

CS 546

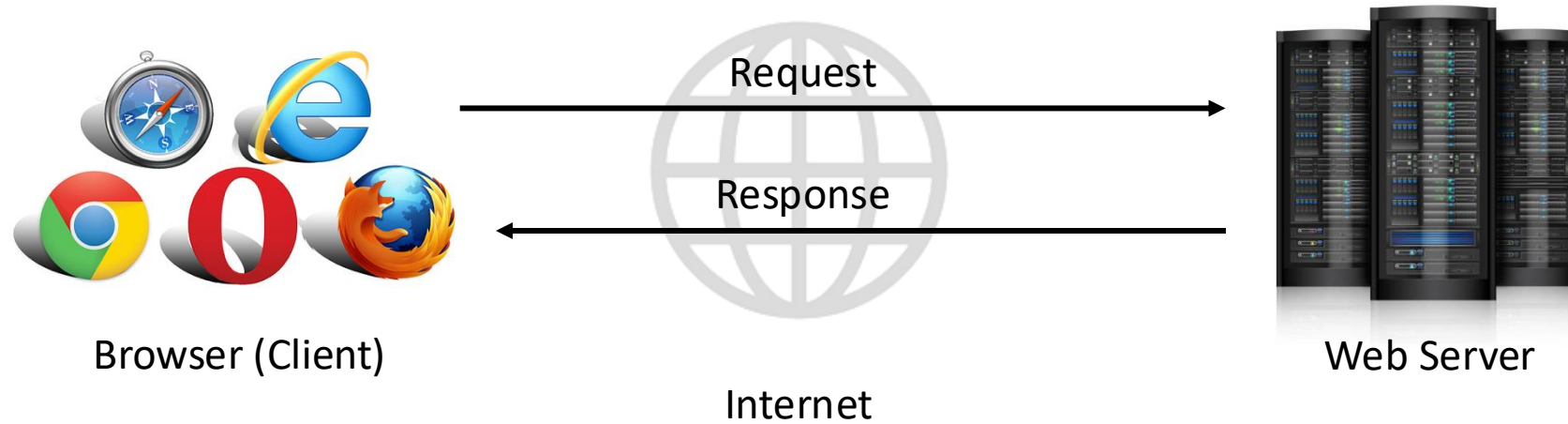
Computer System Security

Lecture 6: Web Security - I

Instructor: Aarushi Goel

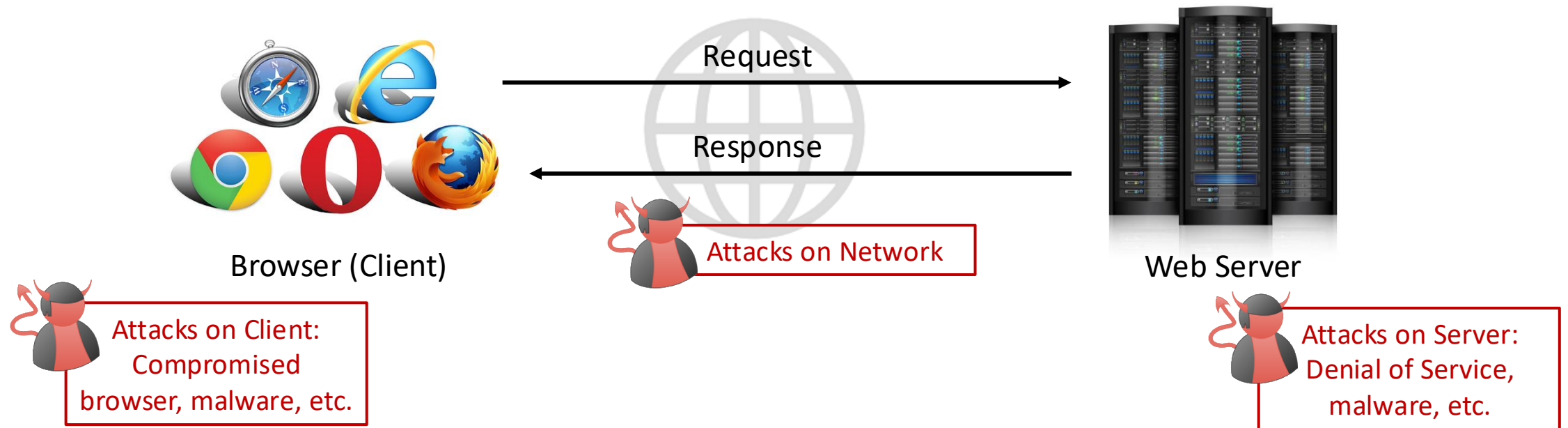
Fall 2026

Web Applications

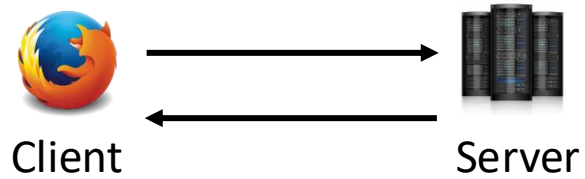


Potential avenues for attacking?

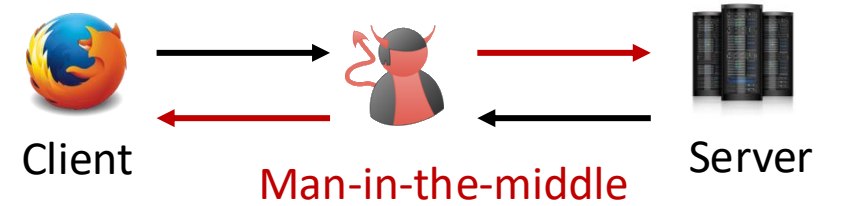
Web Applications



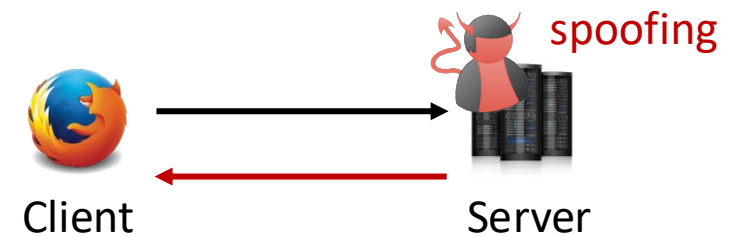
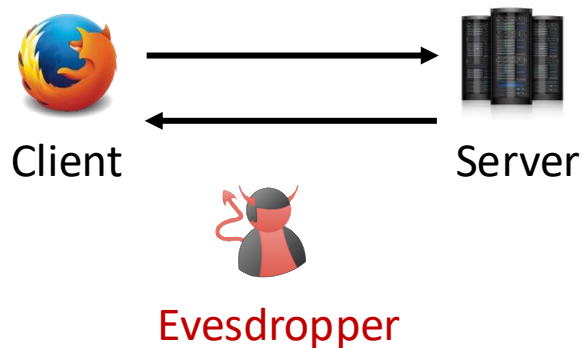
Attacks on Network



Normal Communication



Denial of Service



Attacks on Server

- Attacker controls a malicious website: the website might look professional, legitimate, etc., but the attacker can get users to visit website.
- A benign website is compromised by attacker: attacker inserts malicious content into website and steals sensitive data from website.
- If an attacker gets you to visit a malicious website, they could perform actions on other websites impersonating you or run evil code on your OS.
- Ideally, we would like to mitigate all such attacks.

