

# Network Security 4



**matt blaze** ✓  
@mattblaze

Following

Cryptocurrency somehow combines everything we love about religious fanatics with everything we love about Ponzi schemes.

7:08 AM - 23 Oct 2017

68 Retweets 128 Likes



4

68

128



Tweet your reply

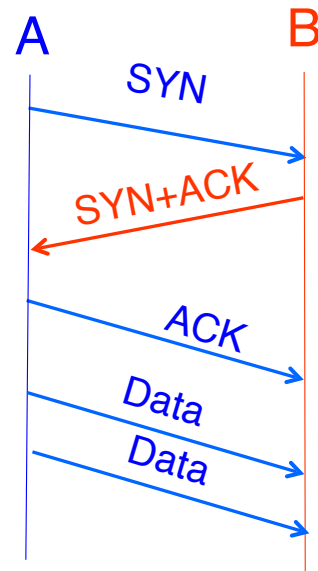


**matt blaze** ✓ @mattblaze · 18m

Replying to @mattblaze

There's a lot to dislike about the world we're in, but at least Ayn Rand didn't have bitcoin to write about.

# 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 that the ACK correctly used the sequence number

# Easy SYN-cookies: HMAC

- On startup create a random key  $k$ ...
- For the server ISN:
  - $\text{HMAC}(k, \text{SIP}|\text{DIP}|\text{SPORT}|\text{DPORT}|\text{client\_ISN})$
- Upon receipt of the ACK
  - Verify that ACK is based off  $\text{HMAC}(k, \text{SIP}|\text{DIP}|\text{SPORT}|\text{DPORT}|\text{client\_ISN})$ 
    - 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 ...









# 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 = “Nick's public key is  $K_{\text{Nick}} = 0xF32A99B...$ ”  
Cert: M,
  - $\{ \text{“Nick's public key ... } 0xF32A99B...” \}_{K^{-1}_{\text{Gavin}}}$   
 $= 0x923AB95E12...9772F$



# *Certificate*



Gavin Newsom hereby asserts:

Nick's public key is  $K_{\text{Grant}} = \mathbf{0xF32A99B...}$

The signature for this statement using

$K_{\text{Gavin}}^{-1}$  is  $\mathbf{0x923AB95E12...9772F}$

# Certificate



Gavin Newsom hereby asserts:

Nick's public key is  $K_{\text{Nick}} = 0xF32A99B...$

The signature for this statement using

$K^{-1}$  **This** is  $0x923AB95E12...9772F$



# Certificate



Gavin Newsom hereby asserts:

Nick's public key is  $K_{\text{Grant}} = 0xF32A99B...$

The signature  $f$  is computed over all of this

$K_{\text{Gavin}}^{-1}$  is  $0x923AB95E12...9772F$

# Certificate



Gavin Newsom hereby asserts:

Grant's public key is  $K_{\text{Grant}} = \mathbf{0xF32A99B...}$

The signature for this statement using

$K_{\text{Gavin}}^{-1}$  is  $\mathbf{0x923AB95E12...9772F}$

and can be  
*validated* using:



# Certificate



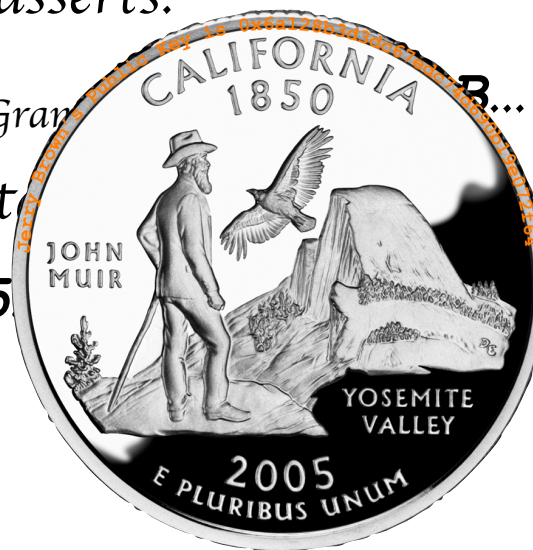
**This:**

Gavin Newsom hereby asserts:

Grant's public key is  $K_{\text{Grant}}$  B...

The signature for this st

$K_{\text{Gavin}}^{-1}$  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 Nick'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  $K_{\text{nick}}$ , then we have high confidence they possess  $K^{-1}_{\text{Nick}}$ 
  - Same for if they show they can sign "using"  $K_{\text{Nick}}$

# Scaling Digital Certificates

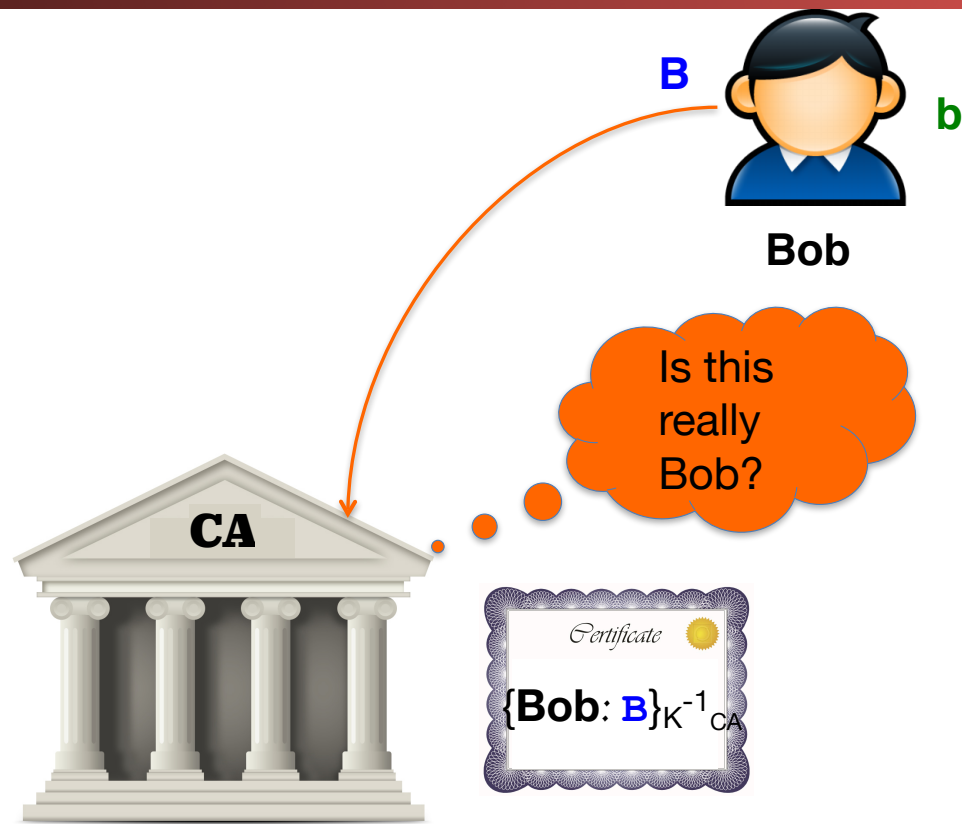
- How can this possibly scale? Surely Gavin can't sign everyone's public key!
- Approach #1: Introduce hierarchy via delegation
  - { "Janet Napolitano's public key is 0x... and I trust her to vouch for UC" }  $K^{-1}_{\text{Gavin}}$
  - { "Carol Christ's public key is 0x... and I trust him to vouch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

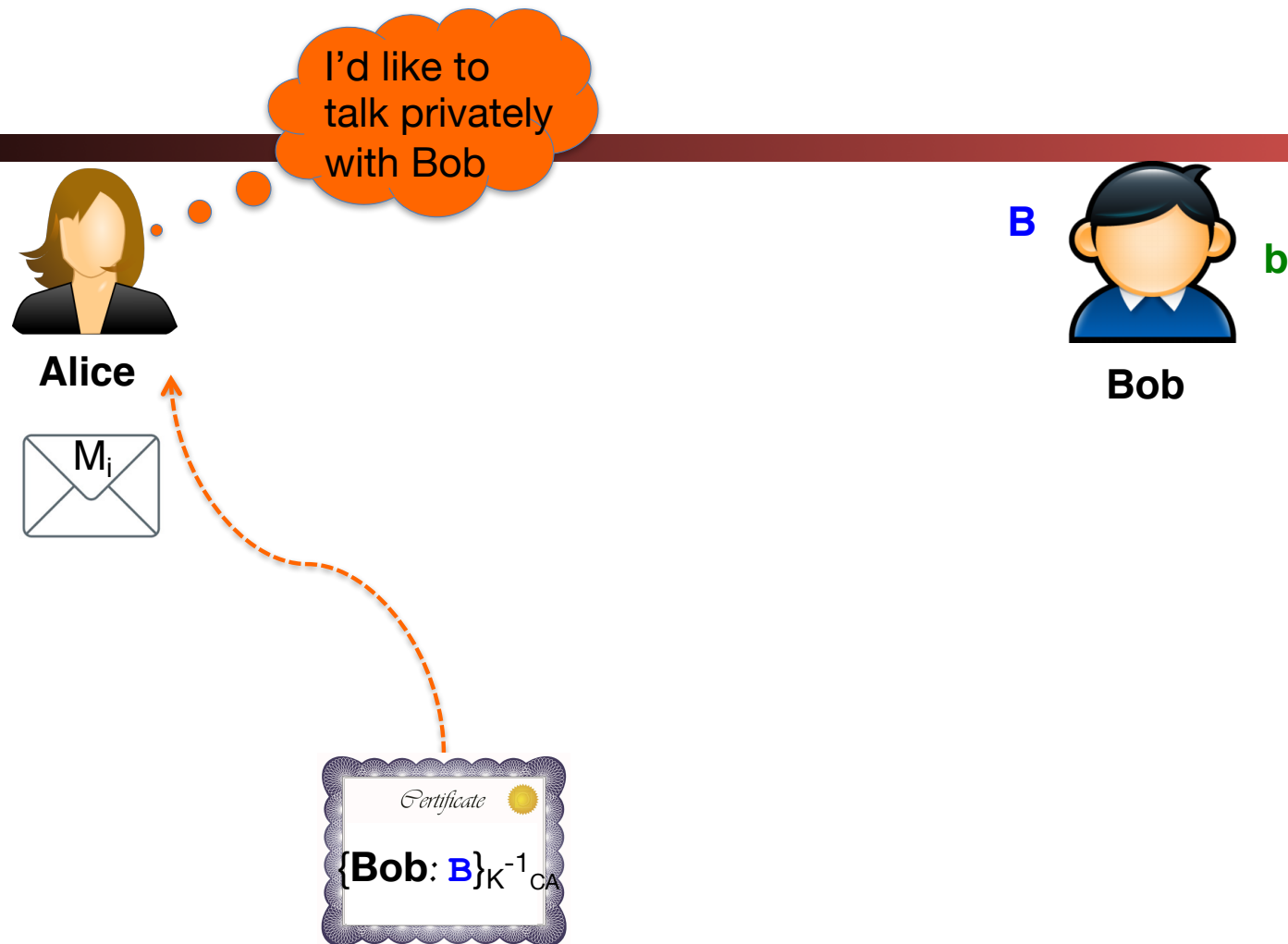
# Scaling Digital Certificates, con't

- Nick 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)

