# CMSC414 Computer and Network Security

## TLS, Denial of Service and Firewalls

Yizheng Chen | University of Maryland
surrealyz.github.io

April 28, 2026

Credits: original slides from instructors and staff from CS161 at UC Berkeley. Blue slides will not be tested.

# Agenda

- TLS
- Denial of Service
- Firewalls

# Recall: TCP and UDP

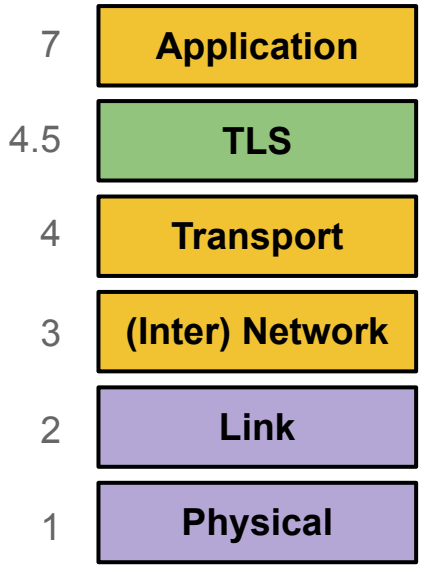
- Transmission Control Protocol (TCP): Reliably sending packets
  - 3-way handshake: Client sends SYN, server sends SYN-ACK, client sends ACK
  - Provides reliability, ordering, and ports
  - Attack: TCP hijacking through data injection or RST injection
    - Blind attacks must guess the client's or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

- **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!

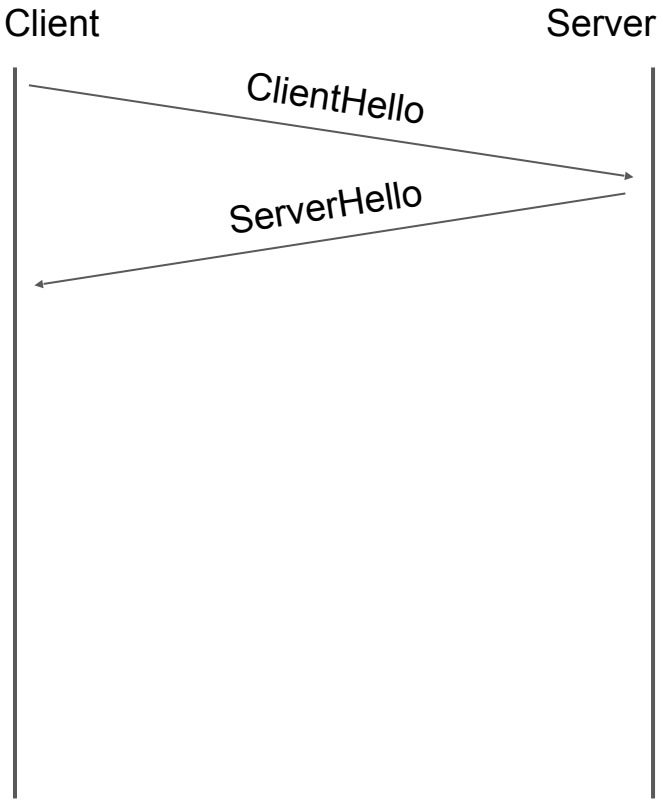


# Today: Secure Internet Communication with TLS

- 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

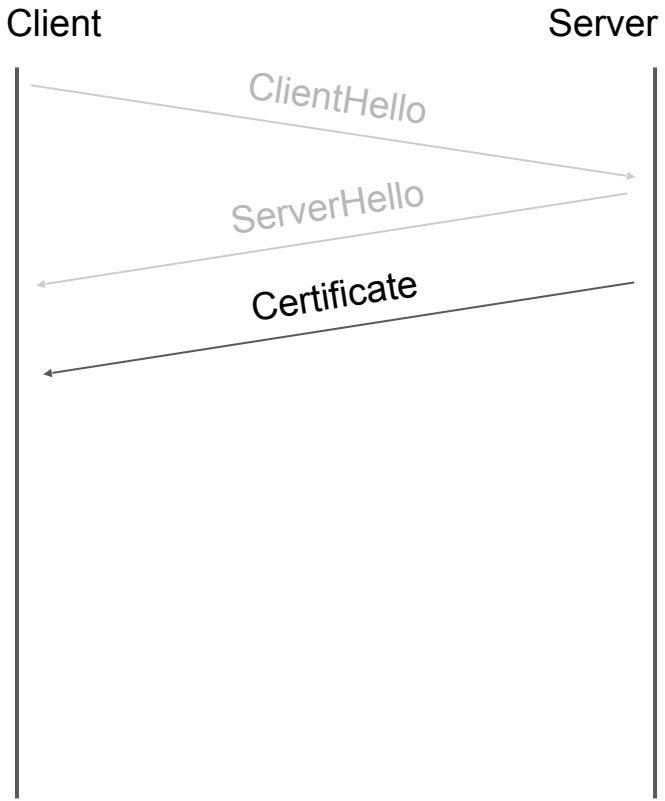
# TLS Handshake Step 1: Exchange Hellos

- Assume an underlying TCP connection has already been formed
- The client sends **ClientHello** with
  - A 256-bit random number $R_B$ (“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



