The exercises are designed for students to finish in an individual capacity. The exercises are not designed to be completed in tutorial sessions but rather to give you some tasks and a starting point to continue and complete on your own.

---

1 TLS

1.1 Lab Tasks

A TLS handshake takes place whenever a user navigates to a website over HTTPS; it involves multiple steps, as the client and server exchange the information necessary for completing the handshake and making further conversation possible. In this lab, we will capture a TLS handshake using Wireshark and analyse the details of the connection establishment.

Open SecureCorp network configuration in GNS3 and start all nodes. You can use one of the workstations to perform the lab tasks.

- **Browse an HTTPS website with lynx**
  Start packet capture on the link between `Internal-Client` and `Switch3`, and browse to monash.edu:

```
cd
lynx https://www.monash.edu
```

  Wireshark should have captured the lynx traffic.

- **Client Hello**
  Open Wireshark and find "Client Hello" packet. TLS wraps all traffic in "records" of different types. We see that the first byte out of our browser is the hex byte 0x16 = 22 which means that this is a "handshake" record:

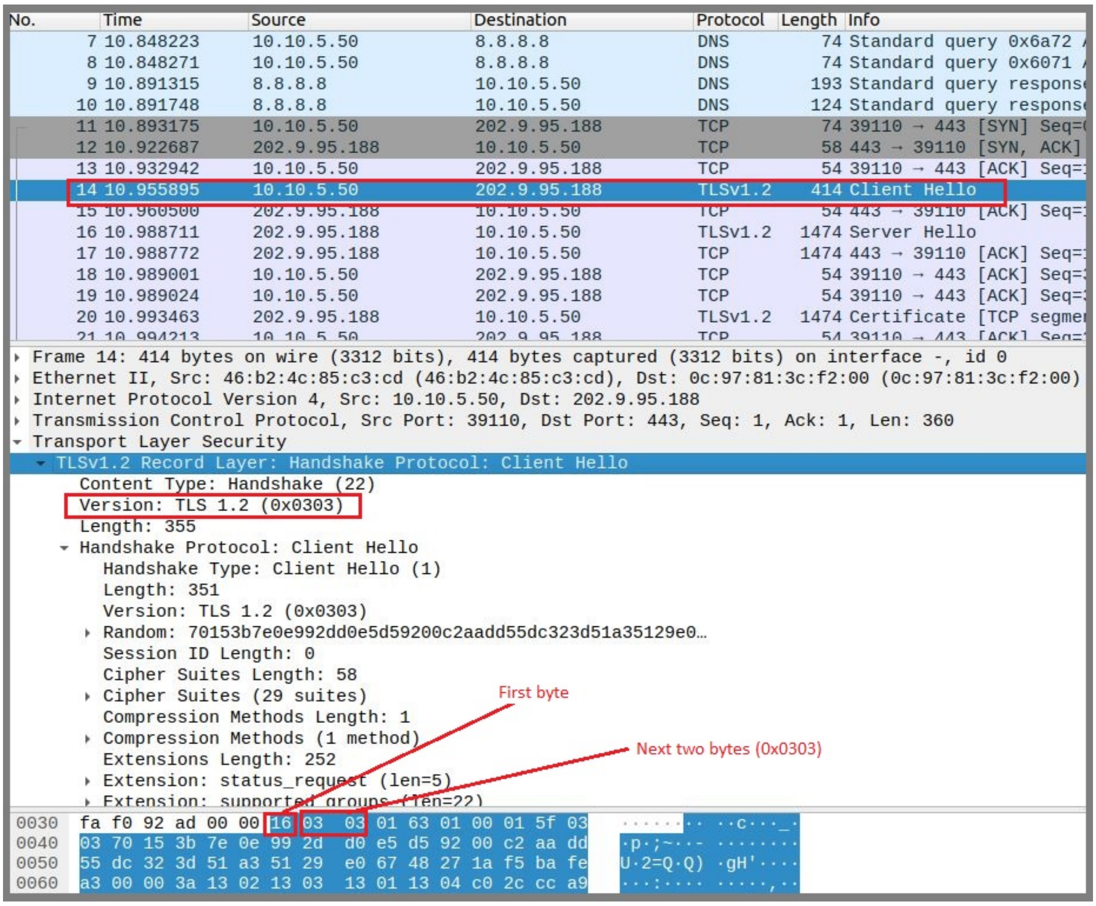


Figure 1: Client Hello

The next two bytes are 0x0303 which indicate that this is a version 3.3 record which shows that TLS 1.2 is essentially SSL 3.3. The handshake record is broken out into several messages. The first is our "Client Hello" message (0x01). There are a few important things here:

