

POLITECNICO DI TORINO

Virtual Private Networks

Daniele Brighenti



April 10, 2025

License

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

You are free:

- **to Share:** to copy, distribute and transmit the work
- **to Remix:** to adapt the work

Under the following conditions:

- **Attribution:** you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **Noncommercial:** you may not use this work for commercial purposes.
- **Share Alike:** if you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

More information on the Creative Commons website.



Acknowledgments

The author would like to thank all the people who contributed to this document.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 4 |
| 2 | Initial network configuration | 5 |
| 2.1 | Creation and configuration of the Linux namespaces | 6 |
| 3 | GRE | 9 |
| 3.1 | Configuration of the GRE-based VPN | 9 |
| 3.2 | Verification and analysis of the GRE-based VPN configuration | 10 |
| 3.3 | Cleaning the environment in preparation of the next exercise | 14 |
| 4 | IPSec | 15 |
| 4.1 | Configuration of the IPSec-based VPN with AH | 16 |
| 4.2 | Verification and analysis of the AH IPSec-based VPN configuration | 18 |
| 4.3 | Configuration of the IPSec-based VPN with ESP | 22 |
| 4.4 | Verification and analysis of the ESP IPSec-based VPN configuration | 23 |
| 4.5 | Cleaning the environment in preparation of the next exercise | 26 |
| 5 | TLS | 27 |
| 5.1 | Configuration of the TLS-based VPN | 27 |
| 5.1.1 | Running the TLS server and client | 32 |
| 5.2 | Verification and analysis of the TLS-based VPN configuration | 32 |

1 Introduction

This lab aims at practicing with Virtual Private Networks (VPNs). VPNs are a network security solution that provides connectivity on a shared infrastructure such that some properties are enforced in that infrastructure, as if they were enforced in a private network. Example of those properties are Quality of Service (QoS), reliability, addressing, but also security. For what concerns security, a VPN can provide security features such as data confidentiality, data authentication, data integrity, and peer authentication.

You will practice with three protocols that can be used for the implementation of the VPN capability: GRE, IPSec, and TLS. You will be thus able to compare the different properties each solution can provide. You will also analyze how those solutions modify the packets crossing a VPN differently, by capturing the packets on different network interfaces through a traffic analyzer.

All the exercises about VPNs will be carried out by using a Virtual Machine on Crownlabs (<https://crownlabs.polito.it/>), running the Kali Linux OS. However, as multiple entities need to interact to exchange packets, a preliminary network configuration based on the creation of network namespaces must be performed before starting to practice with VPNs themselves.

2 Initial network configuration

Namespaces are a feature of the Linux kernel that partition kernel resources such that one set of processes sees one set of resources, while another set of processes sees a different set of resources. The feature works by having the same namespace for a set of resources and processes, but those namespaces refer to distinct resources. Resources may exist in multiple spaces, including networking. From this point of view, a network namespace isolates the system's physical network from the virtual network namespace within a single system. Each network namespace has its own interfaces, routing tables, forwarding rules, etc. Processes can thus be launched and dedicated to one network namespace.

For this activity, you will use network namespaces to simulate a network composed of four main elements. Specifically, the network you are going to simulate is depicted in Figure 2.1. There are two remote sub-networks, identified by the “end point” visual element, which are branches of the same private corporate network. Each one of these two sub-networks are connected to router working as a VPN gateway. Then, the two VPN gateways can communicate over a shared untrusted network infrastructure (e.g., the Internet). The network interfaces of the end points representing the two sub-networks and the two VPN gateways are configured with the IP and MAC addresses listed in Table 2.1.

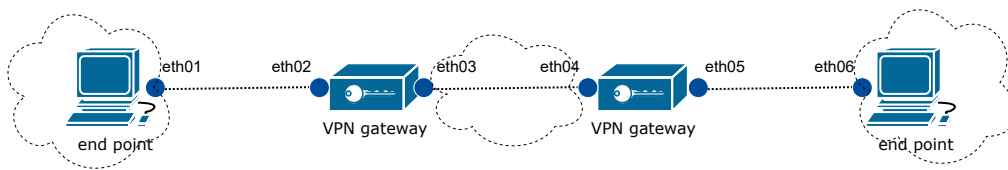


Figure 2.1: Simulated topology.

Table 2.1: Configuration of the network interfaces

| Network interface | IP address | MAC address |
|-------------------|----------------|-------------------|
| eth01 | 192.168.0.2/24 | 00:00:00:00:00:01 |
| eth02 | 192.168.0.1/24 | 00:00:00:00:00:02 |
| eth03 | 10.200.0.2/24 | 00:00:00:00:00:03 |
| eth04 | 10.200.0.1/24 | 00:00:00:00:00:04 |
| eth05 | 192.168.2.2/24 | 00:00:00:00:00:05 |
| eth06 | 192.168.2.1/24 | 00:00:00:00:00:06 |

In this preliminary activity, your task will be to create this network with namespaces in Kali Linux. Be aware that, every time a Kali Linux VM is rebooted, the namespaces are deleted, and the operation must be repeated.

2.1 Creation and configuration of the Linux namespaces

Simulating the network depicted in Figure 2.1 with namespaces, the actual network created in Linux will be the one depicted in Figure 2.2. A Linux bridge is introduced with the objectives to interconnect all virtual Ethernet interfaces of the four namespaces, and to allow you to capture network traffic in an easier way in the next exercises.

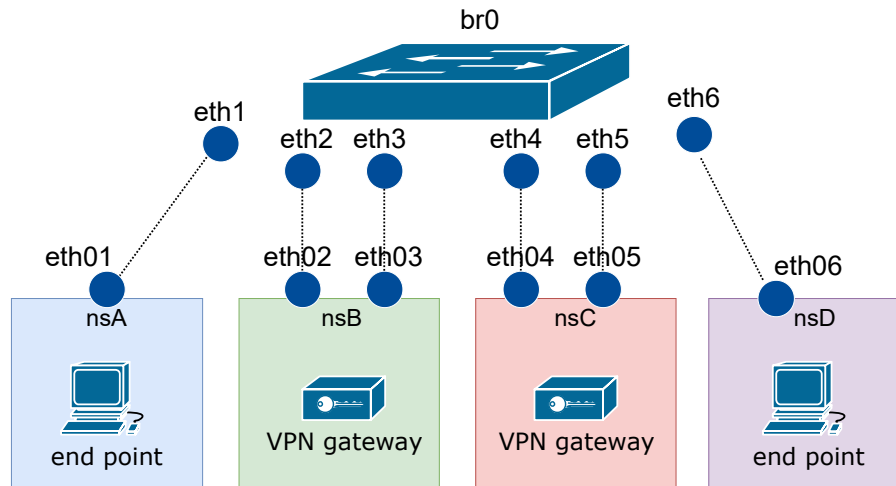


Figure 2.2: Network topology with namespaces.

In greater detail, the network created with Linux namespaces must have the following characteristics:

- the network consists of four network namespaces representing the four different elements of the simulated network topology: *nsA*, *nsB*, *nsC*, and *nsD*;
- the namespaces *nsA* and *nsD* are related to the end points of the simulated network, the namespaces *nsB* and *nsC* are related to its VPN gateways;
- the namespaces are connected to each other via a Linux Bridge, called *bridge*;
- the namespaces *nsA* and *nsD* have a single interface each, whereas the namespaces *nsB* and *nsC* have two interfaces each, in order to represent how the simulated devices are interconnected in the simulated network;
- the namespace interfaces are configured with the IP and MAC addresses listed in Table 2.1;
- the Linux Bridge has six interfaces: *eth1*, *eth2*, *eth3*, *eth4*, *eth5*, and *eth6*;
- the network elements are connected to each other as shown in Figure 2.2.

Before proceeding actual network configuration, it is necessary to disable the feature of sending ICMP Redirect packets, and instead to enable the feature of packet forwarding by executing the following commands:

```
sudo bash
sudo echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects
sudo echo 1 > /proc/sys/net/ipv4/ip_forward
exit
```

In case you forget to execute these commands *before* creating the namespaces, you will be forced to delete the namespaces and recreate them.

After this preliminary operation, you can now create the Linux bridge and the namespaces. For the Linux bridge creation, you can use the following commands:

```
sudo apt-get install bridge-utils -y
sudo brctl addbr bridge
sudo brctl stp bridge off
sudo ip link set dev bridge up
```

Instead, for the creation of the four namespaces, you can use the following commands:

```
sudo ip netns add nsA
sudo ip netns add nsB
sudo ip netns add nsC
sudo ip netns add nsD
```

Next, you have to create all the pairs of network interface `eth0 i -eth i` , with i going from 1 to 6, to properly interconnect them and to active them. In the following, the commands for the pair `eth01-eth1` will be explained as a representative example.

