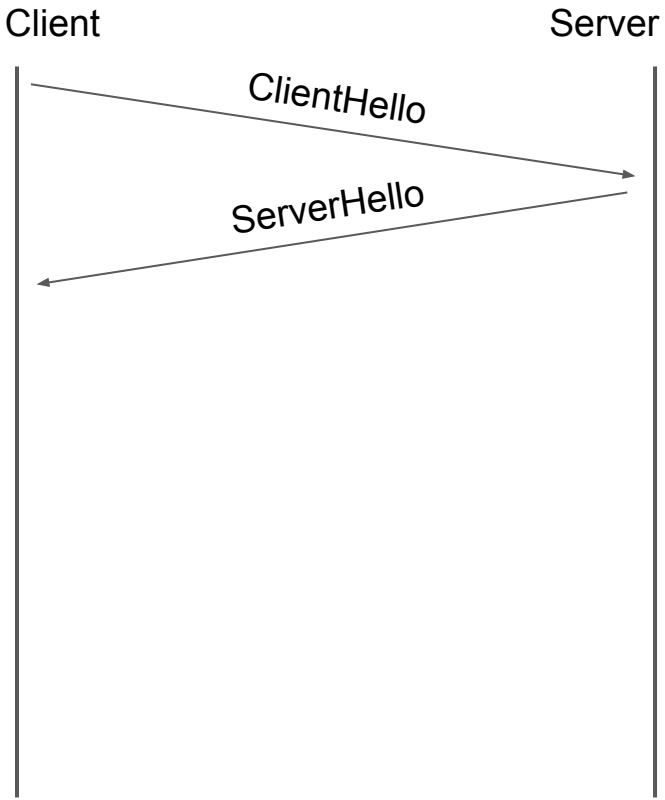
- Goals of TLS
  - **Confidentiality**: Ensure that attackers cannot read your traffic
  - **Integrity**: Ensure that attackers cannot tamper with your traffic
    - Prevent replay attacks
      - The attacker records encrypted traffic and then replays it to the server
      - Example: Replaying a packet 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

