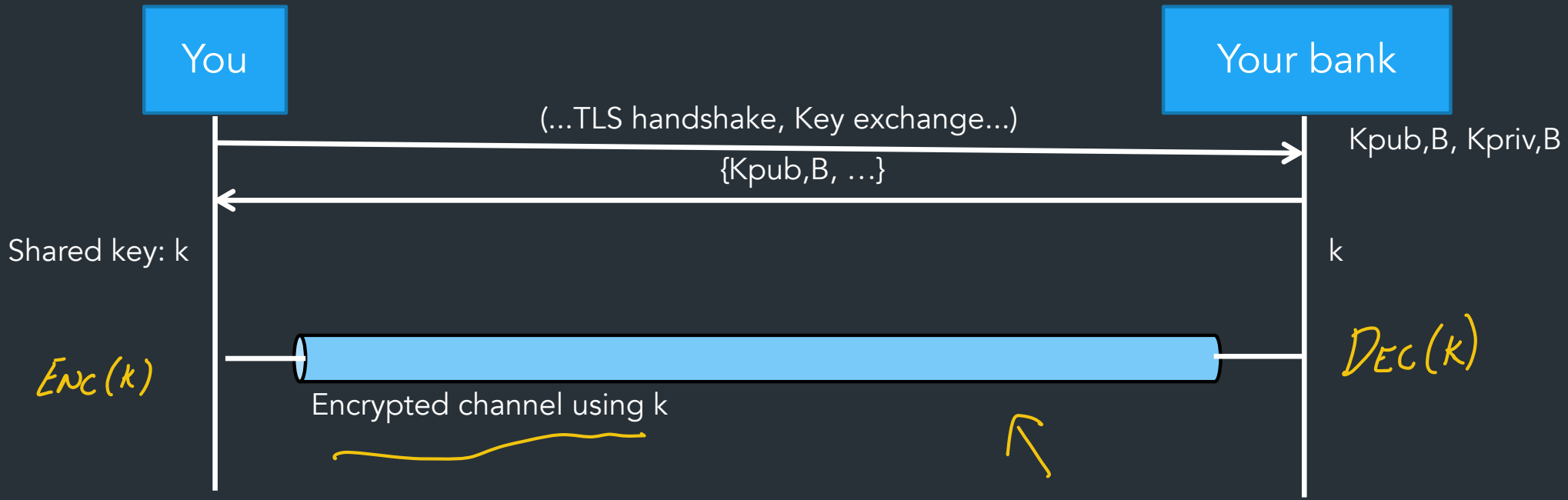
CSCI-1680
More on TLS
How to (try) to be anonymous

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, Scott Shenker and John Jannotti

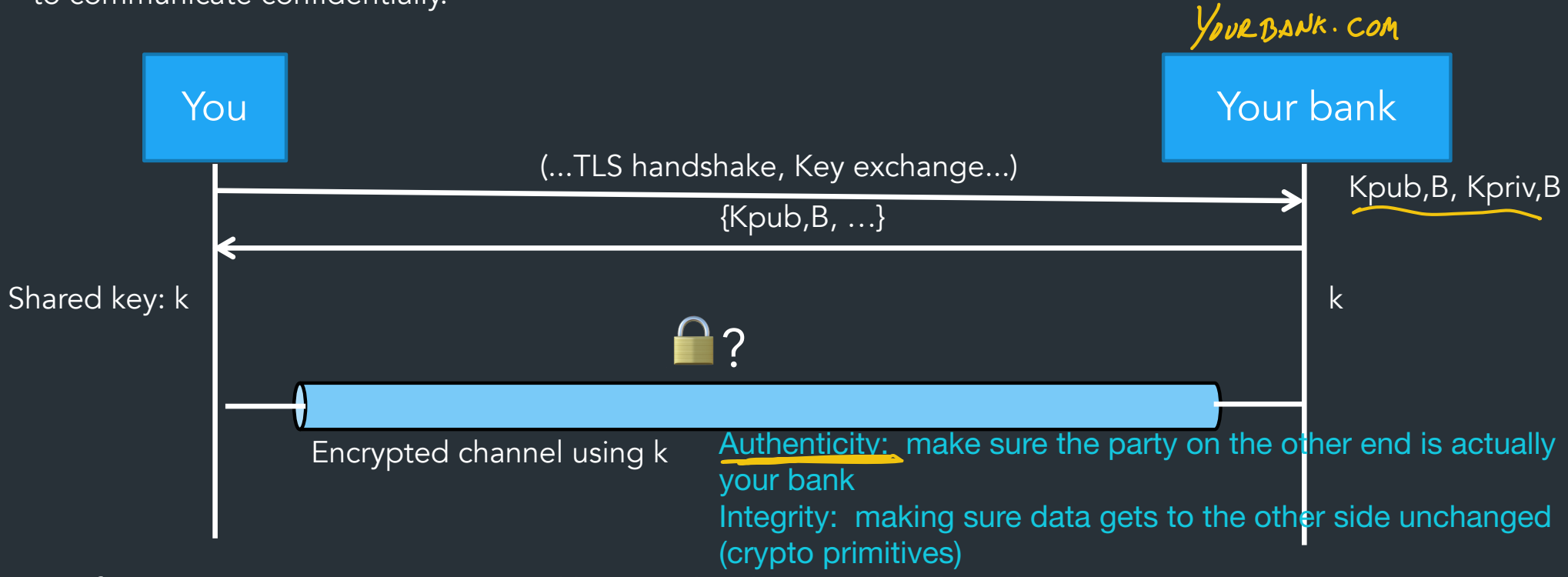
Warmup

When establishing a TLS connection, can (easily) set up a shared key for both parties to communicate confidentially.



Warmup

When establishing a TLS connection, can (easily) set up a shared key for both parties to communicate confidentially.



But if you want to connect to a site like your bank securely, what else is missing?
What do we need besides confidentiality?

Problem: How can we trust K_pub is Your Bank's public key?

Problem: distributing trust

How can we trust Kpub is Your Bank's public key?

Problem: Trust distribution

- Hard to verify real-world identities
- Hard to scale to the whole Internet

Different protocols have different mechanisms

=> TLS (and others): Public Key Infrastructure (PKI) with certificates

PKI: The main idea

"TRUSTED AUTHORITY"

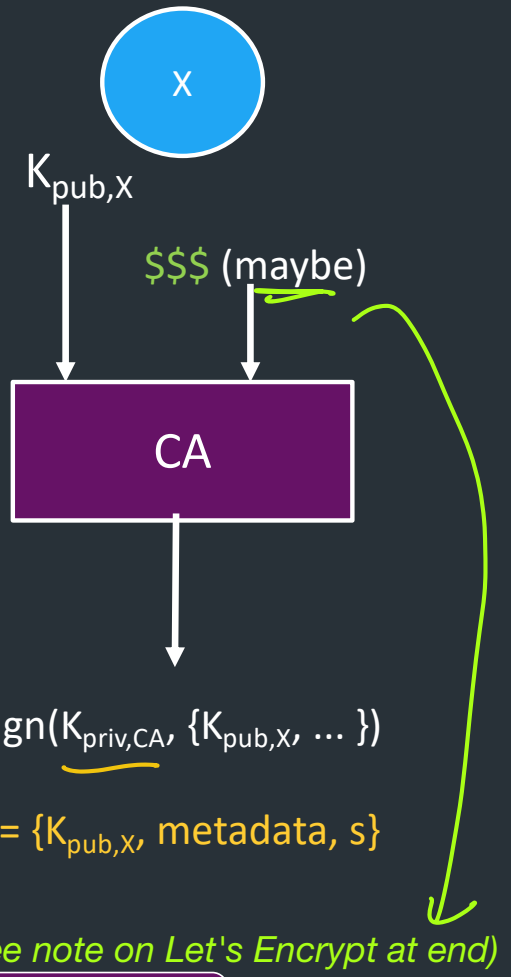
Public keys managed by Certificate Authorities (CAs)

- Everyone knows public key for some root CAs
  - Pre-installed into browser/OS

EVERYONE HAS $K_{pub,CA}$

- If X wants a public key, request from CA
  - CA validates X's identity => if OK signs X's public key
  - Generates certificate
- Client can verify $K_{pub,X}$ from CA's signature:

  Verify($K_{pub,CA}$ Cert) => True/False



=> Delegates trust for individual entity to a more trusted authority

What's in a certificate?

- Public key of entity (eg. yourbank.com)
- Common name: DNS name of server (yourbank.com)
- Contact info for organization
- VALIDITY DATES.
- ...

K_{PUB}

↳ OR A LIST OF NAMES

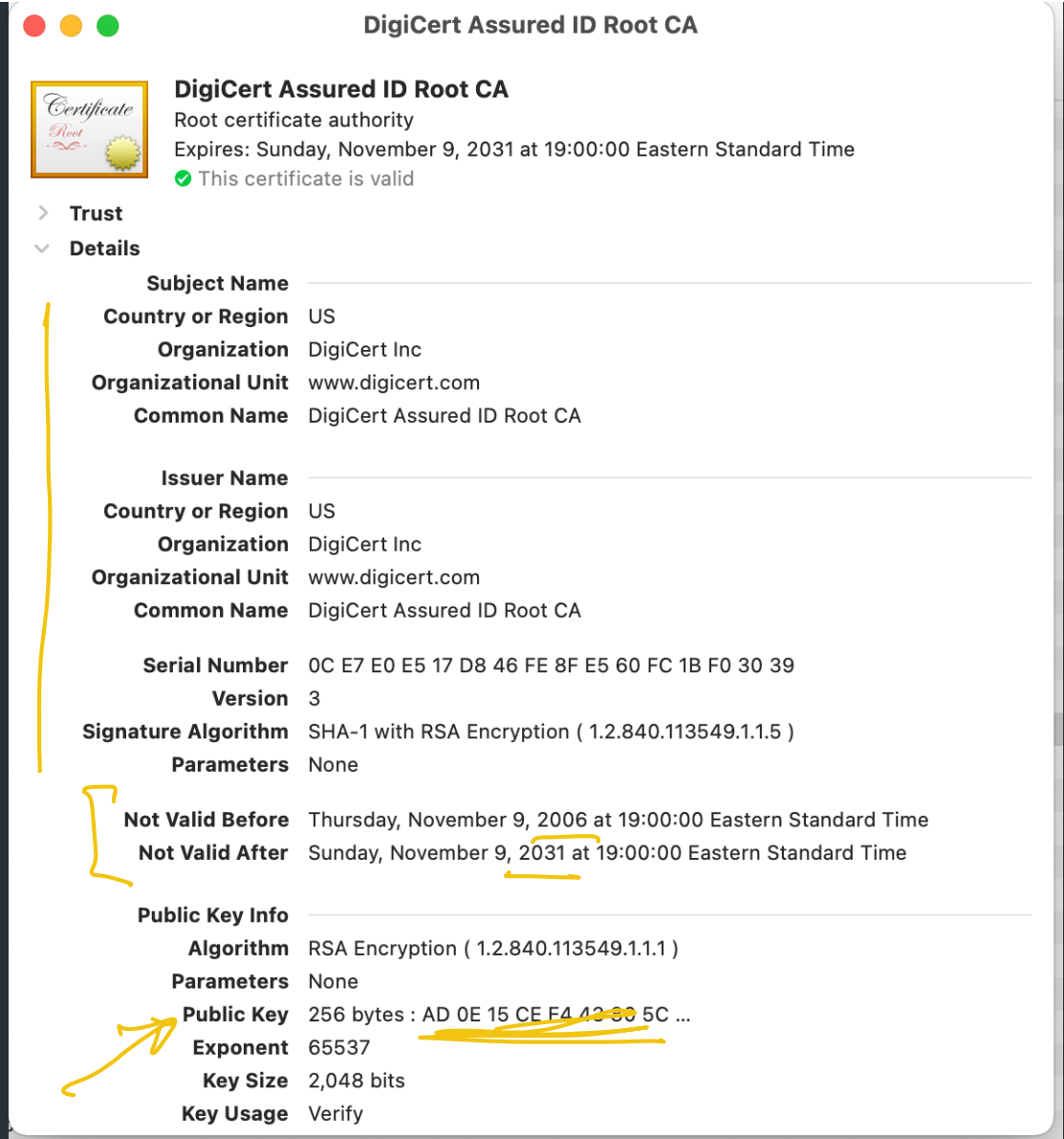
ALL SIGNED BY CA

CERT IS INVALID IF ANY METADATA ALTERED!

What's in a certificate?

- Public key of entity (eg. yourbank.com)
- Common name: DNS name of server (yourbank.com)
- Contact info for organization
- Validity dates (start date, expire date)
- URL of *revocation center* to check if key has been revoked

All of this is part of the data signed by the CA
=> Critical to check all parts during TLS startup!



