IPsec and Transport Layer Security

Laboratory for the class "Information Systems Security" (01TYMOV, 02KRQ)
Politecnico di Torino – AA 2023/24
Prof. Diana Berbecaru

prepared by:
Diana Berbecaru (diana.berbecaru@polito.it)
Enrico Bravi (enrico.bravi@polito.it)

v. 1.0 (17/11/2023)

Contents

1 Purpose of the laboratory

In this laboratory, you will perform exercises aimed to create secure (communication) channels among two nodes, to evaluate their security features, and to measure the performance of the system when the security features are modified. In practice, the exercises use two famous security protocols: IPsec, to enable security at IP level, and TLS, to enable security at transport level.

IPsec (IP Security) is the solution proposed by the IETF (`www.ietf.org`) for securing communication in the IP networks, and it applies both to IPv4 and to IPv6. The main idea in IPsec is to provide security services at network level, so that the applications – operating at the upper level – can avoid the implementation of the security services directly into the applications. IPsec provides mainly authentication, integrity and confidentiality

of the IP packets. IPsec allows a system to define its own security policies, to select the protocols adequate for each security policy that has been chosen, to specify the cryptographic algorithms to be used, and to acquire the necessary cryptographic keys. To communicate securely via IPsec, two network nodes must share at least one Security Association (SA) in order to specify the security features of the channel. The IPsec SA can be specified manually or can be negotiated (securely) by using the IKE protocol.

The exercises proposed for IPsec make use of the IPsec implementation and the corresponding configuration tools available for Linux platform:

- `ip xfrm` is the tool use to manipulate the Security Association and Security Policy databases in IPsec (`man ip xfrm`);
- strongSwan is the daemon used for IKE (v1 and v2) protocol management and supports authentication with Pre-Shared Keys (PSK), X.509 certificates (`man ipsec`);
- the file `/etc/ipsec.conf` defines basically the behaviour of strongSwan (`man ipsec.conf`).

The second part of the laboratory is dedicated to experimenting with the TLS (Transport Layer Security) protocol for protecting data at transport level. For this purpose, we will use the OpenSSL library (and some associated tools) offering in depth support for the configuration and analysis of SSL/TLS channels. In particular, the command `s_client` implements the functionality of an SSL/TLS client (`man s_client`). The command `s_server` implements instead the functionality of a simple SSL/TLS server (`man s_server`).

The support files required for the execution of some proposed exercises are available at the teaching portal as:

`ISS_lab04_support.zip`

2 Useful commands: IPsec and security at IP level

ip xfrm

`ip xfrm` is a tool that allows to configure the Security Association and Security Policy databases in IPsec. It includes the following sub-commands, as show in the man page (`man ip xfrm`):

- `state`, to manage the Security Associations (SA) database as follows:
  - `add | update` to add or update a SA;
  - `list` to list all SAs in the SA Database (SAD);
  - `flush` to remove all SAs from the SAD;
- `policy`, to manage the Security Policies (SP) database as follows:
  - `add | update` to add or update a SP;
  - `list` to list all SPs in the SP Database (SPD);
  - `flush` to remove all SPs from the SPD;
- `monitor`, to show debugging information about the SAs and SPs.

IPsec Security Associations (ip xfrm state)

To create new security associations, you can use the following command:

```
ip xfrm state add [src IP_src ] [dst IP_dst ] [proto type_of_protection ] [ spi
SPI ] [ protection_methods ]
```

A security association is unidirectional, that is it defines which mechanism must be applied for the communication between *IP_src* and *IP_dst*. Thus, to protect the traffic originating from *IP_dst* toward *IP_src* (obtaining thus a bidirectional channel), it is necessary to define another SA.
The *type_of_protection* is one of the options available in IPsec. You must use either AH (`ah`), ESP (`esp`) or compression (`comp`) for this parameter.
The parameter *SPI* (Security Parameter Index) is an integer number (expressed in decimal or in hexadecimal with the prefix `0x`). The SPI provides an index to the SAs present in the SAD, when they are being applied. SPI values between 0 and 255 are reserved for future use by IANA and cannot be used. Typically, the last 3 hexadecimal digits are fixed (to zero), and when you need to specify the SPI values you can start with the value `0x1000`, and increment from the fourth hexadecimal digit for each new SPI, that is `0x2000`, `0x3000`, and so on.
For the *protection_methods*, you can use one of the following three options:

