

CIS 4930: Secure IoT

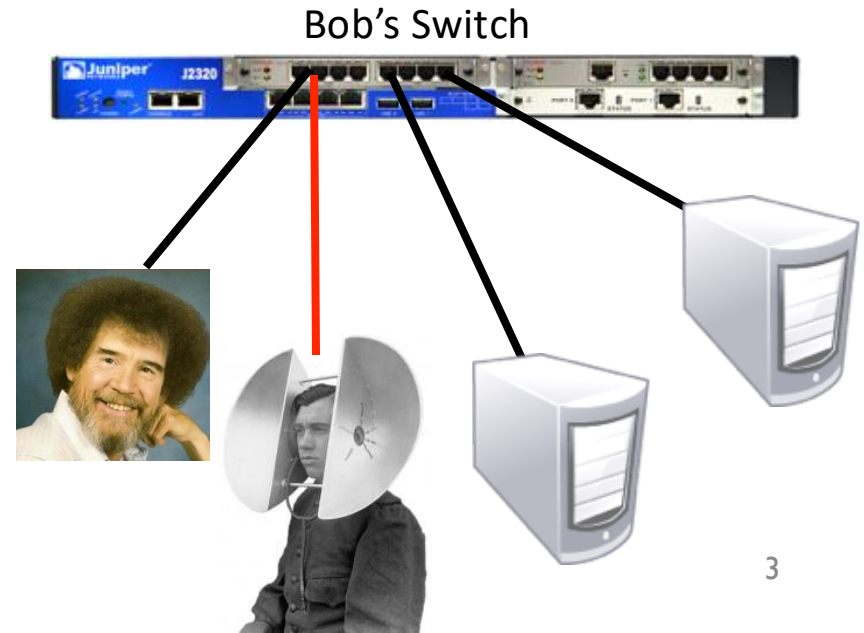
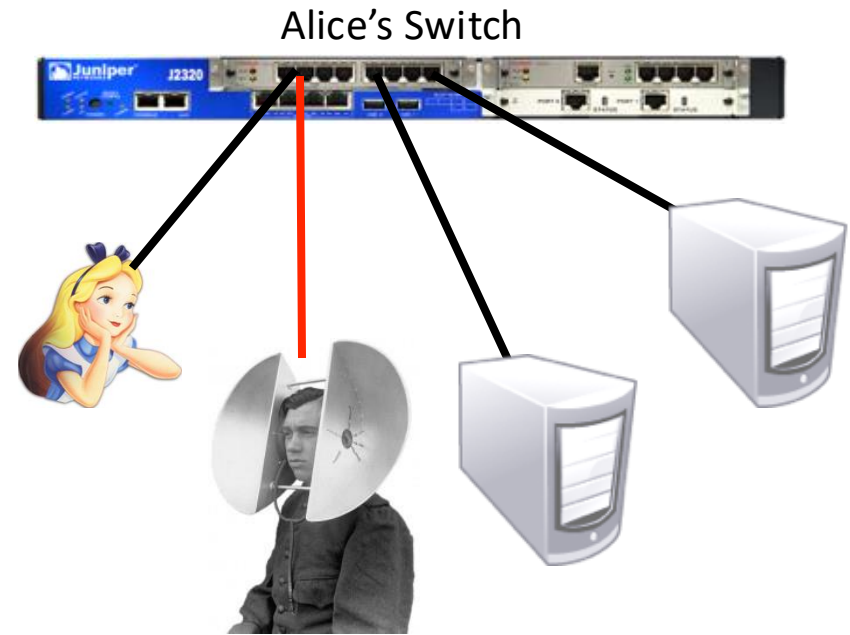
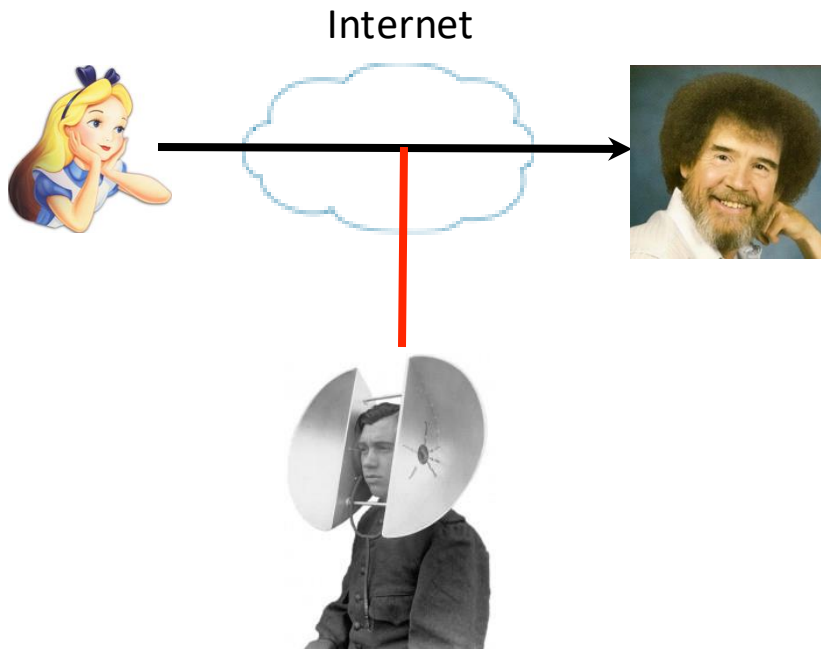
Lecture 6

Prof. Kaushal Kafle

Class Notes

- Grades for homework 1 [posted](#).
- Homework 2 due on **09/19**
- 3-4 groups yet to meet to discuss their project proposals.

