EECS 388

Introduction to Computer Security

Lecture 9:

HTTPS and the Web PKI

September 26, 2023
Prof. Halderman

Web and Network Security

Last week:

- The Web Platform
- Web Attacks and Defenses

This week:

- **HTTPS and the Web PKI**
- HTTPS Attacks and Defenses

Next week:

- Networking 101
- Networking 102

Later:

- Network Defense
- User Authentication
- Online Privacy and Anonymity

Why Do We Need TLS?

Traditionally, HTTP (web), SMTP (email), and many other applications were carried over the Internet using **TCP**, a **plaintext transport protocol**.

TCP provides:

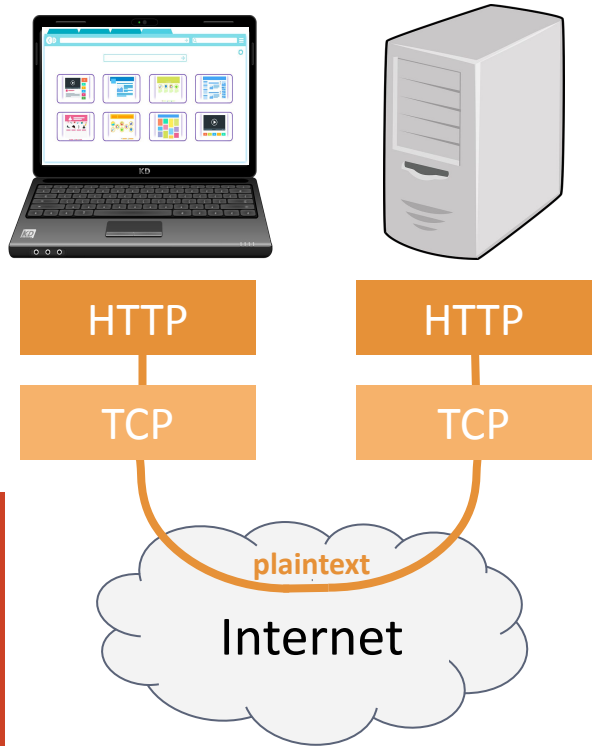
- "Phone call"-like semantics:
  dial, send/receive data stream, hang up

TCP **doesn't** provide:

- Confidentiality
- Integrity
- Authenticity

[Why is this a problem?]

The network is evil and wants to kill you!



TLS (Transport Layer Security)

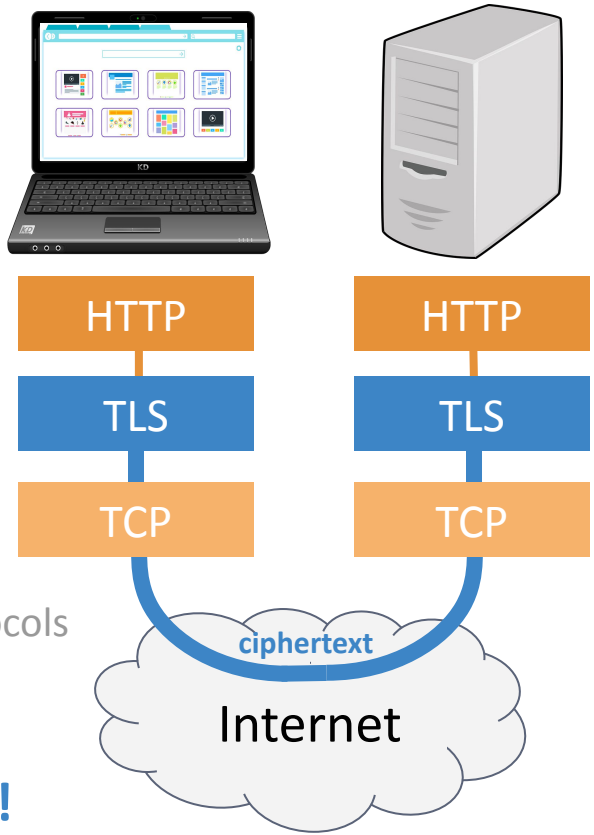
TLS (Transport Layer Security) is a cryptographic protocol that is layered above TCP to provide a **secure channel**.

Commonly used with many application protocols:

HTTP over TLS ⇨ HTTPS

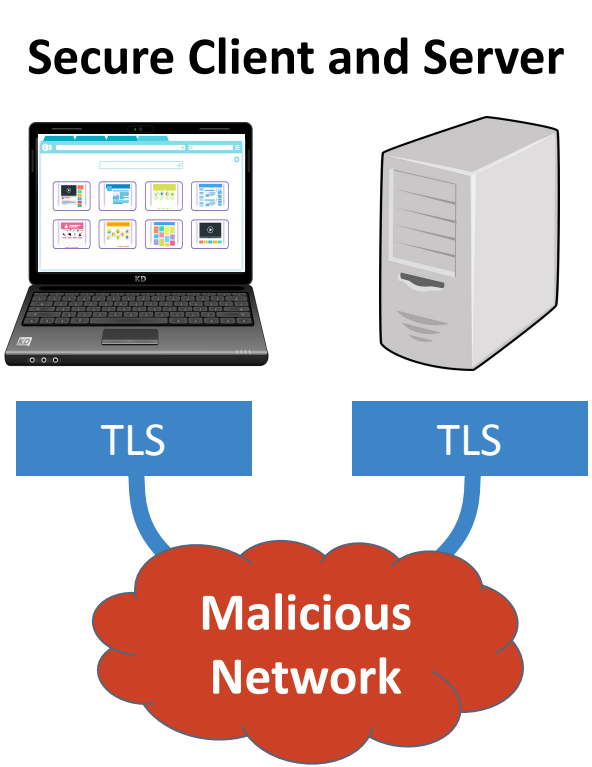
SMTP, RDP, FTP, XMPP, OpenVPN, and others

* SSH and Wireguard use their own, totally different, crypto protocols



High-quality TLS libraries are widely available.
If your program sends data over the Internet, use TLS!

