CS 370
Introduction to Security

Secure Socket Layer (SSL) and Transport Layer Security (TLS)

Yeongjin Jang

Digital Certificate

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

Certificate Viewer: oregonstate.edu

General | Details

Issued To

Common Name (CN)	oregonstate.edu
Organization (O)	Oregon State University
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	InCommon RSA Server CA
Organization (O)	Internet2
Organizational Unit (OU)	InCommon

Validity Period

Issued On	Sunday, June 5, 2022 at 5:00:00 PM
Expires On	Tuesday, June 6, 2023 at 4:59:59 PM

Fingerprints

SHA-256 Fingerprint	7B 57 A4 91 B0 06 29 2E 8E 54 04 FB BB F6 F8 4F 09 56 15 C0 20 59 37 9F E9 F1 A4 27 DC B6 F4 E1
SHA-1 Fingerprint	FC EE 7C 4B AA 30 8F A6 03 E2 22 C5 31 FF 6C C6 92 FF C3 8E

Trust Chain

- oregonstate.edu
  - Verified by InCommon RSA Server CA
- InCommon RSA Server CA
  - Verified by USERTrust RSA Certificate Authority
- USERTrust RSA CA
  - Verified by itself

oregonstate.edu | InCommon RSA Server CA | USERTrust RSA Certification Authority

Subject Name

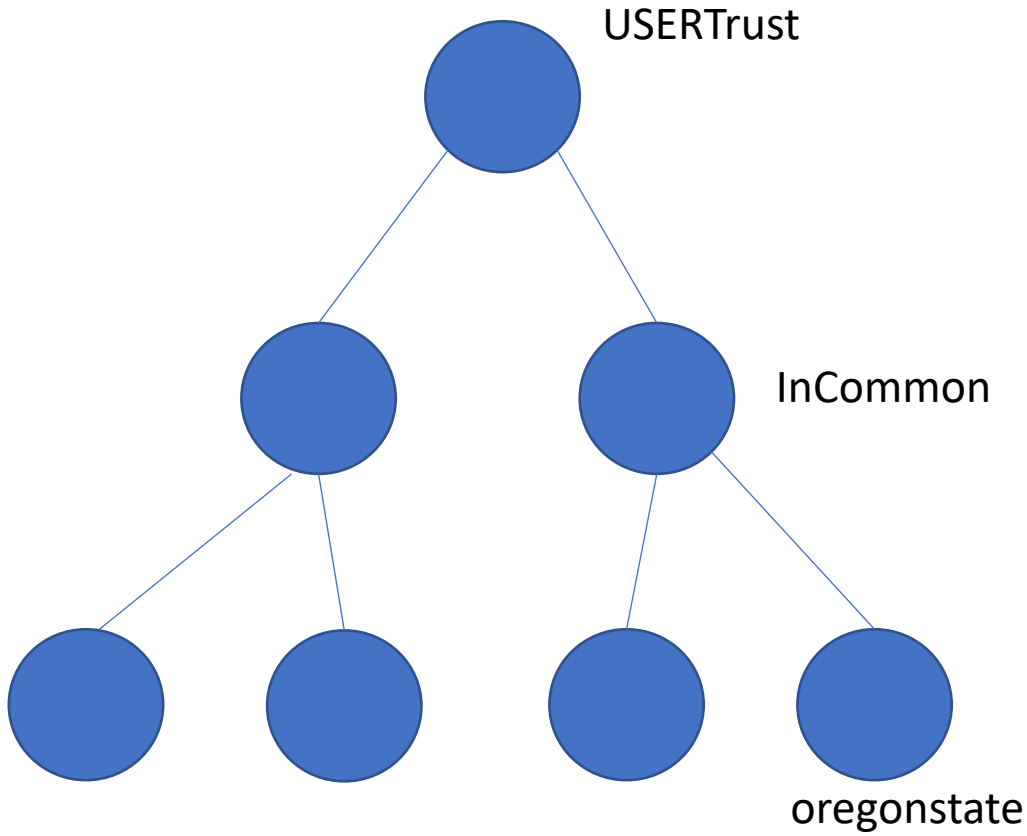
Country	US
State/Province	New Jersey
Locality	Jersey City
Organization	The USERTRUST Network
Common Name	USERTrust RSA Certification Authority

Issuer Name

Country	US
State/Province	New Jersey
Locality	Jersey City
Organization	The USERTRUST Network
Common Name	USERTrust RSA Certification Authority

Public Key Infrastructure (PKI)

- An Infrastructure that provides public key with certificate chain
- Trust anchor: Root CA
  - We set a small set of entities use self-signed cert
- Verify the certificate chain!
  - We must verify the entire chain



Diffie-Hellman Key Exchange

- User A & User B agrees on g and p, where g and p are primes

- User A secretly chooses a, send $A = g^a \bmod p$
- User B secretly chooses b, send $B = g^b \bmod p$

- User A receives B, compute $B^a = (g^b)^a \bmod p = g^{ab} \bmod p$
- User B receives A, compute $A^b = (g^a)^b \bmod p = g^{ab} \bmod p$

- $g^{ab} \bmod p$ is our secret

Diffie-Hellman Key Exchange

- g^{ab} mod p is our secret

- Attacker knows g, p, $A = g^a$ mod p and $B = g^b$ mod p

- $A+B = (g^a + g^b)$ mod p
- $AB = g^{(a+b)}$ mod p
- Hard to compute g^{ab} from those values
  - Discrete logarithm; Can you get a from $A = g^a$ mod p

How Can We Secure the Internet Communication?

- Authentication
  - Get the certificate of each entity
  - Verify their public key
  - Using certificate trust chain!

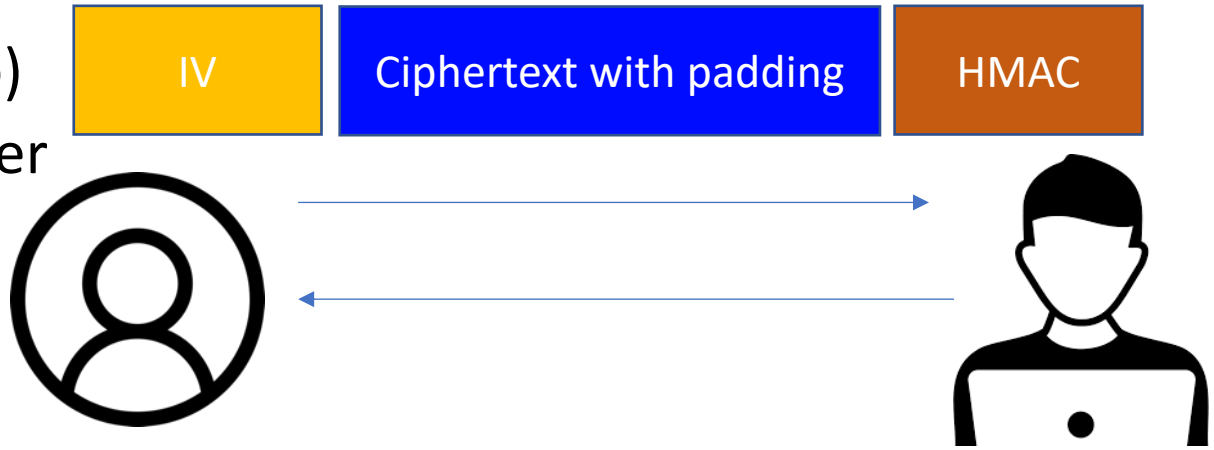
- Key-exchange
  - A computes g^a mod p, and sign that with A's private key
  - B computes g^b mod p, and sign that with B's private key
  - Both can verify the identity of each and then share
    - g^{ab} mod p

Killed
Man-in-the-middle attack!

How Can We Secure the Internet Communication?

- Confidentiality
  - Run SHA-256('cipher key' + g^{ab} mod p)
  - Use that as the key for the block cipher
  - E.g., AES-256-CBC

- Integrity
  - Run SHA-256('mac key' + g^{ab} mod p)
  - Use that as the key for HMAC



A communication channel with
Authenticity
Confidentiality
Integrity
has been established!

Topics for Today

- The Insecure Internet
  - OSI 7 layers, no security at all
- Secure Socket Layer
  - Enabling secure communication over the Internet
  - Later, the name changed to Transport Layer Security
- We will learn how SSL/TLS establishes secure communication channel
  - Over the insecure Internet!

Internet

- Inter + net
  - A network of networks
- 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

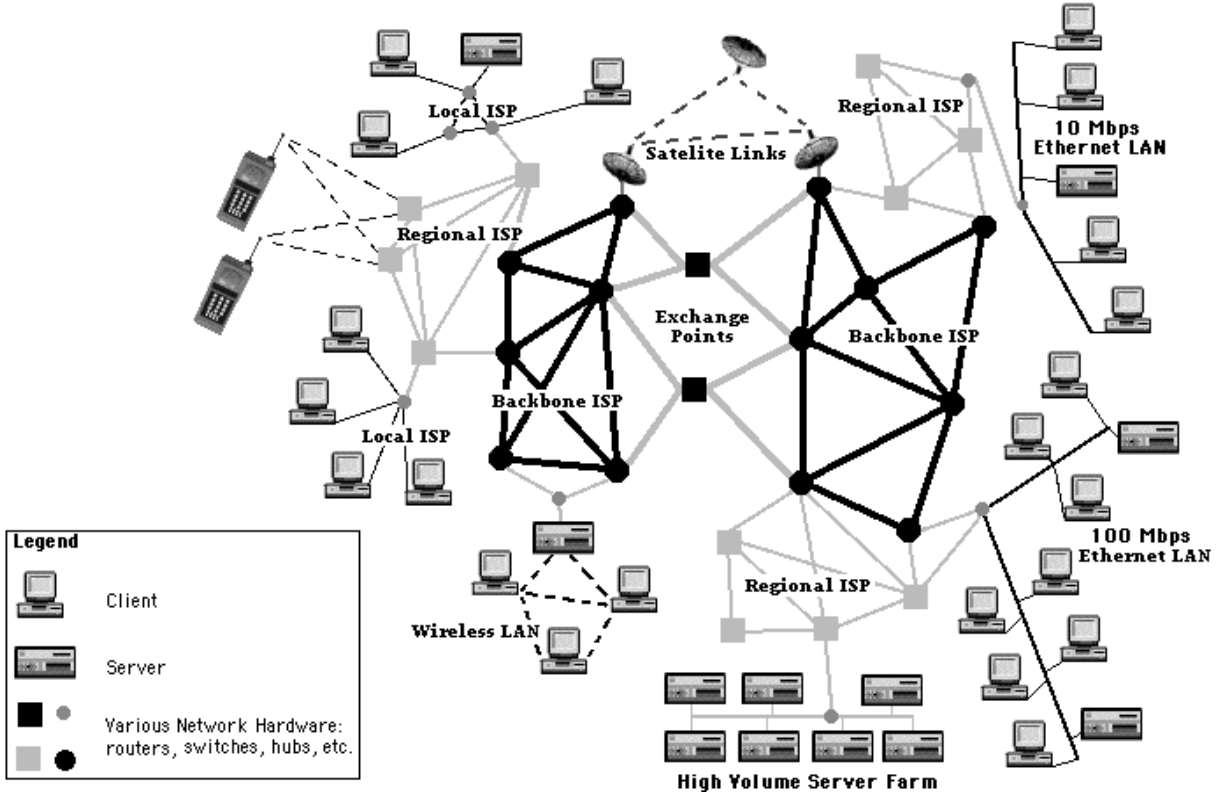


Image is from: https://www.cs.utexas.edu/~mitra/csFall2018/cs329/lectures/fig1.gif