Eavesdropping



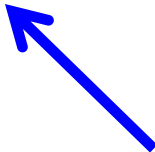
Why is crypto useful?


MacBook-Pro-4 [redacted] \$ echo "Security is Fun" | netcat -v localhost 8080
localhost [127.0.0.1] 8080 (http-alt) open

No.	Time	Source	Destination	Protocol	Length	Info
160	2...	127.0.0.1	127.0.0.1	TCP	56	59584 → 8080 [ACK] Seq=1 Ack=1 Win=408288 Len=0 TSv...
161	2...	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8080 → 59584 [ACK] Seq=1 Ack=1 ...
162	2...	127.0.0.1	127.0.0.1	TCP	72	59584 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=408288 Len=...
163	2...	127.0.0.1	127.0.0.1	TCP	56	8080 → 59584 [ACK] Seq=1 Ack=17 Win=408256 Len=0 TS...
164	2...	127.0.0.1	127.0.0.1	TCP	68	59585 → 19536 [SYN] Seq=0 Win=65535 Len=0 MSS=16344...
165	2...	127.0.0.1	127.0.0.1	TCP	44	19536 → 59585 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
166	2...	127.0.0.1	127.0.0.1	TCP	68	59586 → 19536 [SYN] Seq=0 Win=65535 Len=0 MSS=16344...
167	2...	127.0.0.1	127.0.0.1	TCP	44	19536 → 59586 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
168	2...	127.0.0.1	127.0.0.1	TCP	68	59587 → 19536 [SYN] Seq=0 Win=65535 Len=0 MSS=16344...
169	2...	127.0.0.1	127.0.0.1	TCP	44	19536 → 59587 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
170	2...	127.0.0.1	127.0.0.1	TCP	68	59588 → 19536 [SYN, ECN, CWR] Seq=0 Win=65535 Len=0...
171	2...	127.0.0.1	127.0.0.1	TCP	44	19536 → 59588 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
172	2...	127.0.0.1	127.0.0.1	TCP	68	59589 → 19536 [SYN] Seq=0 Win=65535 Len=0 MSS=16344...
173	2...	127.0.0.1	127.0.0.1	TCP	44	19536 → 59589 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

▶ Frame 162: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 59584 (59584), Dst Port: 8080 (8080), Seq: 1, Ack: 1, Len: 16

```
0000  02 00 00 00 45 00 00 44  2a cb 40 00 40 06 00 00  ....E..D *.@.@...
0010  7f 00 00 01 7f 00 00 01  e8 c0 1f 90 44 fd 6b e1  ....D.k.
0020  d9 cd 38 c7 80 18 31 d7  fe 38 00 00 01 01 08 0a  ..8...1. .8.....
0030  6a 50 15 48 6a 50 15 47  53 65 63 75 72 69 74 79  jP.HjP.G Security
0040  20 69 73 20 46 75 6e 0a  is Fun.
```





Packets: 199 · Displayed: 199 (100.0%) · Load time: 0:0.3 Profile: Default

Why is this bad?

- Its just an instant message, right?

Alice uses the Internet for:

- **Email**
- **Banking**
- Online shopping
- Social networking
- ...