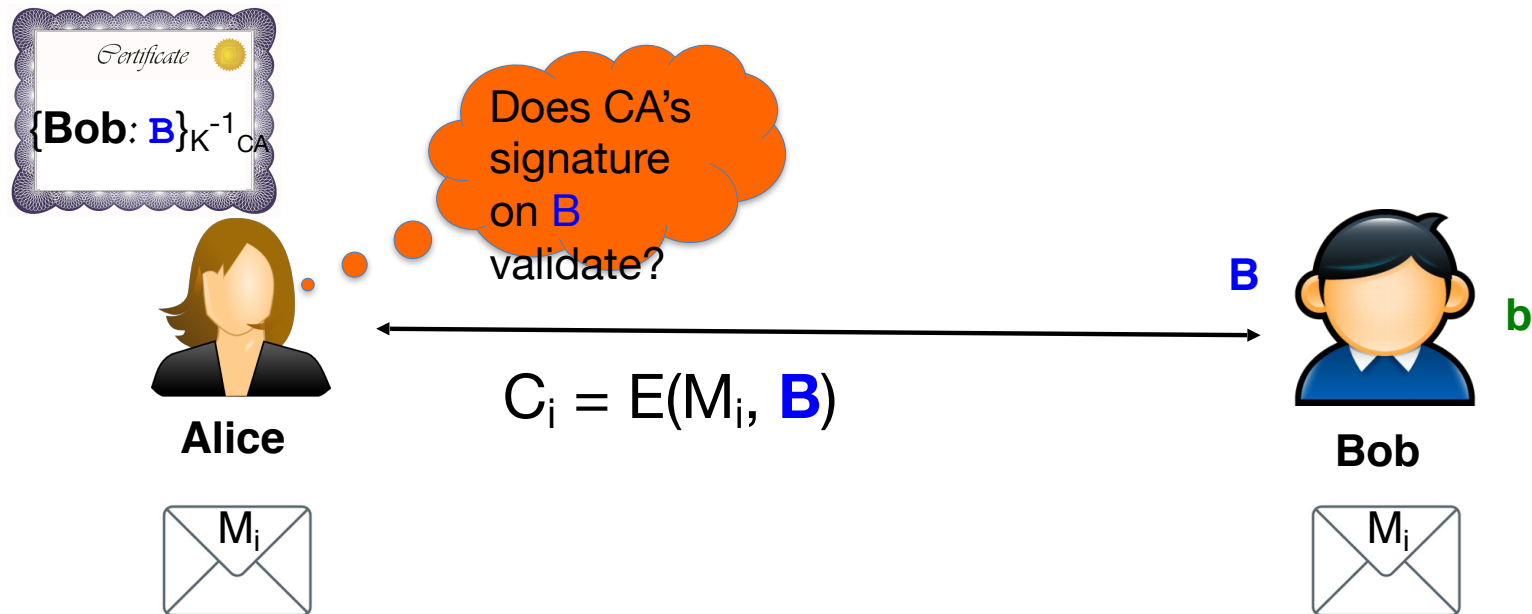
# Certificate Authorities

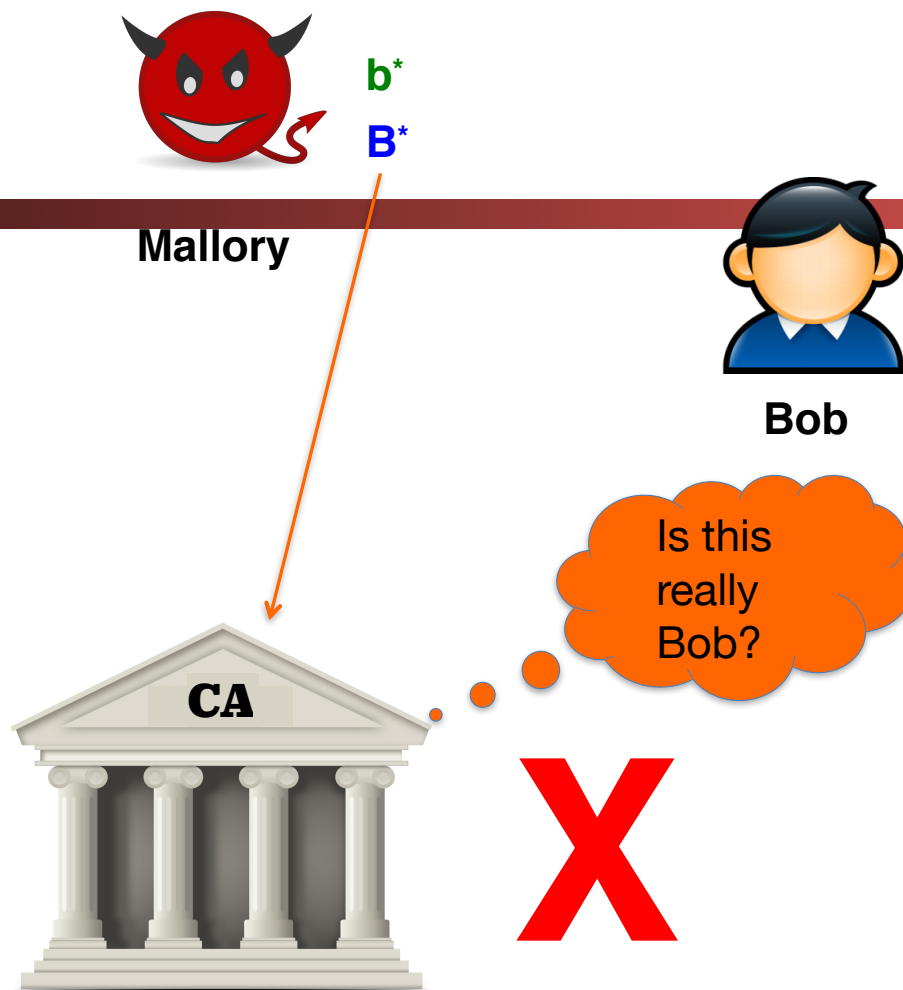
- CAs are trusted parties in a Public Key Infrastructure (PKI)
- They can operate offline
  - They sign (“cut”) certs when convenient, not on-the-fly (... though see below ...)
- Suppose Alice wants to communicate confidentially w/ Bob:
  - Bob gets a CA to issue {Bob’s public key is B}  $K_{CA}^{-1}$
  - 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  $\{M\}K_B$  to Bob

