Stony Brook University

CSE 361: Web Security

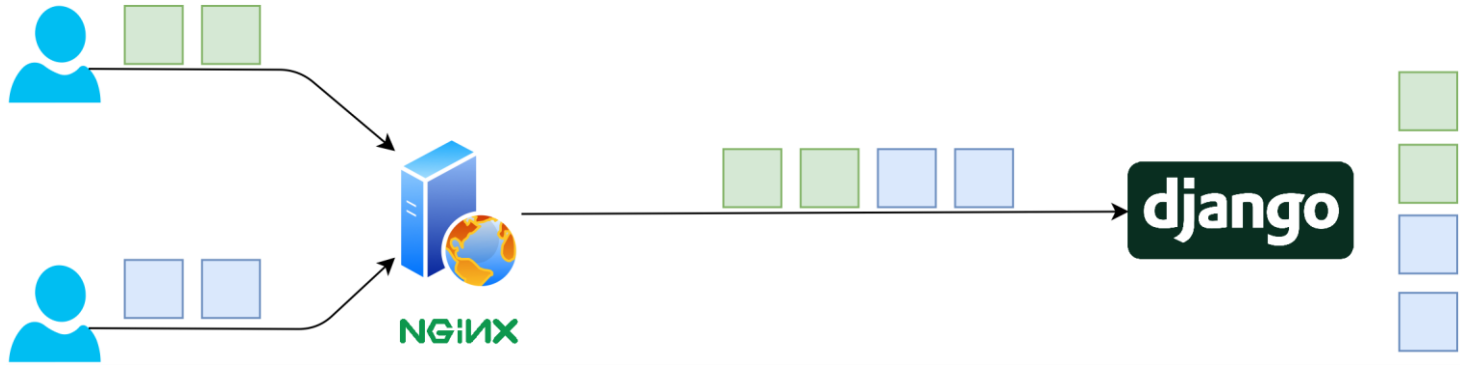
Infrastructure Security

Nick Nikiforakis

HTTP Desync Attacks

HTTP Front-End and Back-End Servers

- In real-world settings, you often have a reverse proxy
  - Multiple incoming requests (different TCP connections)
  - Single TCP connection between front-end and back-end servers



- What happens if nginx and Django have a different understanding of how long a HTTP request is?

HTTP: How does a server determine length of content?

- Option 1: Content-Length: $length header
  - Read the value, subsequently read $length bytes
- Option 2: Transfer-Encoding: chunked
  1. Read single line, treat as hexadecimal representation of $length for data to come
     - Stop reading if $length = 0
  2. Read $length bytes
  3. Go to step 1
- **What happens if you have both?**
  - RFC 2616 says: If a message is received with both a **Transfer-Encoding** header field and a **Content-Length** header field, the **latter MUST be ignored**.

Pitfalls in parsing HTTP headers

- What happens if front-end takes first occurrence of the header, but back-end takes the last?

```
POST / HTTP/1.1
Host: example.com
Content-Length: 6
Content-Length: 5

12345G
```

```
POST / HTTP/1.1
Host: example.com
Content-Length: 6
Content-Length: 5

12345G
```

```
POST / HTTP/1.1
Host: example.com
Content-Length: 6
Content-Length: 5

12345G
```

[https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn]

Pitfalls in parsing HTTP headers

- Now assume a second request, from a benign client

```
POST / HTTP/1.1
Host: example.com
Cookie: session=1234
Content-Length: 6

123456
```

```
POST / HTTP/1.1
Host: example.com
Content-Length: 6
Content-Length: 5

12345G
```

```
POST / HTTP/1.1
Host: example.com
Content-Length: 6
Content-Length: 5

12345
```

```
POST / HTTP/1.1
Host: example.com
Cookie: session=1234
Content-Length: 6

123456
```

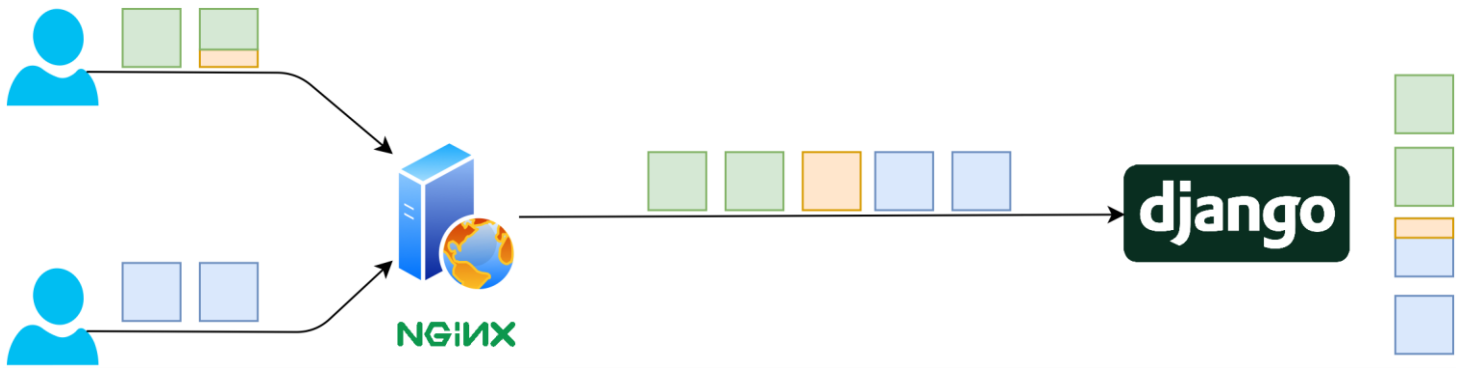
```
GPOST / HTTP/1.1
Host: example.com
Cookie: session=1234
Content-Length: 6

123456
```

[https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn]

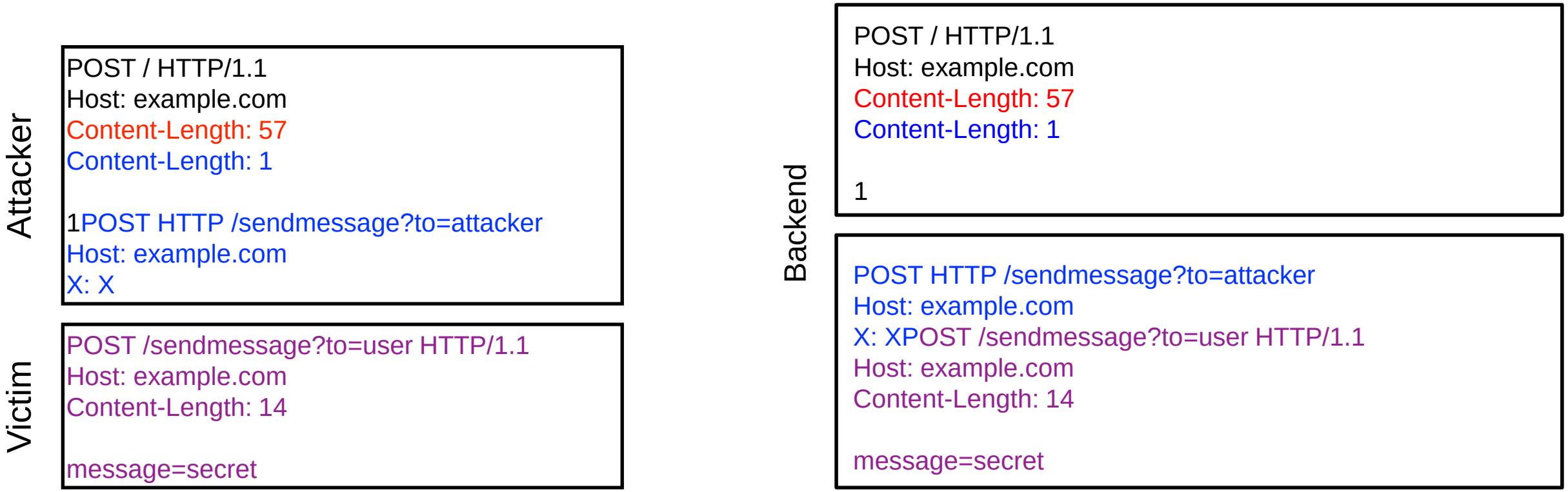
Result: Desync Attacks

- Front- and Back-end have different understanding of requests
  - Can smuggle in requests
  - Can hijack other user's requests



Result: Desync Attacks

- Front- and Back-end have different understanding of requests
  - Can smuggle in requests
  - Can hijack other user's requests



Additional problems in Desync attacks

- Some back-end systems look for the substring "chunked"
  - Attack: use Content-Length to fool front-end into accepting single request, use Transfer-Encoding: Xchunked to force TE for back-end

- Some back-end systems allow for whitespaces other than space
  - Transfer-Encoding:\tchunked ignored by front-end, understood by back-end

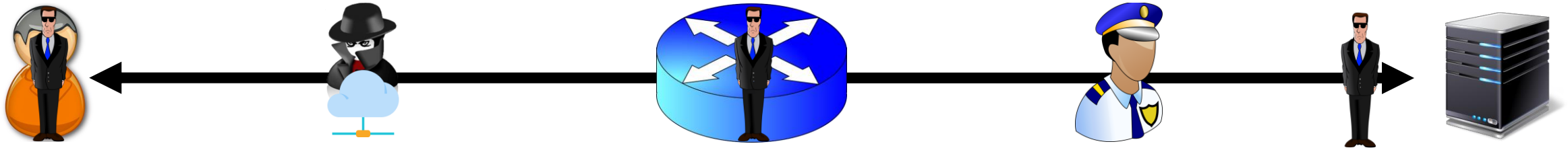
Transport Layer Security

Network Attacker

- Resides somewhere in the communication link between client and server
- Tries to disturb the confidentiality, integrity and authenticity of the connection
  - Observation of traffic (passive eavesdropper)
  - Fabrication of traffic (e.g., injecting fake packets)
  - Disruption of traffic (e.g., selective dropping of packets)
  - Modification of traffic (e.g., changing unencrypted HTTP traffic)
- "Man in the middle"

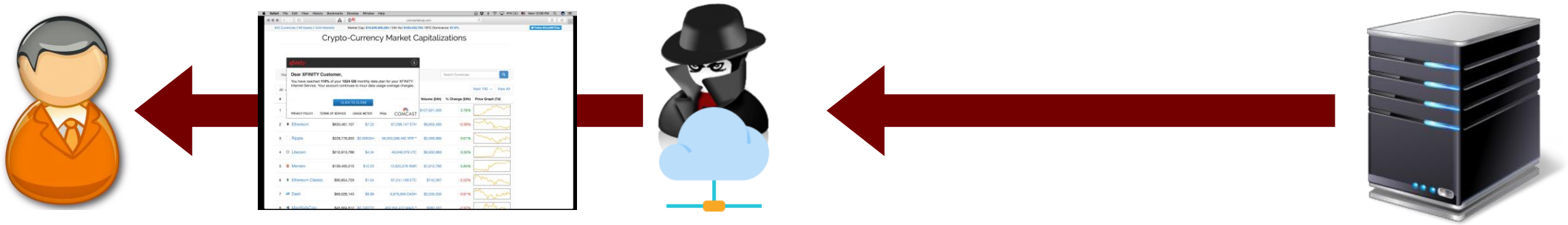
Possible types of a network attacker

- Within same network (ARP poisoning)
- Internet Service Provider (complete access to all traffic)
- Law Enforcement (access to traffic for specific user/to specific server)
- ... GCHQ, NSA, et al. (everywhere really)



Network attackers on the Web

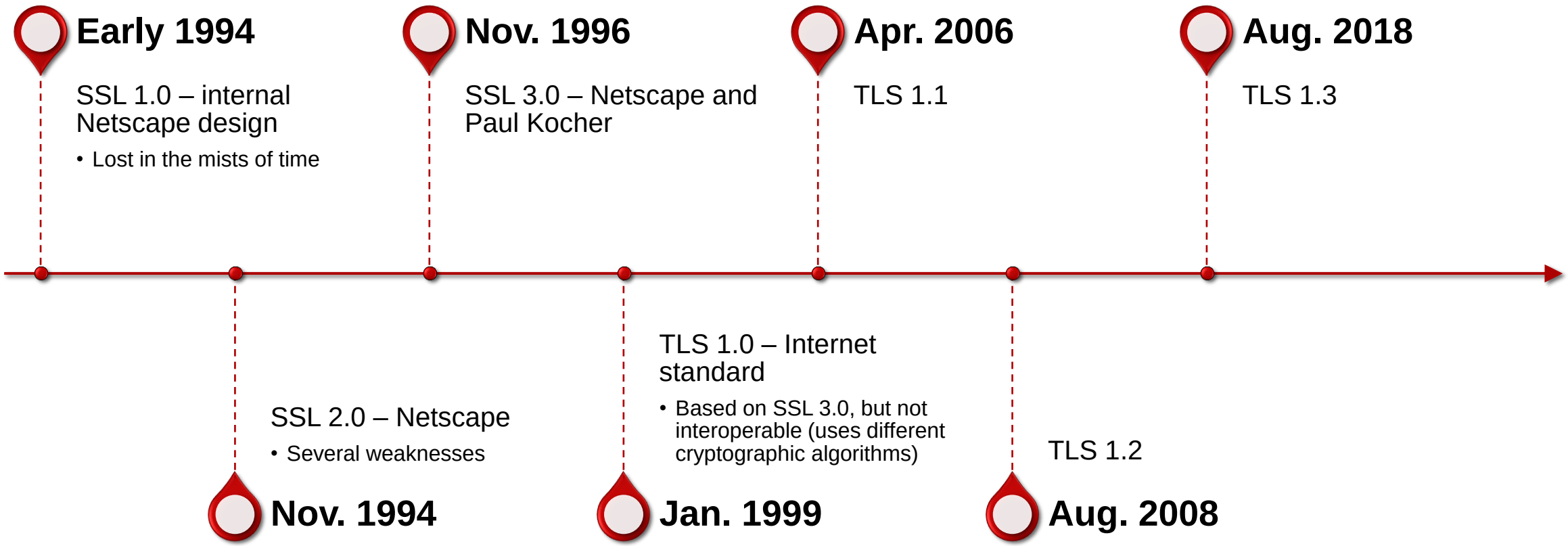
- Active attacker
  - tries to modify, e.g., the response
    - Comcast added advertisements into unencrypted sites
- Passive attacker
  - tries to eavesdrop on exchange to learn information
  - e.g., credentials



