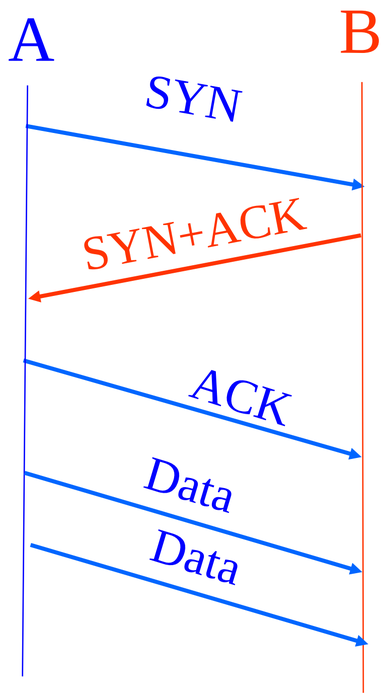
Network Security 4

Announcements

- Project 2 Design Doc due tonight
  - Project 2 due July 29
  - Start your implementation now!
  - Remember to write good tests
- Complete Mid-Summer Survey by Wednesday
  - In HW2 Question 2
  - HW2 due August 1

Reminder: Establishing a TCP Connection



Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on local clock)

Hmm, any way for the attacker to know *this?*

How Do We Fix This?

Use a (Pseudo)-*Random* ISN

Sure – make a non-spoofed connection *first*, and see what server used for ISN y then!

Summary of TCP Security Issues

- An attacker who can observe your TCP connection can manipulate it:
  - Forcefully terminate by forging a RST packet
  - Inject (spoof) data into either direction by forging data packets
  - Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  - Remains a major threat today

Summary of TCP Security Issues

- An attacker who can observe your TCP connection can manipulate it:
  - Forcefully terminate by forging a RST packet
  - Inject (spoof) data into either direction by forging data packets
  - Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  - Remains a major threat today
- If attacker could predict the ISN chosen by a server, could “blind spoof” a connection to the server
  - Makes it appear that host ABC has connected, and has sent data of the attacker’s choosing, when in fact it hasn’t
  - Undermines any security based on trusting ABC’s IP address
  - Allows attacker to “frame” ABC or otherwise avoid detection
  - Fixed (mostly) today by choosing random ISNs

The SYN Flood DOS Attack...

- When a computer receives a TCP connection it decides to accept
  - It is going to allocate a significant amount of state
- So just send lots of SYNs to a server...
  - Each SYN that gets a SYN/ACK would allocate some state
  - So do a ***lot of them***
  - And ***spoof*** the source IP
- Attack is a resource consumption DOS
  - Goal is to cause the server to consume memory and CPU
- Requires that the attacker be able to spoof packets
  - Otherwise would just rate-limit the attacker's IPs

SYN-Flood Counter: SYN cookies

- Observation: Attacker needs to see or guess the server's response to complete the handshake
  - So don't allocate ***anything*** until you see the ACK... But how?
- Idea: Have our initial sequence ***not*** be random...
  - But instead have it be ***pseudo-***random, aka "random"
- So we create the SYN/ACK's ISN using the pseudo-random function
  - And then check than the ACK correctly used the sequence number

Easy SYN-cookies: HMAC

- On startup create a random key ***k***...
- For the server ISN:
  - HMAC(***k***,SIP|DIP|SPORT|DPORT|client_ISN)
- Upon receipt of the ACK
  - Verify that ACK is based off HMAC(***k***,SIP|DIP|SPORT|DPORT|client_ISN)
    - Remember that ACK sequence # == client_ISN + 1
- Only ***then*** does the server allocate memory for the TCP connection
  - HMAC is very useful for these sorts of constructions:
    Give a token to a client, verify that the client presents the token later

Theme of The Rest Of This Lecture...

How Can We Communicate With Someone New?

- Public-key crypto gives us amazing capabilities to achieve confidentiality, integrity & authentication without shared secrets …
- But how do we solve MITM attacks?
- How can we trust we have the true public key for someone we want to communicate with?

- Ideas?

Trusted Authorities

- Suppose there's a party that everyone agrees to trust to confirm each individual's public key
  - Say the Governor of California

- Issues with this approach?
  - How can everyone agree to trust them?
  - Scaling: huge amount of work; single point of failure …
    - … and thus Denial-of-Service concerns
  - How do you know you're talking to the right authority??

Trust Anchors

- Suppose the trusted party distributes their key so everyone has it ...

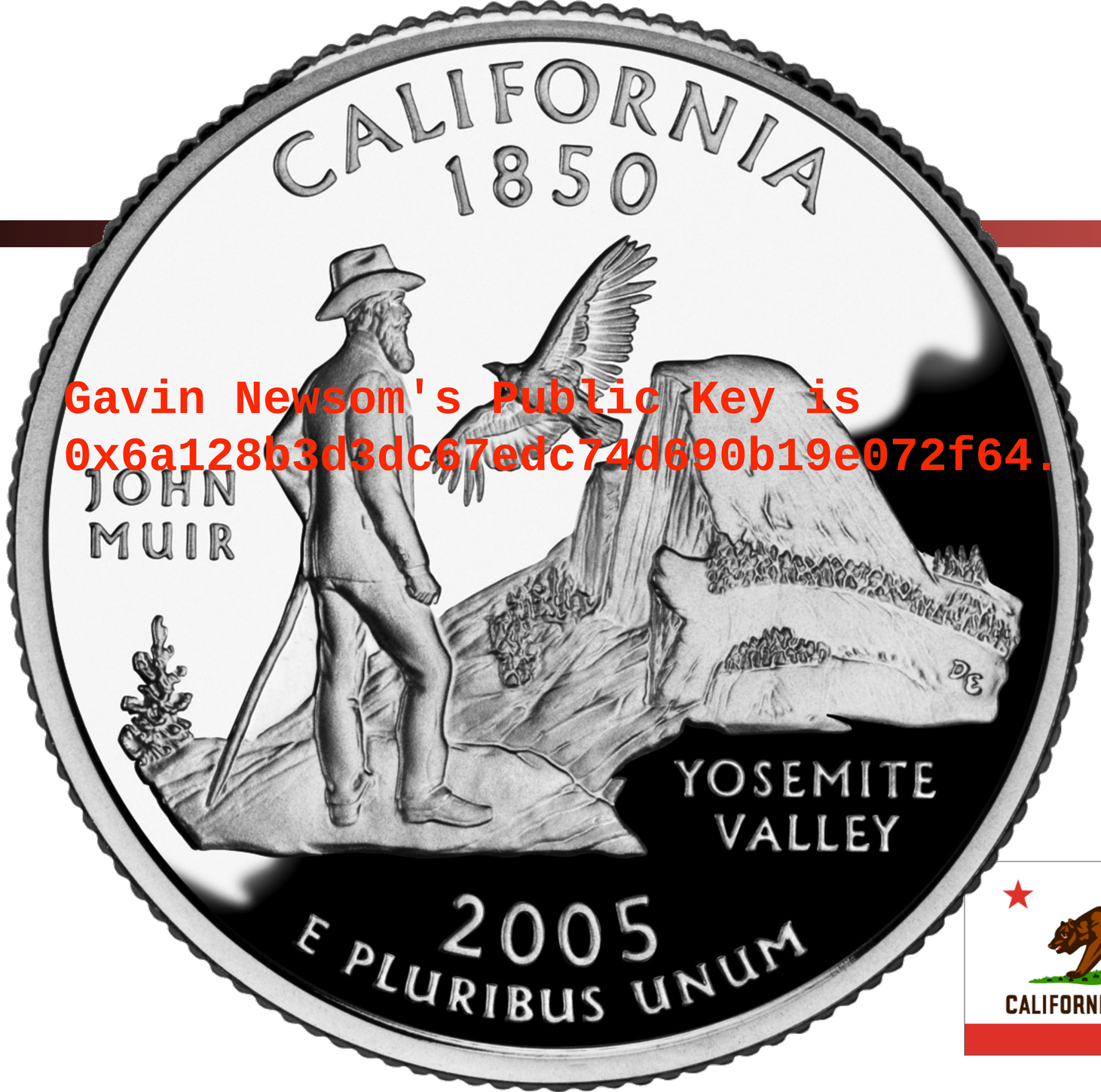

UNITED STATES OF AMERICA
IN GOD WE TRUST
LIBERTY
D
QUARTER DOLLAR


CALIFORNIA
1850
JOHN MUIR
YOSEMITE VALLEY
2005
E PLURIBUS

CALIFORNIA REPUBLIC


CALIFORNIA
1850
JOHN
MUIR
YOSEMITE
VALLEY
2005
E PLURIBUS UNUM


CALIFORNIA REPUBLIC



Gavin Newsom's Public Key is 0x6a128b3d3dc67edc74d690b19e072f64.

