IERG 4210 / IEMS 5718 Web Programming and (Internet) Security

TLS and Web Browser Security

Slide materials adapted from IERG4210 by Prof. Kehuan Zhang, Prof. Sherman Chow, Dr. Adonis Fung

Agenda

- Concepts of TLS/SSL
  - TLS: Transport Layer Security
  - SSL: Secure Socket Layer
  - SSL is obsolete and replaced by TLS, but the term is often used interchangeably
- Brief revision on public key cryptography
  - TLS/SSL is based on public key encryptions
- A high-level overview on the TLS protocol
- TLS implementations in browsers
- Security problems of TLS in real-world

TLS/SSL

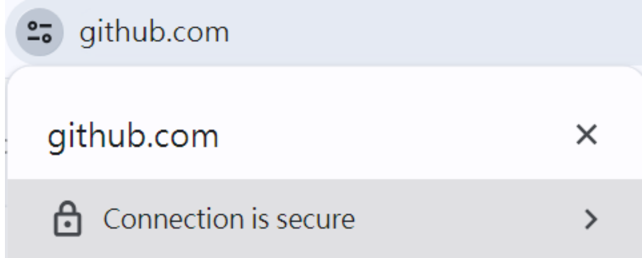
How can the client know it is the valid web page?

Note: TLS is upgraded version of SSL (so we'll simply refer it to TLS)

Note: Current Version is TLS v1.3 (RFC 8446)

Why do we need TLS?

- Web applications include three components:
  - Server, Browser, and communication network (the Internet)
- TLS are designed to solve following security problems
  - How to verify the server’s identity
    - → to ensure that we are not talking to fake and malicious server
  - How to achieve a secure communication over (public) Internet
    - → Internet is public and shared, thus could be attacked easily
    - → packet sniffing, traffic replaying, message modification, …
  - How to ensure the integrity of client (i.e. the browser)?
    - → E.g., online application for renewal of vehicle license
    - → https://www.gov.hk/en/residents/transport/vehicle/renewvehiclelicense.htm
    - → There is a place to let user upload their personal certificate
  - Another Example: EC-Ship Online Portal
    - → https://hongkongpost.hk/en/preparation/posting/ecship/index.html
    - → Digital certificate needs to be installed/imported in browsers



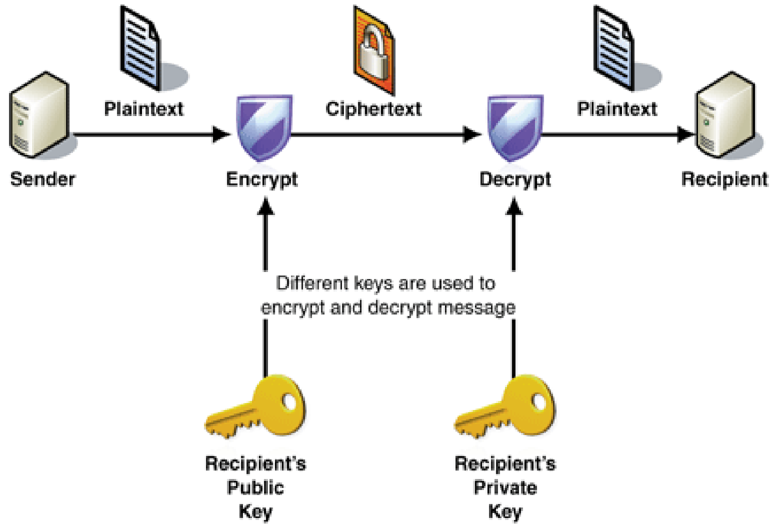
Revision on Public Key Cryptography

- TLS is based on public key encryptions
- What is public key encryption?
  - Also called asymmetric key encryption (vs. Symmetric Key Encryption like AES)
  - There are two encryption keys in a public key scheme
    - → There is only a single key in symmetric key encryption
  - A public key–announced to the public
  - A private key–to be kept as secret and known only to the user (or the server)
- Keys must be used in pairs, but have different security implications
  1. Achieve confidentiality by encrypting with public key:

     $c_1 = Enc(K_{pub}, m_1)$ and $m_1 = Dec(K_{priv}, c_1)$
  2. Achieve integrity by encrypting with private key:

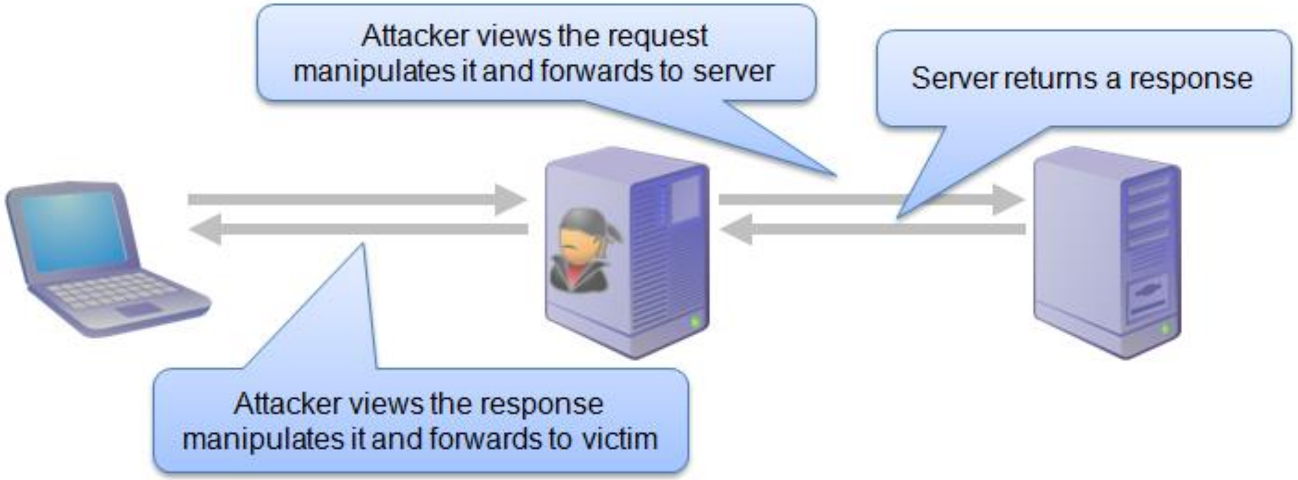
     $c_2 = Enc(K_{priv}, m_2)$ and $m_2 = Dec(K_{pub}, c_2)$
- TLS uses the second scenario
  - i.e., encrypt with private key, more specifically,