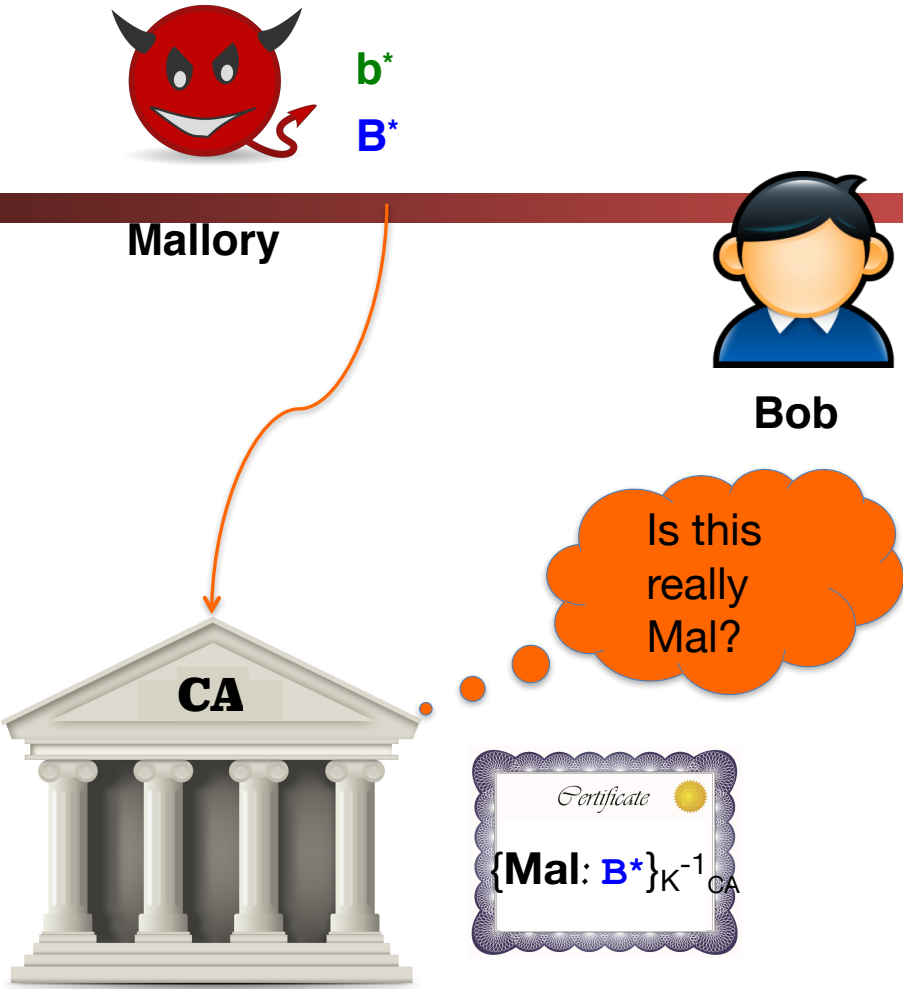


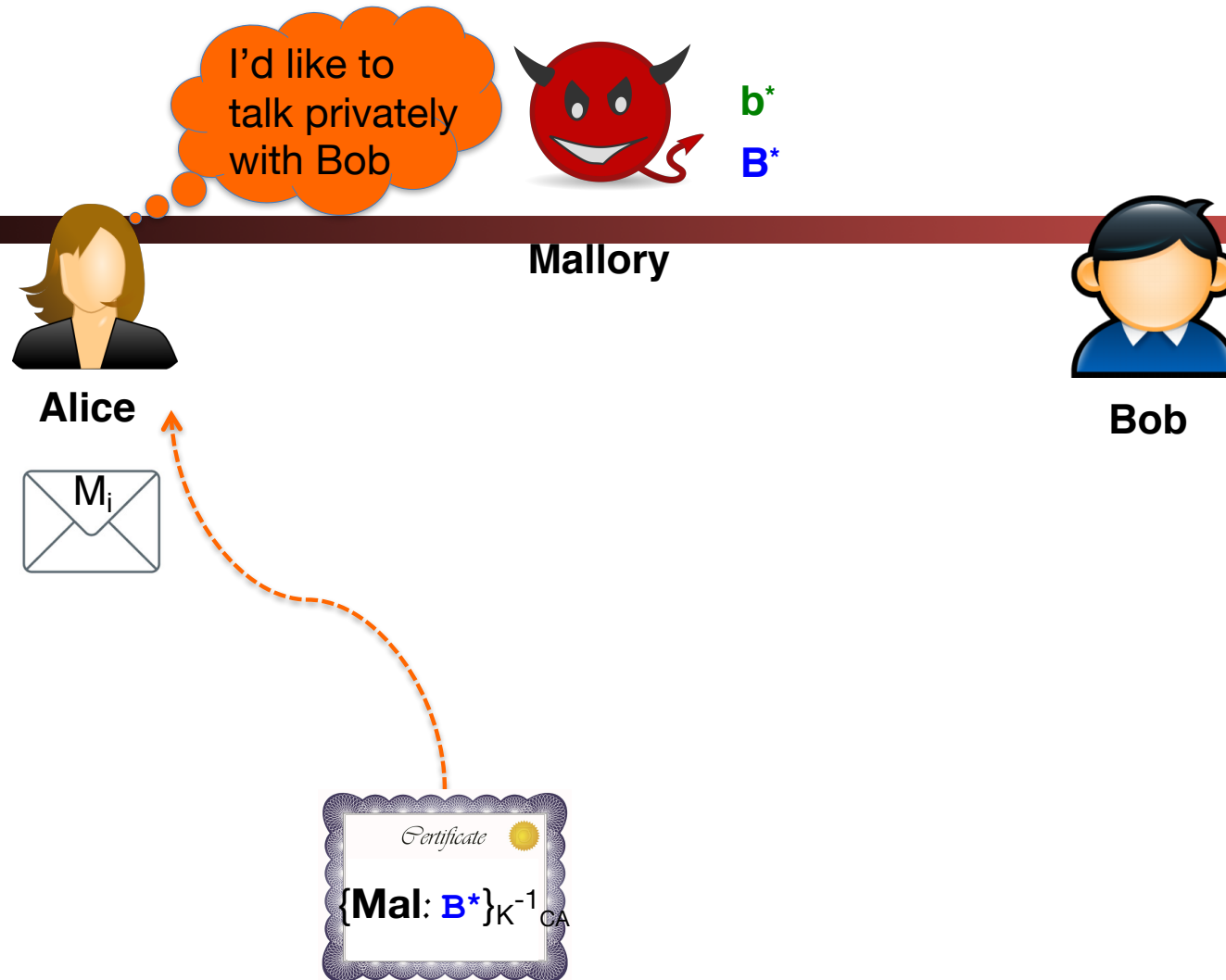


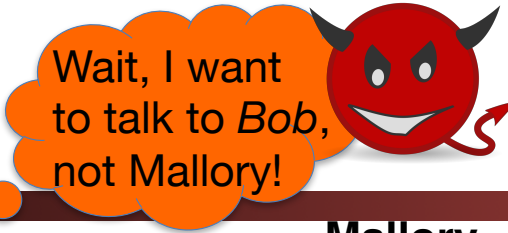












$b^*$   
 $B^*$



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

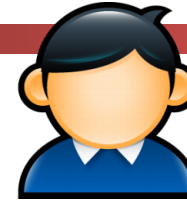


$b^*$   
 $B^*$

Mallory



Alice



Bob



# Revocation

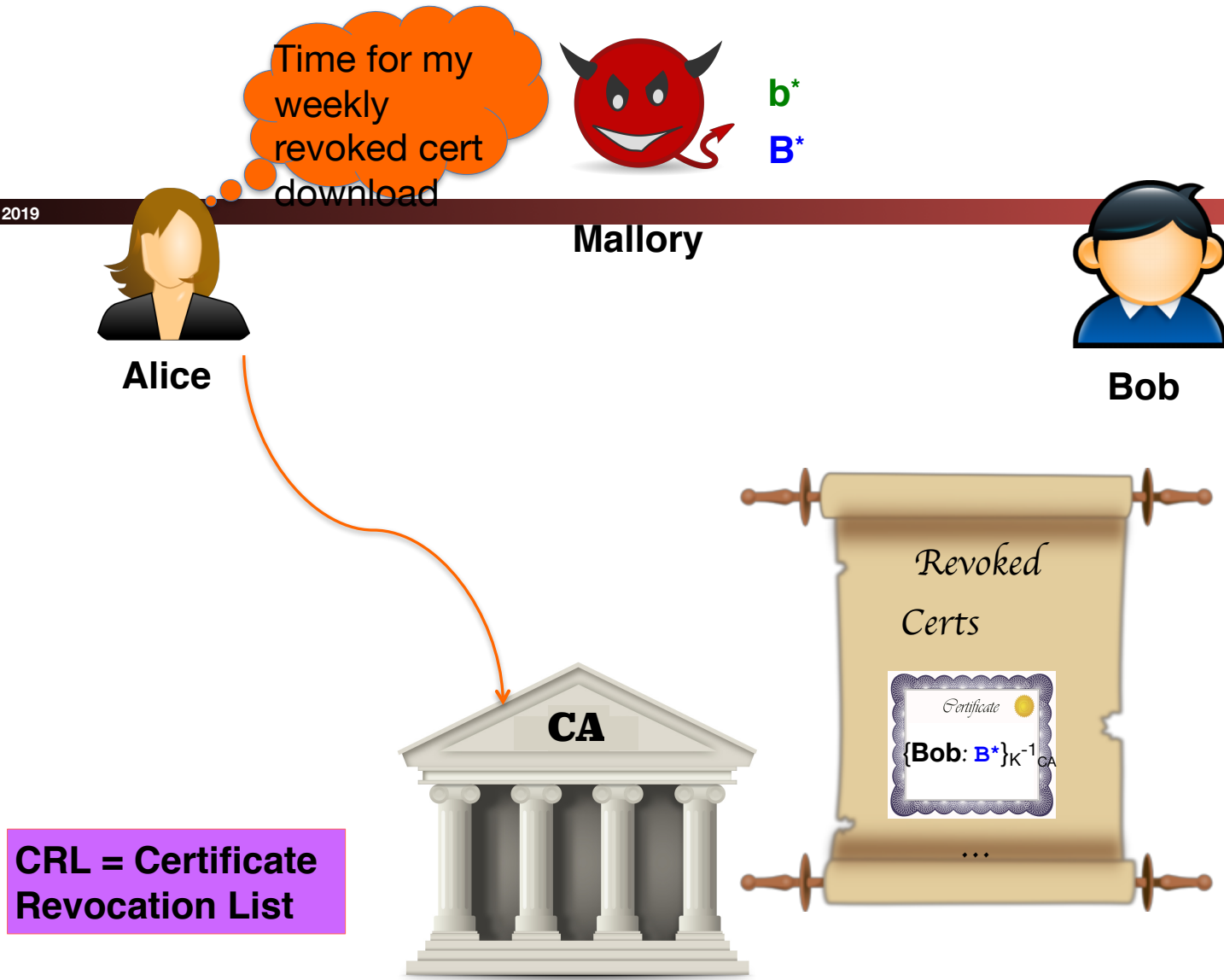
- What do we do if a CA **screws up** and issues a cert in Bob's name to Mallory?
  - E.g. Verisign issued a **Microsoft.com** cert to a **Random Joe**
  - (Related problem: Bob realizes **b** has been **stolen**)
- **How do we recover from the error?**
- **Approach #1: expiration dates**
  - Mitigates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





# Revocation, con't

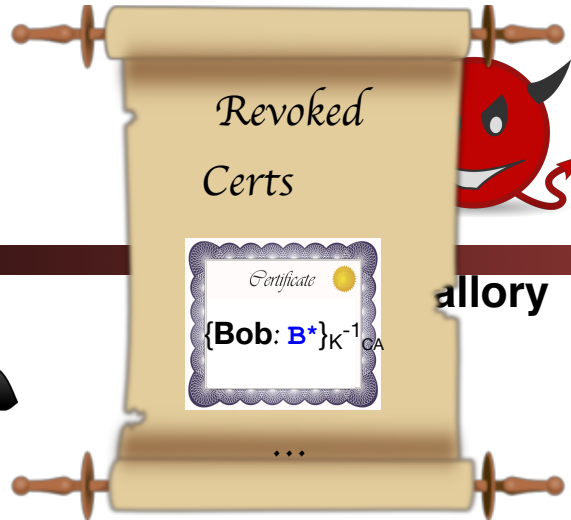
- Approach #2: announce revoked certs
  - Users periodically download cert revocation list (CRL)



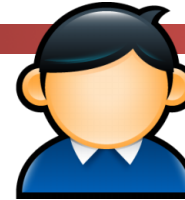
Oof!



Alice



$b^*$   
 $B^*$



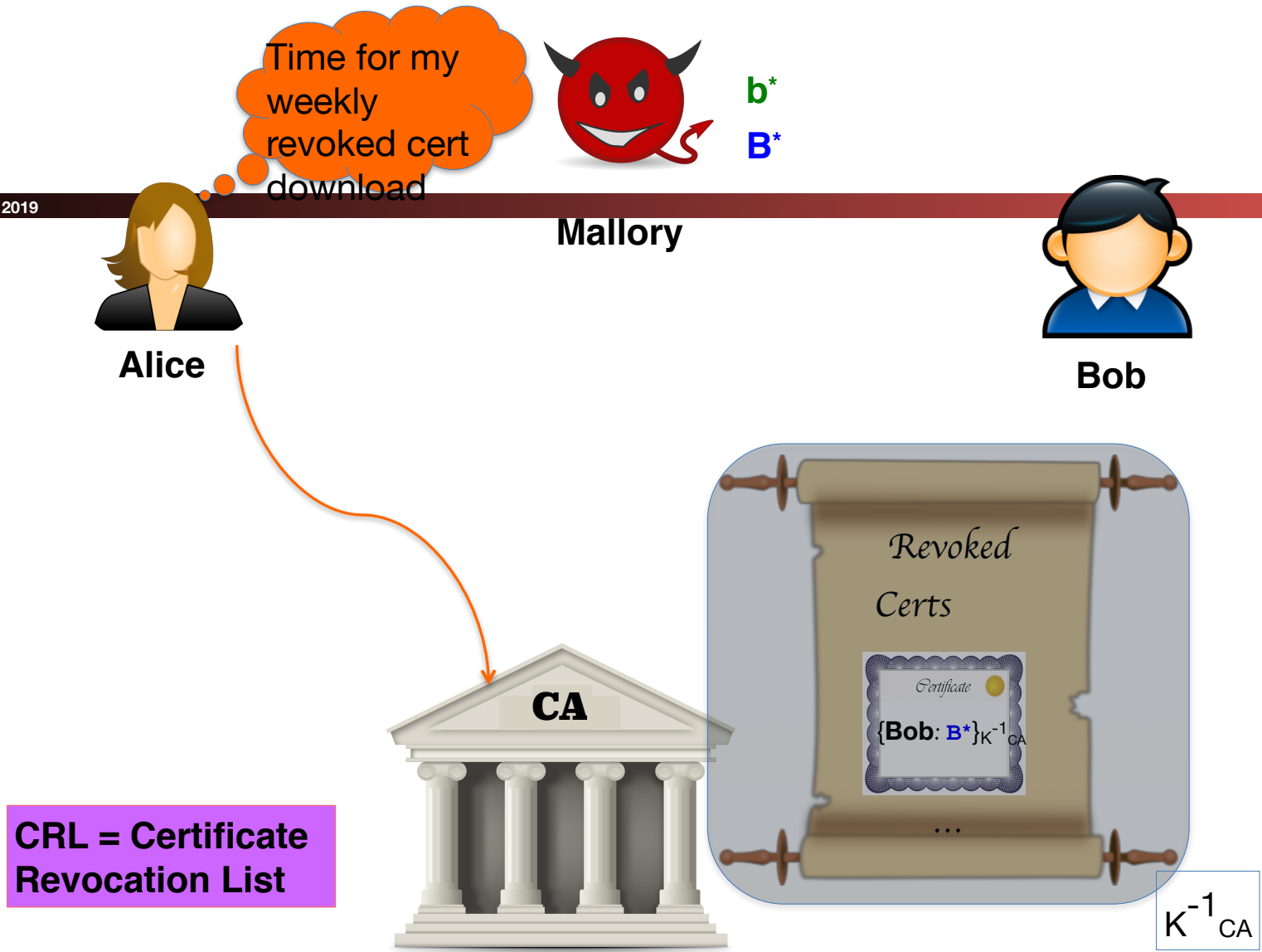
Bob



CRL = Certificate  
Revocation List

# Revocation, con't

- Approach #2: announce revoked certs
  - Users periodically download cert revocation list (CRL)
- Issues?
  - Lists can get large
  - Need to authenticate the list itself – how?



# 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 if Mallory can “DoS” CA (DoS = denial-of-service)



# 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!

# 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

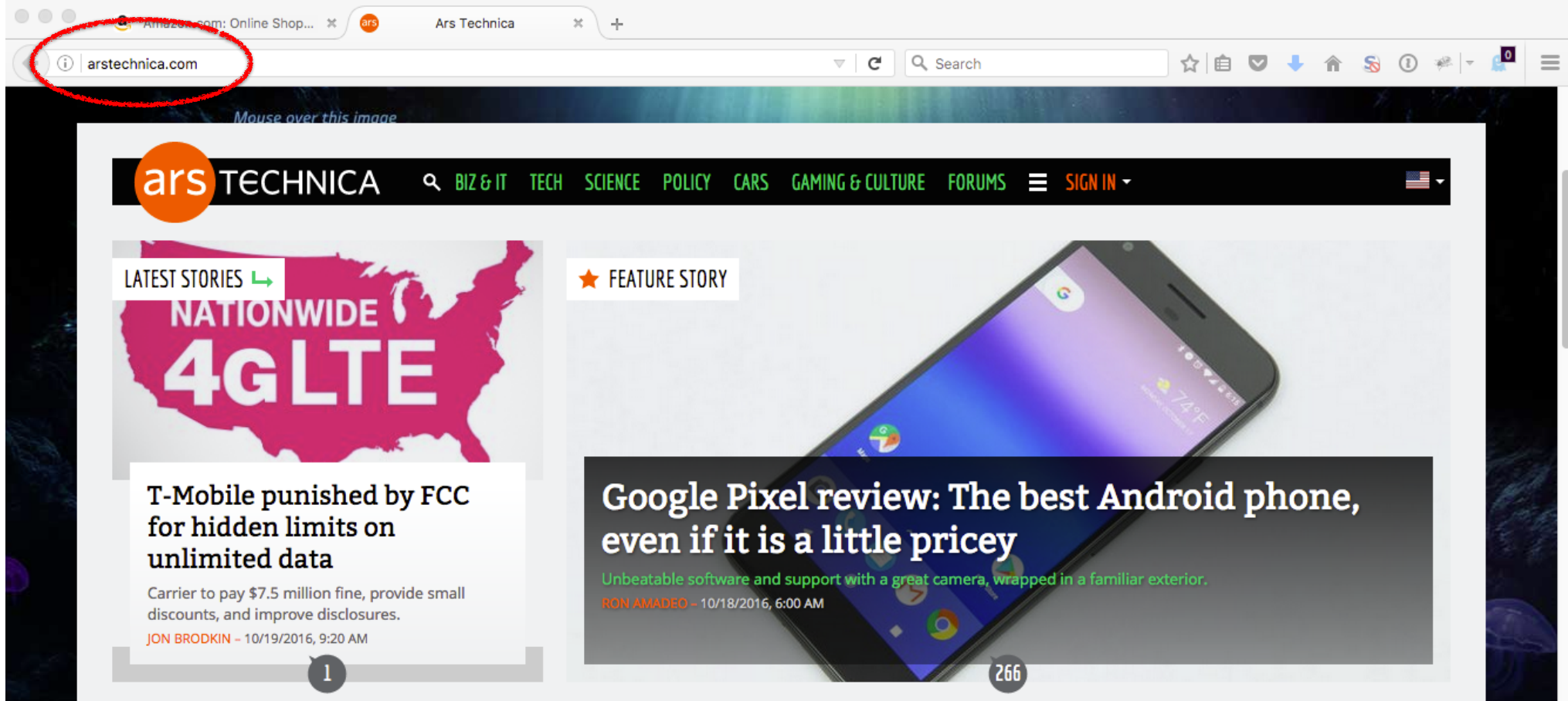
- End-to-end = communication protections achieved all the way from originating client to intended server
  - With no need to trust intermediaries
- Dealing with threats:
  - Eavesdropping?
    - Encryption (including session keys)
  - Manipulation (injection, MITM)?
    - Integrity (use of a MAC); replay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

