EECS 388

Introduction to Computer Security

Lecture 9:

HTTPS and the Web PKI

September 24, 2024

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:

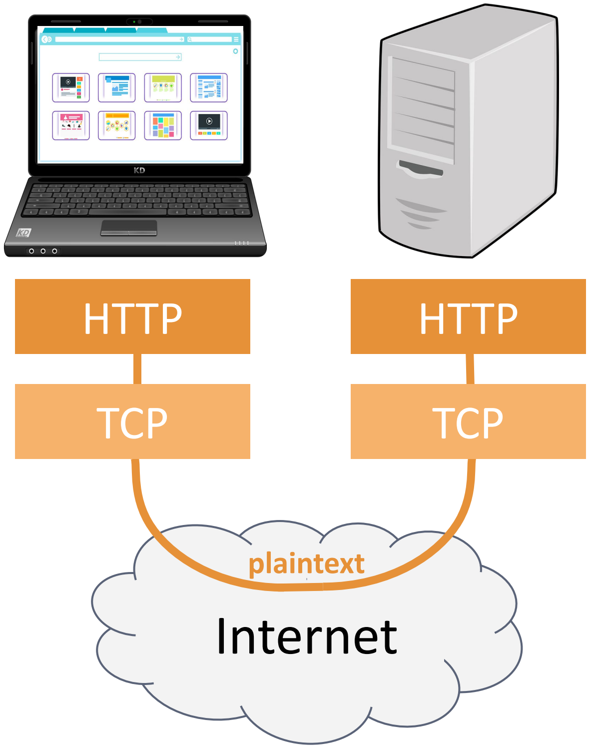
- "Phone call"-like semantics:
  dial, send/receive data stream, hang up

TCP **doesn't** provide:

- Confidentiality
- Integrity
- Authenticity

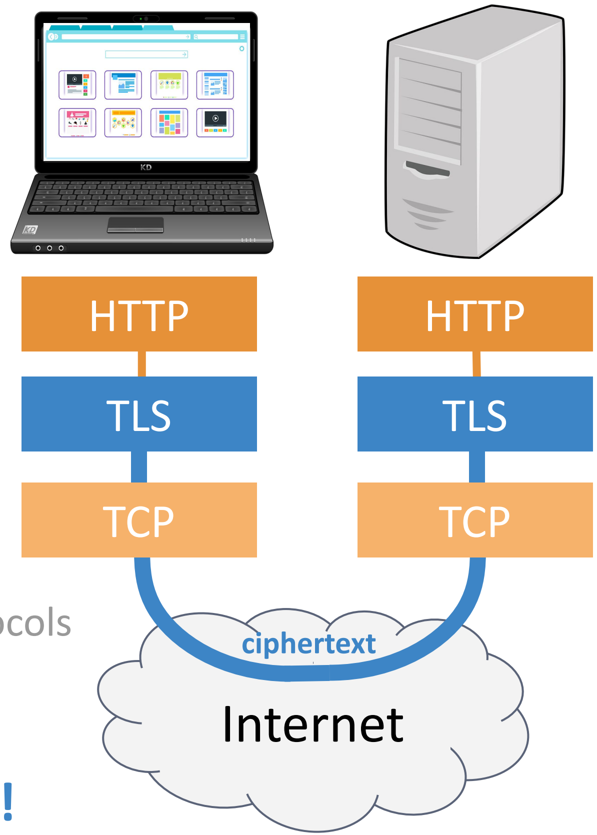
[Why is this a problem?]

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:

