CIS 6930:
IoT Security

Lecture 5

Prof. Kaushal Kafle

Spring 2025

Derived from slides by Adwait Nadkarni, William Enck, Micah Sherr and Patrick McDaniel

Class Notes and Clarifications

- General comments on paper reviews
  - Pay attention to the feedback!
  - The grading metric also looks at your improvements, so not repeating mistakes is key!
  - Small problems in implementation vs larger limitations/drawbacks of the methodology/framework
  - *Your comments should match the accept/reject verdict!*
- General comment on discussions
  - I expect you to have *at least some thoughts* on the research topics we discuss!
  - The point is to express opinions and engage.

Class Notes and Clarifications

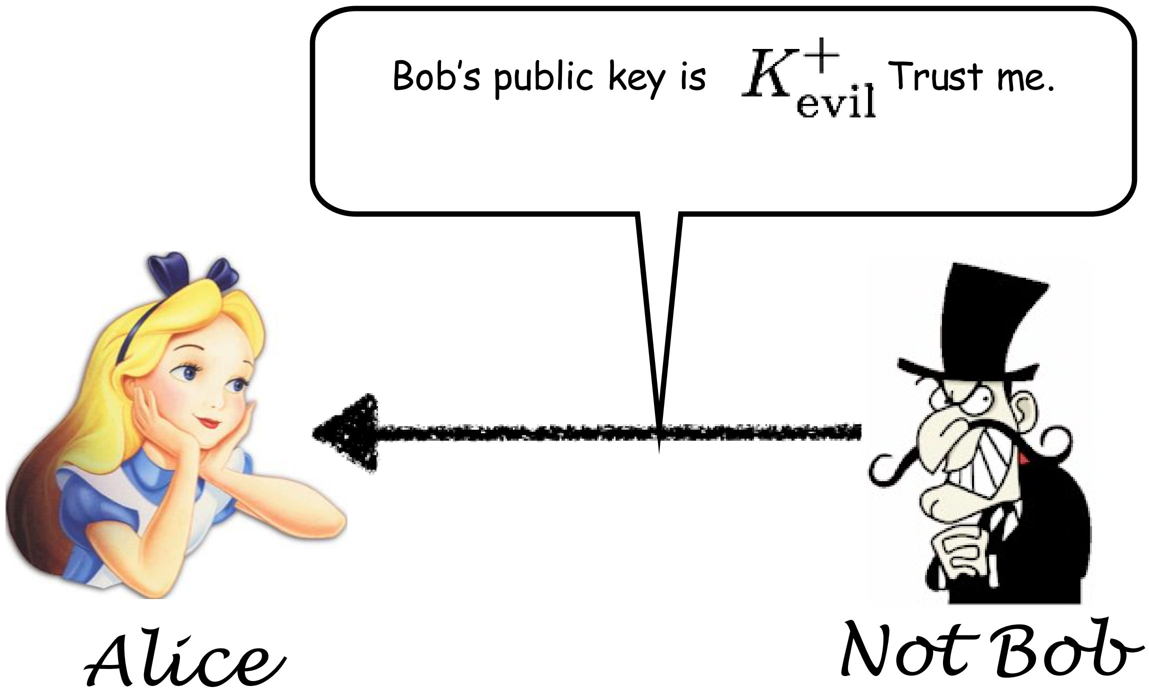
- General comments on presentations
  - Deeply understand and relay the ideas of the paper that you are presenting
    - If you don't understand something yourself, you won't be able to explain it to us.
  - Understand ***why*** the authors are choosing to do things a certain way.
    - Since you are presenting, we'll assume you are the author!
  - *Remember that the presentation is a visual medium!*

Public Key Crypto
(10,000 ft view)

- Separate keys for encryption and decryption
  - Public key: anyone can know this
  - Private key: kept confidential
- Anyone can encrypt a message to you using your public key
- The private key (kept confidential) is required to decrypt the communication
- Alice and Bob no longer have to have *a priori* shared a secret key

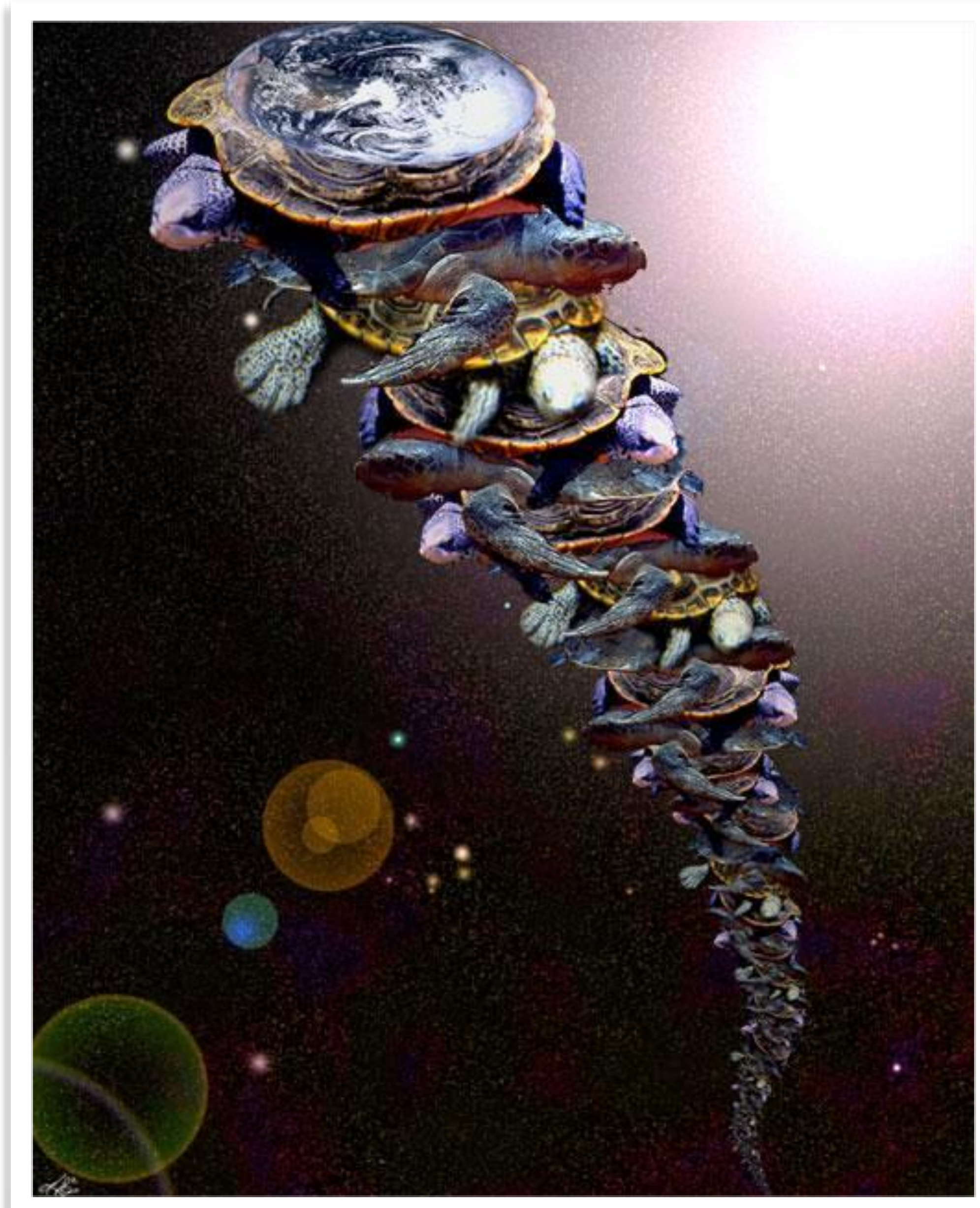
Problem? YES. *How do we know if Alice's key is really Alice's?*

But how do we *verify* we're using the correct public key?



Short answer: We can't.

It's turtles all the way down.

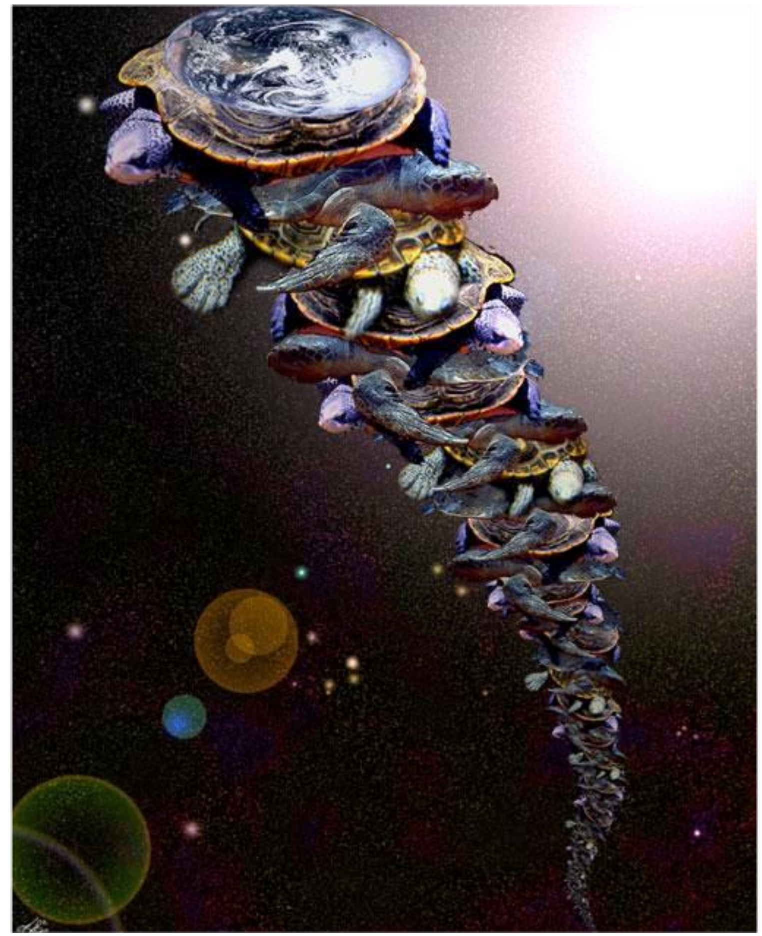


Why not just use a database?

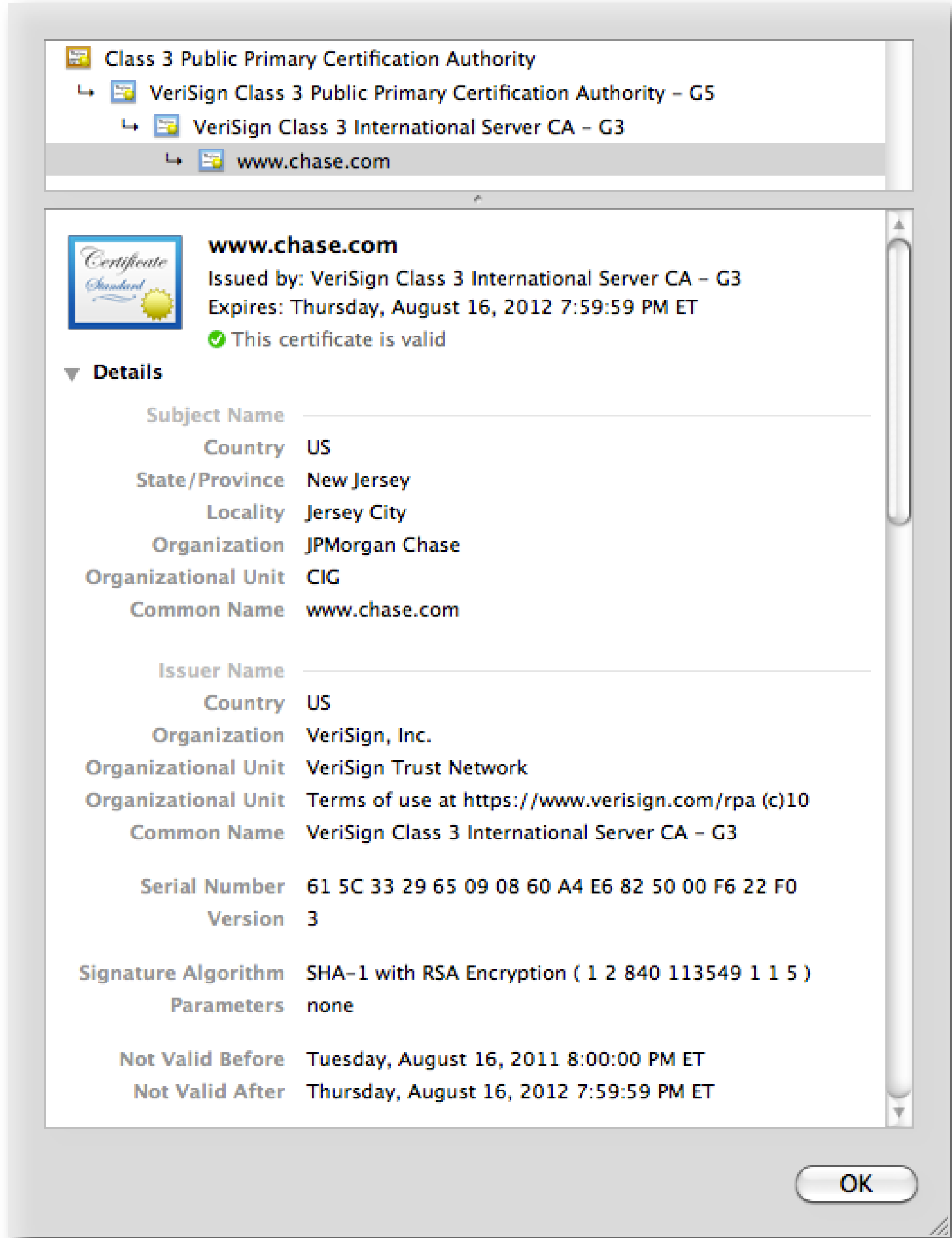
- Every user has his/her own public key and private key.
- Public keys are all published in a database.
- Alice gets Bob's public key from the database
- Alice encrypts the message and sends it to Bob using Bob's public key.
- Bob decrypts it using his private key.
- **What's the problem with this approach?**

Solving the Turtles Problem

- We need **a trust anchor**
  - there must be someone with authority
  - requires *a priori* trust
- Solution: form a trust hierarchy
  - "I believe **X** because…"
  - "**Y** vouches for **X** and…"
  - "**Z** vouches for **Y** and…"
  - "I implicitly trust **Z**."



