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

Administrivia

- Final project: now available
  - Team form: due TODAY (12/2) by 5pm EST
  - Brief proposal: due Friday 12/6 (no late days!)

- Final homework (short): out now, due Mon, 12/9
- Short SRC component: due 12/16 (same as final project)

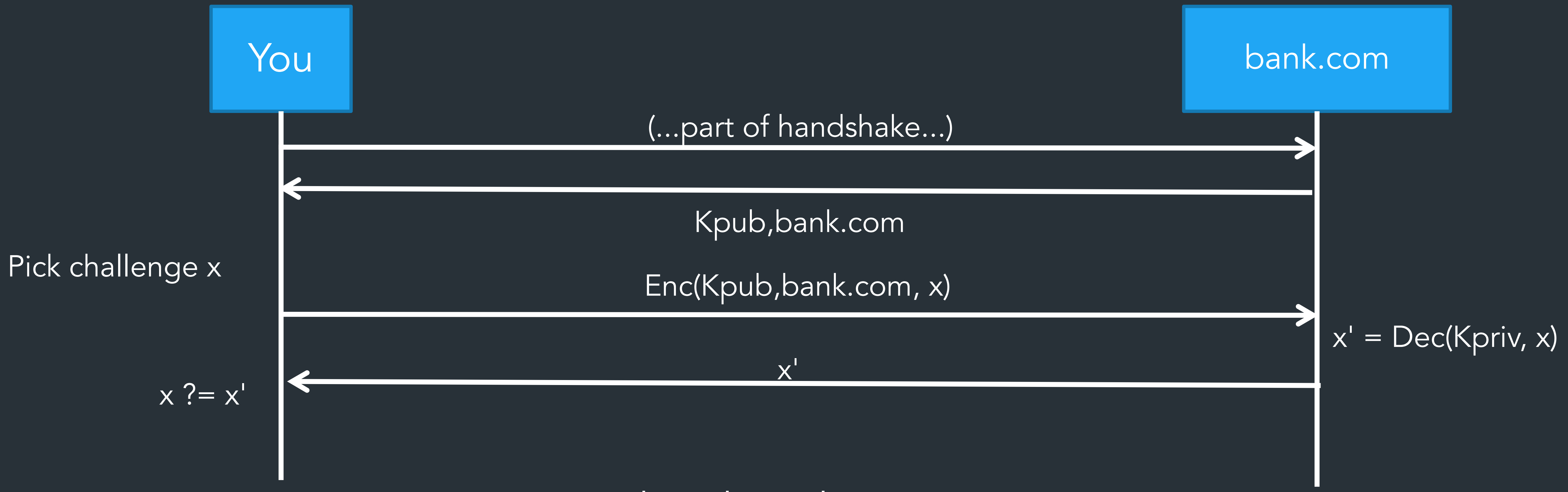
Administrivia

- Final project: now available
  - Team form: due TODAY (12/2) by 5pm EST
  - Brief proposal: due Friday 12/6 (no late days!)

- Final homework (short): out now, due Mon, 12/9
- Short SRC component: due 12/16 (same as final project)

- Most office hours end Friday, some updates this week
  - After 12/6: I will still have hours, but schedule will differ => see calendar

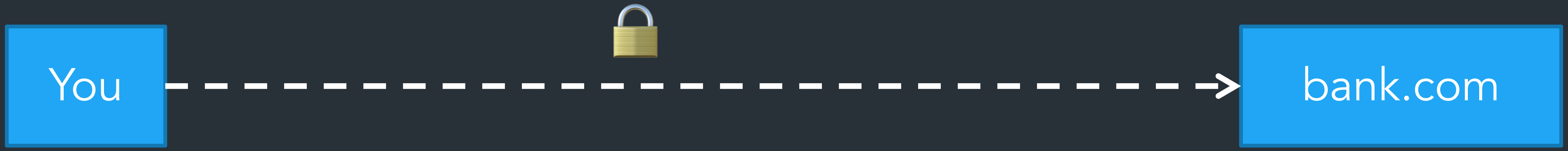
Warmup: the TLS challenge



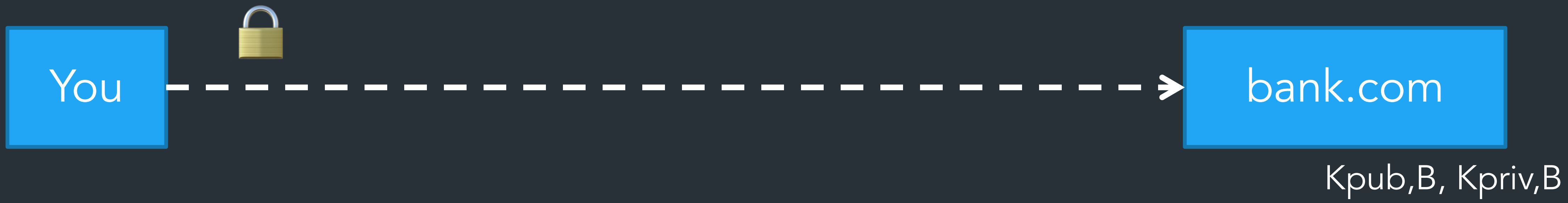
What does this prove?

