CSCI-1680
TLS

Nick DeMarinis

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

Administrivia

- If you haven't scheduled a TCP grading meeting, please do so

- HW4 (short): Out today, due next Friday

- Final project: short proposal due Friday (no late days!)
  - Will send team confirmation/repo link today

This is not a security class (as much as I would like it to be…)

- This isn’t intended to be a lecture on all crypto
- I want you to appreciate the important principles, understand what’s important for TLS (and other protocols like it)

Want to know more?

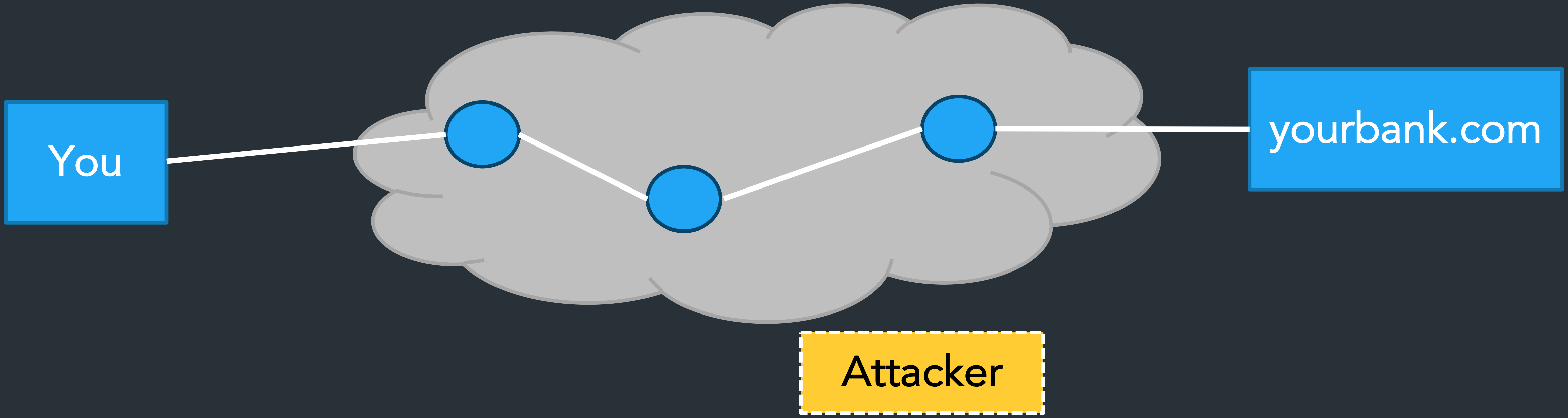
- CS1660 (Spring): Intro to Computer Systems Security
- CS1515 (Spring): Applied cryptography
- CS1510 (Fall): Intro to Cryptography and Computer Security

Internet's Design: Insecure

- Designed for simplicity in a naïve era
- Lots of insecure systems that can be compromised

- No central administration => hard to diagnose, coordinate fixes

What can go wrong?



(some) Key security properties

- Confidentiality

- Authentication

- Integrity

(some) Key security properties

- Confidentiality: prevent adversary from reading the data
  => Protect against *eavesdropping*, *sniffing*
- Authentication: verifying the identity of a message or actor
  => Protect against *spoofing*, *impersonation*
- Integrity: make sure messages arrive in original form
  => Protect against *tampering*

(some) Key security properties

- Confidentiality: prevent adversary from reading the data
  => Protect against *eavesdropping, sniffing*

- Authentication: verifying the identity of a message or actor
  => Protect against *spoofing*, *impersonation*

- Integrity: make sure messages arrive in original form
  => Protect against *tampering*

There are more security properties, but we'll stick to these => Focus of TLS

Other important security properties

- Availability: Will the network deliver data?
  - Protect against infrastructure compromise, DDoS
- Provenance: Who is responsible for this data?
  - Prevent forging responses, denying responsibility; prove who created the data

- Authorization: is actor *allowed* to do this action?
- Appropriate use: is action *consistent with policy*? (spam, copyright, …)
- Anonymity: can someone tell what packets *I* am sending?

TLS: Transport layer security

TLS 1.0 (1999) => TLS 1.3 (2018)

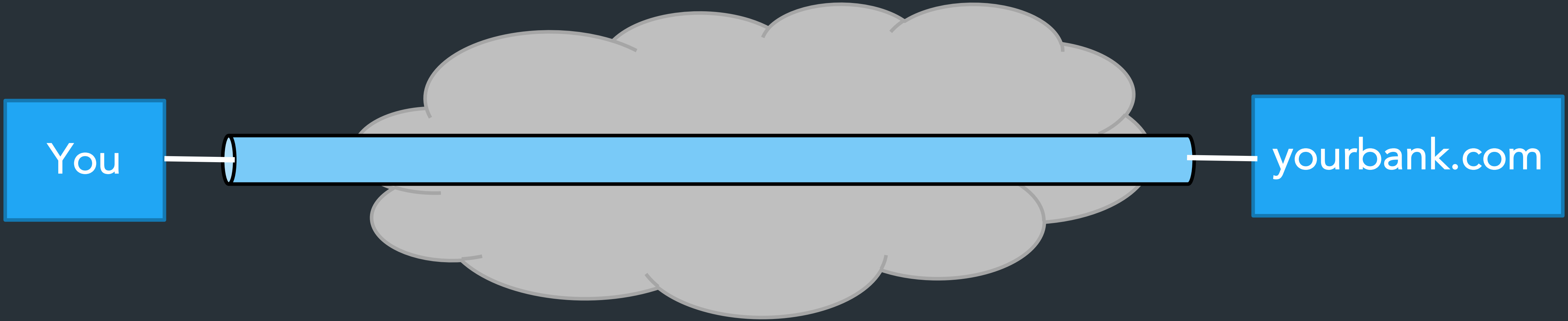
Bidirectional pipe between two parties providing:

- Confidentiality
- Integrity
- Authentication

TLS: Transport layer security

Bidirectional pipe between two parties providing:

- Confidentiality
- Integrity
- Authentication

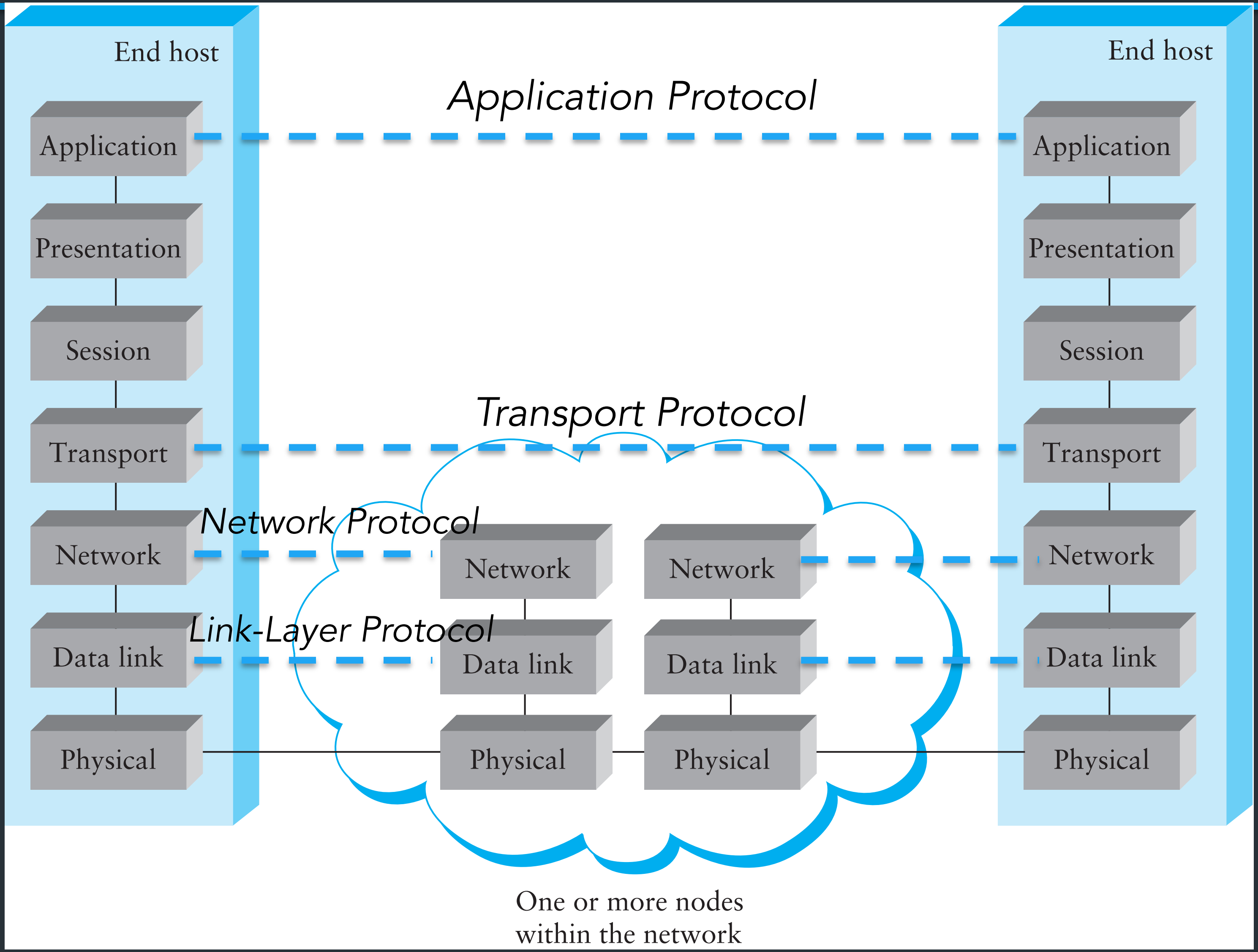


Are these all the security properties we might want? No!

Where does TLS go?

Application	Service: user-facing application. Application-defined messages
Transport	How to support multiple applications?
Network	Moving data between hosts (nodes)
Link	Move data across individual *links*
Physical	Service: move bits to other node across link

Throwback: The OSI model



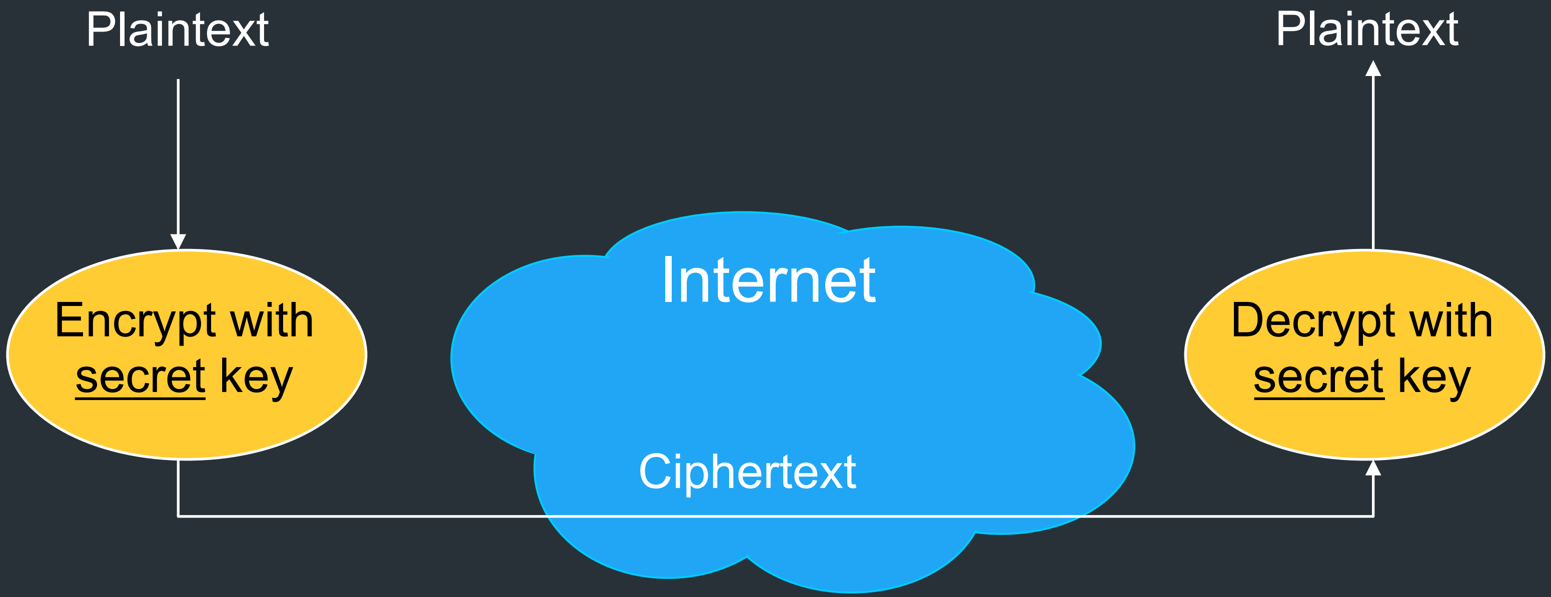
Fundamental crypto properties we need

Symmetric cryptography

- A, B share secret key k
- Examples: AES, Serpent, Whirlpool, DES (old, insecure), …
- Provides: confidentiality (encrypt/decrypt), integrity (MAC)

Symmetric crypto: strong, fast, but parties <u>need to have shared key k</u>
=> Key distribution is hard, why?

