HTTP

HTTP

- HyperText Transfer Protocol
- Invented by Tim Berners Lee @ CERN
- A protocol for delivering *resources* over the web
- TCP/IP connections, default (server) port 80
- HTTP client & HTTP server

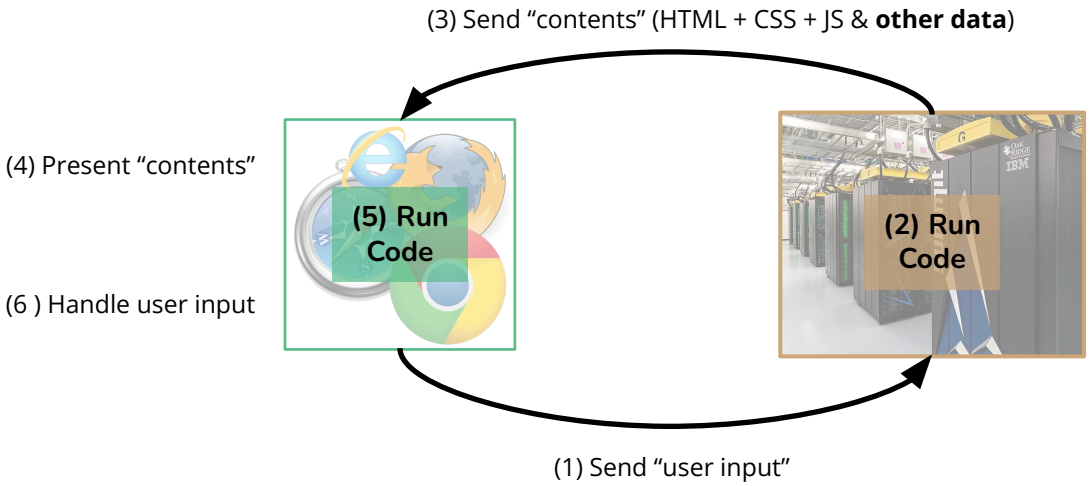
Other network Transfer Protocols

- FTP: File Transfer Protocol
- FTPS: Secure FTP
- SMTP: Simple Message Transfer Protocol
- NTP: Network Time Protocol

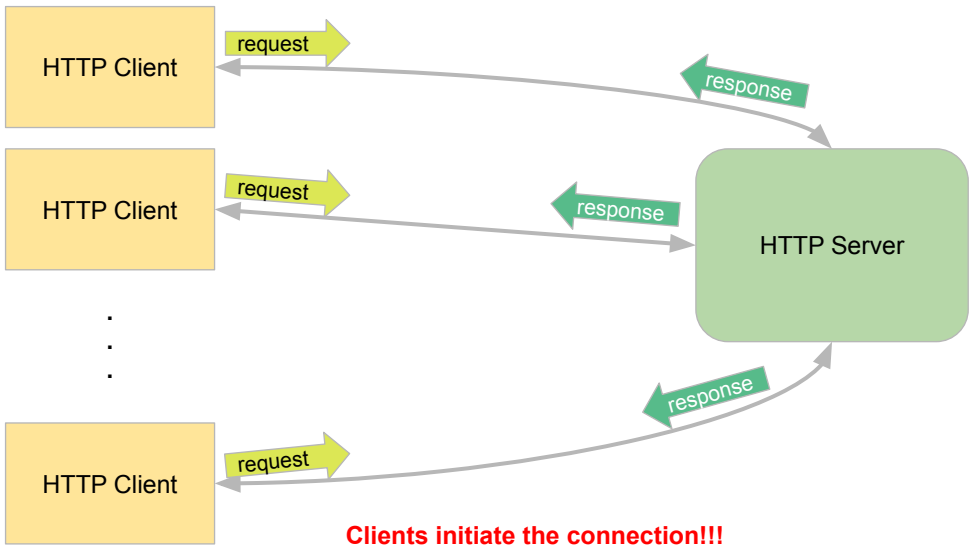
Why learn the details of HTTP?