  generate digital certificate using private key of a Certificate Authority



Man-In-The-Middle (MITM) attack

- Instead of talking directly to the server,

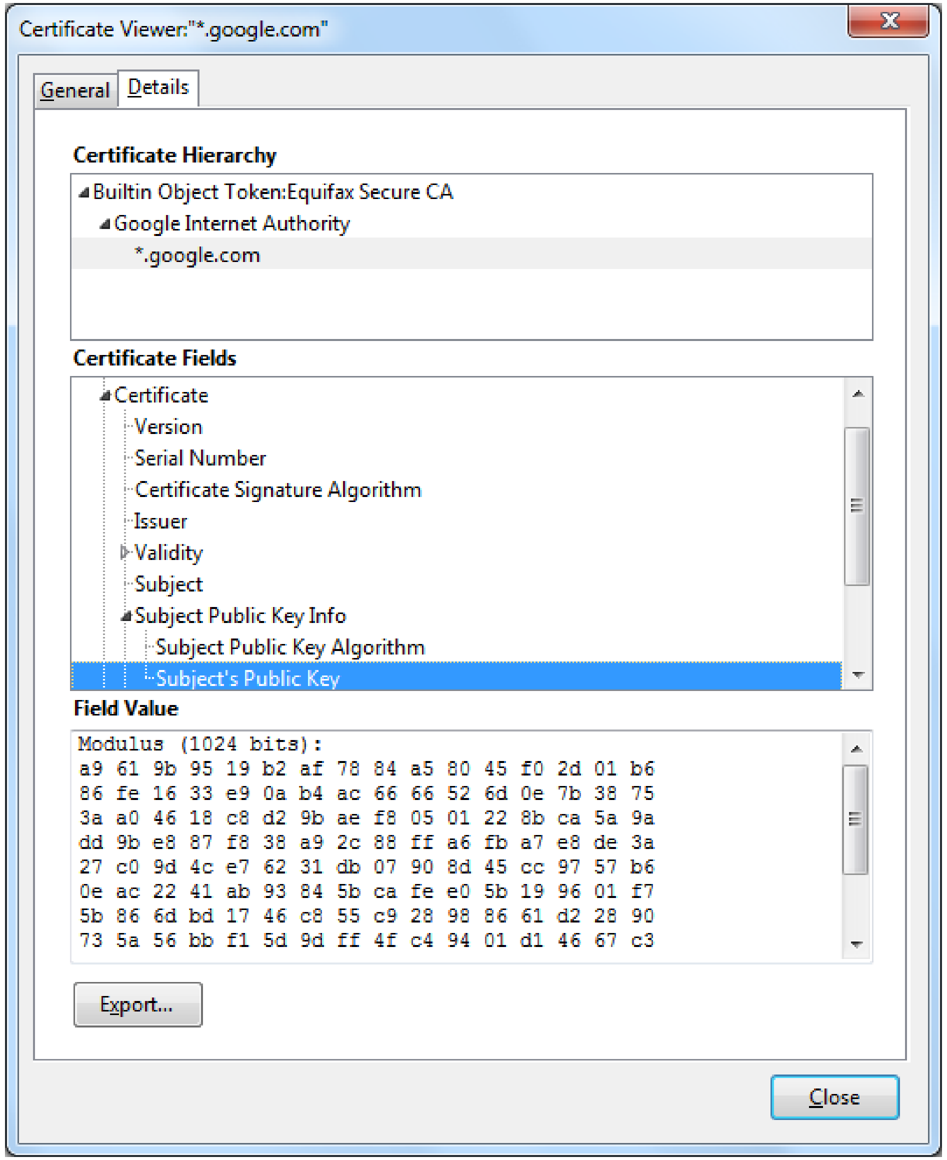


→Note: this is an active attacker, as he tampers content

- If no SSL is used, MITM even launched is transparent to victims
- If SSL is properly configured, MITM can be detected and certificate warnings will appear

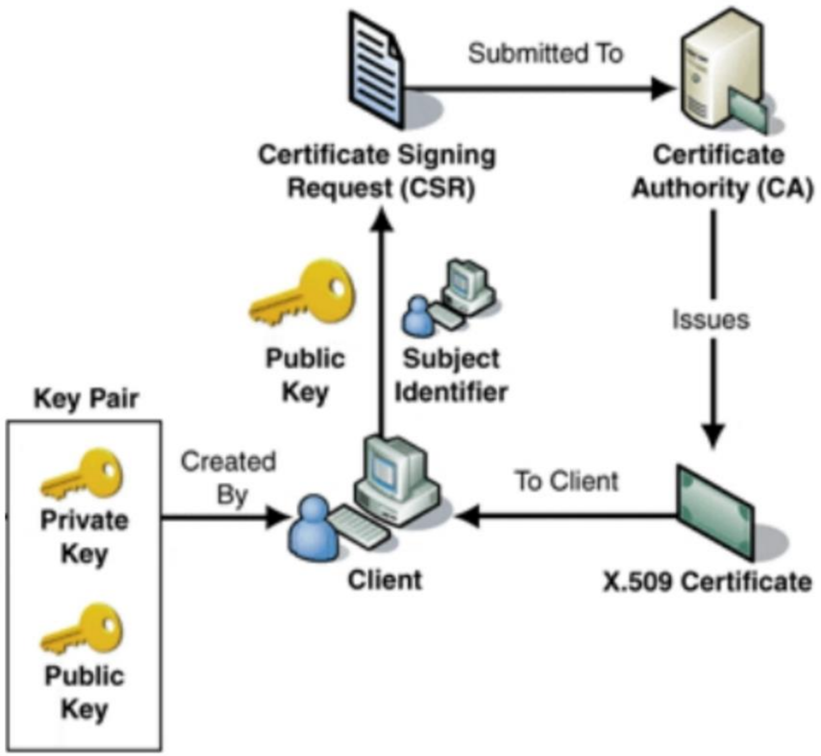
Digital (Public-Key) Certificate

- Everyone can generate public/private key pair.
  - How to bind a key pair to an ID?
- A "trusted" certificate authority (CA) sign on it after checking your ID
- Important fields of a certificate:
  - Subject identifier aka Common Name or "CN" (domain name for server certs)
  - Validity period
  - CA-signed Public Key
  - etc…



A Key Component of TLS - Digital Certificate

- A digital certificate consists of information about:
  - holder of this certificate
  - endorsement from a certificate authority (CA)
- To apply a (server) certificate from the CA:
  - Generate a Key Pair
    - → with Subject equals the domain name
  - Produce a Cert-Signing Req. to the CA
  - CA validates that the applicant is a valid domain name holder and/or can prove his identity
  - Install the issued cert to server



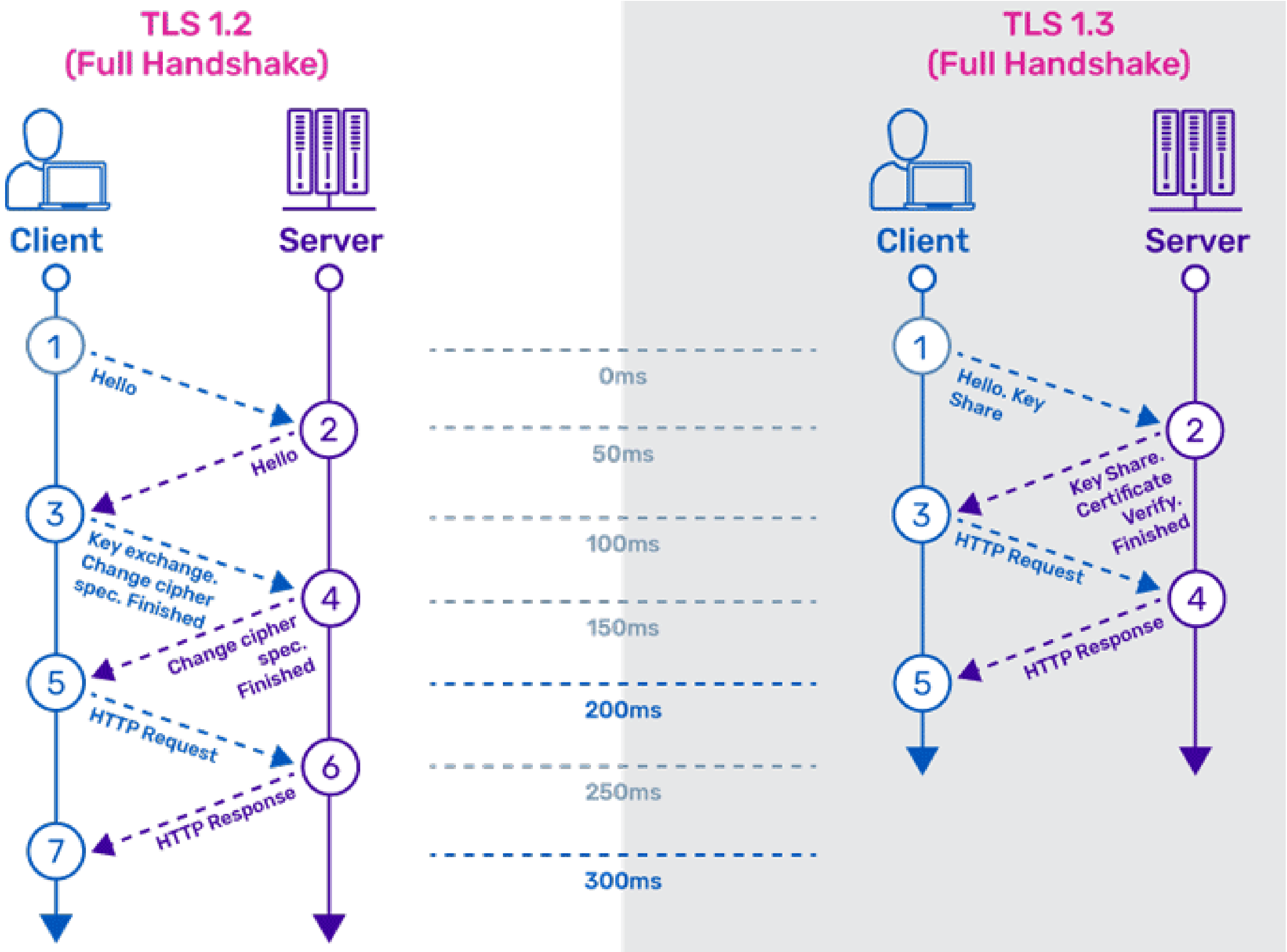
https://www.ssl.com/how-to/manually-generate-a-certificate-signing-request-csr-using-openssl/

