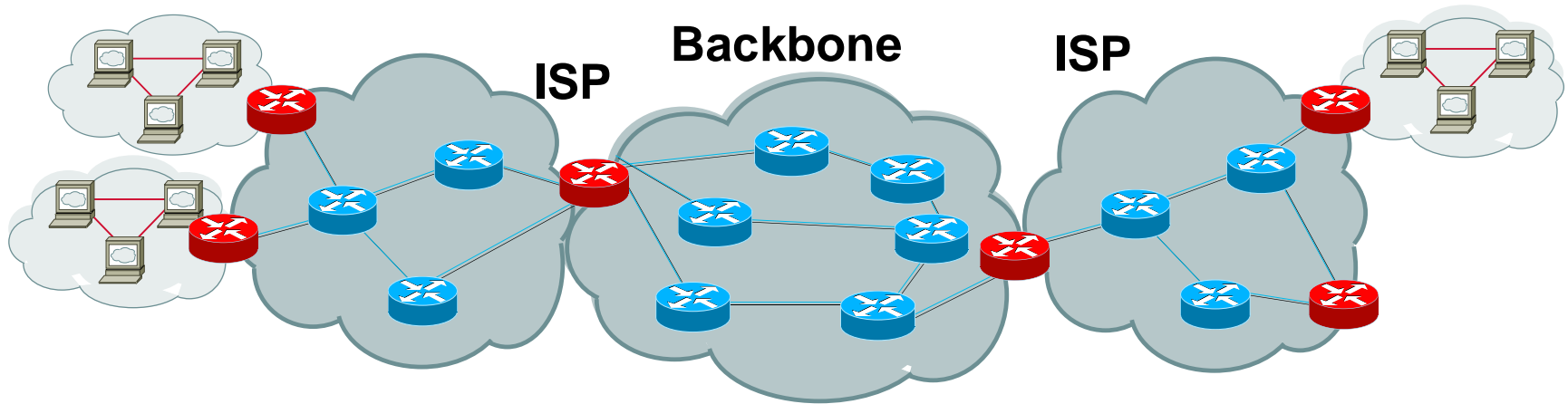


# Module 6: Internet Security

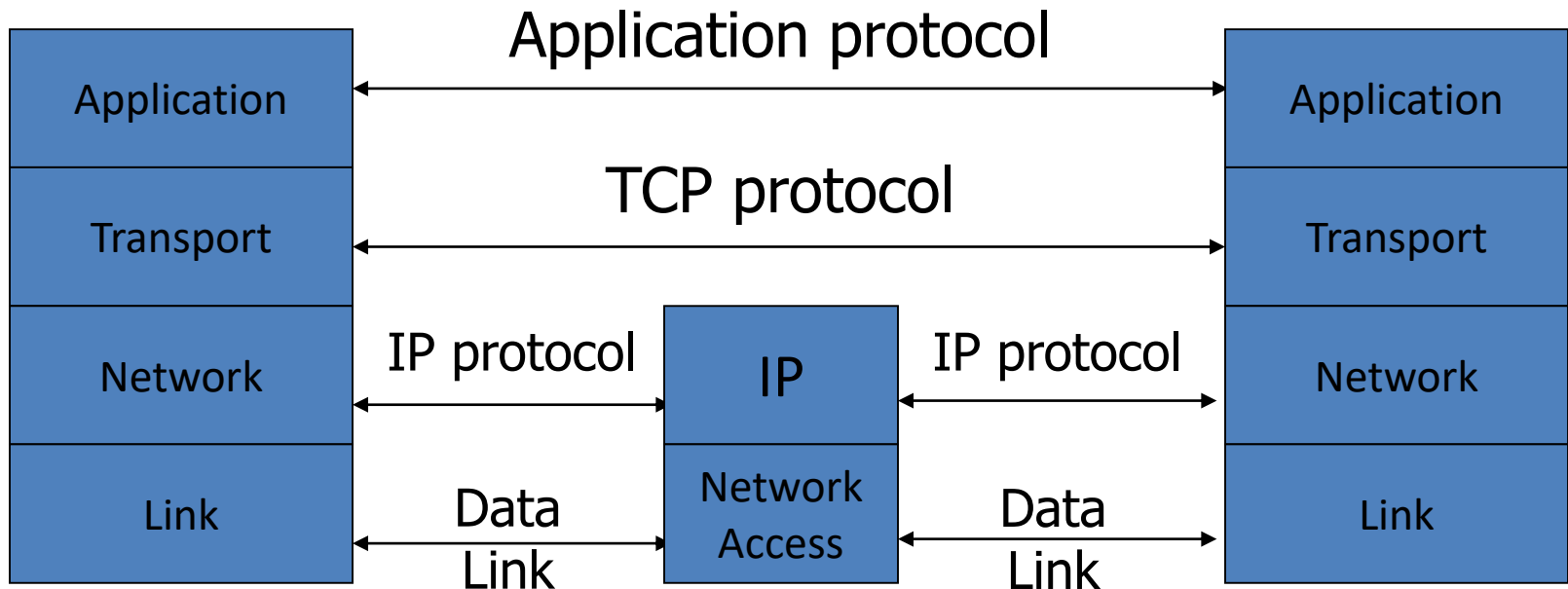
TCP/IP, DNS Security, HTTPS

# Internet Infrastructure

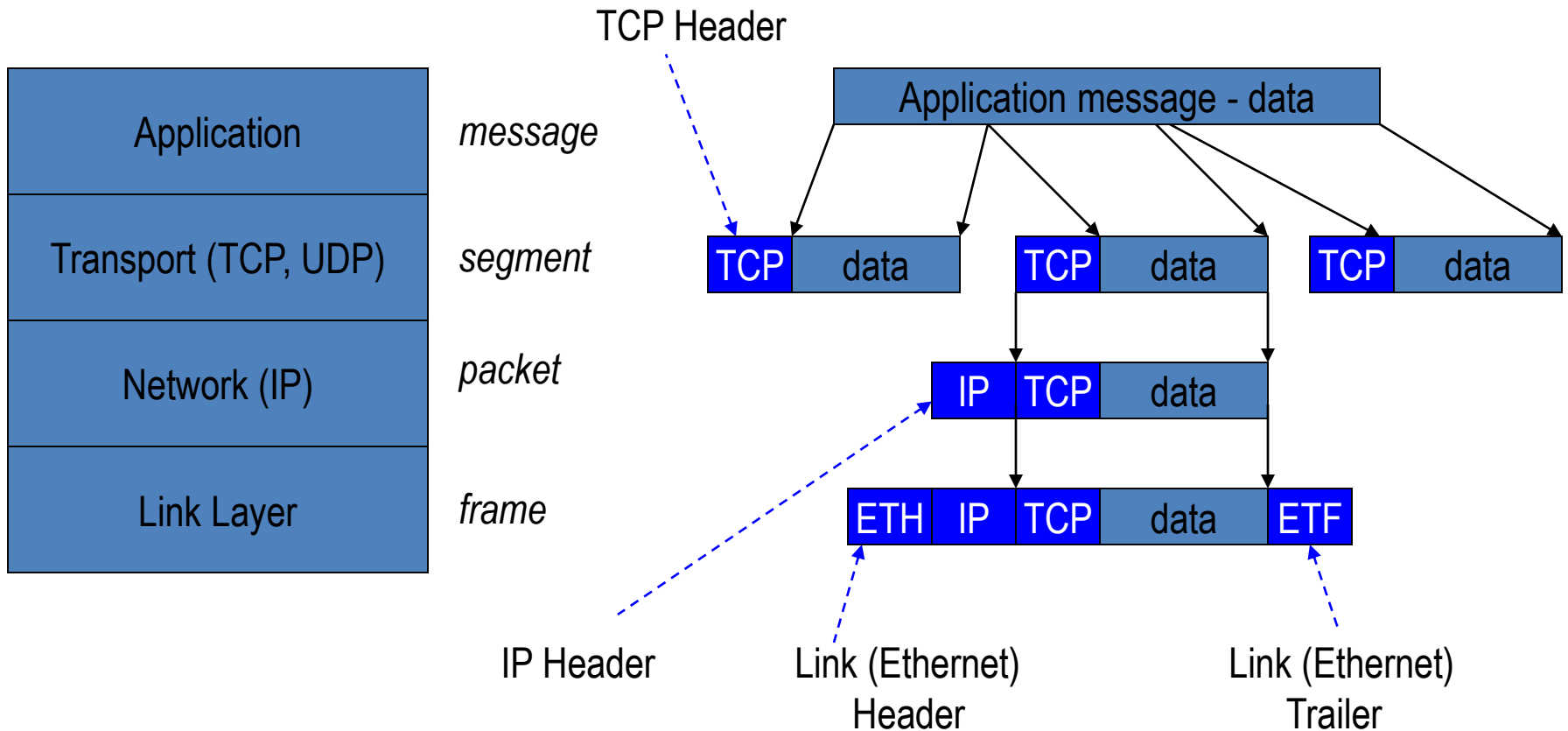


- Local and interdomain routing
  - TCP/IP for routing and messaging
  - BGP for routing announcements
- Domain Name System
  - Find IP address from symbolic name ([www.cse.iitk.ac.in](http://www.cse.iitk.ac.in))

# TCP Protocol Stack



# Data Formats

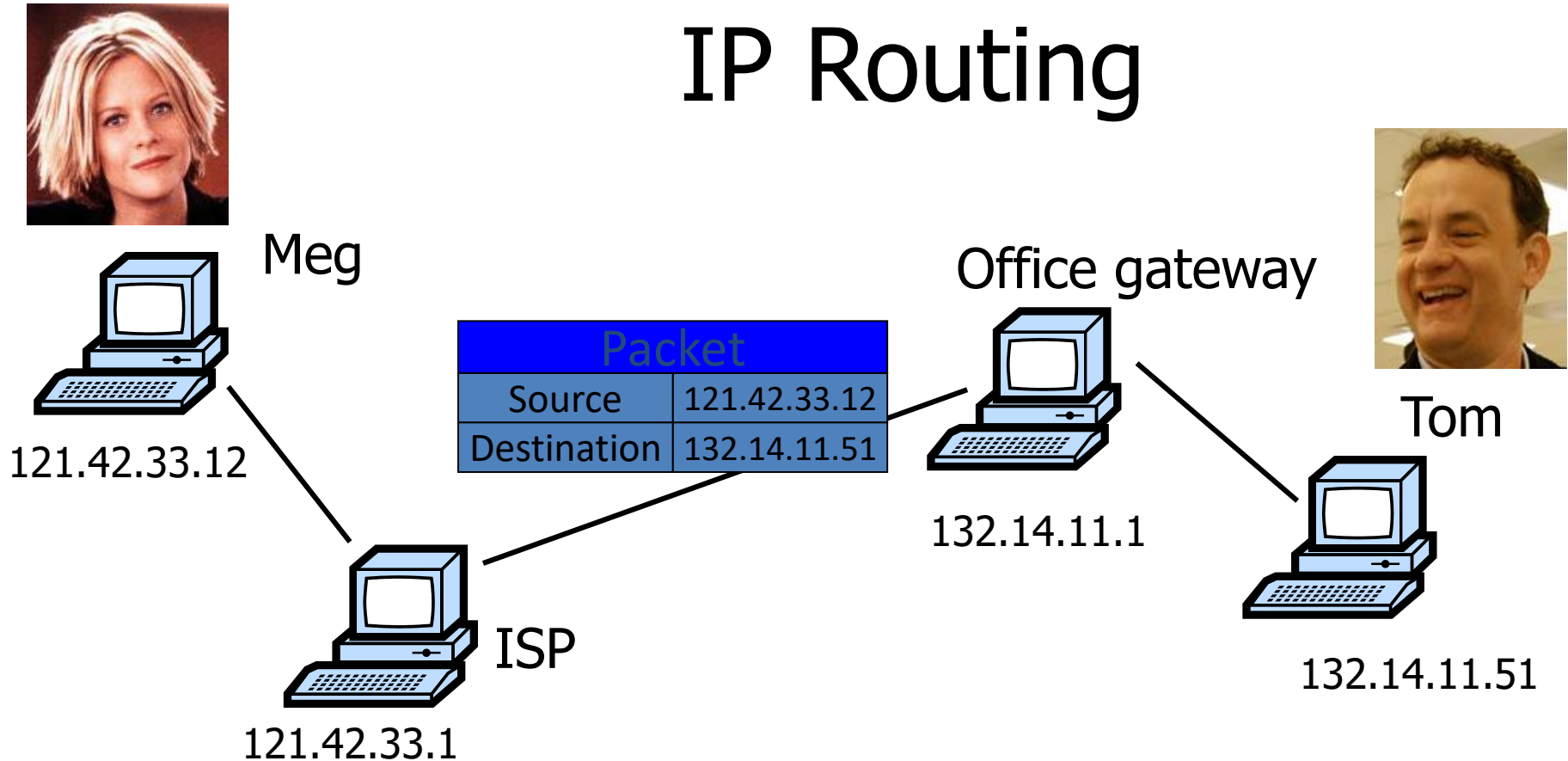


# Internet Protocol

- Connectionless
  - Unreliable
  - Best effort
- Notes:
  - src and dest **ports** not parts of IP hdr

Version	Header Length
Type of Service	
Total Length	
Identification	
Flags	Fragment Offset
Time to Live	
Protocol	
Header Checksum	
Source addr of Originating Host	
Destination addr of Target Host	
Options	
Padding	
IP Data	

# IP Routing



- Typical route uses several hops
- IP: no ordering or delivery guarantees

# IP Protocol Functions (Summary)

- Routing
  - IP host knows location of router (gateway)
  - IP gateway must know route to other networks
- Fragmentation and reassembly
  - If max-packet-size less than the user-data-size
- Error reporting
  - ICMP packet to source if packet is dropped
- TTL field: decremented after every hop
  - Packet dropped if TTL=0. Prevents infinite loops.

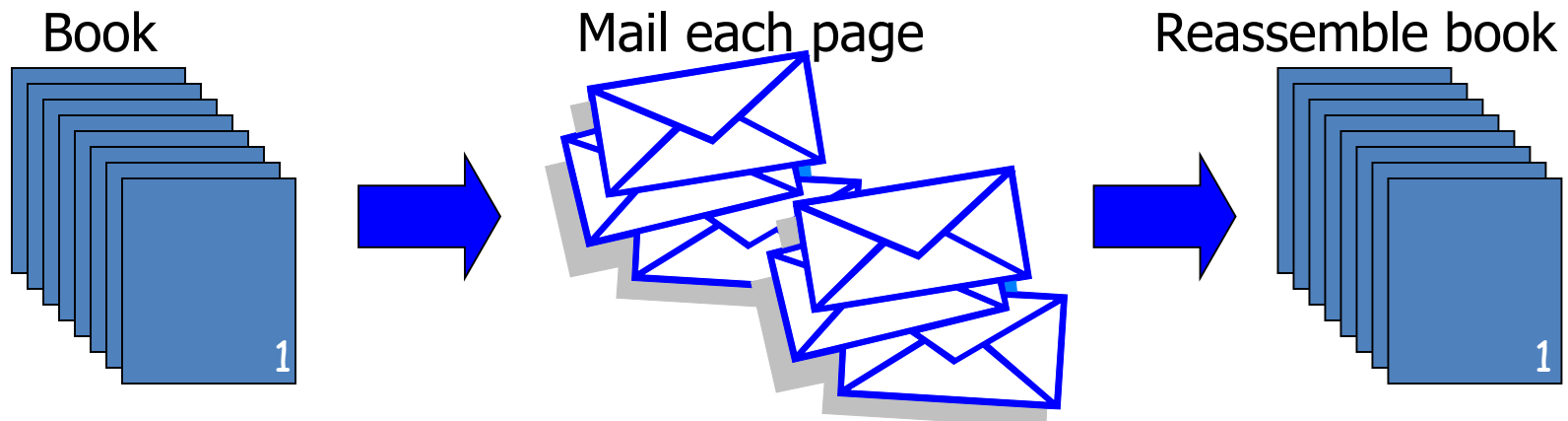
# Problem: no src IP authentication

- Client is trusted to embed correct source IP
  - Easy to override using raw sockets
  - **Libnet**: a library for formatting raw packets with arbitrary IP headers  
(<https://repolinux.wordpress.com/2011/09/18/libnet-1-1-tutorial/>)
- ◆ Anyone who owns their machine can send packets with arbitrary source IP
  - ... response will be sent back to forged source IP
- Implications: (solutions in DDoS lecture)
  - Anonymous DoS attacks;
  - Anonymous infection attacks (e.g. slammer worm)