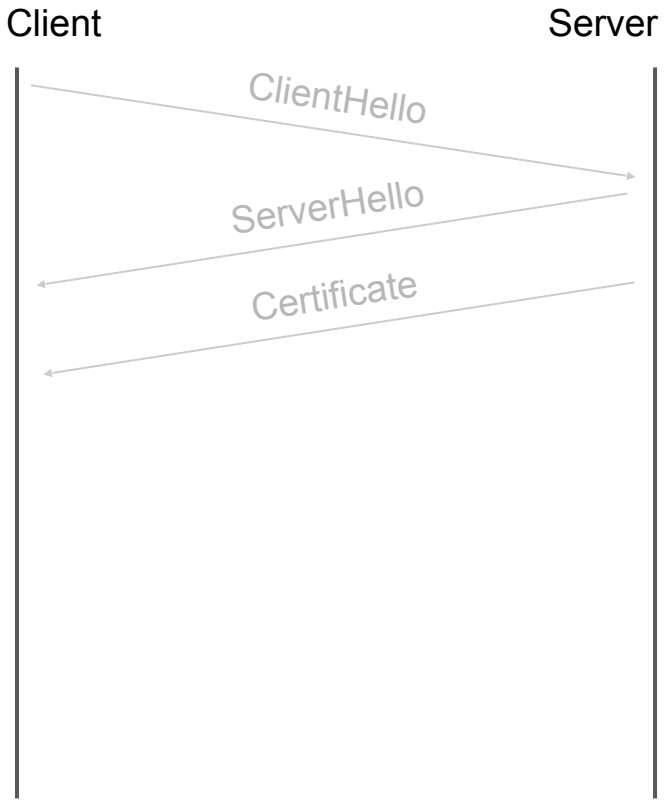
# TLS Handshake Step 2: Certificate

- 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



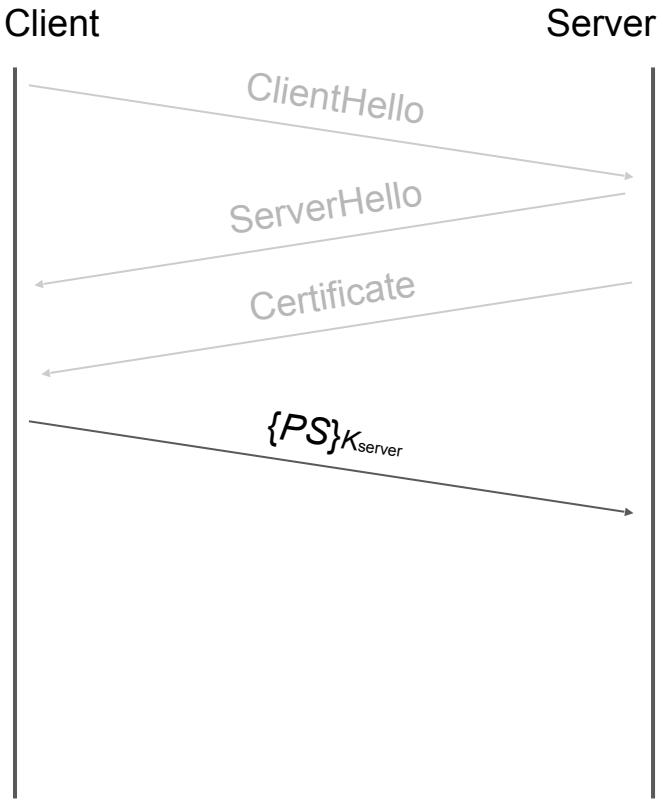
# TLS Handshake Step 3: Premaster Secret

- 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)



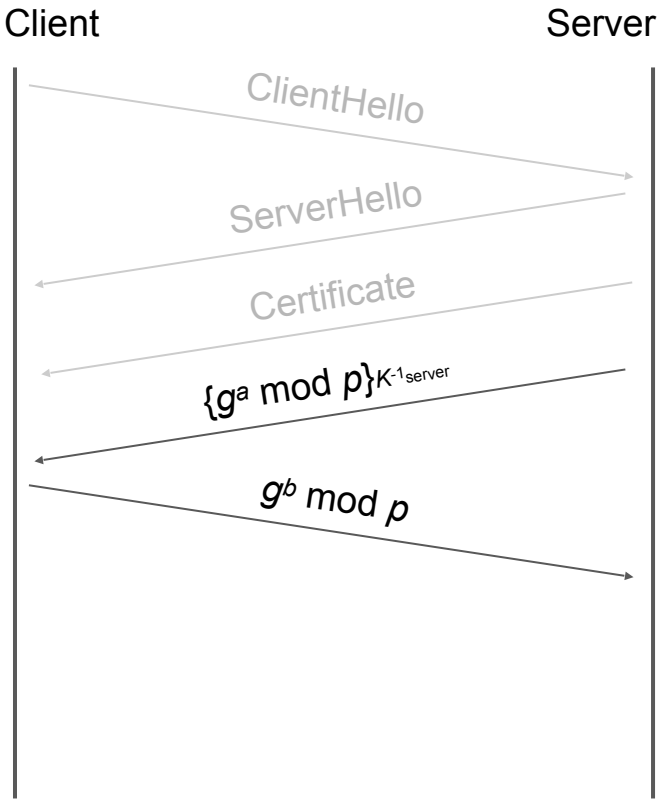
# TLS Handshake Step 3: Premaster Secret (RSA)

- 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*)



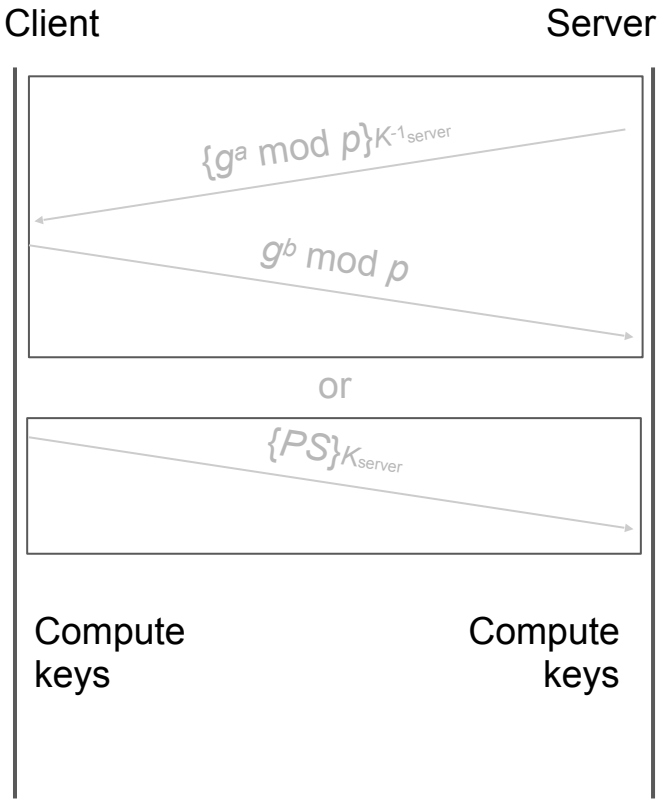
# TLS Handshake Step 3: Premaster Secret (DHE)

- 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 $g^{ab} \bmod p$



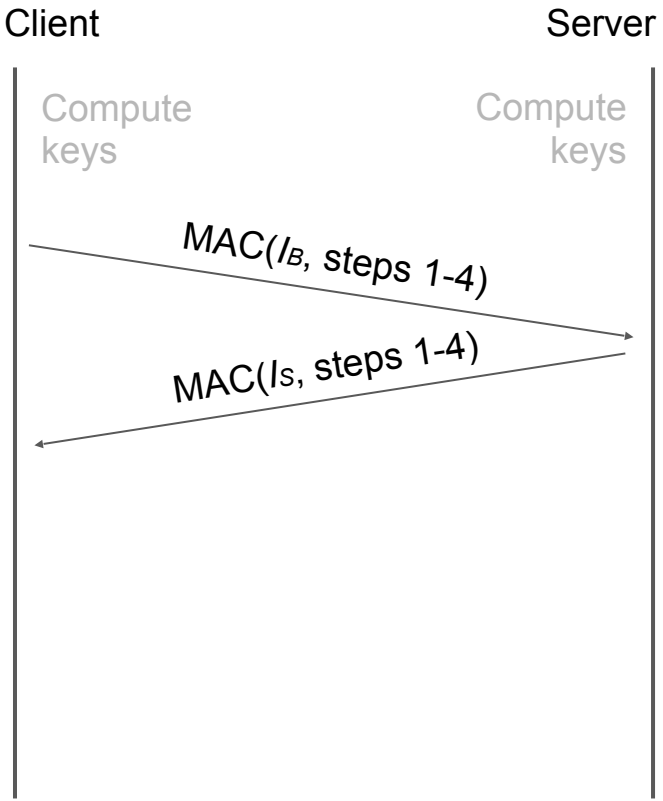
# TLS Handshake Step 4: Derive Symmetric Keys