- `enc`, to specify the encryption algorithm (e.g. `aes`) and the secret key to be used
- `auth`, to specify the digest algorithm used for HMAC (e.g. `sha1`, `sha256`) and the secret key to be used
- `aead`, to specify the authenticated encryption with associated data algorithm, the secret key to be used, and the length of the *ICV* (Integrity Check Value)
- `comp`, to specify the compression algorithm (e.g. `deflate`, `lzs`).

The full list of supported encryption, authentication, and compression algorithms is given in the `ip xfrm state` man page.

IPsec Security Policies (ip xfrm policy)

To define the type of traffic for which a certain SA is applied, it is necessary to define a security policy (SP). To create a SP, you can use the sub-command `policy`, which has the following syntax:

```
ip xfrm policy add [ src IP_src ] [ dst IP_dst ] [ proto [ PROTOCOL ] dir
DIR [ action ACTION ] [ tmpl POLICY ]
```

The security policies are unidirectional too, because they are "selectors" of the SAs to be applied. The field *DIR* must specify either input traffic (`in`), output traffic (`out`), or packets that need to be forwarded (`fwd`).
The field *PROTOCOL* represent a valid protocol name among `tcp`, `udp`, `sctp`, `dccp`, `icmp`, `icmp-ipv6`, and `gre`. In case of the transport protocols, the `sport` and `dport` options can be used to specify the source and destination ports, respectively. In case this parameter is not specified, the SP applies to all protocols.
The field *ACTION* defines the action to be performed whenever a packet matches the policy, and it can either be allowed (`allow`, the default) or blocked (`block`).
The field *POLICY* includes the following parameters:

- `proto`: either `ah` or `esp` must be used for this parameter.
- `mode`, indicates whether IPsec transport or tunnel mode is used. If mode is `tunnel`, you must specify the end-point addresses of the SA as `src` and `dst`, which is used to specify the SA to use. If `mode` is transport, both `src` and `dst` can be omitted.
- `level`, is one of the following: `required` (default) and `use`.`required` means that a SA is required whenever the kernel sends a packet matched with the policy (the operation fails in case at least one SA is not available). `use` means that the kernel uses an SA if it's available, otherwise the kernel keeps normal operation.

strongSwan

If you don't want to or if it is inefficient to create the SAs manually, you can create them dynamically with the IKE protocol. The daemon used for IKE protocol management is named `strongSwan`, which is an opensource IPsec implementation for Linux.
The following command is used to start strongSwan:

```
ipsec start
```

The following command is used to restart strongSwan, e.g. when you need to do so after a change in its configuration:

```
ipsec restart
```

strongSwan's behaviour is largely controlled by the configuration file `/etc/ipsec.conf` in which you need to specify the directives used for the negotiation of the keys (made with the IKEv2 deamon `charon`) and for the negotiation of the SAs.
Additionally you need to use the file `/etc/ipsec.secrets`, which contains secrets like the RSA private keys or the pre-shared keys.
You also have to use the directory `/etc/ipsec.d`, which contains certificates and CRL files that are loaded by the keying daemon `charon`.
The `/etc/ipsec.conf` configuration file of strongSwan consists of three different sections types:

- `config setup` defines general configuration parameters
- `conn` <*name*> defines a connection
- `ca` <*name*> defines a certification authority

In the exercises proposed you will mainly modify the `conn` <*name*> section type. In the `conn` section type, `left` and `right` (that will have to be changed according to your configuration in the file `/etc/ipsec.conf`) denote the two endpoints of an IKE_SA: `left` means the local peer, i.e. the one on which the `ipsec.conf` config file is stored, while `right` is the remote peer.
You can easily remember this by looking at the first letter of the two terms (Left=Local, Right=Remote).
Besides the authentication and keying material, IKE also provides the means to exchange configuration information and to negotiate IPsec SAs, which are often called CHILD_SAs. IPsec SAs define which network traffic is to be secured and how it has to be encrypted and authenticated.
A CHILD_SA actually consist of two components, the actual IPsec SA describing the algorithms and keys used to encrypt and authenticate the traffic and the policies that define which network traffic shall use such an SA. The policies work both ways, that is, only traffic matching an inbound policy will be allowed after decryption.