Attacks on Client/Browser

- Web browser renders web content (HTML pages, scripts) and is responsible for confining web content In other words, browser implementations dictate what websites can do.
- Browsers must ensure that any code on a website must be safely executed on your local computer, without allowing for illegal access to local files, etc.

Interacting with Web Servers

Resources which are identified by a URL (Uniform Resource Locator)

`https://www.cs.rutgers.edu/academics/graduate`

Protocol:

Application layer protocol for exchanging data across the world wide web
e.g., ftp, http, https

FQDN: Fully qualified domain name

Translated to an IP address (128.6.4.2) by DNS
[Host name].[Domain].[TLD].[Root]

- **Host name:** www, mail, etc.
- **Domain:** cs.rutgers
- **Top-level domain (TLD):** edu, com, etc.
- **Root:** Represented by a final trailing dot (.), which is required in DNS configuration files but often hidden by browsers.

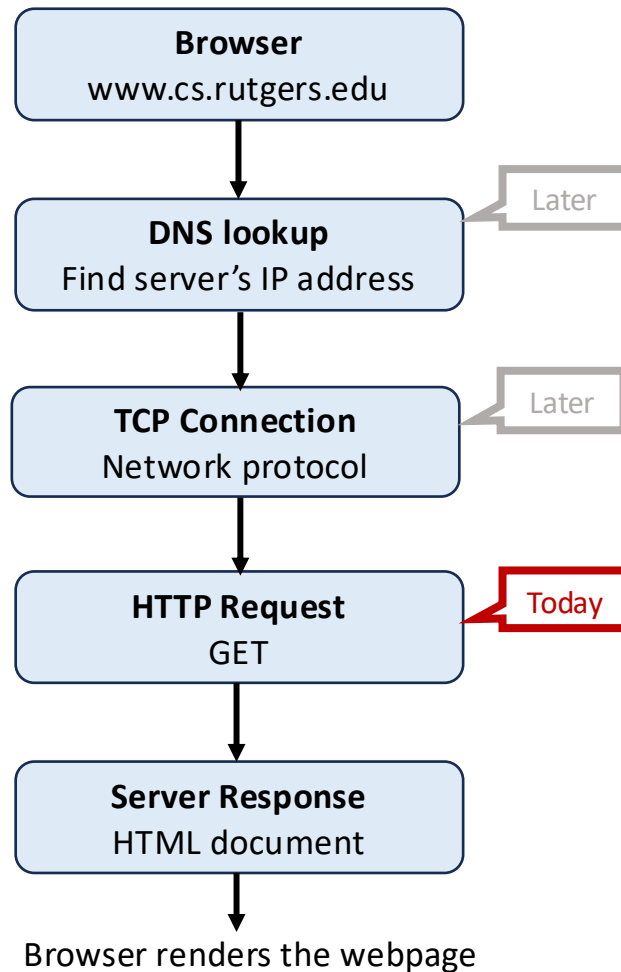
Path to a resource

Here, this is a static content i.e., fixed file returned by the server.

Can also be a dynamic content, i.e., content that the server generates on the fly, e.g. "delete.php?abc..."

HTML (Hypertext Markup Language)

HTML is the standard language used to define the structure and content of a webpage.

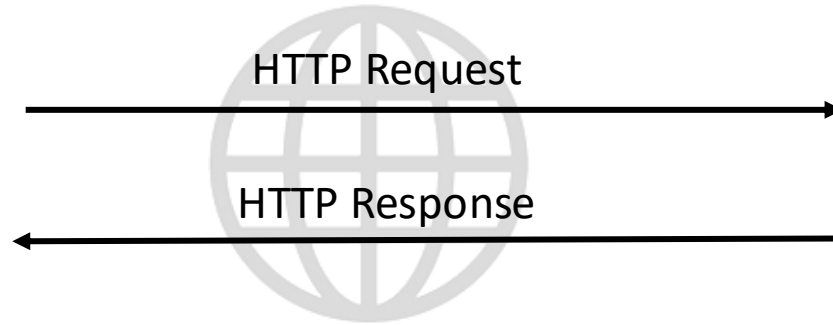


```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Website</title>
  </head>
  <body>
    <h1>Welcome to my website!</h1>
    <p>This is my first webpage.</p>
    <a href="https://www.google.com">Visit Google</a>
  </body>
</html>
```

HTTP (Hypertext Transport Protocol)



Web Browser



Web Server

Requests contain:

- The **URL** of the resource the client wishes to obtain
- **Headers** describing what the browser can do

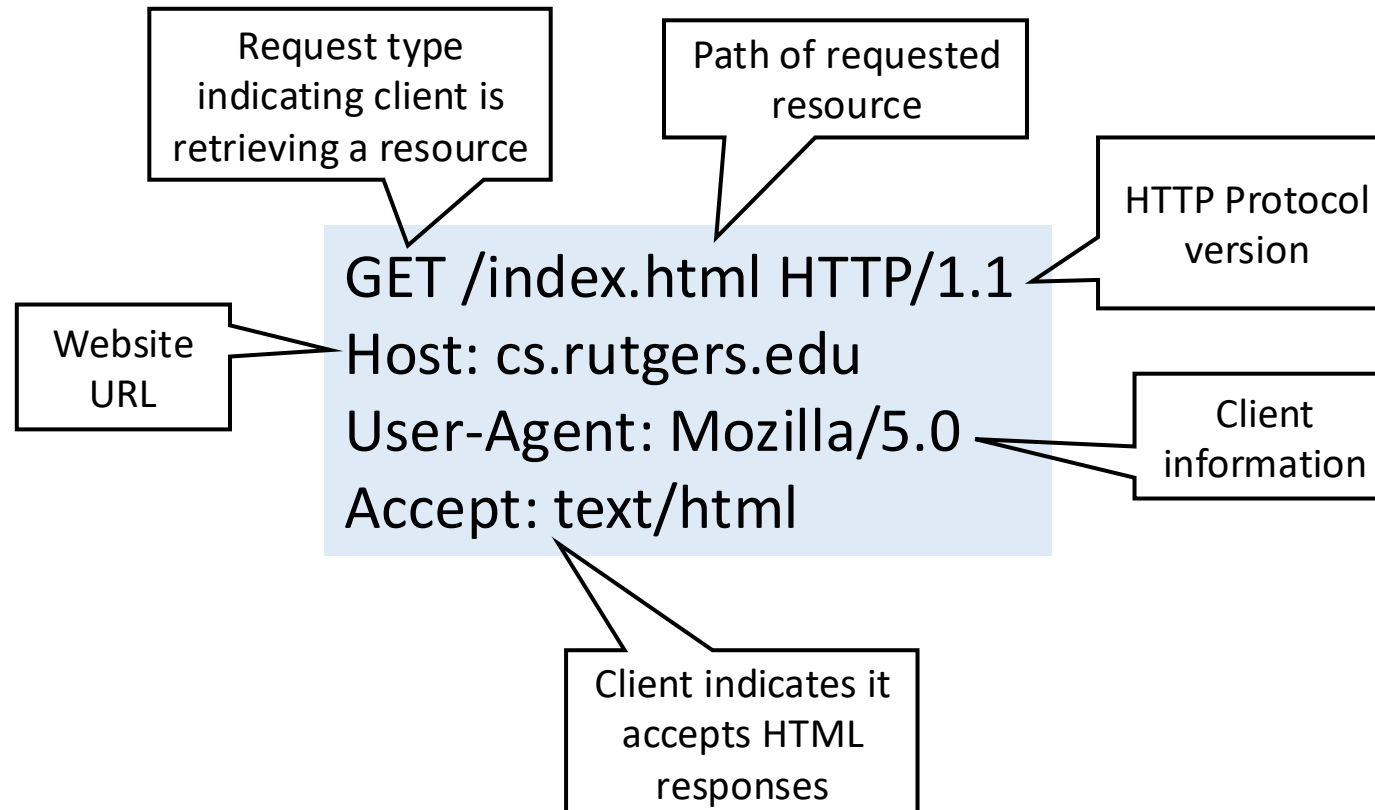
Request types:

- **GET**: all data is in the URL itself
- **POST**: includes the data as separate fields
- Other types: PUT, DELETE, PATCH, etc.

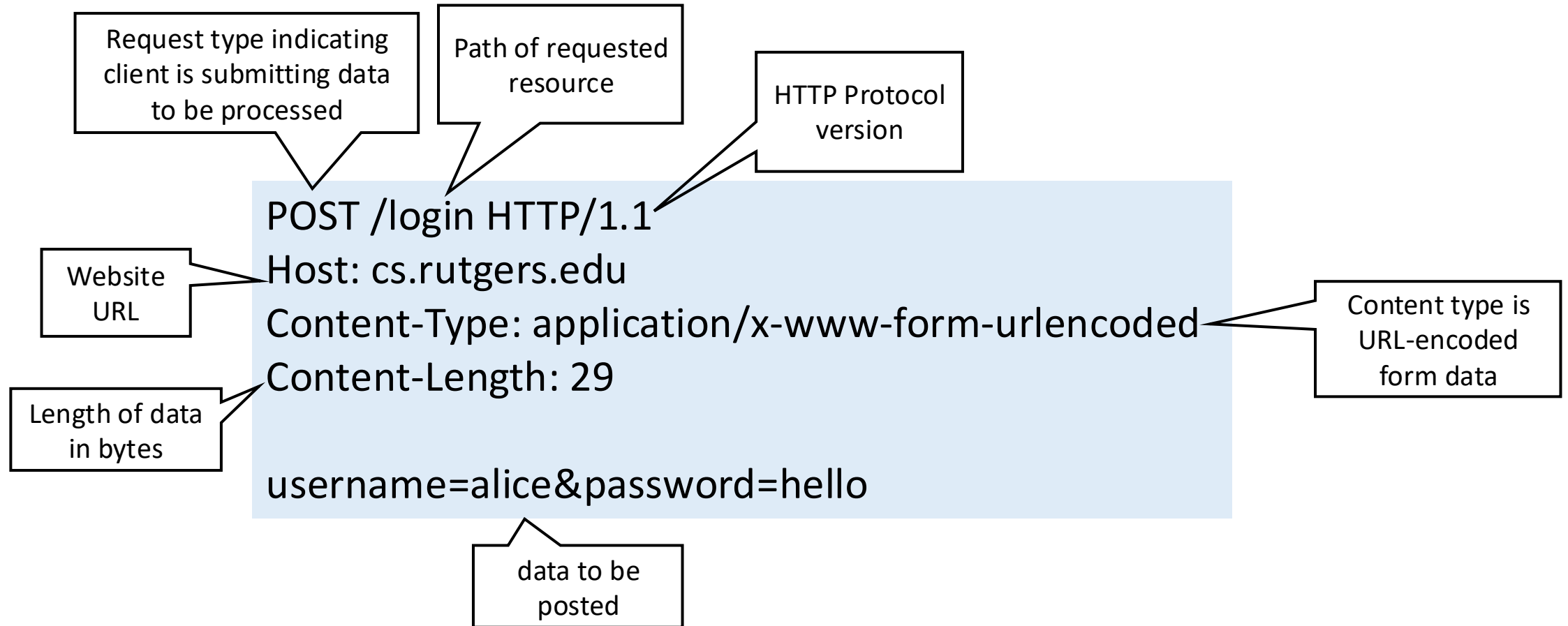
Responses contain:

- **Status** code
- **Headers** describing what the server provides
- **Data**
- **Cookies** (more on these later)
 - Represent *state* the server would like the browser to store

HTTP GET Request



HTTP POST Request



What if “content-length” does not match actual length of data or there are some other inconsistencies in the request?

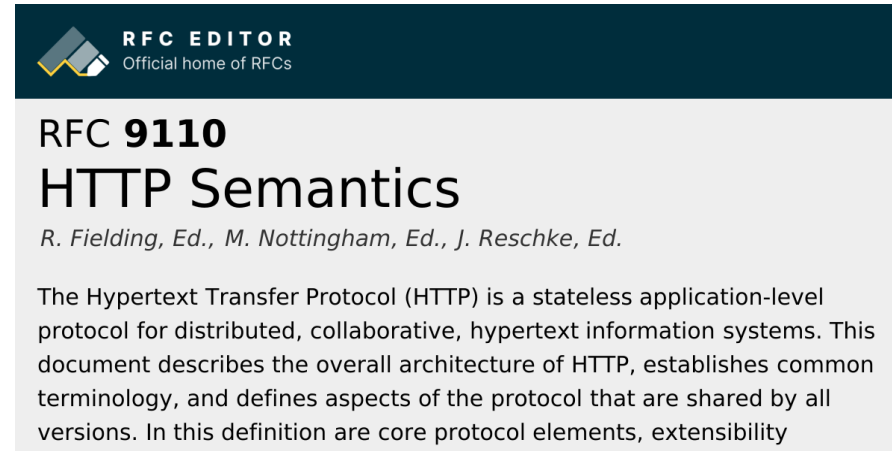
RFC (Request for Comments) Documents



RFC EDITOR
Official home of RFCs

- An RFC is a technical document used to describe Internet protocols, standards, and related technologies.
- When different companies implement protocols like TCP, HTTP, or TLS, they need to agree on exactly how those protocols should work.
- RFCs provide these specifications so that software developed by different companies can communicate correctly.
- For example, Google's Chrome browser and an Apache web server can communicate using HTTP because they implement compatible protocol specifications.
- Researchers, engineers, and other technical contributors draft specifications. Many Internet standards are developed through IETF (internet engineering task force) working groups, where proposals undergo discussion, review, and revision.
- Once approved through the relevant publication process, a document is published with a permanent RFC number.

