CS 370: Introduction to Security

05.02: SSL and TLS

Tu/Th 4:00 – 5:50 PM

Sanghyun Hong

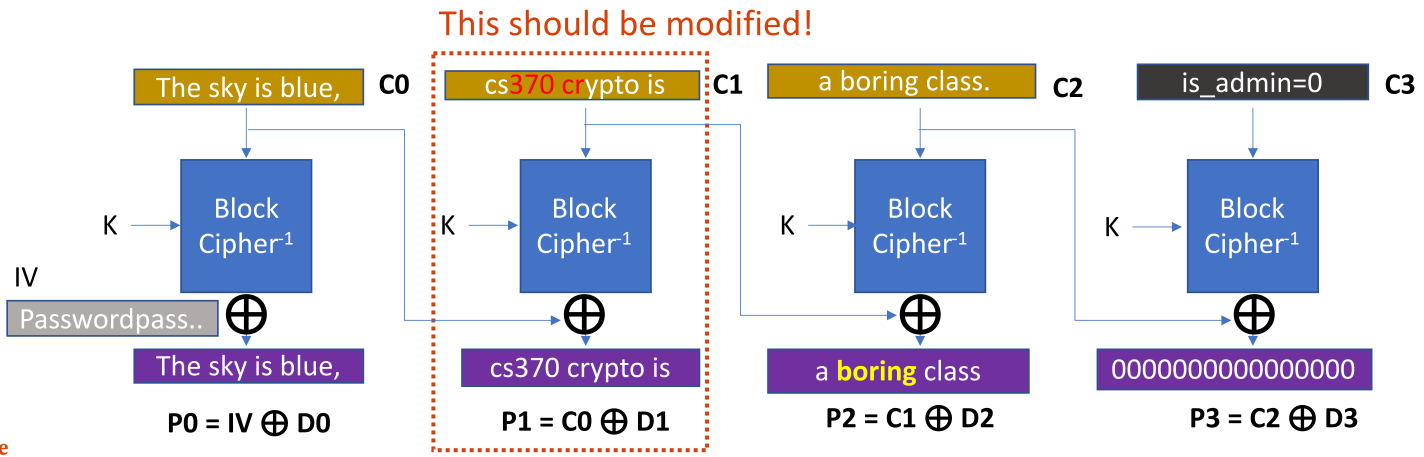
sanghyun.hong@oregonstate.edu

Oregon State University

SAIL
Secure AI Systems Lab

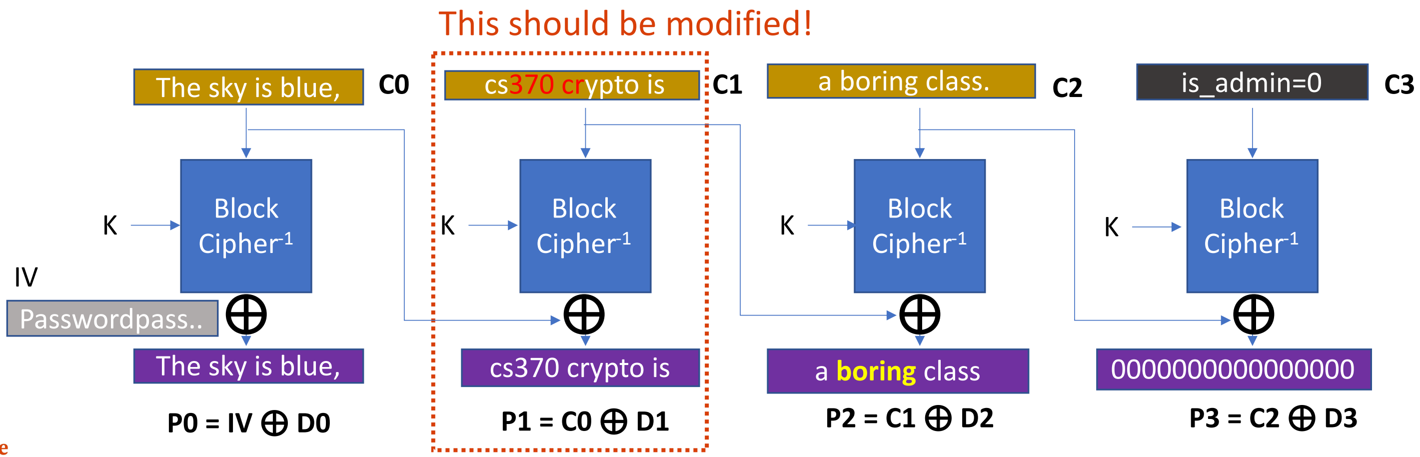
MICRO-LABS: CBC-ATTACK: BORING TO SUPERB

- Job 3
  - Create a copy of this data with
  - The change from ‘boring’ to ‘superb’
  - Use template.py (marked as XXX)
- Hint
  - Find a way to modify the plaintext of the 5th block



Micro-labs: CBC-attack: boring to superb

- We have
  - C1 ^ D2 = ‘boring’
  - C1’ ^ D2 = ‘superb’
  - where C1’ is the modified ciphertext we want



Micro-labs: CBC-attack: boring to superb

- We have
  - C1 ^ D2 = ‘boring’
  - C1’ ^ D2 = ‘superb’
  - where C1’ is the modified ciphertext we want

- Let’s XOR these two:
  - C1 ^ D2 ^ C1’ ^ D2 = ‘boring’ ^ ‘superb’ (we know XOR is associative)
  - C1 ^ C1’ ^ (D2 ^ D2) = ‘boring’ ^ ‘superb’ (we know a ^ a = 0)
  - C1 ^ C1’ ^ 0 = ‘boring’ ^ ‘superb’ (we know a ^ 0 = a)
  - C1 ^ (C1 ^ C1’) = C1 ^ (‘boring’ ^ ‘superb’)
  - (C1 ^ C1) ^ C1’ = C1 ^ (‘boring’ ^ ‘superb’)
  - C1’ = C1 ^ (‘boring’ ^ ‘superb’)

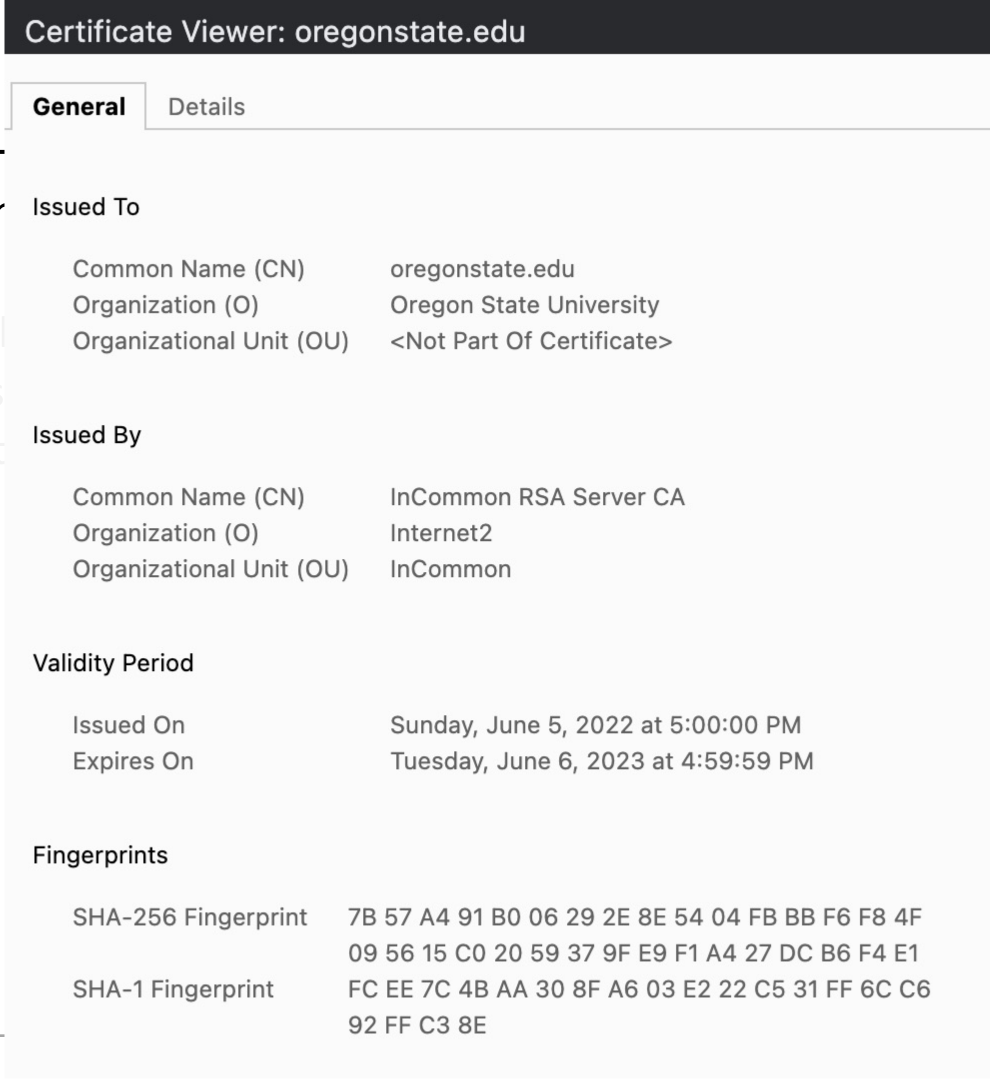
Recap

- How can we secure the Internet communication?
  - How can we make sure we're talking to the right person?
  - How can we establish a secure channel over an insecure channel?
  - How can we encrypt/decrypt the message we send/receive?
  - How can we ensure the message is not altered by an adversary?