1. **Random**

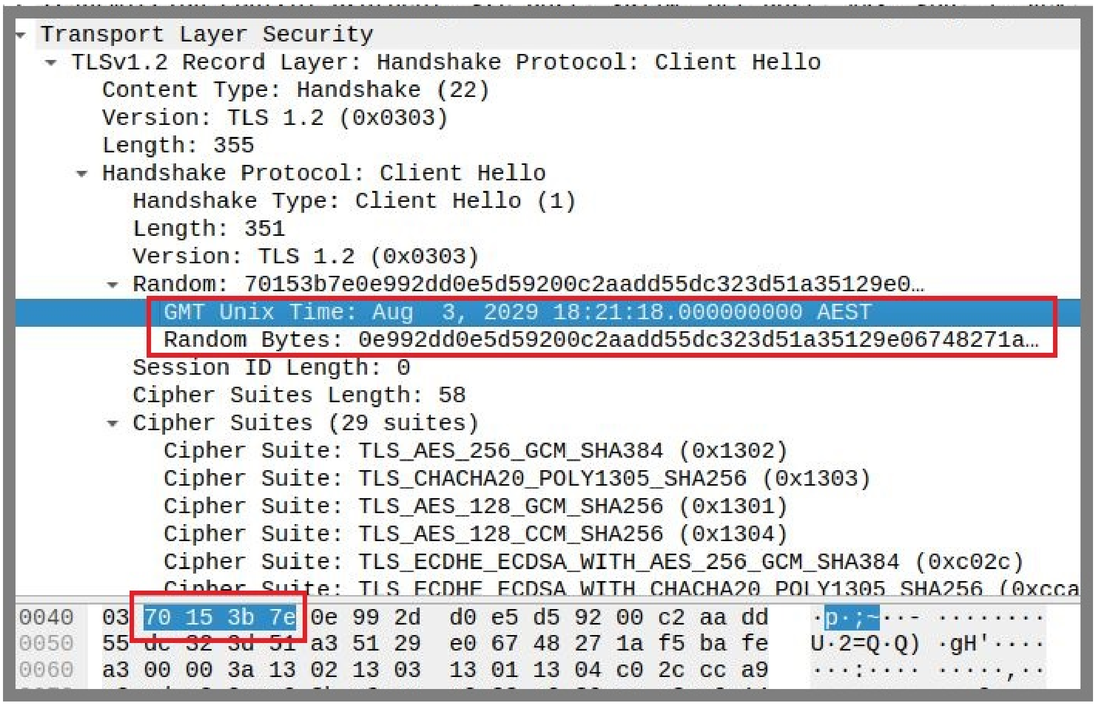


Figure 2: Random:

There are four bytes representing the current Coordinated Universal Time (UTC) in the Unix epoch format, which is the number of seconds since January 1, 1970. In this case, 0x70153b7e. It's followed by 28 random bytes. This will be used later on.

2. **Cipher Suites**:

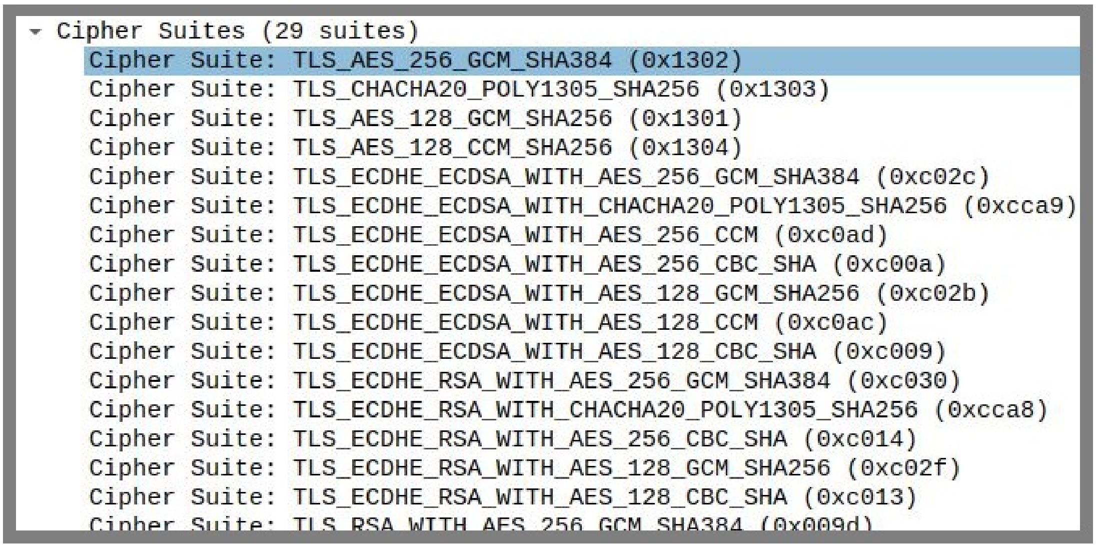


Figure 3: Random

This is a list of all of the encryption algorithms that the browser is willing to support. Its top pick is a very strong choice of `TLS_AES_256_GCM_SHA384` followed by 28 others that it's willing to accept.

3. **server_name extension**:

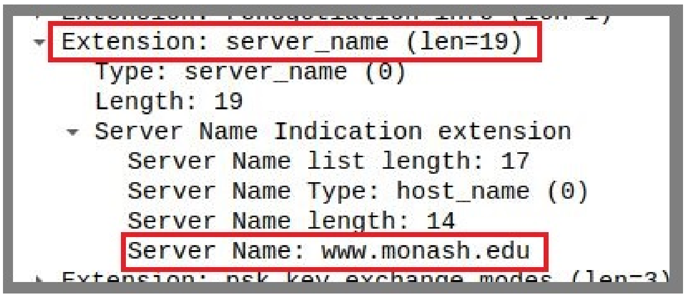


Figure 4: Random

This is a way to tell monash.edu that our browser is trying to reach https://www.monash.edu. This is really convenient because our TLS handshake occurs long before any HTTP traffic. HTTP has a "Host" header which allows hosting companies to host hundreds of websites onto a single IP address. SSL has traditionally required a different IP for each site, but this extension allows the server to respond with the appropriate certificate that the browser is looking for.

4. **Session ID**

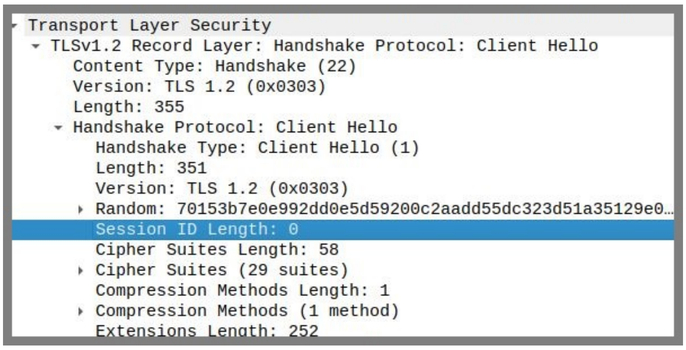


Figure 5: Session ID

Here it's empty/null. If we had previously connected to Monash a few seconds ago, we could potentially resume a session and avoid a full handshake.

- **Server Hello**
  Monash replies with a handshake record. The record has version bytes of 0x0303 meaning that Monash agreed to our request to use TLS 1.2. This record has three sub-messages with some interesting data:

  1. "Server Hello" Message (2)