(Later) How to programmatically initiate HTTP requests from your program

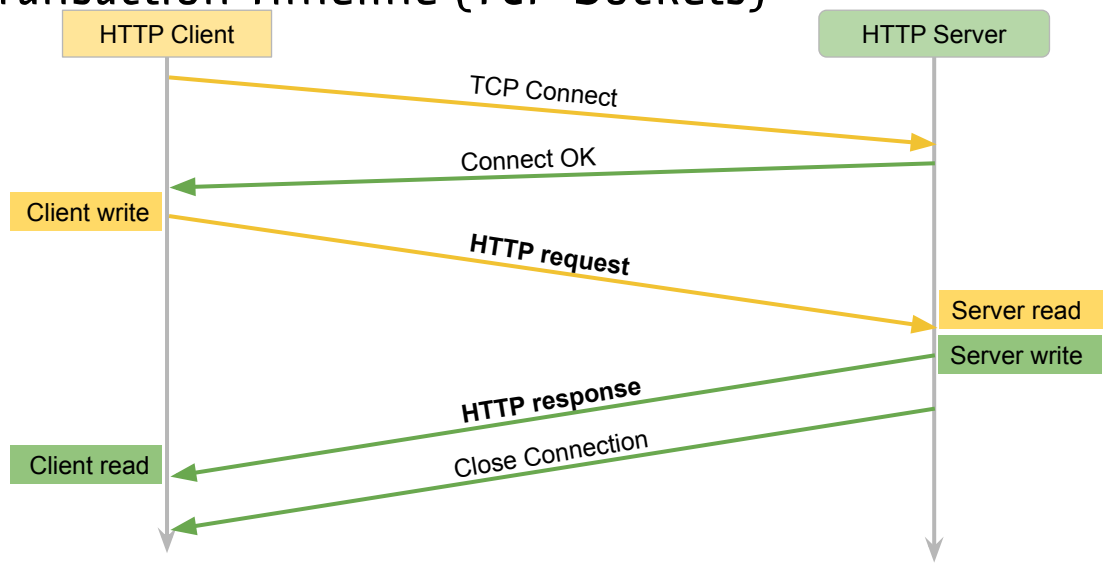
Web Client/Server Architecture



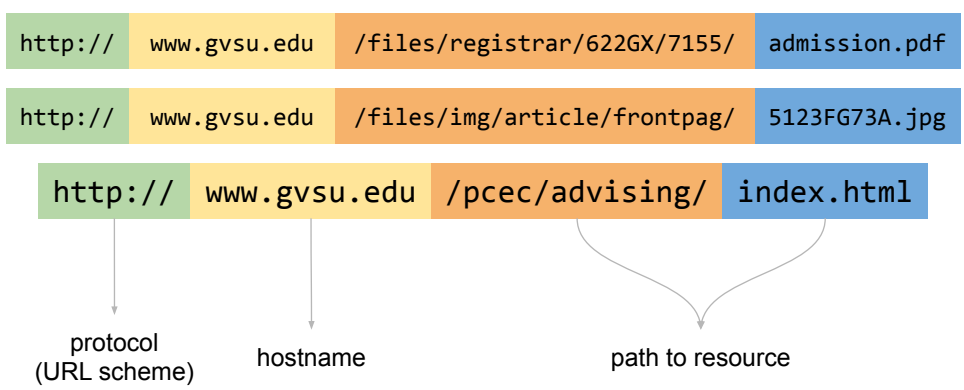
HTTP Communication Model



Transaction Timeline (TCP Sockets)



