IPsec and Transport Layer Security

Laboratory for the class “Information Systems Security” (02TYMUV)
Politecnico di Torino – AA 2024/25
Prof. Antonio Lioy

prepared by:
Flavio Ciravegna (flavio.ciravegna@polito.it)
Andrea Atzeni (andrea.atzeni@polito.it)

v. 1.10 (13/12/2024)

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_lab5_support.zip

1.1 Additional commands

The exercises will require to exchange messages between two computers. Consequently, you will have to start the `ssh` server:

```
systemctl start ssh
```

To copy a file (e.g. `prova`) from the `ssh` client machine into the `/home/kali` directory on the machine running `ssh` server, you can use the command:

```
scp prova kali@IP_address_ssh_server:/home/kali
```

Insert the password ‘kali’ when asked.

The exercises will require also to use the Apache web server. To start the Apache server use the command:

```
systemctl start apache2
```

To run the exercises, you may have to generate also X.509 certificates corresponding to the server and to the client (in this case we provided both certificates in the lab material).

1.2 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.

2 IPSec and security at IP level

2.1 Setting IPsec policies and Security Associations

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

To do that, it is necessary to configure two databases, the SAD and the SPD. In general, what information do they contain? What is their role in the IPsec protocol?

→

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

NOTE

Refer to the manual pages for the `ip xfrm` command to understand its functionality. Then, identify the specific commands needed to flush and list the SAD and SPD contents. Please take note of these commands and their purpose as they will be required throughout the exercises.

→

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
0xaa22334455667788990 0aabbccddeeff

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
```

Answer to the following questions:

1. Which part of the IP packet is protected by the IPsec protocol?

   $\rightarrow$

2. Which IPsec header (AH or ESP) and mode (transport or tunnel) is used?

   What security services are provided by this IPsec header?

   What cryptographic algorithms are used?

   $\rightarrow$

3. Which IPsec directives (policies and SAs) are processed every time Alice sends a new IP packet?

   $\rightarrow$

4. Which IPsec directives (policies and SAs) are processed every time Alice receives a new IP packet?

   $\rightarrow$

5. Why are two SAs necessary?

   $\rightarrow$

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

1. What is the purpose of the SPI field in the exchanged IP packets?

   →

2. Which is the purpose of the Sequence Number field?

   →

2.2 Modification of IPsec policies and Security Associations

Modify the SAs and the policies previously defined according to the following specifications:

NOTE

Every time you modify the configuration, remember to flush the content of SPD and SAD from the previous exercises

1. Protection of IP packets by using ESP with AES-128-CBC and HMAC-SHA1.

   →

2. Protection of IP packets by using AH with HMAC-SHA1.

   →

3. Protection of IP packets by using 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. Protection of IP packets by using 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 a backslash and the keys must have the right length.

How many SAs and policies are needed to implement configuration 3 (above)?

→

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

- What is the difference among configurations 1 and 3 in terms of the structure of the IP packets obtained? What is the difference in terms of the 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 to protect only the TCP protocol messages (and not the messages corresponding to the other protocols). Afterwards, verify that the ping packets are not protected while the TCP traffic remains protected (for instance, the HTTP traffic).

→

2.3 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 folder) 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 and notes the time required to transfer the files; use the command line HTTP client `curl`:

```
curl --trace-time -w "Total time:  %{time_total}s\n" -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

NOTE

It is strongly advisable to measure the transfer time several times for each file and then report the computed average time

	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]					
TLS RSA with AES-128-CBC and SHA1, server auth only					
time [s]					
speed [kB/s]					
TLS RSA with AES-128-CBC and SHA1 with client auth					
time [s]					
speed [kB/s]					

Table 1: Performance measurements results.

Now, perform the same performance measurements for the various IPsec configurations encountered in the previous exercises; write down the results in Table 1, and finally compare them with the previous ones. Which configuration has the best performance?

→

2.4 The IKE protocol

If you don't want to, or it's inefficient to specify the SAs manually, you can create them dynamically using the IKE protocol. The daemon used for IKE protocol management is named `strongSwan`. Take a look at section 1.2 for more details about using `strongSwan`.

2.4.1 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:

1. 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`);
2. overwrites the content of the file `/etc/ipsec.conf` with the one in the file `ipsec.conf_psk_ali`.
3. 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 to `leftsubnet` and `rightsubnet`, they might need to be changed according to your network configuration.
4. overwrites the content of the file `/etc/ipsec.secrets` with the one in the file `ipsec.secrets_psk_ali`.
5. modifies the IP addresses with those of Alice and Bob in the file `/etc/ipsec.secrets`.

