Instituto Superior Técnico

Advanced Network Security and Architectures

Laboratory Report:

Module 2

Group 9

Félix Saraiva – 98752

Rafael Martins – 90629

Miguel Ribeiro – 87553

2022

Table of Contents:

Table of Figures:

1. Lab No. 2.1 – IPSec and VPNs

This laboratory work aims to study IPSec and VPNs. The laboratories analysed are:

- **I.** IPSec using ESP in tunnel mode
- **II.** IPSec with digital certificates and certificate authority
- **III.** IPSec with NAT traversal
- **IV.** GRE over IPSec
- **V.** DMVPN Phase 3
- **VI.** DMVPN over IPSec

1.1. IPSec using ESP in tunnel mode

In this exercise, we connect two branches of an organisation using an IPSec tunnel. After the address configurations, we have configured **OSPF** as routing protocol in the public subnets and static routes for the organisation to be able to communicate externally. The complete configurations are present in Annex A.

IPSec can be used in several situations, such as establishing a secure link between a firewall and a host. Knowing the requirements in terms of configuration to establish an IPSec tunnel is essential.

So, to make IPSec work through your firewalls, you should open UDP port 500 and permit IP protocol numbers 50 and 51 on both inbound and outbound firewall filters. UDP Port 500 should be opened to allow Internet Security Association and Key Management Protocol (ISAKMP) traffic to be forwarded through your firewalls. IP protocol ID 50 should be set to enable IPSec Encapsulating Security Protocol (ESP) traffic to be delivered. Finally, IP protocol ID 51 should be set to allow Authentication Header (AH) traffic to be forwarded. (Clercq, 2012)

Network Architecture:

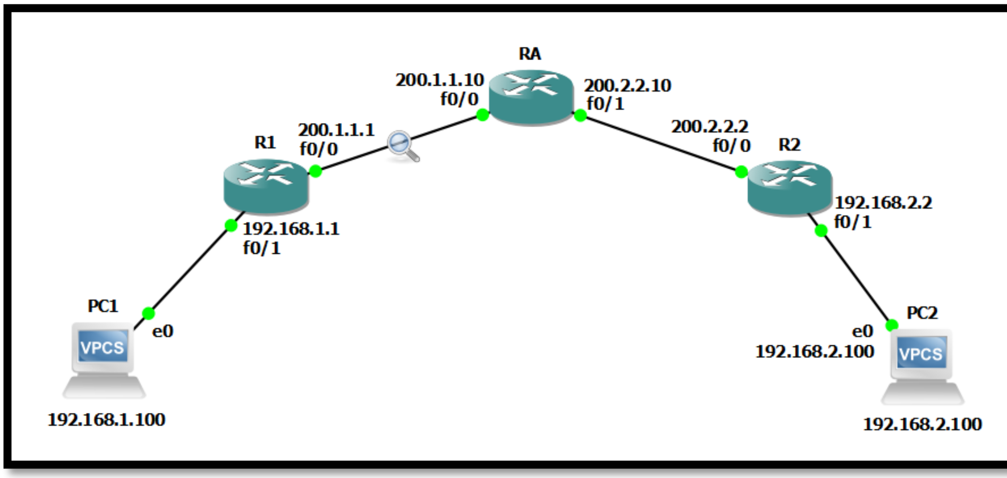
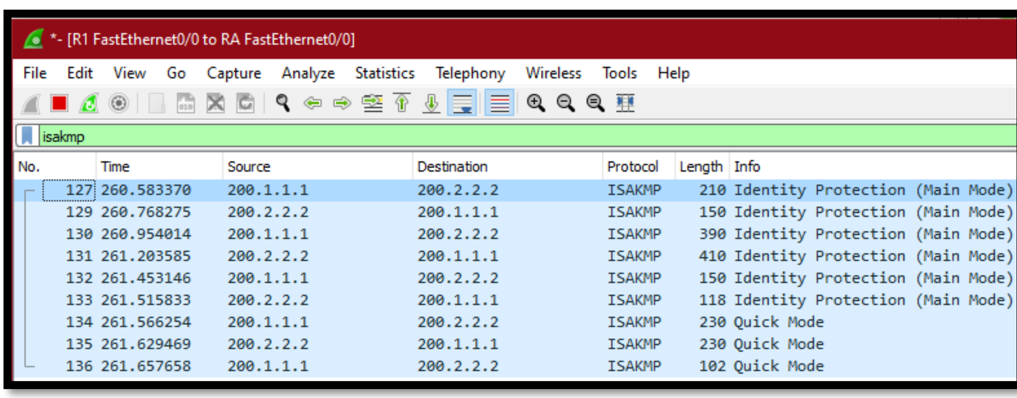


Figure 1 - IPSec using ESP tunnel mode Network Architecture

Regarding the IPSec configuration, we followed the configuration provided for R1 and adapted it for R2, changing the pre-shared key IP, the peer, and the access list. To better analyse the establishment of the IPSec tunnel, we configured a Wireshark capture between R1 and RA.

Proof-of-Concept:

1. The Tunnel establishment involves **IKE Phase 1** and **IKE Phase 2**. Both IKE phases use the **ISAKMP (Internet Security Association and Key Management Protocol)**. IKE Phase 1 consists of six messages in **Main Mode**, and IKE Phase 2 consists of 3 messages in **Quick Mode**, as shown in Figure 2.



No.	Time	Source	Destination	Protocol	Length	Info
127	260.583370	200.1.1.1	200.2.2.2	ISAKMP	210	Identity Protection (Main Mode)
129	260.768275	200.2.2.2	200.1.1.1	ISAKMP	150	Identity Protection (Main Mode)
130	260.954014	200.1.1.1	200.2.2.2	ISAKMP	390	Identity Protection (Main Mode)
131	261.203585	200.2.2.2	200.1.1.1	ISAKMP	410	Identity Protection (Main Mode)
132	261.453146	200.1.1.1	200.2.2.2	ISAKMP	150	Identity Protection (Main Mode)
133	261.515833	200.2.2.2	200.1.1.1	ISAKMP	118	Identity Protection (Main Mode)
134	261.566254	200.1.1.1	200.2.2.2	ISAKMP	230	Quick Mode
135	261.629469	200.2.2.2	200.1.1.1	ISAKMP	230	Quick Mode
136	261.657658	200.1.1.1	200.2.2.2	ISAKMP	102	Quick Mode

Figure 2 - ISAKMP messages

2. In Figure 3, we can see the first message exchange between peers. This message contains the cryptography algorithms supported by R1 to start the **Security Association (SA)** negotiation with R2. In the packet information identified by "**IKE Attribute**", we can observe the protocols to be used.

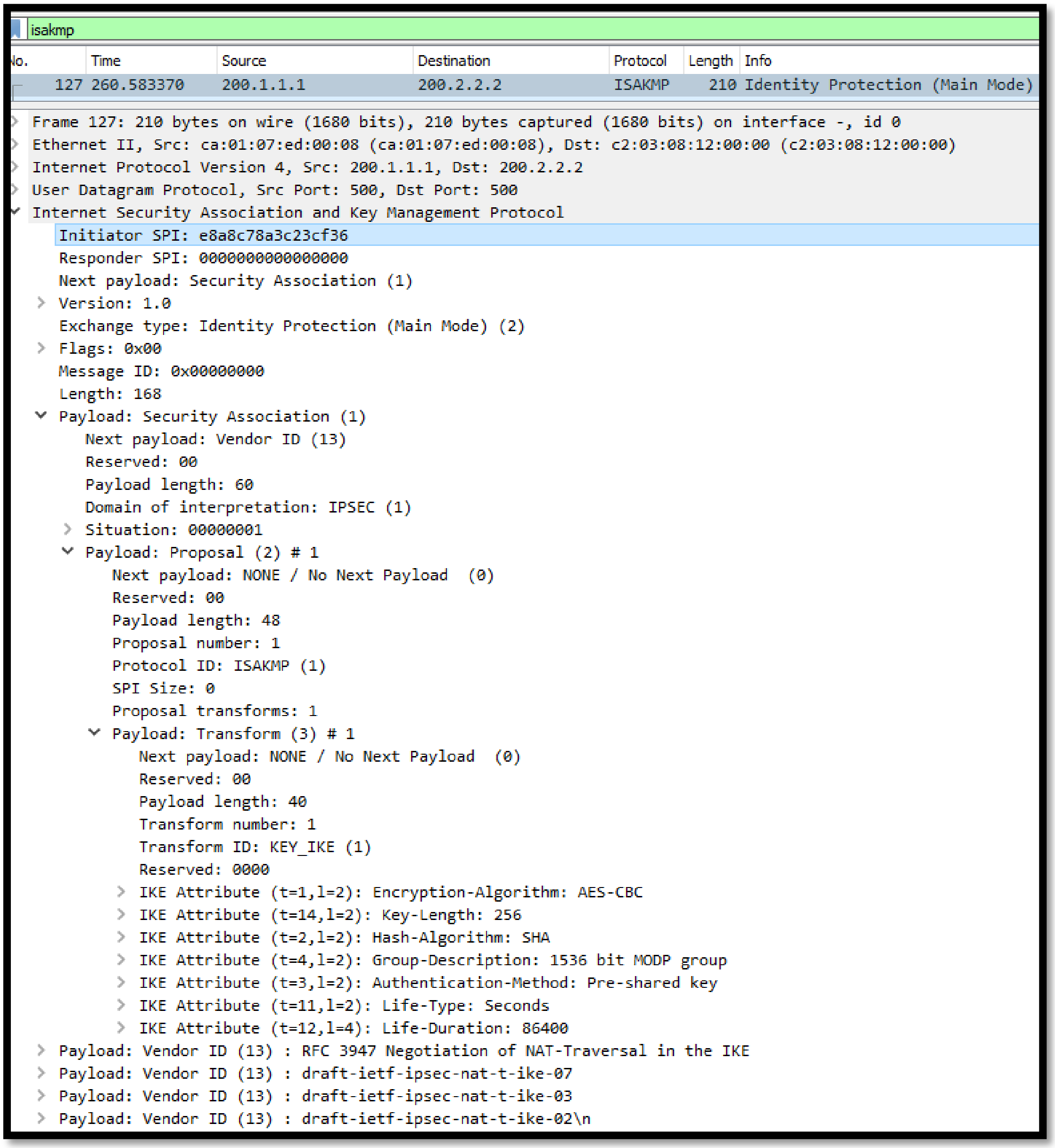


Figure 3 - First ISAKMP message

3. In Figure 4, we have the response from R2. In this packet, we have the confirmation from R2 of the cryptography algorithms to be used in the **SA.**

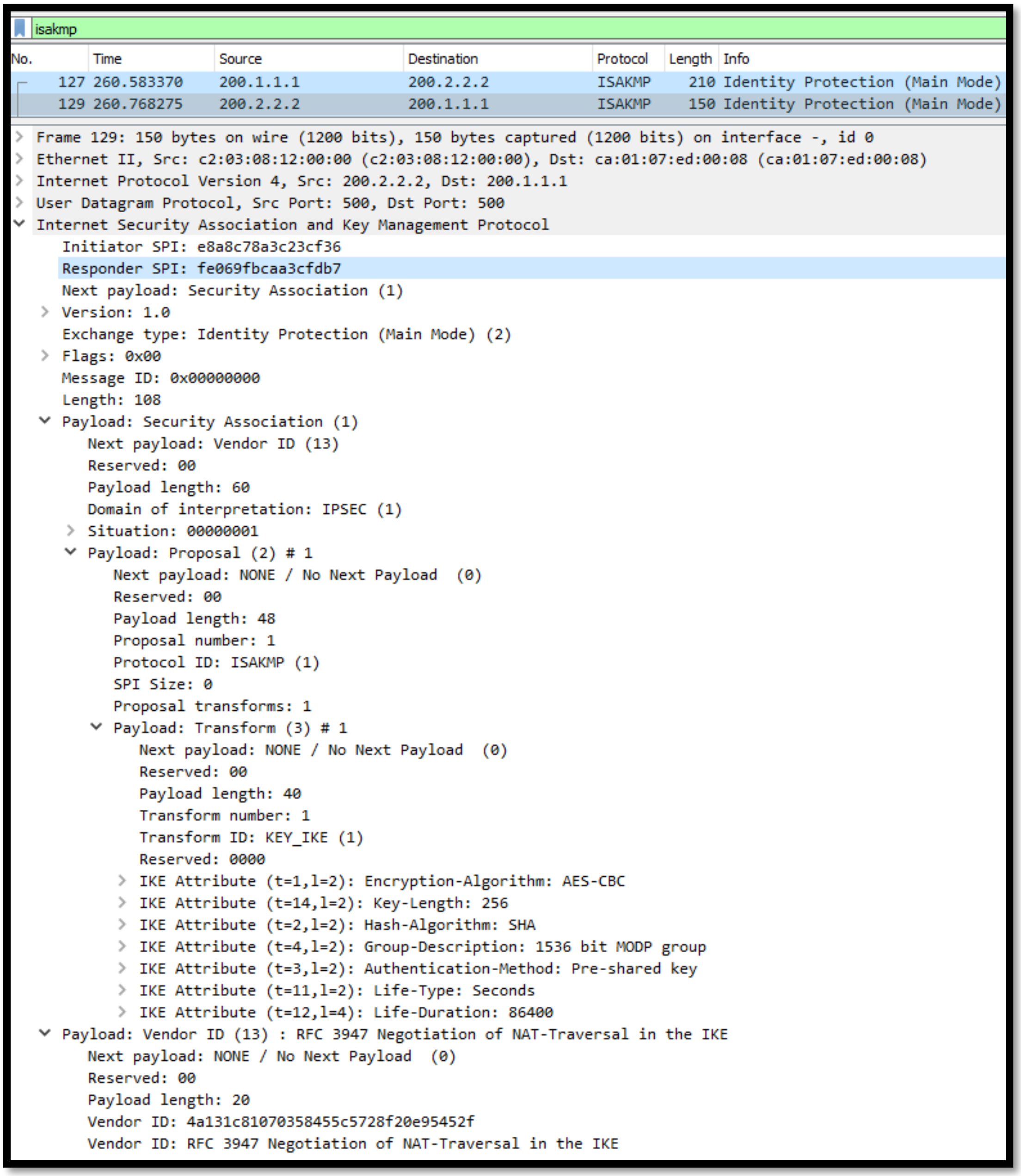


Figure 4 - ISAKMP response from R2

4. The first phase is now completed. Secondly, both parts must compute the shared key using the **Diffie-Helman (DH)** algorithm based on the **DH** group previously agreed (DH group 5 in this case). DH allows both peers to agree on a shared key. We can see this process in the packets in Figure 5 and Figure 6.

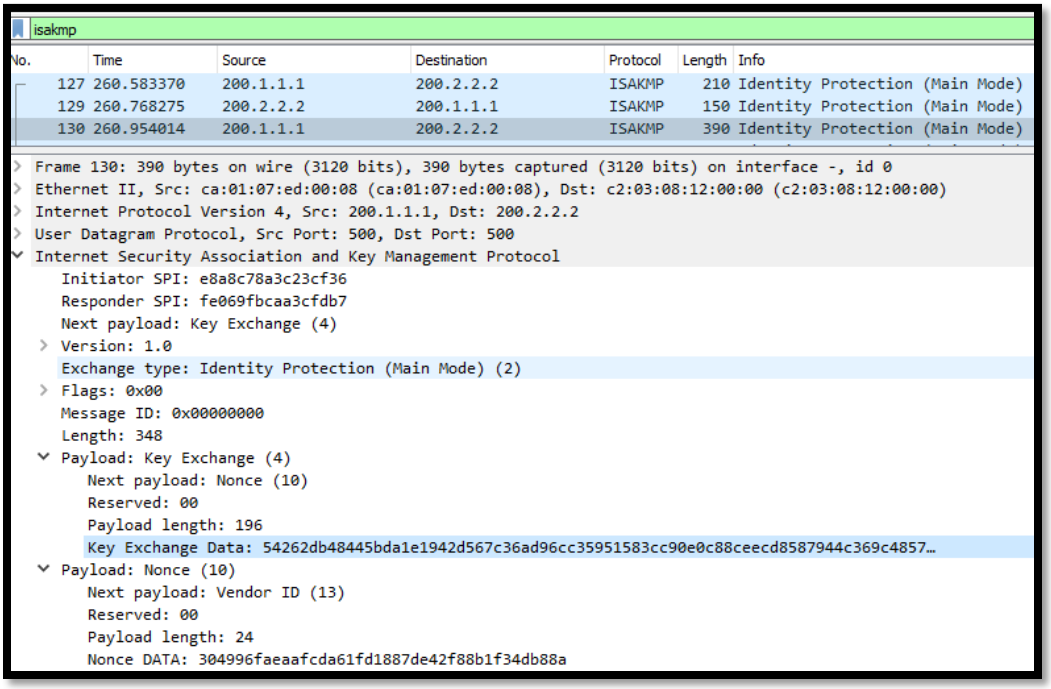


Figure 5 - Key exchange message 3

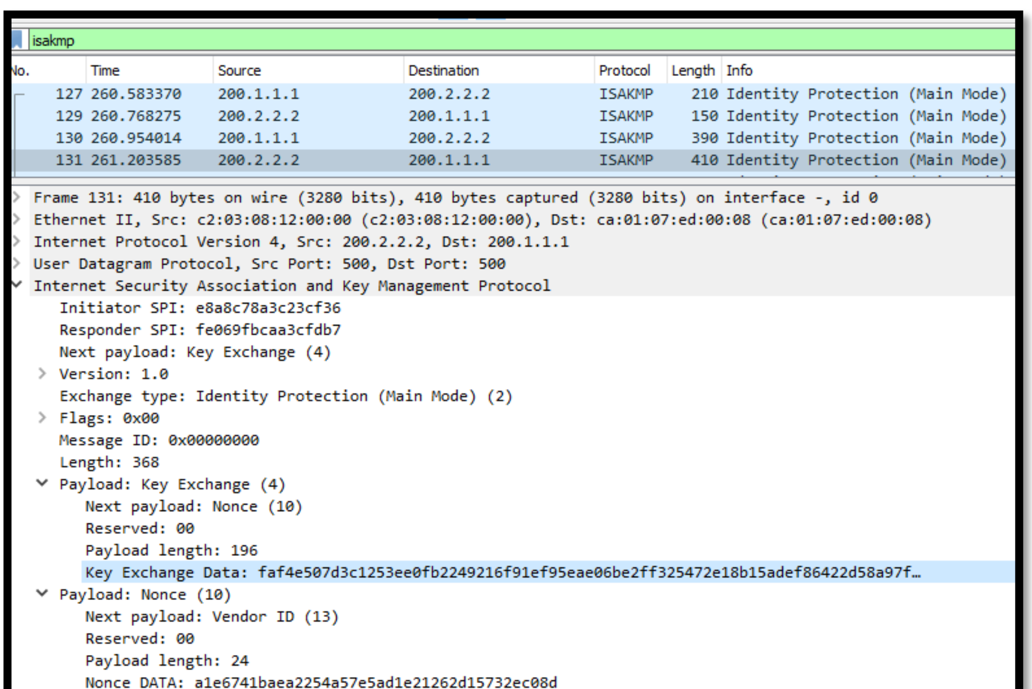


Figure 6 - Key exchange message 4

5. In both packets, the payload Key Exchange contains the value computed from each peer that will be used later in the DH process by the other end to compute the Shared Key. Some literature calls the Diffie-Hellman algorithm a Key Exchange protocol. We can see that this can be, in a way, incoherent.
   The DH protocol is a Public Key Generation Protocol, where each party shares a parameter with the other end for both ends to compute a shared secret from those parameters. What is shared is merely a component that generates the shared key, not the key itself.