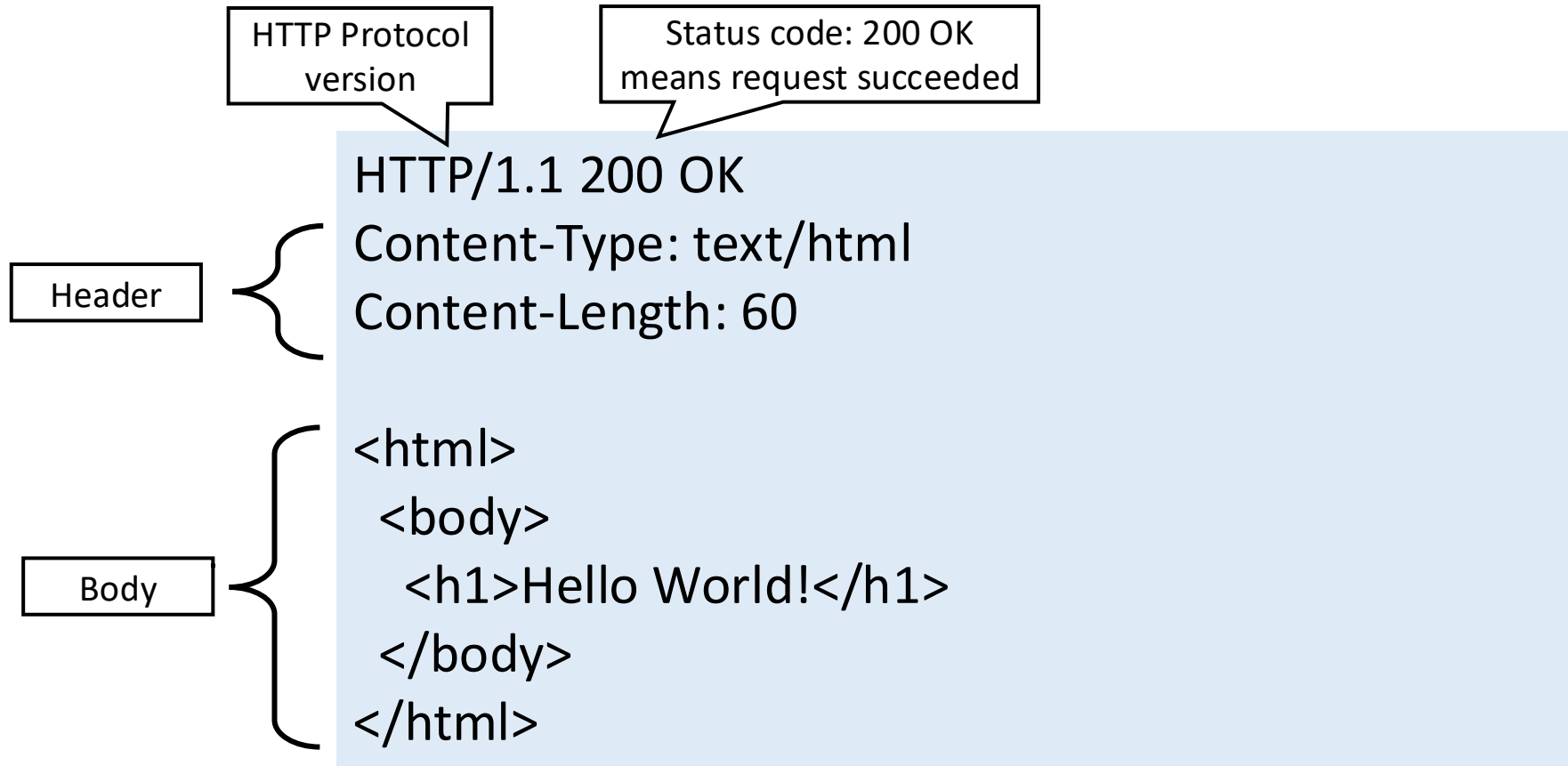
RFC (Request for Comments) Documents



RFC 9110/9112/9113/9114 specify rules governing HTTP(S):

- **Message format:** How HTTP requests and responses are represented.
- **Headers:** How header fields are formatted and parsed.
- **Message boundaries:** How a server determines where one HTTP message ends and another begins.
- **Content-Length:** How the body length is interpreted.
- **Transfer-Encoding:** How message bodies can be transmitted using mechanisms such as chunked encoding.
- **Connection management:** How TCP connections are reused or closed.
- **Security:** How inconsistent parsing can lead to request smuggling and other vulnerabilities.

HTTP Response



HTTP defines how the browser requests and receives content. HTML defines the structure of that content.
HTML could also contain executable JavaScript and elements that trigger network requests and can be exploited by attacker – more on this later

Limitations of HTTP

No Confidentiality

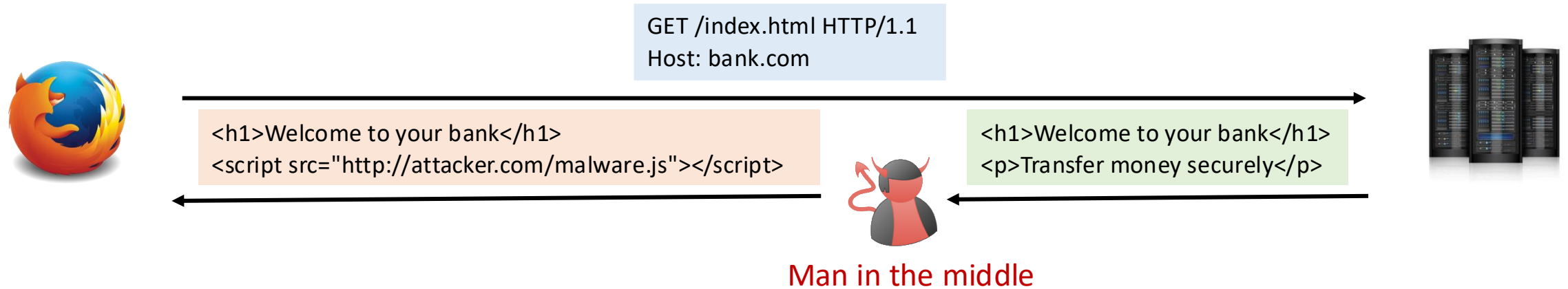
```
POST /login HTTP/1.1  
Host: cs.rutgers.edu  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 29  
  
username=alice&password=hello
```

An attacker who can observe the network traffic (for example, someone controlling an intermediate router) can read the password.

HTTP does not provide any cryptographic mechanism to prevent against this attack!