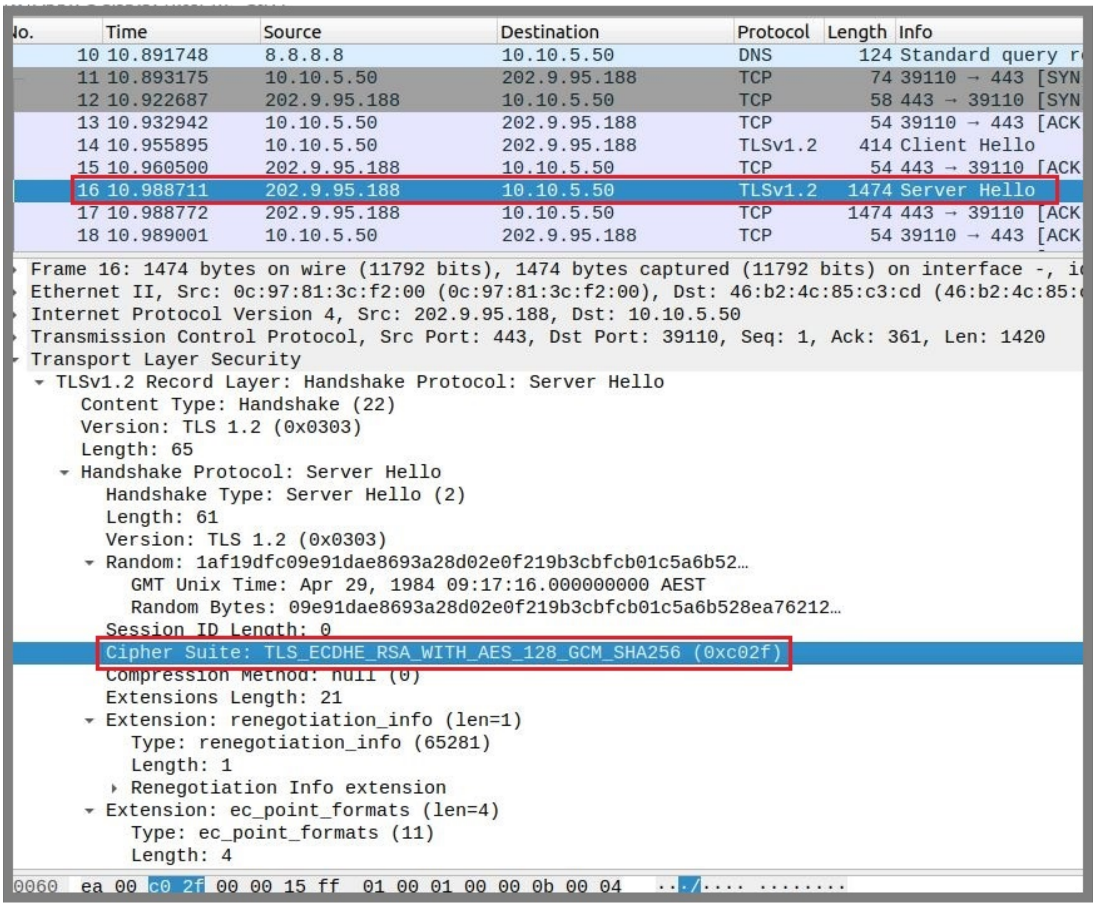


Figure 6: Server Hello

Of 29 cipher suites we offered, Monash didn't pick our first choice, instaed offered to use `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`. So we will be using:

- **ECDHE**: Key exchange with Ephemeral Elliptic Curve Diffie-Hellman
- **RSA**: Signature with RSA (Rivest–Shamir–Adleman)
- **AES_128**: 128-bit Encryption with Advanced Encryption Standard
- **GCM**: AES Mode of Operation with Galois/Counter Mode (GCM)
- **SHA256**: SHA256 hash function to verify the contents of the message

2. **Certificate Message**
The "Server Hello" is not yet done. Find "Certificate" packet which is also part of TLS handshake.

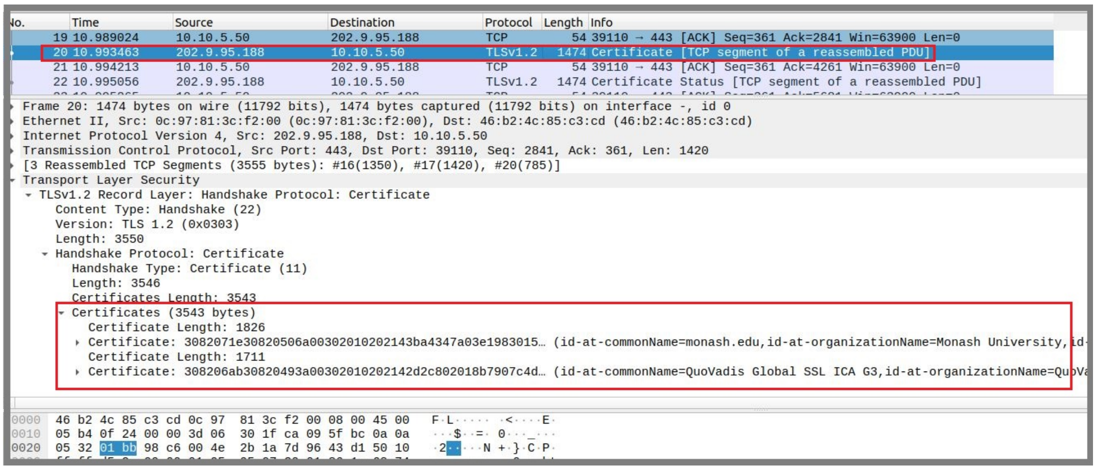


Figure 7: Certificate

Notice that the certificate size is 3543 bytes, but the packet size is 1474 bytes. The certificate is taking more than one packet and Wireshark is showing us the certificate after re-assembling TCP segments (3 segments in this case).

