CS 114: Network Security

Lecture 10 - Transport Layer Security

Prof. Daniel Votipka
Spring 2023

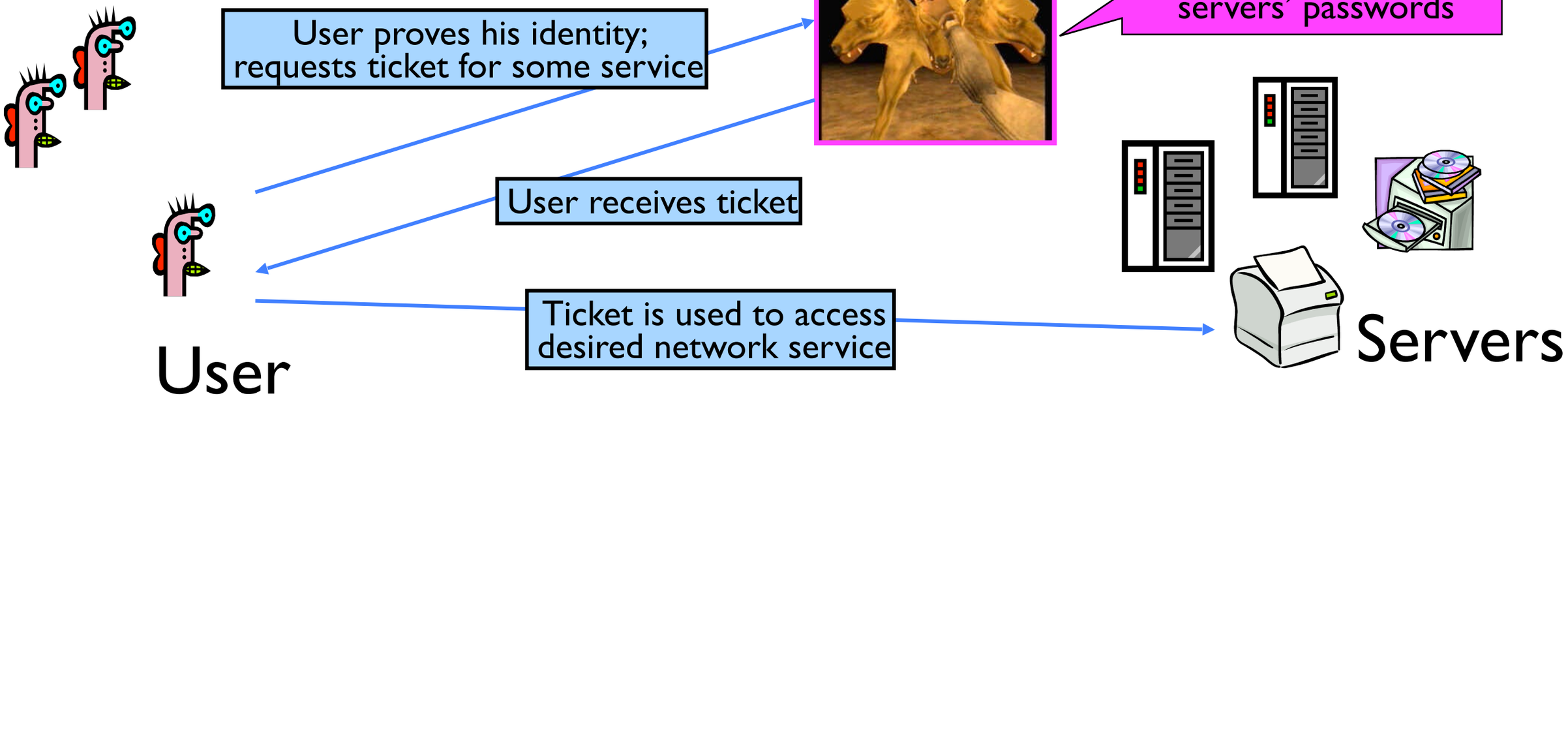
(some slides courtesy of Prof. Micah Sherr)

Administrivia

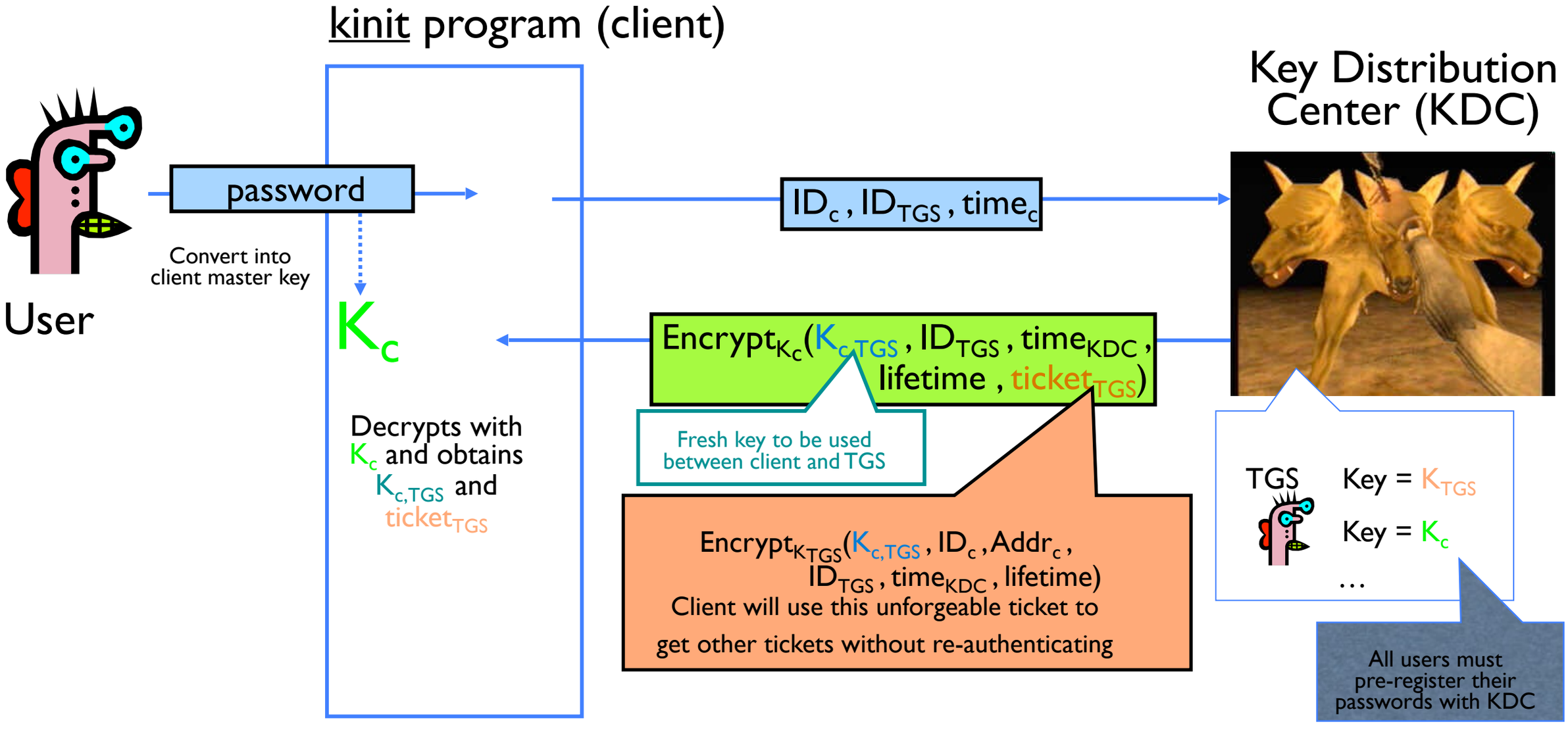
- Exam 1 has mostly been graded
  - Review next week!
- No lecture on Thursday (I will still have office hours)
- Homework 1, part 2 extended to Mar. 7th at 11:59pm
  - Limited TA Office hours (See Piazza) this week
  - HMAC test verified manually
  - LaTeX template in Box