Security in HTTP

- There is no security in HTTP.
- Solution: encapsulate HTTP traffic in secure channel
  - used to be SSL (HTTP via SSL)
  - nowadays Transport Layer Security (TLS)
- TLS adds security to HTTP
  - end-to-end encryption
  - server authentication
  - optional client authentication (rarely used in practice)

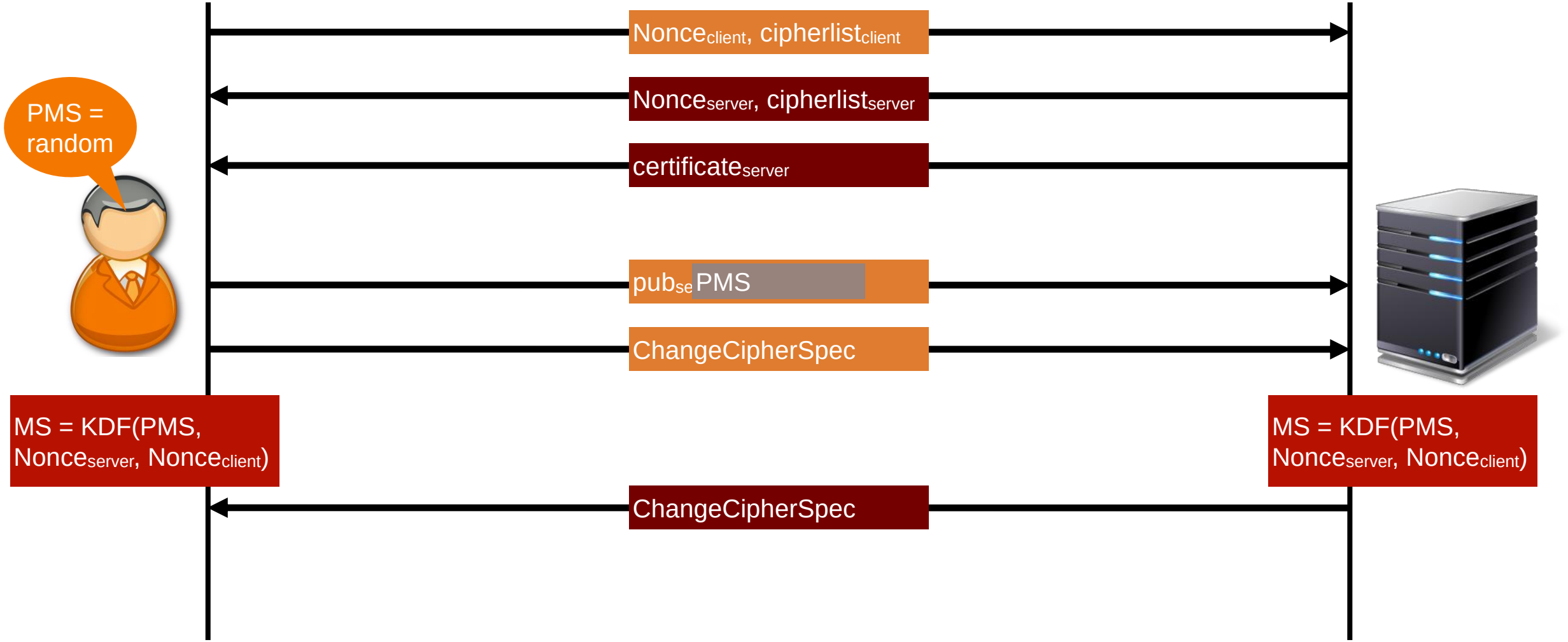
History of the Protocol



Transport Layer Security

- Replaces Secure Sockets Layer (SSL)
- Provides security for connection
  - integrity ensured by HMAC
  - confidentiality ensured by symmetric encryption
  - authentication with public-key cryptography
- Support numerous Cipher Suites
  - define key exchange, encryption and MAC types

(Very simplified) connection establishing in TLS with RSA



TLS Cipher Suites (RFC 5246)

Cipher Suite	Key Exchange	Cipher	MAC
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA
TLS_RSA_WITH_NULL_SHA256	RSA	NULL	SHA256
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	AES_128_CBC	SHA
TLS_RSA_WITH_AES_256_CBC_SHA	RSA	AES_256_CBC	SHA
TLS_RSA_WITH_AES_128_CBC_SHA256	RSA	AES_128_CBC	SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256	RSA	AES_256_CBC	SHA256
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DH_DSS	AES_128_CBC	SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA	DH_RSA	AES_128_CBC	SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DHE_DSS	AES_128_CBC	SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DHE_RSA	AES_128_CBC	SHA
TLS_DH_anon_WITH_AES_128_CBC_SHA	DH_anon	AES_128_CBC	SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA	DH_DSS	AES_256_CBC	SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA	DH_RSA	AES_256_CBC	SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DHE_DSS	AES_256_CBC	SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DHE_RSA	AES_256_CBC	SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA	DH_anon	AES_256_CBC	SHA

No protection

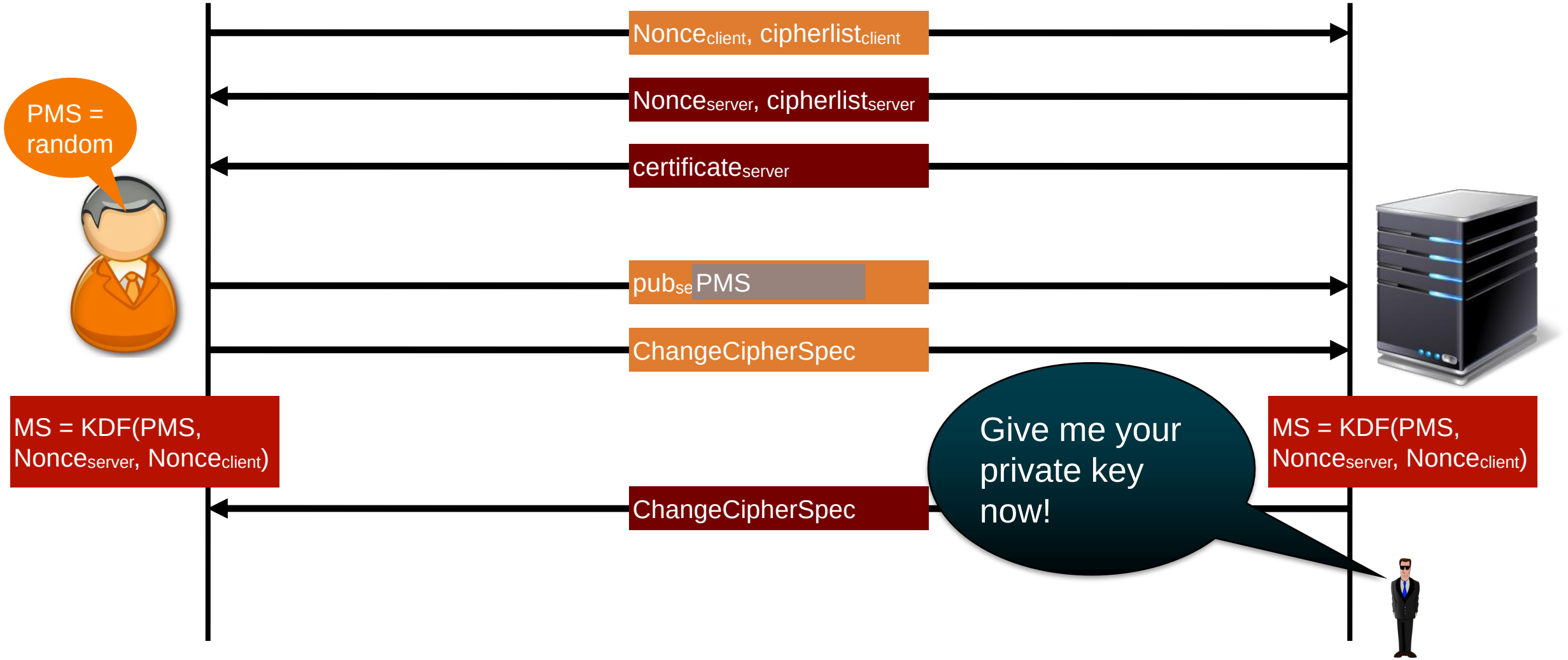
Uses RSA (certificate) for key exchange, AES 256 in CBC mode for encryption and SHA256 as MAC

Uses ephemeral Diffie-Hellman with RSA for key exchange, AES 256 CBC for encryption and SHA256 as MAC

(Very simplified) connection establishing in TLS with RSA

- Client sends list of available ciphers
  - server answers with his list
  - server selects first common suite
    - by default, uses client's priority, not its own
- Server transmits RSA certificate (including public key)
- Client generates PreMasterSecret, sends encrypted PMS to server
- Client and Server derive MasterSecret from PMS and nonces
- Crypto keys derived from MasterSecret used for data exchange

(Very simplified) connection establishing in TLS



Decrypting TLS traffic after the fact

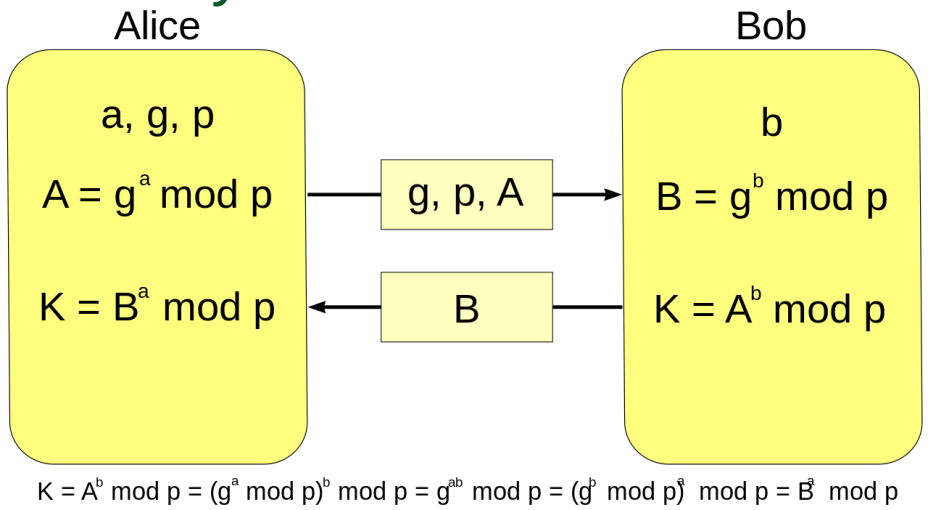
- Requirement for decryption: compute MasterSecret
  - based on nonces and PMS
- Both nonces are transmitted in clear text
- Client-generated PMS is encrypted with public key of server
  - if private key is compromised, attacker may decrypt previously recorded PMS
- Problem: server's RSA key does not change
- Solution: use ephemeral keys for key exchange to achieve forward secrecy

(Perfect) Forward Secrecy

- "Forward Secrecy" refers to inability to decrypt after the fact
  - key material is no longer available
- "Perfect Forward Secrecy" achieved if keys are never reused
  - compromising one key cannot compromise any other keys
- Desirable against Nation State Actors
- Requirement: generate ephemeral keys which cannot be recovered from the traffic
  - even if the private RSA key of the server is leaked

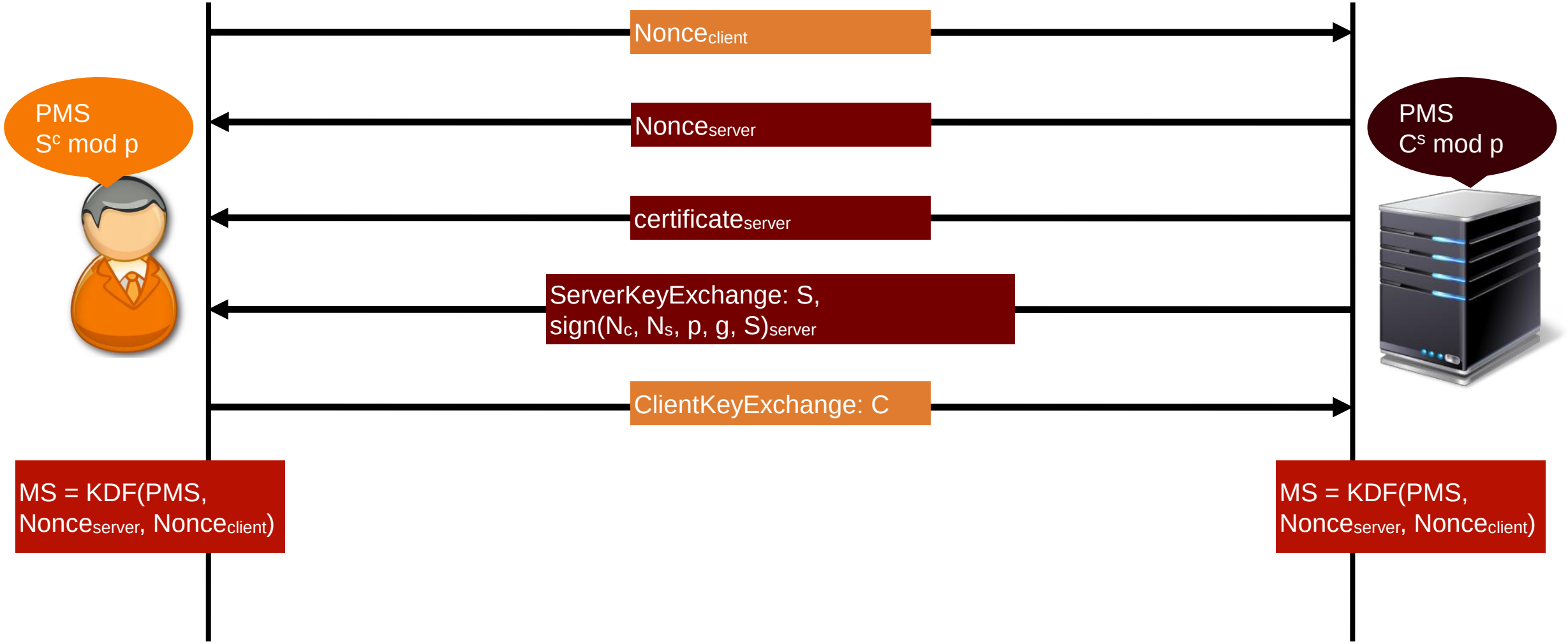
Reminder: Diffie-Hellman

- Two parties want to establish shared key
  - agree on common prime p and generator g (in TLS, given by the server)
  - parties generate private keys a and b
  - compute public keys as $g^{private} \bmod p$
  - knowing private keys allows to calculate common key K
- Bottom line: once you delete ephemeral private key, common secret cannot be recovered