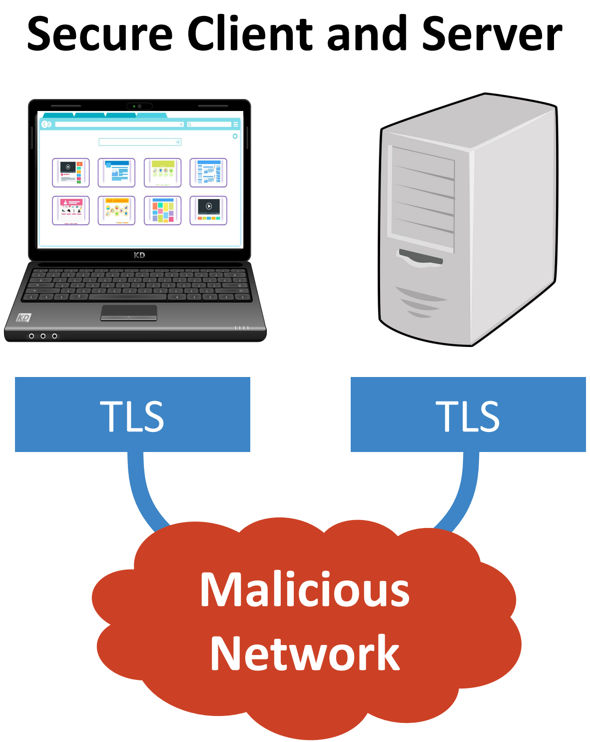
HTTP over TLS ⇨ HTTPS

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

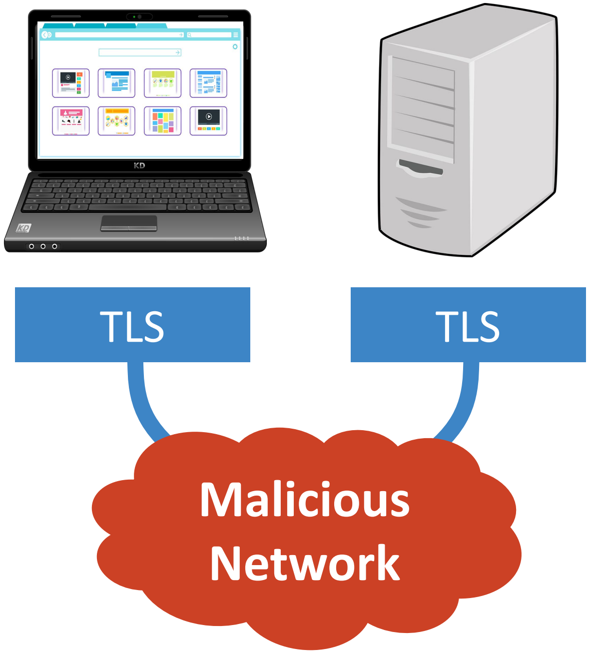
All Internet traffic faces these threats. Examples:

- 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

Secure Client and Server



TLS provides:

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

TLS *does not* 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 >30 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 sometimes used, but dated, complex, and difficult to secure
TLS 1.3 (2018): **Redesign with major improvements, our focus today (RFC 8446)**

TLS Handshake Components

Negotiate Crypto Algorithms

Client — **Server**

Supported algorithms →

Select best

← Chosen algorithm

Find best mutually supported:

Key exchange algorithm
e.g., EC DH 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

Client — **Server**

Client: Make random DH share

Server: Make random DH share

Key share →

← Key share

Client: Compute `master_secret` from shares

Server: Compute `master_secret` from shares

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

Client — **Server**

Server: Hash all handshake messages and sign with private key

← Server's signature

← Certificate chain

Client: Verify hash and signature

Check certificate chain to authenticate server's public key

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.

