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?

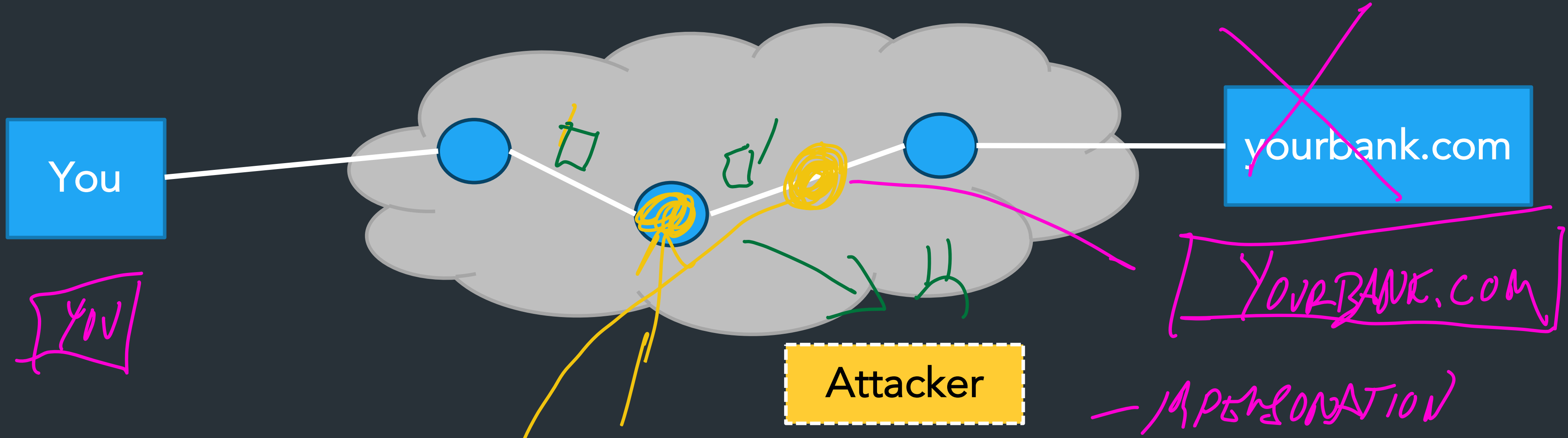
- CS1660 (Spring): Intro to Computer Systems Security
- CS1515 (Spring): Applied cryptography
- CS1510 (Fall): Intro to Cryptography and Computer Security

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?



- EAVESDROPPING
- TAMPERING
- IMPERSONATION
- DENIAL OF AVAILABILITY

(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?

NEXT LECTURE.

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:

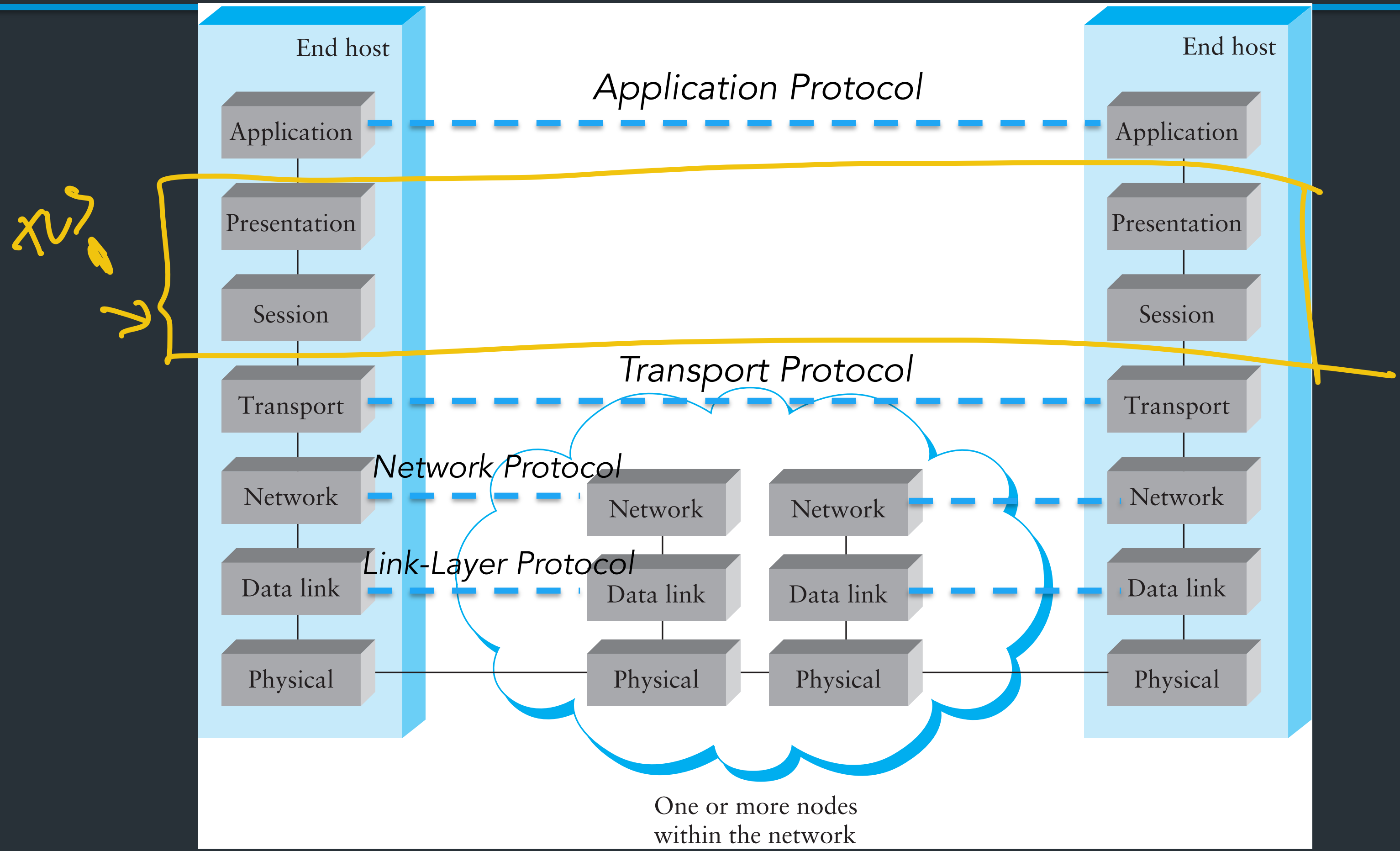
- Confidentiality
- Integrity
- Authentication