Overview of TLS

- SSL (Secure Socket Layer) is a protocol to:
  - secure a data connection between server and client
    → Confidentiality, authenticity, and integrity
  - using both public key and symmetric key cryptography
  - over an insecure network, e.g., the Internet
  - Developed by Netscape in 1994, was renamed to TLS after SSL 3.0
  - Latest version: Transport Layer Security, TLS 1.3

- Architecture
  - TLS Record Protocol
  - TLS Handshake Protocol
  - TLS Change Cipher Spec Protocol
  - TLS Alert Protocol
  - See Full Explanation here

- TLS 1.2 is yet to be deprecated
  - Redefined in [RFC 8446](RFC 8446)
  - Version downgrade protection
  - https://tls12.xargs.org/#client-application-data
- Forward secrecy (TLS 1.3)
  - Past sessions remain secure even the key is leaked (in future)
- All handshake messages after the ServerHello are encrypted.
  - Not in TLS 1.2
- Better KDF, + EC algorithms
  - more efficient

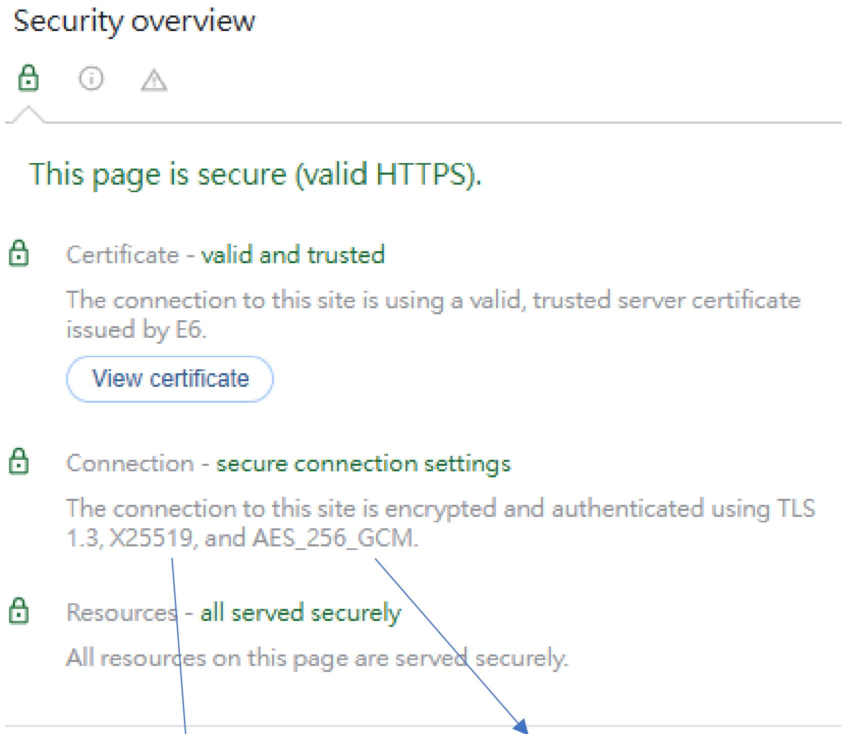


https://www.a10networks.com/glossary/key-differences-between-tls-1-2-and-tls-1-3/

TLS 1.2 vs 1.3

Table 2 A comparative study of TLS v 1.2 and TLS v 1.3 handshake

Factors	TLS v 1.2	TLS v 1.3
R.F.C number	5246 [16]	8446 [17]
Handshake messages	Handshake messages are not encrypted	Handshake messages are encrypted after the server hello step
RTT (round trip time) [23]	2-RTT requires for completing handshake	1-RTT requires for completing handshake
Communication Time [23]	It takes more load time	It takes less load time
PFS (perfect forward secrecy) [24]	It does not provide PFS	It provides PFS
Cipher suites [26]	Support complex cipher suites	Support simplified and secure cipher suites
SNI (server name indication) [27]	SNI is un-encrypted	SNI is encrypted
Cryptographic algorithms	Legacy algorithms are used	Authenticated encryption with associated data (AEAD) algorithms are used
Passive interception	Content type is not encrypted	Content type is encrypted
Latency	Encryption latency is higher	Encryption latency is lower



https://en.wikipedia.org/wiki/Galois/Counter_Mode

elliptic curve Diffie-Hellman key exchange

From: Handshake Comparison Between TLS V 1.2 and TLS V 1.3 Protocol. In Cyber Security in Intelligent Computing and Communications 2022.

TLS Handshake

- Establish shared key; using key exchange protocol or public key cryptography (obsolete)
- Agree on algorithms, modes, parameters
- Perform authentication (e.g., signature under certificate public key)

- TLS 1.2:

1. ClientHello: Nonce N_c+ version, session ID (for resumption), cipher suite, …
   - Key exchange algorithm, authentication (signature), encryption+MAC
2. ServerHello: Nonce N_s, certificate [+ CSR to verify client], KE message.
3. Client replies with KE msg, MAC on handshake transcript [Finish]
4. Server replies with MAC on handshake [Finish]

- Nonce is used to prevent replay.
- Renegotiation attack (not allowed in TLS 1.3)

- TLS 1.3: "combine 1,3; 2+4"
  - allow resumption for 0-RTT (preshared key)

```
uint16 ProtocolVersion;
opaque Random[32];

uint8 CipherSuite[2];    /* Cryptographic suite selector */

struct {
    ProtocolVersion legacy_version = 0x0303;    /* TLS v1.2 */
    Random random;
    opaque legacy_session_id<0..32>;
    CipherSuite cipher_suites<2..2^16-2>;
    opaque legacy_compression_methods<1..2^8-1>;
    Extension extensions<8..2^16-1>;
} ClientHello;
```

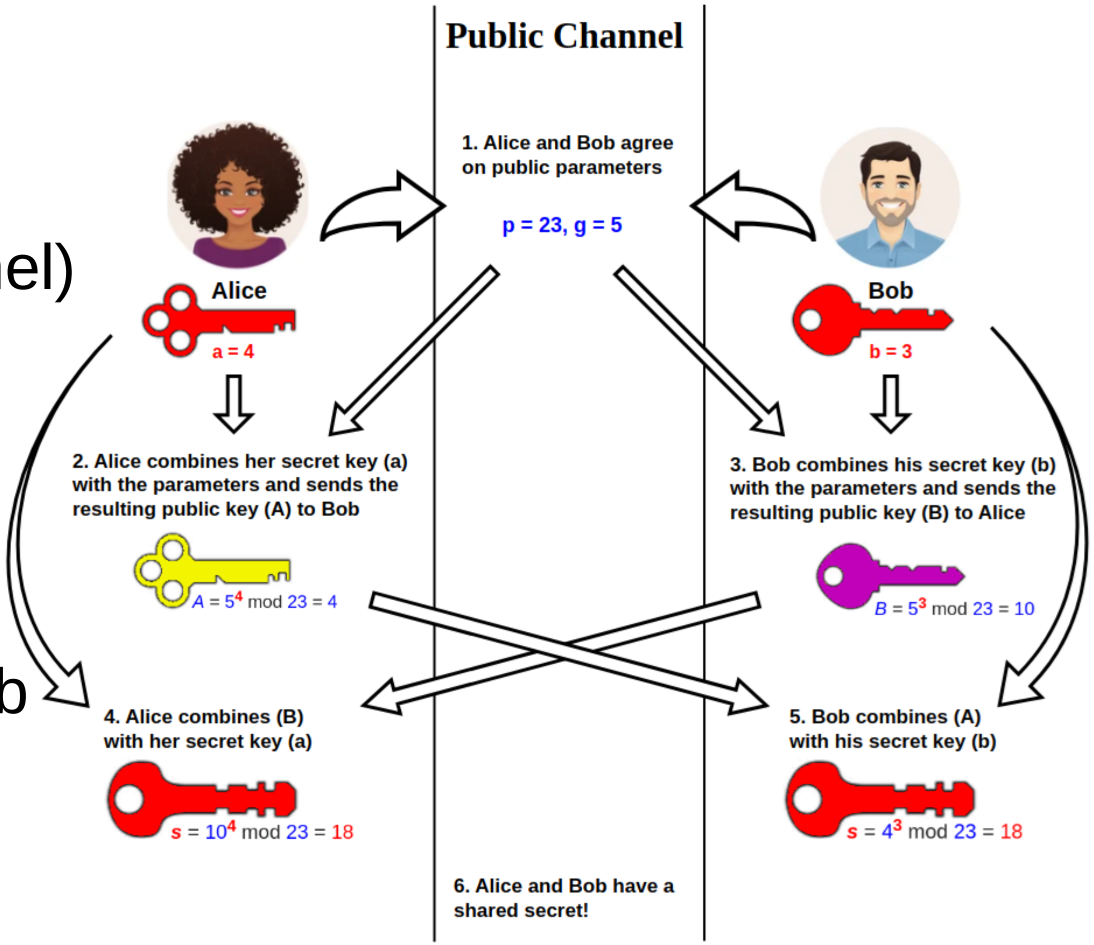
Cookies, key exchange msg, … are here backward-compatible