6. We have now finished the second step of IKE Phase 1.
   As the last of IKE Phase 1, we have authentication of peers using the agreed authentication method. This is done by exchanging hash values between peers, as seen in Figure 7.

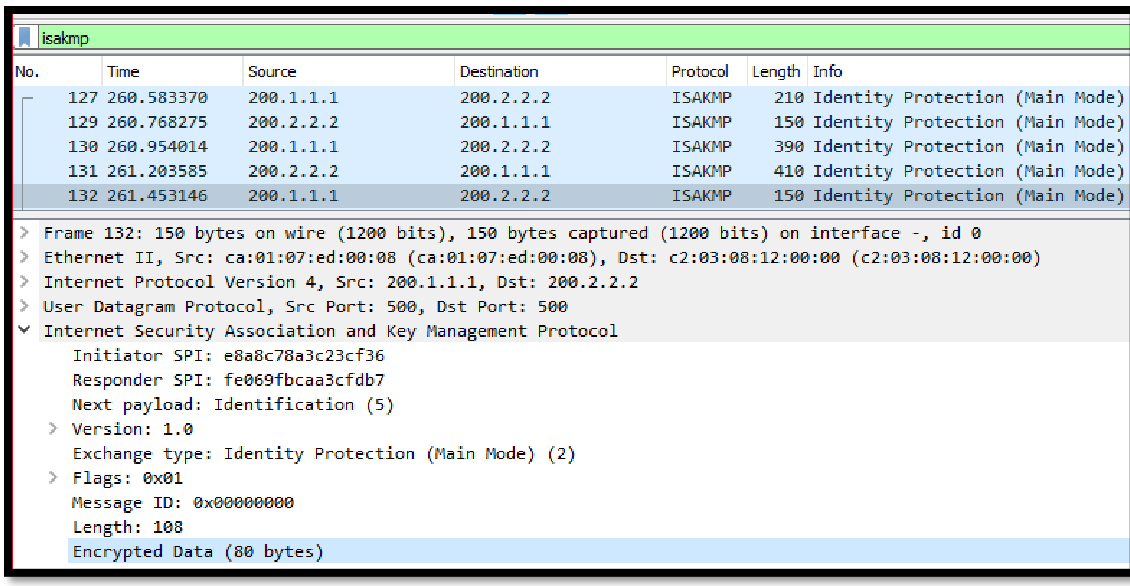
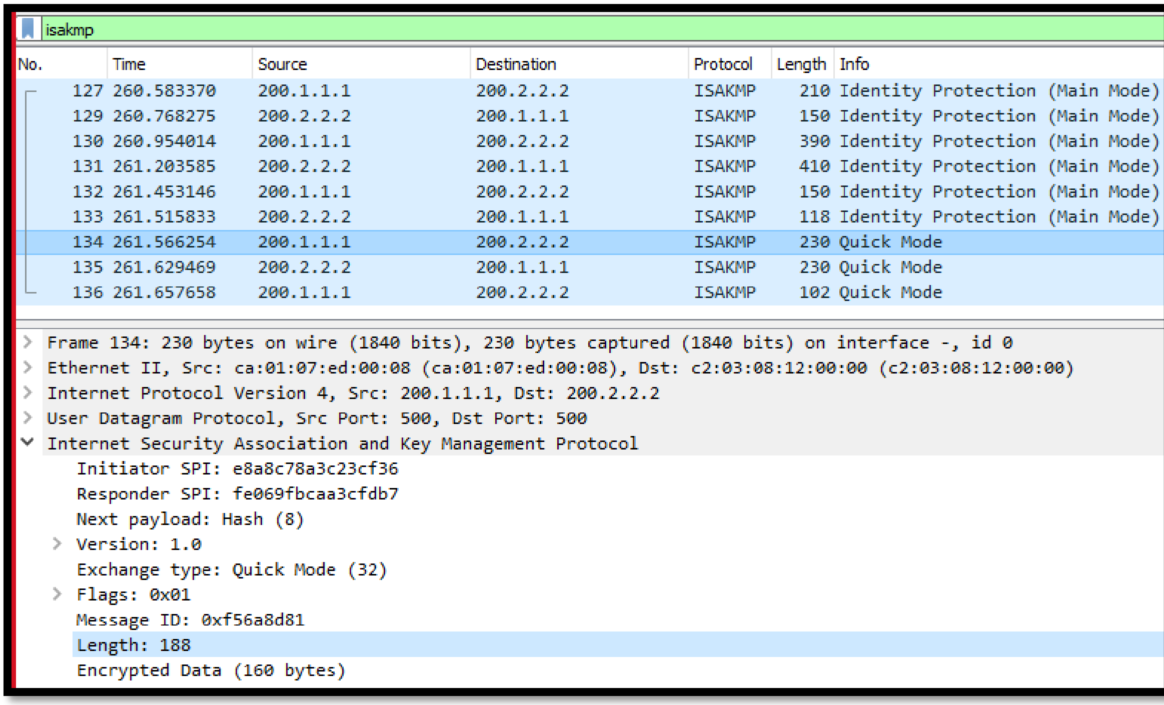


Figure 7 - ISAKMP Authentication of peers

7. Moving on to **IKE Phase 2**, the tunnel created is used to protect data. We have three messages in this phase, as shown in Figure 8.



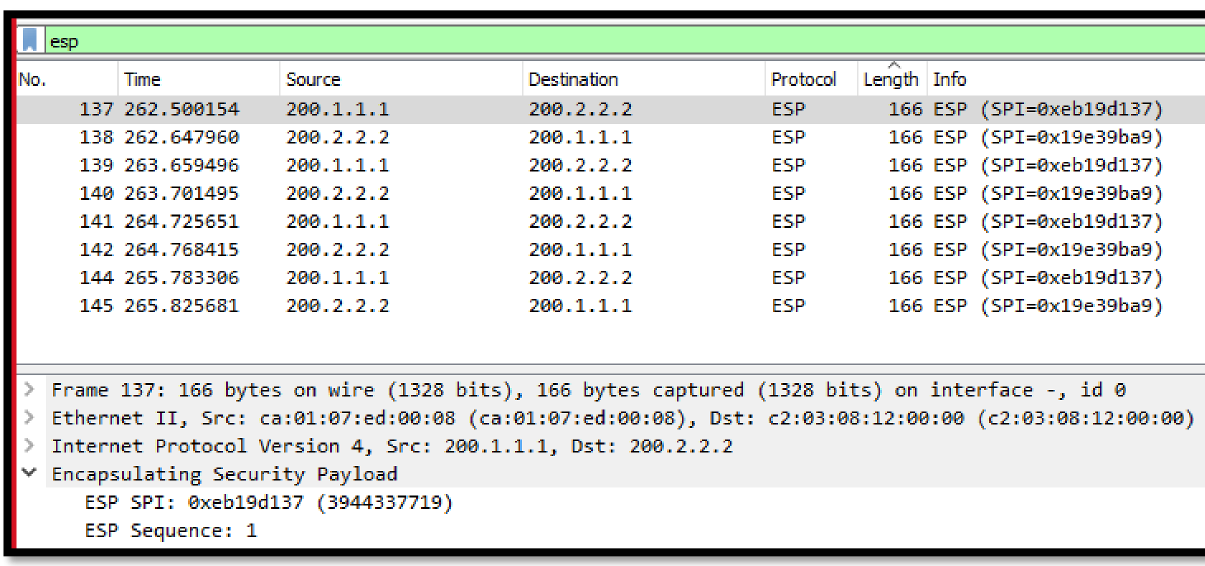
isakmp

No.	Time	Source	Destination	Protocol	Length	Info
127	260.583370	200.1.1.1	200.2.2.2	ISAKMP	210	Identity Protection (Main Mode)
129	260.768275	200.2.2.2	200.1.1.1	ISAKMP	150	Identity Protection (Main Mode)
130	260.954014	200.1.1.1	200.2.2.2	ISAKMP	390	Identity Protection (Main Mode)
131	261.203585	200.2.2.2	200.1.1.1	ISAKMP	410	Identity Protection (Main Mode)
132	261.453146	200.1.1.1	200.2.2.2	ISAKMP	150	Identity Protection (Main Mode)
133	261.515833	200.2.2.2	200.1.1.1	ISAKMP	118	Identity Protection (Main Mode)
134	261.566254	200.1.1.1	200.2.2.2	ISAKMP	230	Quick Mode
135	261.629469	200.2.2.2	200.1.1.1	ISAKMP	230	Quick Mode
136	261.657658	200.1.1.1	200.2.2.2	ISAKMP	102	Quick Mode

```
> Frame 134: 230 bytes on wire (1840 bits), 230 bytes captured (1840 bits) on interface -, id 0
> Ethernet II, Src: ca:01:07:ed:00:08 (ca:01:07:ed:00:08), Dst: c2:03:08:12:00:00 (c2:03:08:12:00:00)
> Internet Protocol Version 4, Src: 200.1.1.1, Dst: 200.2.2.2
> User Datagram Protocol, Src Port: 500, Dst Port: 500
v Internet Security Association and Key Management Protocol
    Initiator SPI: e8a8c78a3c23cf36
    Responder SPI: fe069fbcaa3cfdb7
    Next payload: Hash (8)
  > Version: 1.0
    Exchange type: Quick Mode (32)
  > Flags: 0x01
    Message ID: 0xf56a8d81
    Length: 188
    Encrypted Data (160 bytes)
```

Figure 8 - Three quick mode messages

8. After both phases are completed, it is expected to see the packets exchanged between PC1 and PC2 using **the ESP** protocol. We can verify this in Figure 9, where we ping PC2 from PC1.



esp

No.	Time	Source	Destination	Protocol	Length	Info
137	262.500154	200.1.1.1	200.2.2.2	ESP	166	ESP (SPI=0xeb19d137)
138	262.647960	200.2.2.2	200.1.1.1	ESP	166	ESP (SPI=0x19e39ba9)
139	263.659496	200.1.1.1	200.2.2.2	ESP	166	ESP (SPI=0xeb19d137)
140	263.701495	200.2.2.2	200.1.1.1	ESP	166	ESP (SPI=0x19e39ba9)
141	264.725651	200.1.1.1	200.2.2.2	ESP	166	ESP (SPI=0xeb19d137)
142	264.768415	200.2.2.2	200.1.1.1	ESP	166	ESP (SPI=0x19e39ba9)
144	265.783306	200.1.1.1	200.2.2.2	ESP	166	ESP (SPI=0xeb19d137)
145	265.825681	200.2.2.2	200.1.1.1	ESP	166	ESP (SPI=0x19e39ba9)

```
> Frame 137: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface -, id 0
> Ethernet II, Src: ca:01:07:ed:00:08 (ca:01:07:ed:00:08), Dst: c2:03:08:12:00:00 (c2:03:08:12:00:00)
> Internet Protocol Version 4, Src: 200.1.1.1, Dst: 200.2.2.2
v Encapsulating Security Payload
    ESP SPI: 0xeb19d137 (3944337719)
    ESP Sequence: 1
```

Figure 9 - ESP pings from PC2 to PC1

9. Since the traffic is cyphered with the shared key established in IKE Phase 1, we cannot recognise the ICMP packets originating from the ping. We can see that the source and destination of the packets correspond to the routers at the tunnel's beginning and end.

Additionally, we can see that a new IP header was added by observing the payload. These are evidence that the tunnel is using ESP in tunnel mode.

10. Using the different ISAKMP policies presented below, at R2 (encryption type DES), we concluded that the ISAKMP SA negotiation with R2 has failed, thanks to a mismatch of the proposed encryption protocols. In this case, a NO-PROPOSAL-CHOSEN message is sent, as shown in Figure 10.

```
#Different Policy
#IKE Phase 1 - Configure ISAKMP policy
crypto isakmp policy 10
hash md5
authentication pre-share
group 5
lifetime 86400
encryption des
```

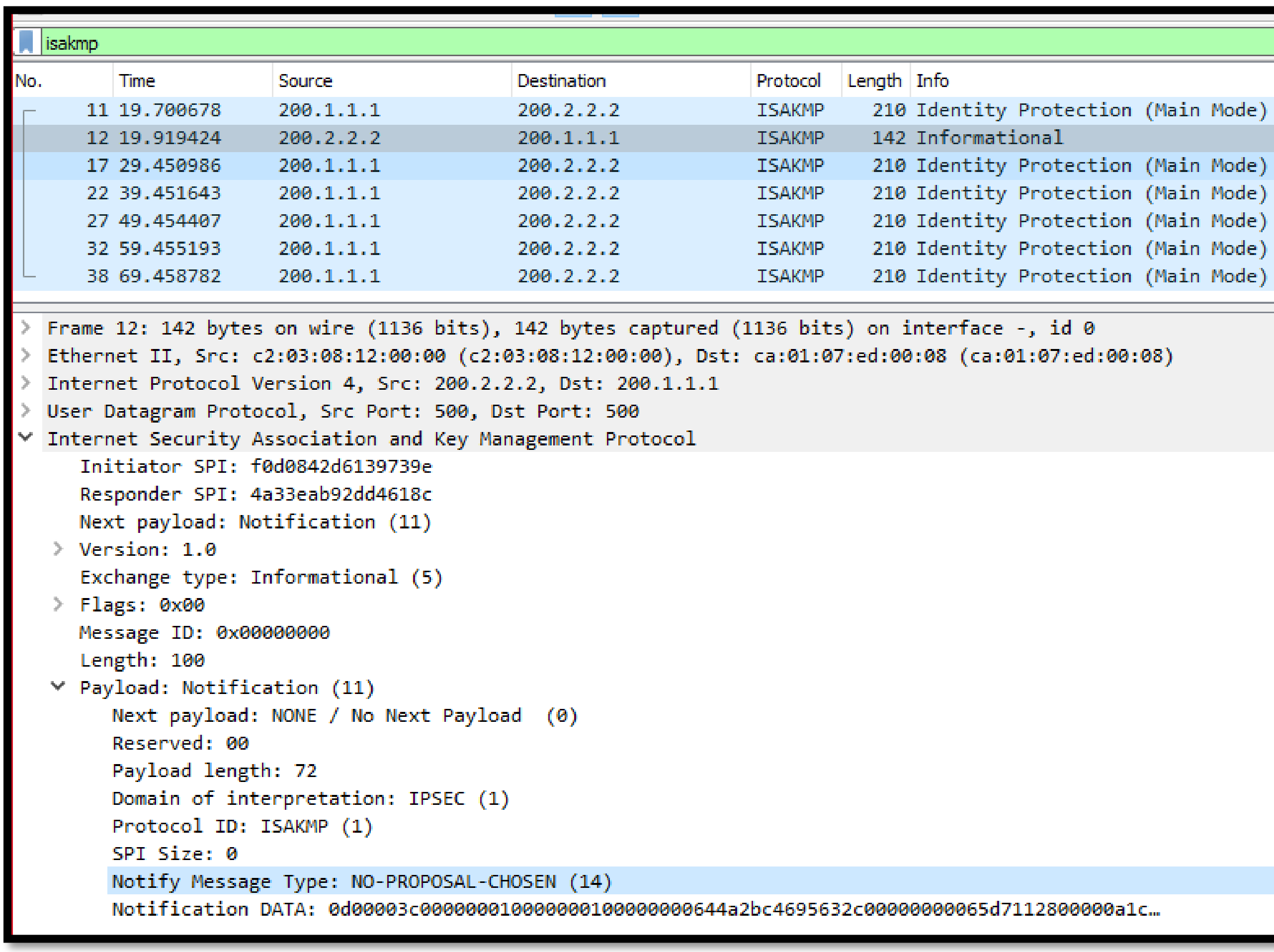


Figure 10 - Informational ISAKMP message NO-PROPOSAL-CHOSEN

1.2. IPSec with digital certificates and certificate authority

In this exercise, we configure a Certification Authority (CA) and IPSec using digital certificates. Authentication in IPSec can be provided through pre-shared keys or digital certificates, which requires a PKI infrastructure, in this case, a CA Server, trusted by both parties. Although more complex, using digital certificates provides a higher level of scalability.

These certificates are tamper-proof and cannot be forged. The most widely used format for digital certificates is X.509. The complete configuration can be found in Annex A.

Network Architecture:

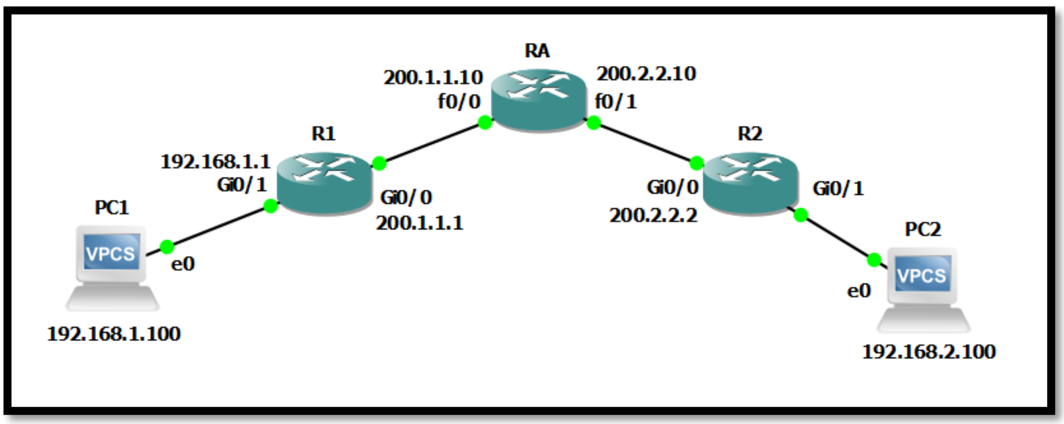


Figure 11 - IPSec with digital certificates Network Architecture

This architecture applies two IOSv routers, R1 and R2, and a 7200 router as RA, which serves as the CA.