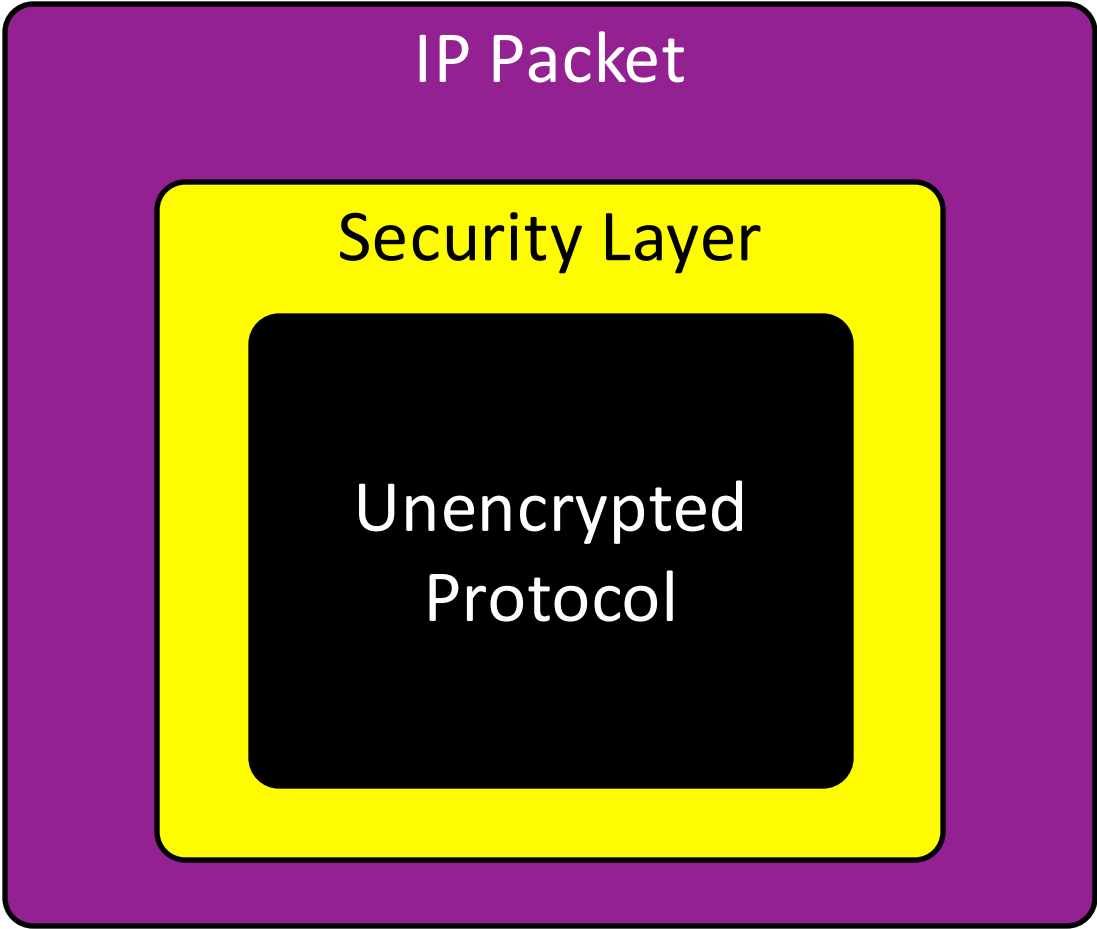
Let's use that crypto stuff

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



Let's build a crypto-wrapper standard instead





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?**

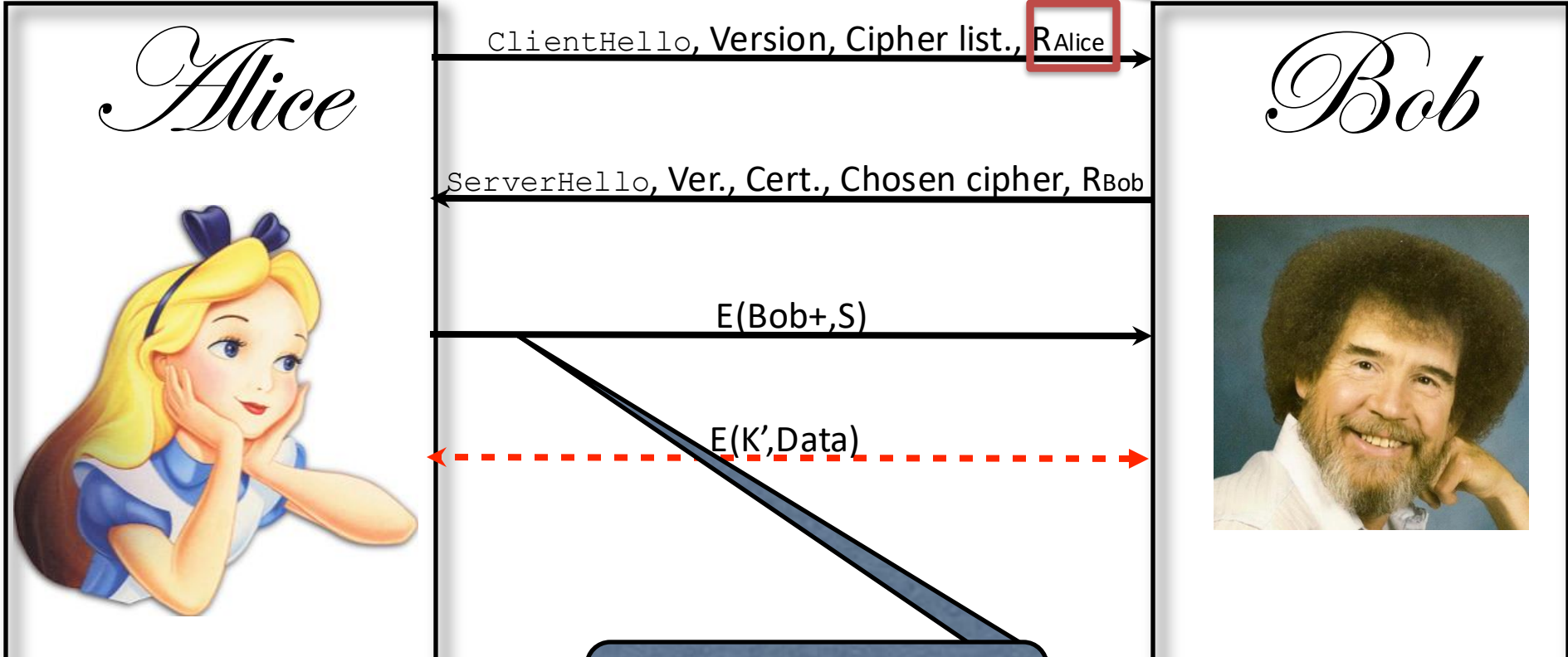


Overview

- 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

Nonce



Alice computes **master secret k** as
 $K=f(S, R_{Alice}, R_{Bob})$

Encryption and integrity keys derived from Master Secret

Alice randomly chooses S
Bob computes **master secret k** as
 $K=f(S, R_{Alice}, R_{Bob})$

Cryptographic Parameters

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

Authentication

Alice



ClientHello, Version, Cipher list, R_{Alice}

ServerHello, Ver., Cert., Chosen cipher, R_{Bob}

$E(Bob+, S)$

$E(K', Data)$

Q: Which parties are authenticated?