Textbook Chapter 31.1

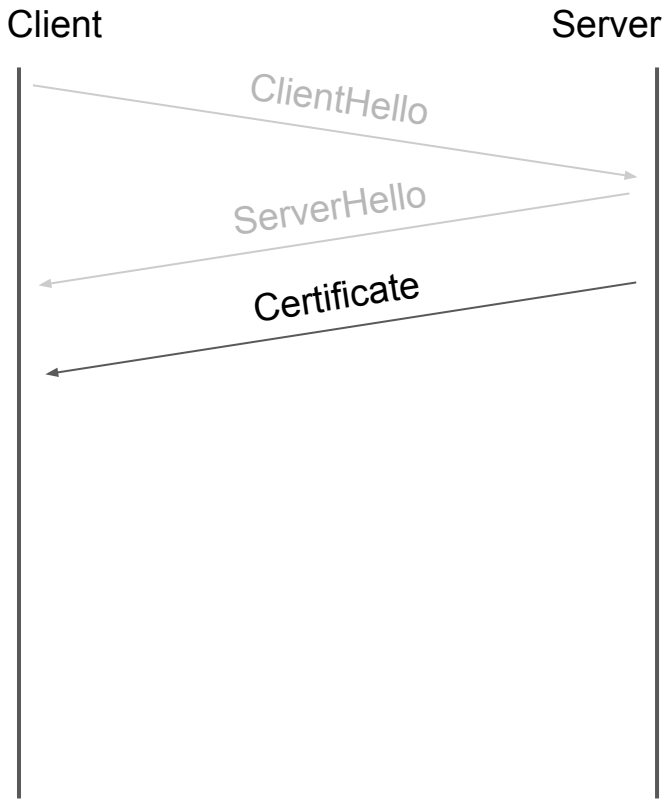
TLS Handshake Step 1: Exchange Hellos

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



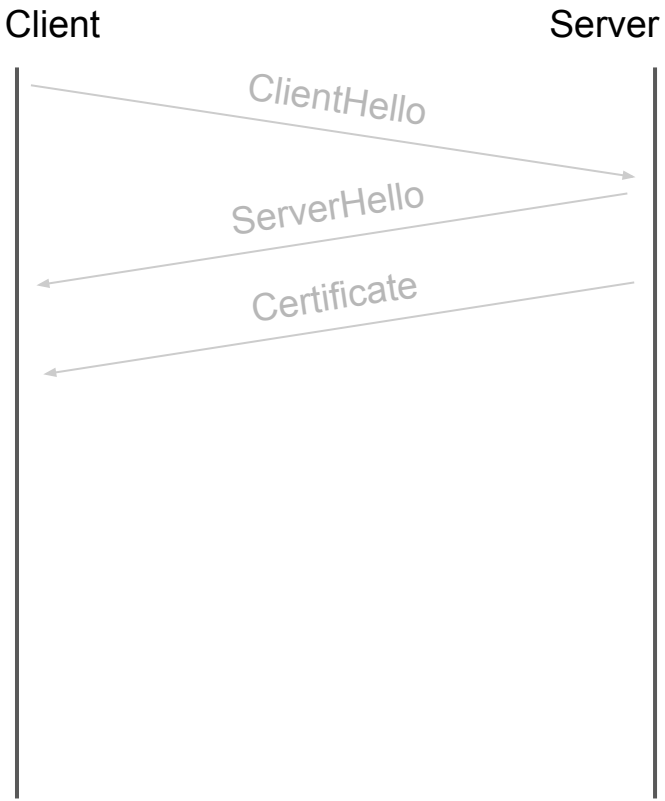
TLS Handshake Step 2: Certificate

- 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



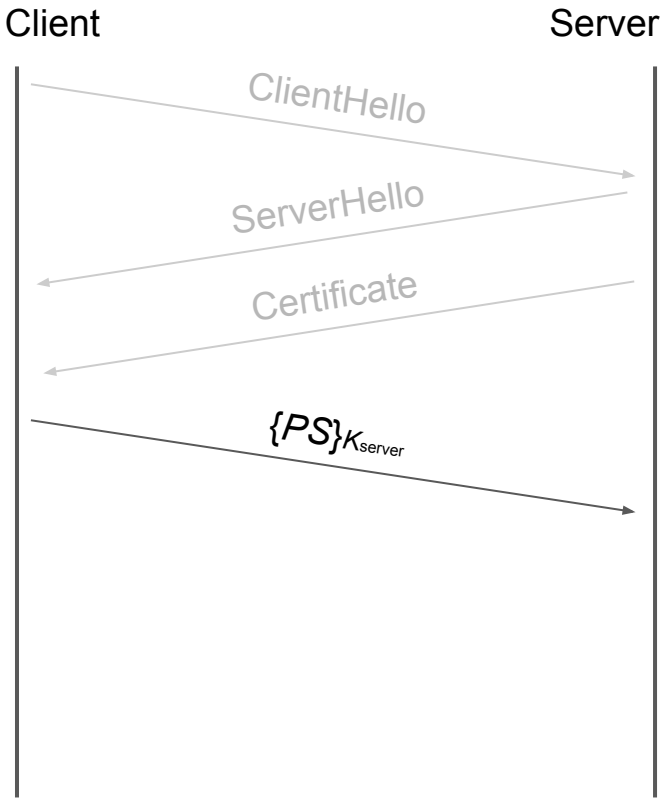
TLS Handshake Step 3: Premaster Secret

- 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)



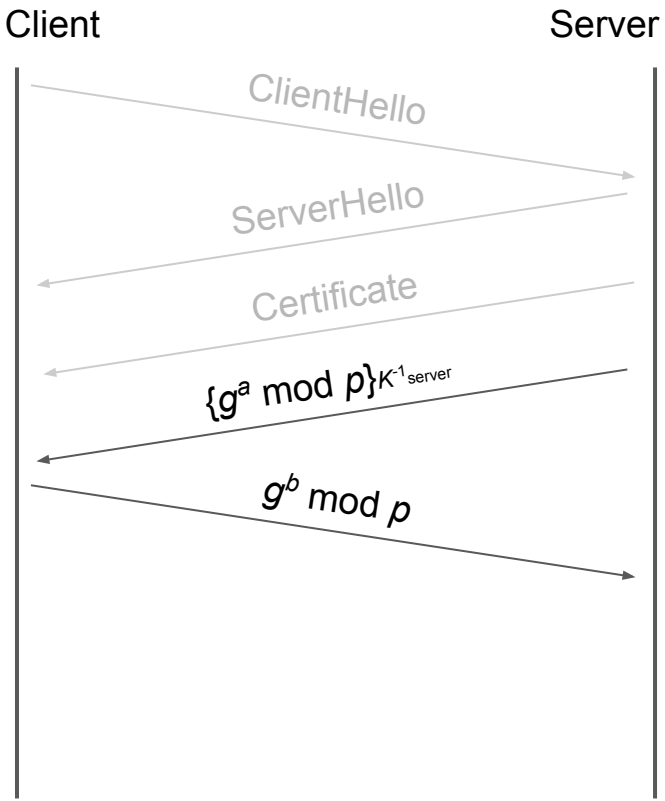
TLS Handshake Step 3: Premaster Secret (RSA)

- 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*)



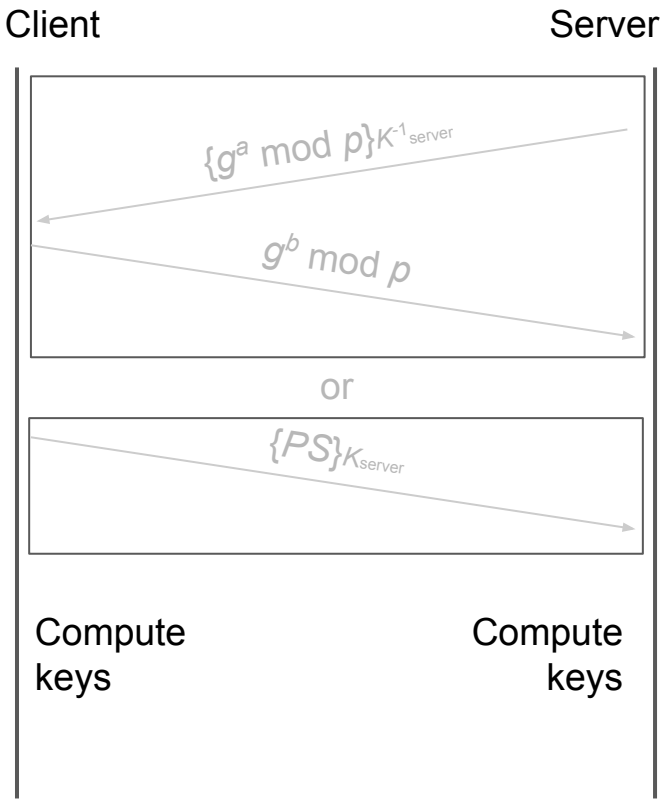
TLS Handshake Step 3: Premaster Secret (DHE)