( What's missing?  
Availability ... )

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

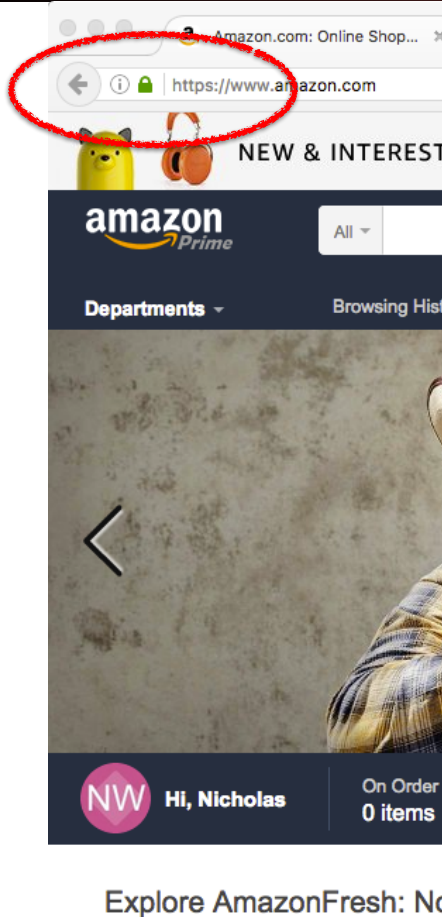
- SSL = Secure Sockets Layer (predecessor)
- TLS = Transport Layer Security (standard)
  - Both terms used interchangeably
- Security for any application that uses TCP
  - Secure = encryption/confidentiality + integrity + authentication (of server, but not of client)
- Multiple uses
  - Puts the 's'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

# An “Insecure” Web Page



# A “Secure” Web Page

Computer Science 161 Spring 2019



The screenshot shows the Amazon homepage. A red circle highlights the lock icon in the browser's address bar, which is next to the URL `https://www.amazon.com`. The Amazon logo and Prime banner are visible at the top. The page content includes a large image of a person's face and various promotional banners at the bottom.

**Lock Icon means:**

- “Your communication between 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”

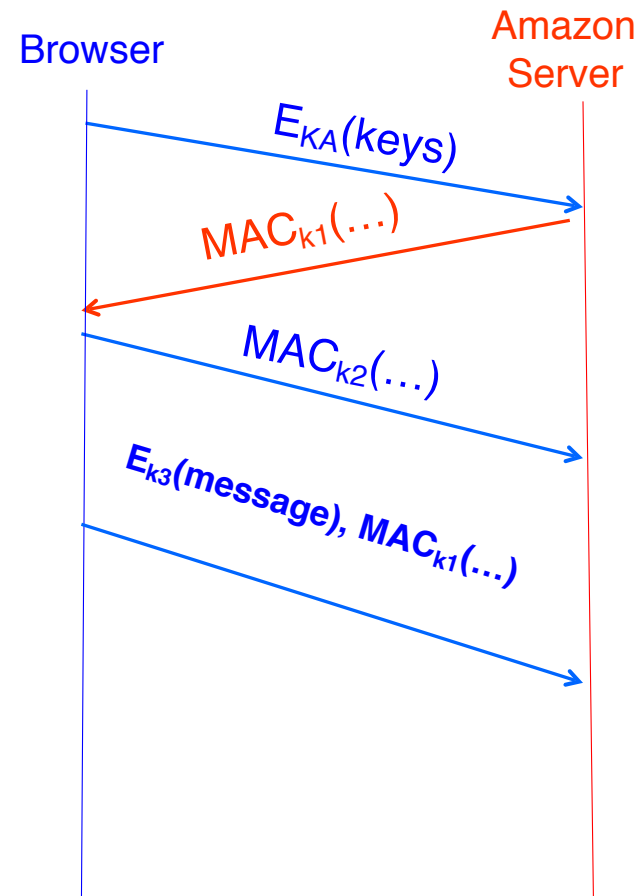
**People *think* lock icon means “Hey, I can trust this site” (no matter where the lock icon itself actually appears).**

Explore AmazonFresh: Now just \$14.99/month [Learn more](#)

Amazon Gift Cards

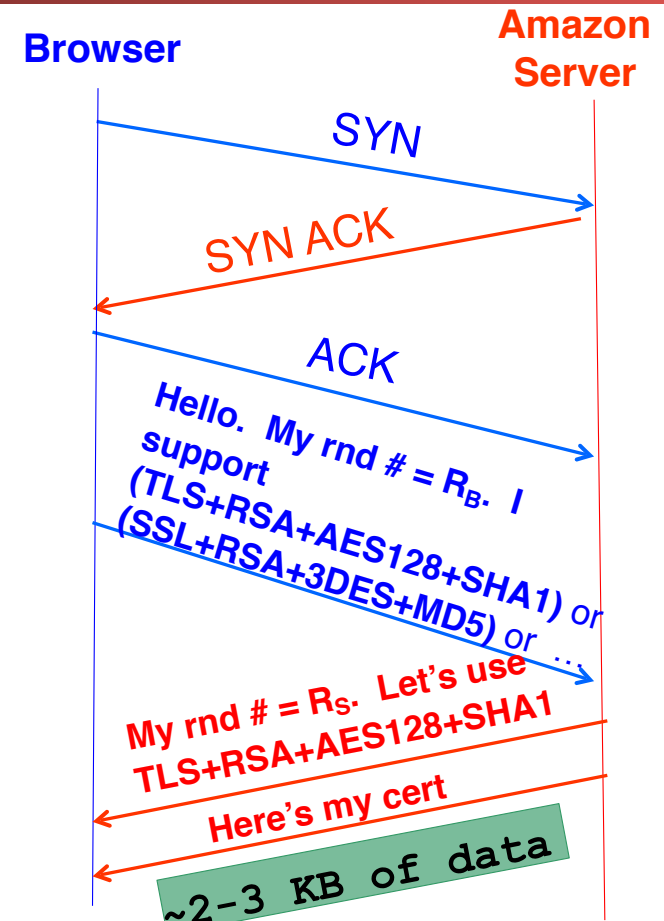
# Basic idea

- 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



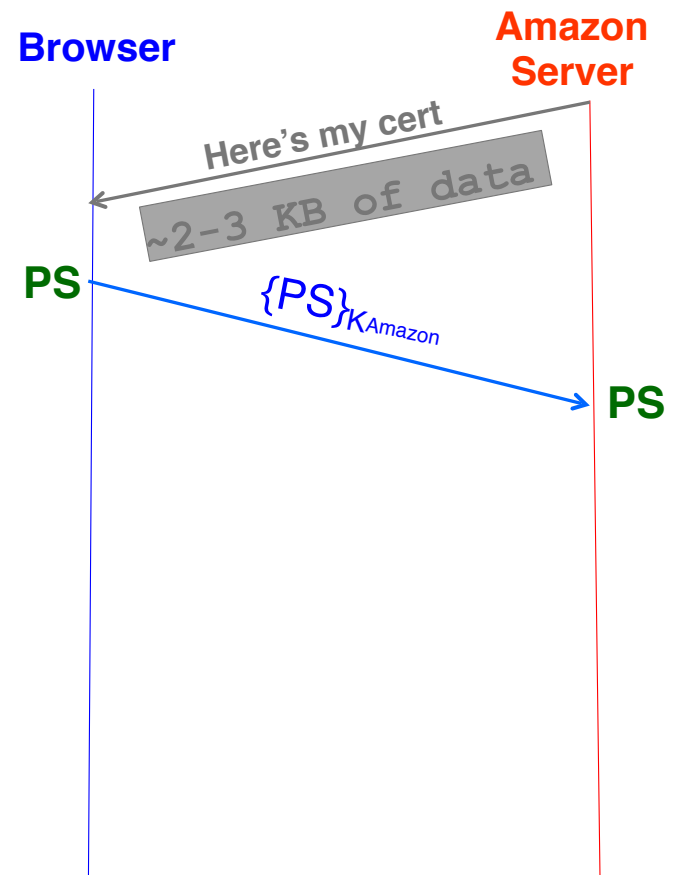
# HTTPS Connection (SSL / TLS)

- 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** cert




# HTTPS Connection (SSL / TLS), cont.

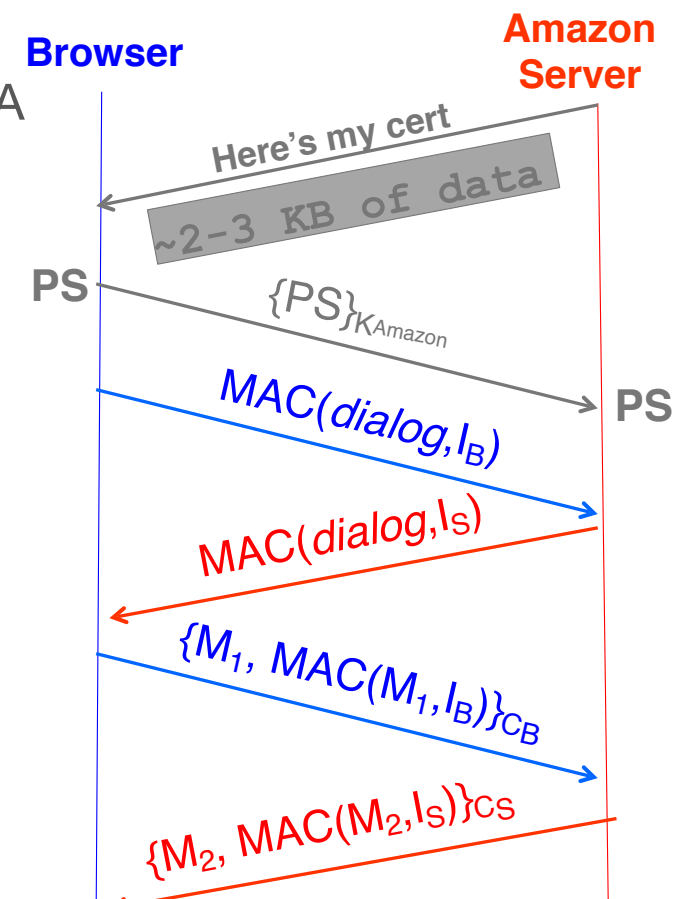
- For RSA, browser constructs “Premaster Secret” PS
- Browser sends PS encrypted using Amazon’s public RSA key  $K_{\text{Amazon}}$
- Using PS,  $R_B$ , and  $R_S$ , browser & server derive symmetric cipher keys ( $C_B$ ,  $C_S$ ) & MAC integrity keys ( $I_B$ ,  $I_S$ )
  - One pair to use in each direction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





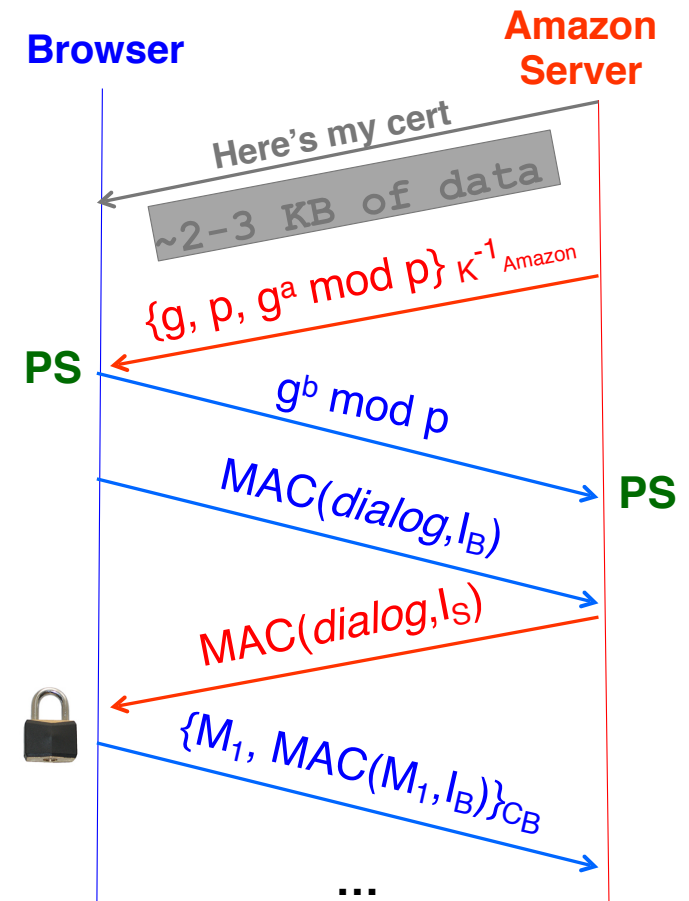
# HTTPS Connection (SSL / TLS), cont.