Bob



Bob Barker

Cipher Suites

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

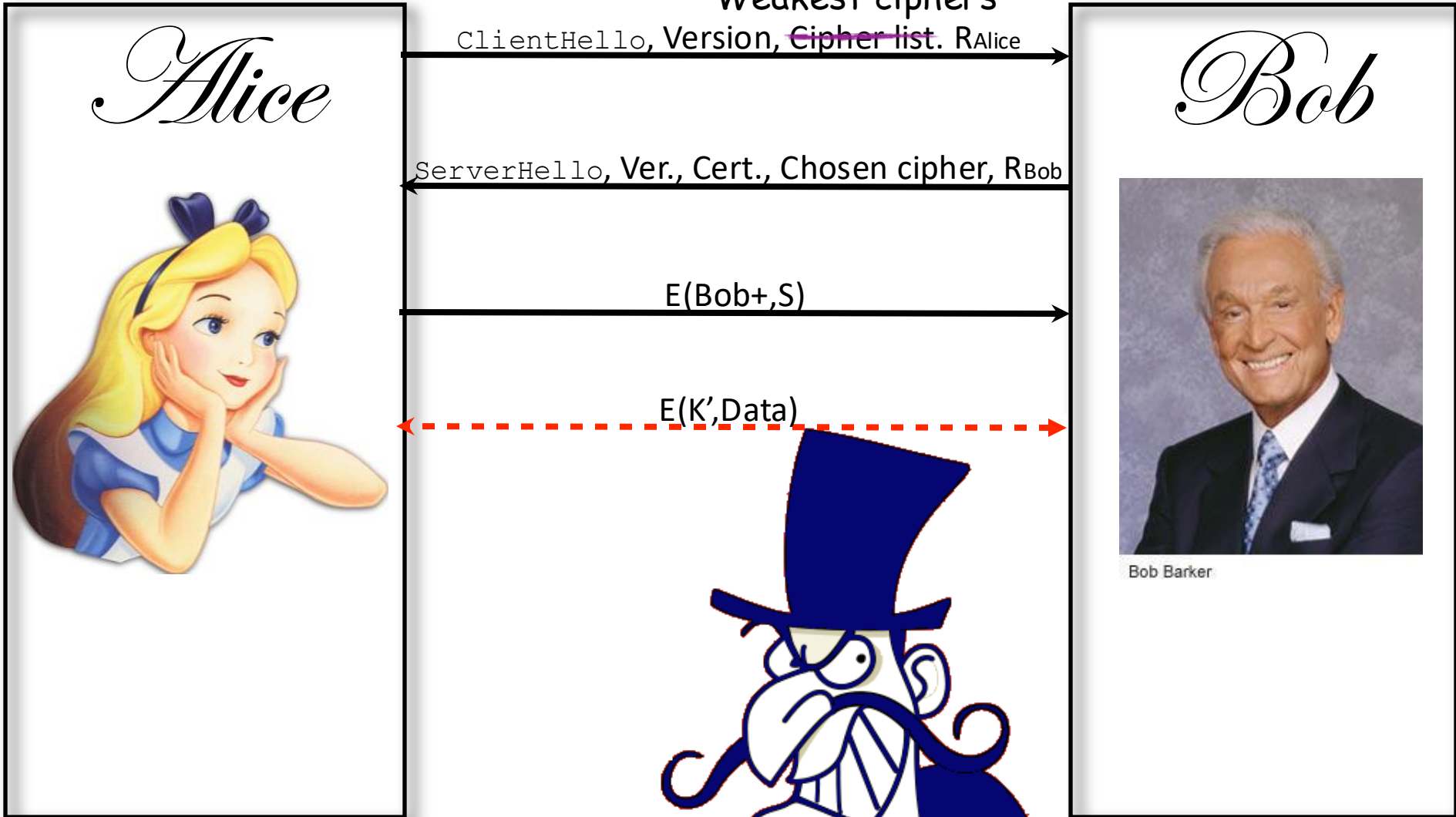
```
% openssl ciphers -v
TLS_AES_256_GCM_SHA384 TLSv1.3 Kx=any Au=any Enc=AESGCM(256) Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any Au=any Enc=CHACHA20/POLY1305(256) Mac=AEAD
TLS_AES_128_GCM_SHA256 TLSv1.3 Kx=any Au=any Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA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

Weakest ciphers



SSLv3 Fixes

Alice



ClientHello, Version, Cipher list, R_{Alice}

ServerHello, Ver., Cert., Chosen cipher, R_{Bob}

$E(Bob+S), h_K(\text{all prior handshake msgs})$

$h_{K'}(\text{keyed hash of handshake msgs})$

$E(K', \text{Data})$

Bob



Bob Barker