RECAP: DIGITAL CERTIFICATE

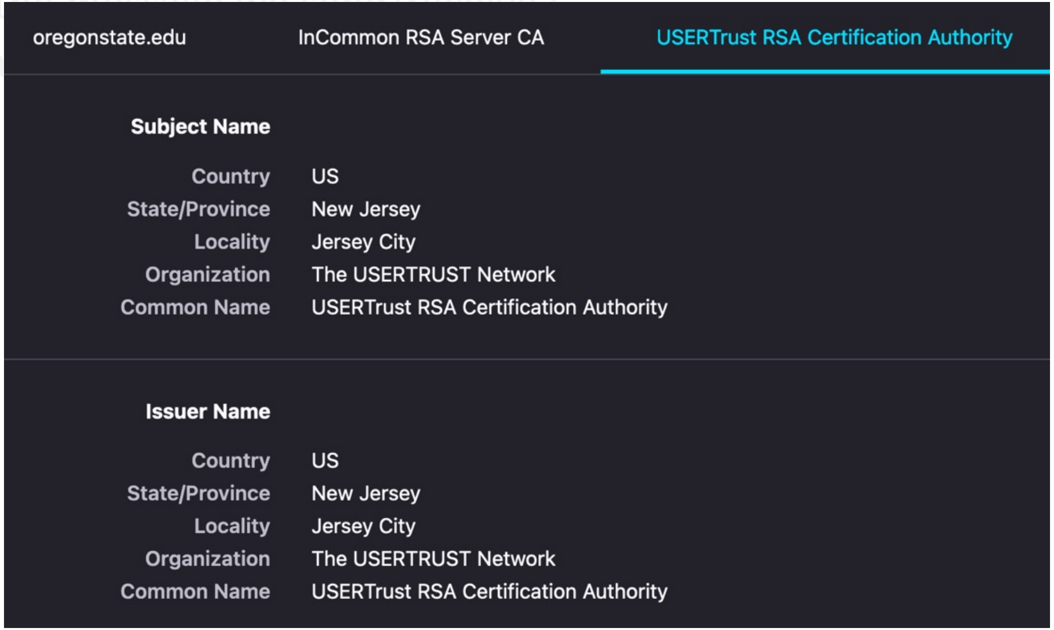
- How can we secure the Internet comm
  - Authentication: a digital certificate
  - How can we establish a secure channel
  - How can we encrypt/decrypt the mess
  - How can we ensure the message is not

- A file that contains:
  - Entity info (CN)
  - Issuer info (CN)
  - Public key
  - Signature



Recap: digital certificate – trust chain

- How can we secure the Internet communication?
  - Authentication: a digital certificate
  - How can we establish a secure channel over an insecure channel?
  - How can we encrypt/decrypt t
  - How can we ensure the messa

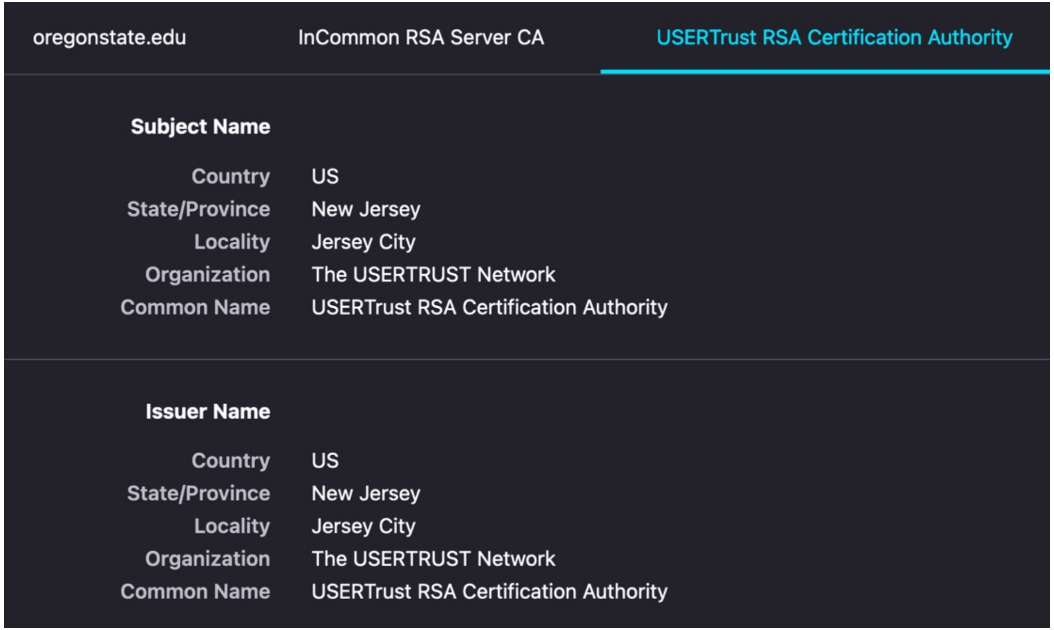


- A file that contains:
  - Entity info (CN)
  - Issuer info (CN)
  - Public key
  - Signature
- Public-key infrastructure (PKI)

Recap: digital certificate – trust chain (cont'd)

- How can we secure the Internet communication?
  - Authentication: a digital certificate

- Public-key infrastructure (PKI)
  - oregonstate.edu
    - Verified by InCommon RSA
  - InCommon RSA Server CA
    - Verified by USERTrust RSA
  - USERTrust RSA CA
    - Verified by itself (Root CA)



Recap

- How can we secure the Internet communication?
  - Authentication: a digital certificate
  - How can we establish a secure channel over an insecure channel?
  - How can we encrypt/decrypt the message we send/receive?
  - How can we ensure the message is not altered by an adversary?

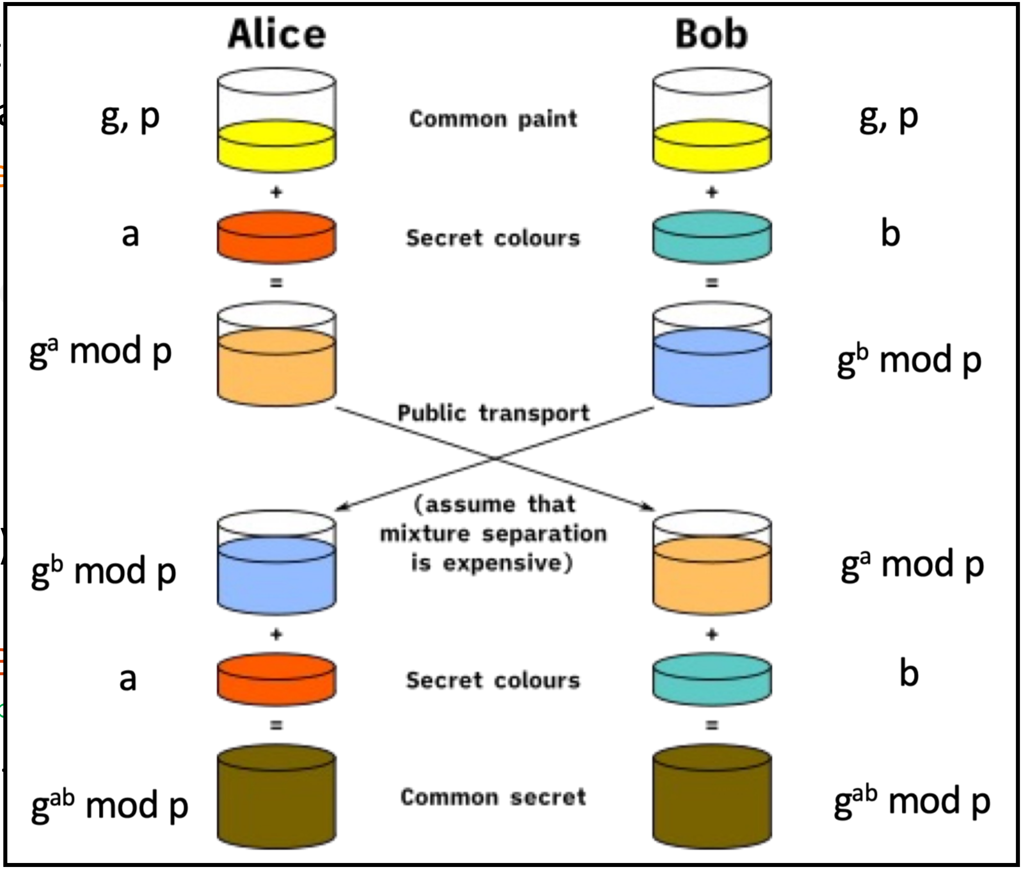
Recap: Diffie-Hellman

- How can we secure the Internet communication?
  - Authentication: a digital certificate
  - Key-exchange: Diffie-Hellman key exchange
  - How can we encrypt/decrypt the message we send/receive?
  - How can we ensure the message is not altered by an adversary?

- Diffie-Hellman
  - Given:
    - g and P (shared secret; public) and a and b (secrets; private)
  - Compute:
    - g^a mod p = A and g^b mod p = B and exchange them
    - $(g^b)^a$ mod p = $(g^a)^b$ mod p = g^{ab} mod p
  - Use g^{ab} mod p as a shared secret

RECAP: DIFFIE-HELLMAN

- How can we secure the Internet
  - Authentication: a digital certifica
  - Key-exchange: Diffie-Hellman ke
  - How can we encrypt/decrypt th
  - How can we ensure the messag



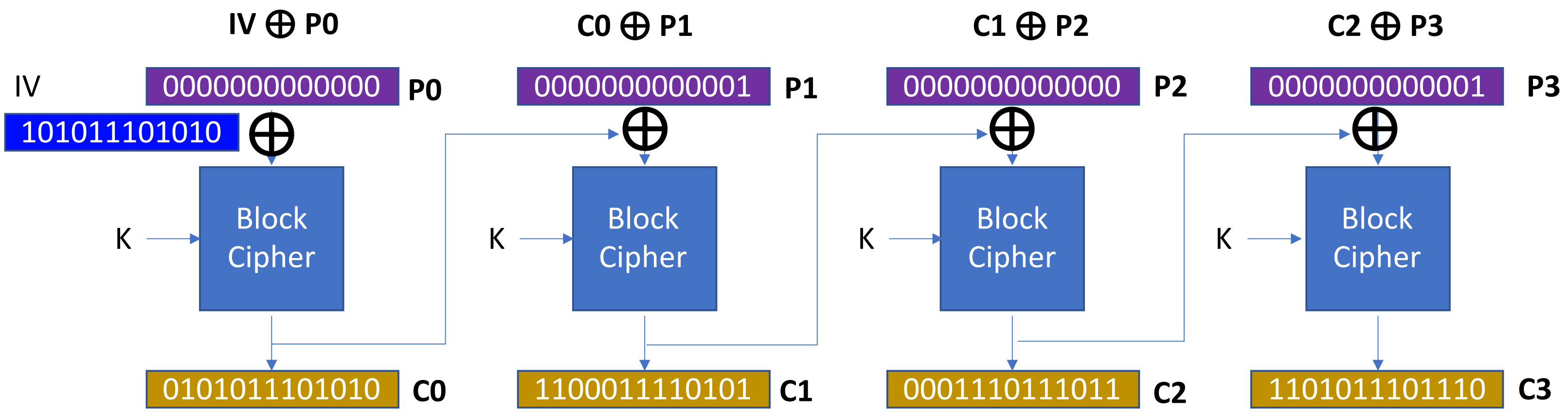
- Diffie-Hellman
  - Given:
    - g and P (shared secret; public)
  - Compute:
    - g^a mod p = A and g^b mod p = B
    - $(g^b)^a$ mod p = $(g^a)^b$ mod p = g^{ab}
  - Use g^{ab} mod p as a shared secre

Recap

- How can we secure the Internet communication?
  - Authentication: a digital certificate
  - Key-exchange: Diffie-Hellman key exchange
  - How can we encrypt/decrypt the message we send/receive?
  - How can we ensure the message is not altered by an adversary?