Kerberos

Kerberos Overview

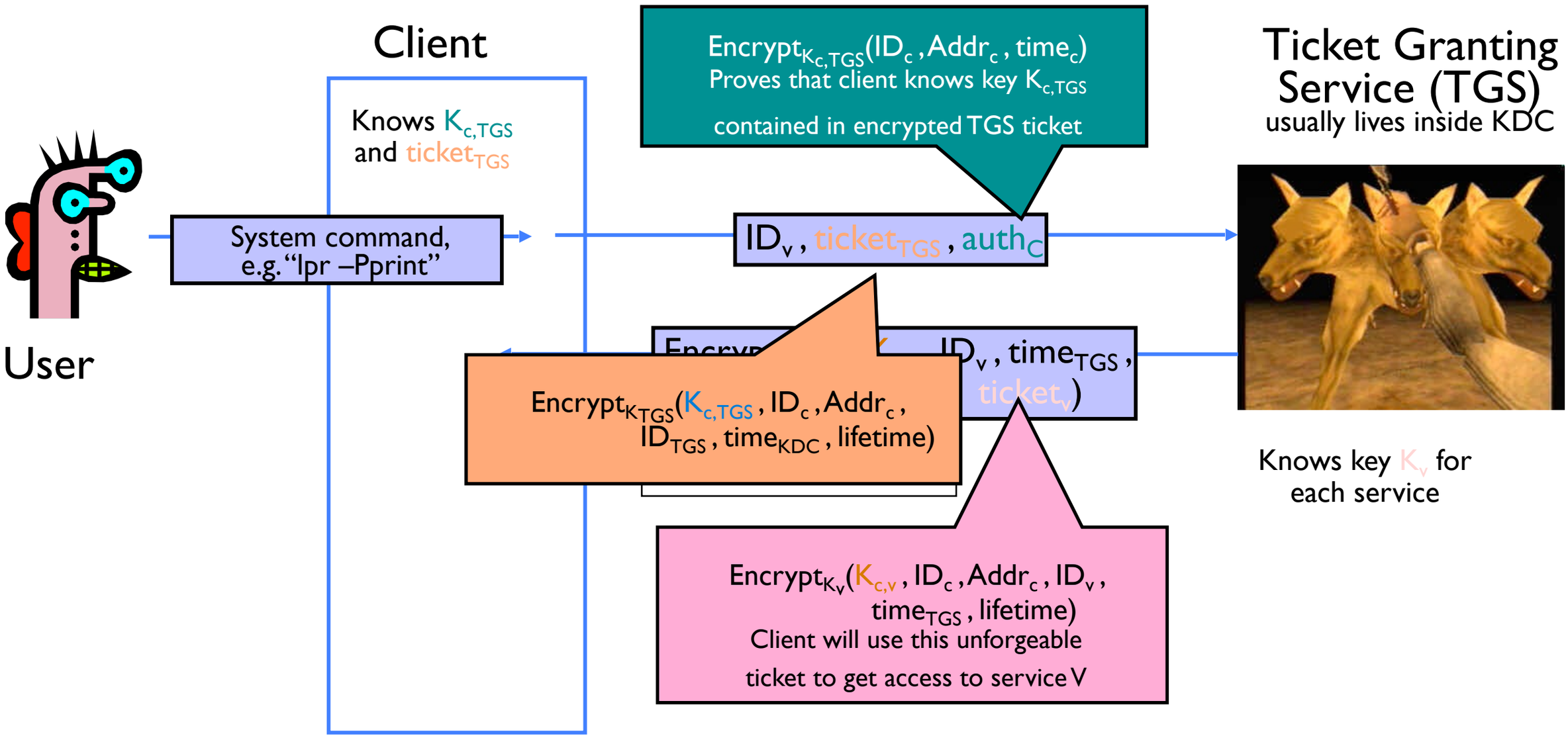


"Single Logon" Authentication



- Client only needs to obtain TGS ticket once (say, every morning)
  - Ticket is encrypted; client cannot forge it or tamper with it

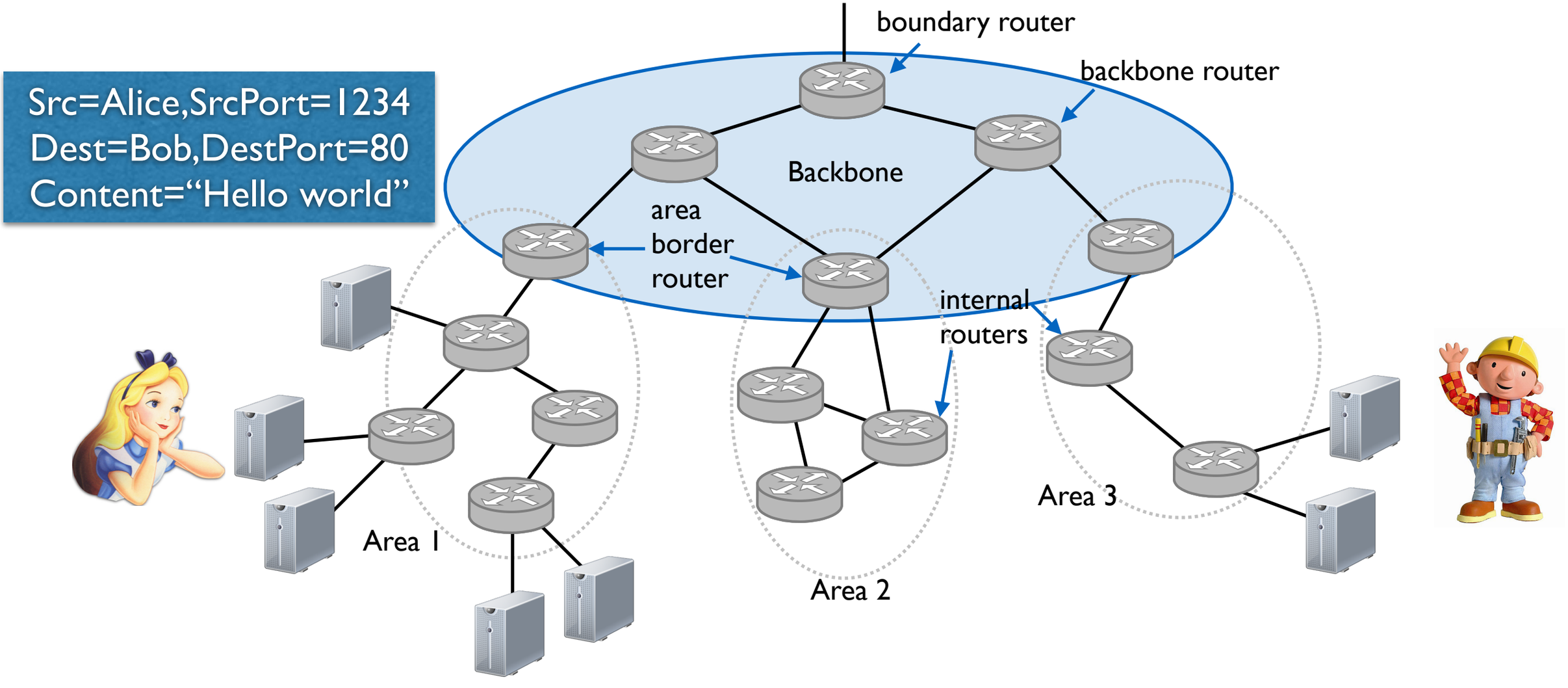
Obtaining a Service Ticket



- Client uses TGS ticket to obtain a service ticket and a short-term key for each network service
  - One encrypted, unforgeable ticket per service (printer, email, etc.)

What is the Internet?

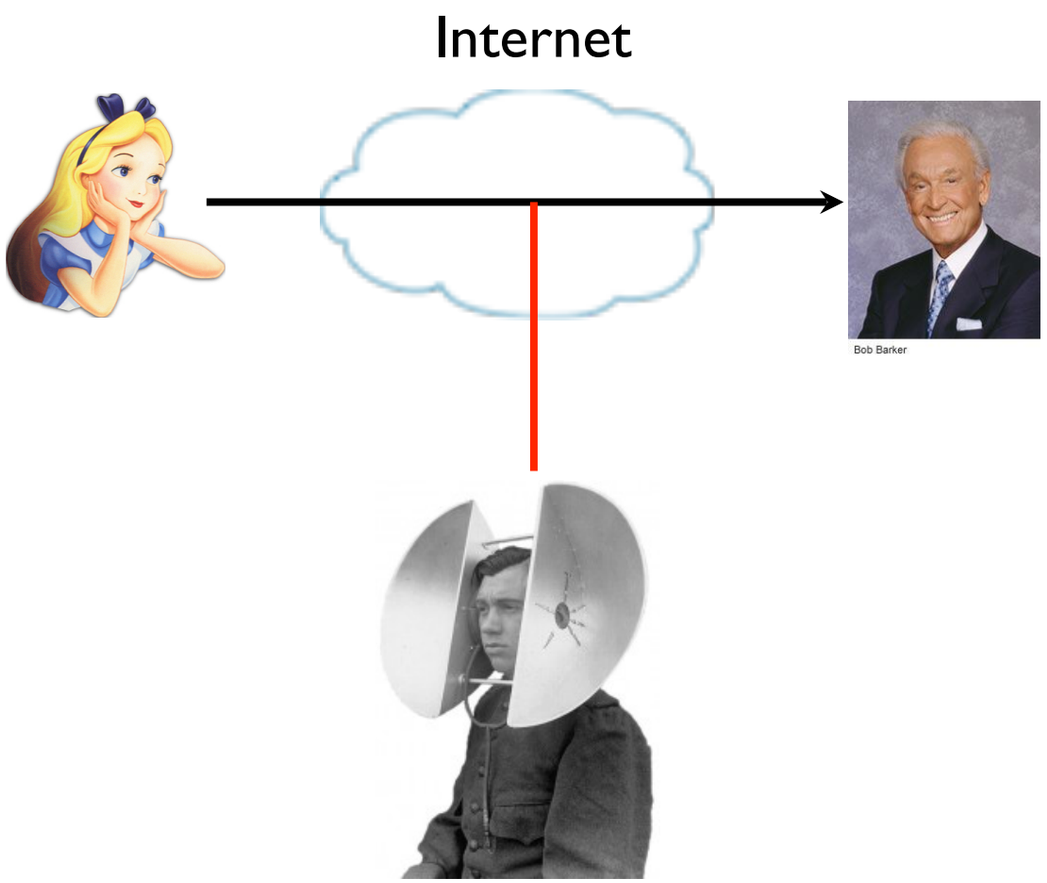
A collection of independently operated
autonomous systems (ASes)