Warmup

TLS: Establish a secure, bidirectional channel between two parties

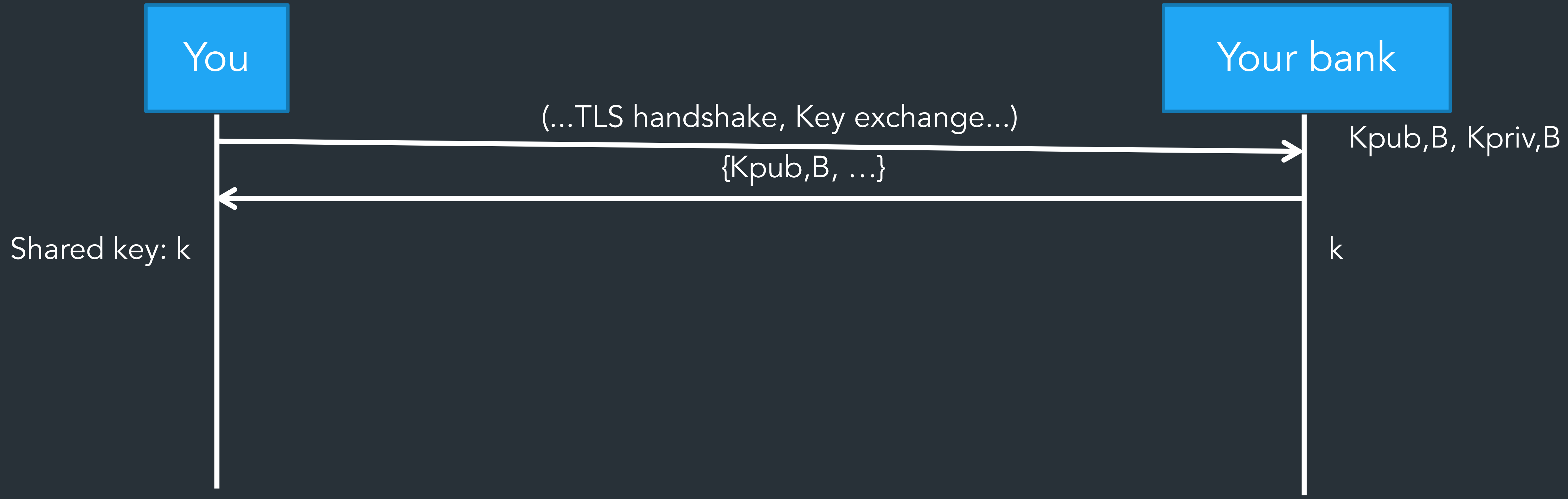


Warmup



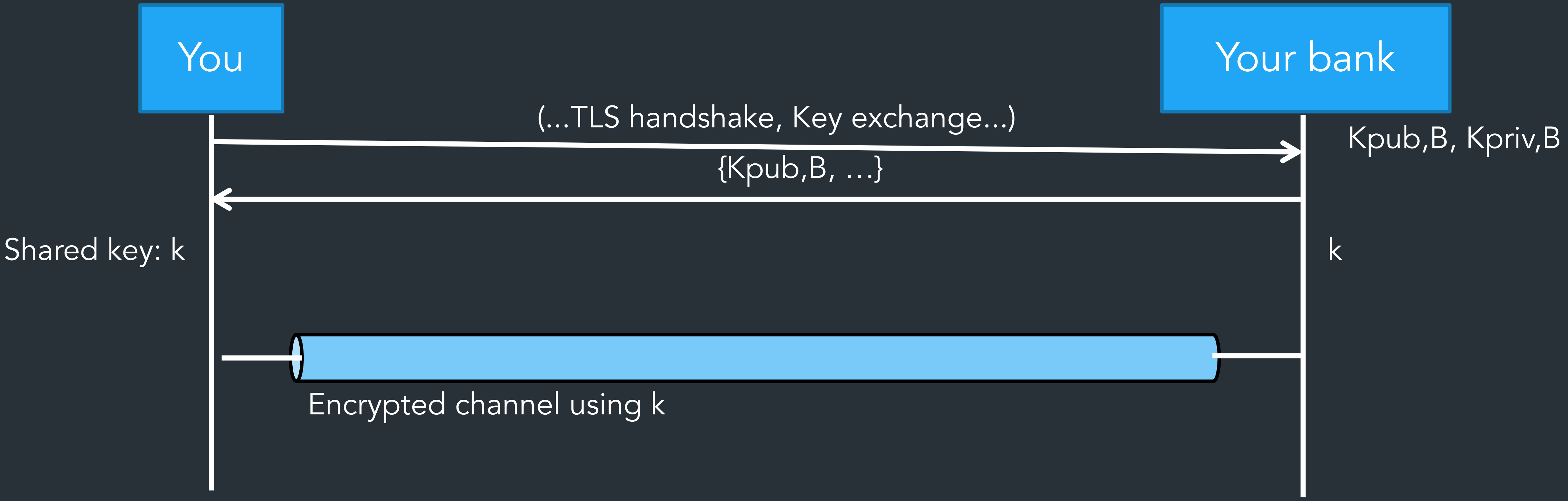
Warmup

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



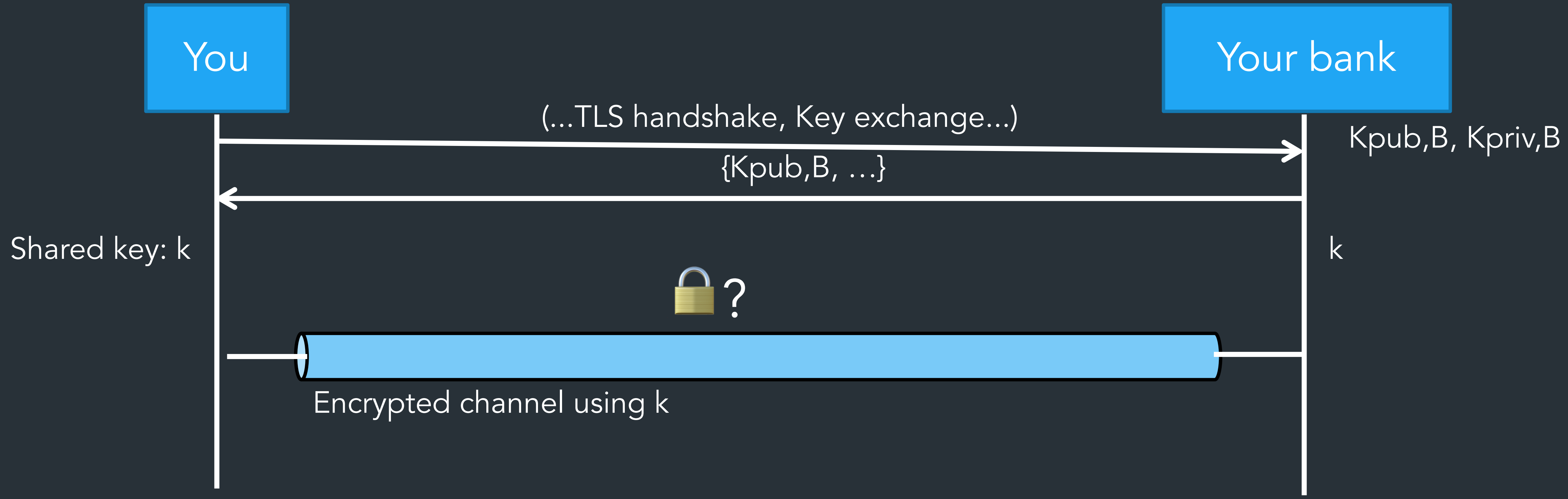
Warmup

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


You
Your bank
(...TLS handshake, Key exchange...)
Kpub,B, Kpriv,B
{Kpub,B, …}
Shared key: k
k
Encrypted channel using k

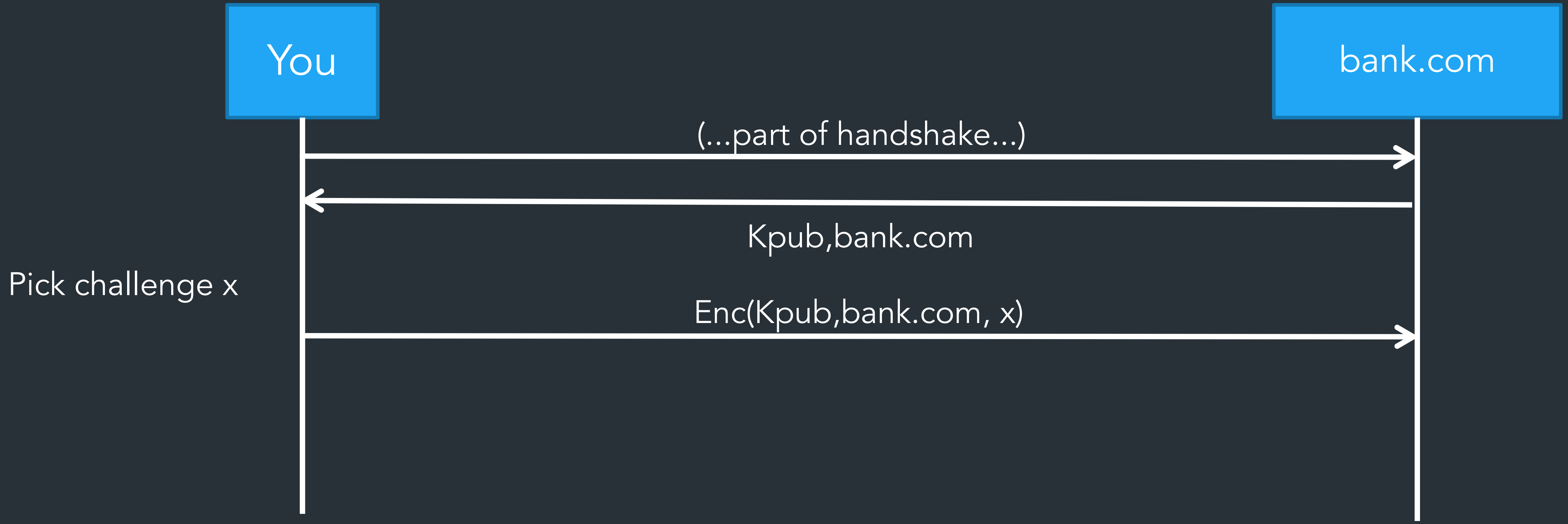
Warmup

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


You
Your bank
(...TLS handshake, Key exchange...)
Kpub,B, Kpriv,B
{Kpub,B, ...}
Shared key: k
k
🔒?
Encrypted channel using k

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

The Challenge



You

bank.com

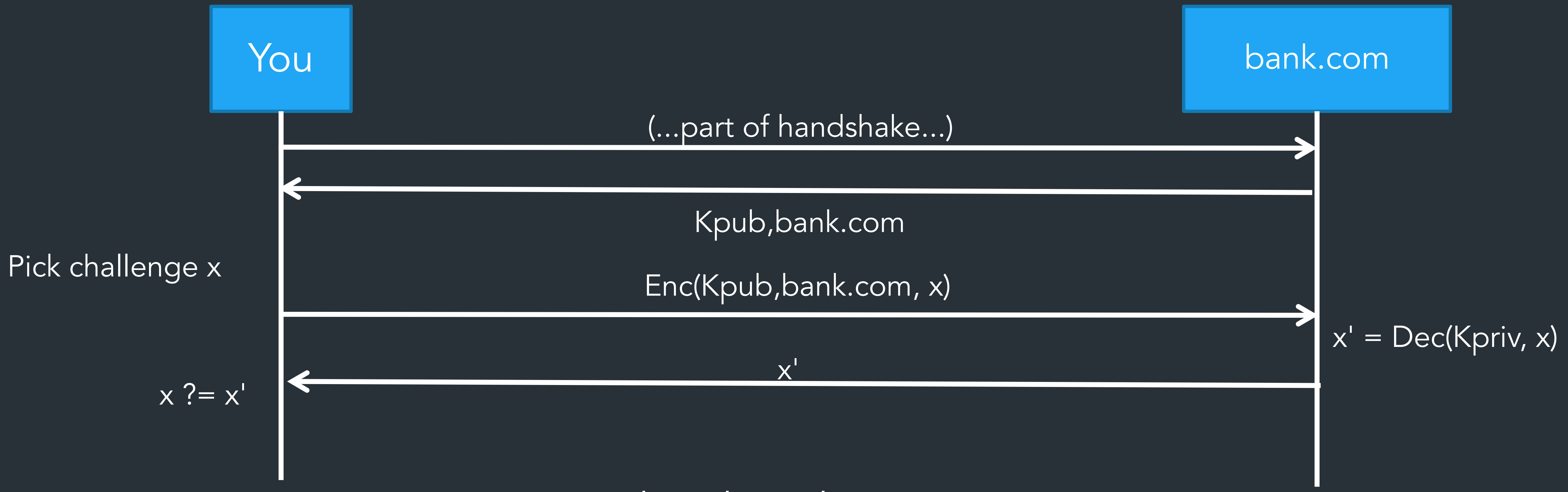
(...part of handshake...)

Kpub,bank.com

Pick challenge x

Enc(Kpub,bank.com, x)

The Challenge



What does this prove?

Authentication challenges

- Challenge proves that the server at bank.com holds K_priv
- Does NOT prove belong to the server belongs to your bank, the real-life bank with your money

Authentication challenges

- Challenge proves that the server at yourbank.com holds K_priv
- Does NOT prove the server belongs to YourBank, the real-life bank that holds your money

"But I'm visiting yourbank.com!"

- DNS can be spoofed
- Possible active network attacker (redirecting your IP traffic to malicious server)
- Domain names can expire and be re-registered...

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

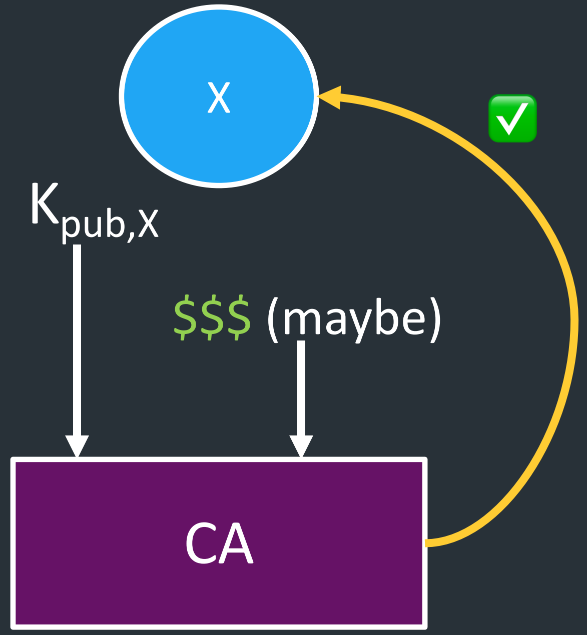
=> TLS (and others): Public Key Infrastructure (PKI) with certificates

PKI: The main idea

Public keys managed by Certificate Authorities (CAs)

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

- If X wants a public key, request from CA
  - CA supposed to validate X's identity…



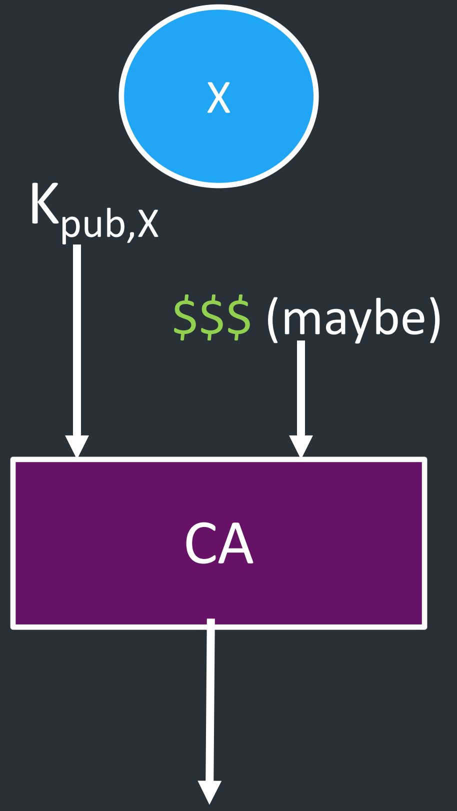
PKI: The main idea

Public keys managed by Certificate Authorities (CAs)

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

- 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



s = Sign($K_{priv,CA}$, {$K_{pub,X}$, … })