OSI layer

- Abstraction layer for computer network
- No security at all
  - Before TLS/SSL kicks in

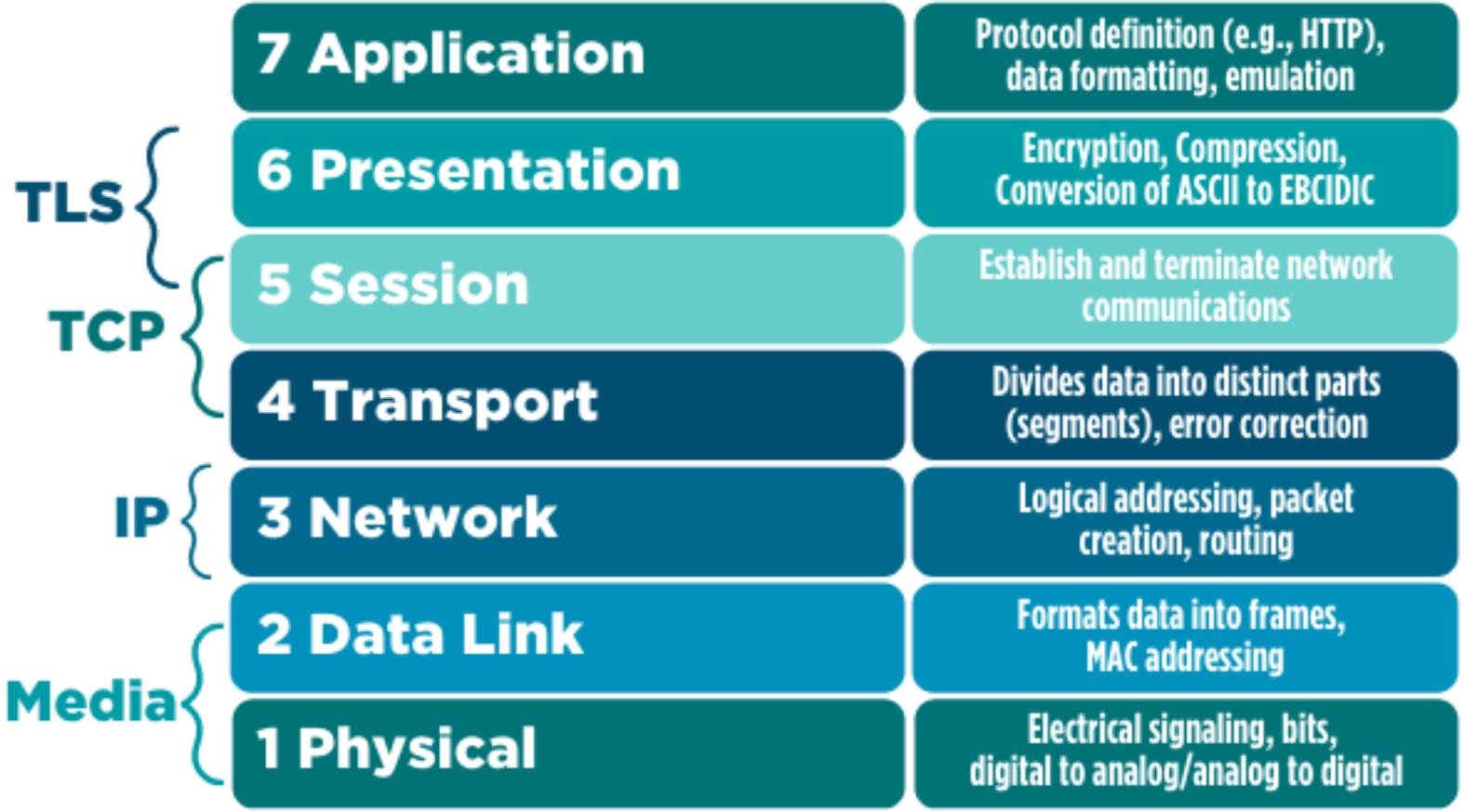
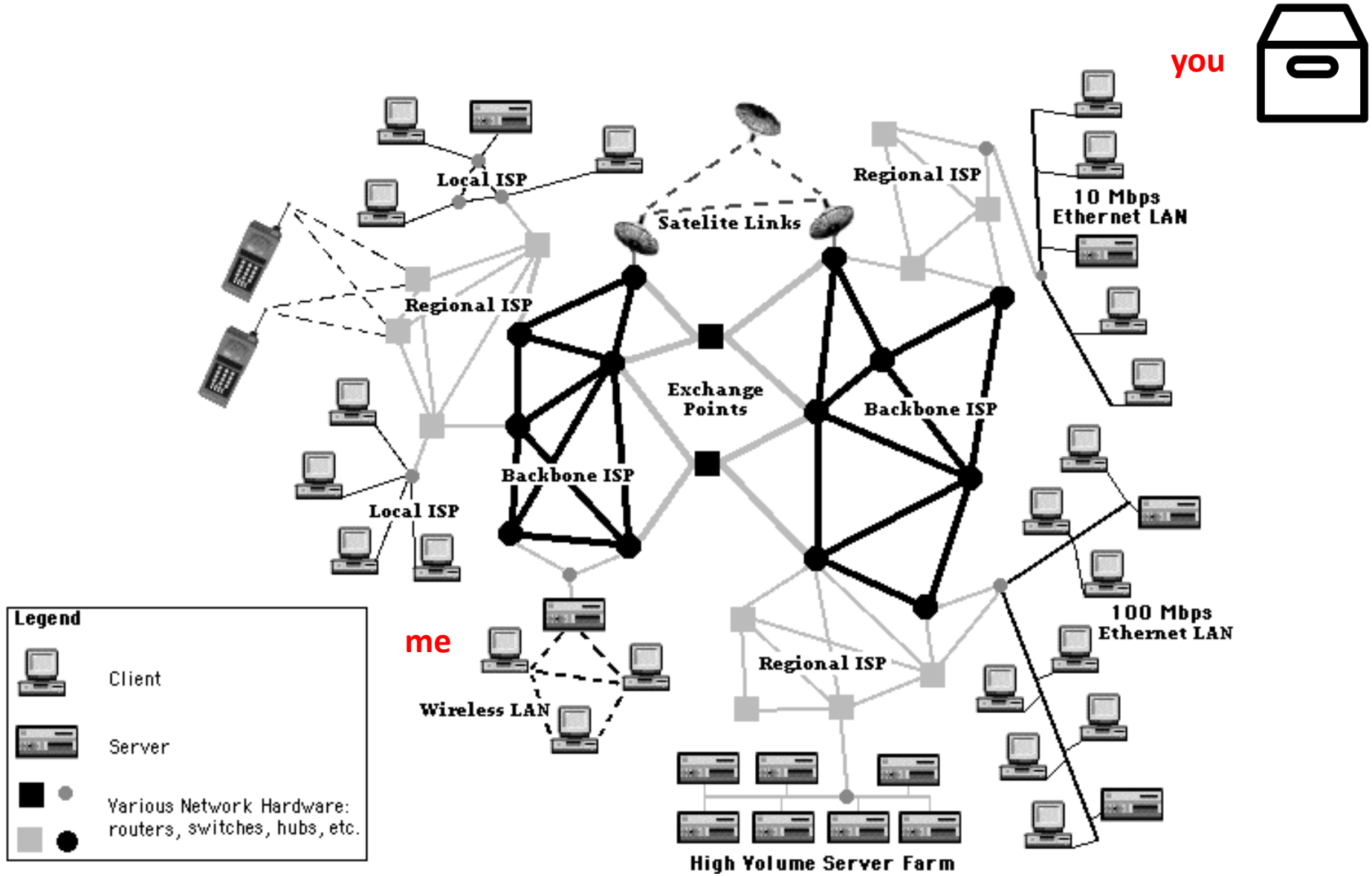


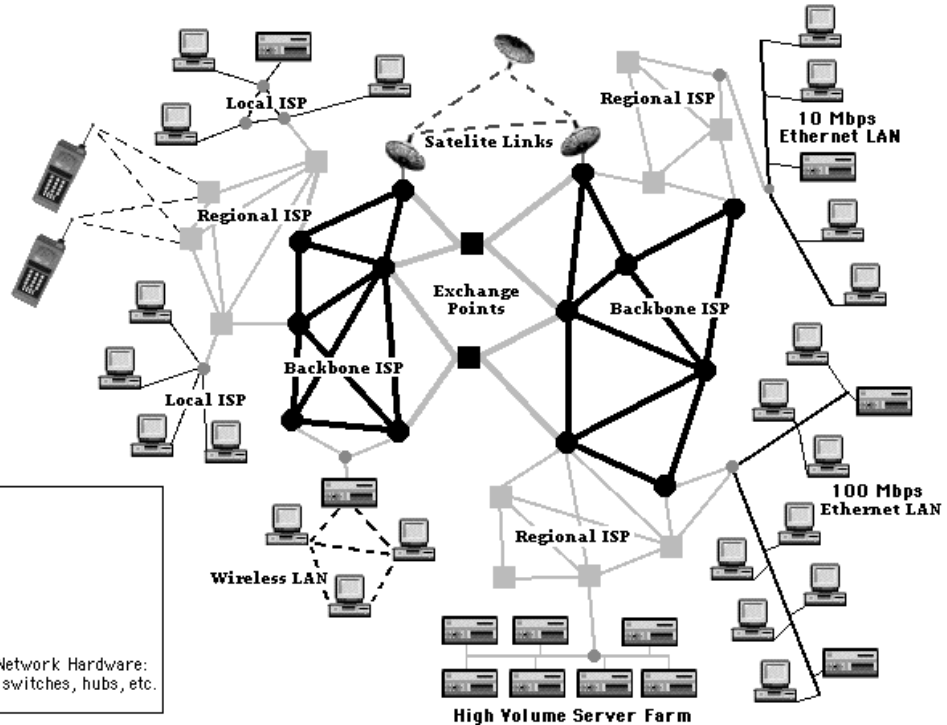
Image is from:
https://comptiacdn.azureedge.net/webcontent/images/default-source/blogs/osi-rm-4.png