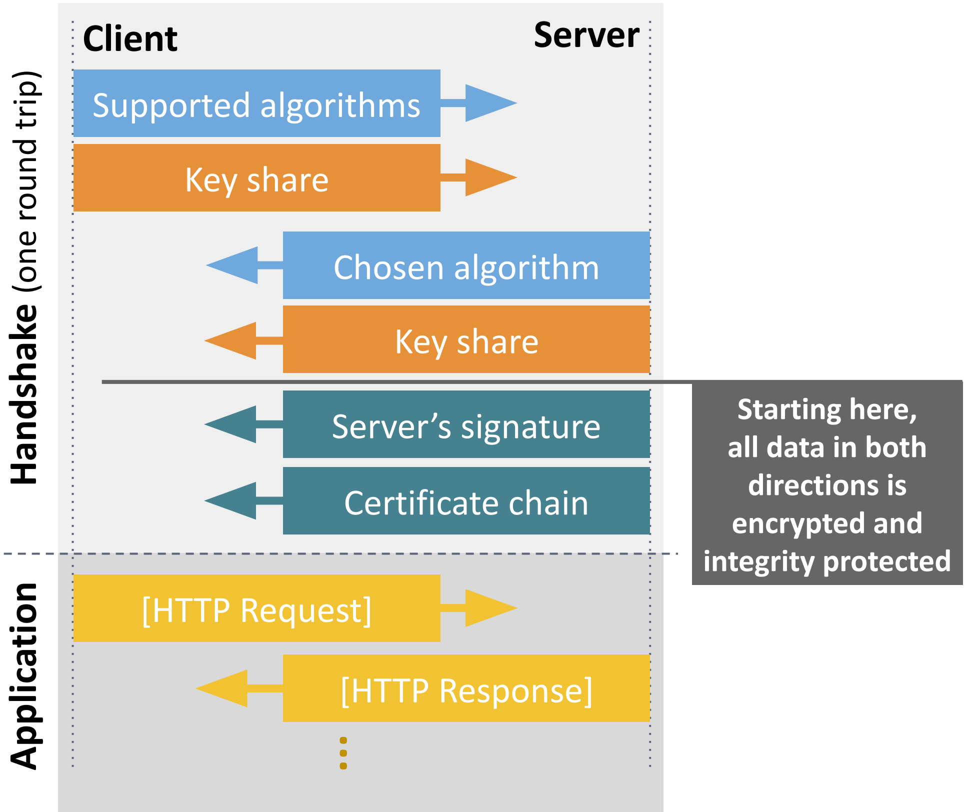
To **minimize latency**, TLS 1.3 handshake works in one round trip.

> **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)**

TLS 1.3



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: 2024-08-27 to 2024-11-25 (90 days)
Public Key: 2048-bit RSA

```
00:98:98:58:eb:ec:cb:b6:77:81:e8:70:0e:87:22:
31:ef:d2:63:63:67:01:9c:90:4e:10:16:94:9c:f5:
19:b6:05:30:56:b6:82:41:62:d4:31:0b:79:c0:d4:
e1:c1:36:13:1f:5c:70:16:21:d0:1c:53:13:8c:3c:
0c:8c:5d:15:47:f8:c7:94:29:41:8f:c2:e3:b2:29...
```

CA's digital signature on the message above:

```
45:1a:5c:2c:ed:d7:52:f1:4e:c0:95:ea:db:c1:91:
d6:62:aa:ad:76:d3:f1:0d:bb:21:3b:95:3a:22:84:
73:5d:1f:e5:2f:61:68:fa:b6:60:f3:d0:4d:2c:59...
```

