727 Crypto Management

Lecture 14

GIOVANNI RUSSELLO

G.RUSSELLO@AUCKLAND.AC.NZ

Pretty Good Privacy (PGP)

Create by Phil Zimmermann, it is used for providing confidentiality and authentication to email and file storage services

PGP consists of

- The best crypto blocks
- Integration of these blocks into a general purpose application that is system agnostic
- Easy to use commands

The code and documentation is freely available but there is also a (low-cost) commercial version

Use of well-known and secure crypto block

It is not under the control of any government

PGP Operation Description

PGP provides different operation modes for both emails and file storage

- Authentication
- Confidentiality
- Confidentiality and Authentication
- Compression

PGP Authentication Mode

1. The sender creates a message
2. A hash function is used to generate a 160-bit hash code of the message
3. The hash code is encrypted using RSA with the sender's private key
4. The receiver decrypts the hash code using the sender's public key
5. The receiver computes the hash code of the message and compares it with the decrypted hash code
6. If the two matches then the message is authentic in the sense that has been generated by the sender

Authentication Mode Considerations

RSA guarantees that only the sender's private key could have signed the message

The hash function guarantees that no one else could generate a message that would match the decrypted hash code

This signature scheme usually provides the signature as an attachment to the email

PGP Confidentiality Mode

Using symmetric encryption, this mode allows protecting the content of emails or files

A symmetric key is used only once and every new transaction requires a new key to be generated

- Because it is used only once (and not for a session) the key is transmitted with the message
- To protect the key, the sender encrypts it with the public key of the receiver – this saves from the hassle of having to distribute the key

The use of symmetric key is justified by its performance: using public-key cryptography for encrypting/decrypting the message or file would be too slow

Because the symmetric key is used only once for each message/file, there is very little knowledge that an attacker could gain by monitoring the message traffic

PGP Confidentiality Mode – STEPS

The sender generates a message and a symmetric key

The message is encrypted with the symmetric key

The symmetric key is encrypted with the receiver's public key

Upon reception, the receiver uses its private key to retrieve the symmetric key and then decrypts the message

NOTE: often in the PGP literature/documentation the symmetric key is referred to as "Session Key" – However there is not such a notion of "session" in PGP

PGP Confidentiality and Authentication Mode

PGP supports the combinations of the two previous mode

First the signature from the plaintext message is generated using the sender private key

A symmetric key is generated

The message and the signature are encrypted using the symmetric key

The symmetric key is encrypted using the receiver's public key

Q: Why is the signature created from the plaintext instead than the encrypted message?

PGP Compression

As default, PGP compresses the message after applying the signature but before the encryption

Q: There are two main reasons why compression is applied after the signature but before encryption: can you explain these reasons?

Secure Sockets Layer (SSL)

SSL and its successor (TLS) were introduced to provide secure communications over the Internet

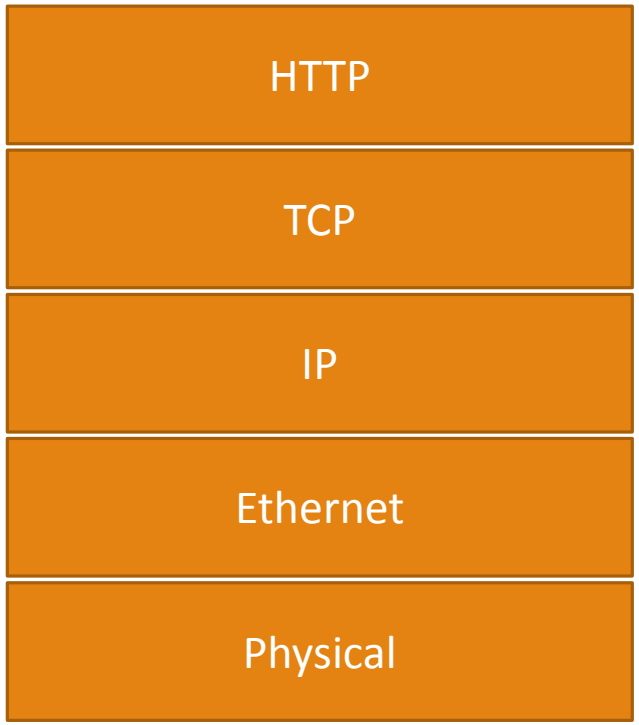
The goal of SSL is to make sure that

- The client is connecting to the “right” website – not a rouge one
- The client and server can securely exchange data without that an eavesdropper is able to “understand” the data