3. **Certificate Status**
   Find "Certificate Status" packet. This message validates whether the server's X.509 digital certificate is revoked or not, it is ascertained by contacting a designated OCSP (Online Certificate Status Protocol) server. The OCSP response, which is dated and signed, contains the certificate status. The client can ask the server to send the "certificate status" message which contains the OCSP response. This approach is known as OCSP Stapling. The process saves bandwidth on constrained networks as it prevents OCSP servers from getting overwhelmed with too many client requests.

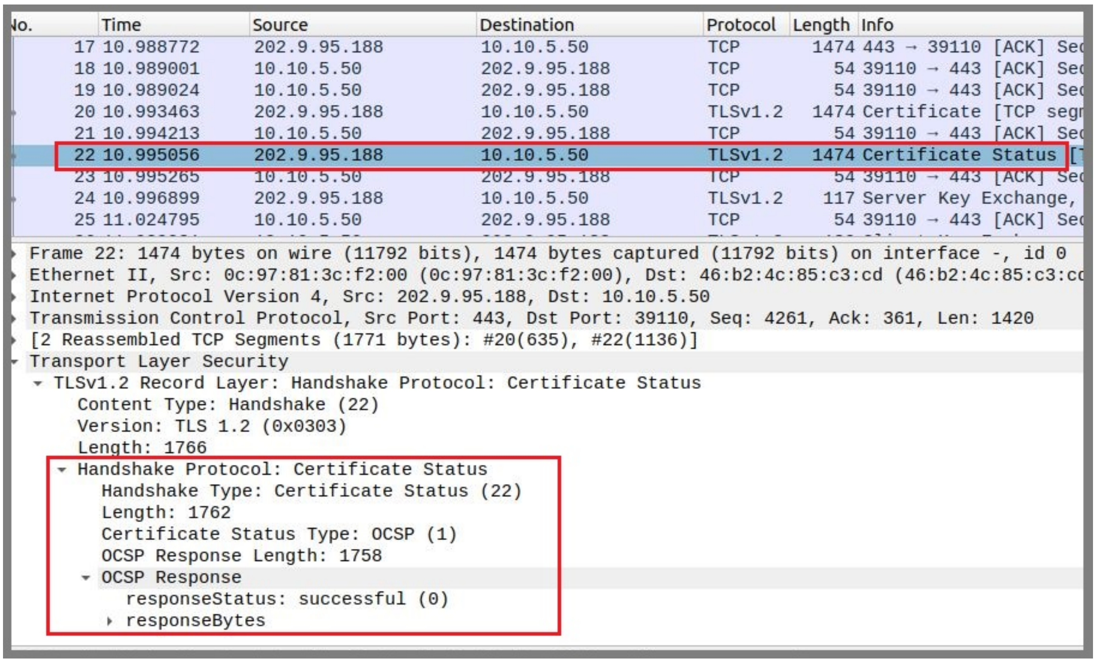


Figure 8: Certificate Status

4. **Server Key Exchange**
   Find packet "Server Key Exchange, Server Hello Done". The "Server Key Exchange" message is optional and sent when the public key present in the server's certificate is not suitable for key exchange or if the cipher suite places a restriction requiring a temporary key. This key is used by the client to encrypt Client Key Exchange later in the process.