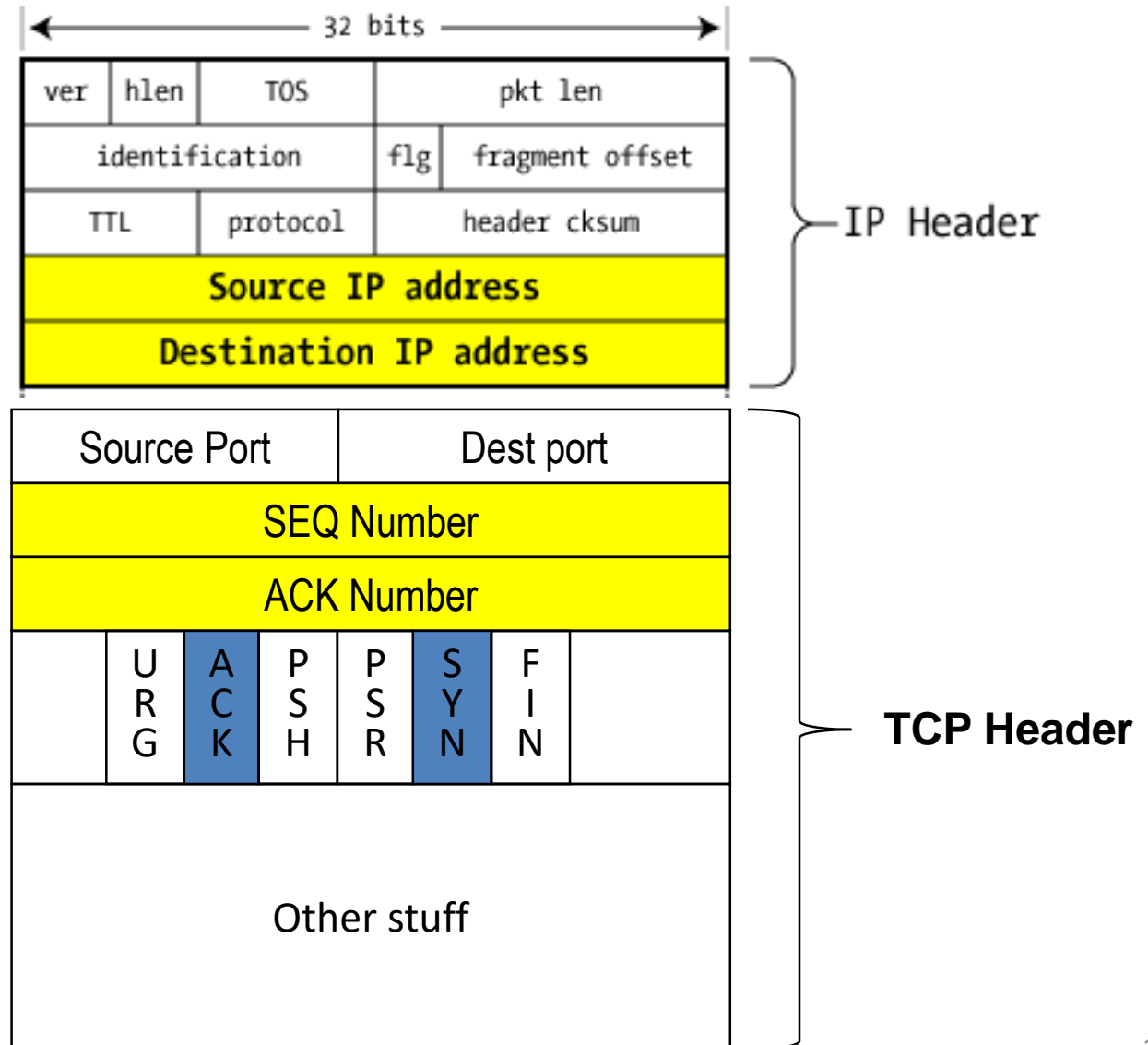
# Transmission Control Protocol

- Connection-oriented, preserves order
  - Sender
    - Break data into packets
    - Attach packet numbers
  - Receiver
    - Acknowledge receipt; lost packets are resent
    - Reassemble packets in correct order

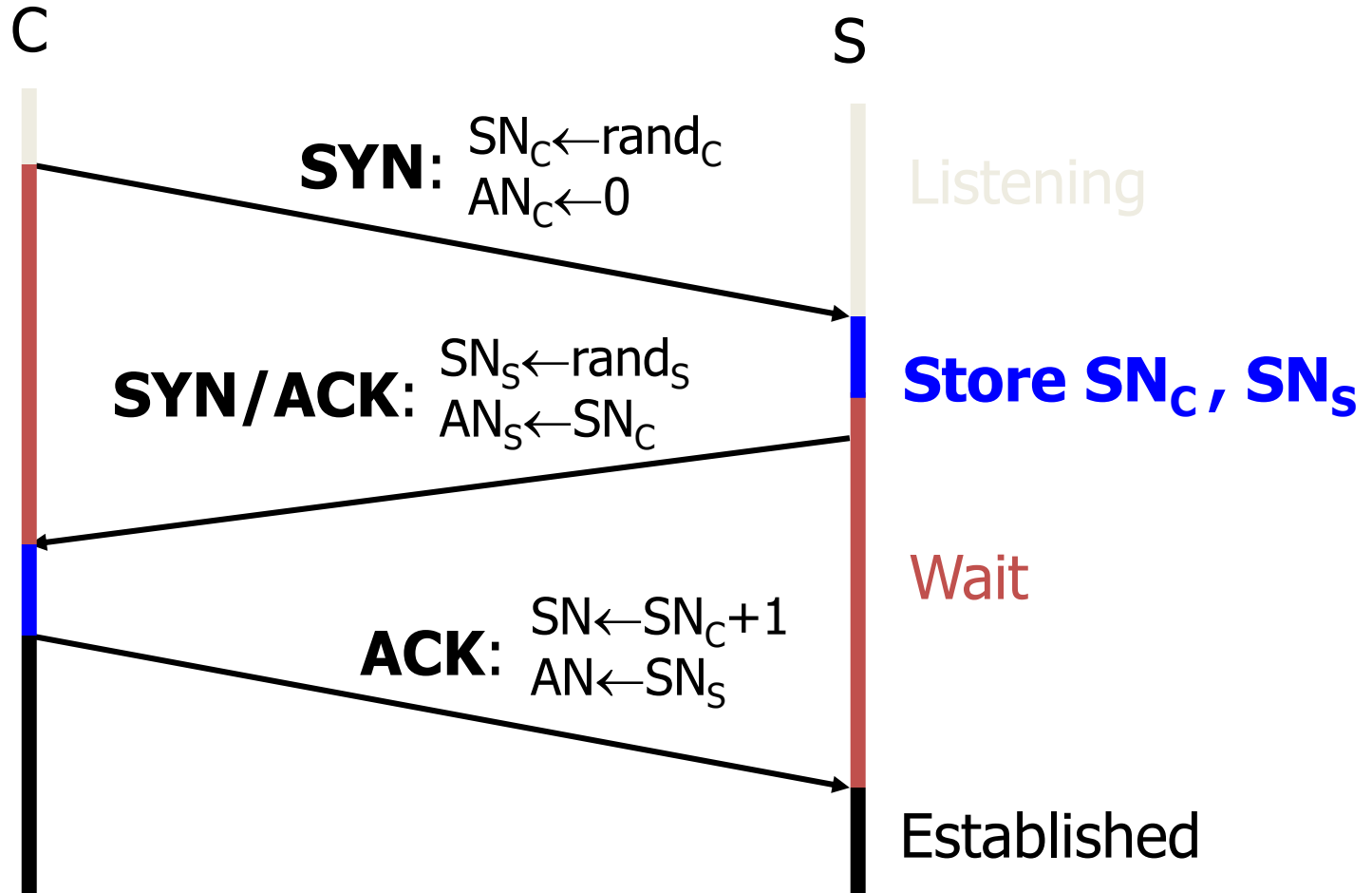


# TCP Header

(protocol=6)



# Review: TCP Handshake



Received packets with SN too far out of window are dropped

<http://packetlife.net/blog/2010/jun/7/understanding-tcp-sequence-acknowledgment-numbers/>

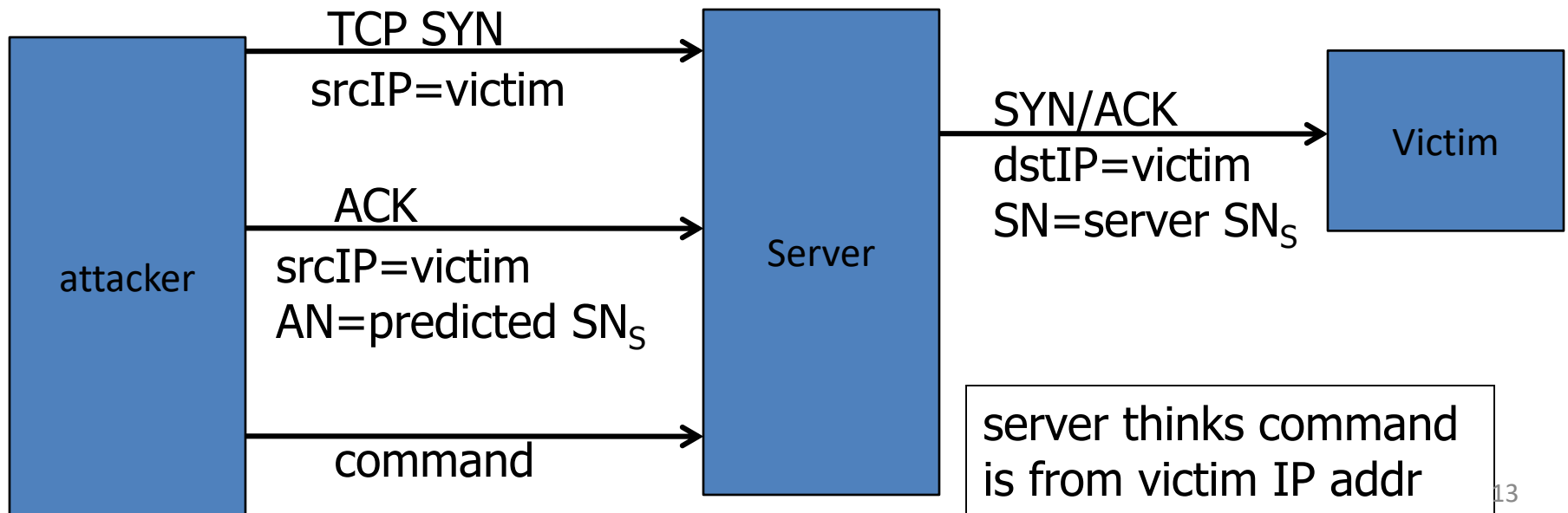
# Basic Security Problems

1. Network packets pass by untrusted hosts
  - Eavesdropping, packet sniffing
  - Especially easy when attacker controls a machine close to victim (e.g. WiFi routers)
2. TCP state easily obtained by eavesdropping
  - Enables spoofing and session hijacking
3. Denial of Service (DoS) vulnerabilities
  - DDoS lecture

# Why random initial sequence numbers?

Suppose initial seq. numbers ( $SN_C$ ,  $SN_S$ ) are predictable:

- Attacker can create TCP session on behalf of forged source IP
  - Random seq. num. does not block attack, but makes it harder



# Example DoS vulnerability: Reset

- Attacker sends a Reset packet to an open socket
  - If correct  $SN_S$  then connection will close  $\Rightarrow$  DoS
  - Naively, success prob. is  $1/2^{32}$  (32-bit seq. #'s).
    - ... but, many systems allow for a large window of acceptable seq. #'s. Much higher success probability.
  - Attacker can flood with RST packets until one works
- Most effective against long lived connections, e.g. BGP

# HTTPS

# Goals for this lecture

## Brief overview of HTTPS:

- How the SSL/TLS protocol works (very briefly)
- How to use HTTPS

## Integrating HTTPS into the browser

- Lots of user interface problems to watch for



# Threat Model: Network Attacker

## Network Attacker:



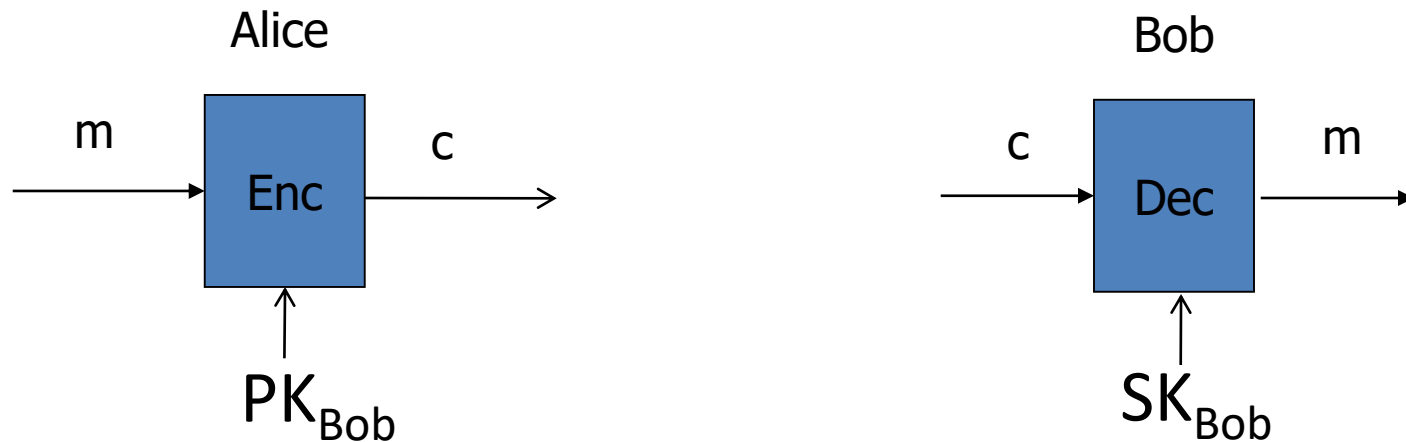
- Controls network infrastructure: Routers, DNS
- Eavesdrops, injects, blocks, modifies packets

## Examples:

- Wireless network at Internet Café
- Internet access at hotels (untrusted ISP)

# SSL/TLS overview

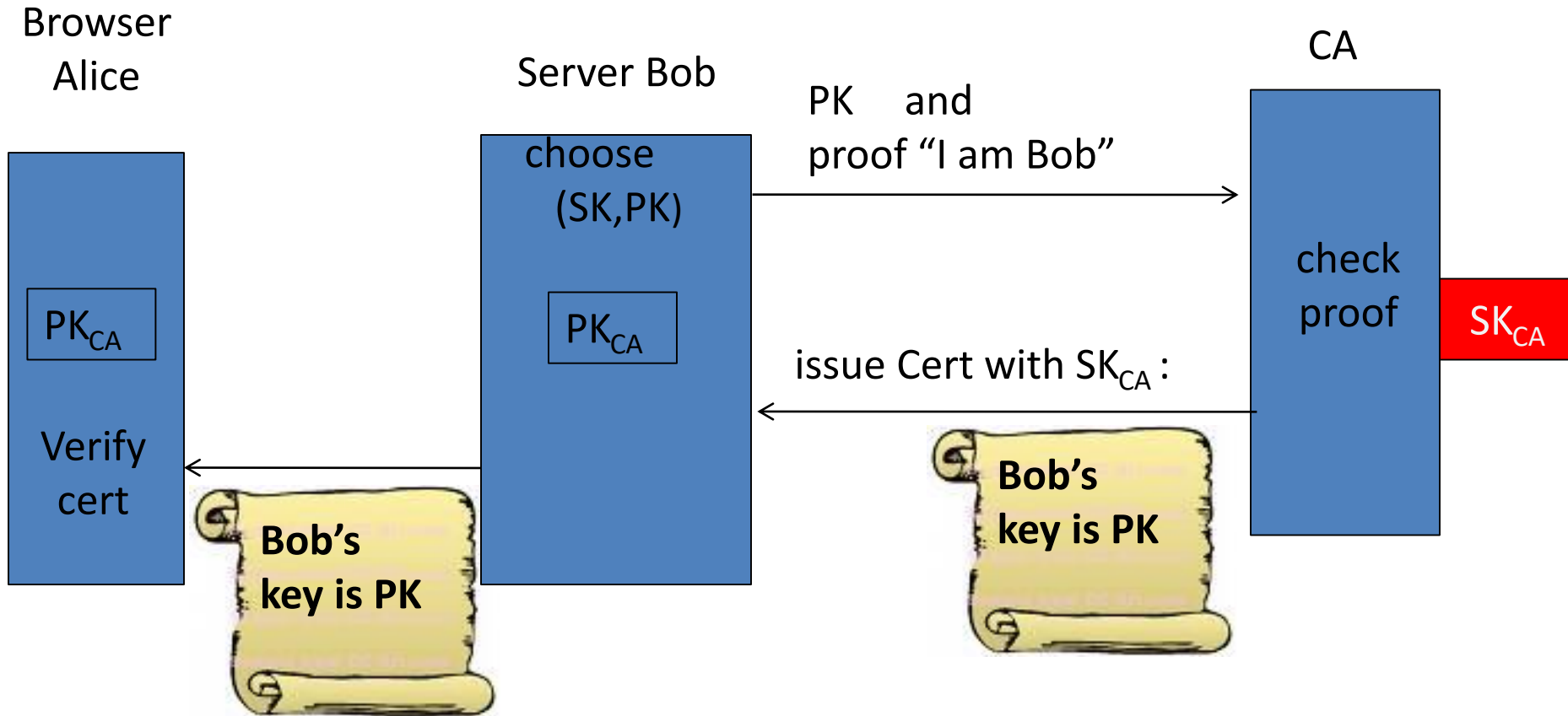
## Public-key encryption:



- Bob generates  $(SK_{Bob}, PK_{Bob})$
- Alice: using  $PK_{Bob}$  encrypts messages and only Bob can decrypt

# Certificates

How does Alice (browser) obtain  $PK_{\text{Bob}}$  ?