Bob performs the following steps:

1. 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`);
2. overwrites the content of `/etc/ipsec.conf` with the one in the file `ipsec.conf_psk_bob`.
3. modifies the IP addresses with those of Alice and Bob in the file `/etc/ipsec.conf`; Pay attention to 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 to `leftsubnet` and `rightsubnet`, they might need to be changed according to your network configuration.
4. overwrites the content of the file `/etc/ipsec.secrets` with the one in the file `ipsec.secrets_psk_bob`.
5. he modifies the IP addresses with those of Alice and Bob in the file `/etc/ipsec.secrets`; Pay attention to 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
```

Analyse with the command `ip xfrm state list` the SAs IPsec generated by the IKE daemon on Alice and Bob machines and answer to the following questions:

How many SAs have been negotiated?

→

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

→

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

→

Inspect 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 analyses it (for example, with `wireshark`).

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

BEWARE

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

3 The TLS protocol

3.1 Setting up a TLS channel

Try to connect (with your browser) to an SSL/TLS server, for instance `https://mail.polito.it`

Inspect the content of the X.509 certificate sent by the server: which fields are used for the server identification? What is the certification path? What algorithms are used to protect the transmitted data?

→

Now, from your machine, try to connect to the same server as above using the `s_client` program (from the command line) instead:

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

Have you specified all the data required to establish a correct TLS connection? How do you explain that the connection was established correctly anyway?

→

Do you manage to identify the sequence of operations executed by `s_client`? In particular, which TLS version and cipher suite have been negotiated?

→

What is the meaning of the fields in `SSL-Session`?

→

3.2 Configuration of a TLS server

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

→

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

1. create a test CA:
   ```
   /usr/lib/ssl/misc/CA.pl -newca
   ```
   When asked, insert a password to protect the private key (e.g., “hello”). Remember the values you inserted for “Country”, “State”, and “Organization” since they are needed 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 (`cacert.pem` in the `demoCA` directory)
- the certificate and the corresponding key for the TLS server (`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
```

What is the purpose of the `-no_dhe` flag?

→

The server now listens on port `4433`. From another terminal on the same machine, 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
```

Try now to connect with a browser: with the parameter `-www`, `s_server` returns a web page containing useful information about the session that has just been created.

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

→

3.3 Use of Session ID

For what purpose is the "Session ID" used 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 demoCA/cacert.pem
-reconnect
```

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

→

3.4 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` to request the 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
```

Write down the `s_client` command necessary to connect to the `s_server` started above:

→

Identify the steps that have changed in the handshake phase of the SSL/TLS protocol. Why does the server send the CA certificate too?

→

Some SSL-enabled servers (for example Microsoft IIS) have the following behaviour when the client authentication is enabled/required for an SSL connection: the SSL/TLS handshake is performed but the server does not send any request for client authentication; once the above SSL handshake has been concluded with server authentication only, the client initiates a second SSL/TLS handshake which is performed instead of the client authentication. The security parameters in the second handshake are resumed with the security parameters negotiated in the first handshake. What are the advantages of this solution?

→

3.5 MITM attack to SSL/TLS

In this exercise, you will use the tools indicated in the first laboratory, specifically in the exercise related to the man-in-the-middle attack. Form groups of three hosts: Alice, Bob, and Chuck. Alice will behave as the web server, Bob as the client, and Chuck as the attacker. The scope is to perform a security attack so that Chuck can sniff the traffic between Alice and Bob.

Now, try to implement the attack using `ettercap`.

1. Chuck copies the file `etter.conf` in the file `/etc/ettercap/etter.conf`, and launches the MITM attack, with the command:
   ```
   ettercap -Tq -M arp /IPaddress_Alice// /IPaddress_Bob//
   ```
2. Bob connects with a browser to the web server of Alice

Can you figure out how this attack works?

→

Suggestion: save the certificate sent by the server and compare it with Alice's server certificate configured in the exercise 3.2.

3.6 Performance measurement

The purpose of this exercise is to measure the times required to transfer data over a channel protected with SSL/TLS. Use the table 1 for this purpose. Form two groups of hosts, Alice and Bob, and proceed as follows.

1. Alice activates the Apache web server, with server authentication:
   - uses the command `a2enmod ssl` to enable the SSL module of Apache;
   - uses the command `a2ensite default-ssl` to enable the Apache SSL site;
   - copies the files `server_cert.pem`, `server_pkey.pem`, and `demoCA/cacert.pem` into the (default) directory for the Apache SSL configuration, which is:
     – `/etc/ssl/certs` for `server_cert.pem` and `demoCA/cacert.pem`
     – `/etc/ssl/private` for `server_pkey.pem`

- updates the content of the file `/etc/apache2/sites-enabled/default-ssl.conf` with the one in the file named `apache2-ssl`, which is provided in the support material for this laboratory;
- to set the use of RSA with AES, replaces in the file `/etc/apache2/mods-available/ssl.conf` the directive `SSLCipherSuite HIGH:MEDIUM:!ADH:!MD5` with `SSLCipherSuite AES:SHA1:!ADH:!MD5`;
- restarts the Apache web server, by using the command:

  ```
  systemctl restart apache2
  ```

  .
- for testing purposes, she tries to connect to (her own) Apache web server with the command:

  ```
  openssl s_client -connect 127.0.0.1:443
  ```

2. Bob obtains the `cacert.pem` from Alice and adds to `/etc/hosts` the value he inserted in the Common Name of the TLS server certificate:

   ```
   <IPaddress_Alice> CN_Alice
   ```

3. Bob downloads the files `10k`, `100k`, `1M`, `10M` and `100M` with the following command:

   ```
   curl -w "Total time:  %{time_total}s\n" --cacert cacert.pem -o /dev/null
   https://IP_Alice/ipsec/file_name
   ```

   and writes down the times obtained in Table 1.

4. Alice now enables the client authentication in TLS, by changing in the file `/etc/apache2/sites-enabled/default-ssl.conf` the directive:

   ```
   SSLVerifyClient none
   ```

   with:

   ```
   SSLVerifyClient require
   ```

   and restarts the Apache web server (`systemctl restart apache2`)

5. Bob now retrieves the `client_pkey.pem` and `client_cert.pem` (he has already obtained `cacert.pem` at step 2). At this point it decrypts the client key with the command:

   ```
   openssl rsa -in client_pkey.pem > client_decrypted_pkey.pem
   ```

6. Bob downloads the files as above with the command:

   ```
   curl -w "Total time:  %{time_total}s\n" --cert client_cert.pem
   --key client_decrypted_pkey.pem --cacert cacert.pem -o /dev/null
   https://IP_Alice/ipsec/file_name
   ```

   and reports the results in Table 1

4 Additional exercises (optional)

This exercise shows the how to configure an application gateway by using Apache web server and a dedicated module named `mod_proxy`. You will have to issue also (and configure) a corresponding certificate in order to connect with HTTPS to the Apache web server.

NOTE

To perform this exercise you need to set up the three groups of hosts that you have seen in the previous laboratory: Alice, Bob and Frank (acting as firewall). You have to configure Alice and Bob to redirect through Frank the traffic exchanged (check the instructions in the previous lab).

TUTORIAL: Application level gateway

4.1 Configuration of mod_proxy as application level gateway

In this exercise, Alice will use an application level gateway in order to get access to the web server running on Bob's host. In particular, we'll use the Apache web server and the module `mod_proxy` to experiment the functionality of an application level web gateway.
Frank performs the following operations to activate and configure the application level gateway:

1. stops the Apache server if active (with the command `systemctl start apache2`).
2. replaces the file `/etc/apache2/sites-enabled/000-default.conf` with the file `frank-apache2-config` provided in the archive.
3. now you need the key and the certificate for the TLS server.

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

   (a) create 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.

   (b) create a certificate request for the TLS server:

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

   (c) 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`).
4. creates the directory `/etc/apache2/ssl` and copy in it the certificates (of the CA and of the server) and the private key
5. loads the Apache modules required to use SSL/TLS

   ```
   a2enmod ssl
   ```
6. loads the Apache modules required for the proxy functionality

   ```
   a2enmod proxy
   ```

```
a2enmod proxy_http
```

7. restarts the Apache web server (with the command `systemctl restart apache2`)

Bob configures his web server:

- replace the `/etc/apache2/sites-enabled/000-default.conf` file with this one `bob-apache2-config` provided in the lab material;
- restart the Apache server `systemctl restart apache2`

Now Alice tries to connect to the web server of Bob, by using the application level gateway of Frank:

- configures the textual browser `lynx` to use Frank as proxy HTTPS, by setting the environment variable `http_proxy`:

  ```
  export http_proxy='https://IP_Fra:443/'
  ```

- connects to the web server of Bob (you can ignore the problem of CA not recognized by the browser by pressing 'y' when asked):

  ```
  lynx http://IP_Bob
  ```

Now modify the authorisation policy of the application gateway of Frank:

- edit the configuration file of the Apache web server, and uncomment the section "Proxy Filter".
- restart the Apache web server.

Verify what happens when connecting with the browser running on Alice's host to the web server running on Bob

```
lynx http://IP_Bob/icons/
```

The rule blocks the access to resources of type *.gif. Thus, by trying to download a gif it is signaled that Alice is not authorized.
Finally, we can protect the services offered by the application gateway of Frank through client authentication:

- edit the configuration file of the Apache web server and uncomment the section "Proxy Auth"
- you must also create a file with the username and password for Apache:

  ```
  htpasswd -c /etc/apache2/htpasswd ali
  ```

- restart the Apache web server.

As above, Alice connects to the web server running on Bob's host with the browser `lynx`:
Now, `lynx` will ask the authentication password of Alice. Which advantages have been obtained when using an application gateway instead of a packet filter?
With this solution we obtain the same advantages as the ones based on the circuit-level gateway, and additionally we have the possibility to set application filters (for example to filter content and vulnerabilities specific to applications).