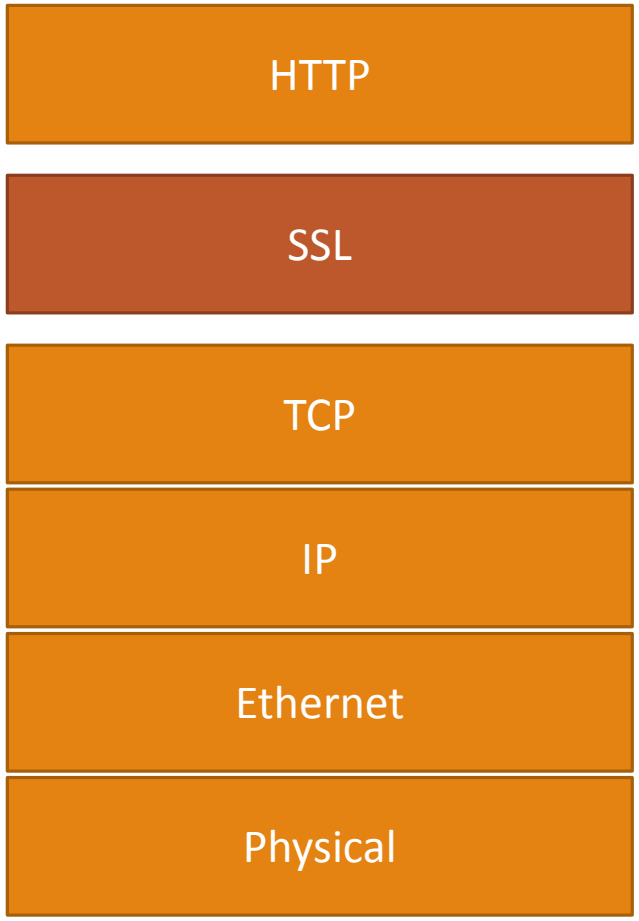
SSL was introduced after the TCP/IP layer was developed and deployed

- It has to work with the de-facto protocols and technology

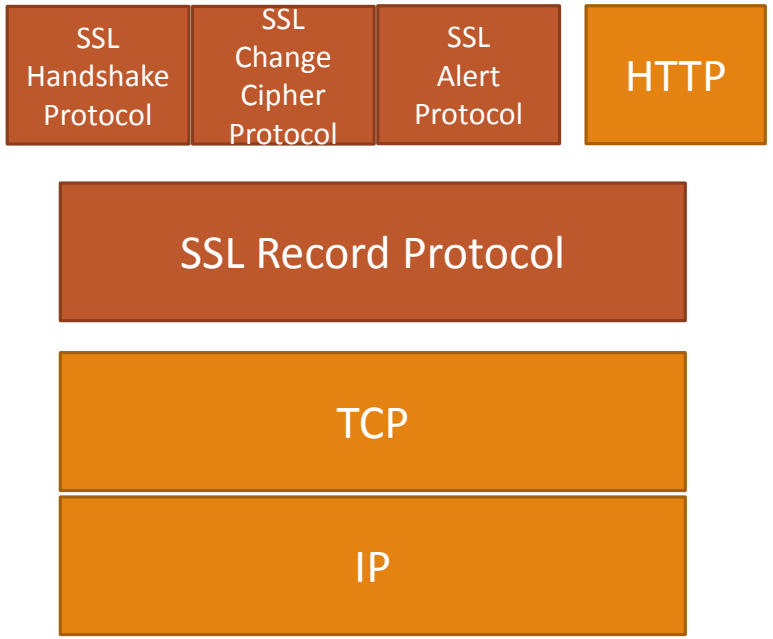
The TCP/IP Stack



The SSL Protocol Stack



The SSL Protocol Stack - Details



SSL Connections and Sessions

A **connection** is a transient transport abstraction to exchange data

- In SSL, a connection is point-to-point relationship between two endpoints
- A connection is always associated with a session

A **session** is an association between a client and a server

- A session is created after a Handshake Protocol to define a set of crypto parameters
- The session crypto parameter can be shared among the connections within a session

In theory, between a client and server there might be multiple sessions. But it is not usually the case

- The use of multiple connections within a session is to save time for crypto operations

Connections and Sessions have states that change during the operational life cycle

