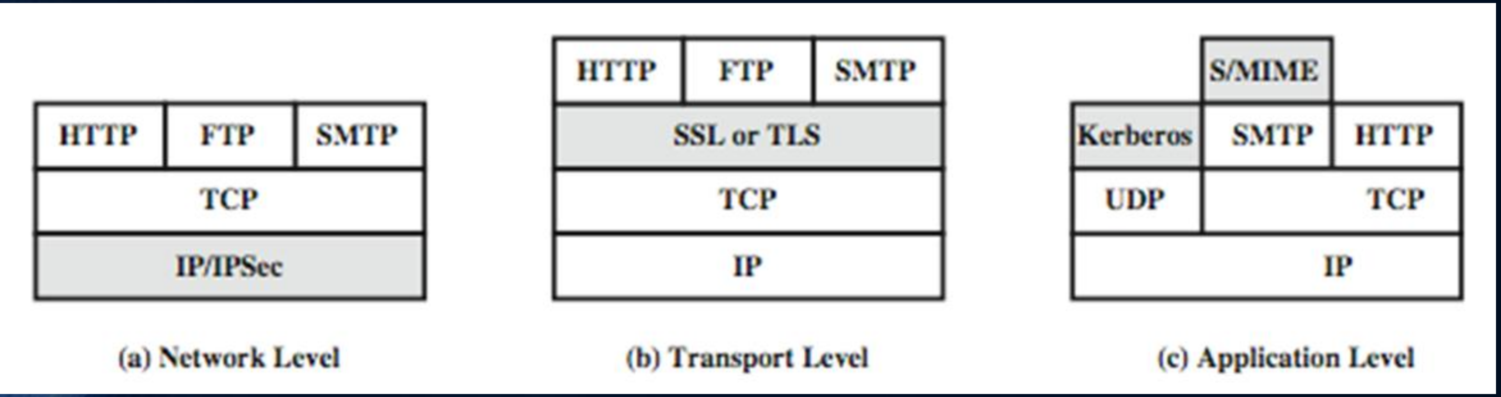
SSL / TLS transport protocol

SSL / TLS

APM@FEUP

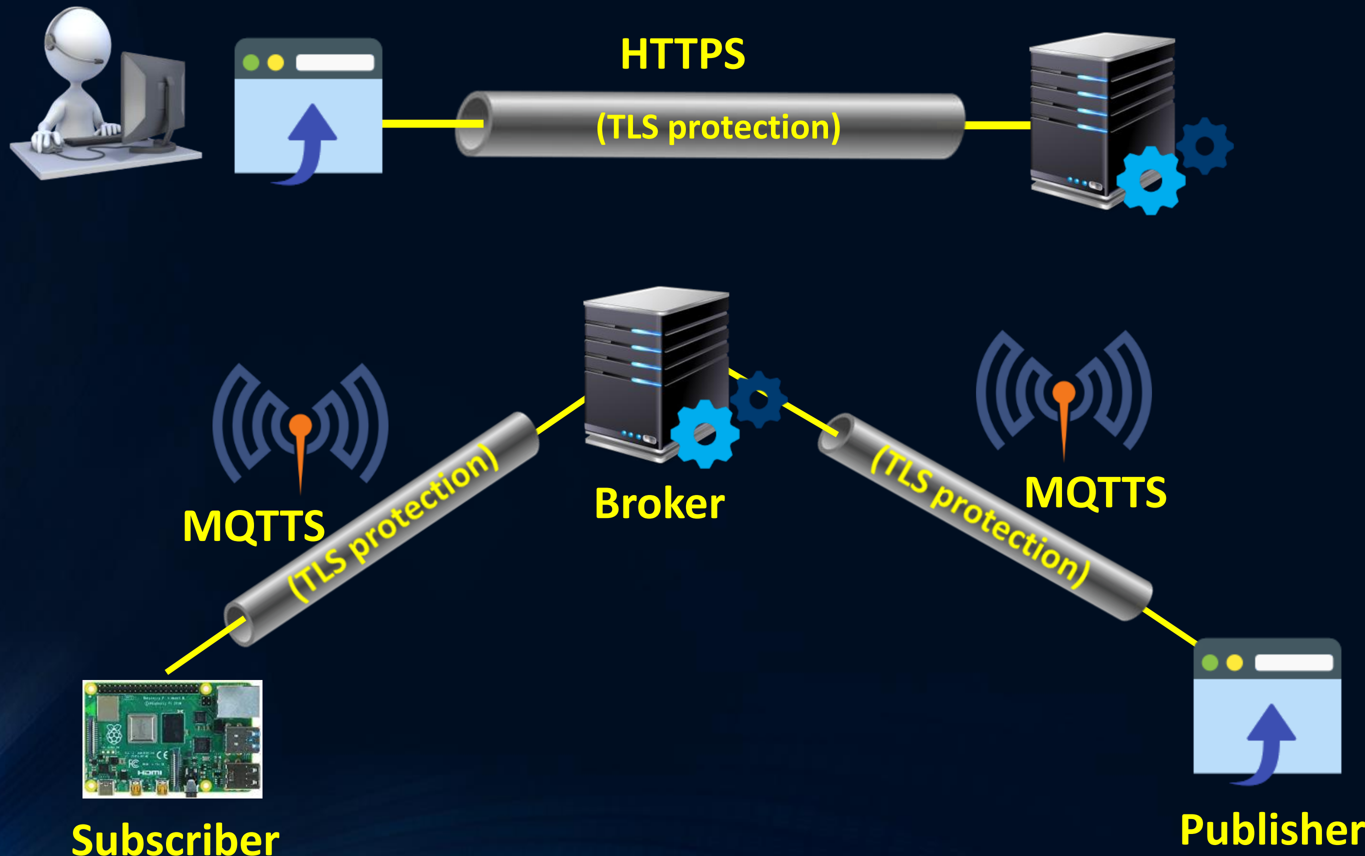
The SSL/TLS protocol

- **Web traffic (as all network traffic) is subject to many threats**
  - integrity, confidentiality, authentication theft, denial of service, …
  - need added security mechanisms
- **Traffic security can appear at several levels in the network**
  - At the lowest protocol level (IPSec) embedded in the network
  - Just above TCP level but used by several high-level protocols
    - Implemented in specific packages (e.g., browsers and web servers)
  - At the application level using underlying standard protocols



TLS as the underlying protocol

TLS can be the security mechanism added to other higher-level protocols like HTTP or MQTT