ServerHello is similar

https://www.youtube.com/watch?v=grRi-aFrbSE

Key Exchange in 1 Page

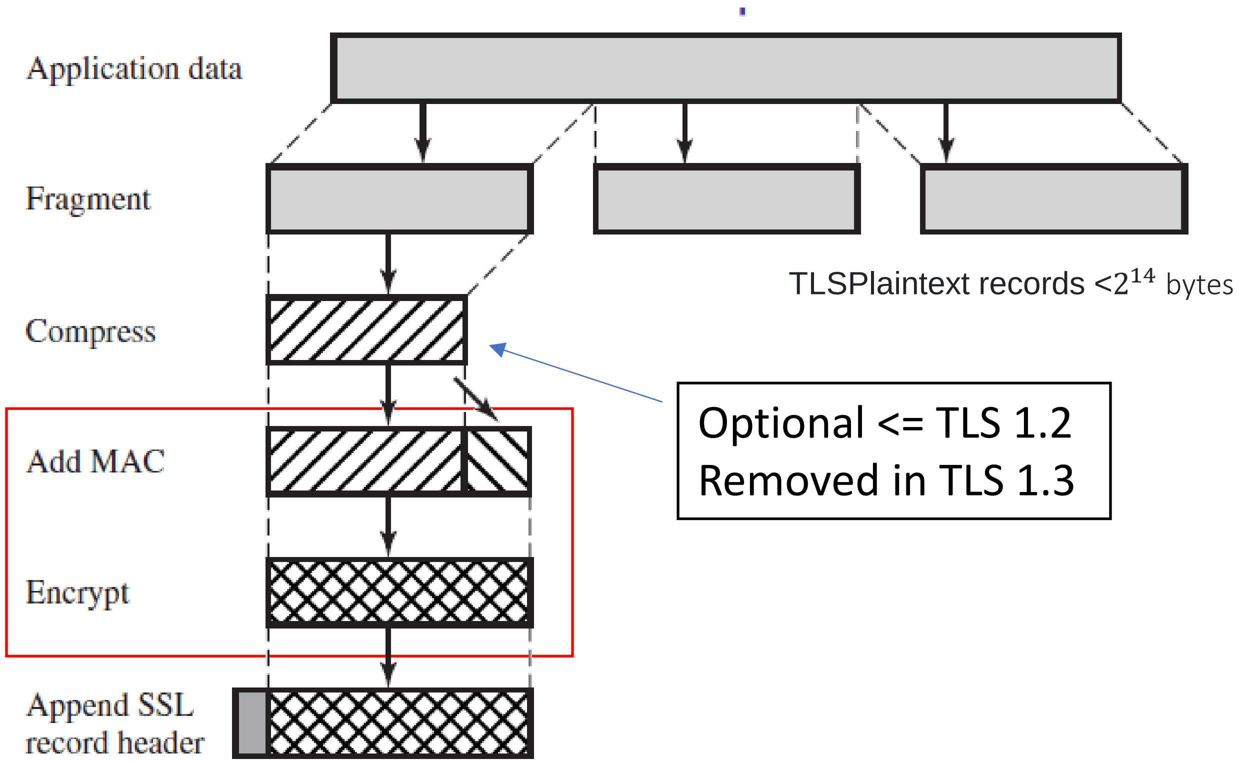
- Diffie-Hellman key exchange (RFC7919 for TLS)
- Rely on the hardness of discrete log
  - Finding x given $g^x \ (mod\ p)$; large p
  - Cannot find a, b from A, B (public channel)
  - Elliptic curve variant (ECDHE) for efficiency
- "Ephemeral": Temporary
  - The shared key is valid in this session
  - still, should ensure Alice is talking to Bob
    - →e.g., sign on the public key "B"



https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

TLS Record Protocol Operations

- Transmit the message with secret key
  - KE in handshake
- MAC: message integrity
- Encryption:
  - message confidentiality
  - (note: symmetric key encryption algorithm will be used here)



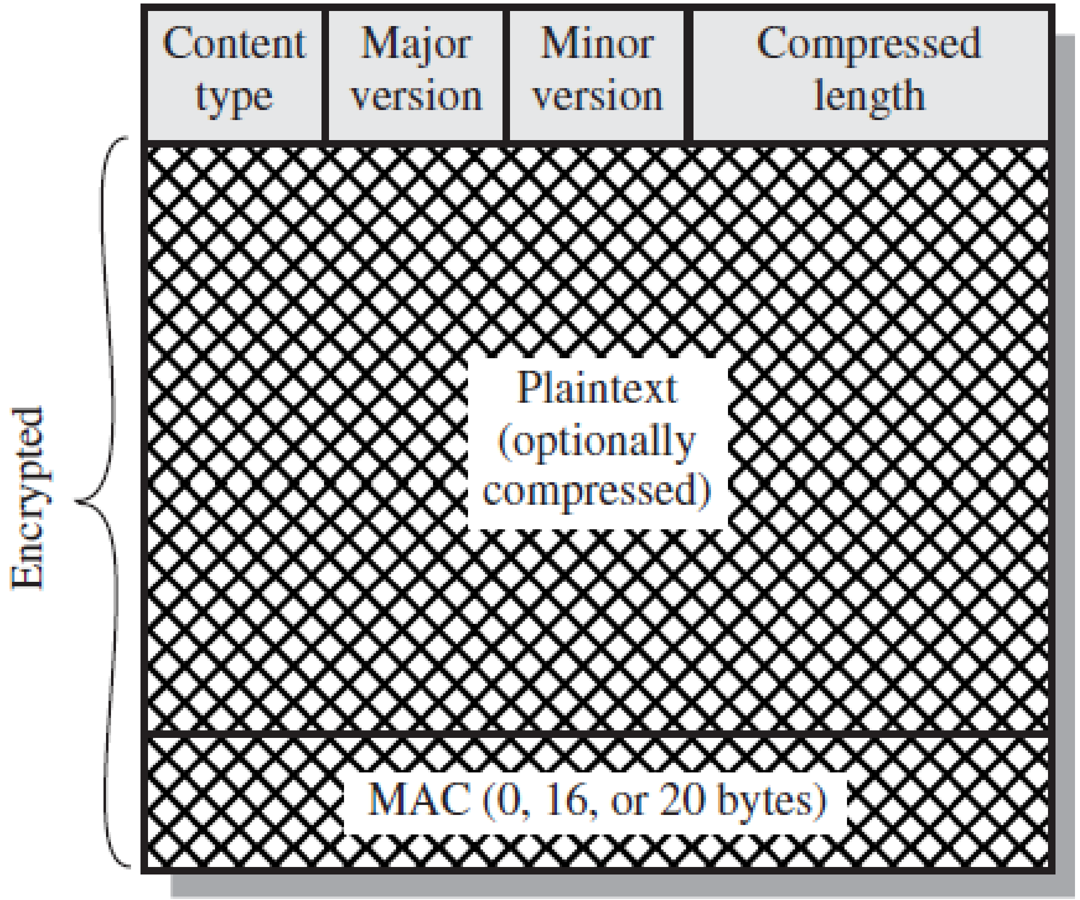
Authenticated Encryption (AEAD) in RFC 9367 (RFC 5116)
(depends on cipher suites)