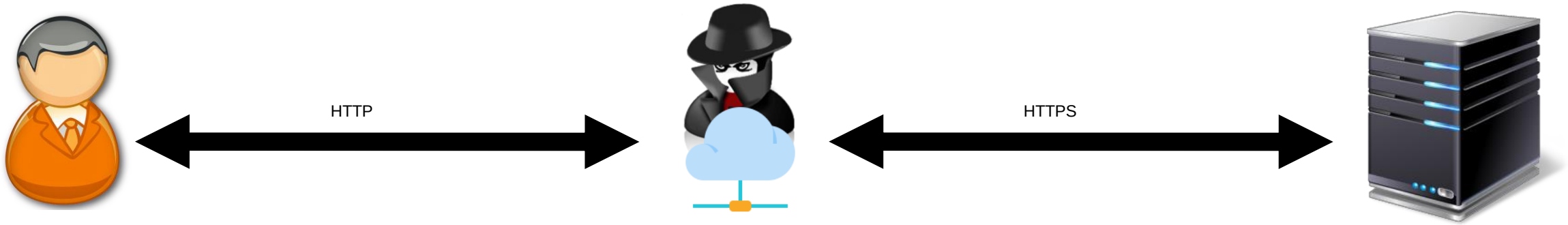
https://upload.wikimedia.org/wikipedia/commons/1/13/Diffie-Hellman-Schl%C3%BCsselaustausch.svg

(Very simplified) connection establishing in TLS with DHE



SSL Stripping

- Specific case of man-in-the-middle attack
  - presented at Blackhat 2009 by Moxie Marlinspike
- Goal: ensure that all traffic from victim is plaintext
  - attacker establishes HTTPS connection and forwards request
  - changes all links/redirects to HTTP
  - while rewriting HTML to show the target as HTTPS

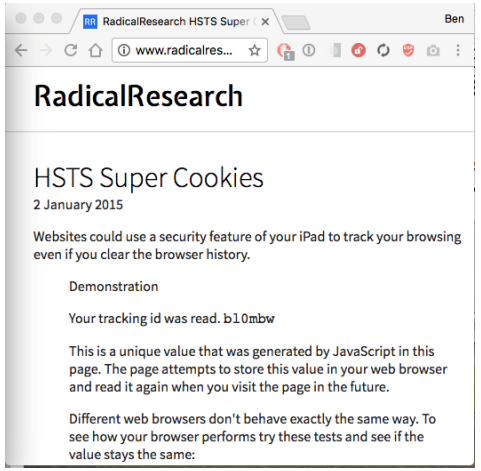


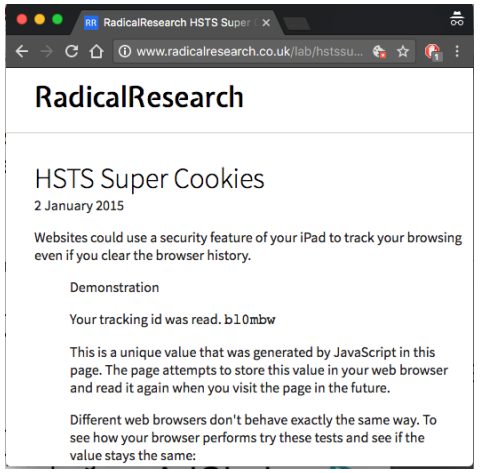
Forcing HTTPS: HTTP Strict Transport Security

- HTTP header (Strict-Transport-Security) send by server
  - only valid if sent via HTTPS
  - `Strict-Transport-Security: max-age=<expiry in seconds>`
    - `includeSubDomains`: header is valid for all subdomains
    - `preload`: allows for inclusion in preload list
  - ensures that site cannot be loaded via HTTP until expiry is reached
- Caveat: need to visit page once to get header
- solution: HSTS preload list (https://hstspreload.org/)
  - only possible with at least 18 weeks max-age, includeSubDomains and automatic redirect from HTTP

Abusing HSTS for Tracking purposes

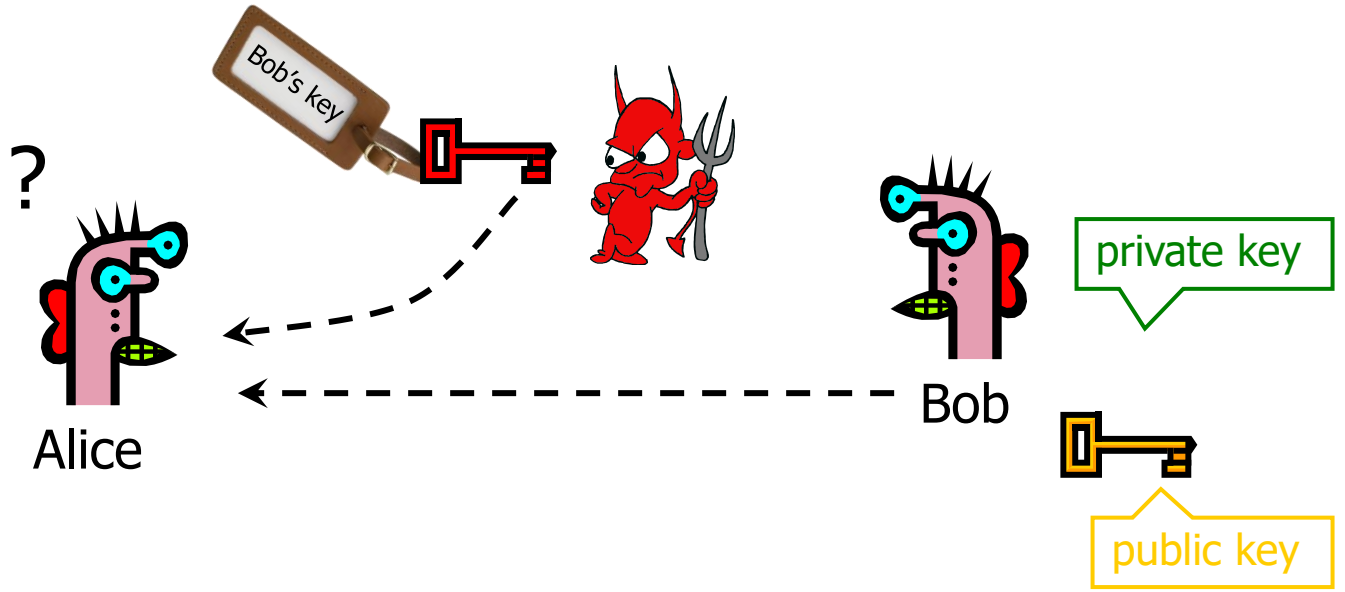
- HTTP Strict Transport Security controls connection behavior
  - once header is transmitted to client, all traffic to domain is forced via HTTPS
  - security feature, persisted across browsing sessions (also in incognito)
- Can be used to track user
  1. generate random ID for user on client
  2. for each bit set to 1, retrieve server resource
     - https://<bit>.tracking.attacker.com/set
     - server sets HSTS header for each response
  3. to read ID, include script from each subdomain
     - **<script src="http://<bit>.tracking.attacker.com/get"></script>**
     - HSTS-enabled subdomains automatically redirected by browser
     - on server, return id[<bit>] = 1 if script is accessed via HTTPS





HTTPS Certificates

Authenticity of Public Keys



Problem: How does Alice know that the public key she received is really Bob's public key?

Distribution of Public Keys

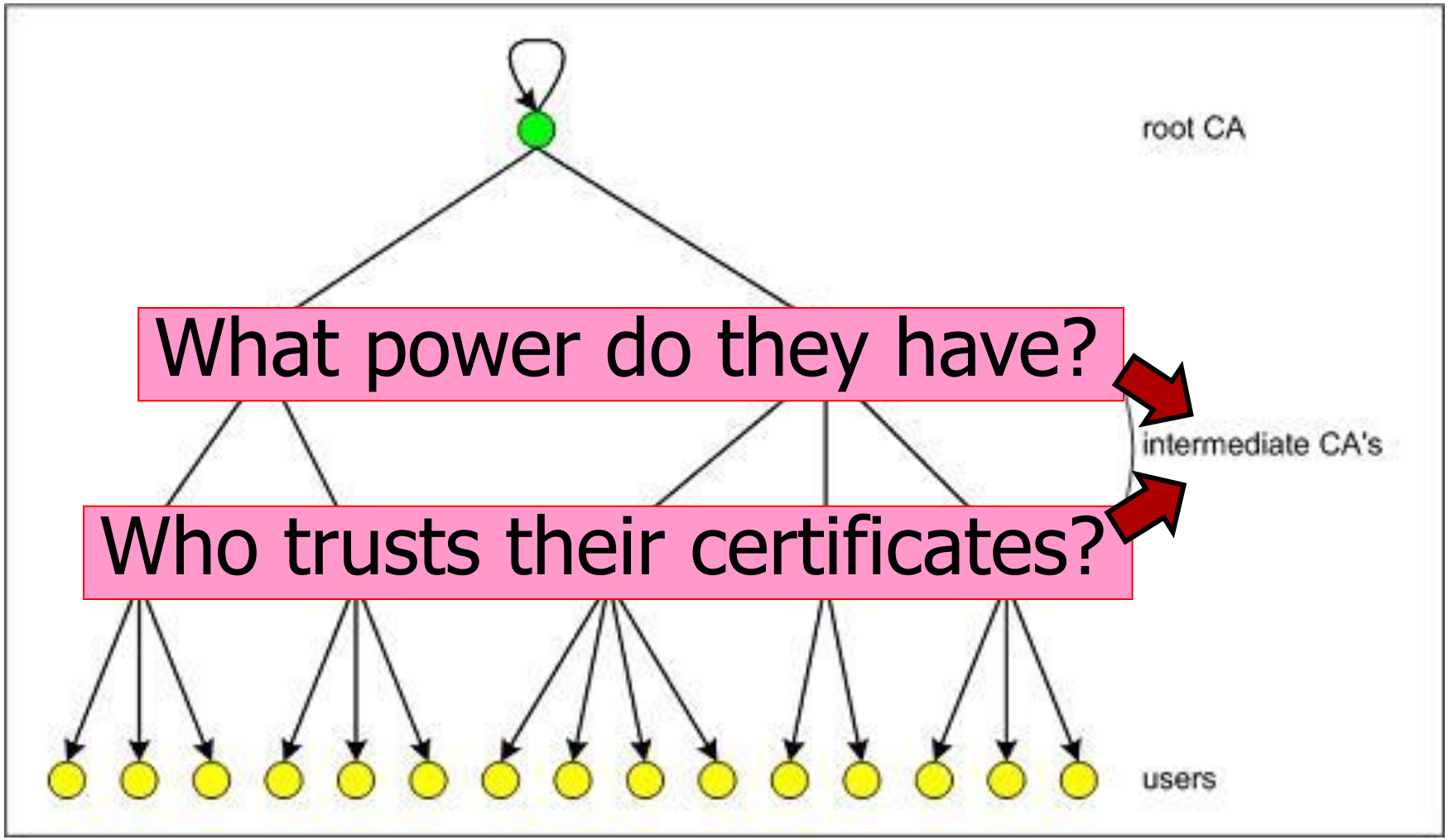
- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - $sig_{Alice}(\text{“Bob”}, PK_B)$
- Common approach: certificate authority (CA)
  - An agency responsible for certifying public keys
  - Browsers are pre-configured with 100+ of trusted CAs
  - A public key for any website in the world will be accepted by the browser if certified by one of these CAs

CA Hierarchy

- Are all Certificate Authorities (CAs), root CAs?
  - Do all websites have to get certificates from a root CA?

