HTTP over SSL/TLS

HTTPS

- Commonly called HTTP Secure or Secure HTTP
- From a web app development perspective
  - HTTPS is the same protocol as HTTP
  - We reuse **all** of our HTTP code

- The difference is that all out requests/responses are encrypted via SSL/TLS
  - SSL (Secure Socket Layer) was renamed to TLS (Transport Layer Security) after SSL 3.0
  - I'll only refer to the protocol as TLS after this note

TLS

- TLS fits between TCP and HTTP on our protocol stack
- All these protocols are modular
  - TCP is not aware that the bytes it's sending are encrypted
  - HTTP is not aware that its requests were encrypted or that its responses will be encrypted

Network/Protocol Stack
HTTP
TCP
IP

Network/Protocol Stack
HTTP
TLS
TCP
IP

TLS

- This allows us to continue to use TCP and HTTP
- We only need to add the TLS layer to our web apps to gain encryption
  - This will not require any changes in the HTTP side of our servers

Network/Protocol Stack
HTTP
TLS
TCP
IP

TLS

- What we want:
  - Two-way encrypted traffic
- What we have:
  - A server with a public/private key pair verified by a CA

- A client could encrypt using the servers public key
- How does the server encrypt responses sent to the client?

TLS Overview

- Client and server negotiate a TLS handshake
- During the handshake, a symmetric encryption key is agreed upon
  - Same key encrypts and decrypts
- Client and server both have this key
- All communication in both directions is encrypted with this key

- With this goal in mind, how do a client and server securely agree on this key without an eavesdropper also knowing the key?

Diffie-Hellman Key Exchange

- Client and server agree on a prime number p with a group generator g
  - A generator for a prime group means that
    - For each value 0 < i < p
    - g^i mod p is a unique value
    - We say g generates the group since multiplying g by itself p times (mop p) will provide every value 1 to p-1

- Both p and g are public

Diffie-Hellman Key Exchange

- Client and server both compute a random number
  - Call the clients number a
  - Call the servers number b

- Both a and b are private
  - Client and server cannot even share these values with each other

Diffie-Hellman Key Exchange

- Client computes g^a mod p
  - Sends this value to the server
  - Server raises this value to the power of b mod p
  - Server now has g^{ab} mod p

- Server computes g^b mod p
  - Sends this value to the client
  - Client raises this value to the power of a mod p
  - Client now has g^{ab} mod p

Diffie-Hellman Key Exchange

- Client and server now have a shared secret $g^{ab} \bmod p$
  - This secret is used as a seed to generate a symmetric encryption key
    - Or used directly as the key

- The only values containing secret values that were sent over the network were
  - $g^a \bmod p$
  - $g^b \bmod p$
- And computing a or b from these values involves computing a discrete logarithm which we believe is prohibitively hard to solve

Symmetric Key Encryption

- Once the Client and server have a shared symmetric, then can encrypt all their communication with this key
- The same key encrypts and decrypts
- Typical choice of algorithm is AES (Advanced Encryption Standard)
  - Very brief description: AES repeatedly scrambles bytes and XOR them with values generated by the encryption key
  - AES does not reduce to a cryptographic primitive
  - Theoretical attacks exist, but no known practical attacks yet

TLS 1.2 Handshake

- Client Hello
  - Here are the algorithms I support
- Server Hello
  - Here are the algorithms we'll use for this connection
- Server sends its certificate
- Client and server both generate their part of the symmetric key based on the chosen algorithms
  - Ex. Generate a and send g^a mod p
  - Server signs its portion with the private key from its certificate
- With the partial key received from the client/server, compute the rest of the symmetric key
- Both parties now have the symmetric key and can encrypt all following traffic

Algorithms Note

- RSA, Diffie-Hellman Key Exchange, and AES were mentioned as examples
- The algorithms change and evolve over time
- Different servers/client may support different sets of algorithms
- TLS is very flexible and allows for any algorithms to be used, so long as the client and server both agree which ones will be used
  - TLS itself does not define how to exchange keys or encrypt and instead defers to the algorithms for details

Forward Secrecy

- Note that the servers keys from its certificate were only used to verify the servers identity during the key exchange
- The encryption of traffic was done with a one-time symmetric key
  - A different key is generated for every TLS connection

- Even if an eavesdropper stored all of the encrypted traffic **and** later stole the servers private key linked to the certificate
  - They are still out of luck (Cannot use this private key to find the symmetric key)
  - This is what we call forward secrecy

HTTPS

- HTTP over TLS (Transport Layer Security)
- TLS is the protocol that dictates the public/private key encryption, key exchange, and symmetric key encryption
- Encrypts the entire HTTP requests/responses using the symmetric key
- **Eavesdroppers can still see TCP/IP**
  - Including source/destination IP addresses!
    - They don't know what you're saying, but they know who you're talking to
  - This is why VPNs are still popular even though most sites use HTTPS in current year