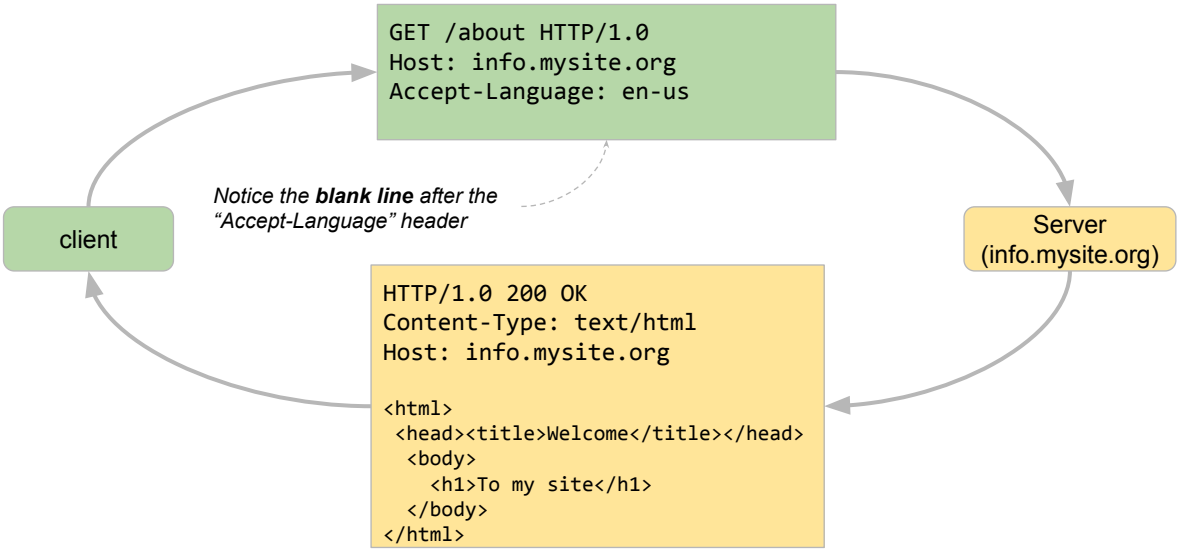
HTTP URL: Uniform *Resource* Locator



HTTP Messages: Request & Response

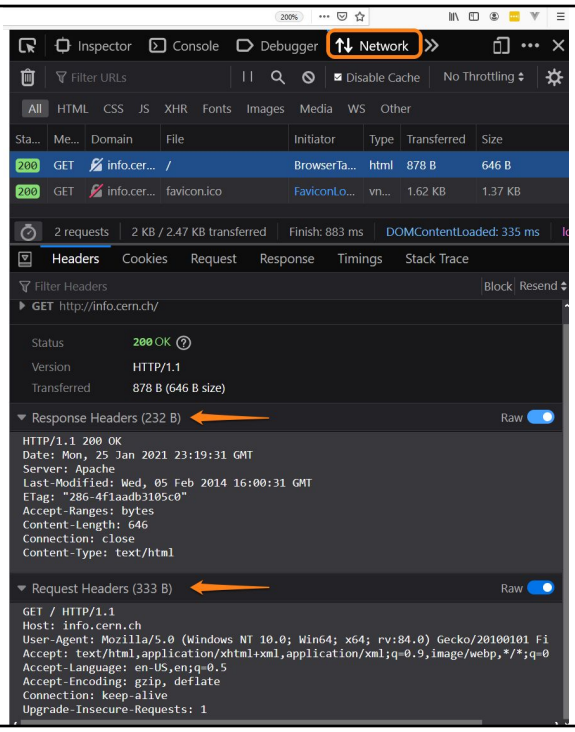
Demo: URL & Web Dev Tools

http://info.mysite.org/about/



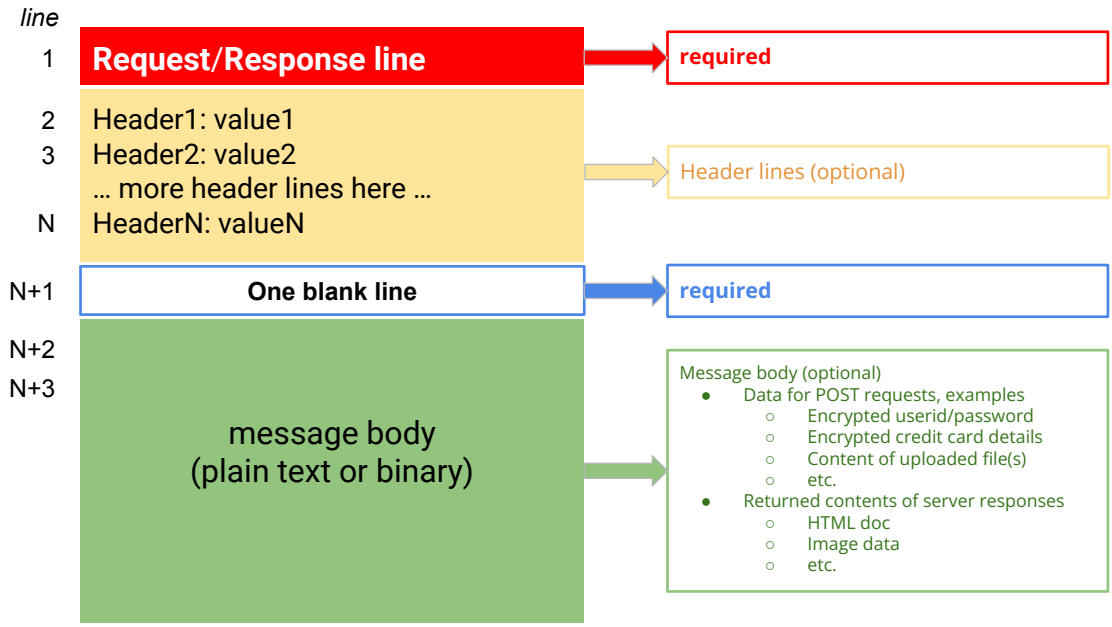
Web Browser DevTools (Network Tab)

http://info.cern.ch




```
curl --verbose http://info.cern.ch
```

(On Linux/OSX/Windows 10 WSL)



HTTP headers of interest to web developers

Header	Description	Example
`Accept`	Inform server media-type to respond	`Accept: image/jpg`
`Accept-Language`	Inform the server which languages the client is able to understand	`Accept-Language: en-US; en-UK`
`Content-Type`	Media type of the returned content	`Content-Type: plain/text`
`Content-Language`	The languages of the content	`Content-Language: en-US`
`Date`	Date and time of the message	`Date: Mon, 21 Aug 2017 18:14:36 GMT`
`ETag`	Identifier used by caching algorithms	`ETag: “"8a9-291e721905000”`
`Host`	Specify the domain name of the intended server (mainly for Virtual Hosting)	`Host: www.personal.me:5555`

HTTP 1.0 Commands (Request Methods)

- GET
- POST
- HEAD (like GET but the server responds only with header, no data)

More-frequently used

- PUT
- DELETE
- OPTIONS

Less-frequently used

Operation	HTTP Request
Create	POST
Read	GET
Update	PUT
Delete	DELETE

POST: upload file to Bb

client

```
POST /path/to/your/course HTTP/1.0
Host: mybb.gvsu.edu

The text/binary contents of your
File to upload to Bb will be included
as attachments here
```

Server (mybb.gvsu.edu)

```
HTTP/1.0 200 OK
Content-Type: text/html
Host: info.mysite.org

Additional message from server goes here
```

HTTP Status Code

Status Code	Description
1xx	Informational messages
2xx	Success messages
3xx	Redirect message
4xx	Error on the client's behalf
5xx	Error on the server's behalf