Session Parameters

A session state is defined by the following parameters

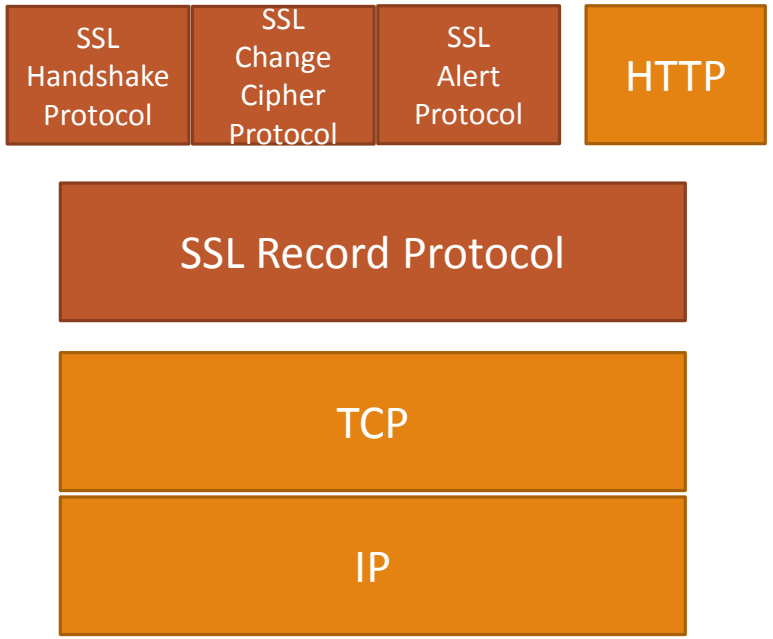
- Session identifier: an arbitrary byte sequence chosen by the server to identify an active or resumable session state
- Peer certificate: an X.509 certificate of the peer
- Compression method: the algorithm used for compressing the data prior to encryption
- Cipher spec: the symmetric algorithm used for the bulk encryption of the data and the hash algorithm used for MAC
- Master secret: the 48-byte secret key shared between the client and server
- Is resumable: a flag that indicates whether the session can be used to start a new connection

Connection Parameters

A connection state is defined by

- Server and client random: byte sequence chosen by the server and client for each connection
- Server write MAC secret: the secret key used in MAC operations on data sent by the server
- Client write MAC secret: the secret key used in MAC operations on data sent by the client
- Server write key: the secret key used by server for encrypting the data sent to the client
- Client write key: the secret key used by the client for encrypting the data sent to the server
- Initialization vector: when a block cipher is used in CBC mode then an IV is maintained for each key. This parameter is first initialized by the handshake protocol
- Sequence number: each party maintains a sequence number on the message transmitted and received. When a change cipher message is received this parameter is reset to zero and cannot exceed 2^64 -1

The SSL Protocol Stack - Details



SSL Record Protocol

This protocol provides two services:

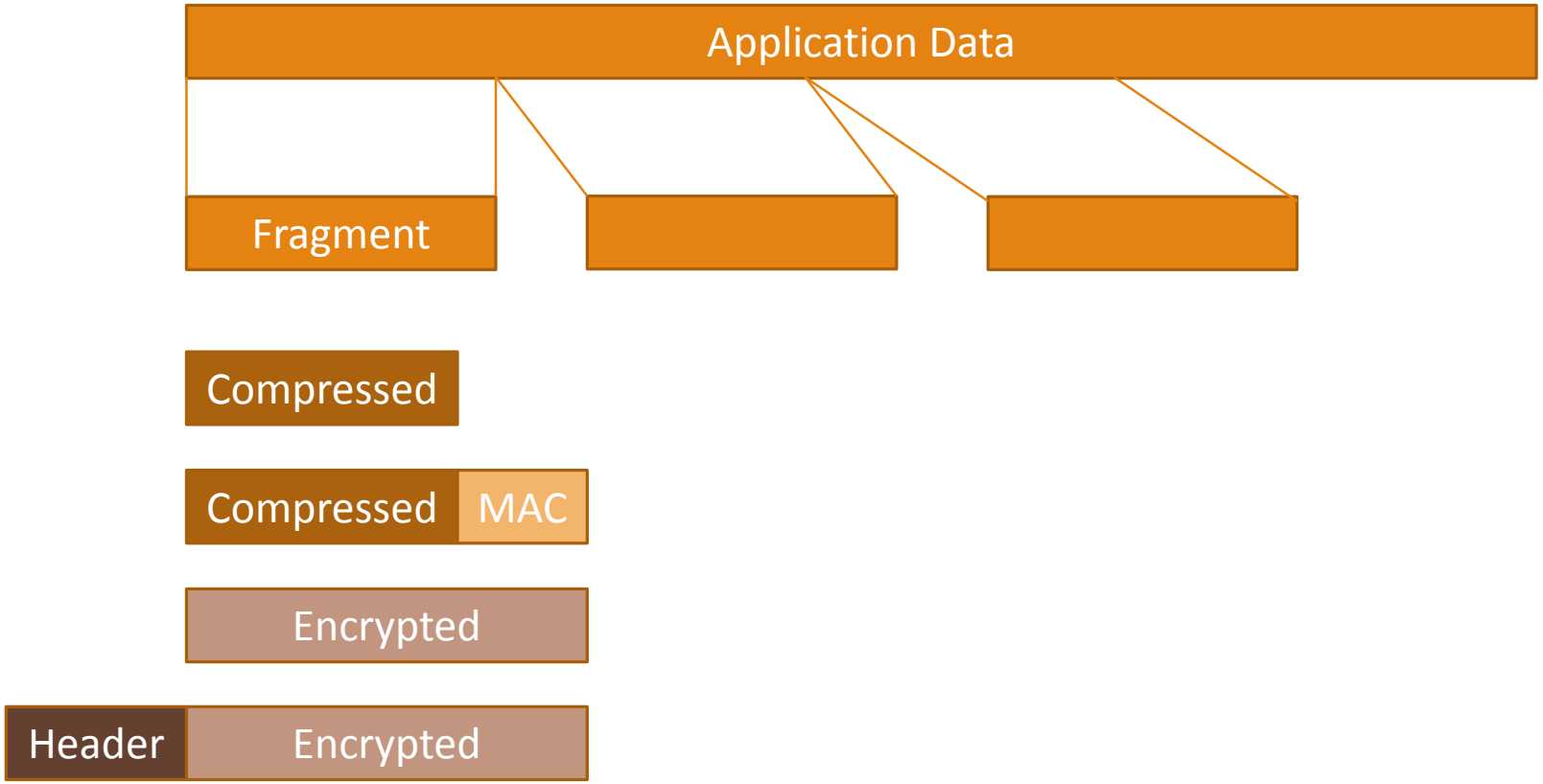
Confidentiality: the Handshake Protocol defines a shared secret key for encrypting the payload exchanged during a SSL session

Message Integrity: the Handshake Protocol defines a shared secret key used in a MAC

The SSL Record Protocol performs the following operations

- Fragments the application data in block (max block size is 2^14 or 16384 bytes)
- Compress the fragments – optional – must be lossless
- Add a MAC of the fragment
- Encrypt the MAC + fragment
- Append a SSL Record Header

Record Protocol Operations



SSL Record Header

Each fragment will contain a header with the following fields:

- **Content Type (8 bits)**: the higher level protocol used to process the enclosed data (`change_cipher_spec, alert, handshake, application_data`)
- **Major Version (8 bits)**: indicates the major version of SSL in use
- **Minor Version (8 bits)**: indicates the minor version of SSL in use
- **Compressed Length (16 bits)**: the length in bytes of the plaintext fragment (or compressed).

SSL High-level Protocols

SSL has three high-level protocols

- Change Cipher Spec Protocol
- Alert Protocol
- Handshake Protocol

The Change Cipher Spec Protocol is the simplest one

- Consists of only one message of one byte with value 1
- When this message is sent, the pending state becomes the current state

Alert Protocol

This protocol is used to alert the other peer that something is wrong

Its messages consists of two bytes: Level and Alert

- Level has two values: warning (1) or fatal(2)

When a fatal message is received the connection is immediately terminated

- Other connections might still be used
- No more connections can be established on the current session

Fatal Alert Messages

The following are always fatal

- **`unexpected_message`**: inappropriate message
- **`bad_record_mac`**: the MAC of a record was not correct
- **`decompression_failure`**: decompression failed (not able to or decompress to greater than allowable length)
- **`handshake_failure`**: the sender was not able to negotiate a valid set of parameters
- **`illegal_parameter`**: a field in the handshake protocol was out of range or inconsistent with the rest of the parameters

Other Alert Messages

These are the rest of the alert messages

- **`close_notify`**: the sender will not send any more messages on this connection. Each party is required to send this alert message when closing the write part of a connection
- **`no_certificate`**: an appropriate certificate is not found
- **`bad_certificate`**: a received certificate was corrupted (e.g., the signature did not verify)
- **`unsupported_certificate`**: the type of received certificate is not supported
- **`certificate_revoked`**: the signer has revoked the received certificate
- **`certificate_expired`**: the received certificate has expired
- **`certificate_unknown`**: something went wrong when processing a certificate