- 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 $g^{ab} \bmod p$



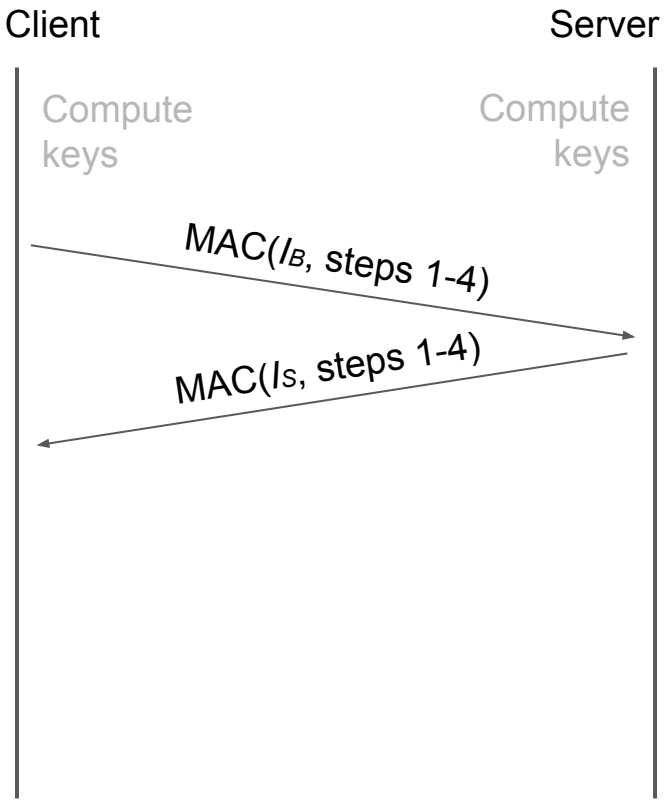
TLS Handshake Step 4: Derive Symmetric Keys

- The server and client each derive symmetric keys from R_B, R_S, 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
  - C_B: For encrypting client-to-server messages
  - C_S: For encrypting server-to-client messages
  - I_B: For MACing client-to-server messages
  - I_S: For MACing server-to-client messages
  - Note: Both client and server know all four keys



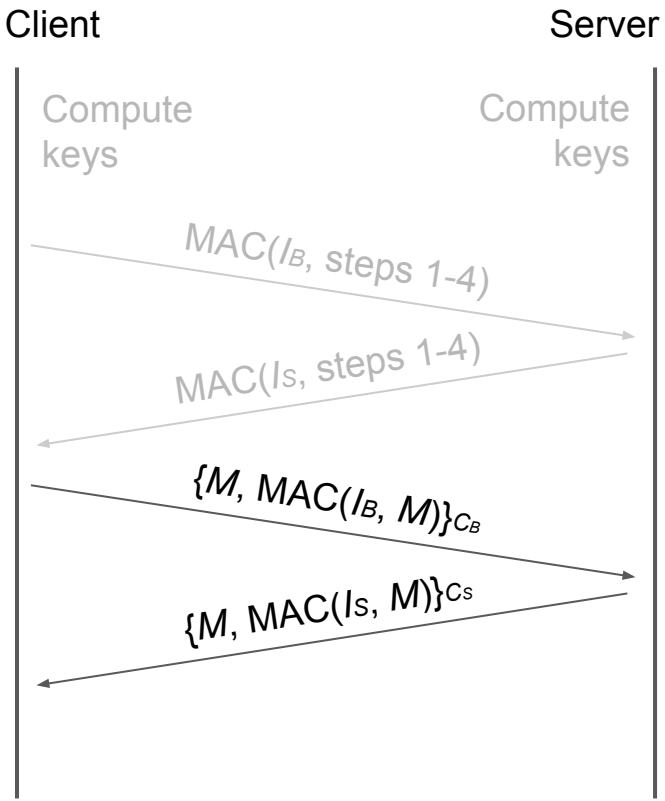
TLS Handshake Step 5: Exchange MACs

- 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
  - Not to be confused with MAC addresses
  - (Yes, since TLS was designed by cryptographers, our message authentication codes are once again called MACs, not MICs)



