SE 4472 / ECE 9064

Information Security

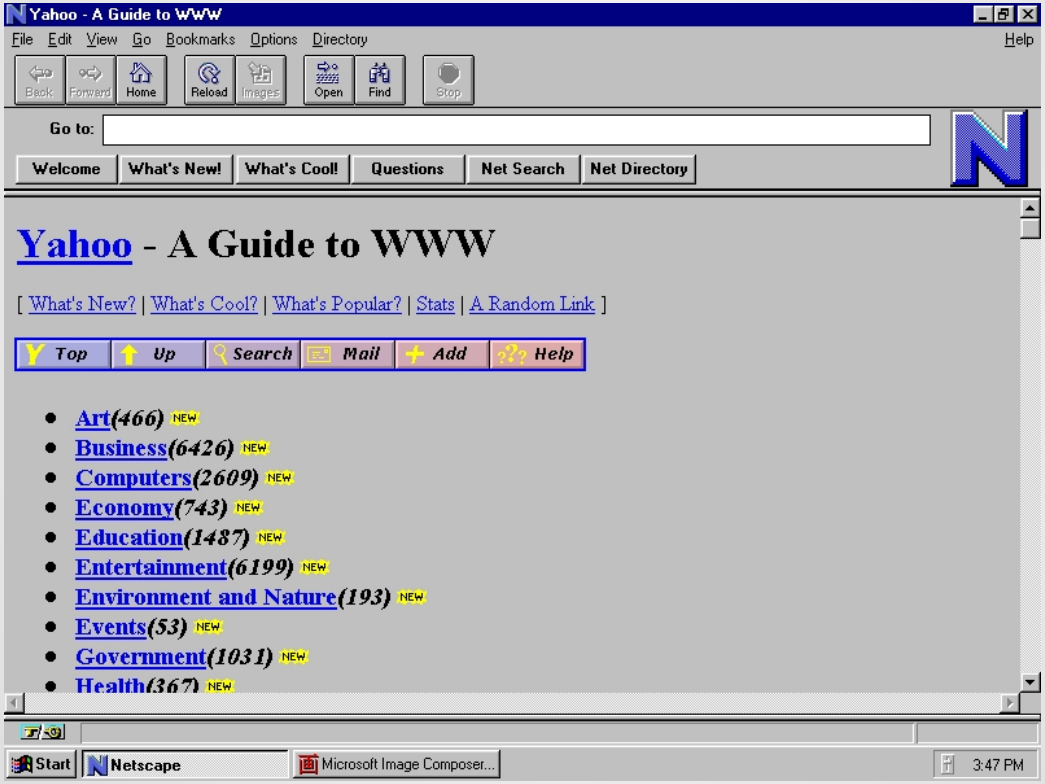
Week 11:

Transport Layer Security (TLS): Putting it all together

Fall 2015
Prof. Aleksander Essex

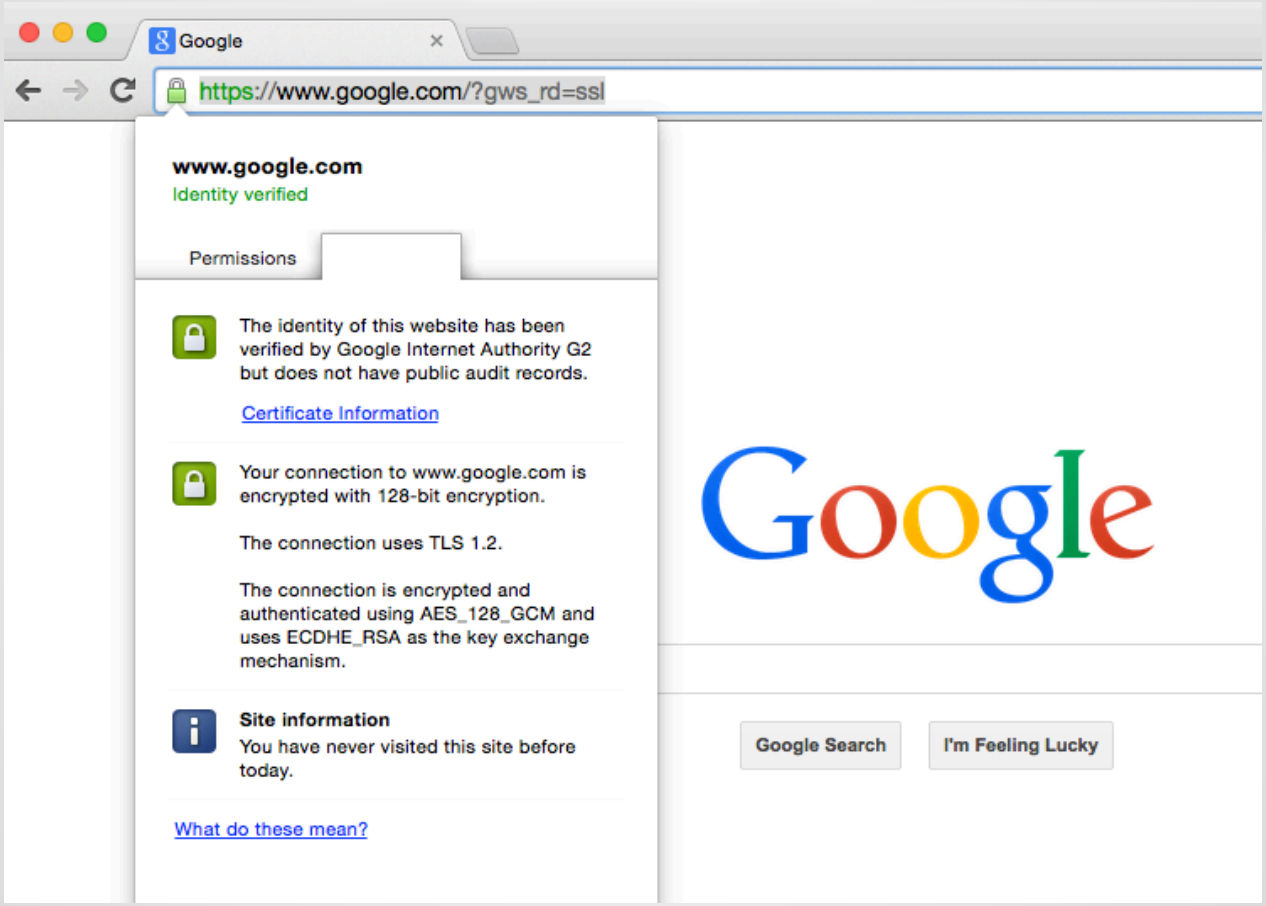
Western Engineering

Security at the Transport Layer



Where we started in this course: no security

Security at the Transport Layer



Where we've ended up in this course: practical security

Security at the Transport Layer

- TLS is the protocol that secures the web
- Combines all the concepts you've learned about so far

SSL/TLS History

- Secure Sockets Layer (SSL)
  - 1.0 – not released. Originally written by Taher Elgamal.
  - 2.0 – 1995. MITM possible through cipher downgrade attacks. Disallowed by IETF in 2011 (RFC 6176)
  - 3.0 – 1996. Major redesign. SHA-1 introduced. POODLE attack (Sept 2014)
- Transport Layer Security (TLS)
  - 1.0 – 1999. Different key derivation functions (HMAC)
  - 1.1 – 2006. Better IV handling mitigates CBC-mode attacks (BEAST)
  - 1.2 – 2008. SHA-256. AES-GCM
  - 1.3 – >2015. Deprecate RSA Kx?