Trust Anchors

- Suppose the trusted party distributes their key so everyone has it ...
- We can then use this to bootstrap trust
  - As long as we have confidence in the decisions that that party makes

Digital Certificates

- Certificate (“cert”) = signed claim about someone’s public key
  - More broadly: a signed *attestation* about some claim

- Notation:
  $\{ M \}_K$ = “message M encrypted with public key k”
  $\{ M \}_{K^{-1}}$ = “message M signed w/ private key for K”

E.g. M = “Rafael's public key is K_{Rafael} = ***0x...6E5D2030***”
Cert: M,
$\{$ “Rafael's public key … ***0x...6E5D2030***” $\}_{K^{-1}Gavin}$
= ***0x923AB95E12...9772F***

Certificate

Gavin Newsom hearby asserts:
Rafael's public key is K_{Rafael} = ***0x...6E5D2030***

The signature for this statement using
$\mathbf{K^{-1}_{Gavin}}$ is ***0x923AB95E12...9772F***

Certificate

Gavin Newsom hearby asserts:
Rafael's public key is K_{Rafael} = ***0x...6E5D2030***

The signature for this statement using
K^{-1}_{Gavin} is ***0x923AB95E12...9772F***

This

Certificate

Gavin Newsom hearby asserts:
Rafael's public key is K_{Rafael} = ***0x...6E5D2030***

The signatu is computed over all of this using
$\mathbf{K^{-1}_{Gavin}}$ is ***0x923AB95E12...9772F***

Certificate

Gavin Newsom hearby asserts:
Rafael's public key is K_{Rafael} = ***0x...6E5D2030***

The signature for this statement using
$\mathbf{K^{-1}_{Gavin}}$ is ***0x923AB95E12...9772F***

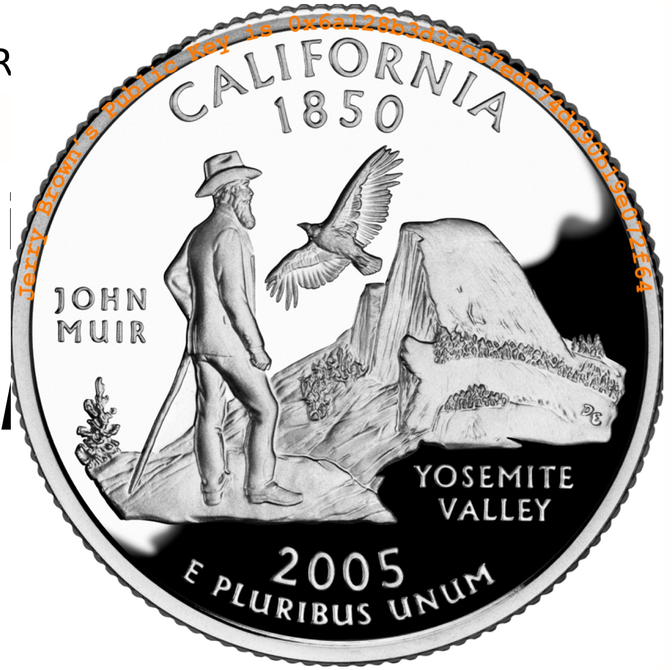
and can be *validated* using:

Certificate

This:

Gavin Newsom hearby asserts:
Rafael's public key is K_R

The signature for th
using
K^{-1}_{Gavin} is ***0x923AB95***



If We Find This Cert Shoved Under Our Door …

- What can we figure out?
  - If we know Gavin’s key, then whether he indeed signed the statement
  - If we trust Gavin’s decisions, then we have confidence we really have Rafael's key

- Trust = ?
  - Gavin won’t willy-nilly sign such statements
  - Gavin won’t let his private key be stolen

Analyzing Certs Shoved Under Doors ...

- ***How*** we get the cert doesn't affect its utility
- ***Who*** gives us the cert doesn't matter
  - They're not any more or less trustworthy because they did
  - Possessing a cert doesn't establish any identity!
- However: if someone demonstrates they can decrypt data encrypted with K_{Rafael}, then we have high confidence they possess K^{-1}_{Rafael}

Same for if they show they can sign using K^{-1}_{Rafael}

Scaling Digital Certificates

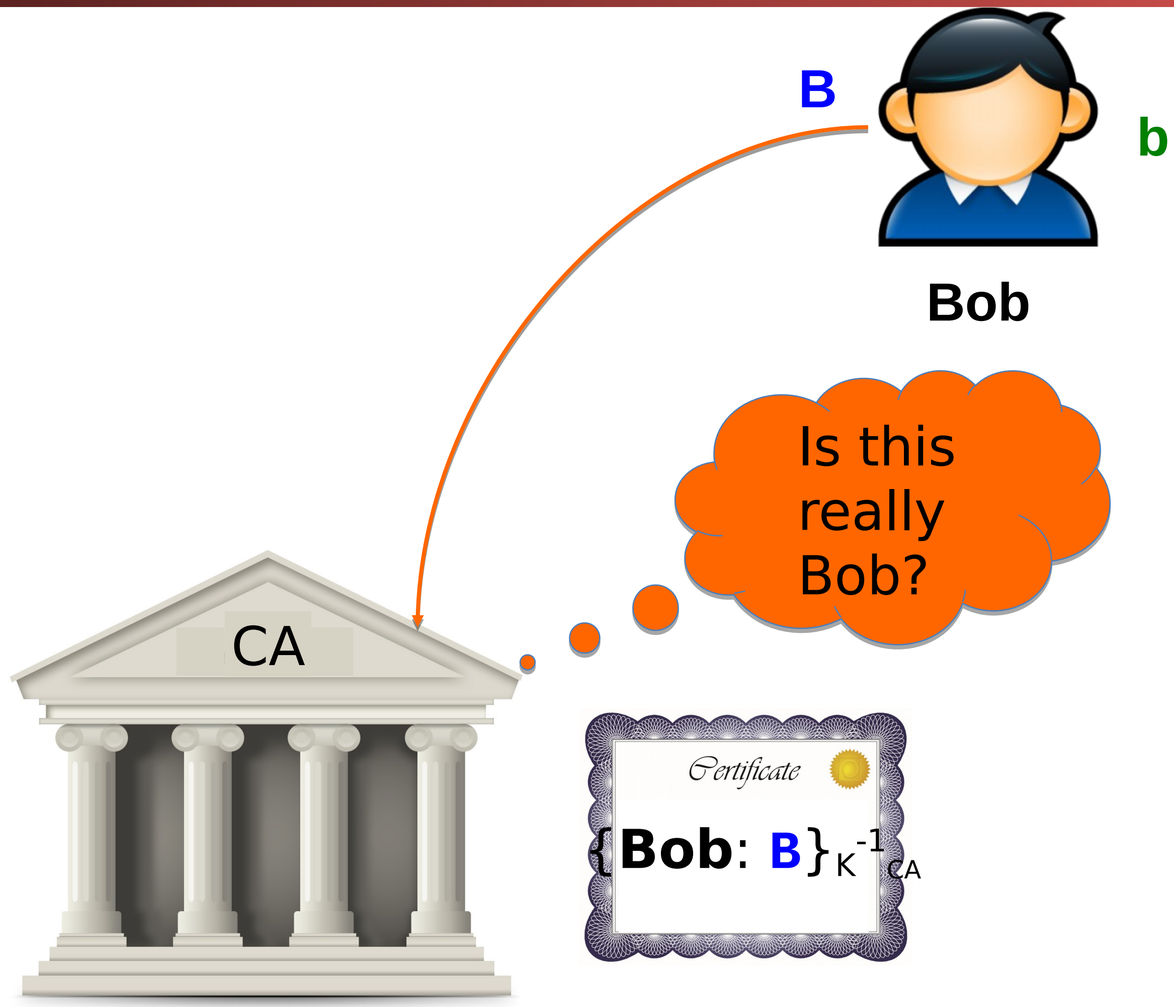
- How can this possibly scale? Surely Gavin can't sign everyone's public key!
- Approach #1: Introduce hierarchy via delegation
  - { "Janet Napolitano's public key is 0x... and I trust her to vouch for UC" }K^{-1}_{Gavin}
  - { "Carol Christ's public key is 0x... and I trust her to vouch for UCB" }K^{-1}_{Janet}
  - { "James Demmel's public key is 0x... and I trust him to vouch for EECS" }K^{-1}_{Carol}
  - { "Rafael Dutra's public key is 0x...6E5D2030" }K^{-1}_{Jim}