Recap: block cipher

- How can we secure the Internet communication?
  - Authentication: a digital certificate
  - Key-exchange: Diffie-Hellman key exchange
  - Confidentiality: ex. CBC with SHA-256('cipher key' + gab mod p) as a key
  - How can we ensure the message is not altered by an adversary?



Recap: block cipher

- How can we secure the Internet communication?
  - Authentication: a digital certificate
  - Key-exchange: Diffie-Hellman key exchange
  - Confidentiality: ex. CBC with SHA-256(‘cipher key’ + gab mod p) as a key
  - How can we ensure the message is not altered by an adversary?

Recap: block cipher

- How can we secure the Internet communication?
  - Authentication: a digital certificate
  - Key-exchange: Diffie-Hellman key exchange
  - Confidentiality: ex. CBC with SHA-256(‘cipher key’ + gab mod p) as a key
  - Integrity: ex. SHA-256(‘MAC key’ + g^{ab} mod p) as the key for HMAC

IV	Block 0	Block 1	HMAC (key || IV+Block0+Block1)

- HMAC = SHA-256(SHA-256(‘MAC key’ + g^{ab} mod p) || IV+Block0+Block1)
- : 7624e1f89ce009f8ec7e6e39781a42c0a27fa38f94db4f05f78b0f301007e06a

RECAP

- How can we secure the Internet communication?
  - Authentication: a digital certificate
  - Key-exchange: Diffie-Hellman key exchange
  - Confidentiality: ex. CBC with SHA-256(‘cipher key’ + gab mod p) as a key
  - Integrity: ex. SHA-256(‘MAC key’ + g^{ab} mod p) as the key for HMAC

A Communication Channel with Authenticity, Confidentiality, and Integrity!

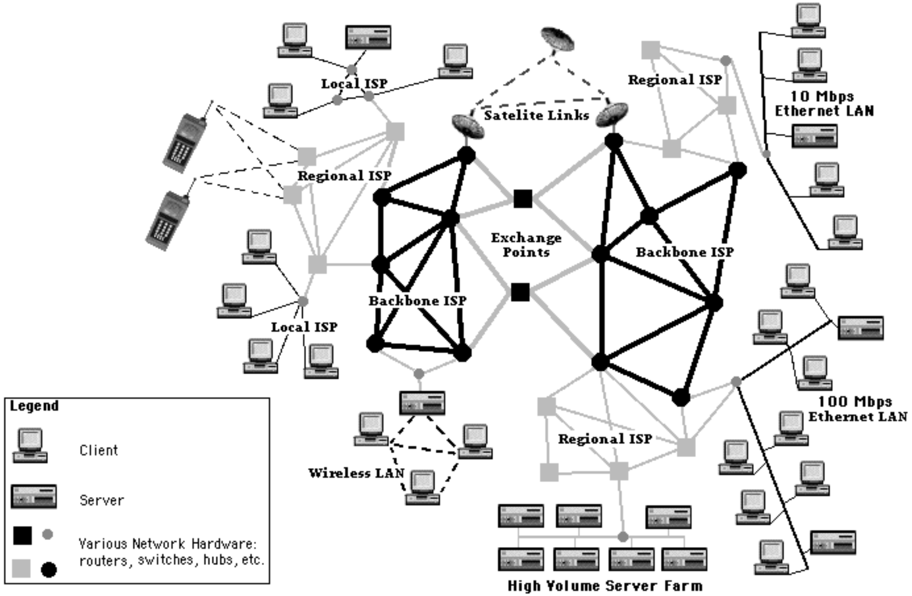
Topics for today

- SSL and TLS security
  - How secure is the Internet?
  - How can we implement secure communication channels?
  - How can we establish such channels between two parties?
  - How can we minimize the impact of security incidents?
  - How do we use to achieve such a goal (in practice)?

The Internet

- The Net
  - A system of computer networks; a network of networks
  - Uses the Internet protocol suite (TCP/IP) to communicate

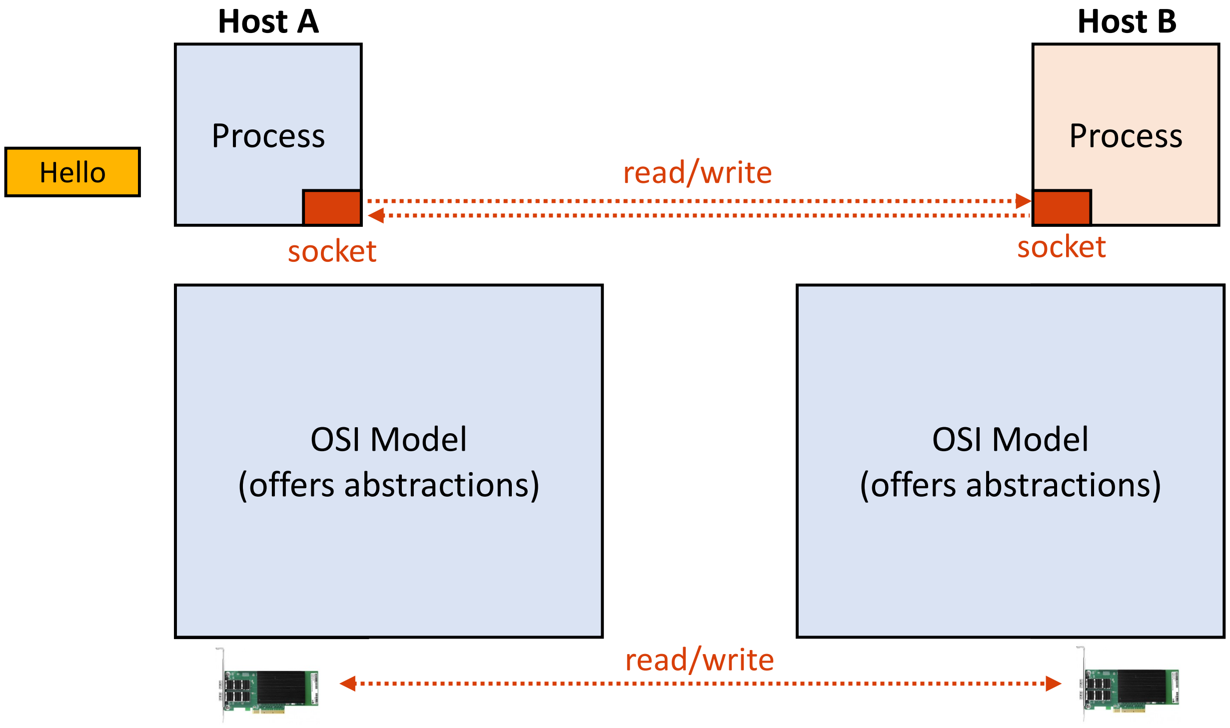
- Design principle
  - Network is complex, $O(N^2)$
  - Manage small network, $O(n^2)$
  - Manage network of networks $O(m^2)$
  - N >>>>> m,n
  - Make it simple!



