Author : Praveen Kumar Pendyala
Email : m@praveen.xyz

Problem 3.1

Task 1.1:
TLS version - 1.0
The client can decide the version
Announced in ClientHello record

Task 1.2

Time	Major Records	Major records data
↓	Client Hello	- TLS Version - Cipher suites - Extensions
	Server Hello	- TLS Version - Cipher Suite - Extensions
	Certificate, Server Key exchange, Server Hello Done	- Certificates - EC Diffie-Hellman Server Params - Pubkey - Signature - Curve type
	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message	- EC Diffie-Hellman Client Params - Pubkey - Change Cipher Spec Message - Encrypted Handshake Message
	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message	- TLS Session Ticket - Change Cipher Spec Message - Encrypted Handshake Message

Task 2.1
IP Address : 173.194.69.94 belongs to Google Inc.

Task 2.2
Yes! Nonce value:
86:f5:0f:e1:f7:c4:7b:52:9c:6f:7a:f7:e0:49:e4:15:9d:6d:a5:e5:be:22:0c:79:e9:03:12:e8

Task 2.3
Yes!
Second listed suite: Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
Public key algorithm: ECDHE_ECDSA
Symmetric key algorithm: AES_256_CBC
Hash algorithm: SHA

Task 3.1
Yes!
Chosen Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA
Public key algo: ECDHE_RSA
Symmetric key algo: RC4_128
Hash algo: SHA

Task 3.2
Yes!
Nonce value:
41:56:1f:a9:19:70:65:06:81:2d:d7:cf:a9:19:fe:21:49:23:f0:d0:6b:77:24:b5:38:91:12:1d

Purpose of nonce: The use of nonces prevent a replay attack. Nonces are also used along with the master_key in the generation of a Shared key.

Task 3.3
Next_Protocol_Negotiation is an extension for application layer protocol negotiation. This allows the application layer to negotiate which protocol should be performed over the secure connection. This helps in protocol negotiation without additional Round trip times.

It's empty for ClientHello because of the extension definition. Quoting Google's technical note (https://technotes.googlecode.com/git/nextprotoneg.html):

"The extension_data field of a next_protocol_negotiation extension in a ClientHello MUST be empty."

This can be seen as an indication that the client indeed supports the extension and the server may choose to send a list of protocols if need be.

Task 3.4
2 certificates.
Certificate 1:
Subject - www.google.de (commonName)
Issuer - Google Internet Authority

Certificate 2:

Subject - Google Internet Authority

Issuer - Equifax Secure Certificate Authority

Certificate1 is a certificate to verify the domain **google.de**, while later is a certificate to verify **Google Internet Authority**, issuer of certificate1

Task 4.1

Yes, the record contains a pre-master secret but, not in clear text.
Length of the encrypted record : 65
The client encrypts the pre-master secret using server's public key and sends it back to the server. Only the server can decrypt this.

Task 5.1

With session-ids, the server needs to keep track of previous sessions that could be continued at some point in time. This results in some extra work (and also cache space) that the server has to do.

The session-ticket, in contrast, is not an identifier but the session data encrypted by the server and stored by client. When a client want to continue a session, it sends the session-ticket and the server, and only the server, can decrypt the whole session information from that.

This reduces the amount of data that server has to cache per session and also useful in load balancing scenarios.

Task 6.1

The change cipher spec message is sent by both the client and server to notify the receiving party that subsequent records will be protected under the just-negotiated CipherSpec and keys.

1. It exists to update the cipher suite to be used in the connection.
2. It permits a change in the SSL session occur without having to renegotiate the connection.

Task 6.2

Just 1 byte

Task 7.1

Yes, they differ.
They encrypt all the protocol information sent by the other party in the previous records. This allows detection of any foul play in the previous messages.

Task 8.1
Yes! It is encrypted using the CipherSpec negotiated by the 2 parties during the connection setup - RC4_128, in this case.

TLS uses MAC-then-Encrypt: Compute the MAC (SHA in this case) on the cleartext, append it to the data, and then encrypt the whole. So, the MAC and the encrypted data are indistinguishable.

<u>Problem 3.2</u>

Task 9.1
Trace protocol : SSH
Trace scenario: Connected to a previously known host over SSH and established a remote shell. No commands were actually issued however.

Trace is attached along with this submission. ARP and DNS requests for the SSH host can also be seen in the trace. Now, if we filter the trace for only SSH records, we can see the following records:

- Client: TCP Packet
- Server: TCP Packet

1. Client: Key Exchange Init
2. Server: Key Exchange Init
3. Client: Diffie-Hellman Key Exchange Init
4. Server: Diffie-Hellman Key Exchange Reply
5. Client: Diffie-Hellman GEX Init
6. Server: Diffie-Hellman GEX Reply
7. Server: New Keys
8. Client: New Keys

If we inspect the first two packets, we can notice that they are SSH version info exchange packets for client and server respectively. We shall now take a deeper look at the actual SSH protocol records.

1. **Client: Key Exchange Init**
   Several parameters are negotiated here. Such as:
   - Key Exchange Algorithms (diffie-hellman etc.,.)
   - Encryption algorithms :
     - Client - Server
     - Server - Client
   - MAC algorithms :
     - Client - Server
     - Server - Client
   - Compression algorithms
     - Client - Server
     - Server - Client

2. **Server: Key Exchange Init**
   Response to the previous request with a choice from the list

3. **Client: Diffie-Hellman Key Exchange Init**
   Negotiation of the Diffie-Hellman parameters about mathematical group (as described in section 3 of RFC4419)

4. **Server: Diffie-Hellman Key Exchange Reply**
   Reply to the above request. Since the payload is encrypted, not much information can be derived from this record in wireshark

5. **Client: Diffie-Hellman GEX Init**
   This is the first actual exchange of DH information. Client sends its public part of Diffie–Hellman key exchange. Following the Wikipedia notation on Diffie–Hellman key exchange, client sends g^a

6. **Server: Diffie-Hellman GEX Reply**
   This finished the DH exchange. Server sends its public part of the DH key. Again, following wiki notation, server sends g^b. After receiving this packet both peers know the secret key (g^{ab}) and establish a secure channel.

7. **Server: New Keys**
8. **Client: New Keys**
   These two are very small packets - both of packet length 12 and padding length 10, Mostly they are simple acknowledgment messages and doesn't contain any significative data.

All future packets over the SSH protocol are encrypted and are tagged as either *Encrypted request packet* or *Encrypted response packet* and no information can be derived from them - hopefully, fingers crossed.

The following sources, not an exhaustive list, have been referred to arrive at this answer.

- "Diffie–Hellman key exchange" - Wikipedia
  http://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange
- "How the SSH Client and Server Communicate" - HP OpenVMS Systems Documentation
  http://h71000.www7.hp.com/doc/83final/ba548_90007/ch01s04.html
- "Understanding Wireshark capture of SSH" - Stack Exchange
  http://serverfault.com/questions/586638/understand-wireshark-capture-for-ssh-key-exchange