- 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



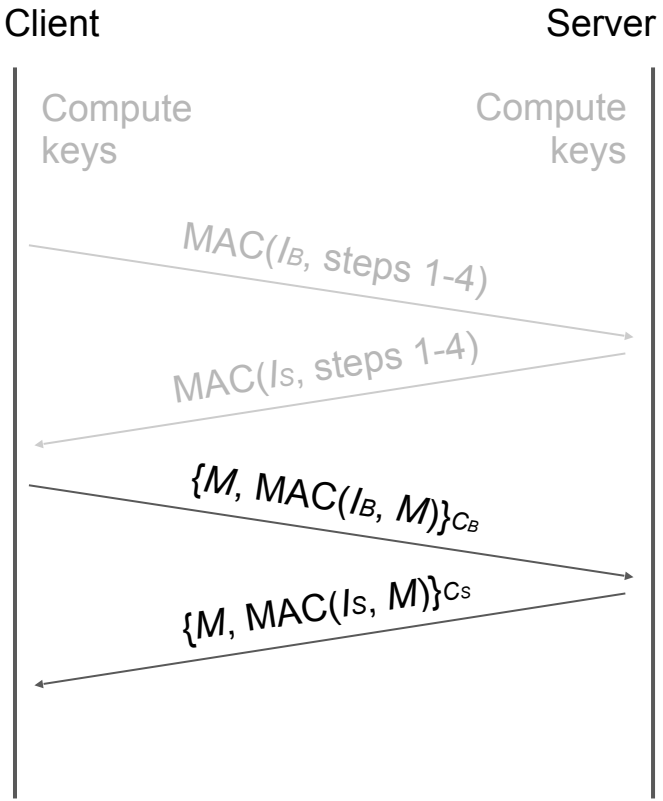
# TLS Handshake Step 5: Exchange MACs

- 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



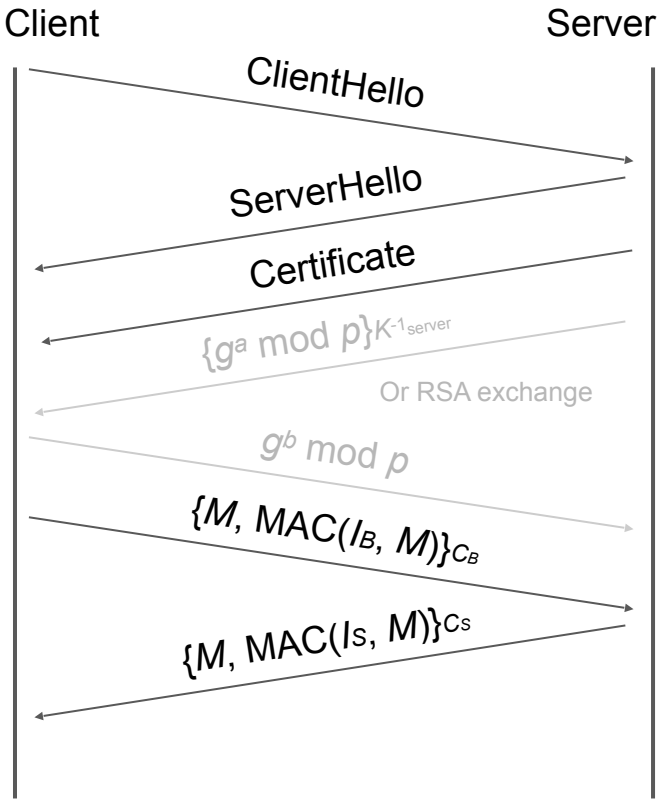
# TLS Handshake Step 6: Send Messages

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



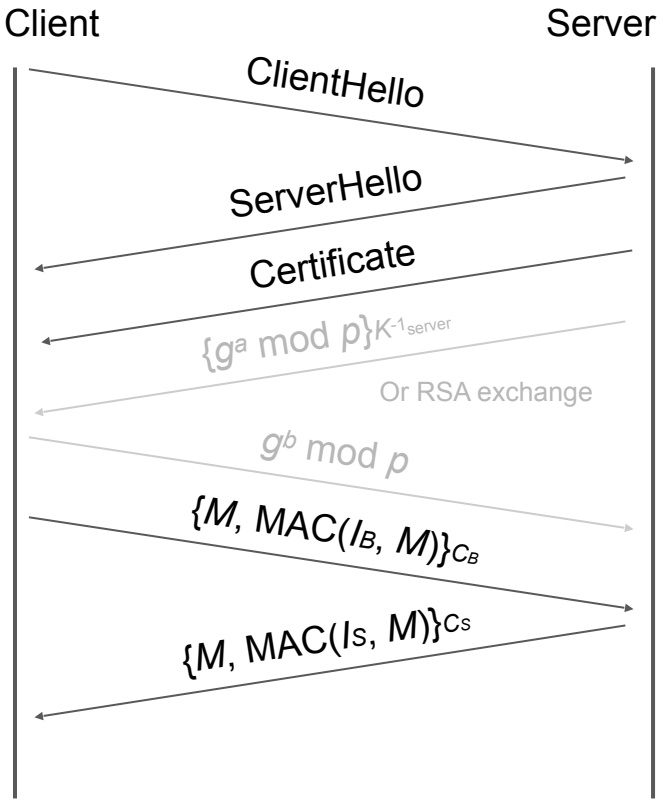
# TLS: Talking to the Legitimate Server

- 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)



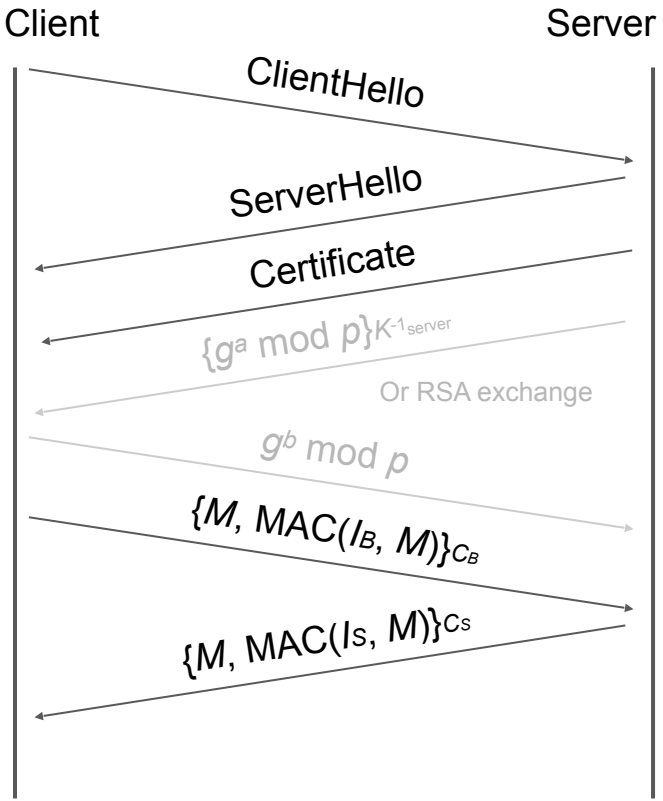
# TLS: Securing Messages

- 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



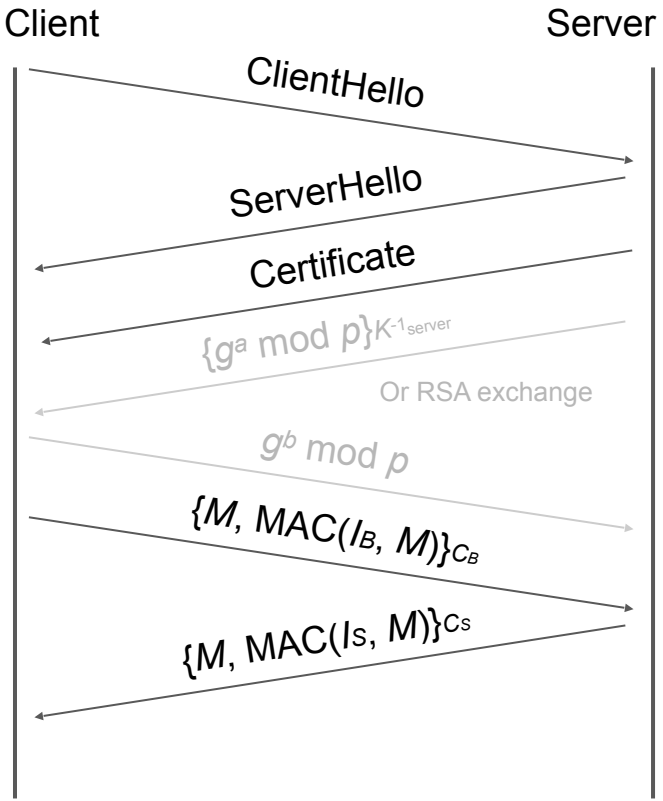
# TLS: Replay Attacks

- How can we be sure that the attacker hasn't replayed old messages from 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