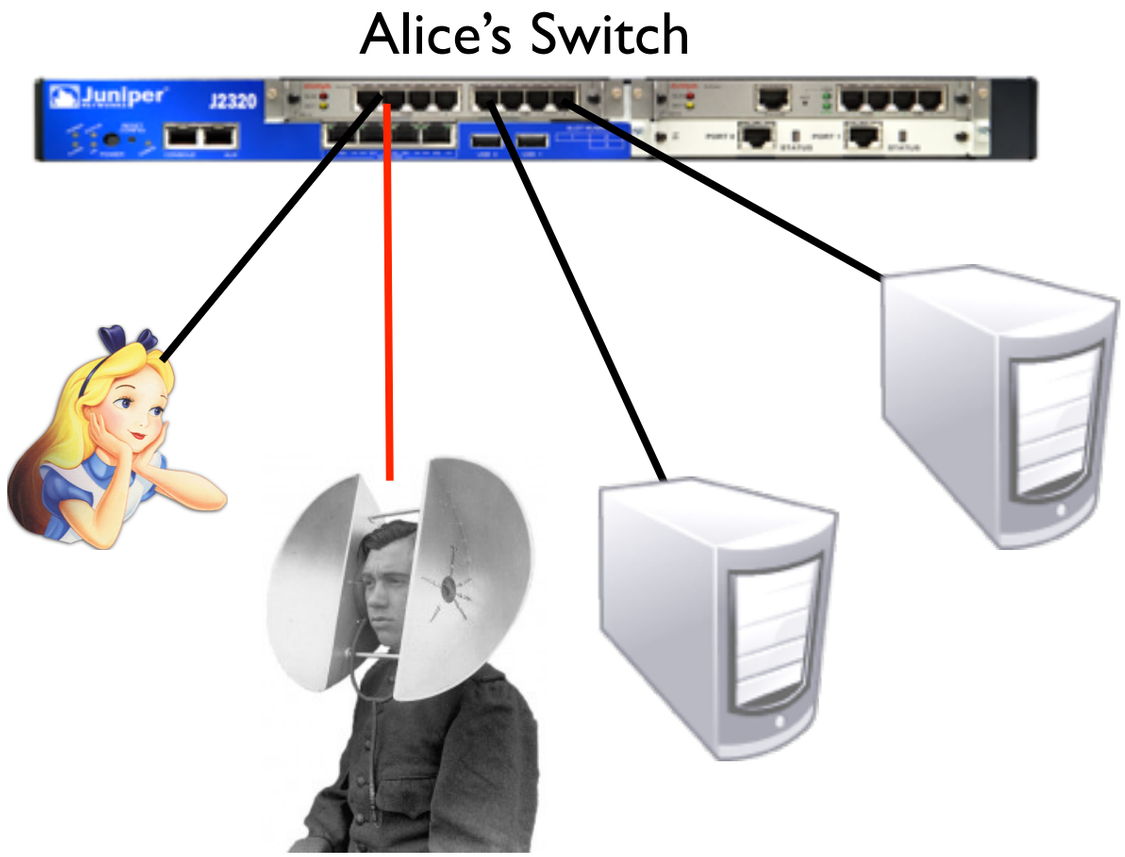


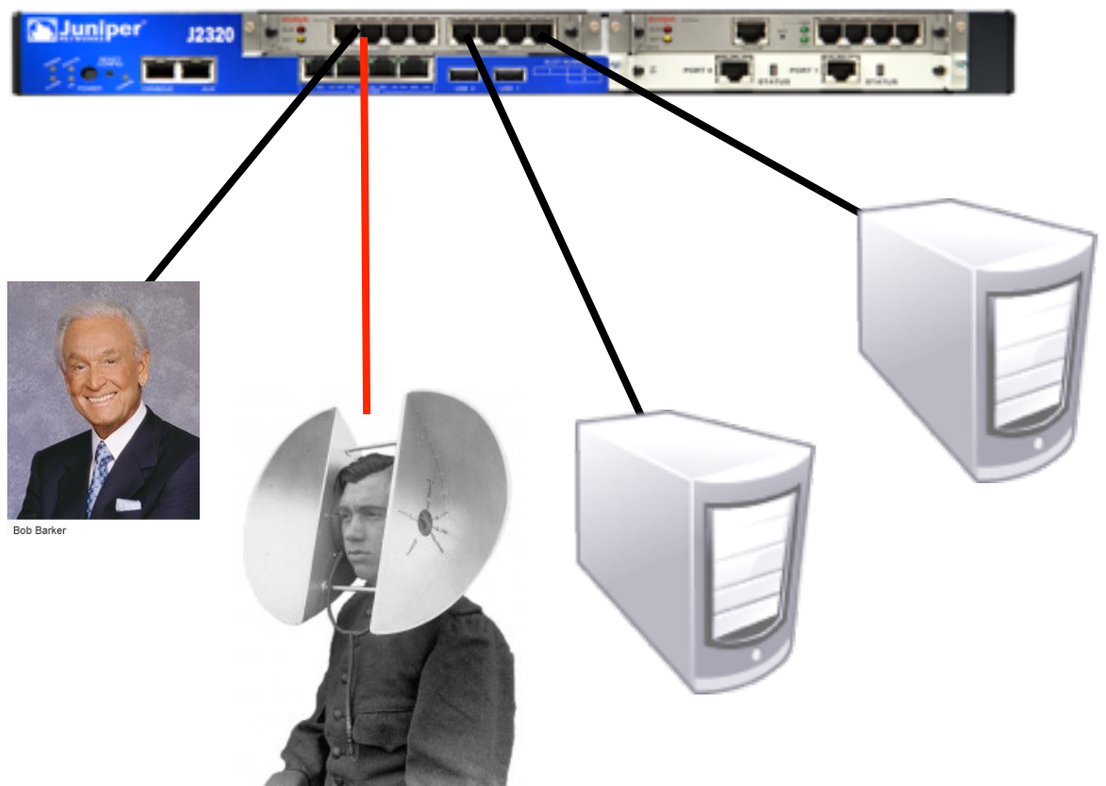
What about security?

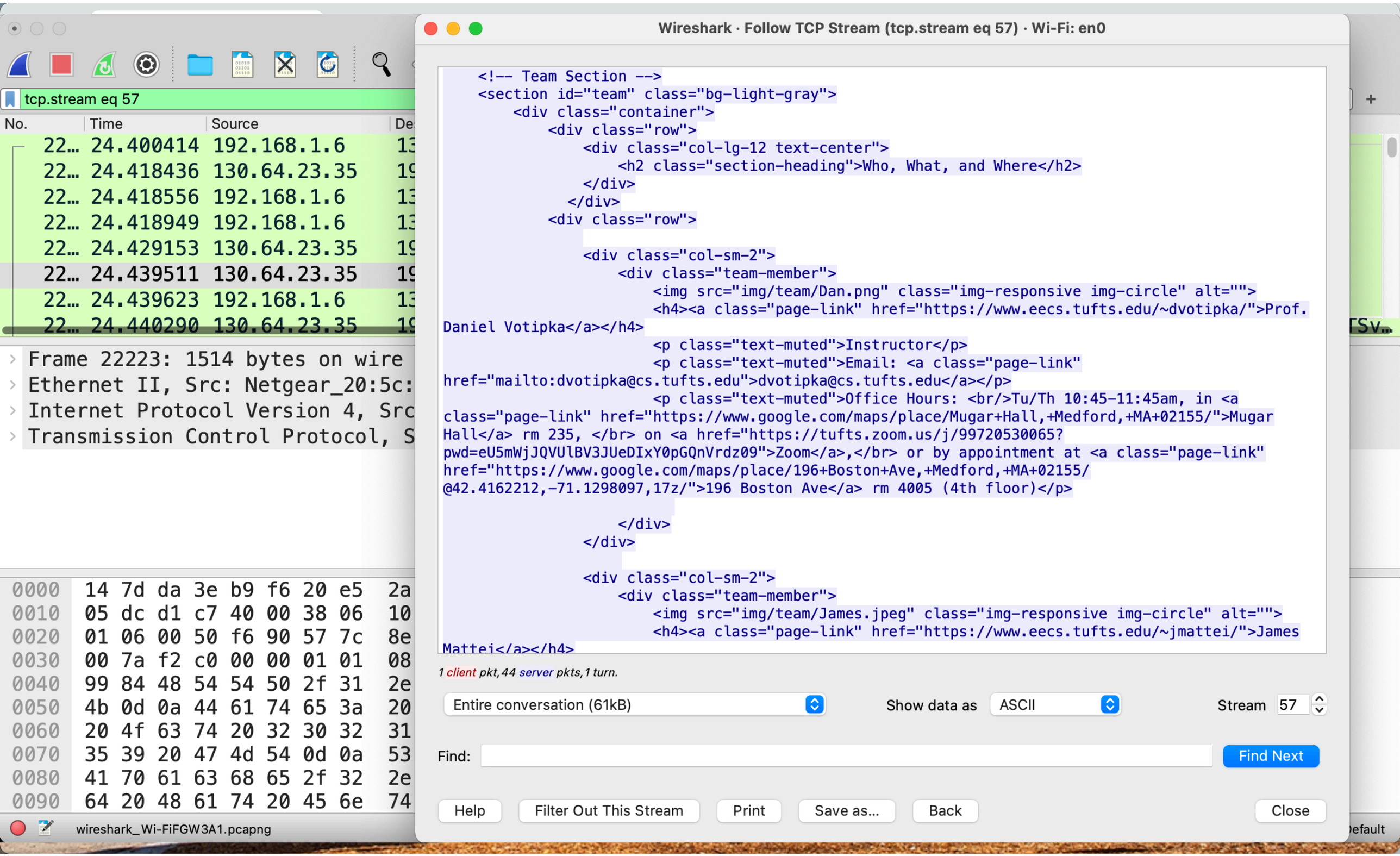
- Where is confidentiality and authenticity?
- No relevant "security" fields in IP, TCP, or UDP headers.