Keyed hash over previous messages ensures integrity protection

SSL/TLS with Server and Client Authentication

Alice



ClientHello, Version, Cipher list, R_{Alice}

ServerHello, Ver., Cert_{Bob}, Cipher, R_{Bob}

CertRequest

E(Bob+, S), Cert_{Alice}

Sig(Alice-, hk(all prior handshake msgs))

hk(keyed hash of handshake msgs)

E(K', Data)

Bob



Bob Barker

Signature proves Alice knows private key associated with her certificate

Problems with TLS/SSL

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

Alice



ClientHello, Version, Cipher list, R_{Alice}

ServerHello, Ver., Cert., Chosen cipher, R_{Bob}

$E(Bob, S)$

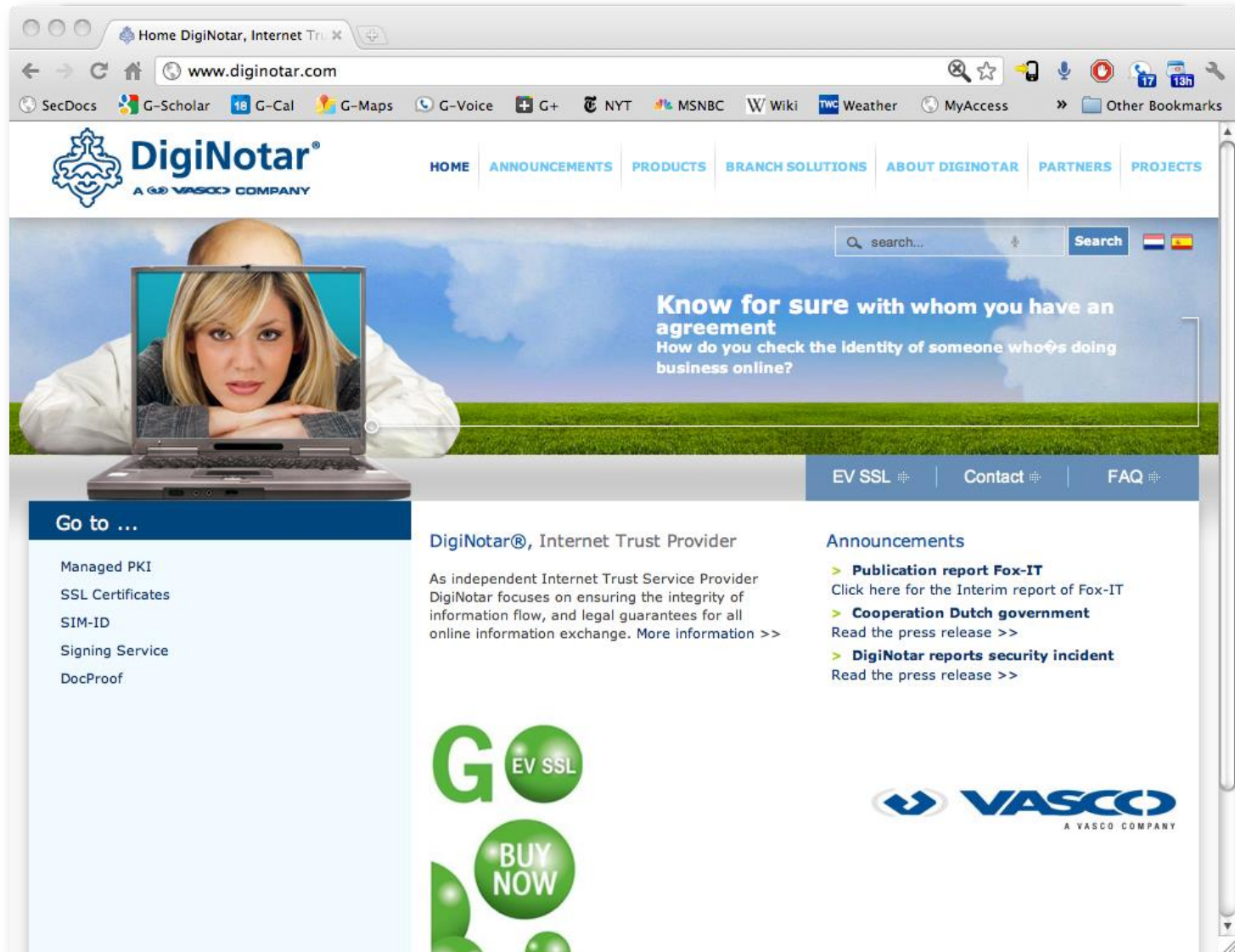
$E(K', Data)$

Bob



Bob Barker

Solution: Use a PKI



Home DigiNotar, Internet Tr...