Browser Certificate

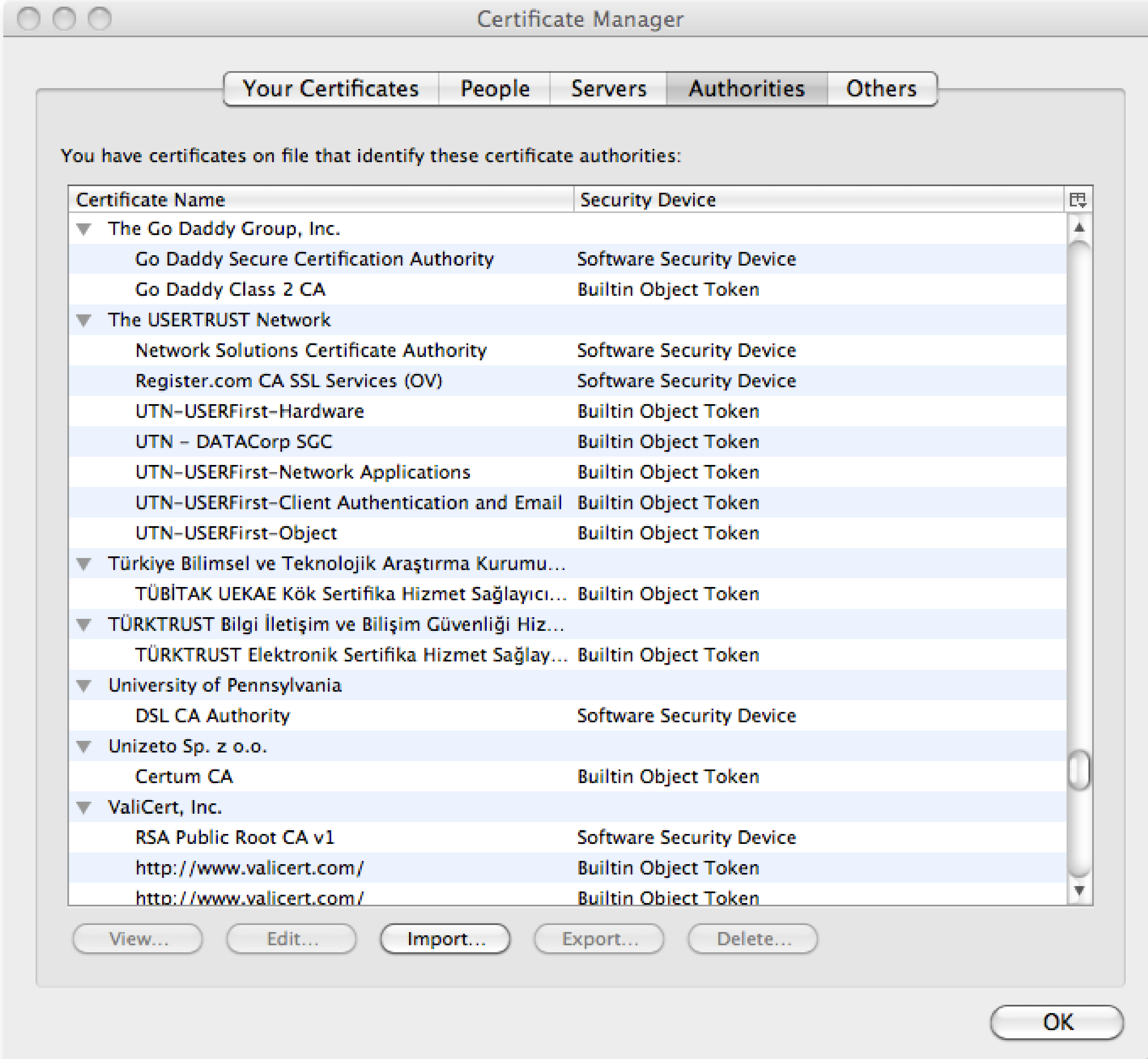


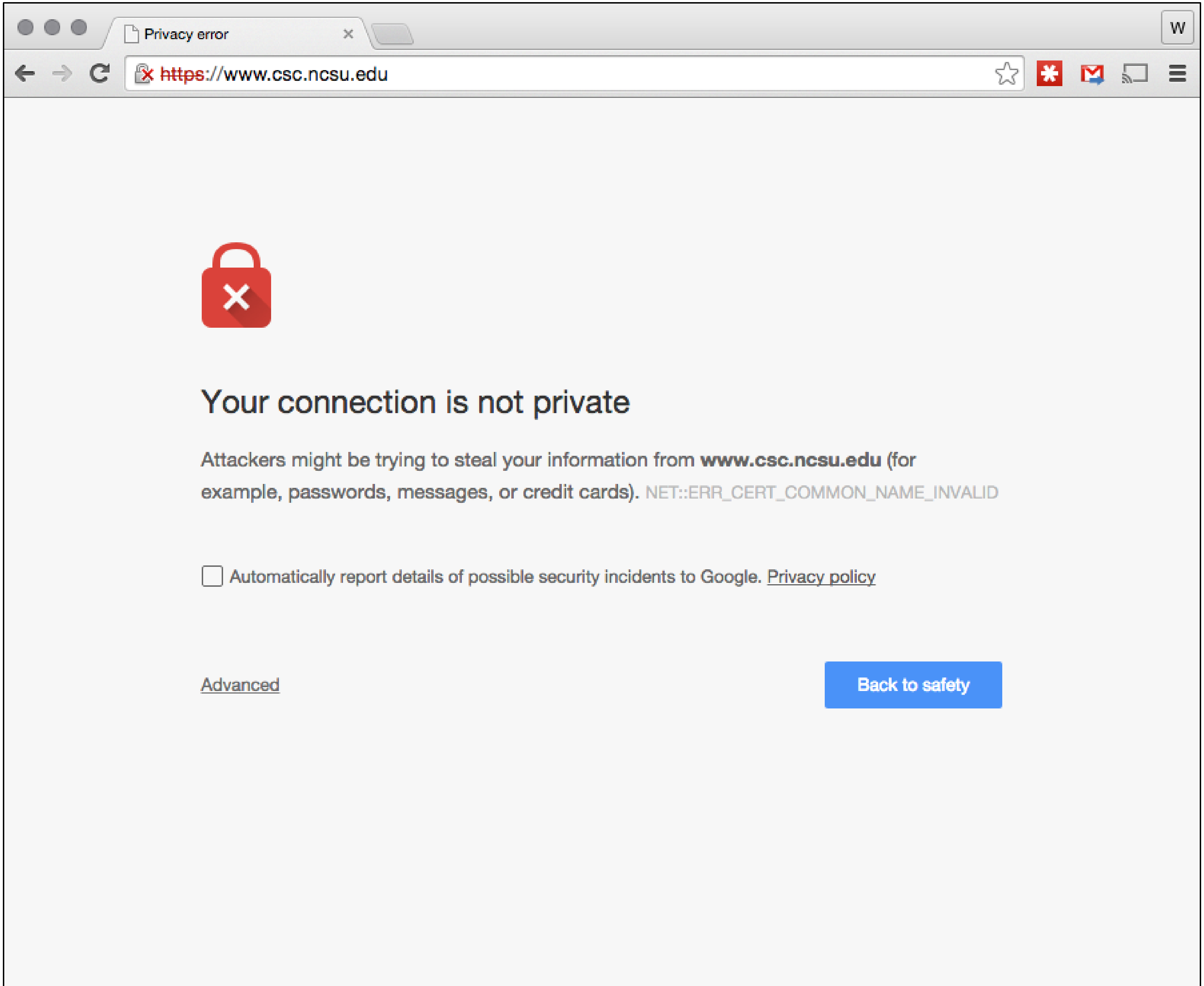
What’s a certificate?

- A certificate …
  - … **makes an association between an identity and a private key**
  - … contains public key information {e,n}
  - … has a validity period
  - … is signed by some *certificate authority* (CA)
  - … identity may have been vetted by a *registration authority* (RA)
- People trust CA (e.g., Verisign) to vet identity

Why do I trust the certificate?

- A collections of *“root” CA certificates (self-signed)*
  - … baked into your browser
  - … vetted by the browser manufacturer
  - … <u>supposedly</u> closely guarded
  - *trust anchor*
- Root certificates used to validate certificate
  - Vouches for certificate’s authenticity


Certificate Manager
Your Certificates
People
Servers
Authorities
Others
You have certificates on file that identify these certificate authorities:
Certificate Name
Security Device
The Go Daddy Group, Inc.
Go Daddy Secure Certification Authority
Software Security Device
Go Daddy Class 2 CA
Builtin Object Token
The USERTRUST Network
Network Solutions Certificate Authority
Software Security Device
Register.com CA SSL Services (OV)
Software Security Device
UTN-USERFirst-Hardware
Builtin Object Token
UTN - DATACorp SGC
Builtin Object Token
UTN-USERFirst-Network Applications
Builtin Object Token
UTN-USERFirst-Client Authentication and Email
Builtin Object Token
UTN-USERFirst-Object
Builtin Object Token
Türkiye Bilimsel ve Teknolojik Araştırma Kurumu...
TÜBİTAK UEKAE Kök Sertifika Hizmet Sağlayıcı...
Builtin Object Token
TÜRKTRUST Bilgi İletişim ve Bilişim Güvenliği Hiz...
TÜRKTRUST Elektronik Sertifika Hizmet Sağlay...
Builtin Object Token
University of Pennsylvania
DSL CA Authority
Software Security Device
Unizeto Sp. z o.o.
Certum CA
Builtin Object Token
ValiCert, Inc.
RSA Public Root CA v1
Software Security Device
http://www.valicert.com/
Builtin Object Token
http://www.valicert.com/
Builtin Object Token
View...
Edit...
Import...
Export...
Delete...
OK