Given Cert = {Kpub,bank, s, ..}

Browser will: Verify(s, Kpub,CA)

↳ {T, F}

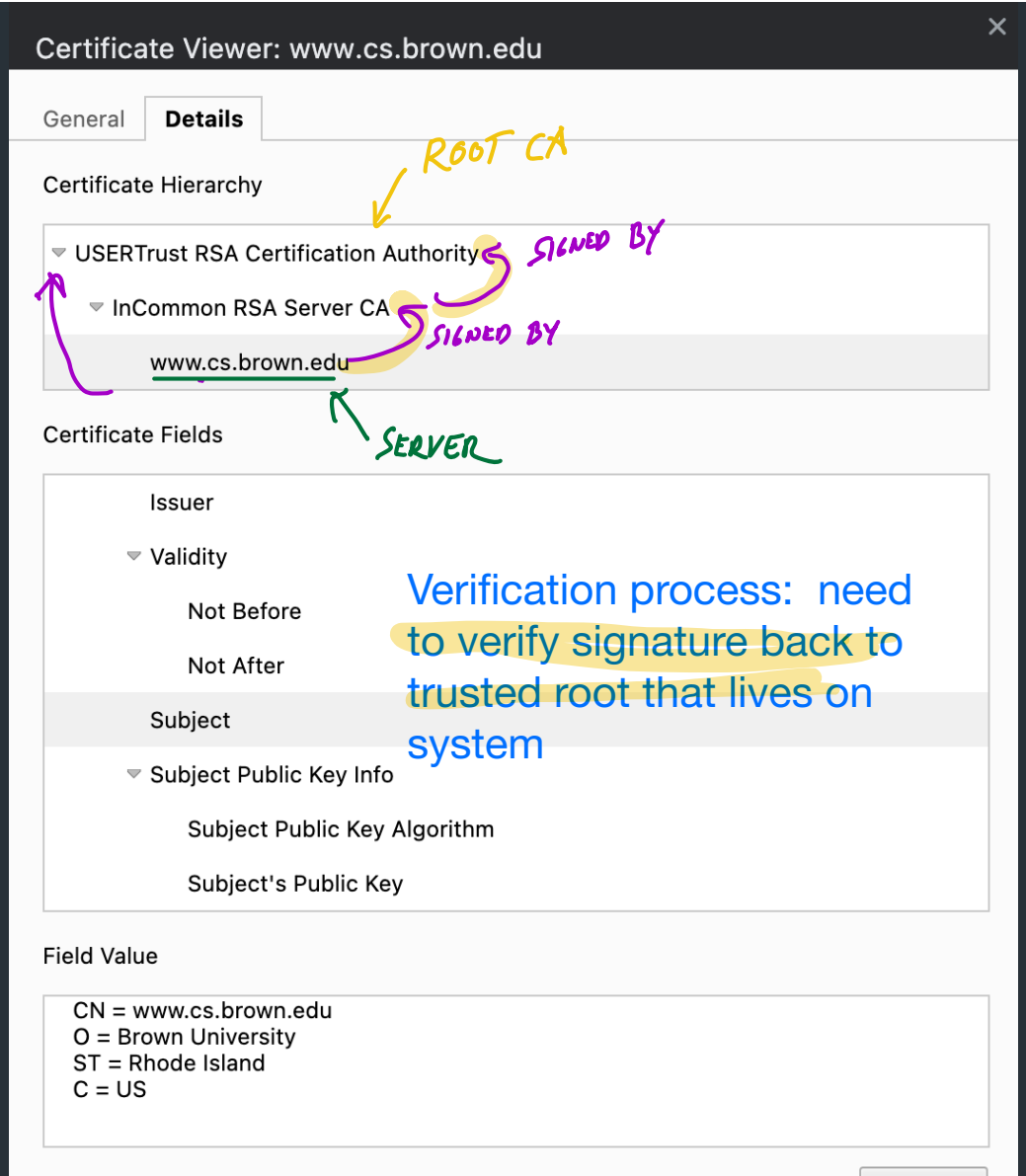
Keychain Access

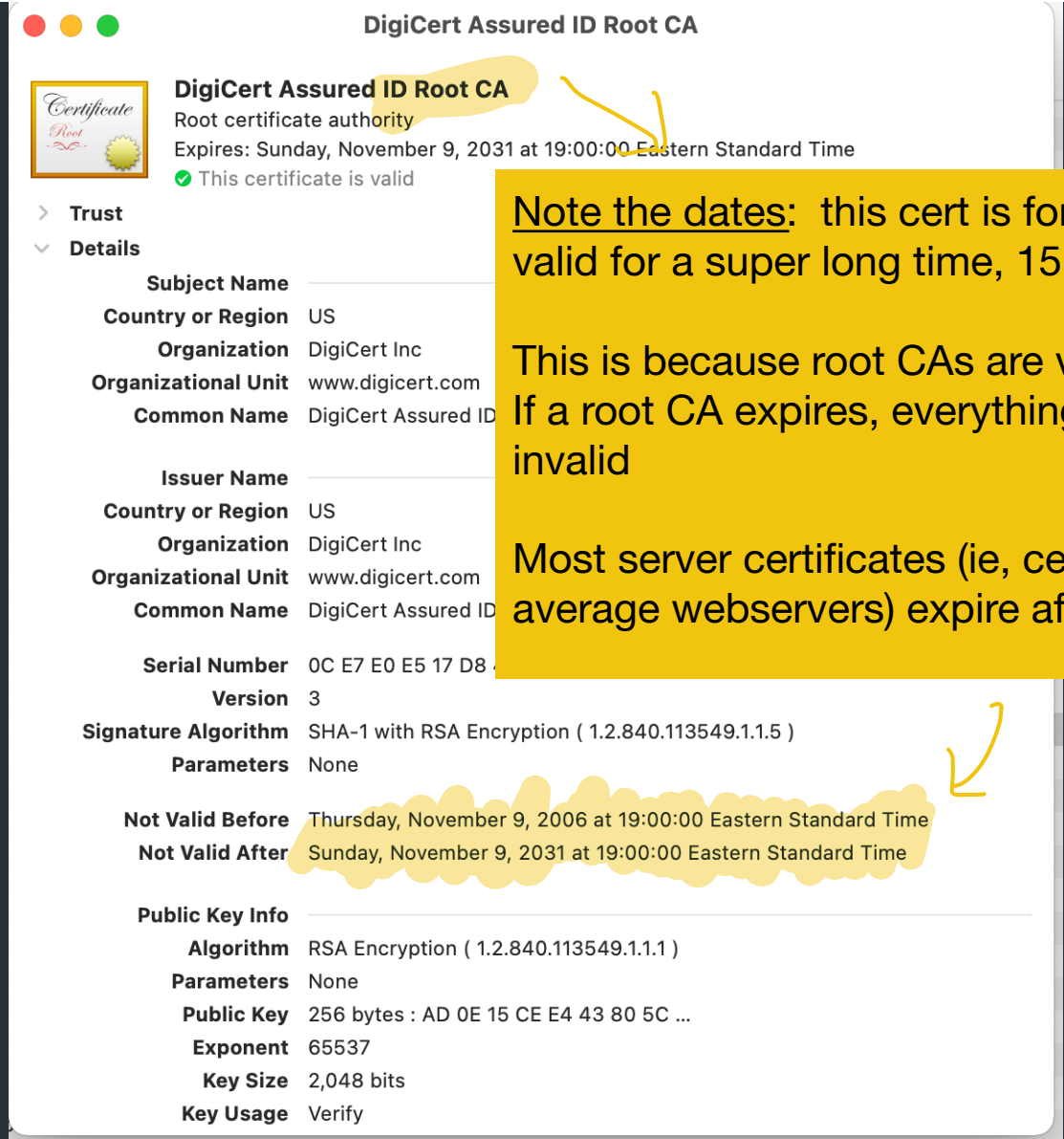
Search

All Items | Passwords | Secure Notes | My Certificates | Keys | Certificates

Amazon Root CA 1
Root certificate authority
Expires: Saturday, January 16, 2038 at 19:00:00 Eastern Standard Time
This certificate is valid

Name	Kind	Date Modified	Expires	Keychain
AAA Certificate Services	certificate	--	Dec 31, 2028 at 18:59:59	System Roots
AC RAIZ FNMT-RCM	certificate	--	Dec 31, 2029 at 19:00:00	System Roots
Actalis Authentication Root CA	certificate	--	Sep 22, 2030 at 07:22:02	System Roots
AffirmTrust Commercial	certificate	--	Dec 31, 2030 at 09:06:06	System Roots
AffirmTrust Networking	certificate	--	Dec 31, 2030 at 09:08:24	System Roots
AffirmTrust Premium	certificate	--	Dec 31, 2040 at 09:10:36	System Roots
AffirmTrust Premium ECC	certificate	--	Dec 31, 2040 at 09:20:24	System Roots
Amazon Root CA 1	certificate	--	Jan 16, 2038 at 19:00:00	System Roots
Amazon Root CA 2	certificate	--	May 25, 2040 at 20:00:00	System Roots
Amazon Root CA 3	certificate	--	May 25, 2040 at 20:00:00	System Roots
Amazon Root CA 4	certificate	--	May 25, 2040 at 20:00:00	System Roots
ANF Global Root CA	certificate	--	Jun 5, 2033 at 13:45:38	System Roots
Apple Root CA	certificate	--	Feb 9, 2035 at 16:40:36	System Roots
Apple Root CA - G2	certificate	--	Apr 30, 2039 at 14:10:09	System Roots
Apple Root CA - G3	certificate	--	Apr 30, 2039 at 14:19:06	System Roots
Apple Root Certificate Authority	certificate	--	Feb 9, 2025 at 19:18:14	System Roots
Atos TrustedRoot 2011	certificate	--	Dec 31, 2030 at 18:59:59	System Roots
Autoridad de Certificacion Firmaprofesional CIF A62634068	certificate	--	Dec 31, 2030 at 03:38:15	System Roots
Autoridad de Certificacion Raiz del Estado Venezolano	certificate	--	Dec 17, 2030 at 18:59:59	System Roots
Baltimore CyberTrust Root	certificate	--	May 12, 2025 at 19:59:00	System Roots
Buypass Class 2 Root CA	certificate	--	Oct 26, 2040 at 04:38:03	System Roots
Buypass Class 3 Root CA	certificate	--	Oct 26, 2040 at 04:28:58	System Roots
CA Disig Root R1	certificate	--	Jul 19, 2042 at 05:06:56	System Roots
CA Disig Root R2	certificate	--	Jul 19, 2042 at 05:15:30	System Roots
Certigna	certificate	--	Jun 29, 2027 at 11:13:05	System Roots
Certinomis - Autorité Racine	certificate	--	Sep 17, 2028 at 04:28:59	System Roots
Certinomis - Root CA	certificate	--	Oct 21, 2033 at 05:17:18	System Roots
Certplus Root CA G1	certificate	--	Jan 14, 2038 at 19:00:00	System Roots
Certplus Root CA G2	certificate	--	Jan 14, 2038 at 19:00:00	System Roots
certSIGN ROOT CA	certificate	--	Jul 4, 2031 at 13:20:04	System Roots
Certum CA	certificate	--	Jun 11, 2027 at 06:46:39	System Roots
Certum Trusted Network CA	certificate	--	Dec 31, 2029 at 07:07:37	System Roots


Certificate Viewer: www.cs.brown.edu
General
Details
Certificate Hierarchy
ROOT CA
USERTrust RSA Certification Authority
SIGNED BY
InCommon RSA Server CA
SIGNED BY
www.cs.brown.edu
SERVER
Certificate Fields
Issuer
Validity
Not Before
Not After
Subject
Subject Public Key Info
Subject Public Key Algorithm
Subject's Public Key
Verification process: need to verify signature back to trusted root that lives on system
Field Value
CN = www.cs.brown.edu
O = Brown University
ST = Rhode Island
C = US



Note the dates: this cert is for a root CA, so it's valid for a super long time, 15 years!

This is because root CAs are very hard to change. If a root CA expires, everything signed by it is invalid

Most server certificates (ie, certs installed on average webservers) expire after 1 year, or less

PKI hierarchy

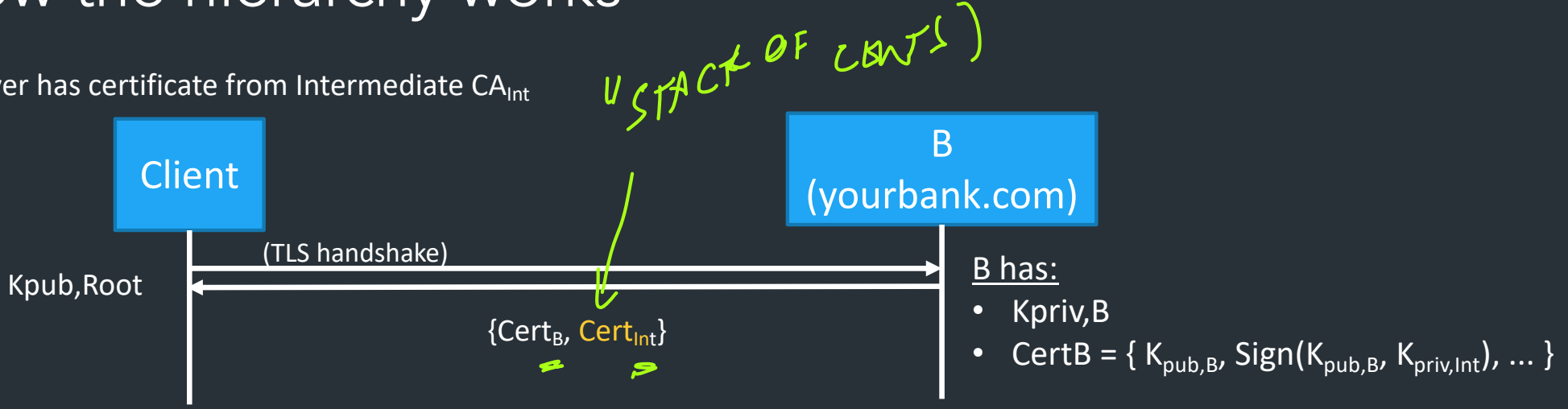
In reality, PKI creates a hierarchy of trust:

- Root CAs: k_{pub} stored in virtually every browser, OS
  - Private keys protected by most stringent security measures (software, hardware, physical)
- Intermediate CAs: k_{pub} signed by root CA
  - Sign certificates for general use (ie, regular websites)
  - Doesn't require same protections as root
- General-use certificates: for a specific webserver

What happens if a root is compromised?