Cert = {$K_{pub,X}$, metadata, s}

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

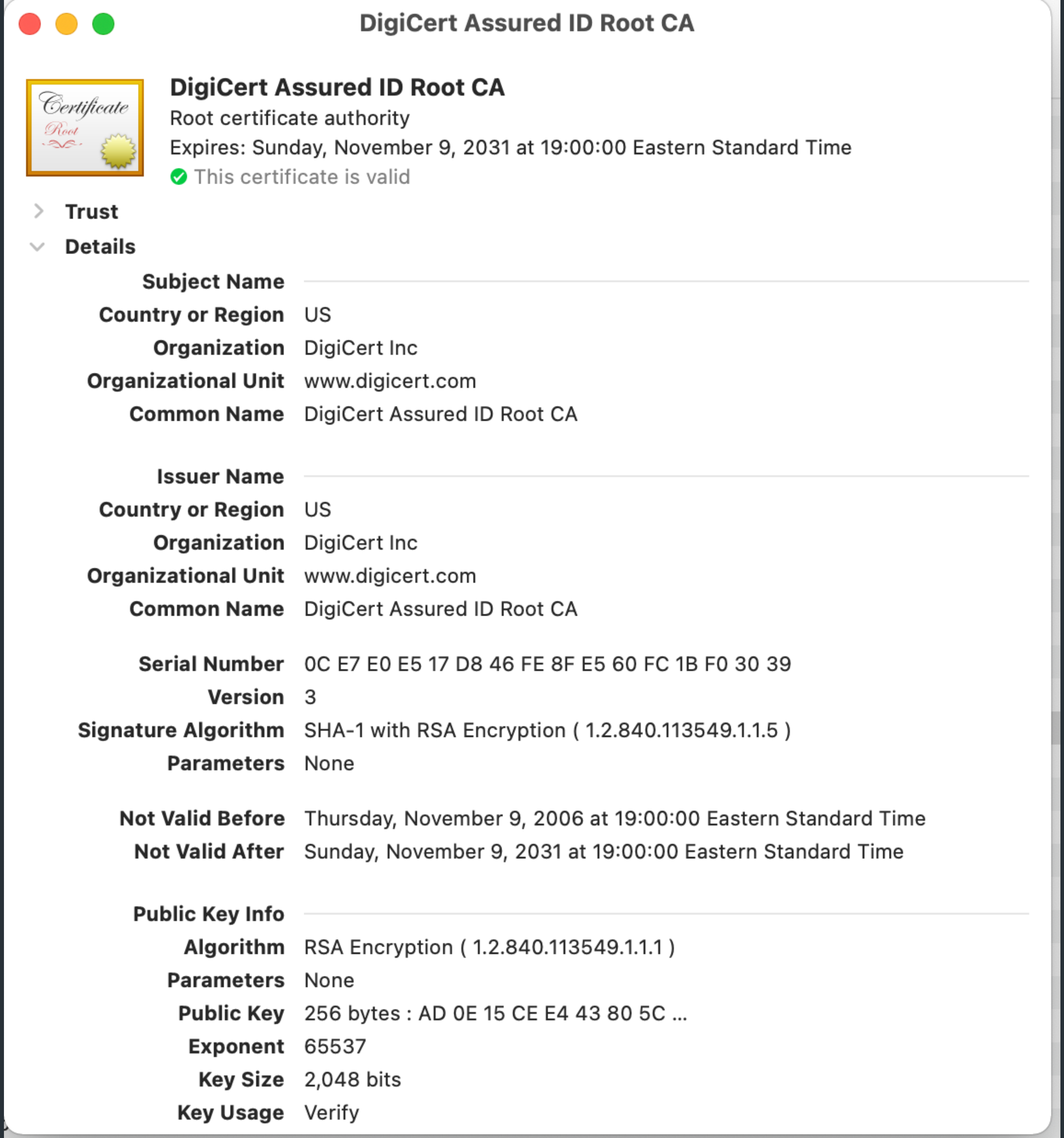
What's in a certificate?

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

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!



DigiCert Assured ID Root CA

DigiCert Assured ID Root CA

Root certificate authority

Expires: Sunday, November 9, 2031 at 19:00:00 Eastern Standard Time

✓ This certificate is valid

Subject Name

Country or Region US

Organization DigiCert Inc

Organizational Unit www.digicert.com

Common Name DigiCert Assured ID Root CA

Issuer Name

Country or Region US

Organization DigiCert Inc

Organizational Unit www.digicert.com

Common Name DigiCert Assured ID Root CA

Serial Number 0C E7 E0 E5 17 D8 46 FE 8F E5 60 FC 1B F0 30 39

Version 3

Signature Algorithm SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)

Parameters None

Not Valid Before Thursday, November 9, 2006 at 19:00:00 Eastern Standard Time

Not Valid After Sunday, November 9, 2031 at 19:00:00 Eastern Standard Time

Public Key Info

Algorithm RSA Encryption (1.2.840.113549.1.1.1)

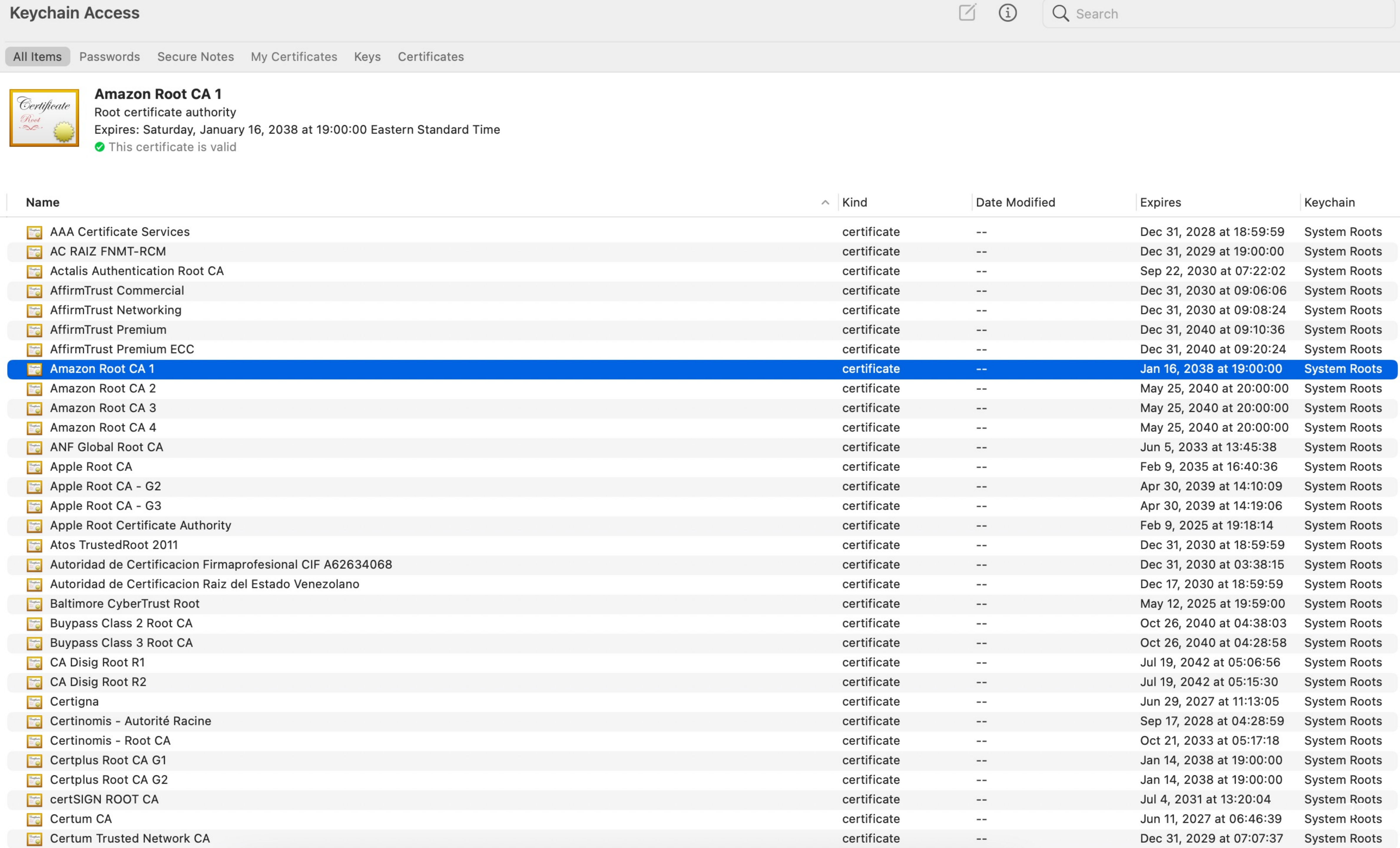
Parameters None

Public Key 256 bytes : AD 0E 15 CE E4 43 80 5C ...