First, you must create the pair of network interfaces with a single command, so as to establish their interconnection since their creation itself:

```
sudo ip link add eth01 type veth peer name eth1
```

After creating the interfaces, you must plug them on the specific network element. The interface `eth1` must be plugged on the Linux bridge:

```
sudo brctl addif bridge eth1
```

Instead, the interface `eth01` must be plugged on the namespace `nsA`:

```
sudo ip link set eth01 netns nsA
```

The interface `eth01` must also be configured with the proper MAC and IP addresses, through the `ifconfig` and `ip addr` commands:

```
sudo ip netns exec nsA ifconfig eth01 hw ether 00:00:00:00:00:01
sudo ip netns exec nsA ip addr add 192.168.0.2/24 dev eth01
```

Finally, you can activate both interfaces:

```
sudo ip netns exec nsA ip link set dev eth01 up
sudo ip link set dev eth1 up
```

You will have to repeat these steps for the other five pairs of virtual interfaces.

At this point, some other general configuration settings are still required.

The loopback interface must be activated on the four created namespaces:

```

sudo ip netns exec nsA ip link set lo up
sudo ip netns exec nsB ip link set lo up
sudo ip netns exec nsC ip link set lo up
sudo ip netns exec nsD ip link set lo up

```

The feature of receiving ICMP redirect packets must be disabled for each namespace, with a specific directive in the INPUT chain of iptables:

```

sudo ip netns exec nsA iptables -A INPUT -p icmp --icmp-type redirect -j DROP
sudo ip netns exec nsB iptables -A INPUT -p icmp --icmp-type redirect -j DROP
sudo ip netns exec nsC iptables -A INPUT -p icmp --icmp-type redirect -j DROP
sudo ip netns exec nsD iptables -A INPUT -p icmp --icmp-type redirect -j DROP

```

The feature of packet forwarding must be enabled in the namespaces *nsB* and *nsC* as they represent VPN gateways that must be able to forward a packet received on a interface to another interface:

```

sudo ip netns exec nsB sysctl -w net.ipv4.ip_forward=1
sudo ip netns exec nsC sysctl -w net.ipv4.ip_forward=1

```

Static routes must be added for namespaces *nsA* and *nsD* so that they can reach IP networks to which they are not directly connected. These networks will be reached by them through the interconnected interface of the VPN gateway located in their same IP network. These static routes can be added with the following commands:

```

sudo ip netns exec nsD ip route add 192.168.1.0/24 via 192.168.2.2 dev eth06
sudo ip netns exec nsD ip route add 192.168.0.0/24 via 192.168.2.2 dev eth06
sudo ip netns exec nsA ip route add 192.168.1.0/24 via 192.168.0.1 dev eth01
sudo ip netns exec nsA ip route add 192.168.2.0/24 via 192.168.0.1 dev eth01

```

For completeness, as side material of this lab activity, we provide you with a file listing all the commands reported in this section, so that you can use it as a reference to complete the network configuration activity.

Note: at this stage, not all namespaces are able to communicate with each other. For example, namespaces *nsA* and *nsD* cannot communicate. Their communication will be enabled by the commands that will be presented in the remainder of the document.

3 GRE

Generic Routing Encapsulation (GRE) is a protocol, defined by RFC 2784, that encapsulates packets in order to route any protocols (including the IP protocol itself) over IP networks. In a packet, the presence of a GRE header is identified by the IP Protocol number 47.

In essence, GRE creates a private point-to-point connection like that of a VPN. GRE creates a private way for packets to travel through an otherwise public network by encapsulating or tunnelling the packets. Tunnel endpoints that encapsulate or de-encapsulate the traffic are used in GRE tunnelling. GRE tunnels are often set up between two routers, with each router acting as the tunnel's end. The routers are configured to send and receive GRE packets directly. Within an outer IP packet, GRE encapsulates a payload, an inner packet that must be transferred to a target network. GRE tunnel endpoints route encapsulated packets via intervening IP networks to convey payloads across GRE tunnels.

3.1 Configuration of the GRE-based VPN

In this activity, the GRE tunnel representing the GRE-based VPN will be created between the two namespaces acting as VPN gateways, i.e., between the namespaces *nsB* and *nsC*. To do so, you have to create two *tunnel* interfaces, one for each namespace of interest.

Starting from the namespace *nsB*, you can use the following command to create a tunnel interface named *gre1* (be aware that the following command is a single one; in this document, it is reported on two lines for sake of visualization):

```
sudo ip netns exec nsB ip tunnel add gre1 mode gre remote 10.200.0.1
    local 10.200.0.2
```

The meaning of the different components of this command is as follows:

- `ip tunnel add gre1`: this instructs the system to add a new tunnel interface named *gre1*;
- `mode gre`: this specifies that the type of tunnel being created is GRE;
- `remote 10.200.0.1`: this specifies the remote end point of the tunnel. In this case, it is the IP address 10.200.0.1, i.e., the IP address of the interface *eth04* of the namespace *nsC*;
- `local 10.200.0.2`: this specifies the local end point of the tunnel. In this case, it is the IP address 10.200.0.2, i.e., the IP address of the interface *eth03* of the namespace *nsB*.

After creating the tunnel interface, you can activate it, so as to allow it to send and receive packets:

```
sudo ip netns exec nsB ip link set gre1 up
```

The same commands must be executed for the namespace *nsC*, exchanging the IP addresses related to the remote and local tunnel end points:

```
sudo ip netns exec nsC ip tunnel add gre1 mode gre remote 10.200.0.2
local 10.200.0.1
sudo ip netns exec nsC ip link set gre1 up
```

These tunnel interfaces must be assigned with new IP addresses, so that they are given an identity in the network, and so as to allow routing tables to make decisions about how to forward packets destined for the tunnel end points. The IP address assignment can be performed with the following commands:

```
sudo ip netns exec nsB ip addr add 192.168.1.2/24 dev gre1
sudo ip netns exec nsC ip addr add 192.168.1.1/24 dev gre1
```

As a natural consequence, some static routes must be added to the routing tables of the two namespaces.

On the one hand, the namespace *nsB* must reach the other tunnel end point, represented by the IP address 192.168.1.1/32, through the tunnel interface *gre1*. On the other hand, the namespace *nsC* must reach the other tunnel end point, represented by the IP address 192.168.1.2/32, through the tunnel interface *gre1*. These two static routes can be added with the following commands:

```
sudo ip netns exec nsB ip route add 192.168.1.1/32 dev gre1
sudo ip netns exec nsC ip route add 192.168.1.2/32 dev gre1
```

Finally, the default route should be included for the two namespaces: 192.168.1.1 for the namespace *nsC*, 192.168.1.2 for the namespace *nsB*. The default routes are added as follows:

```
sudo ip netns exec nsB ip route add default via 192.168.1.1
sudo ip netns exec nsC ip route add default via 192.168.1.2
```

This step concludes the configuration of a GRE tunnel that should allow the namespaces *nsA* and *nsD* to communicate through the public infrastructure that is in between *nsB* and *nsC*. In particular, when crossing that network area, their exchanged packets should be tunneled, so an external IP header with the IP addresses of the two gateways and a GRE header should be added.

3.2 Verification and analysis of the GRE-based VPN configuration

Now you will verify the correct behavior of the configured GRE tunnel by making the namespace *nsA* ping the namespace *nsD*. At the same time, you are strongly recommended to analyze how the packets appear (i.e., their structure in terms of headers, and the values assigned to the header fields), capturing packets on all the six virtual interfaces of the Linux bridge: *eth1*, *eth2*, *eth3*, *eth4*, *eth5*, and *eth6*. To do so, you should open an instance of the Wireshark packet sniffer on each bridge interface.

The ICMP Echo Requests and Replies are then generated with the following command:

```
sudo ip netns exec nsA ping 192.168.2.1 -c 5
```

```

Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
Internet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.2.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x0c55 (3157)
  010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: ICMP (1)
    Header Checksum: 0xab00 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.0.2
    Destination Address: 192.168.2.1
  Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x896a [correct]
    [Checksum Status: Good]
    Identifier (BE): 15322 (0x3bda)
    Identifier (LE): 55867 (0xda3b)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
    [Response frame: 3]
    Timestamp from icmp data: Mar 12, 2024 04:57:18.000000000 EDT
    [Timestamp from icmp data (relative): 0.288927868 seconds]
  Data (48 bytes)

```

Figure 3.1: GRE Capture.

Here, for simplicity, we will just consider an ICMP Echo Request captured on three representative interfaces: *eth1*, *eth3*, and *eth6*. However, as previously mentioned, you are invited to complete the exercise analyzing also the ICMP Echo Requests captured on the other three interfaces, and the ICMP Echo Replies on all six interfaces, so as to complete the comparative analysis.

Figure 3.1 shows the Wireshark capture of an ICMP Echo Request captured on the bridge interface *eth1* (i.e., the one linked to the interface *eth01* of the namespace *nsA*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes 00:00:00:00:00:01 as source MAC address and 00:00:00:00:00:02 as destination MAC address, because the packet is sent by the interface *eth01* of the namespace *nsA* to the interface *eth02* of the namespace *nsB*. In fact, the namespace *nsA* does not have a way to contact the namespace *nsD* directly, but its packets must pass through the gateway represented by the namespace *nsB*.
- There is only one IP header, which includes 192.168.0.2 as source IP address and 192.168.2.1 as destination IP address. They are the IP address of the interface *eth01* of the namespace *nsA* and the IP address of the interface *eth06* of the namespace *nsD*, respectively. The presence of these IP addresses is expected, in compliance with the way IP networking behaves.
- There is no GRE header or a second IP header, because the ICMP Echo Request has not yet reached the GRE tunnel.
- The packet data is plain, and can be read by any intermediate node.