# 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

# 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

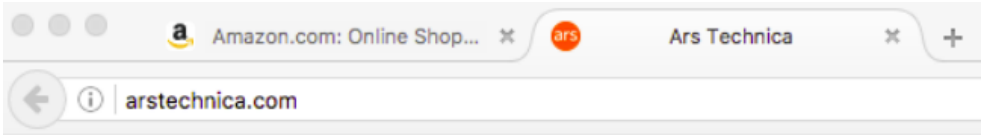
- **Availability**: Keeping the connection open in the face of attackers
- An attacker can stop a TLS connection
  - MITM can drop encrypted TLS packets
  - On-path attacker can still do RST injection to abort the underlying TCP connection
- Result: A TLS connection can still be censored
  - The censor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

# TLS for Applications

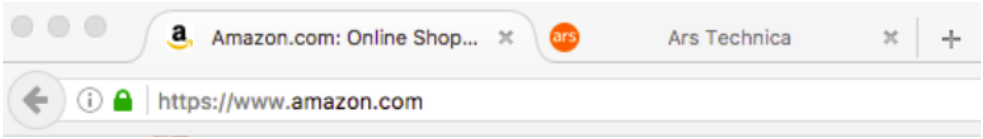
- Recall Internet layering: TLS provides services to higher layers (the application layer)
- **HTTPS**: The HTTP protocol run over TLS
  - In contrast, HTTP runs over plain TCP, with no TLS added
- Other secure application-layer protocols besides HTTPS exist
  - Pretty much anything that runs over TCP 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

# 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



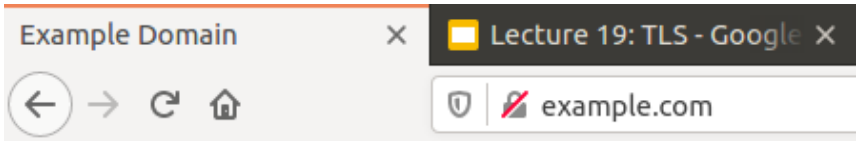

This website uses HTTPS: lock icon

# 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

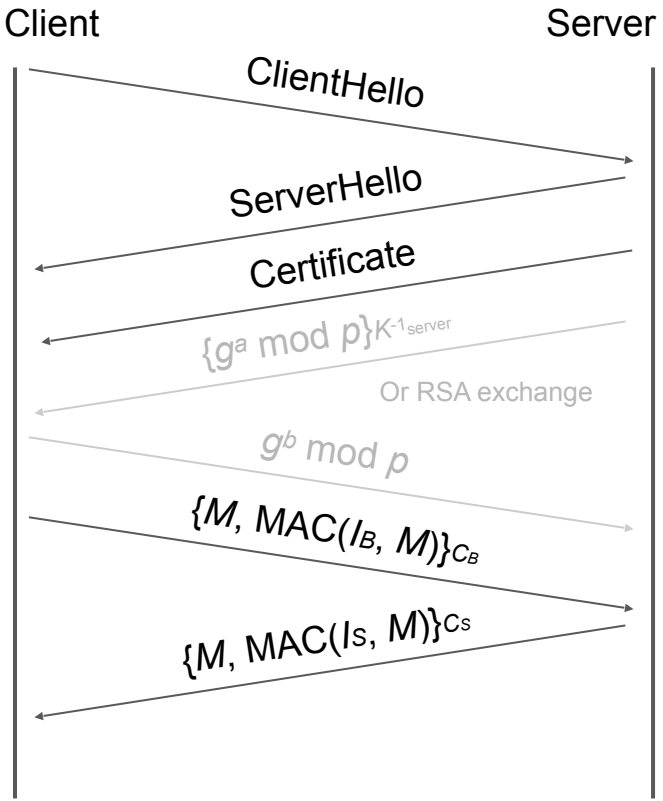



This website uses HTTP: insecure icon


This website uses HTTPS: lock icon

# TLS: 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

# Agenda

- TLS
- Denial of Service
- Firewalls

# Today: Denial of Service and Firewalls

- Denial of service
  - Availability
  - Application-level DoS
    - Algorithmic complexity attacks
  - Network-level DoS
    - Distributed DoS (DDoS)
    - Amplified DoS
  - SYN flooding
    - SYN cookies
  - Defenses
- Firewalls
  - Packet filters
    - Stateless/stateful packet filters
  - Proxy firewalls

# Availability and Denial of Service (DoS)

- **Availability**: Making a service on the network available for legitimate users

- **Denial of service** (**DoS**): An attack that disrupts availability of a service, making it unavailable for legitimate users
  - Reasons for a DoS attack
    - Competitors might DoS each other to benefit their own services
    - Criminals might DoS services unless the services pay a ransom
    - People might DoS services to make a political statement
    - Entities might DoS each other as part of warfare tactics
    - Some people might DoS for fun or revenge (e.g. online games)

# DoS in the News

KrebsonSecurity
In-depth security news and investigation

Link

## Digital Hit Men for Hire

*Brian Krebs* *August 1, 2011*

Cyber attacks designed to knock Web sites off line happen every day, yet shopping for a virtual hit man to launch one of these assaults has traditionally been a dicey affair. That's starting to change: Hackers are openly competing to offer services that can take out a rival online business or to settle a score.

There are dozens of underground forums where members advertise their ability to execute debilitating "distributed denial-of-service" or DDoS attacks for a price. DDoS attack services tend to charge the same prices, and **the average rate for taking a Web site offline is surprisingly affordable: about $5 to $10 per hour; $40 to $50 per day; $350-$400 a week; and upwards of $1,200 per month**.

# DoS in the News

COMPUTERWORLD

Link

## Extortion via DDoS on the rise

*Denise Pappalardo and Ellen Messmer* *May 16, 2005*

**Criminals are increasingly targeting corporations with distributed denial-of-service (DDoS) attacks designed not to disrupt business networks but to be used as tools to extort thousands of dollars from the companies.**

Those targeted are increasingly deciding to pay the extortionists rather than accept the consequences, experts say. While reports of this type of crime have circulated for several years, most victimized companies remain reluctant to acknowledge the attacks or enlist the help of law enforcement, resulting in limited awareness of the problem and few prosecutions.