Limitations of HTTP

No Integrity

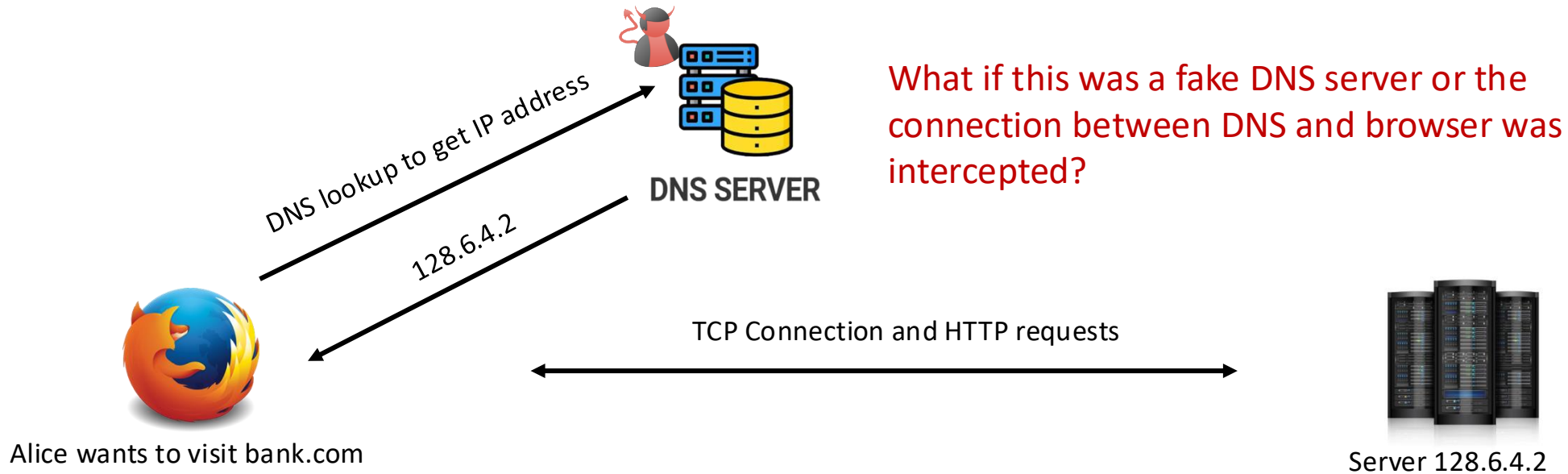


The browser might execute the injected JavaScript, believing it was part of the webpage returned by the bank.

HTTP does not provide any cryptographic mechanism to prevent against this attack!

Limitations of HTTP

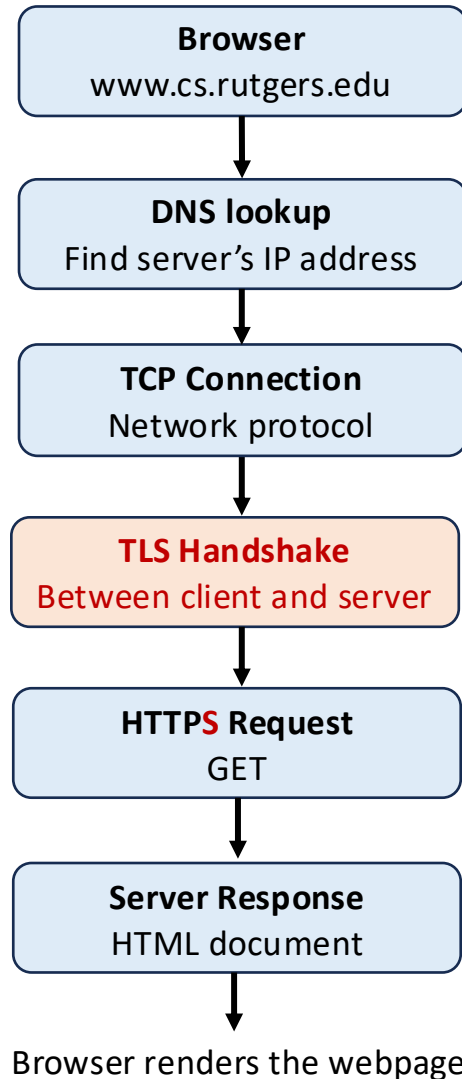
No Server Authentication



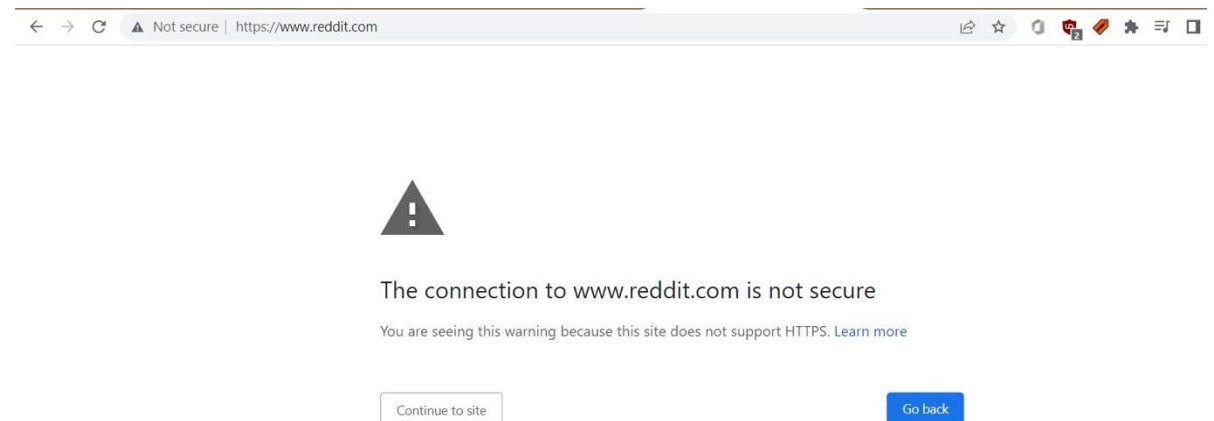
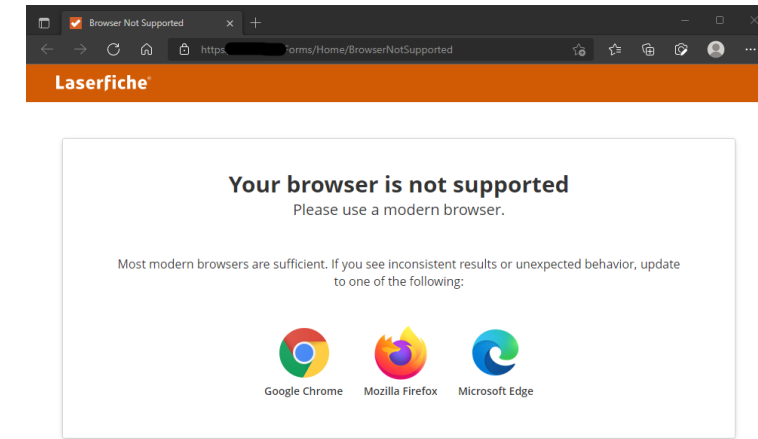
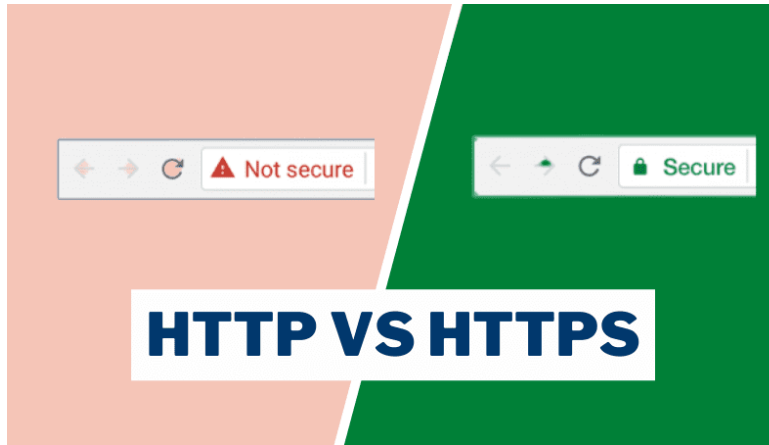
How does Alice know that the server she is communicating with actually belongs to bank.com? An attacker who can redirect Alice's traffic could impersonate the bank.