**Bob uses Cert for an extended period** (e.g. one year)

# Certificates: example

## Important fields:

<b>Serial Number</b>	5814744488373890497	←
<b>Version</b>	3	
<b>Signature Algorithm</b>	SHA-1 with RSA Encryption ( 1.2.840.113549.1.1.5 )	
<b>Parameters</b>	none	
<b>Not Valid Before</b>	Wednesday, July 31, 2013 4:59:24 AM Pacific Daylight Time	
<b>Not Valid After</b>	Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time	
<b>Public Key Info</b>		
<b>Algorithm</b>	Elliptic Curve Public Key ( 1.2.840.10045.2.1 )	
<b>Parameters</b>	Elliptic Curve secp256r1 ( 1.2.840.10045.3.1.7 )	
<b>Public Key</b>	65 bytes : 04 71 6C DD E0 0A C9 76 ...	←
<b>Key Size</b>	256 bits	
<b>Key Usage</b>	Encrypt, Verify, Derive	
<b>Signature</b>	256 bytes : 8A 38 FE D6 F5 E7 F6 59 ...	←

Equifax Secure Certificate Authority  
↳ GeoTrust Global CA  
↳ Google Internet Authority G2  
↳ mail.google.com

 **mail.google.com**  
Issued by: Google Internet Authority G2  
Expires: Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time  
✓ This certificate is valid

▼ Details

**Subject Name** \_\_\_\_\_  
**Country** US  
**State/Province** California  
**Locality** Mountain View  
**Organization** Google Inc  
**Common Name** mail.google.com ←

**Issuer Name** \_\_\_\_\_  
**Country** US  
**Organization** Google Inc  
**Common Name** Google Internet Authority G2

# Certificates on the web

Subject's CommonName can be:

- An explicit name, e.g. `cs.stanford.edu`, or
- A wildcard cert, e.g. `*.stanford.edu` or `cs*.stanford.edu`

matching rules:

“\*” must occur in leftmost component, does not match “.”

example: `*.a.com` matches `x.a.com` but not `y.x.a.com`

(as in RFC 2818: “HTTPS over TLS”)

# Certificate Authorities



Browsers accept  
certificates from a  
large number of CAs

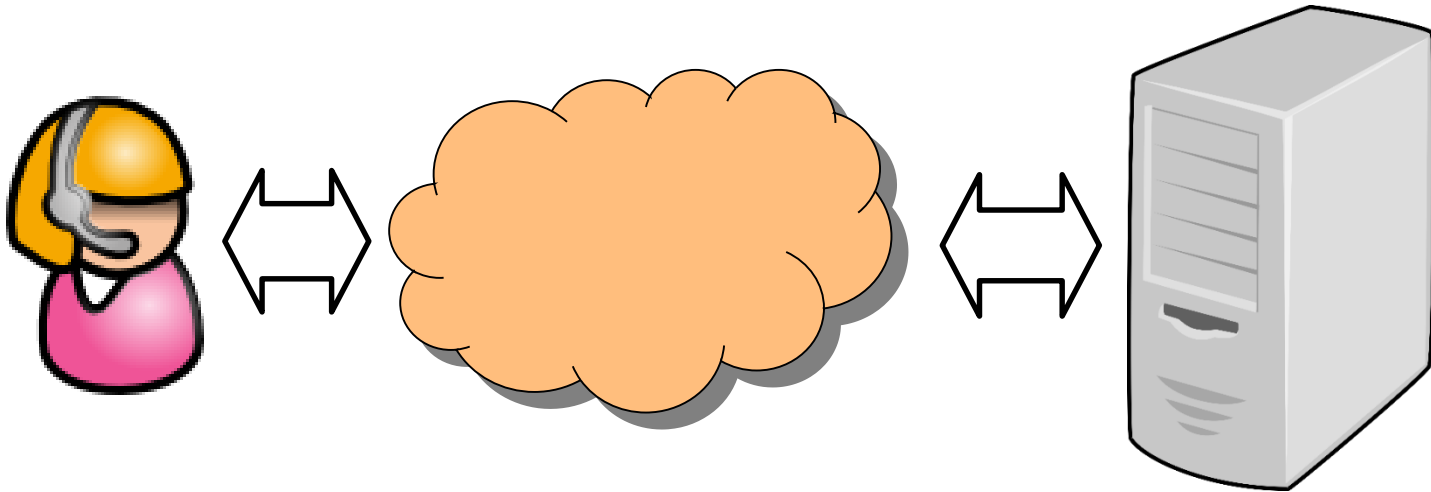
Top level CAs  $\approx$  60

Intermediate CAs  $\approx$  1200

	Entrust.net C...Authority (2048)	Jul 24, 2029 7:15:12 AM
	Entrust.net S...ification Authority	May 25, 2019 9:39:40 AM
	ePKI Root Certification Authority	Dec 19, 2034 6:31:27 PM
	Equifax Secu...rtificate Authority	Aug 22, 2018 9:41:51 AM
	Equifax Secure eBusiness CA-1	Jun 20, 2020 9:00:00 PM
	Equifax Secure eBusiness CA-2	Jun 23, 2019 5:14:45 AM
	Equifax Secu...l eBusiness CA-1	Jun 20, 2020 9:00:00 PM
	Federal Common Policy CA	Dec 1, 2030 8:45:27 AM
	FNMT Clase 2 CA	Mar 18, 2019 8:26:19 AM
	GeoTrust Global CA	May 20, 2022 9:00:00 PM
	GeoTrust Pri...ification Authority	Jul 16, 2036 4:59:59 PM
	Global Chambersign Root	Sep 30, 2037 9:14:18 AM



# Strawman SSL: Need for DHE

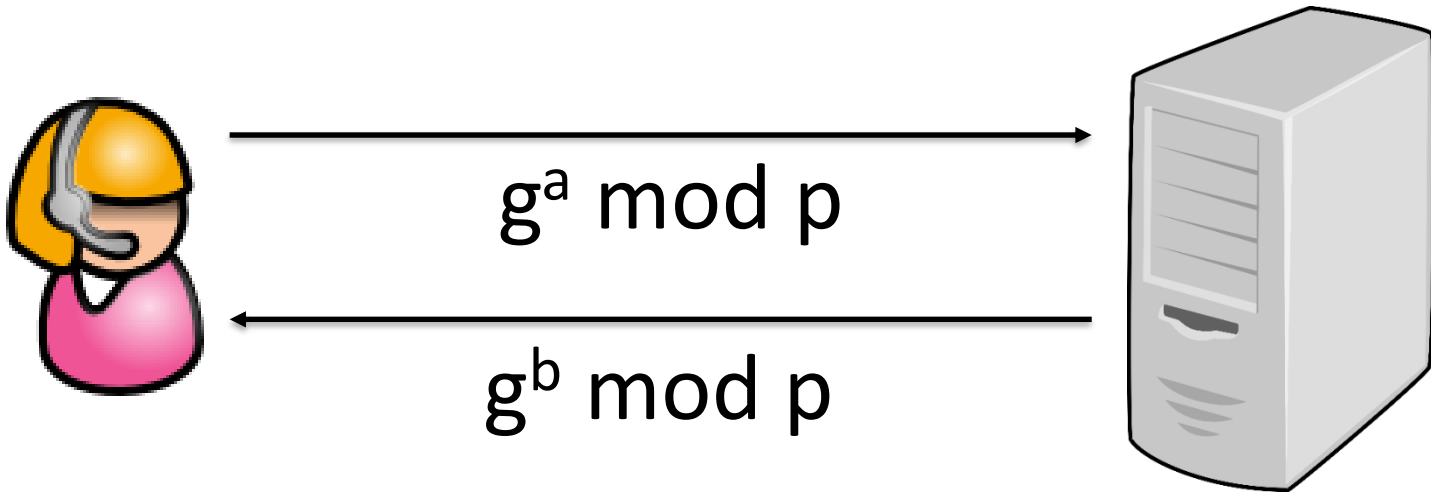


- Alice gets public key of webserver from CA
- Sends session key encrypted using this pubkey
- Server and alice communicate using this key

Problems with this protocol?

- What if server private key is compromised?

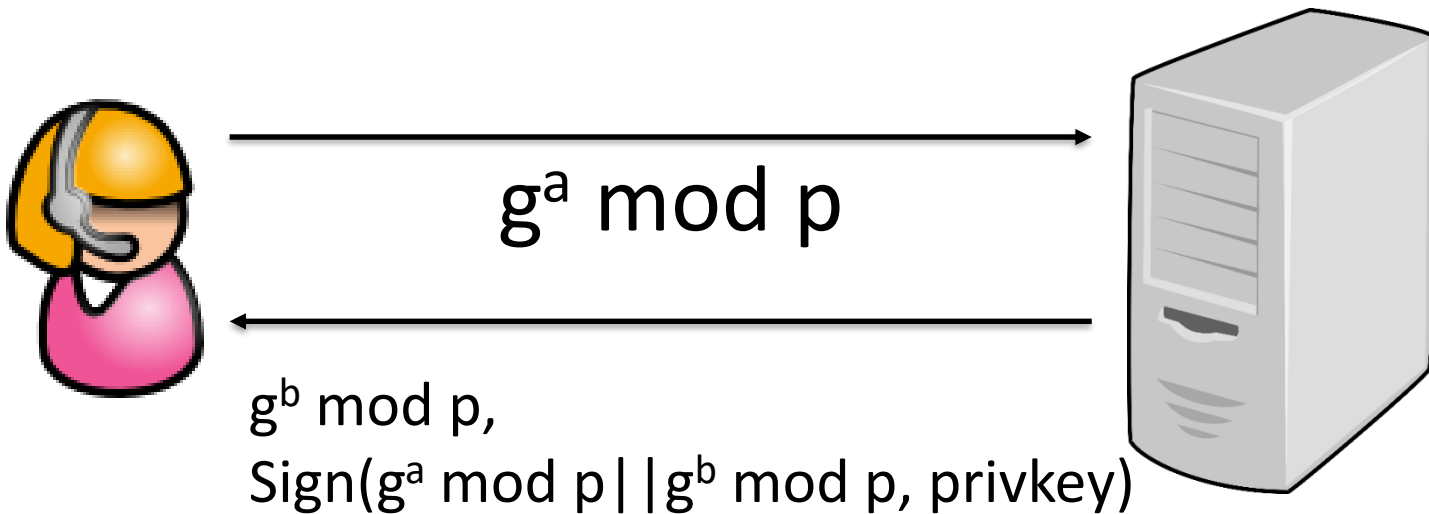
# Diffie-Hellman Key Exchange



- $p$  is a prime,  $g$  is called a generator
- After exchange, both parties know  $g^{ab} \bmod p$
- More importantly, nobody else knows  $g^{ab}$
- This holds even if privkey is compromised **in future**
- Satisfies property of **forward secrecy**



# Authenticated DHE



- Server signs DH parameters with private key
- Prevents MITM attacks
- Why do we need both  $g^a$  and  $g^b$  in the sign?
  - To prevent replays

# SSL

- Connection: a transport that provides a service. Every connection is associated with a session
- Session: association between a client and a server. Created by the Handshake protocol.

# SSL Main Ideas

Phase I: setup connection

- [CS] Negotiate params, encryption suites etc.
- [C] Verify server public key using CA
- [CS] Perform a key exchange
- Derive symmetric keys for this session

Phase II: actually transmit data

# SSL State Information

- SSL session is stateful → SSL protocol must initialize and maintain session state information on either side of the session
- SSL session can be used for several connections → connection state information

# SSL Session State Information: Elements

- Session ID: chosen by the server to identify an active or resumable session state
- Peer certificate: certificate for peer entity (X.509)
- Compression method: algorithm to compress data before encryption
- Cipher spec: specification of data encryption and Message Authentication Code (MAC) algorithms
- Master secret: 48-byte secret shared between client and server
- Is resumable: flag that indicates whether the session can be used to initiate new connections

# SSL Connection State Information: Elements

- Server and client random: 32 bytes sequences that are chosen by server and client for each connection
- Server write MAC secret: secret used for MAC on data written by server
- Client write MAC secret: secret used for MAC on data written by client
- Server write key: key used for data encryption by server and decryption by client
- Client write key: key used for encryption by client and decryption by server
- Initialization vector: for CBC block ciphers
- Sequence number: for both transmitted and received messages, maintained by each party

# SSL Connection State

- Four parts to state
  - Current read state
  - Current write state
  - Pending read state
  - Pending write state
- Handshake:
  - Initial current state is empty
  - Pending state can be made current or reinitialized to empty

# SSL Protocol

Components:

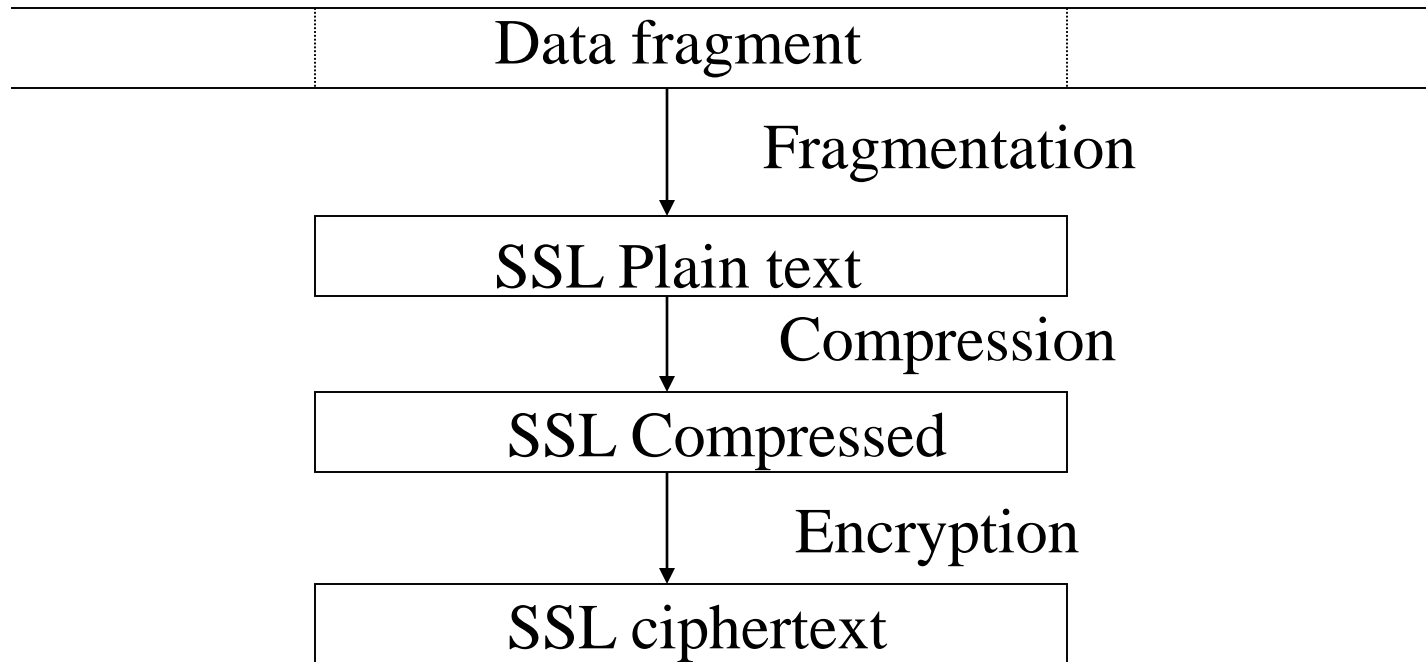
- **SSL Record Protocol**
  - Layered on top of a connection-oriented and reliable transport layer service
  - Provides message origin authentication, data confidentiality, and data integrity
- **SSL sub-protocols**
  - Layered on top of the SSL Record Protocol
  - Provides support for SSL session and connection establishment



# SSL Record Protocol

- Receives data from higher layer SSL sub-protocols
- Addresses
  - Data fragmentation
  - Compression
  - Authentication
  - Encryption

# SSL Record Protocol



# SSL Record Content

- Content type
  - Defines higher layer protocol that must be used to process the payload data (8 bits, only 4 defined)
- Protocol version number
  - Defines SSL version in use (8 bits major, 8 bits minor)
- Length:  $\max 2^{14} + 2048$
- Data payload
  - Optionally compressed and encrypted
  - Encryption and compression requirements are defined during SSL handshake
- MAC
  - Appended for each each record for message origin authentication and data integrity verification

# SSL Sub-protocols

- Alert Protocol
  - Used to transmit alerts via SSL Record Protocol
  - Alert message: (alert level, alert description)
- Handshake Protocol – Complex
  - Used to mutually authenticate client and server and exchange session key
  - Establish new session and connection together or
  - Uses existing session for new connection

# SSL Sub-protocols

- ChangeCipherSpec Protocol
  - Used to change cipher specifications
  - Can be changed at the end of the handshake or later
- Application Protocol
  - Used to directly pass application data to the SSL Record Protocol

# SSL Handshake

- Phase 1: establish security capabilities
- Phase 2: server authentication and key exchange
- Phase 3: client authentication and key exchange
- Phase 4: finish