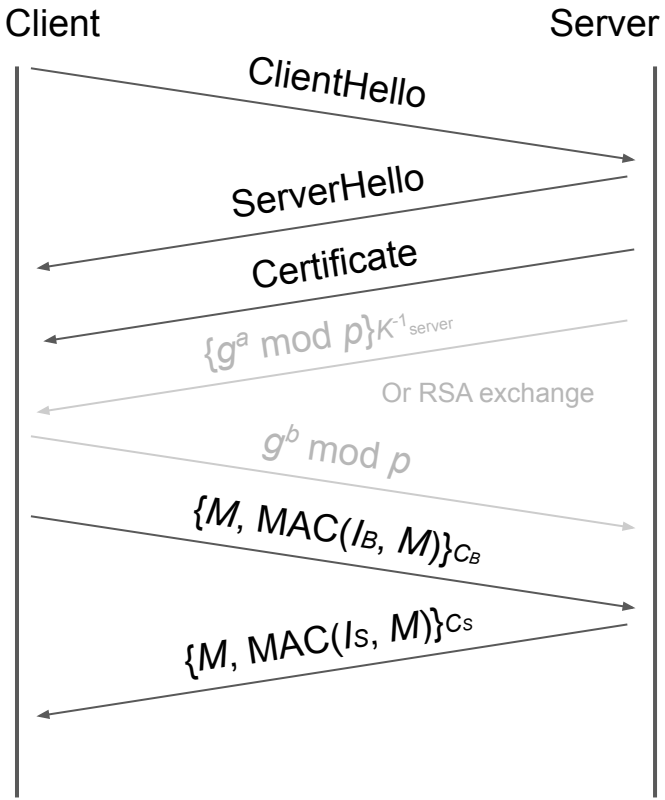
TLS Handshake Step 6: Send Messages

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



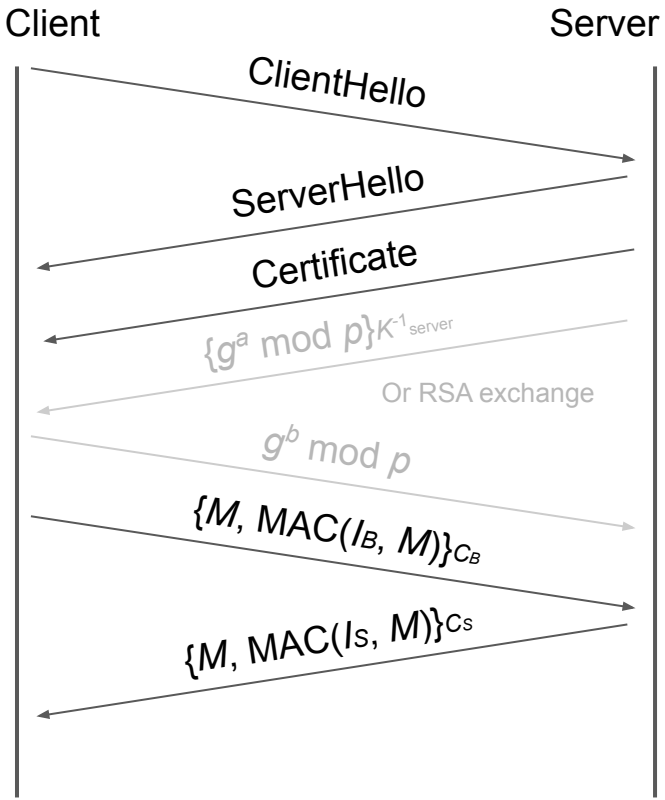
TLS: Talking to the Legitimate Server

- 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)