[1]https://www.cs.utexas.edu/~mitra/csFall2018/cs329/lectures/fig1.gif

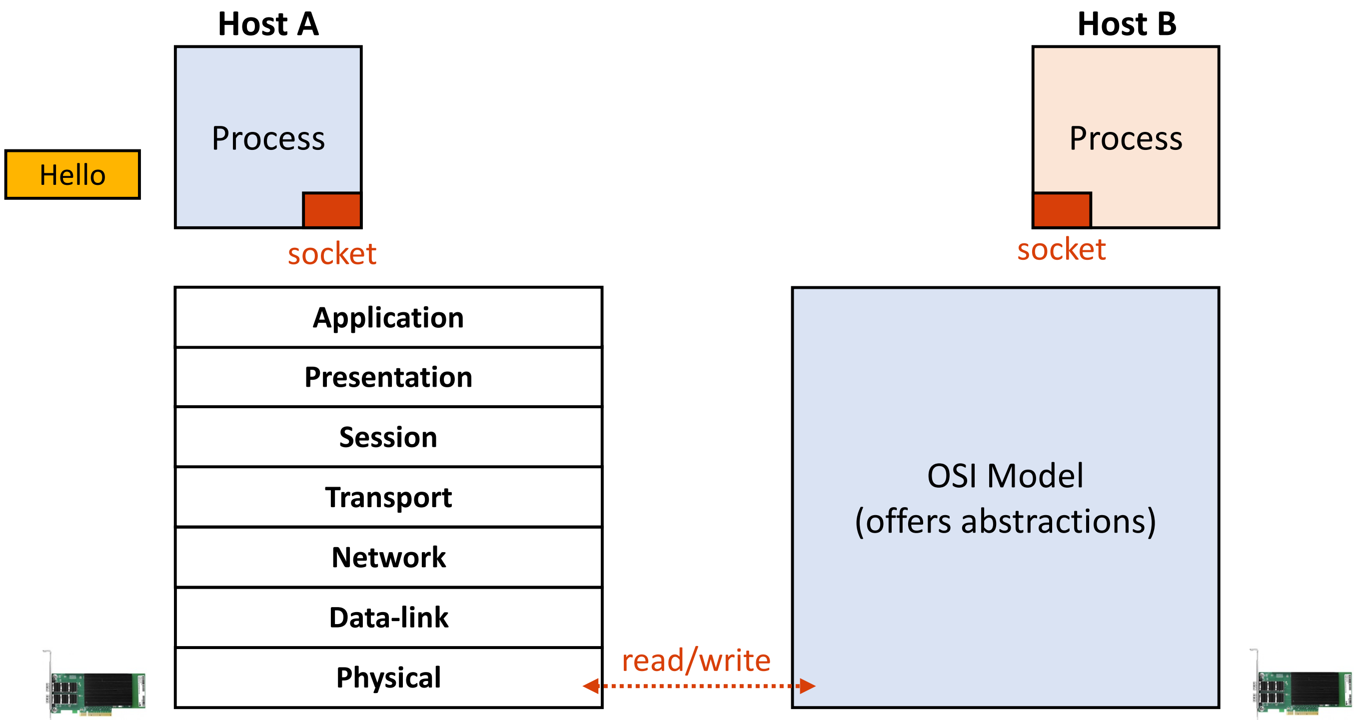
Recap: OSI model

- **O**pen **I**nternet **I**nterface (**OSI**) model



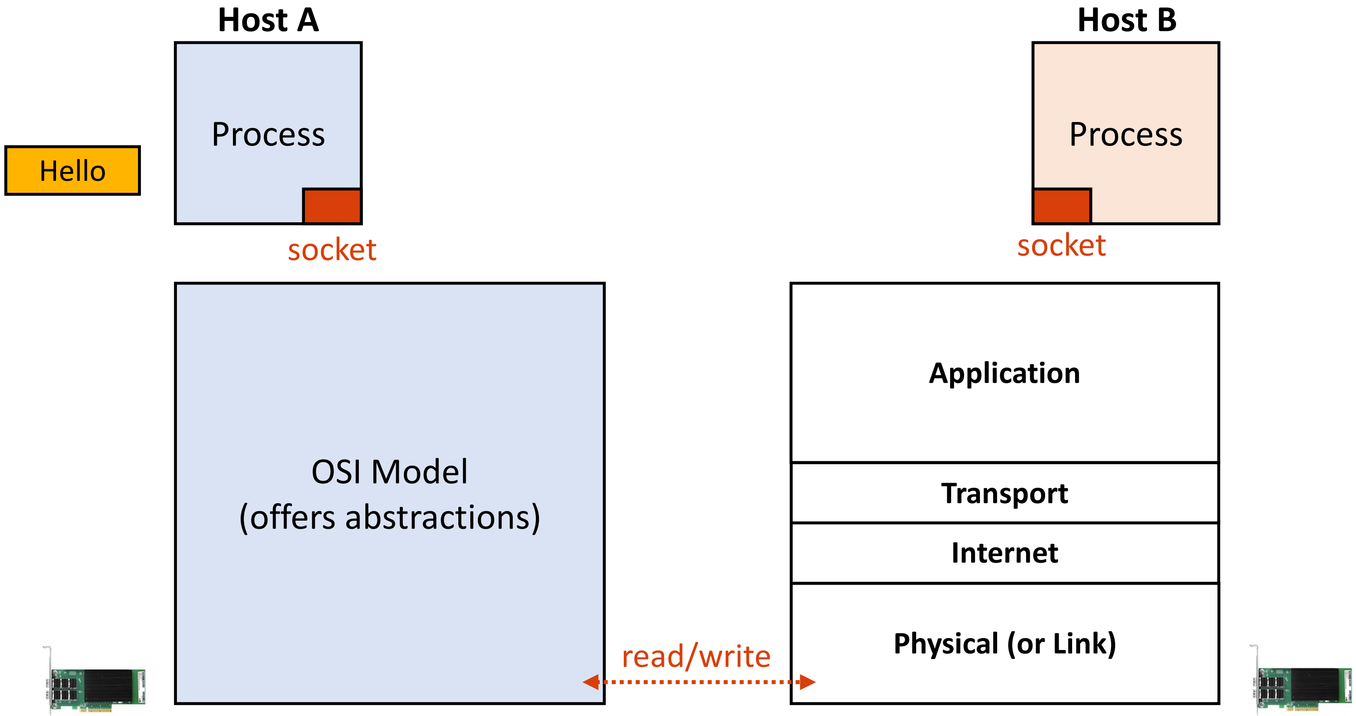
Recap: OSI 7-layer model

- **O**pen **I**nternet **I**nterface (**OSI**) model



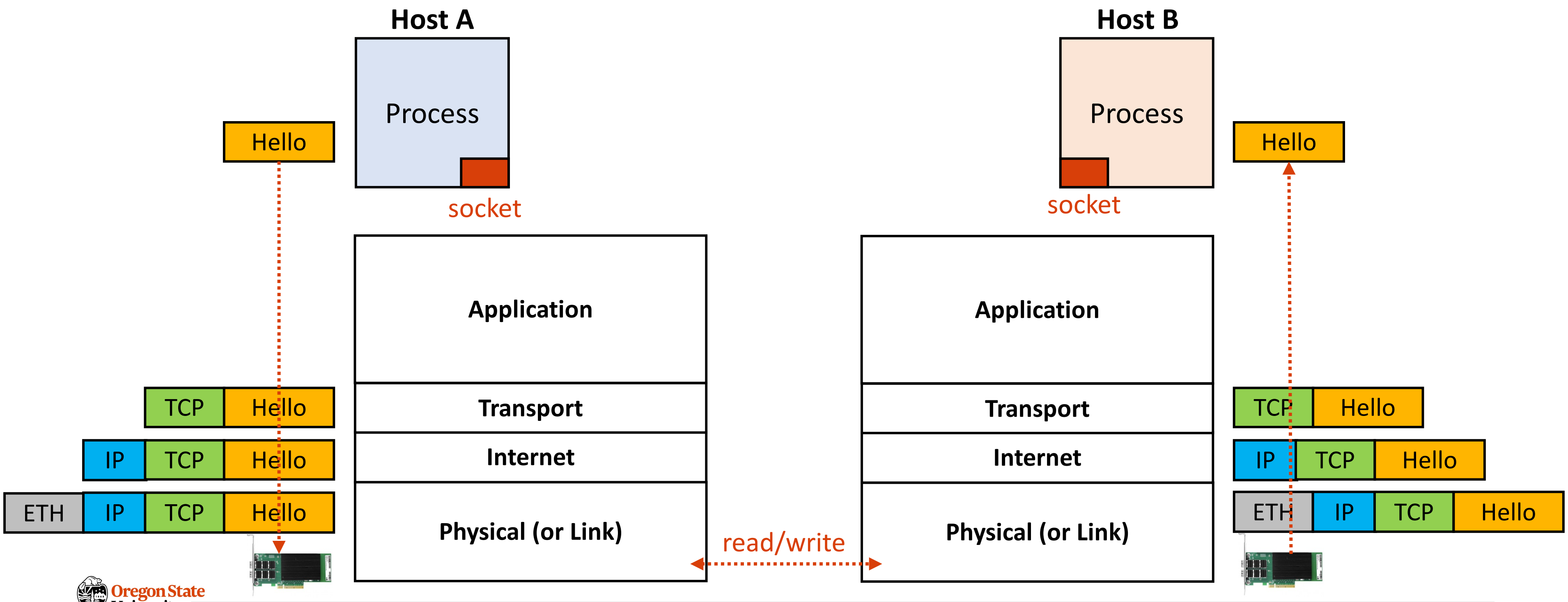
Recap: TCP/IP 4-layer model

- **TCP/IP** 4-layer model



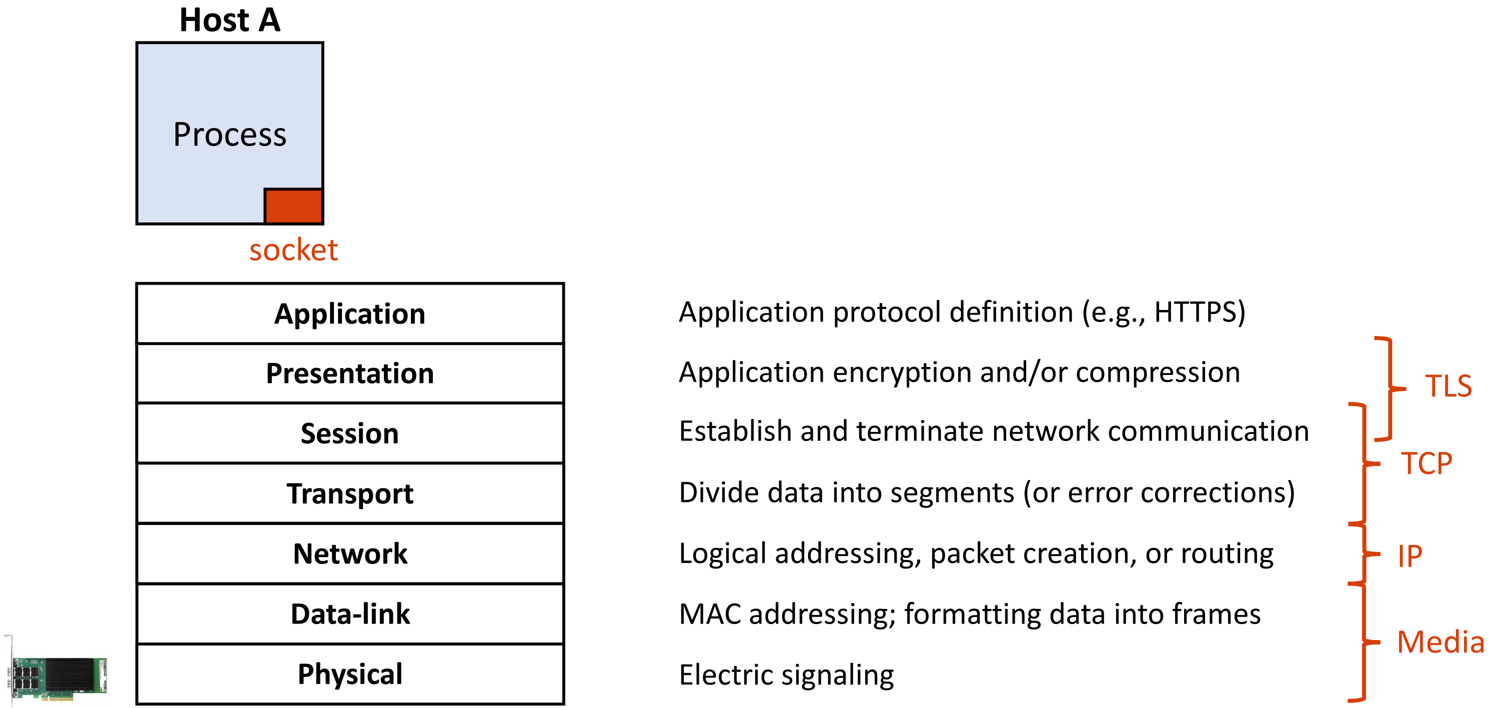
Recap: packet encapsulation

- TCP/IP 4-layer model (OSI 7-layer has more…)