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 <u>individual *links*</u>
Physical	Service: move bits to other node across link

Throwback: The OSI model



Fundamental crypto properties we need

Symmetric cryptography

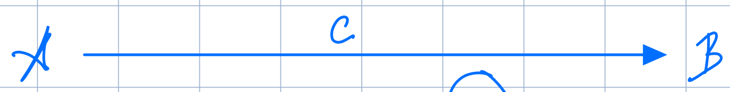
- A, B share secret key k
- Examples: AES, Serpent, Whirlpool, DES (old, insecure), …
- Provides: confidentiality (encrypt/decrypt), integrity (MAC)

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

Brief overview: symmetric crypto

Setup:

- A wants to send message m
- A, B agree on secret key K
  - => Must be exchanged beforehand using some other secure method



(1) A encrypts message with key k, sends ciphertext c
c = Enc(k, m)

(2) B decrypts c to recover original message
m = Dec(k, c)

This provides: confidentiality
Attacker can only see ciphertext. If encryption scheme is strong, adversary can't learn anything about m
- (Unless they can steal k somehow. . .)

Examples: AES, DES (old, insecure), Serpent, Whirlpool, …

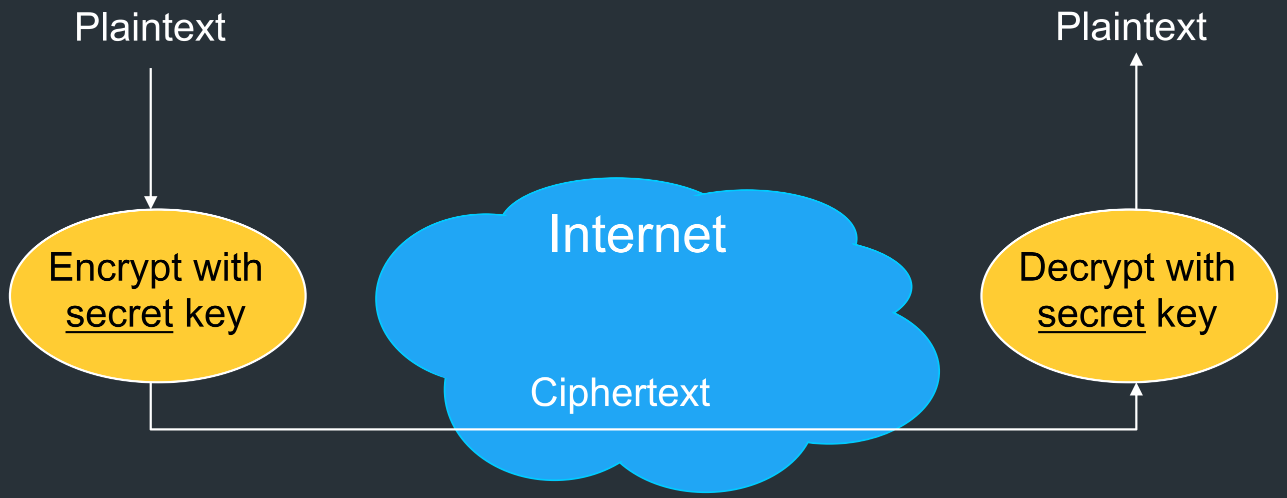
Key properties
- Symmetric crypto is fast (relative to other crypto in our toolkit). Modern CPUs even have hardware support for AES, the most common symmetric scheme
  - => Most data is protected using symmetric crypto

However: need to exchange a shared key beforehand
How to do this if you 1) can't send k over the network in the clear, and 2) need keys for everyone you might talk to?

=> There exist crypto protocols to establish a shared key without sending it over the network (beyond this course—eg. look up Diffie-Hellman Key Exchange (DHKE) for one)

=> Also need to verify the sender's identity before you start communicating (ie, how do you know if they are who they say they are?)

Confidentiality: Symmetric encryption



Brief overview: asymmetric crypto

Setup:

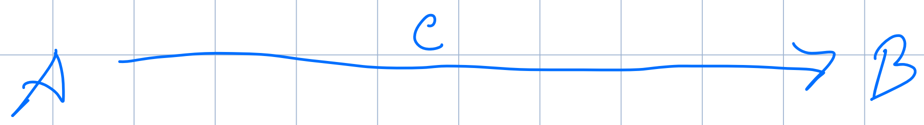
- A and B each have a public/private key pair (eg. Kpub_A, Kpriv_A)
- Kpub is known to everyone, Kpriv is kept secret

From here, there are two fundamental operations we can perform:

- Encryption/Decryption: like before, but now taking advantage of two keys
- Signing and verification: used to verify a message came from a specific party (next page)

I) Asymmetric encryption

1. A encrypts message with B's public key, Kpub,B
2. B can decrypt with its private key Kpriv,B



c = Enc(Kpub_B, m)

m = Enc(Kpriv_B, c)

This is pretty powerful: can encrypt a message for B without a shared key!
However: asymmetric crypto is very slow (orders or magnitude slower than symmetric crypto)