Encrypt-then-MAC (provably secure): ENC(k, M), MAC(ENC(k, M)),

AEAD (key k, nonce, plaintext, AD): C = ENC(k, M), MAC(AD, C)

https://en.wikipedia.org/wiki/File:ChaCha20-Poly1305_Encryption.svg

TLS Record Format

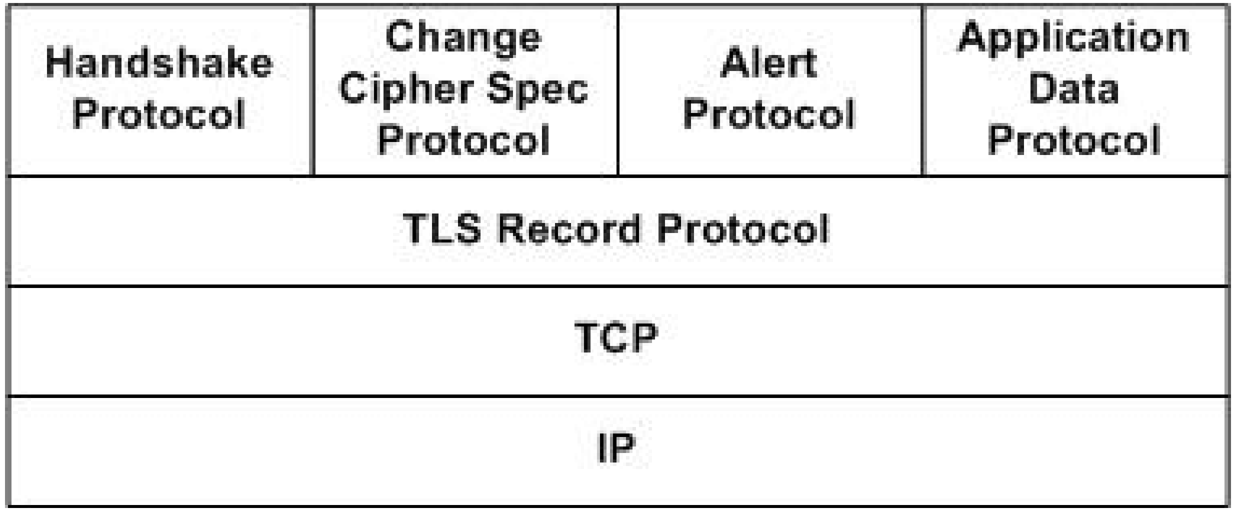



```
enum {
        invalid(0),
        change_cipher_spec(20),
        alert(21),
        handshake(22),
        application_data(23),
        (255)
    } ContentType;

    struct {
        ContentType type;
        ProtocolVersion legacy_record_version;
        uint16 length;
        opaque fragment[TLSPlaintext.length];
    } TLSPlaintext;
```

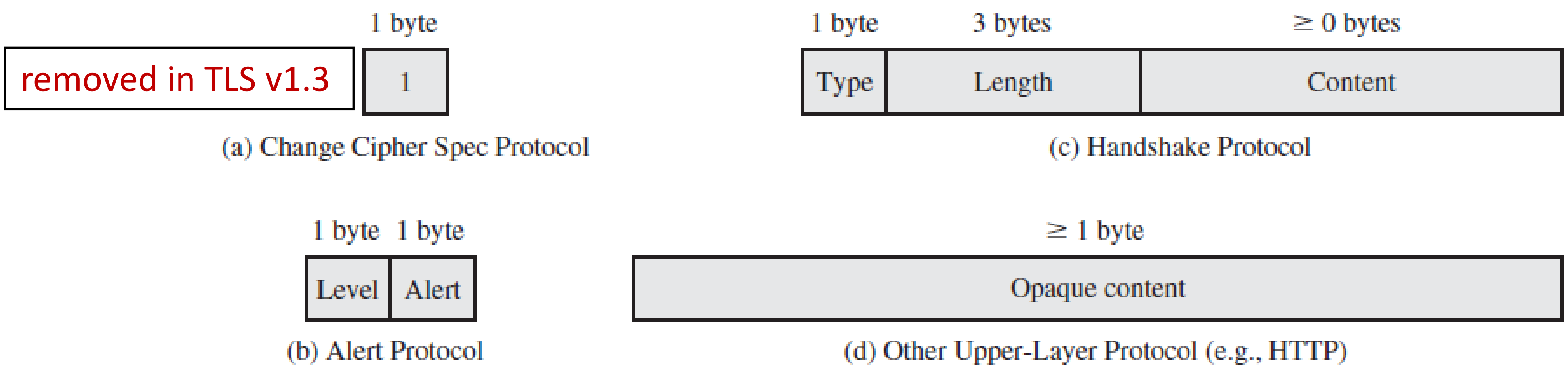
0x0303 for TLS v1.3
Length: 2^14 bytes max

TLS Architecture



- Application data could be data from other application layer protocols (like HTTP, FTP, etc.)
  - HTTP + TLS= HTTPS
- Some functionalities of TLS protocol itself rely on record protocol

TLS Record Protocol Payload



enum { warning(1), fatal(2), (255) } AlertLevel;

- ContentType = 20, 21 (alert), 22 (handshake), 23 (app data)
- More details in the RFC, e.g., AlertLevel and Description
  - (b) closure alerts (notify ending) and error alerts (end the connection)

How TLS Solve Security Problems Mentioned before?

- How to verify the server's identity
  - Each server has a certificate issued by a CA
  - The problem of verifying a server's identity is converted to verify the authenticity of this certificate
  - because CA will verify the server's identity before issuing that certificate and we trust the CA
  - Security of certificate verification is guaranteed by public key cryptography
    - → after hello/KE, everything is encrypted (TLS 1.3 vs cert is not encrypted in 1.2)
- How to achieve a secure communication over (public) Internet?
  - Message Confidentiality: through symmetric key encryption (AEAD)
  - Message Integrity: (keyed) Message Authentication Code/Signature
    - → Replay attack is prevented with a time stamp or unique packet sequence number
- How to ensure the integrity of client (i.e. the browser)?
  - In the same way as we verify the server's identity

TLS implementations in browsers

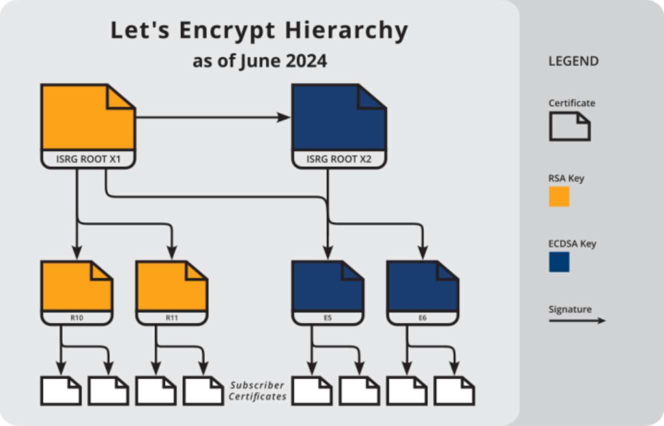
- Browser verifies the web server integrity
  - Checks and verify the server TLS certificate (for approved CA lists)
- How?
  - Common Name checking: does it match the domain name of web server
  - Validity period checking: must fall into validation time and date window (not before and not after)
  - **Signature** checking: using issuer's certificate to verify the integrity of Website's certificate

  1. Get issuer's public key from issuer's certificate
  2. Decrypt the digital signature in website's certificate, get a hash value $v1$

     → Cert is a signature on the hash of the certificate body
  3. Calculate hash value for website's certificate body (i.e., common name, period of validity, public key, etc.), get a hash value $v2$
  4. Pass if $v1 == v2$ (and signature verifies), otherwise verification failed
- But how can we trust the issuer's certificate and public key?
  - It is a problem of finding Root of Trust
  - Normally we trust certificates (pre-)installed with OS
  - So be very careful about manually installing a certificate!!!

Chain of Trust

- A TLS certificate should be signed by a trusted CA
  - Trusted CA also has a certificate, then who issue it?
  - Look at the browser CA list; they also use CAs from your OS
    - → e.g., chrome://certificate-manager/crscerts
    - → vulnerability if someone install malicious CA certs https://attack.mitre.org/techniques/T1553/004/
  - what if some CAs are compromised? e.g., not checking the domain ownership?
    - → Certificate Transparency: log all issued certificates (but rather "new")
    - → https://developer.mozilla.org/en-US/docs/Web/Security/Certificate_Transparency
    - → Revocation



https://letsencrypt.org/certificates/

TLS implementations in browsers (cont.)

- Demo: Handling of Certification inside browser
  - Indication of web sites that use TLS
  - Inspection of certification manually with developer tools
- Basic principle: secure browsing by default
- For example:
  - How to handle websites without using TLS
    → Allow normal browsing, but no downloading by default
  - How to handle websites with invalid certificates?
    → Forbidden browsing by default (without asking user)
    → Is this a good practice? How about usability?
- How to handle websites with mixed (secure and insecure) contents
  - Disable active mixed contents (i.e., contents with script, link, iframe etc.)