# SSL Handshake

## Phase 1

Security capabilities

1. C → S: **CLIENTHELLO**
  2. S → C: **SERVERHELLO**
- 

## Phase 2

Optional server messages

[**CERTIFICATE**]  
[**SERVERKEYEXCHANGE**]  
[**CERTIFICATEREQUEST**]  
**SERVERHELLODONE**

---

## Phase 3

Client key exchange

3. C → S: [**CERTIFICATE**]  
**CLIENTKEYEXCHANGE**  
[**CERTIFICATEVERIFY**]
- 

## Phase 4

Change cipher suite

4. S → C: **CHANGE\_CIPHER\_SPEC**  
**FINISH**

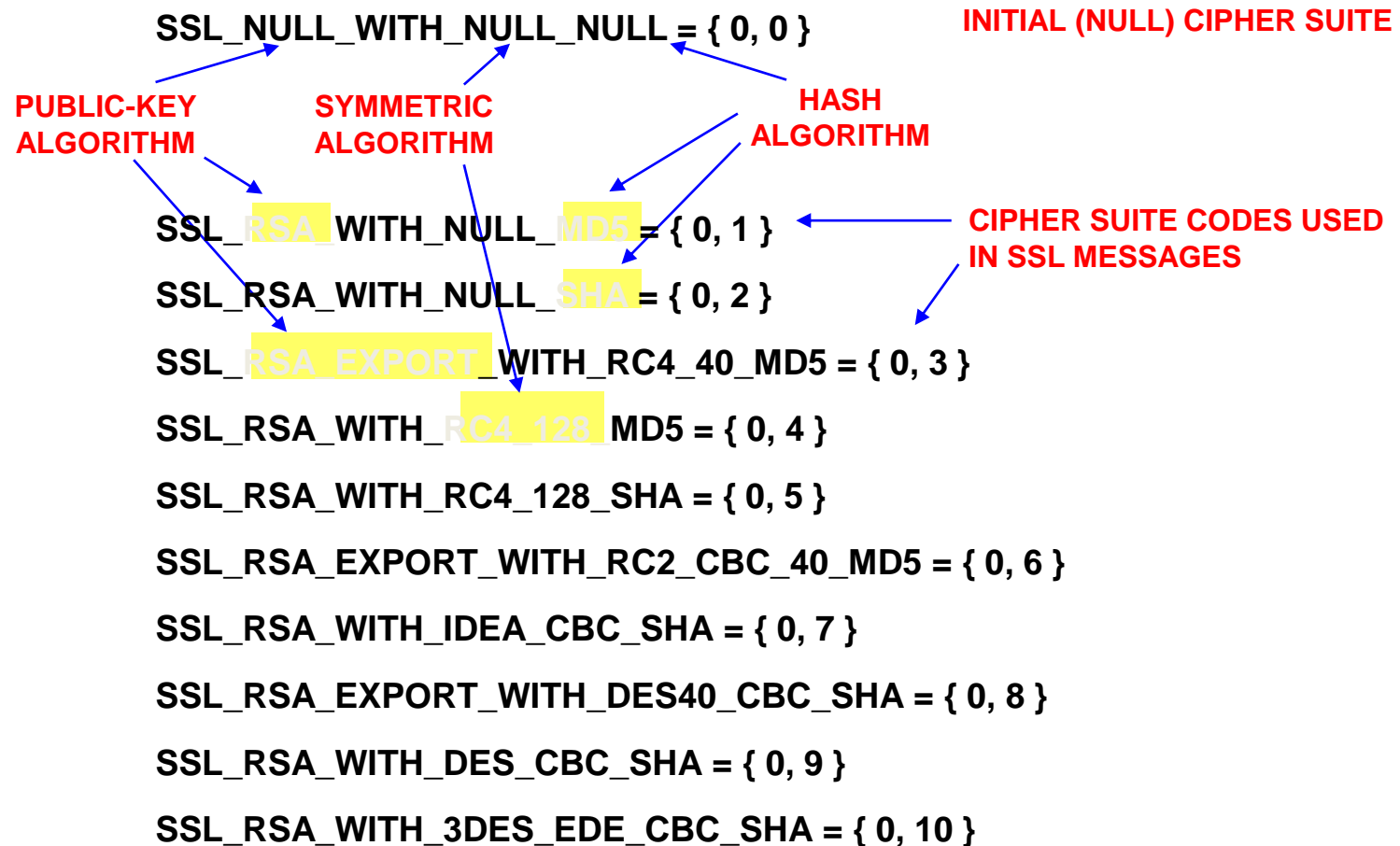
1. C → S: CLIENTHELLO

# SSL Handshake

- CLIENTHELLO message is sent by the client
  - When the **client wants to establish a TCP connection** to the server,
  - When a **HELLOREQUEST message is received**, or
  - When client wants to **renegotiate security parameters** of an existing connection
- Message content:
  - Number of highest SSL understood by the client
  - Client's random structure (32-bit timestamp and 28-byte pseudorandom number)
  - Session ID client wishes to use (ID is empty for existing sessions)
  - List of cipher suits the client supports
  - List of compression methods the client supports



# Cipher Suites



2. S → C: SERVERHELLO  
[CERTIFICATE]  
[SERVERKEYEXCHANGE]  
[CERTIFICATEREQUEST]  
SERVERHELLODONE

# SSL Handshake

- Server processes CLIENTHELLO message
- Server Respond to client with SERVERHELLO message:
  - Server version number: lower version of that suggested by the client and the highest supported by the server
  - Server's random structure: 32-bit timestamp and 28-byte pseudorandom number
  - Session ID: corresponding to this connection
  - Cipher suite: selected by the server for client's list
  - Compression method: selected by the server from client's list

2. S → C: SERVERHELLO  
[CERTIFICATE]  
[SERVERKEYEXCHANGE]  
[CERTIFICATEREQUEST] }  
SERVERHELLODONE

# SSL Handshake

Optional messages: Phase 2– server authentication

- CERTIFICATE:
  - If the server is using certificate-based authentication
  - May contain RSA public key → good for key exchange
- SERVERKEYEXCHANGE:
  - If the client does not have certificate, has certificate that can only be used to verify digital signatures, or uses FORTEZZA token-based key exchange (not recommended)
- CERTIFICATEREQUEST:
  - Server may request personal certificate to authenticate a client

3. C → S: [CERTIFICATE]  
CLIENTKEYEXCHANGE  
[CERTIFICATEVERIFY]  
CHANGE\_CIPHER\_SPEC  
FINISH

# SSL Handshake

- Client processing:
  - Verifies site certification
    - Valid site certification if the server's name matches the host part of the URL the client wants to access
  - Checks security parameters supplied by the SERVERHELLO

3. C → S: [CERTIFICATE]  
CLIENTKEYEXCHANGE  
[CERTIFICATEVERIFY]  
CHANGECIPHERSPEC  
FINISH

# SSL Handshake

- Client messages: Phase 3 – client authentication and key exchange
  - CERTIFICATE
    - If server requested a client authentication, client sends
  - CLIENTKEYEXCHANGE
    - Format depends on the key exchange algorithm selected by the server
      - RSA: 48-byte premaster secret encrypted by the server's public key
      - Diffie-Hellman: public parameters between server and client in SERVERKEYEXCHANGE and CLIENTKEYEXCHANGE msgs.
      - FORTEZZA: token-based key exchange based on public and private parameters
    - Premaster key is transformed into a 48-byte master secret, stored in the session state

3. C → S: [CERTIFICATE]  
CLIENTKEYEXCHANGE  
[CERTIFICATEVERIFY]  
CHANGECIPHERSPEC  
FINISH

# SSL Handshake

- Client messages:
  - CERTIFICATEVERIFY
    - If client authentication is required
    - Provides explicit verification of the use's identity (personal certificate)
  - CHANGECIPHERSPEC
    - Completes key exchange and cipher specification
  - FINISH
    - Encrypted by the newly negotiated session key
    - Verifies that the keys are properly installed in both sites

4. S → C: CHANGECIPHERSPEC  
FINISH

# SSL Handshake

- Phase 4: finish
- Server finishes handshake by sending CHANGECIPHERSPEC and FINISH messages
- After SSL handshake completed a secure connection is established to send application data encapsulated in SSL Record Protocol

# SSL Handshake to Resume Session

1. C → S: **CLIENTHELLO**
2. S → C: **SERVERHELLO**  
**CHANGECIPHERSPEC**  
**FINISH**
3. C → S: **CHANGECIPHERSPEC**  
**FINISH**



# SSL Protocol

- Provides secure TCP connection between client and server by
  - Server authentication
  - Optional client authentication
  - Key exchange services
  - Negotiation
  - Data confidentiality and integrity
  - Message authentication
  - Compression/decompression

# SSL Delay

- Slower than a TCP session (2-10 times)
- Causes:
  - Handshake phase
    - Client does public-key encryption
    - Server does private-key encryption (still public-key cryptography)
    - Usually clients have to wait on servers to finish
  - Data Transfer phase
    - Symmetric key encryption

# Firewall Tunneling

- SSL/TSL: end-to-end security → difficult to interoperate with application gateways
- Firewalls: man-in-the-middle
  - Application protocol being proxied
  - Application protocol being tunneled

# Proxied Protocol

- Proxy server is aware of the specifics of the protocol and understand protocol level processing
- Support:
  - Protocol-level filtering
  - Access control
  - Accounting
  - Logging
- Usually proxied protocols: telnet, ftp, http

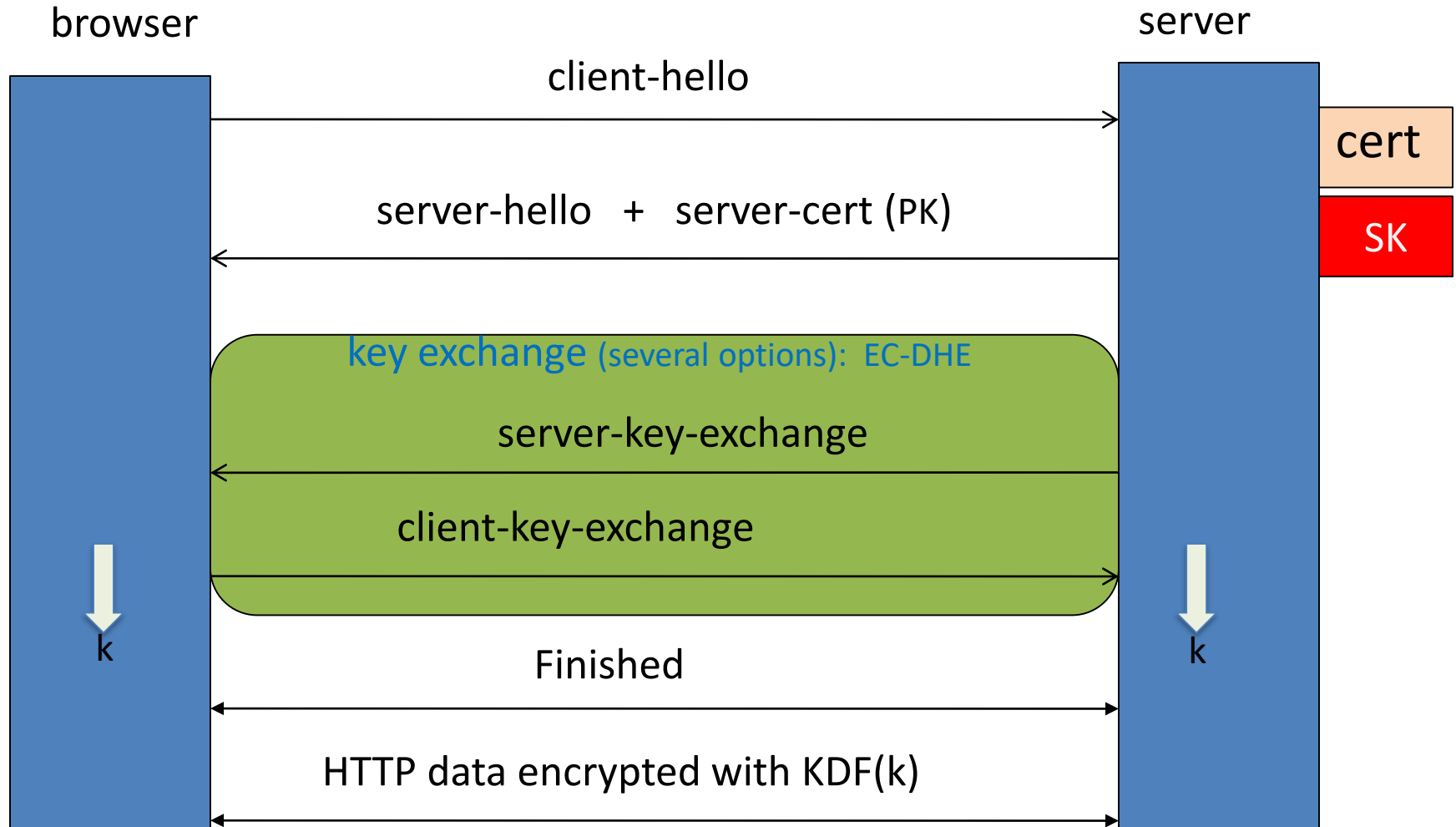
# Tunneled Protocol

- Proxy server:
  - NOT aware of the specifics of the protocol → simply relaying the data between Client and Server
  - Does NOT have access to data being transferred
  - Knows: source and destination addresses (IP and port) and the requesting user (if authentication is supported)
- Cannot support: protocol –level filtering, access control, and logging at the same extend as the proxied version.
- Usually tunneled protocols: SSL-enhanced protocols

# Summary

- Advantages of SSL/TSL:
  - Simplicity
  - Wide deployment
- Disadvantages:
  - Do not secure UDP
  - Work poorly with applications gateways

# Brief overview of SSL/TLS



Most common: server authentication only

# Integrating SSL/TLS with HTTP: HTTPS

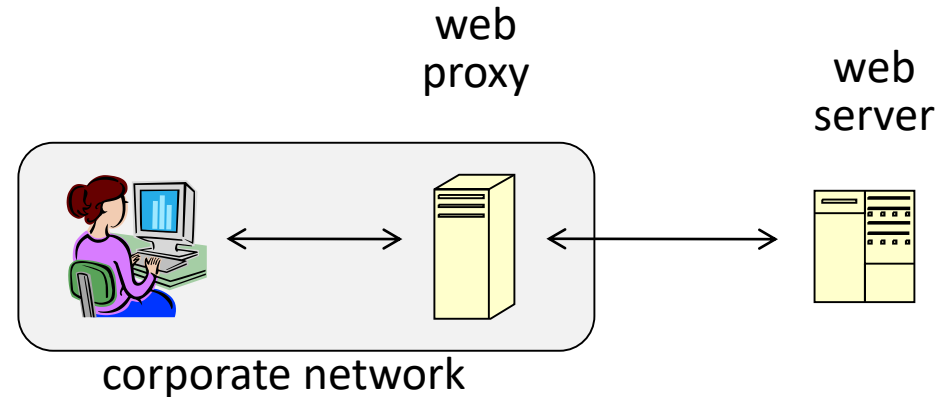
## Two complications

### Web proxies

solution: browser sends

**CONNECT domain-name**

before client-hello



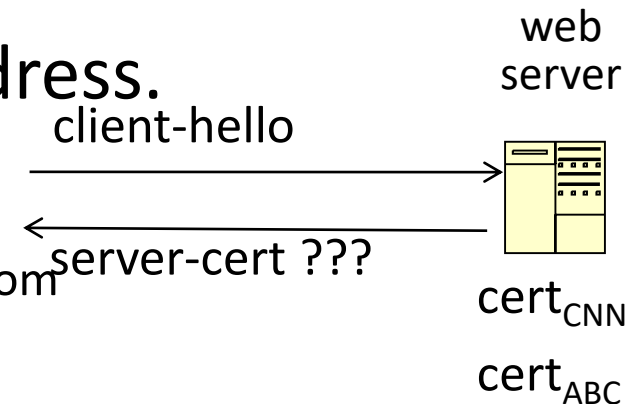
### Virtual hosting:

two sites hosted at same IP address.

solution in TLS 1.1: **SNI** (June 2003)

client\_hello\_extension: server\_name=cnn.com

implemented since FF2 and IE7 (vista)





# HTTPS in the Browser

# The lock icon: SSL indicator



## Intended goal:

- Provide user with identity of page origin
- Indicate to user that page contents were not viewed or modified by a **network attacker**



In reality: many problems (next few slides)

# When is the (basic) lock icon displayed

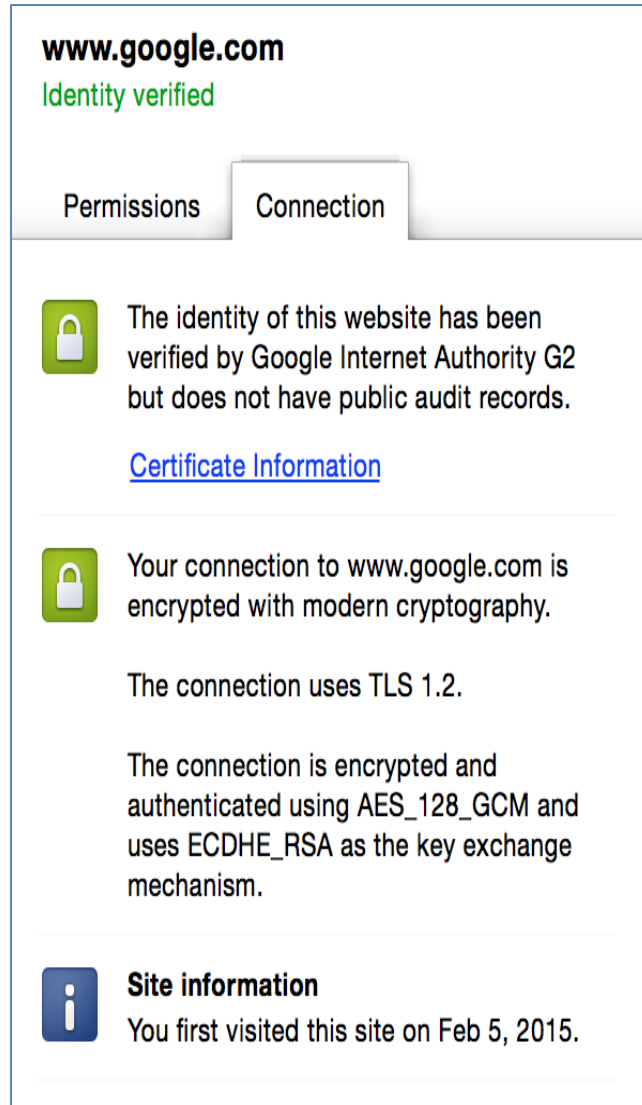


All elements on the page fetched using HTTPS

For all elements:

- HTTPS cert issued by a CA trusted by browser
- HTTPS cert is valid (e.g. not expired)
- CommonName in cert matches domain in URL