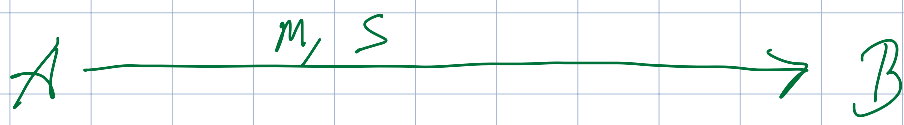
=> Used for authenticating at connection start (more on this in a moment)
=> Use asymmetric crypto to establish a symmetric key at session start, use for rest of connection

II) Signatures and verification

Idea: want to use public key crypto to verify a message came from a certain party

1. A **signs** message (or hash of message) with its private key
   => Produces a **signature**: a small value like a hash

2. B can **verify** the signature using A's public key
   => Outputs a true if message was signed with Kpriv_A, otherwise false



s = Sign(Kriv_A, m)

b = Verify(Kpub_A, m, s)

Also pretty powerful: anyone can verify that m was signed by A's private key
=> Assuming that A's private key is indeed private (only A has it), this means only A could have signed the message

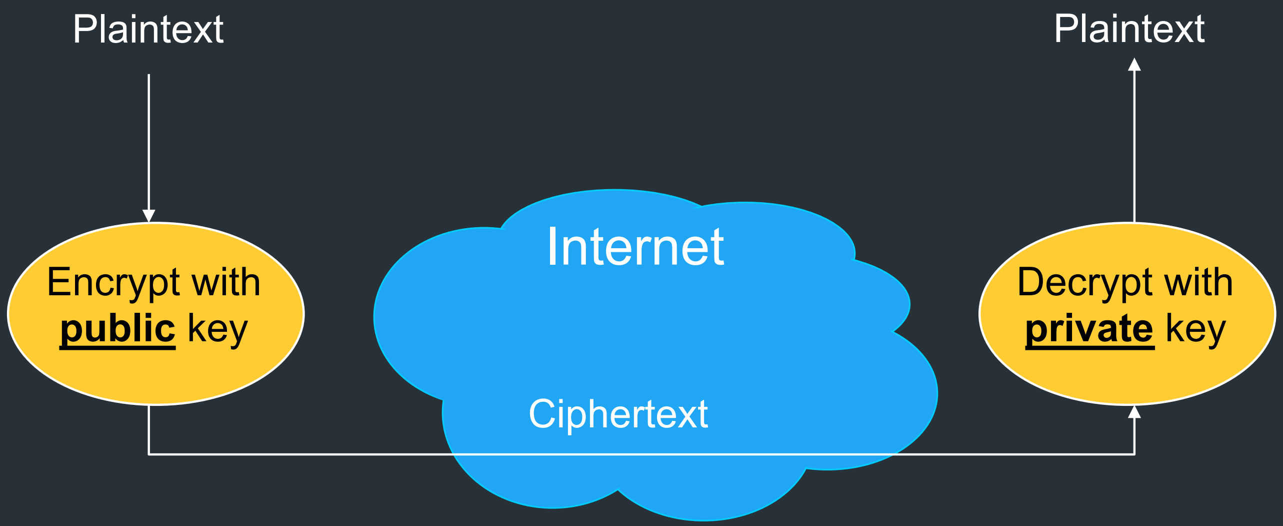
Some notes

- This is also slow, like asymmetric encryption/decryption
- Could also encrypt the message, but also important to sign public info: for example, an open-source developer might sign a software update to prove it's legitimate

- *(not necessary for this class) Signing has another important crypto property: non-repudiation, meaning A can't prove it didn't sign message m*

Public Key / Asymmetric Encryption

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



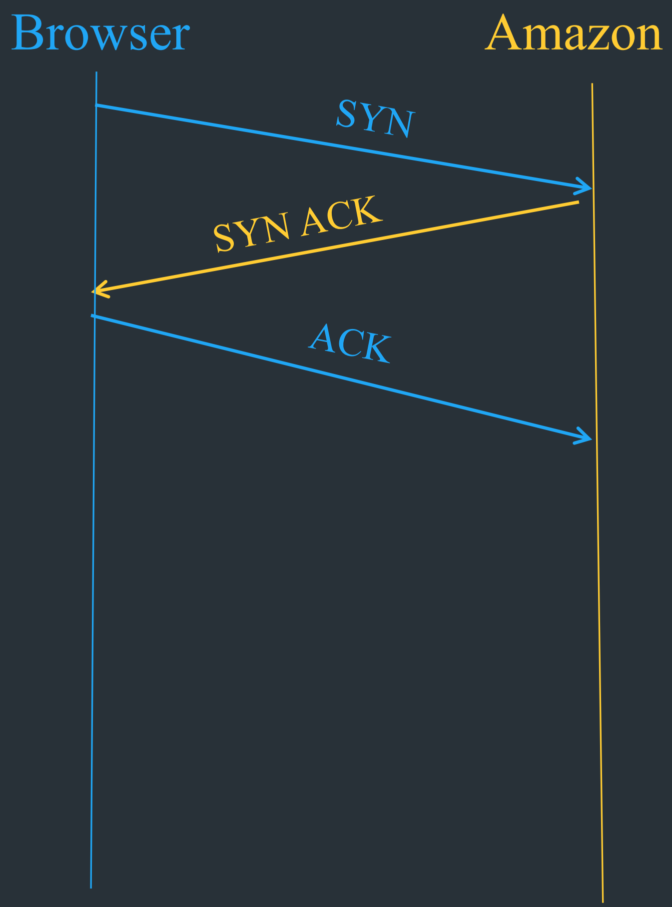
How it works in TLS

- 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)