- For RSA, browser constructs “Premaster Secret” PS
- Browser sends PS encrypted using Amazon’s public RSA key  $K_{\text{Amazon}}$
- 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 displays 
- All subsequent communication encrypted w/ symmetric cipher (e.g., AES128) cipher keys, MACs
  - Sequence #'s thwart replay attacks



# Alternative: Ephemeral Key Exchange via Diffie-Hellman

- For Diffie-Hellman, server generates random  $a$ , sends public parameters and  $g^a \bmod p$ 
  - Signed with server's private key
- Browser verifies signature
- Browser generates random  $b$ , computes  $PS = g^{ab} \bmod p$ , sends  $g^b \bmod p$  to server
- Server also computes  $PS = g^{ab} \bmod p$
- 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 ( $I_B$ ,  $I_S$ ), etc...



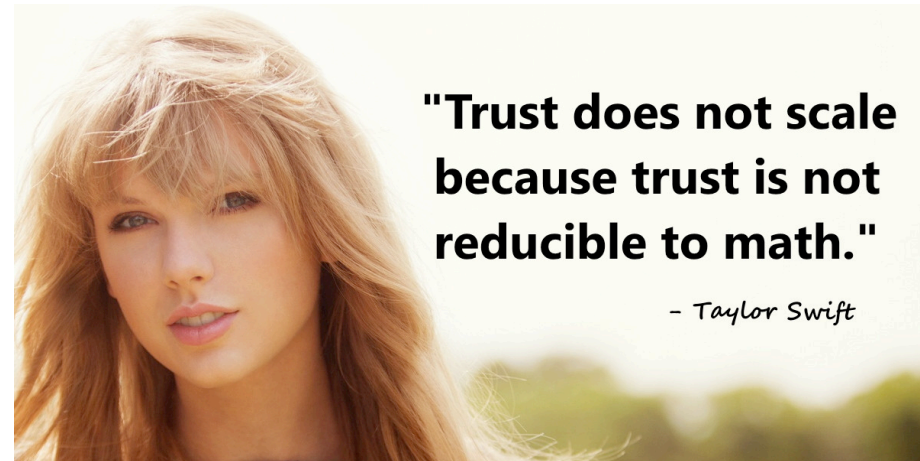
# Big Changes for TLS 1.3

## 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...



**"Trust does not scale  
because trust is not  
reducible to math."**

*- Taylor Swift*

# 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  $K_{\text{Bob}}$  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  $K_{\text{Bob}}$  ...
  - ... 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 ...
  - ... and you trust Verisign to use prudence when she signs other people's keys

# 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 ...
  - assuming signatory is trustworthy, didn't lose private key, wasn't tricked into signing someone else's certificate, and that Amazon didn't lose their key either...

# End-to-End $\Rightarrow$ Powerful Protections

- Attacker runs a sniffer to capture our WiFi session?
  - But: encrypted communication is unreadable
    - No problem!
- 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 a ***denial of service***



# Validating Amazon's Identity, cont.

- Browser retrieves cert belonging to the issuer
  - These are hardwired into the browser – and trusted!
- But what if the browser can't find a cert for the issuer?



## 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 an

The certificate is not trusted by

(Error code: sec\_error\_untruste

### ► I Understand the Risks



# 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
  - Active attacker can read everything, modify, impersonate

# 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!
  - Latency: extra round trips  $\Rightarrow$  1st page slower to 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
- Vulnerabilities introduced by server inconsistencies

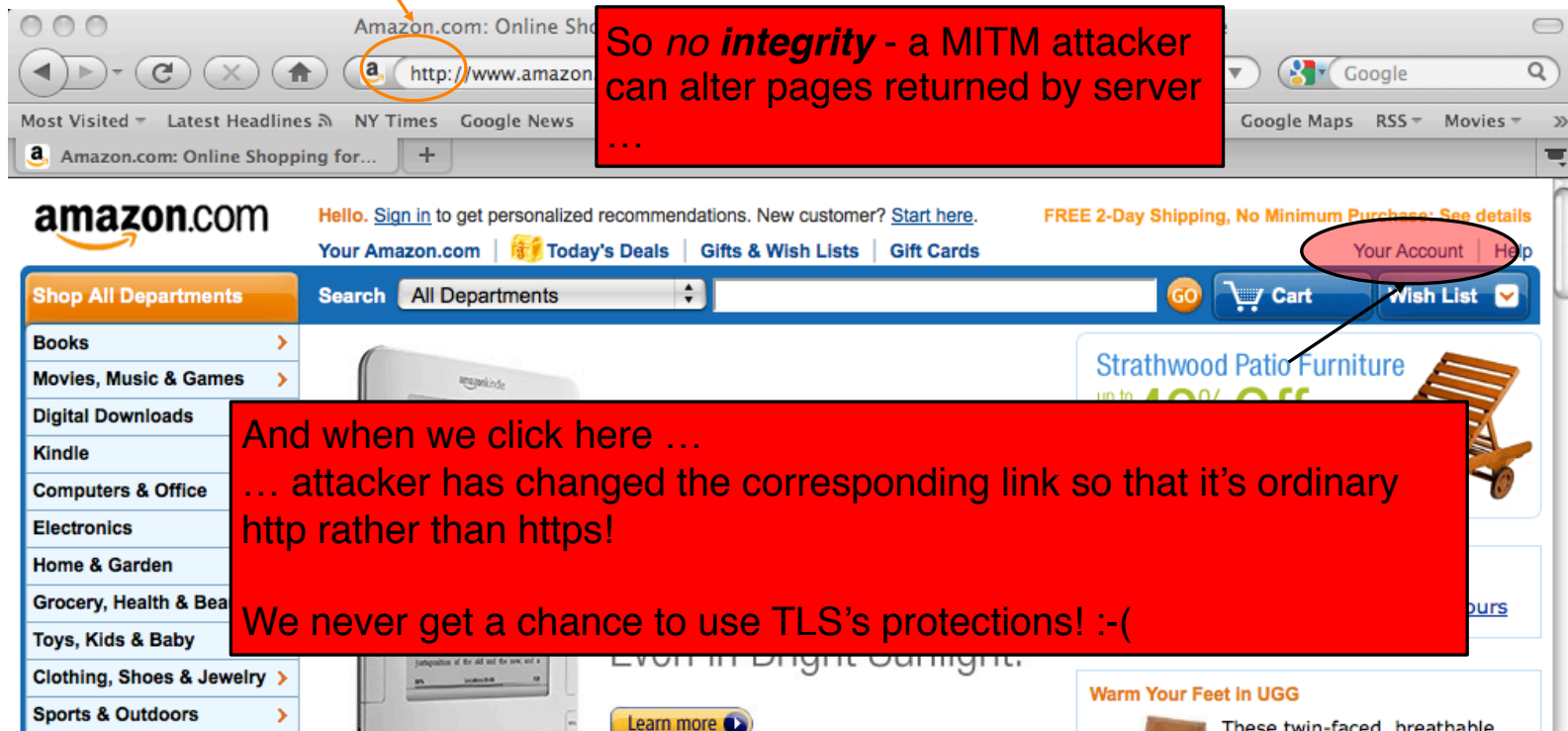
# SSL/TLS Problem: Revocation

- 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](https://en.wikipedia.org/wiki/Online_Certificate_Status_Protocol)