# The lock UI: help users authenticate site



} uninformative



# The lock UI: Extended Validation Certs

## Harder to obtain than regular certs

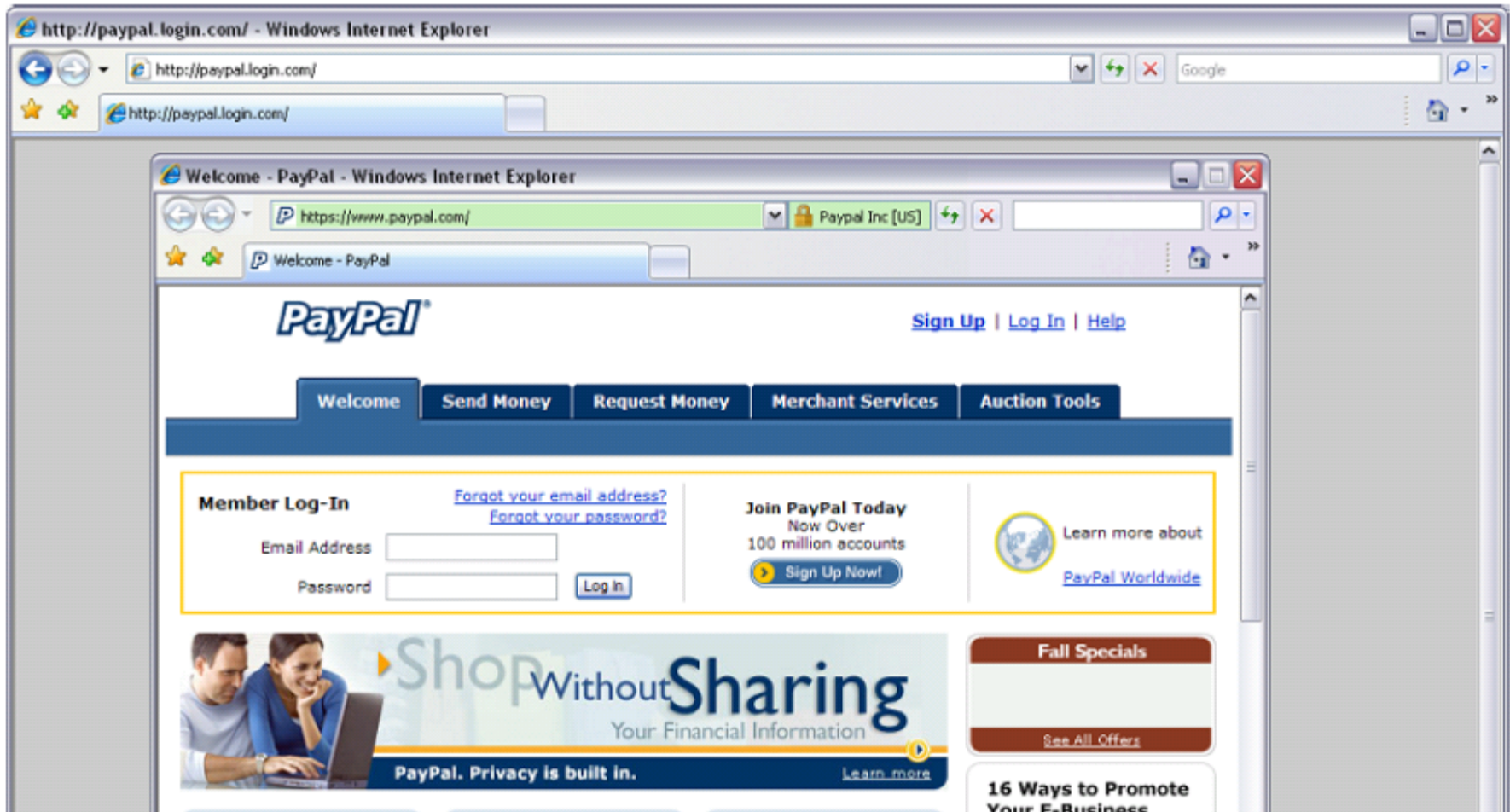
- requires human at CA to approve cert request
- no wildcard certs (e.g. \*.stanford.edu )

Helps block “semantic attacks”:  
www.bankofthevest.com



note: HTTPS-EV and HTTPS are in the same origin

# A general UI attack: picture-in-picture



Trained users are more likely to fall victim to this [JSTB'07]

# HTTPS and login pages: incorrect usage

Users often land on login page over HTTP:

- Type HTTP URL into address bar
- Google links to HTTP page

View source:

```
<form method="post"
```

```
action="https://onlineservices.wachovia.com/..."
```

(old site)

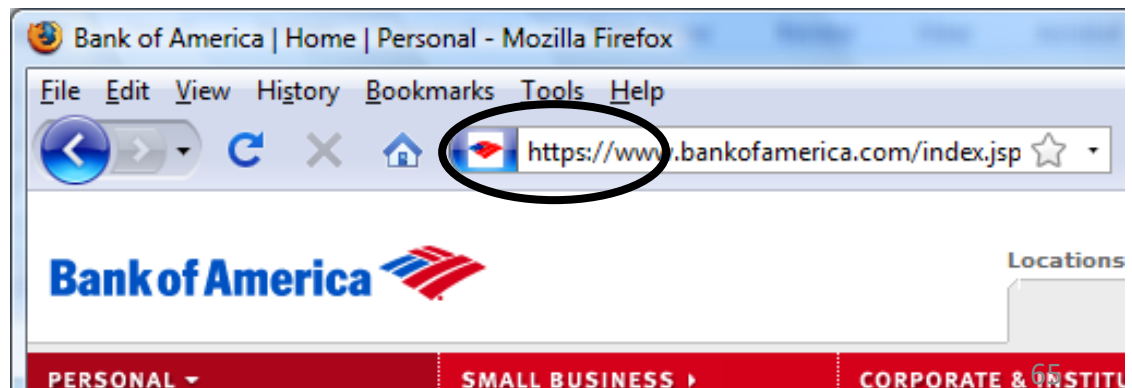


# HTTPS and login pages: guidelines

## General guideline:

Response to <http://login.site.com>

should be Redirect: <https://login.site.com>





# Problems with HTTPS and the Lock Icon

# Problems with HTTPS and the Lock Icon

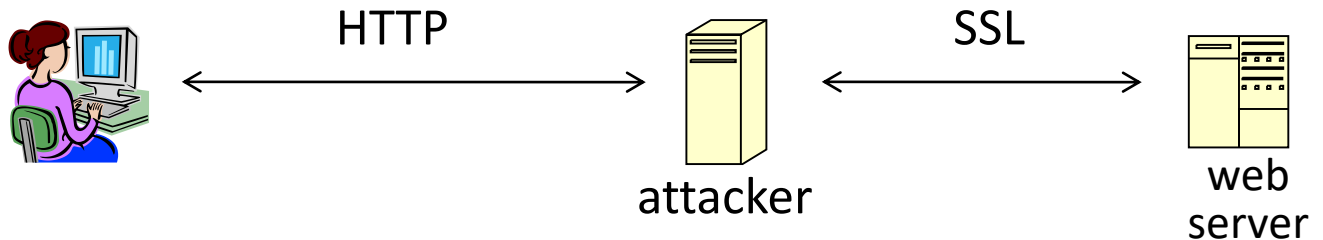
1. Upgrade from HTTP to HTTPS
2. Forged certs
3. Mixed content: HTTP and HTTPS on the same page
4. Does HTTPS hide web traffic?
  - Problems: traffic analysis, compression attacks

# 1. HTTP → HTTPS upgrade

Common use pattern:

- browse site over HTTP; move to HTTPS for checkout
- connect to bank over HTTP; move to HTTPS for login

**SSL\_strip attack:** prevent the upgrade [Moxie'08]



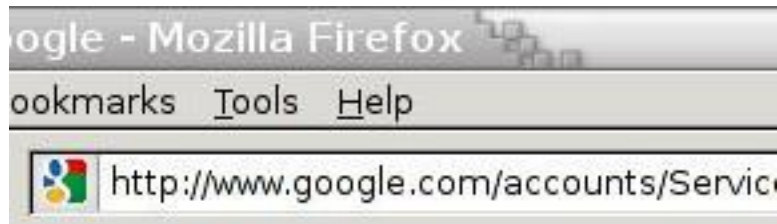
`<a href=https://...>`  $\Rightarrow$  `<a href=http://...>`

Location: `https://...`  $\Rightarrow$  Location: `http://...` (redirect)

`<form action=https://... >`  $\Rightarrow$  `<form action=http://...>`

# Tricks and Details

Tricks: drop-in a clever fav icon (older browsers)



⇒



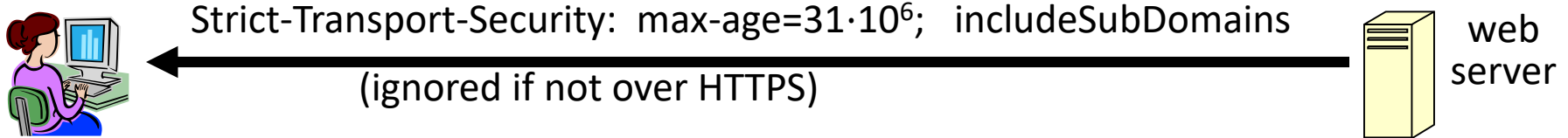
⇒ fav icon no longer presented in address bar



More tricks: inject “Set-cookie” headers to delete existing session cookies in browser. Force login.

Number of users who detected HTTP downgrade: 0

# Defense: Strict Transport Security (HSTS)



Header tells browser to always connect over HTTPS

Subsequent visits must be over HTTPS (self signed certs result in error)

- Browser refuses to connect over HTTP or self-signed cert
- Requires that entire site be served over HTTPS

HSTS flag deleted when user “clears private data” : security vs. privacy

# CSP: upgrade-insecure-requests

The problem: many pages use ``

- Makes it difficult to migrate site to HTTPS

Solution:

Content-Security-Policy: upgrade-insecure-requests

```
  
  
<a href="http://othersite.com/img">
```



```
  
  
<a href="http://othersite.com/img">
```

Always use protocol relative URLs ``

## 2. Certificates: wrong issuance

2011: Comodo and DigiNotar CAs hacked, issue certs for Gmail, Y! Mail, ...

2013: TurkTrust's intermediate CA issued cert for gmail (discovered by pinning)

Result: Intermediate CA blocked; TurkTrust EV revoked in Chrome; TurkTrust removed from Firefox

2014: Indian NIC (intermediate CA trusted by the root CA IndiaCCA) issue certs  
for Google and Yahoo! domains

Result: (1) India CCA revoked NIC's intermediate certificate

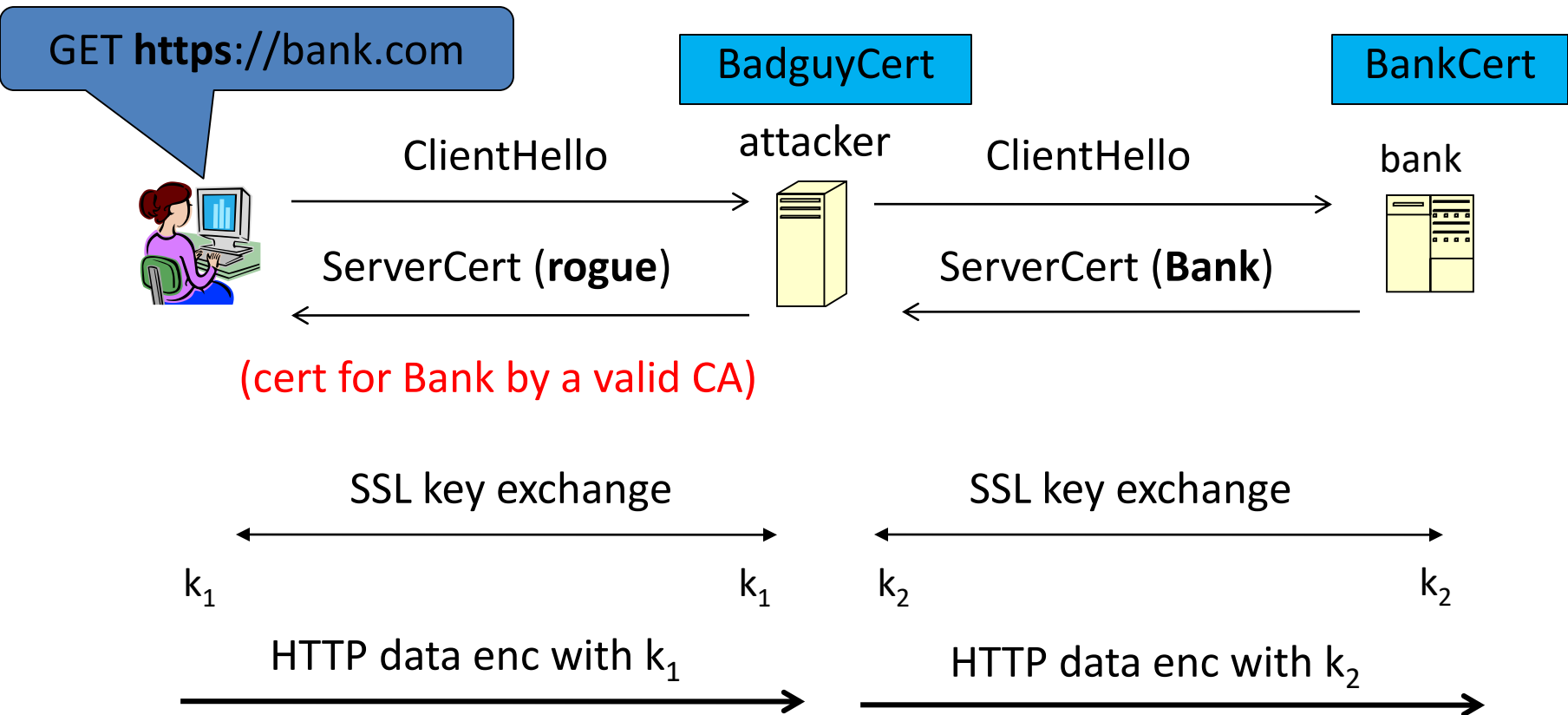
(2) Chrome restricts India CCA root to only seven Indian domains

2015: MCS (intermediate CA cert issued by CNNIC) issues certs for Google domains

Result: CNNIC root no longer recognized by Chrome

⇒ enables eavesdropping w/o a warning on user's session

# Man in the middle attack using rogue cert



Attacker proxies data between user and bank.  
Sees all traffic and can modify data at will.



# What to do? (many good ideas)

## 1. HTTP public-key pinning, TACK

- Let a site declare CAs that can sign its cert (similar to HSTS)
- on subsequent HTTPS, browser rejects certs from other CAs
- TOFU: Trust on First Use

## 2. Certificate Transparency: [LL'12]

- idea: CA's must advertise a log of all certs. they issued
- Browser will only use a cert if it is published on log server
  - Efficient implementation using Merkle hash trees
  - Companies can scan logs to look for invalid issuance

### 3. Mixed Content: HTTP and HTTPS

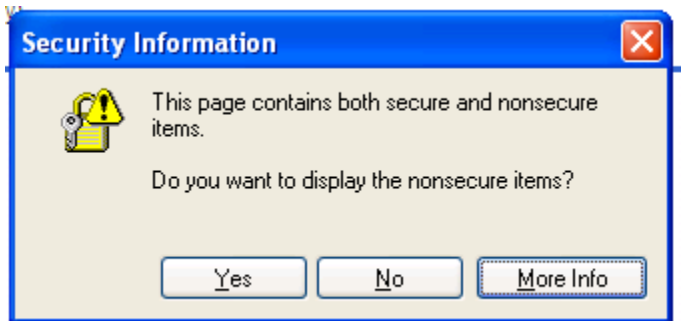
Page loads over HTTPS, but contains HTTP content

(e.g. `<script src="http://.../script.js">` )

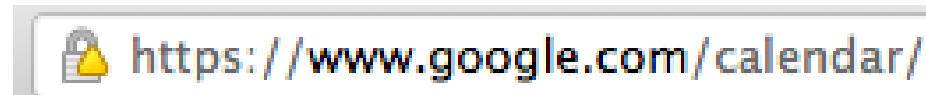
 never write this

⇒ Active network attacker can hijack session  
by modifying script en-route to browser

IE7:



Chrome:



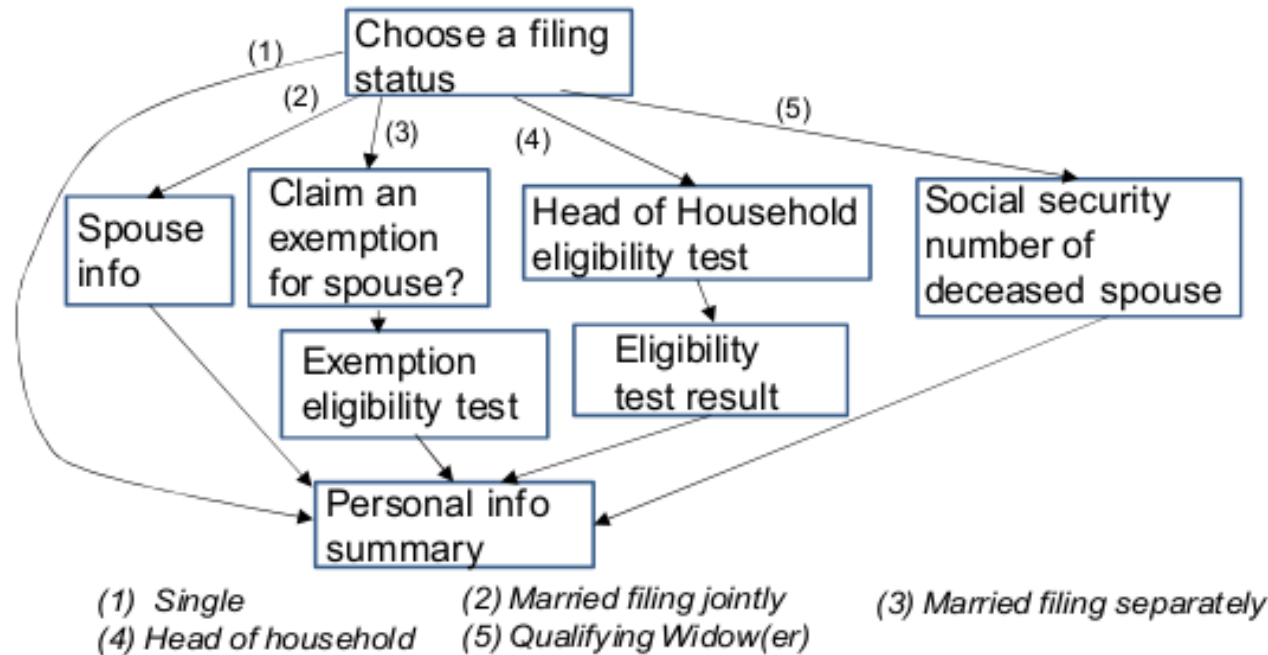
Chrome policy: CSS, script, frame: blocked; images, XHR: allowed

## 4. Peeking through SSL: traffic analysis

- Network traffic reveals length of HTTPS packets
  - TLS supports up to 256 bytes of padding
- AJAX-rich pages have lots and lots of interactions with the server
- These interactions expose specific internal state of the page

[Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow](#). Chen, Wang, Wang, Zhang, Oakland 2010

# Peeking through SSL: an example [CWWZ'10]



Vulnerabilities in an online tax application

No easy fix. Can also be used to ID Tor traffic

# Peeking through SSL: compression [DR'12]

HTTPS: supports compressing data before encryption (16KB records)

Attacker: wants to recover Gmail session cookie (say)

- Places Javascript on some site that issues request:

```
GET gmail.com/__AAAAAAAAAAAAA....AAAAAA  
Cookie: session=__A 6Bh63g53ig4  
Host: gmail.com
```

1<sup>st</sup> byte of cookie is “A” ⇒ record will compress more than when not

- Script tries all possibilities to expose 1<sup>st</sup> byte. Moves to 2<sup>nd</sup> bytes ...

