TLS

CS 161 Fall 2022 - Lecture 19

Last Time: TCP and UDP

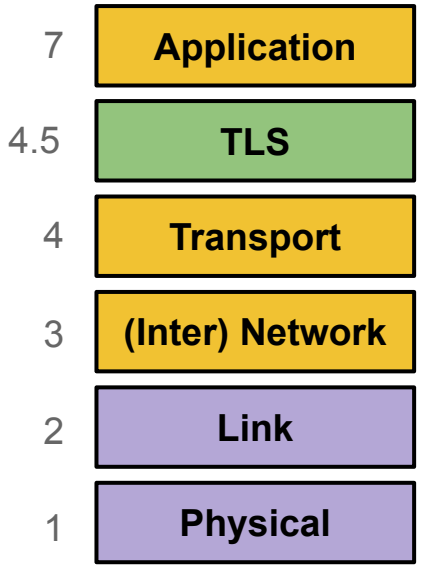
- Transmission Control Protocol (TCP): Reliably sending packets
  - 3-way handshake: Client sends SYN, server sends SYN-ACK, client sends ACK
  - Provides reliability, ordering, and ports
  - Attack: TCP hijacking through data injection or RST injection
    - Blind attacks must guess the client's or 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!



Today: Secure Internet Communication with TLS

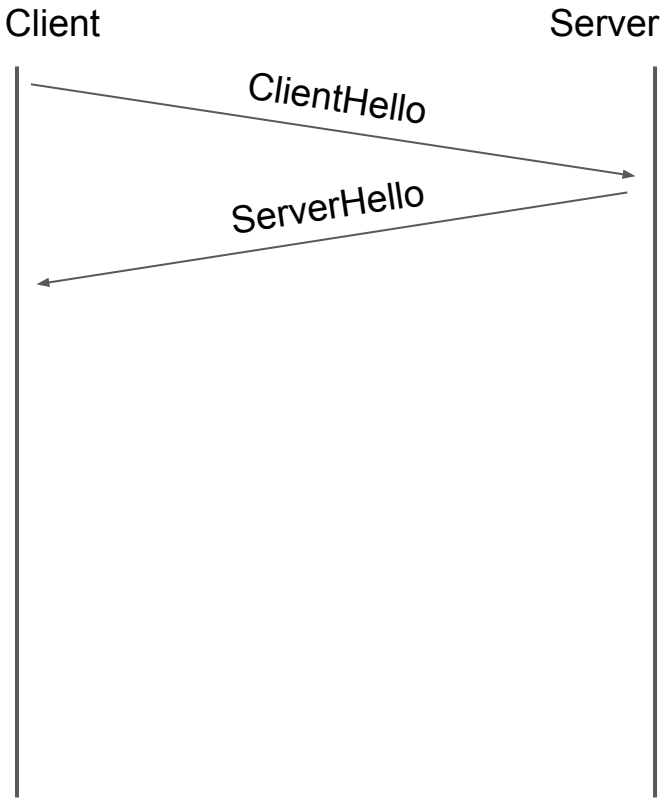
- Goals of TLS
  - **Confidentiality**: Ensure that attackers cannot read your traffic
  - **Integrity**: Ensure that attackers cannot tamper with your traffic
    - Prevent **replay attacks**
      - The attacker records encrypted traffic and then replays it to the server
      - Example: Replaying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