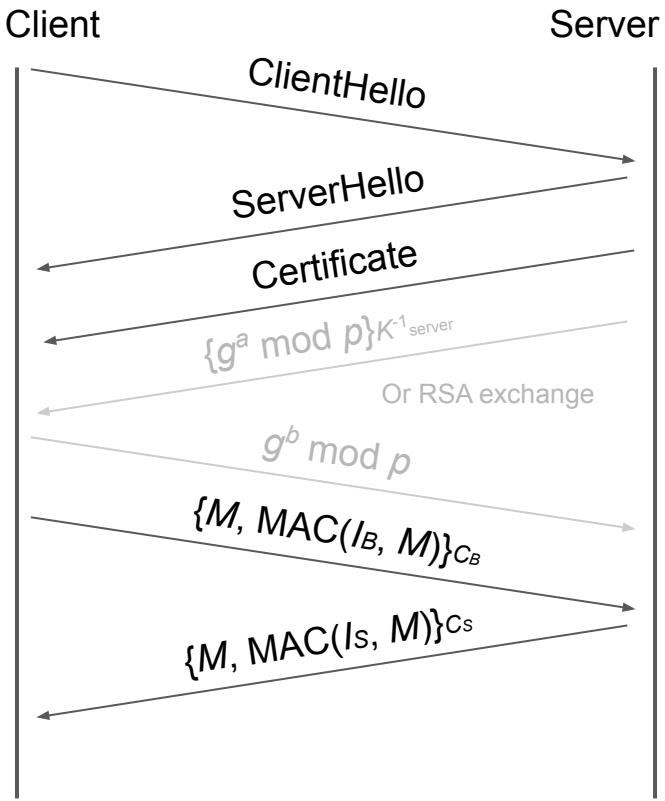
TLS: Securing Messages

- 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



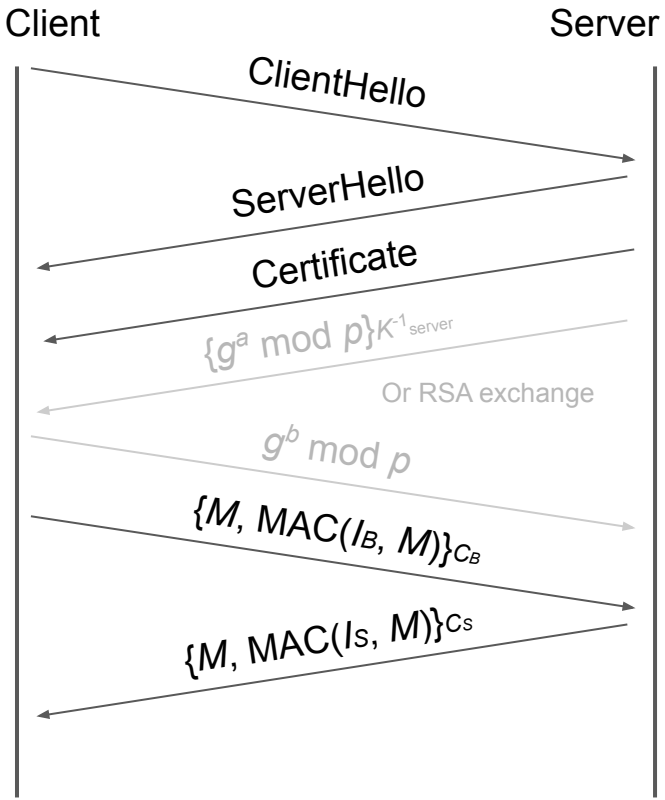
TLS: Replay Attacks

- How can we be sure that the attacker hasn't replayed old messages from a *past* TLS connection?
- Every handshake uses a different R_B and R_S
- The symmetric keys are derived from R_B and R_S
  - The symmetric keys are different for every connection



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

Textbook Chapter 31.1

Forward Secrecy

- Recall 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 R_B, R_S, 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
  - Note: Because DH resists passive adversaries, even if the attacker gets the server's private key they must act as a MITM to intercept traffic!

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 $g^b \bmod p$ in ClientHello
  - If the server agrees to use DHE, the server sends $g^a \bmod p$ (with signature) in ServerHello
  - Potentially saves two messages later in the handshake
- Only supports AEAD mode encryption
  - Recall AEAD (authenticated encryption with additional data): a block cipher mode that guarantees confidentiality and integrity at the same time
  - Eliminates attacks associated with the insecure MAC-then-encrypt pattern

TLS in Practice

Textbook Chapter 31.3

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

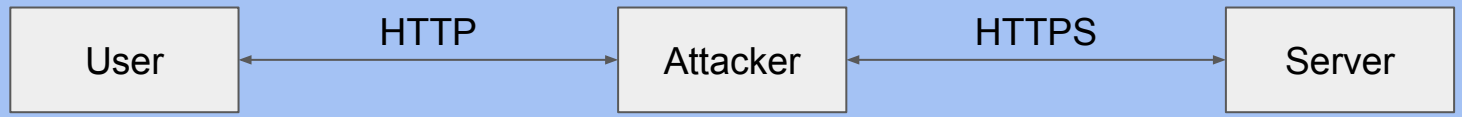
- **Availability**: Keeping the connection open in the face of attackers
- An attacker can stop a TLS connection
  - MITM can drop encrypted TLS packets
  - On-path attacker can still do RST injection to abort the underlying TCP connection
- Result: A TLS connection can still be censored
  - The censor can block TLS connections
  - TLS 1.3 supports “Encrypted SNI”: The server’s name is requested after the key exchange itself
    - But networks that want to enforce a lack of anonymity (censors, businesses, etc.) can recognize and terminate those connections
    - Networks can also use the IP address to guide censorship decisions

TLS for Applications

- Recall 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

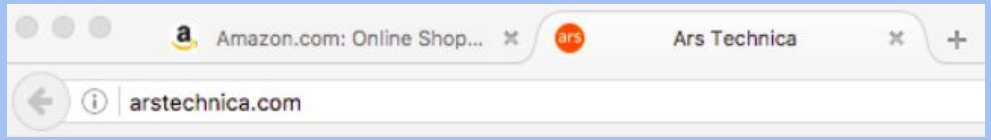
SSL Stripping Attacks

- 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

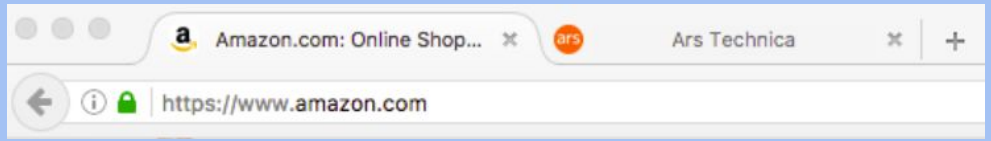


TLS in Browsers

- 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



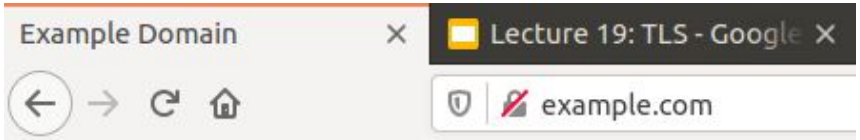
This website uses HTTPS: lock icon

TLS in Browsers

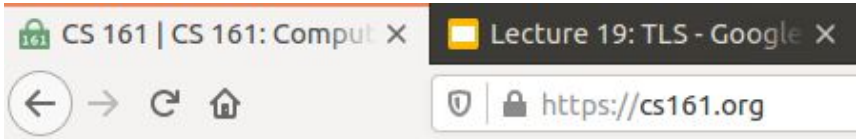
- What users think the lock icon means
  - This website is trustworthy, no matter where the lock icon actually appears
- Attack: The attacker adds their own lock icon somewhere on the page
  - The user thinks they're using TLS, but actually is not using TLS
- Attack: The user might be communicating with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