# “sslstrip”

## (Amazon FINALLY fixed this recently)

Regular web surfing: http: URL



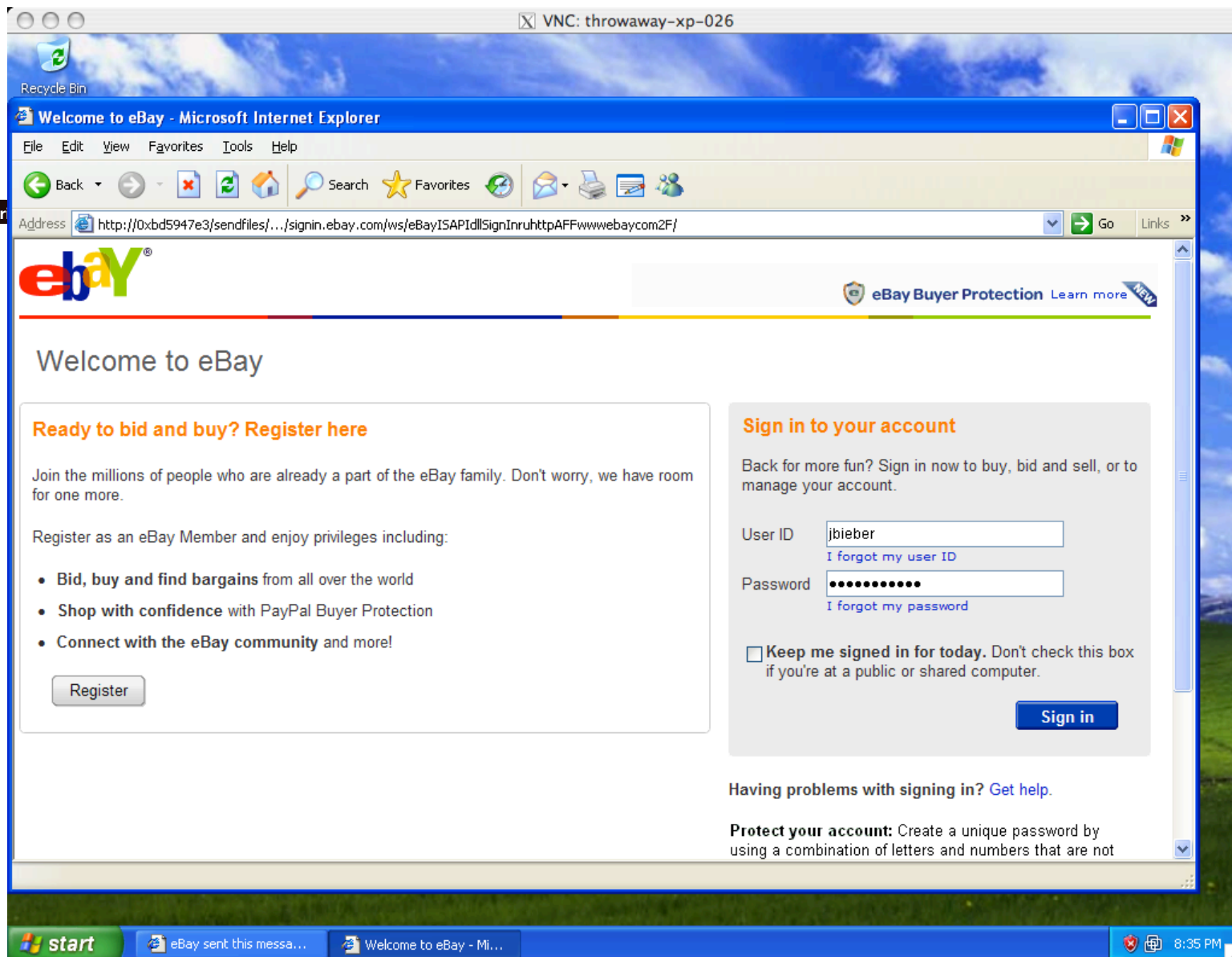


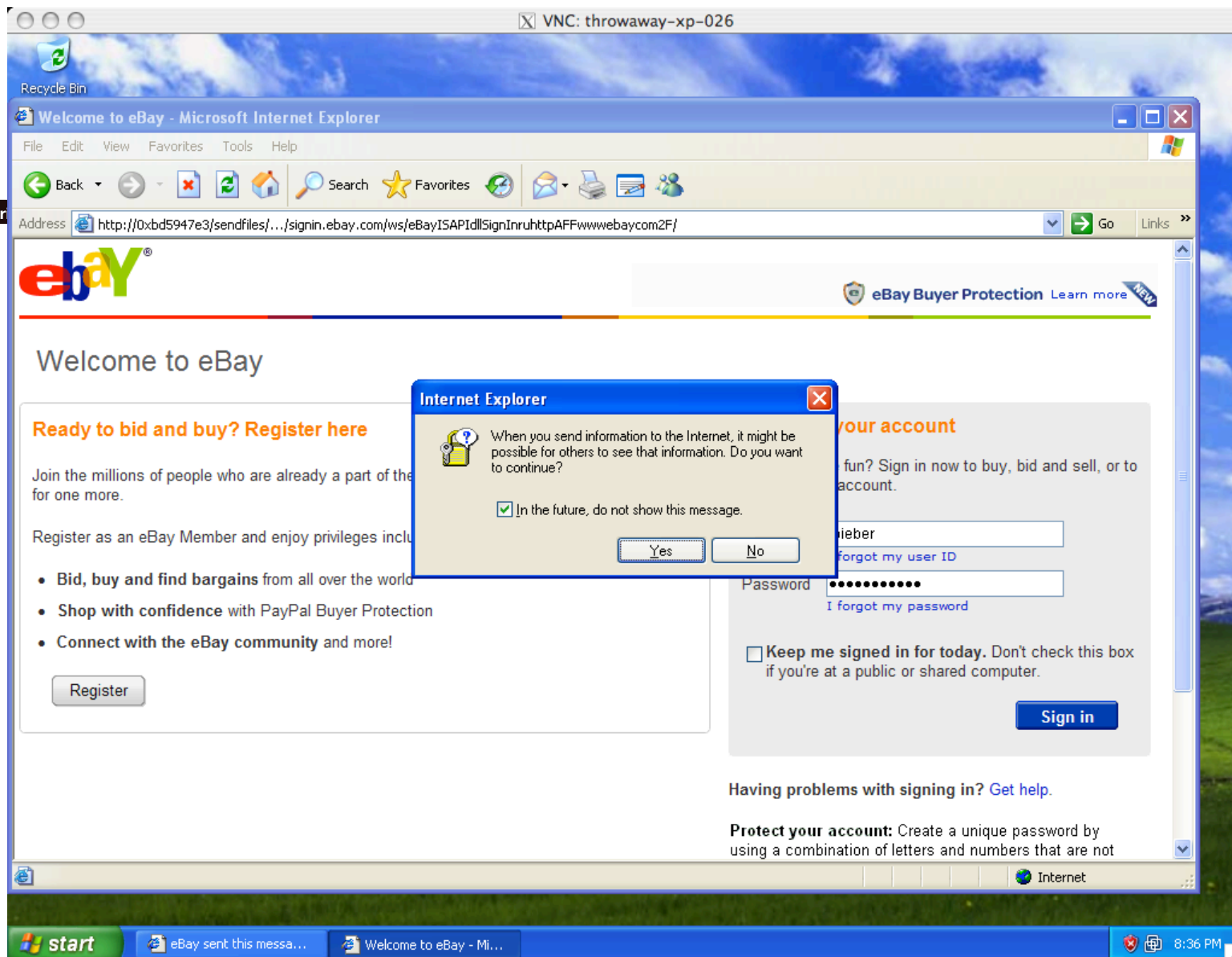
# SSL / TLS Limitations, cont.

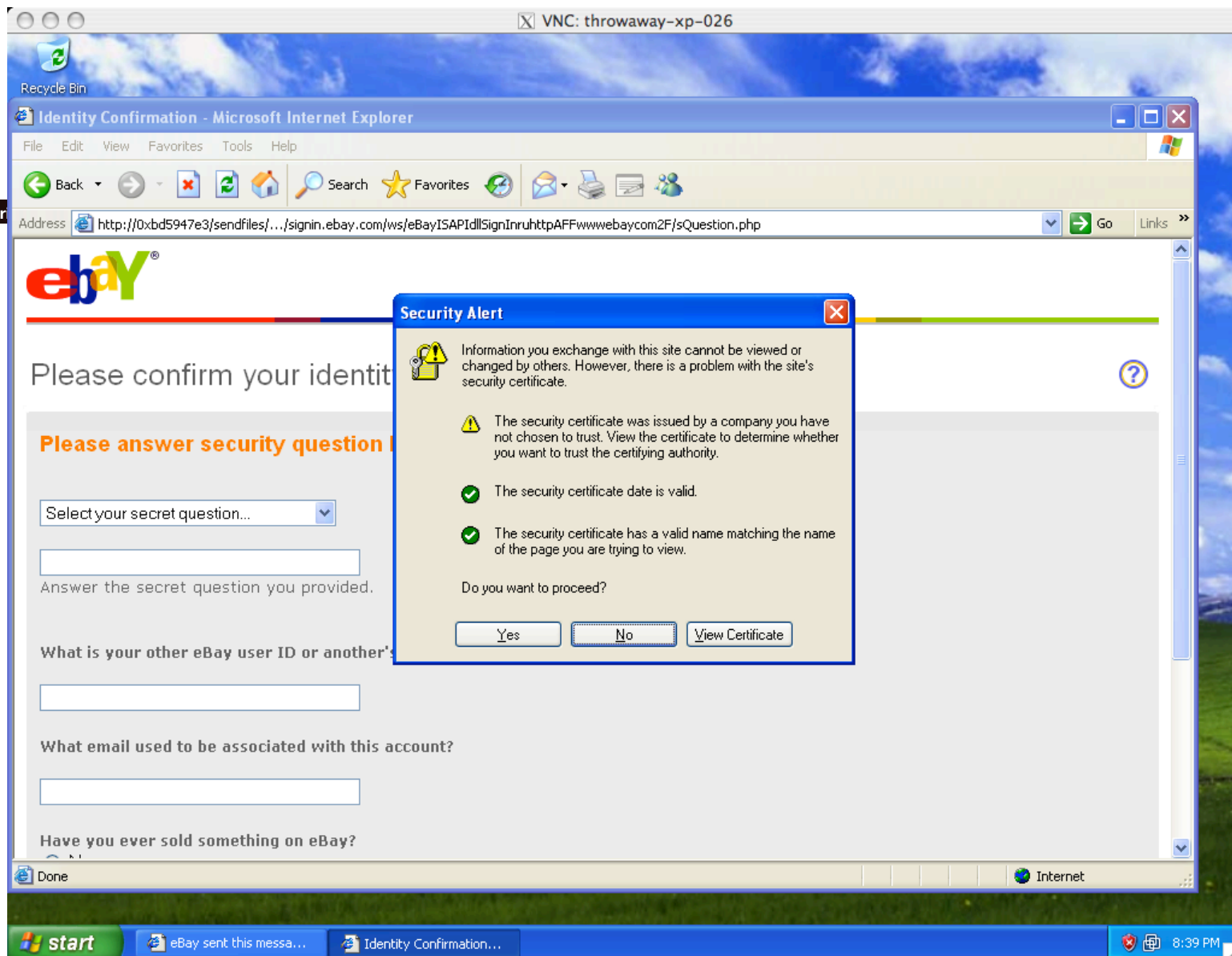
- Problems that SSL / TLS does not take care of ?
- Censorship
- SQL injection / XSS / server-side coding/logic flaws
- Vulnerabilities introduced by server inconsistencies
- Browser and server bugs
- Bad passwords
- What about the trust?