TLS Threat Model: Malicious Networks



TLS assumes client and server are secure, but talking over a **malicious network**.

Two common models of **network adversaries**:

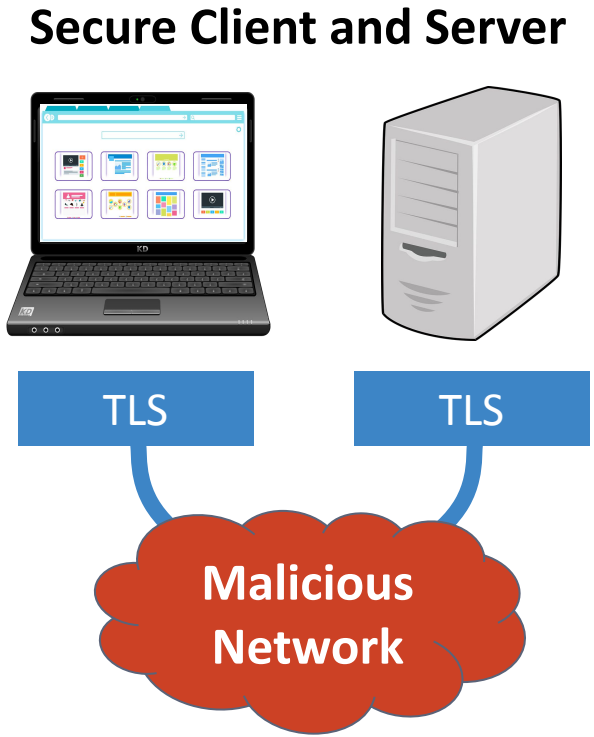
Passive: only eavesdrops
Active: can see, inject, modify, or block

All Internet traffic faces these threats. Examples:

- Government surveillance and censorship
- ISPs harvesting data for tracking, injecting ads
- Compromised routers, WiFi APs, DNS servers
- ARP spoofing, BGP route hijacking

(More about all of these later in the course)

TLS Benefits and Limitations



TLS provides:

- **Confidentiality** and **integrity protection** for application data while in transit.
- Client can **authenticate server's identity**. [Why is this important?]

TLS <u>does not</u> protect against: (for example)

- Malware/intruder on the client/server
- Vulnerabilities in application software
- Phishing, social engineering
- Tracking by the sites you visit
- Metadata analysis (who talked to whom when, for how long)
- Denial of service

TLS Protocol History

Modern TLS is the product of >25 years of design and analysis

Older versions are vulnerable to known attacks and unsafe

SSL (Secure Sockets Layer) – Netscape, proprietary protocol

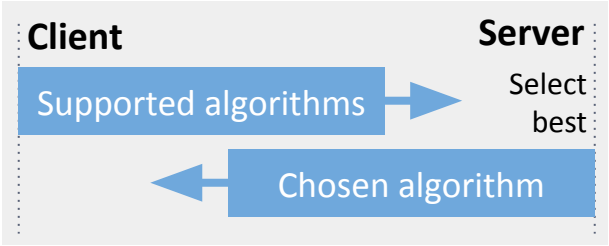
SSL 1.0 (1994): Completely broken, never published
SSL 2.0 (1995): Completely broken, deprecated in 2011
SSL 3.0 (1996): Completely broken, deprecated in 2015; foundation for TLS 1.0

TLS (Transport Layer Security) – IETF standard

TLS 1.0 (1999): Vulnerable, deprecated in 2020
TLS 1.1 (2006): Vulnerable, deprecated in 2020
TLS 1.2 (2008): Still widely used, but dated, complex, and hard to implement securely
TLS 1.3 (2018): **Redesign with major improvements, our focus today (RFC 8446)**

TLS Handshake Components

Negotiate Crypto Algorithms



Find best mutually supported:

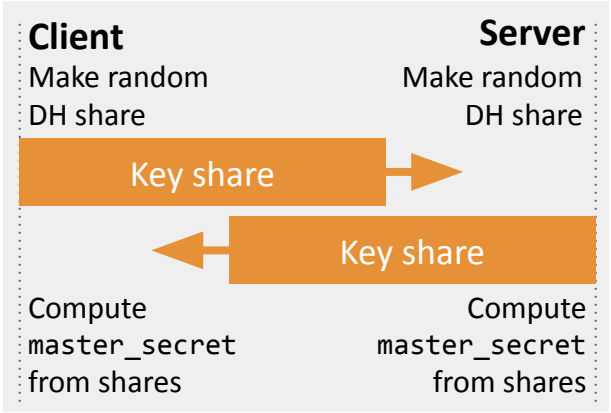
Key exchange algorithm
e.g., EC DHE w/ Curve25519

Signature algorithm
e.g., RSA w/ SHA-256, PKCS #1 pad

Symmetric crypto algorithm
e.g., AES-128 GCM

Why negotiate?