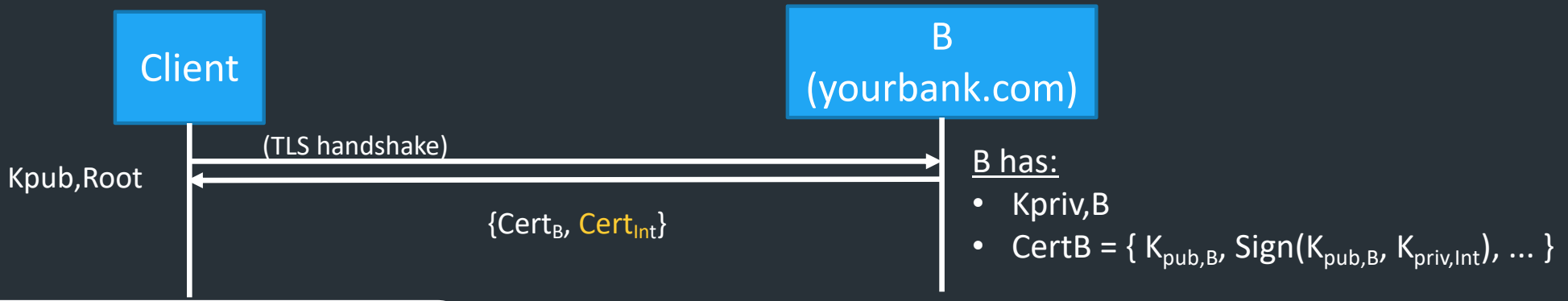
How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}



How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}

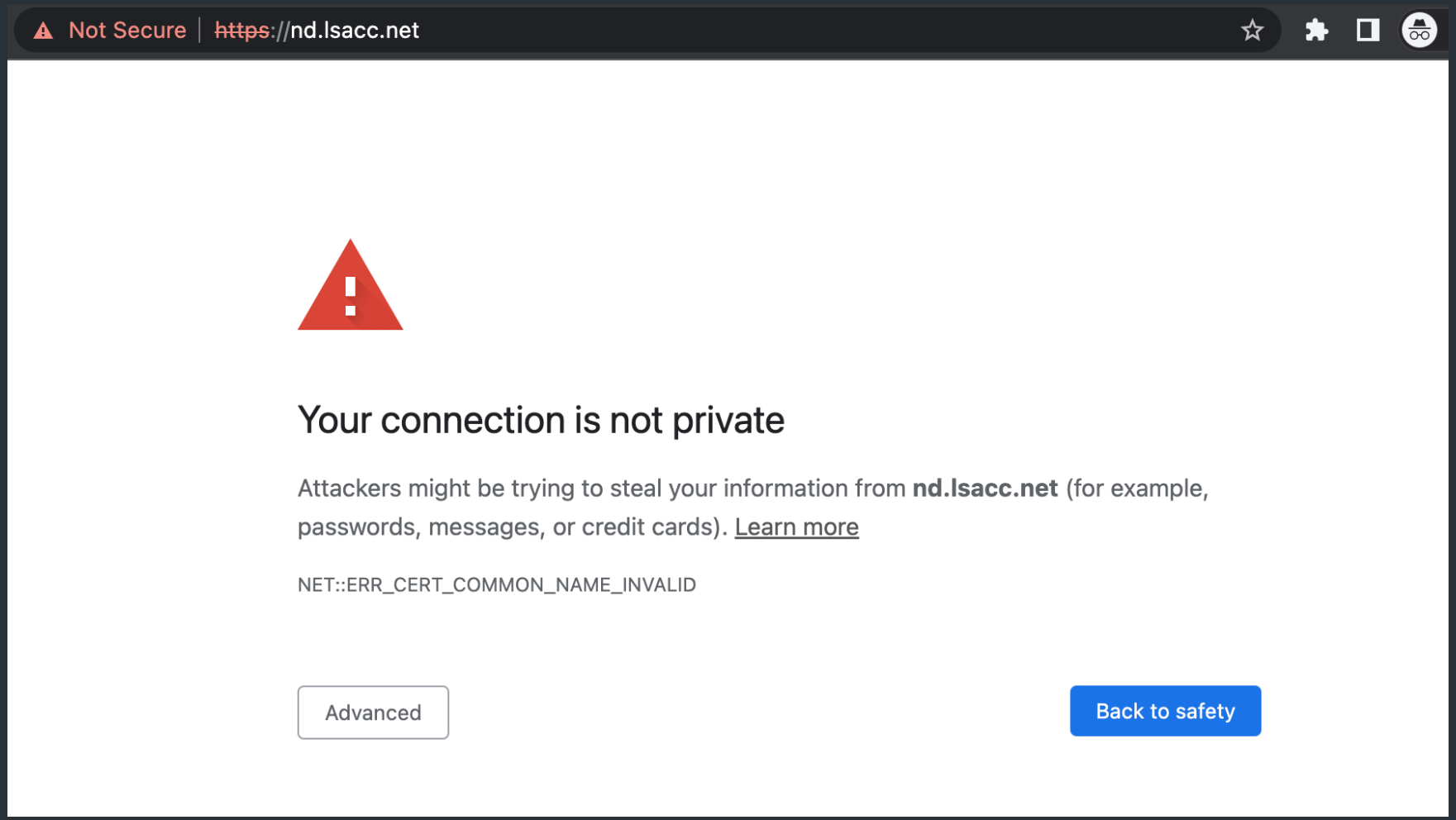


Client's workflow:

- Checks metadata ✅
- Verify($Cert_B$, $K_{pub,Int}$) ✅
- Verify($Cert_{Int}$, $K_{pub,Root}$) ✅

=> To verify integrity, need to verify certificates back to (trusted) root certificate

=> OK if verification passes and metadata correct: 🔒


Not Secure | https://nd.lsacc.net
Your connection is not private
Attackers might be trying to steal your information from nd.lsacc.net (for example, passwords, messages, or credit cards). Learn more
NET::ERR_CERT_COMMON_NAME_INVALID
Advanced
Back to safety

Most common TLS errors you might see

- Common name (eg. yourbank.com) invalid
- Certificate expired => 3mos – 1yr for server certs
- Bad chain of trust (can't verify back to trusted root)

=> Usually a sign of something sketchy, or something wrong with the webserver

When is it okay to click "proceed"? What happens if you do?

=> *Might occur if webserver configured properly, or if you're setting up a system, but not okay for your bank (or Brown ...)*

Most common TLS errors you might see

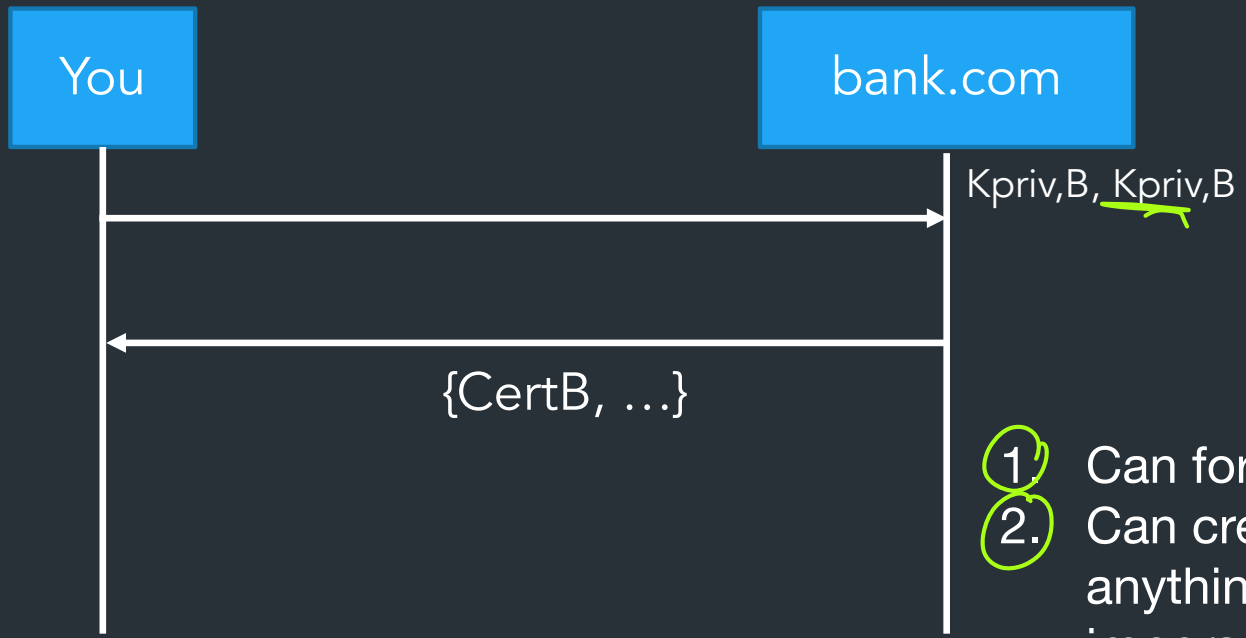
- Common name (eg. yourbank.com) invalid
- Certificate expired
- Bad chain of trust (can't verify to trusted root cert)
- "Certificate is self-signed"???

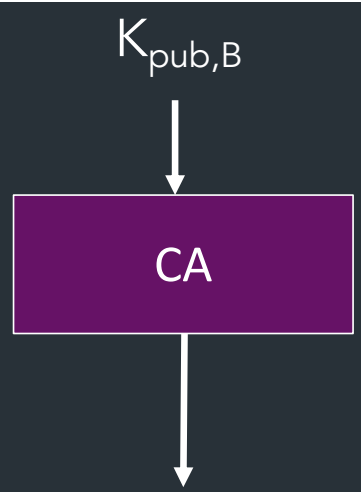
Kpub,X
CertX = {Kpub,X, Sign(Kpub,X, Kpriv,X)}

Self-signed: certificate that signs itself
=> Common for demo services
=> Root CAs are self-signed (that's okay because we trust them)

Warmup

1. What happens if attacker obtains Kpriv,B?
2. What about Kpriv,CA?





$s = Sign(K_{priv,CA}, \{K_{pub,B}, \ldots\})$

$Cert_B = \{K_{pub,B}, metadata, s\}$

1. Can forge messages, impersonate B
2. Can create arbitrary signatures for anything you want => can impersonate ANY website

Warmup

With TLS we get this:

Are we good? Have we solved web security?

Header info for src/dst: who you are, who you're talking to
- Can see protocol (by port numbers)
- Not all applications support TLS/ encryption, so maybe want something more hollistic
=> Would like to do security at a lower llayer than the transport layer

So, are we good?

If we use TLS, is it enough?

Overall, depends on your threat model…

- Server still knows who you are, even if connection is encrypted

- Even encrypted traffic leaks information!

Why?

- Avoiding censorship
- Avoiding surveillance (by person, or an organization)
- Anonymous reporting (journalists, whistleblowers)

Room 641A: alleged wiretapping room in a datacenter for an Internet backbone… https://en.wikipedia.org/wiki/Room_641A

How can we deal with this?