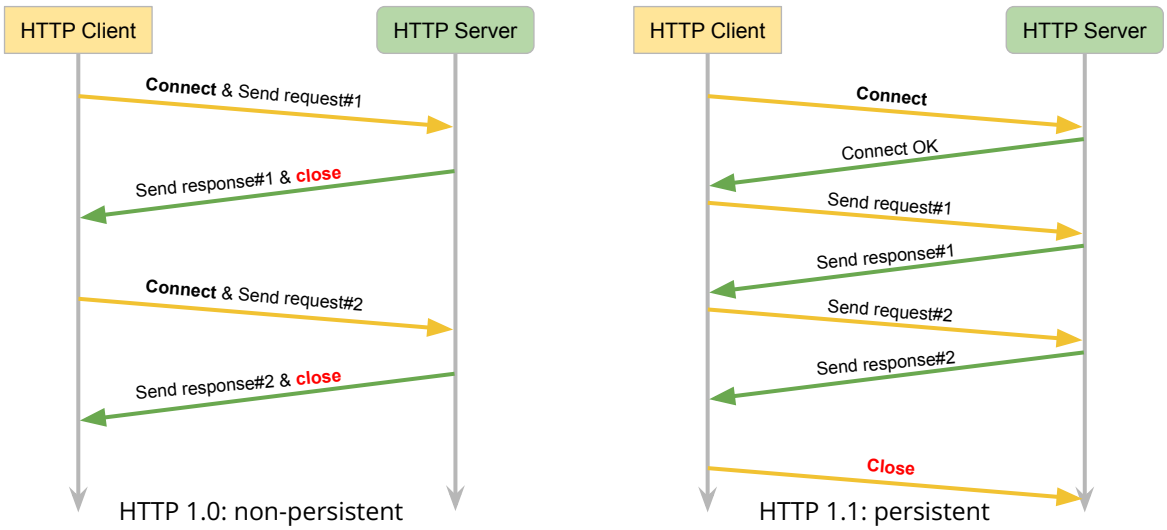
Simple HTTP Server

```
// myFirstHTTPServer.ts
import { createServer, IncomingMessage, ServerResponse } from "http";

const myServer = createServer(
  (req: IncomingMessage, res: ServerResponse) => {
    res.write("<h1>Hello world</h1>");
    res.end();
  }
);

myServer.listen(5000, () => {
  console.debug("Server is listening at port 5000");
});
```

```
# From your project
npx ts-node myFirstServer.ts

# Then from your browser
http://localhost:5000
```

Coding Demo:
NodeJS: http server

HTTP Connections: Persistence



HTTP 1.0

- One request per connection (non-persistent)
- Cache control is **timestamp based** with one-second resolution (inaccurate)

- Client cannot request a portion of a resource
- Responses are delivered in one big chunk

HTTP 1.1

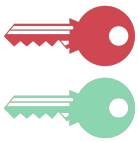
- N requests per connection (persistent)
- Response can be delivered in chunk
- Cache control is **content based**, responses include entity tag (Etag), similar to hash value
- Clients can request **partial content**
  - "`Range:`" header line in HTTP request

- Responses may be delivered in many small chunks

HTTPS

- HTTP Secure
- HTTP over TLS (Transport Layer Security)
- HTTP over SSL (Secure Socket Layer)

- PKI (Public Key Infrastructure)

