ECS655U: Security Engineering

Week6: Example Application of Cryptography: SSL/TLS protocol

Dr Arman Khouzani

February 15, 2019

EECS, QMUL

Learning Outcomes (this lecture)

- A quick review of what we have learned so far, in preparation for the midterm exam next week.
- See how the components that we learned come together through an application case study: TLS.
- Practice the arguments for establishing required security properties.

Learning Outcomes (of the book chapter)

- Familiarity with a range of applications of cryptography, identifying what we have learned:
  - Internet security (SSL/TLS)
  - Wireless Local Area Networks (WEP, WPA)
  - Mobile Telecommunication (GSM, UTMS, 3G/4G)
  - Video Broadcasting (Subscription TV, pay-per-view)
  - Electronic IDs, Secure Payment Cards (chip-and-pin)
  - Anonymity and Anti-Censorship (Tor/I2P)
  - Cryptocurrency (blockchain, timestamping, mining)
- Compare the different environment and security requirement constraints of these applications.
- Appreciating the effect of these constraints on the choice of cryptography and key management.

Announcement

Midterm: next week!

Midterm (20%): Week 7 (Friday February 22), starting at **9:00 am**, ending at 11:00 am, format: Written (similar to the final), location: **ITL Ground Floor**, coverage: weeks 1, 2, 3, 4, 5, 6 (inclusive). **Closed resources**. Calculators NOT allowed.

A Quick Review

Review: Cryptographic Primitives Alone

Table 1: Mapping of primitives used **on their own** to provide security services (Table 1.1 in the book).

	C.	D.I.	D.O.A.	N.R.	E.A.
Encr.	Yes	No	No	No	No
Hash	No	Sometimes	No	No	No
MAC	No	Yes	Yes	Sometimes	No
Dig. Sig.	No	Yes	Yes	Yes	No

Review: Cryptographic Primitives Combined

Table 2: Mapping of primitives that **can be used** to help provide security services. (Table 1.2 in the book).

	C.	D.I.	D.O.A.	N.R.	E.A.
Encr.	Yes	Yes	Yes	Yes	Yes
Hash	Yes	Yes	Yes	Yes	Yes
MAC	No	Yes	Yes	Yes	Yes
Dig. Sig.	No	Yes	Yes	Yes	Yes

SSL/TLS

SSL/TLS

- *Transport Layer Security (TLS)* protocol is the main protocol behind Internet's security (no big deal!)
- Also used for security email, instant messaging, VoIP (again, no big deal!)
- HTTPS: is just HTTP over TLS!
- FTPS: is just FTP over TLS!
- SMTP uses TLS for secure email delivery!

SSL/TLS

Port	Description
21	FTP
889	FTP data over TLS/SSL (FTPS-DATA)
990	FTP control over TLS/SSL (FTPS)
23	telnet
992	telnet protocol over TLS/SSL (telnets)
25	SMTP
465/587	SMTP over TLS/SSL (SMTPS)
80	HTTP
443	HTTP over TLS/SSL (HTTPS)
389	LDAP
636	LDAP over TLS/SSL (SLDAP)

SSL/TLS: History

- First developed in the mid-1990s by Netscape Communications for use with their Navigator browser, and they called it *Secure Sockets Layer (SSL)*.

  - SSLv1 ..... broken at birth
  - SSLv2 ..... many flaws, deprecated by IETF
  - SSLv3 ..... now broken (POODLE + RC4 attacks)

SSL/TLS: History

- In 1999, the *Internet Engineering Task Force (IETF)*, standardised an improved version of (which was no longer tied to any application layer browser or OS) and adopted the name *TLS* (Transport Layer Security).

- TLS 1.0 1999
- TLS 1.1 2006
- TLS 1.2 2008
- TLS 1.3 (significantly different) . . . . . (working draft)

Originally designed for secure e-commerce, now the de facto secure protocol of choice, used everywhere.

SSL/TLS

TLS requires a reliable underlying transport protocol, e.g., *Transmission Control Protocol (TCP)* in Internet, but is under the application layer. (Note: TLS does not require TCP, any *reliable* transport protocol would do.)

- In contrast, SSH (Secure Shell) protocol is at the application layer, and IPsec protocol is at the Internet Procol (IP) layer.

The (green) padlock sign in your browser shows that you are (properly) using HTTP over TLS, i.e., HTTPS.

SSL/TLS

TLS provides a *secure* channel between two entities (e.g. a client web browser and a web server) over a public channel. In particular, it can provide *confidentiality*, *data integrity and origin authentication*, and *entity authentication*.

- TLS is designed to work even in cases where there is no *security association*, i.e., where entities have never before shared any secret information or even agreed on the choice of algorithms, etc.
- How? Using public-key cryptography, and (public key) certificates (and cipher-suite negotiation).