Exponent 65537

Key Size 2,048 bits

Key Usage Verify



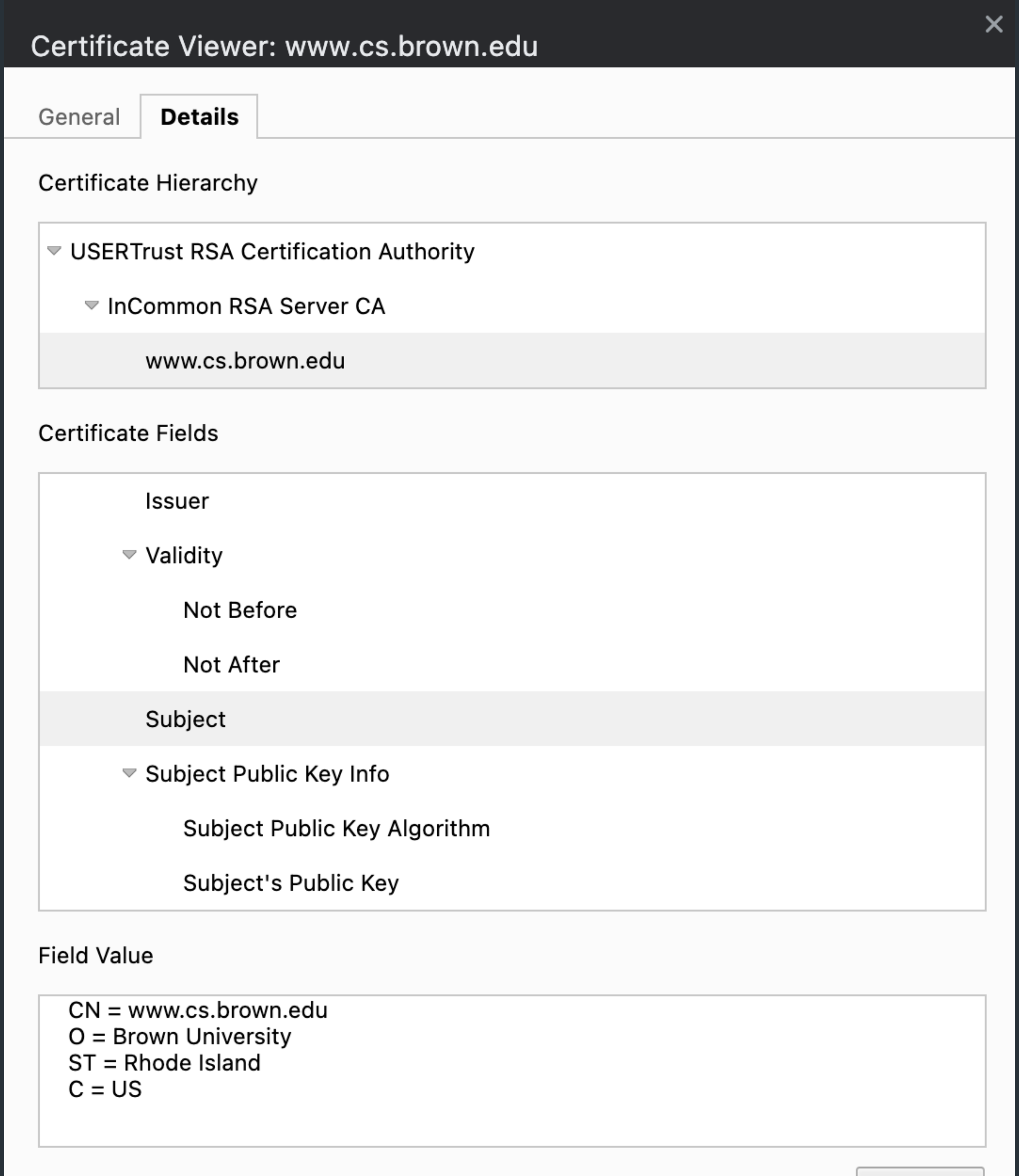
Amazon Root CA 1

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 Hierarchy

- USERTrust RSA Certification Authority
  - InCommon RSA Server CA
    - www.cs.brown.edu

Certificate Fields

- Issuer
- Validity
  - Not Before
  - Not After
- Subject
- Subject Public Key Info
  - Subject Public Key Algorithm
  - Subject's Public Key

Field Value

```
CN = www.cs.brown.edu
O = Brown University
ST = Rhode Island
C = US
```

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)

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

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

PKI hierarchy

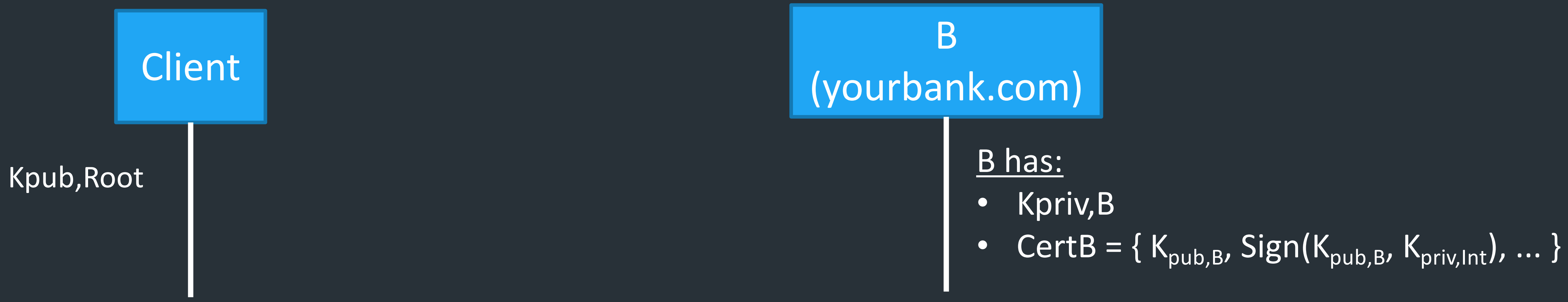
In reality, PKI creates a hierarchy of trust:

- 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}



Client

Kpub,Root

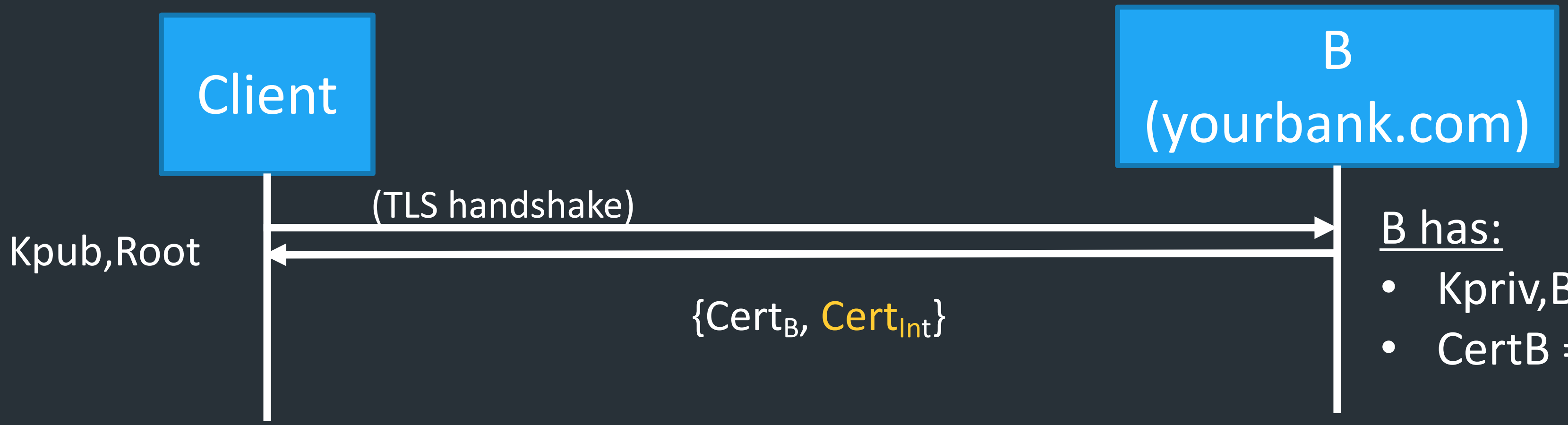
B (yourbank.com)

B has:

- Kpriv,B
- CertB = $\{ K_{pub,B}, Sign(K_{pub,B}, K_{priv,Int}), \ldots \}$

How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}



Client

B

(yourbank.com)

(TLS handshake)

Kpub,Root

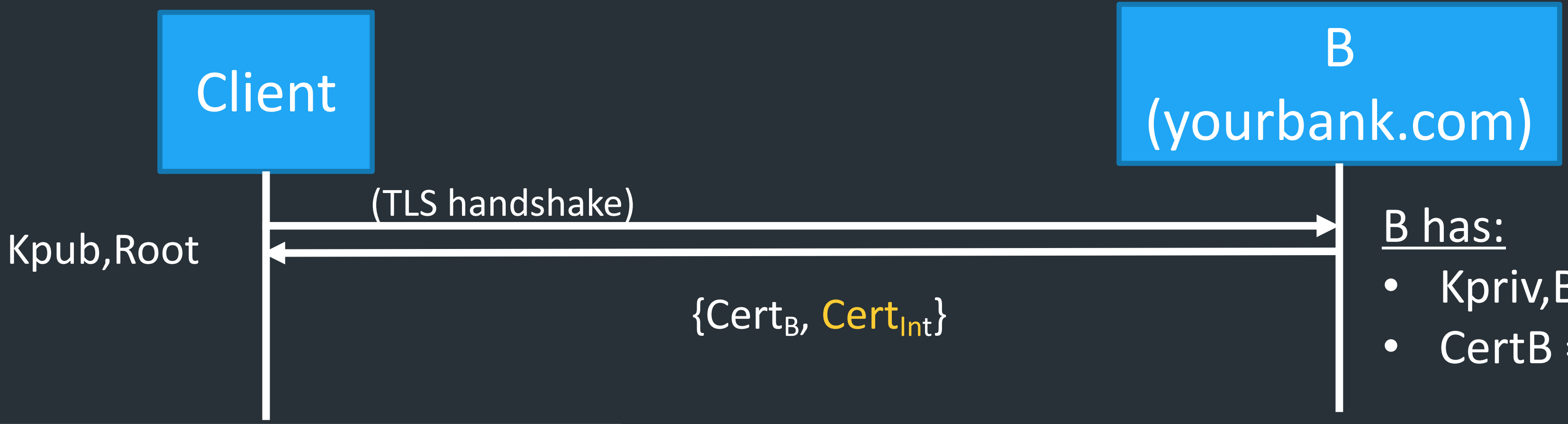
$\{\text{Cert}_{\text{B}}, \text{Cert}_{\text{Int}}\}$

B has:

- Kpriv,B
- CertB = $\{ K_{\text{pub,B}}, \text{Sign}(K_{\text{pub,B}}, K_{\text{priv,Int}}), \ldots \}$

How the hierarchy works

Ex. Server has certificate from Intermediate CA$_{\text{Int}}$



Client

B (yourbank.com)

(TLS handshake)

Kpub,Root

{Cert$_B$, Cert$_{\text{Int}}$}

B has:

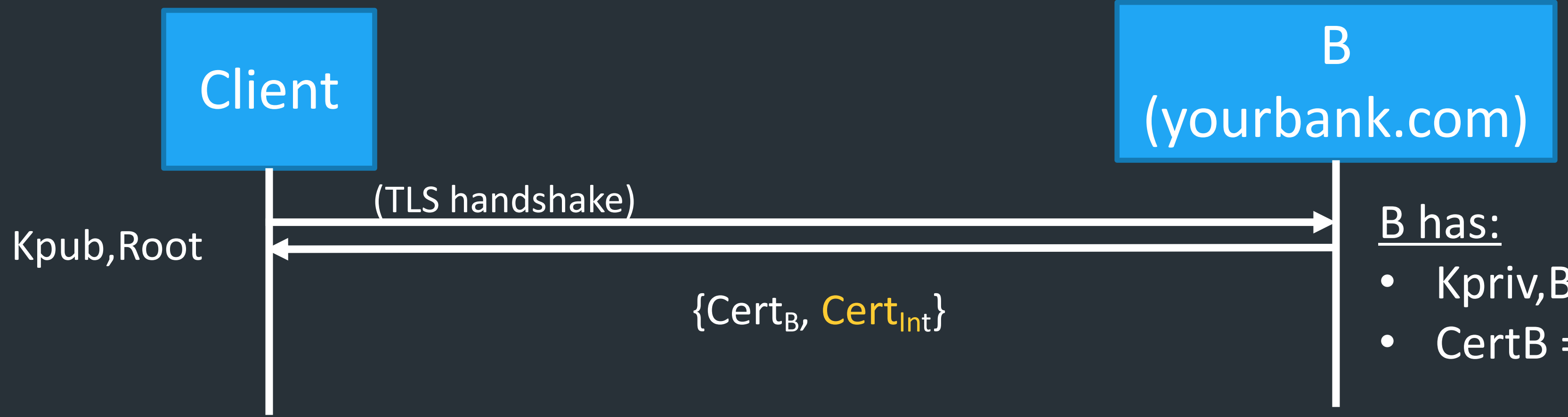
- Kpriv,B
- CertB = { $K_{pub,B}$, Sign($K_{pub,B}$, $K_{priv,Int}$), … }