SSL Handshake Protocol

The Handshake protocol is the most complex part in SSL

The main objective is to establish all the crypto parameters to allow a secure connection between the client and the server

The Handshake protocol is execute before any application data is exchanged

All the messages in this protocol have the same format

- Type (1 byte): the type of message (there are 10 types supported)
- Length (3 bytes): the length of the message in bytes
- Content (>= 0 bytes): the parameter(s) associated with the message type

Handshake Phases

There are 4 phases within the Handshake protocol

- Phase 1: establishing security capabilities of the part involved and set preliminaries parameters (random number)
- Phase 2: server sends keying material through the use of its certificate and might ask for client cert
- Phase 3: client might send certificate (if requested) and certificate verification; also exchange key
- Phase 4: Change cipher suite and finish

If Phase 4 concludes successfully then a secure SSL session is established and the application data can be sent (e.g. access to your online back account)

Phase 1 – Client Hello

This phase is initiated by a client that wants to establish a logical connection with a server

- Here the client and server exchange parameters for the security capabilities

The client sends a **`client_hello`** message with

- **Version**: the highest version of the protocol understood by the client
- **Random**: 32-bit timestamp and 28 bytes random value; these are used as nonces and are used to prevent reply attacks
- **Session ID**: a session identifier; a zero value indicates the client wants to start a new connection on an new session; a nonzero value indicates either to update the parameters of an existing connection or establish a new connection on an existing session
- **Cipher Suite**: a list in decreasing order of preference of the crypto algorithms supported by the client
- **Compression Method**: list of compression methods supported by the client

Client Hello Message Cipher Suite List

Each element of the list contains several field

- Key exchange method: RSA, Fixed Diffie-Hellman, Ephemeral Diffie-Hellman, Anonymous Diffie-Hellman, Fortezza
- Cipher Spec:
  - CipherAlgorithm: RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza
  - MACAlgorithm: MD5 or SHA-1
  - CipherType: Stream or Block
  - IsExportable: True or False
  - HashSize: 0, 16 (MD5) or 20 (SHA-1) bytes
  - KeyMaterial: a sequence of byte containing the data used for generating the keys
  - IV Size: the size of the IV for the CBC encryption

Phase 1 – Sever Hello

The server replies with a **`server_hello`** message that contains the same parameters of the client message but with the following convention:

- The version field contains the lowest suggested by the client and highest supported by the server
- The random field is generated by the server and is independent of the client value
- If the SessionId of the client was nonzero then the same value is used by the server; otherwise the sever create a new session number
- The CipherSuite contains the one selected by the server among the one proposed by the client
- The Compression field contains the one selected by the server among the one proposed by the client

Phase 2 – Server Authentication and Key Exchange

The server sends its x.509 certificate(s) to the client

- The certificate is required for any key-exchange method except anonymous Diffie-Hellman

Next the server sends – if required – a **`server_key_exchange`** message.

- This is not required if the server has sent a certificate with fixed Diffie-Hellman or a RSA key exchange is to be used

Optionally, a server not using the anonymous Diffie-Hellman can request the client for certificates

Finally this phase is always concluded by a **`server_done`** message and wait for the client response

Phase 3 – Client Authentication and Key Exchange

In this phase, the client must verify the server's certificate and check that the server's parameters in the hello message are acceptable

If the server has requested a certificate, the client sends a cert message

- If no suitable certificates are available the client replies with a **`no_certificate`** message

The client proceeds with the **`client_key_exchange`** message where the parameters depends on the type of key exchange used

Finally the client may send a **`certificate_verify`** message where a signature of the parameters exchanged is created with the client private key

- This enable the server to verify that the private key is indeed owned by the client

Phase 4 - Finish

In this phase the client sends a **`change_cipher_spec`** message and copies all the pending CipherSpec into the current CipherSpec

- This message is not considered part of the Handshake protocol but part of the Change Cipher Spec Protocol

Then the client sends a **`finished`** message using the new keys/secrets

- This message verifies that the key exchange and authentication process were successful

In response of this two messages the server sends its own **`change_cipher_spec`** and **`finished`** messages

At this point the handshake is considered completed and the client and server can exchange data securely

Fin!