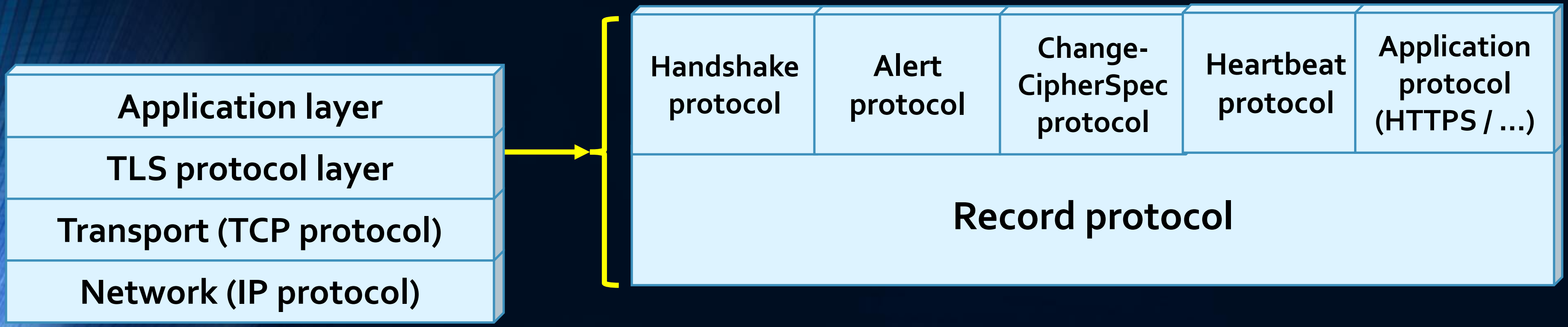


SSL / TLS evolution and standards

- **SSL (secure sockets layer) is a transport protocol**
  - Originally developed by Netscape
  - Makes use of TCP to provide a reliable end-to-end service
  - Version 3 was presented as a draft internet standard
- **The approval of SSL by IETF became the TLS standard**
  - TLS – Transport Layer Security (RFC 2246 – TLS 1.0, 1999)
  - TLS 1.0 is essentially SSL v. 3.1 and was backward compatible not causing disruptions
  - TLS evolved with standards RFC 4346 in 2006 (1.1) and RFC 5246 in 2011
    - Improved cryptography (e.g., SHA-256 and AES, better encryption modes, …)
  - RFC 6167 (2011) and RFC 7568 (2015) refined all TLS versions
    - removed backward compatibility with SSL 2.0 and SSL 3.0
  - TLS 1.3 (RFC 8446) was already approved (2018)
    - operations independent of cipher suites
    - removing support for weaker and lesser used cryptography algorithms
    - Session hash and new signature and key exchange algorithms

TLS stack and architecture

- **TLS protocol is composed of two layers of sub-protocols**
  - Handshake allows encryption, MAC and keys negotiation and authentication
  - Alert conveys alert messages and errors
  - ChangeCipherSpec allows updating the cipher suite in use and making it current
  - Heartbeat checks link operation and prevents disconnection
  - The Record protocol is the data transmission format for exchanging application data

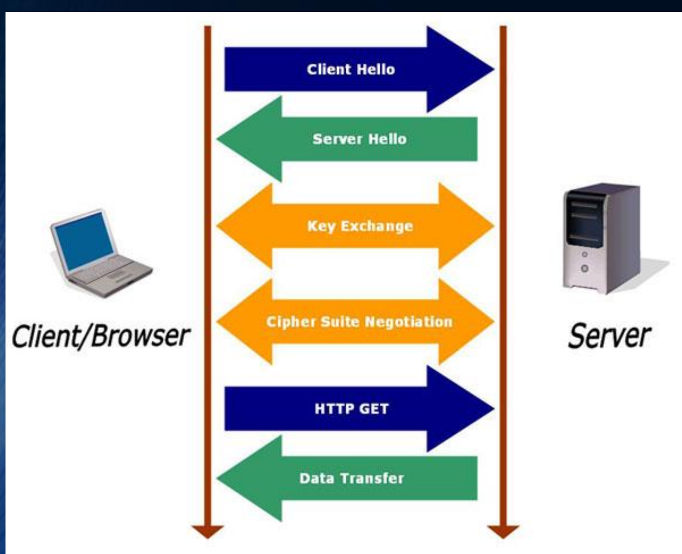


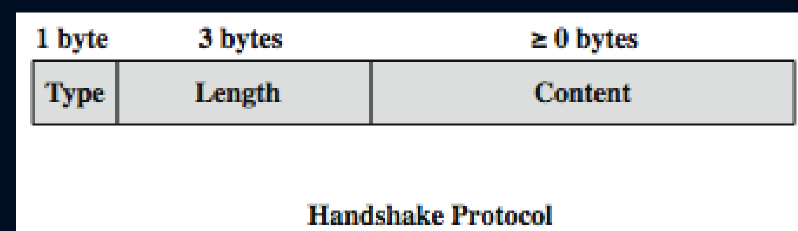
TLS architecture and properties

- **The Handshake protocol establishes a TLS session**
  - Association and link between client (browser) and server
  - Defines a set of cryptographic parameters
  - Conducts a server authentication in the client using a certificate
  - Optionally can do the same relatively to the client (seldom used)
- **The TLS secure channel provides three properties**
  - Confidentiality using symmetric encryption and decryption with keys generated for the session
  - Integrity of data using a message hash and a MAC also with keys generated for the session
  - Server authentication preventing server spoofing and the establishment of connections to attackers unknowingly
    - Using a certificate for the server domain or organization