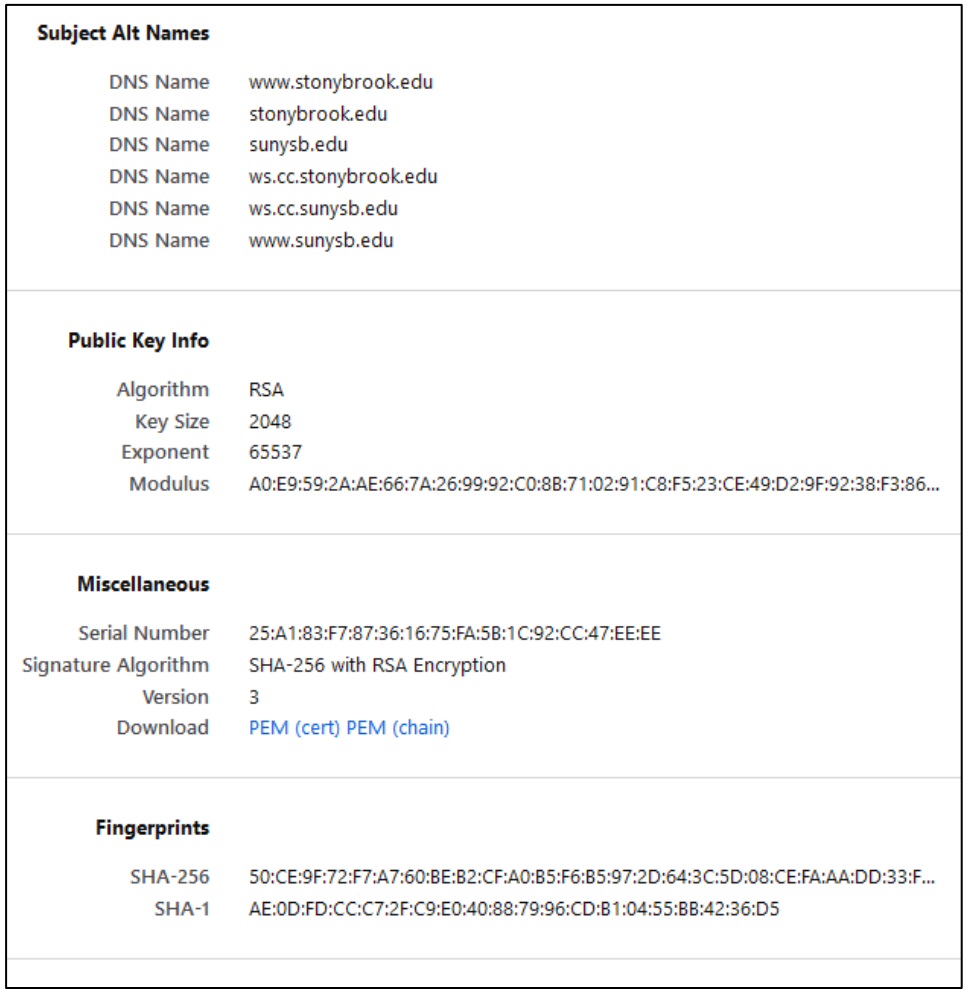
- A Root CA signs certificates for intermediate CAs, they sign certificates for lower-level CAs, etc.
  - Certificate "chain of trust"
    - $sig_{Verisign}$("SBU", PK_{SB}), sig_{SB}("Nick N.", PK_{Nick})
- CA is responsible for verifying the identities of certificate requestors, domain ownership

Certificate Hierarchy



Establishing trust in server's certificate

- Certificate is a
  - "digitally signed document that binds a subject to some other information ... identity certificates bind names to keys... " (Gollmann's Computer Security book)
  - "token that binds an identity to a cryptographic key" (Bishop's Computer Security: Art and Science)
- Need means to establish trust in certificate
- Solution: Public Key Infrastructure
  - implements a chain of trust to presented certificate



Subject Alt Names

DNS Name	www.stonybrook.edu
DNS Name	stonybrook.edu
DNS Name	sunysb.edu
DNS Name	ws.cc.stonybrook.edu
DNS Name	ws.cc.sunysb.edu
DNS Name	www.sunysb.edu

Public Key Info

Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	A0:E9:59:2A:AE:66:7A:26:99:92:C0:8B:71:02:91:C8:F5:23:CE:49:D2:9F:92:38:F3:86...

Miscellaneous

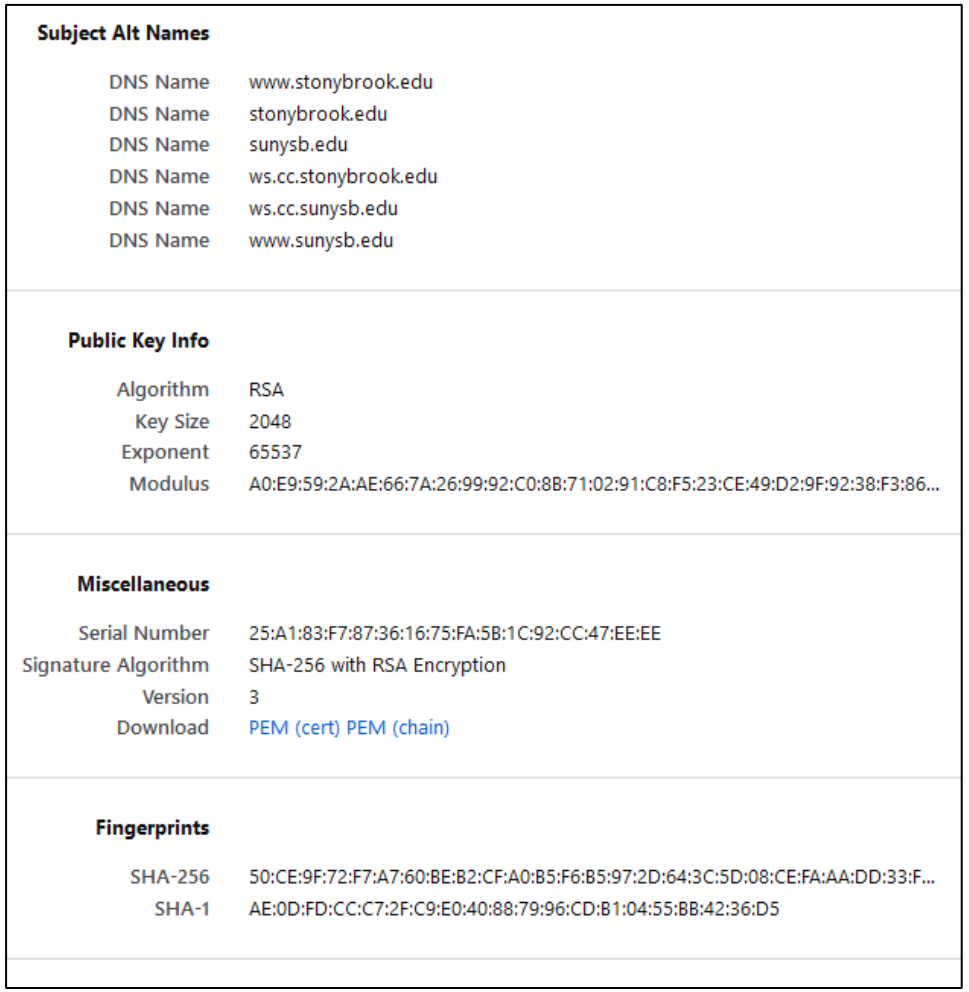
Serial Number	25:A1:83:F7:87:36:16:75:FA:5B:1C:92:CC:47:EE:EE
Signature Algorithm	SHA-256 with RSA Encryption
Version	3
Download	PEM (cert) PEM (chain)

Fingerprints

SHA-256	50:CE:9F:72:F7:A7:60:BE:B2:CF:A0:B5:F6:B5:97:2D:64:3C:5D:08:CE:FA:AA:DD:33:F...
SHA-1	AE:0D:FD:CC:C7:2F:C9:E0:40:88:79:96:CD:B1:04:55:BB:42:36:D5

Establishing trust in server's certificate

- Certificates bind key material to entity
- On technical level, certificate is a combination of
  - the server's public key,
  - its identity (domain name/common name),
  - and validity timestamps
  - (some more information on purpose of certificate and other technical details, e.g., allowed aliases)
- signed with issuer's (Certificate Authority) private key



Subject Alt Names

DNS Name	www.stonybrook.edu
DNS Name	stonybrook.edu
DNS Name	sunysb.edu
DNS Name	ws.cc.stonybrook.edu
DNS Name	ws.cc.sunysb.edu
DNS Name	www.sunysb.edu

Public Key Info

Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	A0:E9:59:2A:AE:66:7A:26:99:92:C0:8B:71:02:91:C8:F5:23:CE:49:D2:9F:92:38:F3:86...

Miscellaneous

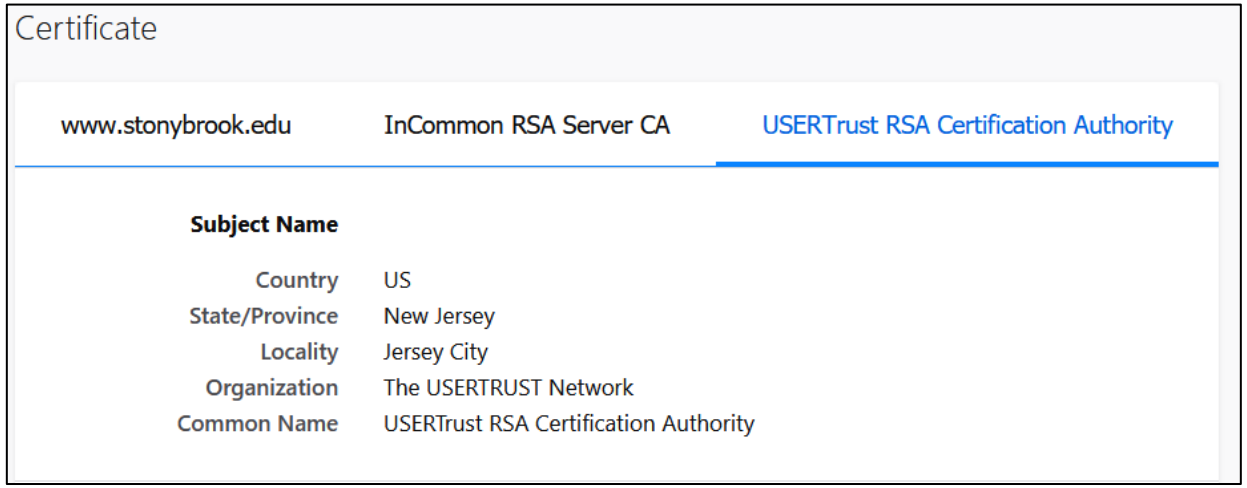
Serial Number	25:A1:83:F7:87:36:16:75:FA:5B:1C:92:CC:47:EE:EE
Signature Algorithm	SHA-256 with RSA Encryption
Version	3
Download	PEM (cert) PEM (chain)

Fingerprints

SHA-256	50:CE:9F:72:F7:A7:60:BE:B2:CF:A0:B5:F6:B5:97:2D:64:3C:5D:08:CE:FA:AA:DD:33:F...
SHA-1	AE:0D:FD:CC:C7:2F:C9:E0:40:88:79:96:CD:B1:04:55:BB:42:36:D5

Establishing trust in server's certificate

- Trusting a certificate boils down to trusting the CA
  - several root CAs integrated into browsers
  - CAs may be allowed to sign other CA's keys
- Example stonybrook.edu
  - certificate signed by InCommon RSA Server CA
  - ... which is signed by USERTrust RSA Certification Authority
  - ... which is in the browser



Root Stores

- The place where all the certificates of trusted certificate authorities (root CA certificates) are held is called a “root store”

- Different operating systems have slightly different processes for approving new CAs but the processes typically boil down to:
  - Comply with various baselines for CAs
  - Comply with local laws
  - Pass yearly audits
  - Show business value to user base of the OS

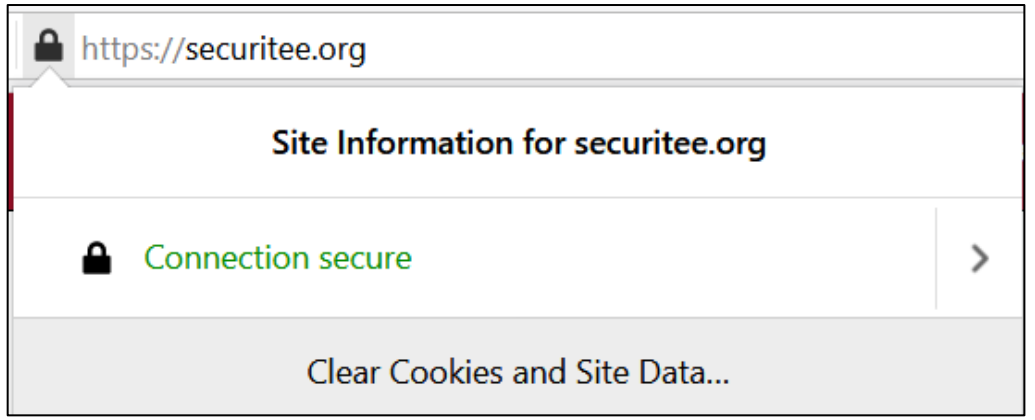
- Most applications use the root stores provided by the underlying operating system
  - On MS Windows, you can access it by typing “certmgr.msc” in the “Run” dialogue
- Firefox is a notable exception since it ships with its own root store
  - Linux distributions commonly use this root store

