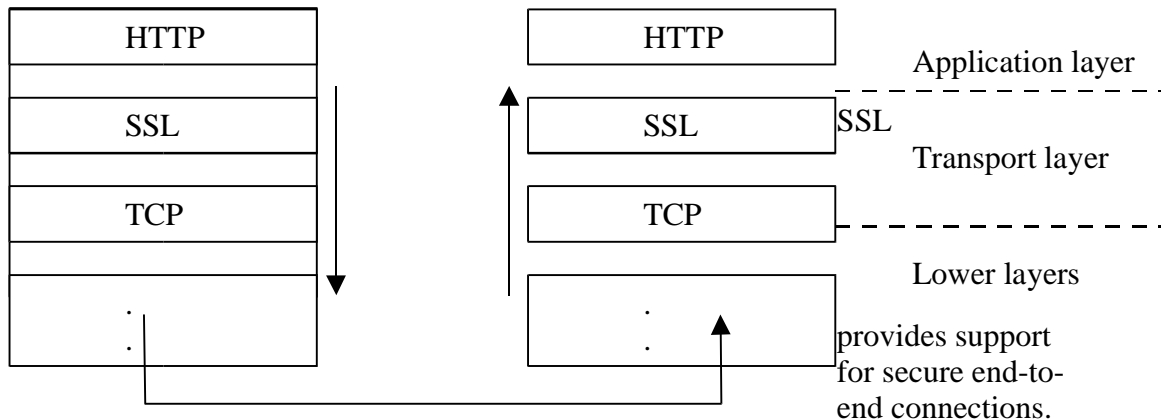


Overview of TLS

- Secure Socket Layer (also known as TLS)
- (a) The SSL protocol operates on top of TCP/IP and provides services to the application layer. This enables higher layer protocols, such as HTTP, to make use of the SSL functionality. The capabilities that SSL provides address fundamental concerns about communication about over the Internet and other TCP/IP networks.



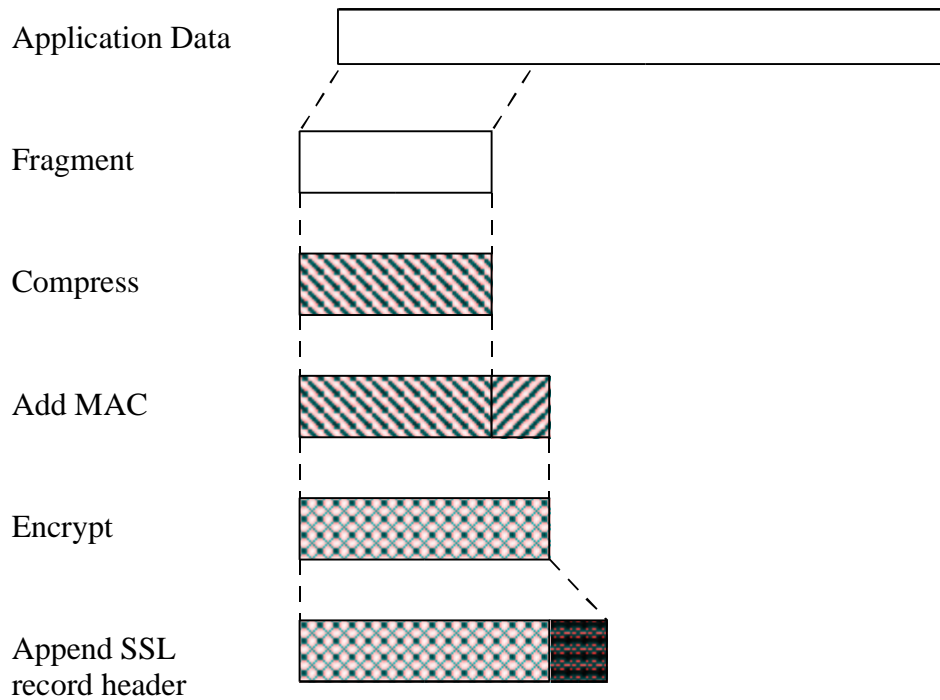
Services SSL provides are:

- **Confidentiality:** A shared secret is established that is used for conventional encryption of the SSL payload
- **Message Integrity:** A shared secret is used to form a message authentication code
- **Authentication:** At least the server is authenticated to the client

The SSL protocol defines three sub-protocols:

- **SSL Record Protocol:** Is responsible for formatting the application data to appropriate SSL format, to be handled by the next SSL layer
- **SSL Handshake Protocol:** The most complex part of SSL. This protocol allows the server and the client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in a SSL record
- **SSL Change Cipher Spec Protocol:** The simplest part of the SSL protocol. The protocol consists of a single message. Its sole purpose is to cause the pending state of the session to be copied into the current state, which updates the cipher suite to be used on this connection
- **SSL Alert Protocol:** The Alert Protocol is used to convey SSL-related alerts to the peer entity. Such as: handshake failure, certificate related alerts, MAC errors etc.