Proof-of-Concept:

1. After assigning all the IP addresses specified in Figure 11 and configuring OSPF with the respective static routes, we configure the CA at router RA. The information on the PKI server can be seen in Figure 12. In Figure 13, we display the information of the CA certificate, and in Figure 14, we can see the public key created for the CA.

```
RA#show crypto pki server
Certificate Server myCA:
    Status: enabled
    State: enabled
    Server's configuration is locked  (enter "shut" to unlock it)
    Issuer name: cn=saarCA
    CA cert fingerprint: 07C75185 EBD04A45 A413EA43 A290747C
    Granting mode is: auto
    Last certificate issued serial number (hex): 1
    CA certificate expiration timer: 17:28:28 UTC Jun 5 2025
    CRL NextUpdate timer: 23:28:28 UTC Jun 6 2022
    Current primary storage dir: nvram:
    Database Level: Minimum - no cert data written to storage
```

Figure 12 - Information of the PKI server

```
RA#show crypto ca certificate
CA Certificate
  Status: Available
  Certificate Serial Number (hex): 01
  Certificate Usage: Signature
  Issuer:
    cn=saarCA
  Subject:
    cn=saarCA
  Validity Date:
    start date: 17:28:28 UTC Jun 6 2022
    end   date: 17:28:28 UTC Jun 5 2025
  Associated Trustpoints: myCA
```

Figure 13 - CA certificate Information

```
RA#show crypto key mypub rsa
% Key pair was generated at: 17:28:28 UTC Jun 6 2022
Key name: myCA
Key type: RSA KEYS
 Storage Device: not specified
 Usage: General Purpose Key
 Key is not exportable.
 Key Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00B2CB6A
  264C6729 2A9DB5FD 12A31B86 41C4FDD0 0F9513CE 1F523173 83DB3FDB C4C9DB43
  6BEF1B7A 21E8210F 0667A296 D50403EF 4E62B037 0EFAE4D9 FBD74469 F70C1226
  FF0D0242 E5A5B078 A86FE61B 6BCE44D3 435F26C6 8DCE99C8 42F614FF 70D3E705
  2B750669 DF2C3D24 EBA9FE2C A214694C 8814E9E3 8F911369 AE62742E 07020301 0001
% Key pair was generated at: 17:28:29 UTC Jun 6 2022
Key name: myCA.server
Key type: RSA KEYS
Temporary key
 Usage: Encryption Key
 Key is not exportable.
 Key Data:
  307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00AE5D46 1DC51415
  A3E4AD9E BAD21CE0 DF449F71 A286175E 30EF7CA6 D3A5EF84 031E1998 D636EC69
  CDABA73C 5E9C3B59 34698AA5 F1C8FC07 30A1ADE4 4EEF43AA 9292A5F3 30347123
  756E774E FD09D25E B8B96E54 CE65AEDC 62F3C54E 2271953B BB020301 0001
RA#
```

Figure 14 - Public key created for the CA

2. We now configure R1 and R2 as the PKI clients who obtain a CA certificate. We used a (Simple Certificate Enrolment Protocol) SCEP-based enrolment. The exchange of HTTP messages related to SCEP can be seen in Figure 15. We can also confirm that the certificate was installed at R1 in Figure 17 and see R1's public key in Figure 16.

http

No.	Time	Source	Destination	Protocol	Length	Info
42	84.665389	200.1.1.1	200.1.1.10	HTTP	211	GET /cgi-bin/pkiclient.exe?operation=GetCACert&message=myTrustpoint HTTP/1.0
45	84.729123	200.1.1.10	200.1.1.1	HTTP	603	HTTP/1.1 200 OK (application/x-x509-ca-cert)
52	84.775835	200.1.1.1	200.1.1.10	HTTP	211	GET /cgi-bin/pkiclient.exe?operation=GetCACaps&message=myTrustpoint HTTP/1.0
55	84.791110	200.1.1.10	200.1.1.1	HTTP	112	HTTP/1.1 200 OK (application/x-pki-message)
81	119.343694	200.1.1.1	200.1.1.10	HTTP	901	GET /cgi-bin/pkiclient.exe?operation=PKIOperation&message=MIIF8gYJKoZIhvcNAQcCoI
86	119.571840	200.1.1.10	200.1.1.1	HTTP	166	Continuation

Figure 15 - SCEP HTTP messages between R1 and RA

```
Certificate
  Status: Available
  Certificate Serial Number (hex): 02
  Certificate Usage: General Purpose
  Issuer:
    cn=saarCA
  Subject:
    Name: Router
    Serial Number: 9VILXPB82TVPLFP1BDZQB
    hostname=Router+serialNumber=9VILXPB82TVPLFP1BDZQB
  Validity Date:
    start date: 17:40:39 UTC Jun 6 2022
    end   date: 17:40:39 UTC Jun 6 2023
  Associated Trustpoints: myTrustpoint

CA Certificate
  Status: Available
  Certificate Serial Number (hex): 01
  Certificate Usage: Signature
  Issuer:
    cn=saarCA
  Subject:
    cn=saarCA
  Validity Date:
    start date: 17:28:28 UTC Jun 6 2022
    end   date: 17:28:28 UTC Jun 5 2025
  Associated Trustpoints: myTrustpoint
```

Figure 17 - Certificate installed at R1

```
Router(config)#do show crypto key mypub rsa
% Key pair was generated at: 17:39:58 UTC Jun 6 2022
Key name: Router
Key type: RSA KEYS
 Storage Device: not specified
 Usage: General Purpose Key
 Key is not exportable.
 Key Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00BD495D
  D084C1BD 1F4A8972 232054B8 9666B653 D00E702B E7CF00FB D6942C79 E621DDCE
  3B80D0A0 D4C2220D 05532DAB 440C8277 27B6CD32 381B8CDB F79BDCC7 B54A1DF2
  FB5E5A4E B0500FC6 157D9F82 676D0DF8 840A5D5A 2F2CC115 1CF86E6E 02031527
  9F8E6CDA 5477F4FA 476D2D0E 4505EB52 0D9C3BBD 45B214B1 8BE6EEBE BF020301
  0001
% Key pair was generated at: 17:40:00 UTC Jun 6 2022
Key name: Router.server
Key type: RSA KEYS
Temporary key
 Usage: Encryption Key
 Key is not exportable.
 Key Data:
  307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00AE125B 3DC8E36A
  8B74ED13 B2D2A788 D4DB2D07 F0A94B2C 85F79FBA 96EA1CAD 82FA83A2 5954F5E0
  A9D11280 CC607B28 9EA41364 919E7265 08CD7610 75E7FB1F D14F8484 206431F4
  F53393BB 155BEE64 94E6D448 3B5A97EB 900DA8FE ADF85D93 F9020301 0001
```

Figure 16 - R1's Public Key

3. Using Wireshark, we can identify the HTTP message that requests the certificate in Figure 18 and the message that carries the requested certificate in Figure 19. We also extracted the certificate to analyse better its contents presented in Certificate General Info and Figure 21.

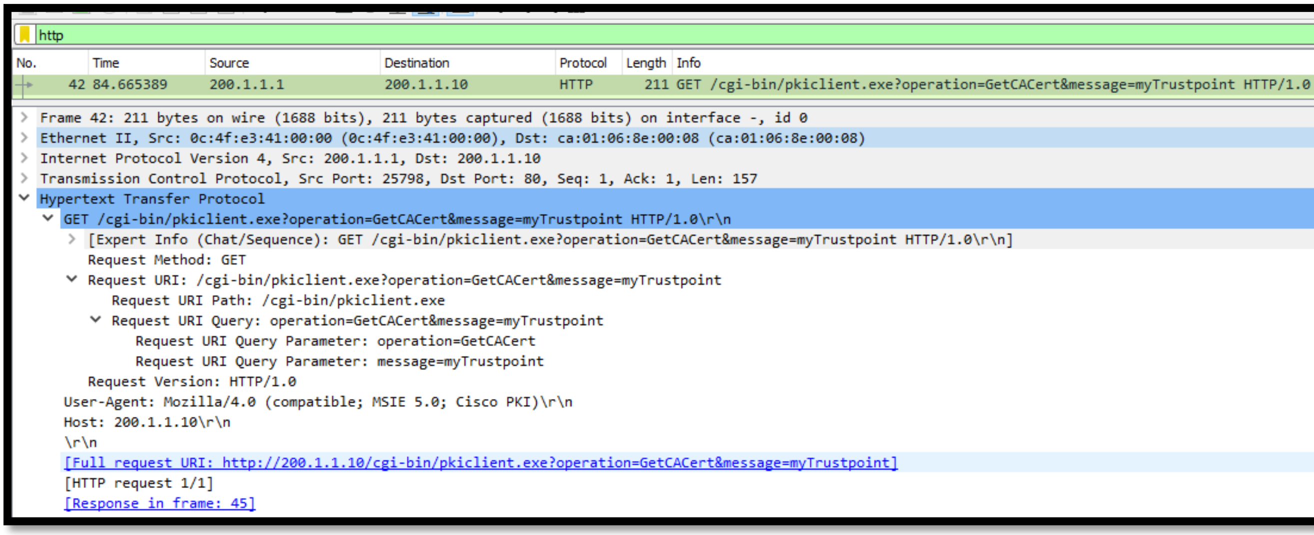


Figure 18 - GET certificate HTTP message

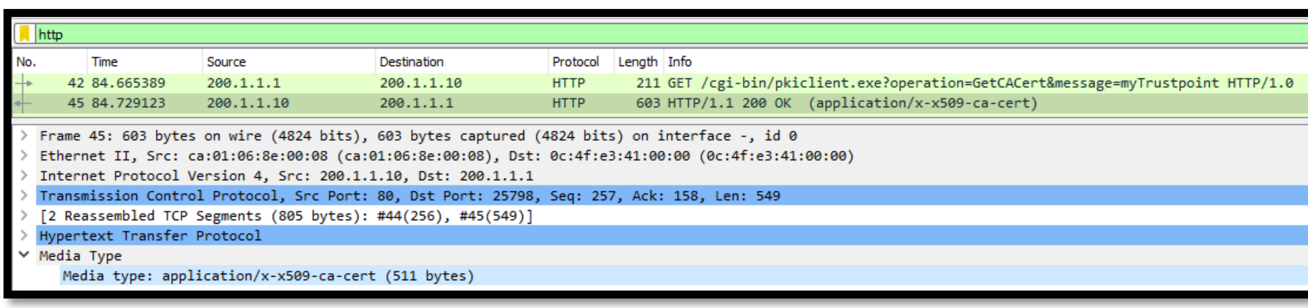


Figure 19 - HTTP OK message containing the CA certificate

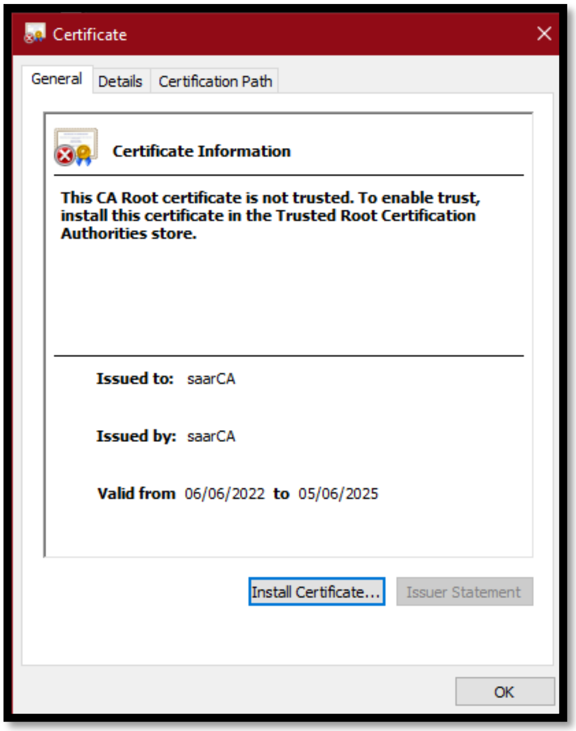


Figure 21 - Certificate General Info

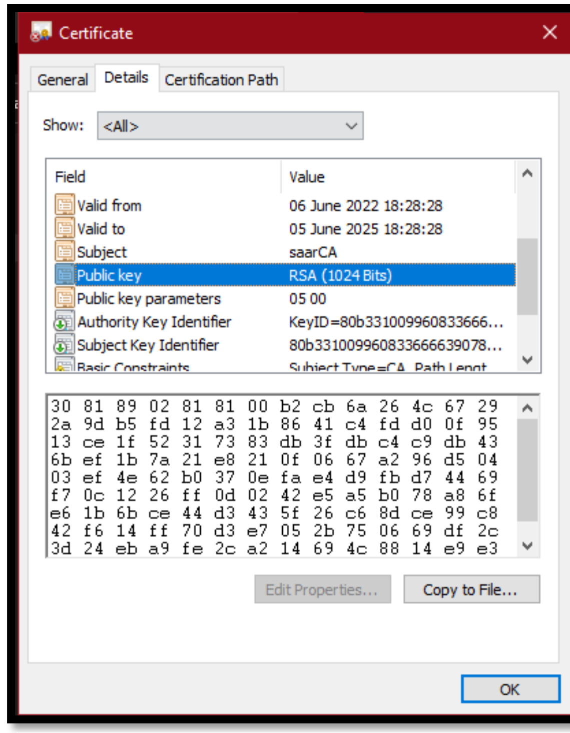


Figure 20 - Certificate Details

4. Using an RSA signature authentication method, we now configure IPSec with Public Signature Keys using crypto maps at R1 and R2. This allows PC1 to communicate with PC2, as shown in Figure 22. In Figure 23, we can see that the main difference concerning IPSec with pre-shared keys is the authentication method type, which in this case is using RSA signatures in Digital certificates.

```
PC1> ping 192.168.2.100

192.168.2.100 icmp_seq=1 timeout
84 bytes from 192.168.2.100 icmp_seq=2 ttl=62 time=35.739 ms
84 bytes from 192.168.2.100 icmp_seq=3 ttl=62 time=21.390 ms
84 bytes from 192.168.2.100 icmp_seq=4 ttl=62 time=22.447 ms
84 bytes from 192.168.2.100 icmp_seq=5 ttl=62 time=21.566 ms
```

Figure 22 - Ping from PC1 to PC2

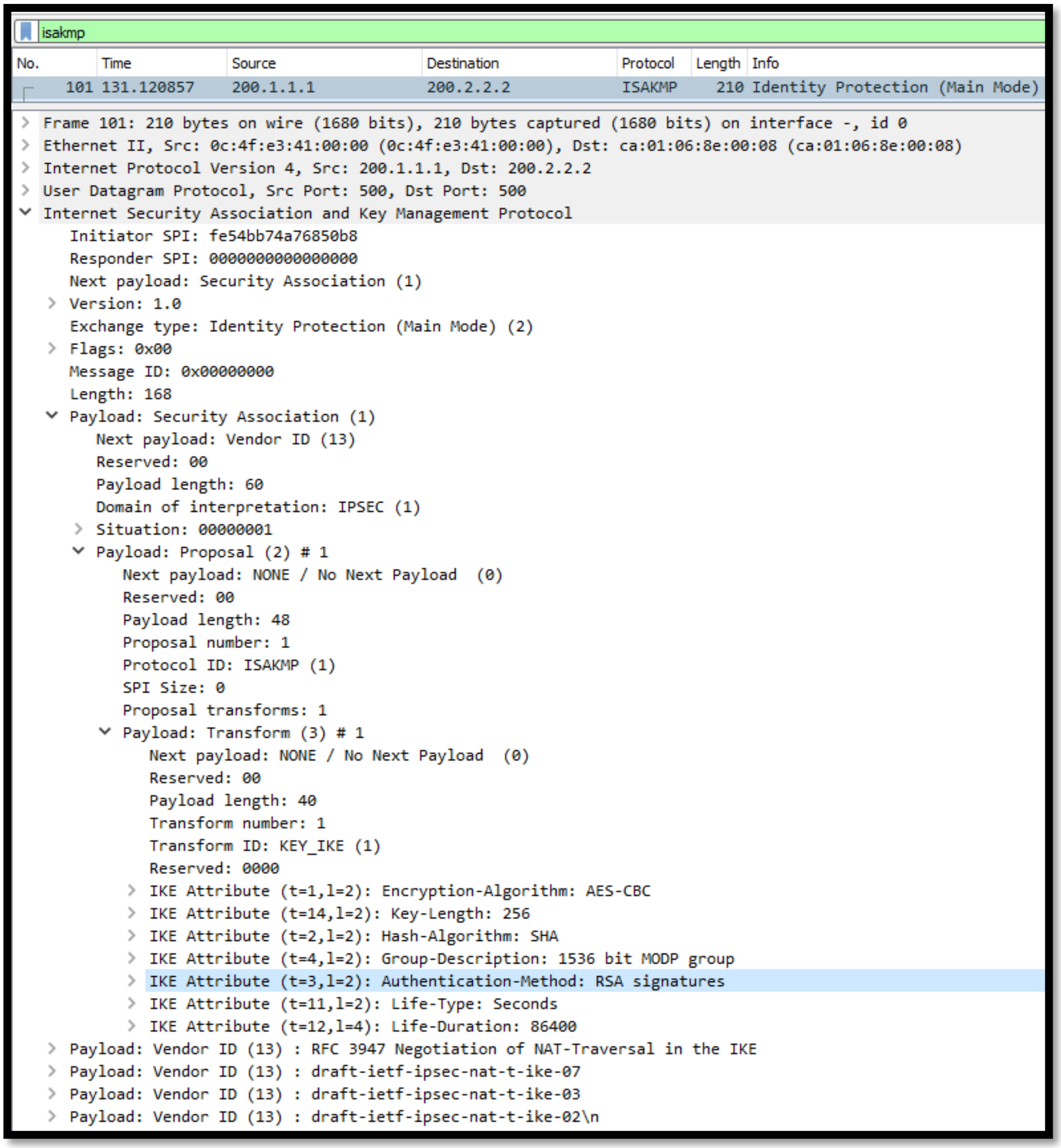


Figure 23 - IKE authentication method using Digital certificates

5. Now that the configurations are complete and we managed to communicate from PC1 to PC2, we can see that each router stores the public key of the other in Figure 24, which shows the public key of R2 at R1. We can also see in Figure 25 that ICMP messages exchanged between R1 and R2 cannot be identified since they are cyphered.

```
Router#show crypto key pubkey-chain rsa name Router
 Key name: Router
 Subject name(X.500 DN name): hostname=Router+serialNumber=93FVZFKVS57NP31AMCWPP
 Key id: 11
 Serial number: 03
 Usage: Signature Key
 Source: Certificate
 Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00A8CCAF
  2BE215D7 19999D64 CBCD13A4 BD95FC72 5CC30544 8ECAC918 58433856 E8F0ACD7
  8A546821 282A35D5 DFE74ED4 F43966FF 98ACC1A2 62EAA82C ACBC7C57 B2CA45B7
  C3CDA18A 682DE2A6 860072AB 1E06C02D 929239AD 85A19A50 E6AC0BFD 4898228C
  96CE484C FDA8F5DE 9A8ED796 03E0F950 AAA5ADD2 444AB00F CB0FF65F 59020301
  0001
```

Figure 24 - R2 Public key at stored in R1

esp

No.	Time	Source	Destination	Protocol	Length	Info
161	212.512533	200.1.1.1	200.2.2.2	ESP	166	ESP (SPI=0x1eb0f314)
162	212.550642	200.2.2.2	200.1.1.1	ESP	166	ESP (SPI=0x12153503)
164	213.575083	200.1.1.1	200.2.2.2	ESP	166	ESP (SPI=0x1eb0f314)
165	213.595133	200.2.2.2	200.1.1.1	ESP	166	ESP (SPI=0x12153503)
166	214.602250	200.1.1.1	200.2.2.2	ESP	166	ESP (SPI=0x1eb0f314)
167	214.616926	200.2.2.2	200.1.1.1	ESP	166	ESP (SPI=0x12153503)

```
> Frame 161: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface -, id 0
> Ethernet II, Src: 0c:4f:e3:41:00:00 (0c:4f:e3:41:00:00), Dst: ca:01:06:8e:00:08 (ca:01:06:8e:00:08)
> Internet Protocol Version 4, Src: 200.1.1.1, Dst: 200.2.2.2
v Encapsulating Security Payload
    ESP SPI: 0x1eb0f314 (514913044)
    ESP Sequence: 1
```

Figure 25 – Cyphered ICMP messages between R1 and R2

1.3. IPSec with NAT traversal

In this exercise, we test the IPSec tunnel with NAT Traversal. After the address configurations, we have configured **OSPF** as the routing protocol on the left side of the organisation and static routing between the two organisations. The complete configuration can be found in Annex A.