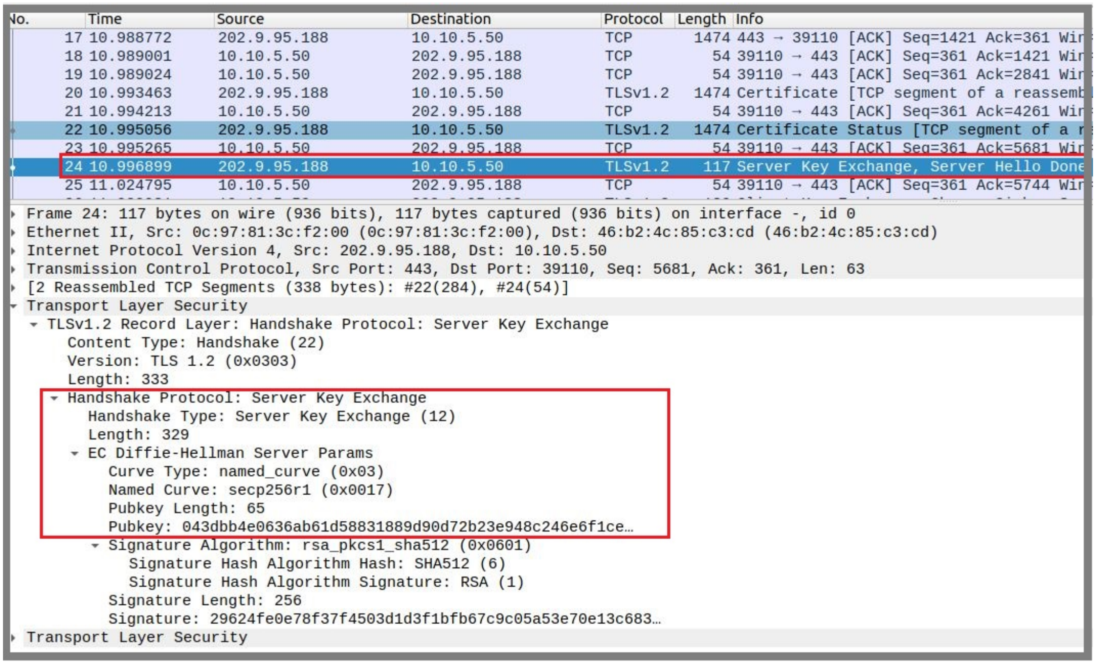


Figure 9: Server Key Exchange

5. **Server Hello Done**
   This is a zero byte message that tells the client that it's done with the "Hello" process and indicate that the server won't be asking the client for a certificate.

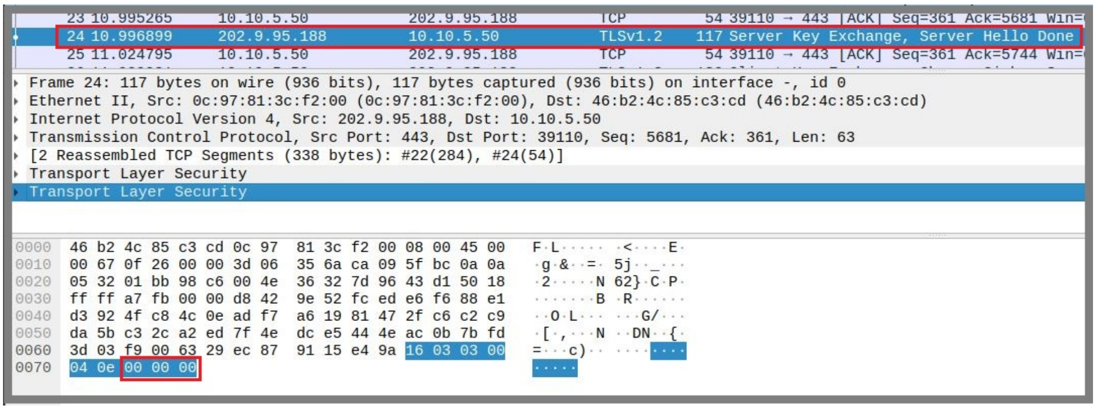


Figure 10: Server Hello Done

The browser will verify the Certificate Authority's signature on the server's certificate, and if all goes well it will continue to the next step, otherwise the browser will complain about the validity of the certificate and let the user decide if they still want to continue.

- **Client response to server**
  Find "Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message" packet.

1. **Client Key Exchange**
   The "Client Key Exchange" message contains: (a) the protocol version of the client which the server verifies if it matches with the original client hello message, (b) pre-master secret – a random number generated by the client and encrypted with the server's public key.