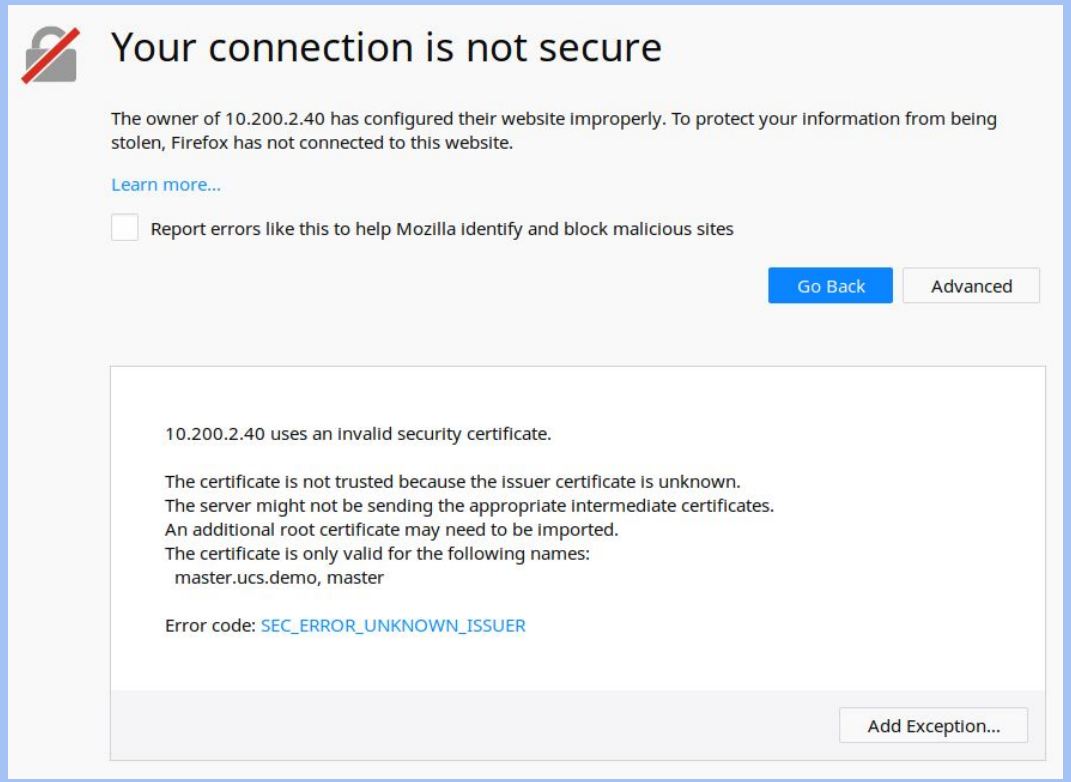
- Consider TLS with Diffie-Hellman
  - An attacker who learns the DHE secret *a* can derive the PS g^{ab} mod *p* (recall g^b mod *p* is sent over the channel)
  - An attacker who knows the PS can derive the symmetric keys (recall R_C and R_S are sent over the channel)
- Consider using a PRNG to generate all random values
  - Includes the server DHE secret *a* and the client DHE secret *b*
- What if the PRNG is sabotaged and doesn't have rollback resistance?
  - Example of sabotage: Dual_EC DRBG with knowledge of the secret used to create the generator
  - Example of sabotage: ANSI X9.31: An AES-based PRNG with a secret key
- Attack: See subsequent PRNG output and work backwards to learn the DHE secret

TLS Trust Issues: Certificate Authorities

Recall: Certificates in TLS

- The server sends its certificate
  - Certificate: The server's domain name and public key, signed by a certificate authority
- The browser verifies the server's certificate
  - The browser checks the domain name in the URL matches the domain name in the certificate
  - The certificate authority's public key is hardwired into the browser (trust anchor)
  - The browser uses the CA's public key to verify the signature
- If the certificate is verified, the browser now knows the server's public key