Client's workflow:

- Checks metadata ✅
- Verify(Cert$_B$, $K_{pub,Int}$) ✅
- Verify(Cert$_{\text{Int}}$, $K_{pub,Root}$) ✅

How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}



B has:

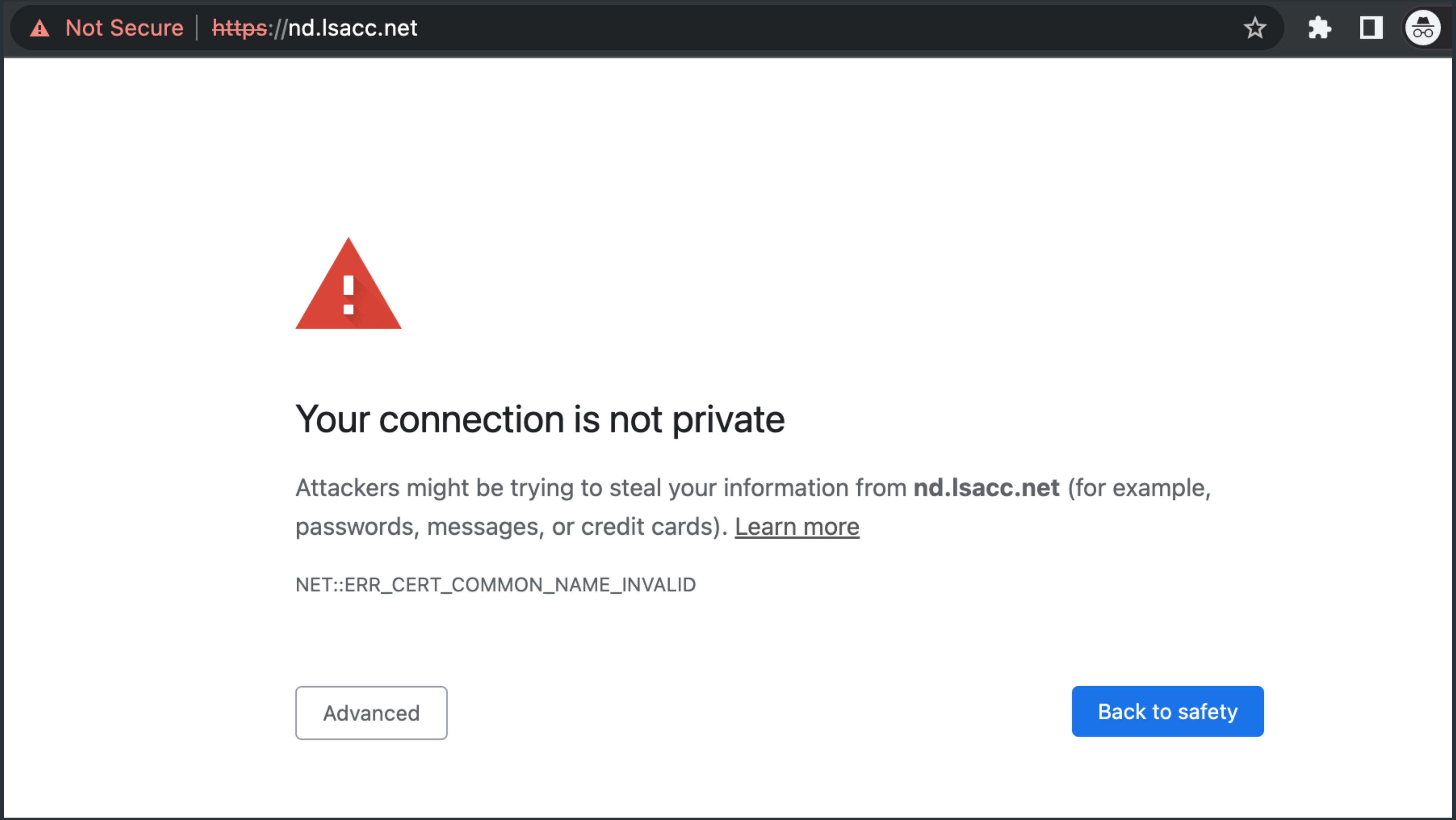
- Kpriv,B
- CertB = { $K_{pub,B}$, Sign($K_{pub,B}$, $K_{priv,Int}$), … }

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: 🔒



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

Most common TLS errors you might see

Most common TLS errors you might see

- Common name (eg. yourbank.com) invalid
- Certificate expired
- Bad chain of trust (can’t verify back to trusted root)

Most common TLS errors you might see

- Common name (eg. yourbank.com) invalid
- Certificate expired
- Bad chain of trust (can’t verify back to trusted root)

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

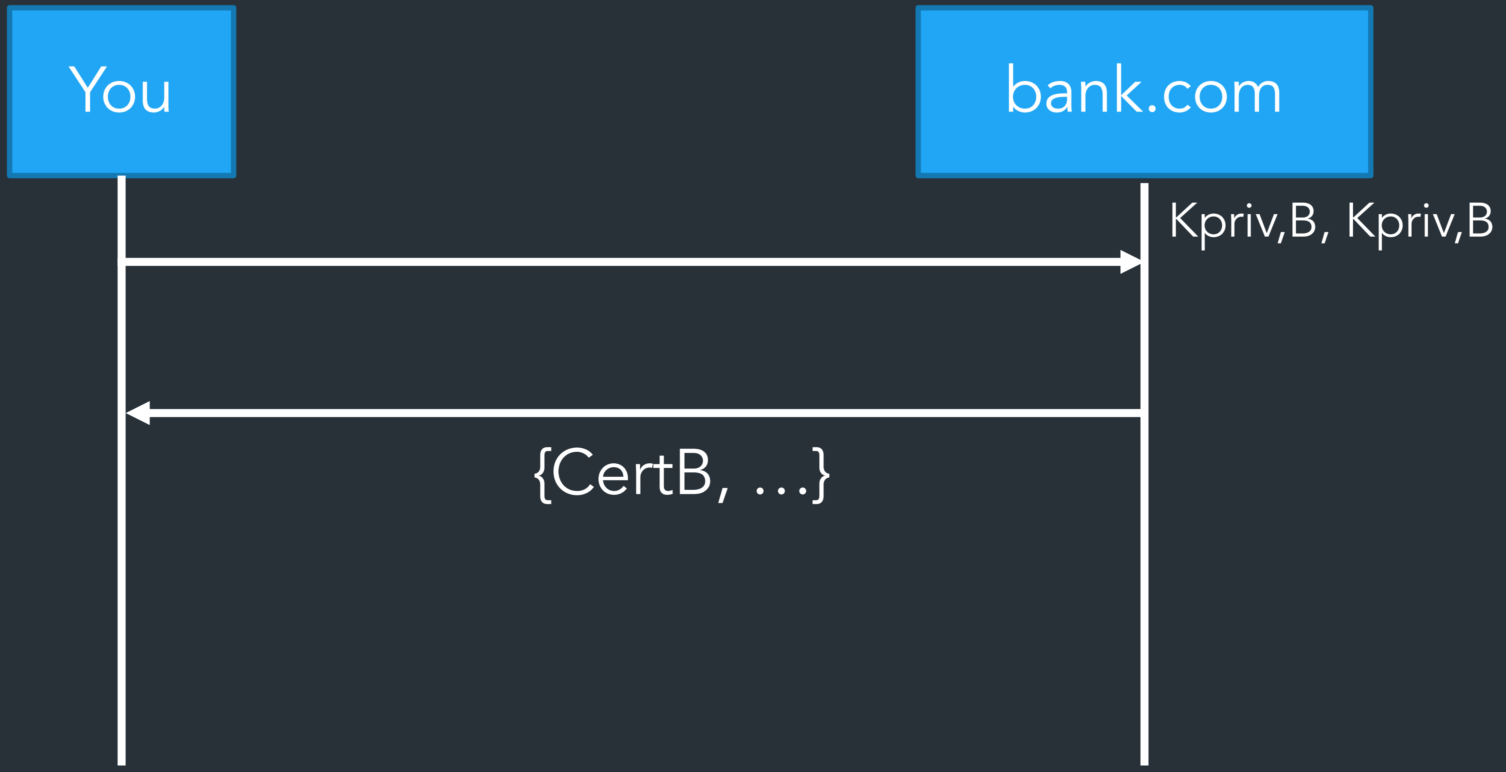
When is it okay to click “proceed”? What happens if you do?

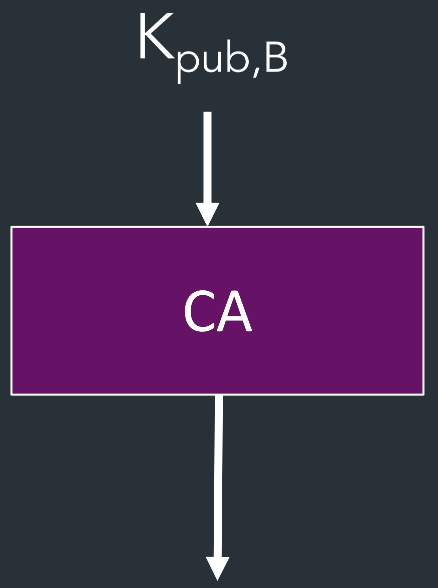
Most common TLS errors you might see

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

Warmup

What happens if attacker obtains Kpriv,B?
What about Kpriv,CA?