TLS Handshake

TLS Handshake

- Phases:
- Phase 1: Establish security capabilities
- Phase 2: Authentication and public key exchange
- Phase 3: Secret key exchange and derivation
- Phase 4: Finish

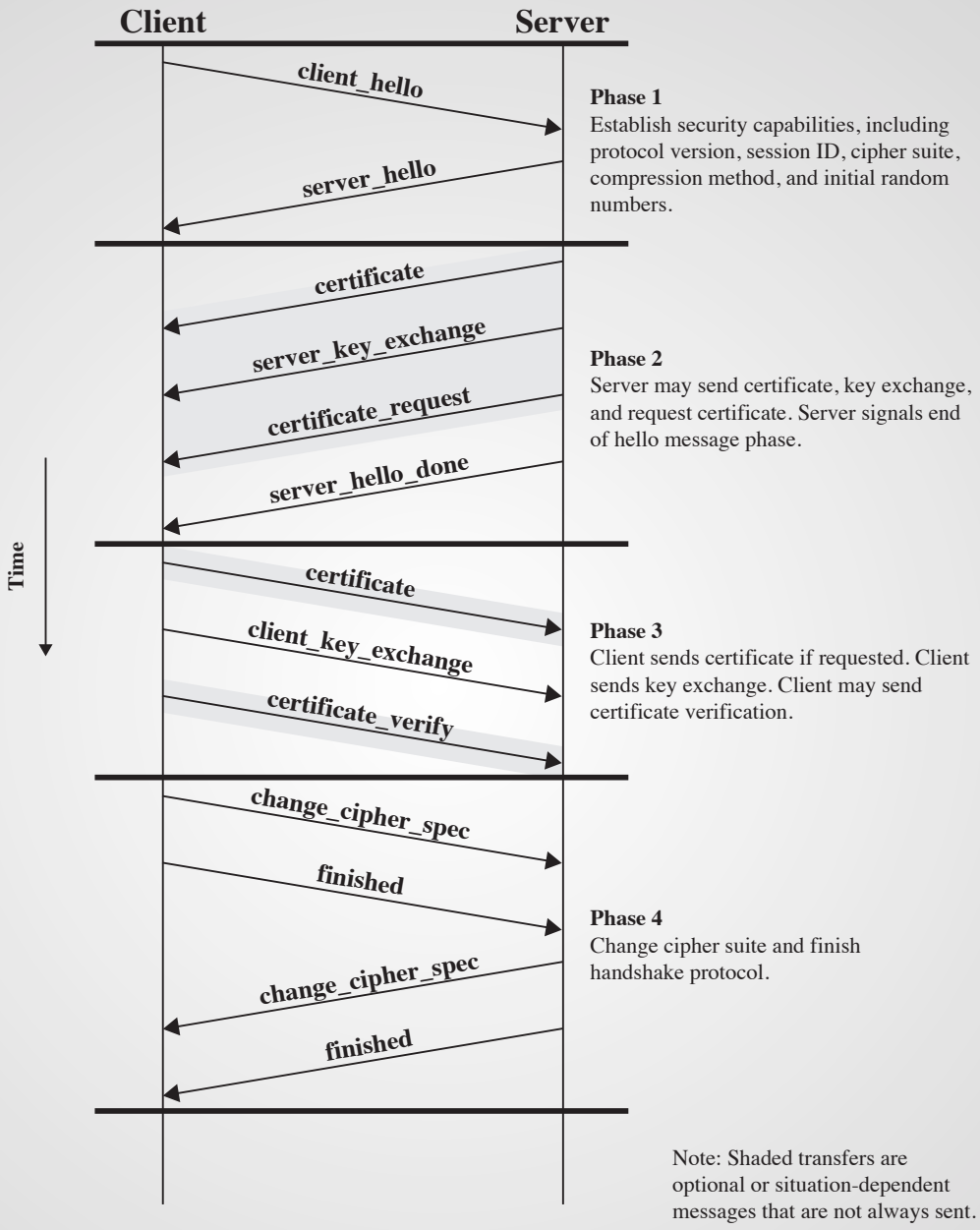


Figure 17.6 Handshake Protocol Action

Source: William Stallings. Cryptography and Network Security.