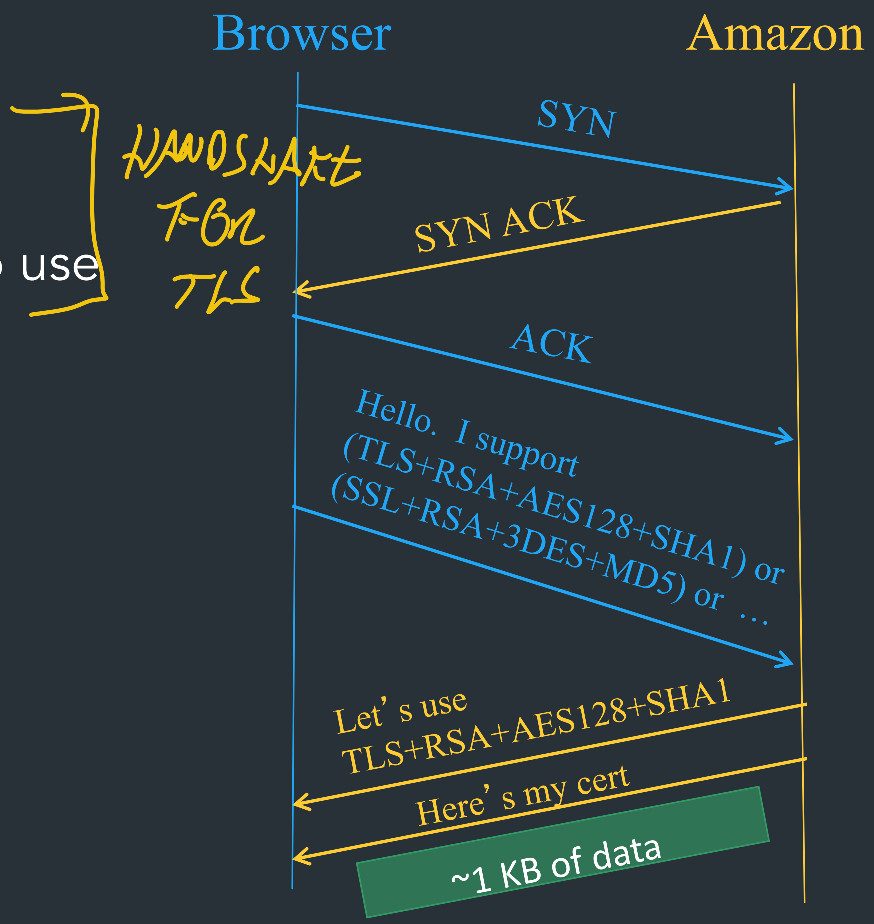
TLS: setup

- 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

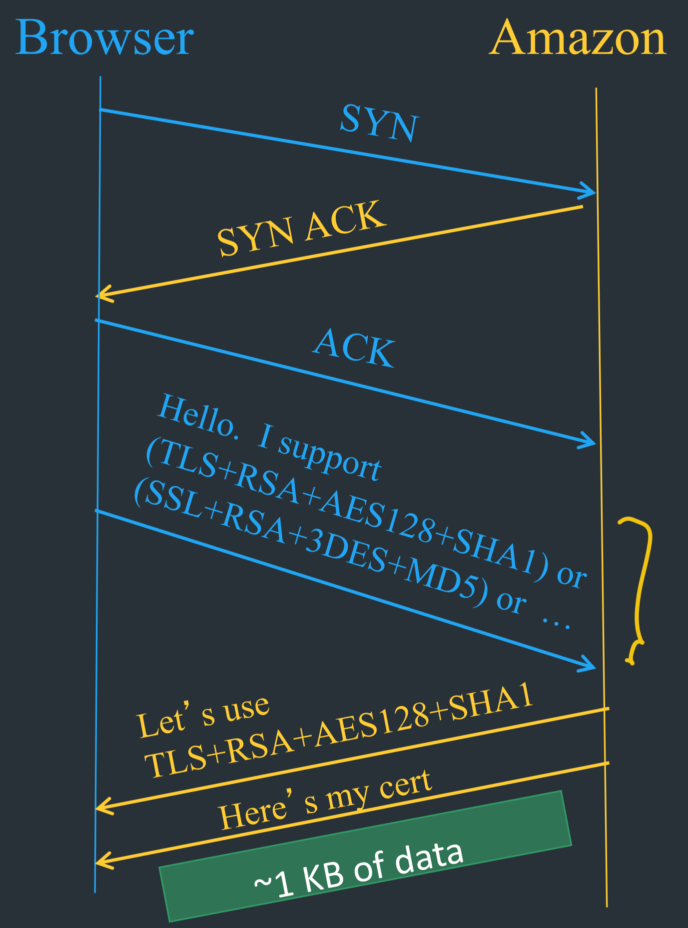


TLS: setup

SOME OF THIS IS IN CLEAR, → TRICKY

- 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



At startup, client/server must agree on what crypto methods to use—these are called ciphersuites
=> These cover what crypto algorithms are used for the different parts (key exchange, what asymmetric crypto to use, what symmetric crypto to use, hashing functions for integrity, etc.)