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

$$\text{Cert}_B = \{K_{pub,B}, \text{metadata}, s\}$$

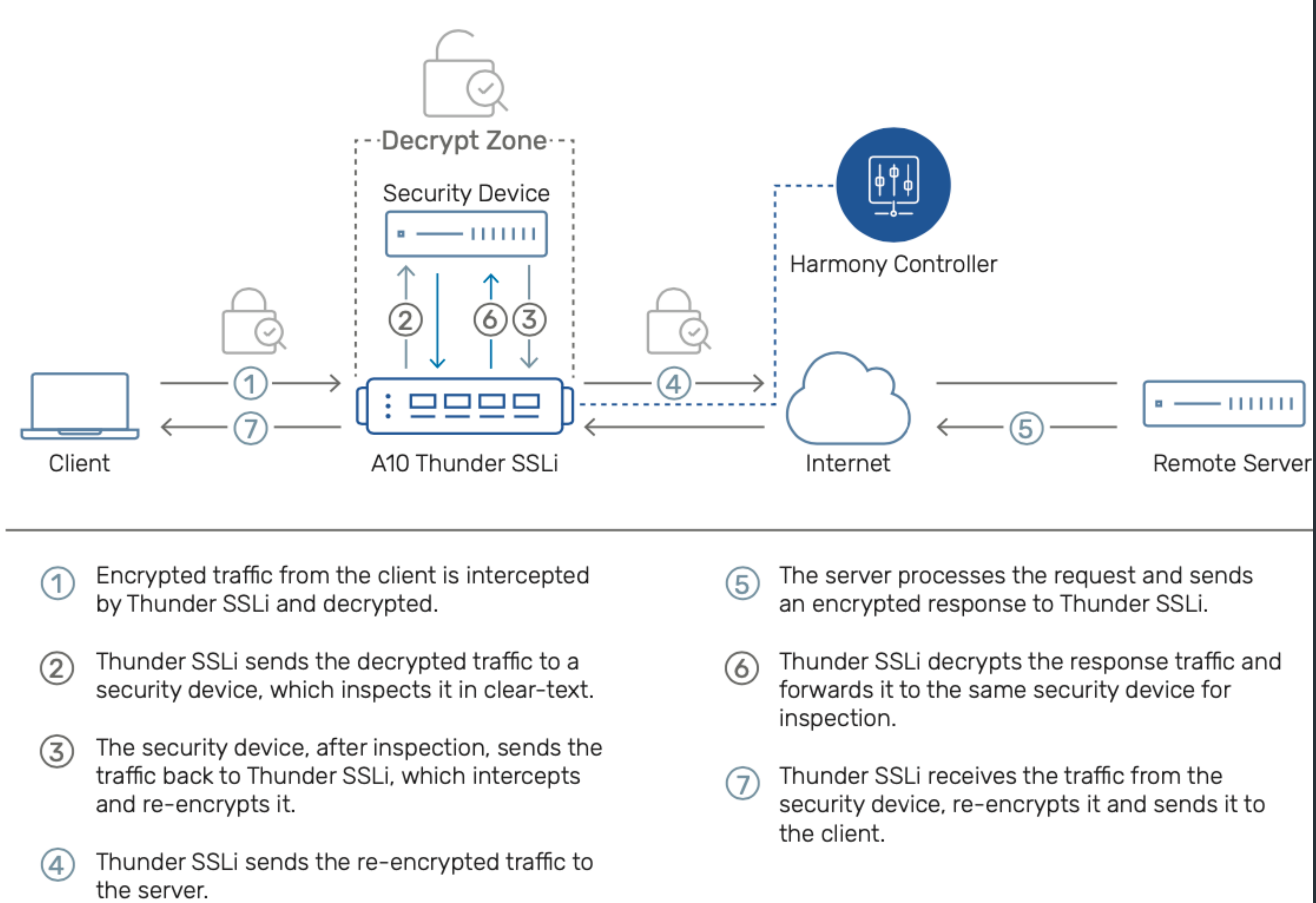
Rogue Certificates?

- 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

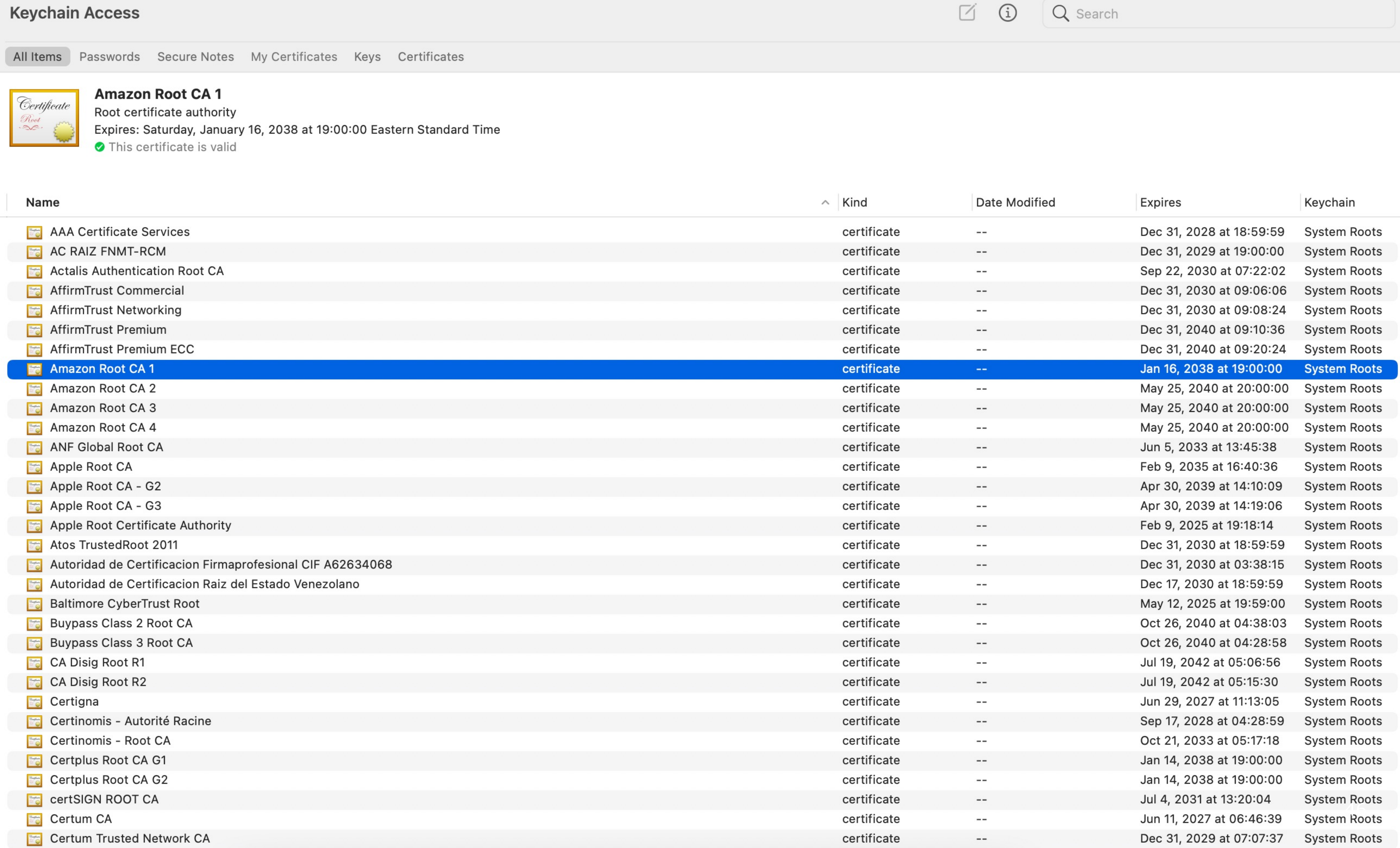
- In 2017, Google questioned the certificate issuance policies and practices of Symantec
- Google’s Chrome would start distrusting Symantec’s certificates unless certain remediation steps were taken
- See back and forth between Ryan Sleevi (Chromium team) and Symantec
- The matter was settled with DigiCert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

TLS “decryption”

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



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.



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

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

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

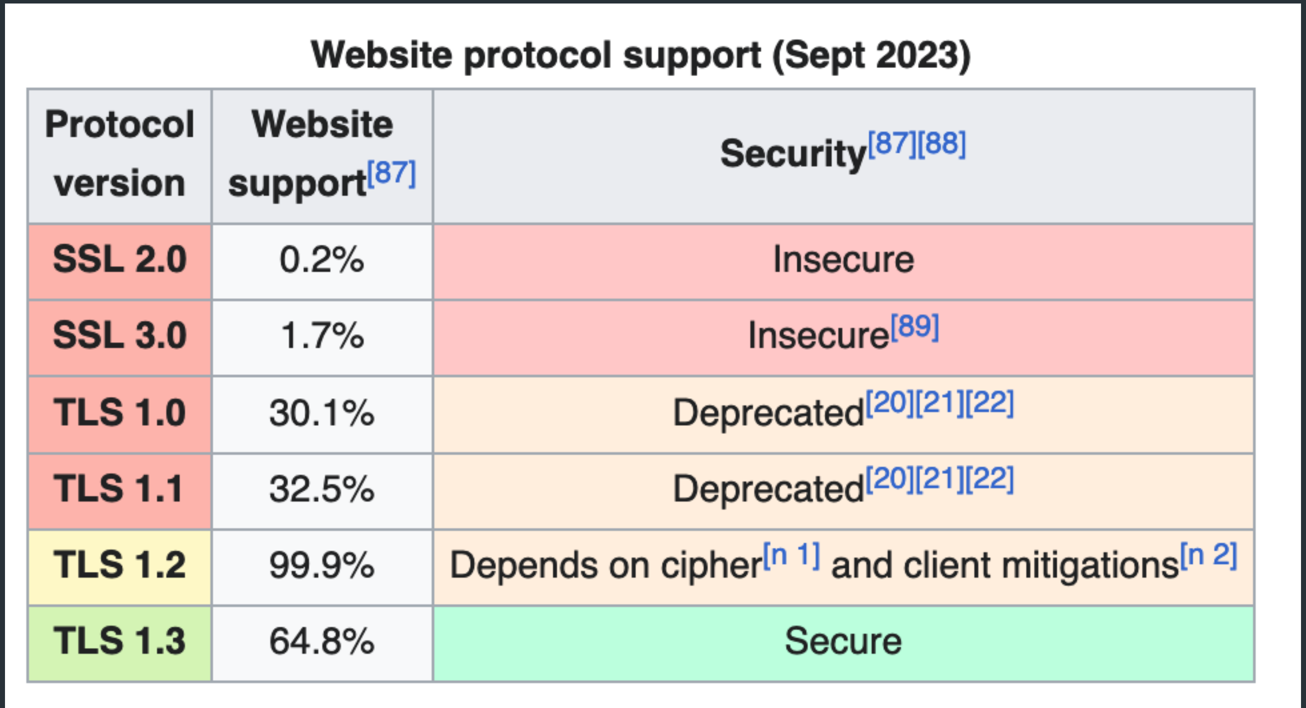
⇒ Once optional, now required by TLS 1.3 (2018)