What to do: do not use compression with HTTPS

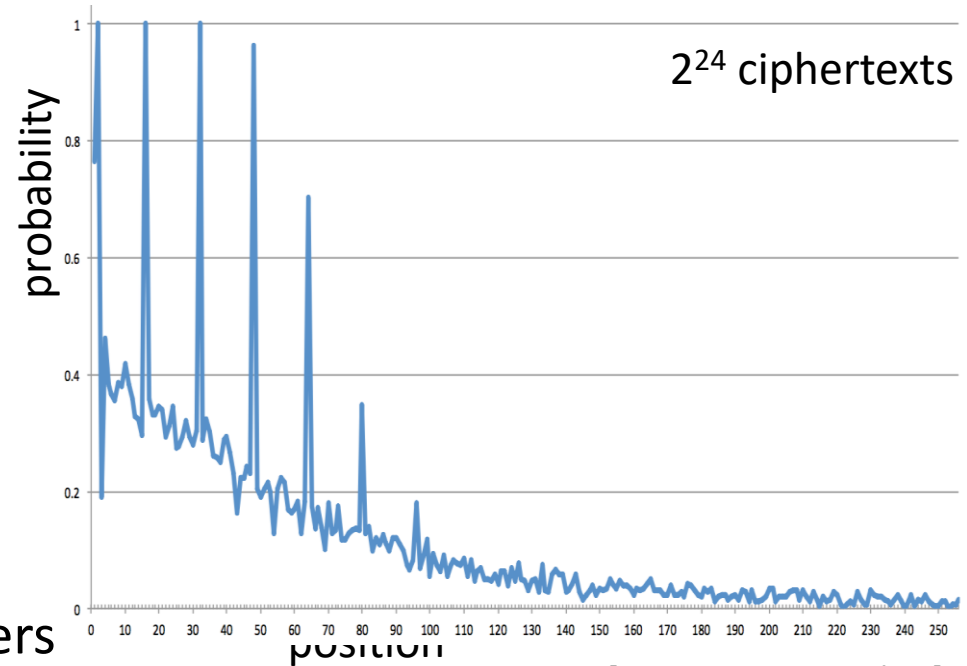
# Peeking through SSL: weak algs.

[ABPPS'13]

RC4: a stream cipher commonly used in HTTPS  
(fast, other options in TLS 1.0 are problematic)

Bad news: [MS'01, M'02, ABPPS'13]

RC4 does not hide  
plaintext well



[Source: ABPPS'13]

What to do:

- Push for TLS 1.2 support in browsers
- If must use RC4, pad HTTP headers so that nothing important in first 512 bytes

# Summary of TLS/SSL

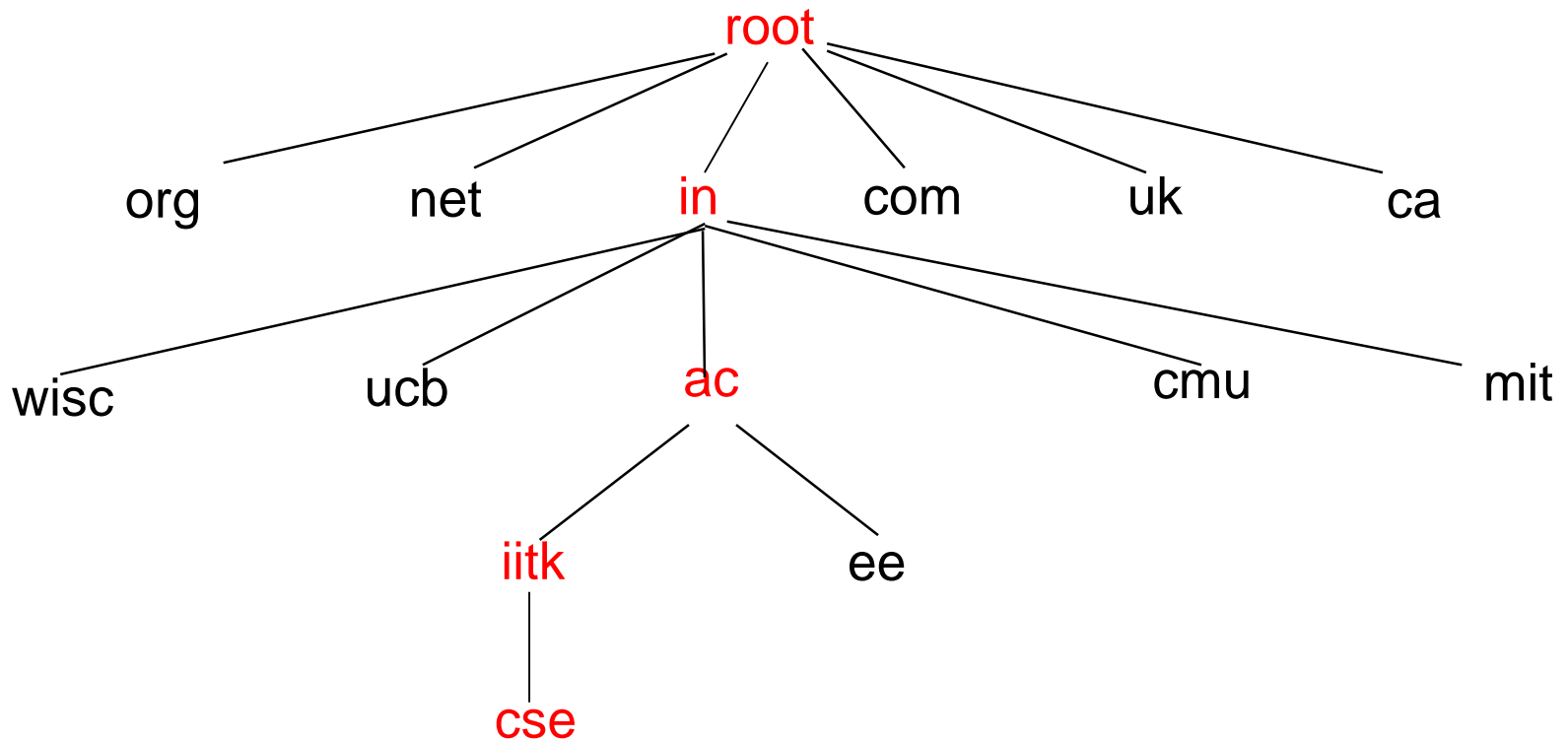
- Very impt component in network security
- You should always use SSL
  - Fairly straightforward w/ libraries (OpenSSL etc.)
  - It's not just for websites  
(<https://www.wired.com/2014/08/isp-bitcoin-theft/>)
- Tons on Attacks on TLS
  - BEAST, Lucky13, CRIME, Freak, DROWN
  - Beware of the subtleties – this stuff is hard!

# Domain Name System



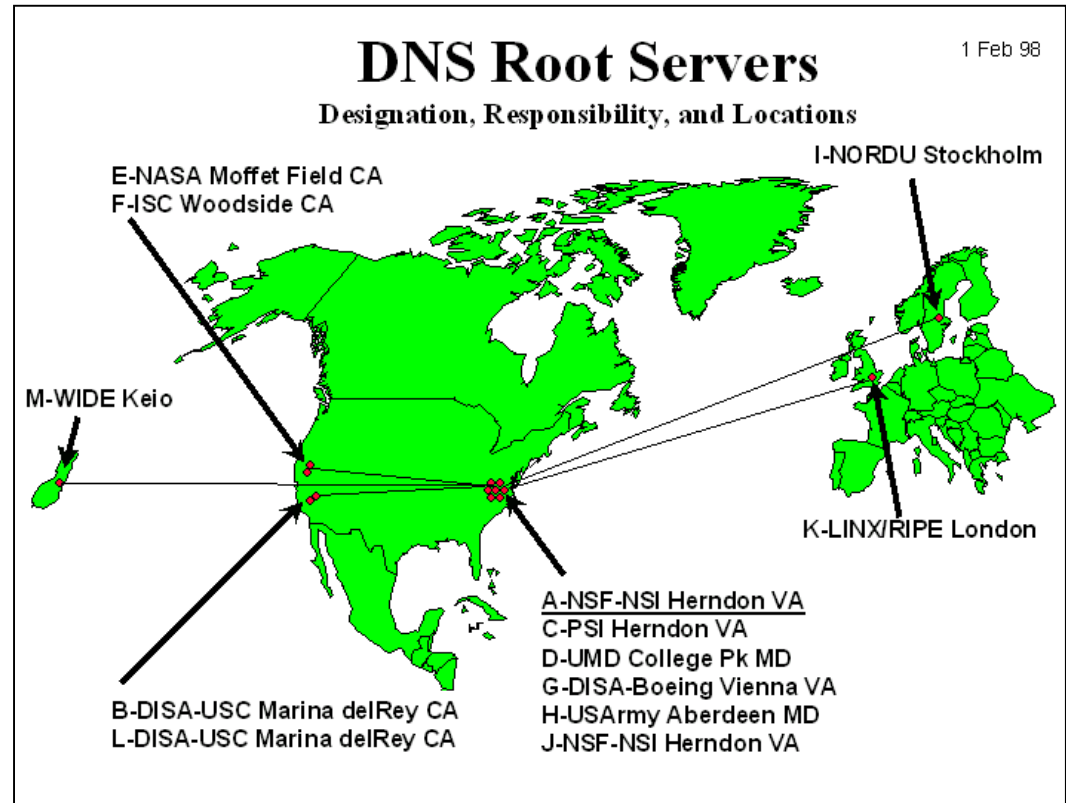
# Domain Name System

- Hierarchical Name Space

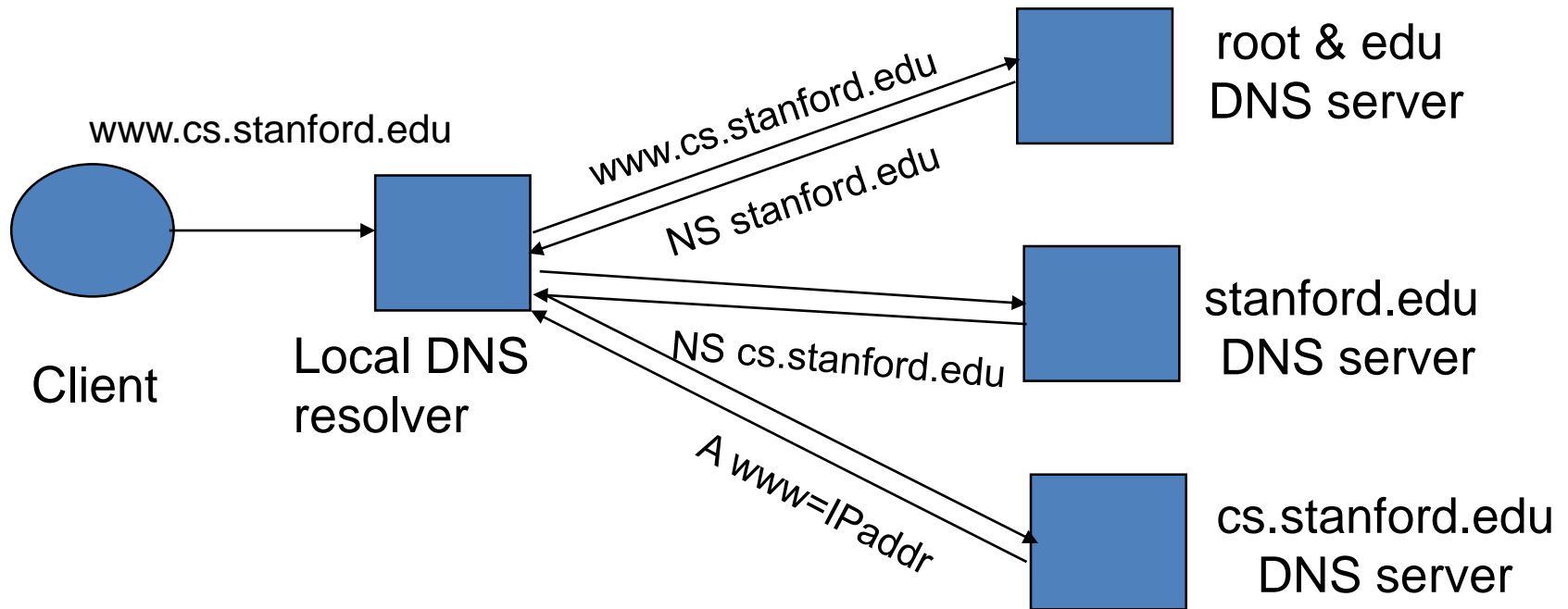


# DNS Root Name Servers

- Hierarchical service
  - Root name servers for top-level domains
  - Authoritative name servers for subdomains
  - Local name resolvers contact authoritative servers when they do not know a name



# DNS Lookup Example



DNS record types (partial list):

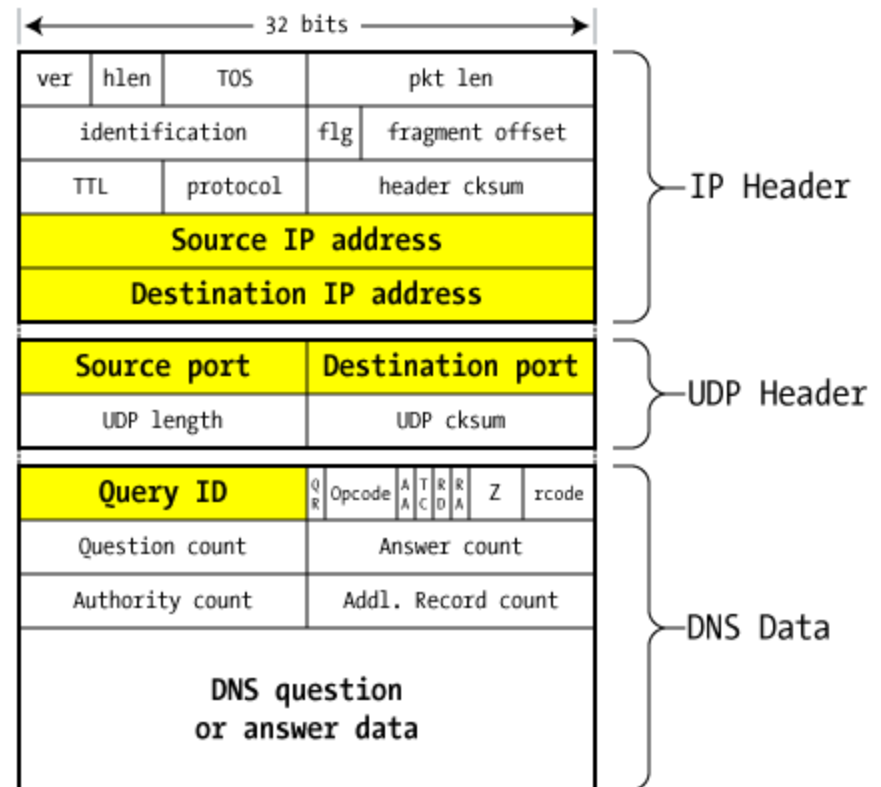
- NS: name server (points to other server)
- A: address record (contains IPv4 address)
- AAAA: address record (IPv6)
- MX: address in charge of handling email
- TXT: generic text (e.g. used to distribute site public keys (DKIM))

# Caching

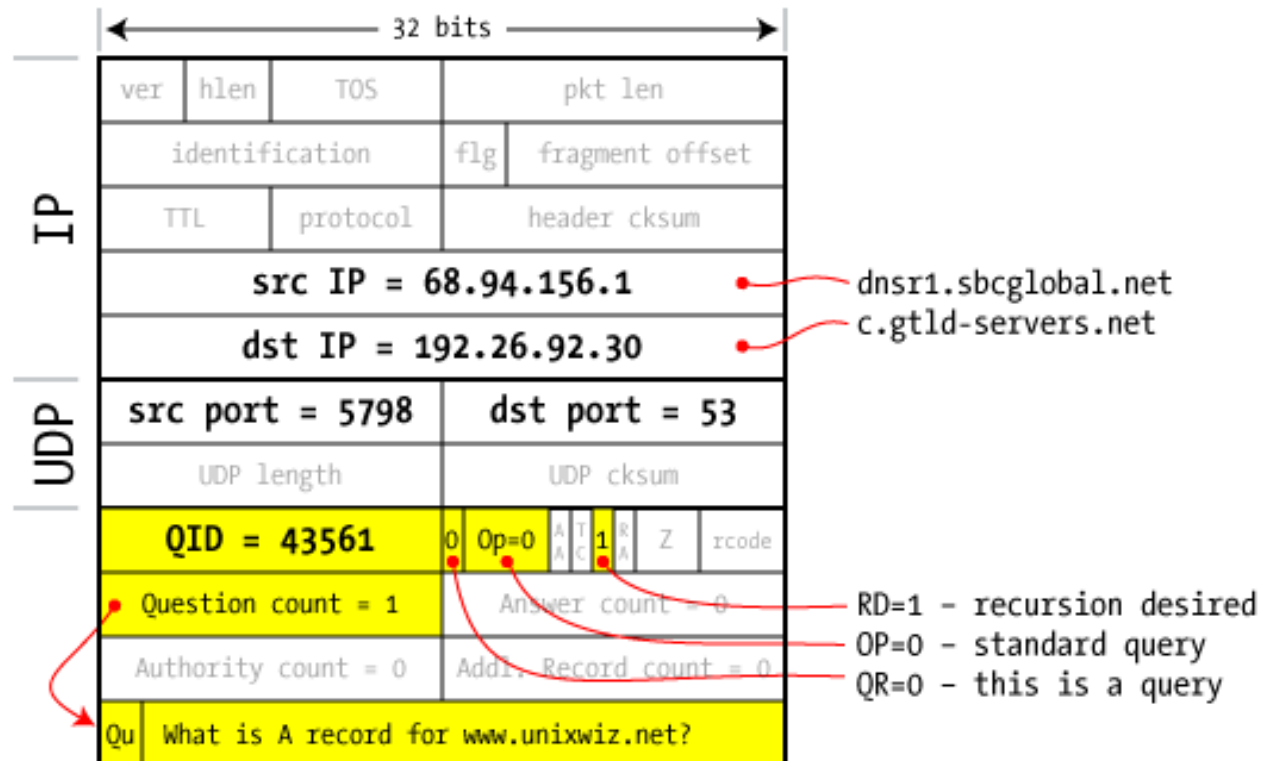
- DNS responses are cached
  - Quick response for repeated translations
  - Useful for finding servers as well as addresses
    - NS records for domains
- DNS negative queries are cached
  - Save time for nonexistent sites, e.g. misspelling
- Cached data periodically times out
  - Lifetime (TTL) of data controlled by owner of data
  - TTL passed with every record