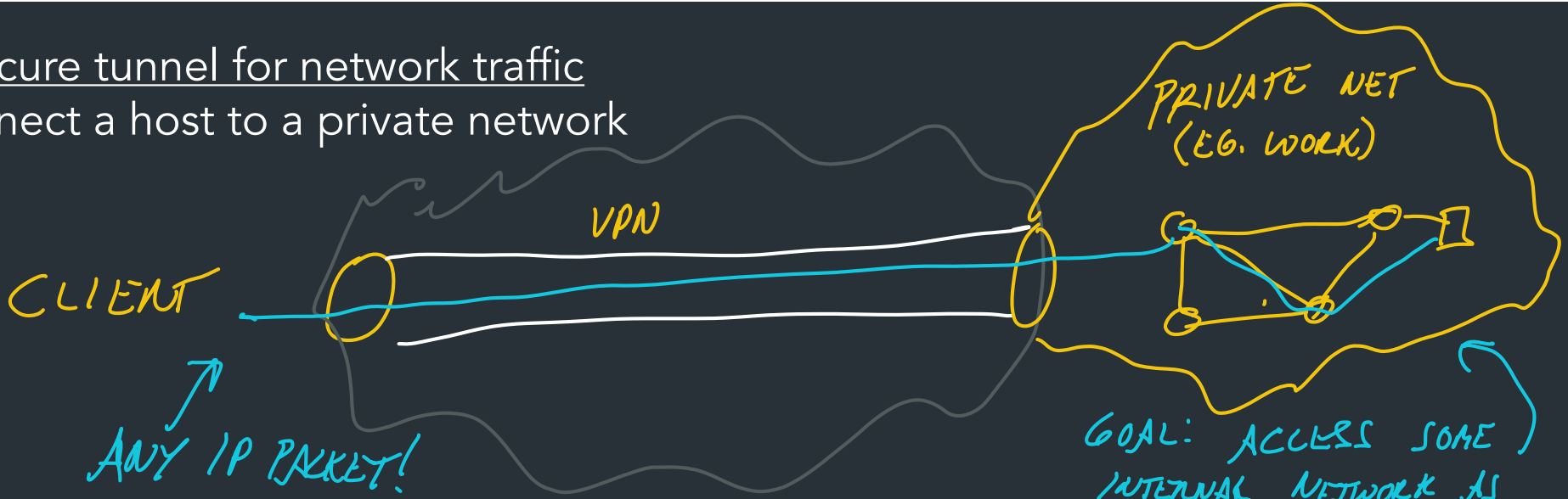
Mechanisms to provide more security at the network layer

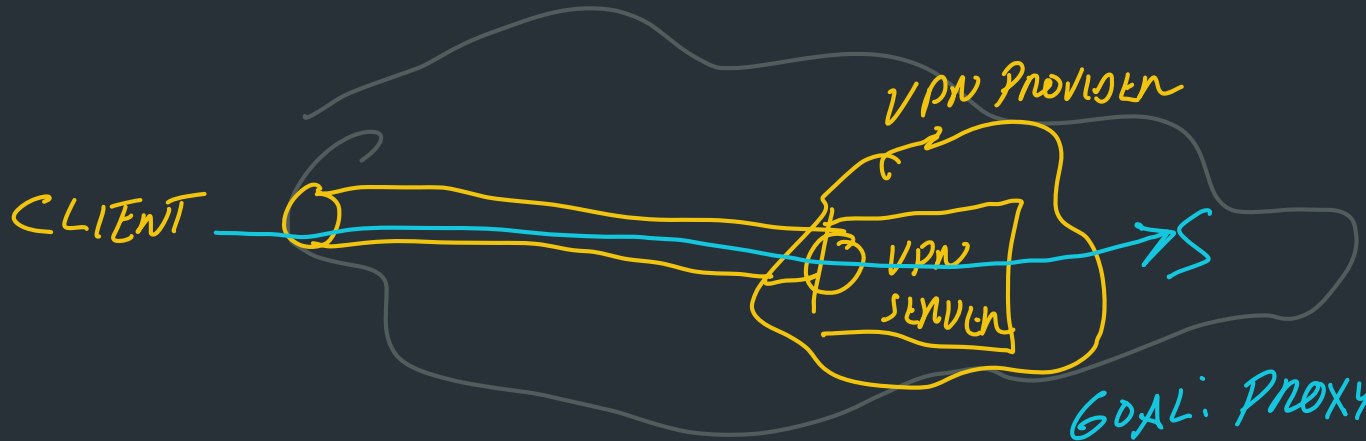
⇒ Security for all your network traffic => not just one 5-tuple

⇒ Can (try to) provide more anonymity

VPN: secure tunnel for network traffic

=> Connect a host to a private network





Virtual Private Network (VPN)

Secure tunnel for arbitrary network traffic (any IP packets)

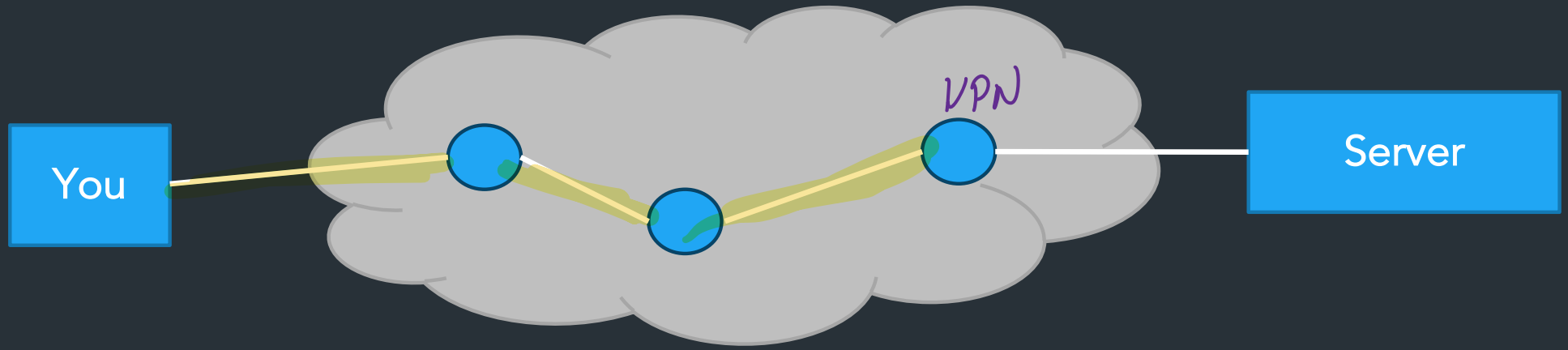
Use for

=> Accessing a private network (remote access internal network)

=> Secure proxy for your traffic: traffic appears to originate from VPN server

Problems?

A bit more context on VPNs... sorry my demo broke!
See end of notes for even more!



Security implications: VPNs

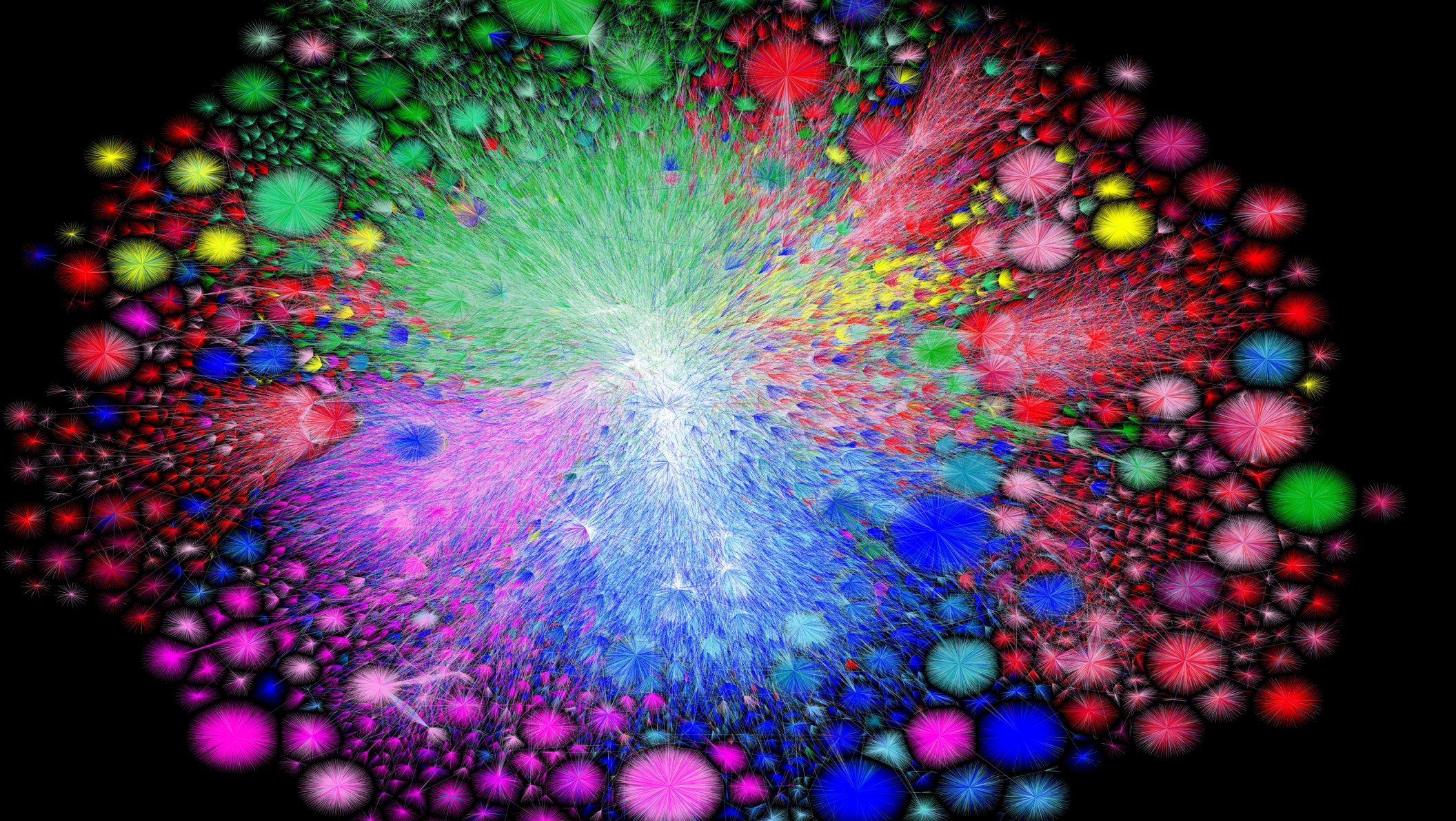
- Traffic still might be in the clear (vulnerable to eavesdropping/ analysis) once it leaves VPN
- Your traffic may blend in with other users (other VPN clients on same IP), but maybe not
  => Possible to de-anonymize user with timing/correlation info

- VPN provider still knows who you are
  => Can we do better? See extra notes for more!

Wrapping up

- This is our last formal lecture
- From here: work on final project

What I hope you have learned



We can't cover (or remember) everything

Hope you learn important tools/principles to understand networking challenges you encounter

Protocols Ways to communicate between *heterogeneous* systems

Network programming

```
conn, err :=  net.Dial("tcp", "10.0.0.1:80")
. . .

someBuf := make([]byte, . . .)
conn.Write(someBuf)
```

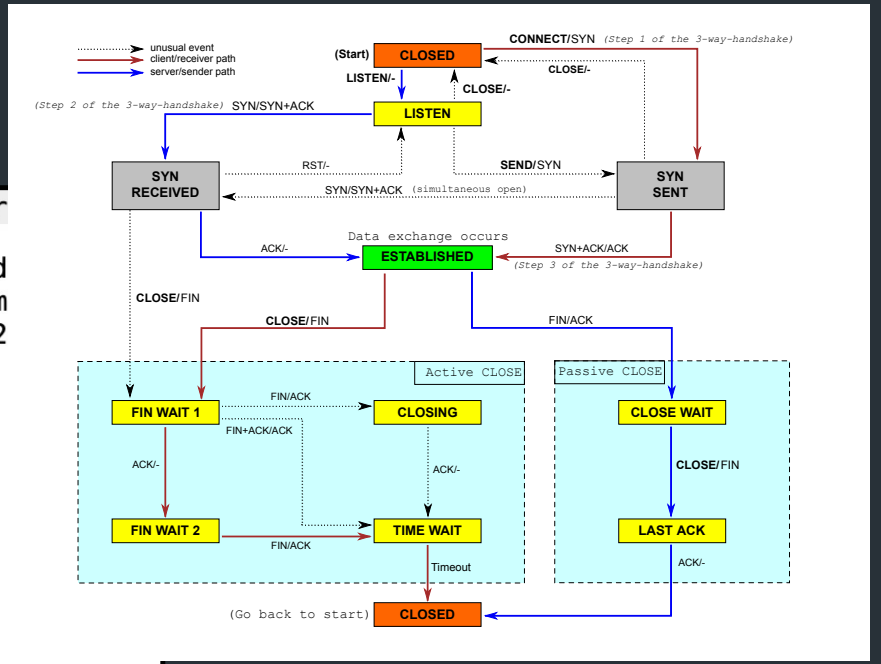
From: draft-ietf-tcpm-rfc793bis-28 Internet Standar

Internet Engineering Task Force (IETF) W. Eddy, Ed
STD: 7 MTI System
Request for Comments: 9293 August 202
Obsoletes: 793, 879, 2873, 6093, 6429, 6528, 6691
Updates: 1011, 1122, 5961
Category: Standards Track
ISSN: 2070-1721

Transmission Control Protocol (TCP)

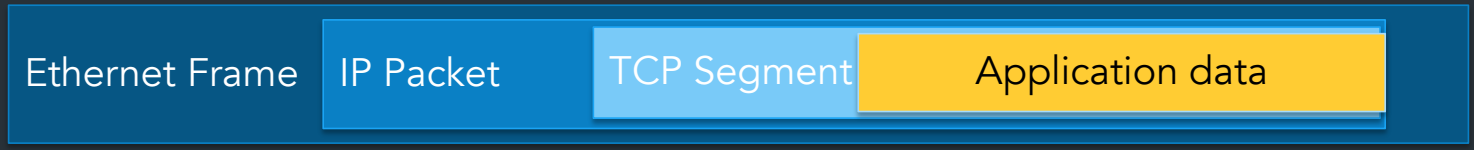
Abstract

This document specifies the Transmission Control Protocol (TCP). TCP is an important transport-layer protocol in the Internet protocol stack, and it has continuously evolved over decades of use and growth