Confidentiality: Symmetric encryption



Confidentiality: Asymmetric encryption

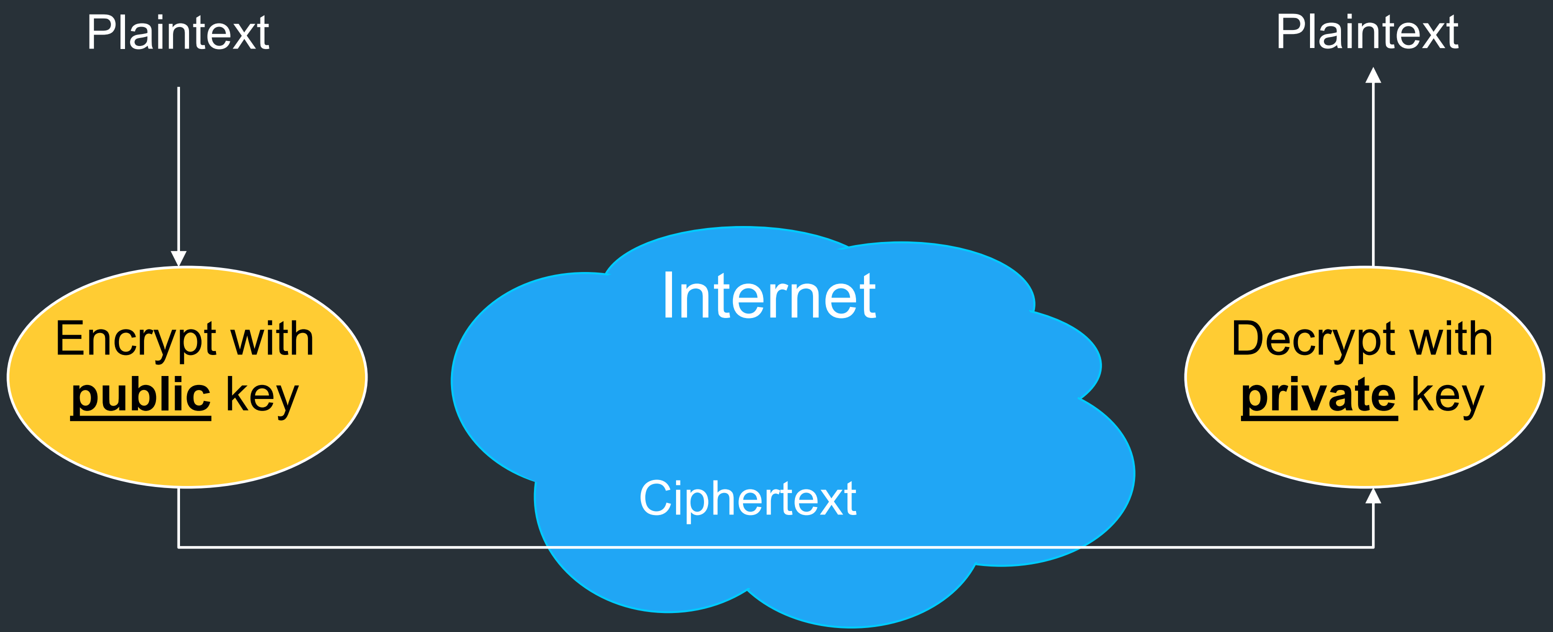
Everyone has two keys: k_pub, k_priv

Confidentiality: Asymmetric encryption

- Everyone has two keys: k_pub, k_priv
  - k_pub: Public key, widely-known
  - k_priv: Private key, kept secret
- Used for: authentication, signing (and confidentiality, integrity)

Public Key / Asymmetric Encryption

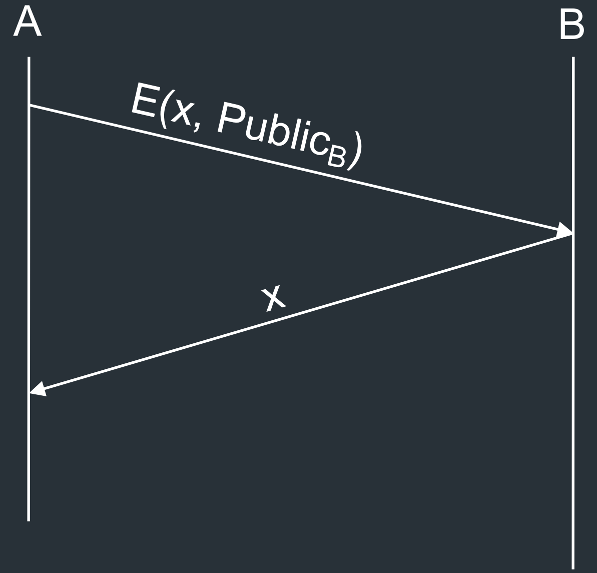
- Sender uses receiver's public key
  - Advertised to everyone
- Receiver uses complementary private key
  - Must be kept secret



What can we do with this?

Public Key Authentication

- Each side need only to know the other side's public key
  - No secret key need be shared
- **A** encrypts a nonce (random number) **x** using B's public key
- **B** proves it can recover **x**
- **A** can authenticate itself to **B** in the same way



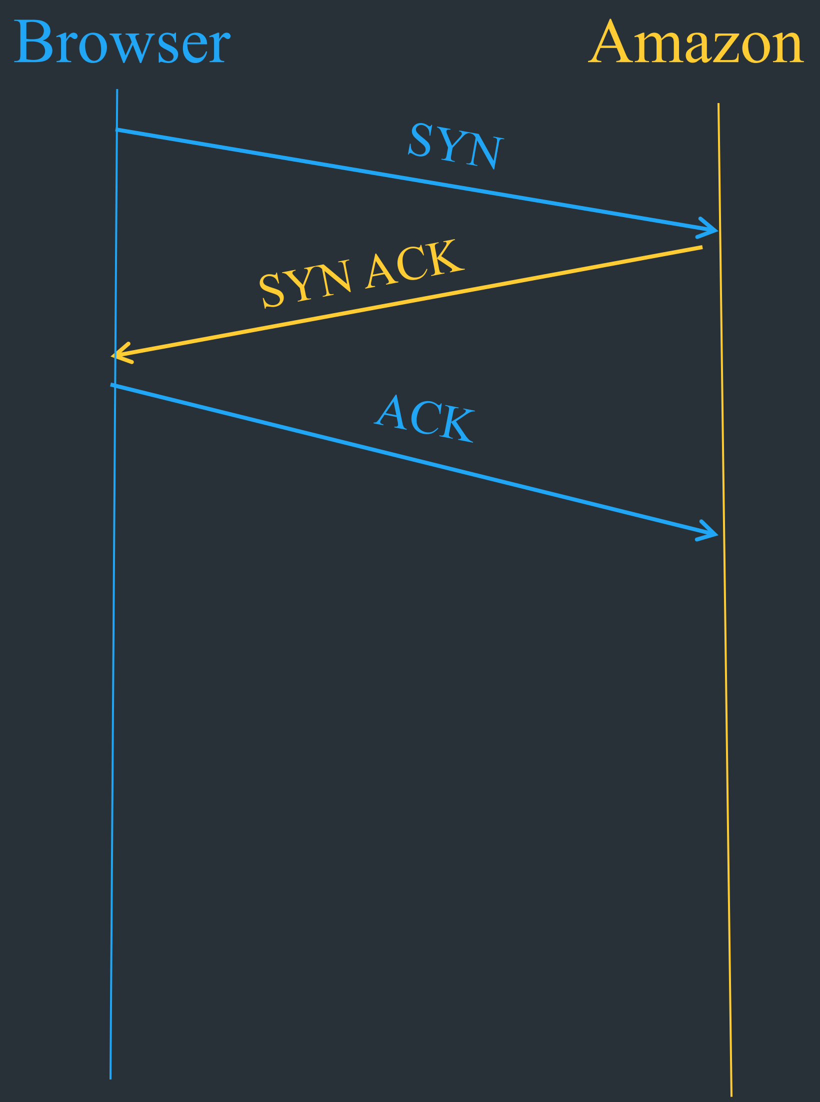
How it works in TLS

- Type in your browser: https://www.amazon.com
- https = "Use HTTP over TLS"
  - TLS = Transport Layer Security
  - SSL = Secure Socket Layer (older version)
  - RFC 4346, and many others

Goal: provide security layer (authentication, encryption) on top of transport layer
=> Fairly transparent to the app (once set up)

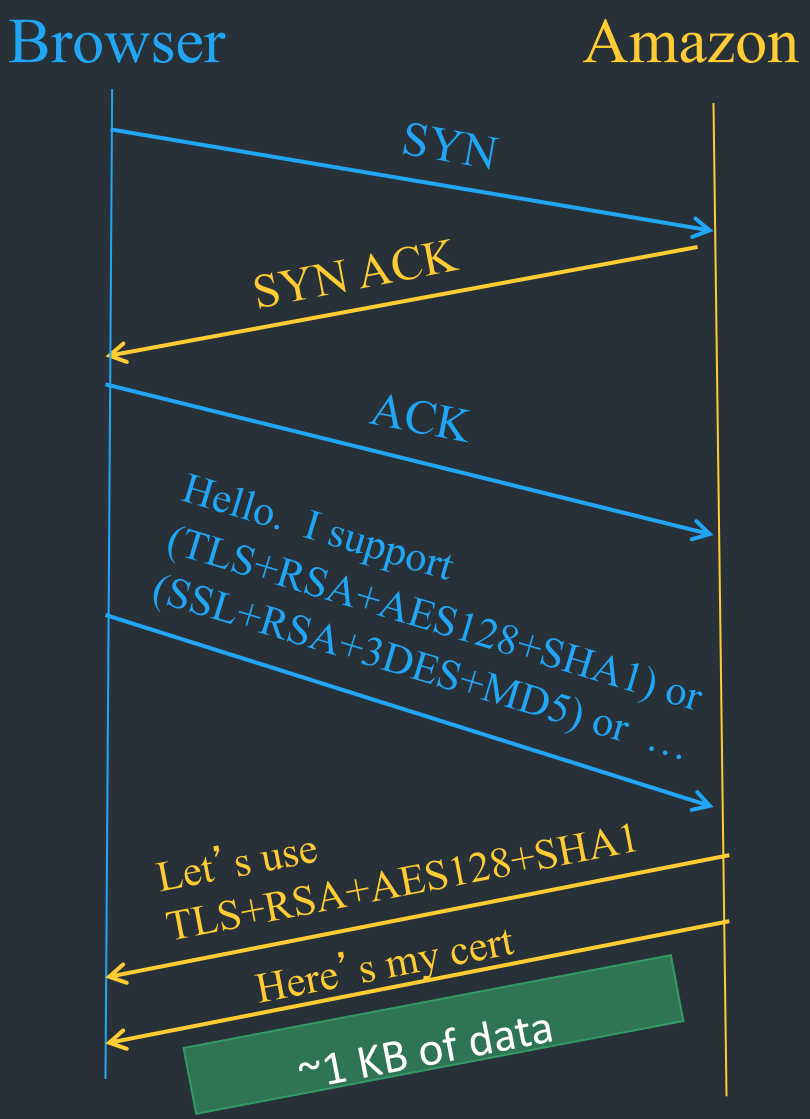
TLS: setup

- First: TCP handshake



TLS: setup

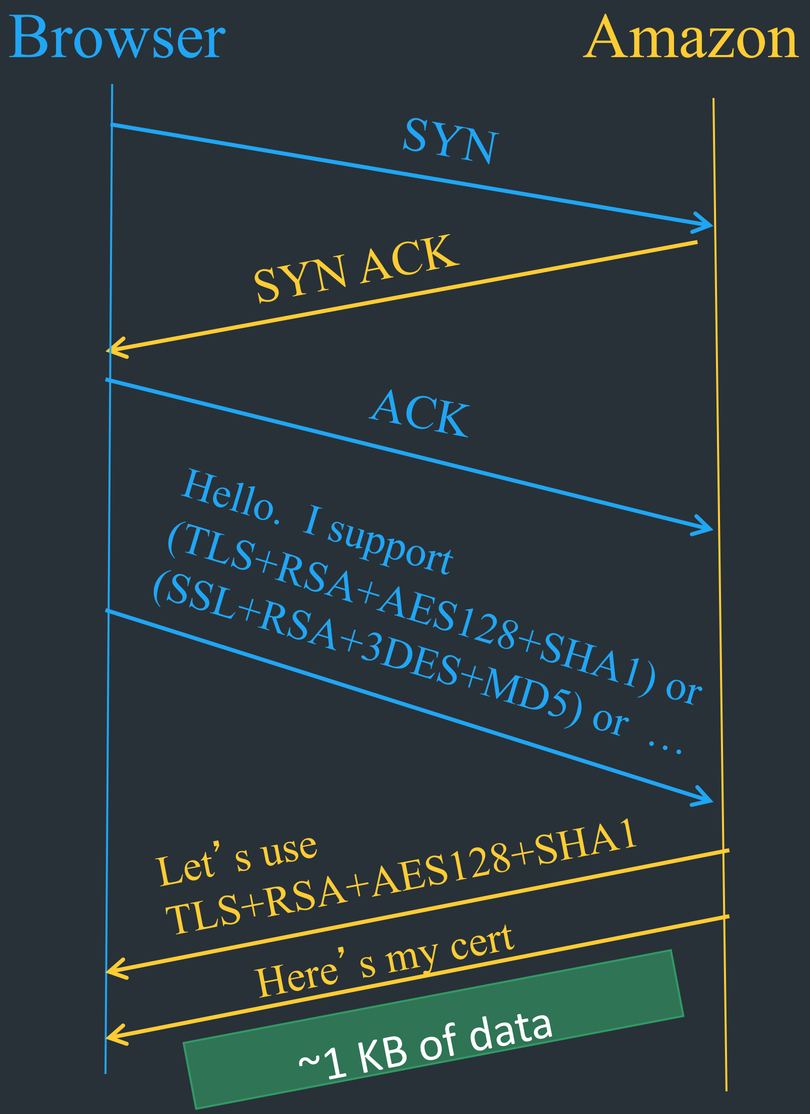
- First: TCP handshake
- Client sends over list of crypto protocols it supports
- Server picks crypto protocols to use for this session