Establish Shared Secret



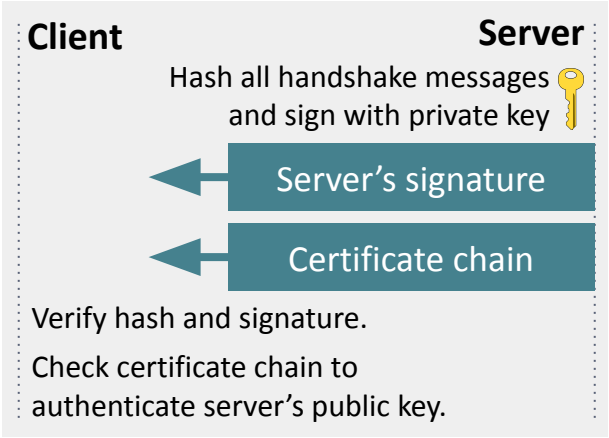
Mutually compute a **shared secret**.

Diffie-Hellman for **forward secrecy**.

Derive symmetric keys from shared secret and encrypt and integrity-check all further data.

Is it the real server or an active attacker?

Authenticate the Server



Server signs, and client verifies, a hash of the entire **handshake transcript** to this point.

How does client authenticate the server's public key?

(Find out in a few slides)

TLS Protocol

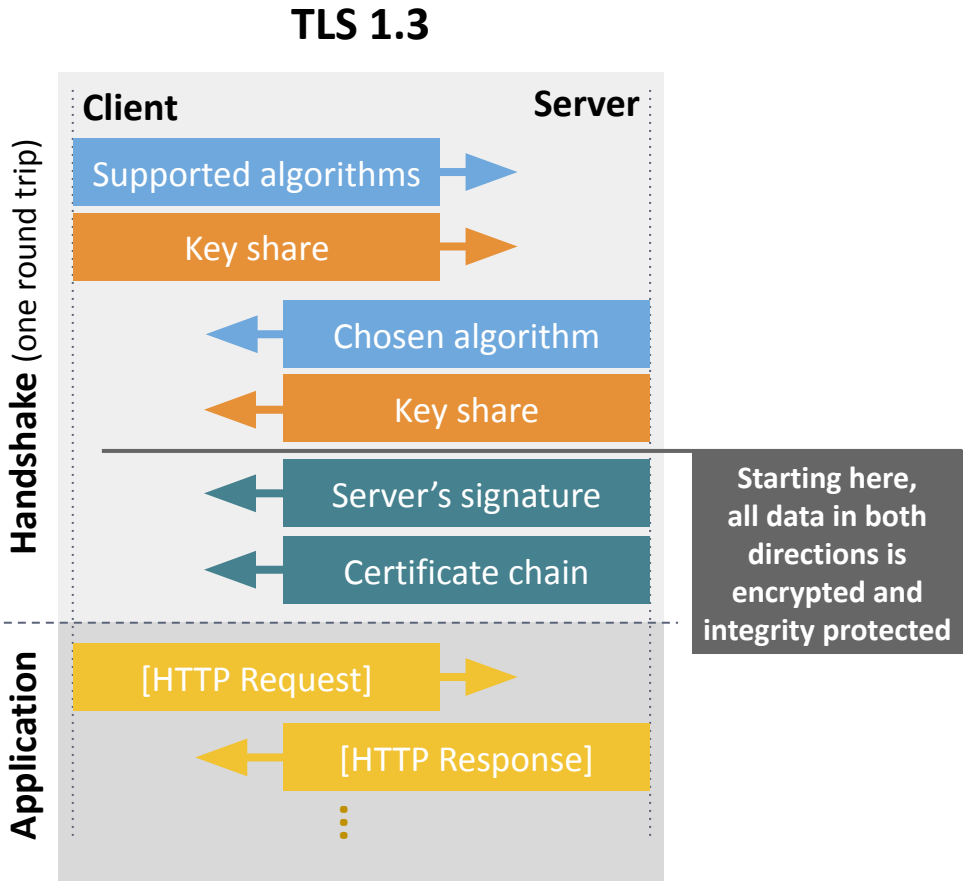
A **network round trip** takes ≈100 ms.

To **minimize latency**, TLS 1.3 handshake works in one round trip.

Clever design: Client *guesses* which key exchange algorithm the server will pick. (If guessed wrong, must add another round-trip to send a new key share.)

Make sure you understand how TLS 1.3 achieves these properties:

- ☐ **Confidentiality** **(AEAD ciphers)**
- ☐ **Message Integrity** **(AEAD ciphers)**
- ☐ **Authentication** of server by client
  (Public key crypto and certificates)



X.509 Certificates

How does client obtain server's public key?

Server presents a **certificate**: a message asserting the server's identity and its public key, signed by a **certificate authority (CA)**.

A CA is an entity trusted by clients to **verify server identities** and issue certificates.

Each major platform and browser includes a set of public keys for the CAs that it *trusts*, called **root CAs**. Clients use these keys to verify certificates.

TLS uses the **X.509 certificate format** (specifies data structure, encoding, etc.)