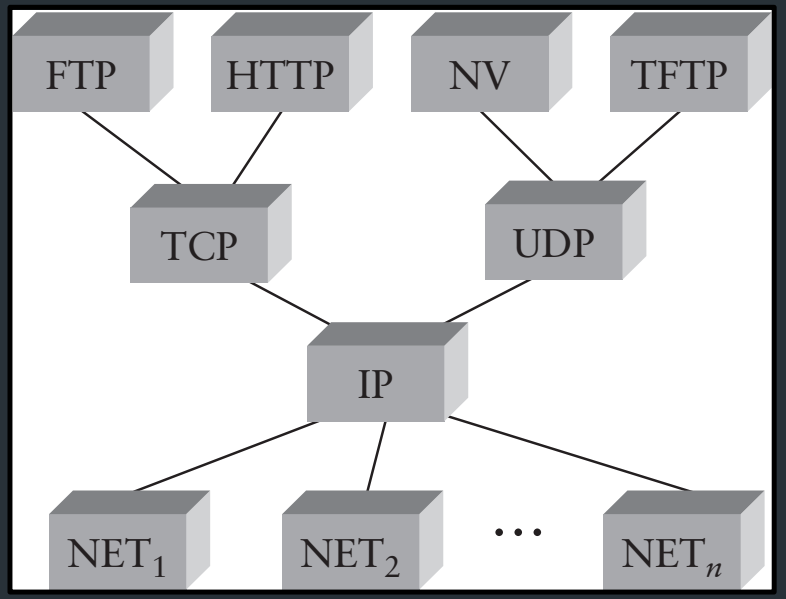
Layering / Encapsulation

Building abstractions and interfaces to hide lower-level details from "higher" layers



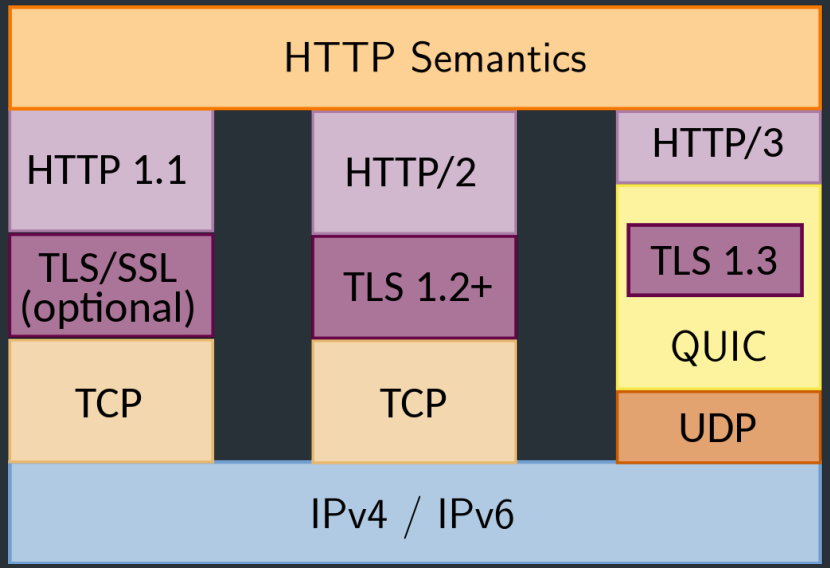
Abstractions are great!

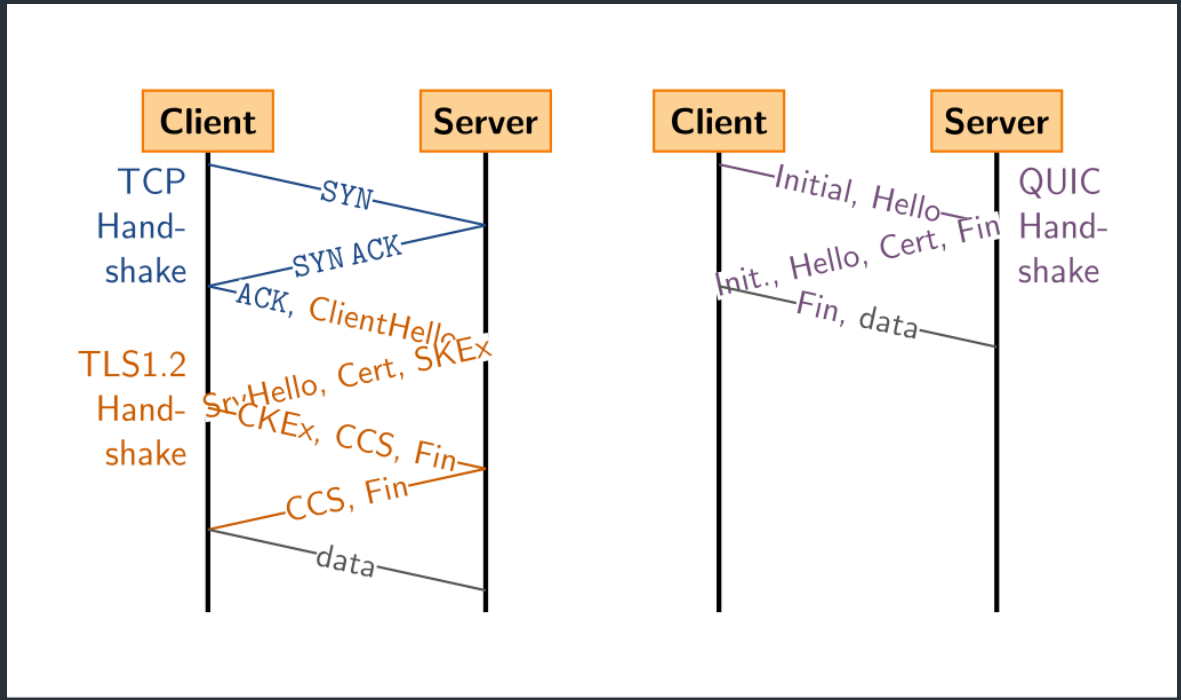
- Can support huge variety of devices, protocols
- Allows independent evolution => new protocols!



… until they aren't

Sometimes, need to break them





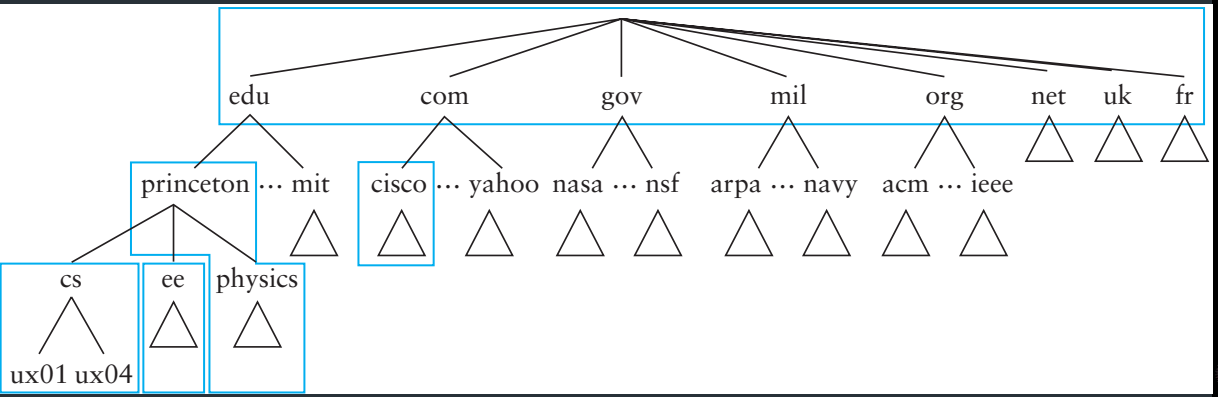
Naming

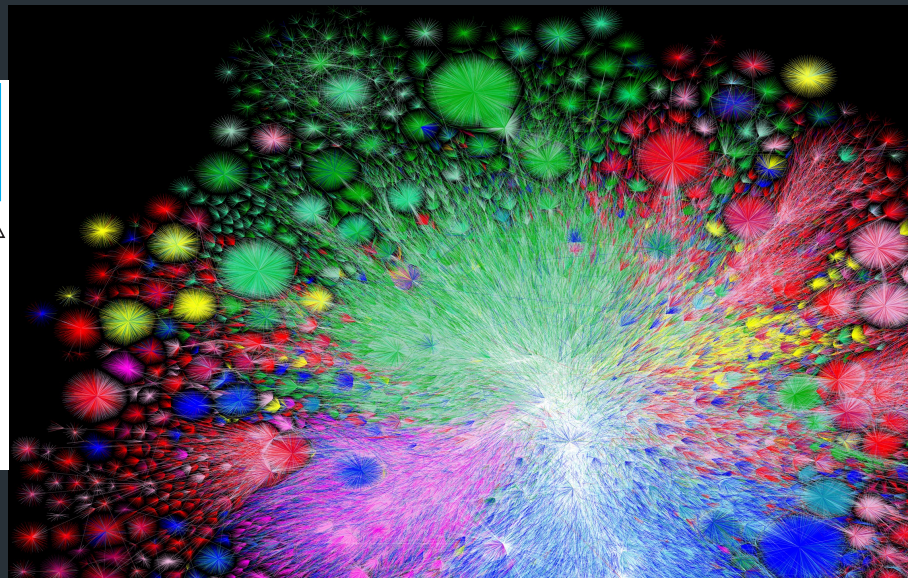
Indirection: abstract low-level info with a higher-level name

=> Human-readable DNS names

=> Scalability: redundancy, proxies, load balancing

Can leverage hierarchy of naming => scalability (IP, DNS, …)





How naming, etc. can be controlled…

Changing DNS servers in response to blocking of Twitter in Turkey (2014)

Writeup, with more links: https://www.thousandeyes.com/blog/internet-censorship-around-the-world

Lots of challenges out there

Our Internet architecture was designed in the 1980s, where modern scale and complexity was unimaginable

Now…

- No one knows how big the Internet is
- No one is in charge
- Anyone can add any application
- Packets traverse many paths, countries, regulatory domains

Other CS courses that may interest you if you liked 1680:

CS 1670: Operating systems (Malte Schwarzkopf)
CS 2680: Grad seminar on networking (Akshay Narayan)
CS 2690: Datacenter and cloud operating systems (Deepti Raghavan)
CS 1675: Designing high-performance network systems
CS 2390: Privacy-Conscious Computer Systems

Thank you!
Please stay in touch!

Extra notes: more content in case you want to read further

These extra notes have two parts:

- *More about TLS*
    - *Ways TLS can be compromised*
    - *How CAs validate your identity (cf. Let's Encrypt)*

- *More about anonymity*
    - *Tor: onion routing*

Hope you enjoy!

Part 1: More on TLS

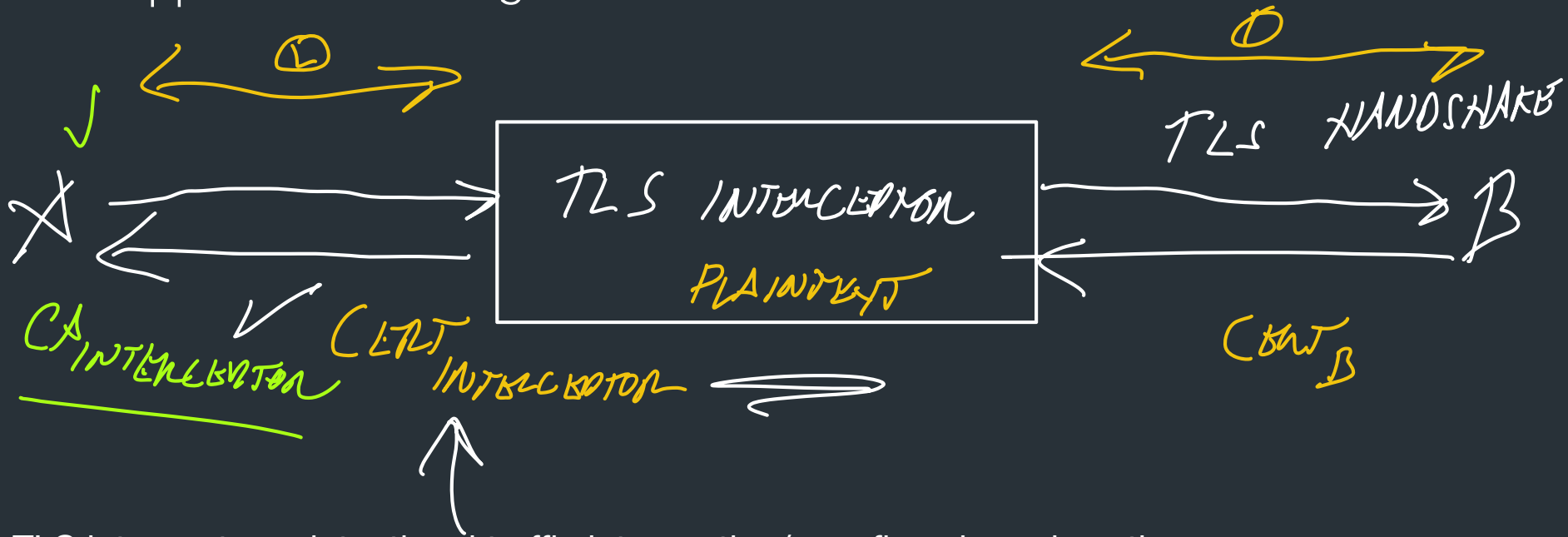
Rogue Certificates?