TLS

The OpenSSL library implements a simple SSL/TLS client and server, which can be used for testing the SSL/TLS protocol very easily.
The OpenSSL command used to start an SSL/TLS client program is `s_client`, whose syntax is given below:

```
openssl s_client [-connect host:port] [-state] [-showcert] [-CAfile file_cert]
[-cipher cipherlist] [-reconnect]
```

where:

- `-connect` *host:port* specifies the host and optional port to connect to. If host and port are not specified then an attempt is made to connect to the local host on port 4433.

- `-state` prints out the SSL session states.
- `-showcerts` displays the whole server certificate chain: normally only the server certificate itself is displayed.
- `-CAfile` *file* indicates the file containing trusted certificates to use during server authentication and to use when attempting to build the client certificate chain.
- `-cipher` *cipherlist* allows to specify the cipher list sent by the client in the ClientHello message of the Handshake protocol. Although the server determines which cipher suite is used it should take the first supported cipher in the list sent by the client. See the OpenSSL `ciphers` command for more information.
- `-reconnect` allows the client to reconnect to the same server 5 times using the same session ID.

The OpenSSL command used to start an SSL/TLS server program is `s_server`, whose syntax and parameters are given below:

```
openssl s_server [-www] [-no_dhe] [-key server_pkey.pem] [-cert server_cert.pem]
[-CAfile file_cert] [-{vV}erify depth] [-cipher ciphersuite_list]
```

where:

- `-www` sends a status message back to the client when it connects. This includes lots of information about the ciphers used and various session parameters. The output is in HTML format so this option will normally be used with a web browser.
- `-no_dhe` if this option is set then no DH parameters will be loaded effectively disabling the ephemeral DH cipher suites.
- `-key` *server_pkey.pem* indicates that *server_pkey.pem* contains the private key of the server.
- `-cert` *server_cert.pem* indicates that *server_cert.pem* contains the certificate of the server.
- `-CAfile` *file* indicates the file containing trusted certificates to use during client authentication and to use when attempting to build the server certificate chain. The list is also used in the list of acceptable client CAs passed to the client when a certificate is requested.
- `-verify` *depth*, `-Verify` *depth* indicates the verify depth to use. This specifies the maximum length of the client certificate chain and makes the server request a certificate from the client. With the -verify option a certificate is requested but the client does not have to send one, with the -Verify option the client must supply a certificate or an error occurs.
- `-cipher` *cipherlist*, this option allows the cipher list used by the server to be modified. When the client sends a list of supported ciphers the first client cipher also included in the server list is used. Because the client specifies the preference order, the order of the server cipherlist irrelevant. See the OpenSSL `ciphers` command for more information.

3 IPSec and security at IP level

3.1 Setting IPsec policies and Security Associations

Form two groups of hosts, named Alice and Bob, and try to establish an IPsec channel among them.

Note: by checking the manual pages for the `ip xfrm`, identify the command need to flush and to list the content of the SAD and SPD and note down their use as it will be needed throughout the execution of the exercises.

Which commands do you need to run to flush the SAD and SPD? Which commands do you need to run to check the content of the SAD and SPD databases?