After the ICMP Echo Request reaches the interface *eth02* of the *nsB*, this namespace sends it towards the namespace *nsC*. In particular, according to the previous configuration of the tunnel interfaces, the GRE tunnel will be used to forward this packet.

Figure 3.2 shows the Wireshark capture of the ICMP Echo Request captured on the bridge interface *eth3* (i.e., the one linked to the interface *eth03* of the namespace *nsB*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

```

* Frame 2: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface eth3, id 0
* Ethernet II, Src: 00:00:00:00:00:03 (00:00:00:00:00:03), Dst: 00:00:00:00:00:04 (00:00:00:00:00:04)
* Internet Protocol Version 4, Src: 10.200.0.2, Dst: 10.200.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  * Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 108
    Identification: 0x0605 (1541)
  * 010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 63
    Protocol: Generic Routing Encapsulation (47)
    Header Checksum: 0x1fcc [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.200.0.2
    Destination Address: 10.200.0.1
  * Generic Routing Encapsulation (IP)
    * Flags and Version: 0x0000
    Protocol Type: IP (0x0800)
  * Internet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.2.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    * Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 84
      Identification: 0x0c55 (3157)
    * 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
      Time to Live: 63
      Protocol: ICMP (1)
      Header Checksum: 0xac00 [validation disabled]
      [Header checksum status: Unverified]
      Source Address: 192.168.0.2
      Destination Address: 192.168.2.1
  * Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x896a [correct]
    [Checksum Status: Good]
    Identifier (BE): 15322 (0x3bda)
    Identifier (LE): 55867 (0xda3b)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
    [Response frame: 3]
    Timestamp from icmp data: Mar 12, 2024 04:57:18.000000000 EDT
    [Timestamp from icmp data (relative): 0.288952646 seconds]
  * Data (48 bytes)

```

Figure 3.2: GRE Capture.

- The Ethernet header includes 00:00:00:00:00:03 as source MAC address and 00:00:00:00:00:04 as destination MAC address, because the packet is sent by the interface *eth03* of the namespace *nsB* to the interface *eth04* of the namespace *nsC*. In fact, the namespace *nsB* does not have a way to contact the namespace *nsD* directly, but its packets must pass through the gateway represented by the namespace *nsC*.
- The GRE tunnel is used. Its usage is confirmed by the presence of two IP headers and the GRE header between them.
- The external IP header includes 10.200.0.2 as source IP address and 10.200.0.1 as destination IP address. They are the IP address of the interface *eth03* of the namespace *nsB* and the IP address of the interface *eth04* of the namespace *nsC*, respectively. These IP addresses are used so that, supposing that between the two gateways there is a shared big network infrastructure such as the Internet, they are used there for routing decisions, instead of the IP addresses of the two remote end points.
- The GRE header specifies that, after it, there is an IP header. This information is provided by the Protocol Type field, which is set to 0x8000, i.e., the value representing the IP protocol.
- The internal IP header includes 192.168.0.2 as source IP address and 192.168.2.1 as destination IP address. They are the IP address of the interface *eth01* of the namespace *nsA* and the IP address of the interface *eth06* of the namespace *nsD*, respectively. It is basically the IP header of the packet that the namespace *nsB* received on the interface *eth02* from the namespace *nsA*. It

must be kept, because it will be finally used by the other tunnel end point to forward the ICMP Echo Request to the final destination.

- The packet data is plain, and can be read by any intermediate node.

After the tunneled ICMP Echo Request reaches the interface *eth04* of the *nsC*, this namespace removes the encapsulation, and sends the decapsulated packet towards the namespace *nsD*.

```

> Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth6, id 0
> Ethernet II, Src: 00:00:00_00:00:05 (00:00:00:00:00:05), Dst: 00:00:00_00:00:06 (00:00:00:00:00:06)
> Internet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.2.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0x0c55 (3157)
> 010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 62
  Protocol: ICMP (1)
  Header Checksum: 0xad00 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.0.2
  Destination Address: 192.168.2.1
> Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x896a [correct]
  [Checksum Status: Good]
  Identifier (BE): 15322 (0x3bda)
  Identifier (LE): 55867 (0xda3b)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 3]
  Timestamp from icmp data: Mar 12, 2024 04:57:18.000000000 EDT
  [Timestamp from icmp data (relative): 0.288970149 seconds]
> Data (48 bytes)

```

Figure 3.3: GRE Capture.

Figure 3.3 shows the Wireshark capture of the ICMP Echo Request captured on the bridge interface *eth6* (i.e., the one linked to the interface *eth06* of the namespace *nsD*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes 00:00:00:00:00:05 as source MAC address and 00:00:00:00:00:06 as destination MAC address, because the packet is sent by the interface *eth05* of the namespace *nsC* to the interface *eth06* of the namespace *nsD*. In fact, the two namespaces are connected to the same IP network and they can communicate directly.
- There is only one IP header, which includes 192.168.0.2 as source IP address and 192.168.2.1 as destination IP address. They are the IP address of the interface *eth01* of the namespace *nsA* and the IP address of the interface *eth06* of the namespace *nsD*, respectively. This header was the encapsulated one of the previous GRE tunnel.
- There is no GRE header or a second IP header, because the ICMP Echo Request left the GRE tunnel at the previous hop.
- The packet data is plain, and can be read by any intermediate node.

From the analysis of these captured packets (and the packets you may capture on other interfaces), it is possible to derive that GRE is a very weak VPN solution from the security point of view. If GRE may offer some networking-oriented features such as basic flow and congestion control, it does not provide security features such as data confidentiality, data authentication, data integrity, or peer authentication. In fact, in this whole exercise you were able to see any part of the header packets, as no encryption was applied to them.

3.3 Cleaning the environment in preparation of the next exercise

Before proceeding with the exercise about IPSec, you need to remove everything that was created specifically for the exercise about GRE. In greater detail, you must delete all the static routes that you have defined for the namespaces *nsB* and *nsC*, and you must delete the two tunnel interfaces:

```
sudo ip netns exec nsB ip route del 192.168.1.1
sudo ip netns exec nsB ip route del 192.168.1.0/24
sudo ip netns exec nsB ip route del default
sudo ip netns exec nsC ip route del 192.168.1.2
sudo ip netns exec nsC ip route del 192.168.1.0/24
sudo ip netns exec nsC ip route del default
sudo ip netns exec nsB ip tunnel del gre1
sudo ip netns exec nsC ip tunnel del gre1
```

4 IPsec

IPsec (IP Security) is the IETF proposal for ensuring security in IP networks. The basic concept behind IPsec is to provide a set of common security services at the network layer to avoid the need to having to duplicate these features in each application. IPsec provides security features such as data confidentiality, data authentication, data integrity, and protection against replay attacks for IP packets. IPsec allows a system to define its own security policies, to select the appropriate protocols for the chosen policies, specify the cryptographic algorithms used, and to acquire the necessary cryptographic keys.

IPsec can be used in transport mode to provide end-to-end security, and in tunnel mode to create a VPN. Whichever mode is selected, IPsec provides two possible headers that can be used to provide security in IP networks:

- the Authentication Header (AH) provides connectionless data integrity and data origin authentication for IP packets and provides protection against IP header modification attacks and replay attacks. These security properties are provided on both the IPsec payload and the IP header on the left of AH.
- the Encapsulating Security Payload (ESP) header provides confidentiality, connectionless data integrity, data origin authentication, an anti-replay service (a form of partial sequence integrity), and limited traffic-flow confidentiality. These security properties are provided only the IPsec payload, so not on the IP header on the left of ESP.

To communicate in IPsec, two nodes must share at least one Security Association (SA) to specify all the security parameters and cryptographic keys necessary for secure communication. Each SA is uniquely identified by three parameters: Security Parameter Index (SPI), IP destination address, and a Security Protocol Identifier (i.e., AH or ESP).

Moreover, some Security Policies (SPs) must be configured to define the rules governing the application of security services (such as authentication, integrity, and confidentiality) to IP packets. These policies determine which traffic should be protected by IPsec and how it should be protected. SPs are typically configured by network administrators and include parameters such as source and destination address, traffic type, action, IPsec protocol (i.e., AH or ESP), encryption and authentication algorithms. SPs establish which SAs are applied to incoming and outgoing traffic, ensuring that only authorized and secure communication occurs.

In summary, Security Associations (SAs) establish the parameters and keys required for secure communication between network entities, while Security Policies (SPs) define the rules governing the application of security services to IP traffic. Together, they form the basis of IPsec security mechanisms.

In this lab activity, you will use AH and ESP in tunnel mode, so as to create an actual IPsec-based VPN and to provide a common ground for comparison with the GRE-based VPN of the previous exercise and the TLS-based VPN of the next exercise.

4.1 Configuration of the IPSec-based VPN with AH

In the first exercise about IPSec, you will use AH as IPSec header.