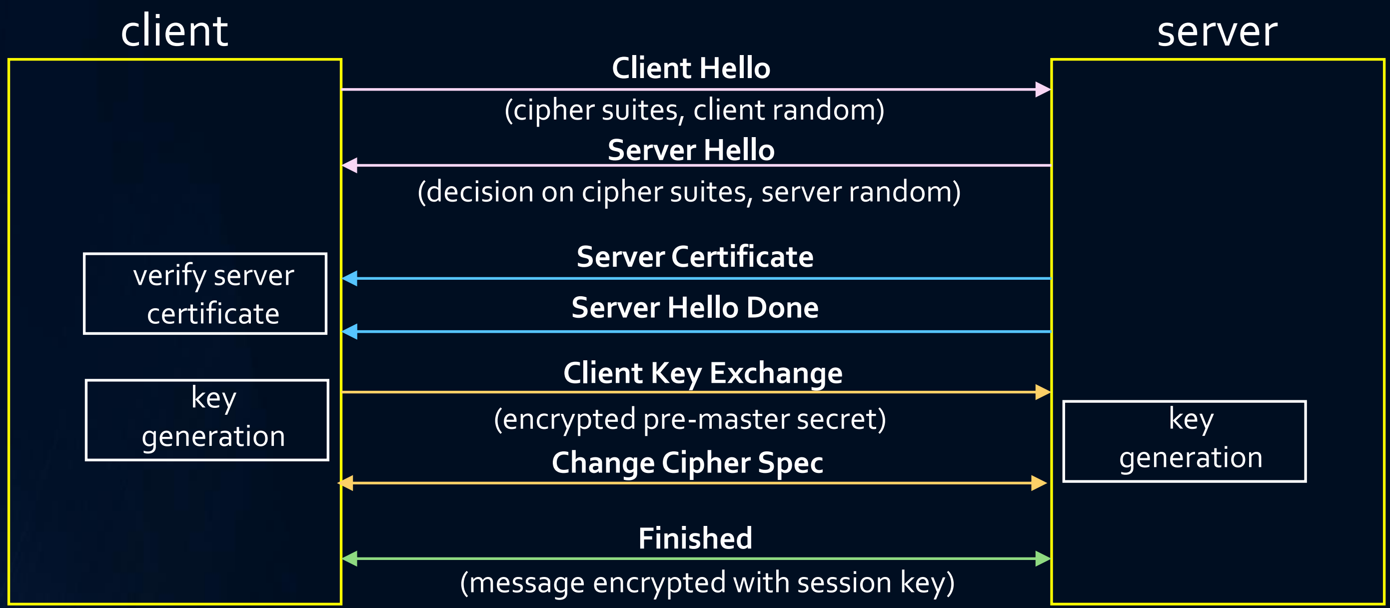
Handshake and data transfers

- **Initiated by the client to establish a TLS session**
  - It has several phases and consists of the exchange of simple messages
  - negotiate encryption, mode, hash and MAC algorithms
  - authenticate the server (optionally also the client)
  - generate and exchange session cryptographic keys
- **The keys established in the handshake protocol are used for HTTP requests through the Record protocol**





Handshake operations



Client Hello – transmits the highest version understood and supported algorithms (crypto algorithms, key exchange, and compression) in preferred order. Also, a value (client_random) composed of a time stamp and a cryptographic random value is transmitted.
Server Hello – transmits the decision of the server concerning version and algorithms. A similar server_random value is also transmitted.