www.diginotar.com

SecDocs G-Scholar 18 G-Cal G-Maps G-Voice G+ NYT MSNBC Wiki TWC Weather MyAccess Other Bookmarks

DigiNotar®
A VASCO COMPANY

HOME ANNOUNCEMENTS PRODUCTS BRANCH SOLUTIONS ABOUT DIGINOTAR PARTNERS PROJECTS

search... Search

Know for sure with whom you have an agreement
How do you check the identity of someone who's doing business online?

EV SSL Contact FAQ

Go to ...

- Managed PKI
- SSL Certificates
- SIM-ID
- Signing Service
- DocProof

DigiNotar®, Internet Trust Provider

As independent Internet Trust Service Provider DigiNotar focuses on ensuring the integrity of information flow, and legal guarantees for all online information exchange. More information >>

Announcements

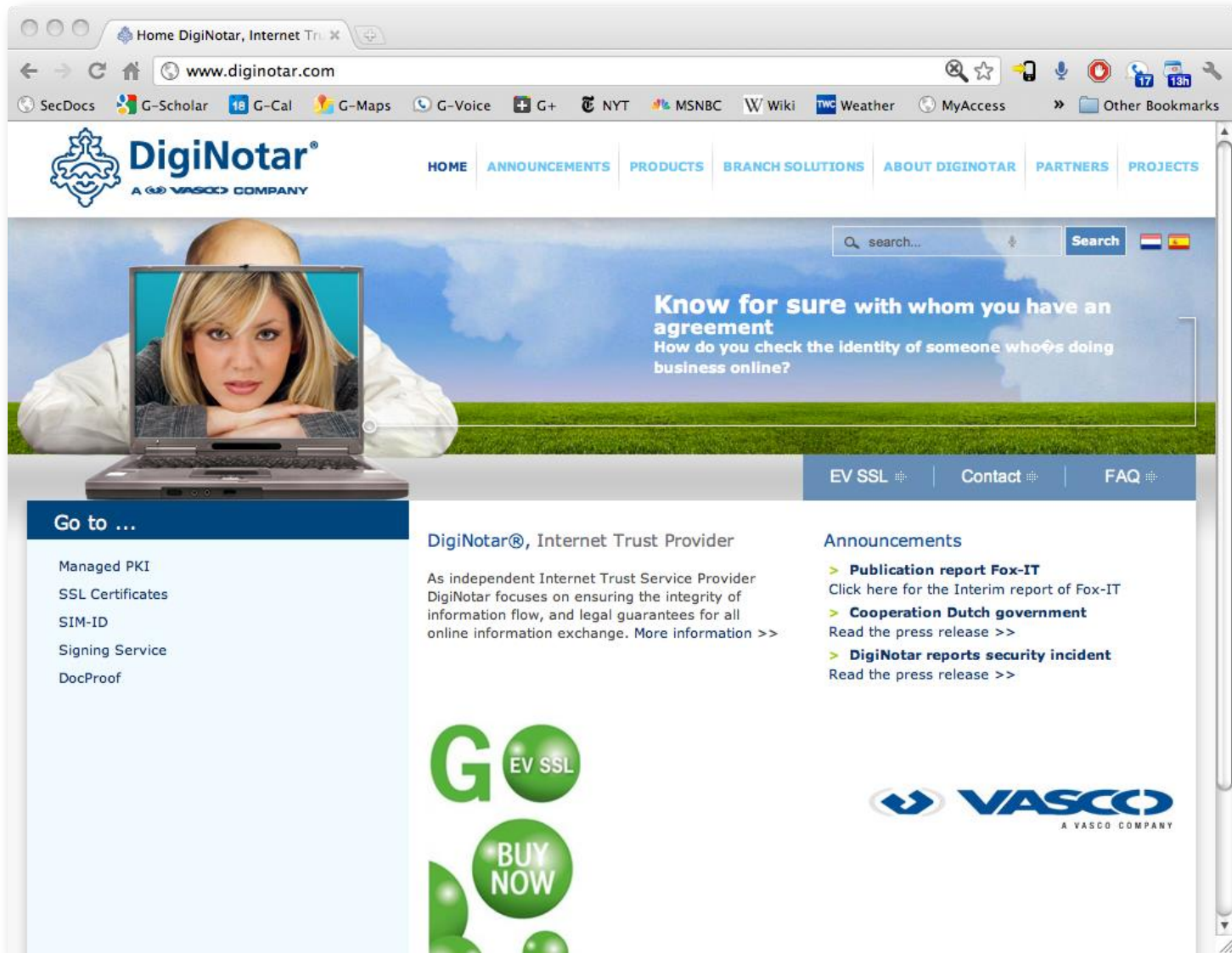