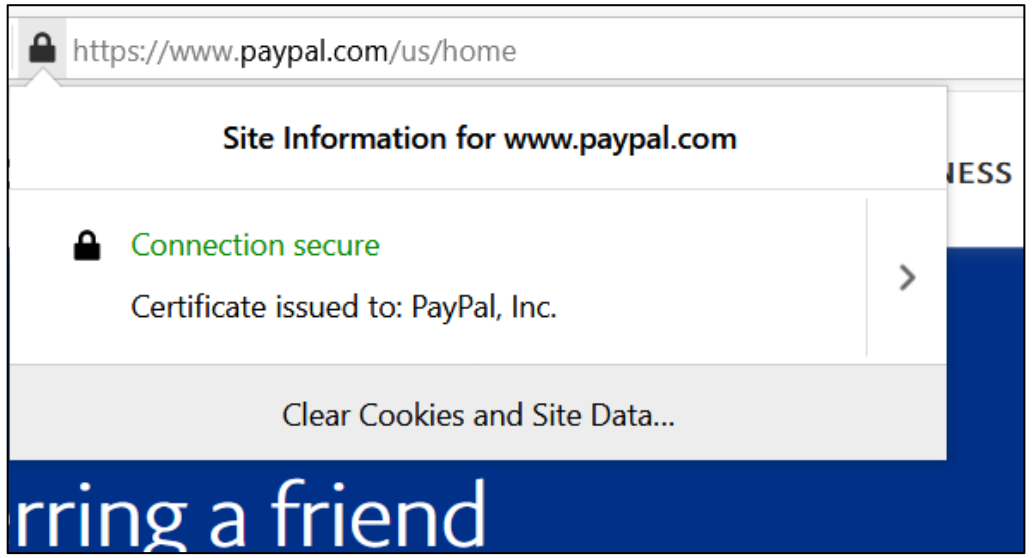
Interlude: Server Name Indication

- Single server may host several domains
  - TLS connection is made to IP address before Host header is sent
  - certificate validation is based on the domain name though
- In "good old days", only one certificate per IP possible
  - one certificate containing all domains hosted on the machine
  - horrible to maintain (e.g., adding a domain, revoking a domain's certificate)
- Solution: Server Name Indication
  - client sends desired domain name to server
  - server decides which certificate to present
  - widely adopted in browsers and servers since 2010

Validating ownership of a domain

- Most CAs use "domain validation"
  - send email to registrant (or webmaster@domain.com)
  - use DNS TXT entry with random token
  - ask owner to host file at certain path on their domain (Let's encrypt)
- More expensive solution: Extended Validation
  - requires proof of legal identity (e.g., Company) and physical location
  - Distinction no longer visible by default
- Problem: what if ownership changes?
  - need means to revoke certificate



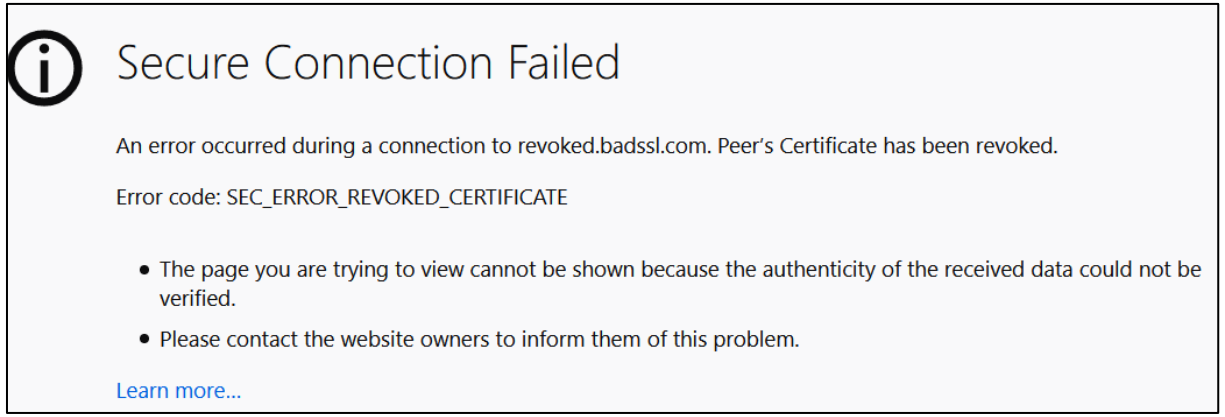


Revoking certificates with CRLs

- Certificate Revocation Lists (CRLs)
  - frequently updated by CAs
  - contains list of all certificates which have been revoked
    - e.g., because of compromised keys
  - downloaded by browsers in regular intervals
- Several issues
  - interval of updates by CAs
  - interval of updates by browsers
  - blacklist does not ensure validity
- CRLs are (being) deprecated from browsers

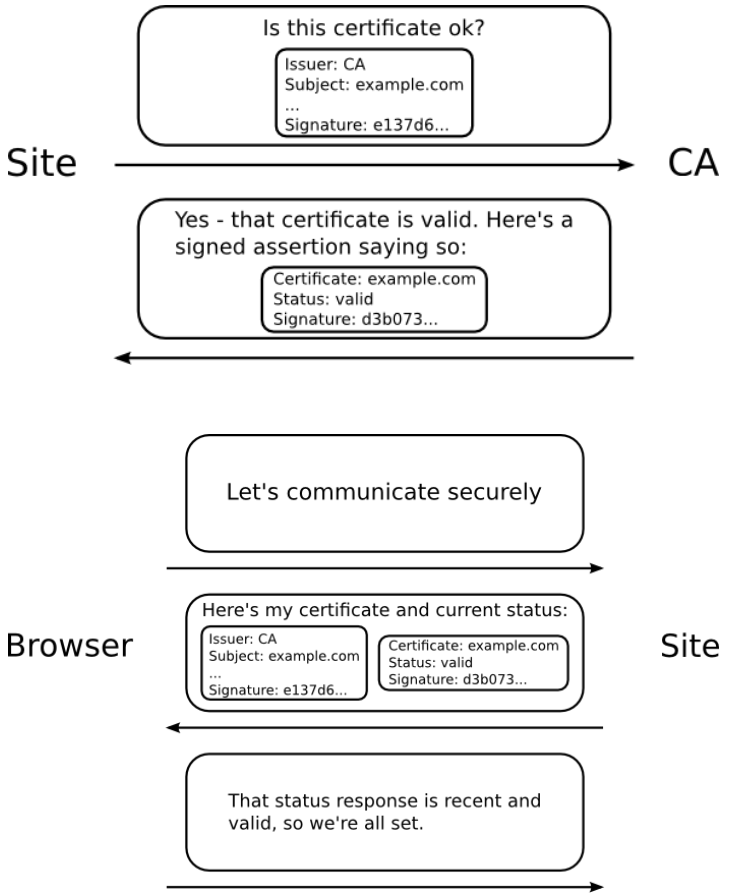
Online Certificate Status Protocol (OCSP)

- Turn CRL approach around
  - client checks validity on request
  - response is signed to allow for verification
- Also has several drawbacks
  - potentially high load on OCSP servers
  - clients often implement "tryLater" incorrectly
    - connect anyways, check again later
  - privacy leak
    - OCSP server knows all HTTPS servers you visited
  - Not enabled by default in all browsers
    - Chrome team claims "performance issues"



OCSP Stapling

- TLS-enabled servers regularly query OCSP server
  - get "ticket" with limited validity
  - OCSP response is "stapled" to TLS handshake
- Yields several benefits
  - less load on OCSP provider
  - no privacy leaks
- ... and a minor drawback
  - slight delay in invalidated certificates
- Certificates can be extended with Must-Staple
  - added to the original certificate



https://blog.mozilla.org/security/2013/07/29/ocsp-stapling-in-firefox/

PKI relies on trusted parties

- PKI is based on chain of trust
- Root CAs have ultimate trust
  - by design no restriction for which domains certificates may be issued
- Several attack scenarios
  - compromised root CA may issue any certificate
  - root CA may create intermediate certificate authorities

- Let's look at some horror stories…

Comodo

COMODO
Creating Trust Online®

- Comodo is one of the trusted root CAs
  - Its certificates for any website in the world are accepted by every browser
- Comodo accepts certificate orders submitted through resellers
  - Reseller uses a program to authenticate to Comodo and submit an order with a domain name and public key, Comodo automatically issues a certificate for this site

Comodo Break-In

COMODO
Creating Trust Online®

- An Iranian hacker broke into instantSSL.it and globalTrust.it resellers, decompiled their certificate issuance program, learned the credentials of their reseller account and how to use Comodo API
  - username: gtadmin, password: globaltrust
- Wrote his own program for submitting orders and obtaining Comodo certificates
- On March 15, 2011, got Comodo to issue 9 rogue certificates for popular sites
  - Including: mail.google.com, login.live.com, login.yahoo.com, login.skype.com, addons.mozilla.org

Consequences

- Attacker needs to first divert users to an attacker-controlled site instead of Google, Yahoo, Skype, but then…
  - For example, use DNS to poison the mapping of mail.yahoo.com to an IP address
- … “authenticate” as the real site
- … decrypt all data sent by users
  - Email, phone conversations, Web browsing

Q: Does HTTPS help? How about EV certificates?

Message from the Attacker

http://pastebin.com/74KXCaEZ

I'm single hacker with experience of 1000 hacker, I'm single programmer with experience of 1000 programmer, I'm single planner/project manager with experience of 1000 project managers …

When USA and Isarel could read my emails in Yahoo, Hotmail, Skype, Gmail, etc. without any simple little problem, when they can spy using Echelon, I can do anything I can. It's a simple rule. You do, I do, that's all. You stop, I stop. It's rule #1 …

Rule#2: So why all the world got worried, internet shocked and all writers write about it, but nobody writes about Stuxnet anymore?... So nobody should write about SSL certificates.

Rule#3: I won't let anyone inside Iran, harm people of Iran, harm my country's Nuclear Scientists, harm my Leader (which nobody can), harm my President, as I live, you won't be able to do so. as I live, you don't have privacy in internet, you don't have security in digital world, just wait and see...

DigiNotar Break-In

- In June 2011, the same “ComodoHacker” broke into a Dutch certificate authority, DigiNotar
  - Message found in scripts used to generate fake certificates:

    “THERE IS NO ANY HARDWARE OR SOFTWARE IN THIS WORLD EXISTS WHICH COULD STOP MY HEAVY ATTACKS MY BRAIN OR MY SKILLS OR MY WILL OR MY EXPERTISE"
- Security of DigiNotar servers
  - All core certificate servers in a single Windows domain, controlled by a single admin password (Pr0d@dm1n)
  - Software on public-facing servers out of date, unpatched
  - Tools used in the attack would have been easily detected by an antivirus… if it had been present

Consequences of DigiNotar Hack

- Break-in not detected for a month
- Rogue certificates issued for *.google.com, Skype, Facebook, www.cia.gov, and 527 other domains
- 99% of revocation lookups for these certificates originated from Iran
  - Evidence that rogue certificates were being used, most likely by Iranian government or Iranian ISPs to intercept encrypted communications
    - Textbook man-in-the-middle attack
  - 300,000 users were served rogue certificates
- DigiNotar filed for bankruptcy

TrustWave

- In Feb 2012, admitted issuance of an intermediate CA certificate to a corporate customer
  - Purpose: “re-sign” certificates for “data loss prevention”
  - Translation: forge certificates of third-party sites in order to spy on employees’ encrypted communications with the outside world
- Customer can now forge certificates for any site in world… and they will be accepted by any browser!
  - What if a “re-signed” certificate leaks out?
- Do other CAs do this?

TurkTrust

- In Jan 2013, a rogue *.google.com certificate was issued by an intermediate CA that gained its authority from the Turkish root CA TurkTrust
  - TurkTrust accidentally issued intermediate CA certs to customers who requested regular certificates
  - Ankara transit authority used its certificate to issue a fake *.google.com certificate in order to filter SSL traffic from its network
- This rogue *.google.com certificate was trusted by every browser in the world

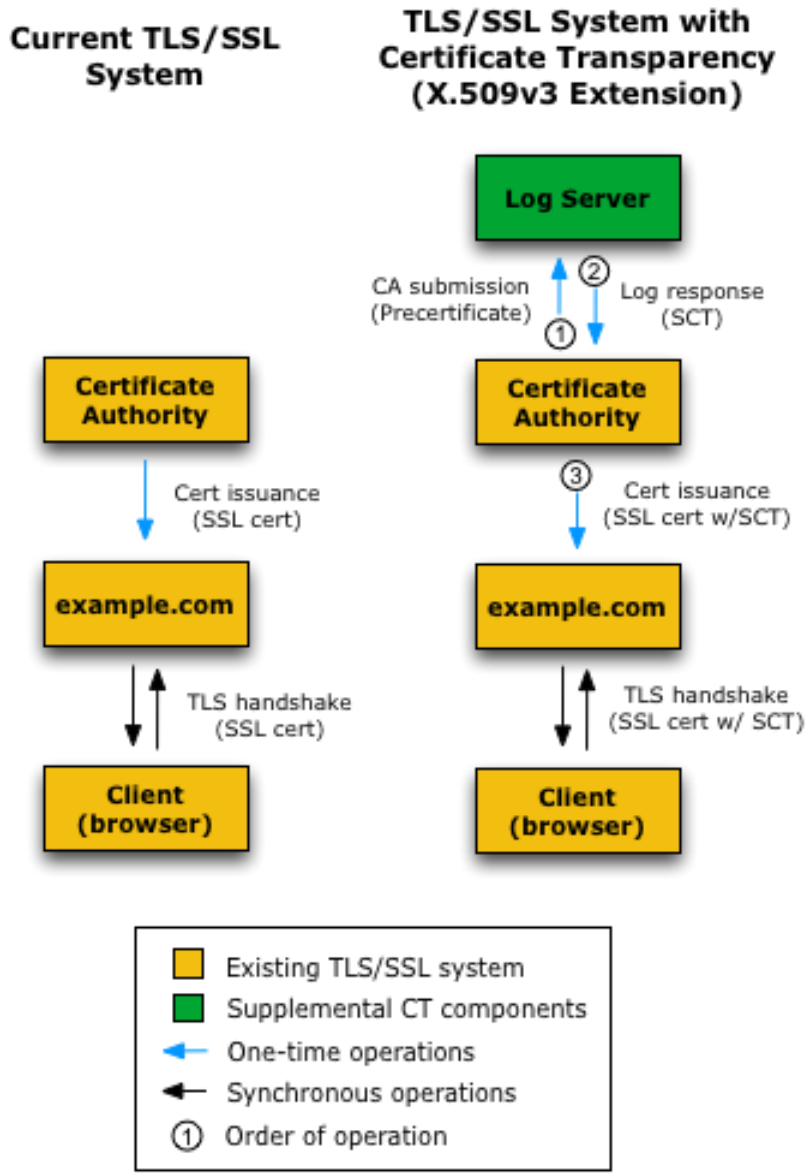
Symantec

- Owns VeriSign and Thawte root CAs
  - represents almost 30% of all valid certificates
- Several "mistakes"
  - improper checks before issuing EV certificates
  - misissued "test" certificates (including google.com, in 2015 and 2017)
  - allowed unauthorized employees access to CA keys
  - up to 30,000 certificates issued without proper validation
- Google Chrome's penalty
  - decrease validity to at most nine months (until Chrome 64)
  - disable all EV functionality for Symantec certificates

How do we deal with compromised / misbehaving CAs?