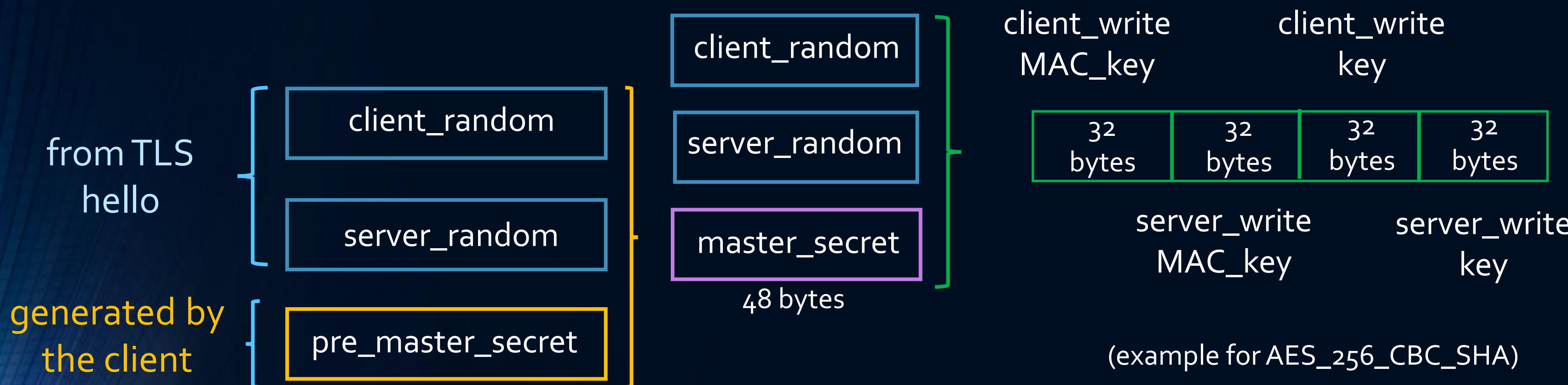
The server must send his certificate, and its thorough verification in the client is crucial.
The **Server Hello Done** message terminates the algorithms negotiation

The **Client Key Exchange** message transmits a secret (pre-master-key) generated in the client and usually encrypted using the public key of the server certificate. This key is used by both client and server, together with the client and server_random values, to generate the session keys.
The **Change Cipher Spec** messages turn the session keys effective.

Finally, the **Finished** messages uses hash, MAC and encryption to test the session keys in both senses.

TLS session keys generation

- **From the pre-master-secret both sides gain the same keys**
  - **The pre-master-key is generated by the client and transmitted to the server (using RSA or DH)**
    - **Then a master-secret is computed in both sides**
    - **Finally, the session keys are computed in both sides**



The master_secret and session keys are generated in both sides using an agreed upon pseudo-random function

u_0 = label || server_random || client_random, where || denotes concatenation and label a string like "master secret"

u_i = HMAC(secret, u_{i-1}), where secret is the premaster or master secret

output = u_1 || u_2 || …, retaining only the necessary bytes

TLS Data Transmission

- **Data transmission is in both directions using records**
  - Follows the TLS Record protocol data format
    - Is used by all the TLS sub-protocols, after key exchange and generation phase in the Handshake protocol
  - Contains a header and a payload
    - The header is divided into
      - Content type (1 byte) indicating the sub-protocol (Handshake, ChangeCipherSpec, Alert, Heartbeat and Application
      - Version (2 bytes) indicating the SSL / TLS version
      - Length (2 bytes) with the payload length in bytes, until a maximum of 2^{14}
    - The payload contains the data transmitted in this record (may be compressed), with an appended MAC code and padding, all encrypted
      - The message presented to the MAC algorithm contains the record sequence number, the compression type and version (if any), the length (after possible compression), and the compressed bytes (fragment)