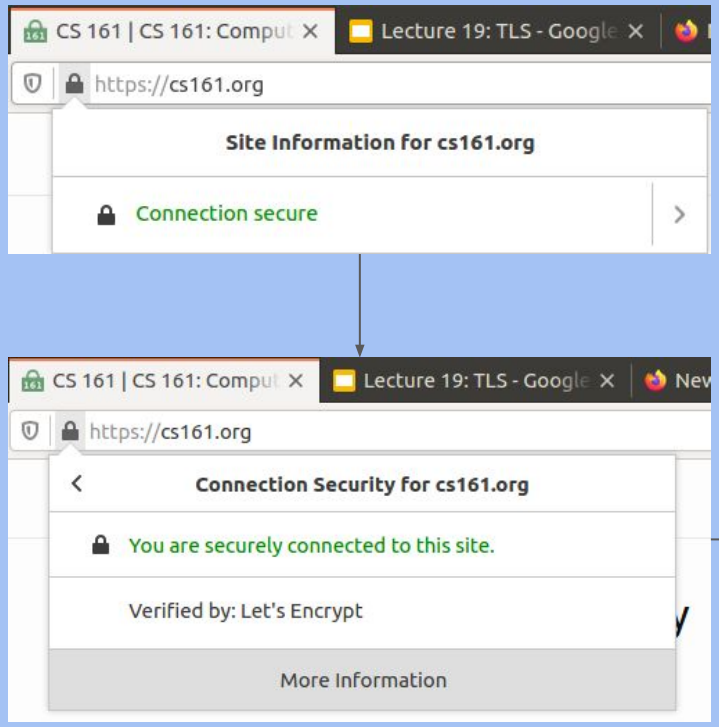
Issues: Unknown Certificate Authority

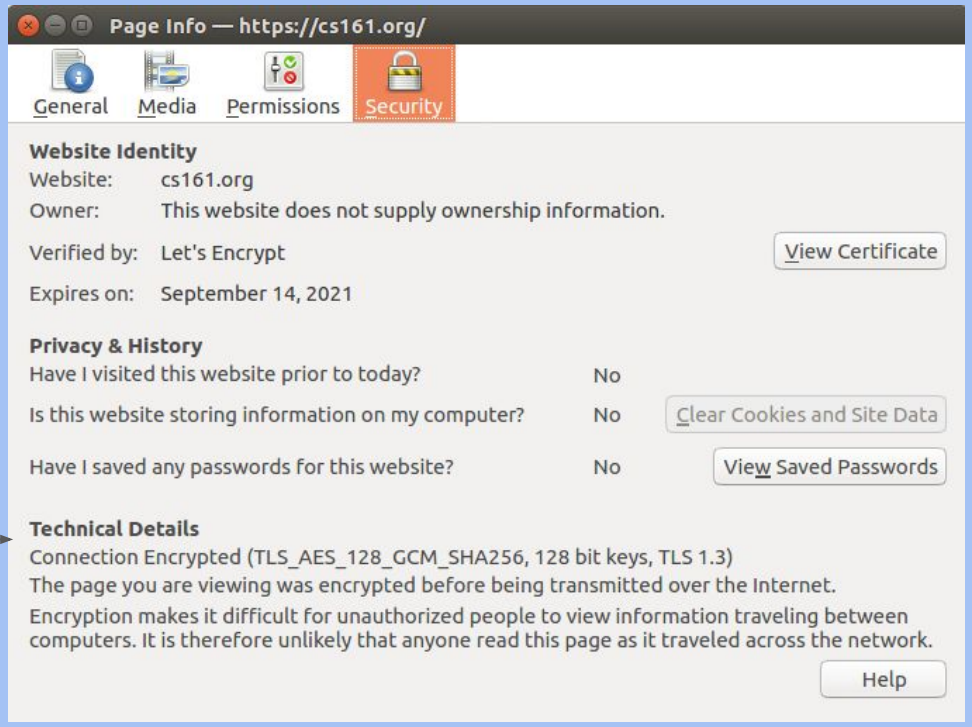


Issues: Unknown Certificate Authority

- What if the browser doesn't have the certificate authority's public key for verification?
- Warn the user that the website is not verified
  - The TLS connection can still proceed, but there is no guarantee that the user is talking to the legitimate server
- What if the user is not talking to the legitimate server?
  - No more end-to-end security
  - An attacker can read and modify messages
  - An attacker can impersonate the server

Verifying Certificates





Verifying Certificates

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 expiration date, but they often don't expire for years
- Solution: Certificate revocation lists
  - The CA occasionally sends out lists of certificates that are no longer valid
  - The browser occasionally downloads the lists
- Solution: Online Certificate Status Protocol (OCSP)
  - The browser queries the CA whether a given certificate is still valid
  - The CA responds either "good" or "revoked," signed with the CA's private key

Issues: Trust Anchors

- How many certificate authorities do we need to implicitly trust?
  - Modern browsers implicitly trust 100–200 root certificate authorities
- A CA might issue a malicious certificate (e.g. stating that attacker's public key belongs to Google) because:
  - The CA is hacked
  - An attacker pays the CA to issue a malicious certificate

Issues: Trust Anchors

COMPUTERWORLD

Link

Solo Iranian hacker takes credit for Comodo certificate attack

Gregg Keizer *March 27, 2011*

Security researchers split on whether 'ComodoHacker' is the real deal

A solo Iranian hacker on Saturday claimed responsibility for stealing multiple SSL certificates belonging to some of the Web's biggest sites, including Google, Microsoft, Skype and Yahoo.

Early reaction from security experts was mixed, with some believing the hacker's claim, while others were dubious.

Takeaway: Trust certificate authorities can be compromised by hackers

Issues: Trust Anchors

c|net

Link

Fraudulent Google certificate points to Internet attack

Elinor Mills *August 29, 2011*

Is Iran behind a fraudulent Google.com digital certificate? The situation is similar to one that happened in March in which spoofed certificates were traced back to Iran.

A Dutch company appears to have issued a digital certificate for Google.com to someone other than Google, who may be using it to try to re-direct traffic of users based in Iran.

Yesterday, someone reported on a Google support site that when attempting to log in to Gmail the browser issued a warning for the digital certificate used as proof that the site is legitimate, according to this thread on a Google support forum site.

Takeaway: Trust certificate authorities can be compromised by hackers

Issues: Trust Anchors

threatpost

Link

Final Report on DigiNotar Hack Shows Total Compromise of CA Servers

Dennis Fisher *October 31, 2012*

The attacker who penetrated the Dutch CA DigiNotar last year had complete control of all eight of the company's certificate-issuing servers during the operation and he may also have issued some rogue certificates that have not yet been identified.

Takeaway: Trust certificate authorities can be compromised by hackers

Issues: Trust Anchors

techdirt

Link

Evidence Suggests DigiNotar, Who Issued Fraudulent Google Certificate, Was Hacked *Years* Ago

Mike Masnick *August 30, 2011*

The big news in the security world, obviously, is the fact that a fraudulent Google certificate made its way out into the wild, apparently targeting internet users in Iran. The Dutch company DigiNotar has put out a statement saying that it discovered a breach back on July 19th during a security audit, and that fraudulent certificates were generated for "several dozen" websites. The only one known to have gotten out into the wild is the Google one.