Test for your web: https://www.jitbit.com/sslcheck/

Reference: https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content

TLS Deployment in Real-world

- not adopted by some websites, even though it provides good security
  - A list of popular websites not using SSL/TLS: https://whynohttps.com/
  - May think it is unnecessary, e.g., no sensitive data
  - May need extra cost ($), effort and administrative overhead
  - May bring performance overhead (i.e., slower browsing speed)
- Over 95% websites (loaded in Chrome) use HTTPS
  - Google transparency report of HTTPS encryption on the Web
    - → https://transparencyreport.google.com/https/overview
  - 21 SSL Statistics that Show Why Security Matters so Much
    - → https://webtribunal.net/blog/ssl-stats
- But are they secure or using it correctly?
  - over 30% has inadequate security (https://www.ssllabs.com/ssl-pulse/)
  - supporting (up to) deprecated TLS/SSL version, e.g., SSL 3.0, TLS v1.0, 1.1
  - insufficient key size, e.g., below or equivalent to RSA 2048
- Most Browser does not allow HTTP2 without TLS (h2c; cleartext)
  - http://www.httpvshttps.com/ (the speedup is due to HTTP/2)
- Post quantum TLS, e.g., https://openquantumsafe.org/applications/tls.html
  - RSA, discrete-log-based cryptosystem would need a replacement
  - symmetric key primitives shall remain "quantum-safe", e.g., SHA3, AES256
  - https://blog.cloudflare.com/pq-2024/#migrating-the-internet-to-post-quantum-key-agreement

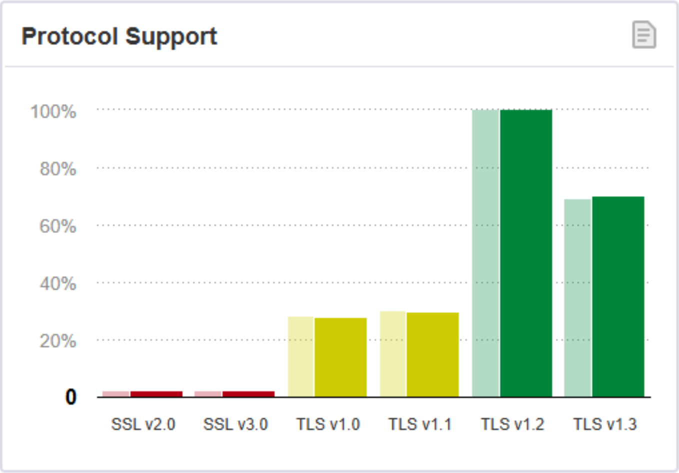
Some Common Security Problems when Deploying TLS

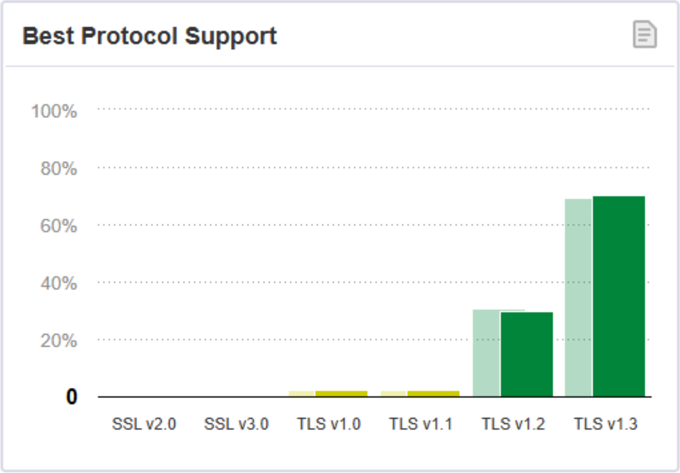
- Use invalid certificate (e.g., common name does not match domain name, self-signed, etc.)
- Certification expired (use automated tools like certbot)
- Mixed content (e.g., download HTTP contents inside an HTTPS page)
  - Browser will auto upgrade them, e.g., <img> <audio> <video> <source>
  - Block download HTTP content in HTTPS page, e.g., script, link, iframe (SOP), …
- No enough protection of cookies for HTTPS sessions (e.g., no secure flag)
- Insecure configuration
  - Weak algorithms (e.g., MD5 and SHA1 has become obsolete since TLS 1.2)
  - Down-gradable to previous insecure TLS/SSL versions
    → some known vulnerabilities: POODLE
  - E.g., Attackers may generate a fake certification by discovering collisions of hash values
  - What is collision, and how that works to generate a fake certification?

- More best practises can be found in https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices

To apply sufficient SSL protection

- Strong Algorithms and Secure Cipher Suites
  - Flaw: SSLv3 (and old TLS version) is insecure, yet still deployed





https://www.ssllabs.com/ssl-pulse/ based on Alexa’s list of the most popular sites in the world

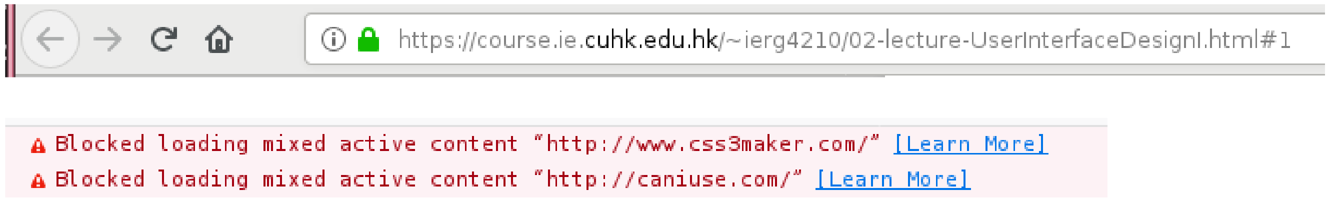
To apply sufficient SSL protection

- Strong Algorithms and Secure Cipher Suites
  - Weak Encryption still common, e.g., RSA1024
- Defense: Modify the mod_ssl setting

- Referring to OWASP Security Misconfiguration
  - Mitigation 1: Keep your software up to date
  - Mitigation 2: Ensure that the security settings in external libraries are understood and configured properly
  - You should also ship secure-by-default applications

Mixed Content (or mixed SSL)

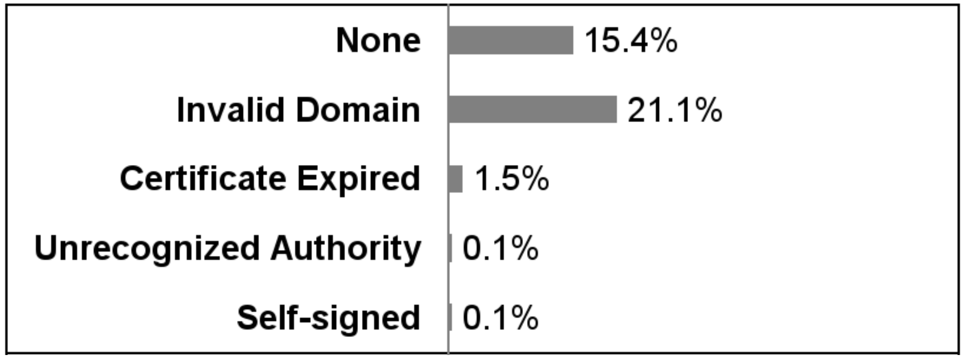
- When a HTTPS page embeds HTTP content
- Some browsers behave differently:
  - Some will block mixed contents silently