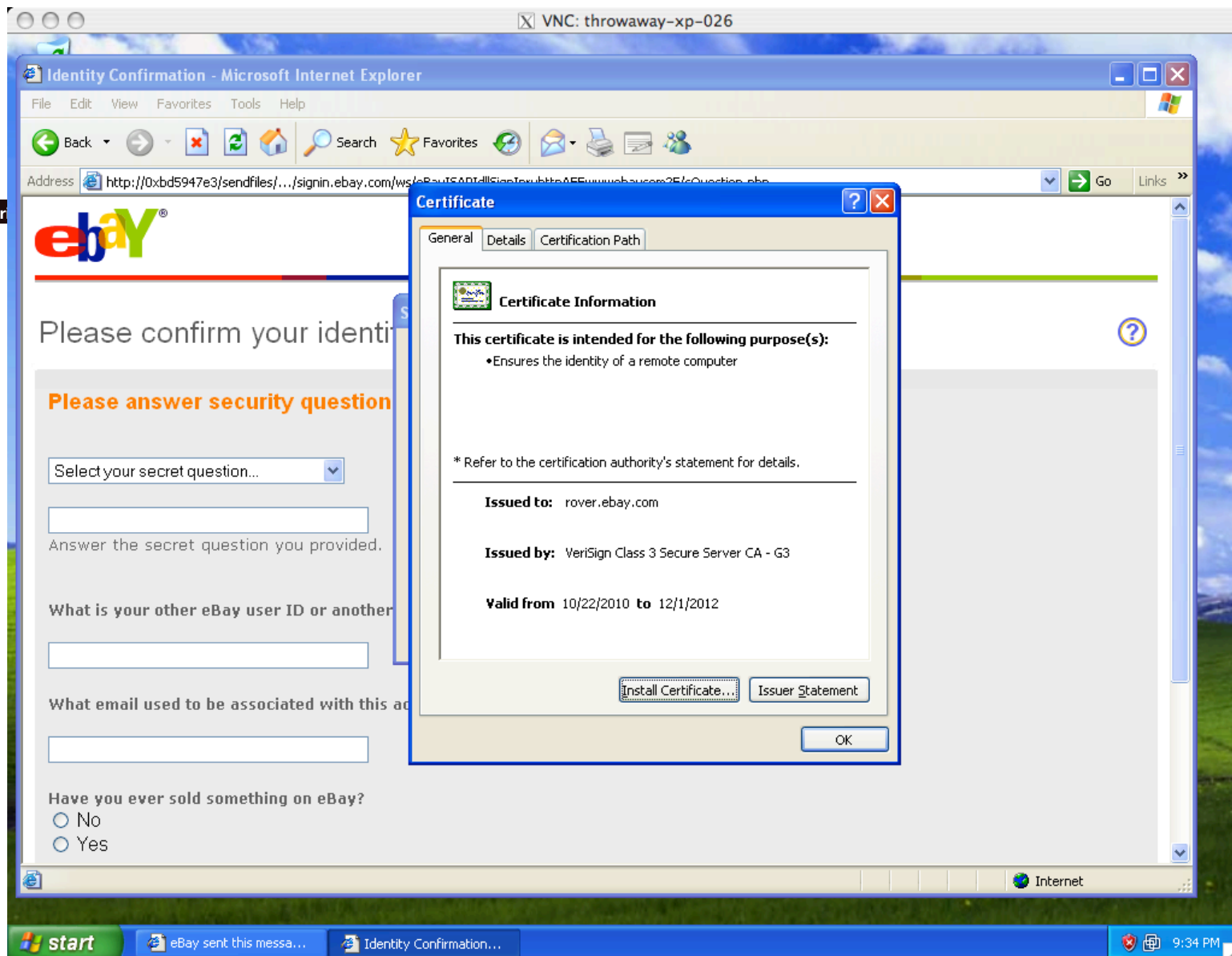
# TLS/SSL Trust Issues

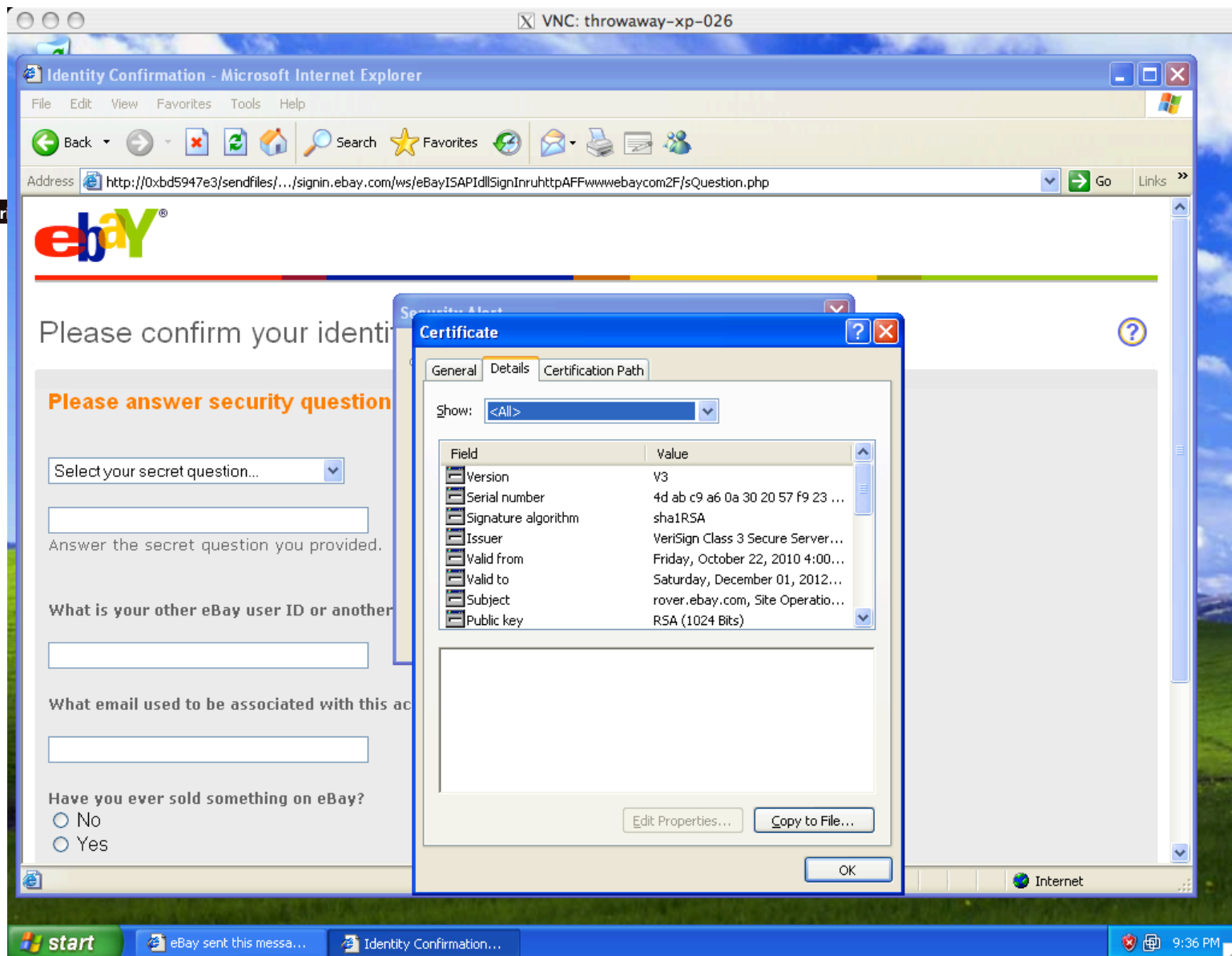
- User has to make correct trust decisions ...