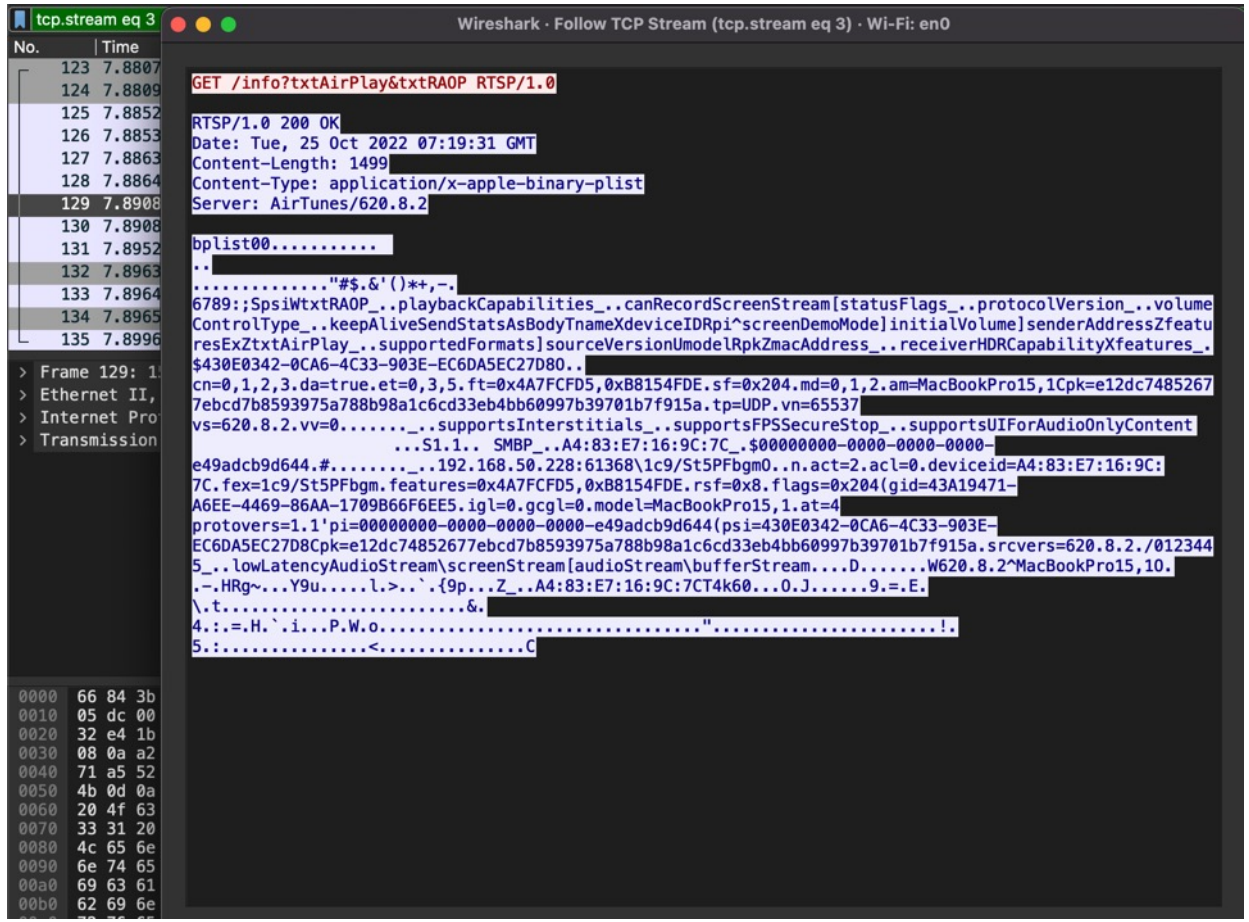
How is a Packet Routed?



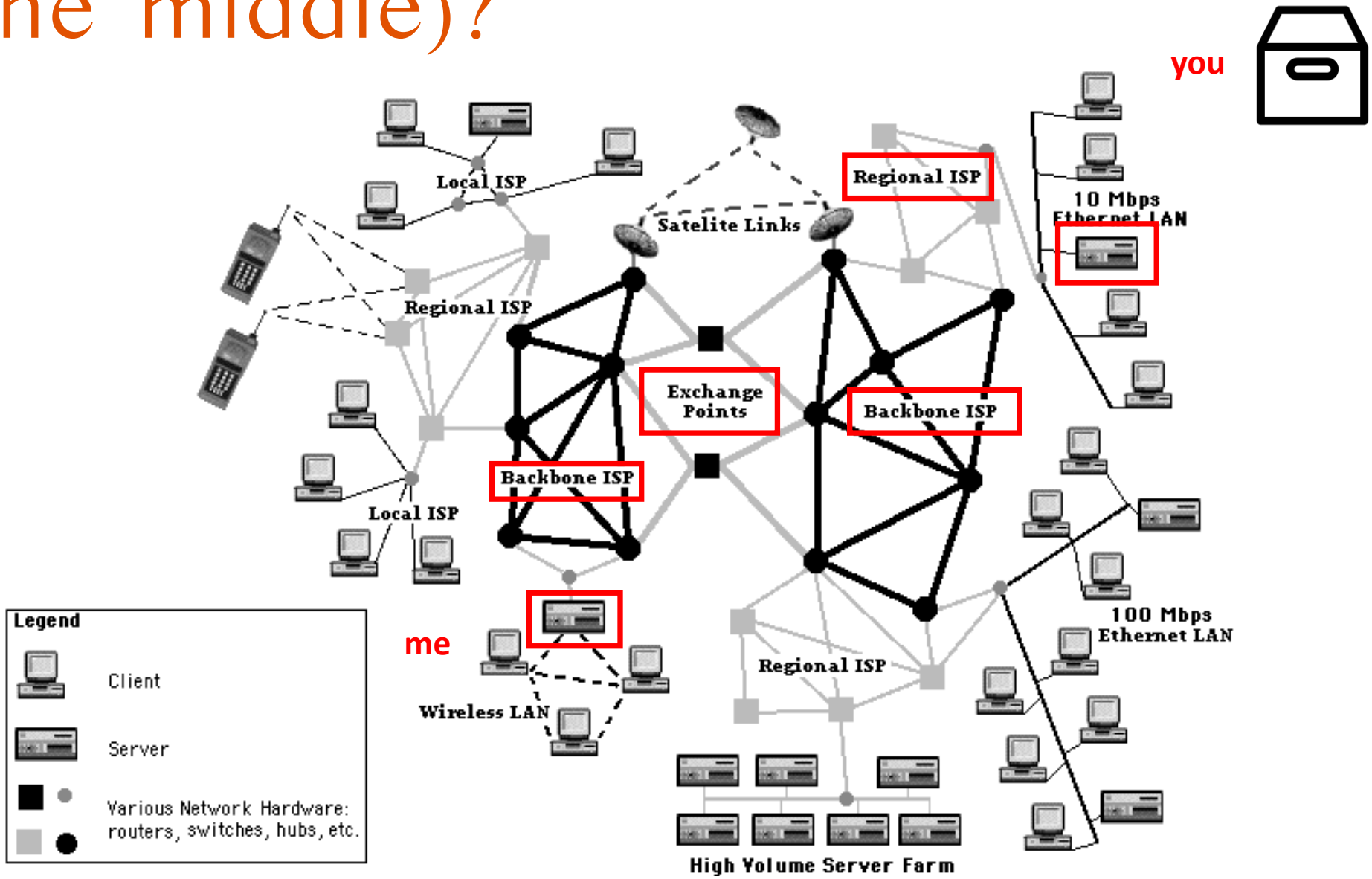
TCP Communication

- Totally insecure
  - Any router in the middle can see your packet content!!





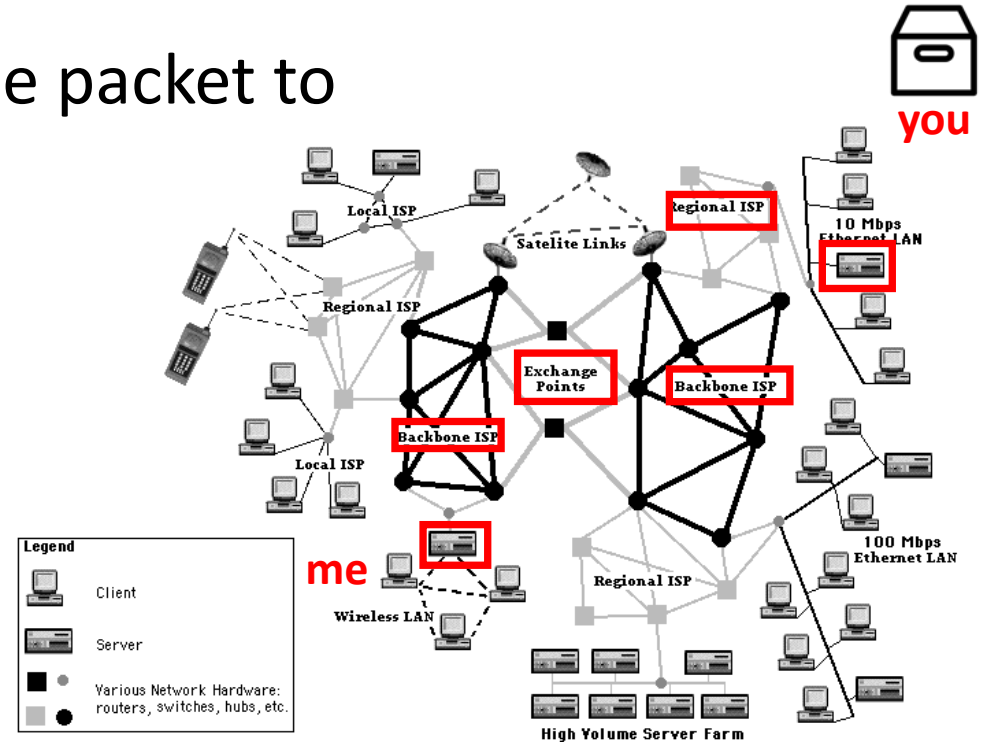
How Can We Trust Routers (in the middle)?



Router Decides Where the Packet Should Go as a Next Step

- What if the router in the middle send the packet to
  - Weird location???

- This is totally possible
  - Yes, router will check the IP but
  - What if routers are malicious???
  - There is no restriction in doing such..



The Internet w/o SSL/TLS

Everybody in the middle knows that I searched ‘dogs’ and they also know the search result…

The Internet with SSL/TLS

Middle mans never know DH exchange keys!!