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

Obtaining a Certificate

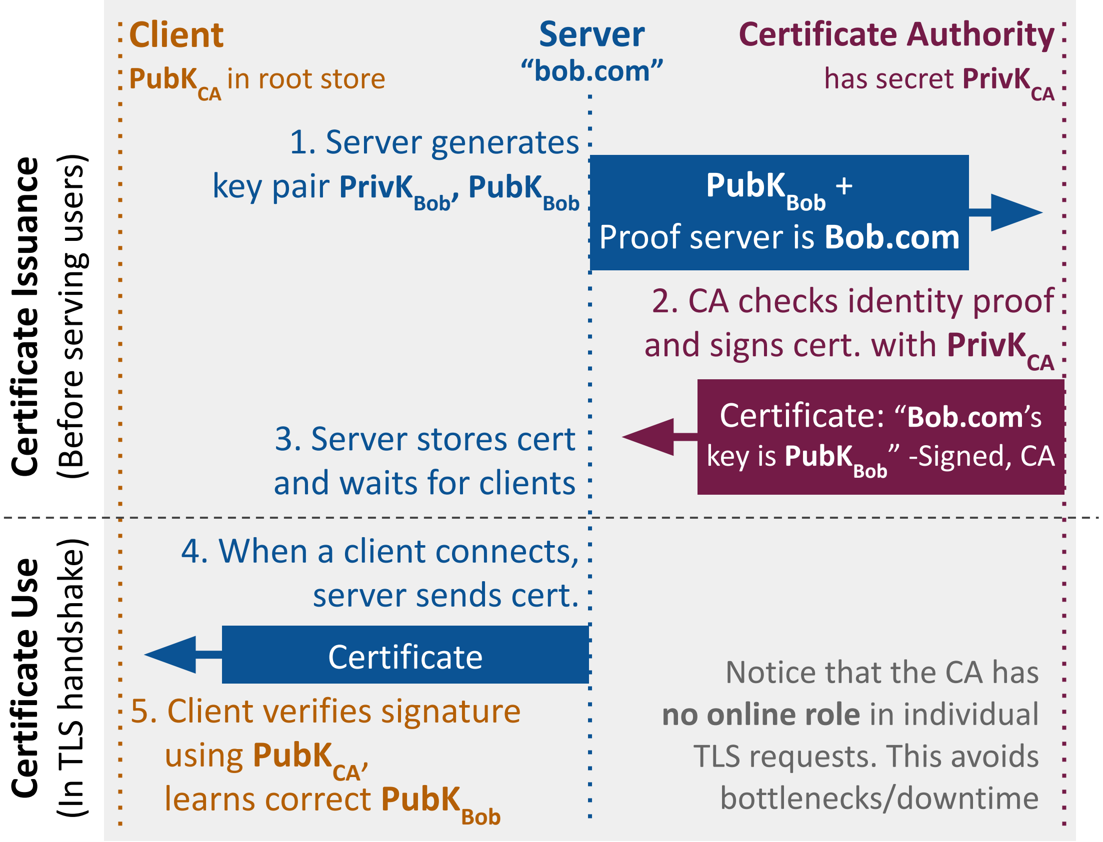
To obtain a certificate, server must **prove its identity** to the CA

Common verification methods:

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

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

Three parties:



Certificate Chains

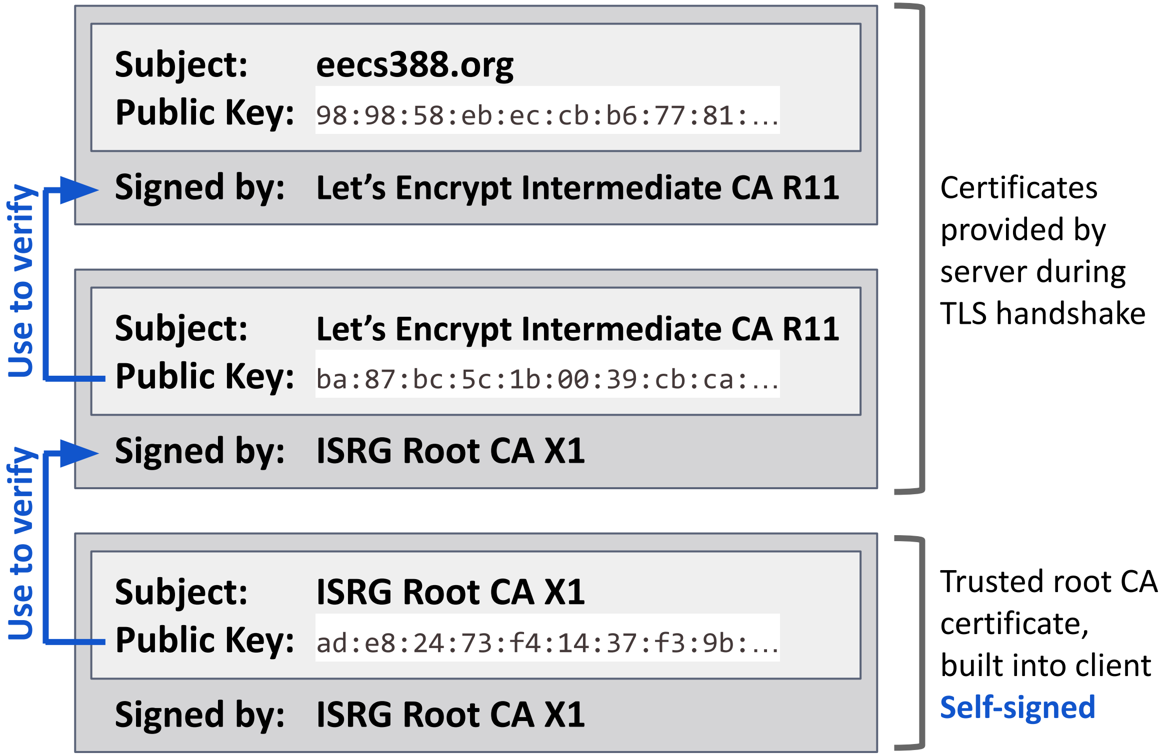
CAs sometimes issue **intermediate CA certificates**, which lend permission to sign further certificates