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

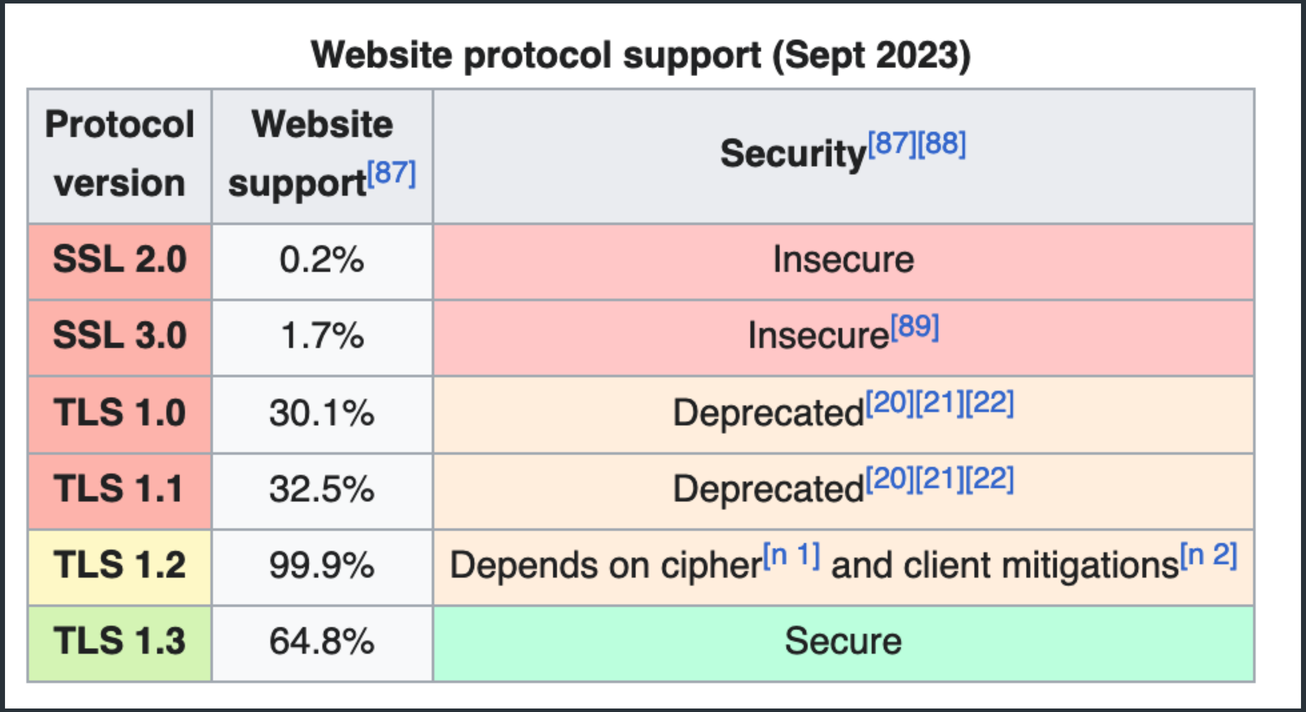
⇒ Once optional, now required by TLS 1.3 (2018)



Website protocol support (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

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



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

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?

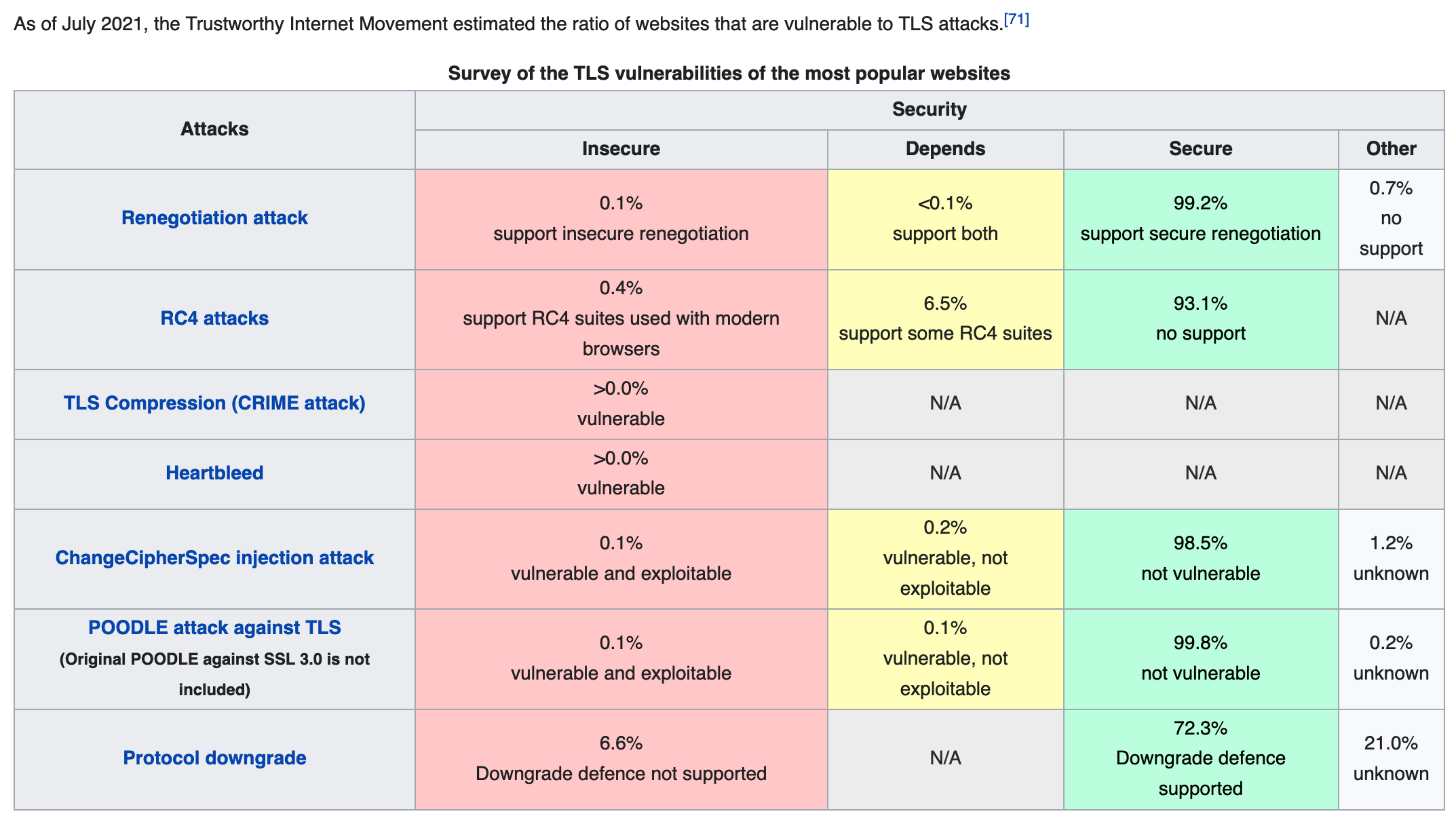
In general, implementing security protocols is hard to get right

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]

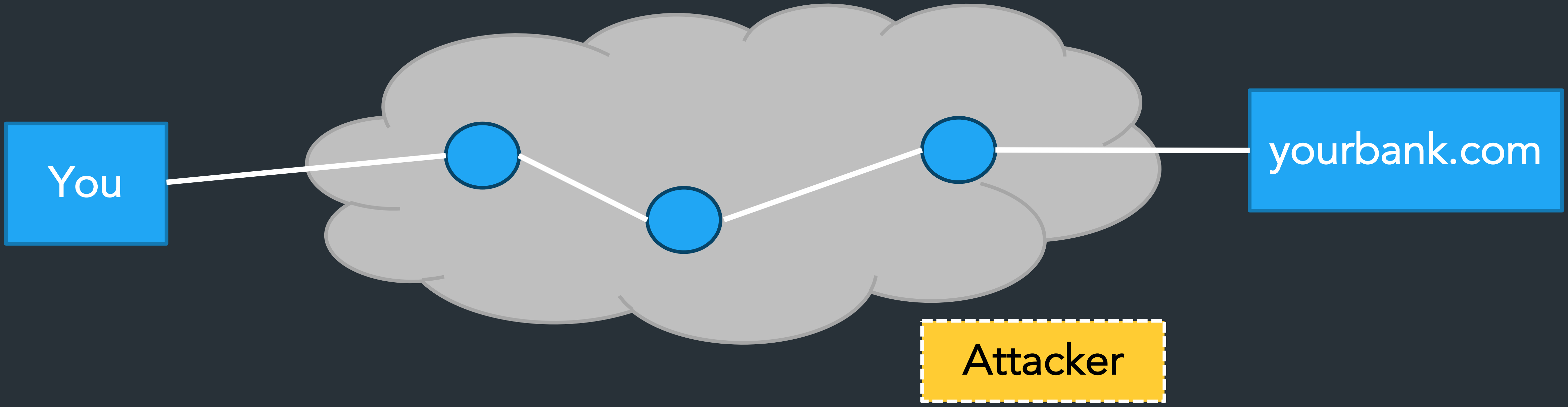
Survey of the TLS vulnerabilities of the most popular websites

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/

So, are we good?

If we use TLS, is it enough?


You
yourbank.com
Attacker

Overall, depends on your threat model…

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

- Even encrypted traffic leaks information!

Overall, depends on your threat model…

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

=> IPs can be traced to location (to varying levels of precision)

=> Your browser may leak info (cookies, mouse usage, etc.)

- Even encrypted traffic leaks information!

=> Name of server: DNS, Server Name Indicator (SNI)

=> Traffic patterns (timing of packets, protocols, …)

Securing the transport layer not enough => info leaks based on other layers

Why?

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

Why?

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

Room 641A: 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

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)

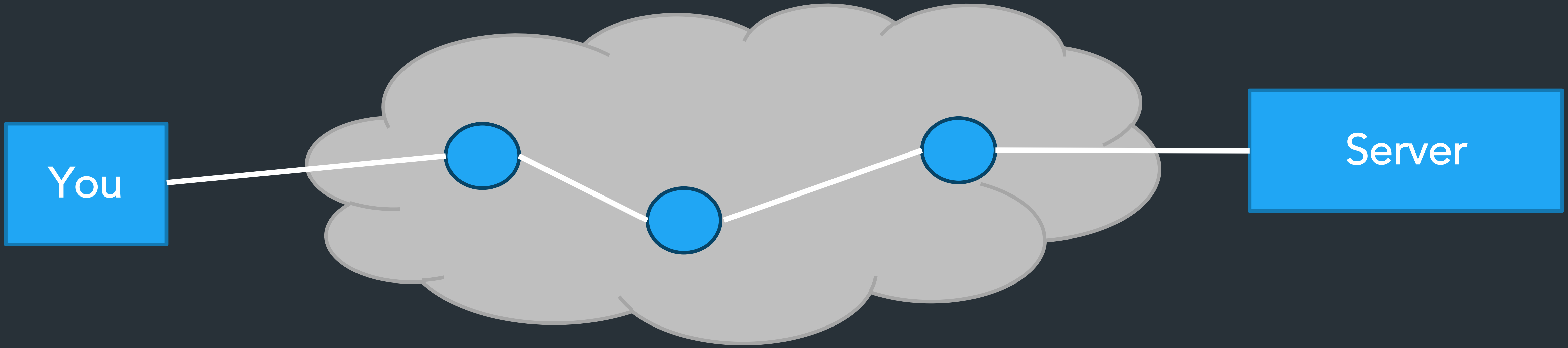
Secure tunnel for arbitrary network traffic (any IP packets)