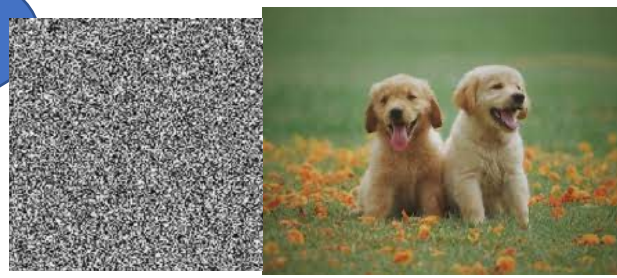
Check certificate, exchange keys, apply encryption with HMAC

Search “Dog”

0x1ce42780dfa1cea
089a9ea00de059ef5

I know these two are communicating but not about the secret key…

Search “Dog”



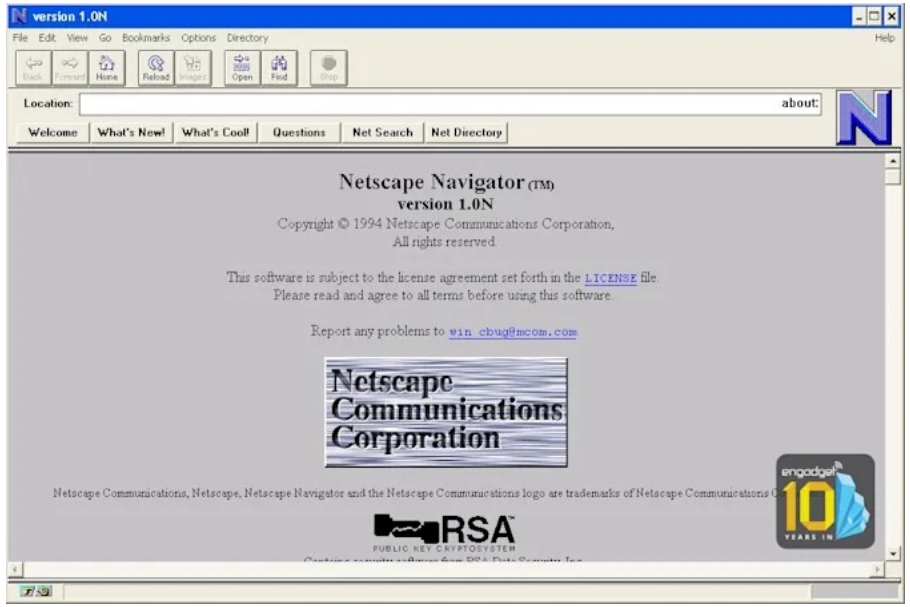
The middlemen will only see the encrypted contents
They will never know the secret key

Secure Socket Layer and Transport Layer Security

- Developed by Netscape in 1995..

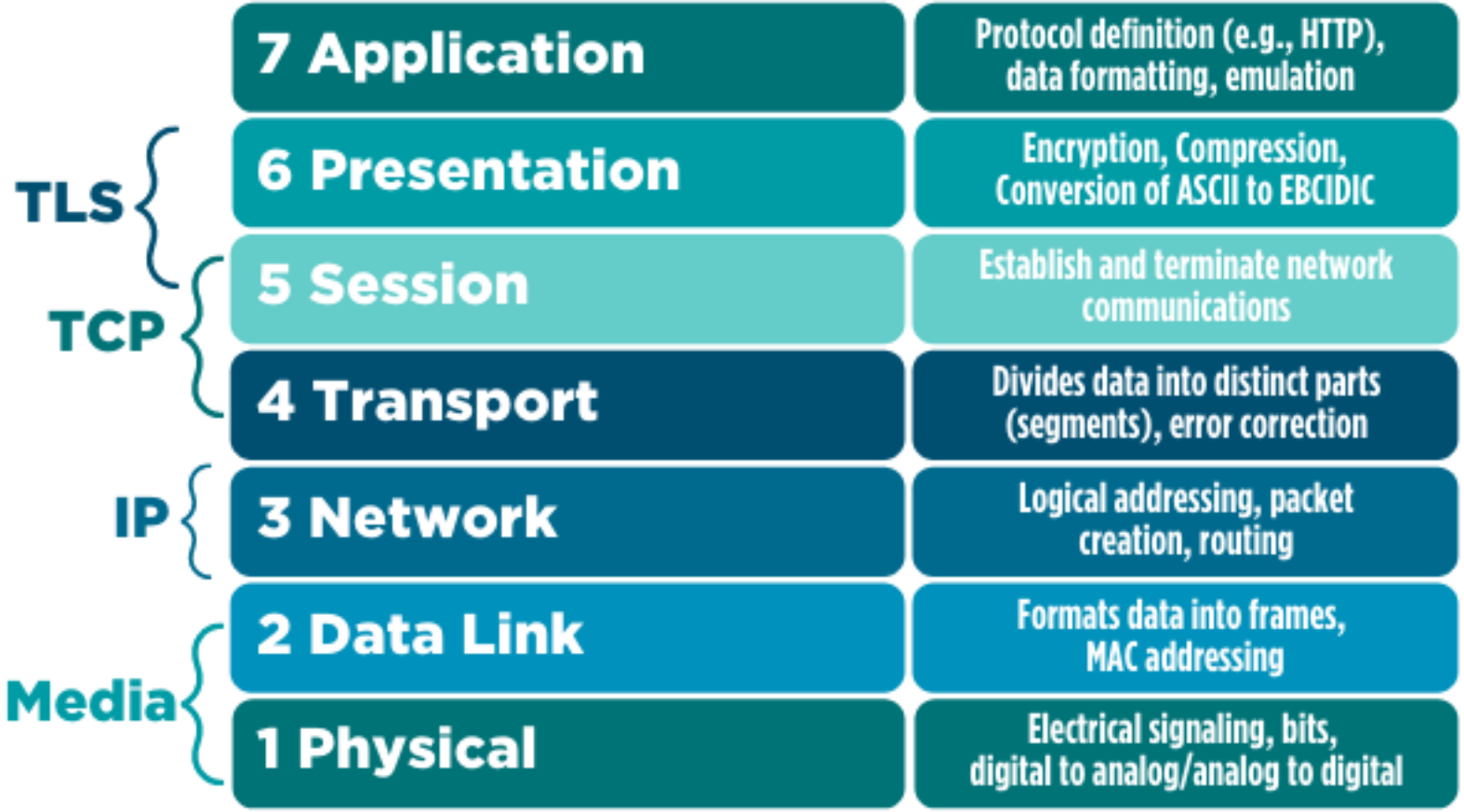
- The Internet is insecure, as a web browser, we need a way to communicate securely!

- Later, standardized by IETF as Transport Layer Security
  - https://www.ietf.org/rfc/rfc2246.txt



Why Transport Layer Security?

- OSI Model

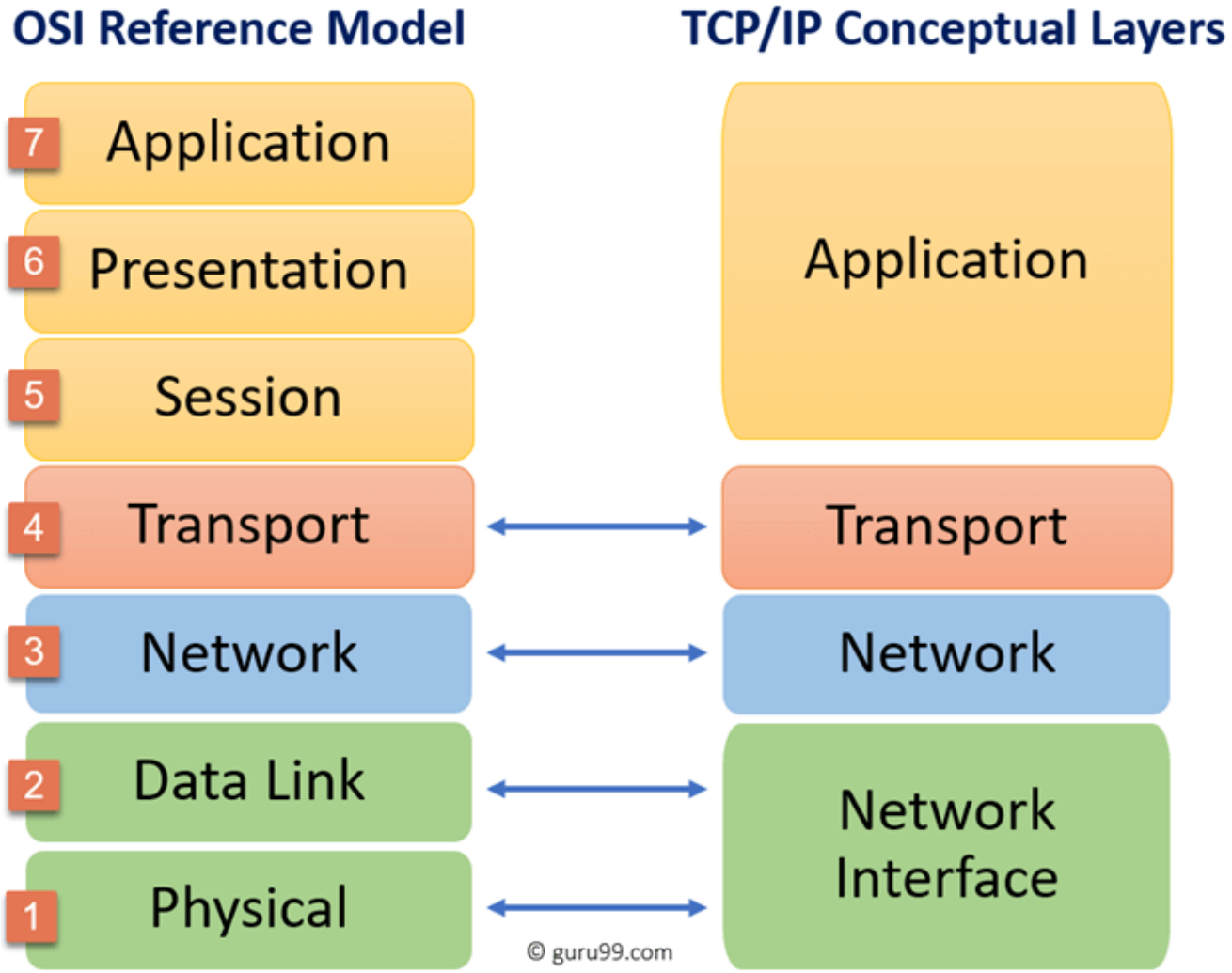


Why Transport Layer Security?

- OSI vs IP Layers

- TLS would like to be
  - Transparent to applications

If transparent,
Applications does not have to change their behavior for adopting security to the network protocols



What an SSL/TLS can do?

- Establish a secure communication channel between
  - Two ends on the Internet
    - Client/Server
    - Server/Server
    - Client/Client
  - We usually use this to verify the server entity
    - You will be authenticated by the server via login interface..

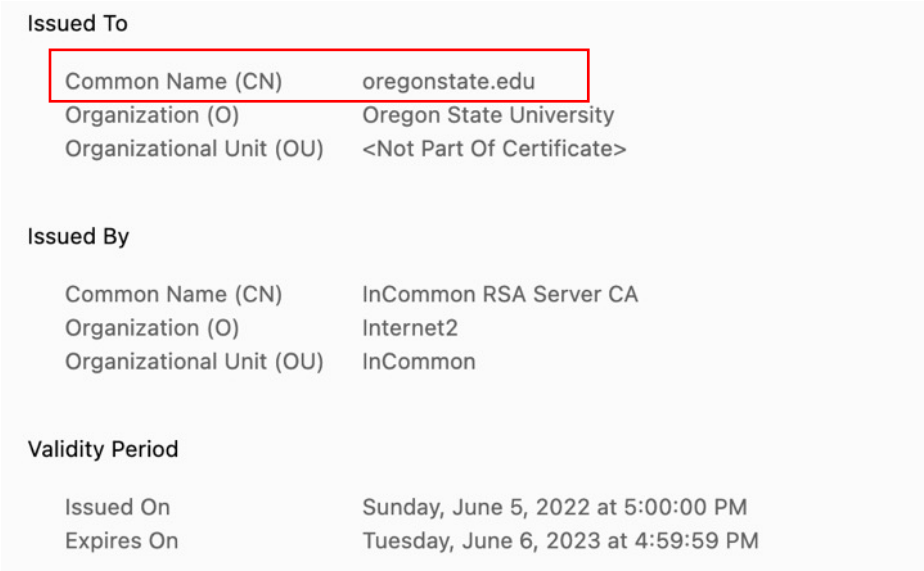
What an SSL/TLS can do?