# DoS in the News


Link

## DDoS makes a phishing e-mail look real

*Munir Kotadia* *November 8, 2006*

**Just as Internet users learn that clicking on a link in an e-mail purporting to come from their bank is a bad idea, phishers seem to be developing a new tactic -- launch a DDoS attack on the Web site of the company whose customers they are targeting and then send e-mails "explaining" the outage and offering an "alternative" URL.**

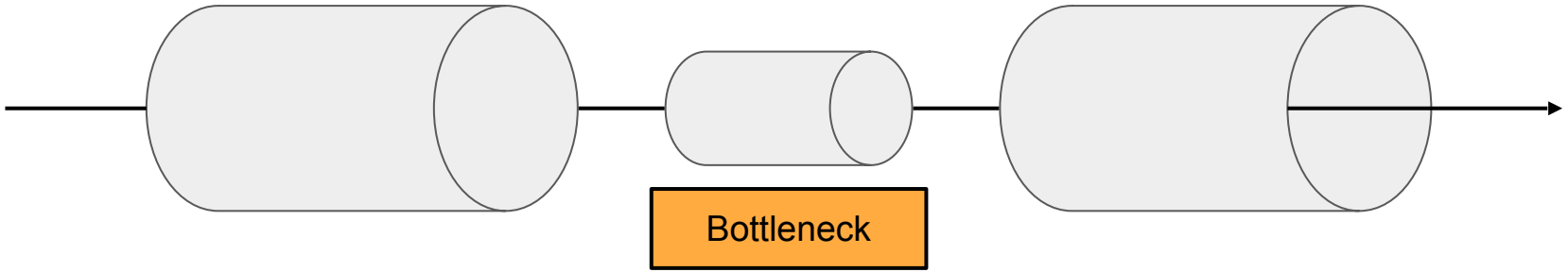
# DoS Attacks: Strategies

- Exploiting program flaws
  - Software vulnerabilities can cause a service to go offline
  - Example: Exploit a buffer overflow to execute a shutdown command to the system
  - Example: Exploit a SQL injection vulnerability to delete the database

- Resource exhaustion
  - Everything on the network has limited resources
  - The attacker consumes all the limited resources so legitimate users can’t use them

# DoS Attacks: Strategies

- Bottlenecks
  - Different parts of the system might have different resource limits
  - The attacker only needs to exhaust the **bottleneck**: the part of the system with the least resources



# DoS Targets

- **Application-level DoS**: Target the high-level application running on the host
- **Network-level DoS**: Target network protocols to affect the host's Internet

# Application-Level DoS

- Target the resources that the application uses
- Exploit features of the application itself
- Some attacks rely on **asymmetry**: A small amount of input from the attack results in a large amount of consumed resources!

# Resource Consumption

- Idea: Force the server to consume all its resources

```
int fd = open('/tmp/junk');
char buf[4096]
while (1) { write(fd, buf, 4096) };
```

Exhausts filesystem space

```
while (1) { malloc(1000000000); }
```

Exhausts RAM

```
while (1) { fork(); }
```

Exhausts processing threads

```
while (1) {
    int fd = open(random_file());
    write(fd, "abcde", 5);
    close(fd);
}
```

Exhausts disk I/O operations

# Algorithmic Complexity Attacks

- Consider an application that runs a sort on user-chosen data
  - What if the attacker intentionally chooses inputs that cause the worst-time runtime to occur?
- **Algorithmic complexity attack**: Supplying inputs that trigger worst-case complexity of algorithms and data structures
  - Defense: Choose algorithms with good worst-case running times
    - Mergesort is safer than quicksort against DoS!

| | Expected runtime | Worst-case runtime |
|---|---|---|
| Mergesort | $O(n \log n)$ | $O(n \log n)$ |
| Quicksort | $O(n \log n)$ | $O(n^2)$ |

# Application-Level DoS: Defenses

- **Identification**: Step 0 of any defense
  - You must be able to distinguish requests from different users before you can do anything else!
  - Requires some method to identify/authenticate users
  - Authenticating users might be expensive and itself vulnerable to DoS

- **Isolation**: Ensure that one user's actions do not affect another user's experience

- **Quotas**: Ensure that users can only access a certain proportion resources
  - Example: Only trusted users can execute expensive requests
  - Example: Limit each user to 4 GB of RAM and 2 CPU cores

# Application-Level DoS: Defenses

- **Proof-of-work**: Force users to spend some resources to issue a request
  - Idea: Make a DoS attack more expensive for the attacker, who now needs to spend resources
  - Example: Add a CAPTCHA, which the attacker will now have to solve (or pay for solving services)

- **Overprovisioning**: Allocate a huge amount of resources
  - Can cost the server a lot of money!
  - Depends on your threat model
  - Often the most effective defense (“security is economics”)
  - **Content delivery network** (CDN): A service that allocates a huge amount of resources for you
    - Example of a CDN: Cloudflare
    - Cloudflare runs your service for you with a huge amount of resources

# Network-Level DoS

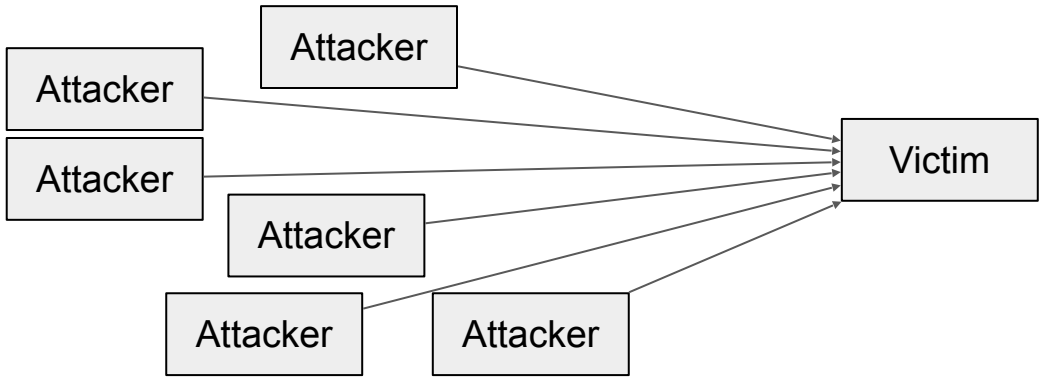
- Approaches target network protocols to affect the victim's Internet access
  - Example: Send a huge amount of packets to the victim

- Overwhelm the victim's **bandwidth** (amount of data it can upload/download in a given time)
  - Example: The server can only upload/download 10 MB/s. The attacker sends the server 20 MB/s.
    - Lots of maximum-sized packets

- Overwhelm the victim's **packet processing capacity**
  - Example: The server can process 10 packets/second. The attacker sends the server 20 packets/second.
    - Lots of minimum-sized packets

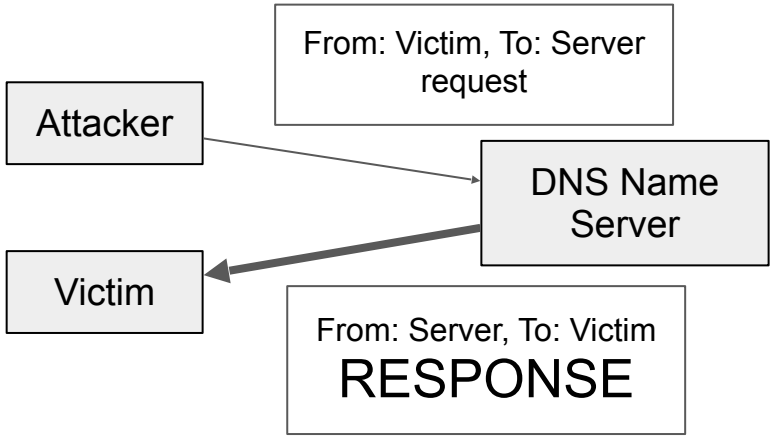
# Distributed Denial-of-Service (DDoS)