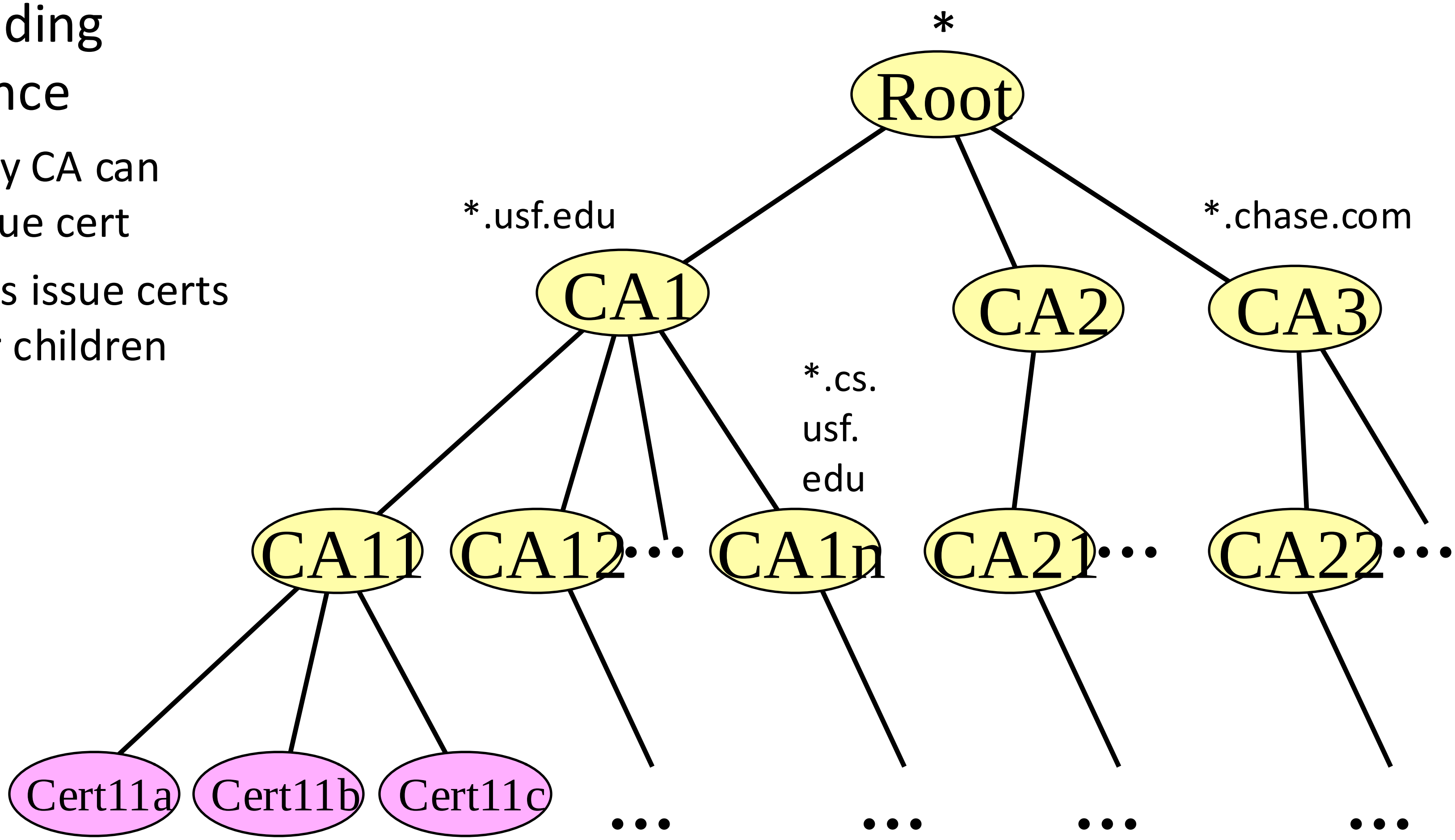

Privacy error
https://www.csc.ncsu.edu
W
Your connection is not private
Attackers might be trying to steal your information from www.csc.ncsu.edu (for example, passwords, messages, or credit cards). NET::ERR_CERT_COMMON_NAME_INVALID
Automatically report details of possible security incidents to Google. Privacy policy
Advanced
Back to safety

Public Key Infrastructure

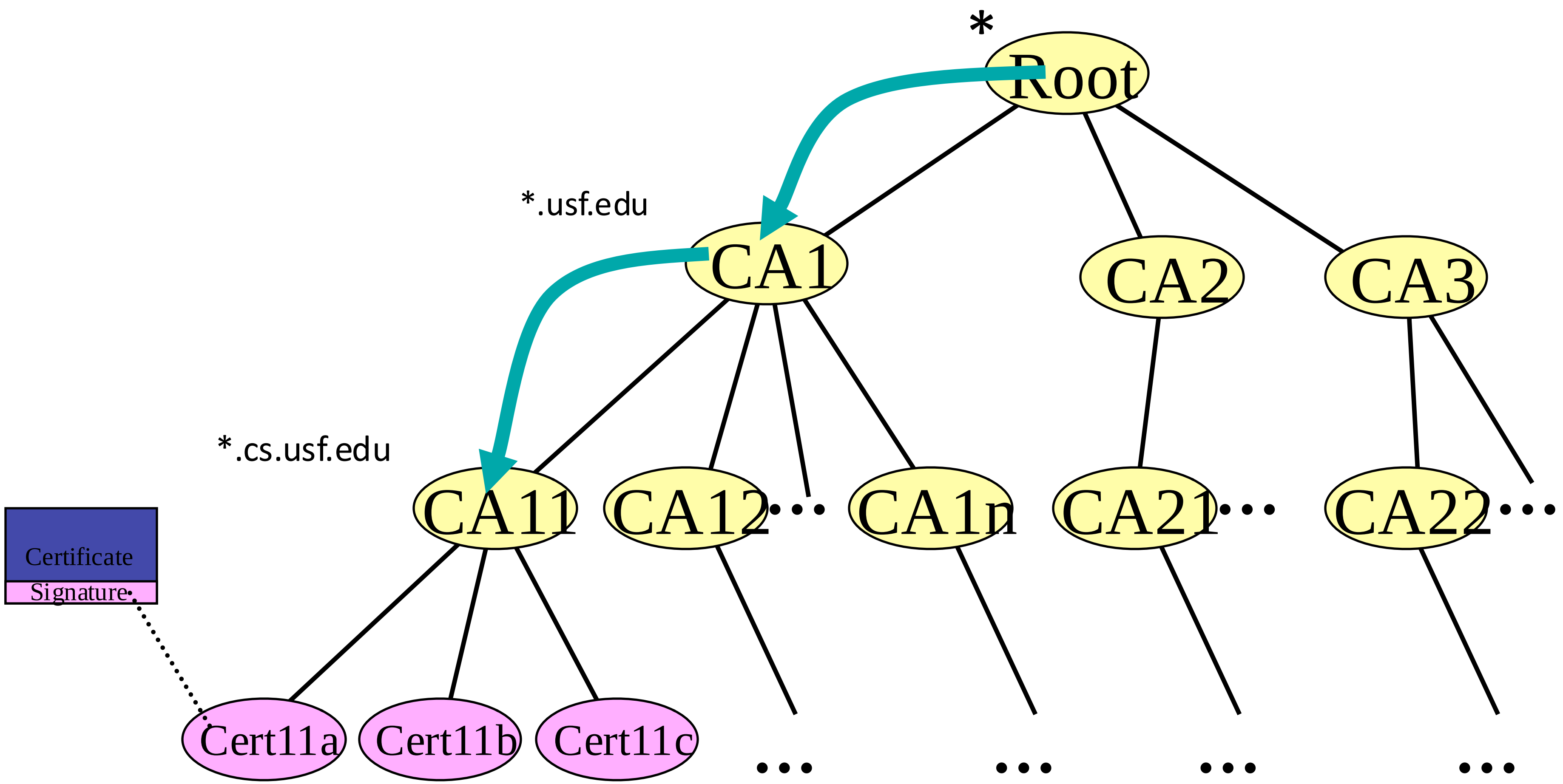
- Hierarchy of keys used to authenticate certificates
- Requires a **root of trust** (i.e., a **trust anchor**)

What is a PKI?

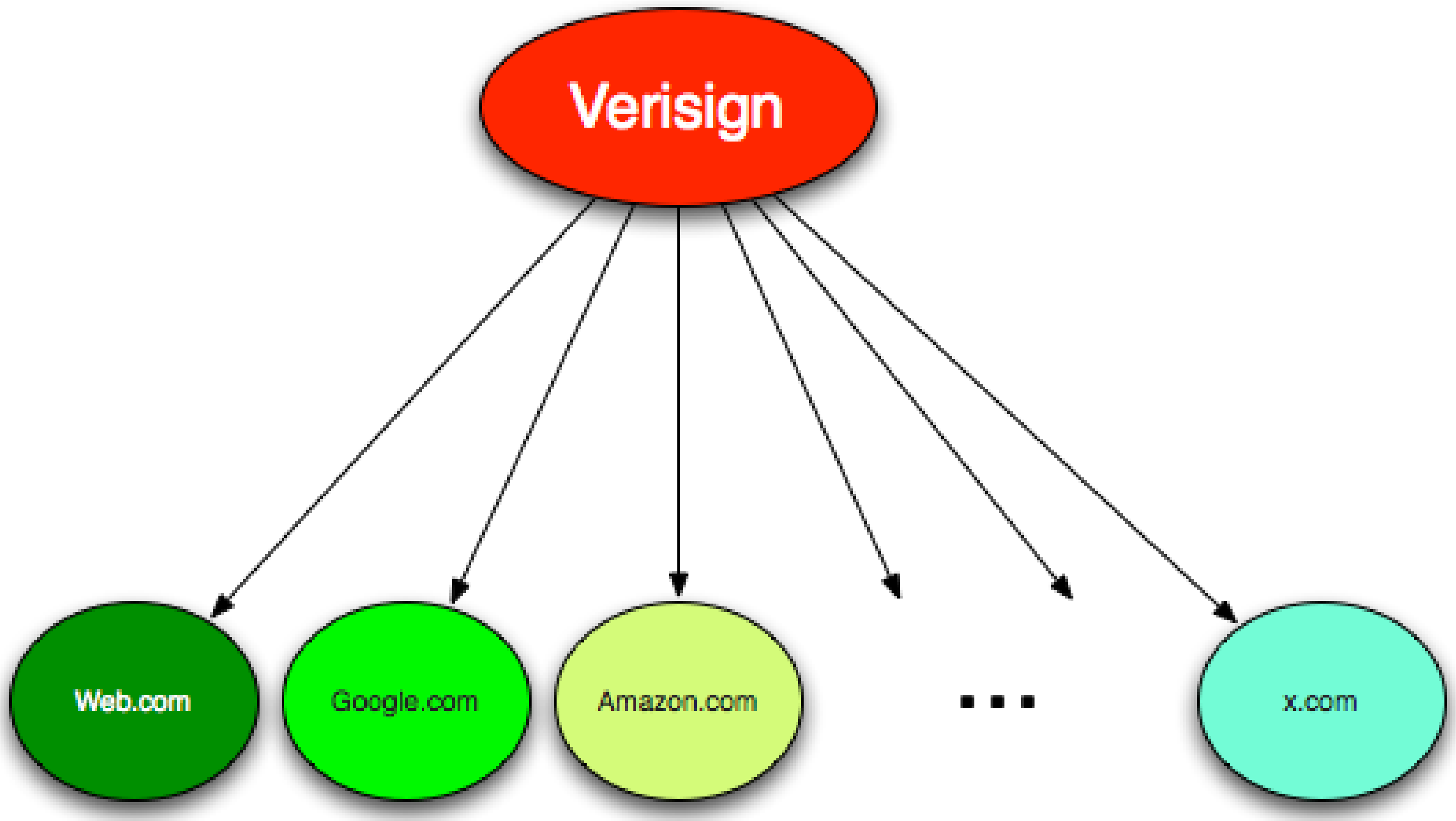
- Rooted tree of CAs
- Cascading issuance
  - Any CA can issue cert
  - CAs issue certs for children