Certificate Transparency

- Goal: be able to trace back malicious certificates
  - e.g., in DigiNotar case
- Proposal: use third party for append-only log
  - after (pre-)certificate submission, log issues Signed Certificate Timestamp (SCT)
  - CA adds SCT to certificate, signs it, hands out
- Chrome only allows Symantec certificate with EV if they are in CT logs
  - enforced since June 2016
- Since April 2018, all new certificates must have an SCT



https://www.certificate-transparency.org/how-ct-works

HTTP Public Key Pinning

- An obvious side-effect of having so many certificate authorities is that even if one goes bad, they can make certificates for any and all TLS-protected websites

- HTTP Public Key Pinning (HPKP) aims to restrict that
  - A web server can send an HPKP header telling the browser to remember the hash of a few certificates and only accept those for future requests
    - HPKP supports pinning both leaf certificates as well as intermediate/root certificates

Unpacking the HPKP header

```
Public-Key-Pins:
  pin-sha256="cUPcTAZWKaASuYWhhneDttWpY3oBAkE3h2+soZS7sWs=";
  pin-sha256="M8HztCzM3elUxkcjR2S5P4hhyBNf6lHkmjAHKhpGPWE=";
  max-age=5184000; includeSubDomains;
  report-uri="https://www.example.org/hpkp-report"
```

- **pin-sha256**: Base64 encoded version of the sha256 fingerprint of the accepted certificate
  - Can have more than one
  - Current spec requires two, one of which is an inactive, backup one
- **max-age**: Number of seconds that the browser should enforce this for
- **includeSubdomains**: Enforce this policy on all subdomains of the website
- **report-uri**: Report violations to this specific URL
  - Presumably useful for identifying MITM attempts

Downsides of HPKP

- Too difficult and too dangerous
  - If your certificate expires, or accidentally deleted, or is compromised and you get a new one without having sent the right fingerprint for that certificate ahead of time, you are effectively DoS-ing yourself
    - Users will not be able to connect to your website, their browser will not allow them to
  - This can also be abused by attackers who get access to your server (RansomPKP attack)
    1. Bad guys get access to server
    2. Push new valid certificate with new pinning
       - Anyone who can prove domain ownership can get a certificate for that domain
    3. Good guys get back control of their website
    4. Now they need the right certificate that the attackers created because their users cannot connect to them
    5. Attackers can sell them the certificate for the “right” price

Downsides of HPKP

- Too
  - If
    - et a
    - ne,
    - PKP

HTTP Public Key Pinning (HPKP)

Deprecated: This feature is no longer recommended. Though some browsers might still support it, it may have already been removed from the relevant web standards, may be in the process of being dropped, or may only be kept for compatibility purposes. Avoid using it, and update existing code if possible; see the compatibility table at the bottom of this page to guide your decision. Be aware that this feature may cease to work at any time.

4. Now they need the right certificate that the attackers created because their users cannot connect to them
5. Attackers can sell them the certificate for the “right” price

DNS Certification Authority Authorization (CAA)

- Can we restrict the CAs who are allowed to issue certificates for our domains?

- DNS entry only meant for authorization of CAs
  - flag currently always set to 0, in the standard for future use
  - **<flag> issue "CA"**
    - allows CA to issue certificates for domain
    - can be set to issue ";" to ensure no certificates are issued
  - **<flag> issuewild "CA"**
    - allows CA to issue wild-card certificates for domain
  - **<flag> iodef "mailto:caa@domain.com"**
    - any policy-violating attempt of a certificate must be reported there
    - should also be notified about any certificates being issued

CAA Quiz

- Let's Encrypt can issue certificates for www.stonybrook.edu
- Let's Encrypt and DigiCert can issue certificates for cs.stonybrook.edu
- Nobody can issue wildcard certs for stonybrook.edu
- Nobody can issue certificates for nossl.stonybrook.edu
- Any violations for the latter should be reported to admins@stonybrook.edu

```
www.stonybrook.edu CAA 0 issue 'letsencrypt.org'
cs.stonybrook.edu CAA 0 issue 'digicert.com'
cs.stonybrook.edu CAA 0 issue 'letsencrypt.org'
stonybrook.edu CAA 0 issuewild ';'
nossl.stonybrook.edu CAA 0 issue ';'
nossl.stonybrook.edu CAA 0 iodef 'mailto:admins@stonybrook.edu'
```

Economics of SSL

- Up until a few years ago, SSL/TLS was something that only websites handling sensitive data would need to get
  - Banks
  - Social Networks
  - Email providers
  - Etc.
- Even though the price of certificates has gone down, today one would need to pay between $10 and $100 for a simple domain-verified (DV) certificate
  - Wildcard certificates (e.g. *.example.com) cost more
  - Symantec, GeoTrust, RapidSSL, Thawte are all common providers

Economics of SSL

- Process for a traditional DV certificate
  1. Go to a commercial certificate provider
  2. Pay the fee
  3. Prove that you own the domain
     - Receive email on an address listed in your WHOIS information
     - Place a file on your root directory with a specific file-name and file-content
     - Add a DNS record specified by the SSL provider
  4. Generate a Certificate Signing Request on your system (e.g. using openssl) and a private key and upload the CSR to the certificate provider
  5. Receive your certificate in an email
  6. Set up your webserver to use that certificate

Economics of SSL

- This was the only way of doing things up until 2016 when Let's Encrypt became publicly available

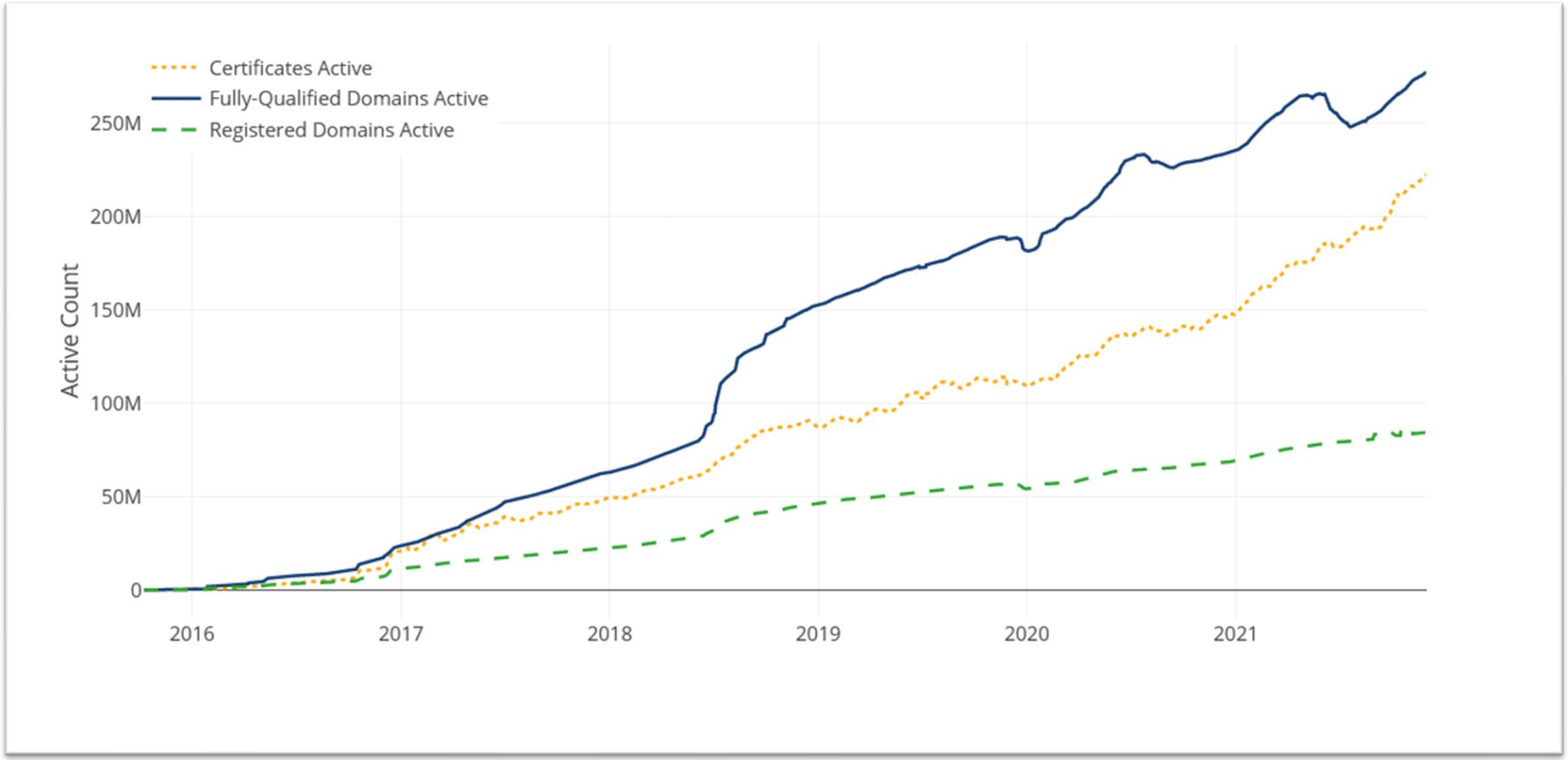
- Let's Encrypt is a certificate authority that provides free domain-verified certificates
  - They were not the first cost-free provider but they had the backing and PR of large companies
  - They provided software that makes the fetching and installation of a certificate, essentially automatic

Let's Encrypt

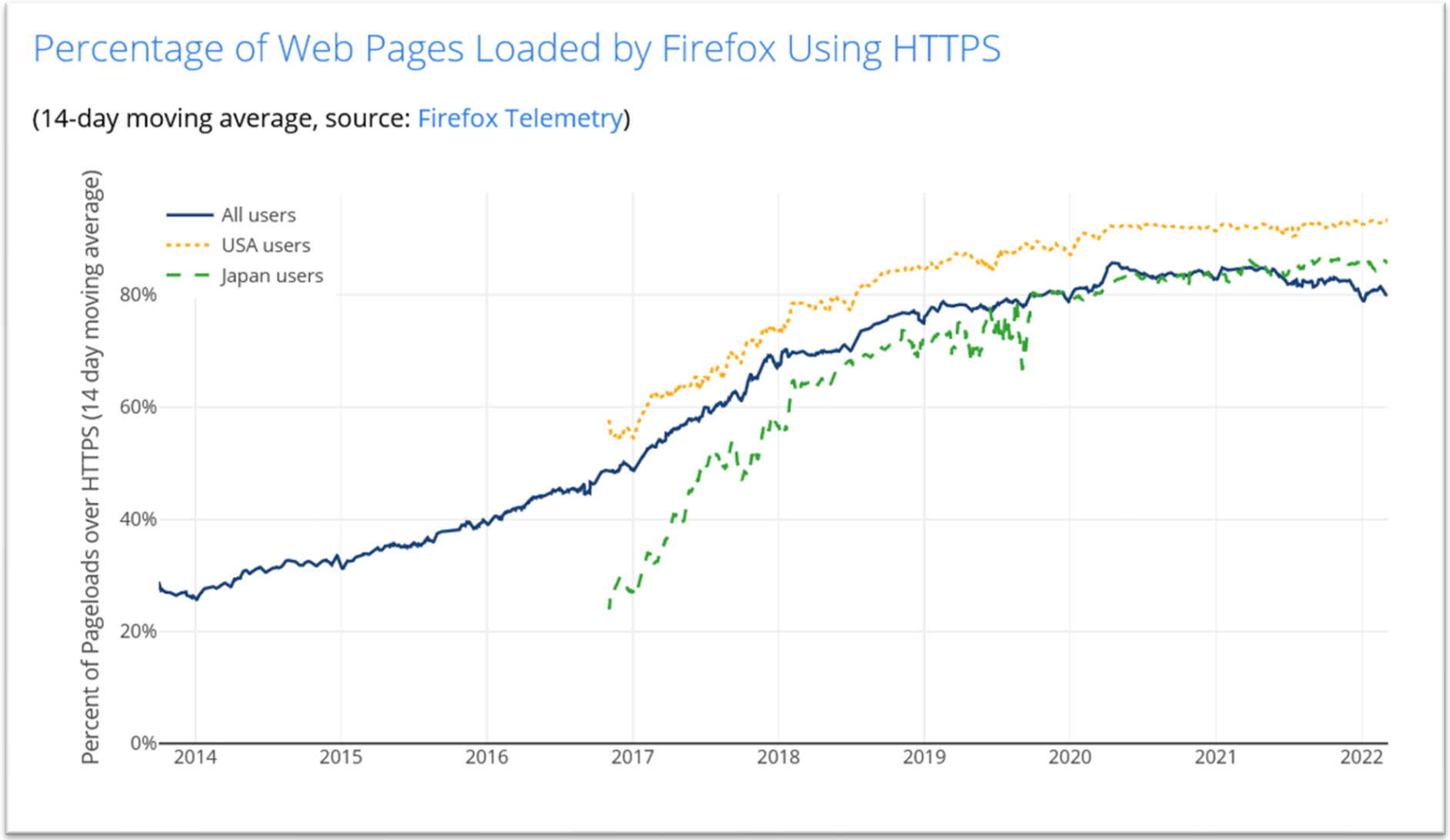
- Let's Encrypt uses a challenge-response protocol called ACME (Automated Certificate Management Environment) to verify that a client is indeed the owner of a domain name, before it issues a free certificate