- **Distributed denial-of-service** (**DDoS**): Use multiple systems to overwhelm the target system
  - Controlling many systems gives the attacker a huge amount of bandwidth
  - Sending packets from many sources makes it hard for packet filters to distinguish DDoS traffic from normal traffic
  - **Botnet**: A collection of compromised computers controlled by one attacker
    - The attacker can tell all the computers on the botnet to flood a given target



# Amplified Denial-of-Service

- **Amplified denial-of-service**: Use an amplifier to overwhelm the target more effectively
  - Idea: Some services send a large response when sent a small request
  - Spoofing a small request that appears to come from the victim results in a large amount of data sent to the victim
  - Example: DNS amplification
    - Requests contain only the question
    - Responses contain answer records, authority records, and additional records



# Amplified Denial-of-Service

- Benefits:
  - The attacker’s identity is concealed because the packets come from the amplification server
  - The attacker is able to overwhelm more bandwidth with relatively little bandwidth
    - Amplification servers often have massive bandwidths to support large numbers of users

- Drawbacks:
  - Requires blind spoofing capability
    - Cannot work over TCP, since TCP spoofing is assumed to be hard, only UDP protocols

# Network-Level DoS: Defenses

- **Packet filter**: Discard any packets that are part of the DoS attack
  - Discard packets where the source IP is the attacker's IP address
  - Find some pattern in the content of the DoS packets to distinguish DoS packets from legitimate packets
  - The packet filter must be before the bottleneck

- Subverting packet filters
  - Spoof DoS packets so that packets look like they're coming from many IP addresses
    - Packet filters can't use IP addresses to filter packets anymore!
    - Hard to defend against
    - Rely on anti-spoofing mechanisms on the network
  - Distributed DoS actually send packets from many IP addresses
    - Packet filters need to be much more sophisticated to defend against DDoS attacks

# Network-Level DoS: Defenses

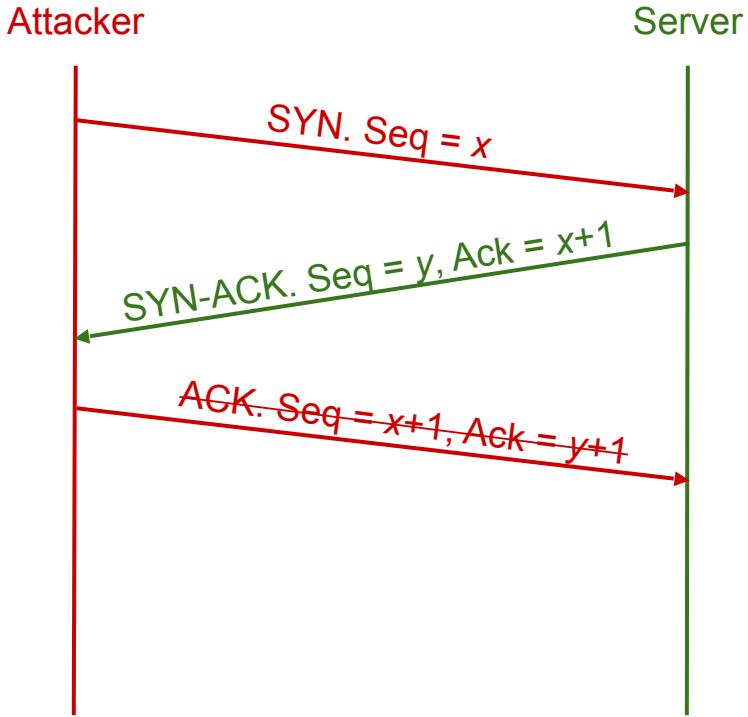
- **Overprovisioning**: Purchase enough networking bandwidth and equipment to make it harder for attackers to overwhelm the network
  - Again, depends on your threat model

# Today: Denial of Service and Firewalls

- Denial of service
  - Availability
  - Application-level DoS
    - Algorithmic complexity attacks
  - Network-level DoS
    - Distributed DoS (DDoS)
    - Amplified DoS
  - SYN flooding
    - SYN cookies
  - Defenses
- Firewalls
  - Packet filters
    - Stateless/stateful packet filters
  - Proxy firewalls

# SYN Flooding

- A type of DoS that exploits many TCP connections
  - Each connection established by the server needs to allocate some memory
    - Used to store sequence numbers, ACK numbers, buffered data, etc.
  - Idea: Establish many connections with the server, causing it to consume a lot of memory
- TCP state is allocated upon receiving a SYN
  - The attacker only needs to send the SYN, so the attacker doesn't consume its own resources!



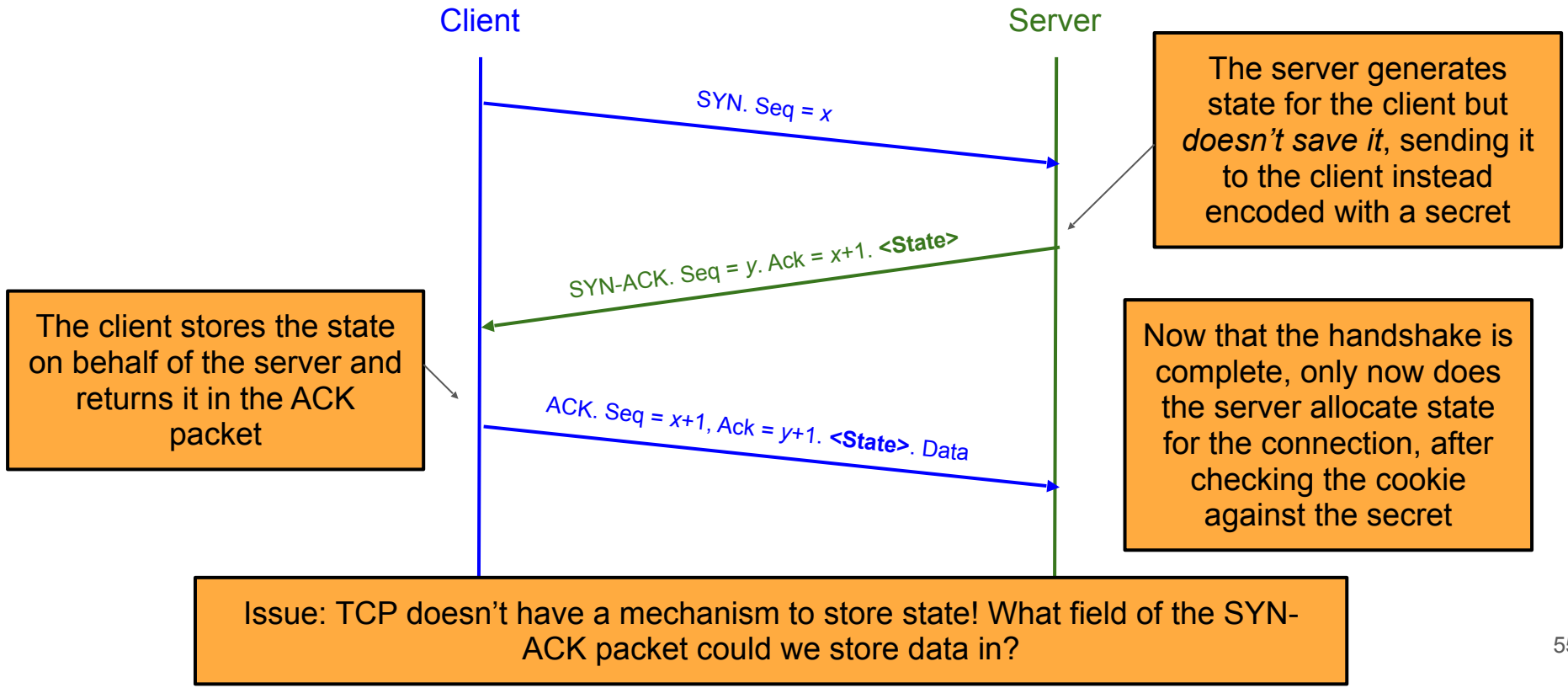
# SYN Flooding: Defenses

- **Overprovisioning**: Ensure the server has a lot of memory
  - Can be expensive and depends on your threat model