Certificate Validation



PKIs in Reality

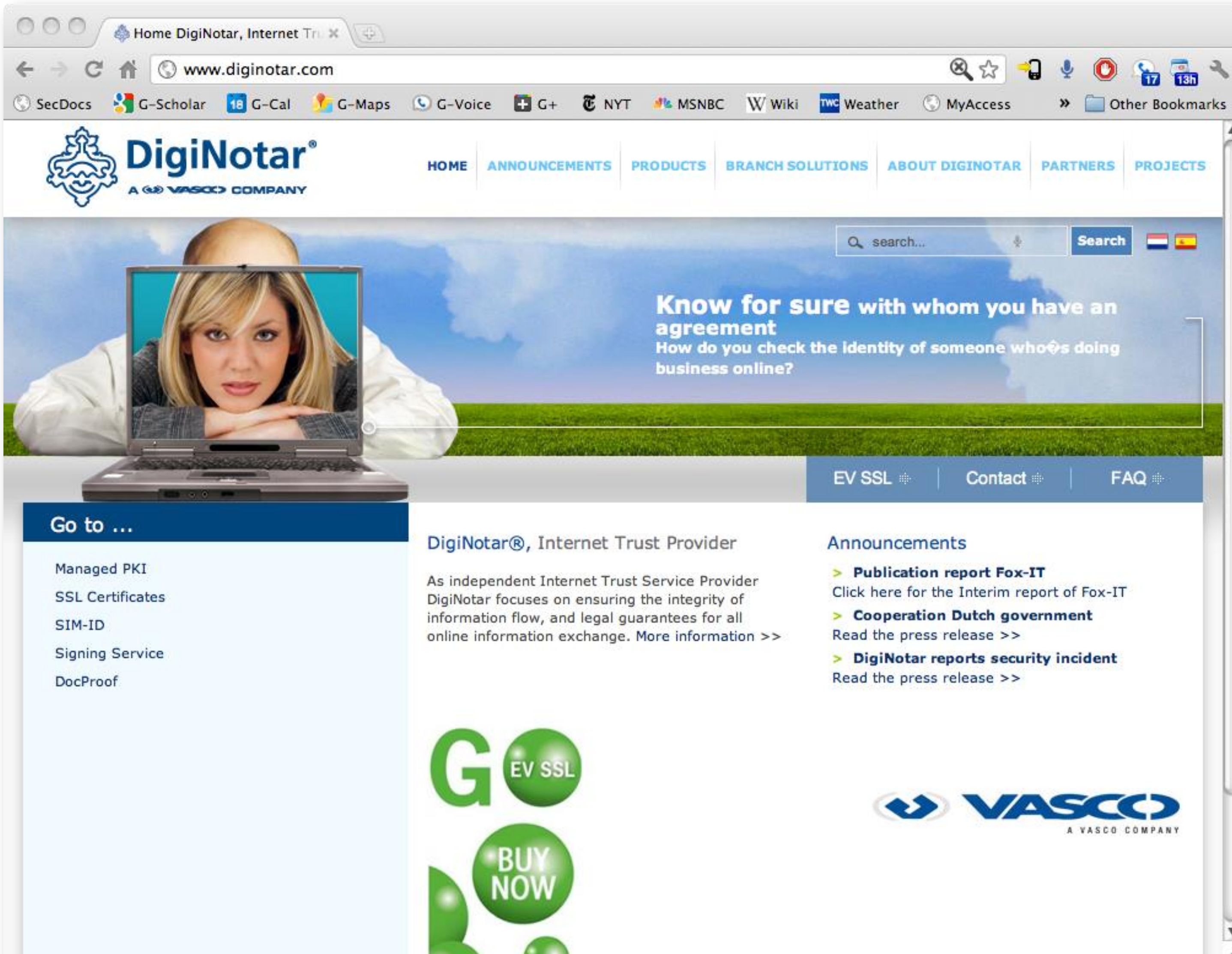


Obtaining a Certificate

1. Alice has some identity document A^{ID} and generates a keypair (A^-, A^+)
2. $A \rightarrow CA$: $\{A^+, A^{ID}\}$, Sig(A^-, $\{A^+, A^{ID}\}$)
   - CA verifies signature -- proves Alice has A^-
   - CA may (and should!) also verify A^{ID} offline
3. CA signs $\{A^+, A^{ID}\}$ with its private key (CA^-)
   - CA attests to binding between A+ and A^{ID}
4. $CA \rightarrow A$: $\{A^+, A^{ID}\}$, Sig(CA^-, $\{A^+, A^{ID}\}$)
   - this is the certificate; Alice can freely publish it
   - anyone who knows CA^+ (and can therefore validate the CA's signature) knows that CA "attested to" $\{A^+, A^{ID}\}$
   - note that CA never learns A^-

- Any CA may sign any certificate
- Browser weighs all root CAs equally
- *Q: Is this problematic?*

The DigiNotar Incident

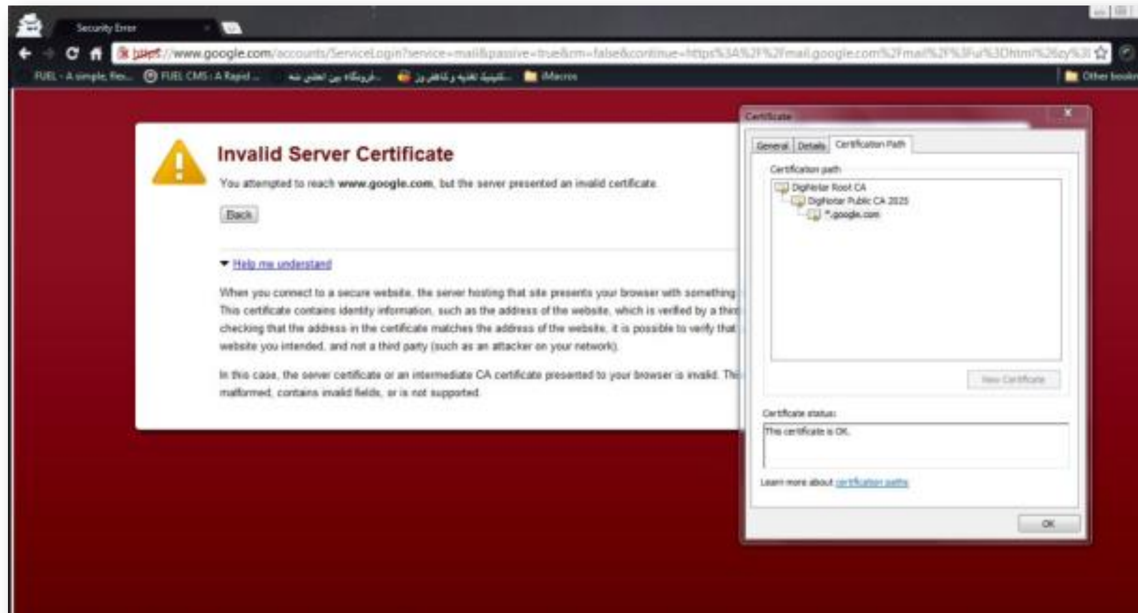


DigiNotar Incident

- DigiNotar is a CA based in the Netherlands that is (well, was) trusted by most OSes and browsers
- July 2011: Issued fake certificate for gmail.com to site in Iran that ran MitM attack...
- ... this fooled most browsers, but...

DigiNotar Incident

- As added security measure, Google Chrome hardcodes fingerprint of Google's certificate
- Since DigiNotar didn't issue Google's true certificate, this caused an error message in Chrome

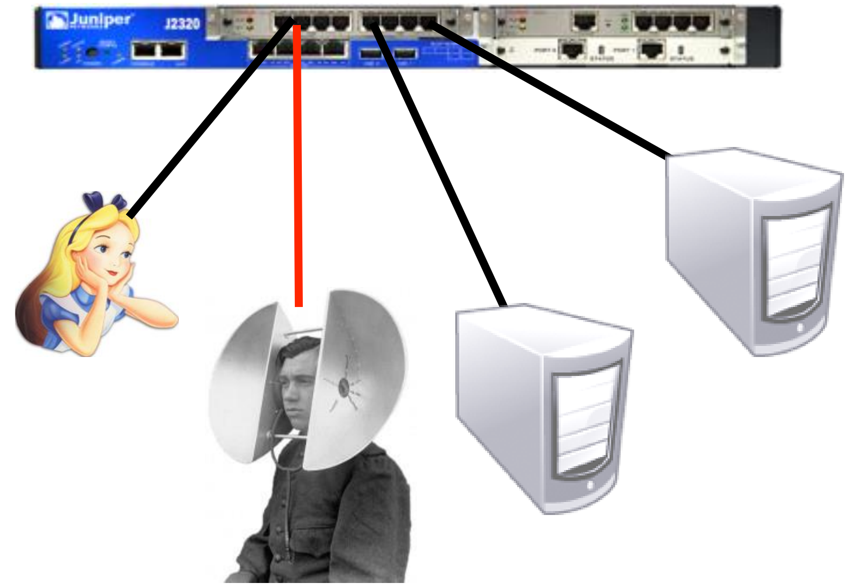


How secure is the verifier?

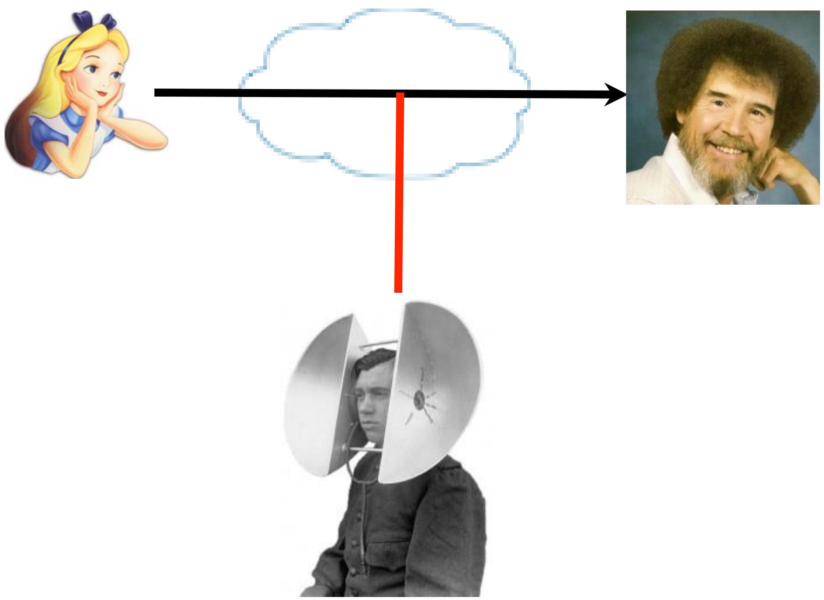
- What happens if attacker is able to insert his public root CA key to the verifier's list of trusted CAs?
- More generally, what are the consequences if the verifier is compromised?
- Q: What's the consequences for IoT devices/apps?

Eavesdropping

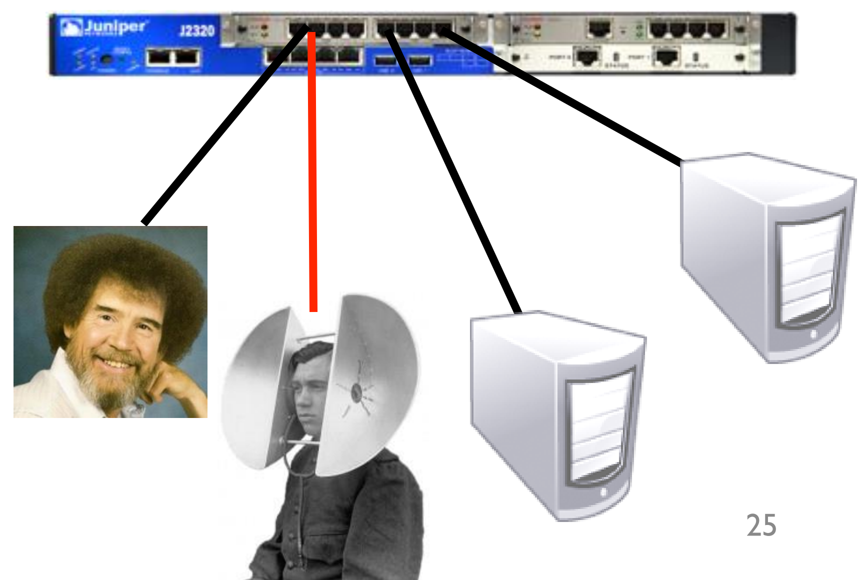
Alice's Switch



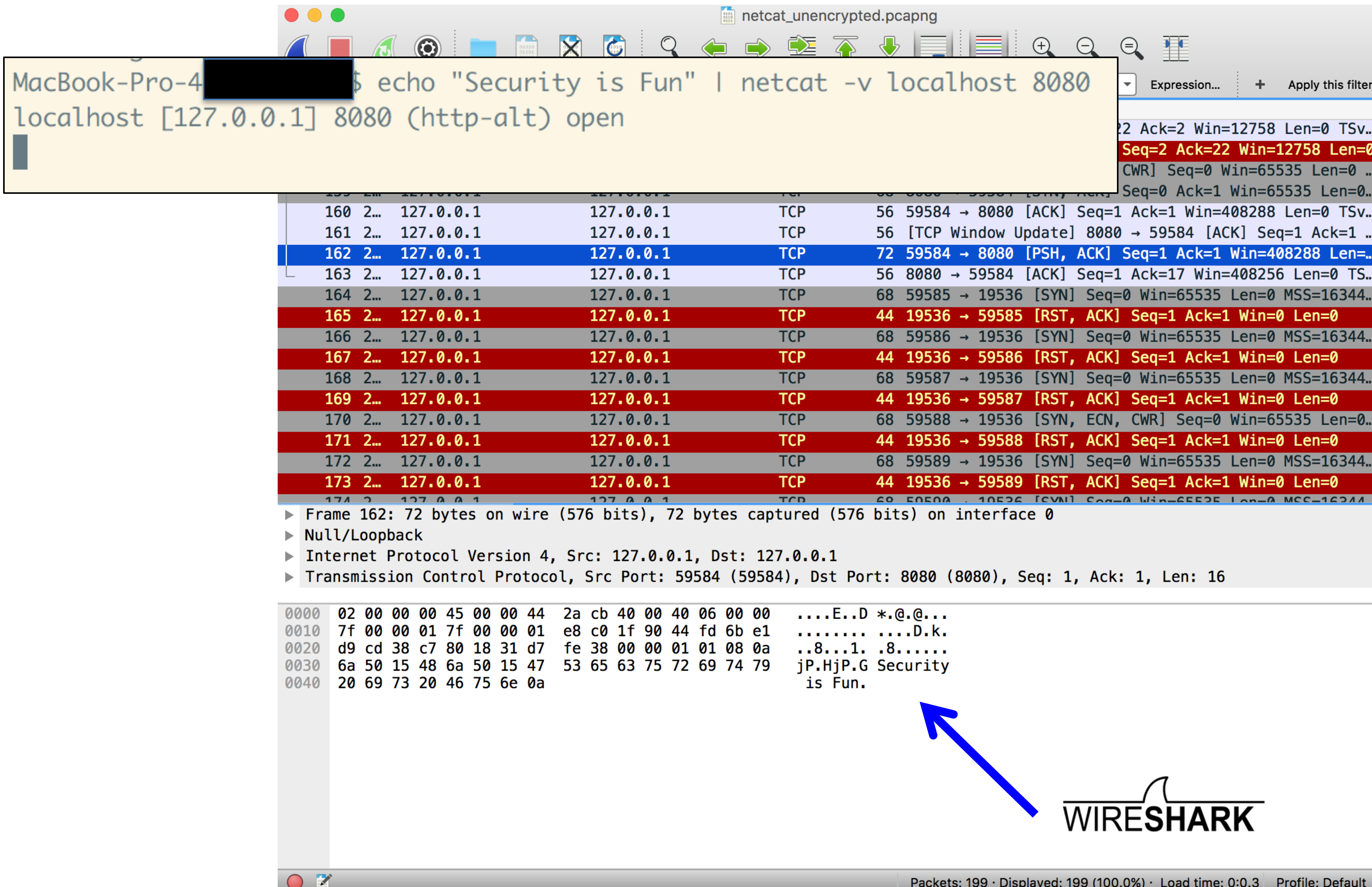
Internet



Bob's Switch



Why is crypto useful?