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

Certificate Chain



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

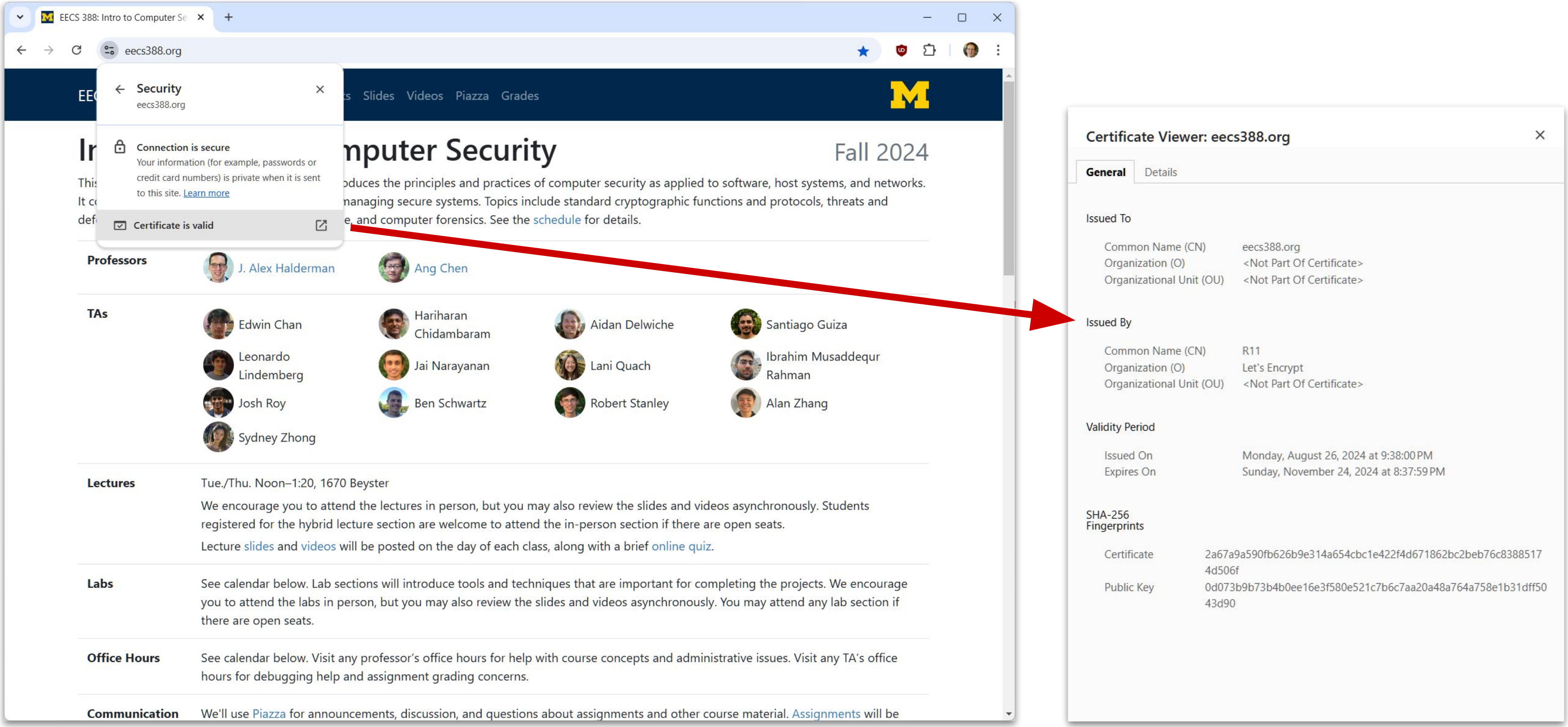
Each platform trusts 100s of root CAs

There are also 1000s of implicitly trusted intermediate CAs

CAs have issued billions of certs

Name
AAA Certificate Services
AC RAIZ FNMT-RCM
ACCVRAIZ1
Actalis Authentication Root CA
AffirmTrust Commercial
AffirmTrust Networking
AffirmTrust Premium
AffirmTrust Premium ECC
Amazon Root CA 1
Amazon Root CA 2
Amazon Root CA 3
Amazon Root CA 4
ANF Global Root CA
Apple Root CA
Apple Root CA - G2
Apple Root CA - G3
Apple Root Certificate Authority
Atos TrustedRoot 2011
Autoridad de Certificacion Firmaprofesional CIF A62634068