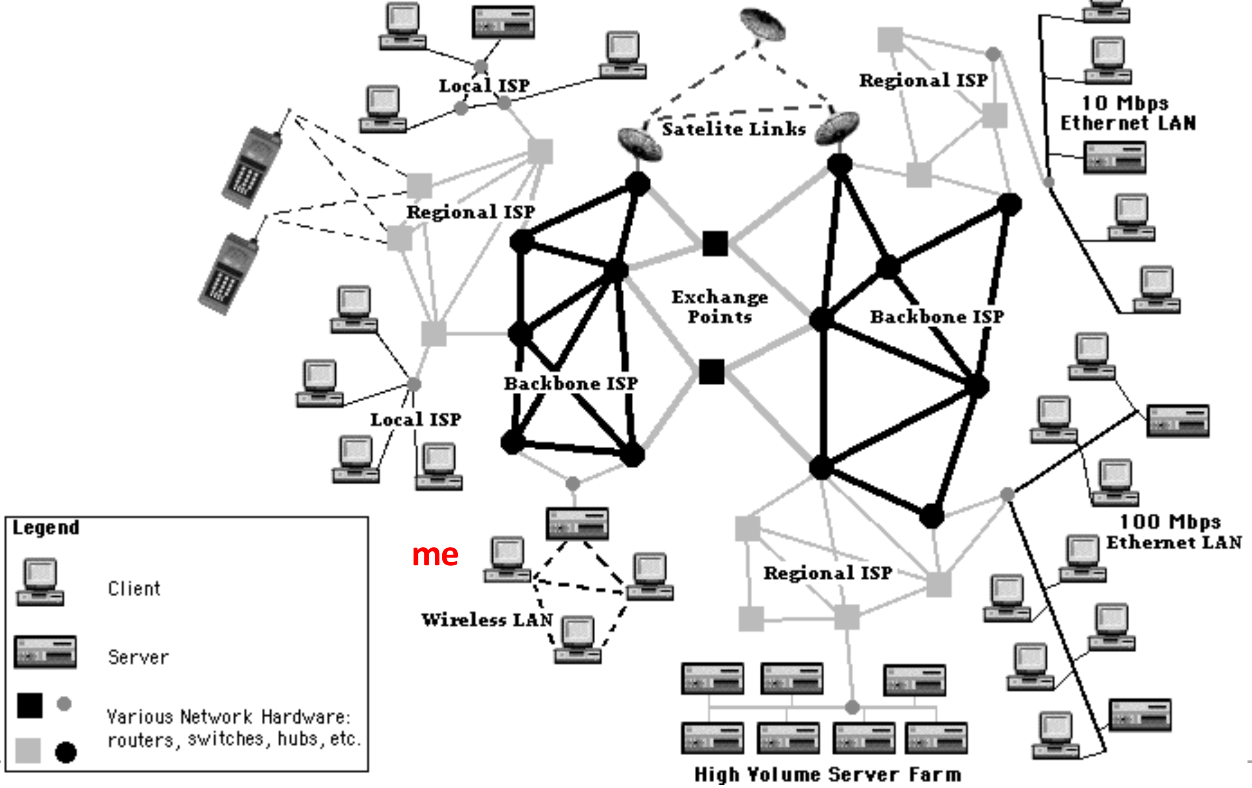


Recap: OSI 7-layer model

- **<u>O</u>**pen **<u>I</u>**nternet **<u>I</u>**nterface (**<u>OSI</u>**) model



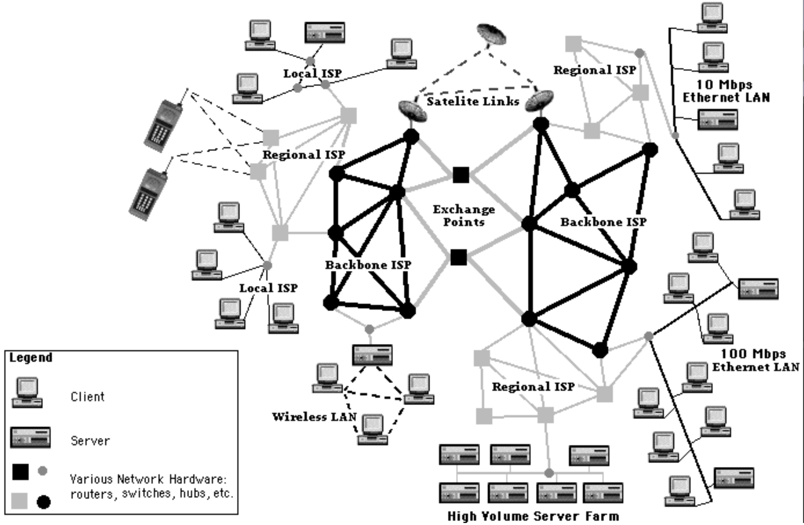
The Internet: packet routing

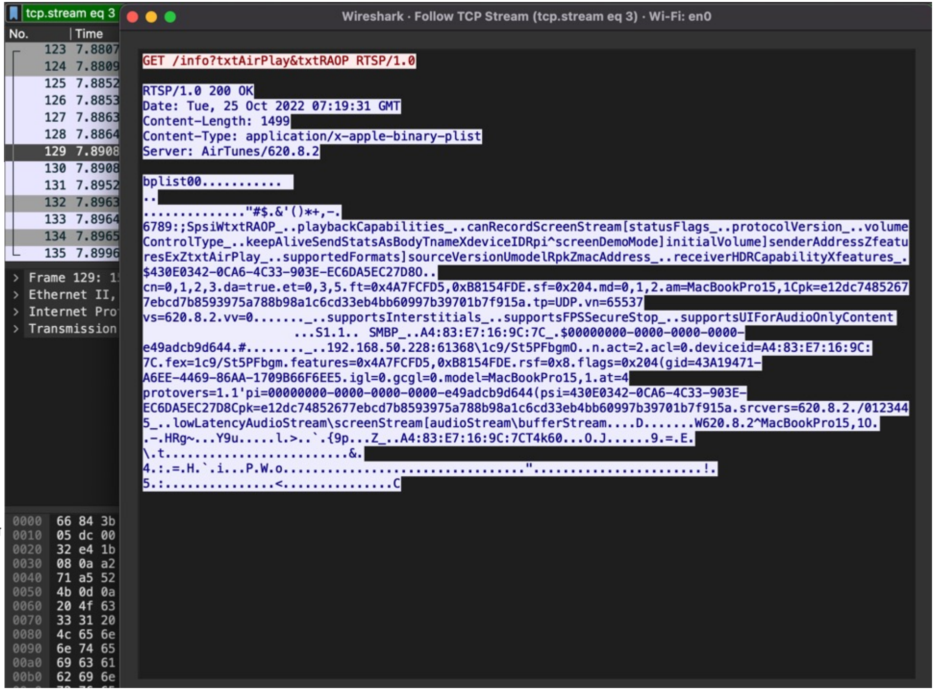


Oregon State University

The Internet: (no) security

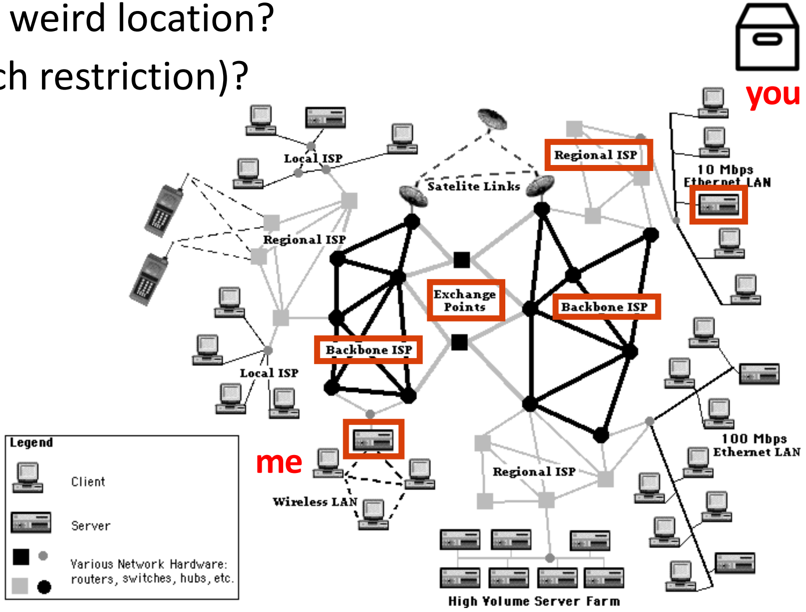
- No security (in TCP communication)
  - Any router in the middle can see any packet content :(





The Internet: (no) security

- Routers:
  - Decide where the packet should go as a next step
  - What if
    - the router in the middle sends a packet to weird location?
    - the router(s) are malicious (there is no such restriction)?



We Cannot Establish Trust in Routers

Topics for today

- SSL and TLS security
  - The Internet is not secure
  - How can we implement secure communication channels?
  - How can we establish such channels between two parties?
  - How can we minimize the impact of security incidents?
  - How do we use to achieve such a goal (in practice)?