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]

→ SYMMETRIC PART (ENC, INTEGRITY)

→ AUTH/ KEY EXCHANGE (ASYM)

TLS + Authentication

TLS Goals

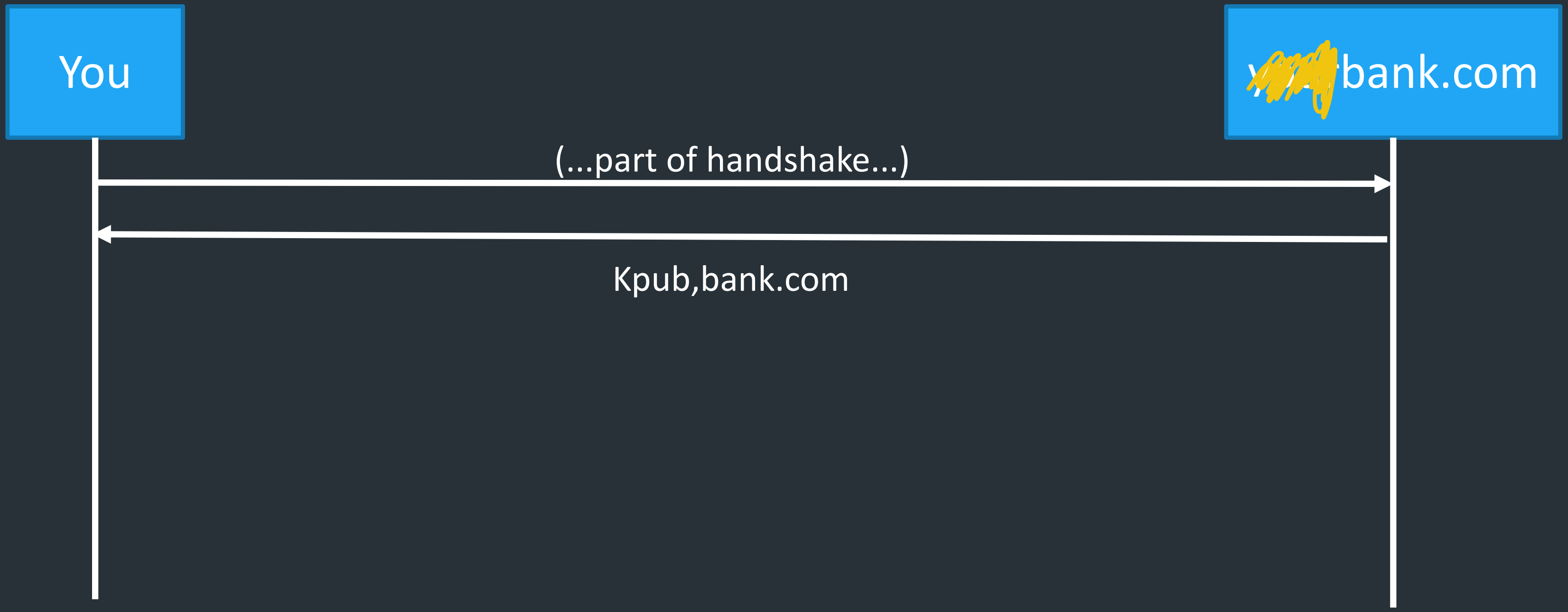
Authentication: verifying that the entity on the other end of the connection is who they claim to be

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

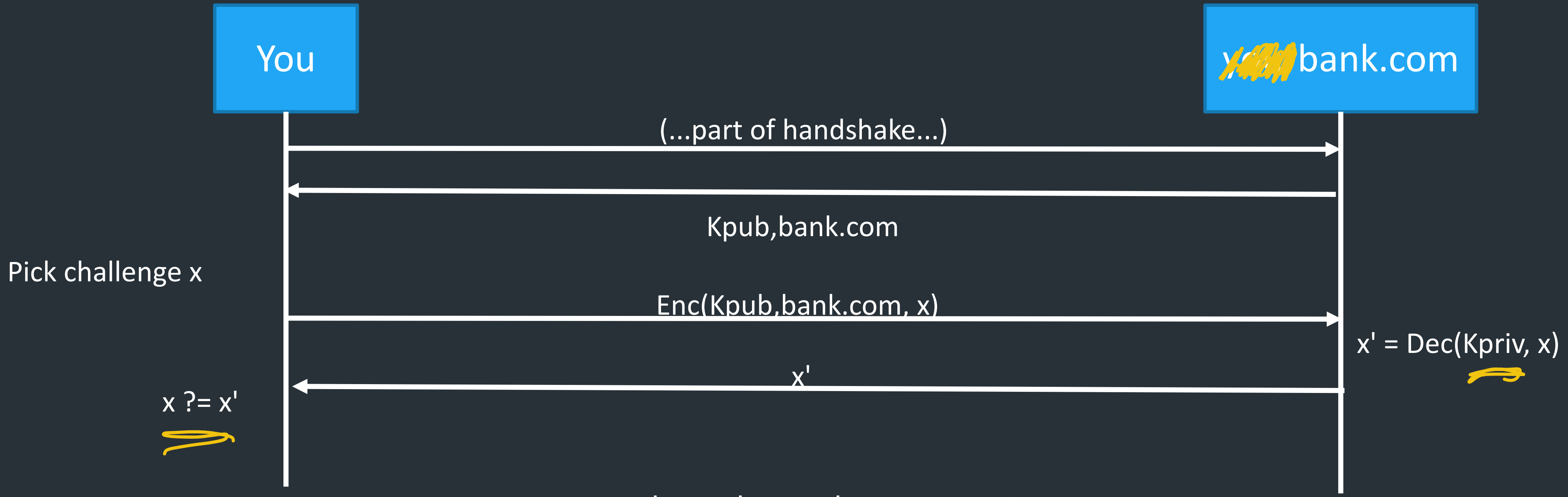
Everything we've talked about so far relies on each server having a public/private key

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

The Challenge



The Challenge



What does this prove?

=> ENDPOINT YOU ARE TALKING TO HAS PRIVATE KEY

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!"