Autoridad de Certificacion Raiz del Estado Venezolano
Baltimore CyberTrust Root
Buypass Class 2 Root CA
Buypass Class 3 Root CA
CA Disig Root R1
CA Disig Root R2
Certigna
Certinomis - Autorité Racine
Certinomis - Root CA
Certplus Root CA G1
Certplus Root CA G2
certSIGN ROOT CA
Certum CA
Certum Trusted Network CA
Certum Trusted Network CA 2
CFCA EV ROOT
Chambers of Commerce Root
Chambers of Commerce Root - 2008
Cisco Root CA 2048
COMODO Certification Authority
COMODO ECC Certification Authority
COMODO RSA Certification Authority

Try It: View a Site’s Certificate



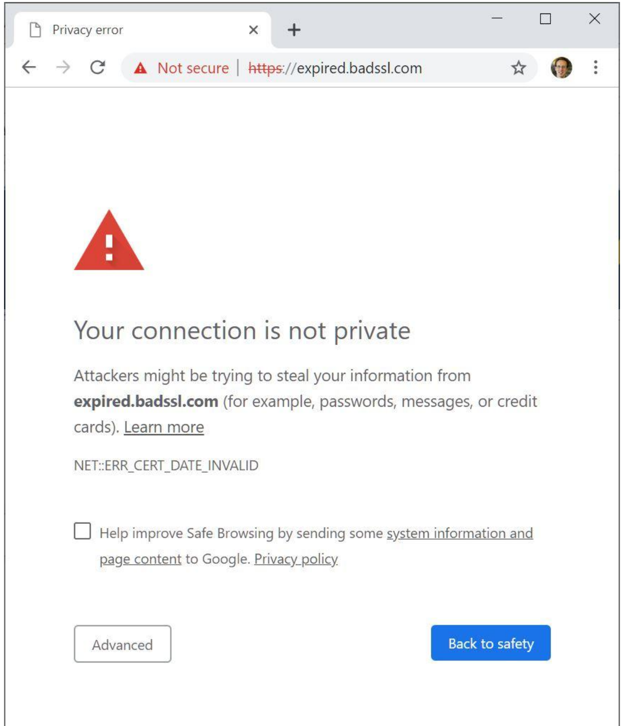
Usability: Certificate Warnings

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 cert.