X.509 Certificate Example

Subject: **eecs388.org**
Issuing CA: Let's Encrypt Authority
Validity: 2023-08-09 to 2023-11-07 (90 days)
Public Key: 2048-bit RSA

```
00:ab:c7:1b:0c:ed:c6:01:f8:ea:a9:b3:cf:08:17:
4f:a2:cb:7c:34:c4:66:12:e6:ef:f3:98:17:79:c9:
65:ee:66:4c:1f:9a:92:7d:33:ee:07:fa:2e:15:62:
f7:b4:f3:1f:d5:4f:2e:b1:67:a8:49:42:bf:e3:cc:
9a:b7:30:46:c2:68:f5:28:a9:64:69:6f:4c:4b:...
```

CA's digital signature on the message above:

```
7d:d4:f1:b2:49:f2:99:ea:e9:3d:b5:f8:24:e6:78:
fb:35:0a:69:a4:6a:fa:8e:1b:5e:6d:38:dd:aa:fb:
7d:24:18:a8:a9:47:77:92:dc:31:d0:77:33:71:...
```

Can include multiple domains or wildcards (*.umich.edu)

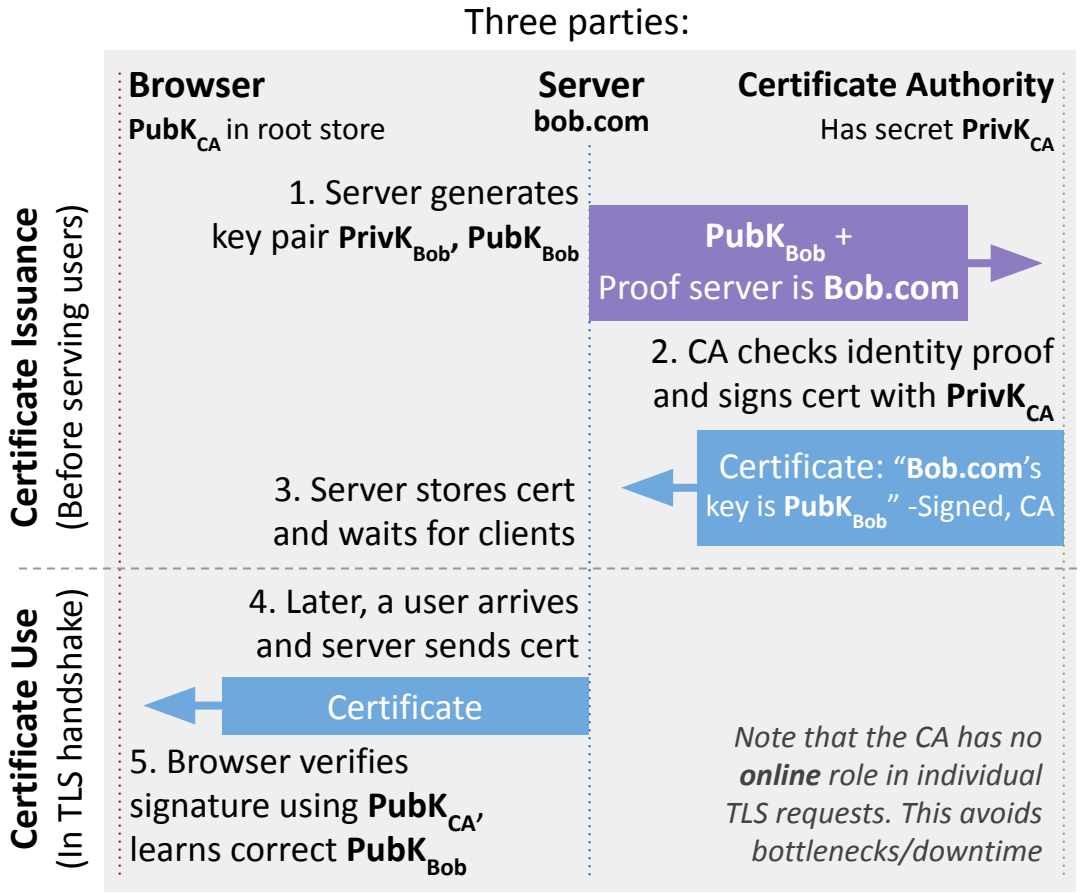
Obtaining a Certificate

To obtain a certificate, server must **prove its identity** to the CA.

Common verification methods:

- **Email:** Receive confirmation code at an email address specified when the domain was registered.
- **HTTP:** Place file with confirmation code at specified URL on domain.
- **DNS:** Add record containing conf. code to domain's DNS zone.

ACME (RFC 8555), an open protocol created by Let's Encrypt, automates these processes (Automatic Certificate Management Environment).