- In 2011, DigiNotar, a Dutch root certificate authority, was compromised
- The attacker created rogue certificates for popular domains like google.com and yahoo.com
- DigiNotar was distrusted by browsers and filed for bankruptcy
- See the incident investigation report by Fox-IT

- In 2017, Google questioned the certificate issuance policies and practices of Symantec
- Google's Chrome would start distrusting Symantec's certificates unless certain remediation steps were taken
- See back and forth between Ryan Sleevi (Chromium team) and Symantec
- The matter was settled with DigiCert acquiring Symantec's certificate business

TLS "decryption"

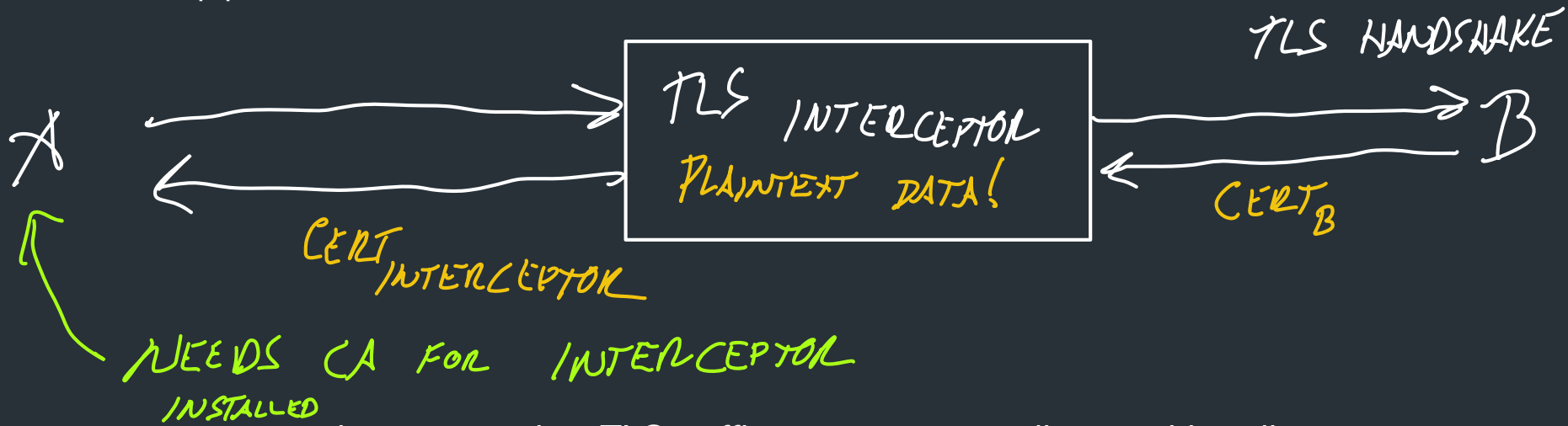
What happens when an organization wants to view TLS traffic on its network?



TLS interceptors: intentional traffic interception/spoofing--how does the browser still think it's valid???

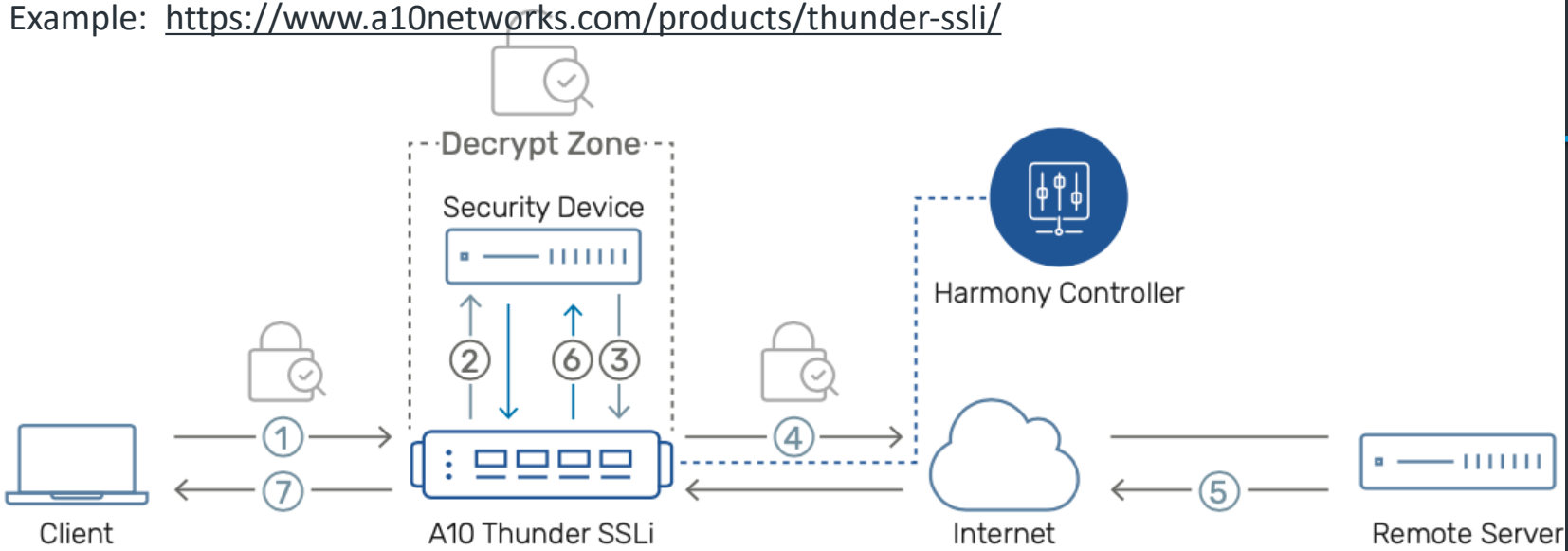
TLS decryption

What happens when an organization wants to view TLS traffic on its network?



Some corporate networks want to view TLS traffic to ensure compliance with policy
=> Forward all traffic through TLS interceptor: client does TLS handshake with interceptor, then interceptor connects to actual server, allowing it to see all data
=> When A does the TLS handshake with the interceptor, it gets back a fake certificate from the interceptor, not B. How does this pass verification? Company needs to install a CA on A
=> *This is intentional traffic interception/spoofing—thoughts?*

Example: https://www.a10networks.com/products/thunder-ssli/



1. Encrypted traffic from the client is intercepted by Thunder SSLi and decrypted.
2. Thunder SSLi sends the decrypted traffic to a security device, which inspects it in clear-text.
3. The security device, after inspection, sends the traffic back to Thunder SSLi, which intercepts and re-encrypts it.
4. Thunder SSLi sends the re-encrypted traffic to the server.
5. The server processes the request and sends an encrypted response to Thunder SSLi.
6. Thunder SSLi decrypts the response traffic and forwards it to the same security device for inspection.
7. Thunder SSLi receives the traffic from the security device, re-encrypts it and sends it to the client.

Larger problem: how do we trust that CAs are issuing certificates properly?

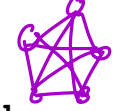
Certificate Transparency (RFC9162, 2021): Recent effort to provide open standard to monitor how certificates are issued

- Verifiable, append-only logs of all certificates issued (built using Merkle trees)
- Browsers, CAs, other interested parties can maintain logs

Modern browser vendors are starting to require that CAs use Certificate Transparency in order to be included as a trusted CA

Example CT monitor: https://crt.sh

Aside: are there other methods of delegating trust?



- Web of trust: small group of parties that sign each other's keys
  => Have a threshold on how many signatures you need to be "trusted"
  => Doesn't scale to entire internet, but exists for small communities (esp. open-source software projects)

- Trust on first use (TOFU)
  - ON first connection, ask user if they trust the public key (y/n)
  - If user says yes, trust key for all time
  - If public key changes later, something sketchy is happening => trust error
  => SSH (by default)

Also: PKI comes up in other ways outside of TLS:
- DNSSEC has a similar hierarchy (root zone ~= trusted CA)
- Similar certificates used for secure email (S/MIME) or some other related authentication standards

Q: If private key is compromised, can attacker decrypt data?

Not if TLS connection uses forward secrecy

⇒ Cannot recover session key if server private key leaked

⇒ Once optional, now required by TLS 1.3 (2018)

Q: If private key is compromised, can attacker decrypt data?

Not if TLS connection uses forward secrecy

⇒ Cannot recover session key if server private key leaked

⇒ Once optional, now required by TLS 1.3 (2018)

Website protocol support (May 2024)

Protocol version	Website support[92]	Security[92][93]
SSL 2.0	0.1%	Insecure
SSL 3.0	1.4%	Insecure[94]
TLS 1.0	27.9%	Deprecated[20][21][22]
TLS 1.1	30.0%	Deprecated[20][21][22]
TLS 1.2	99.9%	Depends on cipher[n 1] and client mitigations[n 2]
TLS 1.3	70.1%	Secure

In practice, TLS 1.3 rollout delayed by many broken TLS implementations (eg. in-network middleboxes/proxies) …

Website protocol support (May 2024)

Protocol version	Website support[92]	Security[92][93]
SSL 2.0	0.1%	Insecure
SSL 3.0	1.4%	Insecure[94]
TLS 1.0	27.9%	Deprecated[20][21][22]
TLS 1.1	30.0%	Deprecated[20][21][22]
TLS 1.2	99.9%	Depends on cipher[n 1] and client mitigations[n 2]
TLS 1.3	70.1%	Secure

In practice, TLS 1.3 rollout delayed by many broken TLS implementations (eg. in-network middleboxes/proxies) …

Remember how we said don't propagate buggy behavior in TCP?

Website protocol support (Sept 2023)

Protocol version	Website support[87]	Security[87][88]
SSL 2.0	0.2%	Insecure
SSL 3.0	1.7%	Insecure[89]
TLS 1.0	30.1%	Deprecated[20][21][22]
TLS 1.1	32.5%	Deprecated[20][21][22]
TLS 1.2	99.9%	Depends on cipher[n 1] and client mitigations[n 2]
TLS 1.3	64.8%	Secure

COMPARE!

In general, implementing security protocols is hard to get right

=> TLS libraries are very critical and need lots of oversight/auditing

=> Servers (and clients) need to be updated with latest standards/fixes

Se

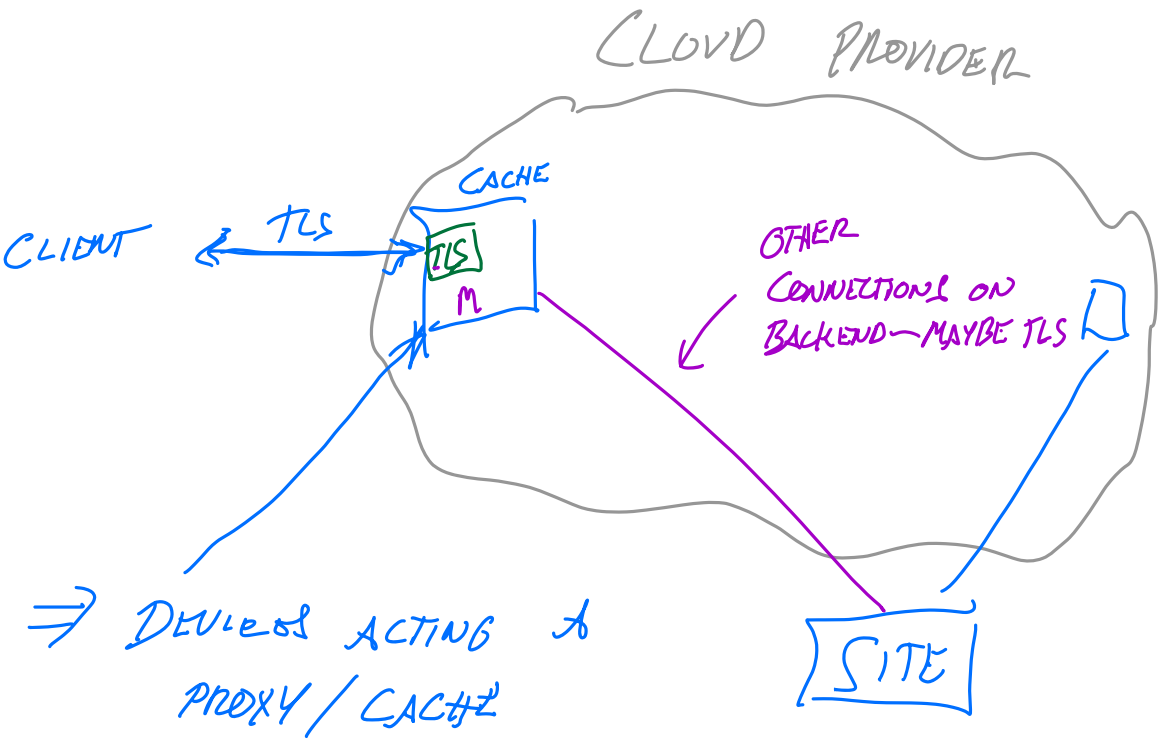
As of July 2021, the Trustworthy Internet Movement estimated the ratio of websites that are vulnerable to TLS attacks.[71]

Survey of the TLS vulnerabilities of the most popular websites

Attacks	Security			
	Insecure	Depends	Secure	Other
Renegotiation attack	0.1% support insecure renegotiation	<0.1% support both	99.2% support secure renegotiation	0.7% no support
RC4 attacks	0.4% support RC4 suites used with modern browsers	6.5% support some RC4 suites	93.1% no support	N/A
TLS Compression (CRIME attack)	>0.0% vulnerable	N/A	N/A	N/A
Heartbleed	>0.0% vulnerable	N/A	N/A	N/A
ChangeCipherSpec injection attack	0.1% vulnerable and exploitable	0.2% vulnerable, not exploitable	98.5% not vulnerable	1.2% unknown
POODLE attack against TLS **(Original POODLE against SSL 3.0 is not included)**	0.1% vulnerable and exploitable	0.1% vulnerable, not exploitable	99.8% not vulnerable	0.2% unknown
Protocol downgrade	6.6% Downgrade defence not supported	N/A	72.3% Downgrade defence supported	21.0% unknown