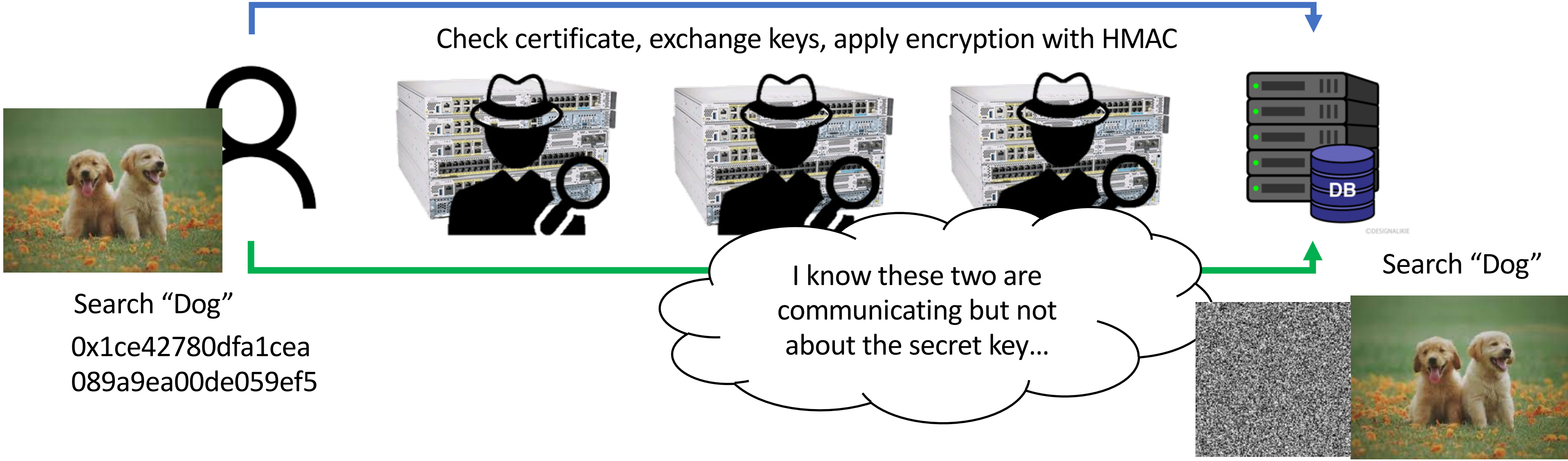
The Internet without security

Search "Dog"

Everybody in the Middle Knows That I Searched 'dogs' and They Also Know the Search Result… Ugh…

The Internet with a secure mechanism (SSL/TLS)

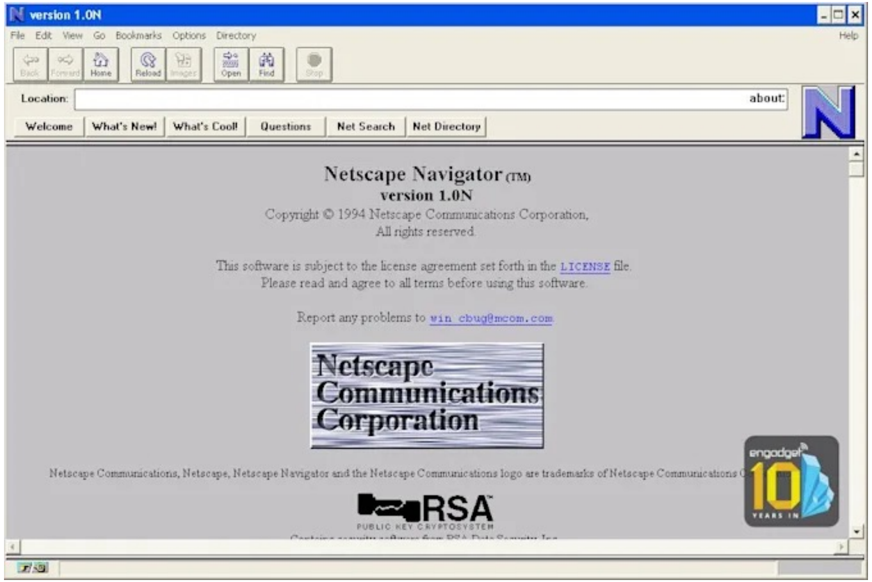
Middle mans never know DH exchange keys!!



The Middlemen Will Only See the Encrypted Contents
They Will Never Know the Secret Key …

SSL/TLS: SECURE SOCKET LAYER AND TRANSPORT LAYER SECURITY

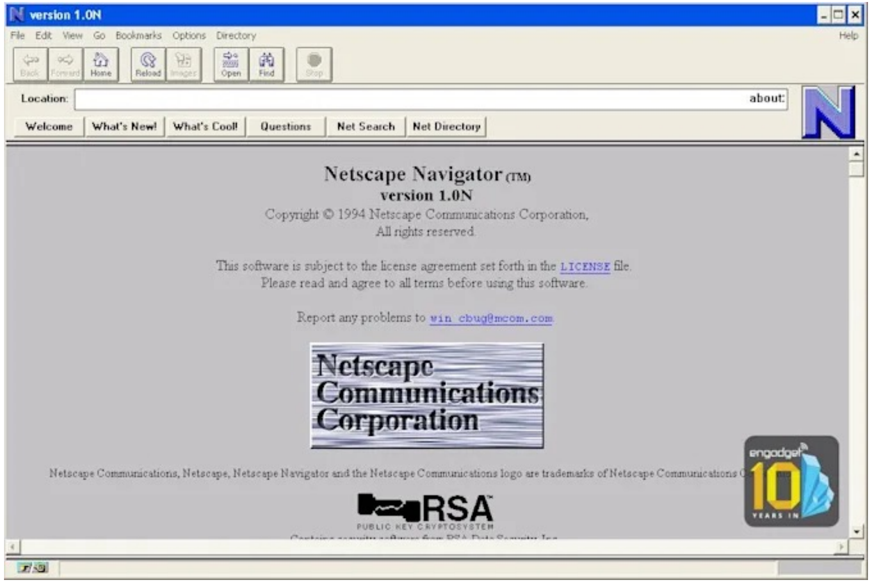
- SSL/TLS
  - Developed by Netscape in 1995
  - Standardized by IETF as TLS
  - https://www.ietf.org/rfc/rfc2246.txt



SSL/TLS: SECURE SOCKET LAYER AND TRANSPORT LAYER SECURITY

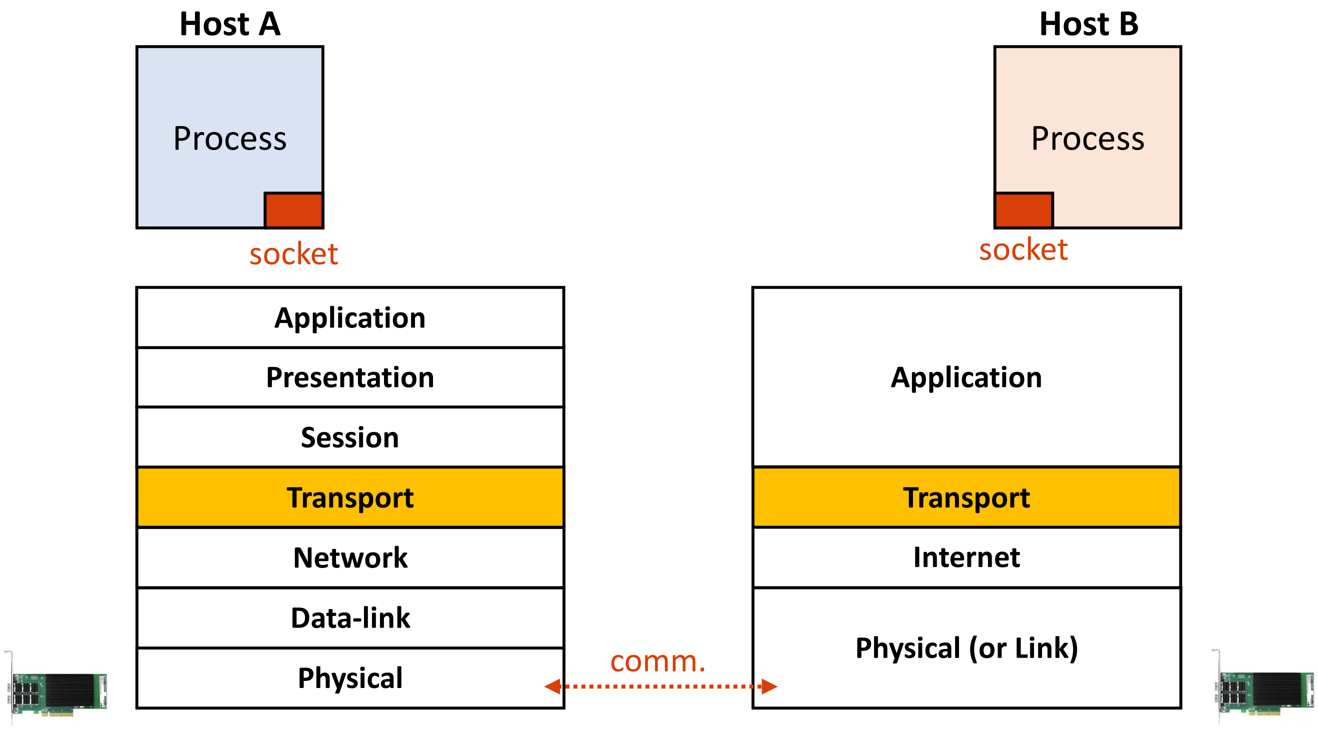
- SSL/TLS
  - Developed by Netscape in 1995
  - Standardized by IETF as TLS
  - https://www.ietf.org/rfc/rfc2246.txt

- "Transport Layer" Security
  - Why?



SSL/TLS: Transport layer security, why?

- Independent from the application running on a host



SSL/TLS: Benefits

- Enable to
  - Establish secure comm channels btw two ends (hosts) on the Internet
    - Client <-> Server (ex. OSU login)
    - Server <-> Server (ex. Amazon requests a transaction with your credit card)
    - Client <-> Client (ex. chat applications)
  - Verify the server entity
    - Use a digital certificate
- end-to-end secure communication channels
  - Authentication: a digital certificate
  - Key-exchange: Diffie-Hellman key exchange
  - Confidentiality: Block ciphers
  - Integrity: HMAC / MAC

Topics for today

- SSL and TLS security
  - The Internet is not secure
  - SSL/TLS for secure communications
  - How can we establish such channels between two parties?
  - How can we minimize the impact of security incidents?
  - How do we use to achieve such a goal (in practice)?

SSL/TLS: Handshaking

Client (You)

- 1. Client hello
  - Send version, random number, available cipher suite, etc..