As preliminary operation, you must set new default routes for the namespaces *nsC* and *nsD*, because the previous ones were removed. With IPSec, you do not need new addresses to be assigned to new interfaces, differently from the previous addresses 192.168.1.2 and 192.168.1.1 assigned to the tunnel interfaces. Therefore, the IP address related to the default route of the namespace *nsC* can simply be 10.200.0.1, i.e., the IP address of the interface *eth04* of the namespace *nsD*. Similarly, the IP address related to the default route of the namespace *nsD* can simply be 10.200.0.2, i.e., the IP address of the interface *eth03* of the namespace *nsc*. These default routes can be configured in the following way:

```
sudo ip netns exec nsB ip route add default via 10.200.0.1
sudo ip netns exec nsC ip route add default via 10.200.0.2
```

Then, you can proceed with the actual IPSec configuration for the namespace *nsC*.

First, you must create two SAs, one for each tunnel direction, as SAs are unidirectional. You can suppose that only data authentication and integrity are requested, and that they are provided by means of the HMAC-SHA1 algorithm. The two commands to be used for the creation of these SAs are the following (be aware that they are only two commands, so on Linux shells they should be executed on two single lines, here they are reported in multiple lines for sake of visualization):

```
sudo ip netns exec nsB ip xfrm state add
src 10.200.0.2 dst 10.200.0.1 proto ah
spi 0x1000 mode tunnel auth hmac\(\sha1\)
0x3a7b9c2f5e8d1a64b7f9e0c2a3d5f8e1a2b4c6d8
sudo ip netns exec nsB ip xfrm state add
src 10.200.0.1 dst 10.200.0.2 proto ah
spi 0x2000 mode tunnel auth hmac\(\sha1\)
0x3a7b9c2f5e8d1a64b7f9e0c2a3d5f8e1a2b4c6d8
```

Here, a brief explanation of the different command components is provided, so that you can also test other kinds of IPSec configurations if you want:

- `ip xfrm`: this component is used to interact with the IP xfrm framework, used to implement the IPsec protocol suite;
- `state`: this component specifies that we are dealing with IPsec SAs, which represent the current security parameters and keys associated with a given unidirectional communication flow;
- `add`: this component is the actual command, specifying the requirement of adding a new IPsec SA;
- `src 10.200.0.2` or `src 10.200.0.1`: this component specifies the source IP address of the traffic that must be protected by this SA (in case of encapsulation, it refers to the external IP header);
- `dst 10.200.0.1` or `dst 10.200.0.2`: this component specifies the destination IP address of the traffic that must be protected by this SA (in case of encapsulation, it refers to the external IP header);
- `proto ah`: this component specifies the IPsec protocol to be used for this SA (in this case, AH instead of ESP);

- `spi 0x1000` or `spi 0x2000`: this component specifies the Security Parameters Index (SPI) for this SA, used to uniquely identify this SA in combination with the destination IP address and the security protocol;
- `mode tunnel`: this component specifies the IPsec mode to be used (in this case, "tunnel" mode, which means the entire original IP packet is encapsulated within a new IP packet);
- `auth hmac\sha1\`: this specifies the authentication algorithm to be used for this SA (in this case, HMAC with SHA-1, which provides data authentication and integrity through hash-based message authentication codes). Be aware that the `\` is actually needed before both brackets.
- `0x3a7b9c2f5e8d1a64b7f9e0c2a3d5f8e1a2b4c6d8`: this component is the hexadecimal representation of the key used for data authentication and integrity (in this case, it is 160 bit long, as requested by HMAC with SHA-1).

Note that the only differences between the two SAs are the SPI and the directional, expressed by the `src` and `dst` components.

Next, you must create the SPs. Specifically, you need two SPs, which can be created with the following two commands:

```
sudo ip netns exec nsB ip xfrm policy add src 192.168.0.2 dst 192.168.2.1
    dir out tmpl src 10.200.0.2 dst 10.200.0.1 proto ah mode tunnel
sudo ip netns exec nsB ip xfrm policy add src 192.168.2.1 dst 192.168.0.2
    dir fwd tmpl src 10.200.0.1 dst 10.200.0.2 proto ah mode tunnel
```

Here, a brief explanation of the different command components is provided, so that you can also test other kinds of IPsec configurations if you want:

- `ip xfrm`: this component is used to interact with the IP xfrm framework, used to implement the IPsec protocol suite;
- `policy`: this component specifies that we are dealing with IPsec SPs, which define the criteria for selecting SAs to protect traffic.;
- `add`: this component is the actual command, specifying the requirement of adding a new IPsec SP;
- `src 192.168.0.2` or `src 192.168.2.1`: this component specifies the source IP address of the traffic to which the SP refers;
- `dst 192.168.2.1` or `dst 192.168.0.2`: this component specifies the destination IP address of the traffic to which the SP refers;
- `dir out` or `dir fwd`: this component specifies the direction of the traffic to which this policy will apply. Specifically, `out` refers to outbound traffic. Instead, `fwd` refers to inbound traffic that is addressed to an IP address that is not installed on the gateway itself (this check is done after decryption, if ESP is used - this is not the case of this example, where AH is used). For completeness, `in` refers to inbound traffic that is addressed to an IP address that is installed on the gateway (e.g., the IP address 10.200.0.2);
- `tmpl`: this component specifies a template describing the transformation the gateway must apply to a packet satisfying the previous conditions. All the following components are connected to this one, as they specify how the transformation must be performed;
- `src 10.200.0.2` or `src 10.200.0.1`: this component specifies the source IP address of the external IP header to be added as encapsulation;

- `dst 10.200.0.1` or `dst 10.200.0.2`: this component specifies the destination IP address of the external IP header to be added as encapsulation;
- `proto ah`: this component specifies the IPsec protocol to be used for this SA (in this case, AH instead of ESP);
- `mode tunnel`: this component specifies the IPsec mode to be used (in this case, "tunnel" mode, which means the entire original IP packet is encapsulated within a new IP packet).

Note that there is no need of having a `dir in` policy, if we simply want to allow that traffic from 192.168.0.2 reaches 192.168.2.1. You can do additional practice by trying other kinds of SAs and SPs, including some policies where the traffic is directed to a locally installed IP address.

In a similar way, you can configure the SAs and the SPs of the namespace *nsC*. You are invited to try to create them by yourself. Here, for sake of completeness, we report the commands that can be used to create them:

```
sudo ip netns exec nsC ip xfrm state add
    src 10.200.0.2 dst 10.200.0.1 proto ah
    spi 0x1000 mode tunnel auth hmacsha1
    0x3a7b9c2f5e8d1a64b7f9e0c2a3d5f8e1a2b4c6d8
sudo ip netns exec nsC ip xfrm state add
    src 10.200.0.1 dst 10.200.0.2 proto ah
    spi 0x2000 mode tunnel auth hmacsha1
    0x3a7b9c2f5e8d1a64b7f9e0c2a3d5f8e1a2b4c6d8
sudo ip netns exec nsC ip xfrm policy add src 192.168.2.1 dst 192.168.0.2
    dir out tmpl src 10.200.0.1 dst 10.200.0.2 proto ah mode tunnel
sudo ip netns exec nsC ip xfrm policy add src 192.168.0.2 dst 192.168.2.1
    dir fwd tmpl src 10.200.0.2 dst 10.200.0.1 proto ah mode tunnel
```

This step concludes the configuration of an IPsec tunnel, based on the AH IPsec protocol, that should allow the namespaces *nsA* and *nsD* to communicate through the public infrastructure that is in between *nsB* and *nsC*. In particular, when crossing that network area, their exchanged packets should be tunneled, so an external IP header with the IP addresses of the two gateways and an AH header should be added. No encryption should be applied, while an HMAC should be computed.

4.2 Verification and analysis of the AH IPsec-based VPN configuration

Now you will verify the correct behavior of the configured IPsec tunnel (with AH) by making the namespace *nsA* ping the namespace *nsD*. At the same time, you are strongly recommended to analyze how the packets appear (i.e., their structure in terms of headers, and the values assigned to the header fields), capturing packets on all the six virtual interfaces of the Linux bridge: *eth1*, *eth2*, *eth3*, *eth4*, *eth5*, and *eth6*. To do so, you should open an instance of the Wireshark packet sniffer on each bridge interface.

The ICMP Echo Requests and Replies are then generated with the following command:

```
sudo ip netns exec nsA ping 192.168.2.1 -c 5
```

Here, for simplicity, we will just consider an ICMP Echo Request captured on three representative interfaces: *eth1*, *eth3*, and *eth6*. However, as previously mentioned, you are invited to complete the

```

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
Internet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.2.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0x013e (318)
  010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: ICMP (1)
  Header Checksum: 0xb617 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.0.2
  Destination Address: 192.168.2.1
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xb113 [correct]
  [Checksum Status: Good]
  Identifier (BE): 34248 (0x85c8)
  Identifier (LE): 51333 (0xc885)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 2]
  Timestamp from icmp data: Mar 13, 2024 10:37:22.000000000 EDT
  [Timestamp from icmp data (relative): 0.274421801 seconds]
  Data (48 bytes)

```

Figure 4.1: IPsec (AH) Capture.

exercise analyzing also the ICMP Echo Requests captured on the other three interfaces, and the ICMP Echo Replies on all six interfaces, so as to complete the comparative analysis.