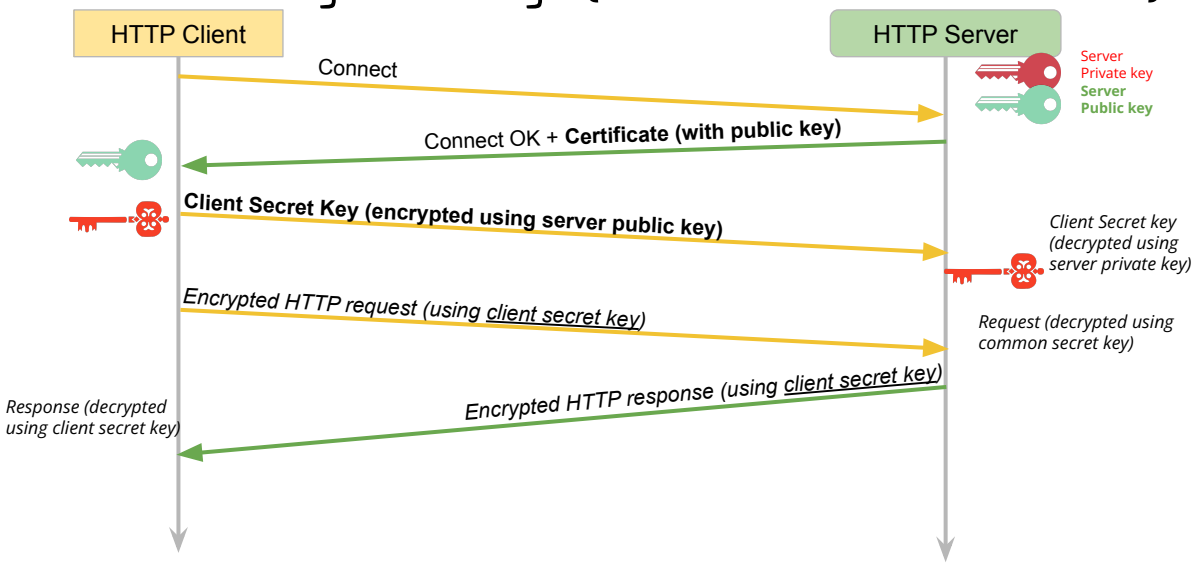


private key

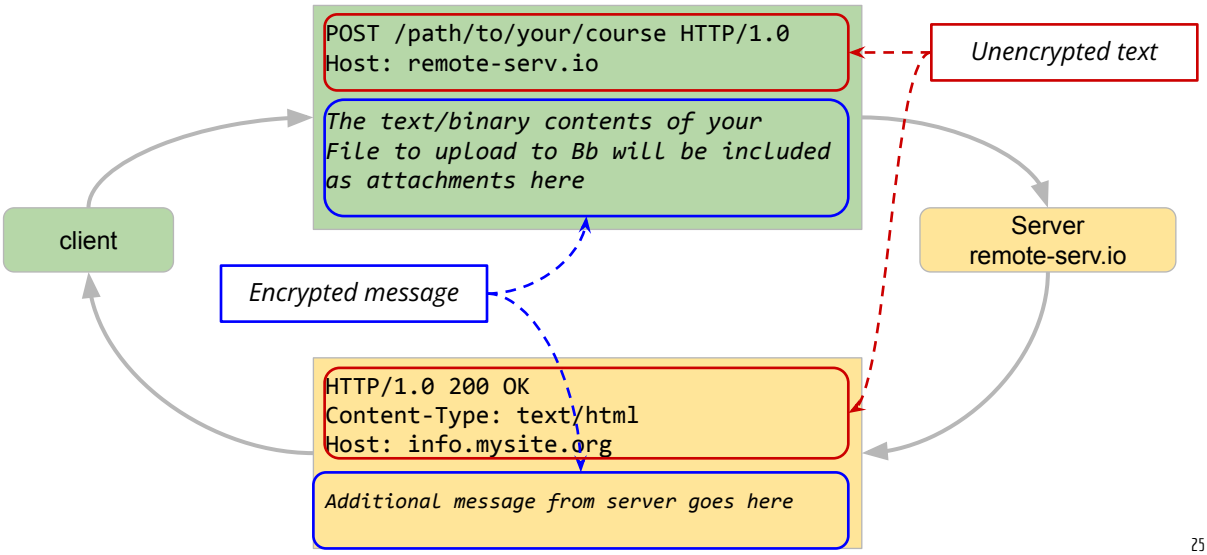
public key

Encrypted Message (with public+private key pair)

Secure Message Exchange (over Persistent Connection)



GET or POST over secure connections



Uploading Sensitive Data over Encrypted Channel

- Embed the sensitive data in a GET request query string

```
GET /place/my/order/?creditcard=xxxxyyyyzzzzuuuu&zip=12345 HTTP/1.0
Host: www.amazon.co.uk
```

unencrypted

- Embed the sensitive data in a POST message payload

```
POST /place/my/order HTTP/1.0
Host: www.amazon.co.uk

creditcard=xxxxyyyyzzzzuuuu
zip=12345
```

unencrypted

encrypted

Certificate and Certificate Authority (CA)

Certificate: *Proof of Your Identity*

Certificate Authority:
Trusted Organizations who issue certificates

Michigan IDs vs. Browser Certificates

Michigan IDs	(Browser) Certificates
A formal proof of your identity	A formal proof of the web server identity
Issued and **signed** by Secretary of State	**Issued** and **signed** by Certificate Authority
Provide other proof of identity (birth certificate, passport) to **apply** for Michigan ID to the SoS	Certificate Signing **Request**: server request a CA to sign the server's identity (public key) using the CA key
The SoS is a trusted government body	Trusted CAs