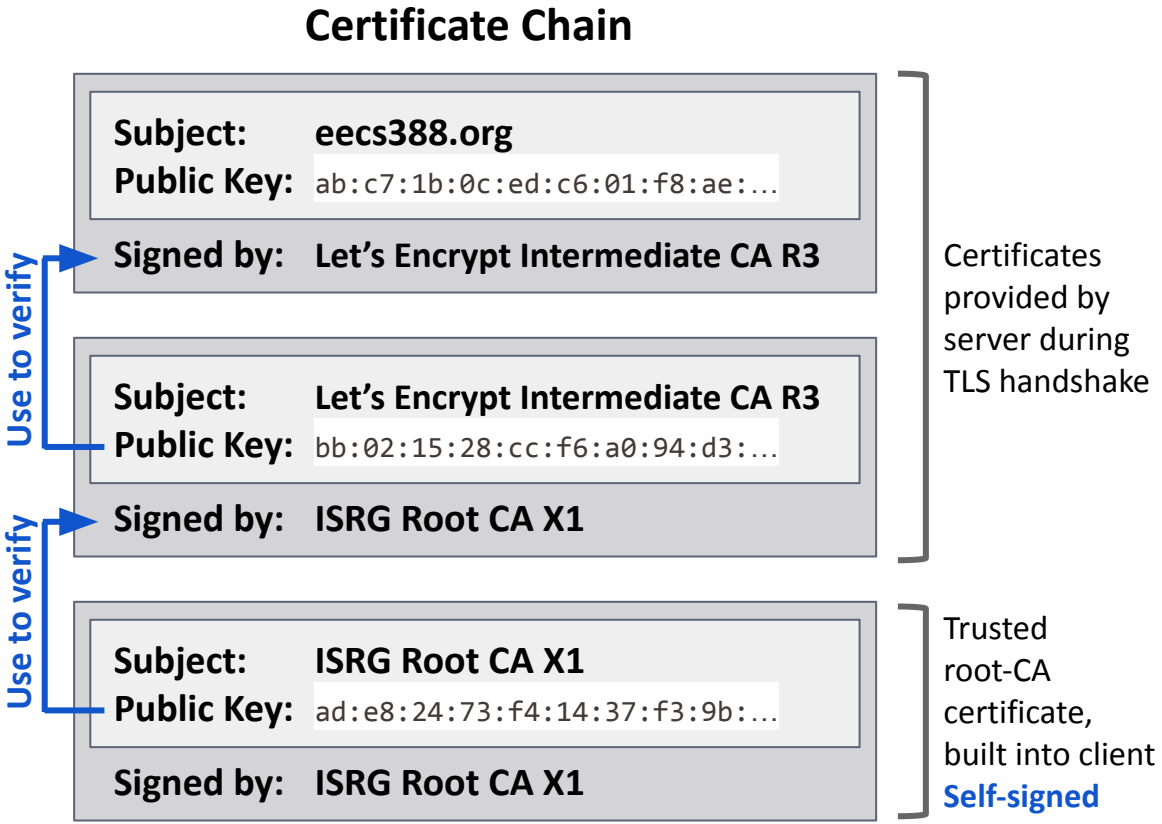
Certificate Chains

CAs sometimes issue **intermediate CA certificates**, which lend permission to sign further certificates.

Used to:

- Delegate trust to other CAs
- Use separate key for issuing certs from long-term root key stored offline [Why?]

Servers provide a **certificate chain**. Client verifies signature in each link of the chain, back to a trusted root CA certificate the client trusts



HTTPS Certificate Ecosystem

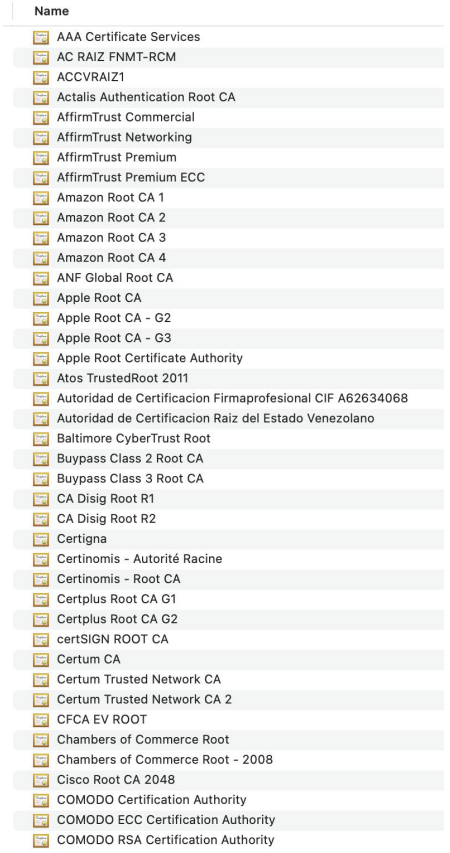
The Web's **public key infrastructure (PKI)** is operated by a community that includes:

- **Certificate authorities**
  Verify identities, issue certificates, manage revocation
- **Browser/platform developers**
  Implement cert. validation and UI; trust or distrust CAs
- **CA/Browser Forum**
  Consortium that sets industry-wide issuance policies