- The most popular implementation of ACME is that of certbot
  - https://certbot.eff.org/

Let's Encrypt growth



SSL Telemetry



Everyone uses Let's Encrypt…including the bad guys

- Since everyone can now have an SSL certificate, so can the bad guys
  - https://www.totallypaypal.com
  - https://www.paypal.com.joesbakery.com
- Websites with SSL have been traditionally thought of as "secure" and "safe" so phishing attacks may work better on SSL websites

Response by Let's Encrypt

- Certificate Authorities are not supposed to be content watchdogs
  - A CA should issue a DV certificate to anyone who can prove ownership of a given domain
    - https://letsencrypt.org/2015/10/29/phishing-and-malware.html
- This will require re-educating users about the meaning of https
  - “Secure” and “safe” are not the same thing
  - HTTPS guarantees “secure” but does not say anything about “safe”

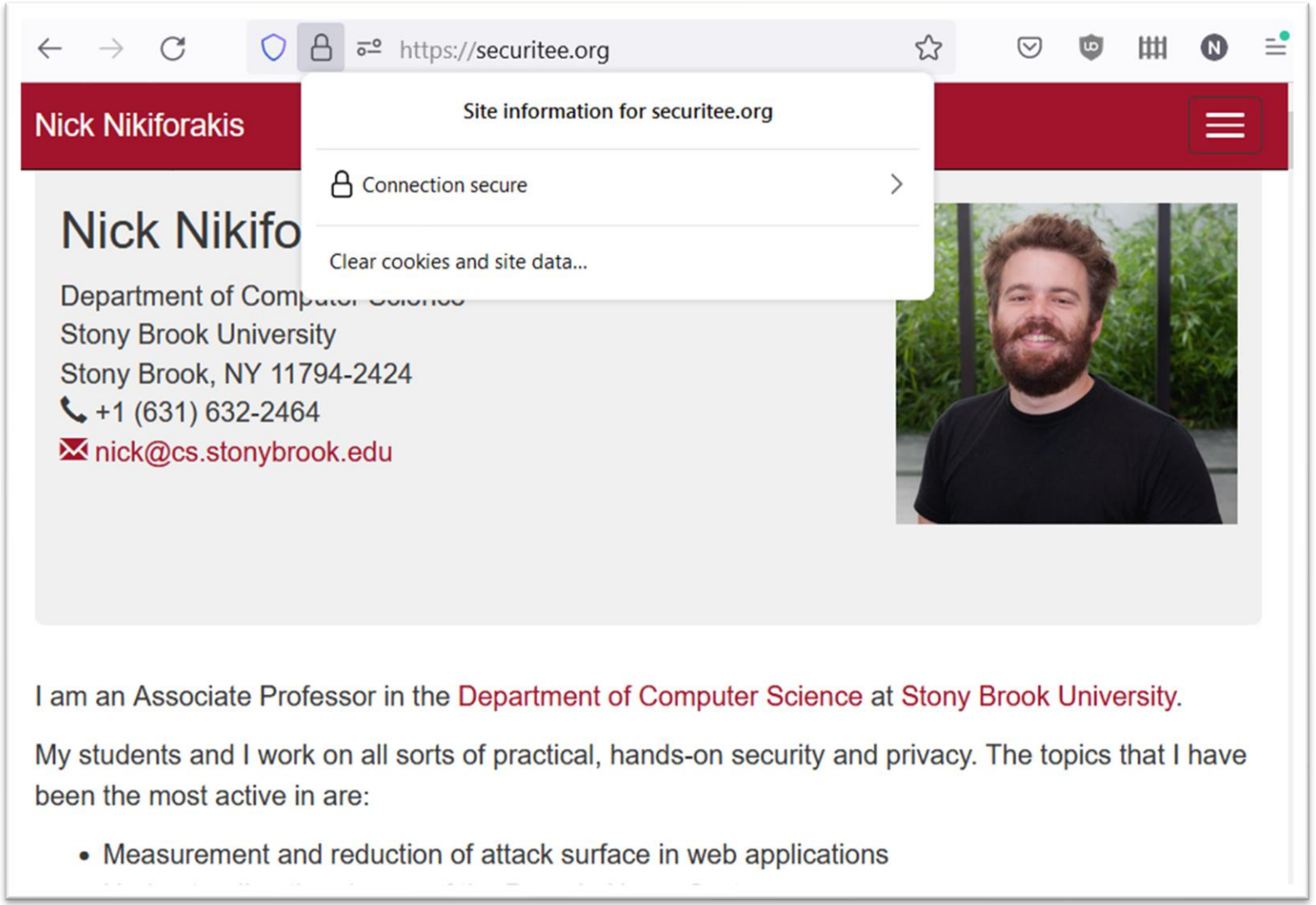
To summarize the differences between the two

Vision for the web and Let's Encrypt

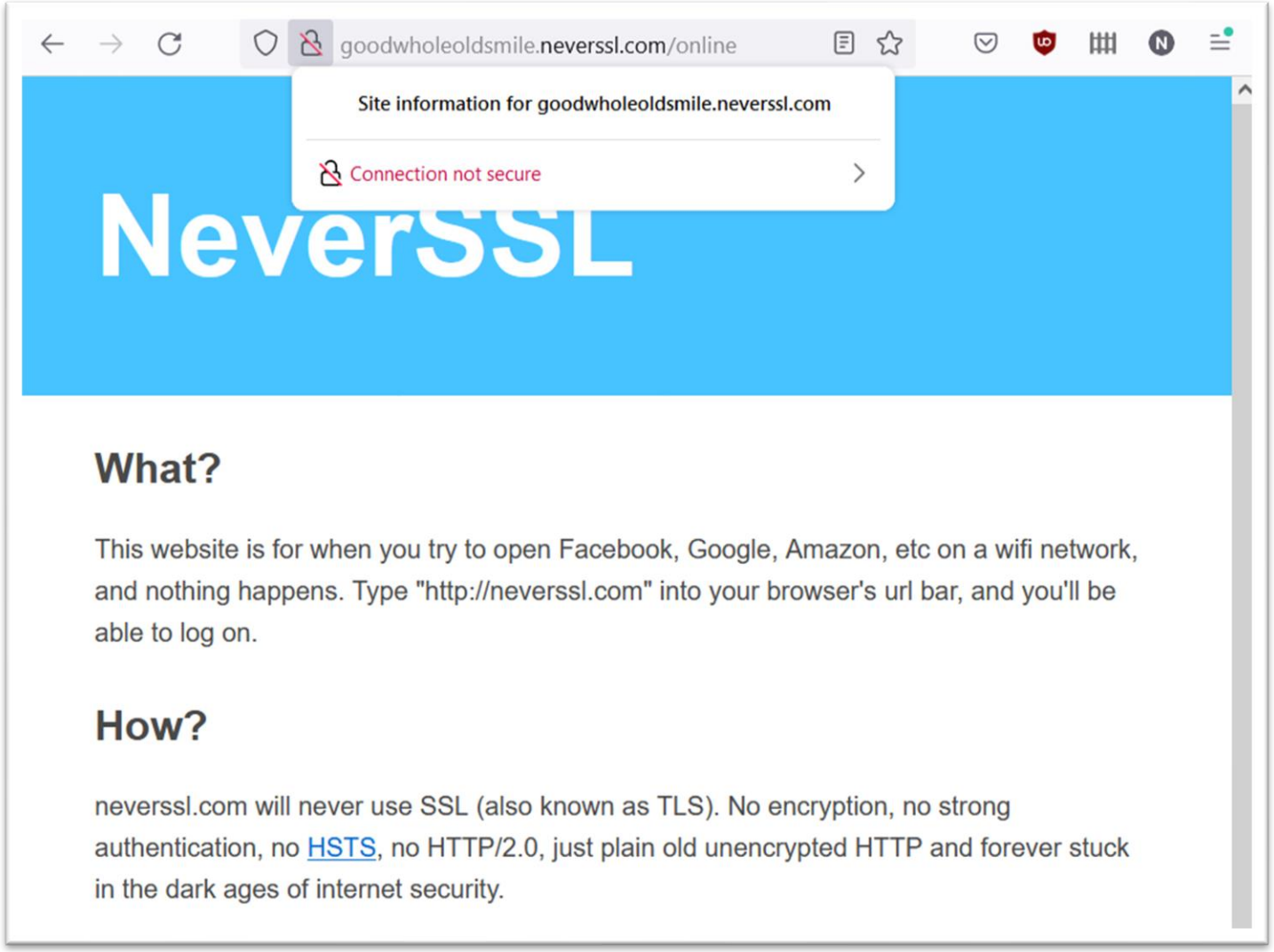
- By removing the financial cost and technical-know-how barriers, the backers of Let's Encrypt are pushing for an HTTPS-only web
  - This is the opposite of what we used to have a few years ago

- Google and Firefox are already penalizing non-HTTPS websites
  - "Default" is now HTTPS, not HTTP