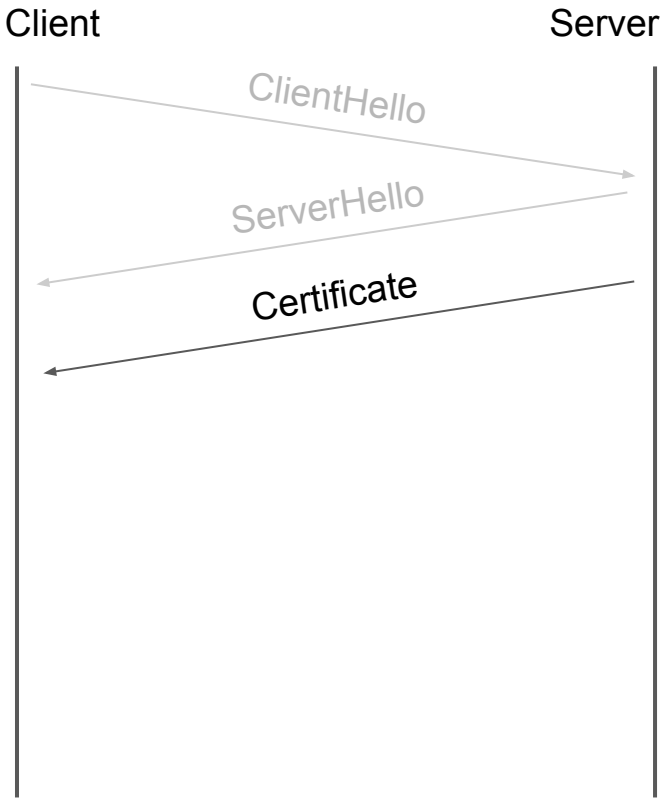
TLS Handshake Step 1: Exchange Hellos

- 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



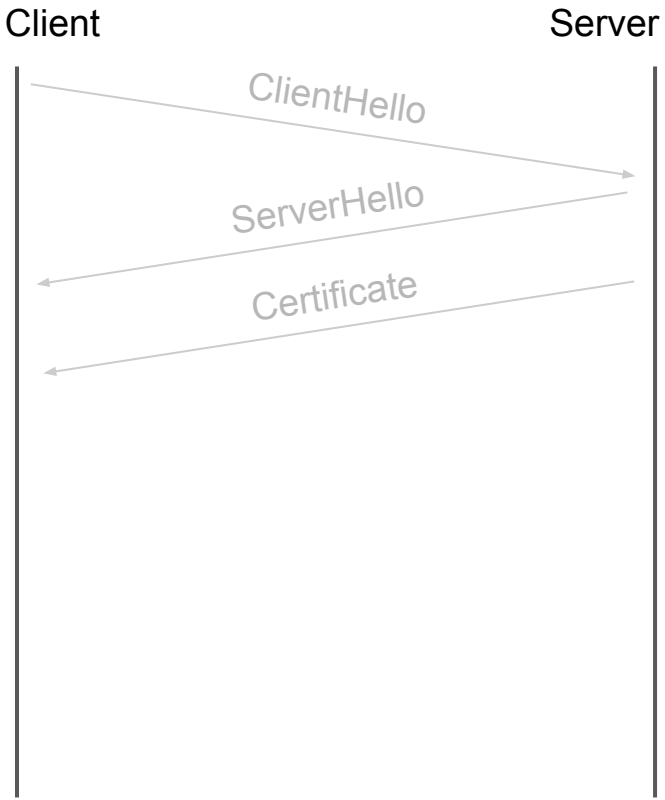
TLS Handshake Step 2: Certificate

- 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



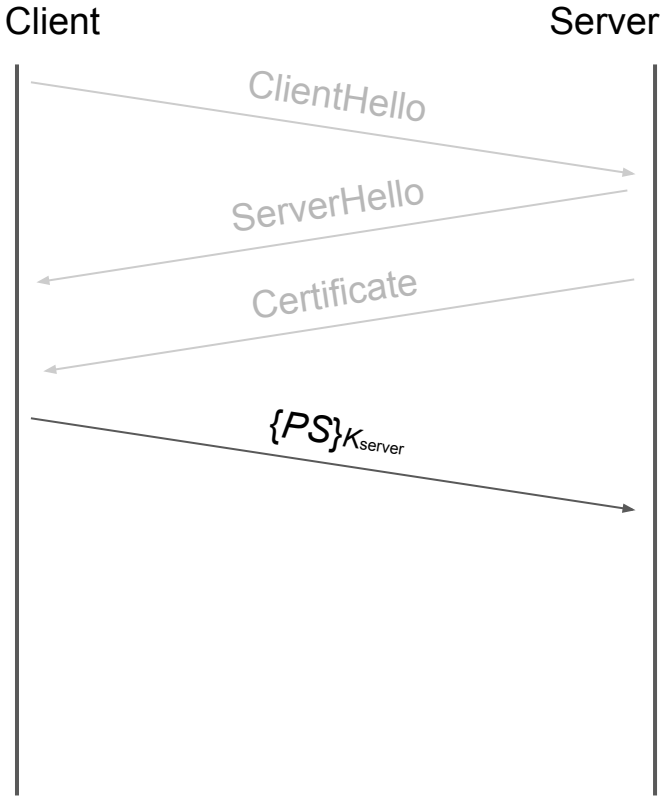
TLS Handshake Step 3: Premaster Secret

- 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)