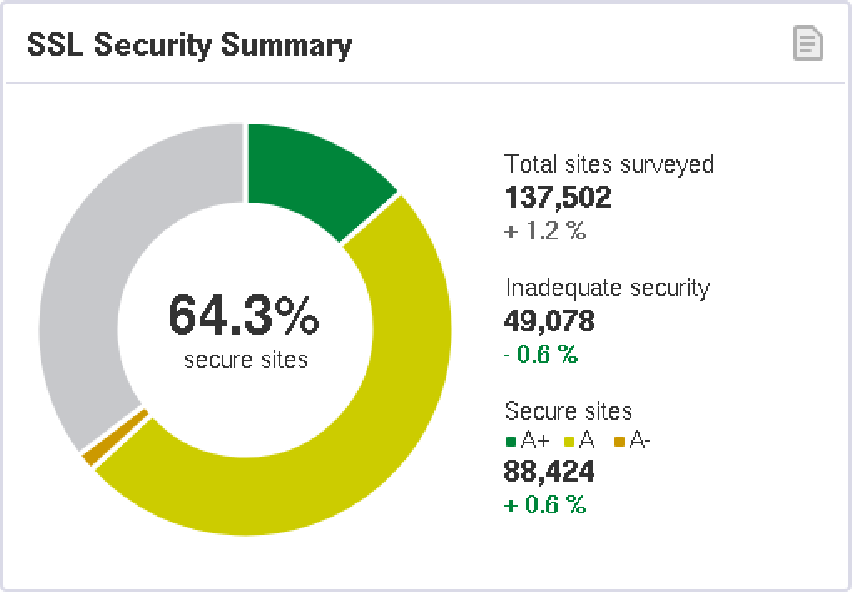


- Usability vs Privacy

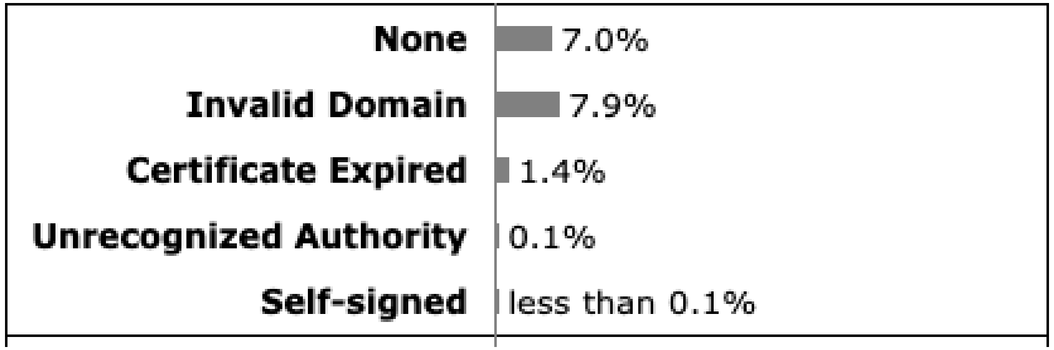
Snapshot of the Trend (2018->2021)

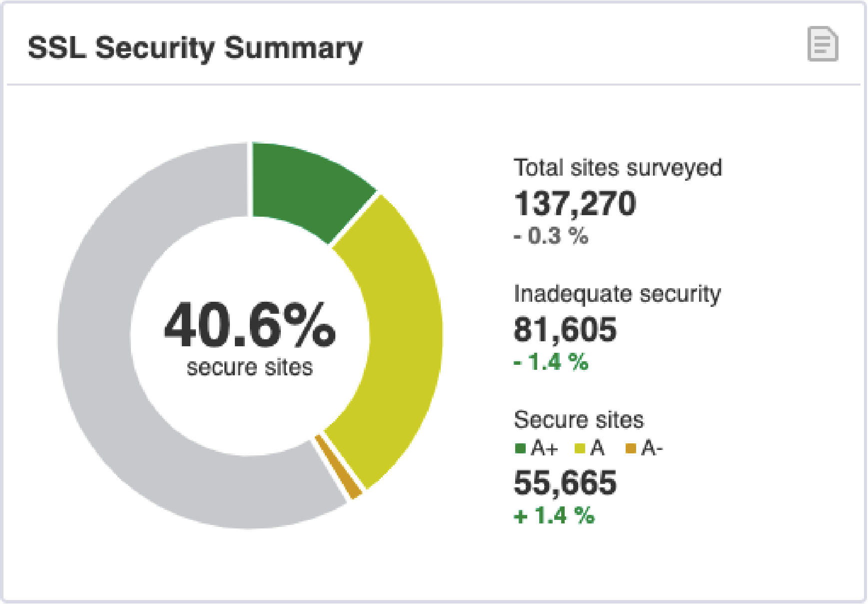
- 2018





2021





Reference: https://www.ssllabs.com/ssl-pulse/
https://w3techs.com/technologies/overview/ssl_certificate/all

Incredible SSL Certificate Statistics

- BuiltWith detects 156,979,428 SSL certificates on the internet.
- Nearly 21% of the Alexa Top 100,000 websites still don’t use HTTPS
- 6.8% of the top 100,000 sites still support (insecure) SSL 2.0 and 3.0 (from Watchguard).
- 68% of websites still support TLS 1.0 (from SSL pulse)
- 3.2% of the browsing time on Chrome is spent on HTTPS pages.
- 95+% of pages loaded in Chrome, on all platforms, were over HTTPS
- 53.5% of sites have inadequate security (from SSL pulse).
- 95.77% of all certificates on the internet are issued by just 9 authorities (from Censys).
- 1 in 10 URLs is malicious (from Symantec).

- •Reference: https://webtribunal.net/blog/ssl-stats

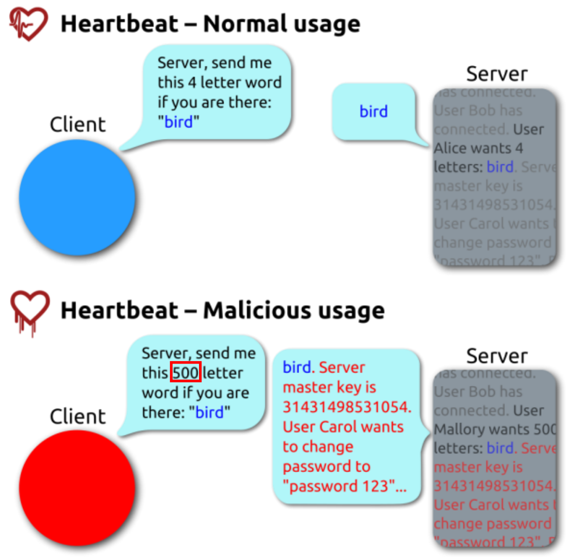
How to use TLS in browser by default

- Often we just type the domain name in the address bar of a browser without specifying what protocol to use
- The browser tend to use HTTPS as the default
  - https://blog.chromium.org/2023/08/towards-https-by-default.html
- The server should either:
  - Redirect HTTP request to HTTPS protocol at the landing page
  - Or use HTTP Strict-Transport-Security (HSTS)
- What is HSTS? How it works?
  - Basically, it contains some headers to tell browsers to automatically and locally convert HTTP requests to HTTPS requests before sending requests out. E.g.:

    Strict-Transport-Security: max-age=<expire-time>

- Reference: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

Typical Attacks to TLS



https://en.wikipedia.org/wiki/Heartbleed

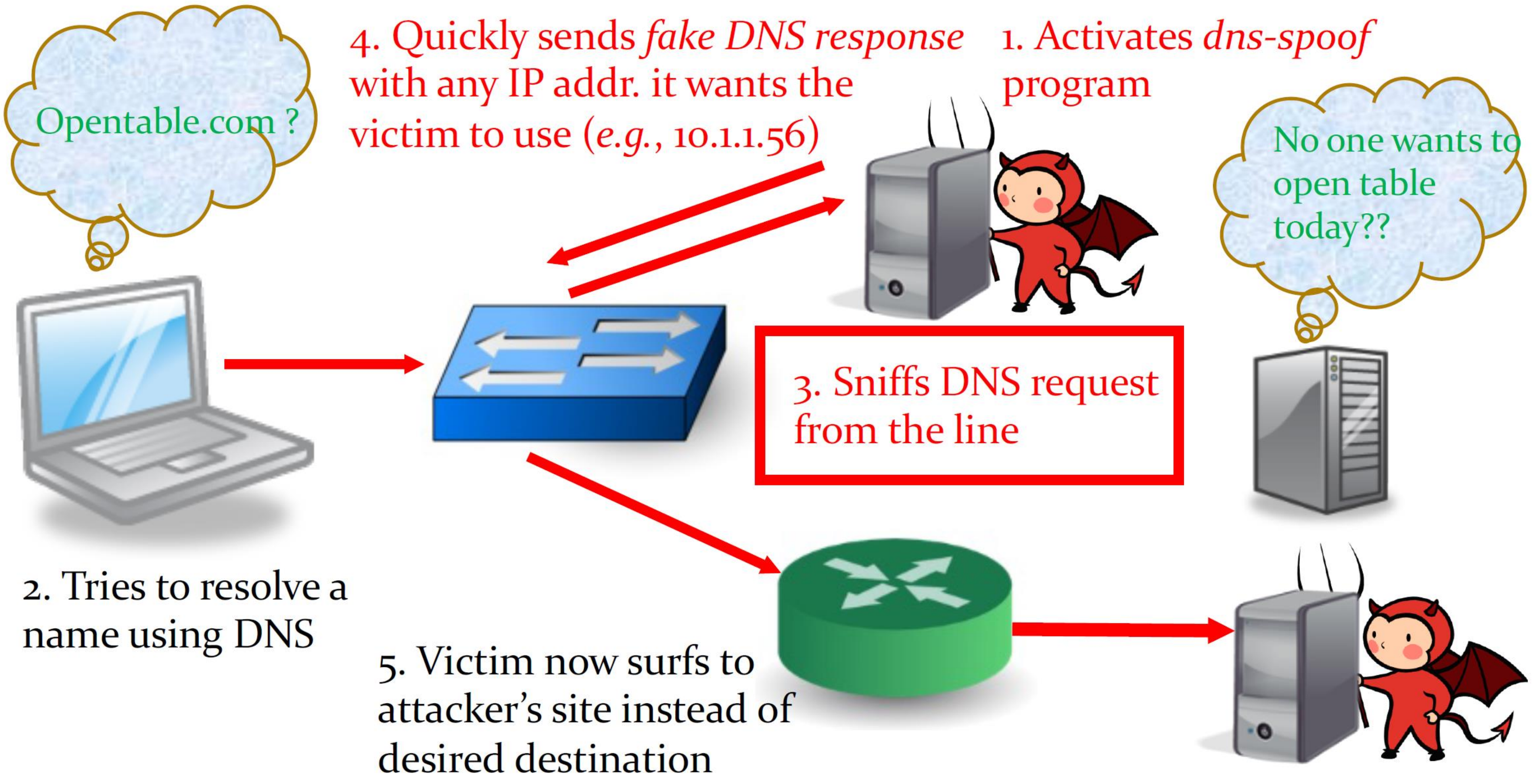
- Man-in-the-middle (MITM)?
  - What is it?
  - Attackers can launch the attack with fake certificate
- Downgrade Attack, POODLE, BEAST
  - Force client and server to fallback to older but vulnerable protocol versions
  - Reference: https://en.wikipedia.org/wiki/Downgrade_attack
- Side-channel Attack, e.g., CRIME in 2012 (CPA+ side channel info from compression)
  - Extract information from encrypted traffics by analyzing packet size, numbers, timing, etc.
  - Reference: http://oakland10.cs.virginia.edu/slides/sidechannel.pdf
- Website Spoofing Attack
  - A malicious website to mimic another legitimate website
  - The malicious website may have a valid certificate (for its own domain name, not the legitimate website)
  - People may be deceived if not checking the URL carefully
  - An example: https://9to5mac.com/2017/04/20/how-to-spot-a-phishingattempt-fake-apple-site/
- Attack on implementation:
  - Heartbleed bug: OpenSSL 1.0.1-1.0.1f; buffer over-read bug
  - CVE-2016-0701 insecure prime number used for DHE, affect < OpenSSL 1.0.2f
  - Insecure cryptography suite: RC4 attacks; nowadays disabled and prohibited
- HSTS Tracking: https://www.usenix.org/sites/default/files/conference/protected-files/foci18_slides_syverson.pdf
  - Mitigation: avoid Mixed content, restrict HSTS to set for the current host or TLD. https://kpwn.de/2023/03/http-strict-transport-security/

Scan your domain: e.g., using https://www.ssllabs.com/ssltest/

Secure Shell Protocol (SSH)

- Public key cryptography to authenticate the remote computer
- Authentication methods:
  - password
  - public key, e.g., DSA, ECDSA, RSA; can support X.509 certs
    - → important to verify the public key provided
  - also can have interactive authentication; RFC 4256
- Terrapin attack in 2023 https://terrapin-attack.com/
  - reduce the security of SSH by downgrade attack via MitM interception.
  - somewhat hard to carry out
- SSH3 (in future) leverage TLS 1.3, QUIC, …

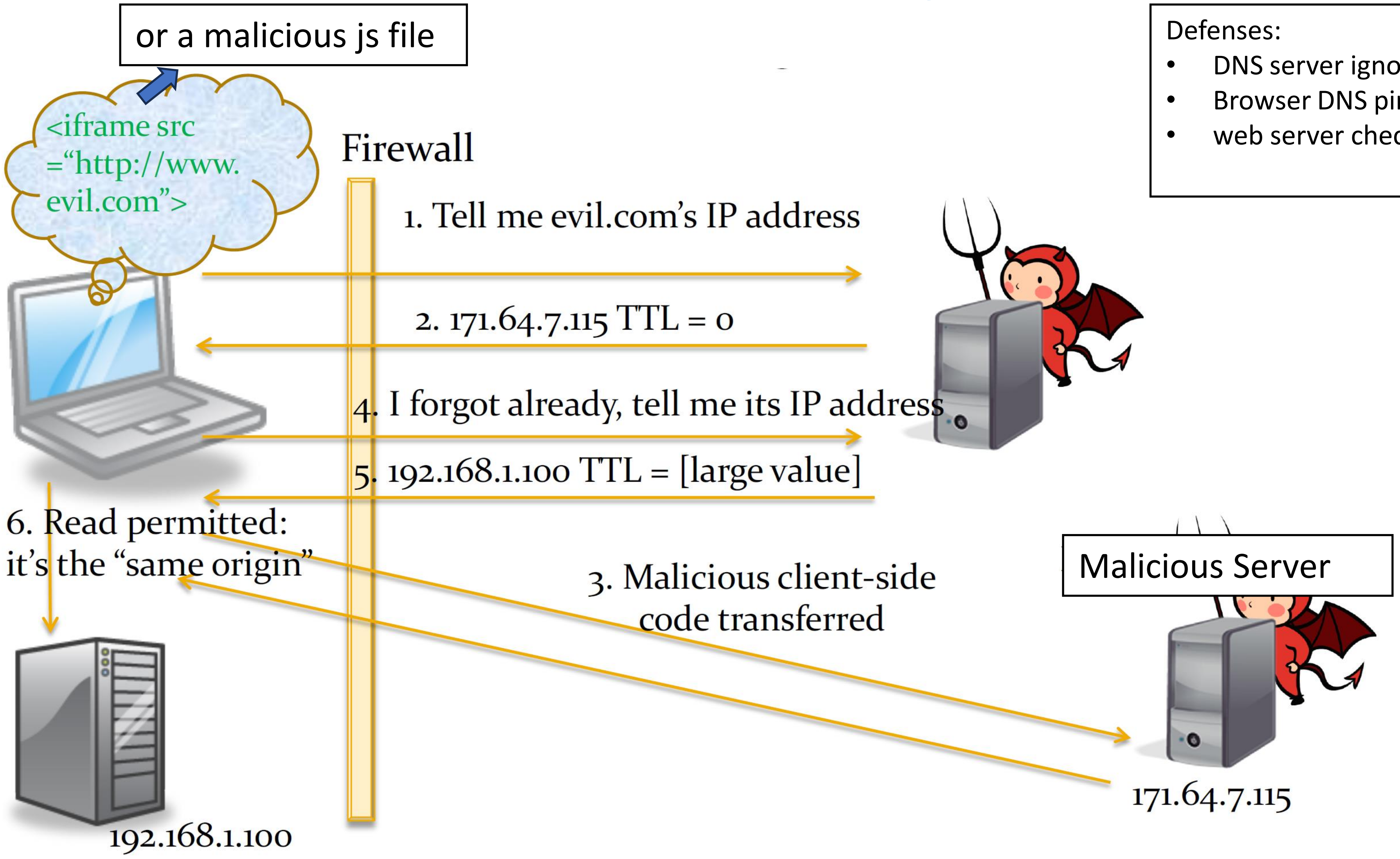
DNS Spoofing



Why is the attack possible/successful?

- I (or My own ISP's DNS server) asked the wrong DNS server
  - No authentication of the DNS server
- Why need to ask another DNS server?
  - Because my local cache / my ISP's DNS server does not know
  - (There is a time-to-live field (TTL) for a DNS record)
- Why can't I notice that I reached a wrong web server?
  - No authentication of the web server
- Secure solutions
  - use DNSSEC, but it's not that deployed or integrated with existing systems
  - use HTTPS, which shall trigger a warning
    - → except if the attacker presents a valid certificate under the domain (malicious CA?)
- More advanced attack: actively spoofing the DNS server
- But again rare in nowadays…
- Core of the attack: keep the wrong mapping
  - The attacker's goal is always to keep it as long lasting as possible?
  - Not necessarily, see the next attack

DNS Rebinding



Defenses:

- DNS server ignores local/private IP
- Browser DNS pinning
- web server checks Host header

extra: **New Techniques for Split-Second DNS Rebinding**
https://www.youtube.com/watch?v=uVGdZ-i2JeI