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



More examples:
https://badssl.com

HTTPS is Becoming Ubiquitous

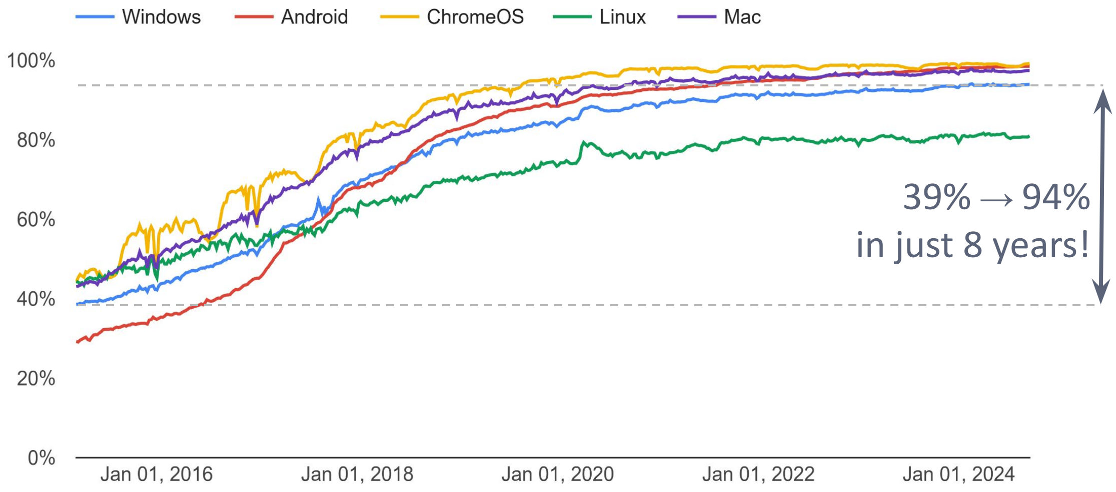
Until recent years, HTTPS was relatively uncommon.

Today it's nearly ubiquitous

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

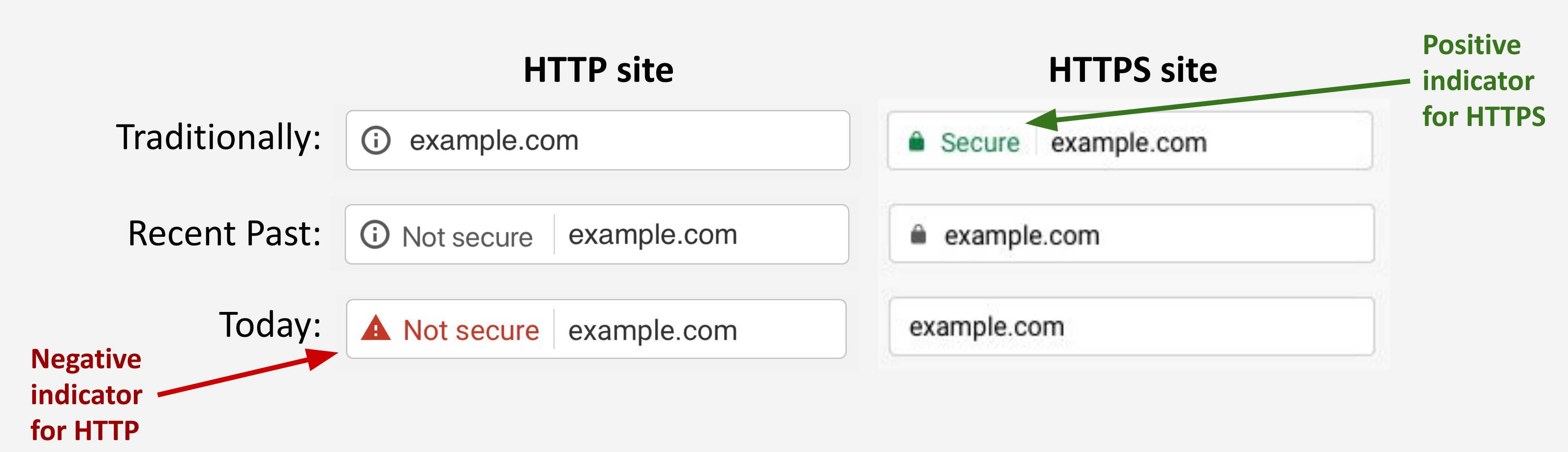
Google gives HTTPS sites higher rank in search results

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.

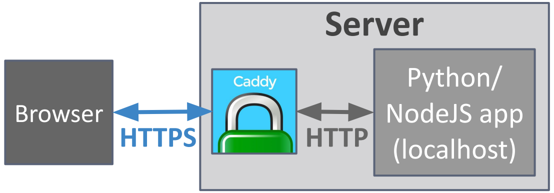
Usability experiments showed users **failed to notice** when the lock icon was missing.