Network Architecture:

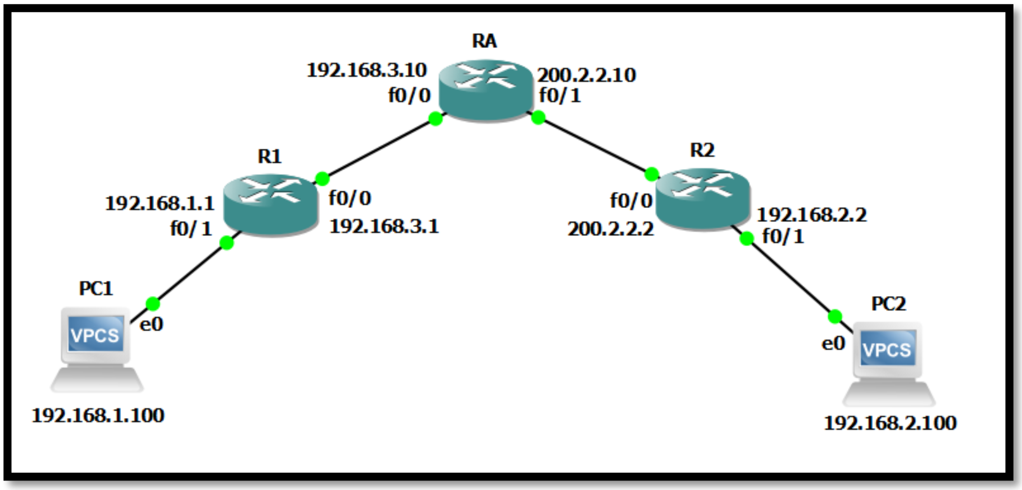


Figure 26 - IPSec with NAT traversal Network Architecture

In this architecture, the organisation's left side now includes two subnets: 192.168.1.0/24 and 192.168.3.0/24. The IPSec tunnel is still between R1 and R2, but RA is now the border router and includes PAT (i.e., NAT overload).

Proof-of-Concept:

Regarding the IPSec configuration, we followed the configuration provided for R2 and adapted it for R1, changing the pre-shared key IP and its peer. To better analyse the establishment of the IPSec tunnel, we configured a Wireshark capture between R1 and RA and between RA and R2.

1. We can see in Figure 27 the packet capture between R1 and RA (Private link). Here we can observe that the source is the IP **192.168.3.1** previously configured in the interface of R1. On the other hand, in Figure 28 (Public link), we can see that the source IP is **200.2.2.10**. Since RA has NAT configured, packets from the private network on the left will be translated to the public address **200.2.2.10**.

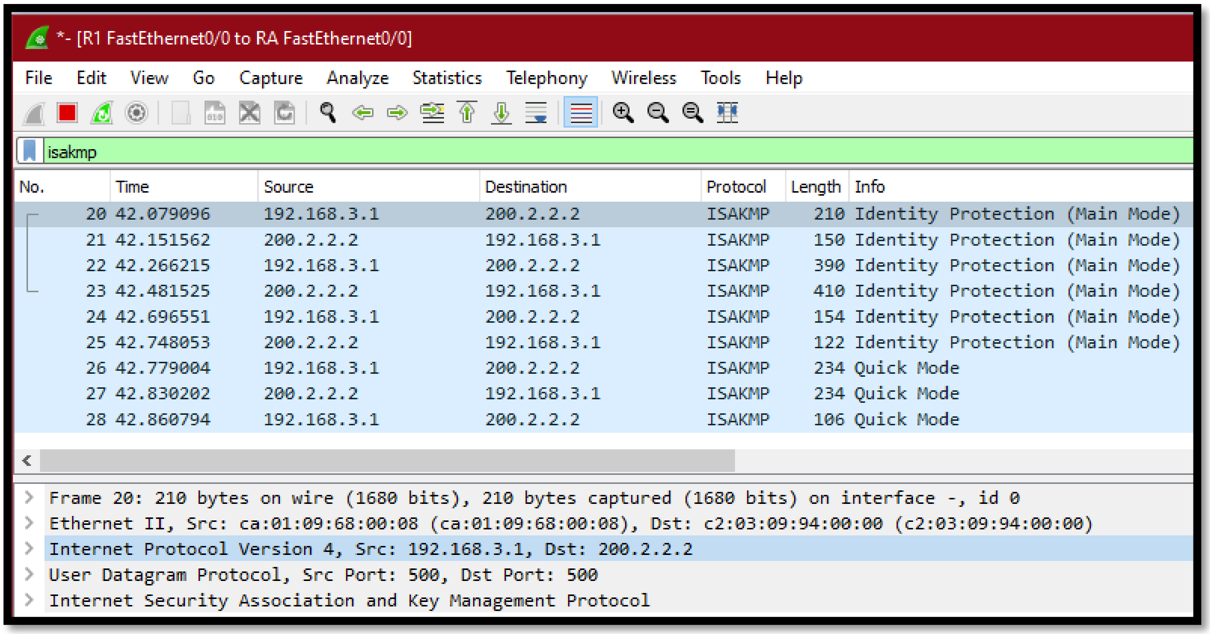


Figure 27 - Packet capture between R1 and RA (Private link)

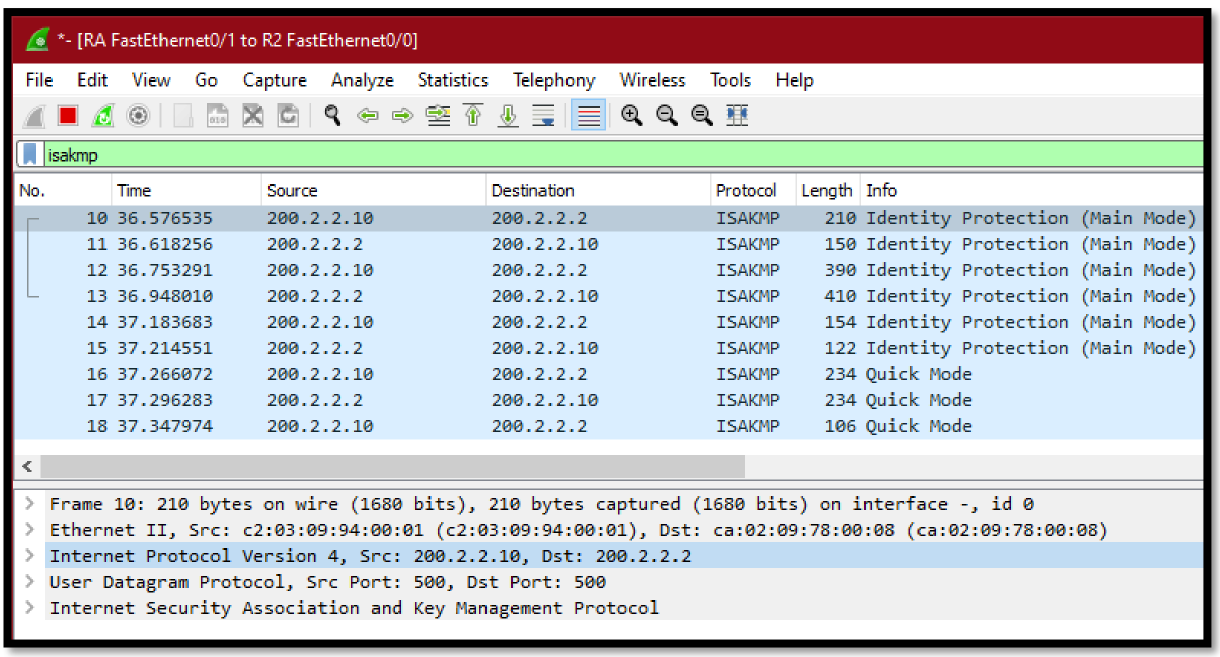


Figure 28 - Packet capture on the public link

2. We have the third and fourth IKE messages in Figure 29 and Figure 30, respectively. We can observe that both packets contain two **NAT-D Payloads**. Furthermore, we can see that these payloads have a Hash value. The first uses the source IP and port for the hash, and the second one uses the destination IP and source. The peer will use these values to verify if NAT was used. If the hash values match, then NAT was not used; if the hashes do not match, then NAT was indeed employed.
   In this situation, the second payload will not match the one calculated by the other end, as the packet received by this peer has a different source IP than the one previously used for the hash. This way, the peer will notice that NAT is being used.

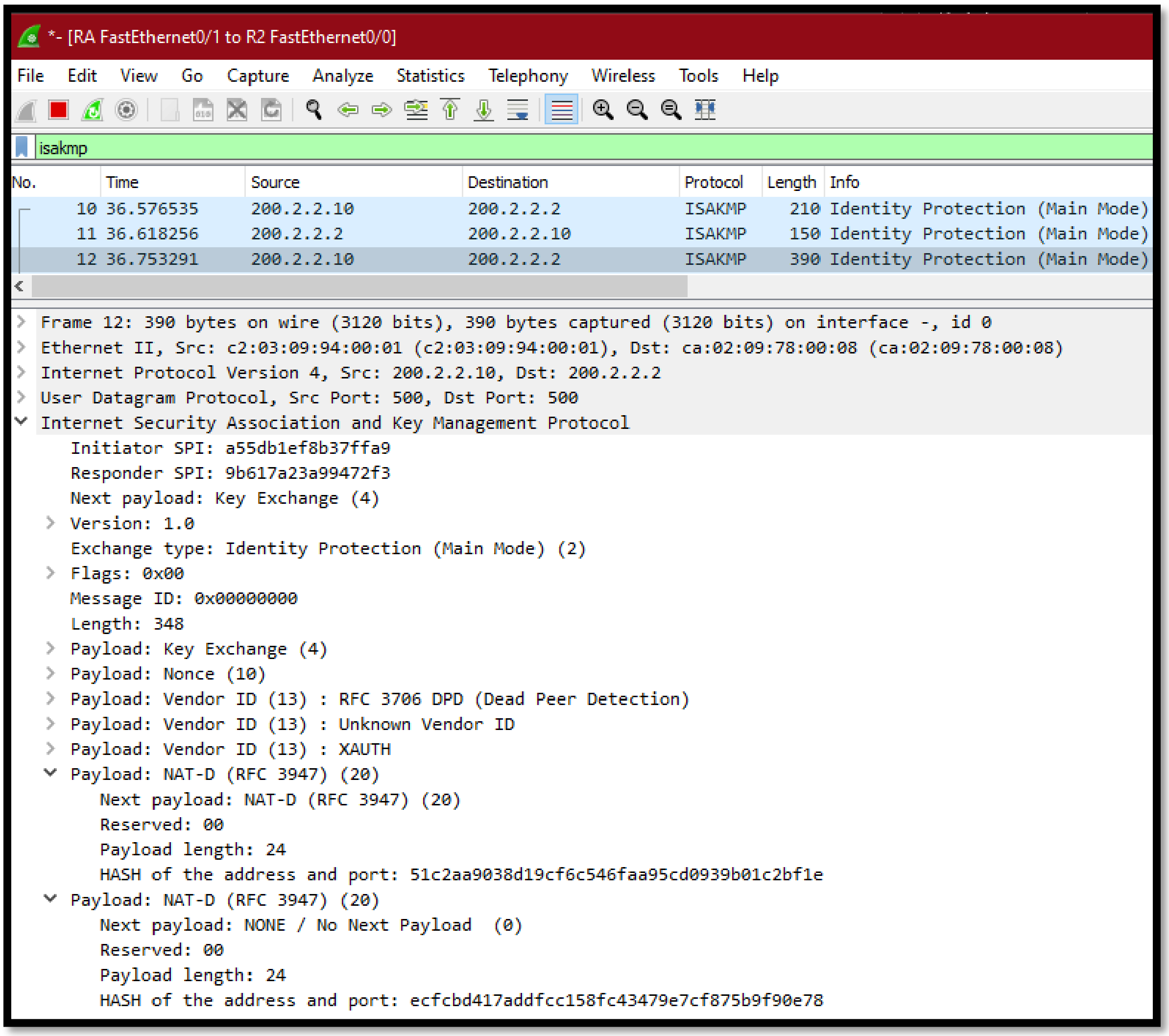


Figure 29 - Third ISAKMP message with NAT-D Payload

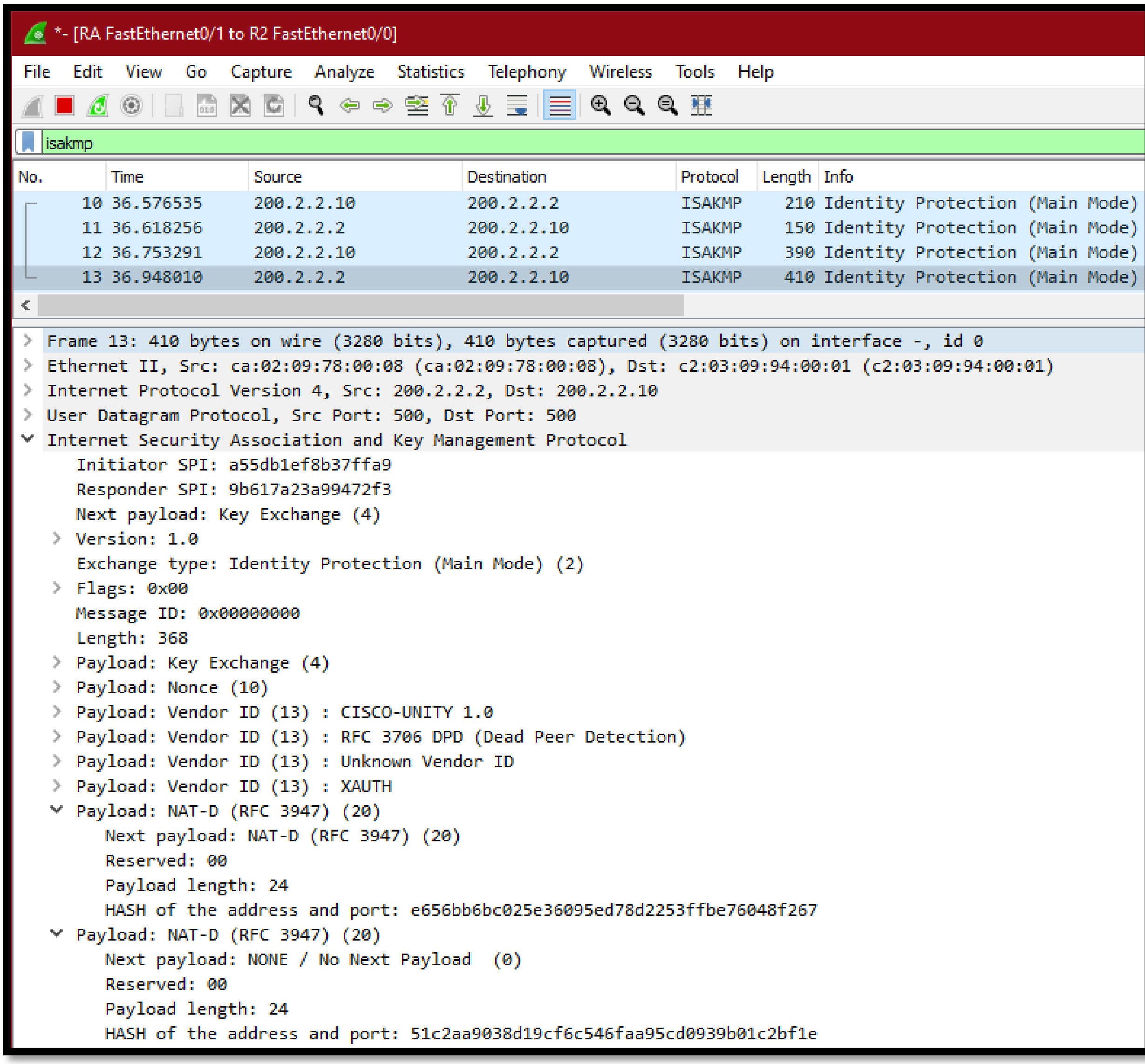


Figure 30 - Fourth ISAKMP message with NAT-D Payload

3. Since **NAT** is detected, **UDP port 4500** will be used from now on, starting in the fifth packet, as shown in Figure 31.