# DNS Packet

- Query ID:
  - 16 bit random value
  - Links response to query



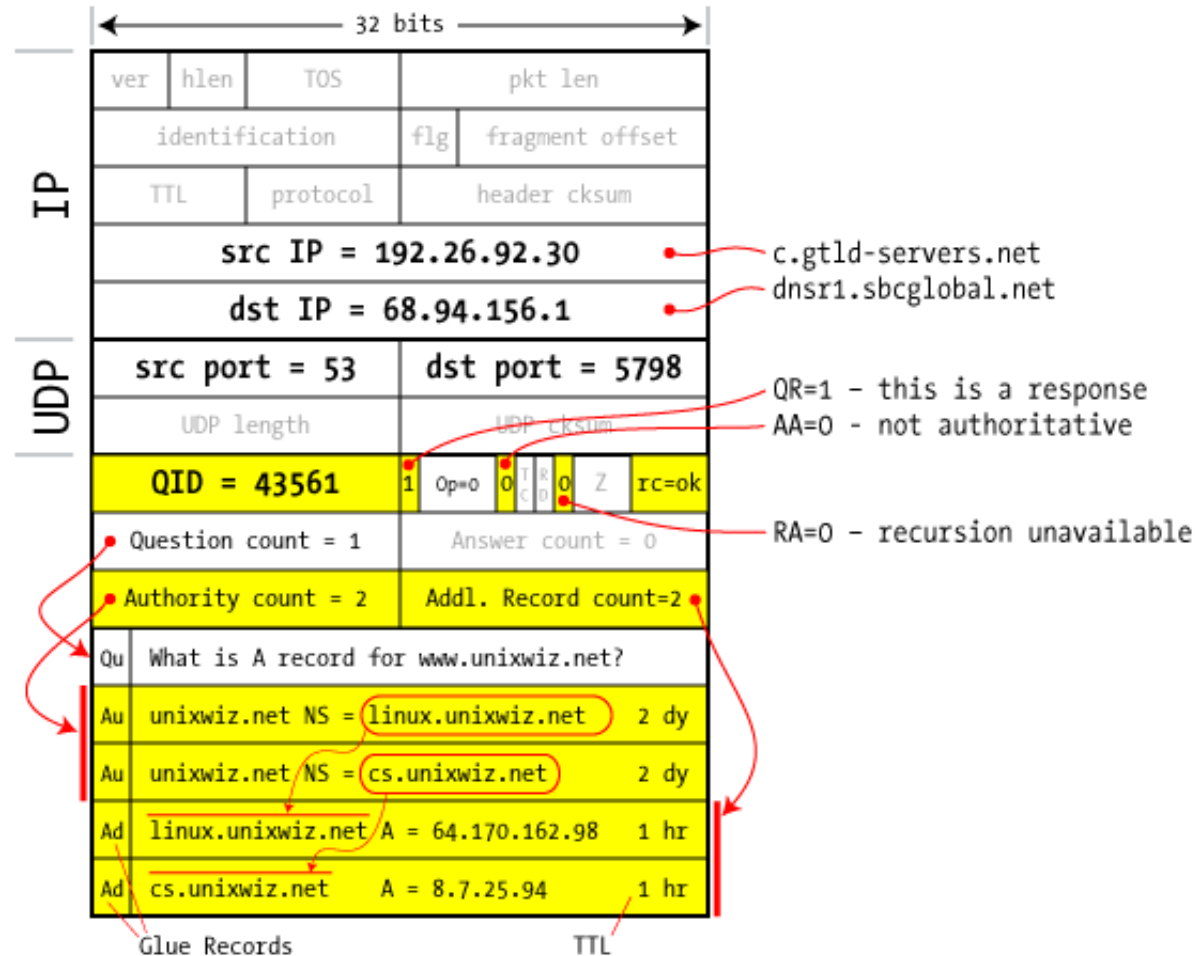
# Resolver to NS request



# Response to resolver

Response contains IP  
addr of next NS server  
(called “glue”)

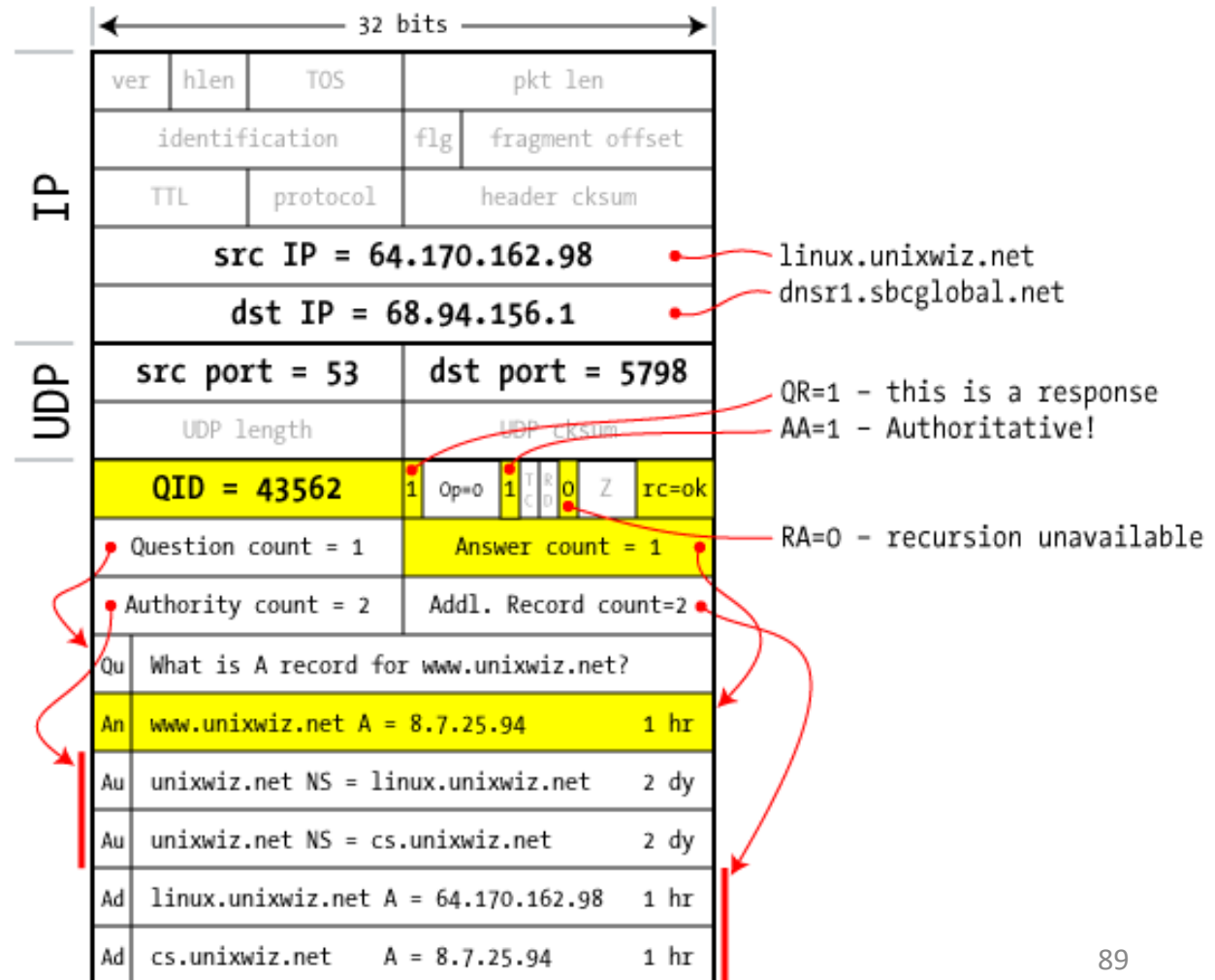
Response ignored if  
unrecognized QueryID



# Authoritative response to resolver

bailiwick checking:  
response is cached if  
it is within the same  
domain of query  
(i.e. **a.com** cannot  
set NS for **b.com**)

final answer →



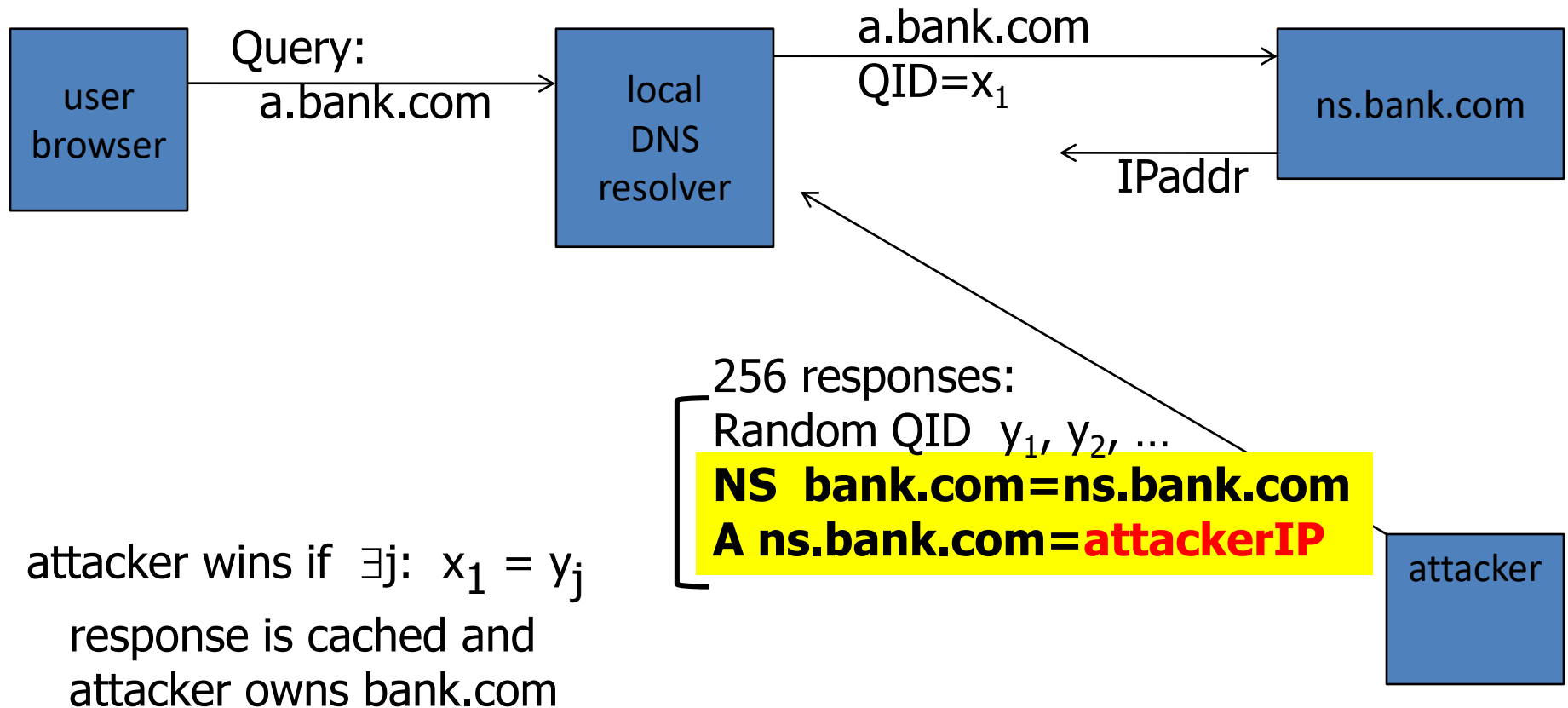


# Basic DNS Vulnerabilities

- Users/hosts trust the host-address mapping provided by DNS:
  - Used as basis for many security policies:  
Browser same origin policy, URL address bar
- Obvious problems
  - Interception of requests or compromise of DNS servers can result in incorrect or malicious responses
    - e.g.: malicious access point in a Cafe
  - Solution – authenticated requests/responses
    - Provided by DNSsec ... but few use DNSsec

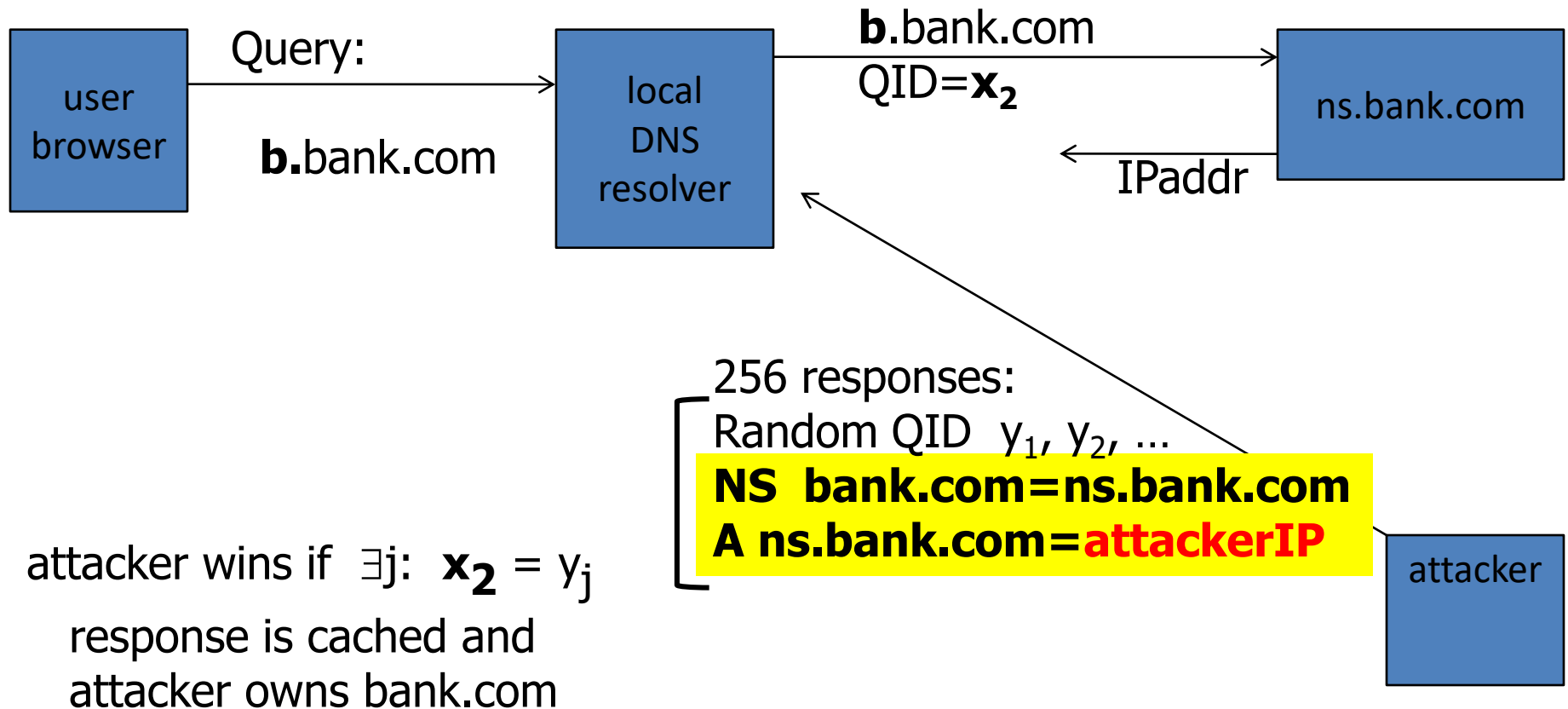
# DNS cache poisoning (a la Kaminsky' 08)