Figure 4.1 shows the Wireshark capture of an ICMP Echo Request captured on the bridge interface *eth1* (i.e., the one linked to the interface *eth01* of the namespace *nsA*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes 00:00:00:00:00:01 as source MAC address and 00:00:00:00:00:02 as destination MAC address, because the packet is sent by the interface *eth01* of the namespace *nsA* to the interface *eth02* of the namespace *nsB*. In fact, the namespace *nsA* does not have a way to contact the namespace *nsD* directly, but its packets must pass through the gateway represented by the namespace *nsB*.
- There is only one IP header, which includes 192.168.0.2 as source IP address and 192.168.2.1 as destination IP address. They are the IP address of the interface *eth01* of the namespace *nsA* and the IP address of the interface *eth06* of the namespace *nsD*, respectively. The presence of these IP addresses is expected, in compliance with the way IP networking behaves.
- There is no AH header or a second IP header, because the ICMP Echo Request has not yet reached the IPsec tunnel.
- The packet data is plain, and can be read by any intermediate node.

After the ICMP Echo Request reaches the interface *eth02* of the *nsB*, this namespace sends it towards the namespace *nsC*. In particular, according to the previous configuration of the VPN, the IPsec tunnel will be used to forward this packet.

Figure 4.2 shows the Wireshark capture of the ICMP Echo Request captured on the bridge interface *eth3* (i.e., the one linked to the interface *eth03* of the namespace *nsB*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes 00:00:00:00:00:03 as source MAC address and 00:00:00:00:00:04 as destination MAC address, because the packet is sent by the interface *eth03* of the namespace

```

* Frame 2: 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits) on interface eth3, id 0
* Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04)
* Internet Protocol Version 4, Src: 10.200.0.2, Dst: 10.200.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  * Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 128
    Identification: 0x0000 (0)
  * 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: Authentication Header (51)
    Header Checksum: 0x24b9 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.200.0.2
    Destination Address: 10.200.0.1
  * Authentication Header
    Next header: IPIP (4)
    Length: 4 (24 bytes)
    Reserved: 0000
    AH SPI: 0x00001000
    AH Sequence: 11
    AH ICV: 510df57eaa5f9e809b716d34
  * Internet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.2.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    * Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 84
      Identification: 0x013e (318)
    * 010. .... = Flags: 0x2, Don't fragment
      ...0 0000 0000 0000 = Fragment Offset: 0
      Time to Live: 63
      Protocol: ICMP (1)
      Header Checksum: 0xb717 [validation disabled]
      [Header checksum status: Unverified]
      Source Address: 192.168.0.2
      Destination Address: 192.168.2.1
  * Internet Control Message Protocol

```

Figure 4.2: IPSec (AH) Capture.

nsB to the interface *eth04* of the namespace *nsC*. In fact, the namespace *nsB* does not have a way to contact the namespace *nsD* directly, but its packets must pass through the gateway represented by the namespace *nsC*.

- The IPSec tunnel is used. Its usage is confirmed by the presence of two IP headers and the AH header between them.
- The external IP header includes 10.200.0.2 as source IP address and 10.200.0.1 as destination IP address. They are the IP address of the interface *eth03* of the namespace *nsB* and the IP address of the interface *eth04* of the namespace *nsC*, respectively. These IP addresses are used so that, supposing that between the two gateways there is a shared big network infrastructure such as the Internet, they are used there for routing decisions, instead of the IP addresses of the two remote end points.
- The AH header specifies that, after it, there is an IP header. This information is provided by the Next Header field, which is set to 4, i.e., the value representing the IP protocol. It also includes the SPI value, a sequence number used for providing protection against replay attacks, and the ICV (Integrity Check Value), that is the result of the HMAC-SHA1 application to provide data authentication and integrity. In this way, the recipient will be able to compare the received ICV with the locally computed one.
- The internal IP header includes 192.168.0.2 as source IP address and 192.168.2.1 as destination IP address. They are the IP address of the interface *eth01* of the namespace *nsA* and the IP address of the interface *eth06* of the namespace *nsD*, respectively. It is basically the IP header of the packet that the namespace *nsB* received on the interface *eth02* from the namespace *nsA*. It must be kept, because it will be finally used by the other tunnel end point to forward the ICMP Echo Request to the final destination.

- The packet data is plain, and can be read by any intermediate node.

After the tunneled ICMP Echo Request reaches the interface *eth04* of the *nsC*, this namespace removes the encapsulation, and sends the decapsulated packet towards the namespace *nsD*.

```

Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth6, id 0
Ethernet II, Src: 00:00:00_00:00:05 (00:00:00:00:00:05), Dst: 00:00:00_00:00:06 (00:00:00:00:00:06)
  Destination: 00:00:00_00:00:06 (00:00:00:00:00:06)
  Source: 00:00:00_00:00:05 (00:00:00:00:00:05)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.2.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0x0143 (323)
  010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 62
  Protocol: ICMP (1)
  Header Checksum: 0xb812 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.0.2
  Destination Address: 192.168.2.1
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xacfb [correct]
  [Checksum Status: Good]
  Identifier (BE): 34248 (0x85c8)
  Identifier (LE): 51333 (0xc885)
  Sequence Number (BE): 2 (0x0002)
  Sequence Number (LE): 512 (0x0200)
  [Response frame: 4]
  Timestamp from icmp data: Mar 13, 2024 10:37:23.000000000 EDT
  [Timestamp from icmp data (relative): 0.280428755 seconds]
  Data (48 bytes)

```

Figure 4.3: IPsec (AH) Capture.

Figure 4.3 shows the Wireshark capture of the ICMP Echo Request captured on the bridge interface *eth6* (i.e., the one linked to the interface *eth06* of the namespace *nsD*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes 00:00:00:00:00:05 as source MAC address and 00:00:00:00:00:06 as destination MAC address, because the packet is sent by the interface *eth05* of the namespace *nsC* to the interface *eth06* of the namespace *nsD*. In fact, the two namespaces are connected to the same IP network and they can communicate directly.
- There is only one IP header, which includes 192.168.0.2 as source IP address and 192.168.2.1 as destination IP address. They are the IP address of the interface *eth01* of the namespace *nsA* and the IP address of the interface *eth06* of the namespace *nsD*, respectively. This header was the encapsulated one of the previous IPsec tunnel.
- There is no AH header or a second IP header, because the ICMP Echo Request left the IPsec tunnel at the previous hop.
- The packet data is plain, and can be read by any intermediate node.

From the analysis of these captured packets (and the packets you may capture on other interfaces), it is possible to derive that AH provides more security features with respect to GRE: data authentication, data integrity, and protection against replay attacks. Besides, all these properties are applied to both the external IP header and to the IPsec payload (including the internal IP header, in tunnel mode). However, it does not provide data confidentiality, as proved by the fact you were able to see any part of the header packets, as no encryption was applied to them.

4.3 Configuration of the IPSec-based VPN with ESP

In the second exercise about IPSec, you will use ESP as IPSec header.

First of all, you need to delete all the SAs and SPs that you previously set:

```
sudo ip netns exec nsB ip xfrm policy flush
sudo ip netns exec nsB ip xfrm state flush
sudo ip netns exec nsC ip xfrm policy flush
sudo ip netns exec nsC ip xfrm state flush
```

Then, you can create the SAs and the SPs of the namespace *nsB*. You can suppose that you only want to provide data confidentiality, without data authentication and integrity (even if ESP supports those properties).

You can create the two SAs with the following commands:

```
sudo ip netns exec nsB ip xfrm state add
    src 10.200.0.2 dst 10.200.0.1 proto esp
    spi 0x1000 mode tunnel enc aes
    0xaa223344556677889900aabbccddeeff
sudo ip netns exec nsB ip xfrm state add
    src 10.200.0.1 dst 10.200.0.2 proto esp
    spi 0x2000 mode tunnel enc aes
    0xaa223344556677889900aabbccddeeff
```

Differently from the commands you used to create the AH-based SAs, here you specify *esp* as IPSec protocol, and then you provide information about the encryption algorithm (*enc*) that must be used for data confidentiality, i.e., *aes*. The hexadecimal string represents the encryption key, that must be shared with the other peer so that it can decrypt the traffic.

Instead, the SPs can be created with the following commands:

```
sudo ip netns exec nsB ip xfrm policy add src 192.168.0.2 dst 192.168.2.1 dir out
    tmpl src 10.200.0.2 dst 10.200.0.1 proto esp mode tunnel
sudo ip netns exec nsB ip xfrm policy add src 192.168.2.1 dst 192.168.0.2 dir fwd
    tmpl src 10.200.0.1 dst 10.200.0.2 proto esp mode tunnel
```

These ones are almost identical to the SPs defined for the AH-based tunnel. The only difference is that you specify *esp* as IPSec protocol.