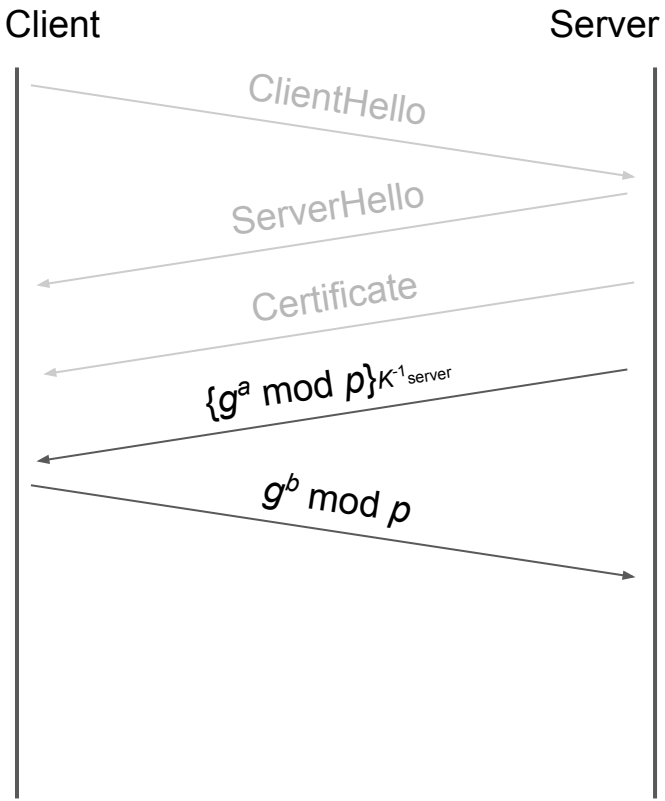
TLS Handshake Step 3: Premaster Secret (RSA)

- 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*)



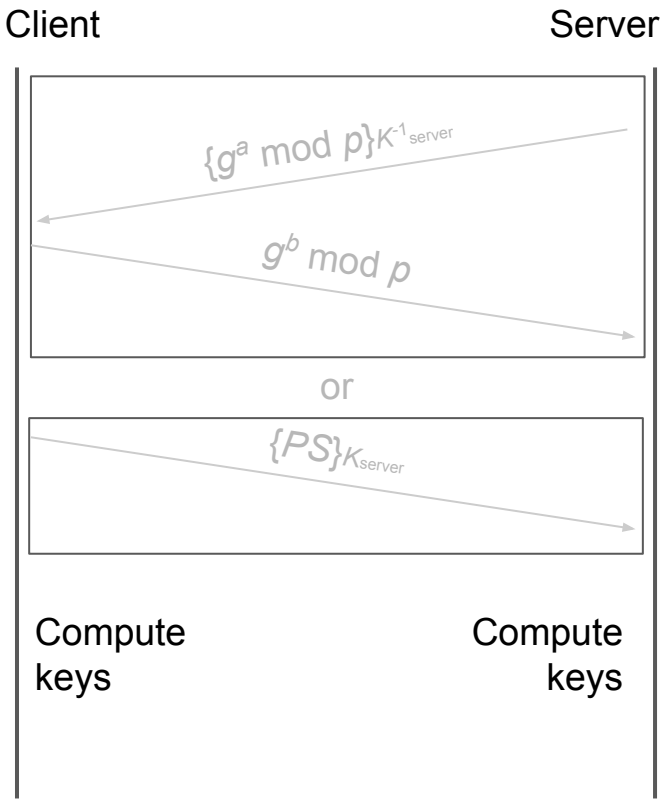
TLS Handshake Step 3: Premaster Secret (DHE)

- 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 $g^{ab} \bmod p$



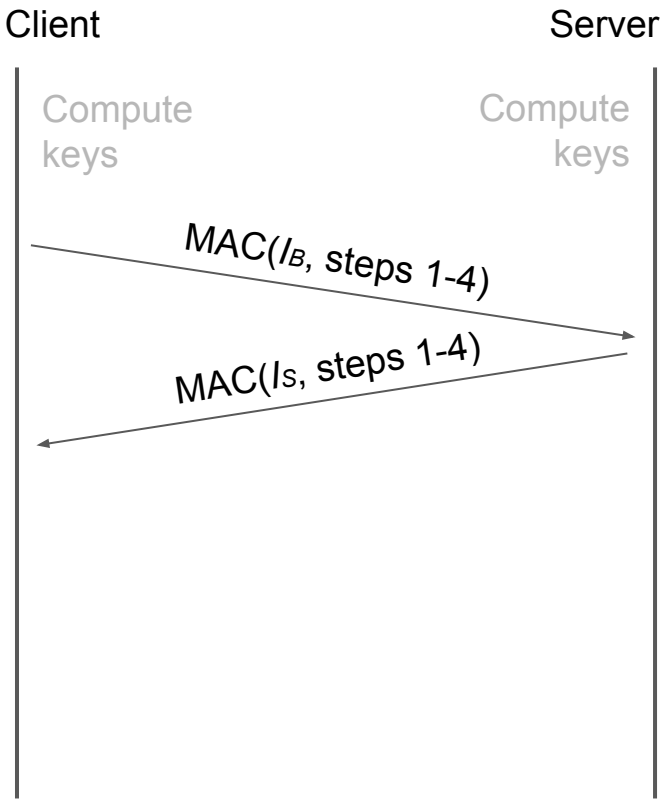
TLS Handshake Step 4: Derive Symmetric Keys