- Its just an instant message, right?

Alice uses the Internet for:

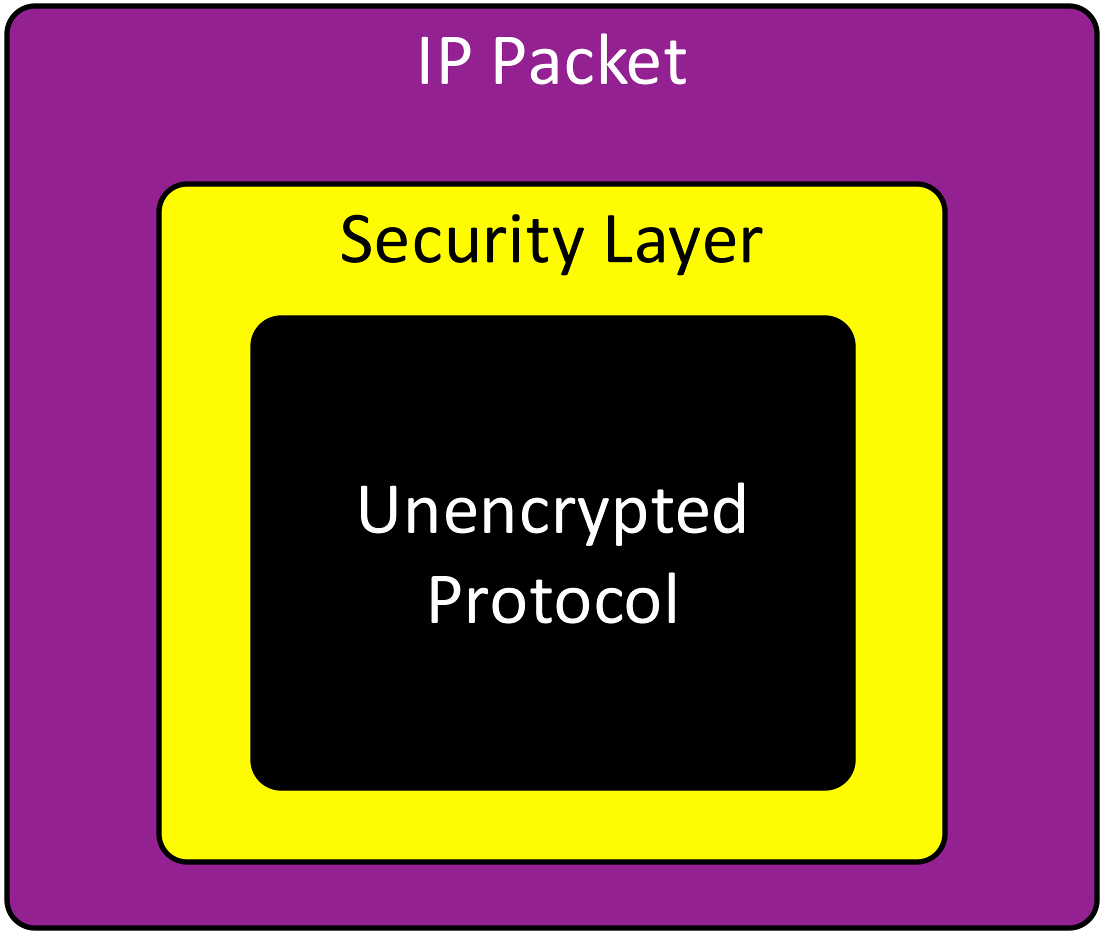
- Email
- Banking
- Online shopping
- Social networking
- …

Why is this bad?

Let’s use that crypto stuff

- Let’s build some new protocols
  - HTTP → SecureHTTP
  - POP → POPSecure
  - IMAP → Cry
  - SMTP → SMTP
  - FTP → FTPS
  - Jabber → SecJabber
  - Telnet → TelCryptNet

Let’s build a crypto-wrapper standard instead


IP Packet
Security Layer
Unencrypted
Protocol

What properties should this crypto-wrapper have?

- Confidentiality
- Integrity
- Authenticity
  - Server
  - Client
  - Mutual authentication

SSL / TLS

History

- **Secure Sockets Layer (SSL)** developed by Netscape (remember them?) in 1995
  - Version 1 never released
  - Version 2 incorporated into Netscape Navigator 1.1
  - Microsoft fixes vulnerabilities in SSLv2 and introduces Private Communications Technology (PCT) protocol
  - Netscape overhauls SSLv2, fixing some more security issues, and releases SSLv3
  - IETF takes over and releases **Transport Layer Security (TLS)**, a non-interoperable upgrade to SSLv3
    - current version is TLS version 1.3, RFC 8446 (August 2018)

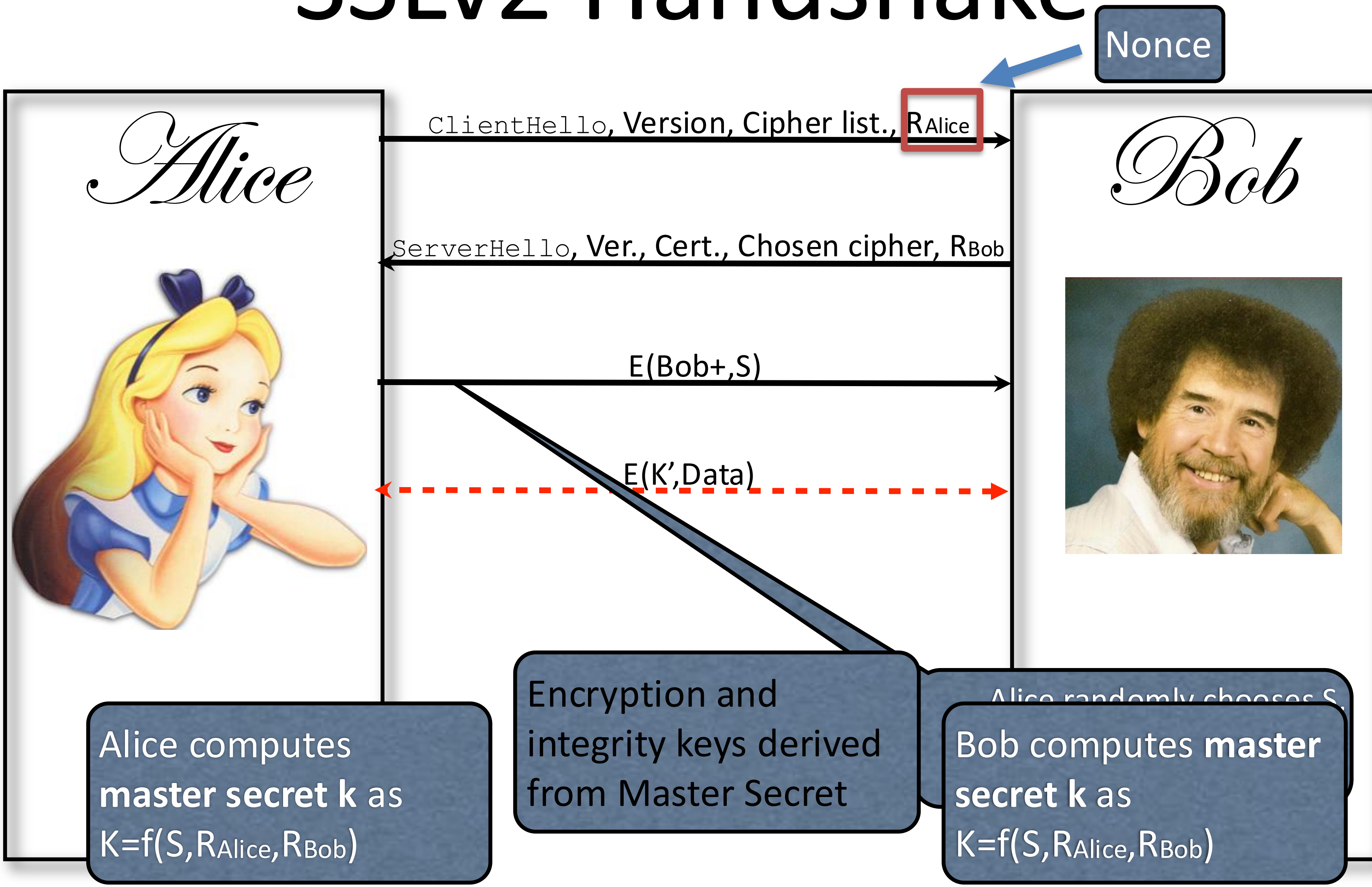
K.I.S.S.

- Application-layer protocol
- Operates over TCP **-- WHY?**

Overview

- Alice (client) initiates conversation with Bob (server)
- Bob sends Alice his certificate
- Alice verifies certificate
- Alice picks a random number S and sends it to Bob, encrypted with Bob's public key
- Both parties derive key material from S
- Client and server exchange encrypted and integrity-protected data

SSLv2 Handshake



Cryptographic Parameters

- Generated from
  - the master secret K
  - Rc
  - Rs
- *Six values* to be generated
  - client authentication and encryption keys
  - server authentication and encryption keys
  - client encryption IV
  - server encryption IV
- Generator functions: $k_i = g_i(K,Rc,Rs)$

Authentication

Alice

ClientHello, Version, Cipher list. RAlice

ServerHello, Ver., Cert., Chosen cipher, RBob

E(Bob+,S)

E(K’,Data)

Q: Which parties are authenticated?