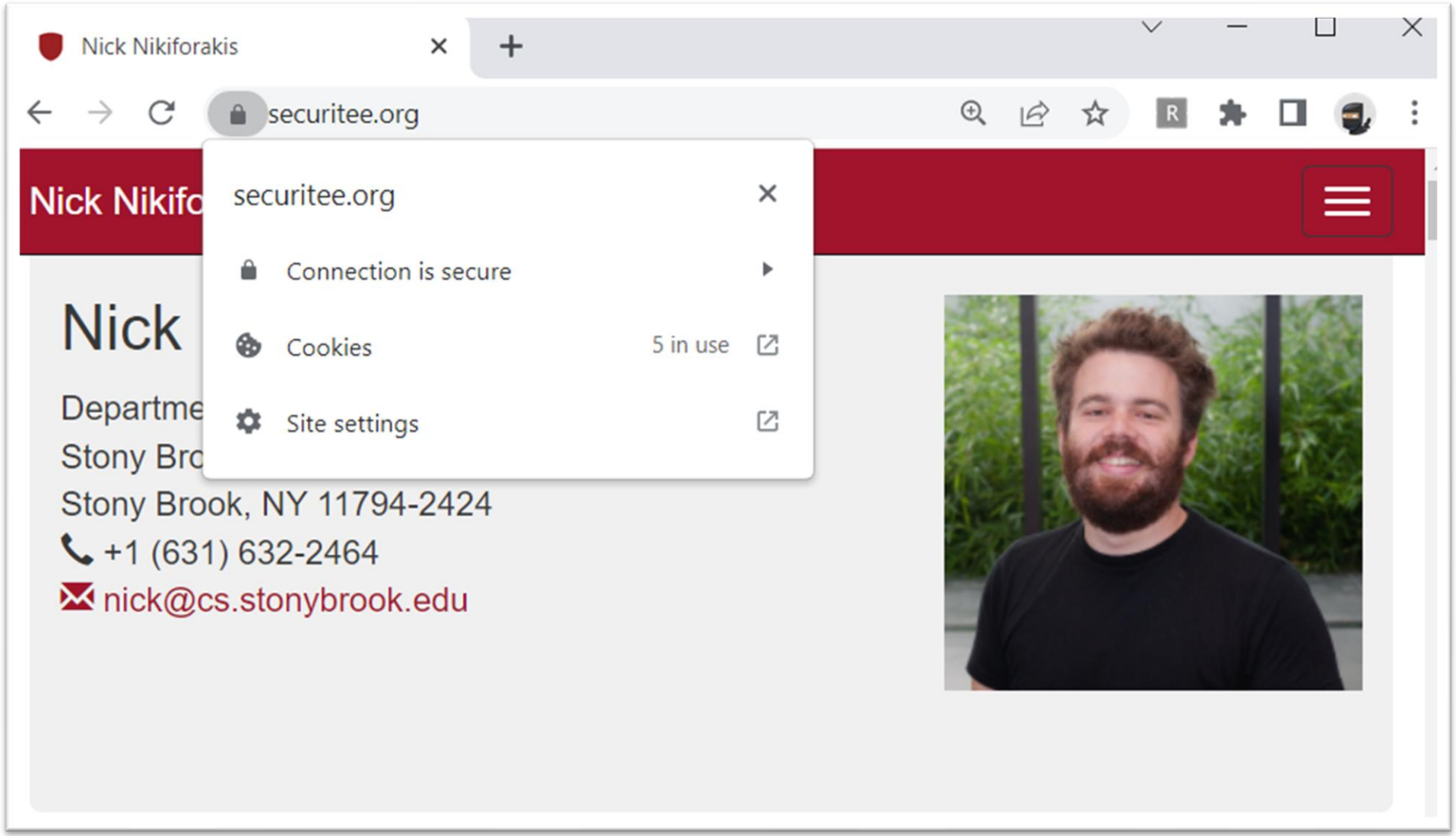
Firefox - HTTPS



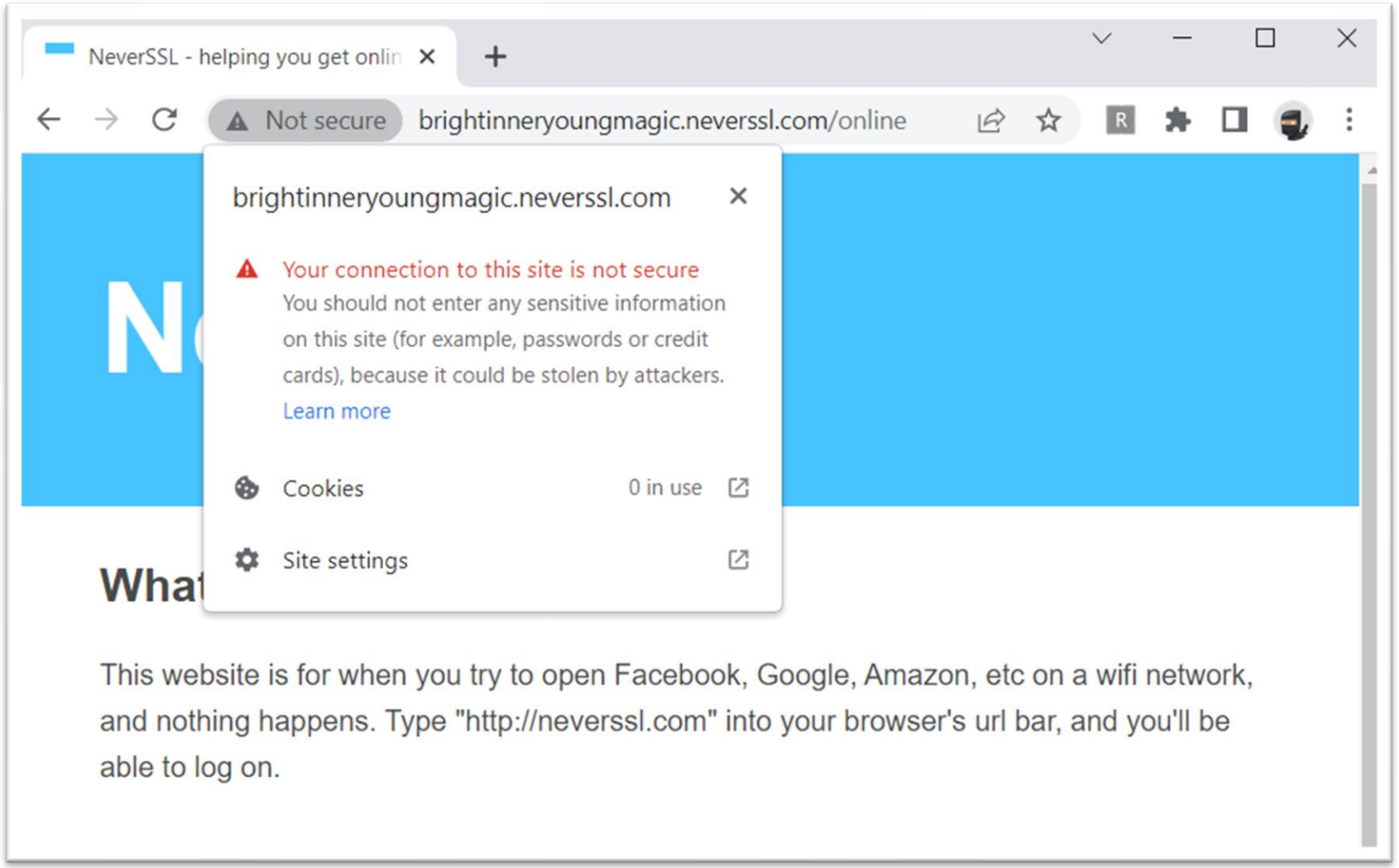
Firefox - HTTPS



Chrome - HTTPS



Chrome - HTTP


NeverSSL - helping you get onlin
Not secure
brightinneryoungmagic.neverssl.com/online
brightinneryoungmagic.neverssl.com
Your connection to this site is not secure
You should not enter any sensitive information on this site (for example, passwords or credit cards), because it could be stolen by attackers.
Learn more
Cookies
0 in use
Site settings
This website is for when you try to open Facebook, Google, Amazon, etc on a wifi network, and nothing happens. Type "http://neverssl.com" into your browser's url bar, and you'll be able to log on.

Not only UI penalties

- Google Chrome engineers are also pushing for making powerful features unavailable on websites that do not utilize HTTPS

- Current list of features
  - Geolocation
  - Device motion/orientation
  - Notifications
  - AppCache
  - getUserMedia

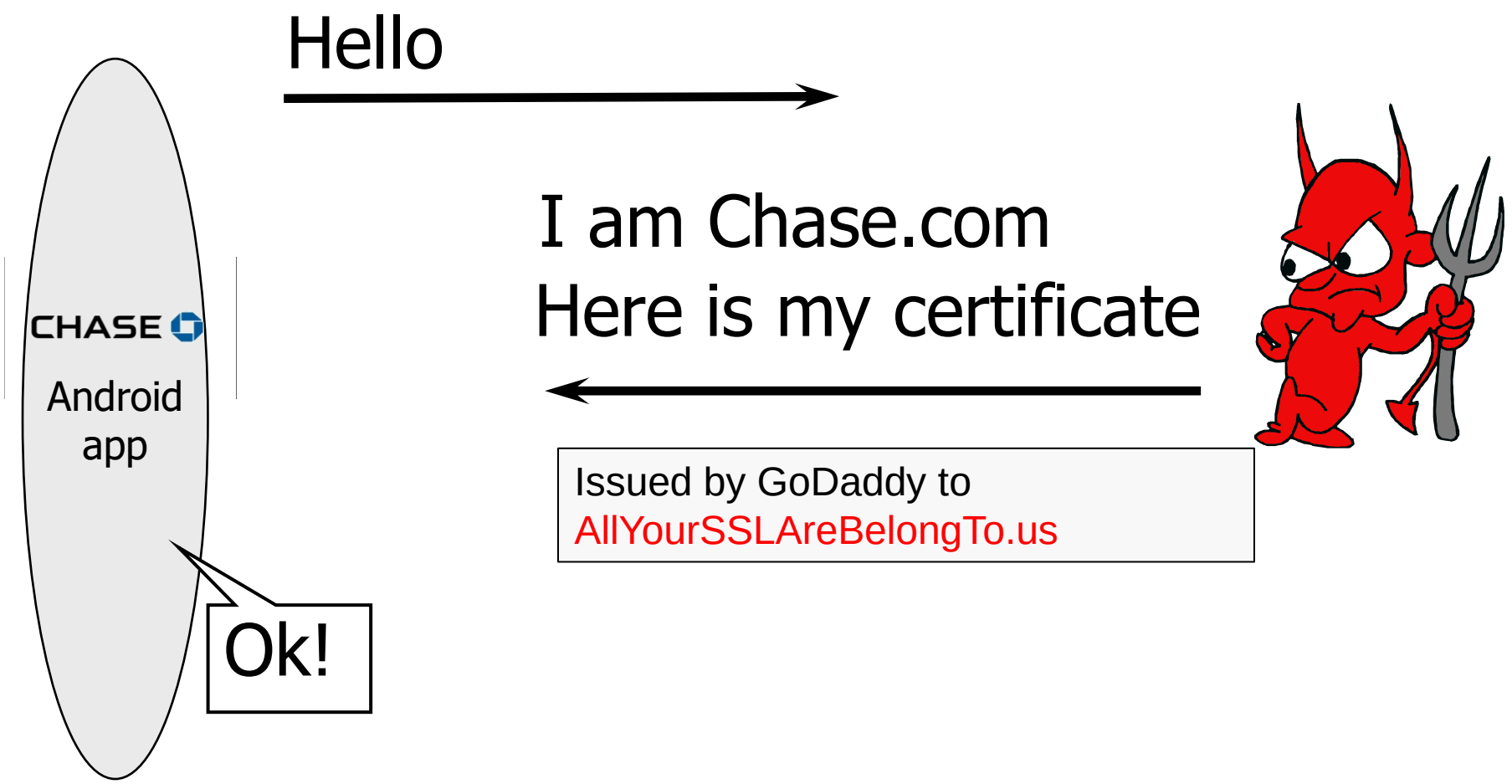
- Google search has also been using SSL support as a ranking feature
  - If your site does not support HTTPS, all other things being equal, it will be lower-ranked than those websites that do support HTTPS

What about TLS implementations?

OpenSSL Incident

- Code overhaul of OpenSSL in 2006
  - using automated tool Valgrind to look for errors
  - discovered reading of uninitialized data
- Code was rewritten to remove usage of uninitialized data
  - actually, data was meant to add randomness
  - instead, PRNG only seeded with PID - only 32k values
- Result: trivial for attacker to precompute all 32k values
  - allowed for decryption of every TLS connection
  - (same for a lot of SSH keys)

SSL/TLS Handshake



Failing to Check Hostname

"Researchers at the University of Texas at Austin and Stanford University have discovered that poorly designed APIs used in SSL implementations are to blame for vulnerabilities in many critical non-browser software packages. Serious security vulnerabilities were found in programs such as Amazon's EC2 Java library, Amazon's and PayPal's merchant SDKs, Trillian and AIM instant messaging software, popular integrated shopping cart software packages, Chase mobile banking software, and several Android applications and libraries. SSL connections from these programs and many others are vulnerable to a man in the middle attack…"

- Threatpost (Oct 2012)

Major payment processing gateways, client software for cloud computing, integrated e-commerce software, etc.

Goto Fail

Here is PayPal's certificate
And here is my signed Diffie-Hellman value

... verify the signature on the DH value using the public key from the certificate

```
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
   goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
   goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail; ...
err = sslRawVerify(...);
...
fail: ... return err ...
```

???

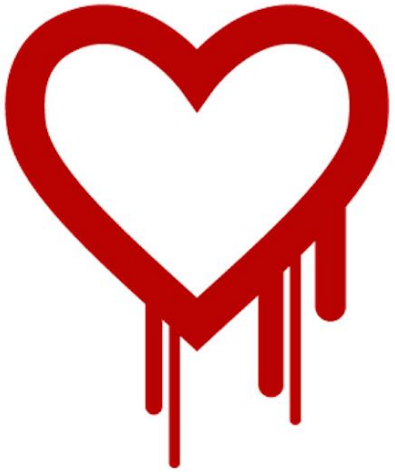
Signature is verified here

https://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/

Complete Fail Against MITM

- Discovered in February 2014
- All OS X and iOS software vulnerable to man-in-the-middle attacks
  - Broken TLS implementation provides no protection against the very attack it was supposed to prevent
- What does this tell you about quality control for security-critical software?

Heartbleed



- "Heartbeat" mechanism in OpenSSL to ensure that a server is still alive
  - Hey server. Are you there? If so, respond with "Parrot" (6 chars)
  - "Parrot"
- Heartbleed vulnerability (buffer overread)
  - Server trusts the client for the length of the string
  - Hey server. Are you there? If so, respond with "Parrot" (999 chars)
  - "Parrot e3 5c 29 2b a3 b7 35 93 db 29 2c 66 0d 48 11 64
    5f f2 9f 4e d7 ca c4 a6 e3 01 24 97 0a dd 75 15
    ec 4a 10 e9 b8 93 98 30 af ba 48 5d d5 57 4d a2
    53 28 a0 0a aa fa 5b c2 dd 56 38 15 74 52 6f 80
    a2 63 6d 22 cc 58 af 50 cf fb 51 9f 7b f0 c9 f4
    4b d5 a5 f1 90 15 09 b5 80 03 02 6b e4 c3 97 0c
    23 c8 fb 76"

Adjacent memory containing passwords, cookies, private keys, etc.

What can we do about these?

- As web application developers
  - Limited options that should still be exercised
    - Attack-surface reduction
    - Firewalls
    - Intrusion detection
    - Quick patching
- Developers of TLS implementations
  - Use fuzzing and static analysis to discover and correct bugs
  - Test suites
  - Switch to more secure programming languages

Summary

14

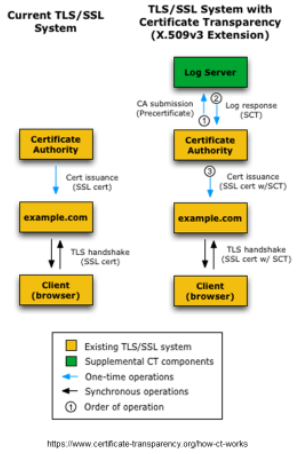
Security in HTTP

- There is no security in HTTP.
- Solution: encapsulate HTTP traffic in secure channel
  - used to be SSL (HTTP via SSL)
  - nowadays Transport Layer Security (TLS)
- TLS adds security to HTTP
  - end-to-end encryption
  - server authentication
  - optional client authentication (rarely used in practice)

39

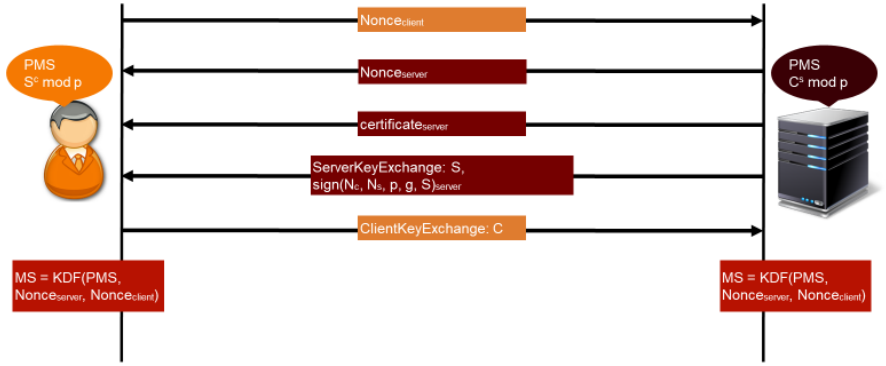
Certificate Transparency

- Goal: be able to trace back malicious certificates
  - e.g., in DigiNotar case
- Proposal: use third party for append-only log
  - after (pre-)certificate submission, log issues Signed Certificate Timestamp (SCT)
  - CA adds SCT to certificate, signs it, hands out
- Chrome only allows Symantec certificate with EV if they are in CT logs
  - enforced since June 2016
- Since April 2018, all new certificates must have an SCT



23

(Very simplified) connection establishing in TLS with DHE



2

Establishing trust in server's certificate

- Trusting a certificate boils down to trusting the CA
  - several root CAs integrated into browsers
  - CAs may be allowed to sign other CA's keys
- Example stonybrook.edu
  - certificate signed by InCommon RSA Server CA
  - ... which is signed by USERTrust RSA Certification Authority
  - ... which is in the browser

Credits

- Original slide deck by Ben Stock
- Some slides by Vitaly Shmatikov
- Modified and extended by Nick Nikiforakis