HTTP does not provide any cryptographic mechanism to prevent against this attack!

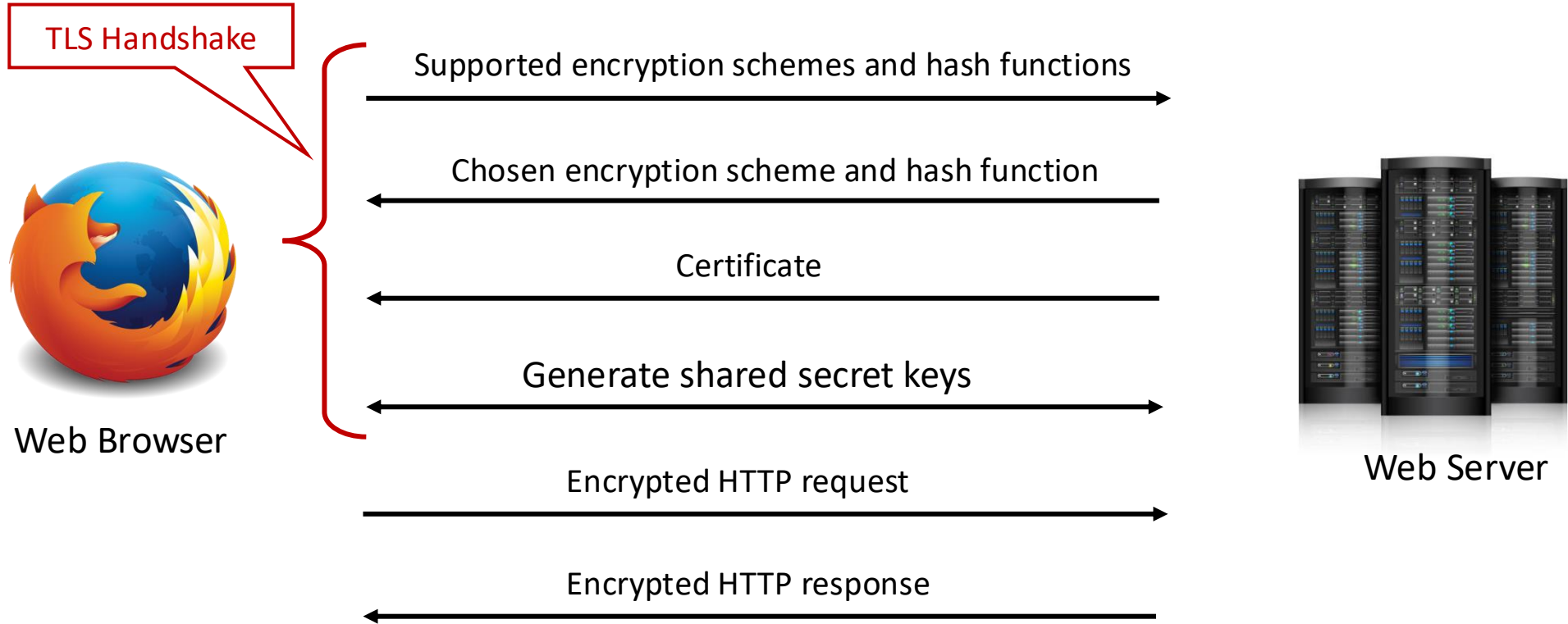
HTTPS (Hypertext Transport Protocol **Secure**)