- > **Publication report Fox-IT**
Click here for the Interim report of Fox-IT
- > **Cooperation Dutch government**
Read the press release >>
- > **DigiNotar reports security incident**
Read the press release >>

GO EV SSL BUY NOW

VASCO
A VASCO COMPANY

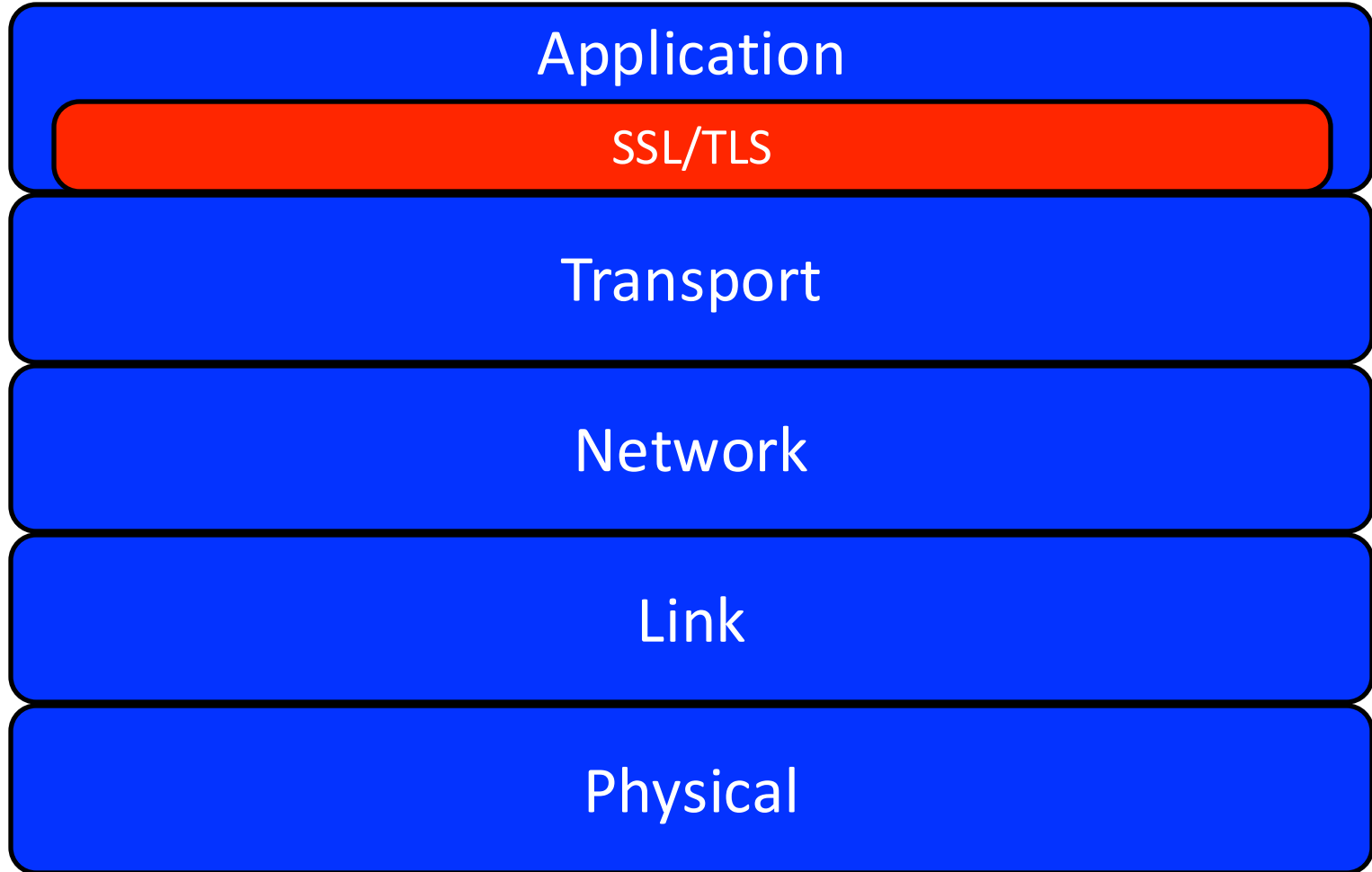
- Any CA may sign any certificate
- Browser weighs all root CAs equally
- *Q: Do you recall why this is problematic?*

Recall: The DigiNotar Incident



SSL/TLS in the Real World

Network Stack, revisited

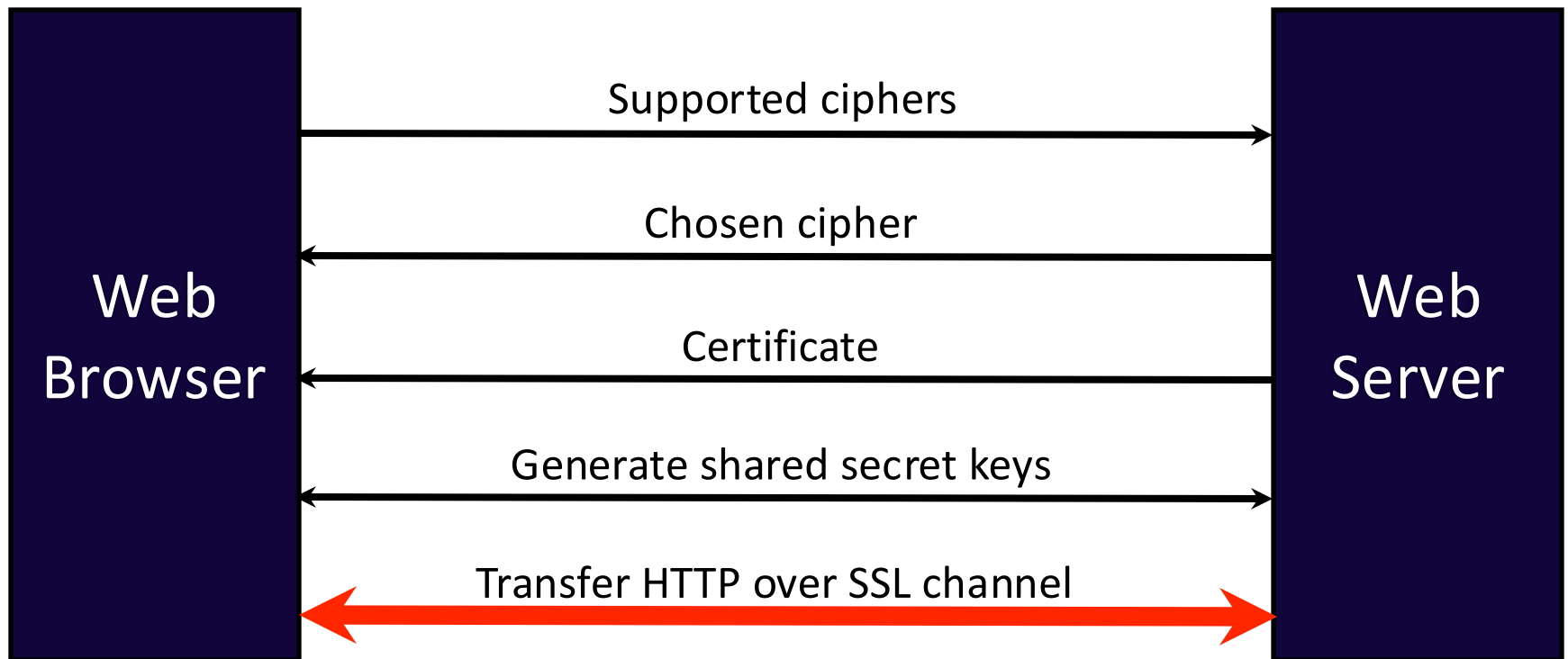


SSL/TLS in the Real World

- 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

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

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());
```