- End-to-end secure communication channel
  - Authenticity (Who are at each ends?)
  - Secure key exchange (Nobody else than those two know the key!!)
  - Confidentiality (No one else than those two can see the message)
  - Integrity (We know if the message is altered)

SSL/TLS Steps

- Authenticity
  - We usually use SSL/TLS to verify the server identity
  - E.g., is the server www.oregonstate.edu? Or www.google.com?

- We do this by checking the digital certificate



Issued To

Common Name (CN)	oregonstate.edu
Organization (O)	Oregon State University
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	InCommon RSA Server CA
Organization (O)	Internet2
Organizational Unit (OU)	InCommon

Validity Period

Issued On	Sunday, June 5, 2022 at 5:00:00 PM
Expires On	Tuesday, June 6, 2023 at 4:59:59 PM

SSL/TLS verifies if the server has PKI-certified digital certificate with the same domain name

SSL/TLS Steps

Client

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

Server

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

1. Client Hello

- The first message that client sends to the server
  - Sends SSL/TLS version, random number, available cipher suite, etc.

```
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.?E ......,.
00000020  bf 27 f0 52 b0 97 6c f0  6c a2 a9 20 bc b7 86 80  .'.R..l. l.. ....
00000030  f2 f1 71 9f e0 7e 7e 4c  c2 51 88 e7 72 2d e0 3c  ..q..~~L .Q..r-.<
00000040  ca cc fa 2c 99 dc b9 56  d0 80 bd 91 00 62 13 02  ...,...V .....b..
00000050  13 03 13 01 c0 30 c0 2c  c0 28 c0 24 c0 14 c0 0a  .....0., .(.$....
00000060  00 9f 00 6b 00 39 cc a9  cc a8 cc aa ff 85 00 c4  ...k.9.. ........
00000070  00 88 00 81 00 9d 00 3d  00 35 00 c0 00 84 c0 2f  .......= .5...../
00000080  c0 2b c0 27 c0 23 c0 13  c0 09 00 9e 00 67 00 33  .+.'.#.. .....g.3
00000090  00 be 00 45 00 9c 00 3c  00 2f 00 ba 00 41 c0 11  ...E...< ./...A..
000000A0  c0 07 00 05 00 04 c0 12  c0 08 00 16 00 0a 00 ff  ........ ........
000000B0  01 00 00 95 00 2b 00 09  08 03 04 03 03 03 02 03  .....+.. ........
000000C0  01 00 33 00 26 00 24 00  1d 00 20 ba 53 26 b5 f2  ..3.&.$. .. .S&..
000000D0  19 5d b0 e0 b5 f4 30 0c  73 e9 2a 1d 86 72 d5 29  .]....0. s.*..r.)
000000E0  6e fc 32 3f d3 0f 31 d6  e2 57 61 00 00 00 18 00  n.2?..1. .Wa.....
000000F0  16 00 00 13 77 77 77 2e  6f 72 65 67 6f 6e 73 74  ....www. oregonst
00000100  61 74 65 2e 65 64 75 00  0b 00 02 01 00 00 0a 00  ate.edu. ........
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  ........ ......h2
00000140  08 68 74 74 70 2f 31 2e  31                       .http/1. 1
```

```
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: bcb78680f2f1719fe07e7e4cc25188e7722de03ccaccfa2c99dcb956d080bd9
    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)
```

1. Client Hello

- Send 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)
```

2. Server Hello

- The first message that the server sends back to the client
  - Sends SSL/TLS version, random number, choose 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)
```

- ECDHE_RSA_AES_128_GCM_SHA256
  - The server chose this cipher from client’s availability

3. Server Certificate

- The server sends the certificate information to the client

```
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=Oregon…
            Certificate Length: 1533
            Certificate: 308205f9308203e1a0030201020210472 0d0fa85461a7e17a1640291846374300d06092a… (id-at-commonName=InCommon RSA Server CA,id-at-organizationalUn…
            Certificate Length: 1413
            Certificate: 308205813082046 9a0030201020210397 2443af922b751d7d36c10dd313595300d06092a… (id-at-commonName=USERTrust RSA Certification Authority,id-at-o…
            Certificate Length: 1078
            Certificate: 308204323082031aa0030201020201300d06092a864886f70d0101050500307b310b30… (id-at-commonName=AAA Certificate Services,id-at-organizationNa…
```

- They send a full chain (PKI) of digital certificates

Key Exchange / Verifying Signature

- Previously, client chooses a key and send an encrypted key to server

- 1) The client knows the server's public key written in the certificate
- 2) The client chooses a random key and
- 3) encrypt that with the server's public key and
- 4) send that to the server

- Only the server can decrypt the key, so it's OK

SSL/TLS Steps

- Previously, client chooses a key and send an encrypted key to server
- Only the server can decrypt the key, so it's OK

- Suppose the server's private key has been stolen in 3 years later
- An attacker has encrypted packet capture
- Can they decrypt the data?
  - YES
  - They know the private key
  - They can decrypt the secret key
  - They can decrypt all the messages

This scheme does not guarantee Forward Secrecy

What is Forward Secrecy?

- Forward Secrecy / Perfect Forward Secrecy
  - Even if the long-term key (e.g., server's private key) has been compromised
  - Can we keep all the old communication secure?

- Software/hardware vulnerabilities and hacking incidents makes protecting private key difficult
  - Heartbleed (https://heartbleed.com/)
  - CVE-2014-0160

SSL/TLS with Ephemeral DH

- If we use fixed private value for DH for all, then, that value could be stolen via vulnerability, hacking, etc..

- Instead, we will generate the private value every time we make a connection, and never reuse the value
  - User A secretly chooses a, send $A = g^a \bmod p$
  - User B secretly chooses b, send $B = g^b \bmod p$
- i.e., we will choose different a and b each time…

SSL/TLS with ECDHE

- ECDHE
  - Elliptic-curve Diffie—Hellman Ephemeral
- Both client and the server will generate new a and b, respectively
- Even if one session is compromised, attackers cannot get the shared secret if they don't know b for other sessions

SSL/TLS Steps

Client

- 1. Client hello

Server

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

4. Server 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

```
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 333
    Handshake Protocol: Server Key Exchange
      Handshake Type: Server Key Exchange (12)
      Length: 329
      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…
```

5. Server Hello Done

- Indicating 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 Steps

Client

- 1. Client hello

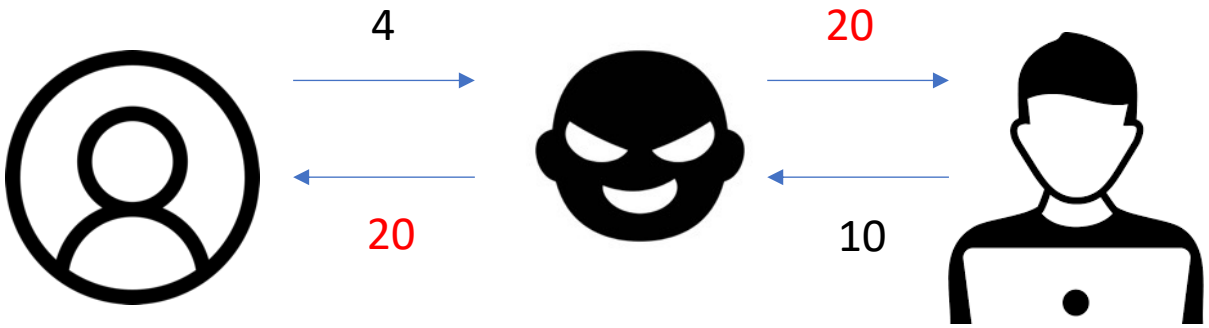
Server

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

Now, a client can
1) Verify server signature and
2) Share a secret via DH!

Diffie-Hellman Weakness: Man-in-the-middle Attack

- 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 Steps

Client

Server

- 1. Client hello
- 2. Server hello
- 3. Server Certificate
- 4. Server Key Exchange
  - Shares DH material, signed by the public key
- 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

6. Client Key Exchange

- The client also shares ECDHE material to the server
  - After doing this, two will share a secret
  - We will generate 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…
```

The client and the server will share a secret after sending this message because Server sent ‘b’ at the step 5. Server Key Exchange

6-1. Client Generates Session Keys

- Now the client knows both ‘a’ and ‘b’ of ECDHE key exchange
  - That is, the client knows the shared secret
- Then, the client 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

7. Change Cipher Spec (client)

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

```
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 I have keys to encrypting data and generating MACs.
Therefore, the server, “let’s communicate securely”!

8. Encrypted Handshake Message

- Encrypted data of previous messages…
  - Purpose: Let the server verify if the client generated 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

8. Encrypted Handshake Message

- Encrypted data of pre
  - Purpose: Let the serv

```
Change Cipher Spec Message
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

0060  04 8e cf 7b 83 8a 37 7b  cd e5 62 cd aa 28 ad 37   ···{··7{ ··b··(·7
0070  95 82 44 29 63 b3 4d 14  03 03 00 01 01 16 03 03   ··D)c·M· ********
0080  00 28 00 00 00 00 00 00  00 00 29 94 d9 97 f6 c8   ·(****** ··)*****
0090  77 dd 20 a2 82 4c 46 49  dc 3e 4c af a9 3b d9 38   w· ··LFI ·>L··;·8
00a0  37 a6 45 12 5f 88 5a a1  21 79                     7·E·_·Z· !y
```

- Compute a SHA256 hash of a concate
  SHA384). This means the Client Hello,
  Exchange messages. Note that you sh
  bytes belonging to the TLS Record hea
- Compute PRF(master_secret, "client fi
- 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 Steps

Client

Server

- 1. Client hello
- 2. Server hello
- 3. Server Certificate
- 4. Server Key Exchange
- 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

9-0. Check Client's Encrypted Handshake Message

- Server verifies client's encrypted handshake message first
  - 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

9. 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 I have keys to encrypting data and generating MACs.
Therefore, the server, “let’s communicate securely”!

10. Encrypted Handshake Message

- Encrypted data of previous messages…
  - Purpose: Let the client verify if the server generated keys correctly

    - 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)

```
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
```

11. Application Data

- Server/Client will send encrypted data to the client
  - Both always sends [encrypted data] [MAC]

  - Server will use server_write_key and server_write_mac_key
  - Client will use client_write_key and client_write_mac_key

ECDHE-RSA-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

Why no Man-in-the-middle?

- Client verifies server's certificate sent at step 3
- Client verifies server's key material using the public key in the cert

- Certificate guarantees the authenticity of
  - Server domain name (e.g., google.com)
  - Server's DH key material

- This means, if DH exchange was done,
  - Only the server knows the shared secret
  - Except the client itself..

SSL/TLS Steps

Client

- 1. Client hello

Server

- 2. Server hello
- 3. Server Certificate
- 4. Server Key Exchange
- 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

Can we guess the key?