In a similar way, you can configure the SAs and the SPs of the namespace *nsC*. You are invited to try to create them by yourself. Here, for sake of completeness, we report the commands that can be used to create them:

```
sudo ip netns exec nsC ip xfrm state add
    src 10.200.0.2 dst 10.200.0.1 proto esp
    spi 0x1000 mode tunnel enc aes
    0xaa223344556677889900aabbccddeeff
sudo ip netns exec nsC ip xfrm state add
    src 10.200.0.1 dst 10.200.0.2 proto esp
    spi 0x2000 mode tunnel enc aes
    0xaa223344556677889900aabbccddeeff
```

```

sudo ip netns exec nsC ip xfrm policy add src 192.168.2.1 dst 192.168.0.2 dir out
    tmpl src 10.200.0.1 dst 10.200.0.2 proto esp mode tunnel
sudo ip netns exec nsC ip xfrm policy add src 192.168.0.2 dst 192.168.2.1 dir fwd
    tmpl src 10.200.0.2 dst 10.200.0.1 proto esp mode tunnel

```

This step concludes the configuration of an IPsec tunnel, based on the ESP IPsec protocol, that should allow the namespaces *nsA* and *nsD* to communicate through the public infrastructure that is in between *nsB* and *nsC*. In particular, when crossing that network area, their exchanged packets should be tunneled, so an external IP header with the IP addresses of the two gateways and an ESP header should be added. Encryption should be applied, so you should expect not be able to see the content of the IPsec payload.

4.4 Verification and analysis of the ESP IPsec-based VPN configuration

Now you will verify the correct behavior of the configured IPsec tunnel (with ESP) by making the namespace *nsA* ping the namespace *nsD*. At the same time, you are strongly recommended to analyze how the packets appear (i.e., their structure in terms of headers, and the values assigned to the header fields), capturing packets on all the six virtual interfaces of the Linux bridge: *eth1*, *eth2*, *eth3*, *eth4*, *eth5*, and *eth6*. To do so, you should open an instance of the Wireshark packet sniffer on each bridge interface.

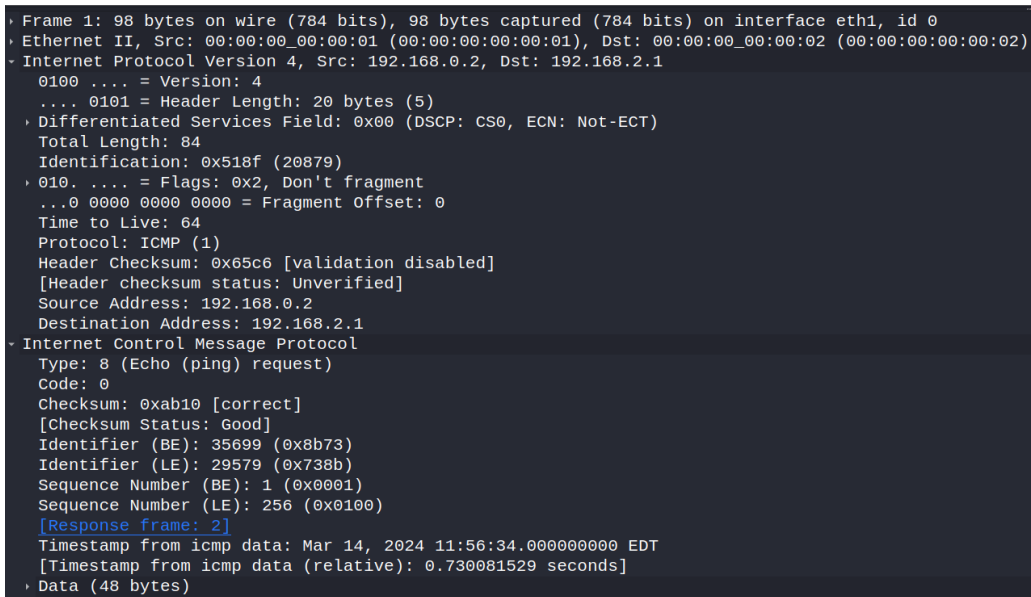
The ICMP Echo Requests and Replies are then generated with the following command:

```

sudo ip netns exec nsA ping 192.168.2.1 -c 5

```

Here, for simplicity, we will just consider an ICMP Echo Request captured on three representative interfaces: *eth1*, *eth3*, and *eth6*. However, as previously mentioned, you are invited to complete the exercise analyzing also the ICMP Echo Requests captured on the other three interfaces, and the ICMP Echo Replies on all six interfaces, so as to complete the comparative analysis.



```

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
Internet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.2.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0x518f (20879)
  010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: ICMP (1)
  Header Checksum: 0x65c6 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.0.2
  Destination Address: 192.168.2.1
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xab10 [correct]
  [Checksum Status: Good]
  Identifier (BE): 35699 (0x8b73)
  Identifier (LE): 29579 (0x738b)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 2]
Timestamp from icmp data: Mar 14, 2024 11:56:34.000000000 EDT
[Timestamp from icmp data (relative): 0.730081529 seconds]
Data (48 bytes)

```

Figure 4.4: IPsec (ESP) Capture.

Figure 4.4 shows the Wireshark capture of an ICMP Echo Request captured on the bridge interface *eth1* (i.e., the one linked to the interface *eth01* of the namespace *nsA*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes 00:00:00:00:00:01 as source MAC address and 00:00:00:00:00:02 as destination MAC address, because the packet is sent by the interface *eth01* of the namespace *nsA* to the interface *eth02* of the namespace *nsB*. In fact, the namespace *nsA* does not have a way to contact the namespace *nsD* directly, but its packets must pass through the gateway represented by the namespace *nsB*.
- There is only one IP header, which includes 192.168.0.2 as source IP address and 192.168.2.1 as destination IP address. They are the IP address of the interface *eth01* of the namespace *nsA* and the IP address of the interface *eth06* of the namespace *nsD*, respectively. The presence of these IP addresses is expected, in compliance with the way IP networking behaves.
- There is no ESP header, because the ICMP Echo Request has not yet reached the IPsec tunnel.
- The packet data is plain, and can be read by any intermediate node.

After the ICMP Echo Request reaches the interface *eth02* of the *nsB*, this namespace sends it towards the namespace *nsC*. In particular, according to the previous configuration of the VPN, the IPsec tunnel will be used to forward this packet.

Figure 4.5 shows the Wireshark capture of the ICMP Echo Request captured on the bridge interface *eth3* (i.e., the one linked to the interface *eth03* of the namespace *nsB*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes 00:00:00:00:00:03 as source MAC address and 00:00:00:00:00:04 as destination MAC address, because the packet is sent by the interface *eth03* of the namespace *nsB* to the interface *eth04* of the namespace *nsC*. In fact, the namespace *nsB* does not have a way to contact the namespace *nsD* directly, but its packets must pass through the gateway represented by the namespace *nsC*.
- In the capture, there is an IP header including 10.200.0.2 as source IP address and 10.200.0.1 as destination IP address. They are the IP address of the interface *eth03* of the namespace *nsB* and the IP address of the interface *eth04* of the namespace *nsC*, respectively. These IP addresses are used so that, supposing that between the two gateways there is a shared big network

```

Frame 1: 154 bytes on wire (1232 bits), 154 bytes captured (1232 bits) on interface eth3, id 0
Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04)
Internet Protocol Version 4, Src: 10.200.0.2, Dst: 10.200.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 140
    Identification: 0x0000 (0)
  010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: Encap Security Payload (50)
    Header Checksum: 0x24ae [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.200.0.2
    Destination Address: 10.200.0.1
  Encapsulating Security Payload
    ESP SPI: 0x00001000 (4096)
    ESP Sequence: 6

```

Figure 4.5: IPsec (AH) Capture.

infrastructure such as the Internet, they are used there for routing decisions, instead of the IP addresses of the two remote end points.

- Then, the captured packet presents the ESP header, including the SPI value and a sequence number.
- ... but then, you are not able to see any other IP header and the rest of the IPSec payload. They are actually present, and the tunnel has been successfully created. However, they are encrypted, so Wireshark cannot decrypt them to show them to the user.

After the tunneled ICMP Echo Request reaches the interface *eth04* of the *nsC*, this namespace removes the encapsulation, and sends the decapsulated packet towards the namespace *nsD*.

Figure 4.6 shows the Wireshark capture of the ICMP Echo Request captured on the bridge interface *eth6* (i.e., the one linked to the interface *eth06* of the namespace *nsD*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes 00:00:00:00:00:05 as source MAC address and 00:00:00:00:00:06 as destination MAC address, because the packet is sent by the interface *eth05* of the namespace *nsC* to the interface *eth06* of the namespace *nsD*. In fact, the two namespaces are connected to the same IP network and they can communicate directly.
- There is only one IP header, which includes 192.168.0.2 as source IP address and 192.168.2.1 as destination IP address. They are the IP address of the interface *eth01* of the namespace *nsA* and the IP address of the interface *eth06* of the namespace *nsD*, respectively. This header was the encapsulated one of the previous IPSec tunnel.
- There is no ESP header, because the ICMP Echo Request left the IPSec tunnel at the previous hop.
- The packet data is plain, and can be read by any intermediate node.