- Victim machine visits attacker's web site, downloads Javascript



# If at first you don't succeed ...

- Victim machine visits attacker's web site, downloads Javascript



success after  $\approx 256$  tries (few minutes) <sup>92</sup>

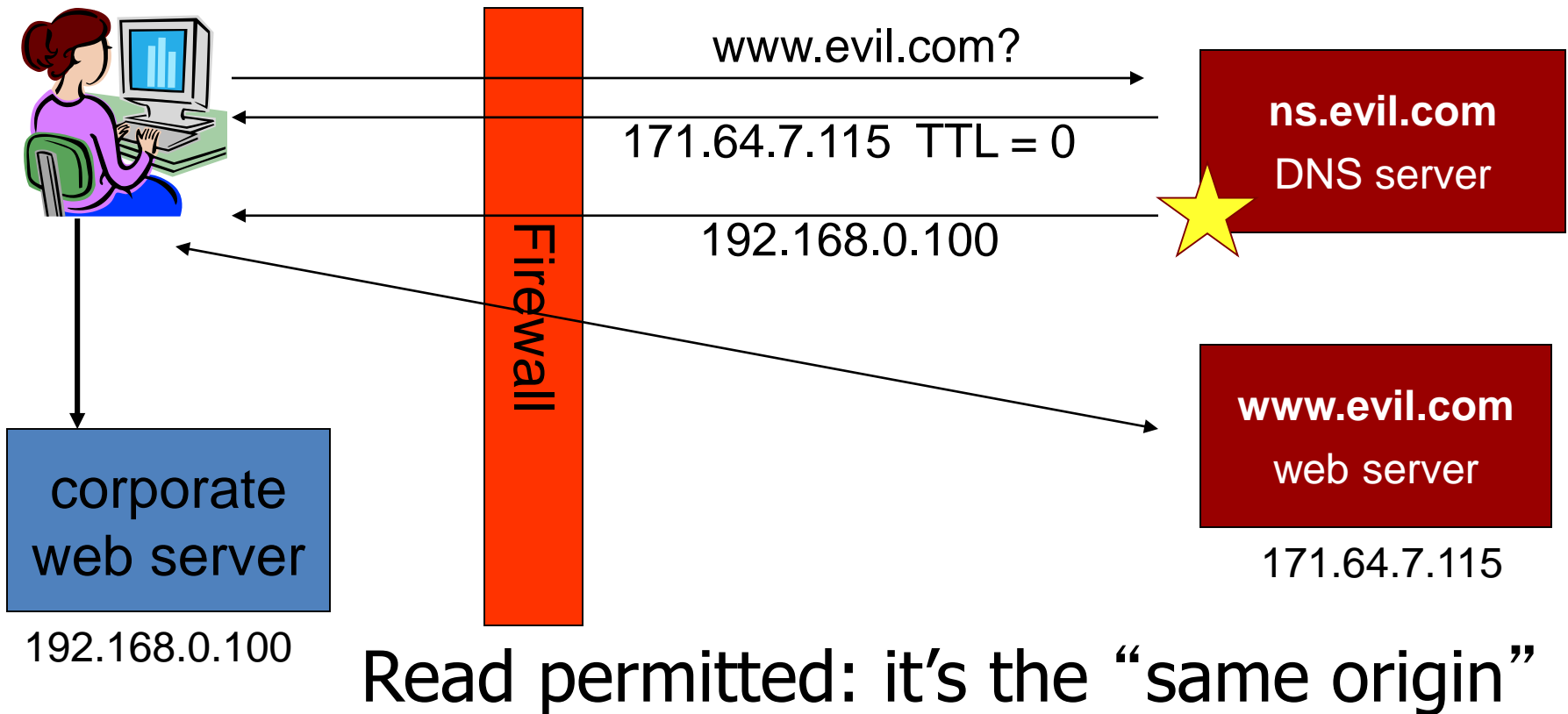
# Defenses

- Increase Query ID size.    How?
- Randomize src port, additional 11 bits
  - Now attack takes several hours
- Ask every DNS query twice:
  - Attacker has to guess QueryID correctly twice (32 bits)
  - ... but Apparently DNS system cannot handle the load

# DNS Rebinding Attack

`<iframe src="http://www.evil.com">`

DNS-SEC cannot  
stop this attack



# DNS Rebinding Defenses

- Browser mitigation: DNS Pinning
  - Refuse to switch to a new IP
  - Interacts poorly w/ proxies, VPN, dynamic DNS, ...
  - Not consistently implemented in any browser
- Server-side defenses
  - Check Host header for unrecognized domains
  - Authenticate users with something other than IP
- Firewall defenses
  - External names can't resolve to internal addresses
  - Protects browsers inside the organization

# Summary

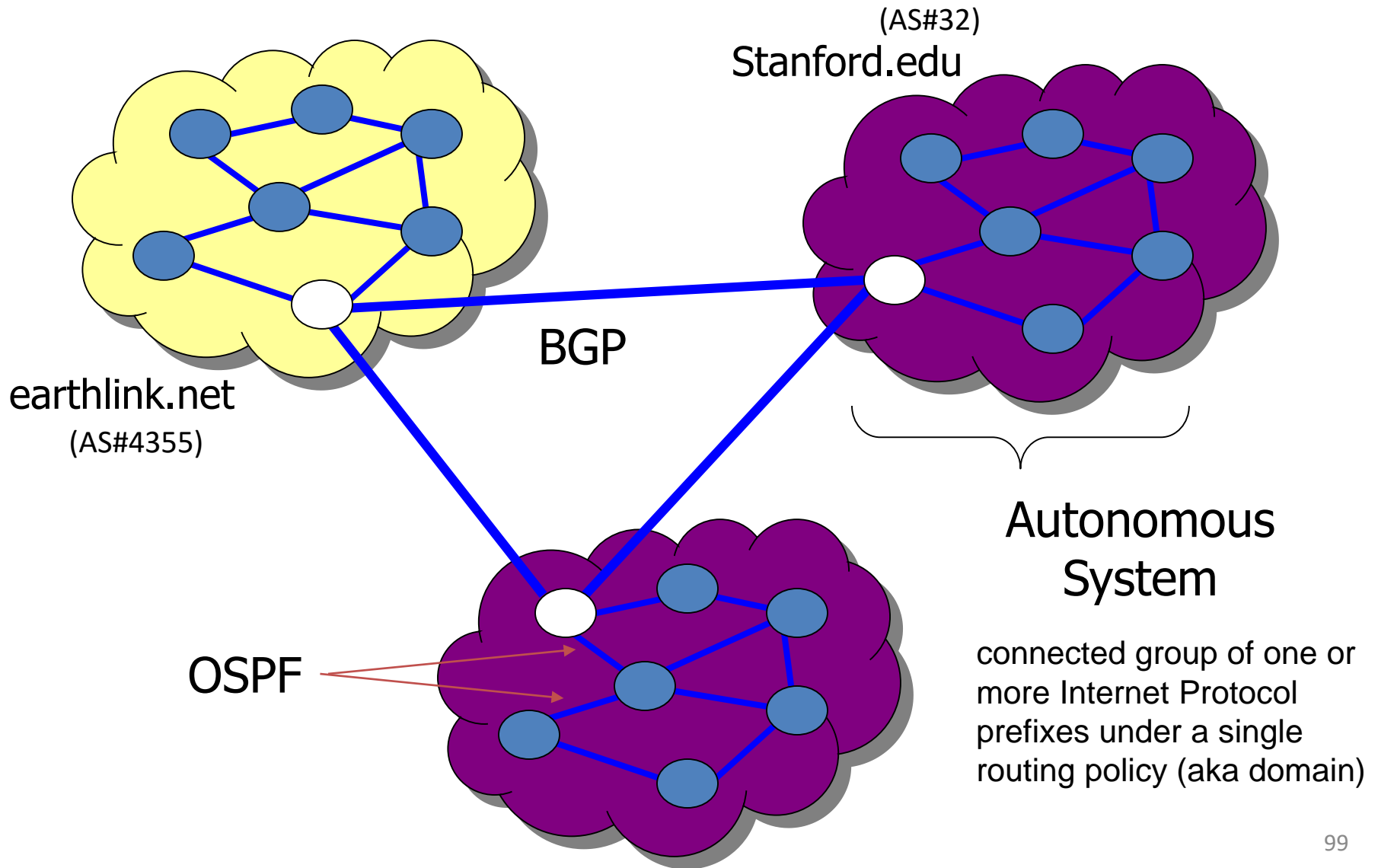
- Core protocols not designed for security
  - Eavesdropping, Packet injection, Route stealing, DNS poisoning
  - Patched over time to prevent basic attacks (e.g. random TCP SN)
- More secure variants exist (next lecture) :
  - IP → IPsec
  - DNS → DNSsec
  - BGP → SBGP

# Routing Security

ARP, OSPF, BGP



# Interdomain Routing

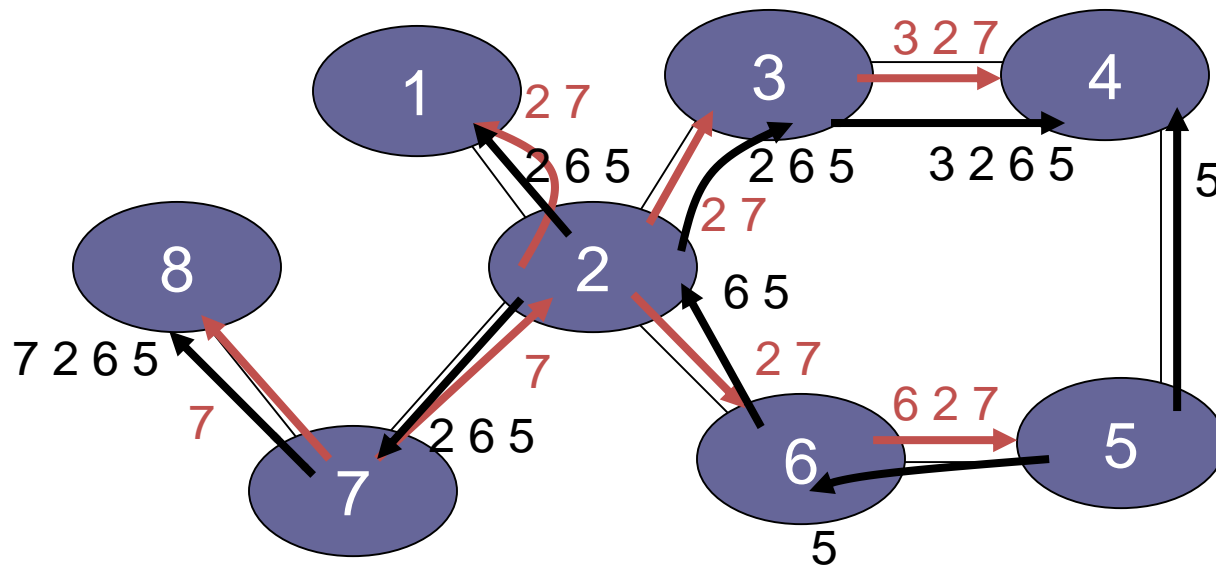


# Routing Protocols

- ARP (addr resolution protocol): IP addr → eth addr  
Security issues: (local network attacks)
  - Node A can confuse gateway into sending it traffic for Node B
  - By proxying traffic, node A can read/inject packets into B's session (e.g. WiFi networks)
- OSPF: used for routing within an AS
- BGP: routing between Autonomous Systems  
Security issues: unauthenticated route updates
  - Anyone can cause entire Internet to send traffic for a victim IP to attacker's address
    - Example: Youtube-Pakistan mishap (see DDoS lecture)
  - Anyone can hijack route to victim (next slides)

# BGP example

[D. Wetherall]



# Security Issues

BGP path attestations are un-authenticated

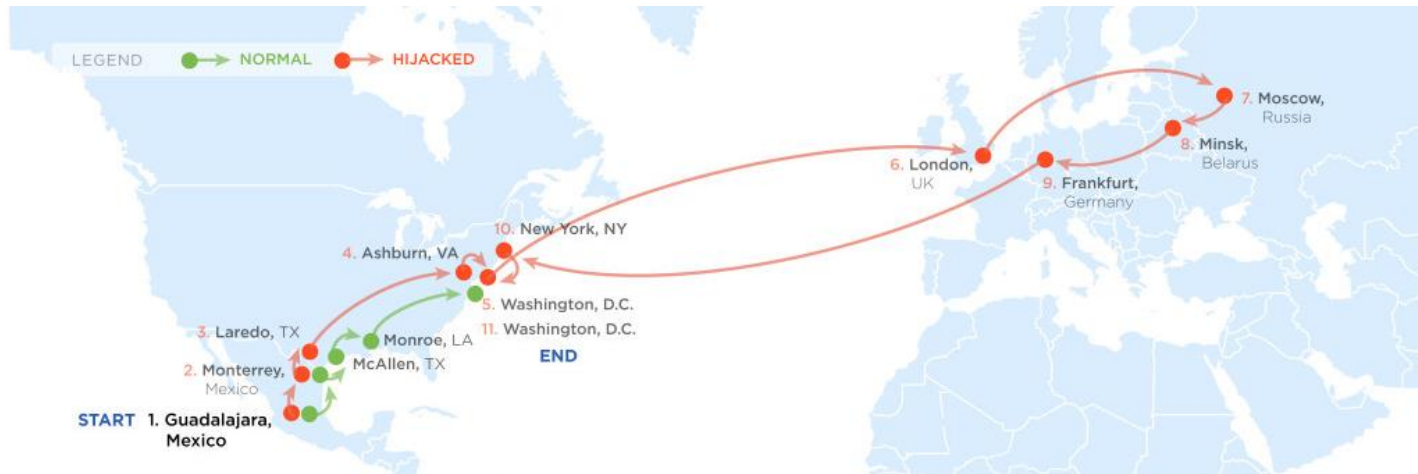
- Anyone can inject advertisements for arbitrary routes
- Advertisement will propagate everywhere
- Used for DoS, spam, and eavesdropping (details in DDoS lecture)
- Often a result of human error

Solutions:

- RPKI: AS obtains a certificate (ROA) from RIR and attaches ROA to path advertisements.  
Advertisements without a valid ROA are ignored.  
Defends against a malicious AS (but not a network attacker)
- SBGP: sign every hop of a path advertisement

# Example path hijack (source: Renesys 2013)

Feb 2013: Guadalajara → Washington DC via Belarus



route  
in effect  
for several  
hours

Normally: Alestra (Mexico) → PCCW (Texas) → Qwest (DC)

Reverse route (DC → Guadalajara) is unaffected:

- Person browsing the Web in DC cannot tell by *traceroute* that HTTP responses are routed through Moscow

# OSPF: routing inside an AS

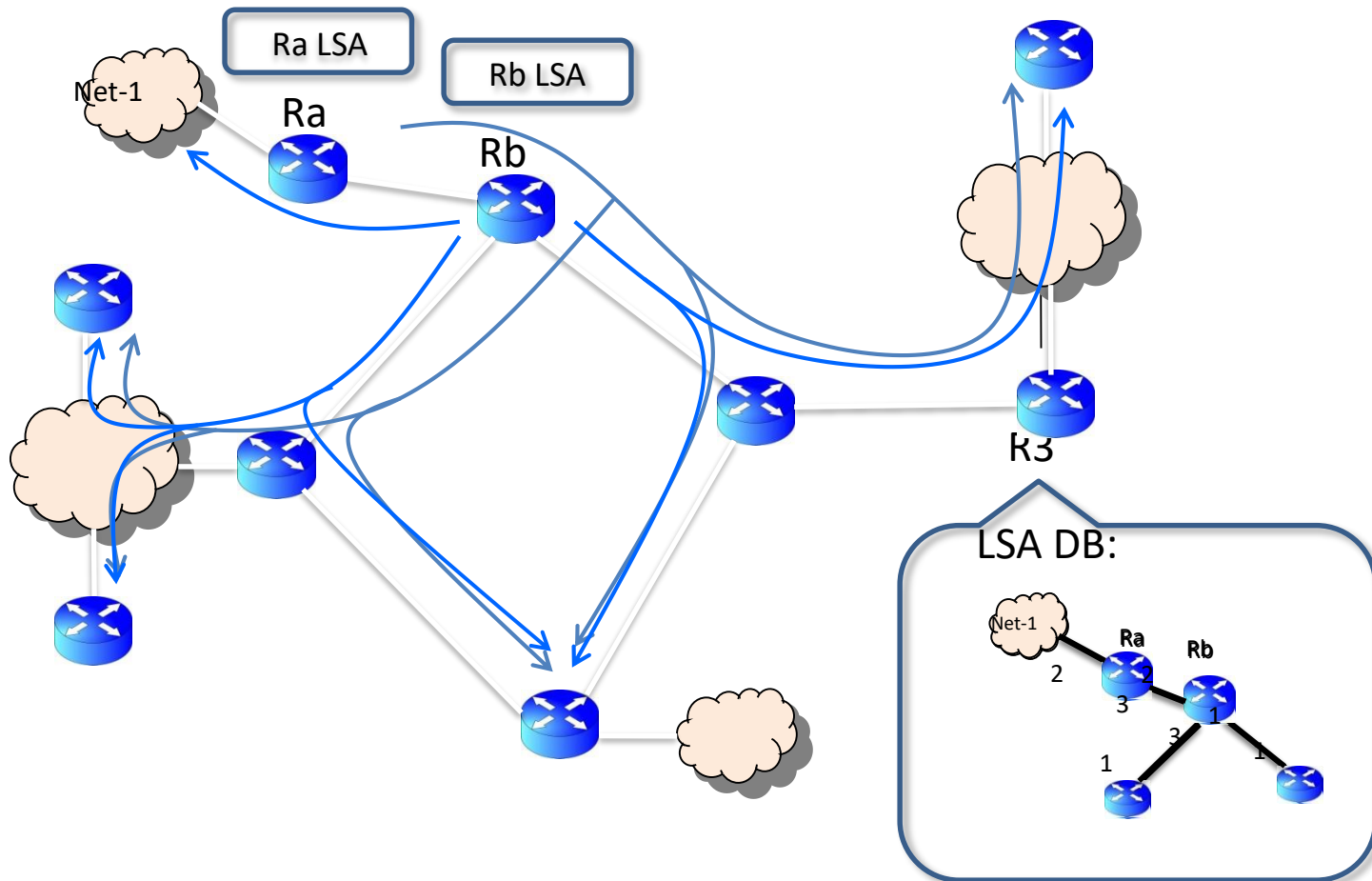
Link State Advertisements (LSA):

- Flooded throughout AS so that all routers in the AS have a complete view of the AS topology
- Transmission: IP datagrams, protocol = 89

Neighbor discovery:

- Routers dynamically discover direct neighbors on attached links --- sets up an “adjacency”
- Once setup, they exchange their LSA databases

# Example: LSA from Ra and Rb



# Security features

- OSPF message integrity (unlike BGP)
  - Every link can have its own shared secret
  - Unfortunately, OSPF uses an insecure MAC:
$$\text{MAC}(k,m) = \text{MD5}(\text{data} \parallel \text{key} \parallel \text{pad} \parallel \text{len})$$
- Every LSA is flooded throughout the AS
  - If a single malicious router, valid LSAs may still reach dest.
- The “fight back” mechanism
  - If a router receives its own LSA with a newer timestamp than the latest it sent, it immediately floods a new LSA
- Links must be advertised by both ends