Eavesdropping








tcp.stream eq 57
No. Time Source
22… 24.400414 192.168.1.6
22… 24.418436 130.64.23.35
22… 24.418556 192.168.1.6
22… 24.418949 192.168.1.6
22… 24.429153 130.64.23.35
22… 24.439511 130.64.23.35
22… 24.439623 192.168.1.6
22… 24.440290 130.64.23.35
Frame 22223: 1514 bytes on wire
Ethernet II, Src: Netgear_20:5c:
Internet Protocol Version 4, Src
Transmission Control Protocol, S
0000 14 7d da 3e b9 f6 20 e5 2a
0010 05 dc d1 c7 40 00 38 06 10
0020 01 06 00 50 f6 90 57 7c 8e
0030 00 7a f2 c0 00 00 01 01 08
0040 99 84 48 54 54 50 2f 31 2e
0050 4b 0d 0a 44 61 74 65 3a 20
0060 20 4f 63 74 20 32 30 32 31
0070 35 39 20 47 4d 54 0d 0a 53
0080 41 70 61 63 68 65 2f 32 2e
0090 64 20 48 61 74 20 45 6e 74
wireshark_Wi-FiFGW3A1.pcapng
Wireshark · Follow TCP Stream (tcp.stream eq 57) · Wi-Fi: en0
<!-- Team Section -->
<section id="team" class="bg-light-gray">

<h2 class="section-heading">Who, What, and Where</h2>

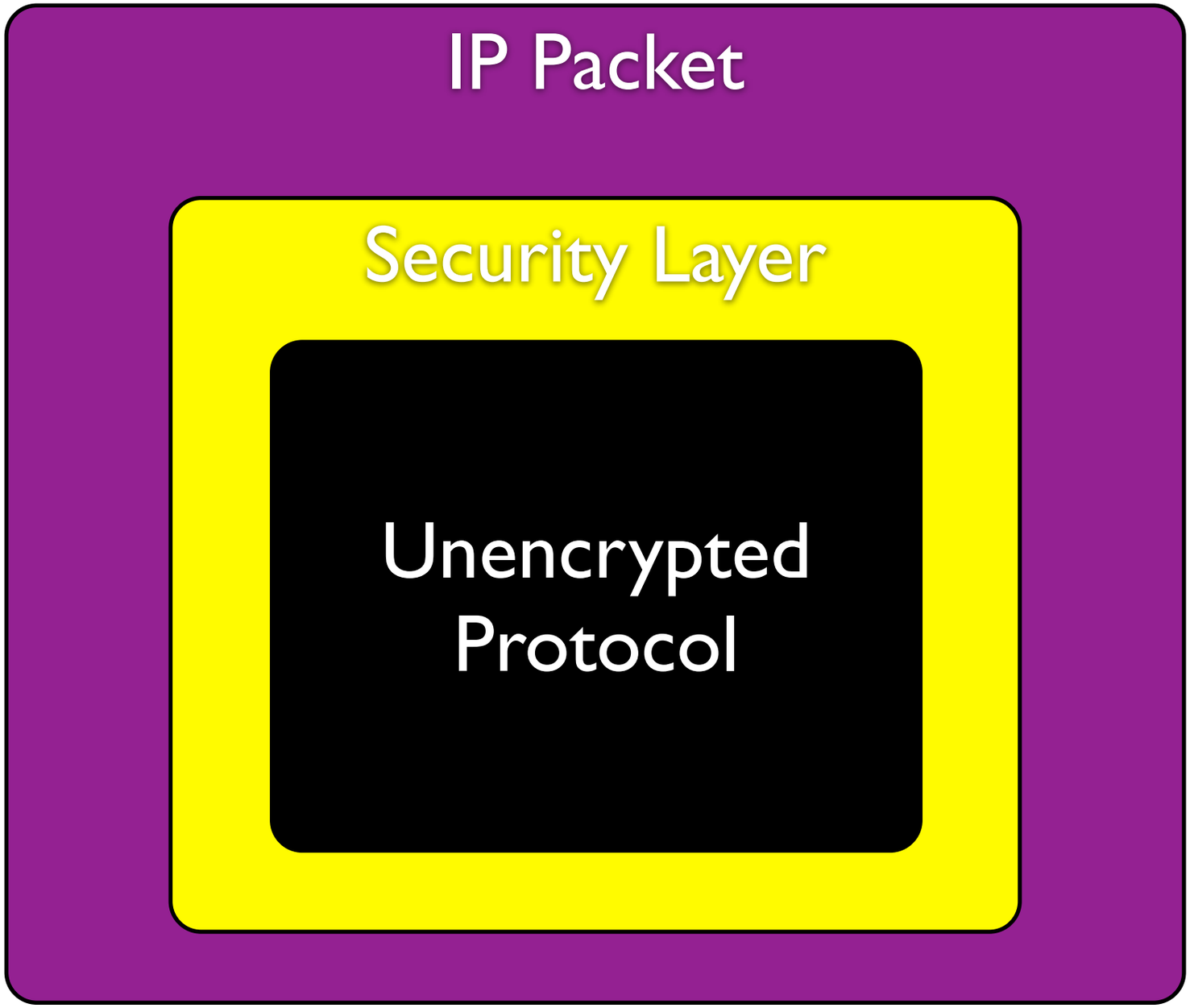
<img src="img/team/Dan.png" class="img-responsive img-circle" alt="">
<h4>Prof. Daniel Votipka</h4>
Instructor
Email: dvotipka@cs.tufts.edu
Office Hours:
Tu/Th 10:45-11:45am, in Mugar Hall rm 235, </br> on Zoom,</br> or by appointment at 196 Boston Ave rm 4005 (4th floor)

<img src="img/team/James.jpeg" class="img-responsive img-circle" alt="">
<h4>James Mattei</h4>
1 client pkt, 44 server pkts, 1 turn.
Entire conversation (61kB)
Show data as ASCII
Stream 57
Find:
Find Next
Help
Filter Out This Stream
Print
Save as...
Back
Close

Let's use that crypto stuff

- Let's build some new protocols
  - HTTP → SecureHTTP
  - POP → POPSecure
  - IMAP → C
  - SMTP → SM
  - 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
- Reliability

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 1.2, RFC 5246

https://tools.ietf.org/html/rfc5246

K.I.S.S.