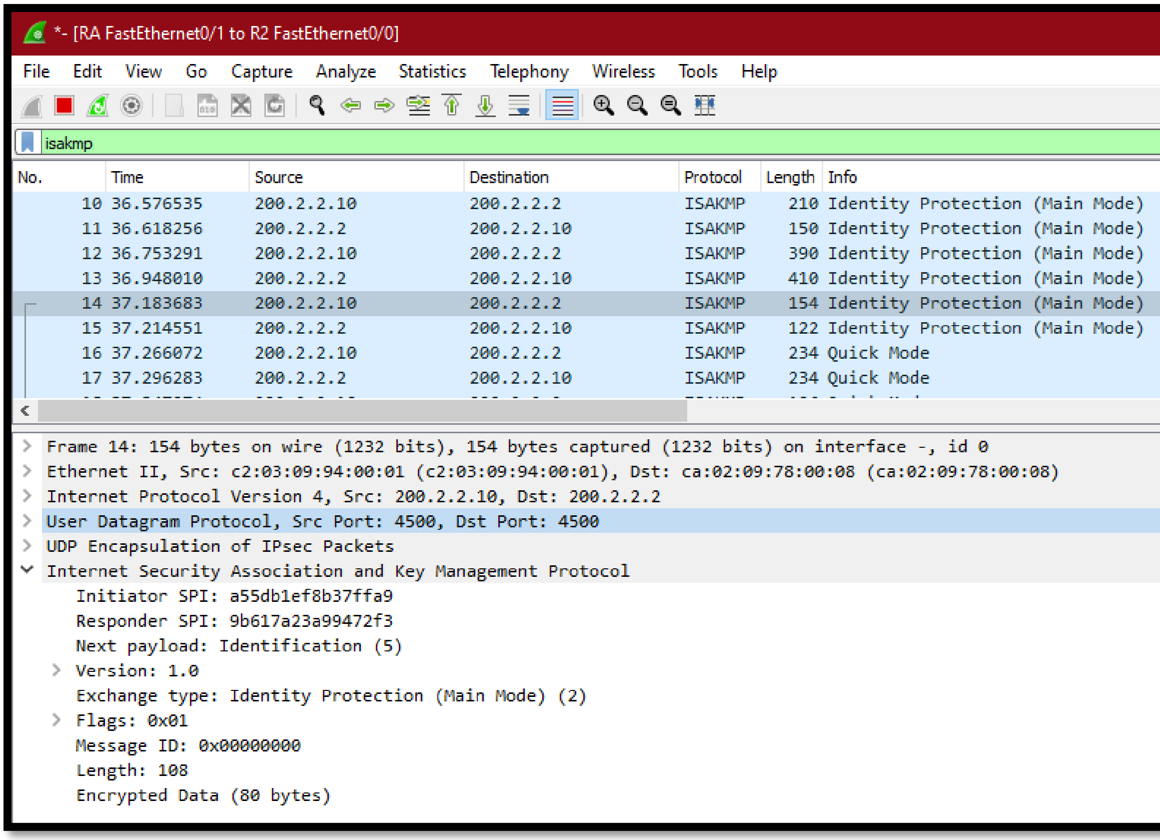


Figure 31 - Fifth ISAKMP packet with port 4500

4. In Figure 32 and Figure 33, we can see the ESP packets in the private and public networks, respectively. As noticed earlier, it is possible to observe that the communication in both networks uses UDP port 4500, allowing the packets to transverse the NAT.

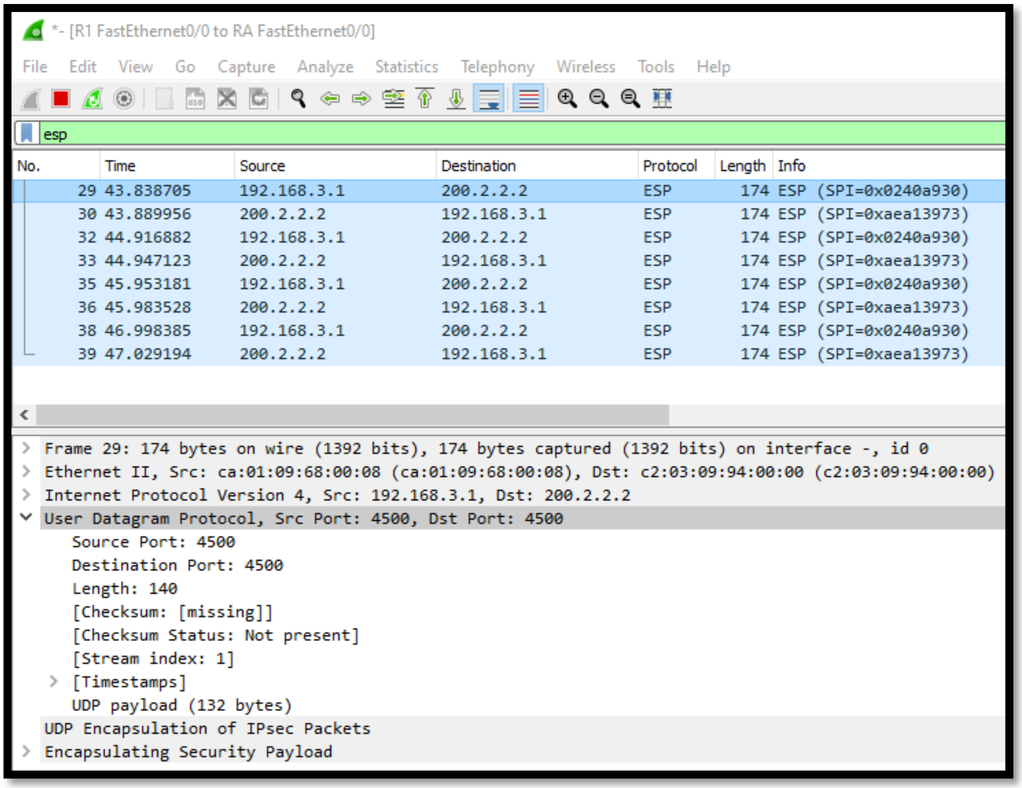


Figure 32 - ESP packets in the private network

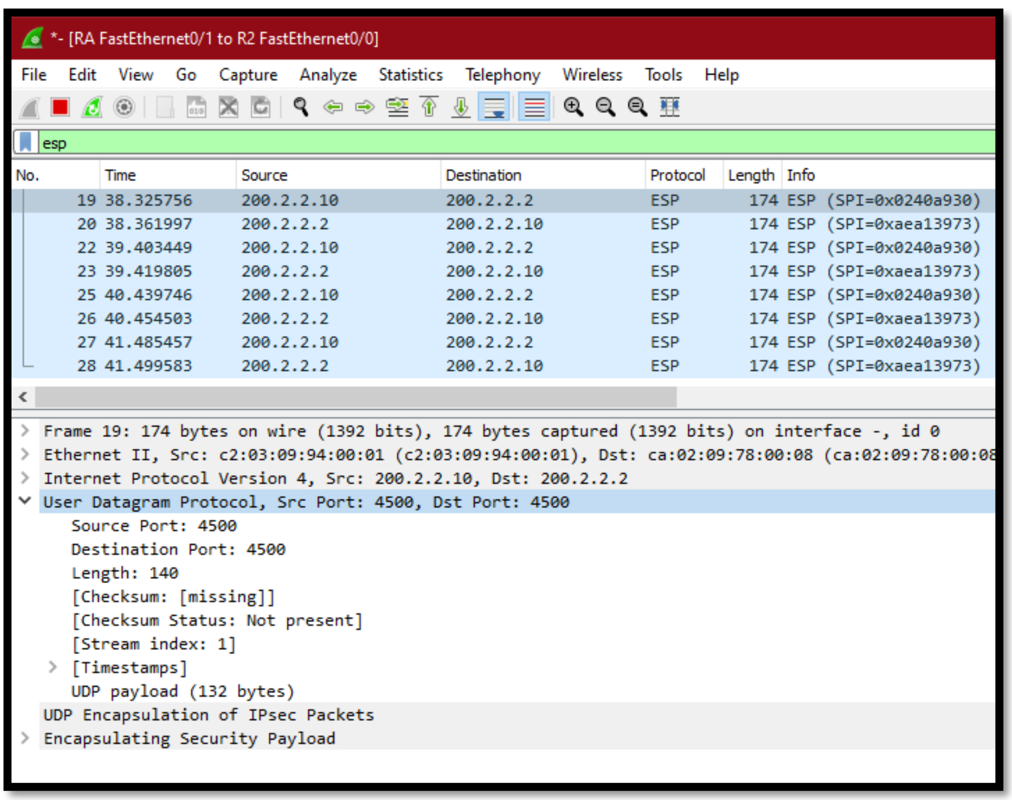


Figure 33 - ESP packets in the public network

1.4. GRE over IPSec

Generic Routing Encapsulation protocol is a simple IP packet encapsulation protocol. It works by encapsulating a payload inside an outer IP packet. Although helpful, GRE is not secure, and IPSec only supports unicast traffic. The solution for having a secure tunnel capable of transporting unicast, multicast, and broadcast IP packets is GRE over IPSec, where broadcast and multicast traffic is encapsulated inside a unicast packet processed by IPSec.

Network Architecture:

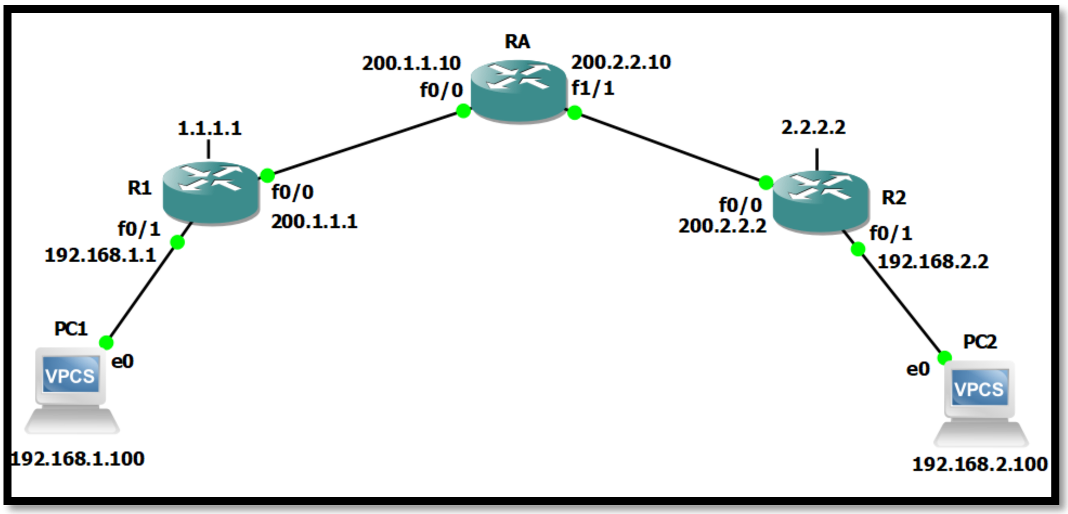


Figure 34 - GRE over IPSec Network Architecture

In this exercise, we configure the network of Figure 34 to support GRE over IPSec (AH protocol in tunnel mode,) where the organisation on the network uses OSPF for routing. The secure tunnel is between R1 and R2, and the public network also uses OSPF. The complete configuration can be found in Annex A.

Proof-of-Concept:

1. After configuring the initial network structures, including the two OSPF areas, we configured the security profile using GRE over IPSec with AH tunnel mode. We captured an ICMP packet at the public network when doing a ping from PC1 to PC2. In this packet, presented in Figure 35, we can identify three encapsulation layers.

```
856 1489.518388   192.168.2.100        192.168.1.100        ICMP      166 Echo (ping) reply     id=
> Frame 856: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface -, id 0
> Ethernet II, Src: ca:01:12:11:00:00 (ca:01:12:11:00:00), Dst: c2:02:12:25:00:00 (c2:02:12:25:00:00)
> Internet Protocol Version 4, Src: 2.2.2.2, Dst: 1.1.1.1
> Authentication Header
> Internet Protocol Version 4, Src: 2.2.2.2, Dst: 1.1.1.1
> Generic Routing Encapsulation (IP)
> Internet Protocol Version 4, Src: 192.168.2.100, Dst: 192.168.1.100
> Internet Control Message Protocol
```

Figure 35 - ICMP Packet from PC1 to PC2 (AH Tunnel mode)

2. The most in-depth layer holds the ICMP packet, which is then encapsulated inside a GRE packet identified by a new IP GRE header, which is again encapsulated and given another IP header provided by the IPsec's tunnel mode. The diagram below provides a visual representation of this explanation.

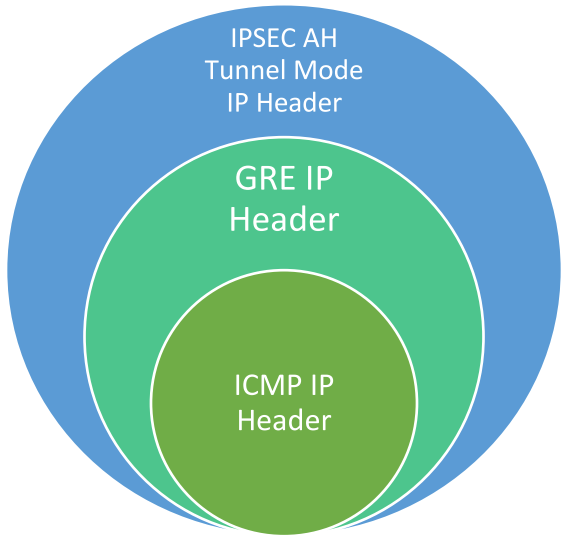


Figure 36 - GRE over IPSec AH Tunnel Mode packet structure

3. The same encapsulation type is also observed in the OSPF packets sent through the tunnel. Figure 38 shows two types of OSPF Hello Packets: one type from the private network and another from the public network. The main difference between them is that the Hello Packets from the public network, presented in Figure 37, only have one IP header since they are not transmitted through GRE over IPsec.

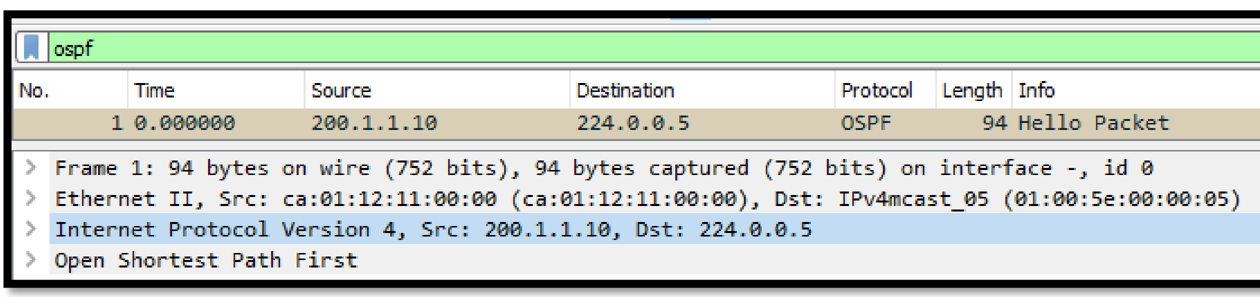
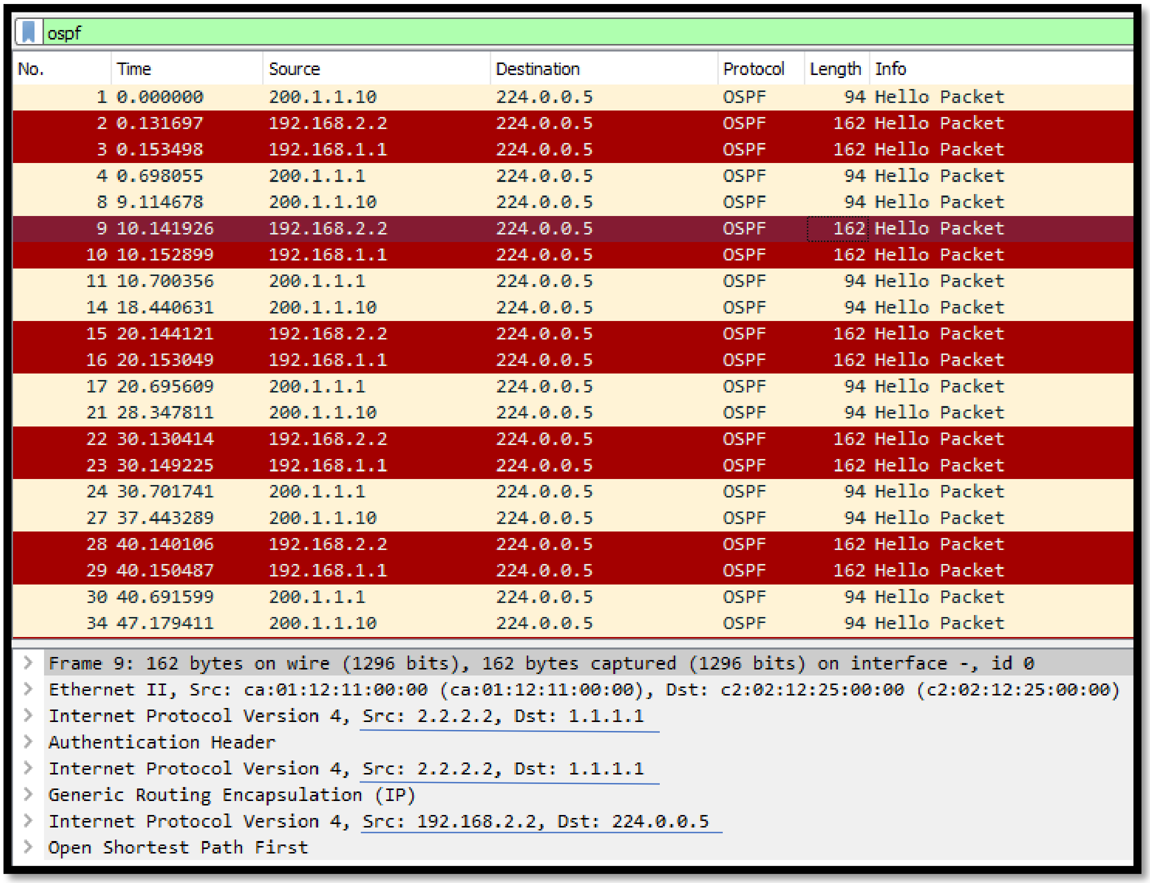


Figure 37 - OSPF Hello Packet Public network


ospf

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	200.1.1.10	224.0.0.5	OSPF	94	Hello Packet
2	0.131697	192.168.2.2	224.0.0.5	OSPF	162	Hello Packet
3	0.153498	192.168.1.1	224.0.0.5	OSPF	162	Hello Packet
4	0.698055	200.1.1.1	224.0.0.5	OSPF	94	Hello Packet
8	9.114678	200.1.1.10	224.0.0.5	OSPF	94	Hello Packet
9	10.141926	192.168.2.2	224.0.0.5	OSPF	162	Hello Packet
10	10.152899	192.168.1.1	224.0.0.5	OSPF	162	Hello Packet
11	10.700356	200.1.1.1	224.0.0.5	OSPF	94	Hello Packet
14	18.440631	200.1.1.10	224.0.0.5	OSPF	94	Hello Packet
15	20.144121	192.168.2.2	224.0.0.5	OSPF	162	Hello Packet
16	20.153049	192.168.1.1	224.0.0.5	OSPF	162	Hello Packet
17	20.695609	200.1.1.1	224.0.0.5	OSPF	94	Hello Packet
21	28.347811	200.1.1.10	224.0.0.5	OSPF	94	Hello Packet
22	30.130414	192.168.2.2	224.0.0.5	OSPF	162	Hello Packet
23	30.149225	192.168.1.1	224.0.0.5	OSPF	162	Hello Packet
24	30.701741	200.1.1.1	224.0.0.5	OSPF	94	Hello Packet
27	37.443289	200.1.1.10	224.0.0.5	OSPF	94	Hello Packet
28	40.140106	192.168.2.2	224.0.0.5	OSPF	162	Hello Packet
29	40.150487	192.168.1.1	224.0.0.5	OSPF	162	Hello Packet
30	40.691599	200.1.1.1	224.0.0.5	OSPF	94	Hello Packet
34	47.179411	200.1.1.10	224.0.0.5	OSPF	94	Hello Packet