Bob

Bob Barker

Cipher Suites

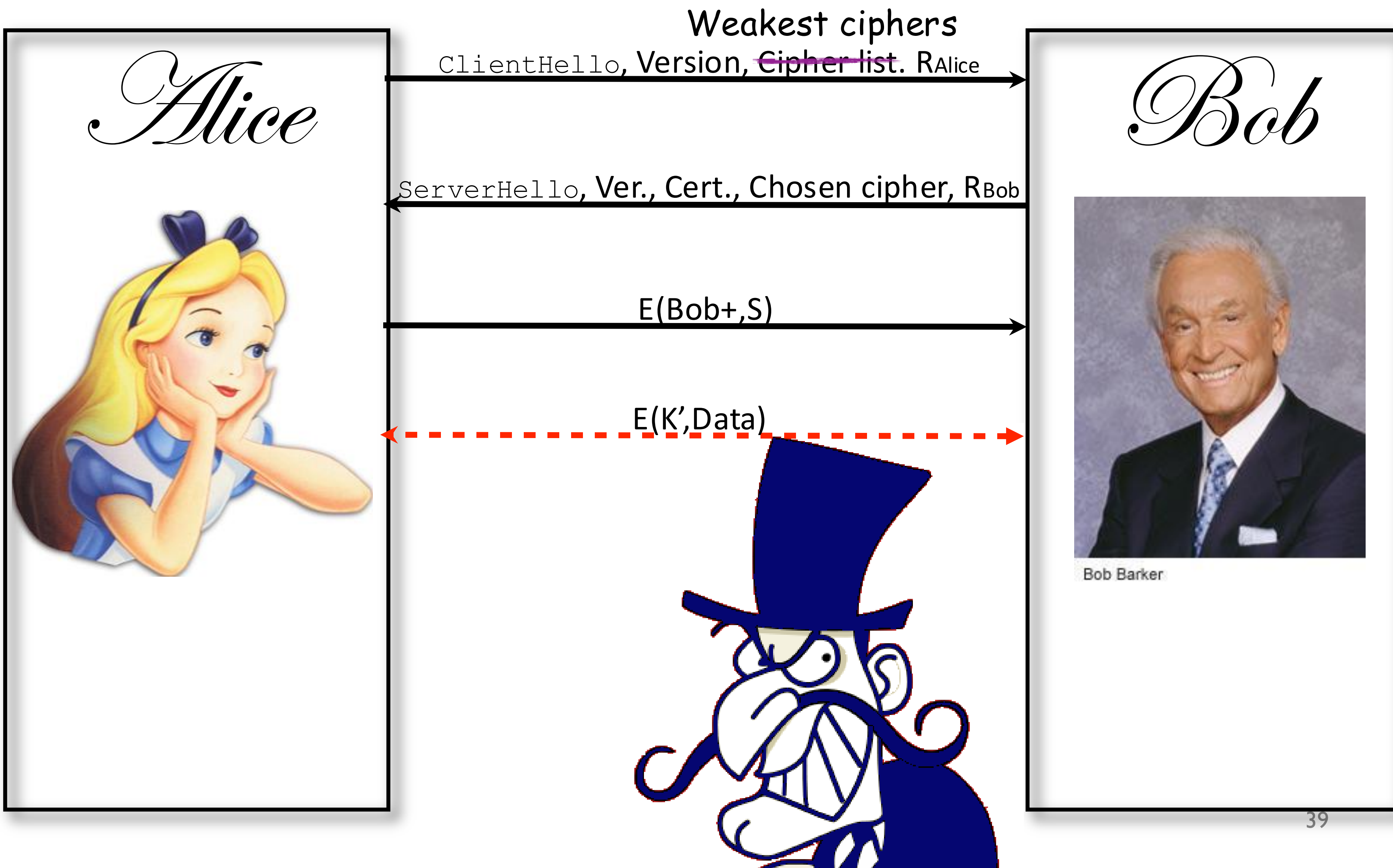
- Alice gives Bob a list of supported cipher suites; Bob makes final choice

```
% openssl ciphers -v
TLS_AES_256_GCM_SHA384    TLSv1.3 Kx=any    Au=any    Enc=AESGCM(256)  Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256  TLSv1.3 Kx=any Au=any Enc=CHACHA20/POLY1305(256)  Mac=AEAD
TLS_AES_128_GCM_SHA256    TLSv1.3 Kx=any     Au=any    Enc=AESGCM(128)       Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384  TLSv1.2 Kx=ECDH    Au=ECDSA Enc=AESGCM(256)  Mac=AEAD
ECDHE-RSA-AES256-GCM-SHA384    TLSv1.2 Kx=ECDH   Au=RSA    Enc=AESGCM(256)    Mac=AEAD
DHE-RSA-AES256-GCM-SHA384      TLSv1.2 Kx=DH        Au=RSA    Enc=AESGCM(256)    Mac=AEAD
ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH Au=ECDSA Enc=CHACHA20/POLY1305(256)  Mac=AEAD
ECDHE-RSA-CHACHA20-POLY1305  TLSv1.2 Kx=ECDH   Au=RSA   Enc=CHACHA20/POLY1305(256)  Mac=AEAD
DHE-RSA-CHACHA20-POLY1305  TLSv1.2 Kx=DH    Au=RSA    Enc=CHACHA20/POLY1305(256)  Mac=AEAD
..
..
..
```

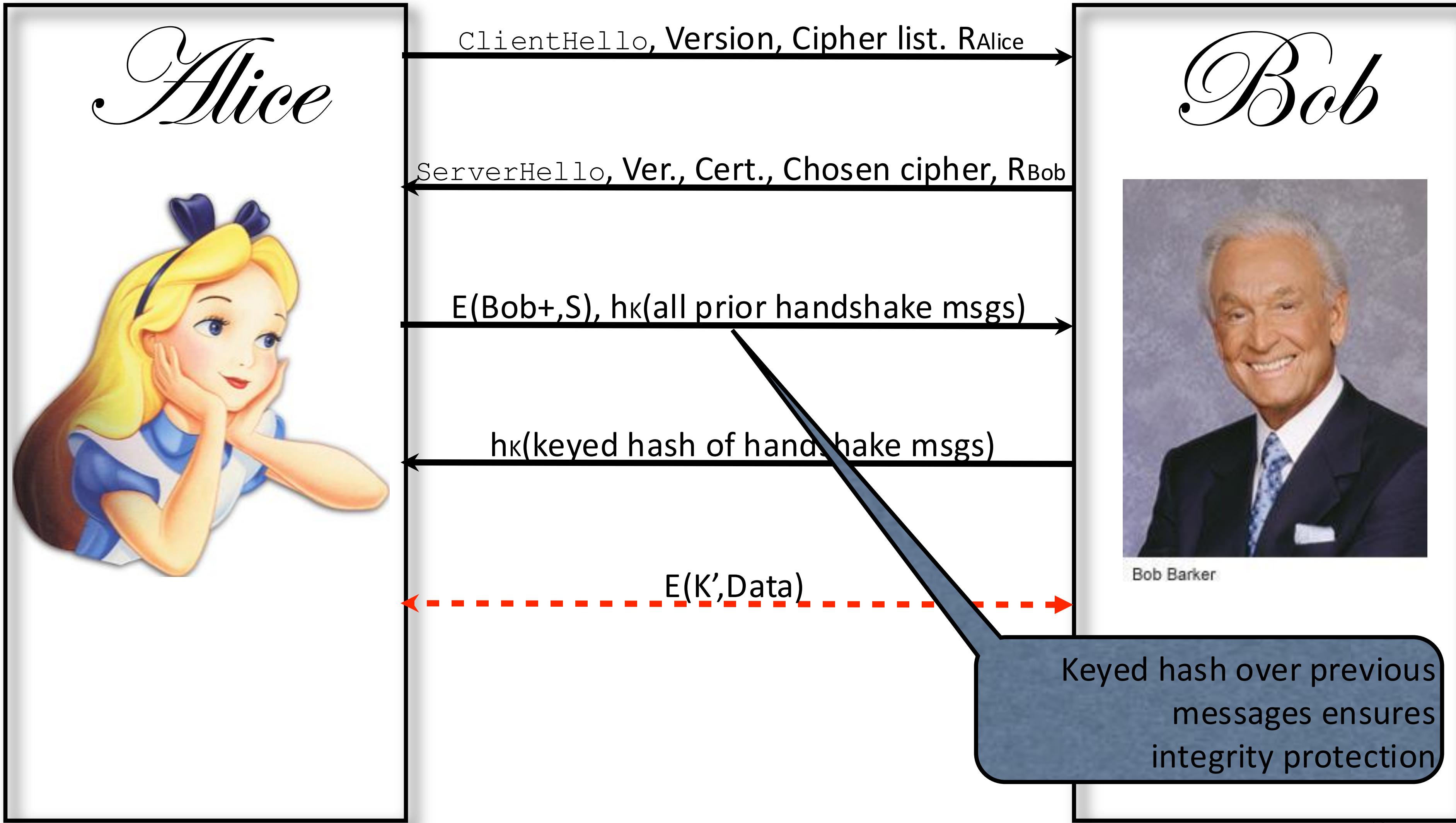
- Includes encryption algorithms, key length, block mode, and integrity checksum algorithm
- Only 5 supported in TLS1.3, >30 in TLS1.2

- Key Exchange algos e.g. RSA, DH, ECDH
- Authentication algos e.g., RSA
- Bulk encryption algos e.g., AES
- MAC algos e.g., SHA-256