Use for

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

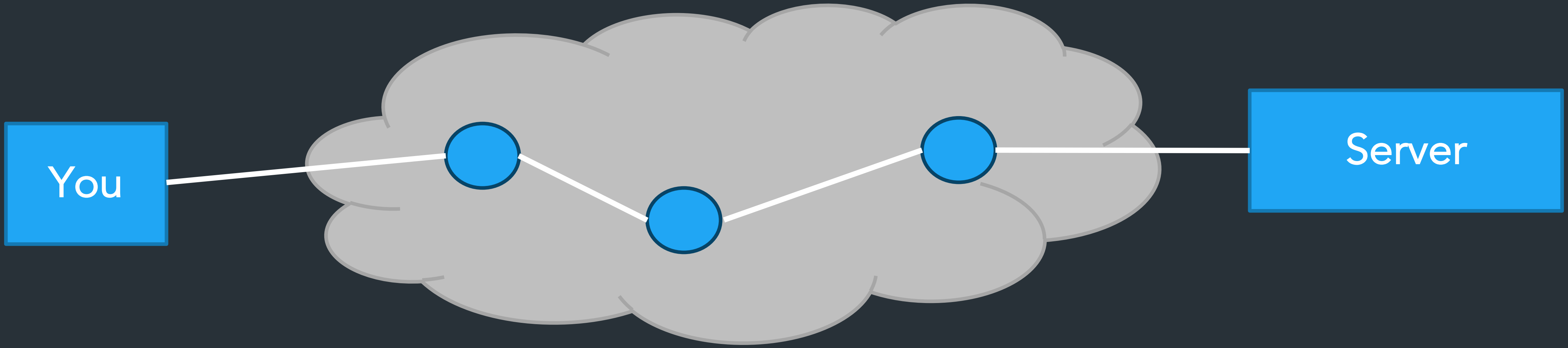
=> Secure proxy for your traffic: traffic appears to originate from VPN server

Problems?

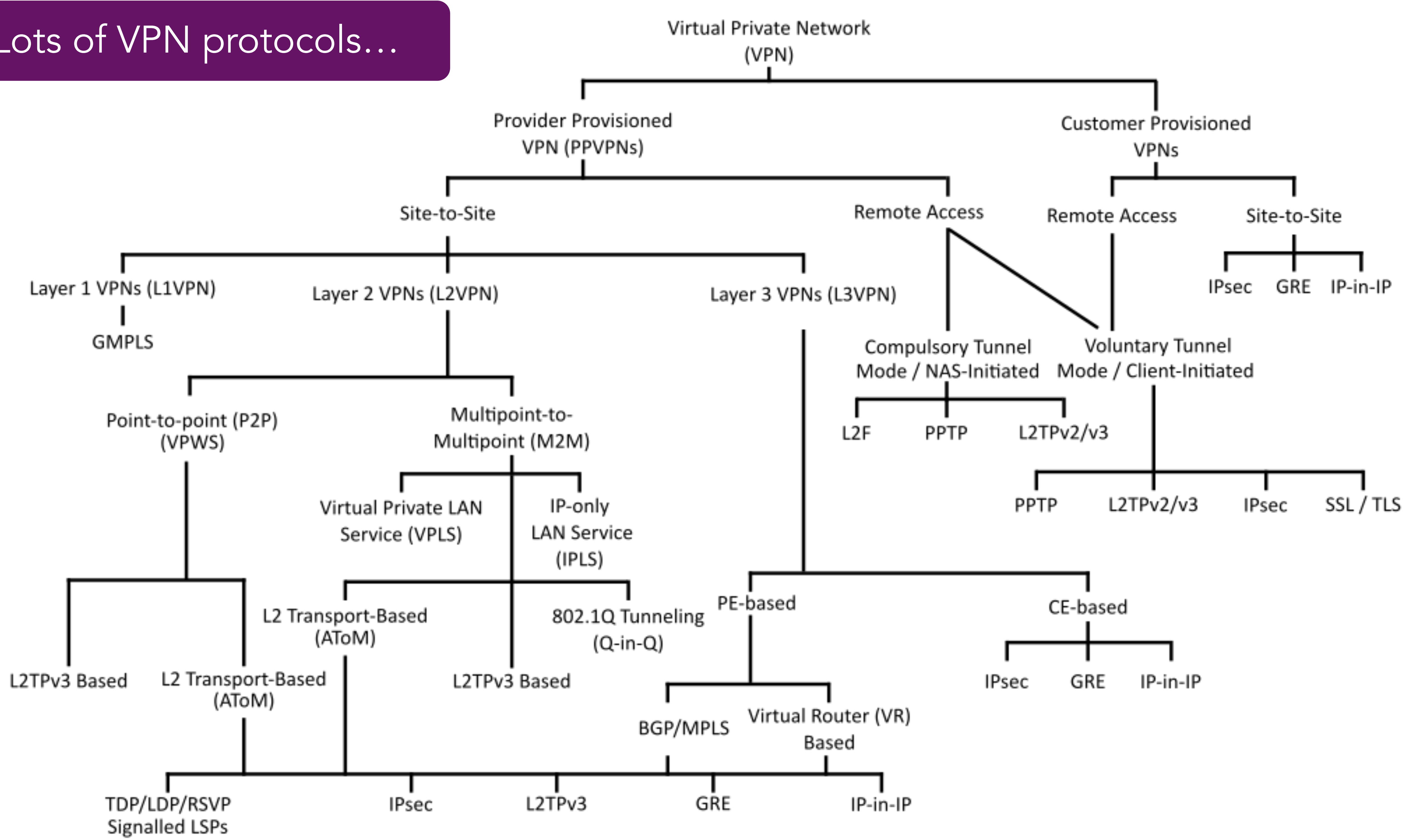


VPN: secure tunnel for network traffic

=> Connect a host to a private network



Lots of VPN protocols…


Virtual Private Network (VPN)
Provider Provisioned VPN (PPVPNs)
Customer Provisioned VPNs
Site-to-Site
Remote Access
Remote Access
Site-to-Site
IPsec
GRE
IP-in-IP
Layer 1 VPNs (L1VPN)
Layer 2 VPNs (L2VPN)
Layer 3 VPNs (L3VPN)
GMPLS
Compulsory Tunnel Mode / NAS-Initiated
Voluntary Tunnel Mode / Client-Initiated
L2F
PPTP
L2TPv2/v3
PPTP
L2TPv2/v3
IPsec
SSL / TLS
Point-to-point (P2P) (VPWS)
Multipoint-to-Multipoint (M2M)
Virtual Private LAN Service (VPLS)
IP-only LAN Service (IPLS)
L2 Transport-Based (AToM)
802.1Q Tunneling (Q-in-Q)
PE-based
CE-based
IPsec
GRE
IP-in-IP
L2TPv3 Based
L2 Transport-Based (AToM)
L2TPv3 Based
BGP/MPLS
Virtual Router (VR) Based
TDP/LDP/RSVP Signalled LSPs
IPsec
L2TPv3
GRE
IP-in-IP

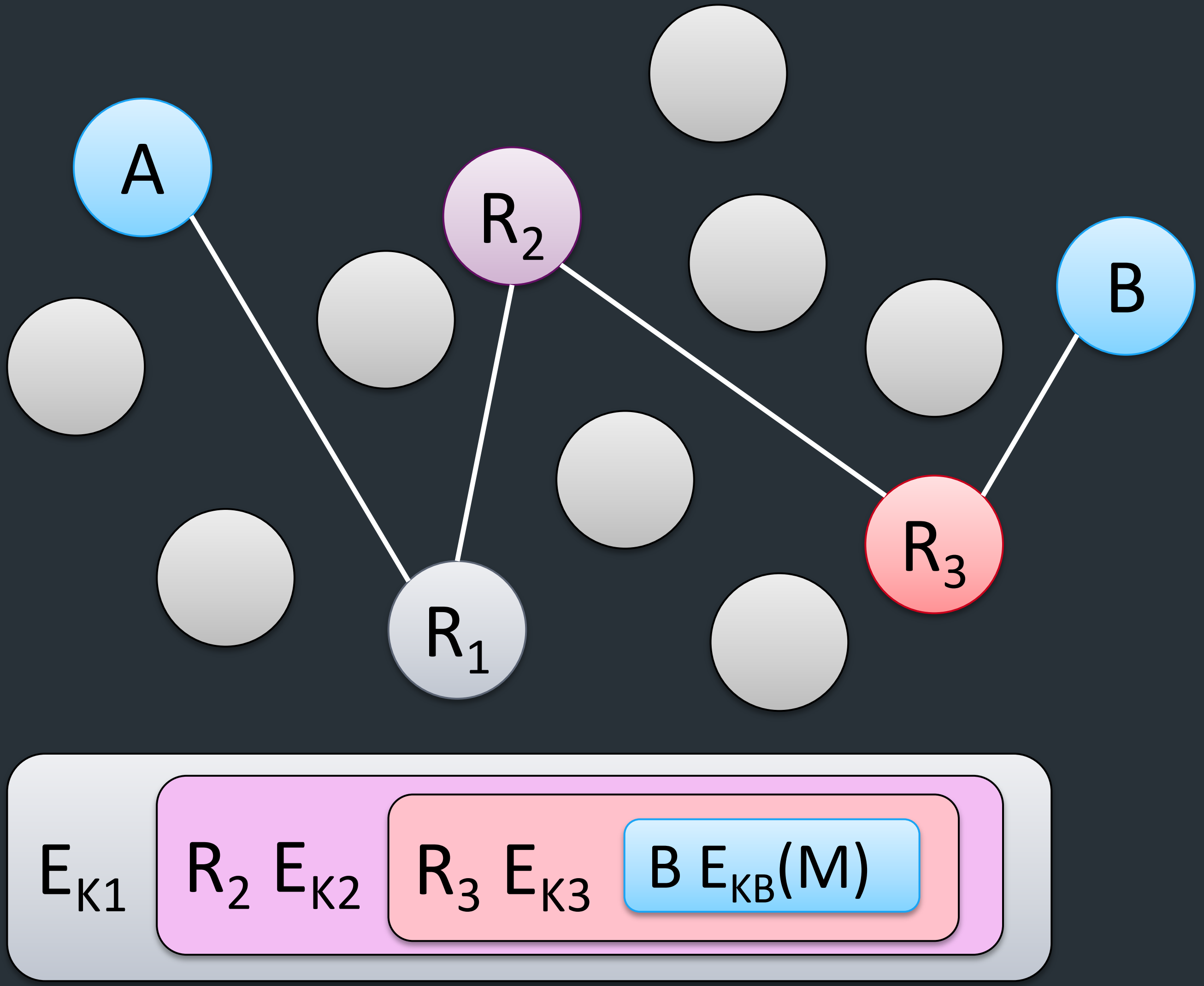
Can we do better?

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

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



E_{K1} R_2 E_{K2} R_3 E_{K3} B $E_{KB}(M)$