TLS Record

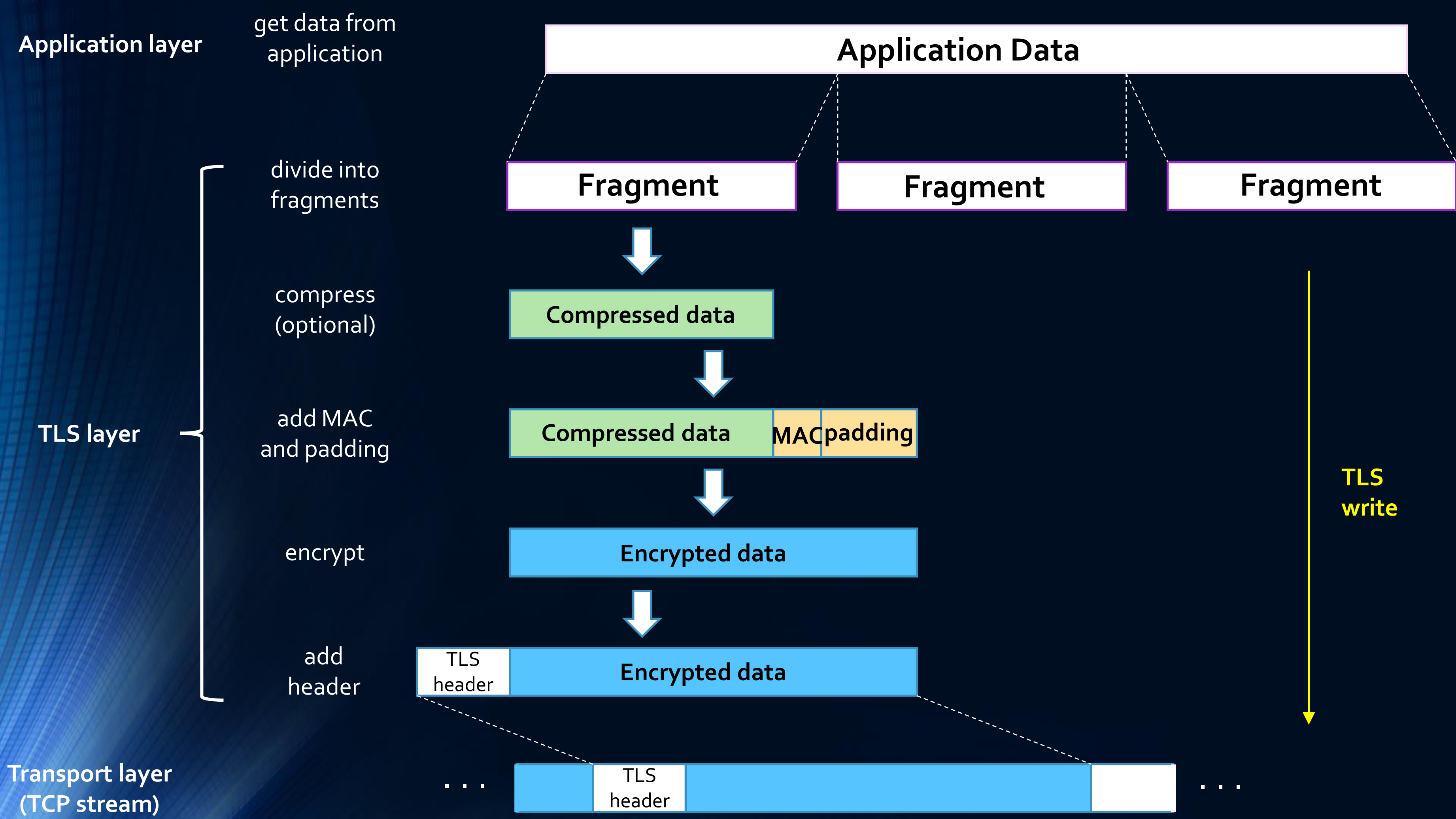
Content type	Version	Length	Data (may be compressed)	MAC	Padding

TLS header

TLS payload (encrypted)