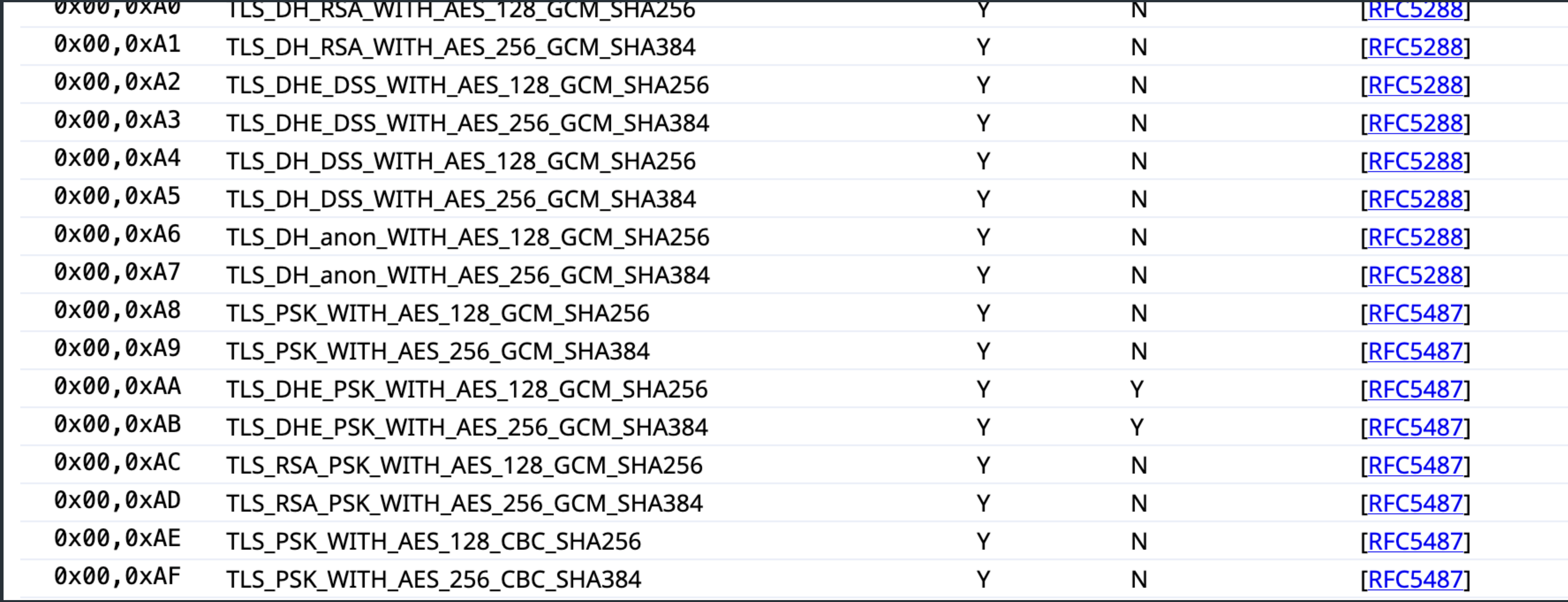

Browser
Amazon
SYN
SYN ACK
ACK
Hello. I support (TLS+RSA+AES128+SHA1) or (SSL+RSA+3DES+MD5) or …
Let's use TLS+RSA+AES128+SHA1
Here's my cert
~1 KB of data

TLS: setup

- First: TCP handshake
- Client sends over list of crypto protocols it supports
- Server picks crypto protocols to use for this session

- Use this to do two things:
  - Create shared session key
  - Verify server's identity


Browser
Amazon
SYN
SYN ACK
ACK
Hello. I support (TLS+RSA+AES128+SHA1) or (SSL+RSA+3DES+MD5) or …
Let' s use TLS+RSA+AES128+SHA1
Here' s my cert
~1 KB of data



0x00,0xA0	TLS_DH_RSA_WITH_AES_128_GCM_SHA256	Y	N	[RFC5288]
0x00,0xA1	TLS_DH_RSA_WITH_AES_256_GCM_SHA384	Y	N	[RFC5288]
0x00,0xA2	TLS_DHE_DSS_WITH_AES_128_GCM_SHA256	Y	N	[RFC5288]
0x00,0xA3	TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	Y	N	[RFC5288]
0x00,0xA4	TLS_DH_DSS_WITH_AES_128_GCM_SHA256	Y	N	[RFC5288]
0x00,0xA5	TLS_DH_DSS_WITH_AES_256_GCM_SHA384	Y	N	[RFC5288]
0x00,0xA6	TLS_DH_anon_WITH_AES_128_GCM_SHA256	Y	N	[RFC5288]
0x00,0xA7	TLS_DH_anon_WITH_AES_256_GCM_SHA384	Y	N	[RFC5288]
0x00,0xA8	TLS_PSK_WITH_AES_128_GCM_SHA256	Y	N	[RFC5487]
0x00,0xA9	TLS_PSK_WITH_AES_256_GCM_SHA384	Y	N	[RFC5487]
0x00,0xAA	TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	Y	Y	[RFC5487]
0x00,0xAB	TLS_DHE_PSK_WITH_AES_256_GCM_SHA384	Y	Y	[RFC5487]
0x00,0xAC	TLS_RSA_PSK_WITH_AES_128_GCM_SHA256	Y	N	[RFC5487]
0x00,0xAD	TLS_RSA_PSK_WITH_AES_256_GCM_SHA384	Y	N	[RFC5487]
0x00,0xAE	TLS_PSK_WITH_AES_128_CBC_SHA256	Y	N	[RFC5487]
0x00,0xAF	TLS_PSK_WITH_AES_256_CBC_SHA384	Y	N	[RFC5487]

TLS + Authentication

TLS Goals

Authentication: verifying that the entity on the other end of the connection is who they claim to be

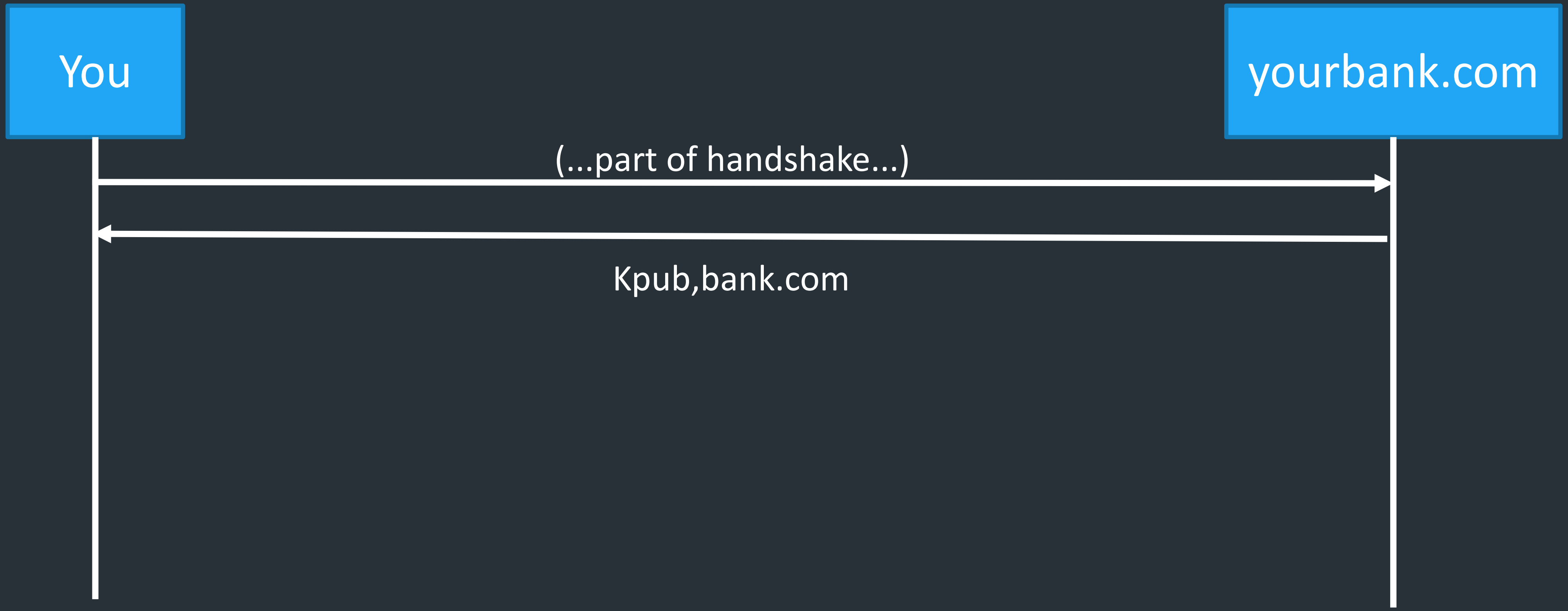
TLS Goals

Authentication: verifying that the entity on the other end of the connection is who they claim to be

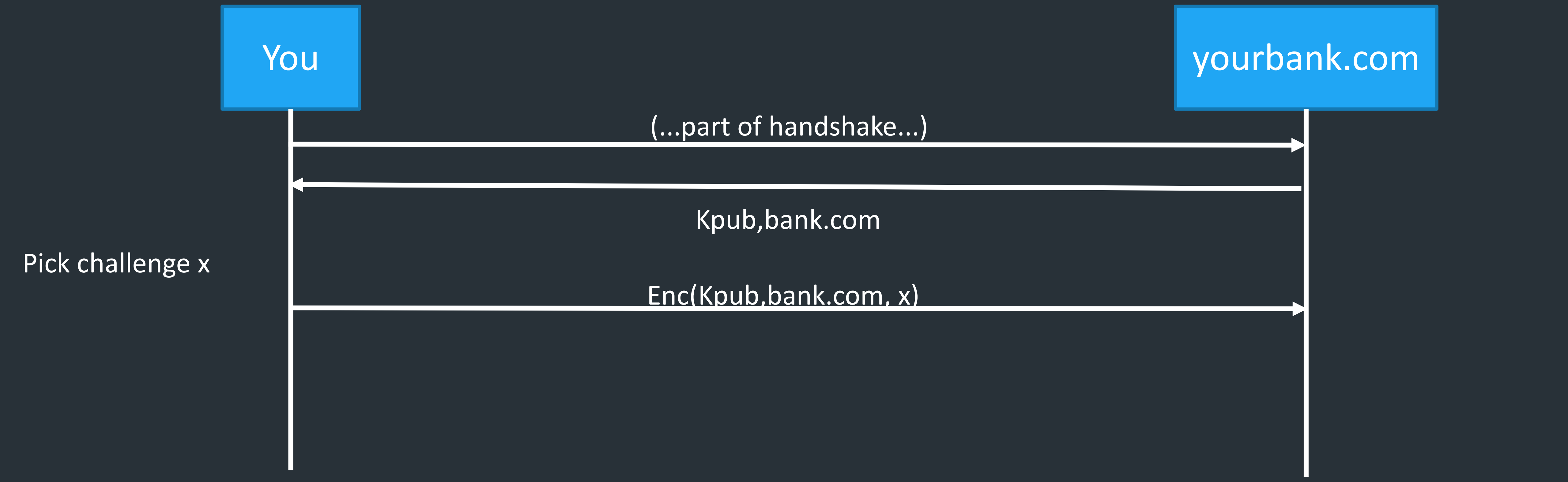
- Technical aspects: crypto
- Social aspects
  - How to distribute keys to entities
  - What to do when things go wrong

TLS: relies on Public Key Infrastructure (PKI) via certificates

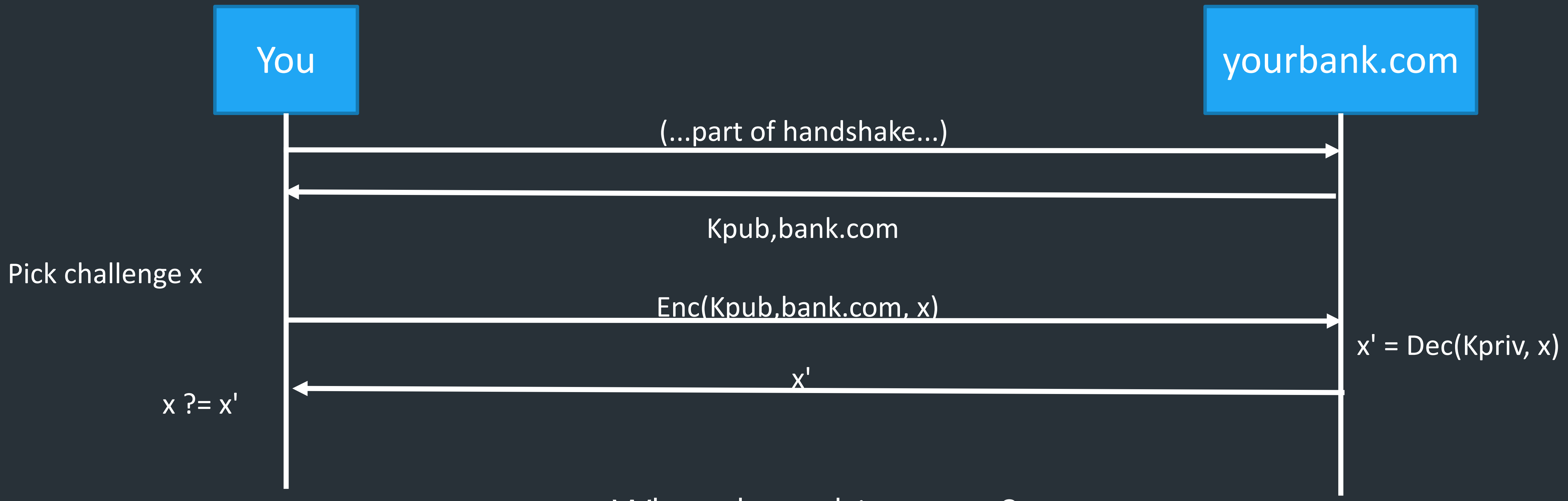
The Challenge



The Challenge



The Challenge



What does this prove?

Authentication challenges

- Challenge proves that the server at yourbank.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 belong to the server belongs to your bank, the real-life bank with your money

"But I'm visiting yourbank.com!"