- 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



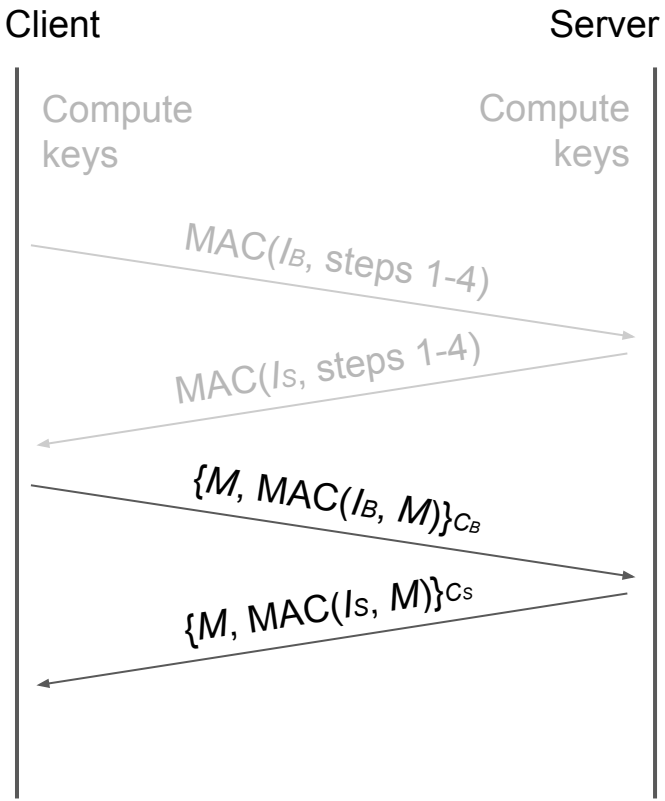
TLS Handshake Step 5: Exchange MACs

- 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



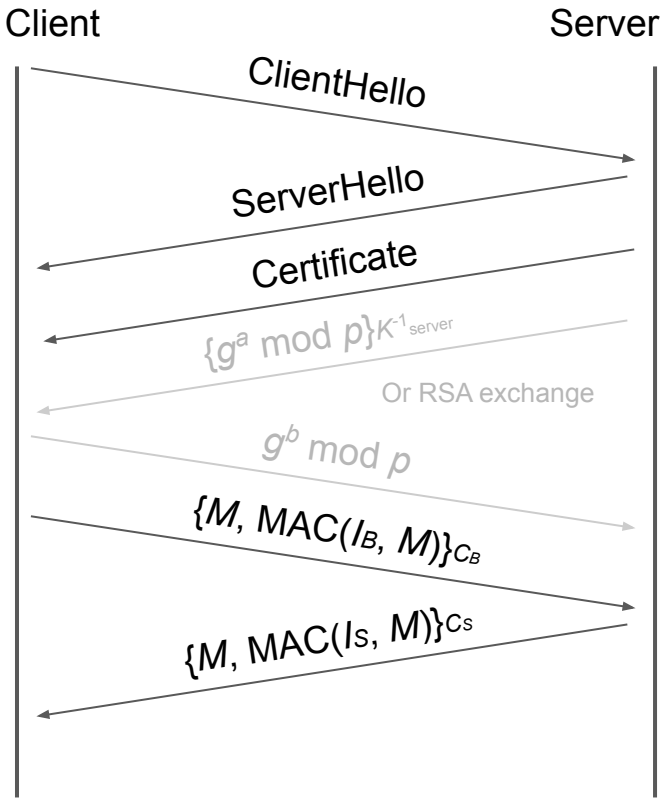
TLS Handshake Step 6: Send Messages

- 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



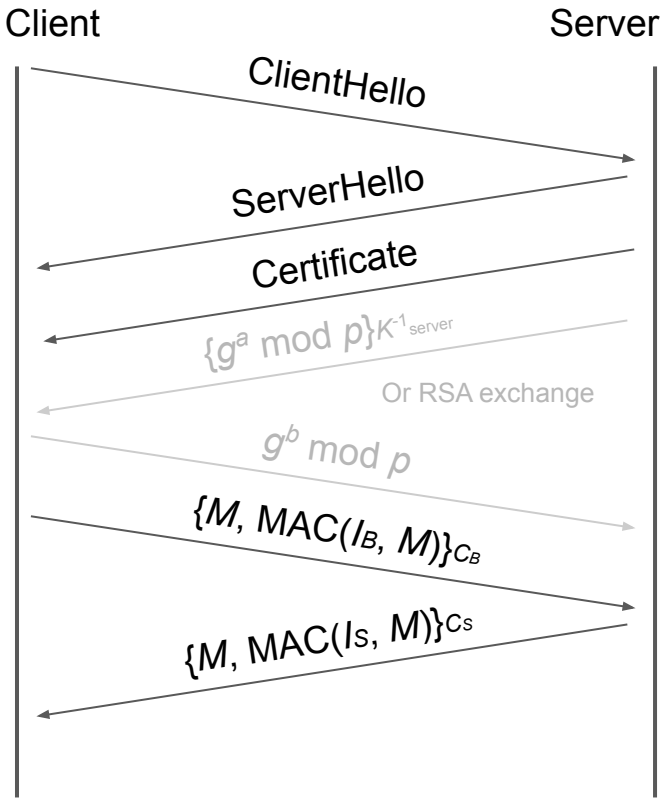
TLS: Talking to the Legitimate Server

- 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)