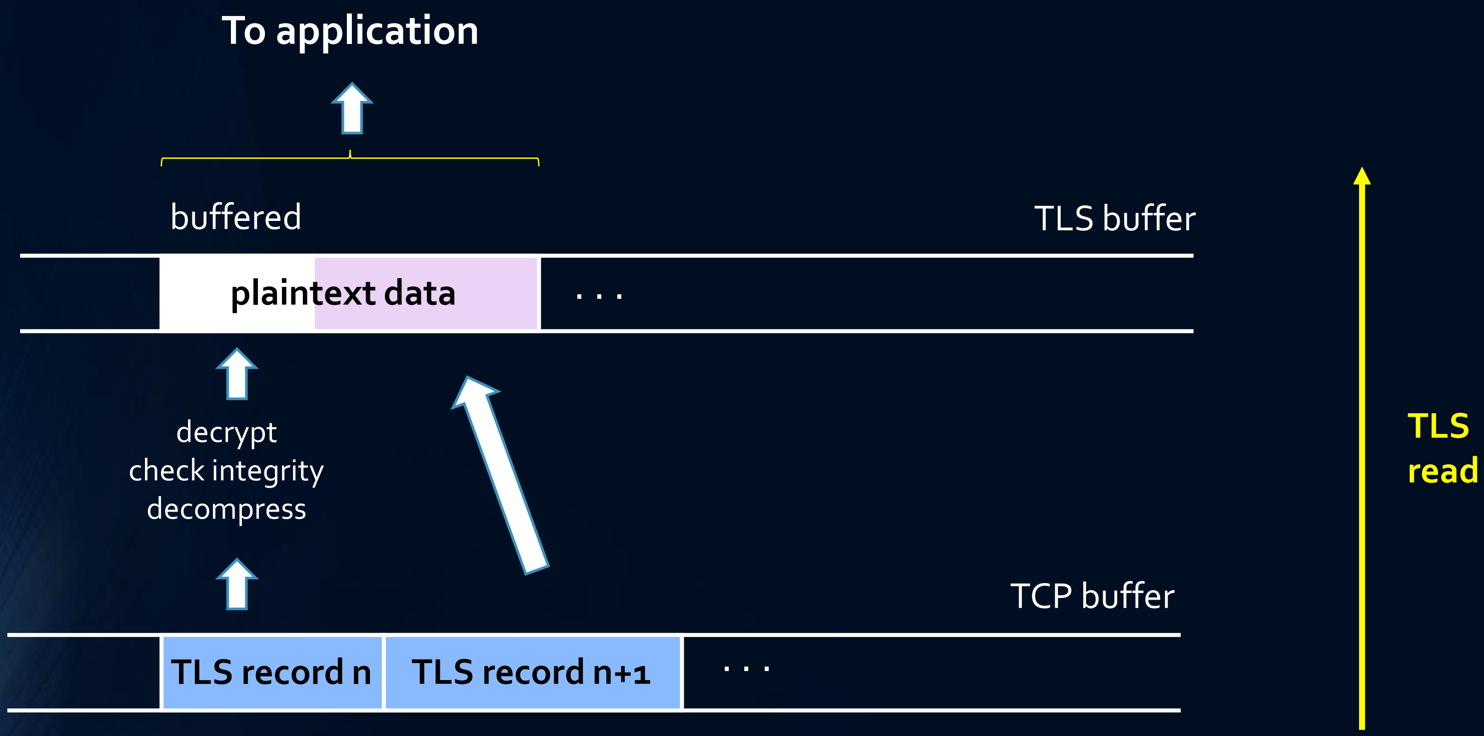
TLS write operation

- Usually, data from HTTP requests or responses (application)



TLS read operation

- **The plaintext data is recovered before sent to application**
  - the browser or web server showing and running the web application



HTTPS

- **HTTP over TLS**
  - Combination of HTTP and TLS to secure communications between browser and server
  - follows the IETF standard RFC 2818
    - specifies TLS handshake followed by normal HTTP requests and responses
    - no fundamental changes from SSL to TLS
  - The URL begins with https://... rather then http://...
  - Uses port 443 instead of port 80 (by default)
- **Allows confidentiality and integrity over the HTTP data**
  - URL addresses
  - document contents
  - form data
  - cookies
  - HTTP headers