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: 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)

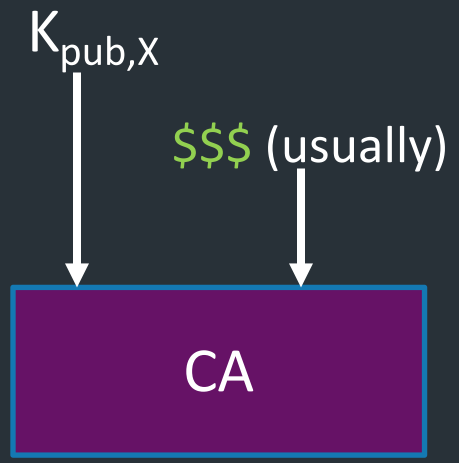
- Everyone knows public key for some root CAs
  - Pre-installed into browser/OS

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, then signs X's public key

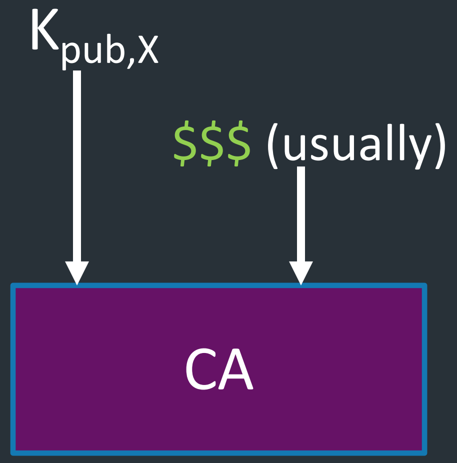


PKI: The main idea

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, then signs X's public key
  - Generates certificate

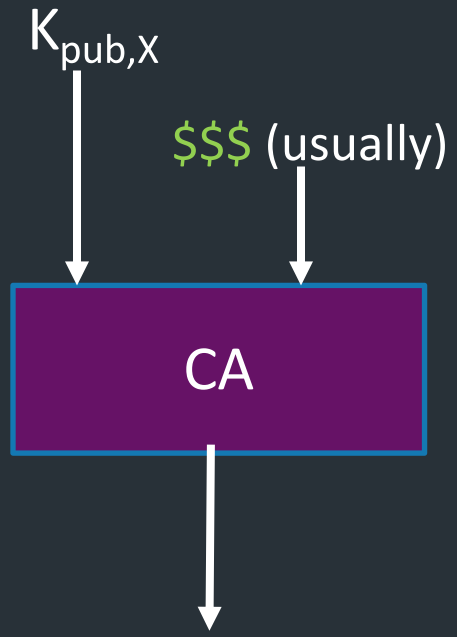


PKI: The main idea

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, then signs X's public key
  - Generates certificate



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

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

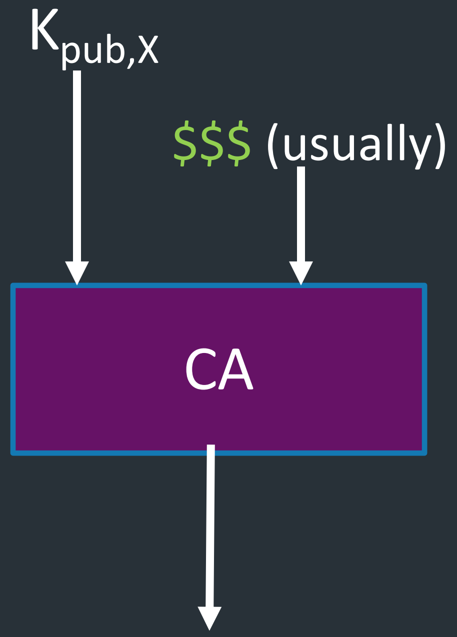
PKI: The main idea

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, then 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}

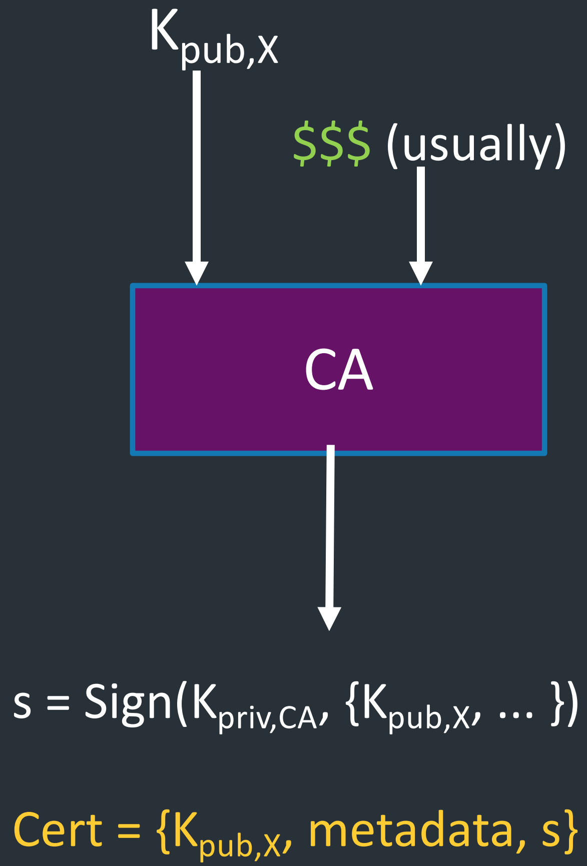
PKI: The main idea

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, then signs X's public key
  - Generates certificate
- Client can verify $K_{pub,X}$ from CA's signature:

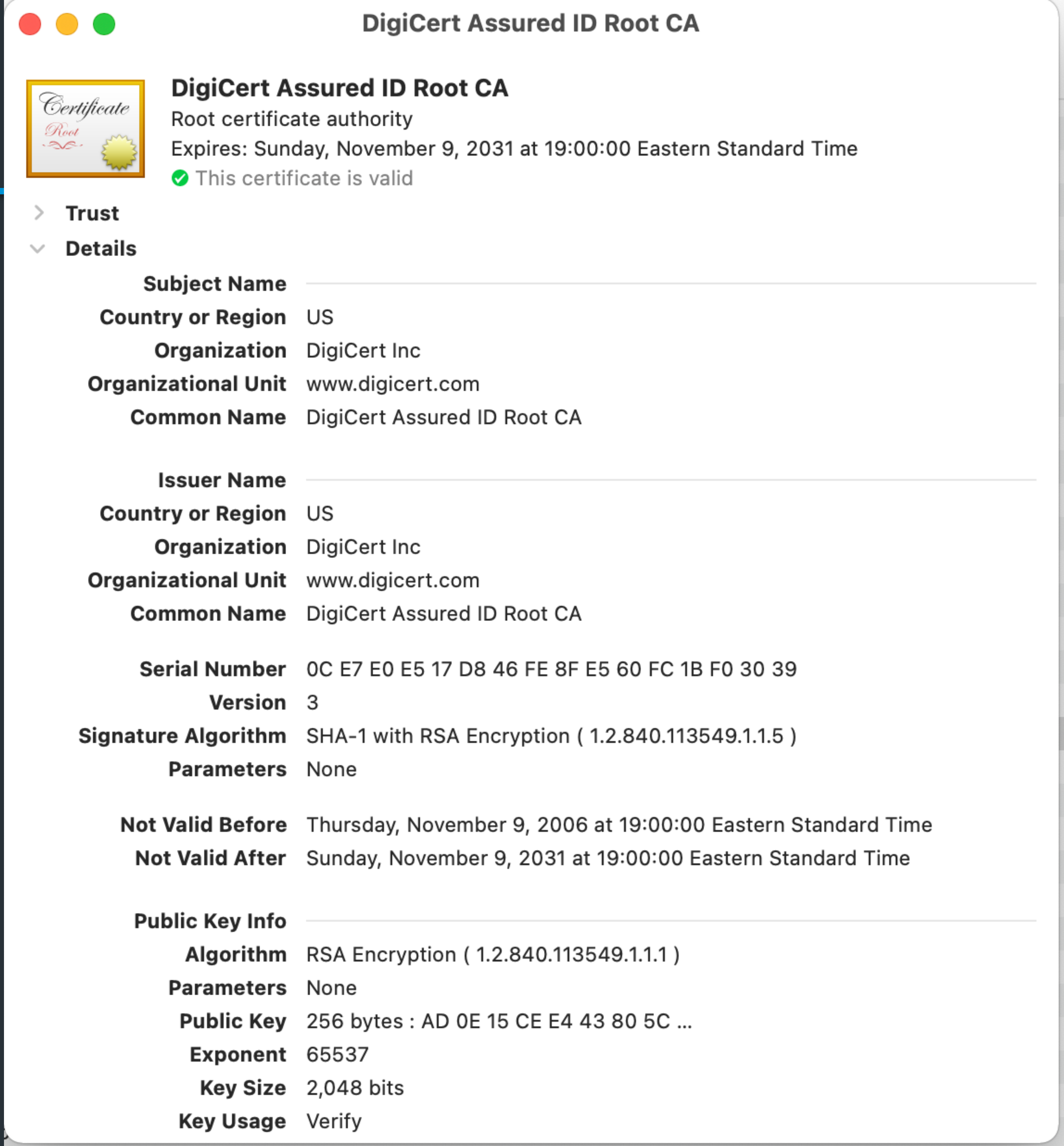
  Verify($K_{pub,CA}$ Cert) => True/False



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

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

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

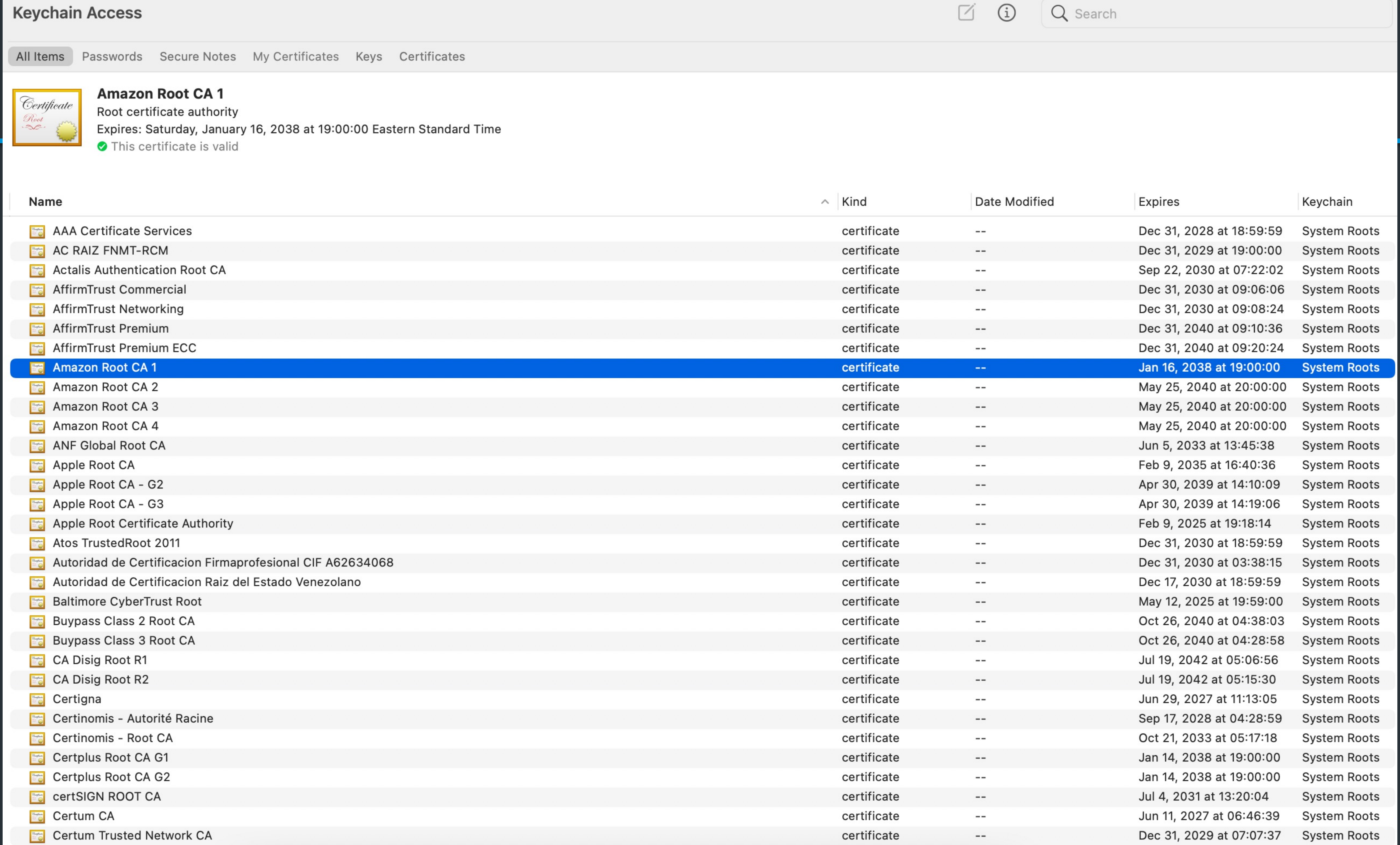


DigiCert Assured ID Root CA

DigiCert Assured ID Root CA
Root certificate authority
Expires: Sunday, November 9, 2031 at 19:00:00 Eastern Standard Time
This certificate is valid