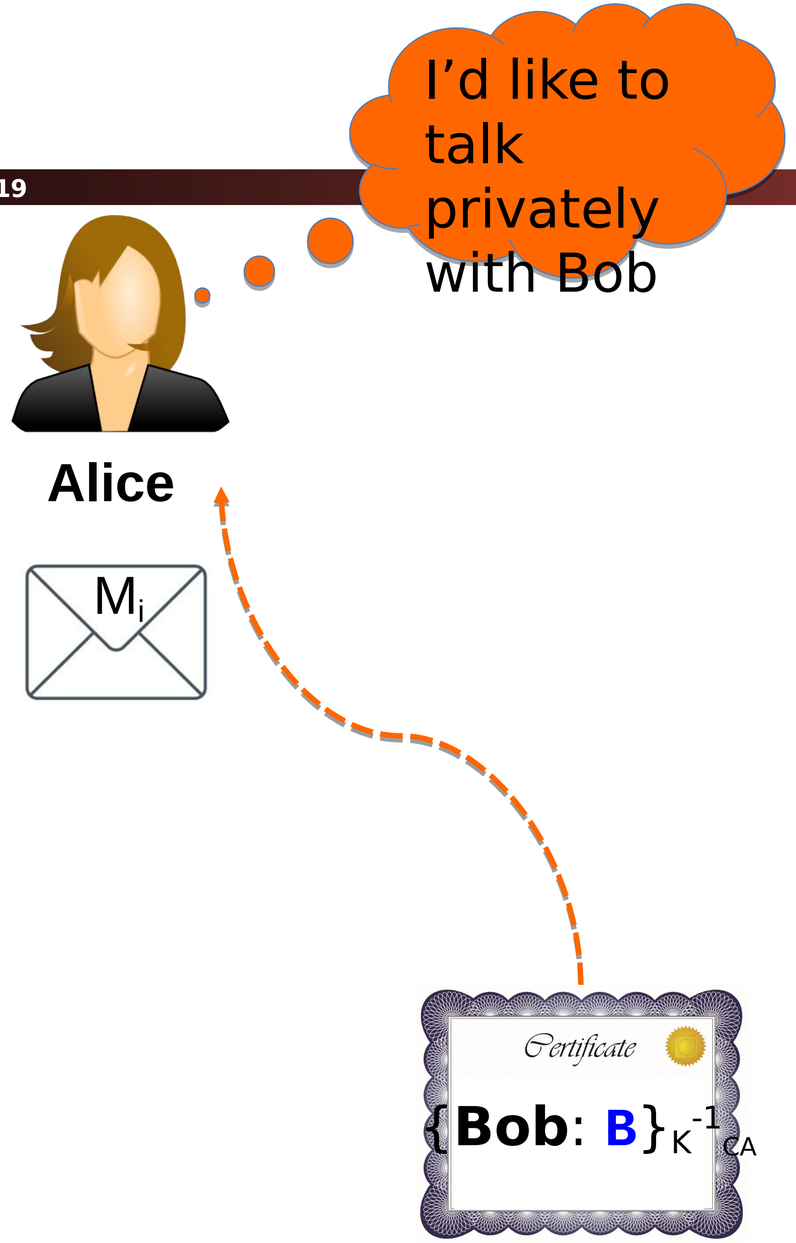
Scaling Digital Certificates, con’t

- Rafael puts this last on his web page
  - (or shoves it under your door)
- Anyone who can gather the intermediary keys can validate the chain
  - They can get these (other than Gavin’s) from anywhere because they can validate them, too
- Approach #2: have multiple trusted parties who are in the business of signing certs …
  - (The certs might also be hierarchical, per Approach #1)

Certificate Authorities

- CAs are trusted parties in a Public Key Infrastructure (PKI)
- They can operate offline
  - They sign (“cut”) certs when convenient, not on-the-fly (… though see below ...)
- Suppose Alice wants to communicate confidentially w/ Bob:
  - Bob gets a CA to issue {Bob’s public key is B} K^{-1}_{CA}
  - Alice gets Bob’s cert any old way
  - Alice uses her known value of K_{CA} to verify cert’s signature
  - Alice extracts B, sends $\{M\}K_B$


B
b
Bob
Is this really Bob?
CA
Certificate
{Bob: B}K-1CA


I’d like to talk privately with Bob
Alice
M_i
Certificate
{Bob: B}$_{K_{CA}^{-1}}$
B
b
Bob

Does CA's signature on B validate?

Alice

M_i

B

$C_i = E(M_i, \mathbf{B})$

b

Bob

M_i

b^*

B^*

Mallory

Bob

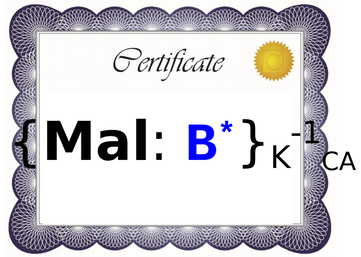
b^*

B^*

Mallory

Bob

Is this really Mal?



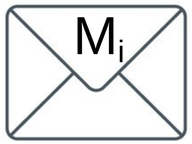
b^*

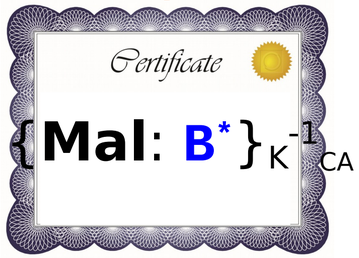
B^*

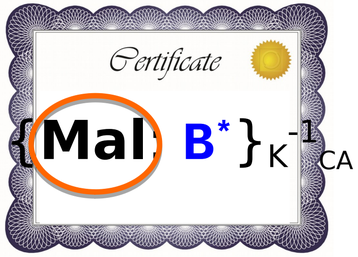
Mallory

Alice

Bob

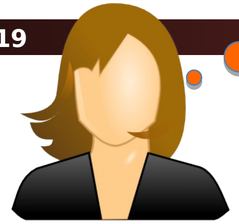






b*

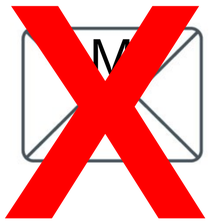
B*



Alice

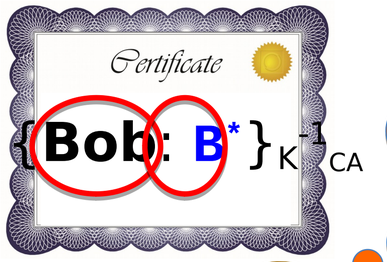
Mallory

Bob



Revocation

- What do we do if a CA screws up and issues a cert in Bob’s name to Mallory?



I'd like to talk privately with Bob

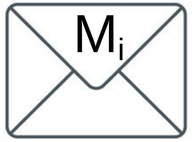
b*

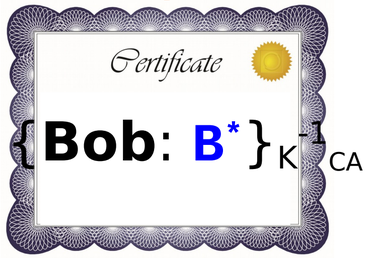
B*

Mallory

Alice

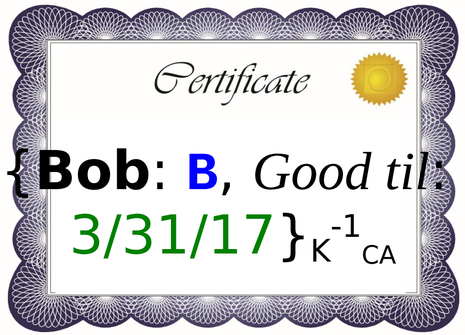
Bob





Revocation

- What do we do if a CA screws up and issues a cert in Bob's name to Mallory?
  - E.g. Verisign issued a ***Microsoft.com*** cert to a *Random Joe*
  - (Related problem: Bob realizes b has been stolen)
- *How do we recover from the error?*
- Approach #1: expiration dates
  - Mitigates possible damage
  - But adds management burden
    - Benign failures to renew will