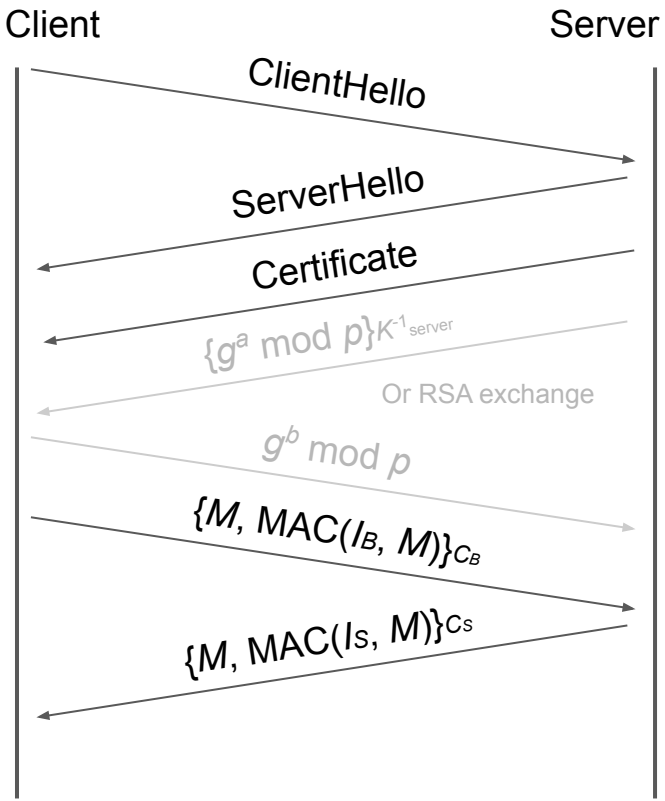
TLS: Securing Messages

- 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



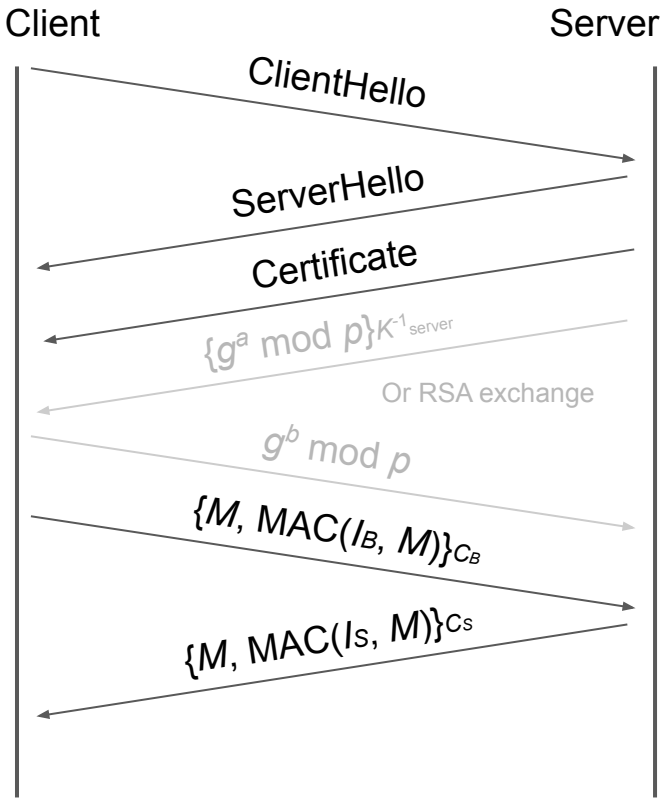
TLS: Replay Attacks

- 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

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

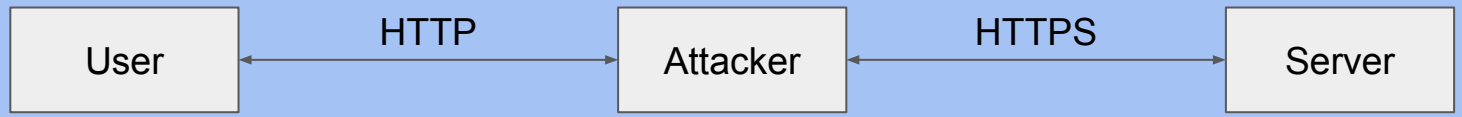
- **Availability**: Keeping the connection open in the face of attackers
- An attacker can stop a TLS connection
  - MITM can drop encrypted TLS packets
  - On-path attacker can still do RST injection to abort the underlying TCP connection
- Result: A TLS connection can still be censored
  - The censor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

TLS 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