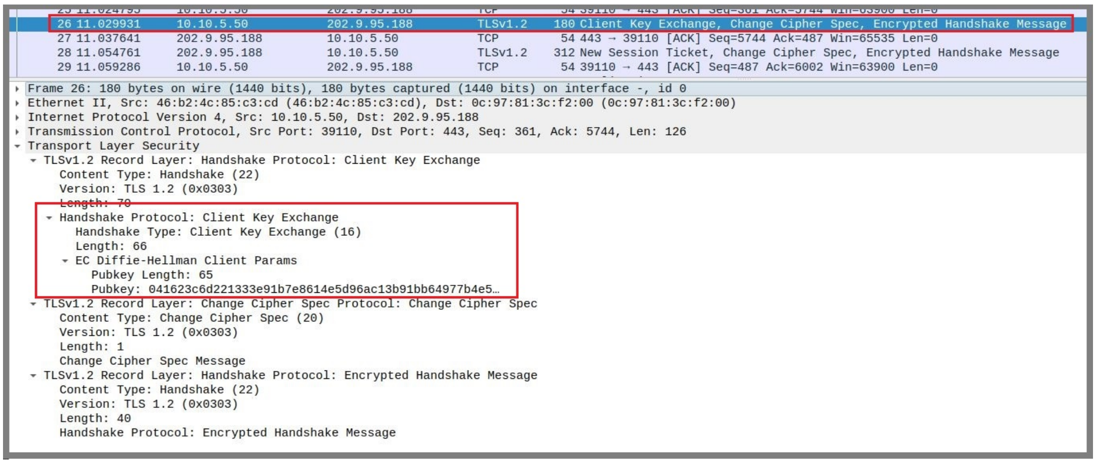


Figure 11: Client Response

If the server can decrypt the message using the private key and can create the master secret locally, then the client is assured that the server has authenticated itself.

2. **Change Cipher Spec**
   This message notifies the server that all the future messages will be encrypted using the algorithm and keys that were just negotiated.
3. **Finished (Encrypted Handshake)**
   The Finished message is complicated as it is a hash of all the messages exchanged previously along with a label ("client finished"). This message indicates that the TLS negotiation is completed for the client.

- **Master secret**
  If we've done everything correctly, both sides (and only those sides) now know the pre-master secret. There's a slight trust issue here from Monash's perspective: the pre-master secret just has bits that were generated by the client, they don't take anything into account from the server or anything we said earlier. We'll fix that by computing the "master secret." This is done by calculating:

  $$master\_secret = PRF(pre\_master\_secret, "mastersecret", ClientHello.random + ServerHello.random)[0..47];$$

  The two random values `ClientHello.random` and `ServerHello.random`, sometimes called "nonces", are randomly generated and sent during the Hello of each parties (Remember?). This is to bound the soon-to-be master key to this session. `PRF` stands for Pseudo-random function, basically some concrete construction that emulates a random oracle: given an input will produce an output computationally indistinguishable from a truly random sequence.

- **Server response to Client**
  The server informs the client that the messages will be encrypted with the existing algorithms and keys. The record layer now changes its state to use the symmetric key encryption.

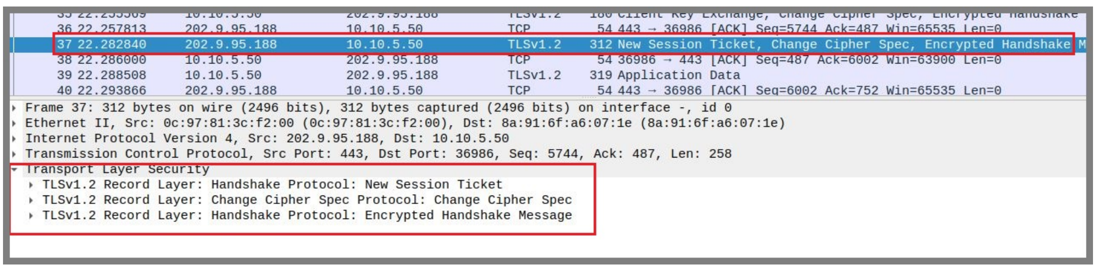
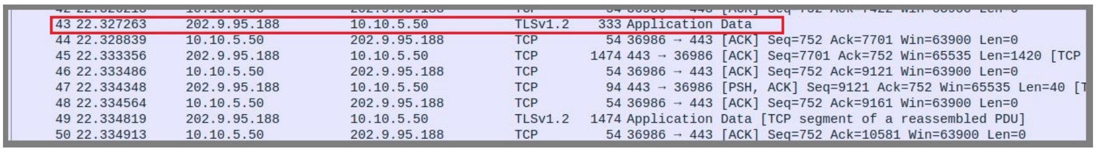


Figure 12: Server Response

- **Application Data Flow**
  The master secret will be used for generating encryption keys (AES_GCM as agreed by both parties), MAC secrets, and IVs. The key generation algorithms are not the scope of this lab, refer to the RFC 5246 (`https://datatracker.ietf.org/doc/html/rfc5246/`) for more details.