Revocation, con’t

- Approach #2: announce revoked certs
  - Users periodically download cert revocation list (CRL)

b^*

B^*

Mallory

Alice

Bob

CRL = Certificate Revocation List

Alice

allory

b*

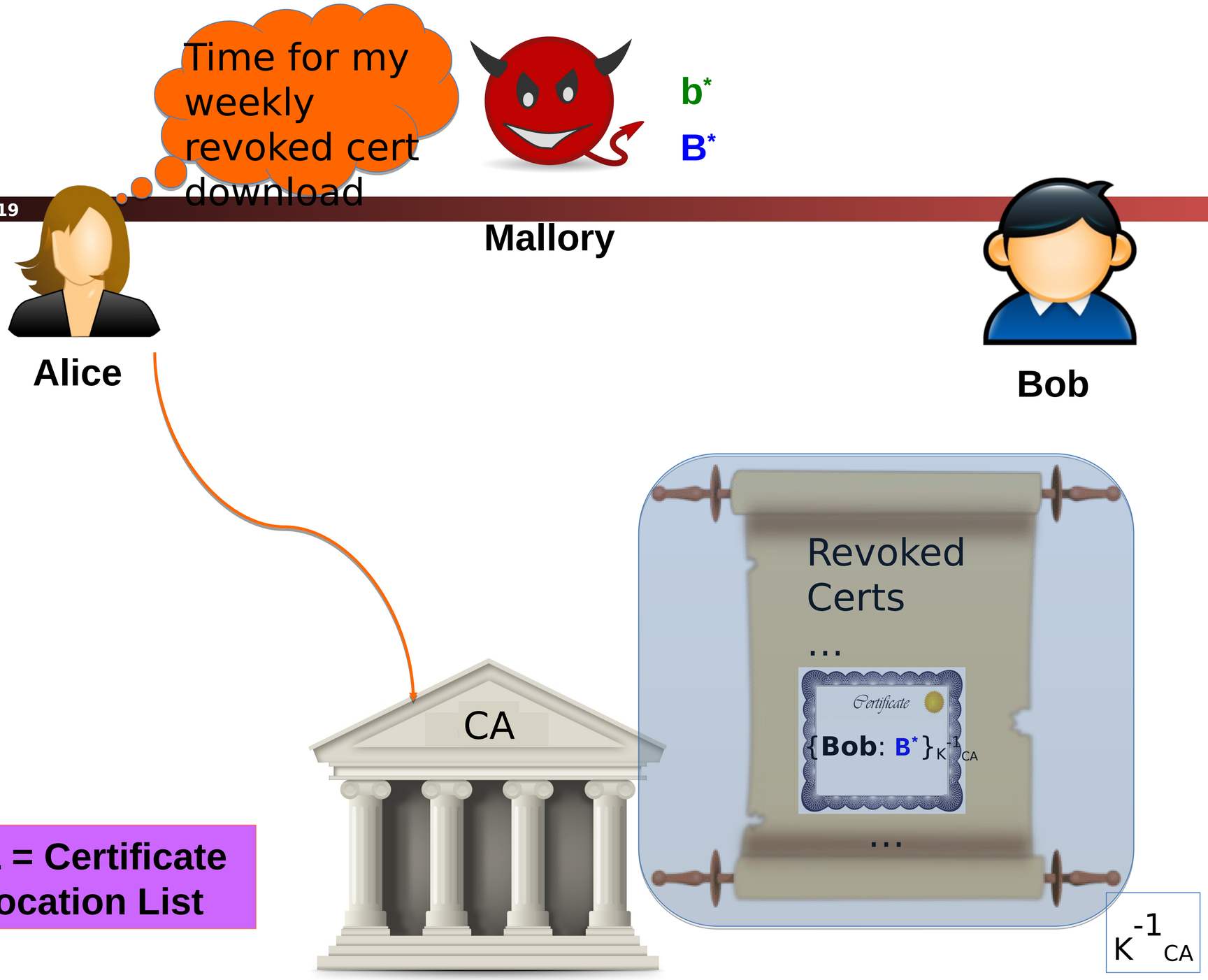
B*

Bob

CRL = Certificate Revocation List

Revocation, con’t

- Approach #2: announce revoked certs
  - Users periodically download cert revocation list (CRL)
- Issues?
  - Lists can get large


Time for my weekly revoked cert download
b*
B*
Mallory
Alice
Bob
CA
Revoked Certs
...
Certificate
{Bob: B*}K-1CA
...
CRL = Certificate Revocation List
K-1CA

Revocation, con’t

- Approach #2: announce revoked certs
  - Users periodically download cert revocation list (CRL)
- Issues?
  - Lists can get large
  - Need to authenticate the list itself – how? Sign it!
  - Mallory can exploit download lag
  - What does Alice do if can’t reach CA for download?
    - Assume all certs are invalid (fail-safe defaults)
      - Wow, what an unhappy failure mode!
    - Use old list: widens exploitation window

The (Failed) Alternative: The “Web Of Trust”

- Alice signs Bob’s Key
  - Bob Sign’s Carol’s
- So now if Dave has Alice’s key, Dave can believe Bob’s key and Carol’s key…
  - Eventually you get a graph/web of trust…
- PGP started out with this model
  - You would even have PGP key signing parties
  - But it proved to be a disaster:
    Trusting central authorities can make these problems so much simpler!

Random fact
2 min break

- Series: random facts about Berkeley professors and instructors
  - Because professors are inhuman...

Bran Castle (Dracula's

The Rest of Today's Lecture:

- Applying crypto technology in practice
- Two simple abstractions cover 80% of the use cases for crypto:
  - “Sealed blob”: Data that is encrypted and authenticated under a particular key
  - Secure channel: Communication channel that can’t be eavesdropped on or tampered with
- Today: TLS – a secure channel
  - In network parlance, this is an “application layer” protocol but…
  - designed to have any application over it, so really “layer 6.5” is a better description

Building Secure End-to-End Channels

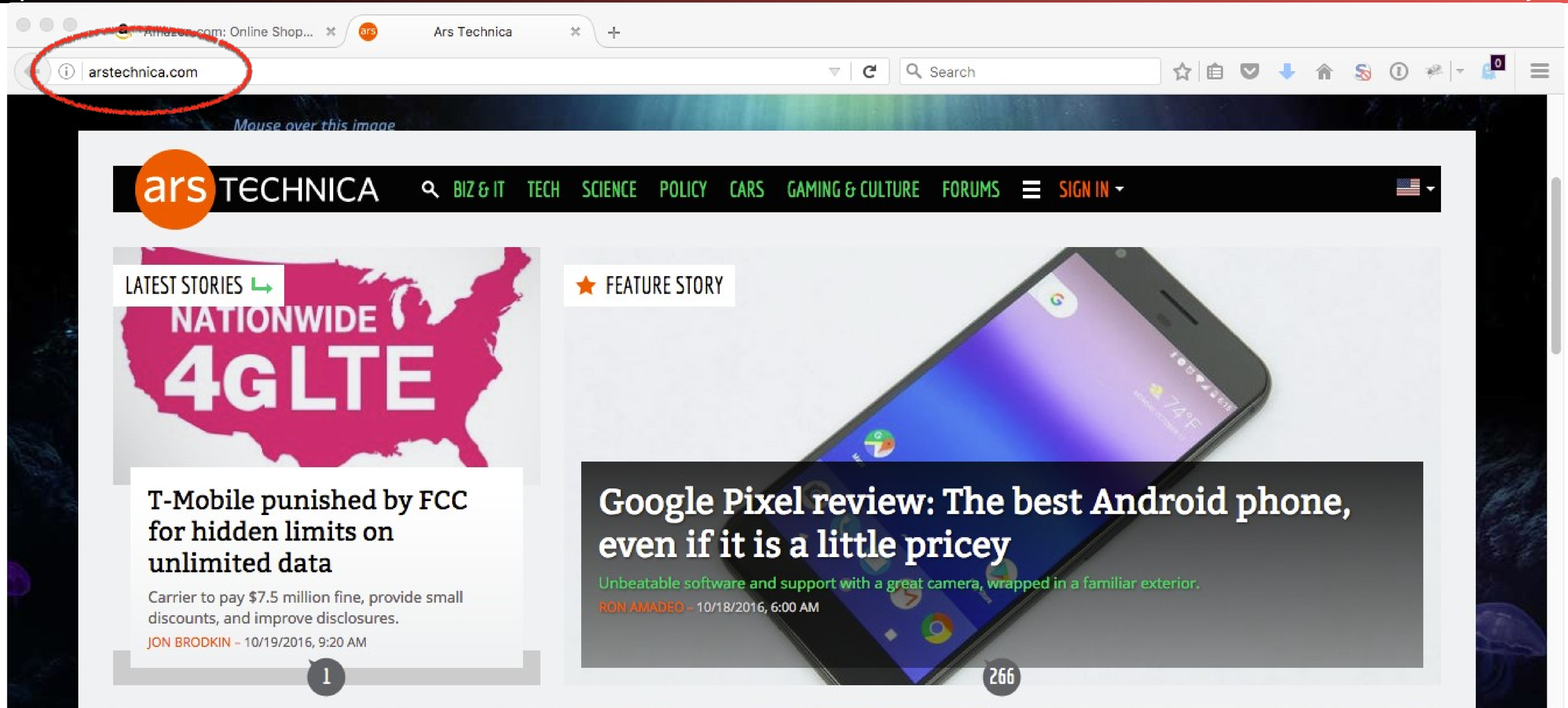
- End-to-end = communication protections achieved all the way from originating client to intended server
  - With no need to trust intermediaries
- Dealing with threats:
  - Eavesdropping?
    - Encryption (including session keys)
  - Manipulation (injection, MITM)?
    - Integrity (use of a MAC); replay protection
  - Impersonation?