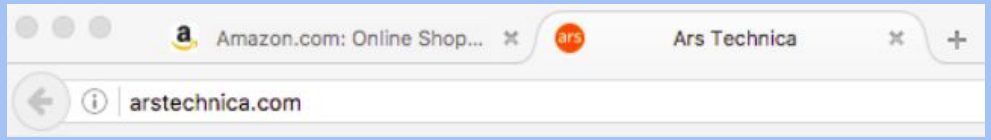
SSL Stripping Attacks

- 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

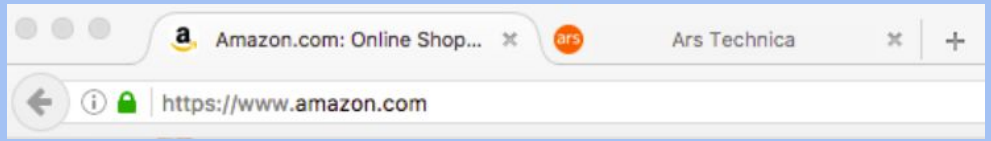


TLS in Browsers

- 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



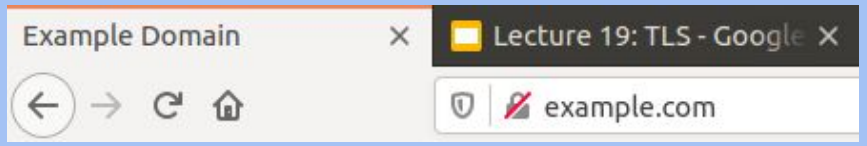
This website uses HTTPS: lock icon

TLS in Browsers

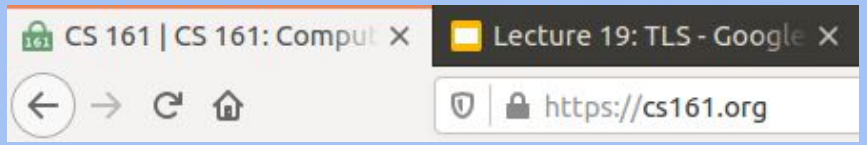
- What users think the lock icon means
  - This website is trustworthy, no matter where the lock icon actually appears
- Attack: The attacker adds their own lock icon somewhere on the page
  - The user thinks they're using TLS, but actually is not using TLS
- Attack: The user might be communicating with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