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

SSL/TLS Message Types

- Handshake
- Alerts
- Change cipher spec
- Data

SSL/TLS Message Types

- Handshake
- Alerts
- Change cipher spec
- Data

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

Alice

Bob

Bob Barker

`ClientHello`, Version, Cipher list., R_{Alice}

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

$E_{Bob+}(S)$

$E_{K'}(Data)$

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

Encryption and integrity keys derived from Master Secret

Alice randomly chooses S,

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

Key exchange/agreement and authentication

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
RSA	Yes	Yes	Yes	Yes	Yes	No	
DH-RSA	No	Yes	Yes	Yes	Yes	No	
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes	
ECDH-RSA	No	No	Yes	Yes	Yes	No	
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
DH-DSS	No	Yes	Yes	Yes	Yes	No	
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[54]	
ECDH-ECDSA	No	No	Yes	Yes	Yes	No	
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
ECDH-EdDSA	No	No	Yes	Yes	Yes	No	
ECDHE-EdDSA (forward secrecy)[55]	No	No	Yes	Yes	Yes	Yes	Defined for TLS 1.2 in RFCs
PSK	No	No	Yes	Yes	Yes		
PSK-RSA	No	No	Yes	Yes	Yes		
DHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
ECDHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
SRP	No	No	Yes	Yes	Yes		
SRP-DSS	No	No	Yes	Yes	Yes		
SRP-RSA	No	No	Yes	Yes	Yes		
Kerberos	No	No	Yes	Yes	Yes		
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes		
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes		
GOST R 34.10-94 / 34.10-2001[56]	No	No	Yes	Yes	Yes		Proposed in RFC drafts

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

SSLv2 Problems

Alice

Bob

Bob Barker

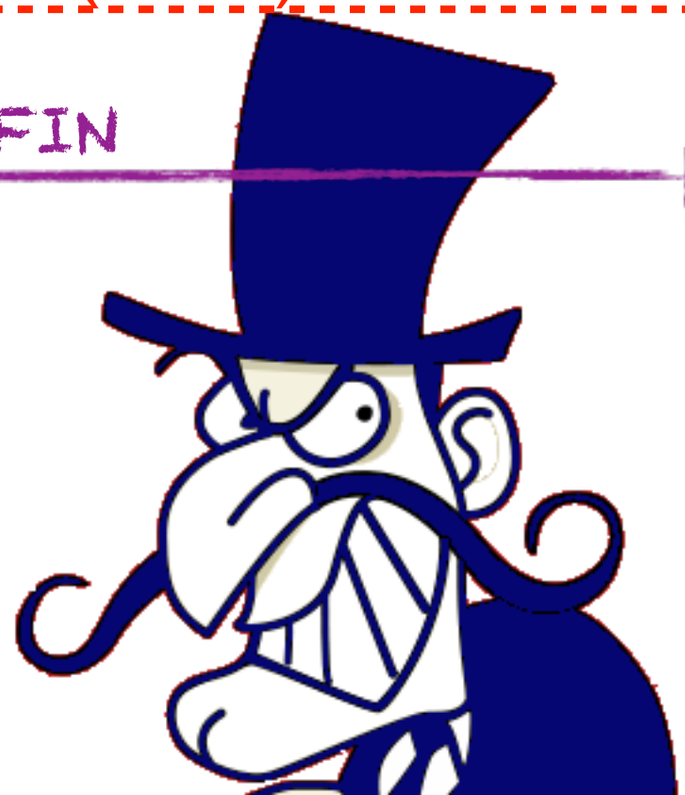
`ClientHello`, Version,
~~Cipher list~~ Weakest ciphers., R_{Alice}

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

$E_{Bob+}(S)$

$E_{K'}$(Data)

TCP FIN



SSLv3 Fixes

Alice

Bob

Bob Barker

`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)

$E_{K'}$(`Finish`)

Keyed hash over previous messages ensures integrity protection

SSL/TLS with Server and Client Authentication

Alice

Bob

Bob Barker

`ClientHello`, Version, Cipher list. R_{Alice}

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

`CertRequest`

$E_{Bob+}(S)$, $Cert_{Alice}$

$Sig(Alice-, h_K(\text{all prior handshake msgs}))$

$h_K(\text{keyed hash of handshake msgs})$

$E_{K'}(Data)$

$E_{K'}(\texttt{Finish})$

Signature proves Alice knows private key associated with her certificate

Handshake cost

- Per-session master secret derived using expensive public key crypto

```
[msherr@NotLinux 11:31 AM] ~> openssl speed rsa1024 aes-128-cbc
Doing aes-128 cbc for 3s on 16 size blocks: 27119786 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 64 size blocks: 7393395 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 256 size blocks: 1883302 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 1024 size blocks: 473817 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 8192 size blocks: 58941 aes-128 cbc's in 3.00s
Doing 1024 bit private rsa's for 10s: 6693 1024 bit private RSA's in 9.99s
Doing 1024 bit public rsa's for 10s: 137389 1024 bit public RSA's in 9.98s
OpenSSL 1.0.0a 1 Jun 2010
built on: Fri Jul 16 10:30:43 EDT 2010
options:bn(64,64) rc4(ptr,char) des(idx,cisc,16,int) aes(partial) idea(int) blowfish(idx)
compiler: /usr/bin/gcc-4.2 -fPIC -fno-common -DOPENSSL_PIC -DZLIB -DOPENSSL_THREADS
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -arch x86_64 -O3 -DL_ENDIAN -DMD32_REG_T=int -Wall
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128 cbc     145122.60k   157725.76k   160708.44k   161729.54k   160948.22k
                  sign    verify    sign/s verify/s
rsa 1024 bits 0.001493s 0.000073s    670.0  13766.4
```

HTTP 1.0

GET /index.html

In HTTP 1.0, each transaction requires a separate TCP connection (i.e., "session")

POST /login.cgi

Session Resumption

- Allows Alice and Bob to construct new encryption & integrity keys using previously shared pre-master secret (S)
  - uses *session-id* to continue SSL session over multiple connections
  - avoids having to repeat public-key crypto operations
- If either Alice or Bob don't remember master secret key, new handshake is required

SSLv3 Handshake

Alice

Bob

Bob Barker

`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)

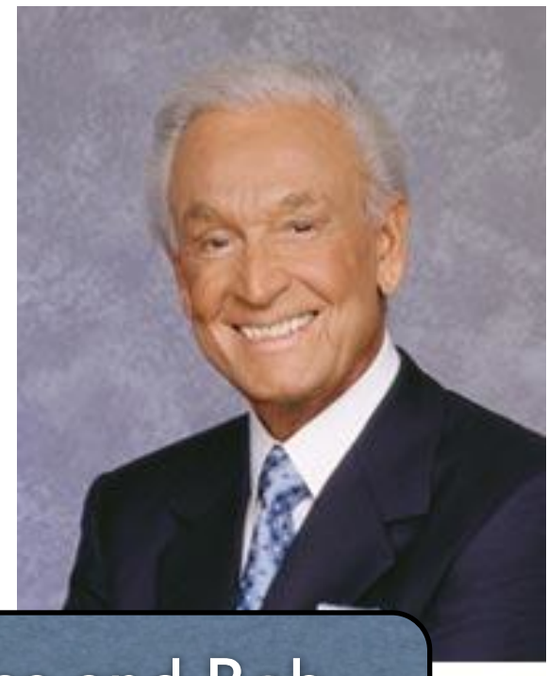
$E_{K'}$(Data)

$E_{K'}$(`Finish`)

Session Resumption

Alice

Bob



session-id, Cipher list, R_{Alice}

session-id, cipher, R_{Bob}

h_K(keyed hash of handshake msgs)

h_K(keyed hash of handshake msgs)

$E_{K'}$(Data)

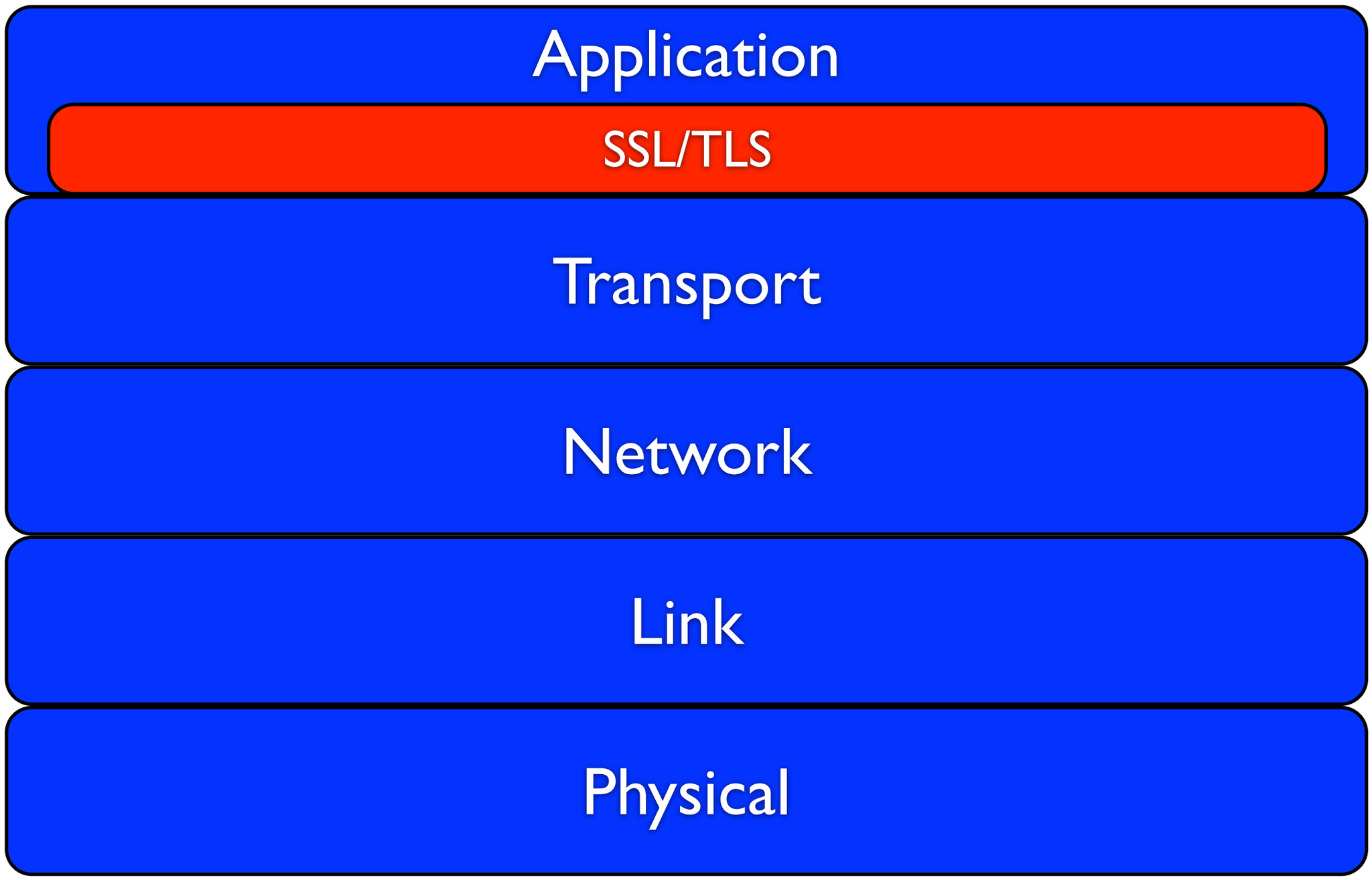
Alice and Bob compute new **master secret k** as $K'=h(S,R_{Alice},R_{Bob})$