  Once the entire TLS Handshake is successfully completed and the peers validated, the applications on the peers can begin communicating with each other.



43	22.327263	202.9.95.188	10.10.5.50	TLSv1.2	333 Application Data
44	22.328839	10.10.5.50	202.9.95.188	TCP	54 36986 → 443 [ACK] Seq=752 Ack=7701 Win=63900 Len=0
45	22.333356	202.9.95.188	10.10.5.50	TCP	1474 443 → 36986 [ACK] Seq=7701 Ack=752 Win=65535 Len=1420 [TCP
46	22.333486	10.10.5.50	202.9.95.188	TCP	54 36986 → 443 [ACK] Seq=752 Ack=9121 Win=63900 Len=0
47	22.334348	202.9.95.188	10.10.5.50	TCP	94 443 → 36986 [PSH, ACK] Seq=9121 Ack=752 Win=65535 Len=40 [T
48	22.334564	10.10.5.50	202.9.95.188	TCP	54 36986 → 443 [ACK] Seq=752 Ack=9161 Win=63900 Len=0
49	22.334819	202.9.95.188	10.10.5.50	TLSv1.2	1474 Application Data [TCP segment of a reassembled PDU]
50	22.334913	10.10.5.50	202.9.95.188	TCP	54 36986 → 443 [ACK] Seq=752 Ack=10581 Win=63900 Len=0

Figure 13: Encrypted Application Data

- **TLS 1.0/1.1**
  There are currently four versions of the TLS protocol in use today: TLS 1.0, 1.1, 1.2 and 1.3.

  In March 2021 IETF officially deprecated TLS 1.0 and TLS 1.1.

  TLS 1.0 was released in 1999, making it a nearly two-decade-old protocol. It has been known to be vulnerable to attacks—such as BEAST and POODLE, in addition to supporting weak cryptography, which doesn't keep modern-day connections sufficiently secure.

  TLS 1.1 doesn't have any known protocol vulnerabilities, though it does share support for bad cryptography like TLS1.0.

  According to Shoadan search engine thousands of website around the world still use TLS1.0/1.1.

  Monash does not support TLS1.0/1.1, let's enable TLS1.0 in openssl and try connecting with Monash.

  Open console in `Internal-Client`, and copy the following in the begining of the file `/etc/ssl/openssl.cnf`:

```
openssl_conf = default_conf
[default_conf]
ssl_conf = ssl_sect
[ssl_sect]
system_default = ssl_default_sect
[ssl_default_sect]
MinProtocol = TLSv1
CipherString = DEFAULT:@SECLEVEL=1
```

  The above configuration will set the minimum TLS protocol to TLS1.0.

  Now browse to Monash using TLS1.0 protocol, we can use `s_client` from openssl to do this:

```
openssl s_client -connect monash.edu:443 -tls1
```

  Notice that Monash did not perform handshake with us. Let's try a website which supports TLS1.0, before executing the below command, start Wireshark on the link between `Internal-Client` and `Switch3`:

```
openssl s_client -connect www.sutherlandshire.nsw.gov.au:443 -tls1
```

  Analyse the handshake traffic in Wireshark and compare it with TLS1.2 (especially compare the cipher suites).

Acknowledgment
Parts of this lab is taken from: http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html

1.2 Additional Review Questions

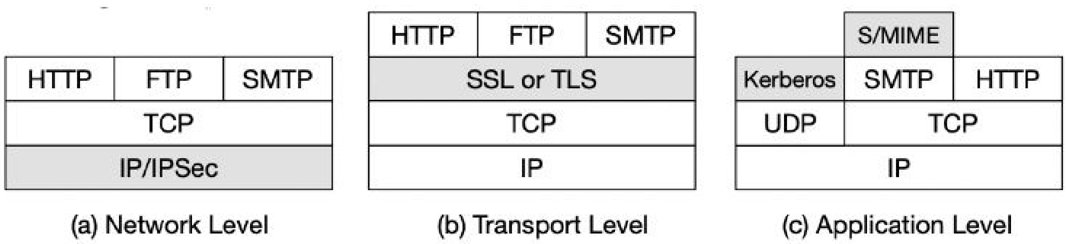


Figure 14: Relative Location of Security Facilities in the TCP/IP Protocol Stack

- What are the advantages of each of the three approaches shown in above figure?
- What services are provided by the SSL/TLS record protocol?
- What steps are involved in the SSL/TLS record protocol transmission?
- What is the purpose of HTTPS?