From the analysis of these captured packets (and the packets you may capture on other interfaces), it is possible to derive that ESP provides encryption with respect to AH. This example does not show

```

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth6, id 0
Ethernet II, Src: 00:00:00_00:00:05 (00:00:00:00:00:05), Dst: 00:00:00_00:00:06 (00:00:00:00:00:06)
Internet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.2.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x518f (20879)
  010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 62
    Protocol: ICMP (1)
    Header Checksum: 0x67c6 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.0.2
    Destination Address: 192.168.2.1
  Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xab10 [correct]
    [Checksum Status: Good]
    Identifier (BE): 35699 (0x8b73)
    Identifier (LE): 29579 (0x738b)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
    [Response frame: 2]
    Timestamp from icmp data: Mar 14, 2024 11:56:34.000000000 EDT
    [Timestamp from icmp data (relative): 0.730131131 seconds]
  Data (48 bytes)

```

Figure 4.6: IPSec (ESP) Capture.

it, but ESP also provides data authentication, data integrity and protection against replay attacks, if an ICV is computed. You may check it by modifying the ESP tunnel configuration as additional exercise. Anyhow, ESP provides all these features only to the IPSec payload, and not also on the IP header on its left, differently from AH.

4.5 Cleaning the environment in preparation of the next exercise

Before proceeding with the exercise about TLS, you need to remove everything that was created specifically for the exercise about IPSec. In greater detail, you must delete the two default routes that you have defined for the namespaces *nsB* and *nsC*, and you must flush the SA and SP tables:

```
sudo ip netns exec nsB ip route del default
sudo ip netns exec nsC ip route del default
sudo ip netns exec nsB ip xfrm policy flush
sudo ip netns exec nsB ip xfrm state flush
sudo ip netns exec nsC ip xfrm policy flush
sudo ip netns exec nsC ip xfrm state flush
```

5 TLS

Transport Layer Security (TLS) is another protocol that can be used to enforce security on communications over untrusted network infrastructures. Originally proposed as Secure Socker Layer (SSL) by Netscape Communications and renamed as TLS when it became an IETF standard, it is logically places in the session layer of the ISO/OSI stack, with the objective to provide security externally from the applications. More specifically, the security properties it may provide are the following ones:

- peer authentication through C.509 certificates during the execution of the TLS hadnshake protocol before the creation of the TLS session (the server authentication is mandatory, whereas the client authentication is optional);
- data confidentiality through the application of encryption algorithms;
- data authentication and data integrity through the computation of a keyed digest or Message Authentication Code (MAC);
- protection from replay and filtering by including a sequence number in the computation of the MAC.

In view of these properties, TLS can be used not only to provide end-to-end security (e.g., for web communications between a browser and a web server), but also to create VPNs.

5.1 Configuration of the TLS-based VPN

In this activity, the TLS tunnel representing the TLS-based VPN will be created between the two namespaces acting as VPN gateways, i.e., between the namespaces *nsB* and *nsC*. In particular, a gateway (namespace *nsC*) will be configured as a TLS server, while the other one (namespace *nsB*) as TLS client.

First, you must install `openvpn`, an open-source program that you will use to create the TLS-based VPN, by executing the following command:

```
sudo apt update
sudo apt install openvpn
```

Another program that must be installed in this preliminary part of the activity about TLS is `easy-rsa`. As previously mentioned, the peer authentication, which is mandatory for the TLS server, is based on X.509 certificates. Therefore, you will need to create a Certification Authority (CA) and to make it emit the certificates. From this point of view, `easy-rsa` is a simplified utility that can ease all these operations, so that you can focus on the tasks related on VPNs as soon as possible. This utility can be installed with the following command:

```
sudo apt install easy-rsa
```

Directory preparation

First, the directory that will contain the certificates, the keys, etc. is prepared:

```
cd /etc/openvpn
sudo make-cadir easy-rsa/
```

The `easy-rsa` directory does not have to exist, because it is created with the previous command and initialized with default contents.

Inside this `easy-rsa` director, you must create the `./pki/openssl-easyrsa.cnf` file and the `./pki/private/` and `./pki/reqs/` directories intended to hold certificates and keys. You can do so with the following commands:

```
sudo chmod 777 /etc/openvpn -R
cd easy-rsa
sudo ./easyrsa init-pki
```

CA creation

Before creating the Certification Authority certificate and its private key, you must edit the vars:

```
sudo gedit vars
```

Specifically, you must set at least the following variables (here an example is reported, but you can modify them according to your preference):

```
set_var EASYRSA_REQ_COUNTRY "IT"
set_var EASYRSA_REQ_PROVINCE "TO"
set_var EASYRSA_REQ_CITY "Turin"
set_var EASYRSA_REQ_ORG "Politecnico di Torino"
set_var EASYRSA_REQ_EMAIL "camail@example.net"
set_var EASYRSA_REQ_OU "DAUIN"

set_var EASYRSA_CA_EXPIRE 3650
set_var EASYRSA_CERT_EXPIRE 825
```

The creation of the Certification Authority is to be done once and lasts for as long as specified by `EASYRSA_CA_EXPIRE`. The command to be executed for its creation is the following:

```
sudo ./easyrsa build-ca nopass
```

Omitting the `nopass` parameter asks for a passphrase, which will be used in the future to sign certificates issued by this CA – it is actually the password needed to unlock the CA private key. The `nopass` parameter avoids the need to type a password in all subsequent operations, to make them faster in this exercise. In a real context, this parameter should be avoided.

During this step, the Common Name of the certification authority is asked. In general, the CN can be identified with the host name of the machine running the PKI infrastructure – here, you can simply put the name you prefer, e.g., `MyCa`.

Of all the files created, the most important are `./pki/ca.crt` and `./pki/private/ca.key`, which are the CA (public) certificate and its private key, respectively.

Generation of the server certificate and private key

In order to generate the server certificate and its private key, you can execute the following command:

```
sudo ./easyrsa build-server-full server nopass
```

During the procedure, you must simply confirm the creation request by typing “yes”.

In the absence of the `nopass` option a PEM pass phrase is asked to protect the server key, this password will have to be typed in every time the OpenVPN server is started.

The private key will be saved in a `./pki/private/server.key` file, the public certificate on the other hand in `./pki/issued/server.crt`.

Diffie-Hellman key generation

A Diffie-Hellman key is also needed for the TLS handshake phase. To create it, you can simply execute the following command:

```
sudo ./easyrsa gen-dh
```

The file `/etc/openvpn/easy-rsa/pki/dh.pem` is thus created. This file will be referred to directly by the OpenVPN configuration file.

Generation of a static key for TLS authentication

Then, you should create a static pre-shared key for TLS authentication:

```
sudo openvpn --genkey secret /etc/openvpn/server/ta.key
```

This key is used in the initial phase of TLS to prevent DoS-type attacks. In this way, a client without this key is blocked before attempting a connection.

Creation of the server configuration file

Finally, you can create the server configuration file:

```
sudo gedit /etc/openvpn/server.conf
```

The content of this file should be as follows:

```
verb 3
status /var/log/openvpn/openvpn-status.log

dev tun
proto udp
local 10.200.0.1
lport 1195
remote 10.200.0.2
```

```

rport 1194

ca /etc/openvpn/easy-rsa/pki/ca.crt
cert /etc/openvpn/easy-rsa/pki/issued/server.crt
key /etc/openvpn/easy-rsa/pki/private/server.key
dh /etc/openvpn/easy-rsa/pki/dh.pem

tls-server

ifconfig 192.168.1.1 192.168.1.2
route 192.168.0.0 255.255.255.0

keepalive 200 600
tls-auth /etc/openvpn/server/ta.key 0
auth-nocache
auth SHA256
cipher AES-256-CBC
persist-key
persist-tun
explicit-exit-notify 1

```

You can notice that most of previously created files are reported here, and that there is an indication of the local and remote address of the TLS tunnel. From those, you can understand that this file will be the configuration for the namespace *nsC*, as its local address is 10.200.0.1. You are also suggested to try to understand the other parameters set in this file.

Generation of the client certificate and private key

Now, it is time to complete the configuration of the TLS tunnel from the client point of view.

In order to generate the client certificate and its private key, you can execute the following command:

```
sudo ./easyrsa build-client-full client nopass
```

During the procedure, you must simply confirm the creation request by typing “yes”.

In the absence of the *nopass* option a PEM pass phrase is asked to protect the client key, this password will have to be typed in every time the OpenVPN client is started.

The private key will be saved in a *./pki/private/client.key* file, the public certificate on the other hand in *./pki/issued/client.crt*.