(google.com) Server

- 2. Server hello
  - Sends server random, version, choose cipher, etc.
- 3. Server Certificate
  - Send certificate to the client

SSL/TLS: Step I – client hello

- The first message a client sends to the server
  - It sends an SSL/TLS version, a random number, an available cipher suite, …

```
00000000  16 03 01 01 44 01 00 01  40 03 03 95 8b 02 ec f4   ....D.
00000010  ca 4d 7d 98 6b 9e 3f 45  8b fa 92 10 f0 9c 2c aa   .M}.k.
00000020  bf 27 f0 52 b0 97 6c f0  6c a2 a9 20 bc b7 86 80   .'.R..
00000030  f2 f1 71 9f e0 7e 7e 4c  c2 51 88 e7 72 2d e0 3c   ..q..~
00000040  ca cc fa 2c 99 dc b9 56  d0 80 bd 91 00 62 13 02   ...,..
00000050  13 03 13 01 c0 30 c0 2c  c0 28 c0 24 c0 14 c0 0a   .....0
00000060  00 9f 00 6b 00 39 cc a9  cc a8 cc aa ff 85 00 c4   ...k.9
00000070  00 88 00 81 00 9d 00 3d  00 35 00 c0 00 84 c0 2f   ......
00000080  c0 2b c0 27 c0 23 c0 13  c0 09 00 9e 00 67 00 33   .+.'.#
00000090  00 be 00 45 00 9c 00 3c  00 2f 00 ba 00 41 c0 11   ...E..
000000A0  c0 07 00 05 00 04 c0 12  c0 08 00 16 00 0a 00 ff   ......
000000B0  01 00 00 95 00 2b 00 09  08 03 04 03 03 03 02 03   .....+
000000C0  01 00 33 00 26 00 24 00  1d 00 20 ba 53 26 b5 f2   ..3.&.
000000D0  19 5d b0 e0 b5 f4 30 0c  73 e9 2a 1d 86 72 d5 29   .]....
000000E0  6e fc 32 3f d3 0f 31 d6  e2 57 61 00 00 00 18 00   n.2?..
000000F0  16 00 00 13 77 77 77 2e  6f 72 65 67 6f 6e 73 74   ....ww
00000100  61 74 65 2e 65 64 75 00  0b 00 02 01 00 00 0a 00   ate.ed
00000110  0a 00 08 00 1d 00 17 00  18 00 19 00 0d 00 18 00   ......
00000120  16 08 06 06 01 06 03 08  05 05 01 05 03 08 04 04   ......
00000130  01 04 03 02 01 02 03 00  10 00 0e 00 0c 02 68 32   ......
00000140  08 68 74 74 70 2f 31 2e  31                        .http/
```

```
⌄ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 324
  ⌄ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 320
      Version: TLS 1.2 (0x0303)
    › Random: 958b02ecf4ca4d7d986b9e3f458bfa9210f09c2caabf27f052b0976cf06ca2a9
      Session ID Length: 32
      Session ID: bcb78680f2f1719fe07e7e4cc25188e7722de03ccaccfa2c99dcb956d080bd91
      Cipher Suites Length: 98
    › Cipher Suites (49 suites)
      Compression Methods Length: 1
    › Compression Methods (1 method)
      Extensions Length: 149
    ⌄ Extension: supported_versions (len=9)
        Type: supported_versions (43)
        Length: 9
        Supported Versions length: 8
        Supported Version: TLS 1.3 (0x0304)
        Supported Version: TLS 1.2 (0x0303)
        Supported Version: TLS 1.1 (0x0302)
        Supported Version: TLS 1.0 (0x0301)
```

SSL/TLS: Step I – client hello

- It sends supported cipher suites:
  - TLS_ECDHE_RSA_WITH AES_128_GCM_SHA256 ECDHE_RSA_AES_128_GCM_SHA256

```
Cipher Suites (49 suites)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x006b)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
    Cipher Suite: TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xccaa)
    Cipher Suite: Unknown (0xff85)
    Cipher Suite: TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256 (0x00c4)
    Cipher Suite: TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA (0x0088)
    Cipher Suite: TLS_GOSTR341001_WITH_28147_CNT_IMIT (0x0081)
    Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Cipher Suite: TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256 (0x00c0)
    Cipher Suite: TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (0x0084)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x0067)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
    Cipher Suite: TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256 (0x00be)
    Cipher Suite: TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA (0x0045)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256 (0x00ba)
```

Note: ECDHE_RSA_WITH_AES_128_GCM_SHA256

- ECDHE
  - Key exchange algorithm. Elliptic Curve Diffie—Hellman Ephemeral
- RSA
  - Digital Signature algorithm. We use this for checking authenticity
- AES-128-GCM
  - Symmetric cipher algorithm/mode. We will use AES-128 in GCM mode
- SHA256
  - HMAC algorithm. We will use SHA256 to construct an HMAC

SSL/TLS: Step II – server hello

- The first message a client sends to the server
  - It sends an SSL/TLS version, a random number, an available cipher suite, …

```
⌄ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 102
  ⌄ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 98
      Version: TLS 1.2 (0x0303)
    › Random: 7937be8da9875cf054f0ed18b7efec590e2fb8823ffb7afb87fdffed322822dc
      Session ID Length: 32
      Session ID: 6adeb8c9532bf74b3f5d9940e83f470e46ac3f49054c667dfe8255a6342bea6e
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Compression Method: null (0)
```

- The server choose a cipher based on the client's availability
  - **Chosen:** TLS_ECDHE_RSA_AES_128_GCM_SHA256

SSL/TLS: Step III – server certificate

- The first message a client sends to the server
  - It sends an SSL/TLS version, a random number, an available cipher suite, …
- The server choose a cipher based on the client's availability
  - **Chosen:** TLS_ECDHE_RSA_AES_128_GCM_SHA256
- The server next sends the certificate information to the client
  - It sends a full chain (PKI) of digital certificates

```
⌄ TLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 6037
  ⌄ Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 6033
      Certificates Length: 6030
    ⌄ Certificates (6030 bytes)
        Certificate Length: 1994
      › Certificate: 308207c6308206aea003020102021030bc9131f05e7eef26b3844d426b816c300d06092a… (id-at-commonName=oregonstate.edu,id-at-organizationName
        Certificate Length: 1533
      › Certificate: 308205f9308203e1a00302010202104720d0fa85461a7e17a1640291846374300d06092a… (id-at-commonName=InCommon RSA Server CA,id-at-organizat
        Certificate Length: 1413
      › Certificate: 3082058130820469a00302010202103972443af922b751d7d36c10dd313595300d06092a… (id-at-commonName=USERTrust RSA Certification Authority,
        Certificate Length: 1078
      › Certificate: 308204323082031aa003020102020101300d06092a864886f70d0101050500307b310b30… (id-at-commonName=AAA Certificate Services,id-at-organiz
```

SSL/TLS: Step IV – key exchange / verifying signature

- Key exchange
  - The client knows the server's public key written in their certificate
  - The client chooses a random key and encrypt that with the server's public key
  - The encrypted key will be sent to the server
  - It's only the server who can decrypt the key (good)

Are We Secure Now? Can We See A Potential Security Issues?

SSL/TLS: Potential security problem

- Key exchange
  - The client knows the server's public key written in their certificate
  - The client chooses a random key and encrypt that with the server's public key
  - The encrypted key will be sent to the server
  - It's only the server who can decrypt the key (good)
- Suppose:
  - 3 years later, the server's private key is stolen
  - From then, the attacker can decrypt the all the data (private key, messages, …)
  - What if the attacker also has all the encrypted messages before the breach?

Topics for today

- SSL and TLS security
  - The Internet is not secure
  - SSL/TLS for secure communications
  - SSL/TLS handshakes (hello-s)
  - How can we minimize the impact of security incidents?
  - How do we use to achieve such a goal (in practice)?

SSL/TLS: Requires forward security

- Forward Secrecy / Perfect Forward Secrecy
  - We want to keep all the communication secure
  - Even if the server's private key (i.e., the long-term key) has been breached

- Example of such breaches
  - Heartbleed (https://heartbleed.com/): CVE-2014-0160

SSL/TLS: Solution – Ephemeral Diffie-Hellman

- The key idea:
  - Do not use a fixed private value for all the DH
  - This can lead to a serious information breach (stolen private key)

- Ephemeral DH
  - Generate the private value every time we make a connection
  - Never reuse the value
    - User A secretly chooses a, send $A = g^a \bmod p$
    - User B secretly chooses b, send $B = g^b \bmod p$
    - User A and B will choose different a and b for the next time

SSL/TLS: ECDHE

- Elliptic-curve Diffie-Hellman Ephemeral (ECDHE)
  - Both the client and server will generate new a and b, respectively
  - Make it difficult for an adversary to infer the shared secret even if the session is compromised (they don't know b for other sessions)

SSL/TLS: Handshaking

Client (You)

- 1. Client hello

(google.com) Server

- 2. Server hello
- 3. Server Certificate
- 4. Server Key Exchange
  - Shares DH material, signed by the public key
- 5. Server Hello Done

SSL/TLS: Step IV – KEY EXCHANGE

- The server sends ECDHE material to the client
  - ECDHE public value (pubkey) is signed by the RSA private key
  - The public key is available in the certificate

```
v Transport Layer Security
  v TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 333
    v Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 329
      v EC Diffie-Hellman Server Params
          Curve Type: named_curve (0x03)
          Named Curve: secp256r1 (0x0017)
          Pubkey Length: 65
          Pubkey: 04d3be5c83a346d31403c9803f753af4c583cd3504d550f5e1be0368c624acf4fa7e1b85…
        > Signature Algorithm: rsa_pkcs1_sha512 (0x0601)
          Signature Length: 256
          Signature: 5fe6444e7ae294aa7815516c91c19eadd1a5edc72e1a690916a4acb89669eb219a669970…
```

SSL/TLS: Step V – server hello done

- The server sends ECDHE material to the client
  - ECDHE public value (pubkey) is signed by the RSA private key
  - The public key is available in the certificate
- The server hello done
  - Indicate that the server has finished sending required values to the client

```
⌄ Transport Layer Security
  ⌄ TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 4
    ⌄ Handshake Protocol: Server Hello Done
        Handshake Type: Server Hello Done (14)
        Length: 0
```

SSL/TLS: Handshaking

Client (You)

- 1. Client hello

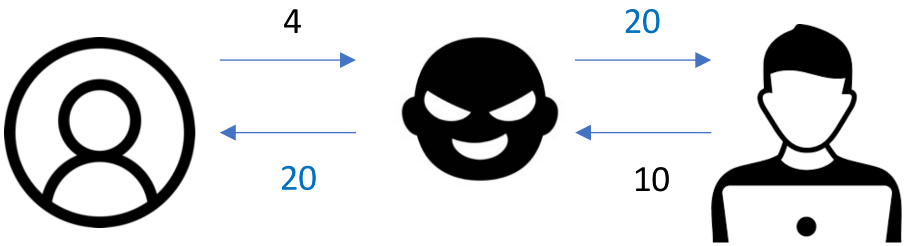
(google.com) Server

- 2. Server hello
- 3. Server Certificate
- 4. Server Key Exchange
  - Shares DH material, signed by the public key
- 5. Server Hello Done

Now, the Client Can Verify Server Signature and Share a Secret via DH!