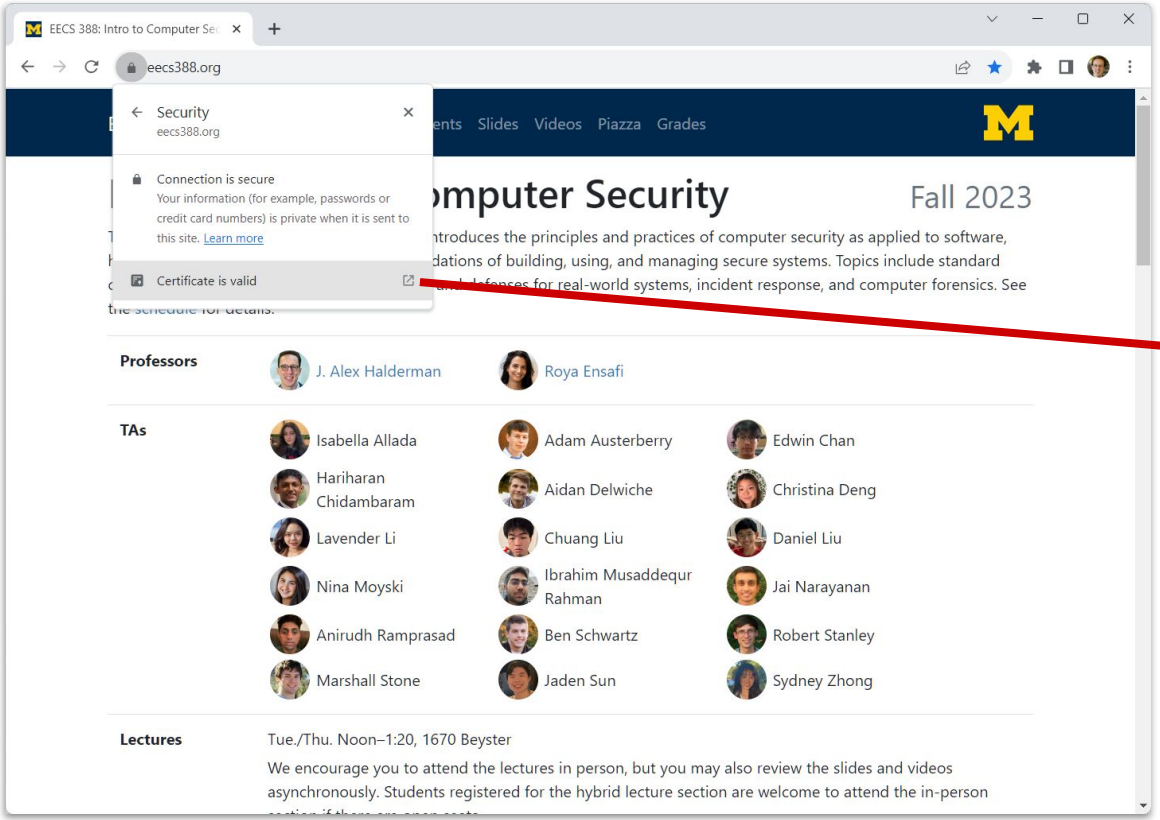
Each platform trusts 100+ root CAs

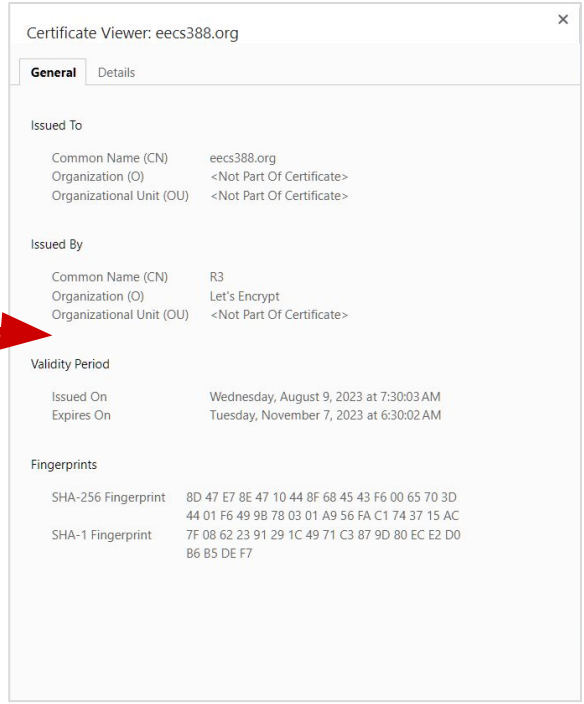
There are also 1000s of implicitly trusted intermediate CAs

CAs have issued billions of certs



Try It: View a Site's Certificate





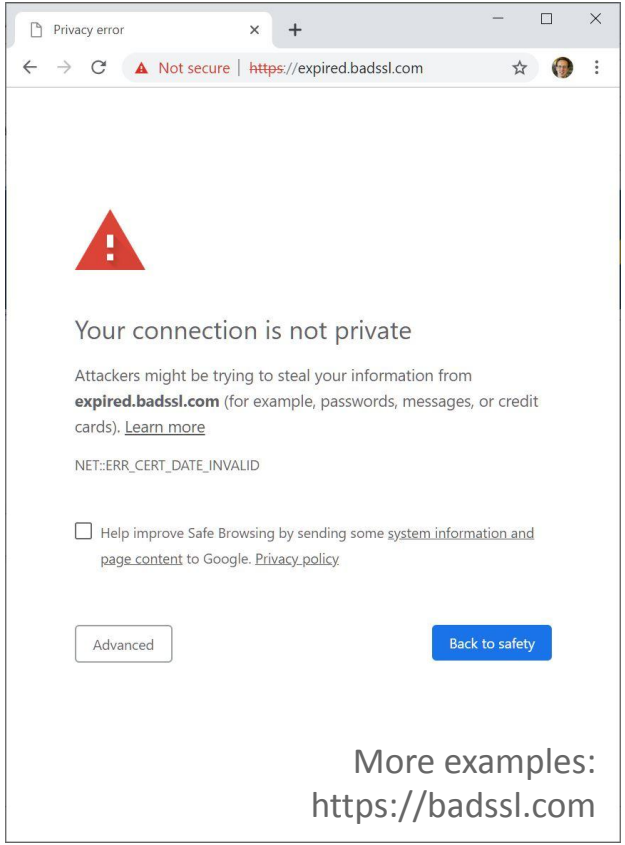
Usability: Certificate Warnings

Browser will show a **certificate warning** if:

- Certificate has expired
- Domain in URL bar doesn't match any of the domains in the cert.
- Cert. chain doesn't lead to a trusted root CA
- CA has **revoked** the certificate

Most warnings are due to expiration or other misconfiguration, but cause could be a network attack,
so browsers make warnings scary/hard to bypass.