Creation of the client configuration file

Now you can client the server configuration file:

```
sudo gedit /etc/openvpn/client.conf
```

The content of this file should be as follows:

```

verb 3
tls-client

dev tun
proto udp
local 10.200.0.2
lport 1194
remote 10.200.0.1
rport 1195

ifconfig 192.168.1.2 192.168.1.1
route 192.168.2.0 255.255.255.0

persist-tun
persist-key
keepalive 200 600
ping-timer-rem

remote-cert-tls server
resolv-retry infinite

key-direction 1
data-ciphers AES-256-GCM:AES-128-GCM
auth SHA256
auth-nocache

<ca>
-----BEGIN CERTIFICATE-----
include here the CA certificate
-----END CERTIFICATE-----
</ca>

<cert>
-----BEGIN CERTIFICATE-----
include here the client certificate
-----END CERTIFICATE-----
</cert>

<key>
-----BEGIN PRIVATE KEY-----
include here the client private key
-----END PRIVATE KEY-----
</key>

<tls-auth>
-----BEGIN OpenVPN Static key V1-----
include here the static key
-----END OpenVPN Static key V1-----
</tls-auth>

```

You should complete this configuration by including the actual content of the last four components, as their values depend on the certificates and keys you created on your machine. Specifically, in order, you can retrieve those pieces of information from the following files:

- `/etc/openvpn/easy-rsa/pki/ca.crt` for the CA certificate;
- `/etc/openvpn/easy-rsa/pki/issued/client.crt` for the client certificate;
- `/etc/openvpn/easy-rsa/pki/private/client.key` for the client private key;
- `/etc/openvpn/server/ta.key` for the static key.

5.1.1 Running the TLS server and client

Finally, you are at the end of the configuration phase. First, you must launch the TLS server:

```
sudo ip netns exec nsC openvpn --config /etc/openvpn/server.conf
```

Second, you can proceed with launching the TLS client:

```
sudo ip netns exec nsB openvpn --config /etc/openvpn/client.conf
```

If you see `Initialization Sequence Completed` printed in their shells, it means that the TLS tunnel has been set up successfully, and you can proceed with the last task of this lab activity.

5.2 Verification and analysis of the TLS-based VPN configuration

Now you will verify the correct behavior of the configured TLS tunnel by making the namespace *nsA* ping the namespace *nsD*. At the same time, you are strongly recommended to analyze how the packets appear (i.e., their structure in terms of headers, and the values assigned to the header fields), capturing packets on all the six virtual interfaces of the Linux bridge: *eth1*, *eth2*, *eth3*, *eth4*, *eth5*, and *eth6*. To do so, you should open an instance of the Wireshark packet sniffer on each bridge interface.

The ICMP Echo Requests and Replies are then generated with the following command:

```
sudo ip netns exec nsA ping 192.168.2.1 -c 5
```

Here, for simplicity, we will just consider an ICMP Echo Request captured on three representative interfaces: *eth1*, *eth3*, and *eth6*. However, as previously mentioned, you are invited to complete the exercise analyzing also the ICMP Echo Requests captured on the other three interfaces, and the ICMP Echo Replies on all six interfaces, so as to complete the comparative analysis.

Figure 5.1 shows the Wireshark capture of an ICMP Echo Request captured on the bridge interface *eth1* (i.e., the one linked to the interface *eth01* of the namespace *nsA*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes `00:00:00:00:00:01` as source MAC address and `00:00:00:00:00:02` as destination MAC address, because the packet is sent by the interface *eth01* of the namespace *nsA* to the interface *eth02* of the namespace *nsB*. In fact, the namespace *nsA* does not have a way to contact the namespace *nsD* directly, but its packets must pass through the gateway represented by the namespace *nsB*.


```

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
Internet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.2.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0x4549 (17737)
  010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: ICMP (1)
  Header Checksum: 0x720c [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.0.2
  Destination Address: 192.168.2.1
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x3e4e [correct]
  [Checksum Status: Good]
  Identifier (BE): 55208 (0xd7a8)
  Identifier (LE): 43223 (0xa8d7)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 2]
  Timestamp from icmp data: Mar 15, 2024 05:43:27.000000000 EDT
  [Timestamp from icmp data (relative): 0.440050836 seconds]
  Data (48 bytes)

```

Figure 5.1: TLS Capture.

- There is only one IP header, which includes 192.168.0.2 as source IP address and 192.168.2.1 as destination IP address. They are the IP address of the interface *eth01* of the namespace *nsA* and the IP address of the interface *eth06* of the namespace *nsD*, respectively. The presence of these IP addresses is expected, in compliance with the way IP networking behaves.
- There is no UDP or OpenVPN header, because the ICMP Echo Request has not yet reached the TLS tunnel.
- The packet data is plain, and can be read by any intermediate node.

After the ICMP Echo Request reaches the interface *eth02* of the *nsB*, this namespace sends it towards the namespace *nsC*. In particular, according to the previous configuration of the VPN, the TLS tunnel will be used to forward this packet.

Figure 5.2 shows the Wireshark capture of the ICMP Echo Request captured on the bridge interface *eth3* (i.e., the one linked to the interface *eth03* of the namespace *nsB*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes 00:00:00:00:00:03 as source MAC address and 00:00:00:00:00:04 as destination MAC address, because the packet is sent by the interface *eth03* of the namespace *nsB* to the interface *eth04* of the namespace *nsC*. In fact, the namespace *nsB* does not have a way to contact the namespace *nsD* directly, but its packets must pass through the gateway represented by the namespace *nsC*.
- In the capture, there is an IP header including 10.200.0.2 as source IP address and 10.200.0.1 as destination IP address. They are the IP address of the interface *eth03* of the namespace *nsB* and the IP address of the interface *eth04* of the namespace *nsC*, respectively. These IP addresses are used so that, supposing that between the two gateways there is a shared big network infrastructure such as the Internet, they are used there for routing decisions, instead of the IP addresses of the two remote end points.
- Then, the captured packet has two headers: an UDP header and an OpenVPN header. They are used to perform the encapsulation of the ICMP Echo Request packet. The usage of UDP is

```

* Frame 4: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits) on interface eth3, id 0
* Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04)
* Internet Protocol Version 4, Src: 10.200.0.2, Dst: 10.200.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
* Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 136
  Identification: 0xc769 (51049)
* 010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: UDP (17)
  Header Checksum: 0x5d69 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 10.200.0.2
  Destination Address: 10.200.0.1
* User Datagram Protocol, Src Port: 1194, Dst Port: 1195
  Source Port: 1194
  Destination Port: 1195
  Length: 116
  Checksum: 0x1618 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
  [Timestamps]
  UDP payload (108 bytes)
* OpenVPN Protocol
  Type: 0x48 [opcode/key_id]
  Peer ID: 4293911
  Data (104 bytes)

```

Figure 5.2: TLS Capture.

determined by the way you created the TLS server and client configuration files. From this point of view, you may have also selected TCP.

- After those headers, you are not able to see the rest of the IPsec payload (you cannot see the MAC, either). They are actually present, and the tunnel has been successfully created. However, they are encrypted, so Wireshark cannot decrypt them to show them to the user.

After the tunneled ICMP Echo Request reaches the interface *eth04* of the *nsC*, this namespace removes the encapsulation, and sends the decapsulated packet towards the namespace *nsD*.

Figure 5.3 shows the Wireshark capture of the ICMP Echo Request captured on the bridge interface *eth6* (i.e., the one linked to the interface *eth06* of the namespace *nsD*), when the namespace *nsA* pings the namespace *nsD*. The most noteworthy analysis considerations about this captured packet are listed in the following:

- The Ethernet header includes 00:00:00:00:00:05 as source MAC address and 00:00:00:00:00:06 as destination MAC address, because the packet is sent by the interface *eth05* of the namespace *nsC* to the interface *eth06* of the namespace *nsD*. In fact, the two namespaces are connected to the same IP network and they can communicate directly.
- There is only one IP header, which includes 192.168.0.2 as source IP address and 192.168.2.1 as destination IP address. They are the IP address of the interface *eth01* of the namespace *nsA* and the IP address of the interface *eth06* of the namespace *nsD*, respectively. This header was the encapsulated one of the previous TLS tunnel.
- There is no UDP or OpenVPN header, because the ICMP Echo Request left the TLS tunnel at the previous hop.
- The packet data is plain, and can be read by any intermediate node.

From the analysis of these captured packets (and the packets you may capture on other interfaces), it is possible to derive that TLS provides many security features you may want for a VPN. With respect to IPsec based on ESP, TLS also provides peer authentication thanks to the initial TSL handshake phase. Moreover, it follows an authenticate-then-encrypt strategy, as the MAC computed for data

```

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth6, id 0
Ethernet II, Src: 00:00:00_00:00:05 (00:00:00:00:00:05), Dst: 00:00:00_00:00:06 (00:00:00:00:00:06)
Internet Protocol Version 4, Src: 192.168.0.2, Dst: 192.168.2.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0x4549 (17737)
  010. .... = Flags: 0x2, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 62
  Protocol: ICMP (1)
  Header Checksum: 0x740c [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.0.2
  Destination Address: 192.168.2.1
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x3e4e [correct]
  [Checksum Status: Good]
  Identifier (BE): 55208 (0xd7a8)
  Identifier (LE): 43223 (0xa8d7)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 2]
  Timestamp from icmp data: Mar 15, 2024 05:43:27.000000000 EDT
  [Timestamp from icmp data (relative): 0.440203984 seconds]
  Data (48 bytes)

```

Figure 5.3: TLS Capture.

authentication, data integrity and protection against replay attacks is encrypted, so that it is not sent as plain data.