— DNS COULD BE SPOOFED

— IP TRAFFIC MAY BE REDIRECTED (BGP SPOOFING)

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

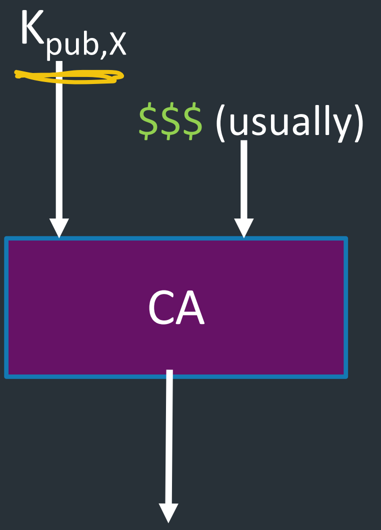
Public keys managed by Certificate Authorities (CAs)

- Everyone knows public key for some <u>root CAs</u>
  - 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 ⇒ EVERYONE HAS $K_{PUB,CA}$
- If X wants a public key, request from CA
  - CA validates X's identity, then signs X's public key
  - Generates certificate



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

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

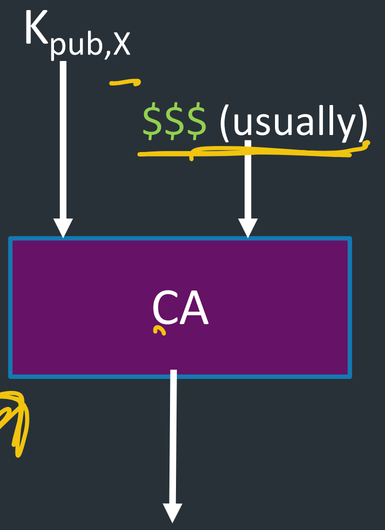
PKI: The main idea

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}

=> 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

PUBLIC KEY

Keychain Access

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

Search

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: 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)

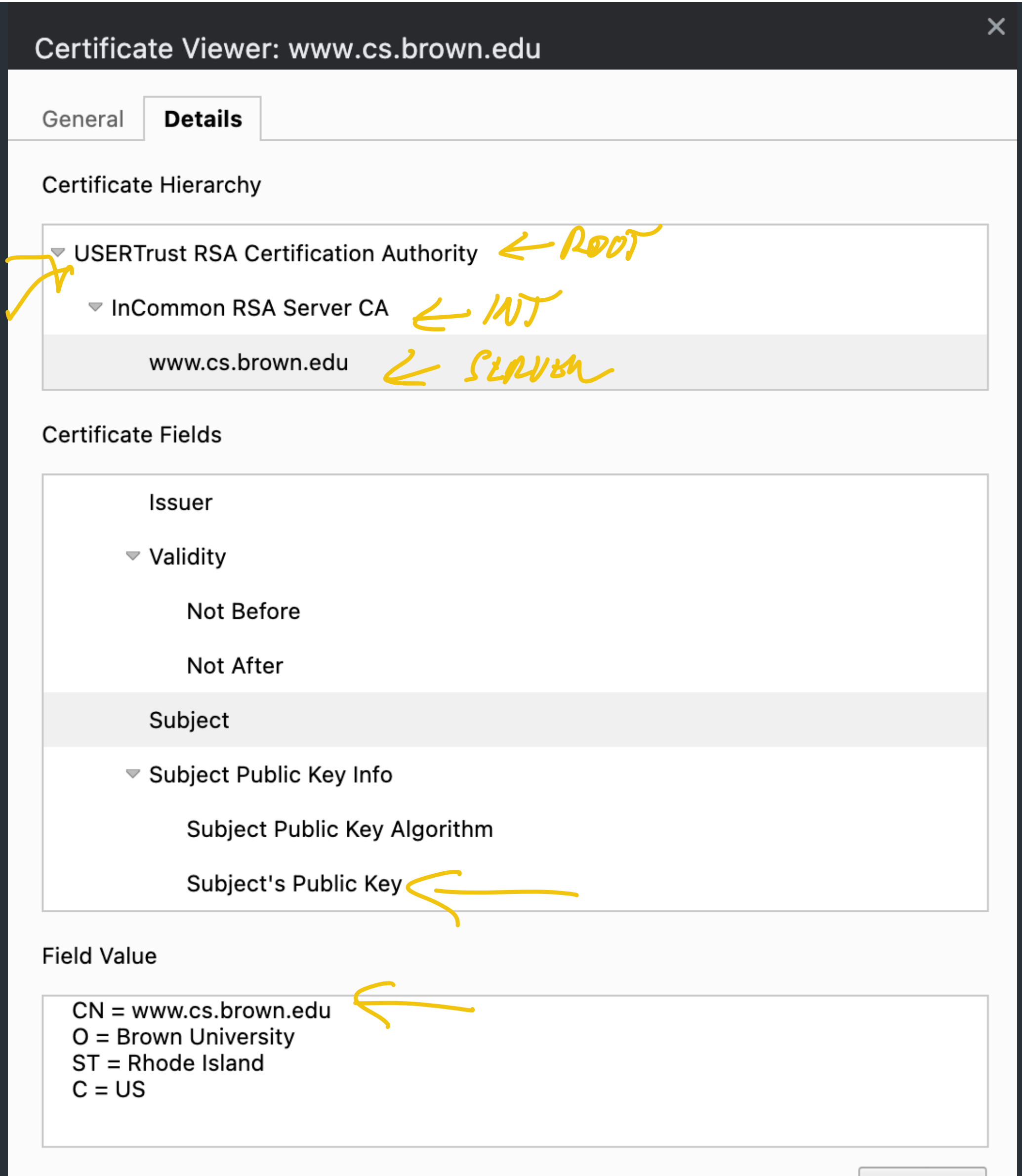
- Trust on first use (TOFU)
  - ON first connection, ask user if they trust the public key (y/n)
    - If user says yes, trust key for all time
    - If public key changes later, something sketchy is happening => trust error
    - => SSH (by default)