Phase 1: Security Capabilities

- Client_Hello
  - Highest SSL/TLS version supported
  - Client nonce
  - Session ID
- Server_Hello
  - Highest SSL/TLS version supported
  - Appropriate ciphersuite
    - Kx (e.g., RSA, DHE/ECDHE)
    - Cipher algorithm (e.g., AES)
    - Hash function (e.g., SHA1)
  - Server nonce

TLS Ciphersuites

Example: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

- Key agreement
  - RSA, DHE, ECDHE
- Signature scheme
  - RSA, DSA, ECDSA
- Block cipher and mode of operation
  - AES_256_CBC, 3DES_EDE, AES-GCM (i.e., AES in CTR)
- Hash function
  - MD5, SHA1, SHA256, SHA512

Chrome's Ciphersuite Preferences

Cipher Suites (in order of preference)	
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b) Forward Secrecy	128
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) Forward Secrecy	128
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x9e) Forward Secrecy	128
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc14) Forward Secrecy	256
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc13) Forward Secrecy	256
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc15) Forward Secrecy	256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a) Forward Secrecy	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) Forward Secrecy	256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x39) Forward Secrecy	256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009) Forward Secrecy	128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) Forward Secrecy	128
TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x33) Forward Secrecy	128
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)	128
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)	256
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)	128
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)	112

TLS Ciphersuites

- The hash function is used as a pseudo-random function to derive keys.
- Depending on the
- What about this:
  - TLS_RSA_WITH_AES_128_GCM_SHA256
  - What is the key agreement method? What is the signature scheme? What is the MAC?

Phase 2: Authentication and Public-key Exchange

- Certificate message
  - Server sends certificate chain
- Server_key_exchange
  - If using DHE/ECDHE, server sends its public key and signature

Client checks certificate chain, and signature on Pk if using DHE/ECDHE

Phase 3: Key Exchange/Derivation

1. Exchange *pre-master secret*

- If using RSA for Kx
  - Client generates 48-byte *pre-master secret*, encrypts with public key and sends to server
- If using DHE/ECDHE
  - Parties compute Diffie-Hellman shared secret (becomes *pre-master secret*)

2. Derive *master secret*

3. Derive symmetric keys
   - Use key derivation function to derive key material (see next slide)

Pseudo-random Function (PRF)

- TLS makes use of an HMAC as a PRF
- Used to expand secrets into keys
- Recall HMAC(K,m) accepts a message and a key
- First we define P_hash, which takes a secret (and a seed value) and expands it into a desired number of bytes:

```
P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) +
                       HMAC_hash(secret, A(2) + seed) +
                       HMAC_hash(secret, A(3) + seed) + ...
```

where + indicates concatenation.

A() is defined as:

```
A(0) = seed
A(i) = HMAC_hash(secret, A(i-1))
```

Pseudo-random Function (PRF)

- The PRF is constructed as follows:

  PRF(secret, label, seed) = P_<hash>(secret, label + seed)

- The label is a UTF-8 string. For example, the label "slithy toves" would be represent the following bytes:

  73 6C 69 74 68 79 20 74 6F 76 65 73

Master Secret

- Derived from the pre-master secret (either RSA or Diffie-Hellman shared secret):

```
master_secret = PRF(pre_master_secret, "master secret",
                    ClientHello.random + ServerHello.random)
                    [0..47];
```

- Used to generate the “key block”:

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

Key Block

- The key block consists of all the values used in the symmetric-key operations
- TLS generates separate keys for client and sever (though both ends have all keys):

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]

Phase 4: Finished

- Parties exchange Finished messages
- An HMAC'd copy of everything the client (resp. the server) has seen in the handshake so far

  PRF(master_secret, finished_label, Hash(handshake_messages))