→

To create the SAs and policies among Alice and Bob, Alice executes the following instructions:

```
ip xfrm state add src IP_Alice dst IP_Bob proto esp spi 0x1000 enc aes
0xaa223344556677889900aabbccddeeff

ip xfrm state add src IP_Bob dst IP_Alice proto esp spi 0x2000 enc aes
0xbb223344556677889900aabbccddeeff

ip xfrm policy add src IP_Alice dst IP_Bob dir out tmpl proto esp mode transport
level required

ip xfrm policy add src IP_Bob dst IP_Alice dir in tmpl proto esp mode transport
level required
```

To create the SAs and policies among Alice and Bob, Bob executes the following instructions:

```
ip xfrm state add src IP_Alice dst IP_Bob proto esp spi 0x1000 enc aes
0xaa223344556677889900aabbccddeeff

ip xfrm state add src IP_Bob dst IP_Alice proto esp spi 0x2000 enc aes
0xbb223344556677889900aabbccddeeff

ip xfrm policy add src IP_Alice dst IP_Bob dir in tmpl proto esp mode transport
level required

ip xfrm policy add src IP_Bob dst IP_Alice dir out tmpl proto esp mode transport
level required
```

Respond to the following questions:

- which type of traffic is protected with IPsec?

  →

- which IPsec protocol is used? which security services are provided? Which are the cryptographic algorithms used?

  →

- what are respectively the roles of the Security Association Database (SAD) and of the Security Policy Database (SPD) in the IPsec architecture?

  →

- which directives are processed every time Alice sends a new IP packet?

→

- which directives of the policy are processed every time Alice receives a new IP packet?

→

- why are two SAs necessary?

→

Run the commands with the IP addresses of your machines (corresponding to Alice and Bob). Activate the SA and SP and verify that they have been activated correctly by listing the content of the SAD and SPD.

Now, send ping requests to your partner machine, analyse the traffic generated (for example by using the tool `wireshark`), and respond to the following questions:

- which is the purpose of the field SPI present in the IP packets exchanged?

→

- which is the scope of the field Sequence Number?

→

3.2 Modification of IPsec policies and Security Associations

Modify the SA and the SP defined previously so that to use (in this order) the following security mechanisms:

1. ESP with AES-128-CBC and HMAC-SHA1;

→

2. AH with HMAC-SHA1;

→

3. ESP with AES-128-CBC and AH with HMAC-SHA1.

> NOTE
>
> The ESP+AH mode became NOT RECOMMENDED starting from the RFC-8221, section 4 (https://www.ietf.org/rfc/rfc8221.html#section-4). It is preferable to use the solution provided at point 4 since it introduces less overhead.

→

4. ESP with AES-128-GCM (RFC-4106) Authenticated Encryption with Associated Data (AEAD).

NOTE

Note that AEAD requires the length of the Integrity Check Value (ICV) in bits along with the key. The RFC-4106 (https://tools.ietf.org/html/rfc4106) specifies that the AEAD implementation MUST support ICVs 16 octets long, and may support 8 or 12 octets as well (Section 6). Moreover, the keying material requested for AES-GCM-ESP with 128 bit key is 20 octets long: the first 16 octets are the 128-bit key, and the remaining four octets are used as the salt value in the nonce (Section 8.1)

→

NOTE

Use `man ip xfrm` to list all the cryptographic algorithms available. Please remember that parenthesis "(. . .)" must be escaped with back-slash and the keys must have the right length.

How many SAs and SPs do you need to define to implement the security mechanism 3 (above)?

→

For each of the three configurations described above, activate the modified SA, send a ping to your partner's machine, 3analyse the packets exchanged (for example by using the tool `wireshark`), and respond to the following questions:

- which is the difference among the configurations 1 and 3 in terms of the structure of the IP packets obtained? which is the difference if terms of security of the data contained in the IP packet?

  →

- when is the configuration 2 useful?

  →

- what advantage do we have using the configuration 4?

  →

Choose one of the previous configurations and modify the SPs so that to protect only the TCP protocol messages (and not the messages corresponding to the other protocols). Verify subsequently that the ping packets are not protected, while the TCP traffic remains protected (for example the HTTP traffic).

→

Finally, indicate which information is contained in the SAD and in the SPD.

→

3.3 Performance measurement

First of all, it is necessary to measure the time required for data exchange over a channel in "normal" conditions, that is without IPsec.