Most modern browsers have switched to negative indicators (warnings) for HTTP.

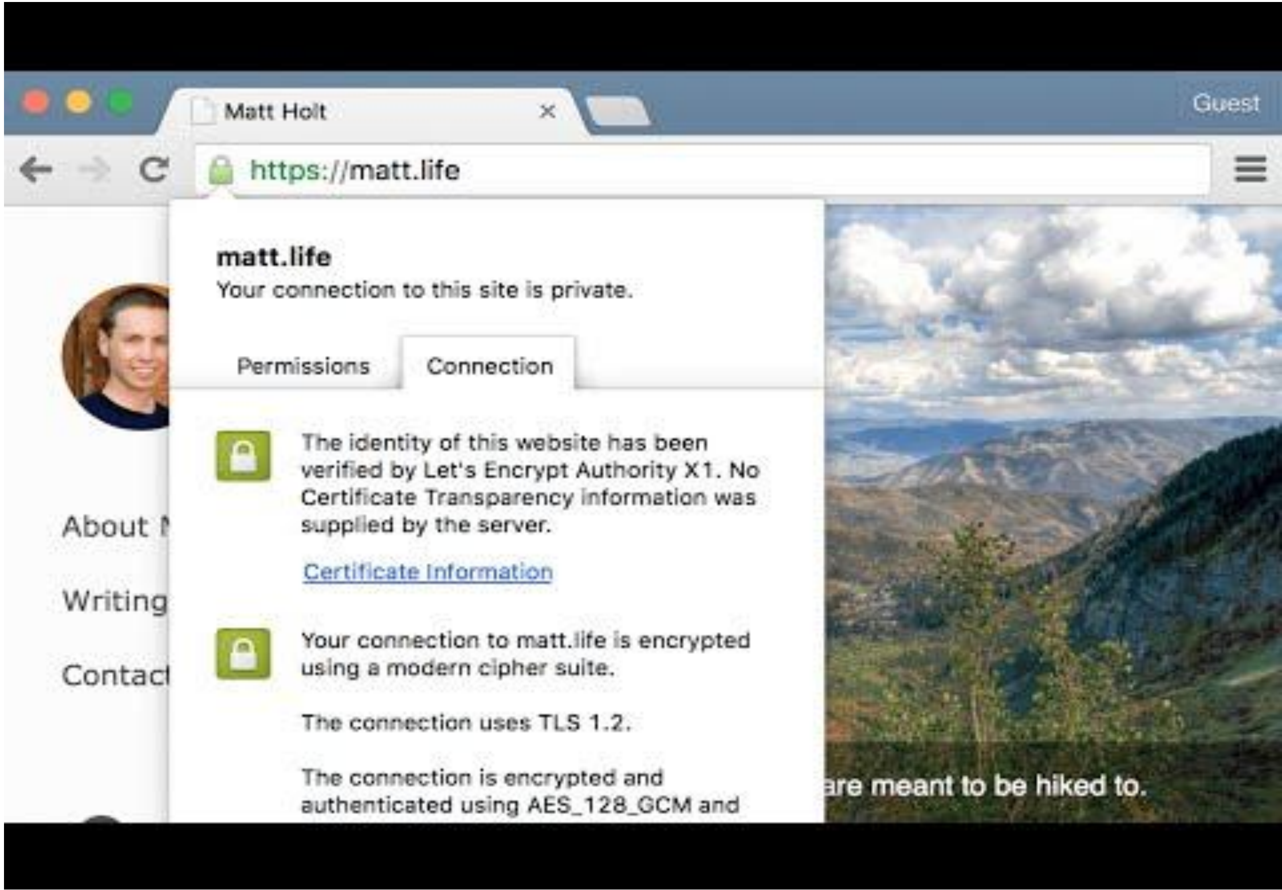
Try It: Setting Up HTTPS

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 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 week, Thursday, October 3, at 6 PM

Midterm Exam is Friday, October 18, 7-8:30 PM

Thursday

Attacking HTTPS

Implementation flaws,
social engineering attacks,
cryptographic failures

Next Week

Networking

Networking 101

Network Attacks and Defenses