- Key generation is designed to make guessing the key difficult

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

- Server chooses their own random
- Client chooses their own random
- Shared secret is naturally looks like a random number (DH)
- PRF, we use either SHA256 or SHA384… good luck, hackers..

Why we hash all the previous messages??

- Verification of the final hash means
  - Server/Client sent/received the same messages
  - Server/Client have generated the same shared secret (via DH)
  - Server/Client have generated the same set of encryption/MAC keys

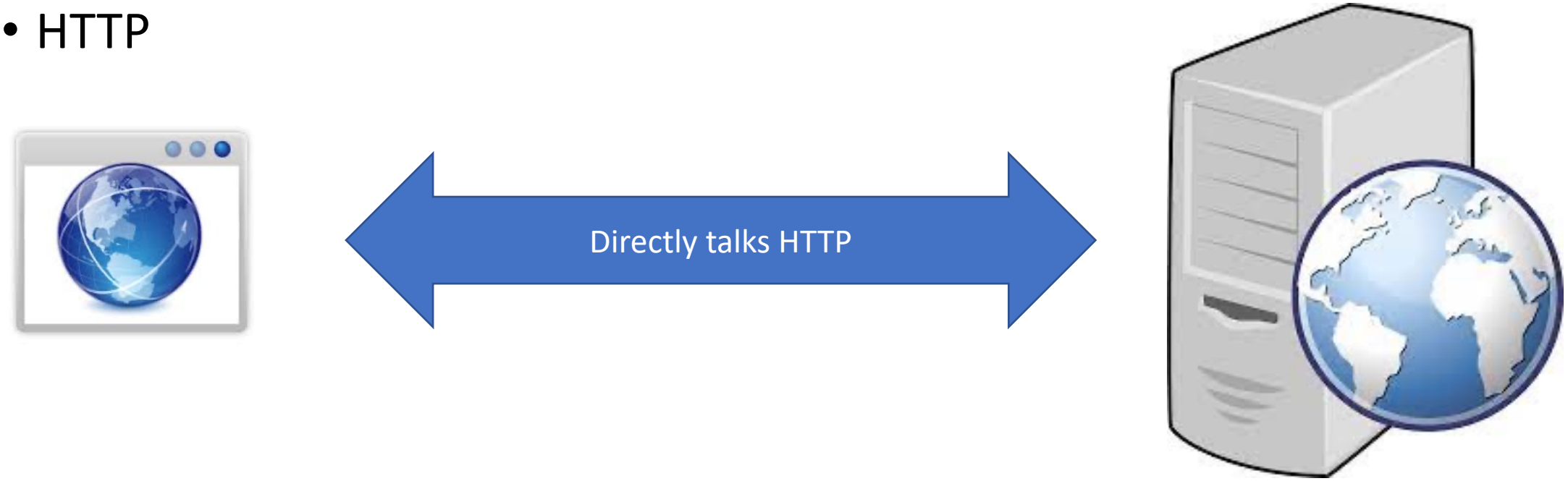
- Compute a SHA256 hash of a concatenation of all the handshake communications (or SHA384 if the PRF is based on SHA384). This means the Client Hello, Server Hello, Certificate, Server Key Exchange, Server Hello Done and Client Key Exchange messages. Note that you should concatenate only the handshake 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

How Do We USE SSL/TLS?

- HTTP(s)
  - 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
- How?

A Web Server Example

- HTTP



A Web Server Example

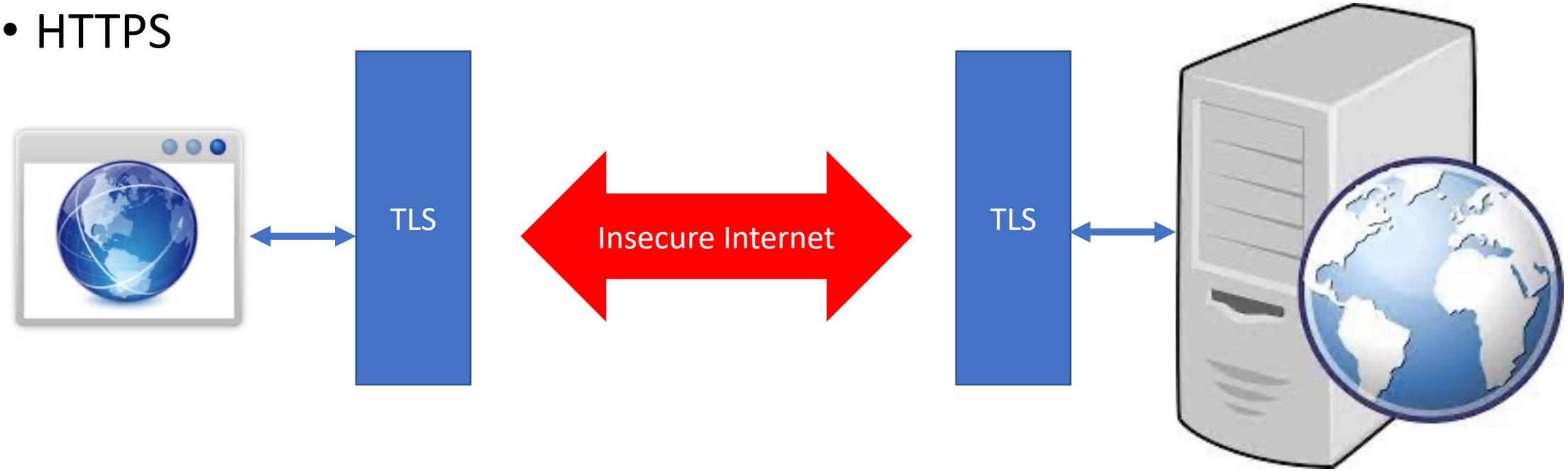
```
GET / HTTP/1.0
```

```
HTTP/1.0 200 OK
Date: Tue, 25 Oct 2022 12:53:12 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO
P3P: CP="This is not a P3P policy! S
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
```

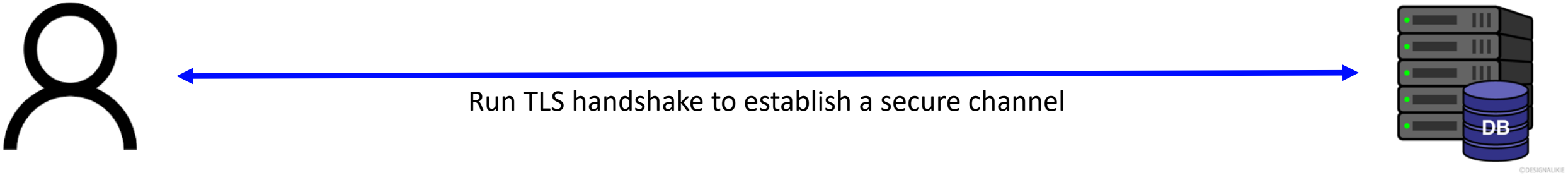
This is how HTTP works

A Web Server Example

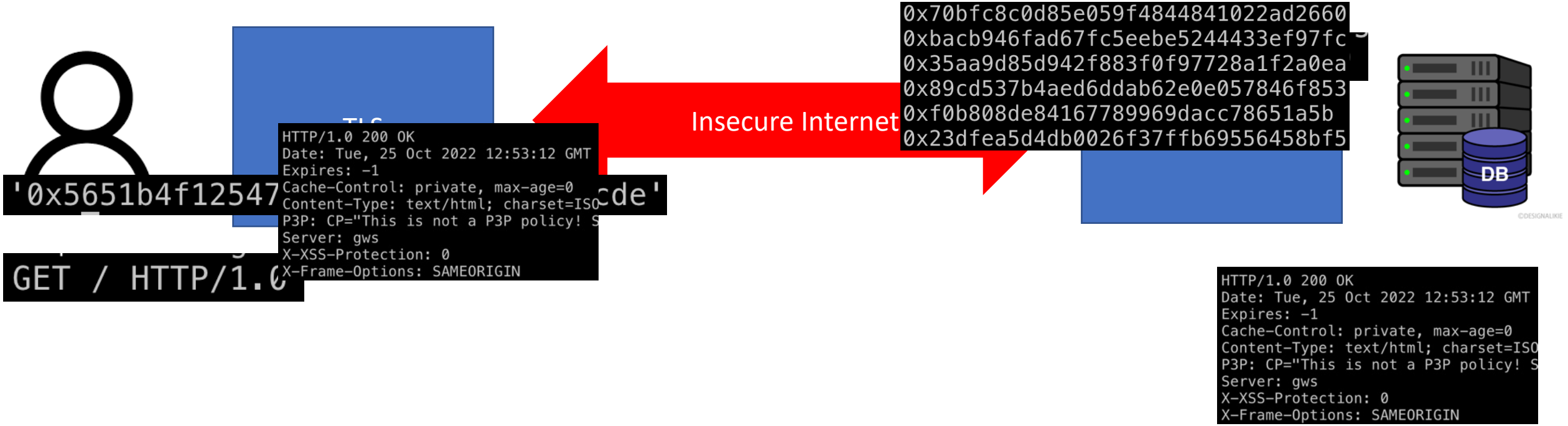
- HTTPS



A Web Server Example



A Web Server Example



HTTP/1.0 200 OK
Date: Tue, 25 Oct 2022 12:53:12 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO
P3P: CP="This is not a P3P policy! S
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN

HTTP Packets

4	0.010756057	10.248.25.87	142.250.69.196	HTTP	144 GET / HTTP/1.1

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: curl/7.81.0
Accept: */*

HTTP/1.1 200 OK
Date: Tue, 25 Oct 2022 13:25:43 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2022-10-25-13; expires=Thu, 24-Nov-2022 13:25:43 GMT; path=/; domain=.google.com;
Secure
Set-Cookie: AEC=AakniGOAPvX70HdROvGjd5tdhzmMk-ZntDxb9jZGhAdPNSmqmwQc2AumlRI; expires=Sun, 23-Apr-2023
13:25:43 GMT; path=/; domain=.google.com; Secure; HttpOnly; SameSite=lax
Set-Cookie: NID=511=MkIeDpP817QKD-9oufZM9-
MAnHFlDpvagPc6jwK6l-2onKyCQID83aSymrg5ss1SUexUDpaSsNb9MrcxpnaXhezc9engEZrNmX4qgoG7Zodt4Fy-
HP9FQI6DbeY6GLGCma0MBOnUmze5m6Ys-i6jSvc6WFJUkye67iOSgFuG72c; expires=Wed, 26-Apr-2023 13:25:43 GMT;
path=/; domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