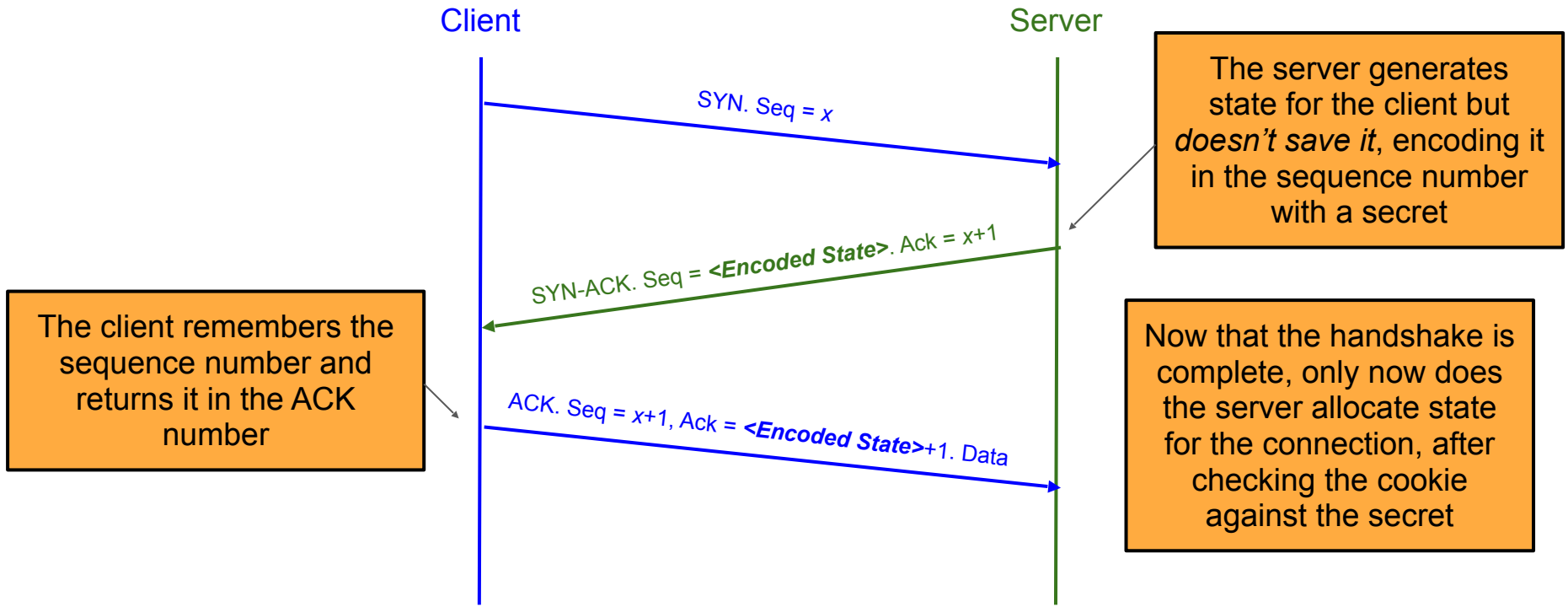
- **Filtering**: Ensure that only legitimate connections will create state
  - Same problems as standard packet filtering for network-level DoS attacks
  - Hard to distinguish legitimate traffic so early in the connection
  - Attacker can spoof source address since they only need to send the SYN, not the ACK

- **SYN cookies**: Don’t store state!
  - Relies on the client to store the server’s state
  - The client returns the state to the server in the ACK packet of the handshake

# Idealized SYN Cookies



# Practical SYN Cookies



# Practical SYN Cookies

- Observation: The server doesn’t create state until the handshake is completed, so the attacker can’t spoof source addresses
  - Filtering becomes easier with SYN cookies

- We can generalize this: Instead of holding state in the server, encode it with a secret and send it to the client, who will return it when it is next needed
  - Requires enough bits to encode the state
  - We must make sure that checking the state against the secret is inexpensive, or this becomes another DoS vector!

# Today: Denial of Service and Firewalls

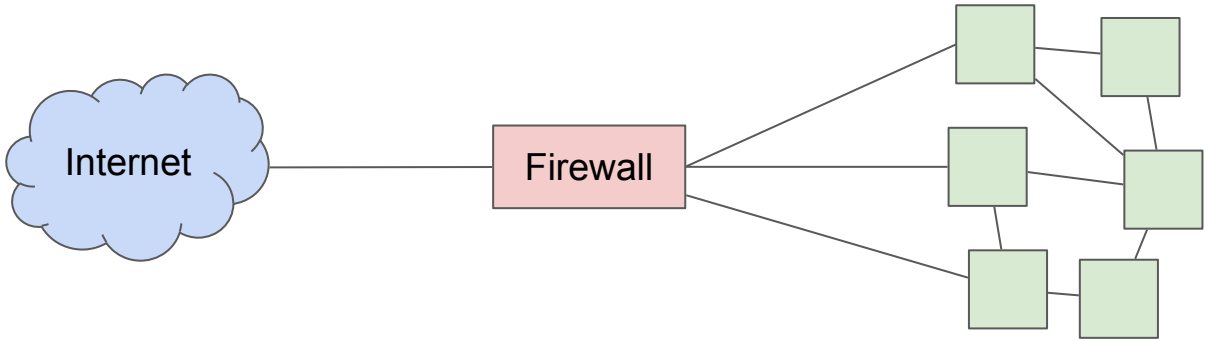
- Denial of service
  - Availability
  - Application-level DoS
    - Algorithmic complexity attacks
  - Network-level DoS
    - Distributed DoS (DDoS)
    - Amplified DoS
  - SYN flooding
    - SYN cookies
  - Defenses
- Firewalls
  - Packet filters
    - Stateless/stateful packet filters
  - Proxy firewalls

# Motivation: Scalable Defenses

- How do you protect a set of systems against external attack?
  - Example: A company network with many servers and employee computers
- Observation: More network services = more risk
  - Each network connection creates more opportunities for attacks (greater attack surface)
  - Turning off all network services is often infeasible (print services, SSH services, etc.)
- Observation: More networked machines = more risk
  - What if you have to secure hundreds of systems?
  - What if the systems have different hardware, operating systems, and users?
  - What if there are some systems in the network that you aren't aware of?
- Instead of securing individual machines, we want to secure the entire network!