(What's missing? *Availability …*)

Building A Secure End-to-End Channel: SSL/TLS

- SSL = Secure Sockets Layer (predecessor)
- TLS = Transport Layer Security (standard)
  - Both terms used interchangeably
- Security for any application that uses TCP
  - Secure = encryption/confidentiality + integrity + authentication (of server, but not of client)
- Multiple uses
  - Puts the ‘s’ in “https”
  - Secures mail sent between servers (STARTTLS)
  - Virtual Private Networks

An “Insecure” Web Page



A “Secure” Web Page

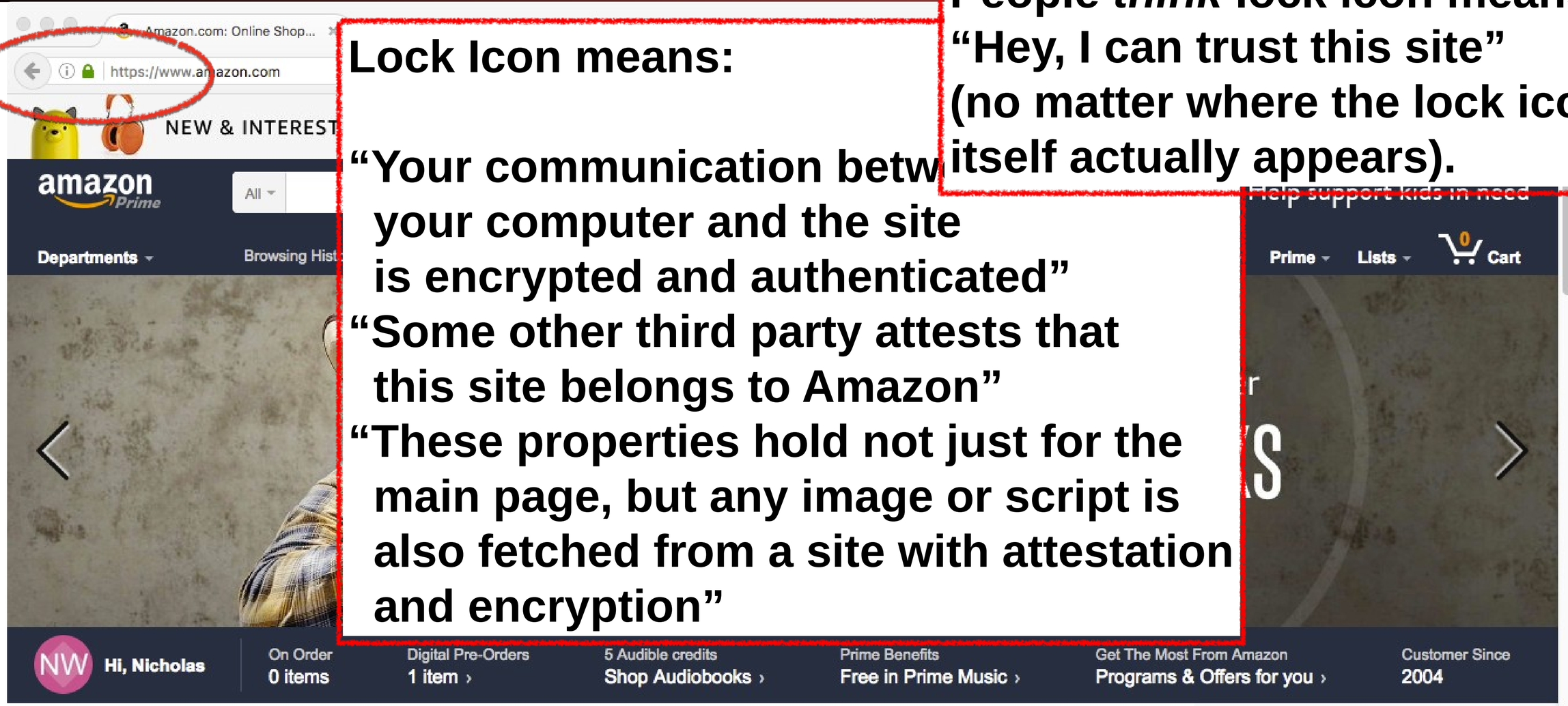
People *think* lock icon means “Hey, I can trust this site” (no matter where the lock icon itself actually appears).

Lock Icon means:

“Your communication betw your computer and the site is encrypted and authenticated”

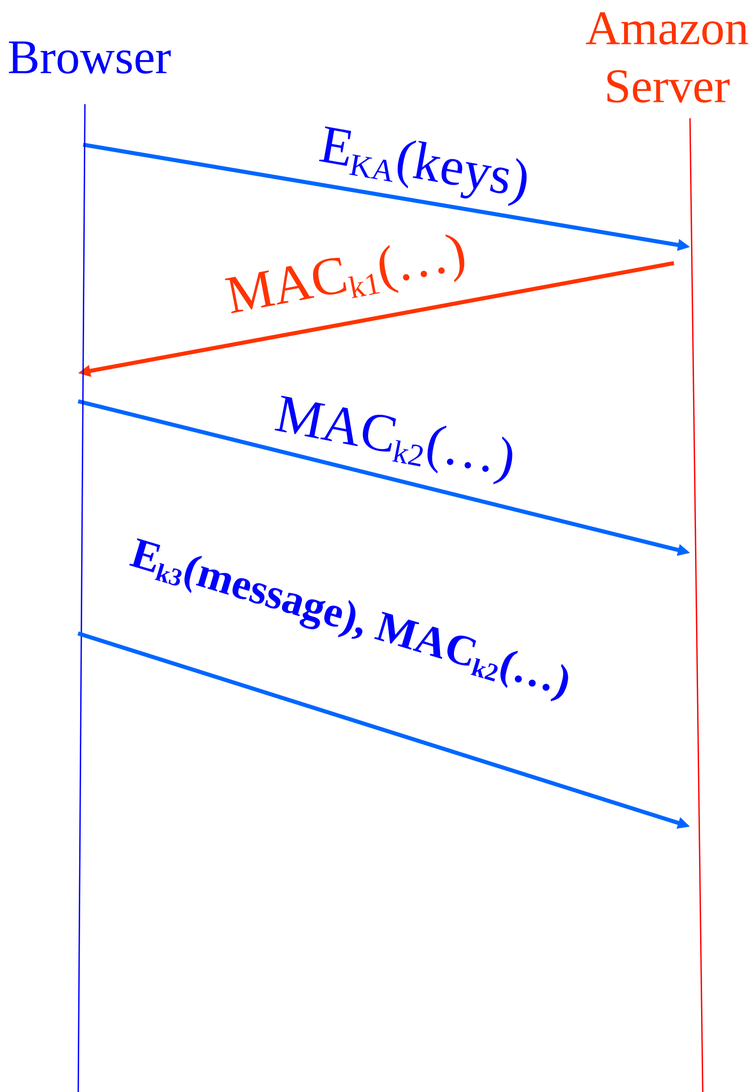
“Some other third party attests that this site belongs to Amazon”