Takeaway: Trust certificate authorities can be compromised by hackers

Issues: Trust Anchors

- DigiNotar: A certificate authority that was hacked
  - All web browsers removed DigiNotar from the list of trusted CAs
- WoSign: An untrustworthy certificate authority
  - Also removed by all browsers
  - A user who controls **`nweaver.github.com`** can create certificates for any subdomain of **`github.com`**
- **Takeaway**: It is hard to implicitly trust the root CAs (trust anchors) in TLS
- **Takeaway**: TLS CA ecosystem is largely secured through human relationships
  - The browser vendors will pronounce a “death sentence” on bad certificate authorities

Solving Trust Issues

- **Certificate pinning**: The browser restricts which CAs are allowed to issue a certificate for each website
  - Example: Only the Google CA is allowed to sign certificates for Google websites
  - Now creating a fake certificate for a specific website requires attacking a particular CA
- **Certificate transparency**: Public logs provided by CAs
  - Specifics are out of scope
  - High-level idea: Use hash chains to keep a record of all issued certificates
  - The server can tell the browser to only accept certificates from CAs implementing transparency
  - The server can then make sure that a CA never screws up

Solving Trust Issues

- Other solutions implementing to “trust but verify” the certificate you received
  - EFF’s SSL Observatory: Check against certificates seen by other dedicated computers, called “observatories,” placed around the Internet
  - ICSI’s Certificate Notary: Check against certificates used in common Internet traffic, by tapping into common Internet channels
  - Specifics of these protocols are out of scope

Certificate Authority Example: Let's Encrypt

- TLS requires every website to obtain and maintain certificates
  - Cost overhead: Certificates might cost money
  - Some management overhead involved
- **Let's Encrypt**: The world's largest certificate authority
  - Issues certificates for free
  - Tries to make obtaining certificates as easy as possible:
    Uses the ACME protocol to validate sites
  - Certificates only last for a short period of time (*forcing sites to automate renewing them*)
- Steps of issuing a certificate (can all be automated with a script)
  - The server requests a certificate
  - Let's Encrypt gives the server a file and tells the server to upload the file at a specific location
  - The server uploads the file to the website
  - Let's Encrypt verifies that the file has appeared on the website (thus verifying the server's identity) and issues the certificate to the server

Using Let's Encrypt in Go

- It is literally just a 2-liner to use Let's Encrypt with Go to turn an HTTP Go server into HTTPS:

```
import "golang.org/x/crypto/acme/autocert"

...
   log.Fatal(http.Serve(autocert.NewListener("example.com"), handler))
```

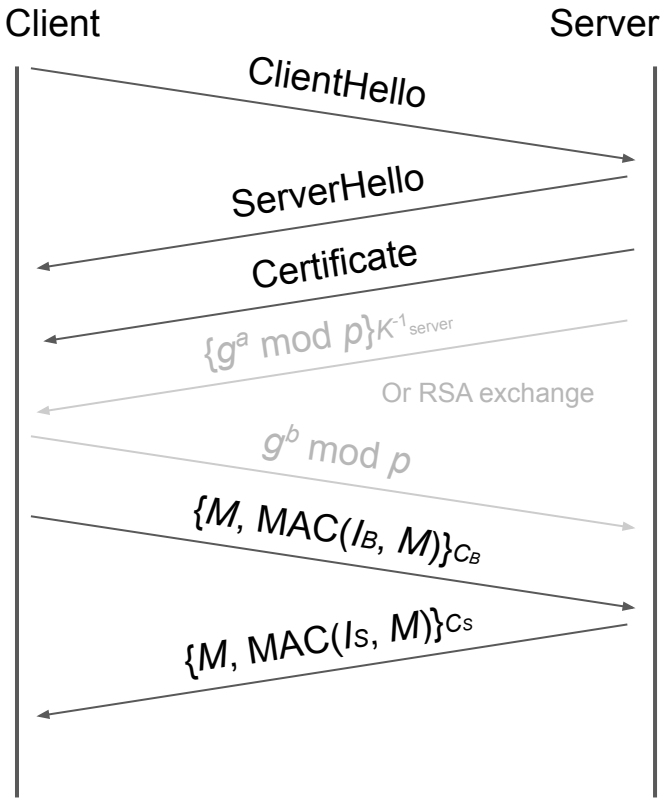
- **Takeaway**: Enabling HTTPS in web servers today is often very simple!

TLS Today

- Between Let's Encrypt and browser changes, TLS is no longer something special
  - It used to require a fair bit of effort and non-trivial money to set up!
- Instead, TLS is the default
- Biggest limitation with HTTPS:
  - A page loaded over HTTPS can *only* include content from other HTTPS pages: no items from HTTP pages
  - Advertisement networks were the biggest bottleneck in transitioning the web to almost universal support of HTTPS

TLS: Summary

- TLS Handshake
  - Nonces make every handshake different (prevents replay attacks across connections)
  - Certificate proves server's public key
  - RSA or DHE proves that the server owns the private key
  - RSA or DHE helps client and server agree on a shared secret key
  - MAC exchange ensures no one tampered with the handshake
  - Messages are sent with symmetric encryption and MACs
  - Record numbers prevent replay attacks within a connection



TLS: Summary

- Security properties
  - DHE TLS: Forward secrecy
  - RSA TLS: No forward secrecy
  - End-to-end security: Secure even if all intermediate parties are malicious
  - Not anonymous: Attackers can determine who you're talking to
  - No availability: Connections can be dropped or censored
- Can be used by the application layer (e.g. HTTPS)
- Trusting certificate authorities can be hard