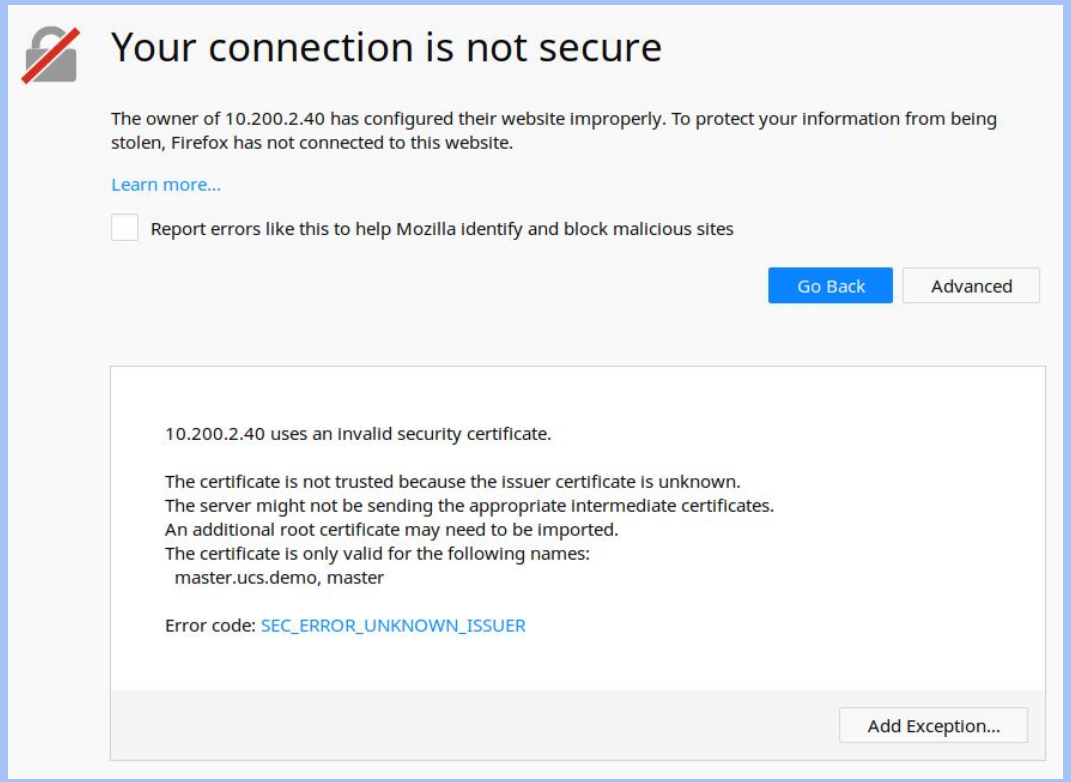
- Consider TLS with Diffie-Hellman
  - An attacker who learns the DHE secret *a* can derive the PS g^{ab} mod *p* (recall g^b mod *p* is sent over the channel)
  - An attacker who knows the PS can derive the symmetric keys (recall R_C and R_S are sent over the channel)
- Consider using a PRNG to generate 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