SSL/TLS

TLS uses a wide range of cryptographic primitives:

- Public-key cryptography is used to enable symmetric key establishment.
- Digital signatures are used to sign certificates and facilitate entity authentication.
- Symmetric encryption is used to provide confidentiality.
- MACs are used to provide data origin authentication and facilitate entity authentication.
- Hash functions are used as components of MACs and digital signatures, and for key derivation.

SSL/TLS: cipher-suite

The two entities agree on the **cipher-suite**, *example*:

`TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`

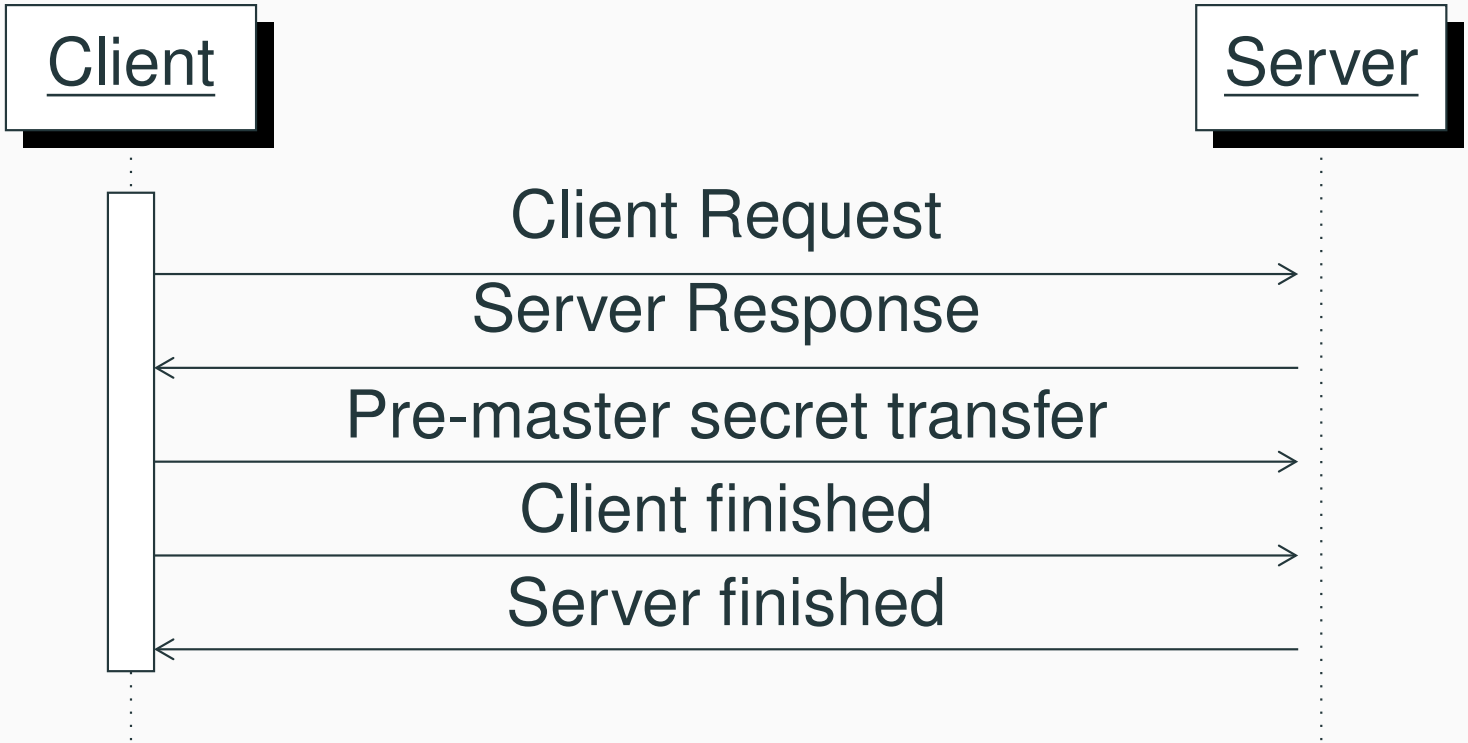
- `TLS` the protocol;
- `ECDHE` the key exchange algorithm;
- `ECDSA` the digital signature algorithm (for verifying the certificates);
- `AES_256_CBC` the bulk (symmetric key) encryption algorithm;
- `SHA384` indicates the hash in HMAC algorithm.