First of all, clear the SAD and SPD databases:

```
ip xfrm state flush
```

```
ip xfrm policy flush
```

Form two groups of hosts, let's say Alice and Bob, and proceed as follows:

1. Alice creates a new directory `/var/www/html/ipsec/` and in this directory she creates a series of files of size 10 kB, 100 kB, 1 MB, 10 MB, 100 MB, naming them respectively `10K`, `100K`, `1M`, `10M`, `100M`; to create a file of a certain size, you can use the following OpenSSL command:

   ```
   openssl rand -out name_file size_in_bytes
   ```

2. Alice starts the Apache server.

3. Bob downloads the files created by Alice (e.g. with the command `wget`), and notes the time required to transfer the files; use the command line HTTP client `curl`:

   ```
   curl --trace-time -v -o /dev/null http://IP_Alice/ipsec/nome_file
   ```

4. write down in the Table 1 the time necessary for transferring the data (i.e. transfer time), and the corresponding speed obtained for each data file downloaded (it is strongly advisable to measure the transfer time several times and to calculate and write down the average time obtained)

	10 kB	100 kB	1 MB	10 MB	100 MB
No IPsec/TLS					
time [s]					
speed [kB/s]					
ESP transport AES-128-CBC					
time [s]					
speed [kB/s]					
ESP transport AES-128-CBC HMAC-SHA1					
time [s]					
speed [kB/s]					
AH transport HMAC-SHA1					
time [s]					
speed [kB/s]					
ESP AES-128-CBC + AH HMAC-SHA1					
time [s]					
speed [kB/s]					

Table 1: Performance measurements results.

Now, execute the same performance measurements for the various IPsec configurations encountered in the previous exercises; write down the results in the Table 1, and compare them with the previous ones.

→

3.4 The IKE protocol

If you don't want to or if it is inefficient to create the SAs manually, you can create them dynamically with the IKE protocol. The daemon used for IKE protocol management is named `strongSwan`. Take a look inside section 2 for more details about using `strongSwan`.

3.4.1 strongSwan and pre-shared-keys

Download the support files available in the archive in the directories `Ali_machine` and `Bob_machine` in the directory `strongswan/IKE_with_psk(Ali-Bob)`.

Now, form two groups of hosts, Alice and Bob.

Alice performs the following steps:

- removes the entries in the SAD database with the command `ip xfrm state flush`; next she removes the entries in the SPD database with the command `ip xfrm policy flush` (in this exercise, the SAs and SPs will be created automatically by the daemon `strongSwan`);
- overwrites the content of the file `/etc/ipsec.conf` with the one in the file `ipsec.conf_psk_ali`.
- she modifies the IP addresses with those of Alice and Bob in the file `/etc/ipsec.conf`; For example, in the file provided, 192.168.27.128 is the IP address of Alice, while 192.168.27.132 is the IP address of Bob. Pay attention also at the `leftsubnet` and `rightsubnet`, they might need to be changed according to your network configuration.
- overwrites the content of the file `/etc/ipsec.secrets` with the one in the file `ipsec.secrets_psk_ali`.
- she modifies the IP addresses with those of Alice and Bob in the file `/etc/ipsec.secrets`.

Bob performs the following steps:

- removes the entries in the SAD database with the command `ip xfrm state flush`; next he removes the entries in the SPD database with the command `ip xfrm policy flush` (in this exercise, the SAs and SPs will be created automatically by the daemon `strongSwan`);
- overwrite the content of the file `/etc/ipsec.conf` with the one in the file `ipsec.conf_psk_bob`.
- he modifies the IP addresses with those of Alice and Bob in the file `/etc/ipsec.conf`; Pay attention at the order, in the file provided 192.168.27.128 is the IP address of Alice, while 192.168.27.132 is the IP address of Bob. Pay attention also at the `leftsubnet` and `rightsubnet`, they might need to be changed according to your network configuration.
- overwrites the content of the file `/etc/ipsec.secrets` with the one in the file `ipsec.secrets_psk_bob`.
- he modifies the IP addresses with those of Alice and Bob in the file `/etc/ipsec.secrets`; Pay attention at the order, in the file provided 192.168.27.128 is the IP address of Alice, while 192.168.27.132 is the IP address of Bob.