“These properties hold not just for the main page, but any image or script is also fetched from a site with attestation and encryption”



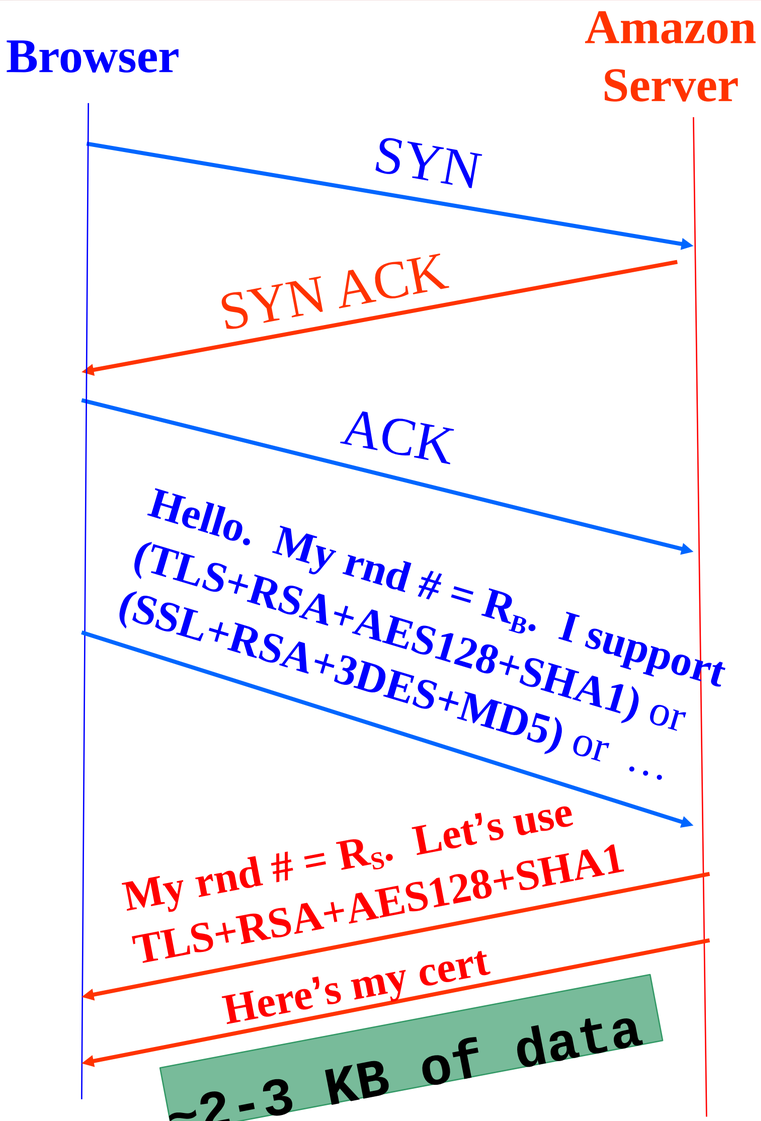
Basic idea

- Browser (client) picks some symmetric keys for encryption + authentication
- Client sends them to server, encrypted using RSA public-key encryption
- Both sides send MACs
- Now they use these keys to encrypt and authenticate all subsequent messages, using symmetric-key crypto



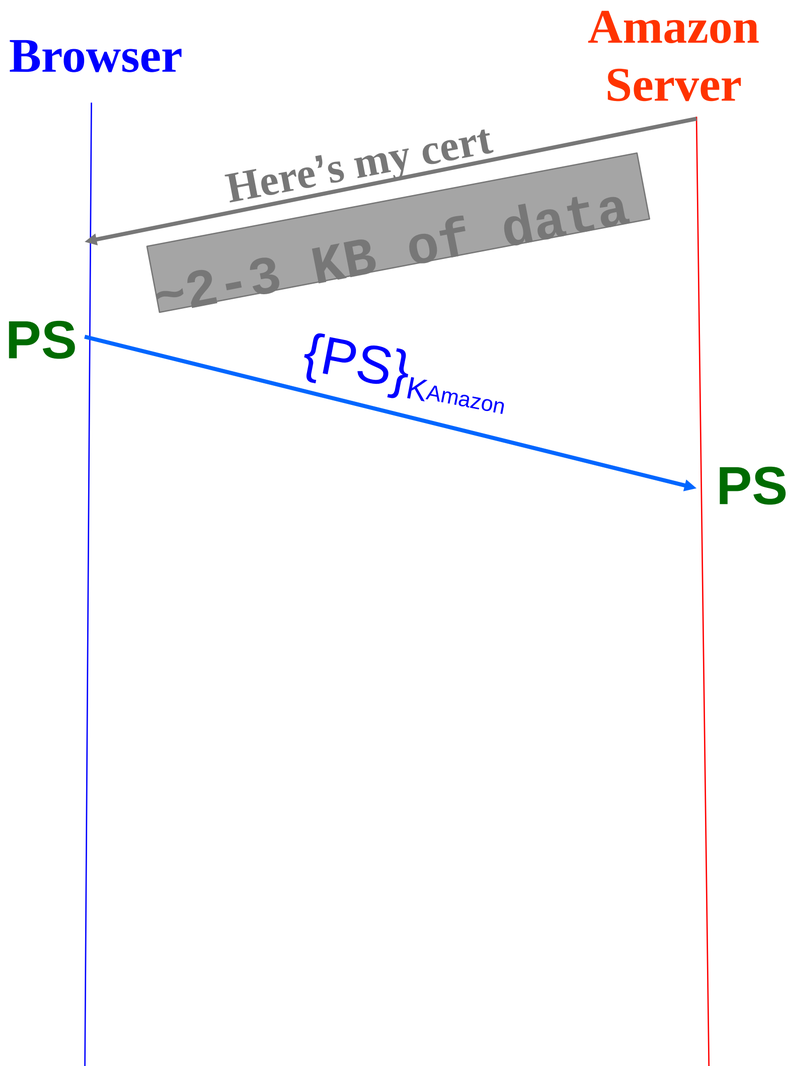
HTTPS Connection (SSL / TLS)

- Browser (client) connects via TCP to Amazon's HTTPS server
- Client picks 256-bit random number R_B, sends over list of crypto protocols it supports
- Server picks 256-bit random number R_S, selects protocols to use for this session
- Server sends over its certificate
  - (all of this is in the clear)
- Client now ***validates***



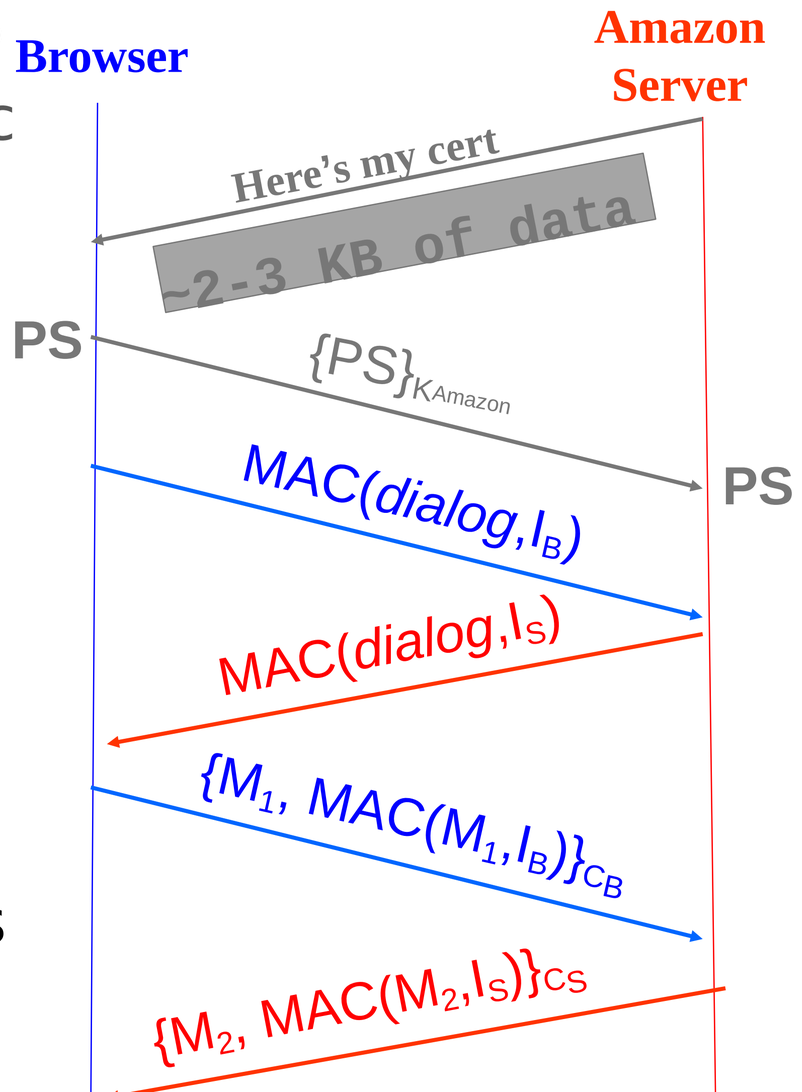
HTTPS Connection (SSL / TLS), cont.