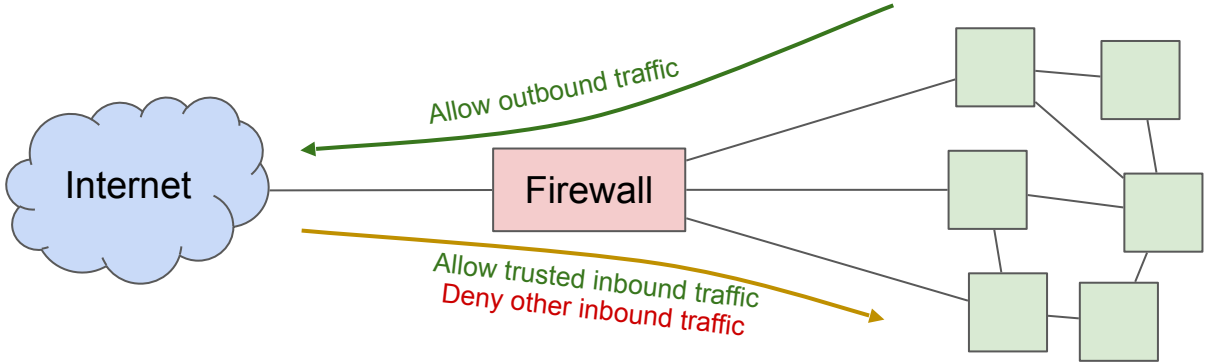
# Firewalls and Security Policies

- Idea: Add a single point of access in and out of the network, with a monitor
  - “Ensure complete mediation”
  - Any traffic that could affect vulnerable systems must pass through the firewall
- Network access is controlled by a **policy**
  - Defines what traffic is allowed to exit the network (**outbound policy**)
  - Defines what traffic is allowed to enter the network (**inbound policy**)
  - Policy model based on our threat model: We usually assume users “inside” the network are trusted, and those outside are not



# Firewalls and Security Policies

- What's the policy of a standard home network?
  - Outbound policy: Allow outbound traffic
    - Users inside the network can connect to any service
  - Inbound policy: Only some traffic is able to enter the network
    - Allow inbound traffic in response an outbound connection
    - Allow inbound traffic to certain, trusted services (e.g. SSH)
    - Deny all other inbound traffic



# Default Security Policies?

- How should we handle traffic that isn't explicitly allowed or denied?
  - **Default-allow policy**: Allow all traffic, but deny those on a specified **denylist**
    - As problems arise, add them to the denylist
  - **Default-deny policy**: Deny all traffic, but allow those on a specified **allowlist**
    - As needs arise (or users complain), add them to the allowlist?

- Which default policy is best?
  - Default-allow is more flexible, but flaws are vulnerabilities and can be catastrophic
  - Default-deny is more conservative, but flaws are less painful
  - Default-deny is generally accepted to be the best default policy ("consider fail-safe defaults")

# Stateless Packet Filters

- Firewalls are often **packet filters**, which inspect network packets and chooses what to do with them
  - Option #1: Allow the packet to pass through the firewall, forwarding it onwards
  - Option #2: Deny the packet from passing through the firewall, dropping it

- Stateless packet filters
  - Packet filters that have no history
  - All decisions must be made using only the information in the packet itself
  - Can have trouble implementing complex policies that require knowledge of history

# Stateless Packet Filters

- Consider implementing the typical home network policy from earlier:
  - Allow outbound traffic
  - Allow inbound traffic in response to an outbound connection
  - Deny all other inbound traffic

- Issue: How do we know what inbound traffic is in response to an outbound connection?
  - TCP: Can be implemented with a hack
    - Allow inbound traffic with the ACK flag set
    - Deny inbound traffic without an ACK flag set
    - If the internal computer sees an ACK packet without having formed a connection, it will ignore it or send a RST
  - UDP: Impossible to implement
    - UDP “connections” are typically implemented at the application layer, so we can’t inspect much

# Stateful Packet Filters

- A better idea: Keep state in the implementation of the packet filter
  - The filter keeps track of inbound/outbound connections
    - Notice: All connections have packets going in both directions, so a stateless filter could not do this
  - Rules define what connections are allowed or denied
  - Ultimately, packets are still either forwarded or dropped

- Example rules:
  - `allow tcp connection 4.5.5.4:* -> 3.1.1.2:80`
    - Allow connections from `4.5.5.4` to `3.1.1.2` with destination port 80
  - `allow tcp connection *:*/int -> *:80/ext`
    - Allow outbound connections with destination port 80
  - `allow tcp connection *:*/int -> *:*/ext`
    - Allow all outbound connections
  - `allow tcp connection *:*/ext -> 1.2.2.3:80`
    - Allow inbound connections to `1.2.2.3` with destination port 80

# Stateful Packet Filters

- Stateful packet filters can also track the state of well-known applications
  - Example: Decoding and tracking HTTP requests/responses
  - Example: Tracking the files sent in an FTP (File Transfer Protocol) connection

# Other Types of Firewalls

- **Proxy firewall**: Instead of forwarding packets, form two TCP connections: One with the source, and one with the destination
  - The firewall is really just a MITM
  - Avoids problems with packets, since the firewall has direct access to the TCP byte streams

- **Application proxy firewall**: Certain protocols allow for proxying at the application layer
  - Example: HTTP proxies will make an HTTP request on behalf of the user then return the HTTP response to the client

# Alternatives to Allowing Firewall Traffic

- **Virtual private network** (**VPN**): A set of protocols that allows direct access to an internal network via an external connection
  - Creates an encrypted tunnel to allow internal network traffic to be sent securely over the Internet
  - Intuition: The encrypted tunnel is an emulated Ethernet cable that allows you to connect “inside” the network
  - The firewall allows VPN traffic, which allows arbitrary traffic to be tunneled inside

# Firewall Pros and Cons

- Pros
  - Centralized management of security policies (single point of control)
  - Transparent operation to end users
  - Mitigates security vulnerabilities on end hosts (e.g. block anything that looks like shellcode)
- Cons
  - Reduced network connectivity
    - Some applications don’t work well inside a firewall
  - Vulnerability to “insiders”
    - Employees could be bribed or threatened
    - Devices are often brought from into the network outside (e.g. cell phones, laptops)
    - Once one device is compromised, attackers can quickly spread through the network
    - Could be mitigated by layering firewalls for more sensitive devices

# Summary: Denial of Service

- **Availability**: Making sure users are able to use a service
  - DoS attacks availability of services
- **Application-level DoS**: Attacks the high-level applications
  - Algorithmic complexity attacks: Attack using inputs that cause the worst-case runtime of an algorithm
  - Defense: Identification, isolation, and quotas
  - Defense: Proof of work
- **Network-level DoS**: Attacks the network of a service
  - Typically floods either the network bandwidth or the packet processing capacity
  - Distributed DoS: Use multiple computers to flood a network at the same time
  - Amplified DoS: Use an amplifier to turn a small input into a large output, spoofing packets so the reply goes to the victim
  - Defense: Packet filtering
- All DoS attacks can be defended against by overprovisioning

# Summary: SYN Cookies

- **SYN flooding**: A type of DoS that causes a server to allocate state for unfinished TCP connections, upon receiving a SYN packet
  - **SYN cookies**: Instead of allocating state when receiving a SYN, send the state back to the client in the sequence number
  - The client returns the state back to the server, which it only then allocates state for

# Summary: Firewalls

- **Firewalls**: Defend many devices by defending the network
  - **Security policies** dictate how traffic on the network is handled
- **Packet filters**: Choose to either forward or drop packets
  - **Stateless packet filters**: Choose depending on the packet only
  - **Stateful packet filters**: Choose depending on the packet and the history of the connection
  - Attackers can subvert packet filters by splitting key payloads or exploiting the TTL
- **Proxy firewalls**: Create a connection with both sides instead of forwarding packets