SSL/TLS: handshake and record

TLS consists of two parts:

- **Handshake Protocol:** performs all the tasks requiring agreement between the two entities before they set up the secure TLS channel. In particular:
  - agree on the cipher suite to be used;
  - establish entity authentication; and
  - establish the keys needed to secure the channel.
- **Record Protocol:** This protocol implements the secure channel, involving:
  - formatting the data (breaking it up into blocks, etc);
  - computing MACs on the data; and
  - encrypting the data.

SSL/TLS: Handshake Protocol Description



SSL/TLS: Handshake Protocol Description

Client Request (`ClientHello`): This message from the client initiates the communication session. It includes:

- a session ID, (a unique identifier for the session – if the client wants to reuse some parameters from an older session);
- a pseudorandom number r_C, used for freshness;
- a list of cipher suites that the client supports. e.g.:
  - `TLS_RSA_WITH_AES_256_CBC_SHA256`,
  - `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`,
  - ...

  The list is in order of preference of the client.

SSL/TLS: Handshake Protocol Description

Server Response: with some initialisation data:

- ▷ the session ID;
- ▷ a pseudorandom number r_S, which is the server's freshness contribution to the protocol;
- ▷ the particular cipher suite chosen from the list offered by the client, which includes a specification of the method to establish shared keys;
- ▷ a copy of the server's public-key certificate, including details of any certificate chain required to verify it;
- ▷ if the server has chosen Ephemeral DH to establish shared keys, then it also generates a fresh set of parameters for it and sends the public values alongside a digital signature on them.

SSL/TLS: Handshake Protocol Description

- ▷ The client now checks that the server's public-key certificate is valid. This will involve checking the validity of all the public-key certificates in the chain, as well as checking any relevant CRLs.
- ▷ If the Ephemeral DH protocol is being used, then the client also verifies the digital signature on the DH parameters.

Question: after this stage, has the client authenticated the server?

SSL/TLS: Handshake Protocol Description

Pre-master Secret Transfer: The client and server now need to agree on a shared *pre-master* secret K_P, which will be used to derive the keys used to secure the session. The most popular techniques are:

1. if RSA has been selected for key establishment, then the client pseudorandomly generates K_P, encrypts it using the server's public key and sends to the server;
2. if Ephemeral DH has been selected for key establishment, then the client generates a fresh temporary DH key pair and sends the public value to the server, after which both client and server compute the shared DH secret K_P.

SSL/TLS: Handshake Protocol Description

The client and server are both now in a position to derive the keys required to secure the TLS session:

- First, they use a *key derivation function* that takes K_P as key and also r_C and r_S as inputs, to compute the master secret K_M.
  Question: How do they know which key derivation function to use?
- The client and server both then use K_M to derive keys to be used for MAC and encryption.
- From this point on, all exchanged messages are cryptographically protected.

SSL/TLS: Handshake Protocol Description

Client Finished: The client computes a MAC (typically based on HMAC) on the hash of all the messages sent thus far. Then encrypts it and sends it to the server.

Server Finished: The server checks the MAC received from the client. The server then computes a MAC on the hash of all the messages sent thus far. This MAC is then encrypted and sent to the client.

- ▷ Finally, the client checks the MAC received from the server.

TLS: Handshake Protocol Analysis

- **Agreement of cryptographic algorithms.** This is achieved at the end of the second protocol message (when the server informs the client which cipher suite has been selected from the list provided by the client).

TLS: Handshake Protocol Analysis

- **Entity authentication of the server.** assuming the protocol run has been successful, we argue:

1. The entity who sent the *Server Finished* message must know the master secret K_M, since the final check was correct and relied on knowledge of K_M.
2. Any entity who knows K_M must also know the pre-master secret K_P, since K_M is derived from K_P.

TLS: Handshake Protocol Analysis

3. Any entity other than the client who knows K_P must know the private key corresponding to the public-key certificate sent in the message *Server Response*, since:
   - ▷ if RSA has been selected for key establishment, this private key is necessary to decrypt K_P in the message Pre-master Secret Transfer;
   - ▷ if Ephemeral DH has been selected for key establishment, then this private key was used to sign a verified digital signature on public DH parameters needed to compute K_P.

TLS: Handshake Protocol Analysis

4. The only entity with the ability to use the private decryption key is the genuine server, since the public-key certificate provided by it in the message *Server Response* was checked and found to be valid.
5. The server is currently “alive” because K_M is derived from fresh pseudorandom values (K_P and r_C) generated by the client and thus cannot be an old value.

TLS: Handshake Protocol Analysis

- **Key establishment.** TLS establishes several keys, all of which are derived from the master secret K_M, which is a value established during the Handshake Protocol. The master secret is derived from the pre-master secret K_P, which is a value only the client and the server know.

Note that the Client Finished and Server Finished messages also provide assurance that none of the messages exchanged during the Handshake Protocol have been tampered with, which is particularly important since the opening messages of the protocol have no cryptographic protection.