Frame 9: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface -, id 0
Ethernet II, Src: ca:01:12:11:00:00 (ca:01:12:11:00:00), Dst: c2:02:12:25:00:00 (c2:02:12:25:00:00)
Internet Protocol Version 4, Src: 2.2.2.2, Dst: 1.1.1.1
Authentication Header
Internet Protocol Version 4, Src: 2.2.2.2, Dst: 1.1.1.1
Generic Routing Encapsulation (IP)
Internet Protocol Version 4, Src: 192.168.2.2, Dst: 224.0.0.5
Open Shortest Path First

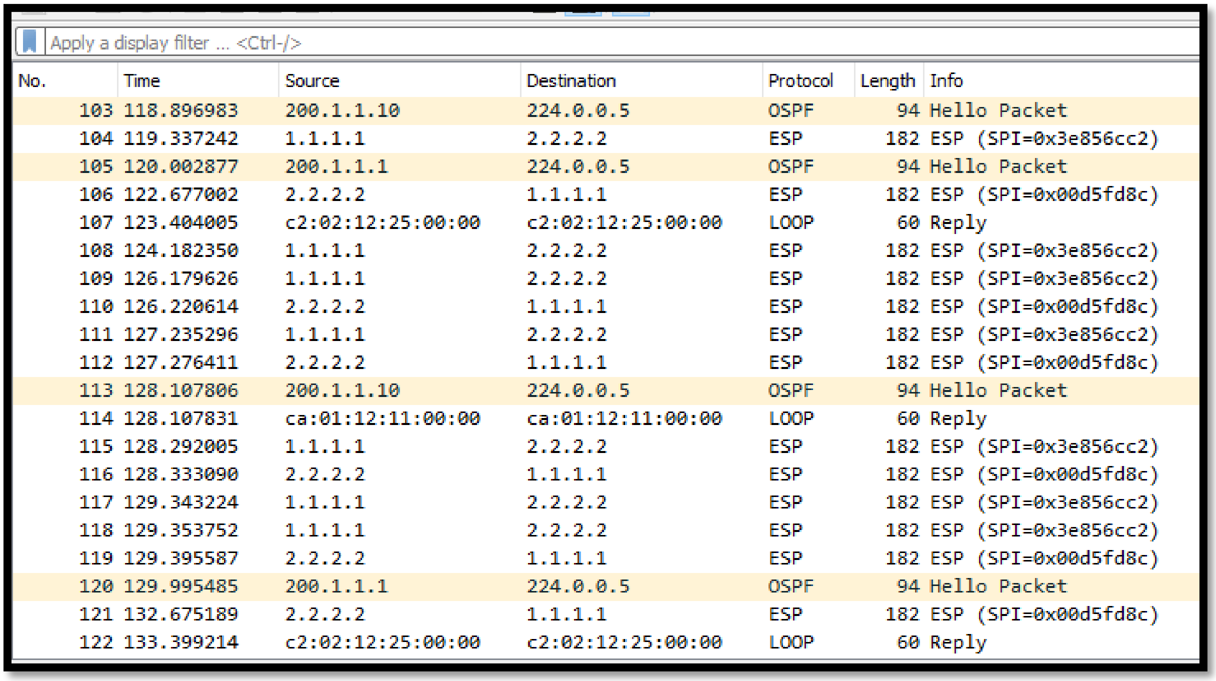
Figure 38 - OSPF Hello Packet Private network

4. By analysing the OSPF LSDB related to the internal network presented in Figure 39, we can also conclude that two types of LSAs are used. Each router generates the Router LSA for each area it is located. In the link-state ID, we can find the originating router's ID. The second type are Network LSAs that are generated by the DR, where the link-state ID is the interface IP address of the DR.

```
            OSPF Router with ID (200.2.2.2) (Process ID 2)

                Router Link States (Area 0)

LS age: 263
Options: (No TOS-capability, DC)
LS Type: Router Links
Link State ID: 200.1.1.1
Advertising Router: 200.1.1.1
LS Seq Number: 80000005
Checksum: 0x3D6D
Length: 48
Number of Links: 2

  Link connected to: a Stub Network
   (Link ID) Network/subnet number: 192.168.1.0
   (Link Data) Network Mask: 255.255.255.0
    Number of TOS metrics: 0
     TOS 0 Metrics: 10

  Link connected to: another Router (point-to-point)
   (Link ID) Neighboring Router ID: 200.2.2.2
   (Link Data) Router Interface address: 0.0.0.9
    Number of TOS metrics: 0
     TOS 0 Metrics: 11111


LS age: 244
Options: (No TOS-capability, DC)
LS Type: Router Links
Link State ID: 200.2.2.2
Advertising Router: 200.2.2.2
LS Seq Number: 80000005
Checksum: 0xD0D5
Length: 48
Number of Links: 2

  Link connected to: a Stub Network
   (Link ID) Network/subnet number: 192.168.2.0
   (Link Data) Network Mask: 255.255.255.0
    Number of TOS metrics: 0
     TOS 0 Metrics: 10

  Link connected to: another Router (point-to-point)
   (Link ID) Neighboring Router ID: 200.1.1.1
   (Link Data) Router Interface address: 0.0.0.9
    Number of TOS metrics: 0
     TOS 0 Metrics: 11111
```

Figure 39 - Router 2 OSPF LSDB Private network

5. Now, we changed the IPSec protocol to ESP and looked at the packets sent while pinging PC2 from PC1. As we can see in Figure 40, it is impossible to distinguish the ICMP packets from the OSPF packets of the private network, as all these packets are cyphered from the point of view of the public network.



No.	Time	Source	Destination	Protocol	Length	Info
103	118.896983	200.1.1.10	224.0.0.5	OSPF	94	Hello Packet
104	119.337242	1.1.1.1	2.2.2.2	ESP	182	ESP (SPI=0x3e856cc2)
105	120.002877	200.1.1.1	224.0.0.5	OSPF	94	Hello Packet
106	122.677002	2.2.2.2	1.1.1.1	ESP	182	ESP (SPI=0x00d5fd8c)
107	123.404005	c2:02:12:25:00:00	c2:02:12:25:00:00	LOOP	60	Reply
108	124.182350	1.1.1.1	2.2.2.2	ESP	182	ESP (SPI=0x3e856cc2)
109	126.179626	1.1.1.1	2.2.2.2	ESP	182	ESP (SPI=0x3e856cc2)
110	126.220614	2.2.2.2	1.1.1.1	ESP	182	ESP (SPI=0x00d5fd8c)
111	127.235296	1.1.1.1	2.2.2.2	ESP	182	ESP (SPI=0x3e856cc2)
112	127.276411	2.2.2.2	1.1.1.1	ESP	182	ESP (SPI=0x00d5fd8c)
113	128.107806	200.1.1.10	224.0.0.5	OSPF	94	Hello Packet
114	128.107831	ca:01:12:11:00:00	ca:01:12:11:00:00	LOOP	60	Reply
115	128.292005	1.1.1.1	2.2.2.2	ESP	182	ESP (SPI=0x3e856cc2)
116	128.333090	2.2.2.2	1.1.1.1	ESP	182	ESP (SPI=0x00d5fd8c)
117	129.343224	1.1.1.1	2.2.2.2	ESP	182	ESP (SPI=0x3e856cc2)
118	129.353752	1.1.1.1	2.2.2.2	ESP	182	ESP (SPI=0x3e856cc2)
119	129.395587	2.2.2.2	1.1.1.1	ESP	182	ESP (SPI=0x00d5fd8c)
120	129.995485	200.1.1.1	224.0.0.5	OSPF	94	Hello Packet
121	132.675189	2.2.2.2	1.1.1.1	ESP	182	ESP (SPI=0x00d5fd8c)
122	133.399214	c2:02:12:25:00:00	c2:02:12:25:00:00	LOOP	60	Reply

Figure 40 - R1 to RA connection using ESP tunnel mode

6. We now change the IPSec protocol back to AH and the IPSec mode to transport to see the differences between transport and tunnel mode. By analysing the ICMP and OSPF packets, we can see several changes regarding encapsulation. In the ISMP and OSPF packets from the private networks, presented in Figure 42 and Figure 43, there are only two IP headers, concluding that the repeated AH IP header has been removed.
   We can see how this mode increases the efficiency of the network by removing the repeated header and therefore conclude that using transport mode can be more appropriate than tunnel mode when using GRE over IPSec.

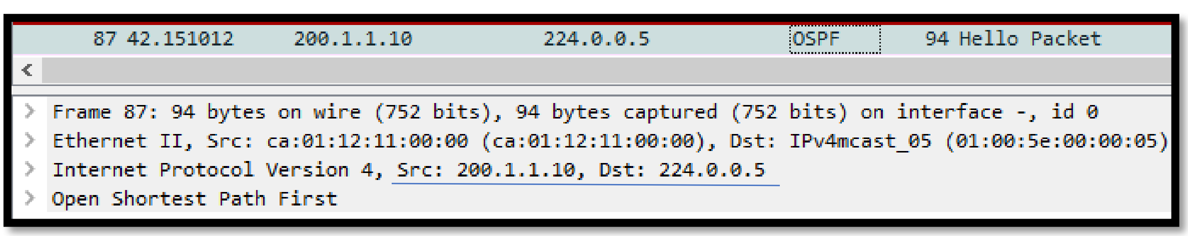


Figure 41 - OSPF Hello Public Network transport mode

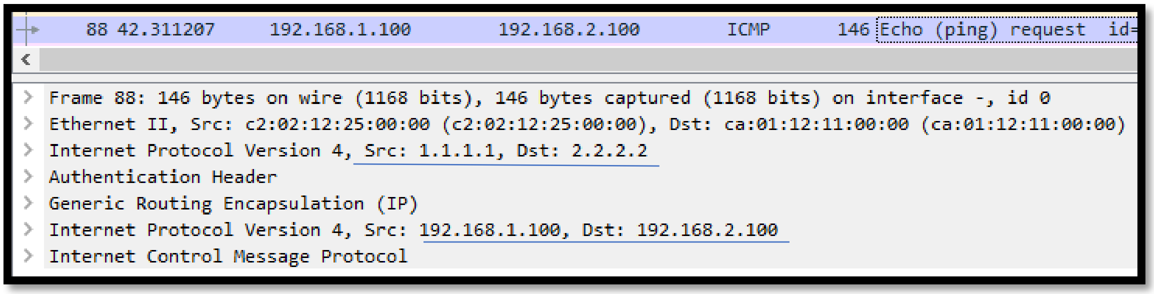


Figure 42 - ICMP packet PC1 to PC2 AH transport mode

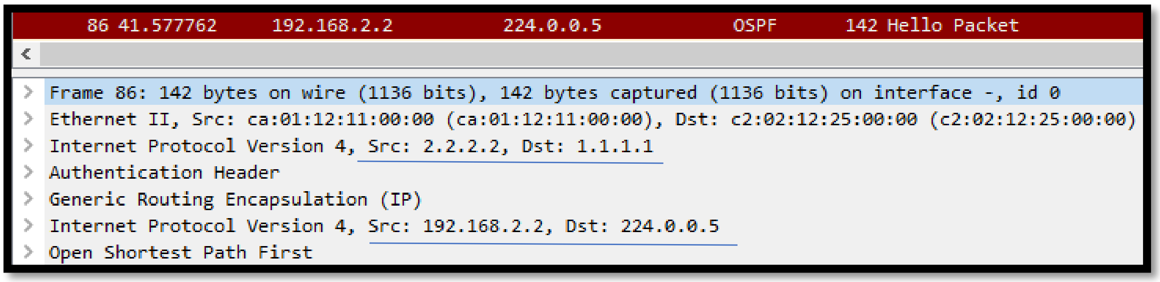


Figure 43 - OSPF packet Private Network AH transport mode

1.5. DMVPN Phase 3

Dynamic Multipoint Virtual Point Network presents a more scalable and dynamic VPN with Hub and Spoke topology. It allows the configuration of a single GRE tunnel interface and one IPSec profile on the hub router that manages all other routers. This feature automatically creates tunnels from hub to spoke or spoke to spoke. It resorts to NHRP, and public addresses are also called NBMA (Public addresses are also called NBMA (Non-Broadcast Multiple Access). (Network Lessons, 2022)

This technology has three different phases corresponding to three configuration options with increasing levels of flexibility. In this lab, we will present the third and most flexible phase.

Network Architecture:

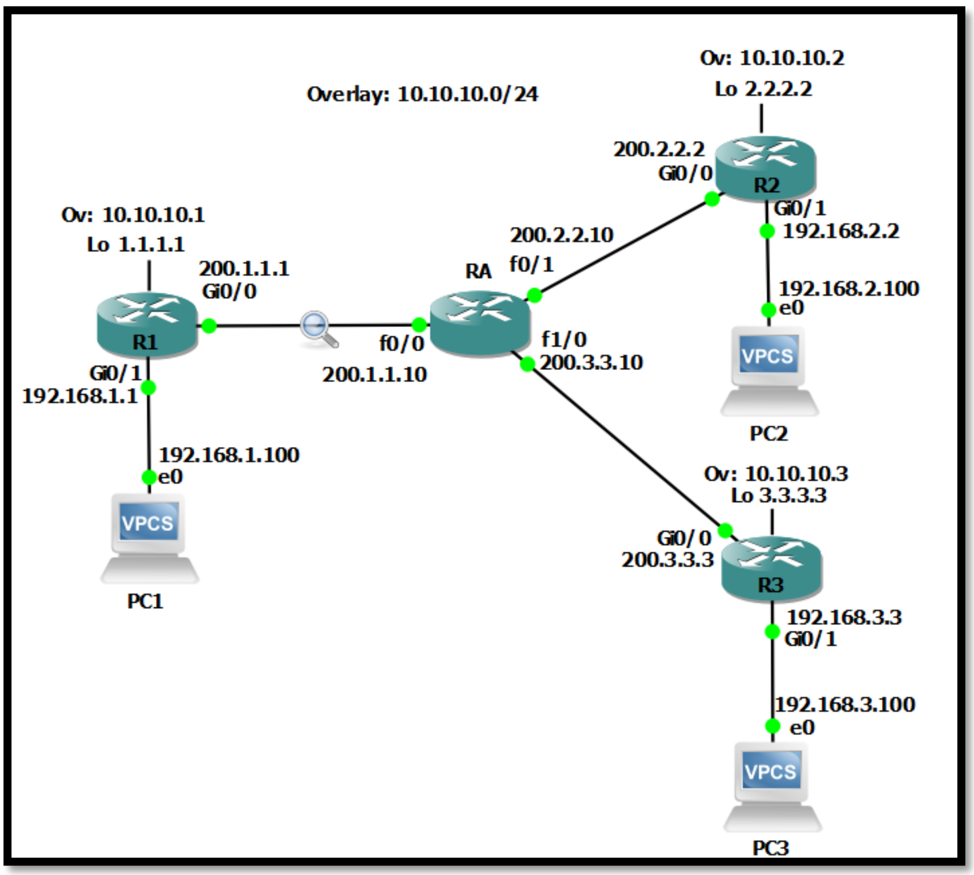


Figure 44 - DMVPN Network Architecture

This architecture represents an organisation's network with three sites: PC1, PC2 and PC3. In this DMVPN lab, R1 is the Hub, and R2 and R3 both are Spoke routers; all these three routers are Cisco IOSv routers. The complete configuration of this lab can be found in Annex A.

Proof-of-Concept:

After implementing the required IP addresses and basic network configurations, we configured the DMVPN Phase 3 solution with NHRP redirects and shortcuts. In this lab exercise, the routing protocols at the organisation network are RIPv2 and OSPF in the outside network.

1. When both OSPF and RIPv2 are implemented correctly in the network, we start by configuring the Hub's (R1) tunnel interface with an NHRP multicast dynamic map with redirect. Then we configure both Spokes' tunnels (R2 and R3) with an NHRP multicast map and shortcut to the Hub.

2. By analysing the RIP and OSPF packets presented in Figure 45 and Figure 46, we could see that their outer IP addresses are different in the public network. RIP packets contain an external IP of the overlay network built on R1 as the source IP address and employ a RIPv2 multicast address (224.0.0.9) as the destination IP, sending routing information to all RIPv2 routers on the private network. Where OSPF uses R1's public interface IP address as the source address and an OSPF multicast address (224.0.0.5) has a destination address, sending Hello packets to all OSPF routers on the public network.