- 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);
```

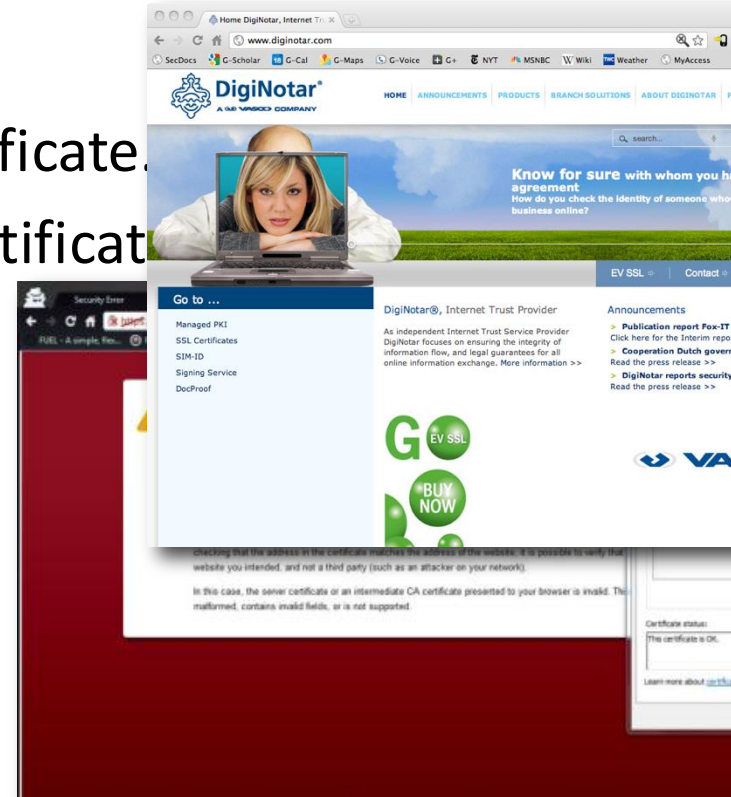
Step 2: Create custom TrustManager

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 certificates.
- 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

The End