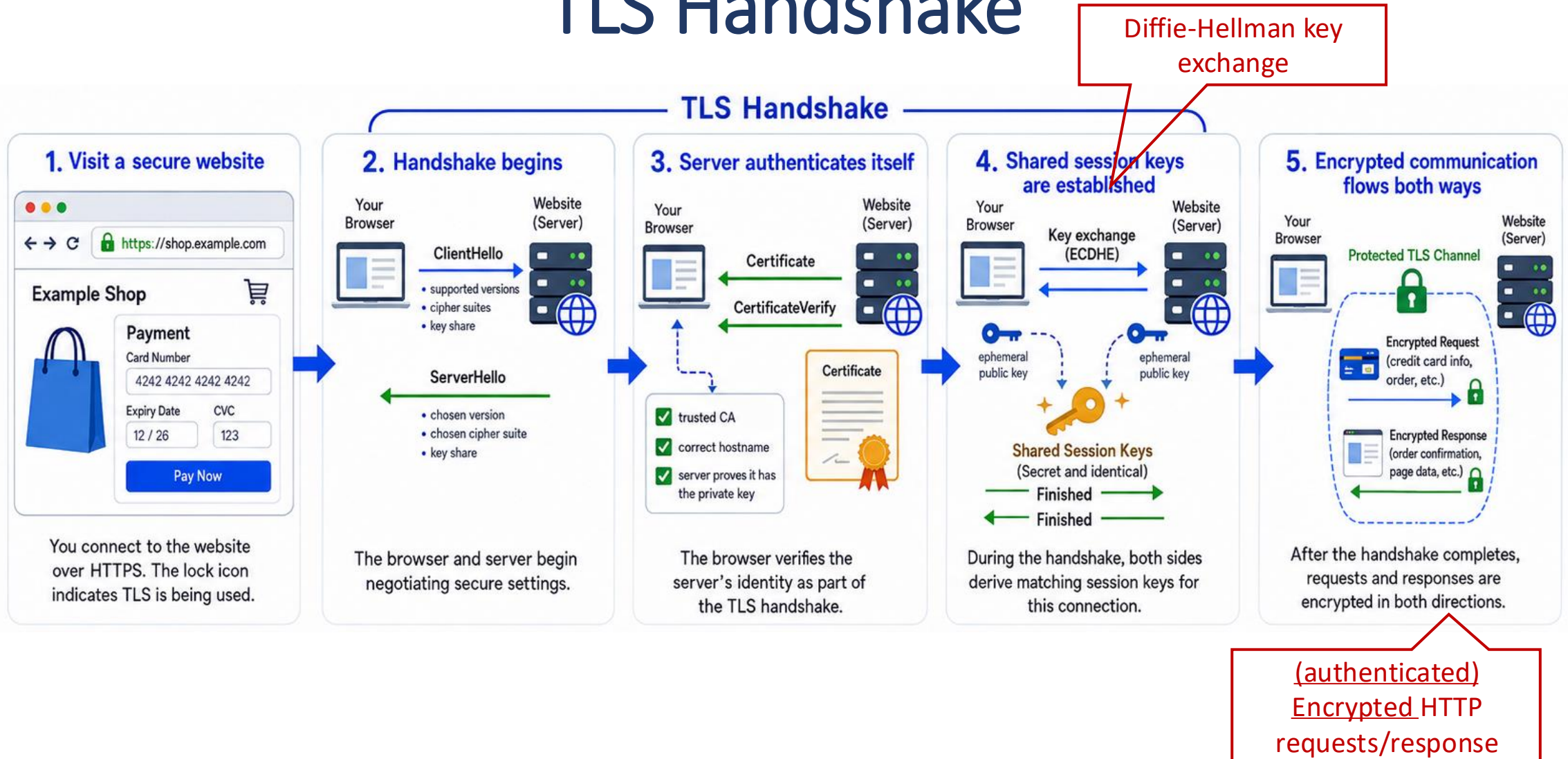
Transport layer security



HTTPS over TLS

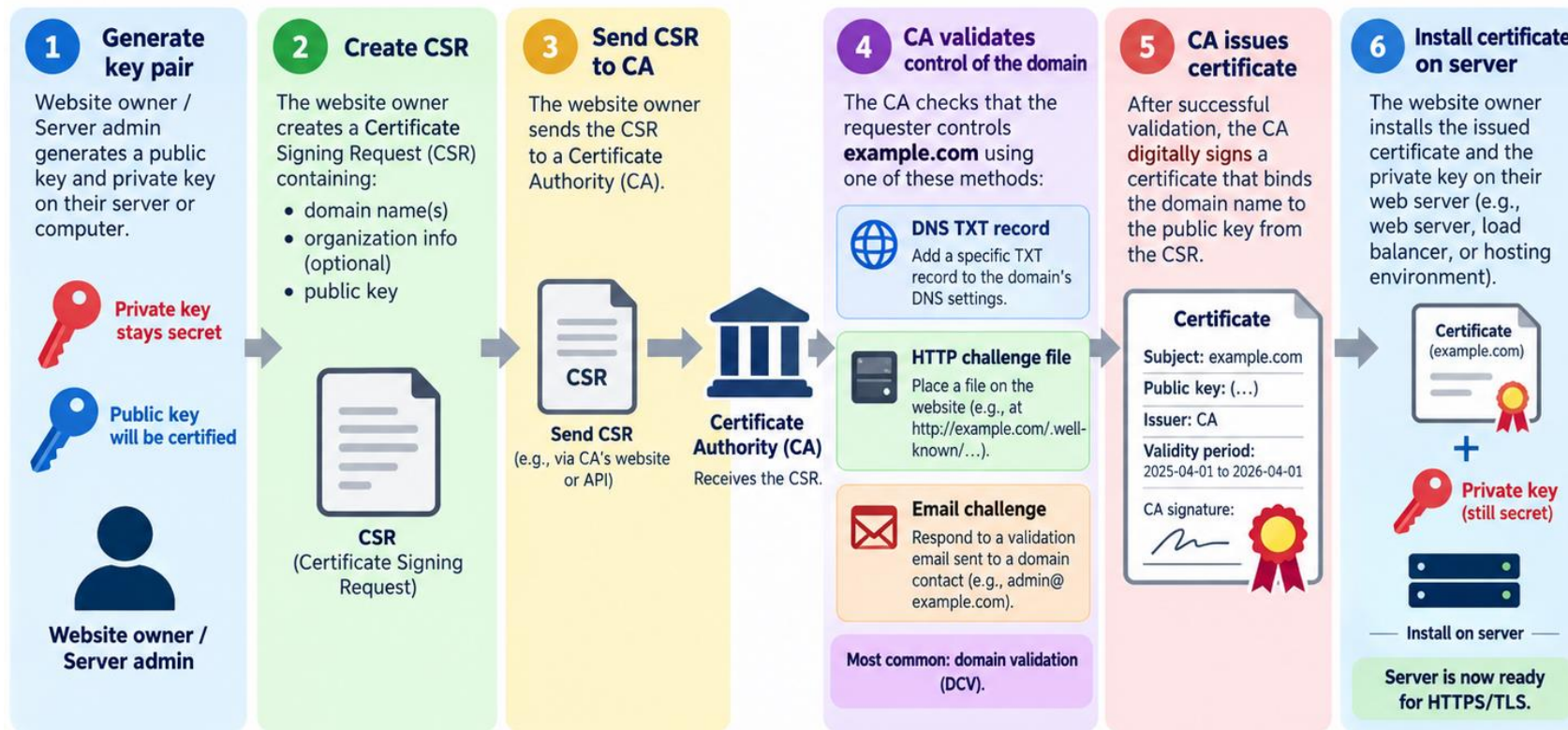


TLS Handshake



How are Certificates Issued?

A **Certificate Authority (CA)** is a trusted organization that issues digital certificates, allowing browsers to verify that a website is authorized to use a particular domain name. Examples: Verisign, Comodo, Thawt, LetsEncrypt, etc.



Certificate Types

DV (Domain validation)

This is only domain validation, i.e., CA is only checking if the requester owns the domain. It does not check whether the requester is running a trustworthy business.

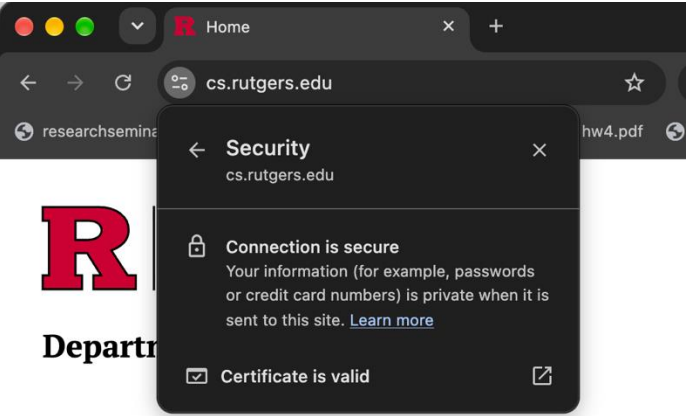
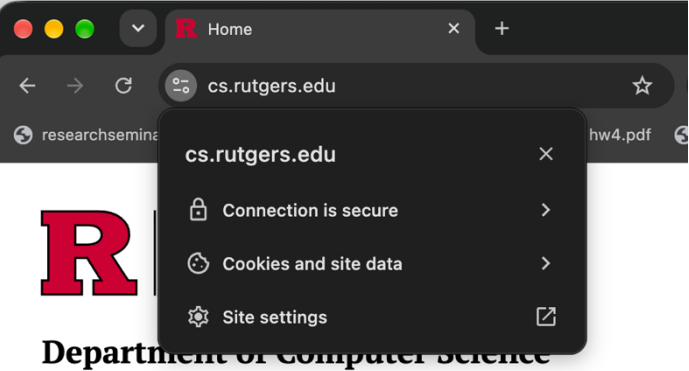
OV (Organization validation)

Prove ownership of the organization that owns the domain

EV (Extended validation)

More extensive validation, increasingly unpopular (\$\$)

cs.rutgers.edu Certificate



Certificate Viewer: www.cs.rutgers.edu

General

Details

Certificate Hierarchy

Sectigo Public Server Authentication Root E46

InCommon ECC OV SSL CA 3

www.cs.rutgers.edu

Certificate Fields

Certificate

Version

Serial Number

Certificate Signature Algorithm

Issuer

Validity

Not Before

Not After

Field Value

CN = InCommon ECC OV SSL CA 3

O = InCommon, LLC

C = US

Export...