Key exchange/agreement and authentication

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
RSA	Yes	Yes	Yes	Yes	Yes	No	
DH-RSA	No	Yes	Yes	Yes	Yes	No	
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes	
ECDH-RSA	No	No	Yes	Yes	Yes	No	
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
DH-DSS	No	Yes	Yes	Yes	Yes	No	
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[54]	
ECDH-ECDSA	No	No	Yes	Yes	Yes	No	
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
ECDH-EdDSA	No	No	Yes	Yes	Yes	No	
ECDHE-EdDSA (forward secrecy)[55]	No	No	Yes	Yes	Yes	Yes	Defined for TLS 1.2 in RFCs
PSK	No	No	Yes	Yes	Yes		
PSK-RSA	No	No	Yes	Yes	Yes		
DHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
ECDHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
SRP	No	No	Yes	Yes	Yes		
SRP-DSS	No	No	Yes	Yes	Yes		
SRP-RSA	No	No	Yes	Yes	Yes		
Kerberos	No	No	Yes	Yes	Yes		
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes		
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes		
GOST R 34.10-94 / 34.10-2001[56]	No	No	Yes	Yes	Yes		Proposed in RFC drafts

SSL/TLS in the Real World

Network Stack, revisited

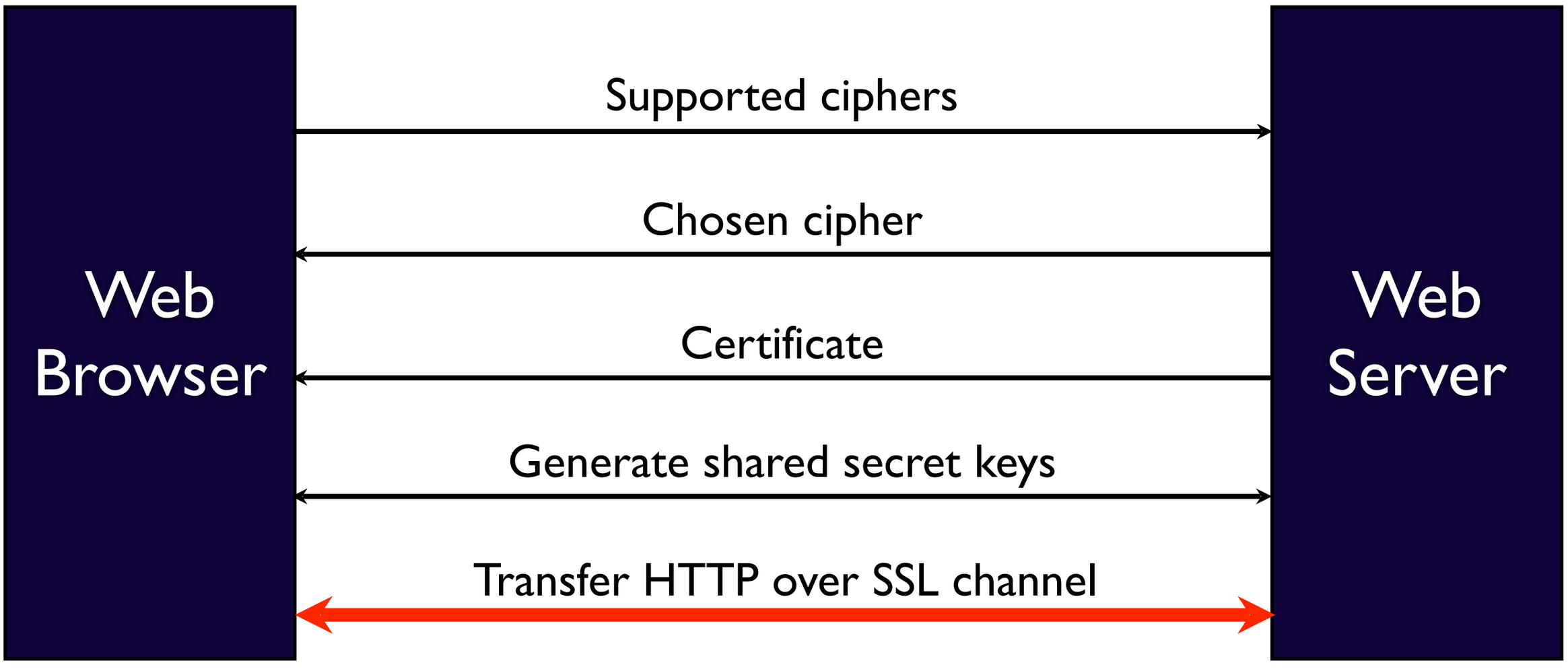


SSL/TLS in the Real World

- Most (modern) browsers support SSLv3, TLS 1.2
- 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 



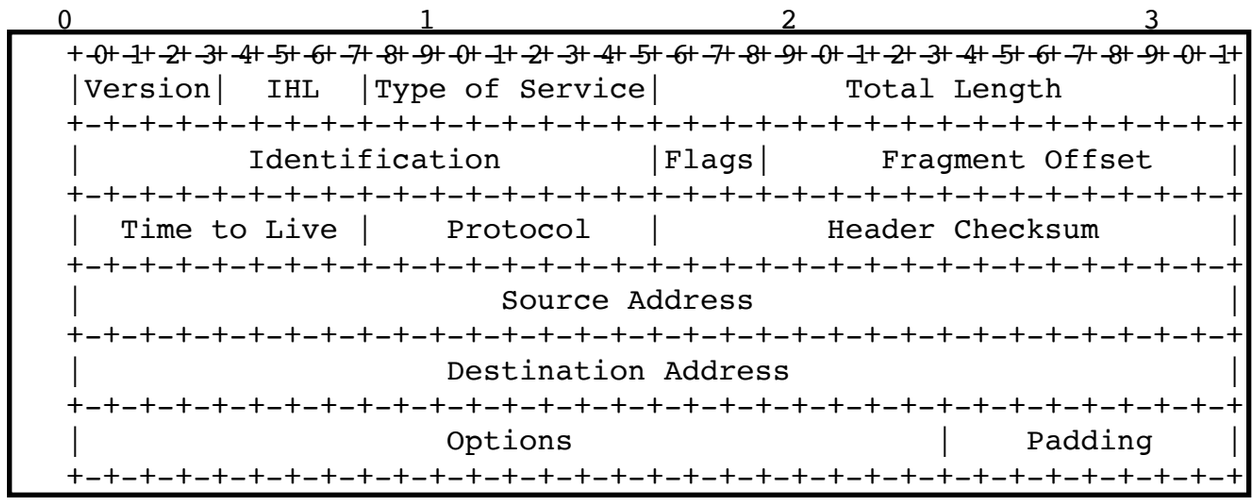
foo.py

New Open Recent Revert Save Print Undo Redo Cut Copy Paste Search Preferences Help