Browser Demo: Certificates

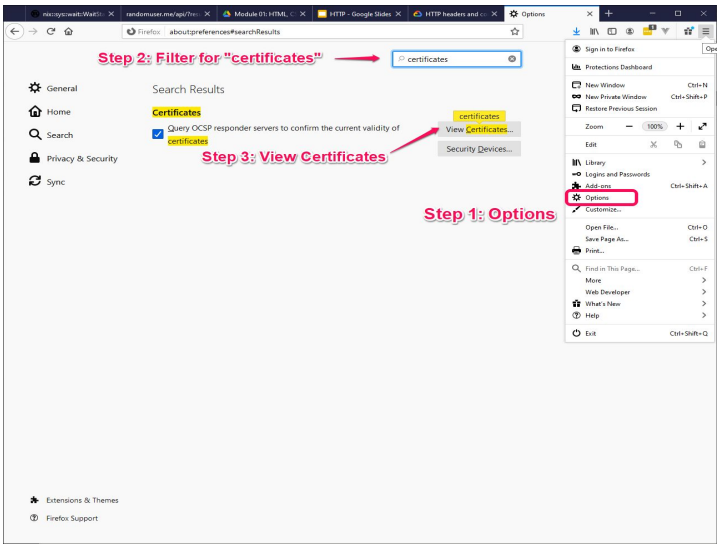
(1) From HTTPS connection

(2) From Settings => View Certificates

Trusted Certificate Authorities

Screenshot of FireFox.

Other browsers follow similar steps.



Obtaining Web Certificates ("Web ID Cards")

Proof of identity
(passport, GOV ids, birth certificates)

Certificate
(signed by CA)

Private+Public
Key Pair

Certificate Authority
(CA)
Known to Browsers

Web Certificate
(signed by CA's
private key)

Self-Signed Certificates (for Development)

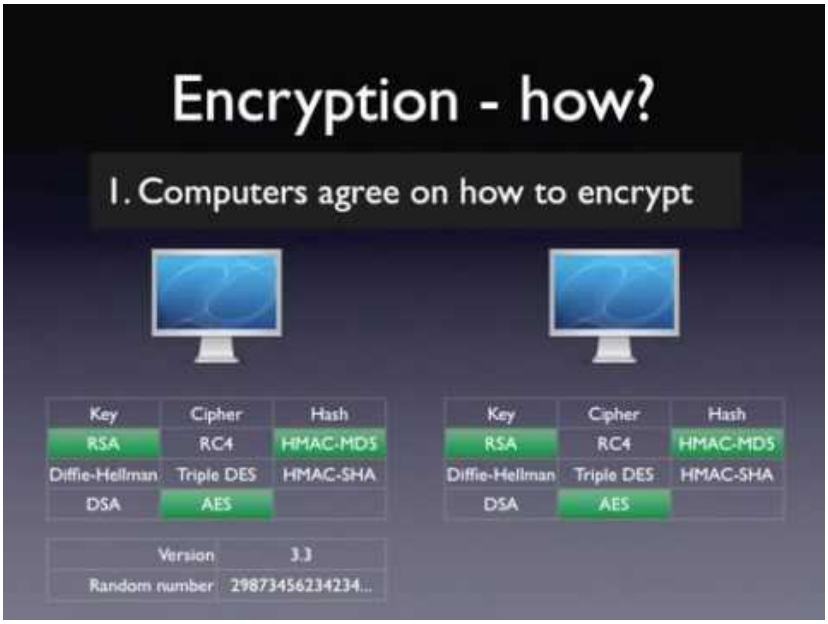
Proof of identity

Private+Public
Key Pair

Certificate (Signing)
Authority
Unknown to browsers

Certificate
(signed by CA)

Web Certificate
(signed by CA's
private key)



HTTP/1.1

- HTTP messages encoded in text format
- Require multiple connections to achieve concurrency
- Uncompressed response headers
- No resource prioritization

HTTP/2

- HTTP messages encoded in binary format
  - Message = request or response
- Multiple concurrent channels on a single connection
- Compressed response headers
- Resource prioritization (important requests complete more quickly)