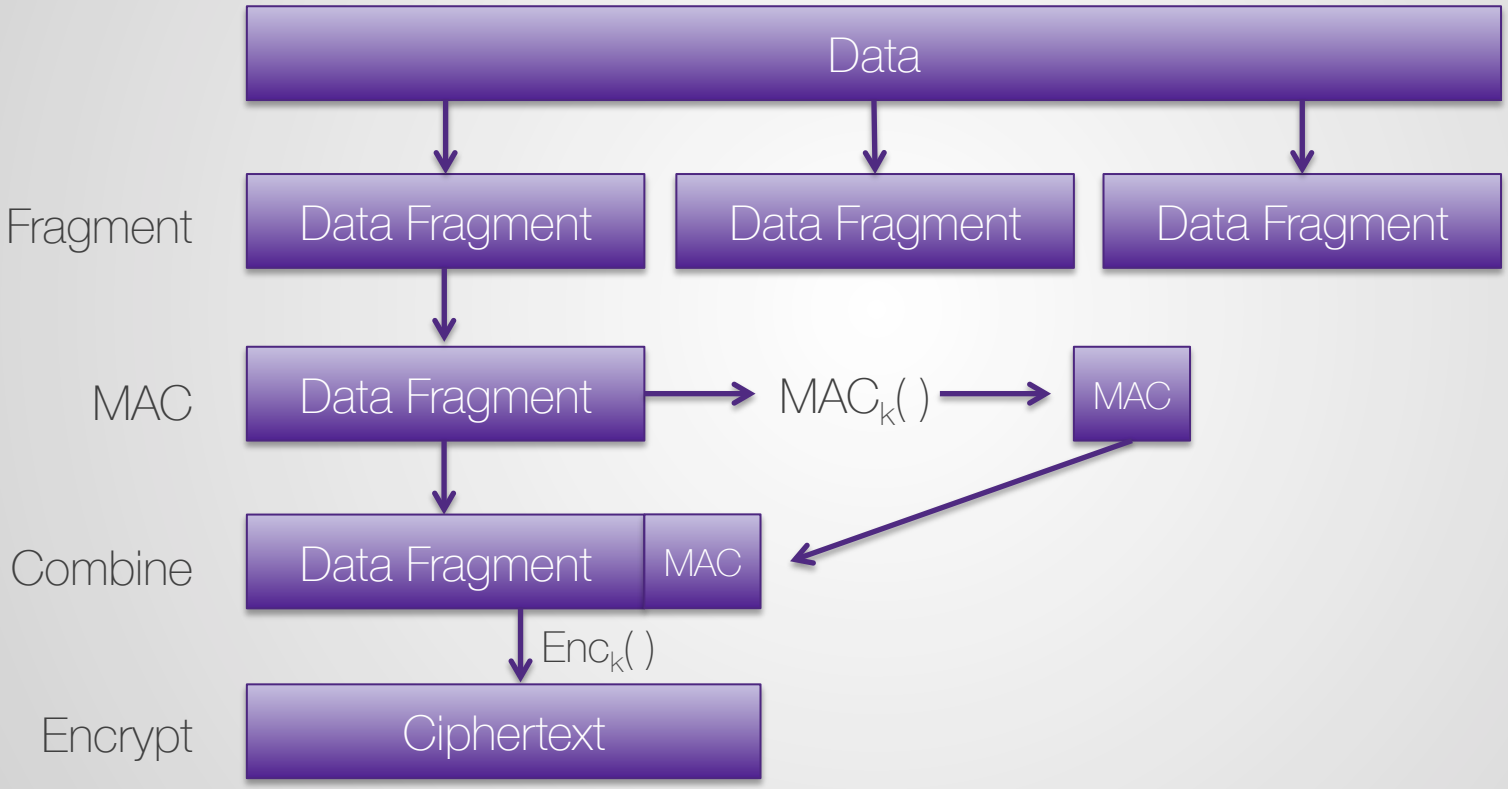
- Prevents a variety of subtle man-in-the-middle attacks
- Once client (resp. server) sends its Finished message and receives and validates the Finished message from its peer, it can start using the TLS connection.

Wireshark

Let’s see a TLS handshake in Wireshark

TLS Record Protocol

SSL/TLS Record Protocol



TLS Attacks

A Few SSL/TLS Attacks

Major SSL/TLS attack focus on trying to recover a encrypted single session cookie/token

- BEAST (TLS 1.0)
  - Predictable IV ("optimization": reuse last block of ciphertext as IV of current block)
- CRIME
  - Exploits TLS compression. Eve injects data into plaintext. If ciphertext is smaller, she knows injected data is similar to target data, and vice versa if ciphertext is larger.
- POODLE (SSL 3.0)
  - MAC was taken message, padding applied after (i.e., padding is unauthenticated). What could possibly go wrong?