- For RSA, browser constructs "Premaster Secret" PS
- Browser sends PS encrypted using Amazon's public RSA key K_{Amazon}
- Using PS, R_B, and R_S, browser & server derive symmetric cipher keys (C_B, C_S) & MAC integrity keys (I_B, I_S)
  - One pair to use in each direction
  - Done by seeding a pRNG in common between the browser and the server:



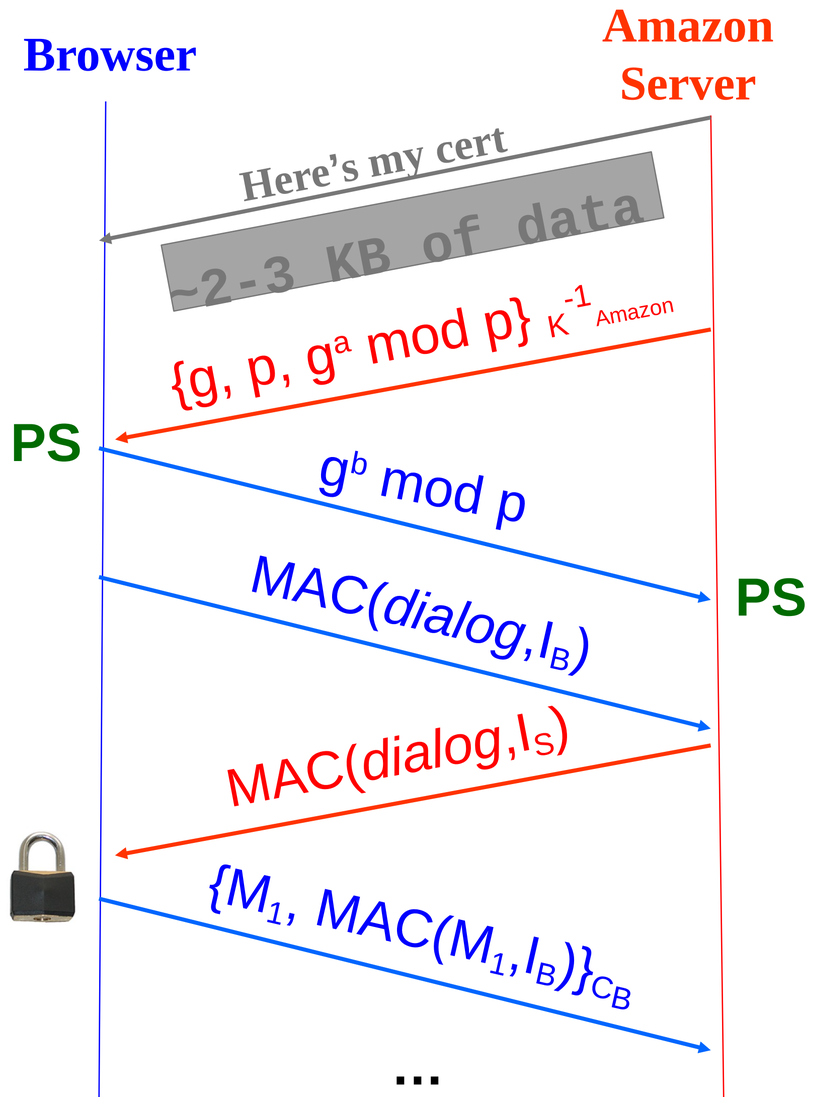
HTTPS Connection (SSL / TLS), cont.

- For RSA, browser constructs “Premaster Secret” PS
- Browser sends PS encrypted using Amazon’s public RSA key KAmazon
- Using PS, R_B, and R_S, browser & server derive symm. cipher keys (C_B, C_S) & MAC integrity keys (I_B, I_S)
  - One pair to use in each direction
- Browser & server exchange MACs computed over entire dialog so far
- If good MAC, Browser displa🔒
- All subsequent communication encrypted w/ symmetric cipher (e.g., AES128) cipher keys, MACs



Alternative: Ephemeral Key Exchange via Diffie-Hellman

- For Diffie-Hellman, server generates random a, sends public parameters and g^a mod p
  - Signed with server's private key
- Browser verifies signature
- Browser generates random b, computes PS = g^{ab} mod p, sends g^b mod p to server
- Server also computes PS = g^{ab} mod p
- Remainder is as before: from PS, R_B, and R_S, browser & server derive symm. cipher keys (C_B, C_S) and MAC integrity keys (I_B, I_S

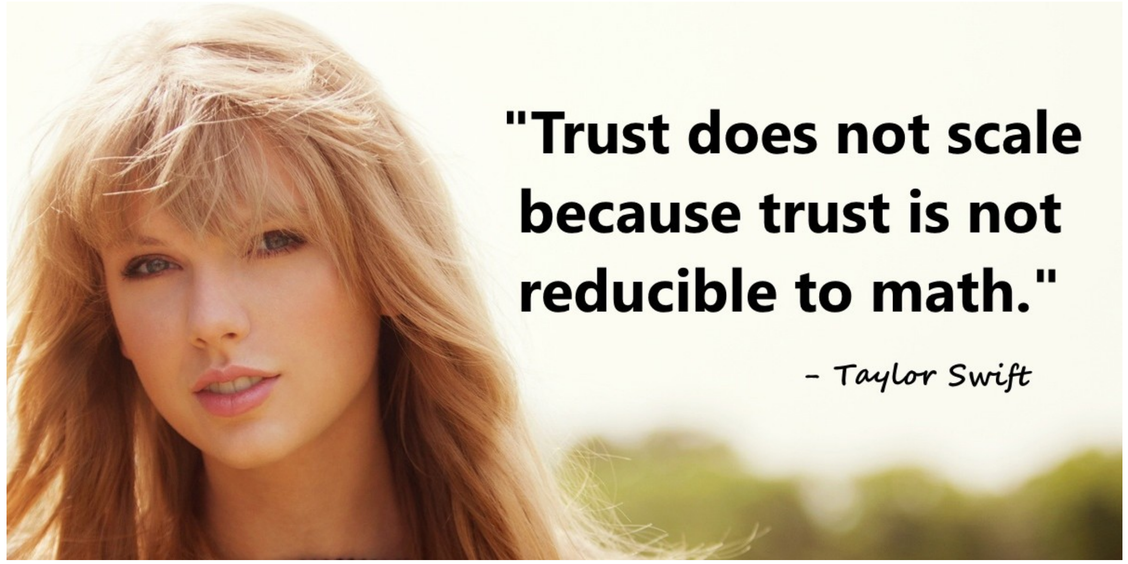


Big Changes for TLS 1.3
Diffie/Hellman and ECDHE only

- The RSA key exchange has a substantial vulnerability
  - If the attacker is ever able to compromise the server and obtain its RSA key… the attacker can decrypt any traffic captured
  - RSA lacks ***forward secrecy***
- So TLS 1.3 uses DHE/ECDHE only
- TLS 1.3 also speeds things up:
  - In the client hello, the client includes $\{g^b \bmod p\}$ for preferred parameters
    - If the server finds it suitable, the server returns $\{g^a \bmod p\}$
  - Saves a round-trip time
- Also only supports AEAD mode encryptions and limited ciphersuites (e.g. GCM)

But What About that “Certificate Validation”

- Certificate validation is used to establish a chain of “trust”
  - It actually is an ***attempt*** to build a scalable trust framework
- This is commonly known as a Public Key Infrastructure (PKI)
  - Your browser is trusting the “Certificate Authority” to be responsible...