Trust
Details

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



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

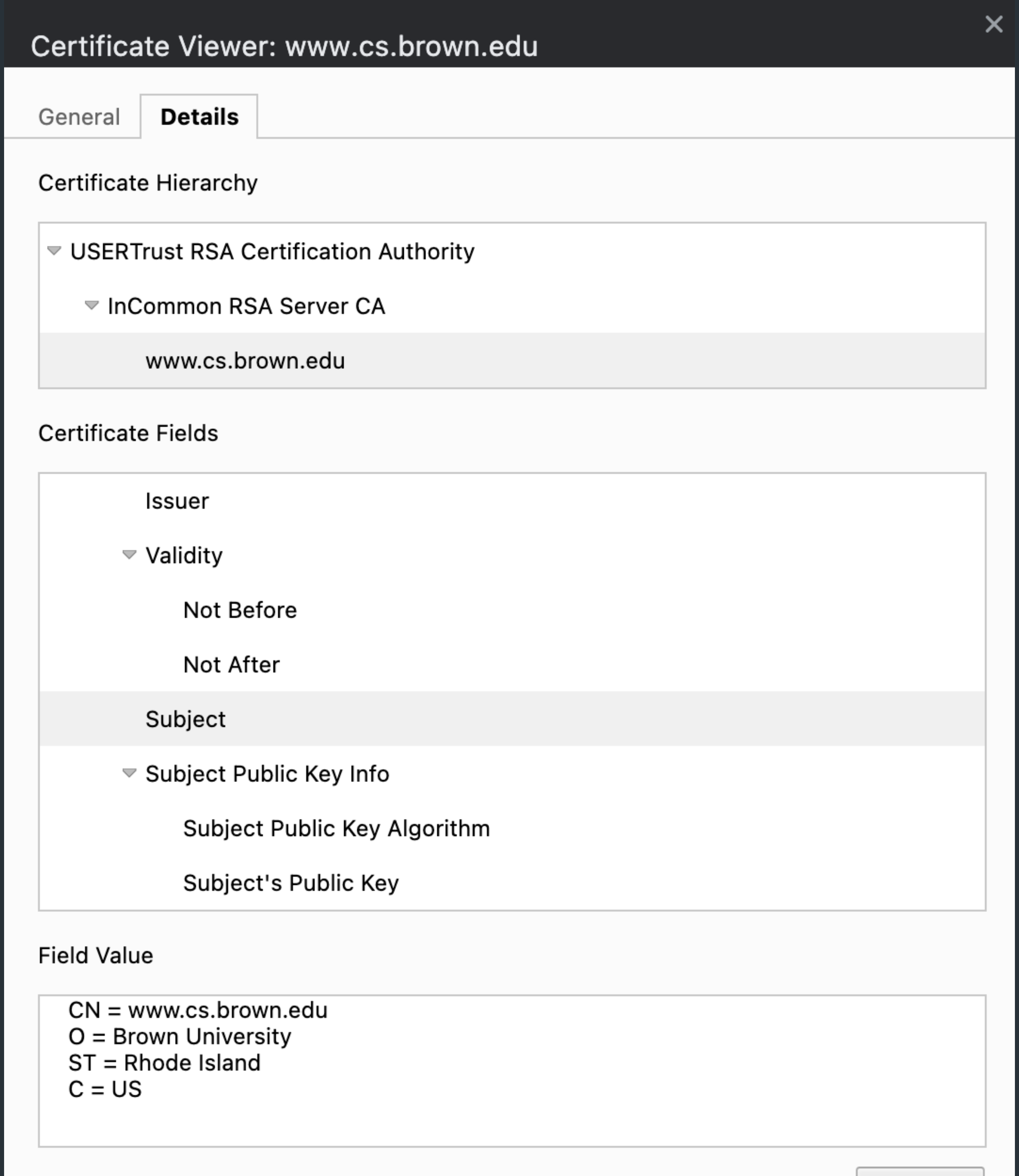
What's in a certificate?

- 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!


Certificate Viewer: www.cs.brown.edu

General | **Details**

Certificate Hierarchy

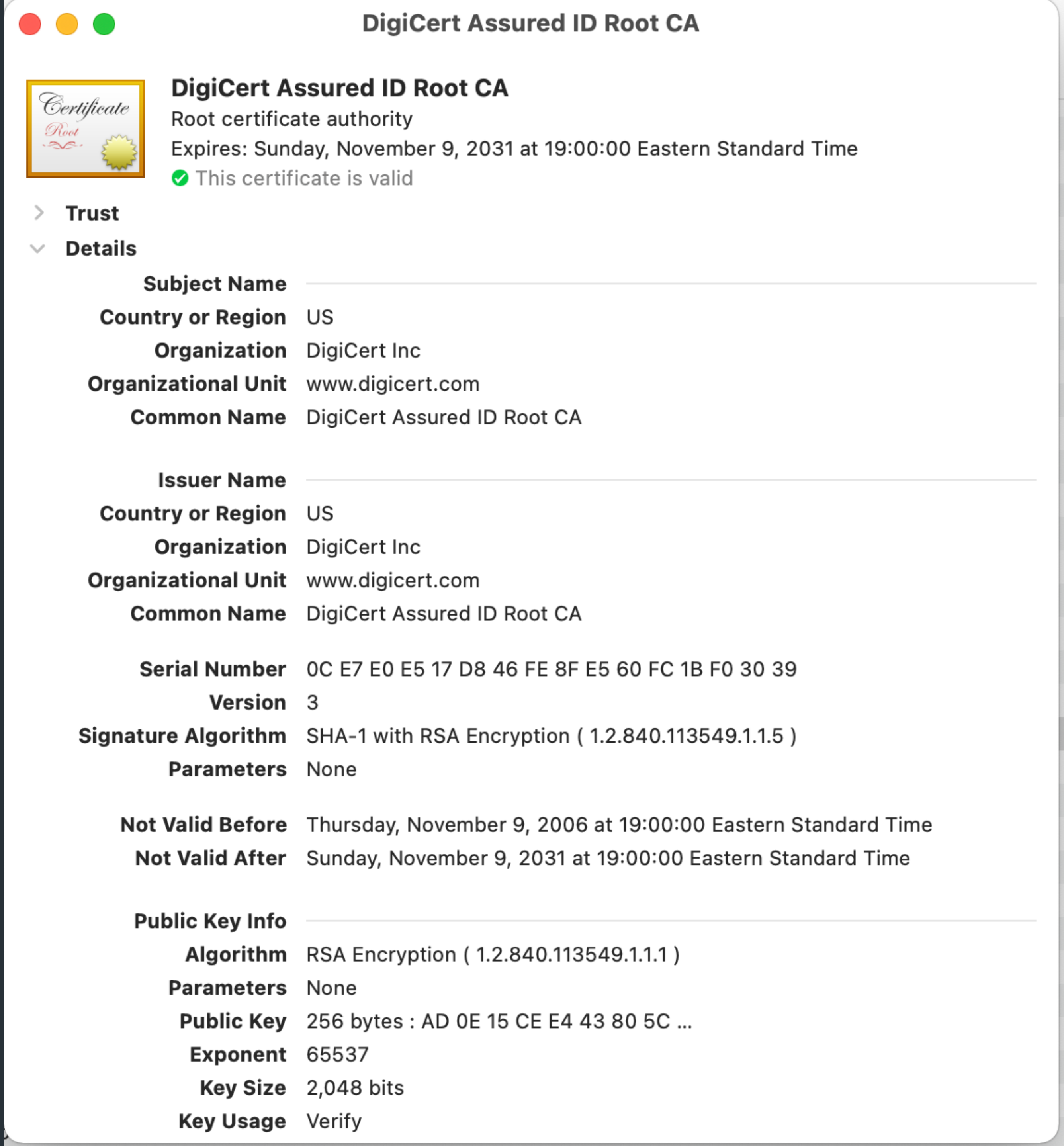
- USERTrust RSA Certification Authority
  - InCommon RSA Server CA
    - www.cs.brown.edu