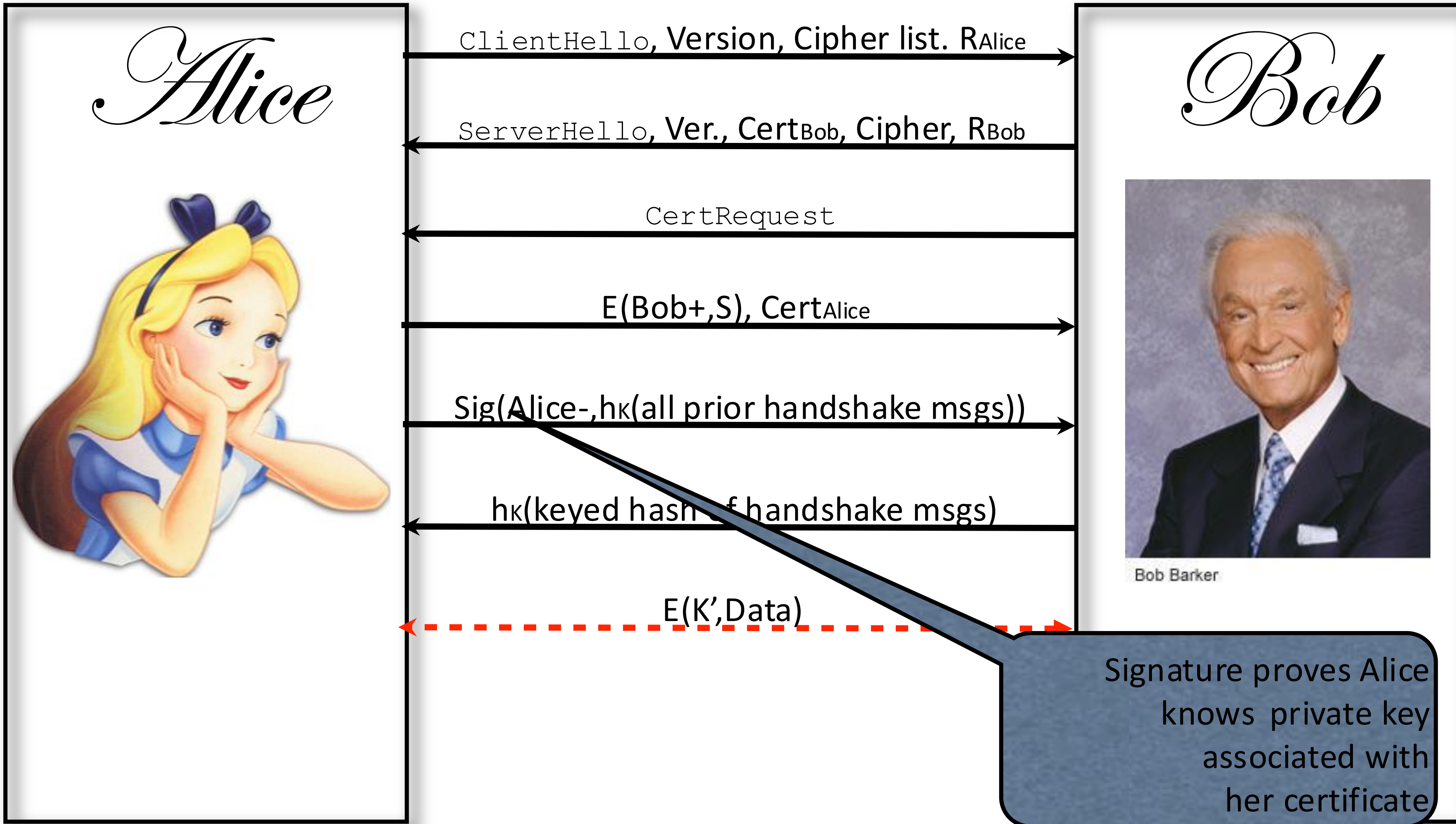
SSLv2 Problems



SSLv3 Fixes

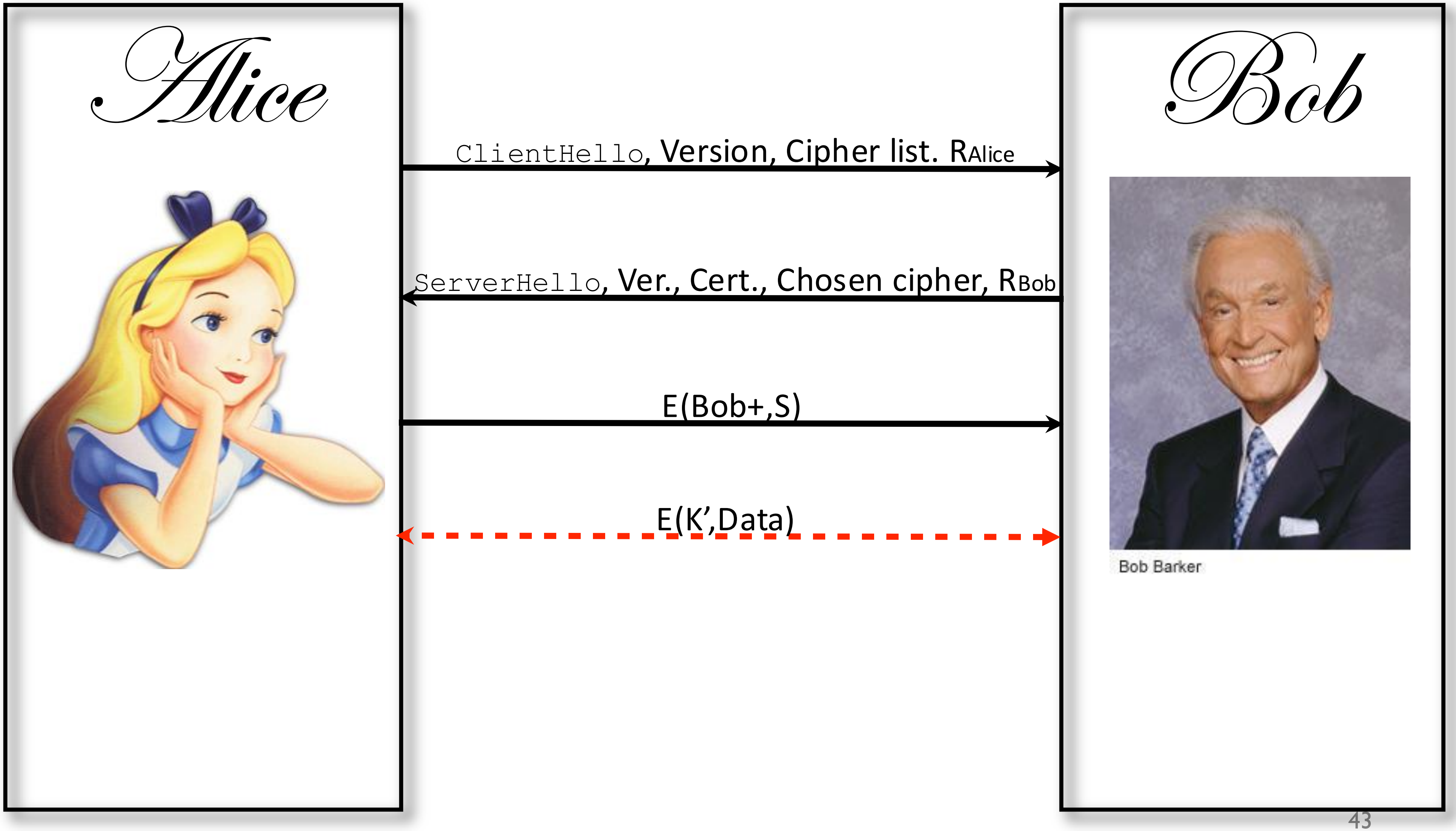

Alice
Bob
ClientHello, Version, Cipher list. R_{Alice}
ServerHello, Ver., Cert., Chosen cipher, R_{Bob}
E(Bob+,S), h_K(all prior handshake msgs)
h_K(keyed hash of handshake msgs)
E(K',Data)
Bob Barker
Keyed hash over previous messages ensures integrity protection

SSL/TLS with Server and Client Authentication

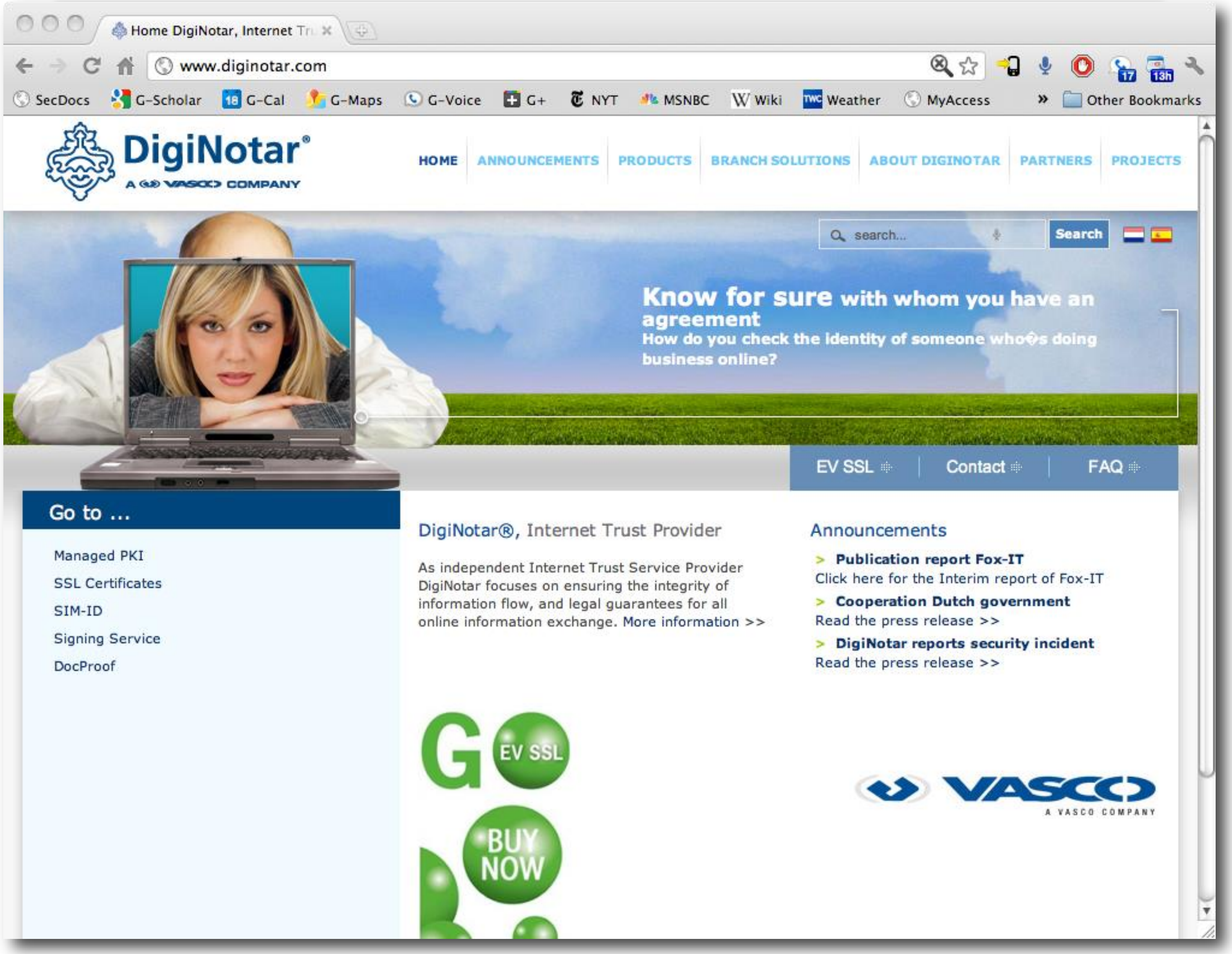


Problems with TLS/SSL

If Bob's cert isn't verified, how do you know you're actually talking to Bob?

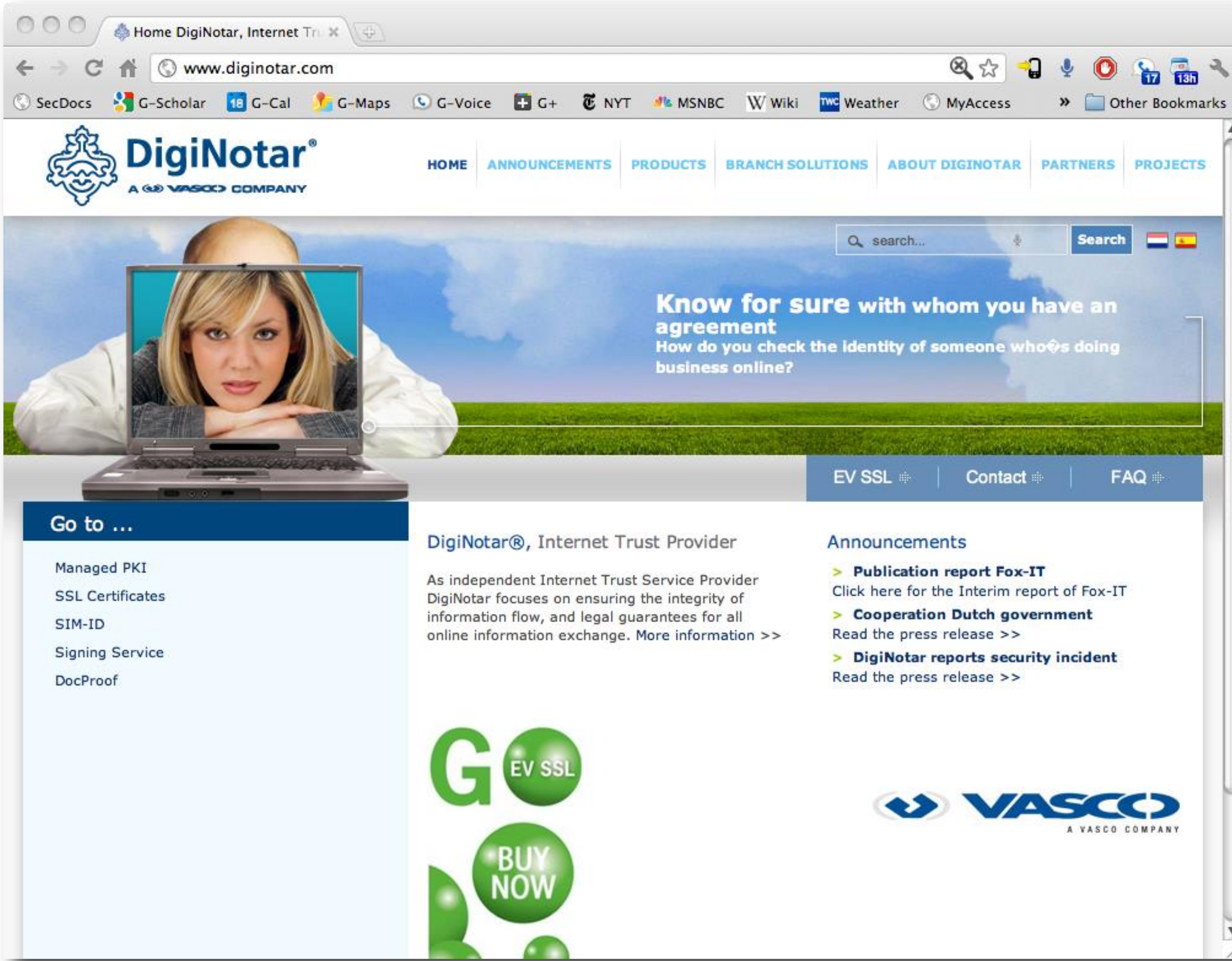


Solution: Use a PKI



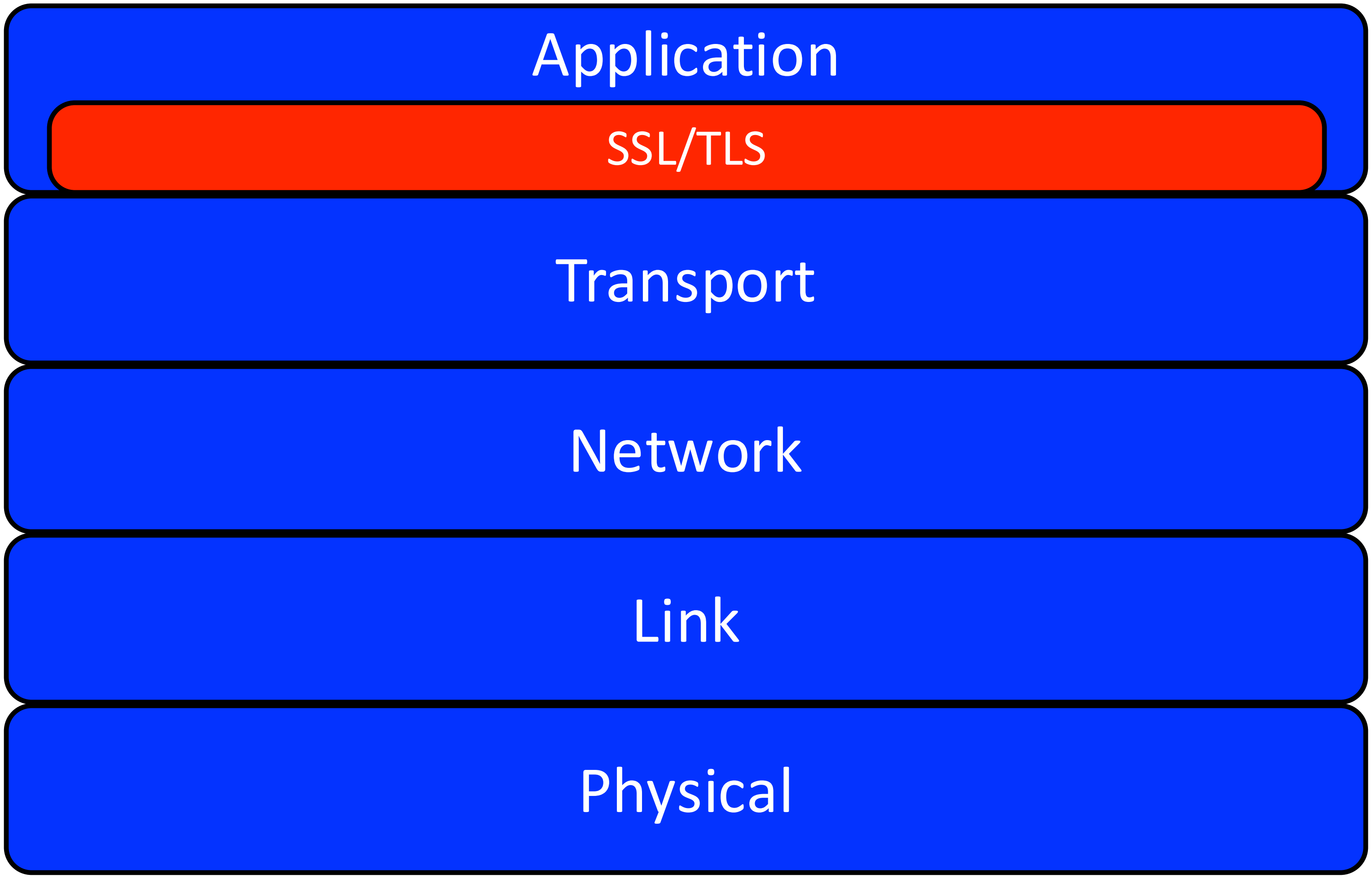
- Any CA may sign any certificate
- Browser weighs all root CAs equally
- *Q: Do you recall why this is problematic?*