Wikipedia table, source: https://www.ssllabs.com/ssl-pulse/

Additional Stuff: Caching + TLS

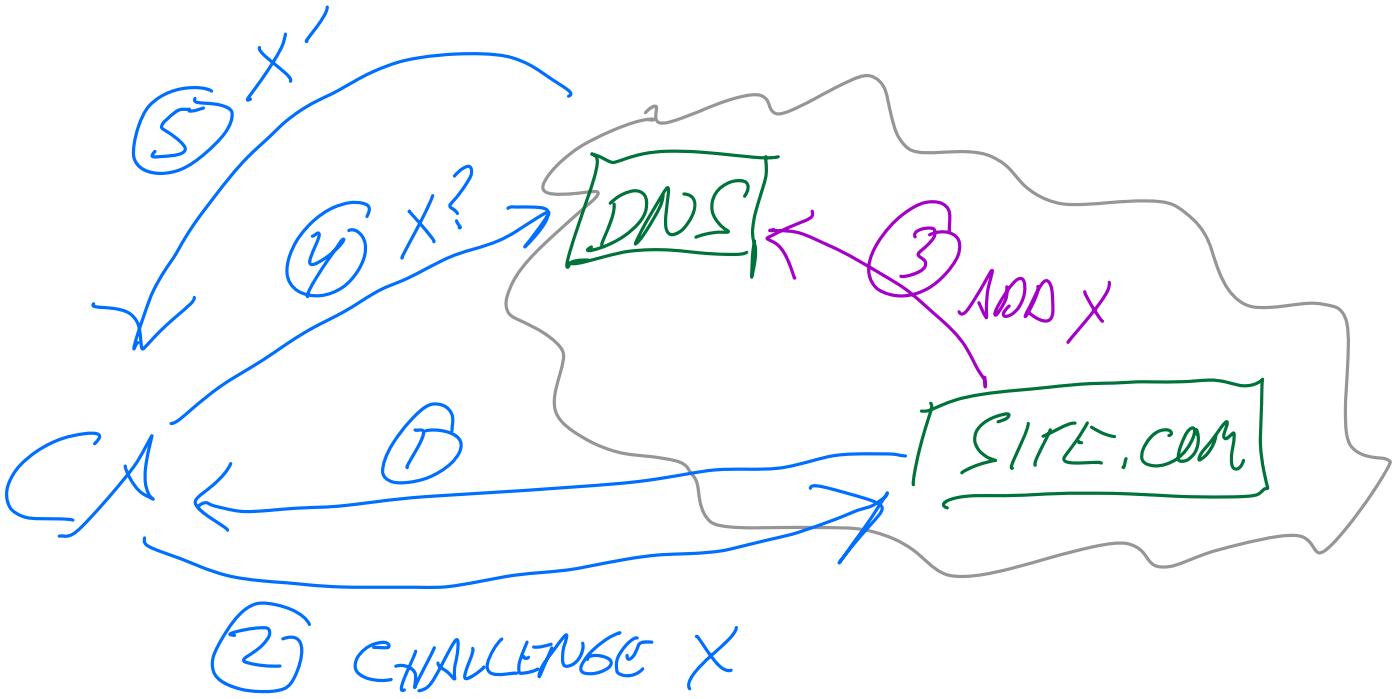


How does caching work with TLS?
- Client makes a TLS connection to some endpoint at cloud provider (cache, etc), not the backend server
- From there, the cache can see the client's request, then respond with cached data or query backend server
  => Cache needs to have certificate
  => Traffic is decrypted in the cloud provider (may or may not be what you want)

How does a CA validate a certificate request?

Before signing a certificate, a CA should check the requestor's identity in some way. Two ways to do this:

- Organization validation (less common): manually verify contact info, in-person, etc.
- Domain validation (most common): verify that the requestor is in control of the domain name where they are requesting the certificate



How domain validation works:

1. Admin of some site site.com asks CA for certificate
2. CA issues challenge with random value X, asks requestor (admin, etc) to make it viewable on their site. Examples:
   A. eg. Add a DNS record on site.com containing challenge value (TXT record)
   B. Make challenge available on website (ACME protocol)
3. The CA checks for challenge value (DNS lookup for site, etc.) => finds challenge X'
4. If X == X', it means that the requestor can prove control of the site

Eg. Let's Encrypt (2014): Free CA that issues certificates using this method => now extremely common, issues >1M certificates per day

Problem: what if attacker can hijack DNS? Could spoof validation process with spoofed responses, BGP hijacking, …

One solution: need to verify challenge from multiple vantage points (ASes) to avoid querying from one bad server/path

Part 2: More on anonymity

Q: How does a VPN client get all traffic to go to the VPN?

A VPN client will create a special network interface (on Linux, called a TUN interface) and then update the routing table to redirect (usually) all traffic to the VPN

Example: Normal routing table

```
default via 138.16.161.1    # Brown's router
138.16.161.0/24 wifi0
```

=> When you start a VPN client and make a connection, it adds an interface to this list

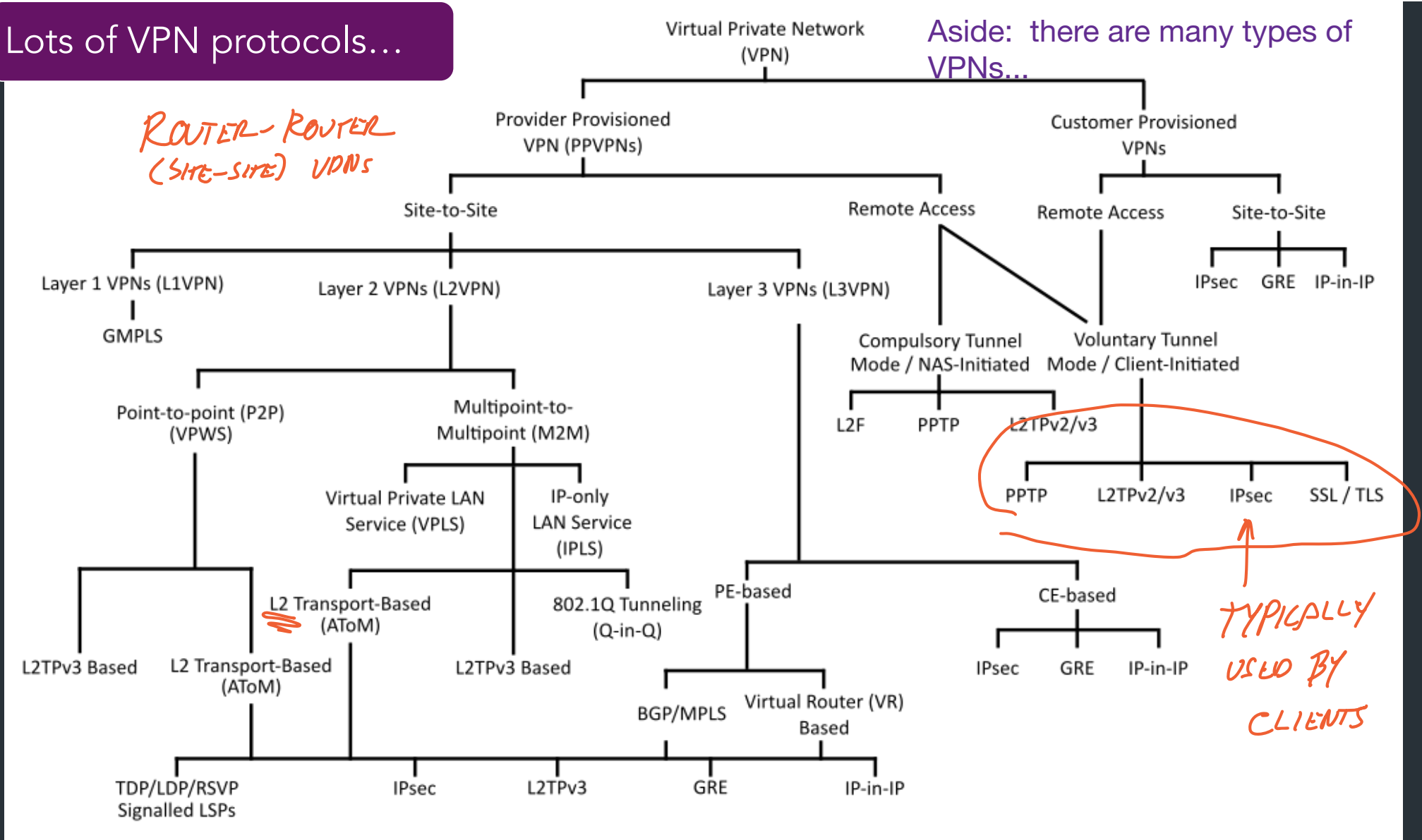
```
default via 10.2.3.4 priority 1
default via 138.16.161.1 priority 2  # Brown's router
10.2.3.4 via tun0
138.16.161.0/24 via wifi0
```

A bit more complicated than this in practice:

- Some VPN clients just send some traffic to the VPN (eg. company network)
- Need to make sure that traffic for the VPN client itself can still access the Internet
- If connection drops, need to adjust rules carefully so that 1) traffic you want to be encrypted isn't leaked, and 2) VPN client can still reconnect

Lots of VPN protocols…

Aside: there are many types of VPNs…

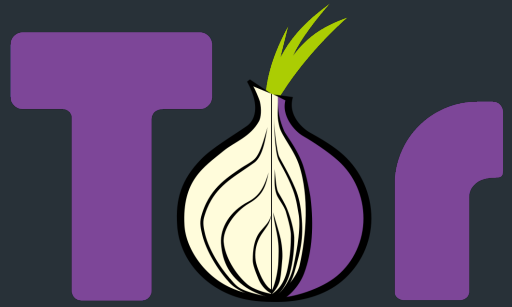


ROUTER-ROUTER (SITE-SITE) VPNs

TYPICALLY USED BY CLIENTS

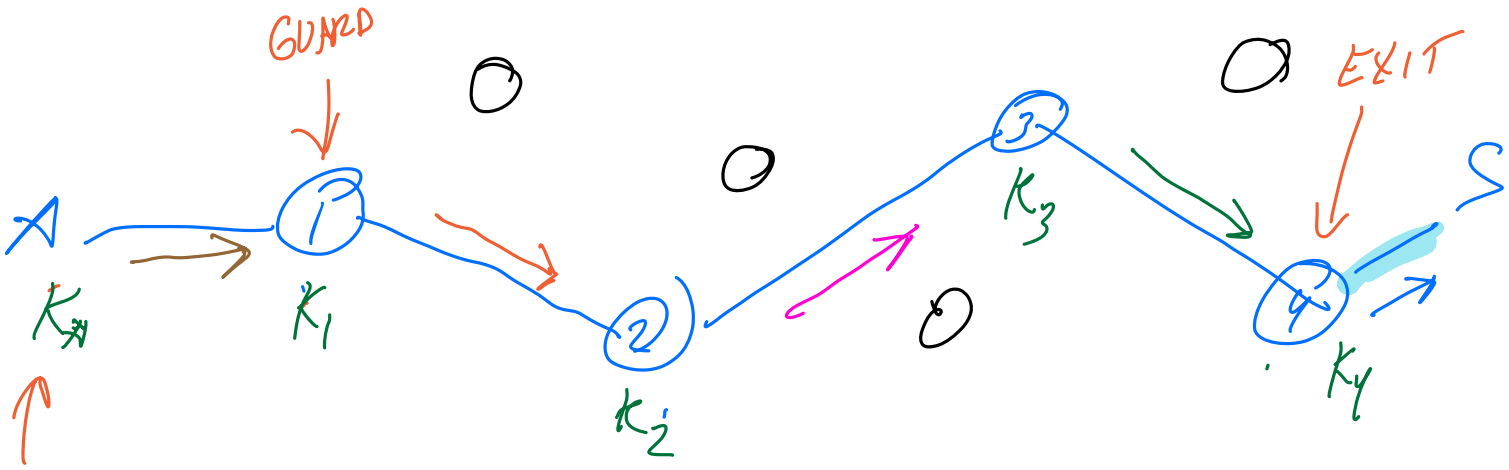
Can we do better?