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
```

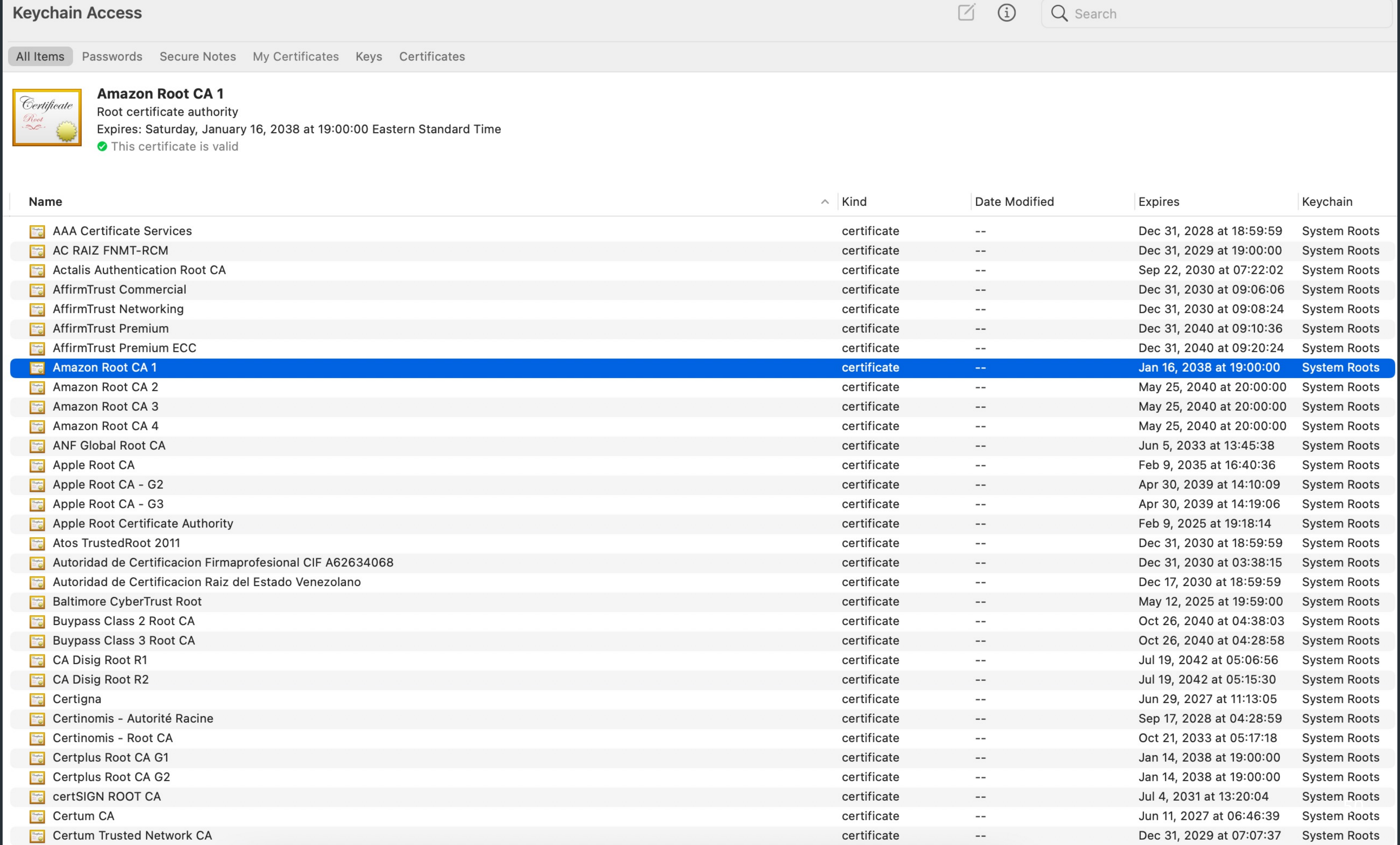

DigiCert Assured ID Root CA

DigiCert Assured ID Root CA
Root certificate authority
Expires: Sunday, November 9, 2031 at 19:00:00 Eastern Standard Time
✓ This certificate is valid

› **Trust**

⌄ **Details**

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



Keychain Access

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

PKI hierarchy

In reality, PKI creates a hierarchy of trust:

- <u>Root CAs</u>: k_{pub} stored in virtually every browser, OS
  - Private keys protected by most stringent security measures (software, hardware, physical)
- <u>Intermediate CAs</u>: 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)

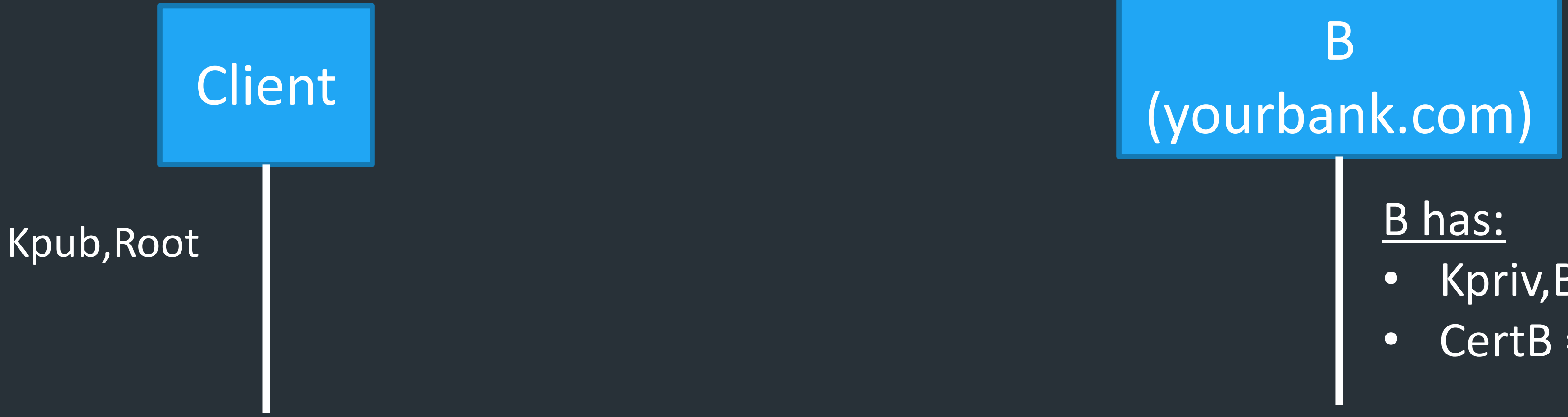
- Intermediate CAs: k_{pub} signed by root CA
  - Sign certificates for general use (ie, regular websites)
  - Doesn't require same protections as root

- General-use certificates: for a specific webserver

What happens if a root is compromised?

How the hierarchy works

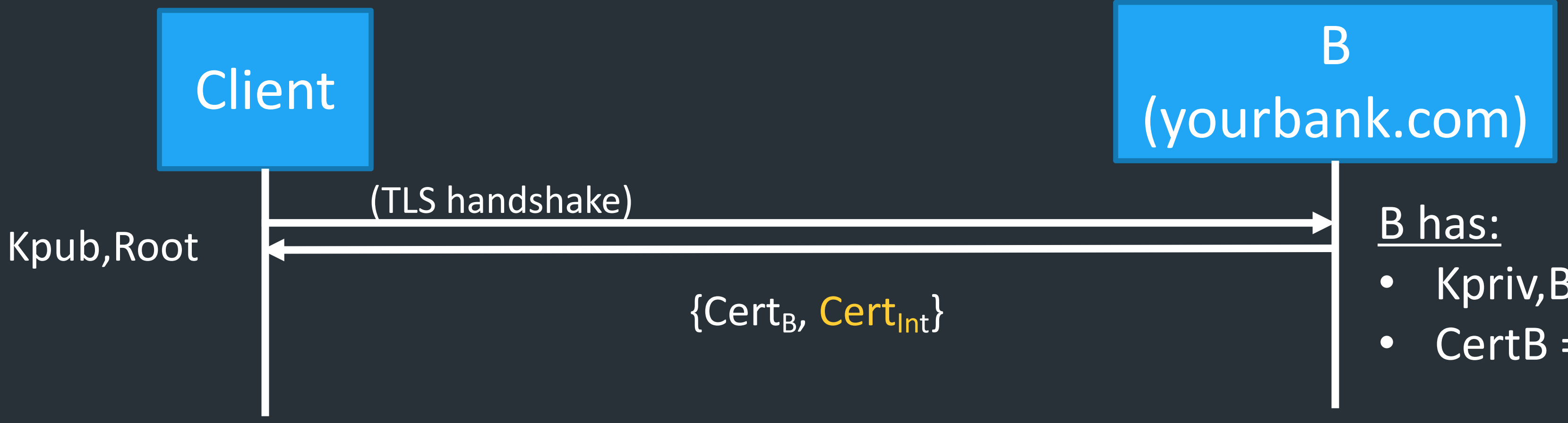
Ex. Server has certificate from Intermediate CA_{Int}



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

How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}

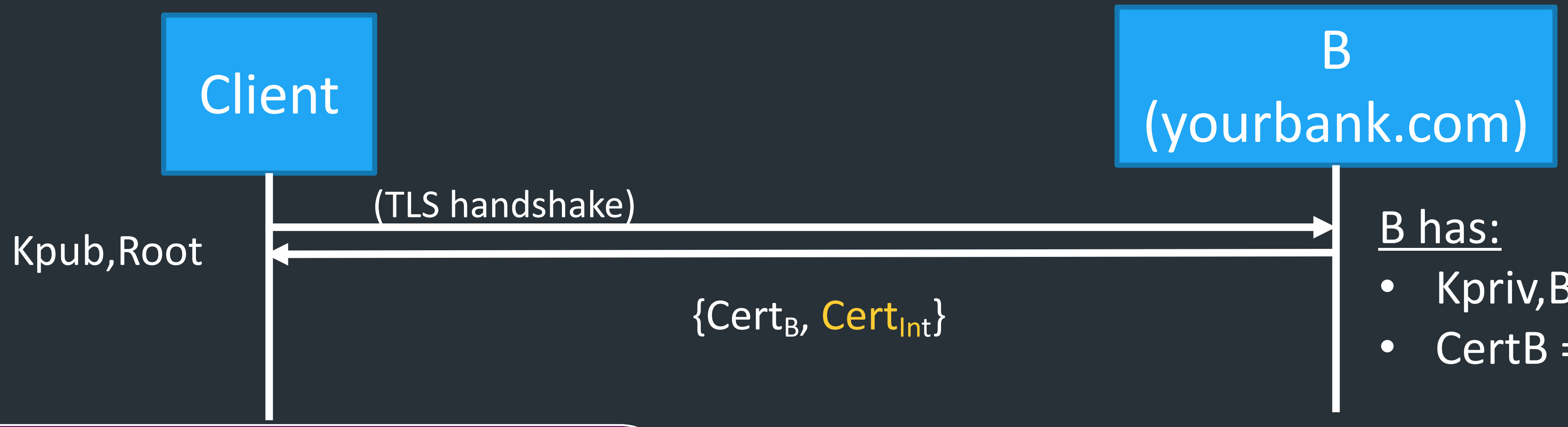


B has:

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

How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}

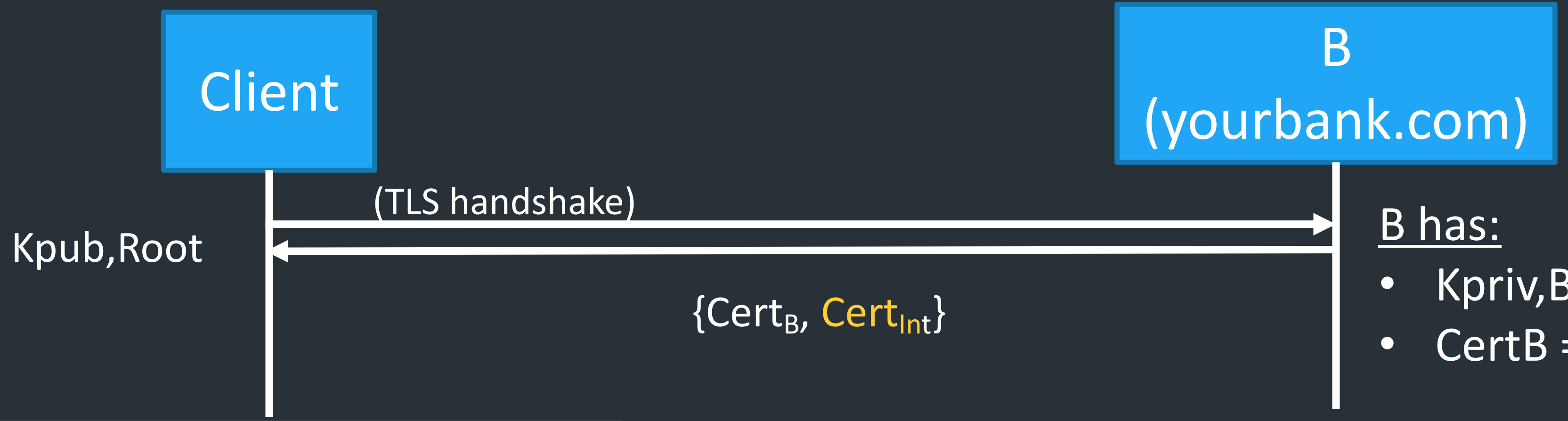


B has:
- Kpriv,B
- CertB = { $K_{pub,B}$, Sign($K_{pub,B}$, $K_{priv,Int}$), … }

Client's workflow:
- Checks metadata ✅
- Verify($Cert_B$, $K_{pub,Int}$) ✅
- Verify($Cert_{Int}$, $K_{pub,Root}$) ✅

How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}



<u>B has:</u>

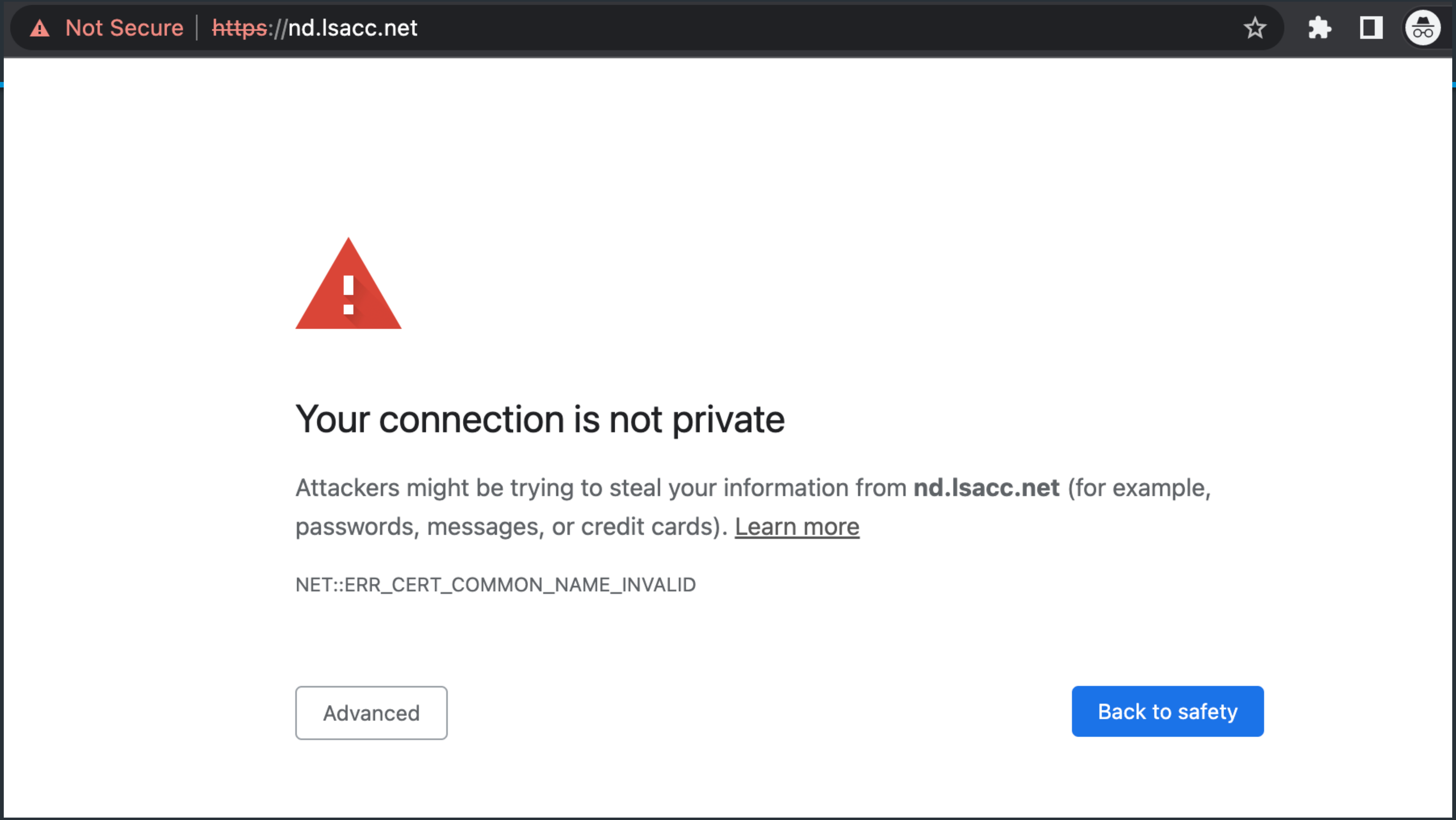
- Kpriv,B
- CertB = { $K_{pub,B}$, Sign($K_{pub,B}$, $K_{priv,Int}$), … }

<u>Client's workflow:</u>

- 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

Most common TLS errors you might see

- Common name (eg. yourbank.com) invalid
- Self-signed
- Certificate expired

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

Most common TLS errors you might see

- Common name invalid
- Self-signed
- Certificate expired

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

=> Might occur if webserver configured improperly, or if you're setting up a system

Rogue Certificates?

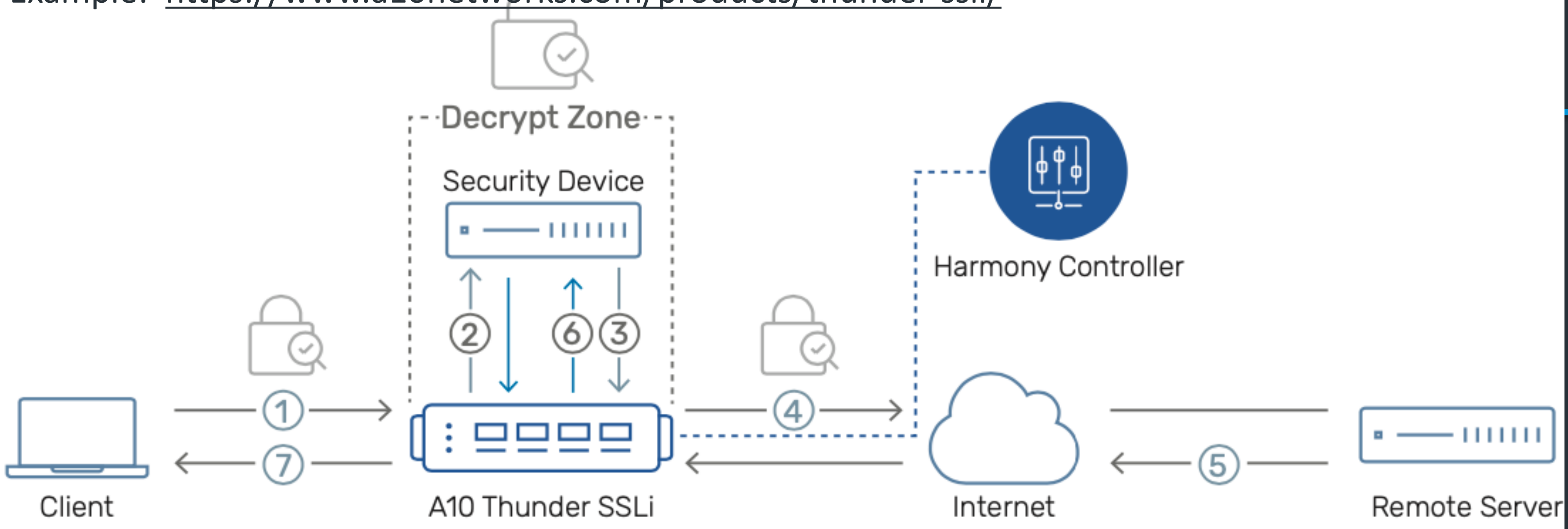
- In 2011, DigiNotar, a Dutch root certificate authority, was compromised
- The attacker created rogue certificates for popular domains like google.com and yahoo.com
- DigiNotar was distrusted by browsers and filed for bankruptcy
- See the incident investigation report by Fox-IT

- 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?

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.

PKIs, TLS, and HTTPS

The story so far

- Asymmetric crypto: each entity gets a key in two parts
  - K_{priv}: Private key, kept secret
  - K_{pub}: Public key, shared with everyone

- Can provide important security properties