348f
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta
content="Search the world's information, including webpages, images, videos and more. Google has many
special features to help you find exactly what you're looking for." name="description"><meta
content="noodp" name="robots"><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta
content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</
title><script nonce="zhUe3NfmQtn_Ha4HtJSi3A">(function(){window.google={kEI:'1-
NXY8iGIs2T0PEP5fy9CA',kEXPI:'0,18167,1284369,56873,6059,206,4804,2316,383,246,5,5367,1123753,1197698,703
,302561,77529,16114,19398,9286,22431,1361,284,12036,17579,4998,13228,3847,6885,3737,22741,5081,1594,1278
,2742,149,1943,1983,214,4100,109,3405,606,2023,1777,520,14670,605,2622,2845,7,4808,791,28171,1851,2614,1
2710,432,3,1590,1,5444,149,11323,2652,4,1528,2304,7039,22023,5708,7356,16639,16808,1435,5827,2530,4094,1
7,4035,3,3541,1,14263,27894,2,14019,2373,342,4931,6470,9868,1755,5679,1021,2380,22668,6074,4568,6258,234
18,1252,5835,14968,4332,2204,5280,445,2,2,1,10956,15676,8155,7381,2,3,15965,873,6577,3048,10007,9,1921,5
784,3995,19130,12192,4832,17016,122,700,4,1,2,2,2,2,8652,107,821,4337,785,1765,978,3023,2756,3546,2,2017
,14,82,950,1758,168,1014,751,202,1866,125,6416,1,1015,51,2197,488,922,613,1323,346,109,364,466,683,899,2
```

Packet 4.1 client pkt, 9 server pkts, 1 turn. Click to select.

Entire conversation (16kB) | Show data as ASCII | Stream 0

HTTPS Packets

```
40  4.482276498   10.248.25.87   142.250.69.196   TLSv1…   146  Change Cipher Spec, Application Data

> Frame 40: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface wlp0s20f3, id 0
> Ethernet II, Src: IntelCor_6c:c3:5c (98:2c:bc:6c:c3:5c), Dst: IETF-VRRP-VRID_01 (00:00:5e:00:01:01)
> Internet Protocol Version 4, Src: 10.248.25.87, Dst: 142.250.69.196
> Transmission Control Protocol, Src Port: 44148, Dst Port: 443, Seq: 518, Ack: 4303, Len: 80
v Transport Layer Security
  > TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  v TLSv1.3 Record Layer: Application Data Protocol: http-over-tls
      Opaque Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 69
      Encrypted Application Data: 8684730f1612223931bb38393d0213d0b0e0dd0aa2ea6b908fab28f13c4ef8f8beba27d2…
      [Application Data Protocol: http-over-tls]

0000  00 00 5e 00 01 01 98 2c  bc 6c c3 5c 08 00 45 00   ··^····, ·l·\··E·
0010  00 84 73 f7 40 00 40 06  cd 6f 0a f8 19 57 8e fa   ··s·@·@· ·o···W··
0020  45 c4 ac 74 01 bb de 97  e7 2a d2 93 69 1c 80 18   E··t···· ·*··i···
0030  01 f5 f9 83 00 00 01 01  08 0a 09 e8 fa 1d 62 2d   ........ ......b-
0040  cc 7d 14 03 03 00 01 01  17 03 03 00 45 86 84 73   ·}····· ····E··s
0050  0f 16 12 22 39 31 bb 38  39 3d 02 13 d0 b0 e0 dd   ···"91·8 9=******
0060  0a a2 ea 6b 90 8f ab 28  f1 3c 4e f8 f8 be ba 27   ···k···( ·<N····'
0070  d2 67 e8 e4 2e 71 28 62  13 11 7d fb a1 58 fc 0c   ·g··.q(b ··}··X··
0080  1d 5b da 7c 91 3f 6d 9f  bb 1d 6c 0b 67 ce 18 23   ·[·|·?m· ··l·g··#
0090  b9 d8                                              ··
```

```
00000000  16 03 01 02 00 01 00 01  fc 03 03 cb 6c ea fb 9f   ........ ....l...
00000010  71 f0 1d 41 6a 19 4d 76  10 3b 3a e2 eb e5 1d 63   q..Aj.Mv .;:....c
00000020  92 d2 da d2 d2 46 98 73  16 b6 75 20 f8 43 a8 eb   .....F.s ..u .C..
00000030  05 41 47 7e 53 47 37 ad  39 78 32 5a f7 88 ae c1   .AG~SG7. 9x2Z....
00000040  64 77 d6 51 e6 e4 ac ef  03 26 6a a2 00 3e 13 02   dw.Q.... .&j..>..
00000050  13 03 13 01 c0 2c c0 30  00 9f cc a9 cc a8 cc aa   .....,.0 ........
00000060  c0 2b c0 2f 00 9e c0 24  c0 28 00 6b c0 23 c0 27   .+./...$ .(.k.#.'
00000070  00 67 c0 0a c0 14 00 39  c0 09 c0 13 00 33 00 9d   .g.....9 .....3..
00000080  00 9c 00 3d 00 3c 00 35  00 2f 00 ff 01 00 01 75   ...=.<.5 ./.....u
00000090  00 00 00 13 00 11 00 00  0e 77 77 77 2e 67 6f 6f   ........ .www.goo
000000A0  67 6c 65 2e 63 6f 6d 00  0b 00 04 03 00 01 02 00   gle.com. ........
000000B0  0a 00 16 00 14 00 1d 00  17 00 1e 00 19 00 18 01   ........ ........
000000C0  00 01 01 01 02 01 03 01  04 33 74 00 00 00 10 00   ........ .3t.....
000000D0  0e 00 0c 02 68 32 08 68  74 74 70 2f 31 2e 31 00   ....h2.h ttp/1.1.
000000E0  16 00 00 00 17 00 00 00  31 00 00 00 0d 00 2a 00   ........ 1.....*.
000000F0  28 04 03 05 03 06 03 08  07 08 08 08 09 08 0a 08   (....... ........
00000100  0b 08 04 08 05 08 06 04  01 05 01 06 01 03 03 03   ........ ........
00000110  01 03 02 04 02 05 02 06  02 00 2b 00 05 04 03 04   ........ ..+.....
00000120  03 03 00 2d 00 02 01 01  00 33 00 26 00 24 00 1d   ...-.... .3.&.$..
00000130  00 20 31 6b 2c 95 bb 6c  06 fb 83 c0 b9 82 1d ee   . 1k,..l ........
00000140  5f 85 0c da 5c 31 9d b6  dc 00 72 d5 06 08 90 d4   _...\1.. ..r.....
00000150  85 60 00 15 00 af 00 00  00 00 00 00 00 00 00 00   .`...... ........
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
000001A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
000001B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
000001C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
000001D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
000001E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
000001F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00000200  00 00 00 00 00                                      .....
```

Application Data

id 0

01)

8beba27d2…

```
00000000  16 03 03 00 7a 02 00 00  76 03 03 82 f4 4b ce 9f   ....z... v....K..
00000010  b3 46 18 0f 31 0b 53 1f  4d a0 e6 17 07 3a 83 f6   .F..1.S. M....:..
00000020  06 c0 4c a2 eb 2c a3 6f  b3 c2 f8 20 f8 43 a8 eb   ..L..,.o ... .C..
00000030  05 41 47 7e 53 47 37 ad  39 78 32 5a f7 88 ae c1   .AG~SG7. 9x2Z....
00000040  64 77 d6 51 e6 e4 ac ef  03 26 6a a2 13 02 00 00   dw.Q.... .&j.....
00000050  2e 00 33 00 24 00 1d 00  20 85 51 b9 c0 6e b7 59   ..3.$...  .Q..n.Y
00000060  4e 79 54 6a dc f5 c2 5b  7d 0b 5e 59 a7 50 a4 37   NyTj...[ }.^Y.P.7
00000070  58 20 c8 6a d6 58 7d 55  31 00 2b 00 02 03 04 14   X .j.X}U 1.+.....
```

```
00000000  16 03 01 02 00 01 00 01  fc 03 03 cb 6c ea fb 9f
00000010  71 f0 1d 41 6a 19 4d 76  10 3b 3a e2 eb e5 1d 63
00000020  92 d2 da d2 d2 46 98 73  16 b6 75 20 f8 43 a8 eb
00000030  05 41 47 7e 53 47 37 ad  39 78 32 5a f7 88 ae c1
00000040  64 77 d6 51 e6 e4 ac ef  03 26 6a a2 00 3e 13 02
00000050  13 03 13 01 c0 2c c0 30  00 9f cc a9 cc a8 cc aa
00000060  c0 2b c0 2f 00 9e c0 24  c0 28 00 6b c0 23 c0 27
00000070  00 67 c0 0a c0 14 00 39  c0 09 c0 13 00 33 00 9d
00000080  00 9c 00 3d 00 3c 00 35  00 2f 00 ff 01 00 01 75
00000090  00 00 00 13 00 11 00 00  0e 77 77 77 2e 67 6f 6f
000000A0  67 6c 65 2e 63 6f 6d 00  0b 00 04 03 00 01 02 00
000000B0  0a 00 16 00 14 00 1d 00  17 00 1e 00 19 00 18 01
000000C0  00 01 01 01 02 01 03 01  04 33 74 00 00 00 10 00
000000D0  0e 00 0c 02 68 32 08 68  74 74 70 2f 31 2e 31 00
000000E0  16 00 00 00 17 00 00 00  31 00 00 00 0d 00 2a 00
000000F0  28 04 03 05 03 06 03 08  07 08 08 08 09 08 0a 08
00000100  0b 08 04 08 05 08 06 04  01 05 01 06 01 03 03 03
00000110  01 03 02 04 02 05 02 06  02 00 2b 00 05 04 03 04
00000120  03 03 00 2d 00 02 01 01  00 33 00 26 00 24 00 1d
00000130  00 20 31 6b 2c 95 bb 6c  06 fb 83 c0 b9 82 1d ee
00000140  5f 85 0c da 5c 31 9d b6  dc 00 72 d5 06 08 90 d4
00000150  85 60 00 15 00 af 00 00  00 00 00 00 00 00 00 00
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000001A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000001B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000001C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000001D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000001E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000001F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000200  00 00 00 00 00
```

```
00000000  16 03 03 00 7a 02 00 00  76 03 03 82 f4 4b ce
00000010  b3 46 18 0f 31 0b 53 1f  4d a0 e6 17 07 3a 83
00000020  06 c0 4c a2 eb 2c a3 6f  b3 c2 f8 20 f8 43 a8
00000030  05 41 47 7e 53 47 37 ad  39 78 32 5a f7 88 ae
00000040  64 77 d6 51 e6 e4 ac ef  03 26 6a a2 13 02 00
00000050  2e 00 33 00 24 00 1d 00  20 85 51 b9 c0 6e b7
00000060  4e 79 54 6a dc f5 c2 5b  7d 0b 5e 59 a7 50 a4
00000070  58 20 c8 6a d6 58 7d 55  31 00 2b 00 02 03 04
```