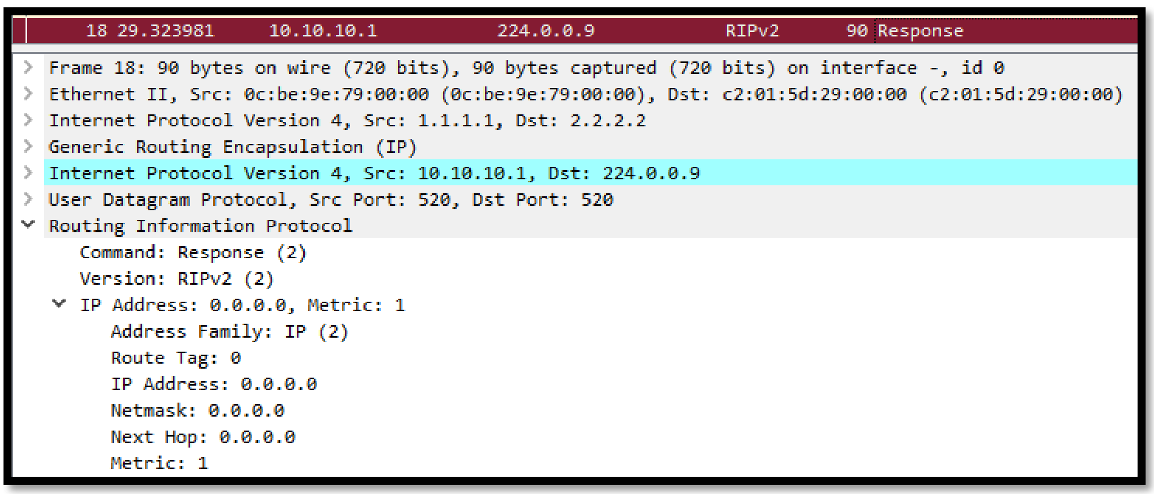


Figure 45 - RIPv2 packet on the public network

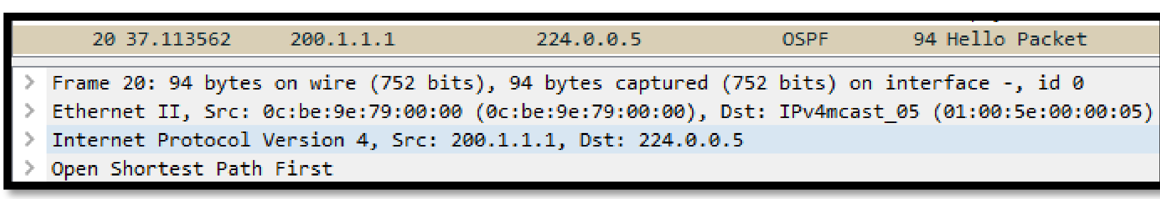


Figure 46 - OSPF packet on the public network

3. To better understand its functionality, we removed the command **ip summary-address rip 0.0.0.0 0.0.0.0** from the tunnel configuration of R1. This command summarises routes in RIPv2 to improve scalability and efficiency. Using this option means there is no entry for child routes in the RIP routing table, reducing the table size and allowing the router to handle more paths. Figure 47 and Figure 48 show a RIP packet being sent without the ip summary-address option and R1's routing table, respectively.

```
5 5.175264    10.10.10.1    224.0.0.9    RIPv2    90 Response
> Frame 5: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface -, id 0
> Ethernet II, Src: 0c:be:9e:79:00:00 (0c:be:9e:79:00:00), Dst: c2:01:5d:29:00:00 (c2:01:5d:29:00:00)
> Internet Protocol Version 4, Src: 1.1.1.1, Dst: 2.2.2.2
> Generic Routing Encapsulation (IP)
> Internet Protocol Version 4, Src: 10.10.10.1, Dst: 224.0.0.9
> User Datagram Protocol, Src Port: 520, Dst Port: 520
v Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  v IP Address: 192.168.1.0, Metric: 1
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 192.168.1.0
      Netmask: 255.255.255.0
      Next Hop: 0.0.0.0
      Metric: 1
```

Figure 47 - RIPv2 packet with no ip summary-address

```
Gateway of last resort is not set

      1.0.0.0/32 is subnetted, 1 subnets
C        1.1.1.1 is directly connected, Loopback0
      2.0.0.0/32 is subnetted, 1 subnets
O        2.2.2.2 [110/12] via 200.1.1.10, 00:11:37, GigabitEthernet0/0
      3.0.0.0/32 is subnetted, 1 subnets
O        3.3.3.3 [110/3] via 200.1.1.10, 00:11:07, GigabitEthernet0/0
      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        10.10.10.0/24 is directly connected, Tunnel0
L        10.10.10.1/32 is directly connected, Tunnel0
      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, GigabitEthernet0/1
L        192.168.1.1/32 is directly connected, GigabitEthernet0/1
R     192.168.2.0/24 [120/1] via 10.10.10.3, 00:00:16, Tunnel0
      200.1.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        200.1.1.0/24 is directly connected, GigabitEthernet0/0
L        200.1.1.1/32 is directly connected, GigabitEthernet0/0
O     200.2.2.0/24 [110/11] via 200.1.1.10, 00:12:15, GigabitEthernet0/0
O     200.3.3.0/24 [110/2] via 200.1.1.10, 00:12:15, GigabitEthernet0/0
Router#
```

Figure 48 - R1's Routing table

4. In addition to such a configuration, we ran the command **no ip split-horizon**. Split-horizon prohibits a router from advertising a route through an interface that the router uses to reach the destination. This is a convenient feature to prevent loops in a network. In Figure 49, we can see the Hub (R1) sending a RIPv2 packet announcing all the routes it has learned back into the network, which would never occur with split-horizon. (fryadmin, 2011)

```
 214 401.138780    10.10.10.1          224.0.0.9          RIPv2      130 Response
<
> Frame 214: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface -, id 0
> Ethernet II, Src: 0c:be:9e:79:00:00 (0c:be:9e:79:00:00), Dst: c2:01:5d:29:00:00 (c2:01:5d:29:00:00)
> Internet Protocol Version 4, Src: 1.1.1.1, Dst: 2.2.2.2
> Generic Routing Encapsulation (IP)
> Internet Protocol Version 4, Src: 10.10.10.1, Dst: 224.0.0.9
> User Datagram Protocol, Src Port: 520, Dst Port: 520
v Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  v IP Address: 10.10.10.0, Metric: 1
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 10.10.10.0
      Netmask: 255.255.255.0
      Next Hop: 0.0.0.0
      Metric: 1
  v IP Address: 192.168.1.0, Metric: 1
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 192.168.1.0
      Netmask: 255.255.255.0
      Next Hop: 0.0.0.0
      Metric: 1
  v IP Address: 192.168.2.0, Metric: 2
      Address Family: IP (2)
      Route Tag: 0
      IP Address: 192.168.2.0
      Netmask: 255.255.255.0
      Next Hop: 10.10.10.3
      Metric: 2
```

Figure 49 – RIPv2 packet with no Split-horizon and no summary-address

5. While analysing the traffic of the DMVPN, we could also capture packets such as the one in Figure 50. NHRP or Next Hop Resolution Protocol allows a Next Hop Client (NHC) to dynamically register with Next Hop Servers. In this case, the NHC are the spoke routers, and the NHS is the hub router. This protocol uses an NHRP cache that stores static and dynamic entries with mapping information from packets such as the one in Figure 50. The NHRP cache of Router 1 can be seen in Figure 51. (Trenner, 2021)

```
 204 388.626635    2.2.2.2             1.1.1.1             NHRP      130 NHRP Registration Request, ID=7
<
> Frame 204: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface -, id 0
> Ethernet II, Src: c2:01:5d:29:00:00 (c2:01:5d:29:00:00), Dst: 0c:be:9e:79:00:00 (0c:be:9e:79:00:00)
> Internet Protocol Version 4, Src: 2.2.2.2, Dst: 1.1.1.1
> Generic Routing Encapsulation (NHRP)
> Next Hop Resolution Protocol (NHRP Registration Request)
```

Figure 50 - NHRP registration request packet

```
Router#show dmvpn
Legend: Attrb --> S - Static, D - Dynamic, I - Incomplete
        N - NATed, L - Local, X - No Socket
        T1 - Route Installed, T2 - Nexthop-override
        C - CTS Capable, I2 - Temporary
        # Ent --> Number of NHRP entries with same NBMA peer
        NHS Status: E --> Expecting Replies, R --> Responding, W --> Waiting
        UpDn Time --> Up or Down Time for a Tunnel
==========================================================================

Interface: Tunnel0, IPv4 NHRP Details
Type:Hub, NHRP Peers:2,

 # Ent  Peer NBMA Addr Peer Tunnel Add State  UpDn Tm Attrb
 ----- --------------- --------------- ----- -------- -----
     1 2.2.2.2              10.10.10.2    UP 00:01:15     D
     1 3.3.3.3              10.10.10.3    UP 00:01:03     D
```

Figure 51 - R1's NHRP cache

1.6. DMVPN over IPSec

Since DMVPN is usually used with the internet as the underlay network, it is best to encrypt its tunnels. To demonstrate how to make DMVPN secure, we will now use IPSec and add it to the DMVPN Phase 3 configuration.

Network Architecture:

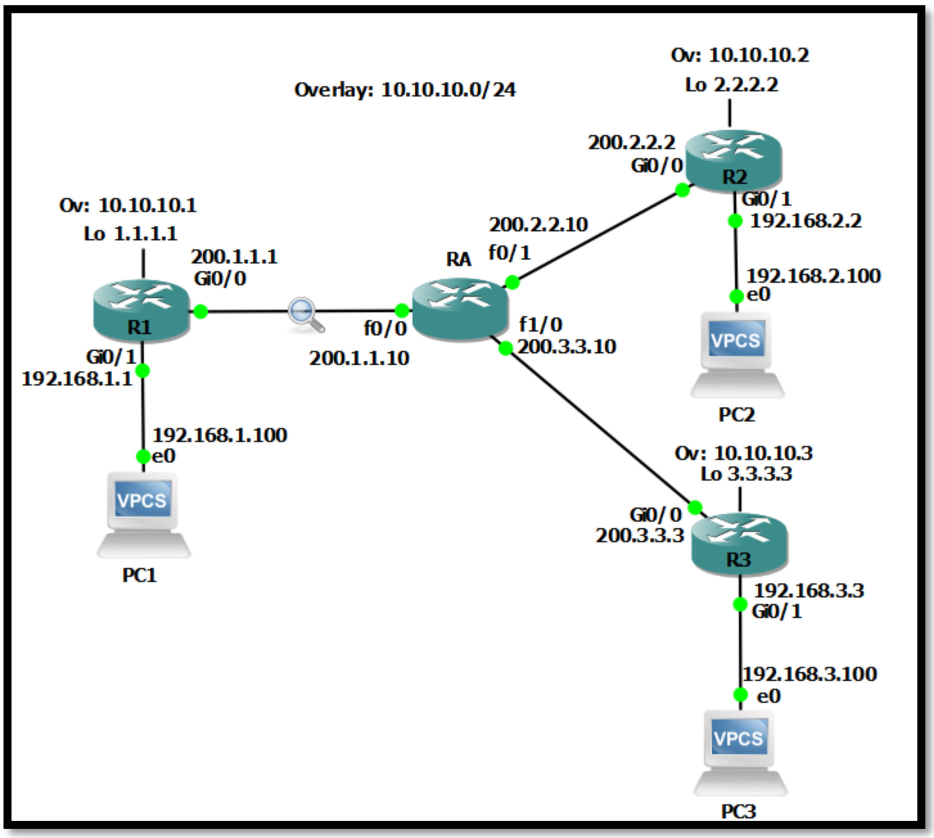


Figure 52 - DMVPN over IPSec Network Architecture

In this exercise, we maintain the same architecture as the previous laboratory exercise. Just like before, R1 is the Hub, and R2 and R3 both are Spoke routers; all these three routers are Cisco IOSv routers. The complete configuration of this lab can be found in Annex A.

Proof-of-Concept:

Using the previously configured network, our task in this exercise was a simple implementation of an IPSec policy profile to the configuration of the tunnel interface of all routers.

1. After implementing the IPSec tunnel, we analysed the routing tables and NHRP caches. Just like shown in Figure 53 and Figure 54, there are no differences compared with the application of DMVPN without IPSec.

```
Gateway of last resort is not set

      1.0.0.0/32 is subnetted, 1 subnets
C        1.1.1.1 is directly connected, Loopback0
      2.0.0.0/32 is subnetted, 1 subnets
O        2.2.2.2 [110/12] via 200.1.1.10, 00:09:53, GigabitEthernet0/0
      3.0.0.0/32 is subnetted, 1 subnets
O        3.3.3.3 [110/3] via 200.1.1.10, 00:09:53, GigabitEthernet0/0
      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        10.10.10.0/24 is directly connected, Tunnel0
L        10.10.10.1/32 is directly connected, Tunnel0
      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, GigabitEthernet0/1
L        192.168.1.1/32 is directly connected, GigabitEthernet0/1
R     192.168.2.0/24 [120/1] via 10.10.10.2, 00:00:00, Tunnel0
R     192.168.3.0/24 [120/1] via 10.10.10.3, 00:00:11, Tunnel0
      200.1.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        200.1.1.0/24 is directly connected, GigabitEthernet0/0
L        200.1.1.1/32 is directly connected, GigabitEthernet0/0
O     200.2.2.0/24 [110/11] via 200.1.1.10, 00:09:53, GigabitEthernet0/0
O     200.3.3.0/24 [110/2] via 200.1.1.10, 00:09:53, GigabitEthernet0/0
Router#
```

Figure 53 - R1's routing table

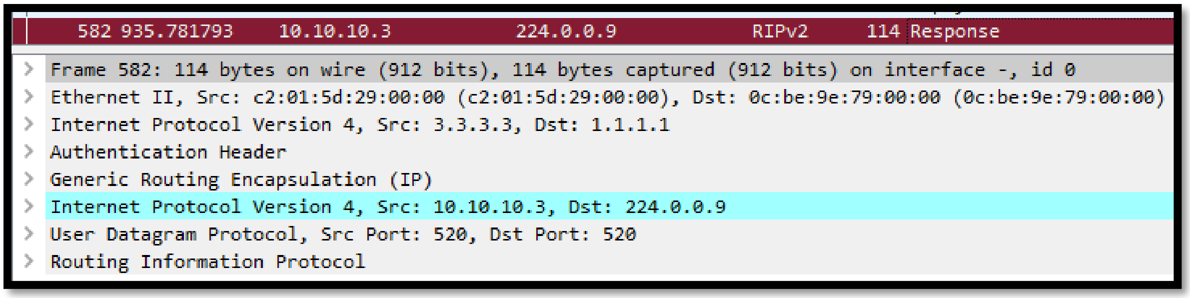
```
Interface: Tunnel0, IPv4 NHRP Details
Type:Hub, NHRP Peers:2,

 # Ent  Peer NBMA Addr Peer Tunnel Add State  UpDn Tm Attrb
 ----- --------------- --------------- ----- -------- -----
     1 2.2.2.2              10.10.10.2    UP 00:07:42     D
     1 3.3.3.3              10.10.10.3    UP 00:07:09     D

Router#
```

Figure 54 - R1's NHRP cache

2. Then, we analysed the packets going through the public network and tried to identify what packets are protected by IPSec; first, we explored an AH configuration. In this analysis, we concluded that IPSec protects all packets between the hub and spokes (overlay) but not the packets in the public network (underlay). In Figure 55 and Figure 56, we can see an NHRP Registration reply and a RIPv2 response packet, both protected by the AH header. Although in Figure 57, we can see an OSPF Hello packet unprotected by IPSec.

```
598 954.615921    1.1.1.1            2.2.2.2            NHRP      174 NHRP Registration Reply, ID=6, Code=Success
> Frame 598: 174 bytes on wire (1392 bits), 174 bytes captured (1392 bits) on interface -, id 0
> Ethernet II, Src: 0c:be:9e:79:00:00 (0c:be:9e:79:00:00), Dst: c2:01:5d:29:00:00 (c2:01:5d:29:00:00)
> Internet Protocol Version 4, Src: 1.1.1.1, Dst: 2.2.2.2
> Authentication Header
> Generic Routing Encapsulation (NHRP)
> Next Hop Resolution Protocol (NHRP Registration Reply)
```

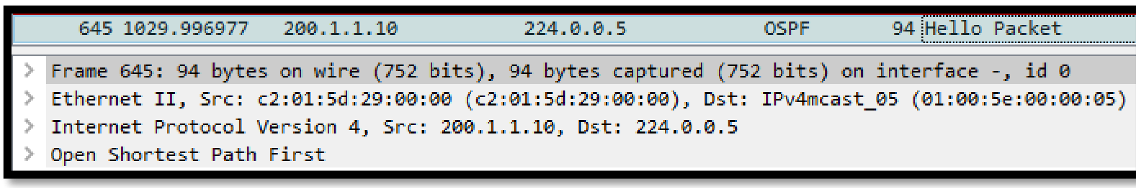
Figure 55 - NHRP Registration Reply with IPSec AH transport mode



582 935.781793 10.10.10.3 224.0.0.9 RIPv2 114 Response

Frame 582: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface -, id 0
Ethernet II, Src: c2:01:5d:29:00:00 (c2:01:5d:29:00:00), Dst: 0c:be:9e:79:00:00 (0c:be:9e:79:00:00)
Internet Protocol Version 4, Src: 3.3.3.3, Dst: 1.1.1.1
Authentication Header
Generic Routing Encapsulation (IP)
Internet Protocol Version 4, Src: 10.10.10.3, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol

Figure 56 - RIPv2 Response packet with IPSec AH transport mode

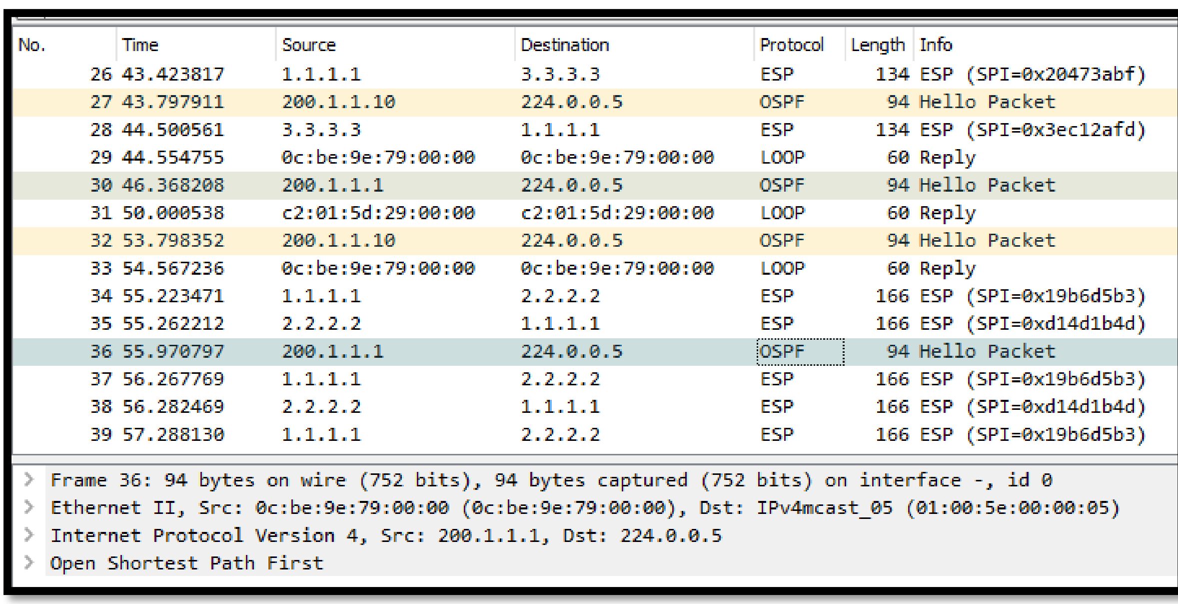


645 1029.996977 200.1.1.10 224.0.0.5 OSPF 94 Hello Packet

Frame 645: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface -, id 0
Ethernet II, Src: c2:01:5d:29:00:00 (c2:01:5d:29:00:00), Dst: IPv4mcast_05 (01:00:5e:00:00:05)
Internet Protocol Version 4, Src: 200.1.1.10, Dst: 224.0.0.5
Open Shortest Path First

Figure 57 - OSPF Hello packet with no IPSec protection

3. Then, we did the same for an ESP configuration. And we concluded that the same happened. As shown in Figure 58, the only packet we can identify is the OSPF Hello packets transmitted between the underlay network. Both RIPv2, NHRP and ICMP packets from the overlay network are encrypted and protected by IPSec.