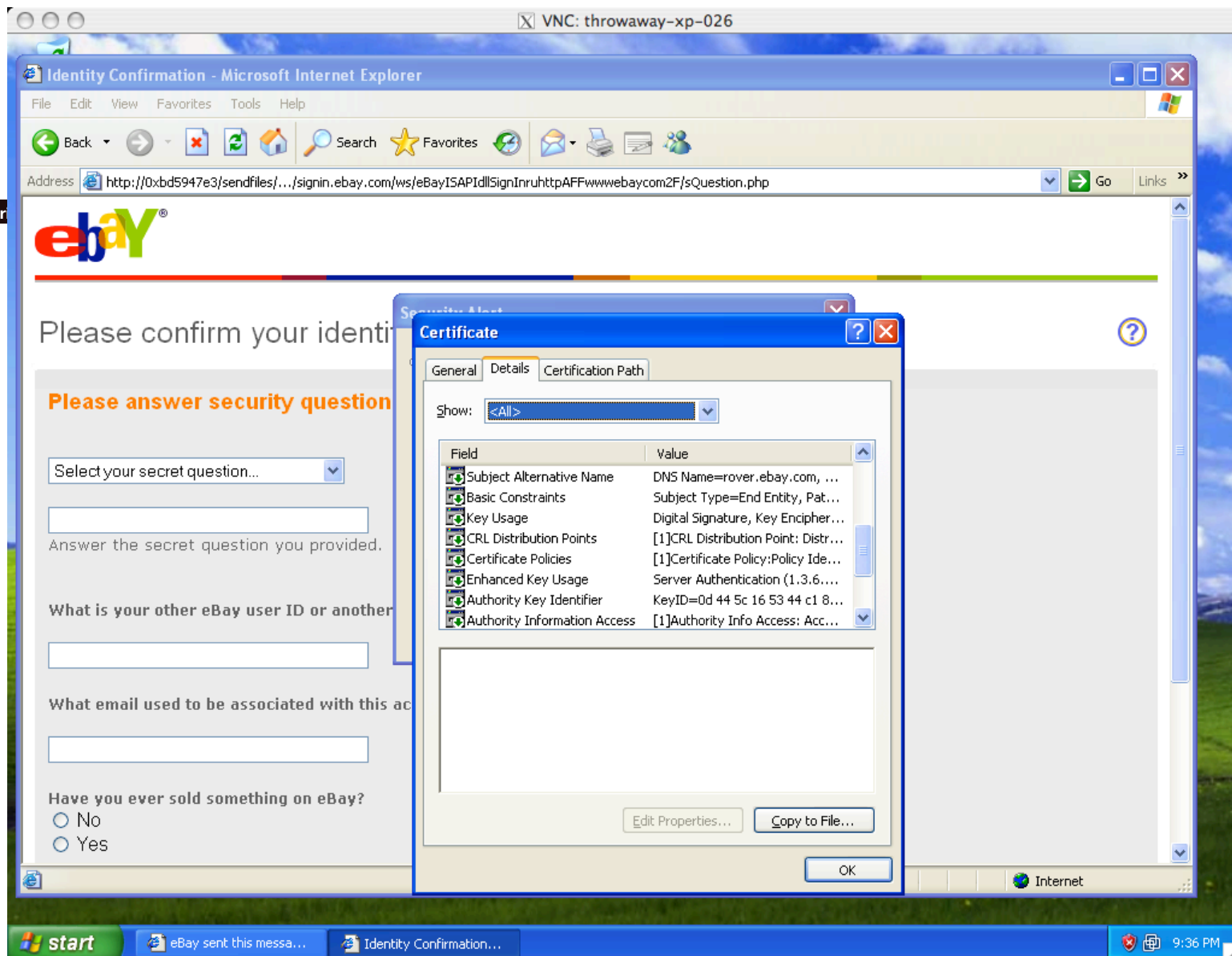




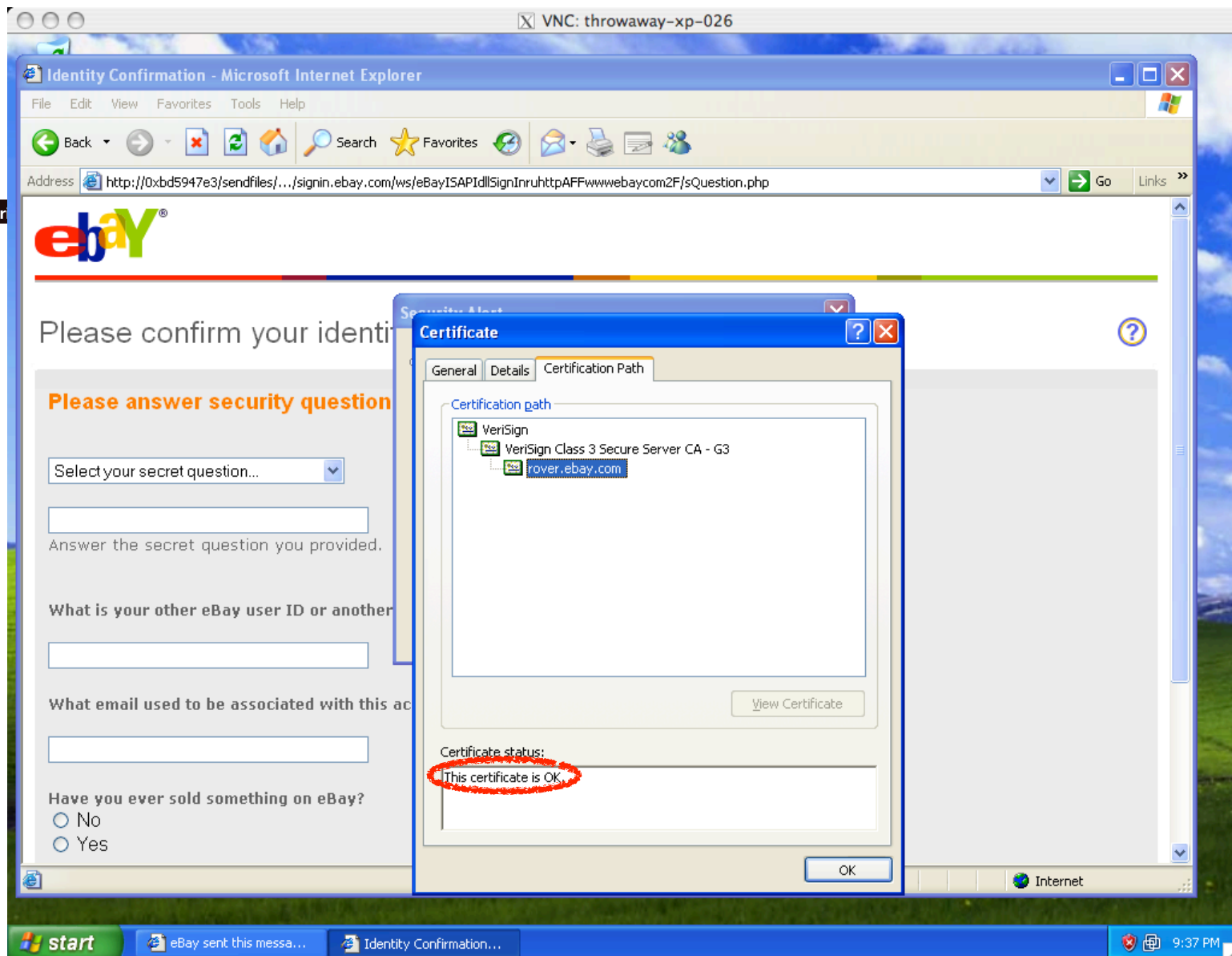




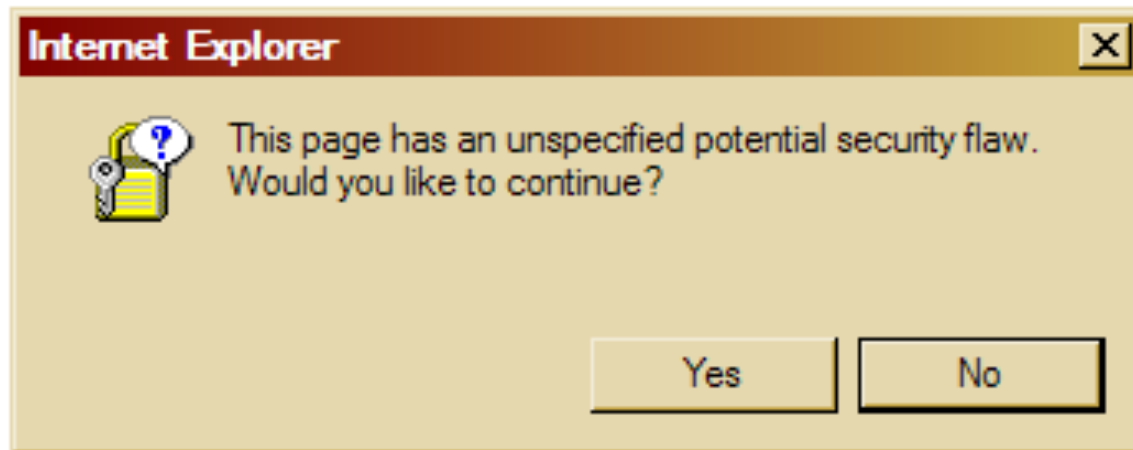








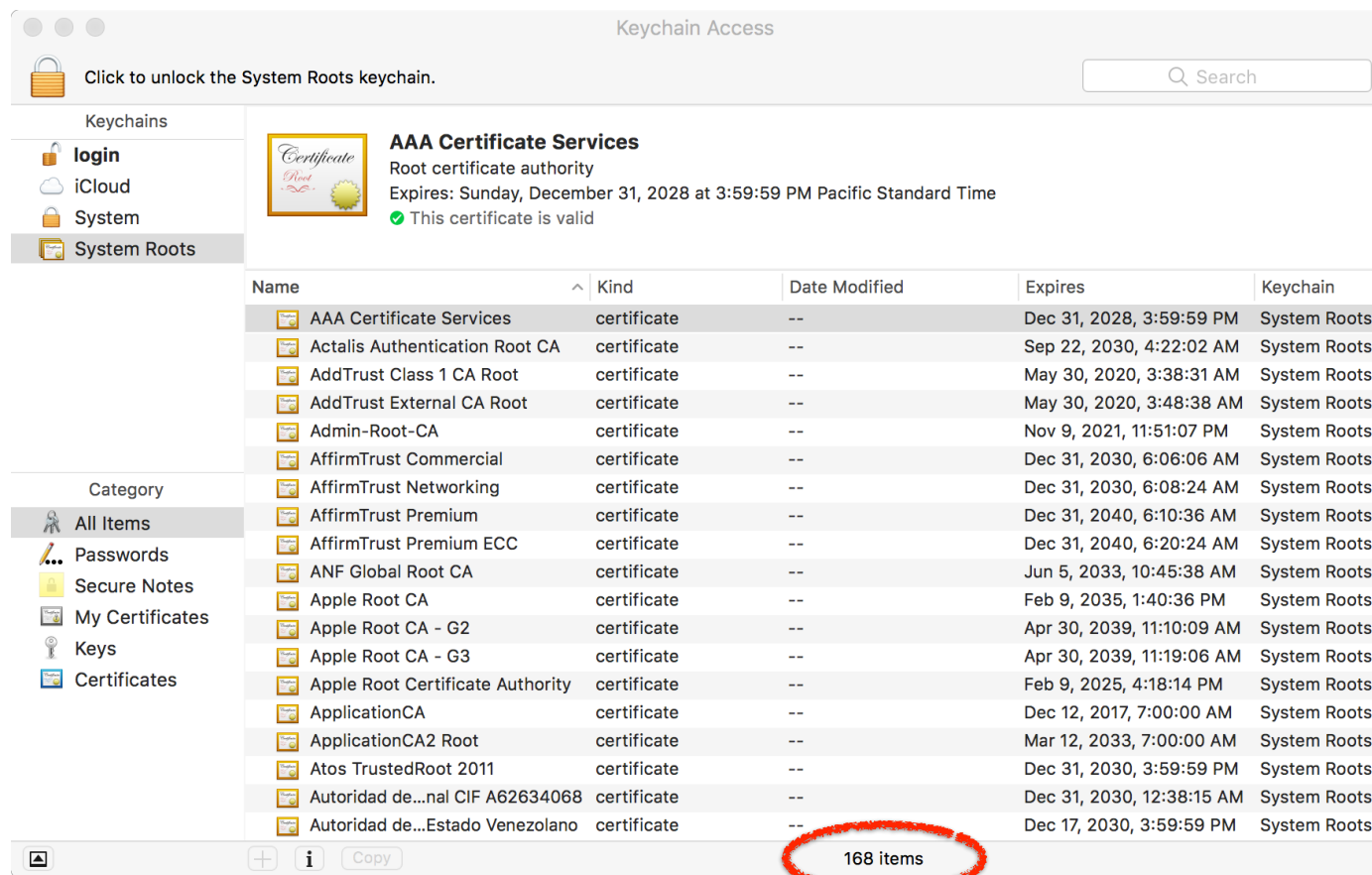
The equivalent as seen by most Internet users:



(note: an actual Windows error message!)

# TLS/SSL Trust Issues, cont.

- “*Commercial certificate authorities protect you from anyone from whom they are unwilling to take money.*”
  - Matt Blaze, circa 2001
- So how many CAs do we have to worry about, anyway?



# TLS/SSL Trust Issues

- “*Commercial certificate authorities protect you from anyone from whom they are unwilling to take money.*”
  - Matt Blaze, circa 2001
- So how many CAs do we have to worry about, anyway?
- Of course, it's not just their greed that matters ...

**News**

# Solo Iranian hacker takes credit for Comodo certificate attack

Security researchers split on whether 'ComodoHacker' is the real deal

By Gregg Keizer

March 27, 2011 08:39 PM ET

 [Comments \(5\)](#)

 [Recommended \(37\)](#)

 [Like](#) 84

---

Computerworld - A solo Iranian hacker on Saturday claimed responsibility for stealing multiple SSL certificates belonging to some of the Web's biggest sites, including Google, Microsoft, Skype and Yahoo.

Early reaction from security experts was mixed, with some believing the hacker's claim, while others were dubious.

# Fraudulent Google certificate points to Internet attack

& Weaver

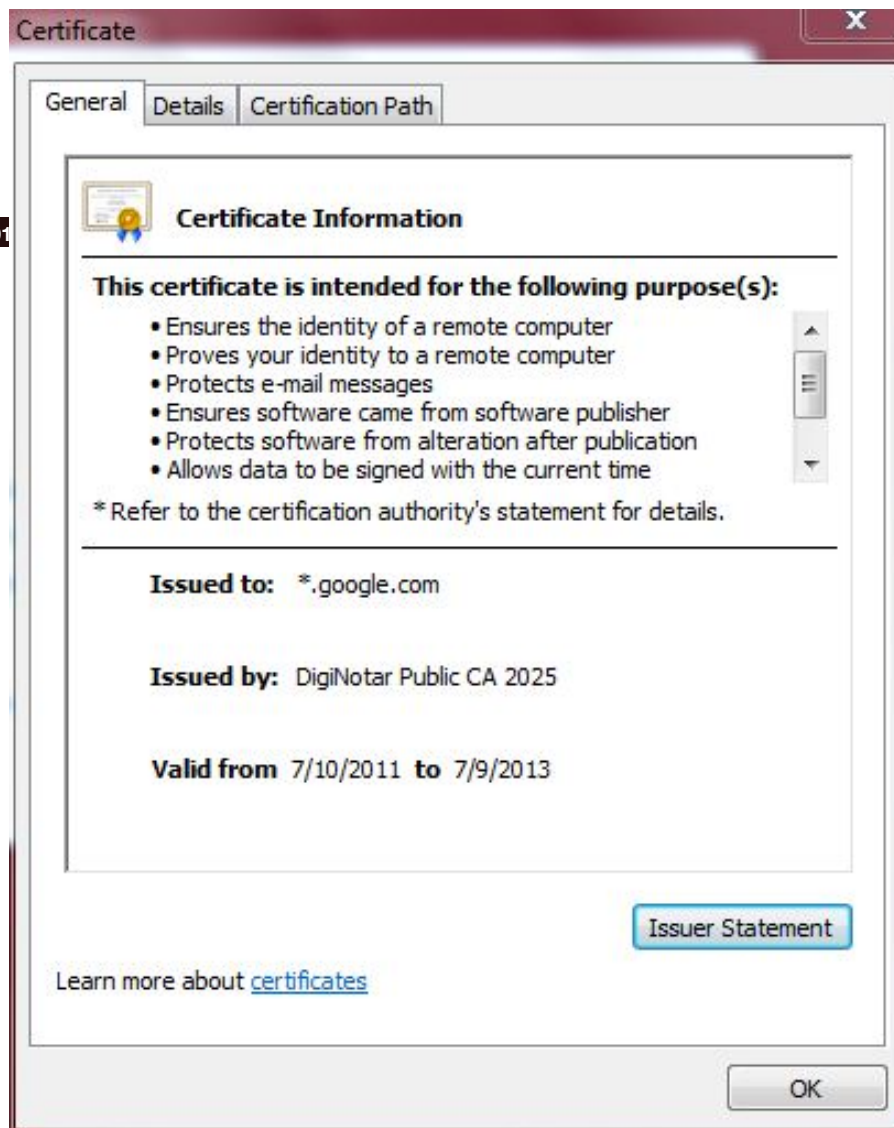
Is Iran behind a fraudulent Google.com digital certificate? The situation is similar to one that happened in March in which spoofed certificates were traced back to Iran.



by [Elinor Mills](#) | August 29, 2011 1:22 PM PDT

A Dutch company appears to have issued a digital certificate for Google.com to someone other than Google, who may be using it to try to re-direct traffic of users based in Iran.

Yesterday, someone reported on a Google support site that when attempting to log in to Gmail the browser issued a warning for the digital certificate used as proof that the site is legitimate, according to [this thread](#) on a Google support forum site.



**This appears to be a fully valid cert using normal browser validation rules.**

**Only detected by Chrome due to its introduction of cert “pinning” – requiring that certs for certain domains must be signed by specific CAs rather than any generally trusted CA**



October 31, 2012, 10:49AM

## Final Report on DigiNotar Hack Shows Total Compromise of CA Servers

The attacker who penetrated the Dutch CA DigiNotar last year had complete control of all eight of the company's certificate-issuing servers during the operation and he may also have issued some rogue certificates that have not yet been identified. The final report from a

### Evidence Suggests DigiNotar, Who Issued Fraudulent Google Certificate, Was Hacked *Years Ago*

from the *diginot* dept

The big news in the security world, obviously, is the fact that a **fraudulent Google certificate made its way out into the wild**, apparently targeting internet users in Iran. The Dutch company DigiNotar has put out a statement saying that **it discovered a breach** back on July 19th during a security audit, and that fraudulent certificates were generated for "several dozen" websites. The only one known to have gotten out into the wild is the Google one.

# The DigiNotar Fallout

- The result was the “CA Death Sentence”:
  - Web browsers removed it from the trusted root certificate store
- This happened again with “WoSign”
  - A Chinese CA
- WoSign would allow an interesting attack
  - If I controlled nweaver.github.com...
  - WoSign would allow me to create a certificate for `*.github.com!?!?`
  - And a bunch of other shady shenanigans

# TLS/SSL Trust Issues

- “Commercial certificate authorities protect you from anyone from whom they are unwilling to take money.”
  - Matt Blaze, circa 2001
- So how many CAs do we have to worry about, anyway?
- Of course, it’s not just their greed that matters ...
- ... and it’s not just their diligence & security that matters ...
  - *“A decade ago, I observed that commercial certificate authorities protect you from anyone from whom they are unwilling to take money. That turns out to be wrong; they don't even do that much.”* - Matt Blaze, circa 2010

# So the Modern Solution: Invoke Ronald Reagan, “Trust, but Verify”

- Static Certificate Pinning:  
The chrome browser has a list of certificates or certificate authorities that it trusts for given sites
  - Now creating a fake certificate requires attacking a *particular* CA
- HPKP Certificate Pinning:  
The web server provides hashes of certificates that should be trusted
  - This is “Leap of Faith”: The first time you assume it is honest but you will catch future changes
- Transparency mechanisms:
  - Public logs provided by certificate authorities
  - Browser extensions (EFF’s TLS observatory)
  - Backbone monitors (ICSI’s TLS notary)

# And Making It Cheap: LetsEncrypt...

- Coupled to the depreciation of unencrypted HTTP...
  - Need to be able to have HTTPS be just about the same complexity...
- Idea: Make it easy to "prove" you own a web site:
  - Can you write an arbitrary cookie at an arbitrary location?
- Build ***automated*** infrastructure to do this
  - Script to create a private key
  - Generate a certificate signing request
  - PKI authority says "here's a file, put it on the server"
  - Script puts it on the server