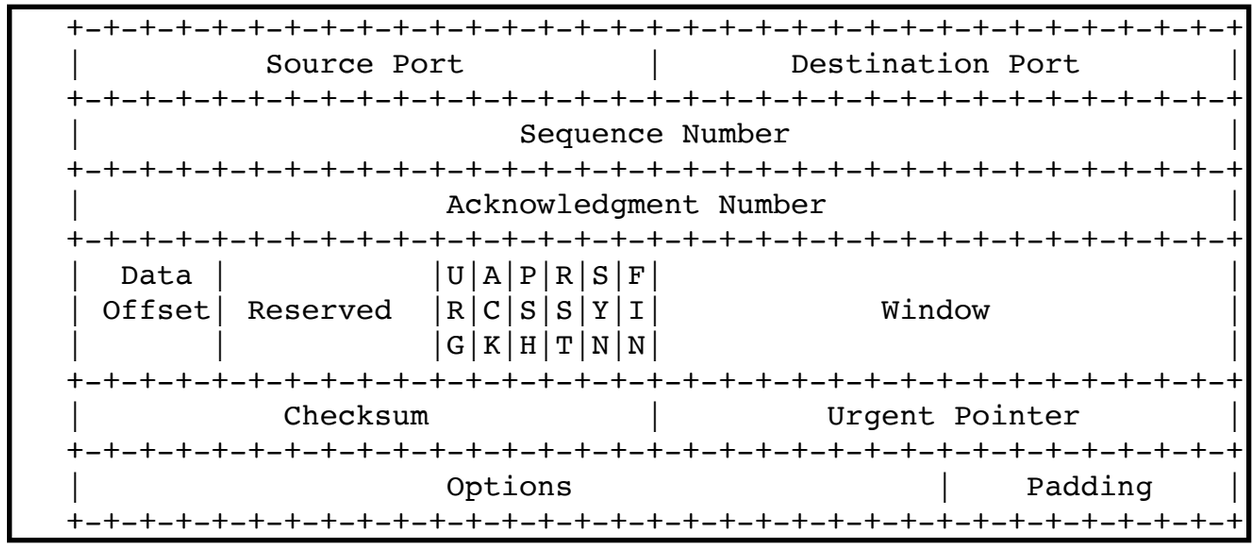
```
import ssl
import socket

dst = “cs.tufts.edu/cs/114”

s = socket.socket(socket.AF_INET)
context = ssl.create_default_context()
conn = context.wrap_socket(s, server_hostname=dst)
conn.connect((dst, 443))
```

U:--- **foo.py** All (10,0) (Py Outl)

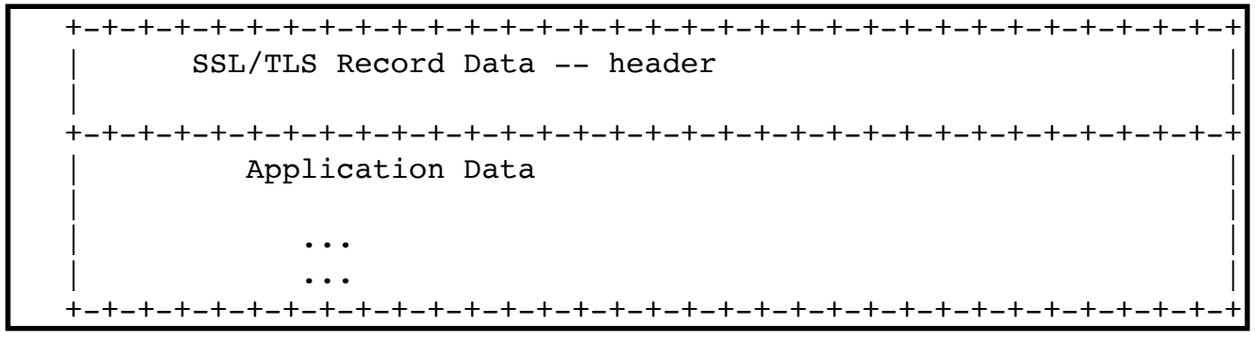
Let's look at one of those packet things



IP Header:
At least 20 bytes



TCP Header:
At least 20 bytes



TLS Header & Application Data

ssl.pcap [Wireshark 1.6.2]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

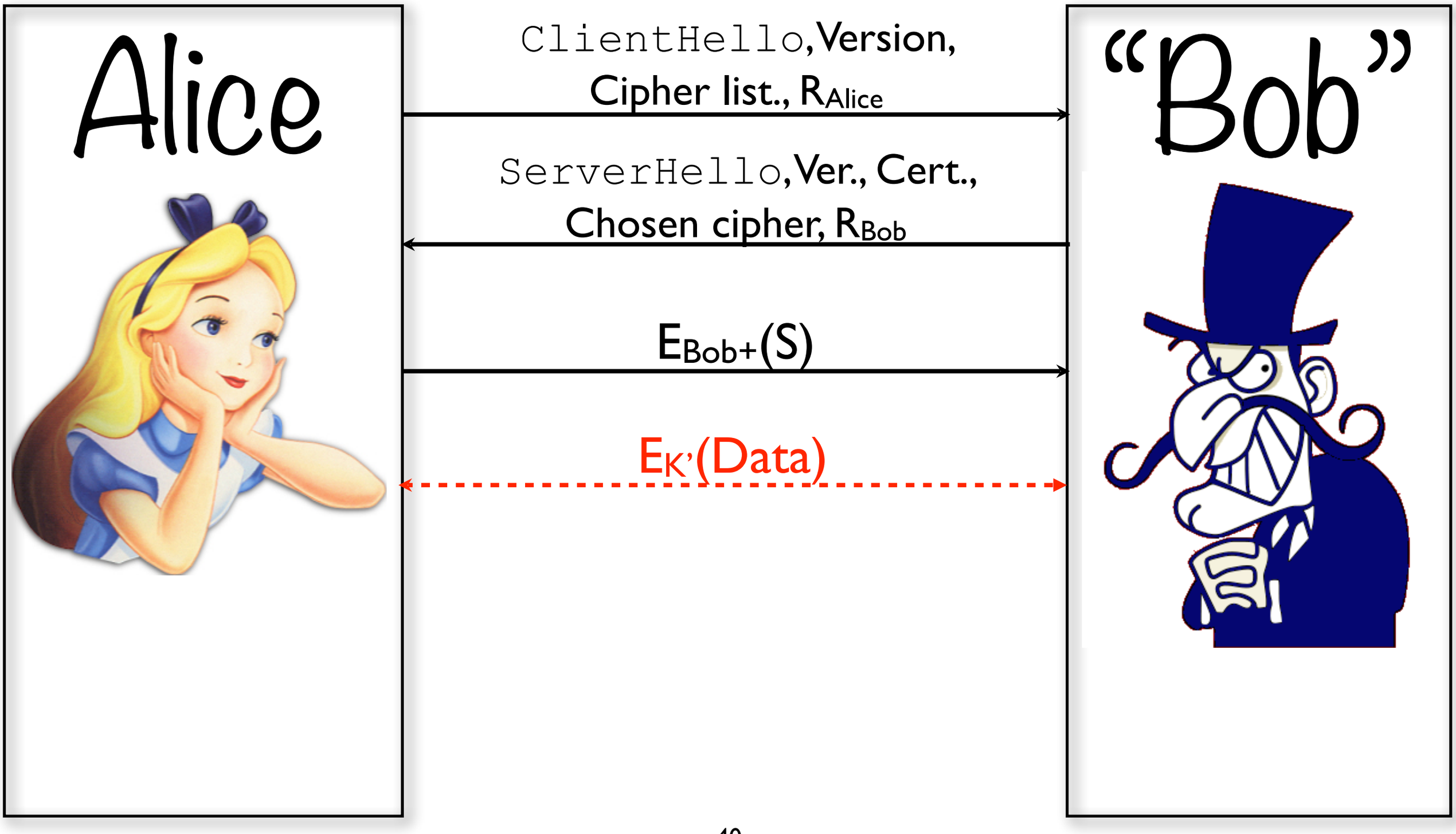
Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	141.161.20.42	141.161.20.48	TCP	78	56324 > https [SYN] Seq=0 Win=65535 Len=0 MSS=1460 W
2	0.001292	141.161.20.48	141.161.20.42	TCP	74	https > 56324 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0
3	0.001326	141.161.20.42	141.161.20.48	TCP	66	56324 > https [ACK] Seq=1 Ack=1 Win=524280 Len=0 TSv
4	0.002661	141.161.20.42	141.161.20.48	TLSv1	255	Client Hello
5	0.003212	141.161.20.48	141.161.20.42	TCP	66	https > 56324 [ACK] Seq=1 Ack=190 Win=15616 Len=0 TS
6	0.013737	141.161.20.48	141.161.20.42	TLSv1	1514	Server Hello