Automation (e.g., ACME) can help servers avoid problems that lead to warnings.



HTTPS is Becoming Ubiquitous

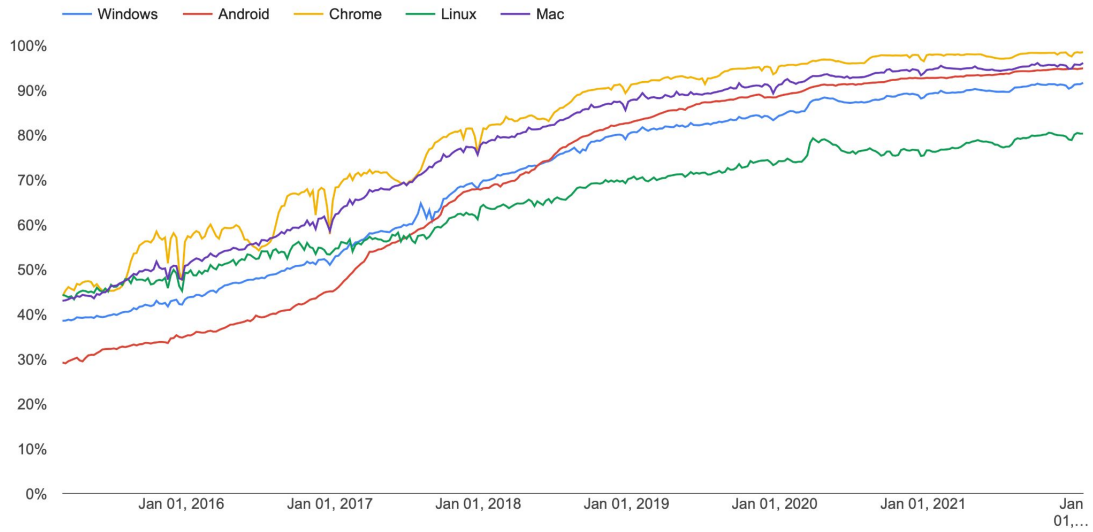
Until recent years, HTTPS was relatively uncommon.

Today it's nearly ubiquitous.

Let's Encrypt provides free certs and is integrated with many servers and hosting providers.

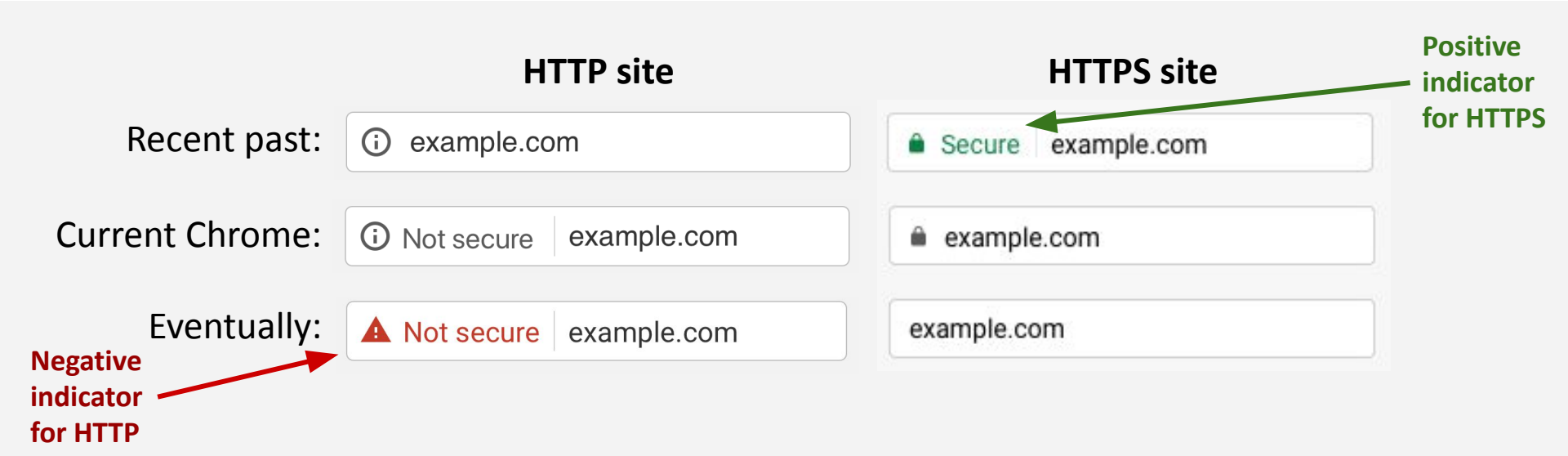
Google gives HTTPS sites higher rank in search results.