Recall: The DigiNotar Incident



SSL/TLS in the Real World

Network Stack, revisited

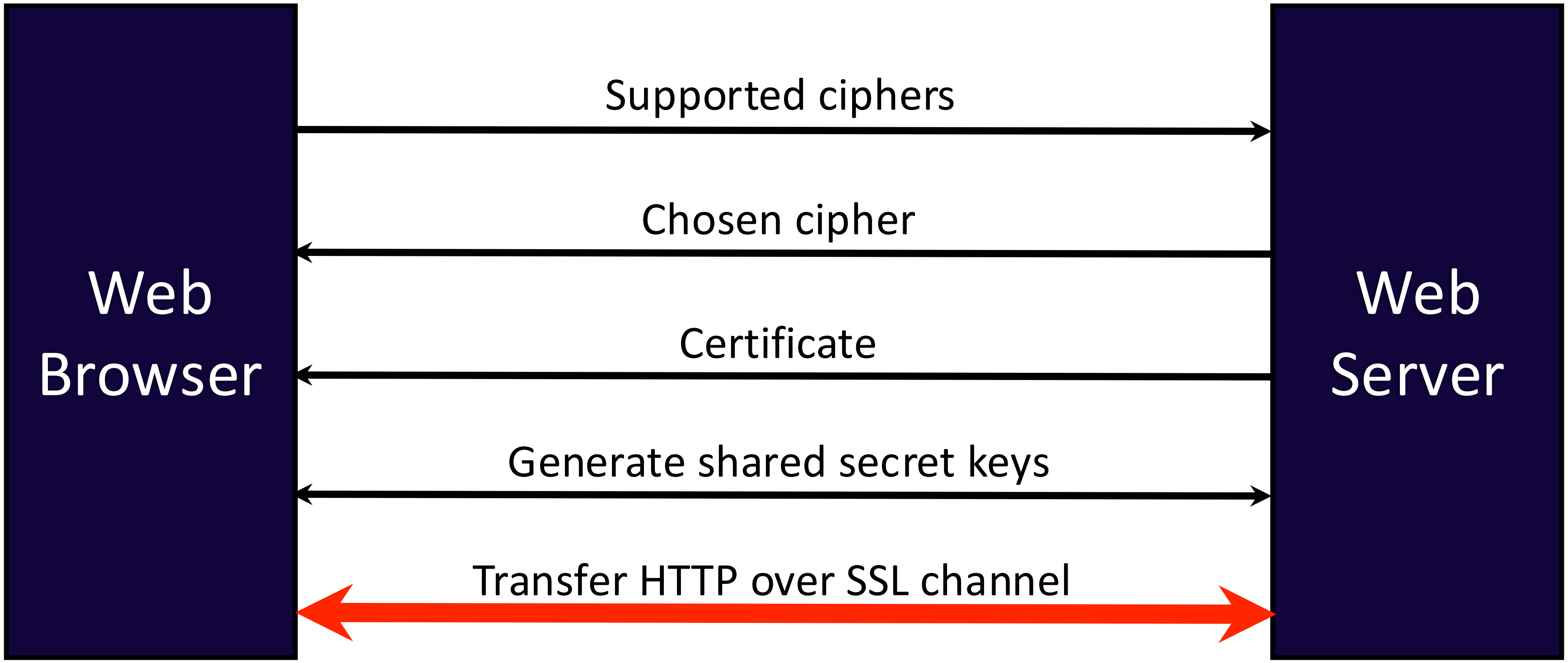


SSL/TLS in the Real World

- All (modern) browsers support TLS 1.2, TLS1.3
  - SSLv3 deprecated in most major browsers
- Client authentication very rare **-- WHY?**
- Implementations:
  - HTTP (80) → HTTPS (443)
  - POP (110) → POP3S (995)
  - IMAP (143) → IMAPS (993)
  - SMTP (25) → SMTP with SSL (465)
  - FTP (20,21)→ FTPS (989,990)
  - Telnet (23) → Telnets (992)

SSL/TLS and the Web

- HTTPS: Tunnel HTTP over SSL/TLS
- Add golden lock symbol



The verifier matters

- SSL is an *application layer protocol*
  - Software developers must use it correctly

- Pre-Smartphone World
  - Small set of applications that use SSL (E.g., Web Browser)
  - Lots of attention to those apps

→

- Smartphone World
  - Possibly *millions of applications that use SSL*
  - Many apps do not verify certificates correctly – **Implications?**
  - Developers change default configuration – **WHY?**

SSL Verification in Apps

- Even popular apps are vulnerable to incorrect SSL use
  - Banking
  - Document storage
  - Social Networks (Facebook, before *Firesheep*)
  - …..and **IoT apps**
  - …
- Common mistakes: Generally, in HTTPS use.
  1. Not using SSL
  2. Mixed SSL use
  3. Accepting all certificates
  4. Accepting all hostnames (i.e., regardless of the CN)
  5. Trusting all CAs

Not using SSL

- What happens when you don't use SSL? E.g., http://www.mybank.com/loggedin?sessionid=11
  - If I can *guess*, *infer*, or *steal* the session ID, game over
- Are there any use cases where not using SSL would be okay?
  - It depends. However, unless confidentiality and authenticity are *never* going to be important to the app, use SSL!

Lesson 1: Always use SSL (i.e., mostly HTTPS)

Mixed SSL use

- Mixed use of HTTP and HTTPS on the same site.
- **Use case 1:** Login page is not HTTPS, but the login form is submitted to a HTTPS page.
  - MiTM can *replace HTTPS links with HTTP* (i.e., SSL Stripping)
- **Use case 2:** Login page is HTTPS, but the rest of the website may be HTTP
  - *Unencrypted cookies/session IDs!* (e.g., Firesheep)

Lesson 2: Use HTTPS throughout

Certificate Validation

- Apps can override the *TrustManager* interface

69

```
SSLContext sslContext = SSLContext.getInstance("SSL");

// set up a TrustManager that trusts everything
sslContext.init(null, new TrustManager[] { new X509TrustManager() {
            public X509Certificate[] getAcceptedIssuers() {
                    System.out.println("getAcceptedIssuers =============");
                    return null;
            }

            public void checkServerTrusted(X509Certificate[] certs,
                          String authType) {
                    System.out.println("checkServerTrusted =============");
            }
} }, new SecureRandom());
```

https://stackoverflow.com/questions/2703161/how-to-ignore-ssl-certificate-errors-in-apache-httpclient-4-0

- What is wrong with this example? It accepts all server certificates!

Lesson 3: Always validate the server's certificate

Using self-signed certificates

- *The right way:* Certificate Pinning
  - i.e., hardcode your self-signed certificate.

```
CertificateFactory cf = CertificateFactory.getInstance("X.509");
// From https://www.washington.edu/itconnect/security/ca/load-der.crt
InputStream caInput = new BufferedInputStream(new FileInputStream("load-der.crt"));
Certificate ca;
try {
    ca = cf.generateCertificate(caInput);
    System.out.println("ca=" + ((X509Certificate) ca).getSubjectDN());
} finally {
    caInput.close();
}
```

Step 1: Read in your certificate

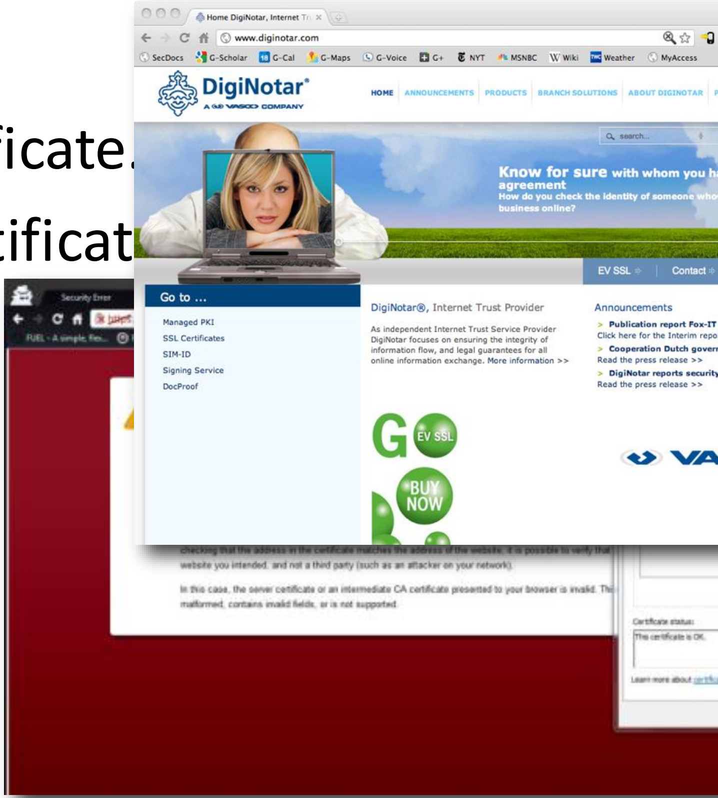
```
// Create a KeyStore containing our trusted CAs
String keyStoreType = KeyStore.getDefaultType();
KeyStore keyStore = KeyStore.getInstance(keyStoreType);
keyStore.load(null, null);
keyStore.setCertificateEntry("ca", ca);

// Create a TrustManager that trusts the CAs in our KeyStore
String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);

// Create an SSLContext that uses our TrustManager
SSLContext context = SSLContext.getInstance("TLS");
context.init(null, tmf.getTrustManagers(), null);
```

Step 2: Create custom TrustManager

Step 3: Compare server certificate with the hard-coded one ✓ ✗

Using self-signed certificates

- *The right way:* Certificate Pinning
  - i.e., hardcode your self-signed certificate.
  - Allows *secure* use of self-signed certificat
- Variation:
  - Pinning own CA certificate
  - Gives you more flexibility.
- How to change the certificate?
  - App updates!
- Don't have to trust 100s of Root CAs!



Lesson 4: Certificate pinning, if done correctly, is more secure than *default SSL use.*

Hostname Verification

- Back to basics: What does a certificate provide?
  - Binding between a *public key* and *identity*

```
HostnameVerifier hostnameVerifier = org.apache.http.conn.ssl.SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER;

DefaultHttpClient client = new DefaultHttpClient();

SchemeRegistry registry = new SchemeRegistry();
SSLSocketFactory socketFactory = SSLSocketFactory.getSocketFactory();
socketFactory.setHostnameVerifier((X509HostnameVerifier) hostnameVerifier);
registry.register(new Scheme("https", socketFactory, 443));
SingleClientConnManager mgr = new SingleClientConnManager(client.getParams(), registry);
DefaultHttpClient httpClient = new DefaultHttpClient(mgr, client.getParams());

// Set verifier
HttpsURLConnection.setDefaultHostnameVerifier(hostnameVerifier);
```

https://stackoverflow.com/questions/2012497/accepting-a-certificate-for-https-on-android?lq=1

- Any certificate issued by any trusted CA will be accepted!
  - i.e., HostName= google.com, but cert has CN=foogle.com? ✓

Lesson 5: Never override the HostNameVerifier