Tor

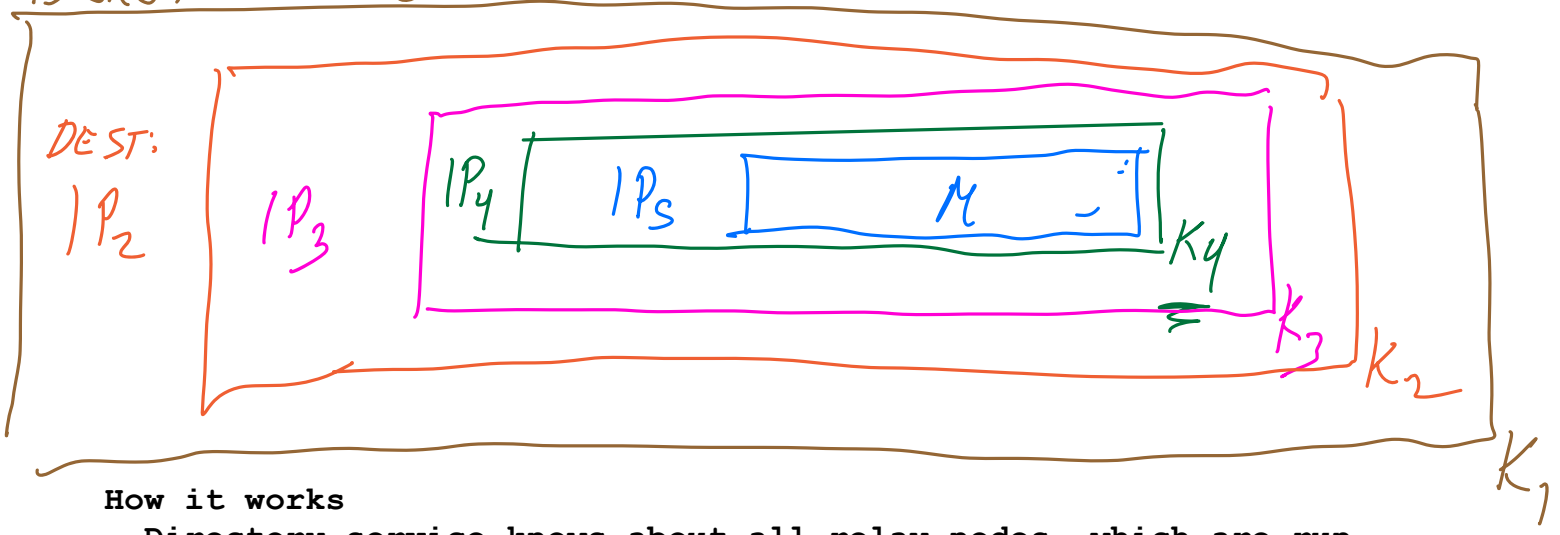


- Onion routing service: build encrypted circuit on tor relay network
- Network of relays, mainly operated by volunteers
- Started in 1990s from Naval Research Lab, now maintained by The Tor Project (a non-profit)

ONION ROUTING



PACKET RECEIVED BY NODE 1:



How it works

- Directory service knows about all relay nodes, which are run by many volunteers
- At connection start, establish a "circuit" or path from a subset of relays => know a symmetric key for each relay
  - First node is "guard node"
  - Last relay node is "exit node"

Send packets with layers of encryption => at each hop, relay can decrypt its layer of the encryption with its key

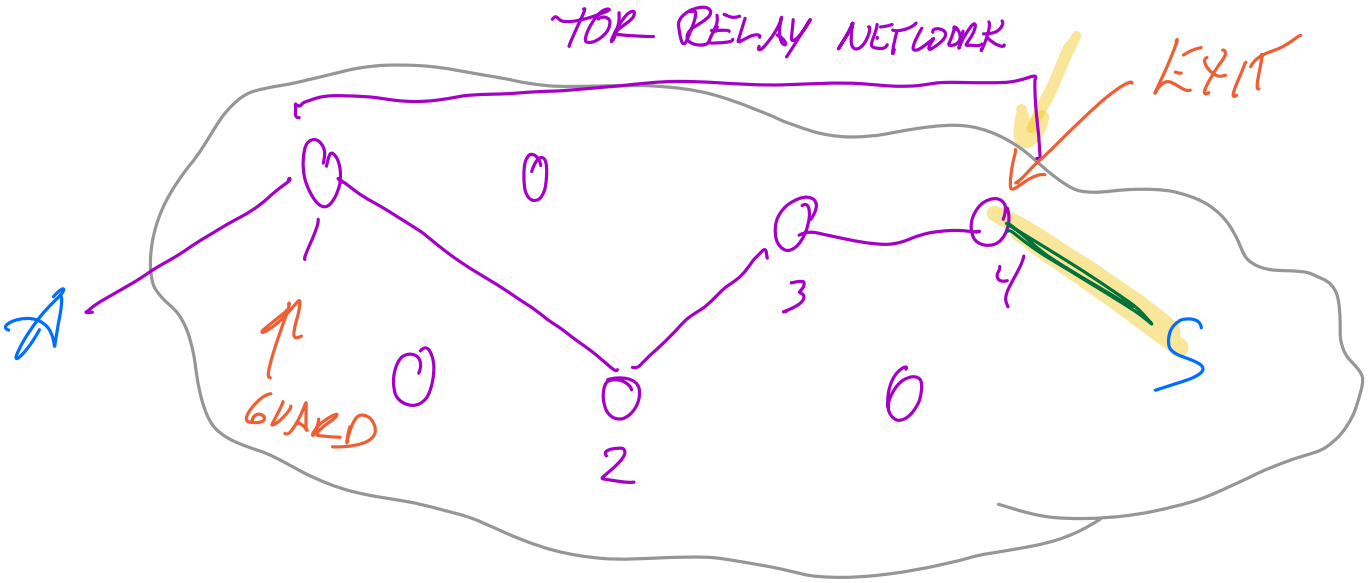
- Unless it's the last node, it can only see the destination of the next node and the encrypted packet

  (At exit node, the packet is in cleartext => goes to destination)

Thus:

- Only the guard node knows about the host
- Only exit node knows the packet's true destination
- Relays in middle only know about next hop

HOW TOR WORKS: RECAP



- ONLY GUARD NODE KNOWS A
- ONLY EXIT NODE KNOWS S
- IDEALLY, RELAYS OWNED BY MANY DIFFERENT PARTIES

Last hop => traffic is leaving tor network to reach destination server => not protected!

- If not using TLS or other protocol-level security, data is in the clear

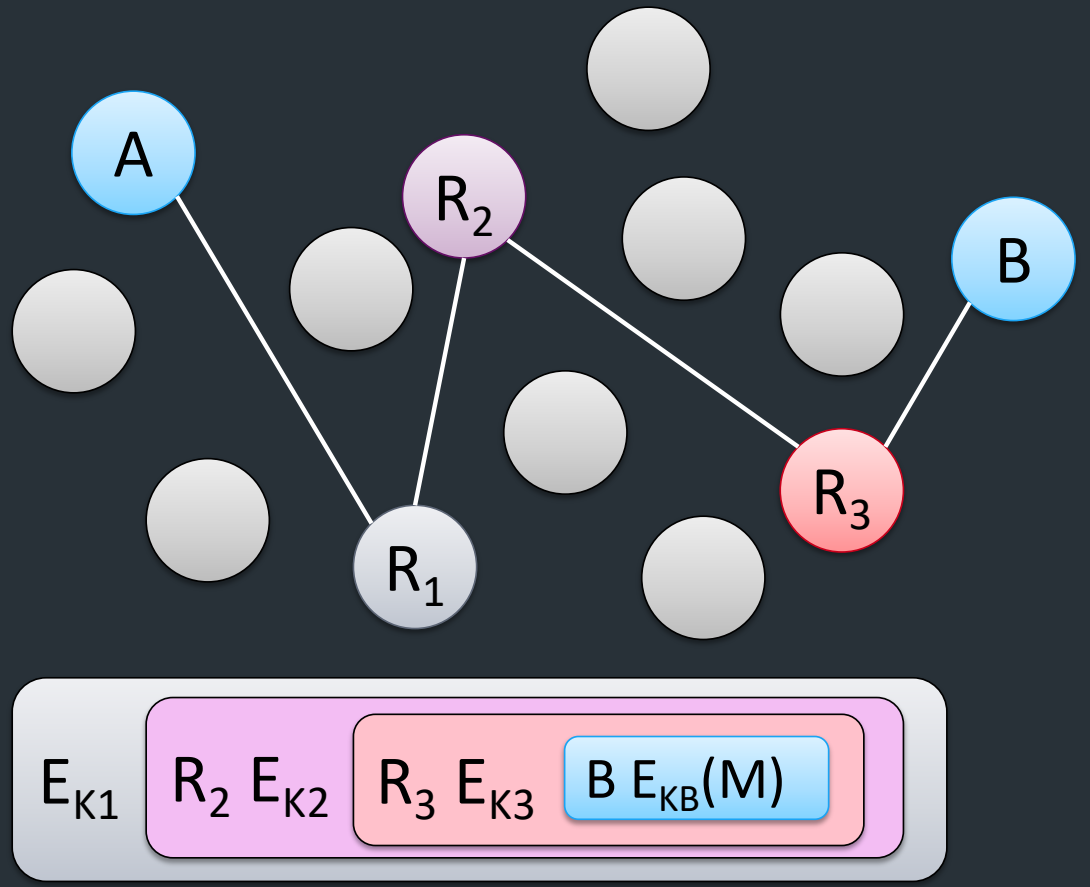
- Depending on the protocol/messages, may leak information that identifies you (eg. cookies, protocol info that contains your IP address)

Q: Why does tor require its own browser? (other than because it's easy)

=> If you used your normal browser, your existing browser state (cookies, etc) can be sent when you visit pages => more likely to identify you

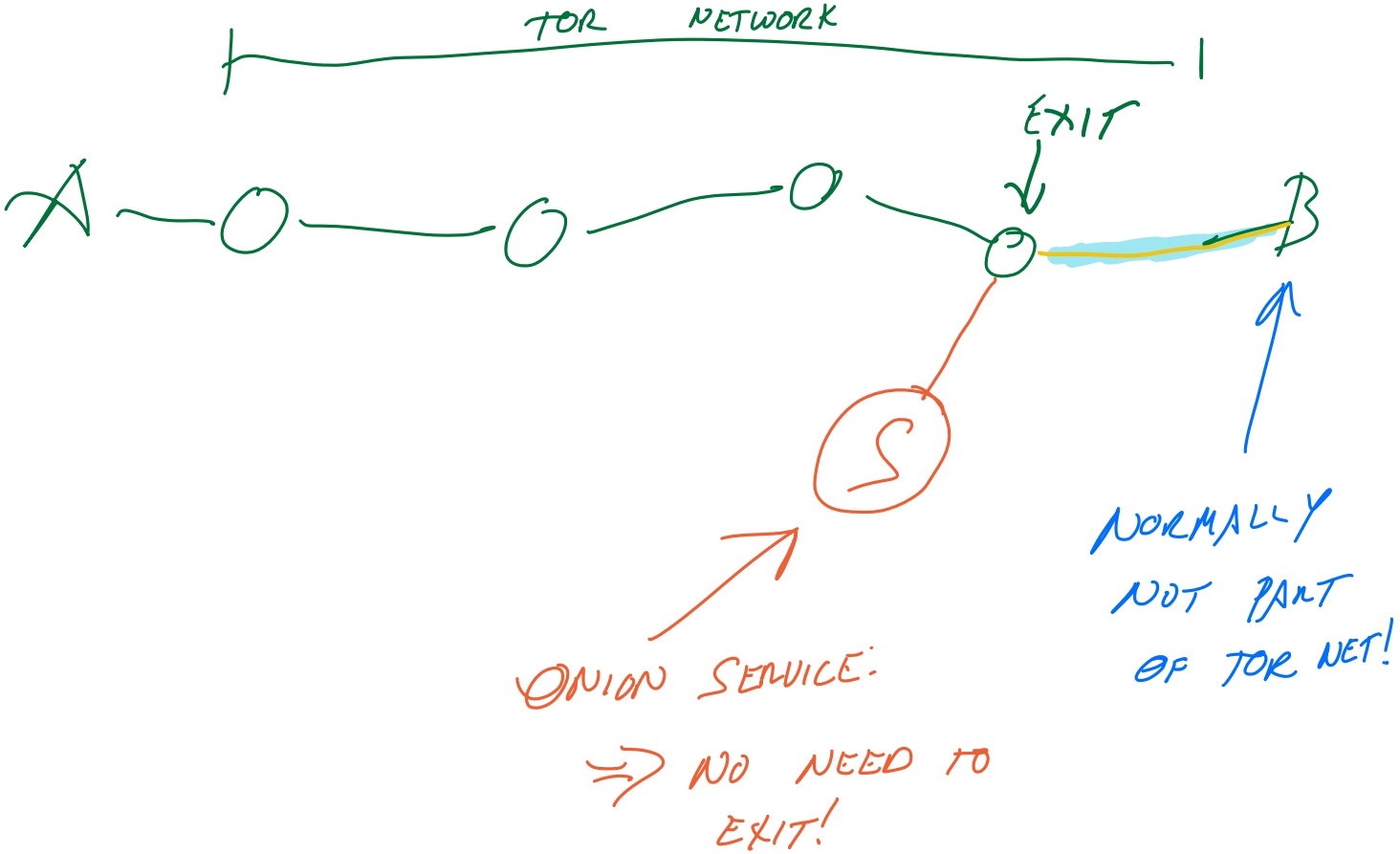
Onion Routing

- Layered encryption
  - Build onion inside out
- Routing
  - Peel onion outside in
- Each router knows only previous and next



Normally: final destination of user's connection (eg. B) is a regular server on the Internet and not aware of tor
=> Traffic is decrypted when it reaches the exit node, send to the public Internet

But what if the server also joins the tor network?
=> Onion services
=> No need for an exit node! Need a "name" for reaching them...



What if the server wants to help?

Onion services: server connects to tor directly => no need for an exit node!

- Accessible via .onion domain: special DNS TLD not in root zone
- Site addresses based on public key of server, client looks up using distributed hash table (DHT)

Examples

- *New York Times:*
  https://www.nytimesn7cgmftshazwhfgzm37qxb44r64ytbb2dj3x62d2lljsciiyd.onion
- Facebook
  https://facebookwkhpilnemxj7asaniu7vnjjbiltxjqhye3mhbshg7kx5tfyd.onion
- Cloudflare public DNS
  `dns4torpnlfs2ifuz2s2yf3fc7rdmsbhm6rw75euj35pac6ap25zgqad.onion`