Recap: Diffie-Hellman's weakness to man-in-the-middle

- Suppose C intercepts communication between A and B
  - A chooses a = 4
    - A = 5^4 mod 23 = 625 mod 23 = 4
  - B chooses b = 3
    - B = 5^3 mod 23 = 125 mod 23 = 10
  - C chooses c = 5
    - C = 55 mod 23 = 3125 mod 23 = 20



- C sends 20 to both A and B

```
EC Diffie-Hellman Server Params
  Curve Type: named_curve (0x03)
  Named Curve: secp256r1 (0x0017)
  Pubkey Length: 65
  Pubkey: 04d3be5c83a346d31403c9803f753af4c583cd3504d550f5e1be0368c624acf4fa7e1b85…
> Signature Algorithm: rsa_pkcs1_sha512 (0x0601)
  Signature Length: 256
  Signature: 5fe6444e7ae294aa7815516c91c19eadd1a5edc72e1a690916a4acb89669eb219a669970…
```

SSL/TLS: Handshaking

Client (You)

(google.com) Server

Previous steps (omitted)

- 5. Server Hello Done

- 6. Client Key Exchange
  - Shares DH material after verifying server signature for server's DH material
- 7. Change Cipher Spec
- 8. Encrypted Handshake Message

SSL/TLS: Step VI – client key exchange

- The client also sends ECDHE material to the server
  - After this, two parties will share a secret
  - We will run the encryption and MAC key by using the shared secret

```
⌄ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 70
  ⌄ Handshake Protocol: Client Key Exchange
      Handshake Type: Client Key Exchange (16)
      Length: 66
    ⌄ EC Diffie-Hellman Client Params
        Pubkey Length: 65
        Pubkey: 043cc5f595ea1dca4b3beb1306dec9444e5323177ef9b2c5470dd910d2ce252f672a1dc8…
```

SSL/TLS: Step VI – client generates a session key

- Now the client knows both ‘a’ and ‘b’ of ECDHE key exchange
  - The client can compute the shared secret
  - The client then computes the following keys from the shared secret

```
To generate the key material, compute

   key_block = PRF(SecurityParameters.master_secret,
                   "key expansion",
                   SecurityParameters.server_random +
                   SecurityParameters.client_random);

until enough output has been generated.  Then, the key_block is
partitioned as follows:

   client_write_MAC_key[SecurityParameters.mac_key_length]
   server_write_MAC_key[SecurityParameters.mac_key_length]
   client_write_key[SecurityParameters.enc_key_length]
   server_write_key[SecurityParameters.enc_key_length]
   client_write_IV[SecurityParameters.fixed_iv_length]
   server_write_IV[SecurityParameters.fixed_iv_length]
```

These are from
1. Client Hello and
2. Server Hello

SSL/TLS: Step VII – Change Cipher Spec (Client)

- Secure communication:
  - The client sends the server a message
  - that now both should use encrypted communication after this point

```
⌄ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
```

Now, We Encrypt Messages and Generate MACs for the Client's!

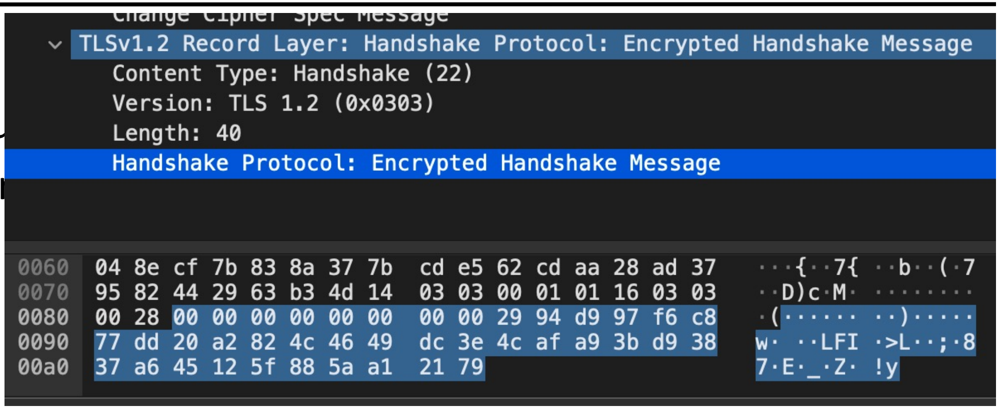
SSL/TLS: Step VIII – Encrypted handshake message

- The server asks
  - the encrypted versions of previous messages
  - to verify whether the client generated the keys correctly

- Compute a SHA256 hash of a concatenation of all the handshake communications (or SHA384 if the PRF is based on SHA384). This means the Client Hello, Server Hello, Certificate, Server Key Exchange, Server Hello Done and Client Key Exchange messages. Note that you should concatenate only the handshake part of each TLS message (i.e. strip the first 5 bytes belonging to the TLS Record header)
- Compute PRF(master_secret, "client finished", hash, 12) which will generate a 12-bytes hash
- Append the following header which indicates the hash is 12 bytes: 0x14 0x00 0x00 0x0C
- Encrypt the 0x14 0x00 0x00 0x0C | [12-bytes hash] (see the Encrypting / Decrypting data section). This will generate a 64-bytes ciphertext using AES-CBC and 40 bytes with AES-GCM
- Send this ciphertext wrapped in a TLS Record

SSL/TLS: Step VIII – Encrypted handshake message

- The server asks
  - the encrypted versions of previou
  - to verify whether the client gener



- Compute a SHA256 hash of a concatenation of all the handshake communications (or SHA384 if the PRF is based on SHA384). This means the Client Hello, Server Hello, Certificate, Server Key Exchange, Server Hello Done and Client Key Exchange messages. Note that you should concatenate only the handshake part of each TLS message (i.e. strip the first 5 bytes belonging to the TLS Record header)
- Compute PRF(master_secret, "client finished", hash, 12) which will generate a 12-bytes hash
- Append the following header which indicates the hash is 12 bytes: 0x14 0x00 0x00 0x0C
- Encrypt the 0x14 0x00 0x00 0x0C | [12-bytes hash] (see the Encrypting / Decrypting data section). This will generate a 64-bytes ciphertext using AES-CBC and 40 bytes with AES-GCM
- Send this ciphertext wrapped in a TLS Record

SSL/TLS: Handshaking

Client (You)

(google.com) Server

Previous steps (omitted)

- 5. Server Hello Done

- 6. Client Key Exchange
  - Shares DH material after verifying server signature for server's DH material
- 7. Change Cipher Spec
- 8. Encrypted Handshake Message

- 9. Change Cipher Spec
- 10. Encrypted Handshake Message

SSL/TLS: Step XV – Check client's encrypted handshake messages

- The server verifies the client's encrypted handshake messages
  - After generating client_write_key
  - Decrypt the message
  - Compute the same value
  - Compare!

- Compute a SHA256 hash of a concatenation of all the handshake communications (or SHA384 if the PRF is based on SHA384). This means the Client Hello, Server Hello, Certificate, Server Key Exchange, Server Hello Done and Client Key Exchange messages. Note that you should concatenate only the handshake part of each TLS message (i.e. strip the first 5 bytes belonging to the TLS Record header)
- Compute PRF(master_secret, "client finished", hash, 12) which will generate a 12-bytes hash
- Append the following header which indicates the hash is 12 bytes: 0x14 0x00 0x00 0x0C
- Encrypt the 0x14 0x00 0x00 0x0C | [12-bytes hash] (see the Encrypting / Decrypting data section). This will generate a 64-bytes ciphertext using AES-CBC and 40 bytes with AES-GCM
- Send this ciphertext wrapped in a TLS Record

SSL/TLS: Step XV – Change cipher spec (server)

- The server lets the client know
  - that we will use encrypted communication after this message

```
⌄ Transport Layer Security
  ⌄ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
```

Now, We Encrypt Messages and Generate MACs for the Server's!

SSL/TLS: Step X – Encrypted handshake message

- The client asks
  - the encrypted version of previous messages
  - to verify whether the server generated keys correctly

```
⌄ TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 40
    Handshake Protocol: Encrypted Handshake Message
```

- It needs to compute a hash of the same handshake communications as the client as well as the decrypted "Encrypted Handshake Message" message sent by the client (i.e. the 16-bytes hash starting with 0x1400000C)
- It will call PRF(master_secret, "server finished", hash, 12)

SSL/TLS: Step XI - Sending application data

- Now, the server and client
  - will send encrypted data to the client
  - both will always send [encrypted data] [MAC]
    - The server will use server_write_key and server_write_mac_key
    - The client will use client_write_key and client_write_mac_key

Teaser: how do we use SSL/TLS?

- HTTP(s)
  - HTTP: Hypertext Transfer Protocol
  - A network protocol for accessing World Wide Web

- http:// vs. https://
  - http:// ← this directive let web browsers connect directly via HTTP
  - https:// ← this directive let web browsers connect HTTP via TLS

Topics for today

- SSL and TLS security
  - The Internet is not secure
  - SSL/TLS for secure communications
  - SSL/TLS handshakes (hello-s)
  - (Perfect) Forward Security
  - How do we use to achieve such a goal (in practice)? (next lecture)

Thank You!

Tu/Th 4:00 – 5:50 PM

Sanghyun Hong

sanghyun.hong@oregonstate.edu

Oregon State University

SAIL
Secure AI Systems Lab