Alice starts up strongSwan with the command:

```
ipsec start
```

Bob starts up strongSwan with the command:

```
ipsec start
```

Alice starts up the connection configured in the `/etc/ipsec.conf` with the command:

```
ipsec up host-host
```

Analyze with the command `ip xfrm state list` the SA IPsec generated by the IKE daemon on Alice and Bob machines and respond at the following questions:

How many SAs have been negotiated?

→

Which kind of security header and which mode is used in the SA? ESP of AH? which algorithms are used? which is the key length used for each algorithm?

→

In which phase of the IKE protocol is used the configured PSK?

→

Analyze with the command `ip xfrm policy list` the IPsec policies generated by the IKE daemon on Alice and Bob machines.

Alice generates traffic towards Bob (for example, icmp and http traffic) and analyse it (for example with `wireshark`).

If you want to observe several times the IKE negotiation, you need to empty out SAD and SPD, or you need to restart `strongSwan` and activate the connection (with `ipsec up host-host`).

ATTENTION

To proceed with the execution of the following exercise, you need to stop `strongswan` and clear the SAD and SPD.

4 The TLS protocol

4.1 Setting up a TLS channel

Depicted in Fig. 1 is a typical instance of a TLS handshake (server authentication only).

The description of the steps is given shortly below:

1. ClientHello: Client sends the random number R_c and the supported cipher suites.
2. ServerHello: Server sends the random number R_s and the chosen cipher suite.
3. Certificate: Server sends its X.509v3 certificate (and chain)
4. ServerKeyExchange: (in case of DH key agreement) Server sends its DH public part (signed)
5. ServerHelloDone: Server signals end of handshake
6. ClientKeyExchange: (in case of DH key agreement) Client sends its DH public part
In case of RSA (key exchange): Client sends the PS encrypted with the public key of the server
Client and server derive master secret, then the keys for encryption ($K_{enc}c$ and $K_{enc}s$), the keys for calculation of MACs ($K_{mac}c$) and $K_{mac}s$ and the IVs.
7. ChangeCipherSpec, Finished: Client sends MAC($K_{mac}c$, handshake messages)
8. ChangeCipherSpec, Finished: Server sends MAC(handshake messages, $K_{mac}s$)
9. ApplicationData: Client sends data protected with the keys and algorithms negotiated
10. ApplicationData: Server sends data protected with the keys and algorithms negotiated

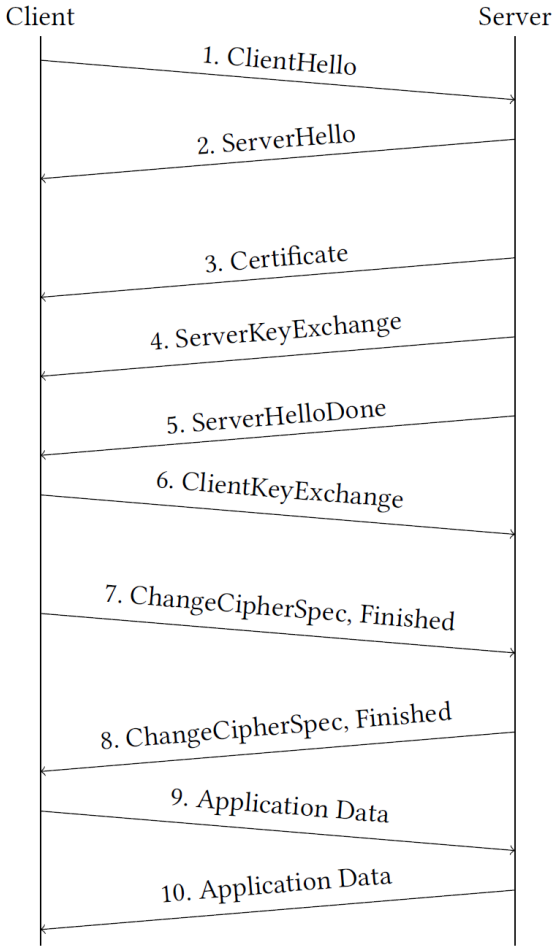

Figure 1: TLS handshake

Now, from your machine, try to connect to a TLS-enabled, by using `s_client` program (from the command line):

```
openssl s_client -connect mail.polito.it:443 -state -showcerts -verify 1
```