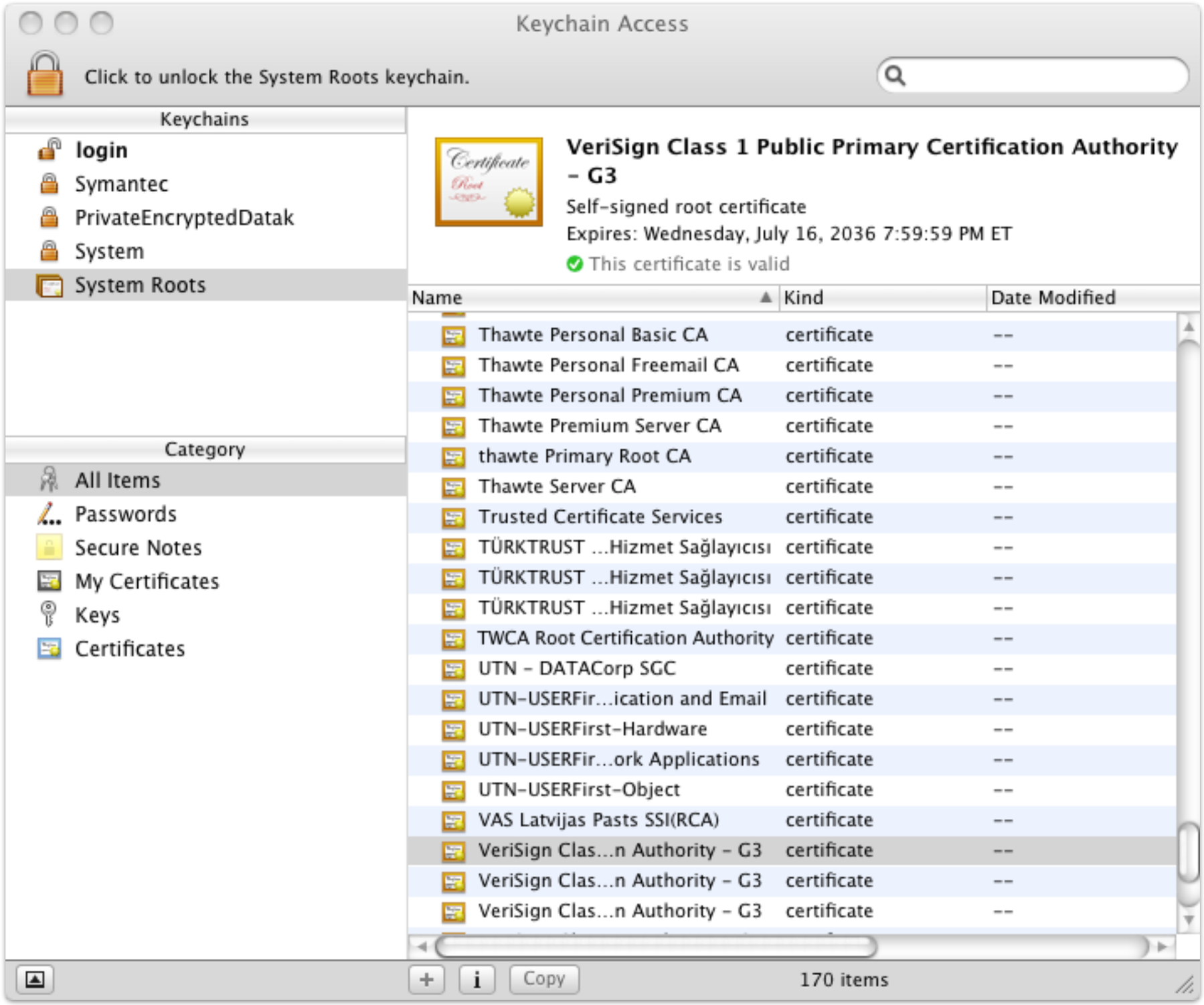
```
▷ Frame 4: 255 bytes on wire (2040 bits), 255 bytes captured (2040 bits)
▷ Ethernet II, Src: Apple_06:e3:44 (d4:9a:20:06:e3:44), Dst: Dell_5b:5a:bf (00:13:72:5b:5a:bf)
▽ Internet Protocol Version 4, Src: 141.161.20.42 (141.161.20.42), Dst: 141.161.20.48 (141.161.20.48)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 241
    Identification: 0x8632 (34354)
  ▷ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
  ▷ Header checksum: 0x7038 [correct]
    Source: 141.161.20.42 (141.161.20.42)
    Destination: 141.161.20.48 (141.161.20.48)
▽ Transmission Control Protocol, Src Port: 56324 (56324), Dst Port: https (443), Seq: 1, Ack: 1, Len: 189
    Source port: 56324 (56324)
    Destination port: https (443)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 190    (relative sequence number)]
    Acknowledgement number: 1    (relative ack number)
    Header length: 32 bytes
  ▷ Flags: 0x18 (PSH, ACK)
    Window size value: 65535
    [Calculated window size: 524280]
    [Window size scaling factor: 8]
```

Problems with TLS/SSL

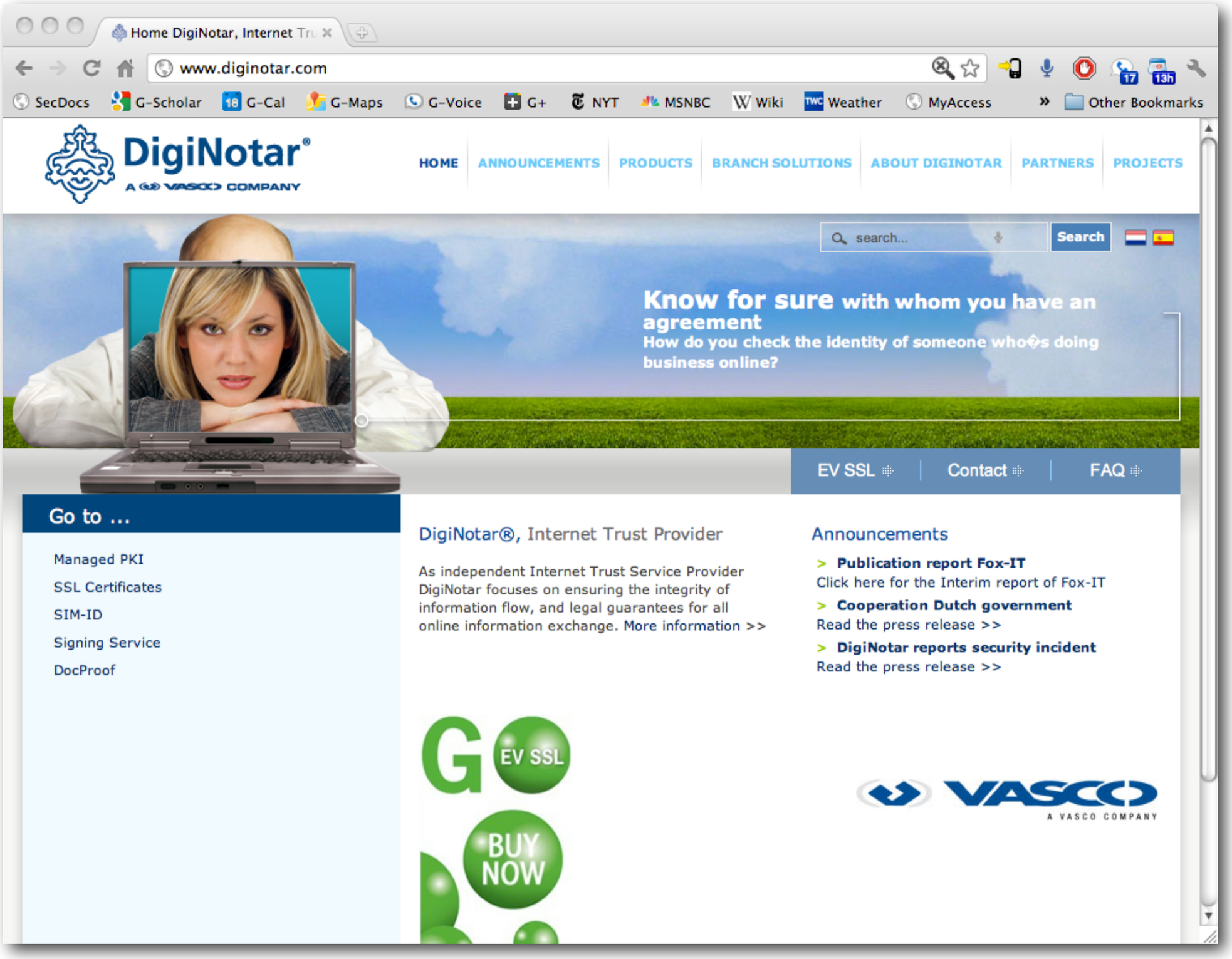
If Bob's cert isn't verified, how do you know you're actually talking to Bob?



Solution: Use a PKI



The 2011 DigiNotar Incident



Exploiting the network

- The Internet is extremely vulnerable to attack
  - it is a huge open system ...
    - which adheres to the end-to-end principle
      - smart end-points, dumb network
- Can you think of any large-scale attacks that would be enabled by this setup?