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

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
ROOT
InCommon RSA Server CA
INT
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
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
Issuer Name	
Country or Region	US
Organization	DigiCert Inc
Organizational Unit	www.digicert.com
Common Name	DigiCert Assured ID
Serial Number	0C E7 E0 E5 17 D8
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

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

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, PKI creates a hierarchy of trust:

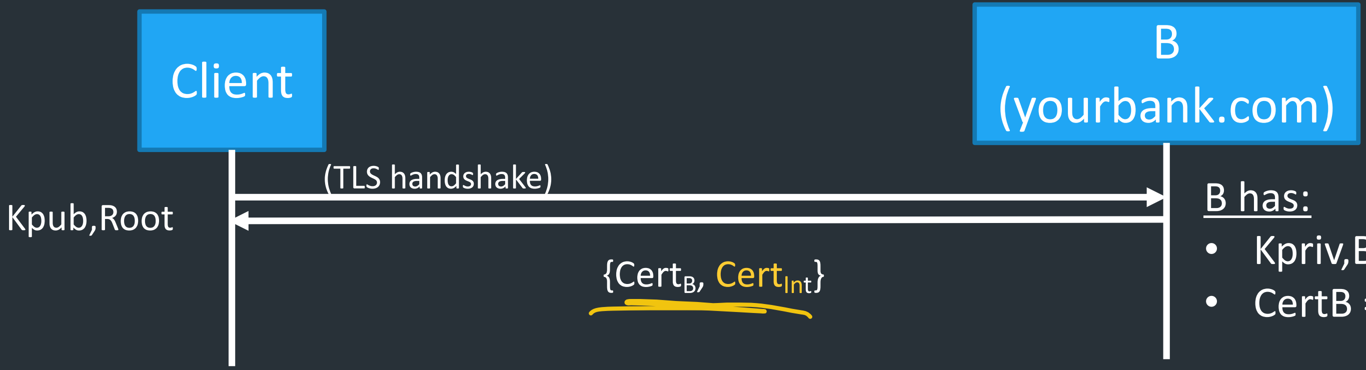
- Root CAs: k_{pub} stored in virtually every browser, OS
  - Private keys protected by most stringent security measures (software, hardware, physical)
- Intermediate CAs: k_{pub} signed by root CA
  - Sign certificates for general use (ie, regular websites)
  - Doesn't require same protections as root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

COULD SIGN ANY CERTIFICATE!

What happens if a root is compromised?

How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}

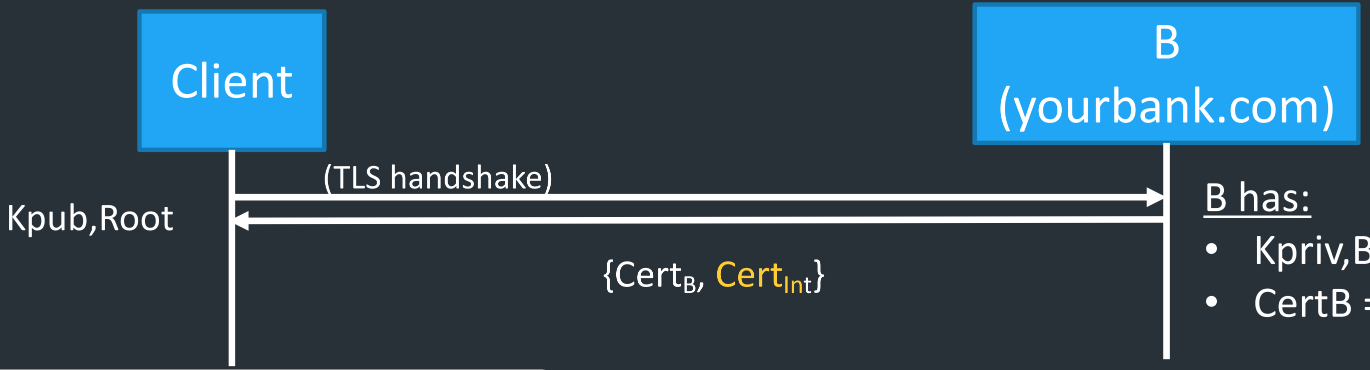


B has:

- 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:

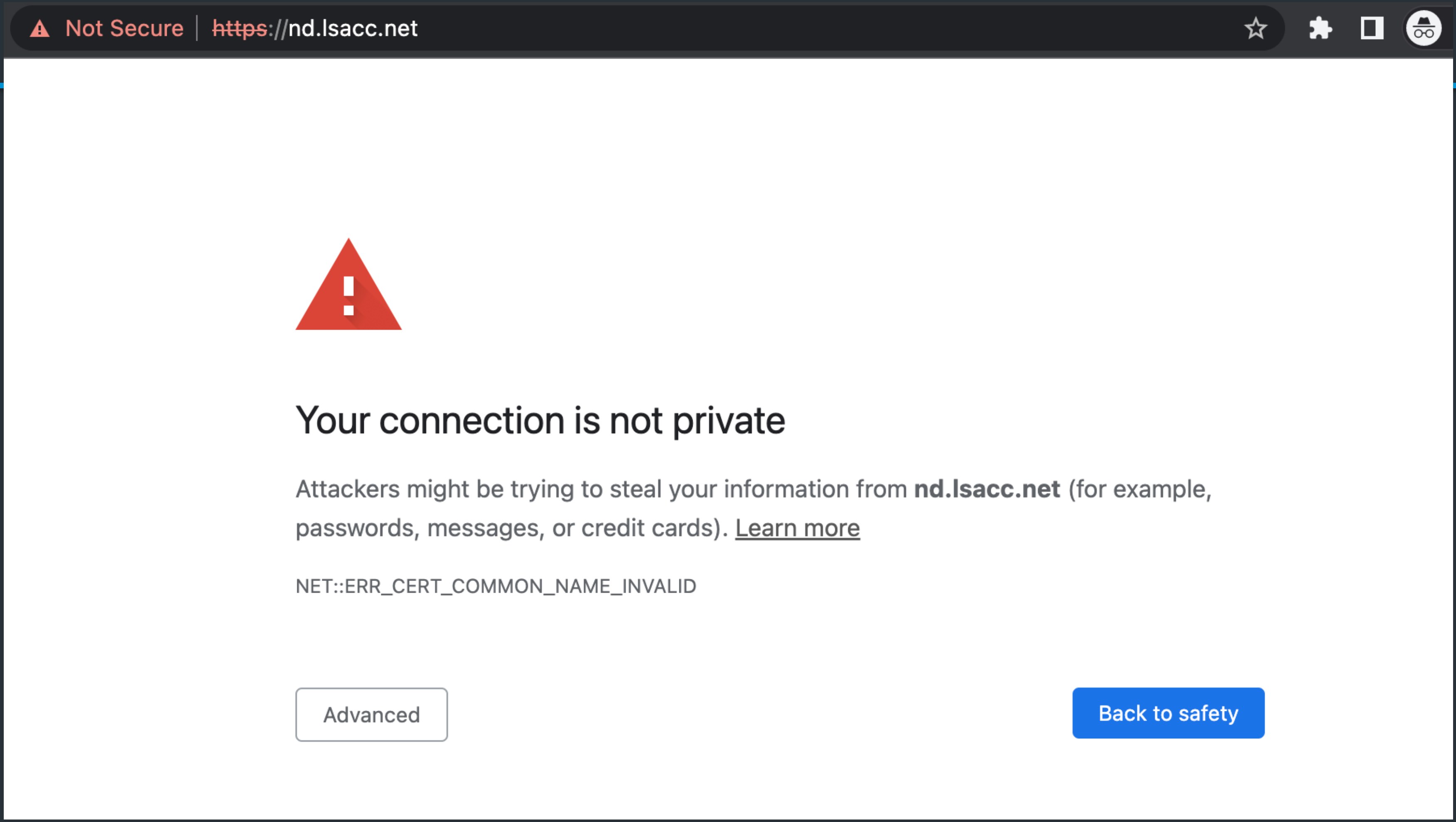
- Kpriv,B
- CertB = { $K_{pub,B}$, Sign($K_{pub,B}$, $K_{priv,Int}$), … }

Client's workflow:

- Checks metadata ✅ ← DATES, NAMES, . . .
- 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 invalid (NAME IN CERT ≠ DOMAIN NAME)
- 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 BUT NOT FOR YOUR BANK!! (OR BROWN)

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

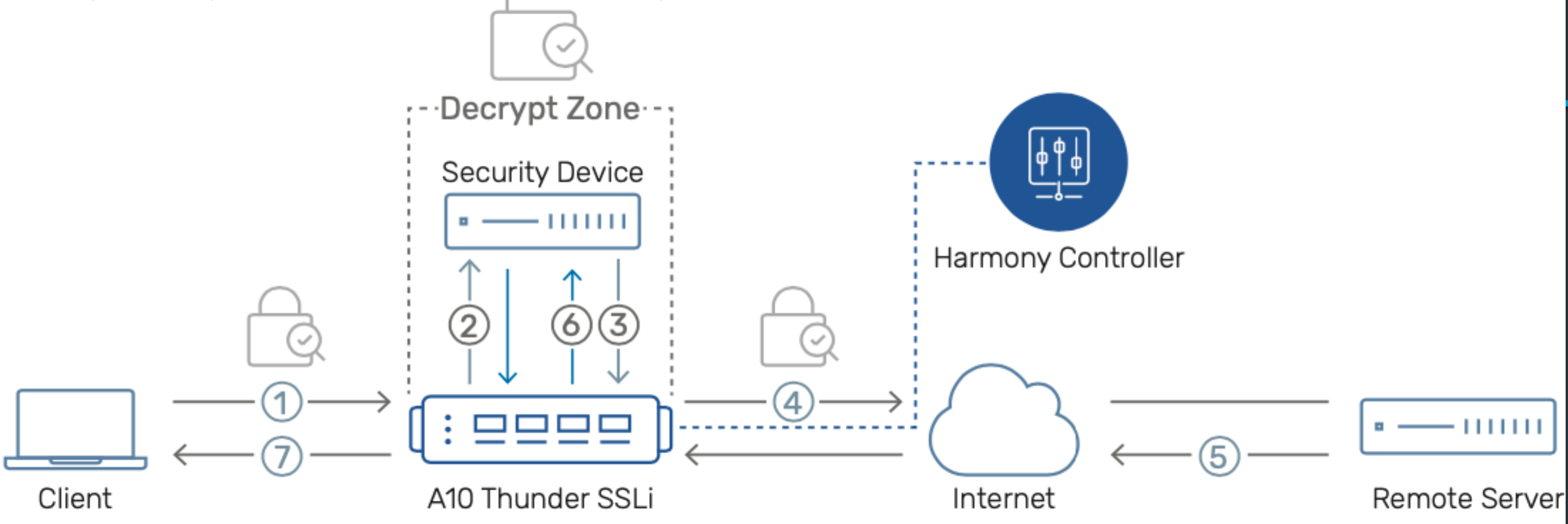
- In 2017, Google questioned the certificate issuance policies and practices of Symantec
- Google's Chrome would start distrusting Symantec's certificates unless certain remediation steps were taken
- See back and forth between Ryan Sleevi (Chromium team) and Symantec
- The matter was settled with DigiCert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

TLS decryption

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

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.

PKIs, TLS, and HTTPS

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/

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