  - Authentication/Integrity: A *signs* message with $K_{priv,A}$, anyone with $K_{pub,A}$ can verify message came from A
  - Confidentiality: A *encrypts* message to B with $K_{pub,B}$, B can decrypt with $K_{priv,B}$

- But: how do we know if we can trust a public key?

Public Key Infrastructure (PKI)

Public key crypto is *very* powerful …

- … but the realities of tying public keys to real world identities turn out to be quite hard

- PKI: *Trust distribution* mechanism
  - Authentication via Digital Certificates
- Note: Trust doesn't mean someone is honest, just that they are who they say they are…

Managing Trust

- The most solid level of trust is rooted in our direct personal experience
  - E.g., Alice's trust that Bob is who they say they are
  - Clearly doesn't scale to a global network!

- In its absence, we rely on *delegation*
  - Alice trusts Bob's identity because Charlie attests to it ….
  - …. and Alice trusts Charlie

Managing Trust, con't

- Trust is not particularly transitive
  - Should Alice trust Bob because she trusts Charlie …
  - … and Charlie vouches for Donna …
  - … and Donna says Eve is trustworthy …
  - … and Eve vouches for Bob's identity?

- Two models of delegating trust
  - Rely on your set of friends and their friends
    - "Web of trust" -- e.g., PGP
  - Rely on trusted, well-known authorities (*and those they trust…*)
    - "Trusted root" -- e.g., HTTPS

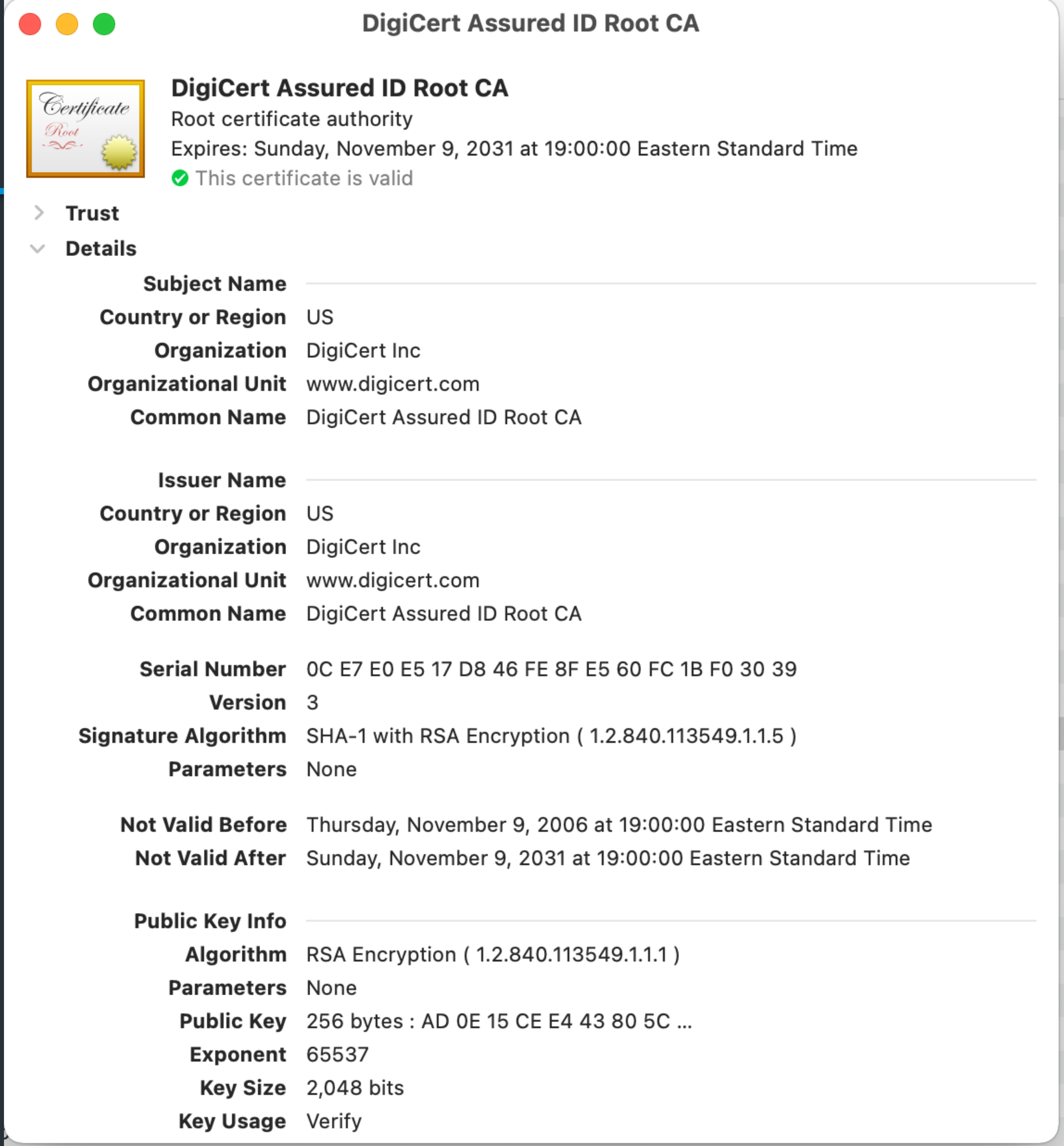
PKI Conceptual framework

Public keys managed by Certificate Authorities (CAs)

- Everyone knows public key for some root CAs
- To publish a public key for entity X, root CA R *signs* X's public key
  - What this means: CA agrees that this is X's public key
  - Creates a Certificate: {$K_{pub,X}$, signature, metadata}

- Given signature, anyone who knows the root can verify
  - Delegates trust of Kpub,X to CA
  - If you trust the CA, you now trust X

- Root CAs: pre-installed in your system/browser

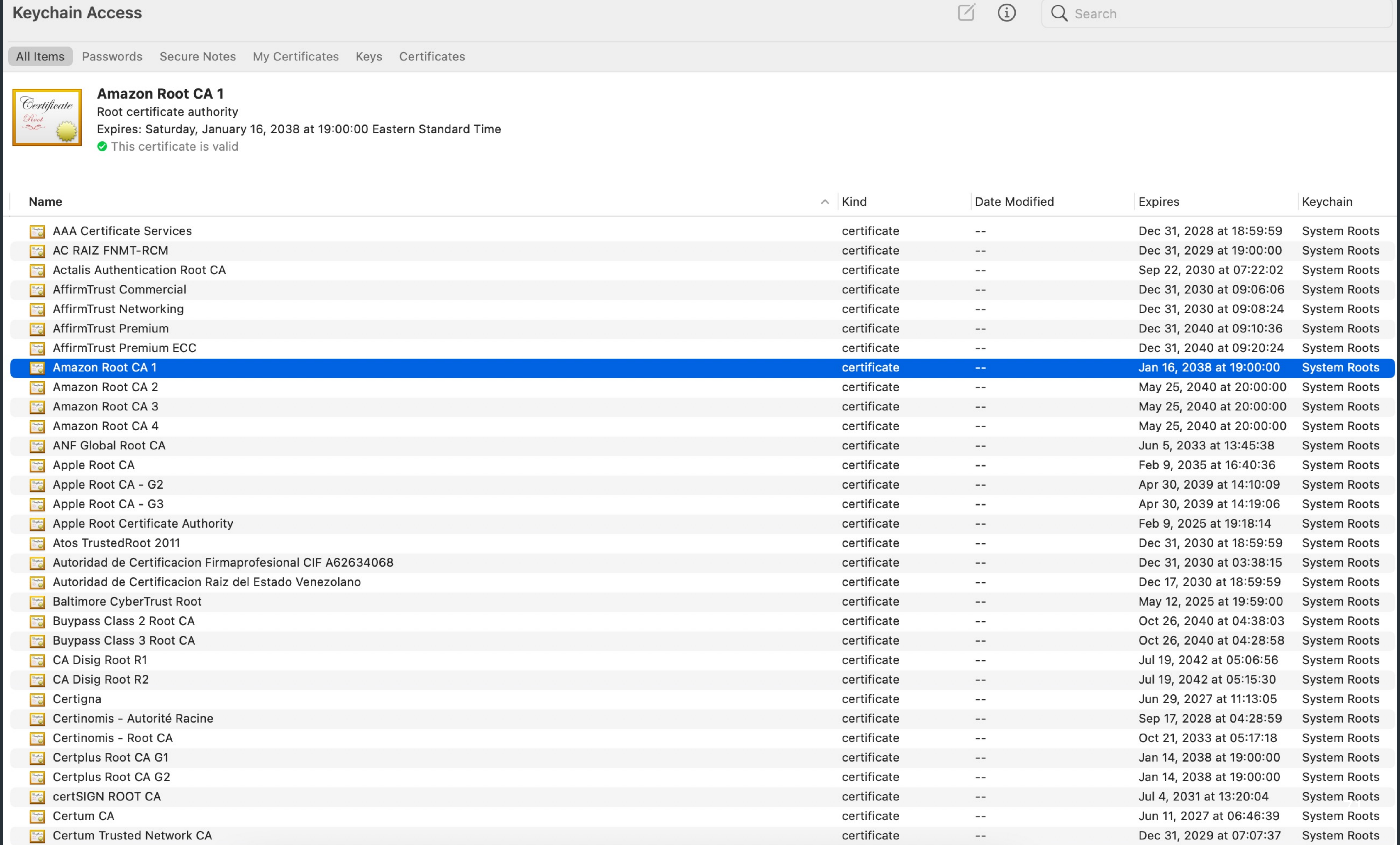

DigiCert Assured ID Root CA

DigiCert Assured ID Root CA
Root certificate authority
Expires: Sunday, November 9, 2031 at 19:00:00 Eastern Standard Time
✓ This certificate is valid

> Trust

∨ Details

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



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

PKI hierarchy

- In reality, hierarchy of trust
- Root CAs sign certificates for Intermediate CAs
- Intermediate CAs sign certificates for general users/sites

The further up the hierarchy, the more protections it needs

- CA's often use Hardware Security Modules (HSMs), other physical protections…
- What happens if a CA is compromised?

PKI Example

Inside the Server's Certificate

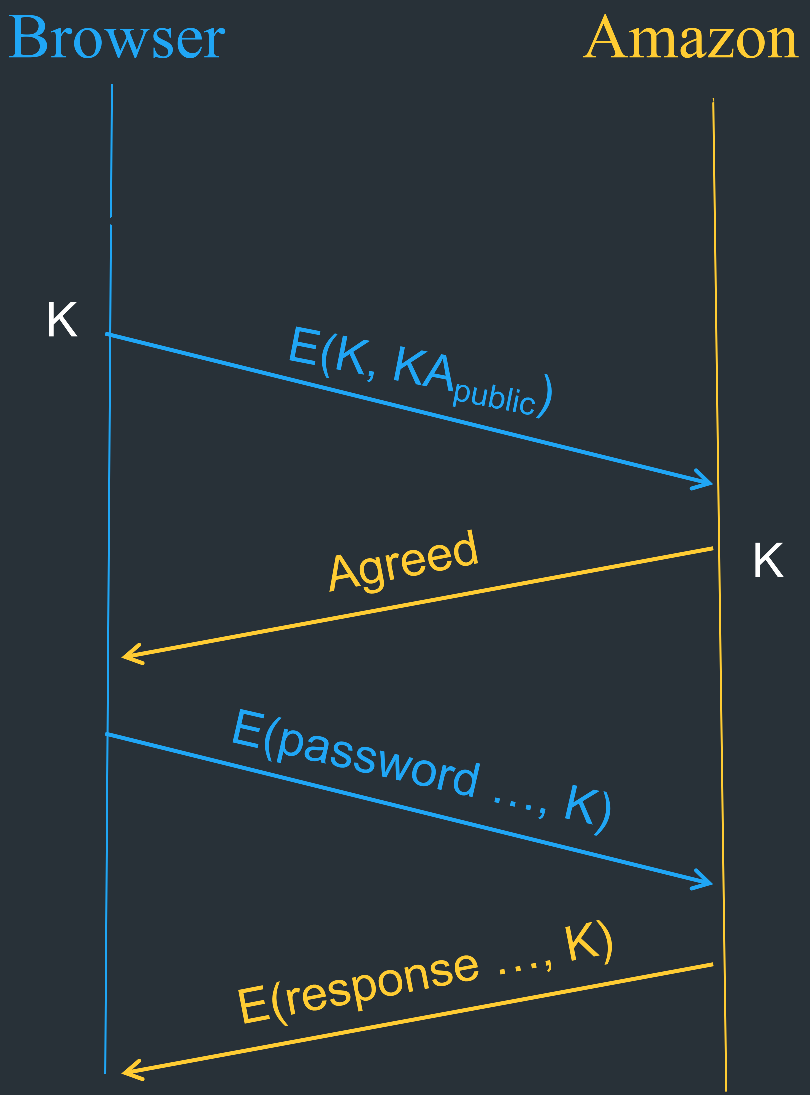
- **Common name: Domain name** for cert (e.g., amazon.com)
- Amazon's **public key**
- A bunch of auxiliary info (physical address, type of cert, expiration time)
- URL to *revocation center* to check for revoked keys
- Name of certificate's **signatory** (who signed it)