The SSL Record Protocol



The SSL Record protocol is comprised of x steps. This is the part of the protocol that formats the message blocks from the higher layers into the appropriate SSL format. These steps are:

- Fragmentation : Each message is fragmented into 16Kb blocks or less
- Compression : Compression is optionally applied on the fragmented data blocks
- Message Authentication Code: A MAC is computed over the compressed data. For this purpose a shared secret key is used.

Hash (MAC_write_secret || pad_2 || hash (MAC_write_secret || pad_1 || seq_num || SSLCompressed.type || SSLCompressed.length || SSLCompressed.fragment))

Hash is MD5, SHA-1 or both. SSLCompressed.type is the higher level protocol used to process this fragment.

- Encryption : The compressed message with the MAC is then encrypted
- SSL Record header: Finally the SSL record header is appended on the encrypted blob. The header consists of the following fields:
 - Content type: the higher layer protocol used to process the enclosed fragment
 - Major version: Major version of SSL used
 - Minor version: Minor version in use
 - Compressed length: The length in bytes of the plaintext fragment

Content type	Major ver.	Minor ver.	Compressed length
Plaintext (Optionally compressed)			
MAC (0, 16 or 20 bytes)			

↑
Encrypted
↓

SSL is sometimes incorporated into applications directly. Such applications are web-browsers, web-servers and email-clients. SSL can also be incorporated at a lower level, maybe as part of the underlying protocol, in which case it is transparent to the applications.

(b) The SSL Handshake protocol is responsible for:

- Establishing a logical connection between the server and the client
- Authenticating the server and the client, and negotiating all the necessary security parameters for key exchange both for bulk encryption and for MAC operations.

Next we describe the exchange of messages between a client and a server. The key exchange method and the cipher suite used are negotiated at the beginning of the session.

We choose RSA as the key exchange method, and for the CipherSpec we choose 168bit 3DES as the cipher algorithm used for bulk encryption and MD5 with SHA1 as the chosen MAC algorithm. This is the strongest cipher suite. Both SSL 2.0 and 3.0 support it. Client authentication is not required.

After the initial client_hello and server_hello messages, where the Cipher Suite is negotiated and agreed, the next step is server authentication and key exchange.

1. Negotiate security parameters: The connection is initiated by the client. The client sends a client_hello message which includes the following parameters which are used to create the master_secret:

- Suggested Cipher suites
- ClientHello.random

The server will then respond by sending back a message, which contains:

- Chosen Cipher suite
- Server Certificate
- ServerHello.random

2. Server authentication: The server begins the authentication phase by sending his certificate. The message contains one or a chain of X.509 certificates. The client will check the validity of the identity the certificate claims to represent. This is done by checking amongst other things, like validity period, domain name etc., and the validity of the CA's digital signature. In the case of a web-browser trusted Certificate Authorities are embedded into the browser. The browser can also contact a central server to check if a public key has been revoked in the meantime.

3. Master Secret creation: The server certificate sent to the client contains the server's public key. The client can then use the public key to encrypt the pre_master_secret, which is generated by the client. The pre_master_secret is used by both the server and the client, to create the master_secret. The creation is in two stages. First the pre_master_secret is created and exchanged, and then both sides calculate the master secret.

When RSA is used the pre_master_secret is a one-time 48-byte value, created by the client. The client will then create the master_secret:

```
master_secret = MD5 (pre_master_secret || SHA1 ('A' || premaster_secret
||Clienthello.random||ServerHello.random))||
MD5 (pre_master_secret || SHA1 ('BB' || premaster_secret
||Clienthello.random||ServerHello.random))||
MD5 (pre_master_secret || SHA1 ('CCC' || premaster_secret
||Clienthello.random||ServerHello.random))
```

The client and the server also need a key to use for MAC operations. This is derived from a PRF which uses as the seed of the function the master_secret and as salt the initial random values from both sides.

```
key_block = MD5 (master_secret || SHA1 ('A' || premaster_secret
||Clienthello.random||ServerHello.random))||
MD5 (master_secret || SHA1 ('BB' || premaster_secret
||Clienthello.random||ServerHello.random))||
MD5 (master_secret || SHA1 ('CCC' || premaster_secret
||Clienthello.random||ServerHello.random))....
```

This is done until the necessary number of bytes is generated for :
client_write MAC, server_write MAC, client_write key, server_write key. The final ciphertext block from each record is preserved for use as the client IV, server IV for the following record.

The client then sends the premaster key encrypted with the server's public key and the PRF value.

The server can then decrypt the premaster secret and carry out the same calculations as the client in order to generate the master secret. The server can decrypt and check the validity of the messages exchanged by calculating the PRF value out of the messages exchanged so far with the client. If the two values match the server accepts the transition of the state from pending to current and the derived keys are used for the rest of the session.