Certificate Viewer: www.cs.rutgers.edu

General

Details

Issued To

Common Name (CN)

Organization (O)

Organizational Unit (OU)

www.cs.rutgers.edu

Rutgers, The State University of New Jersey

<Not Part Of Certificate>

Issued By

Common Name (CN)

Organization (O)

Organizational Unit (OU)

InCommon ECC OV SSL CA 3

InCommon, LLC

<Not Part Of Certificate>

Validity Period

Issued On

Expires On

Wednesday, June 24, 2026 at 8:00:00 PM

Saturday, January 9, 2027 at 6:59:59 PM

SHA-256 Fingerprints

Certificate

Public Key

8ad8b5ff615bca2151e92b057768725fd1f8ca9f5fc8021839383c17a5a584f2

b20a9ac9b774fb67d6370a5808f5de23065b1257d19f3797b02a917a8eeab84a

Certificate Verification?

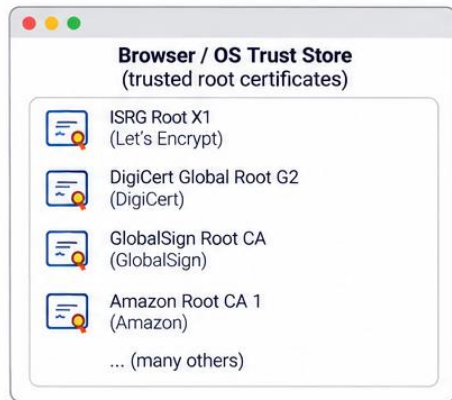
- During the TLS handshake, when the browser receives a certificate from the server, it verifies whether the signature verifies w.r.t. the public verification key of the signing CA.
- The server's public key included in the certificate is essentially a verification key corresponding to a signature scheme.
- The server signs all TLS handshake messages using the corresponding signing key. The browser can be assured that these messages are coming from the intended server by verifying the signatures using the verification key that was included in the certificate. Recall Encrypt-then-MAC.

This works if the browser can trust the CA to be honest and not colluding with the server.
How can the browser trust the CA?

Trustworthy CAs

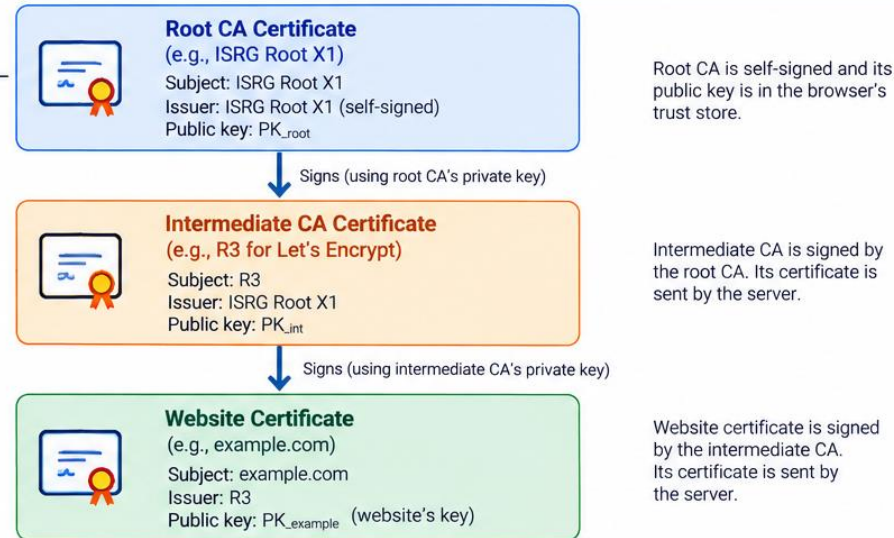
1 Root CAs are built into browsers/OS

Browsers and operating systems come with a pre-installed list of trusted root CA certificates (root store). Each entry contains the CA's public key, which is trusted by the browser vendor.



2 CA hierarchy: root → intermediate → end-entity

Root CAs (self-signed) can delegate signing authority to intermediate CAs, which in turn issue certificates for websites (end-entity certificates).



3 During a TLS handshake, the server sends a certificate chain

The server sends its certificate (for example.com) along with any intermediate certificates (e.g., R3).



4 The browser builds and verifies the chain

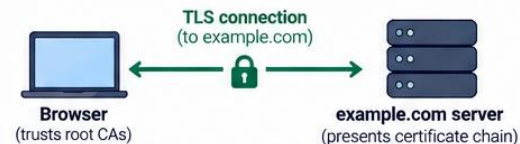
The browser checks that each certificate is signed by the next CA in the chain, up to a root CA that it already trusts.



certificat HBBB coBnnt → (sS)

5 If the chain is valid, the server is trusted

If the chain leads to a root CA in the browser's trust store and all checks succeed (signatures, validity periods, domain name, etc.), the browser accepts the server's certificate and continues the TLS handshake.



6 Key point

The browser knows which CAs to trust because it is manually configured (by the browser/OS vendor) with a list of trusted root CA certificates and their public keys.

Trustworthy CAs

 **Certificate Manager**

 Local certificates

 Your certificates

 **Chrome Root Store**

Chrome Root Store

The Chrome Root Store contains certificates from Certificate Authorities trusted by the Chrome Root Program, and is continually reviewed on an ongoing basis. [Learn more](#)

Trusted Certificates

[Export](#)

Actalis Authentication Root CA	55926084EC963A64B96E2ABE01C...		
Amazon Root CA 3	18CE6CFE7BF14E60B2E347B8DFE...		
Amazon Root CA 2	1BA5B2AA8C65401A82960118F80...		
Amazon Root CA 1	8ECD6884F3D87B1125BA31AC3F...		
Amazon Root CA 4	E35D28419ED02025CFA69038CD6...		
Certum Trusted Network CA	5C58468D55F58E497E743982D2B...		
Autoridad de Certificacion Firmaprofesional CIF A62634068	57DE0583EFD2B26E0361DA99DA9...		
Buypass Class 2 Root CA	9A114025197C5BB95D94E63D55C...		
Buypass Class 3 Root CA	EDF7EBBCA27A2A384D387B7D401...		
OU=certSIGN ROOT CA G2,O=CERTSIGN SA,C=RO	657CFE2FA73FAA38462571F332A2...		
CFCA EV ROOT	5CC3D78E4E1D5E45547A04E6873...		

CA Compromise

- **2011 – DigiNotar:** Attackers compromised CA infrastructure, obtained their signing key and generated fraudulent certificates. Fraudulent Google certificate were then used in actual MITM attacks. Mozilla subsequently removed **DigiNotar from its trusted root program**.
- **2011 – Comodo:** Attacker compromised a registration authority partner. The attacker exploited the compromised account to obtain nine fraudulent certificates for major websites: google, yahoo, mozilla, etc. The **certificates were revoked** after detection. Importantly, investigators did not believe Comodo's root private key had been compromised.
- **2013 – Turktrust:** mistakenly issued two intermediate CA certificates to entities that should have received ordinary website certificates. One of these intermediate certificates was subsequently used to issue a fraudulent certificate for *.google.com. Delegating CA authority is substantially more dangerous than issuing an ordinary website certificate.