- A public-key **signature** of a hash of all this
  - Constructed using the signatory's private RSA key

Validating Amazon’s Identity

- Browser retrieves cert belonging to the **signatory**
- If it can’t find the cert, then warns the user that site has not been verified
  - And may ask whether to continue
  - *Could* still proceed, just without authentication
- Browser uses public key in signatory’s cert to decrypt signature
  - Compares with its own hash of Amazon’s cert
- Assuming signature matches, now have high confidence it’s indeed Amazon
  - … assuming signatory is trustworthy

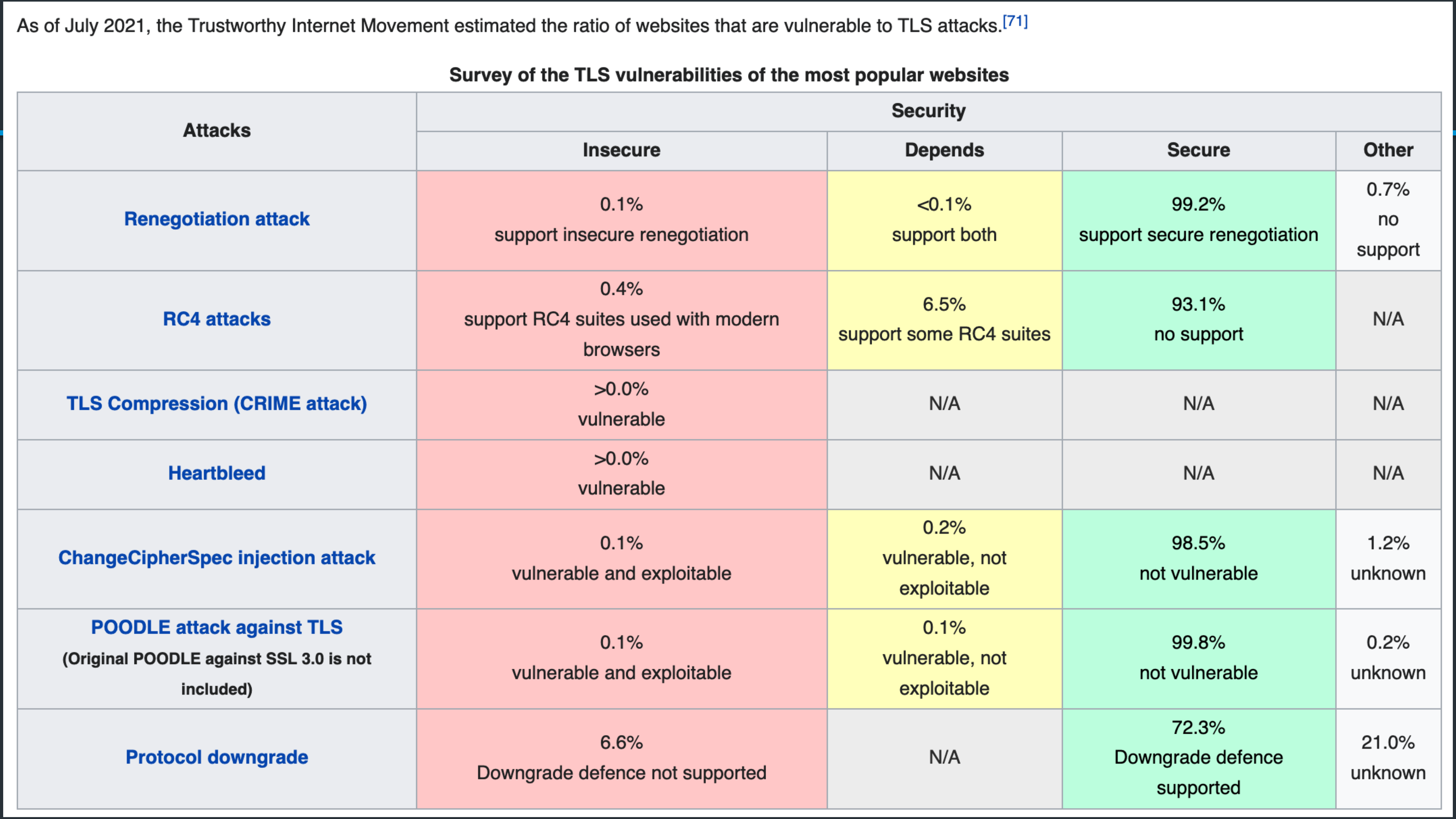
HTTPS Connection (SSL/TLS), con't

- Browser constructs a random *session key* K
- Browser encrypts K using Amazon's public key
- Browser sends E(K, KA_{public}) to server
- Browser displays 🔒
- All subsequent communication encrypted w/ symmetric cipher using key K
  - E.g., client can authenticate using a password



When does this break down?

- TLS is hard to implement
- Need to trust the CAs
- Users need to understand warnings

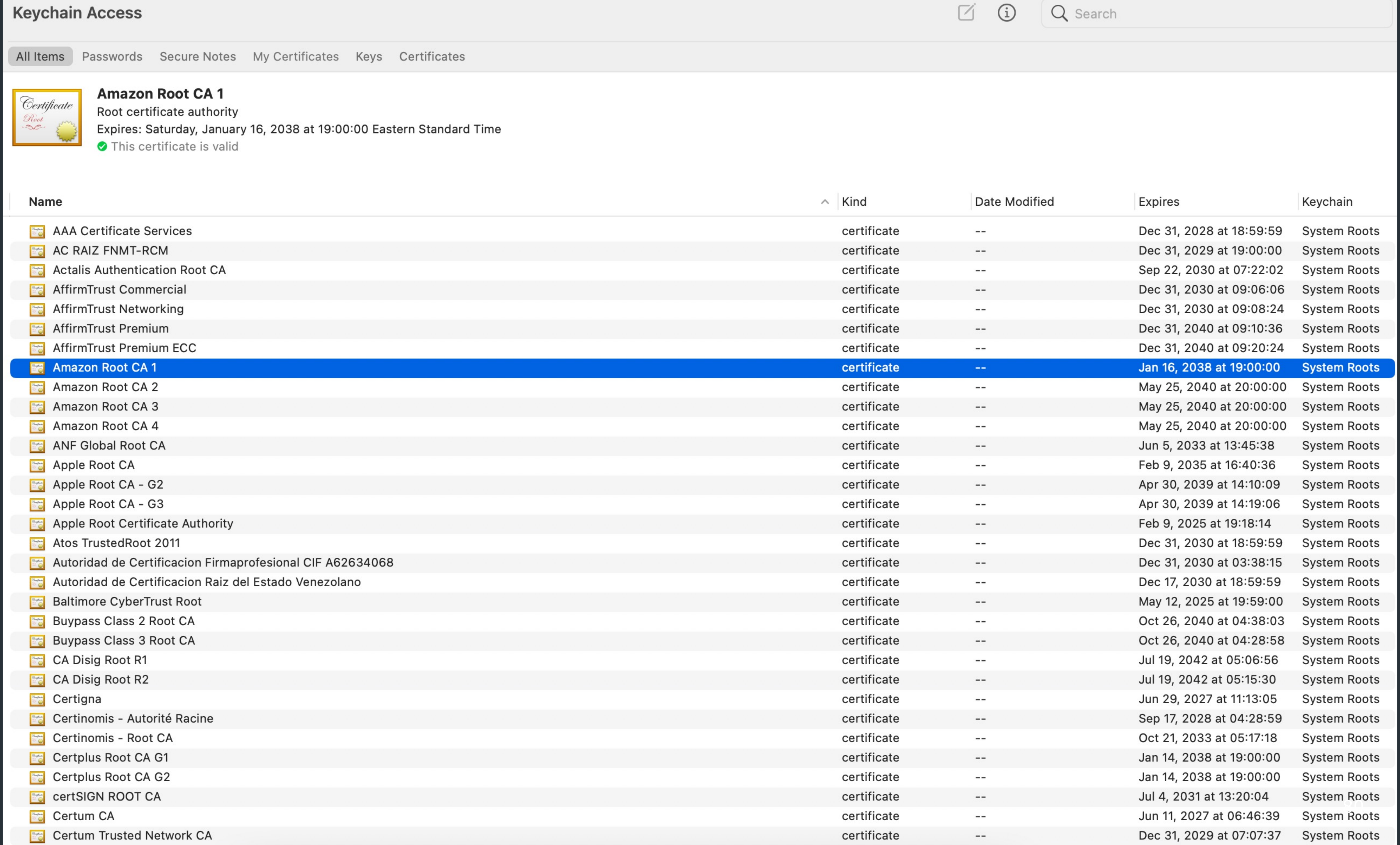


As of July 2021, the Trustworthy Internet Movement estimated the ratio of websites that are vulnerable to TLS attacks.[71]

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/



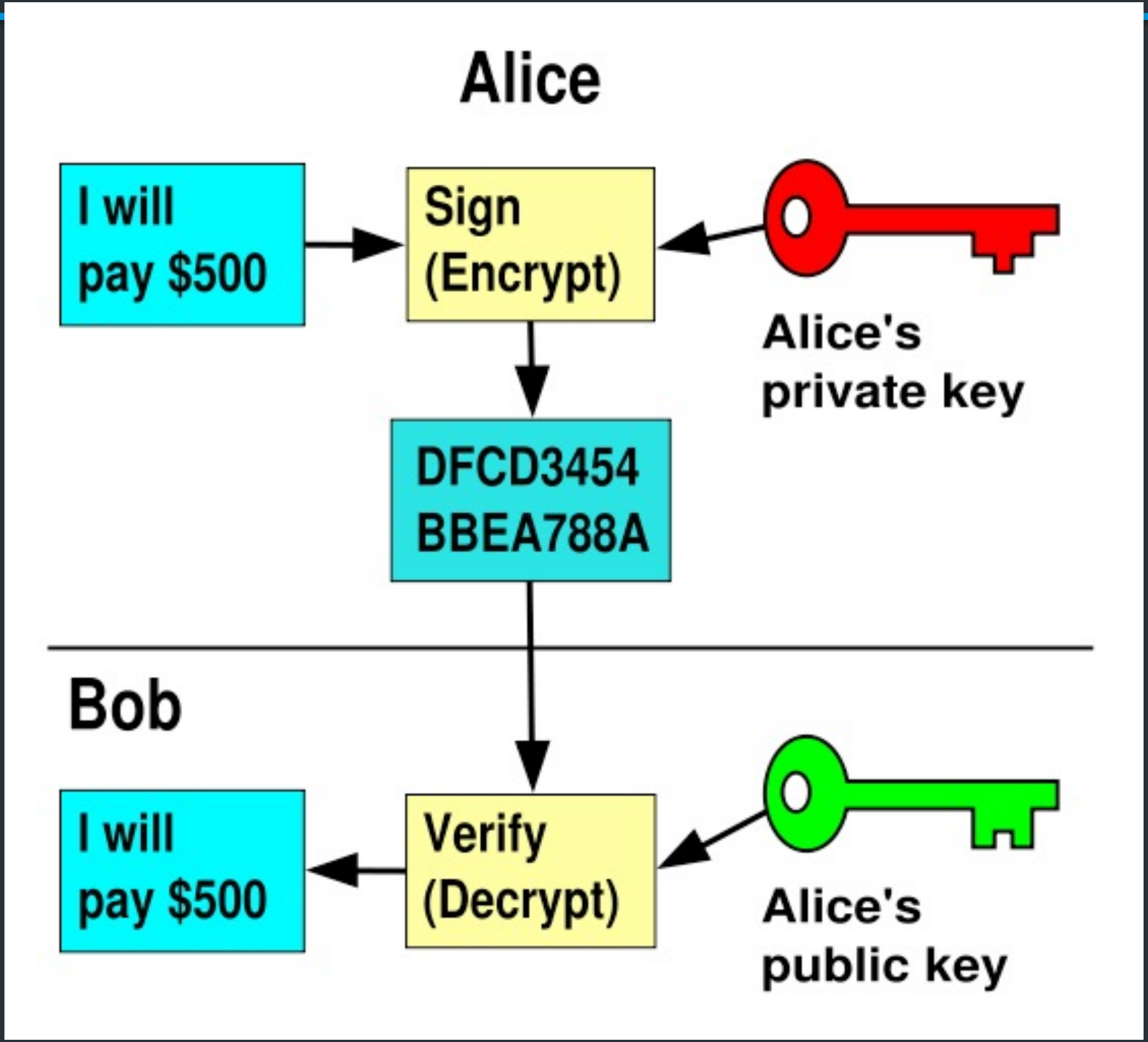
Amazon Root CA 1
Root certificate authority
Expires: Saturday, January 16, 2038 at 19:00:00 Eastern Standard Time
✓ This certificate is valid

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

Digital Signatures

- Suppose Alice has published public key K_E
- If she wishes to prove who she is, she can send a message x encrypted with her **private** key K_D
  - Therefore: anyone w/ public key K_E can recover x, verify that Alice must have sent the message
  - It provides a digital signature
  - Alice can't deny later deny it $\Rightarrow$ non-repudiation

RSA Crypto & Signatures, con't



Summary of Our Crypto Toolkit

- **If** we can securely distribute a key, then
  - Symmetric ciphers (e.g., AES) offer fast, presumably strong confidentiality
- Public key cryptography can make this easier (can share public keys anywhere)
  - But not as computationally efficient
  - Use public key crypto to exchange session key, which is used for symmetric encryption
  - And not guaranteed secure
    - but **major** result if not

Summary of Our Crypto Toolkit, con't

- Cryptographically strong hash functions provide major building block for integrity (e.g., SHA-256)
  - As well as providing concise digests
  - And providing a way to prove you know something (e.g., passwords) without revealing it (non-invertibility)
  - But: worrisome recent results regarding their strength (MD5, SHA1)
- Public key also gives us signatures
  - Including sender non-repudiation
- Turns out there's a crypto trick based on similar algorithms that allows two parties *who don't know each other's public key* to securely negotiate a secret key even in the presence of eavesdroppers
  - Look up: Diffie-Hellman Key Exchange