Certificates

- Cert = signed statement about someone's public key
  - Note that a cert does not say anything about the identity of who gives you the cert
  - It simply states a given public key K_{Bob} belongs to Bob …
    - … and backs up this statement with a digital signature made using a different public/private key pair, say from Verisign (a "Certificate Authority")
- Bob then can prove his identity to you by you sending him something encrypted with K_{Bob} …
  - … which he then demonstrates he can read
- … or by signing something he demonstrably uses
- Works provided you trust that you have a valid copy of Verisign's public key …

Validating Amazon's Identity

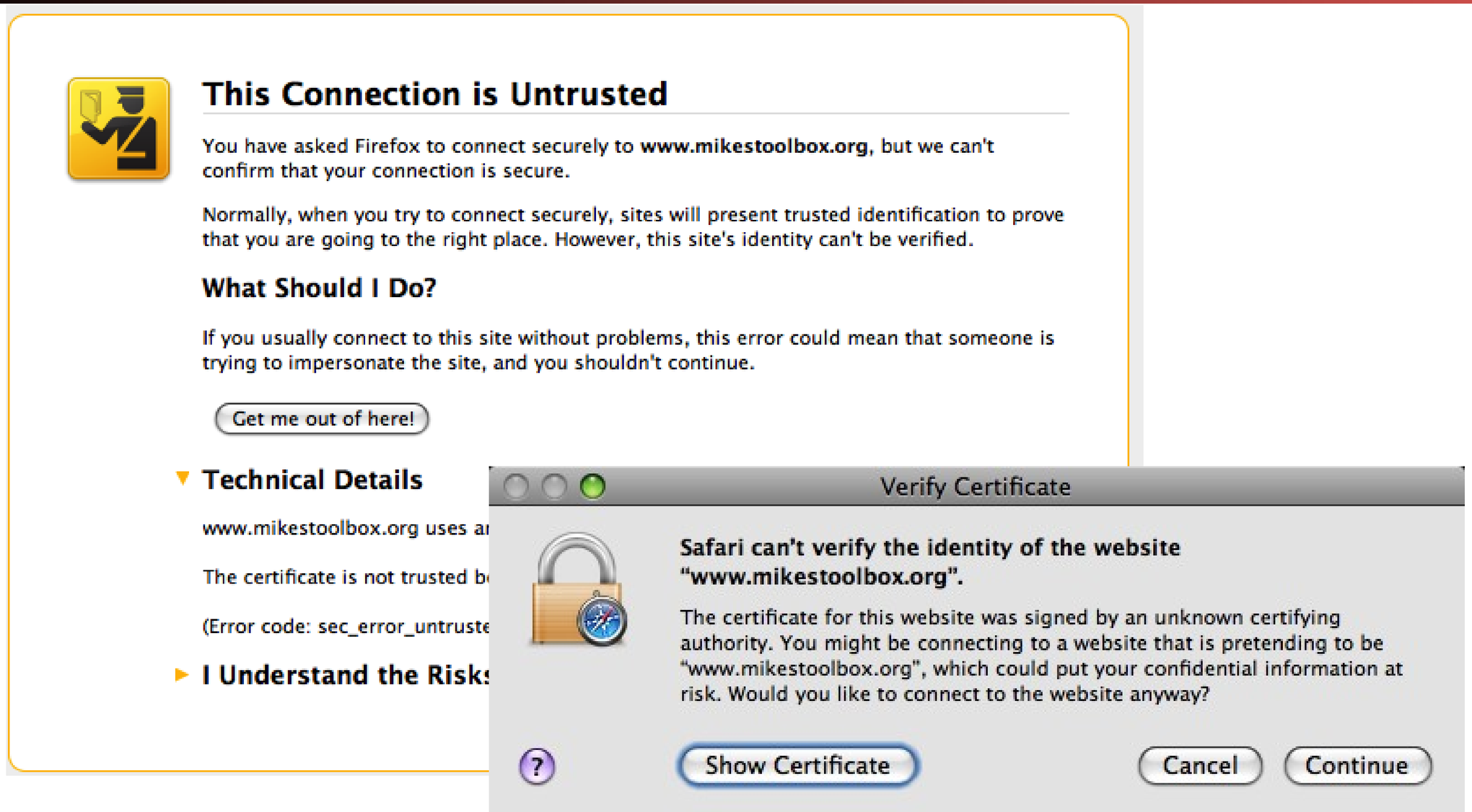
- Browser compares domain name in cert w/ URL
  - Note: this provides an ***end-to-end*** property
    (as opposed to say a cert associated with an IP address)
- Browser accesses separate cert belonging to issuer
  - These are hardwired into the browser – ***and trusted***!
  - There could be a chain of these …
- Browser applies issuer's public key to verify signature **S**, obtaining the hash of what the issuer signed
  - Compares with its own SHA-1 hash of Amazon's cert
- Assuming hashes match, now have high confidence it's indeed Amazon's public key …

End-to-End ⇒ Powerful Protections

- Attacker runs a sniffer to capture our WiFi session?
  - But: encrypted communication is unreadable
    - Attacker doesn’t learn contents, but learns metadata (browsing history)!
- DNS cache poisoning?
  - Client goes to wrong server
  - But: detects impersonation
    - No problem!
- Attacker hijacks our connection, injects new traffic
  - But: data receiver rejects it due to failed integrity check since all communication has a mac on it
    - No problem!
- Only thing a ***full man-in-the-middle*** attacker can do is inject RSTs, inject invalid packets, or drop packets: limited to DoS

Validating Amazon’s Identity, cont.

- Browser retrieves cert belonging to the issuer
  - These are hardwired into the browser – and trusted!


This Connection is Untrusted
You have asked Firefox to connect securely to www.mikestoolbox.org, but we can't confirm that your connection is secure.
Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.
What Should I Do?
If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.
Get me out of here!
Technical Details
www.mikestoolbox.org uses ar
The certificate is not trusted b
(Error code: sec_error_untruste
I Understand the Risks
Verify Certificate
Safari can't verify the identity of the website "www.mikestoolbox.org".
The certificate for this website was signed by an unknown certifying authority. You might be connecting to a website that is pretending to be "www.mikestoolbox.org", which could put your confidential information at risk. Would you like to connect to the website anyway?
Show Certificate
Cancel
Continue

Validating Amazon's Identity, cont.

- Browser retrieves cert belonging to the issuer
  - These are hardwired into the browser – and trusted!
- What if browser can't find a cert for the issuer?
- If it can't find the cert, then warns the user that site has not been verified
  - Can still proceed, just without authentication
- Q: Which end-to-end security properties do we lose if we incorrectly trust that the site is whom we think?
- A: All of them!
  - Goodbye confidentiality, integrity, authentication

SSL / TLS Limitations

- Properly used, SSL / TLS provides powerful end-to-end protections
- So why not use it for everything??
- Issues:
  - Cost of public-key crypto (fairly minor)
    - Takes non-trivial CPU processing (but today a minor issue)
    - Note: symmetric key crypto on modern hardware is effectively free
  - Hassle of buying/maintaining certs (fairly minor)
    - LetsEncrypt makes this almost automatic
  - Integrating with other sites that don’t use HTTPS
    - Namely, you can’t: Non-HTTPS content won’t load!

SSL / TLS Limitations, cont.

- Problems that SSL / TLS does not take care of ?
- Censorship:
  - The censor sees the certificate in the clear, so knows who the client is talking to
  - Optional Server Name Identification (SNI) is also sent in the clear
  - The censor can then inject RSTs or block the communication
- SQL injection/XSS/CSRF/server-side coding/logic flaws

SSL/TLS Problem: Revocation

- A site screws up and an attacker steals the private key associated with a certificate, what now?
  - Certificates have a timestamp and are only good for a specified time
    - But this time is measured in years!?!?
- Two mitigations:
  - Certificate revocation lists
    - Your browser occasionally calls back to get a list of "no longer accepted" certificates
  - OSCP
    - Online Certificate Status Protocol: https://en.wikipedia.org/wiki/Online_Certificate_Status_Protocol

“sslstrip”
(Amazon FINALLY fixed this recently)

Regular web surfing: http: URL

So *no* ***integrity*** - a MITM attacker can alter pages returned by server …

And when we click here …
… attacker has changed the corresponding link so that it’s ordinary http rather than https!

We never get a chance to use TLS’s protections! :-(

SSL / TLS Limitations, cont.

- Problems that SSL / TLS does not take care of ?
- Censorship
- SQL injection / XSS / server-side coding/logic flaws
- Vulnerabilities introduced by server inconsistencies
- Browser and server bugs
- Bad passwords