Many recent browser APIs are available only for HTTPS sites.



Percentage of pages loaded over HTTPS in Chrome

Usability: Security Indicators



In the past, browsers used **positive security indicators** (lock icon) for HTTPS.

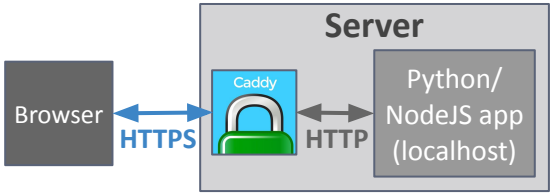
Usability experiments show that users **fail to notice** when the lock icon is missing.

With HTTPS growth, browsers switching to **negative indicators** (warnings) for HTTP.

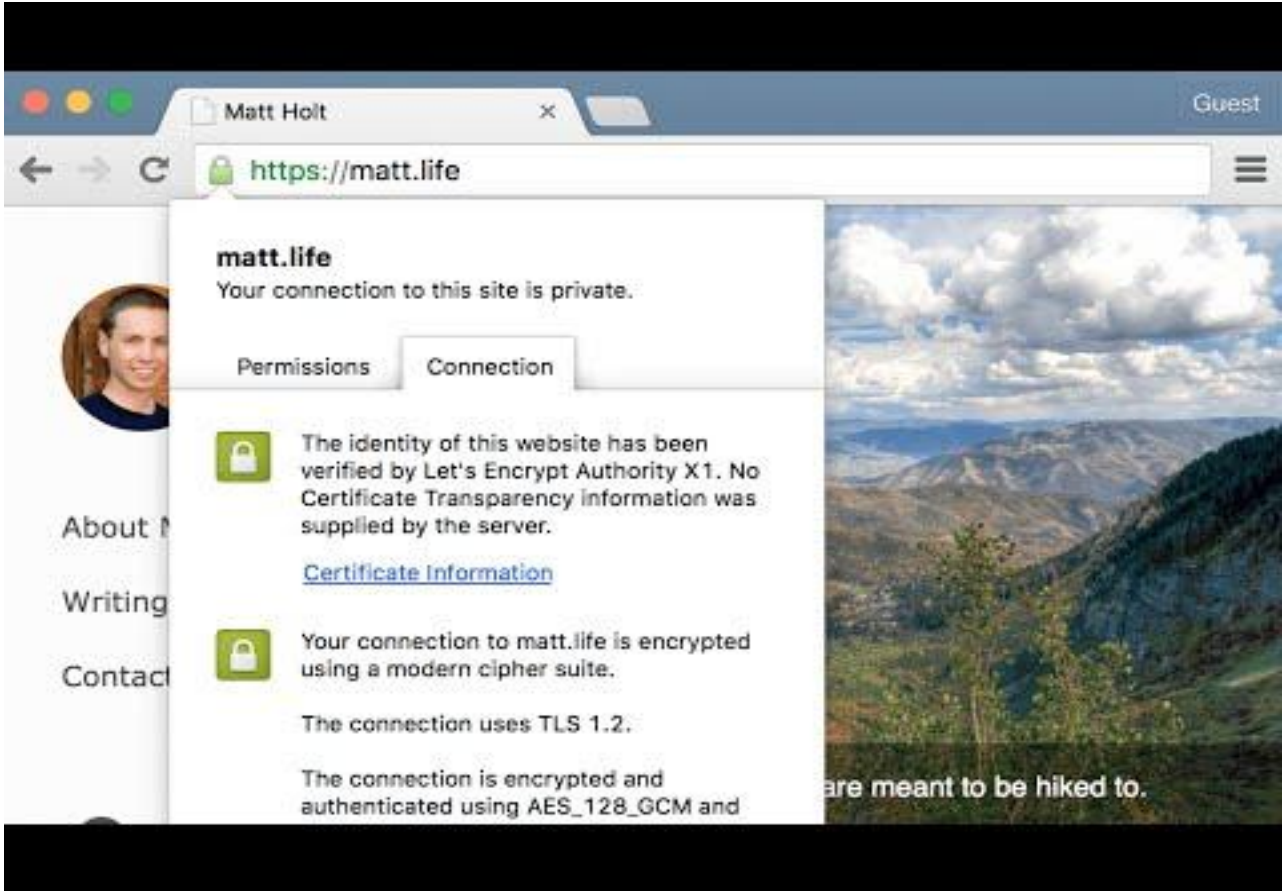
Try It: Setting Up HTTPS

Enabling HTTPS with **Caddy** and **Let's Encrypt**

Caddy can also act as an **HTTPS front-end** for web apps you write in other languages:



https://caddyserver.com



Further Reading

Let's Encrypt: How It Works

https://letsencrypt.org/how-it-works/

The New Illustrated TLS Connection

Every byte explained and reproduced

https://tls13.xargs.org/

RFC 8446

The Transport Layer Security (TLS) Protocol Version 1.3

https://tools.ietf.org/html/rfc8446

Coming Up

Reminders:

Lab Assignment 2 due this Thursday at 6 PM

Project 2 due next Thursday, October 5, at 6 PM

Midterm Exam is Friday, October 20, 7-8:30 PM

Thursday

Attacking HTTPS

Implementation flaws,
social engineering attacks,
cryptographic failures

Next Week

Networking

Networking 101
Network Attacks and Defenses