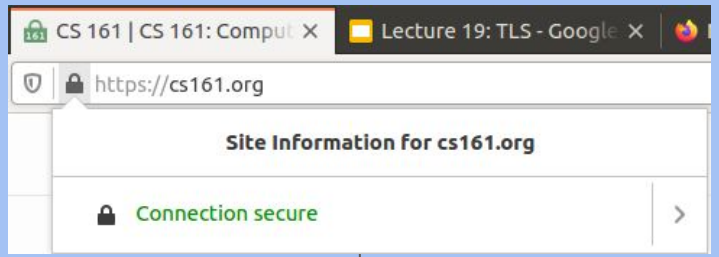
Issues: Unknown Certificate Authority

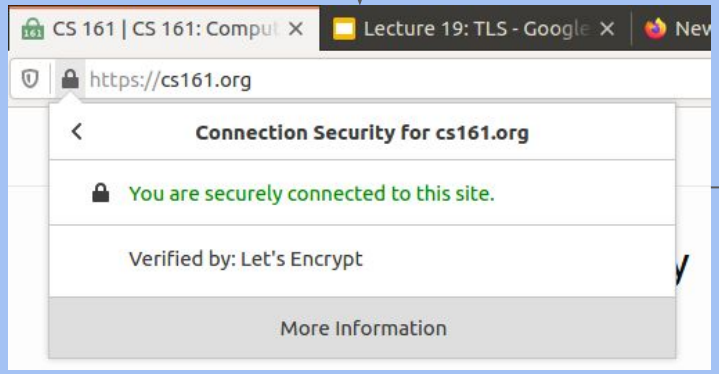


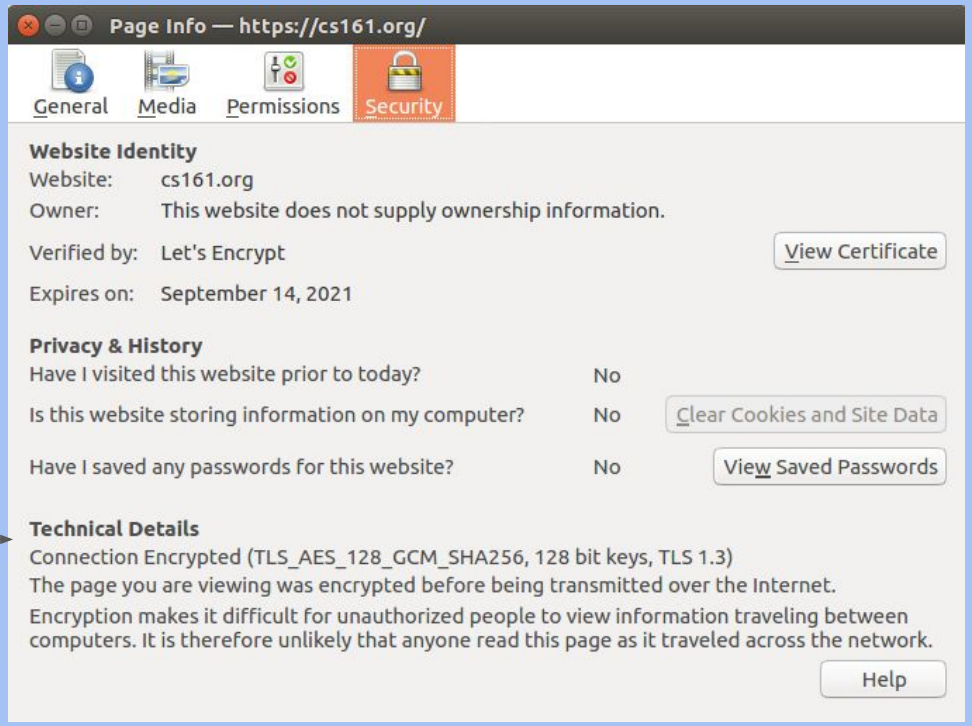
Issues: Unknown Certificate Authority

- 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

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

Solving Trust Issues

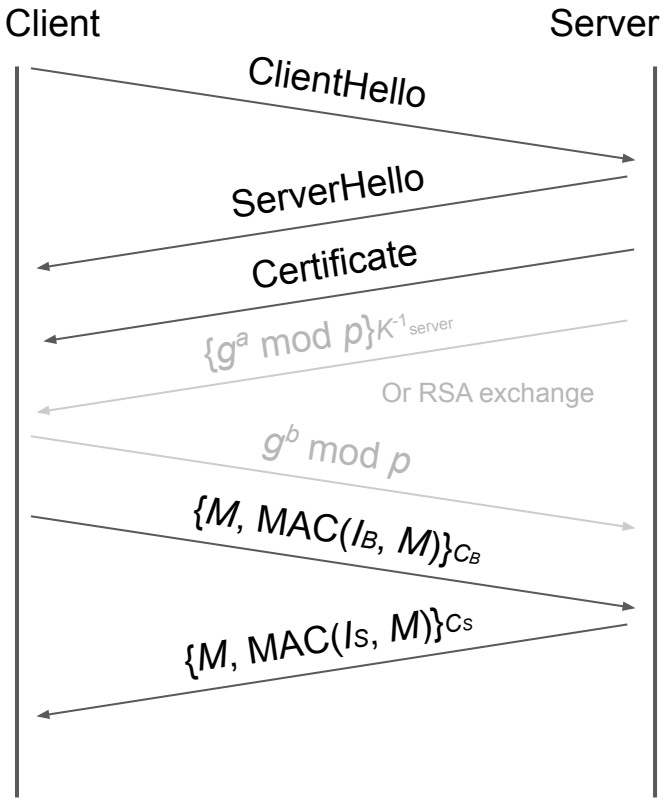
- Other solutions implementing to “trust but verify” the certificate you received
  - EFF’s SSL Observatory: Check against certificates seen by other dedicated computers, called “observatories,” placed around the Internet
  - ICSI’s Certificate Notary: Check against certificates used in common Internet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

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
- 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
  - 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

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