TLS: Handshake Protocol Analysis

Question: Which one of the goals of AKE do we have?

- ▷ *Mutual entity authentication*
- ▷ *Mutual data origin authentication*
- ▷ *Mutual key establishment*
- ▷ *Key confidentiality*
- ▷ *Key freshness*
- ▷ *Mutual key confirmation*
- ▷ *Unbiased key control*

TLS: Handshake Protocol w/ Client Authentication

- The simple Handshake Protocol does not provide mutual entity authentication, only entity authentication of the server.
- For some applications, this may be fine, e.g. client authentication is done at the application layer (using a password, token, etc.), or the server simply may not care! (e.g., when a user purchases goods from an online store, merchant may only care about getting paid at the end!)

TLS: Handshake Protocol w/ Client Authentication

Mutual authentication is achievable by adding an extra message from the client to the server after the message *Pre-master Secret Transfer*, as follows:

- **Client Authentication Data:** The client sends a copy of its public-key certificate to the server.
  - The public key in this certificate is used as a verification key.
  - The client hashes all the protocol messages so far and *digitally signs* the hash using the client's signature key.

TLS: Handshake Protocol wi/ Client Authentication

Mutual authentication is achievable by adding an extra message from the client to the server after the message *Pre-master Secret Transfer*, as follows:

- **Client Authentication Data:** The client sends a copy of its public-key certificate to the server.
  - The public key in this certificate is used as a verification key.
  - The client hashes all the protocol messages so far and *digitally signs* the hash using the client's signature key.

TLS: Handshake Protocol w/ Client Authentication

- ▷ The server then checks that the client's public-key certificate (chain) is valid.
- ▷ The server also verifies the client's digital signature at the end.

TLS: Analysis of Handshake w/ Client Auth.

Entity authentication assurance of the client follows:

1. The entity who sent the *Client Authentication Data* message must know the signature key corresponding to the public key in the client's certificate (why?)
2. The only entity who knows the signature key is the genuine client (why?)
3. The client is currently "alive" (why?)

SSL/TLS: Record Protocol

The TLS's **Record Protocol** is used to instantiate the secure channel after the **Handshake Protocol** has successfully completed.

- Before running the Record Protocol, both client and server derive the necessary cryptographic data:
  - symmetric session keys for encryption;
  - symmetric MAC keys for data integrity/origin authentication (and by extension, entity authentication);
  - and any required IVs.

SSL/TLS: Record Protocol

- These are all generated using a key derivation function to compute a “key block”.
- This key derivation function uses K_M as a key and takes as input, among other data, r_C and r_S.
- The key block is then “chopped up” to provide the necessary cryptographic data.

SSL/TLS: Record Protocol

In particular, the following four symmetric keys are extracted from the key block:

- K_{ECS} for symm. encryption from client to server;
- K_{ESC} for symm. encryption from server to client;
- K_{MCS} for MACs from the client to the server; and
- K_{MSC} for MACs from the server to the client.

Note: these keys are never explicitly exchanged! But how did we get them to be the same on both parties?

SSL/TLS: Record Protocol

For data sent from the client to the server, e.g., the process is:

1. Compute a MAC on the data (and various other inputs) using key K_{MCS};
2. append the MAC to the data and then pad, if necessary, to a multiple of the block length; and
3. encrypt the resulting message using key K_{ECS}.
4. Upon receipt of the protected message, the server decrypts it using K_{ECS} and then verifies the recovered MAC using K_{MCS}.

Recall: so TLS is using a MAC-then-Encrypt scheme.
Question: what were the alternatives? their pros/cons?

SSL/TLS: TLS 1.3

Motivations for TLS 1.3:

- **Weaknesses.** the discovery of a number of attacks on TLS, e.g. *Renegotiation attack (2009)*, Downgrade attacks: *FREAK* and *Logjam*, Cross-protocol attacks:*DROWN*, *BEAST* attack, *CRIME* and *BREACH* attacks, *POODLE*, *Sweet32*, . . .
- **Efficiency.** The Handshake Protocol is somewhat inefficient, requiring two round trips between the client and server.

SSL/TLS: TLS 1.3

Main differences between TLS 1.3 and earlier versions:

- **Perfect Forward Secrecy.** (what is it?) This is achieved by removing support for key establishment based on RSA and mandating the use of DHE.
- **New Handshake Protocol.** The handshake is made more efficient by only requiring one full round trip, and more of it is encrypted. Further, the messages in the new Handshake Protocol are protected by dedicated keys (the two Finished messages of were protected using the same keys as used in the Record Protocol).
- **Authenticated encryption modes.** Encryption "must" use an authenticated-encryption mode.