```
00000205  14 03 03 00 01 01 17 03  03 00 45 86 84 73 0f 16   ........ ..E..s..
00000215  12 22 39 31 bb 38 39 3d  02 13 d0 b0 e0 dd 0a a2   ."91.89= ........
00000225  ea 6b 90 8f ab 28 f1 3c  4e f8 f8 be ba 27 d2 67   .k...(.< N....'.g
00000235  e8 e4 2e 71 28 62 13 11  7d fb a1 58 fc 0c 1d 5b   ...q(b.. }..X...[
00000245  da 7c 91 3f 6d 9f bb 1d  6c 0b 67 ce 18 23 b9 d8   .|.?m... l.g..#..
00000255  17 03 03 00 29 ae d9 ce  dc e1 eb c5 15 ed ab 31   ....)... .......1
00000265  09 28 e9 65 87 98 4a 7a  76 e9 4b 19 f7 8a 12 d9   .(.e..Jz v.K.....
00000275  07 f3 87 8d 9d e1 dc 6e  af 3e 52 bd 94 81 17 03   .......n .>R.....
00000285  03 00 2c ff 1d 93 26 f3  b8 64 16 37 40 d9 4b 87   ..,...&. .d.7@.K.
00000295  56 6a 20 78 46 14 01 12  fd 1e f8 82 8e 01 44 53   Vj xF... ......DS
000002A5  b2 e6 c8 01 ca fe 25 86  d4 b4 39 1d 18 85 f9      ......%. ..9....
000002B4  17 03 03 00 1e 2d 05 11  4e c9 af f5 05 89 07 05   .....-.. N.......
000002C4  27 55 03 a0 0b 74 35 c7  25 d9 03 89 4e 97 87 70   'U...t5. %...N..p
000002D4  a0 ba 26 17 03 03 00 39  31 66 19 54 9b d1 e9 c5   ..&....9 1f.T....
000002E4  f4 bc 2f 43 ff 0d 91 be  e8 11 ef f9 90 35 07 7e   ../C.... .....5.~
000002F4  4c de 3e 05 b5 b6 2a 34  7b 83 9d b6 48 32 e5 a9   L.>...*4 {...H2..
00000304  17 12 f2 94 3c a2 27 2c  75 da 77 8f 98 71 6a 1d   ....<.', u.w..qj.
00000314  47                                                 G
```

```
000010CE  17 03 03 02 45 f8 f4 1d  68 b1 7e e5 a2 c6 1f ec   ....E... h.~.....
000010DE  2a 27 d0 d9 cb 69 5d 4a  31 7b d4 54 43 e2 8f e7   *'...i]J 1{.TC...
000010EE  e9 d0 d7 1e 8b 4f da 2a  8e 41 26 91 2a 27 d2 bc   .....O.* .A&.*'..
000010FE  a9 de 8f 07 57 b5 72 01  11 2f 42 c4 e9 8f 41 80   ....W.r. ./B...A.
0000110E  29 84 2b b7 8b db 8a a6  63 19 70 a3 c8 7c 28 85   ).+..... c.p..|(.
0000111E  17 00 86 d0 ea 02 30 f3  1f 8e 6b a0 c9 19 77 de   ......0. ..k...w.
0000112E  31 4f 61 e3 d8 4b 8e dc  c6 c7 f2 32 fa 70 f0 e1   1Oa..K.. ...2.p..
0000113E  bb af 9c 79 e0 a9 f1 50  6c da d7 e2 36 eb 0b bb   ...y...P l...6...
0000114E  09 f2 a3 7d a0 13 46 2e  3a 81 5c 77 d4 05 c5 2e   ...}..F. :.\w....
0000115E  6f ba 65 49 52 1d f5 0b  1b 7d db c5 f9 1d ab ec   o.eIR... .}......
0000116E  39 d3 40 0a 4b e3 f6 80  56 e2 e7 c5 d3 b8 df 79   9.@.K... V......y
0000117E  b5 8f 07 48 61 30 a8 19  08 00 f5 51 d1 20 a6 b8   ...Ha0.. ...Q. ..
0000118E  29 92 52 ae 46 89 ce 2d  43 a9 b1 ec 62 0f 69 f2   ).R.F..- C...b.i.
0000119E  ff 34 67 5f 92 94 9f 9a  3d e6 36 0c 73 b9 8f 5a   .4g_.... =.6.s..Z
000011AE  2c bb 91 24 fd 94 8f c4  72 f2 41 6a 49 86 f7 aa   ,..$.... r.AjI...
000011BE  8e 17 16 c6 0e 48 92 cf  7b b3 a5 74 ee b6 f4 f4   .....H.. {..t....
000011CE  cb 39 a6 f0 e1 15 a0 46  52 1c ab b9 ab ea d9 82   .9.....F R.......
000011DE  fb a2 77 08 3d 05 65 20  18 7f e3 dd 44 f4 2b 38   ..w.=.e  ....D.+8
000011EE  e7 23 9e 7f c6 29 83 dd  0b f0 e4 d0 b7 a9 fe 18   .#...).. ........
000011FE  83 8f 77 cc 9f 88 42 df  ad a2 41 76 8f 16 38 4e   ..w...B. ..Av..8N
0000120E  9f ea 72 24 c0 92 fd f0  b8 b3 05 2b f2 97 f4 6b   ..r$.... ...+...k
```

Why TLS is good?

- TLS establishes a secure communication channel
  - Over the insecure Internet
  - Adds authenticity, confidentiality, and integrity to the channel
- Applications don't have to change their protocol
  - Just wrap the protocol with TLS
- How can we make HTTP secure?
  - Wrap with TLS: https://...

How Can We Use TLS?

- Many libraries are available
  - OpenSSL
  - libsodium
  - bouncycastle
  - SSL/TLS support in many other languages (Python, etc.)

- Just wrap with this
  - What does it mean???

A Python Example

- Suppose you have a protocol that
  - 1) You need to send ‘Hello’ to the server
  - 2) You need to receive a number from the server
  - 3) You need to add 1 to the number and send back to the server

- Let’s write programs (server/client) for it.

Non-TLS Client

- The protocol
  - 1) You need to send ‘Hello’ to the server
  - 2) You need to receive a number from the server
  - 3) You need to add 1 to the number and send back to the server

```python
#!/usr/bin/env python3

import socket

def main():
    client_socket = socket.socket()
    client_socket.connect(('127.0.0.1', 31337))

    print("Sending Hello...")
    client_socket.send(b'Hello\n')

    b_number = client_socket.recv(5)
    s_number = str(b_number, 'utf-8')
    print("Received %s" % s_number.strip())

    i_number = int(s_number)
    i_answer = i_number + 1

    print("Returning %d" % i_answer)
    b_answer = bytes(str(i_answer), 'utf-8')
    client_socket.send(b_answer)

    result = client_socket.recv(10)
    print(result)

    client_socket.close()

if __name__ == '__main__':
    main()
```

Non-TLS Server

- The protocol
  - 1) You need to send ‘Hello’ to the server
  - 2) You need to receive a number from the server
  - 3) You need to add 1 to the number and send back to the server

```python
#!/usr/bin/env python3

import os
import random
import socket
import sys

def main():
    server_socket = socket.socket()
    server_socket.bind(('127.0.0.1', 31337))
    server_socket.listen(10)

    while True:
        conn, addr = server_socket.accept()
        if os.fork():
            conn.close()
        else:
            message = conn.recv(6)
            if (message != b'Hello\n'):
                sys.exit(-1)
            print(b"Received %s" % message)
            number = random.randint(1000,9999)
            print("Sending %d" % number)
            conn.send(bytes(str(number)+'\n', 'utf-8'))
            message = conn.recv(5)
            print(b"Received %s" % message)
            if number+1 != int(str(message, 'utf-8')):
                conn.send(b"Incorrect\n")
                print("Incorrect")
            else:
                conn.send(b"  Correct\n")
                print("Correct")
            conn.close()
            sys.exit(0)


if __name__ == '__main__':
    main()
```

Diff: Non-TLS vs TLS Client

```python
#!/usr/bin/env python3

import socket

def main():
    client_socket = socket.socket()

    client_socket.connect(('127.0.0.1', 31337))

    print("Sending Hello...")
    client_socket.send(b'Hello\n')

    b_number = client_socket.recv(5)
    s_number = str(b_number, 'utf-8')
    print("Received %s" % s_number.strip())

    i_number = int(s_number)
    i_answer = i_number + 1

    print("Returning %d" % i_answer)
    b_answer = bytes(str(i_answer), 'utf-8')
    client_socket.send(b_answer)

    result = client_socket.recv(10)
    print(result)

    client_socket.close()

if __name__ == '__main__':
    main()
```

```python
#!/usr/bin/env python3

import socket
import ssl

def main():
    client_socket = socket.socket()

    context = ssl.create_default_context()
    context.check_hostname = False          # bad example
    context.verify_mode = ssl.CERT_NONE     # bad example

    client_socket.connect(('127.0.0.1', 31337))
    ssl_client_socket = context.wrap_socket(client_socket)

    print("Sending Hello...")
    ssl_client_socket.send(b'Hello\n')

    b_number = ssl_client_socket.recv(5)
    s_number = str(b_number, 'utf-8')
    print("Received %s" % s_number.strip())

    i_number = int(s_number)
    i_answer = i_number + 1

    print("Returning %d" % i_answer)
    b_answer = bytes(str(i_answer), 'utf-8')
    ssl_client_socket.send(b_answer)

    result = ssl_client_socket.recv(10)
    print(result)

    ssl_client_socket.close()

if __name__ == '__main__':
    main()
```

Diff: Non-TLS vs TLS Server

```python
#!/usr/bin/env python3
import os
import random
import socket
import sys

def main():
    server_socket = socket.socket()
    server_socket.bind(('127.0.0.1', 31337))
    server_socket.listen(10)

    while True:
        conn, addr = server_socket.accept()
        if os.fork():
            conn.close()
        else:
            message = conn.recv(6)
            if (message != b'Hello\n'):
                sys.exit(-1)
            print(b"Received %s" % message)
            number = random.randint(1000,9999)
            print("Sending %d" % number)
            conn.send(bytes(str(number)+'\n', 'utf-8'))
            message = conn.recv(5)
            print(b"Received %s" % message)
            if number+1 != int(str(message, 'utf-8')):
                conn.send(b"Incorrect\n")
                print("Incorrect")
            else:
                conn.send(b"  Correct\n")
                print("Correct")
            conn.close()
            sys.exit(0)


if __name__ == '__main__':
    main()
```

```python
#!/usr/bin/env python3

import os
import random
import socket
import ssl
import sys

def main():
    server_socket = socket.socket()
    server_socket.bind(('127.0.0.1', 31337))
    server_socket.listen(10)

    context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    context.load_cert_chain('cert.pem', 'key.pem')
    ssl_server_socket = context.wrap_socket(server_socket, server_side=True)

    while True:
        conn, addr = ssl_server_socket.accept()
        if os.fork():
            conn.close()
        else:
            message = conn.recv(6)
            if (message != b'Hello\n'):
                sys.exit(-1)
            print(b"Received %s" % message)
            number = random.randint(1000,9999)
            print("Sending %d" % number)
            conn.send(bytes(str(number)+'\n', 'utf-8'))
            message = conn.recv(5)
            print(b"Received %s" % message)
            if number+1 != int(str(message, 'utf-8')):
                conn.send(b"Incorrect\n")
                print("Incorrect")
            else:
                conn.send(b"  Correct\n")
                print("Correct")
            conn.close()
            sys.exit(0)


if __name__ == '__main__':
    main()
```

Summary

- The Internet is insecure by default
- TLS provides a transparent way to establish a secure communication channel and use it
- TLS handshake uses digital signature (RSA), key-exchange algorithm (ECDHE), symmetric cipher (AES-128-GCM) and HMAC (SHA256)
  - Provides authenticity, confidentiality, and integrity to the channel
- Applications can enjoy the added security by just wrapping sockets
  - Only for TCP applications
  - DTLS is for UDP communications