Start Wireshark and observe the TLS handshake messages exchanged between your client and the remote TLS server.

Which cipher suite has been negotiated?

→

Which TLS version has been negotiated?

→

Now run the following command:

```
openssl s_client -connect fancyssl.hboeck.de:443 -state -showcerts -verify 1
```

Intercept with Wireshark the TLS handshake messages exchanged between your machine and the TLS server

in the above command.

Which TLS version has been negotiated?

→

Illustrate it with the caption (Wireshark) in your screen.

Which cipher suite has been negotiated?

→

Which algorithm has been used for key exchange?

→

4.2 Configuration of a TLS server

Try now to configure a TLS server. What do you need in the first place?

You can use the following OpenSSL commands to generate a certificate for the TLS server:

1. create first a test CA by exploiting OpenSSL:

   ```
   /usr/lib/ssl/misc/CA.pl -newca
   ```

   When asked, insert a password to protect the private key, e.g. "ciao". Remember the values you have inserted for "Country", "State", and "Organization", as you will have to use the same values in the following steps.

2. create a certificate request for the TLS server:

   ```
   openssl req -new -keyout server_pkey.pem -out server_creq.pem
   ```

3. issue the new certificate for the TLS server:

   ```
   openssl ca -in server_creq.pem -out server_cert.pem
   ```

At this point you should have: the certificate of the CA (in the `cacert.pem` in the `demoCA` directory), and the certificate and the corresponding key for the TLS server (in the files `server_cert.pem`, `server_pkey.pem`).

Start the `s_server` with the following command:

```
openssl s_server -www -no_dhe -key server_pkey.pem -cert server_cert.pem
```

The server listens on the port `4433`. Try to connect with `s_client` and check out the result:

```
openssl s_client -connect 127.0.0.1:4433 -state -showcerts -CAfile
demoCA/cacert.pem
```

Which is the purpose of the parameter `-no_dhe` used so far?

→

What are and in which case can be used the ephemeral mechanisms for key exchange in TLS?

→

4.3 Use of Session ID

For what purpose it is used the "Session ID" in TLS?

→

Start the server with the following command:

```
openssl s_server -www -no_dhe -key server_pkey.pem -cert server_cert.pem -CAfile
demoCA/cacert.pem -tls1_2
```

Execute the following `s_client` command, which opens up several consecutive connections by exploiting the Session ID:

```
openssl s_client -connect 127.0.0.1:4433 -state -CAfile cacert.pem -reconnect
```

Which parameters of the TLS session remain unchanged in successive connections?

4.4 Client authentication in TLS

Now try to configure an SSL/TLS server with client authentication. What do you need in the first place?

→

You can generate a client certificate in the following way:

1. create a certificate request for the client certificate:

   ```
   openssl req -new -keyout client_pkey.pem -out client_creq.pem
   ```

2. issue a new certificate:

   ```
   openssl ca -in client_creq.pem -out client_cert.pem
   ```

Configure now `s_server` so that to request client authentication during the handshake phase of the SSL/TLS protocol:

```
openssl s_server -www -no_dhe -key server_pkey.pem -cert server_cert.pem -CAfile
demoCA/cacert.pem -Verify 1
```

Run the `s_client` command necessary to connect to the `s_server` started above:

```
openssl s_client -connect 127.0.0.1:4433 -state -showcerts -CAfile cacert.pem
-cert client_cert.pem -key client_pkey.pem
```

Identify the steps that have changed in the handshake phase of the SSL/TLS protocol.

→