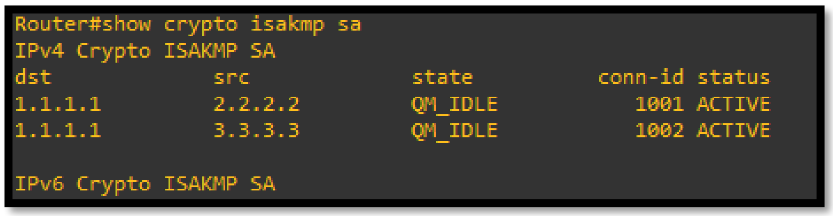


No.	Time	Source	Destination	Protocol	Length	Info
26	43.423817	1.1.1.1	3.3.3.3	ESP	134	ESP (SPI=0x20473abf)
27	43.797911	200.1.1.10	224.0.0.5	OSPF	94	Hello Packet
28	44.500561	3.3.3.3	1.1.1.1	ESP	134	ESP (SPI=0x3ec12afd)
29	44.554755	0c:be:9e:79:00:00	0c:be:9e:79:00:00	LOOP	60	Reply
30	46.368208	200.1.1.1	224.0.0.5	OSPF	94	Hello Packet
31	50.000538	c2:01:5d:29:00:00	c2:01:5d:29:00:00	LOOP	60	Reply
32	53.798352	200.1.1.10	224.0.0.5	OSPF	94	Hello Packet
33	54.567236	0c:be:9e:79:00:00	0c:be:9e:79:00:00	LOOP	60	Reply
34	55.223471	1.1.1.1	2.2.2.2	ESP	166	ESP (SPI=0x19b6d5b3)
35	55.262212	2.2.2.2	1.1.1.1	ESP	166	ESP (SPI=0xd14d1b4d)
36	55.970797	200.1.1.1	224.0.0.5	OSPF	94	Hello Packet
37	56.267769	1.1.1.1	2.2.2.2	ESP	166	ESP (SPI=0x19b6d5b3)
38	56.282469	2.2.2.2	1.1.1.1	ESP	166	ESP (SPI=0xd14d1b4d)
39	57.288130	1.1.1.1	2.2.2.2	ESP	166	ESP (SPI=0x19b6d5b3)

Frame 36: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface -, id 0
Ethernet II, Src: 0c:be:9e:79:00:00 (0c:be:9e:79:00:00), Dst: IPv4mcast_05 (01:00:5e:00:00:05)
Internet Protocol Version 4, Src: 200.1.1.1, Dst: 224.0.0.5
Open Shortest Path First

Figure 58 - Capture of packets transmitted with IPSec ESP transport mode

4. We applied the transport mode in these configurations because, as previously explained in the exercise, GRE over IPSec tunnel mode uses three IP headers. In contrast, the transport mode only uses two, reducing the transmission overhead.
   We also analysed the SAs created between each pair of routers and concluded that there would be a new SA for each tunnel created, as shown in Figure 59.



```
Router#show crypto isakmp sa
IPv4 Crypto ISAKMP SA
dst             src             state          conn-id status
1.1.1.1         2.2.2.2         QM_IDLE           1001 ACTIVE
1.1.1.1         3.3.3.3         QM_IDLE           1002 ACTIVE

IPv6 Crypto ISAKMP SA
```

Figure 59 – R1's SAs in DMVPN over IPSec

2. Lab No. 2.2 – Networking of virtual containers

This lab work aims to study the networking of virtual containers, covering Linux network namespaces, Docker, and Docker Swarm. The laboratories analysed are:

I. Connecting network namespaces to external networks
II. Macvlan networks with VLANs
III. Linux VxLAN with multicast routing
IV. Docker Swarm without data encryption
V. Docker Swarm with data encryption

2.1. Connecting network namespaces to external networks

Linux network namespaces are a Linux kernel feature that allows the isolation of network environments through virtualization. Using this feature allows to create separate network interfaces and routing tables that are isolated from the rest of the system. This feature is the basis of container technologies like Docker or Kubernetes. (Adams, 2021)

In this exercise, we will connect two Linux network namespaces, using a bridge interface, to external networks.

Network Architecture:

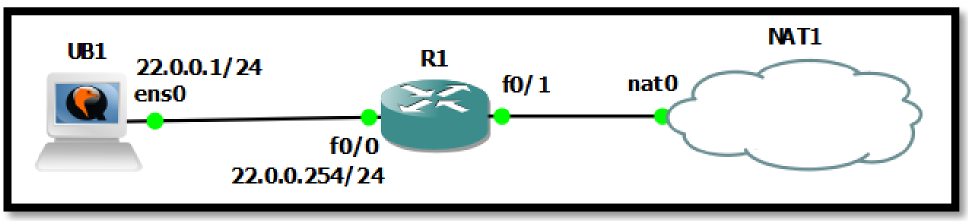


Figure 60 - Network namespaces connecting to external networks network architecture

In this network topology, we use an Ubuntu Server with a cisco router R1 as its default gateway.

Proof-of-Concept:

1. We started by the creation of two network namespaces red and blue identified in Figure 61

```
root@gns3:/home/gns3# ip netns
blue (id: 1)
red (id: 0)
```

Figure 61 - Active NNSes

2. Then we create a bridge interface and virtual cables and connect the bridge interface to the NNSes. These interfaces are presented in Figure 62.

```
root@gns3:/home/gns3# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
 group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode
 DEFAULT group default qlen 1000
    link/ether 0c:ba:28:67:00:00 brd ff:ff:ff:ff:ff:ff
3: v-net-0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mo
de DEFAULT group default qlen 1000
    link/ether e6:4a:ff:24:fe:61 brd ff:ff:ff:ff:ff:ff
4: veth-red-br@if5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue mas
ter v-net-0 state UP mode DEFAULT group default qlen 1000
    link/ether ea:f9:49:89:2f:39 brd ff:ff:ff:ff:ff:ff link-netnsid 0
6: veth-blue-br@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue ma
ster v-net-0 state UP mode DEFAULT group default qlen 1000
    link/ether e6:4a:ff:24:fe:61 brd ff:ff:ff:ff:ff:ff link-netnsid 1
root@gns3:/home/gns3#
```

Figure 62 - Bridge, red and blue interfaces

3. To connect to the external network, we configured a default gateway at the NNSes and NAT at the host so that the 11.0.0.0/24 addresses get translated into the public 22.0.0.0/24 addresses. The default gateway is the bridge's IP address, and NAT is implemented through iptables.

4. After all the configurations presented in [Annex B](), we were able to reach the outside networks as we can see in Figure 63.

```
root@gns3:/home/gns3# ip netns exec blue ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=125 time=27.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=125 time=27.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=125 time=22.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=125 time=23.4 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=125 time=28.3 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4008ms
rtt min/avg/max/mdev = 22.457/25.861/28.381/2.409 ms
```

Figure 63 - Successful communication with external networks

2.2. Macvlan networks with VLANs

Macvlan is a technology that configures multiple MAC addresses on a single physical interface. Having a parent physical ethernet interface Macvlan allows the configuration of sub-interfaces of that same parent interface, attributing unique MAC and IP addresses to each sub-interface. This technology helps bind VMs or containers to a specific sub-interface or implement communication between containers located in different nodes. (Hi Cube, 2022)

Using these networks with IEEE 802.1q trunking allows mapping each Docker host interface to a macvlan network, extending the Layer 2 domain from the VLAN into the macvlan network.

Network Architecture:

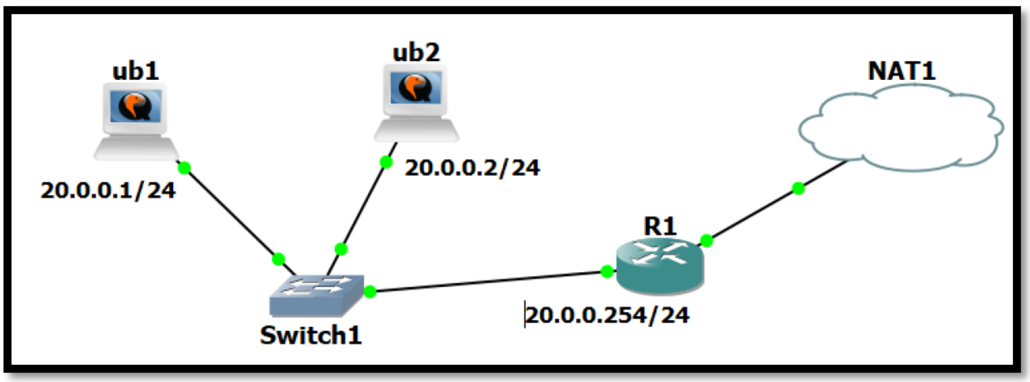


Figure 64 - Macvlan networks with VLANs Network Architecture

In this network topology, we used a 3725 router connected to NAT and a Switch that connects two Ubuntu hosts, ub1 and ub2, each with two alpine containers. The complete configuration can be found in Annex B.

Proof-of-Concept:

To complete this laboratory exercise, we created two Macvlan networks: 11.0.0.0/24 and 12.0.0.0/24. Both were configured in ub1 and ub2. For that, we used the commands:

```
1. docker network create -d macvlan --subnet=11.0.0.0/24 -o parent=ens3.11 my_8021q_net_vlan1
2. docker network create -d macvlan --subnet=12.0.0.0/24 -o parent=ens3.12 my_8021q_net_vlan2
```

1. The **–o** flag is used to specify the sub-interface used by each VLAN. The result of the creation of the Macvlan networks can be found in Figure 65 (for ub1, the results are similar for ub2). In Figure 66, we can observe the sub-interfaces created in ub1.

```
root@gns3:/home/gns3# docker network ls
NETWORK ID     NAME               DRIVER    SCOPE
926efd14c2f3   bridge             bridge    local
23e3b578cb62   host               host      local
4fd78a61bfae   my_8021q_net_vlan1 macvlan   local
cb9e455f7d59   my_8021q_net_vlan2 macvlan   local
baafe6ec79a6   none               null      local
root@gns3:/home/gns3#
```

Figure 65 - ub1 docker networks

```
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN g
roup default
    link/ether 02:42:42:e9:73:e2 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
4: ens3.11@ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 0c:ee:a5:40:00:00 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::eee:a5ff:fe40:0/64 scope link
       valid_lft forever preferred_lft forever
5: ens3.12@ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 0c:ee:a5:40:00:00 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::eee:a5ff:fe40:0/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 66 - Sub-interfaces in ub1

2. We then proceeded to create two containers in both ub1 and ub2. For each host, we have a container attached to VLAN1 and a container attached to VLAN2. These containers can be found in Figure 67 and Figure 68.

```
root@gns3:/home/gns3# docker container ls
CONTAINER ID   IMAGE    COMMAND     CREATED              STATUS             PORTS     NAMES
9c0f78f78330   alpine   "/bin/sh"   About a minute ago   Up About a minute            cont1
5eb2f9d9f02d   alpine   "/bin/sh"   About a minute ago   Up About a minute            cont0
root@gns3:/home/gns3#
```

Figure 67 - Containers in ub1

```
root@gns3:/home/gns3# docker container ls
CONTAINER ID   IMAGE    COMMAND     CREATED          STATUS          PORTS     NAMES
833696937de1   alpine   "/bin/sh"   21 seconds ago   Up 20 seconds             cont3
4a67f4ebc571   alpine   "/bin/sh"   40 seconds ago   Up 38 seconds             cont2
root@gns3:/home/gns3#
```

Figure 68 - Containers in ub2

3. We then tested the connection with a ping from cont0 to cont2 (both attached to vlan1, with cont0 in ub1 and cont2 in ub2) and from cont1 to cont3 (both linked to vlan2, with cont1 in ub1 and cont3 in ub2). We can see in Figure 69 and in Figure 70 that both pings were successful.

```
root@gns3:/home/gns3# docker exec cont0 ping 11.0.0.12
PING 11.0.0.12 (11.0.0.12): 56 data bytes
64 bytes from 11.0.0.12: seq=0 ttl=64 time=1.775 ms
64 bytes from 11.0.0.12: seq=1 ttl=64 time=0.600 ms
64 bytes from 11.0.0.12: seq=2 ttl=64 time=0.672 ms
^C
root@gns3:/home/gns3#
```

Figure 69 - Ping from cont0 to cont2 (VLAN1)

```
root@gns3:/home/gns3# docker exec cont1 ping 12.0.0.13
PING 12.0.0.13 (12.0.0.13): 56 data bytes
64 bytes from 12.0.0.13: seq=0 ttl=64 time=1.771 ms
64 bytes from 12.0.0.13: seq=1 ttl=64 time=0.494 ms
64 bytes from 12.0.0.13: seq=2 ttl=64 time=0.485 ms
64 bytes from 12.0.0.13: seq=3 ttl=64 time=0.469 ms
^C
root@gns3:/home/gns3#
```

Figure 70 - Ping from cont1 to cont3 (VLAN2)

4. To confirm isolation, we tried to ping cont3 from cont0. The ping should not be successful since the containers are attached to different VLANs. We can ensure that cont0 could not reach cont3 by looking at Figure 71.

```
root@gns3:/home/gns3# docker exec cont0 ping 12.0.0.13
PING 12.0.0.13 (12.0.0.13): 56 data bytes
^C
root@gns3:/home/gns3#
```

Figure 71 - Ping from cont0 to cont3 (VLAN1 to VLAN2)

5. We have configured a Wireshark capture between ub1 and the switch to learn more about the communication between the containers. As we can see in Figure 72 and Figure 73, ICMP packets from both VLANs have an 802.1Q VLAN header containing the VLAN ID. VLAN ID allows to identify the source VLAN of the packet sent. Packets from containers in the same subnet will share the same VLAN ID.

```
715 139.663998    11.0.0.10         11.0.0.12         ICMP      102 Echo (ping) request  id
<
> Frame 715: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface -, id 0
> Ethernet II, Src: 02:42:0b:00:00:0a (02:42:0b:00:00:0a), Dst: 02:42:0b:00:00:0c (02:42:0b:00:00:0c)
v 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 11
    000. .... .... .... = Priority: Best Effort (default) (0)
    ...0 .... .... .... = DEI: Ineligible
    .... 0000 0000 1011 = ID: 11
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 11.0.0.10, Dst: 11.0.0.12
> Internet Control Message Protocol
```

Figure 72 - ICMP packet between VLAN1 containers

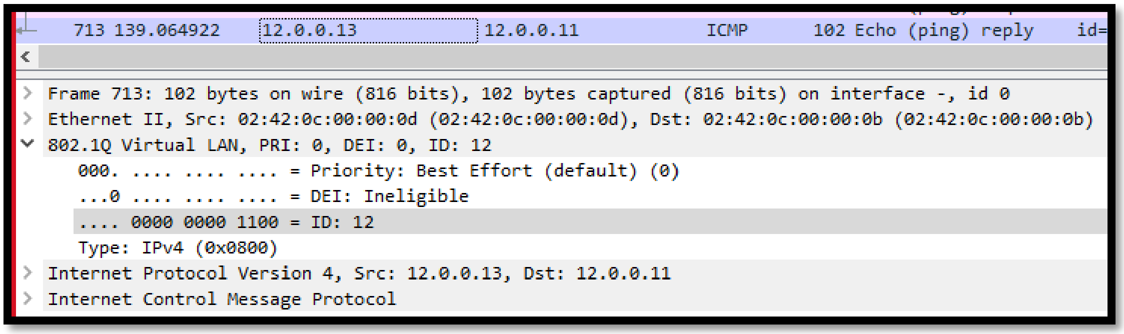


Figure 73 - ICMP packet between VLAN2 containers

2.3. Linux VxLAN with multicast routing

Multicast routing is used to assure connection between networks. Multicasting consists of a group of devices receiving the same messages or packets. For this process to work, multiple devices must share an IP address. Traffic sent to that IP address will reach all devices in that Multicast group. (cloudflare, 2022)

Network Architecture:

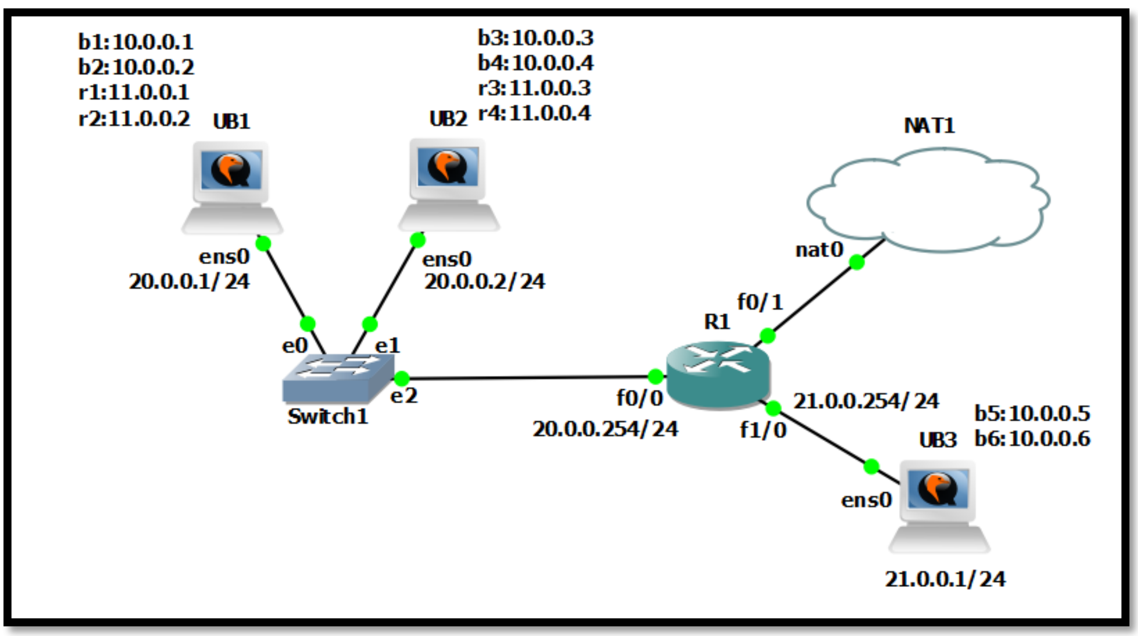


Figure 74 - Linux VxLAN with multicast routing Network Architecture

In this network topology, we use a 3725 router connected to NAT, a Ubuntu machine (UB3) and a Switch that connects two Ubuntu hosts, UB1 and UB2, each Ubuntu machine with four alpine containers except UB3 which only has two in one network (blue). The complete configuration can be found in Annex B.

Proof-of-Concept:

1. The Internet Group Management Protocol (IGMP) will be the protocol responsible for allowing multiple devices to share the IP address. After configurations, we can see in Figure 75 the IGMP table in R1 showing the association between multicast groups and subnets.

```
R1#show ip igmp groups
IGMP Connected Group Membership
Group Address    Interface                Uptime    Expires   Last Reporter    Group Accounted
239.1.1.3        FastEthernet1/0          00:23:45  00:02:02  21.0.0.1
239.1.1.3        FastEthernet0/0          00:39:21  00:02:59  20.0.0.1
239.1.1.6        FastEthernet0/0          00:39:16  00:02:58  20.0.0.1
224.0.1.40       FastEthernet0/0          00:42:56  00:02:07  20.0.0.254
```

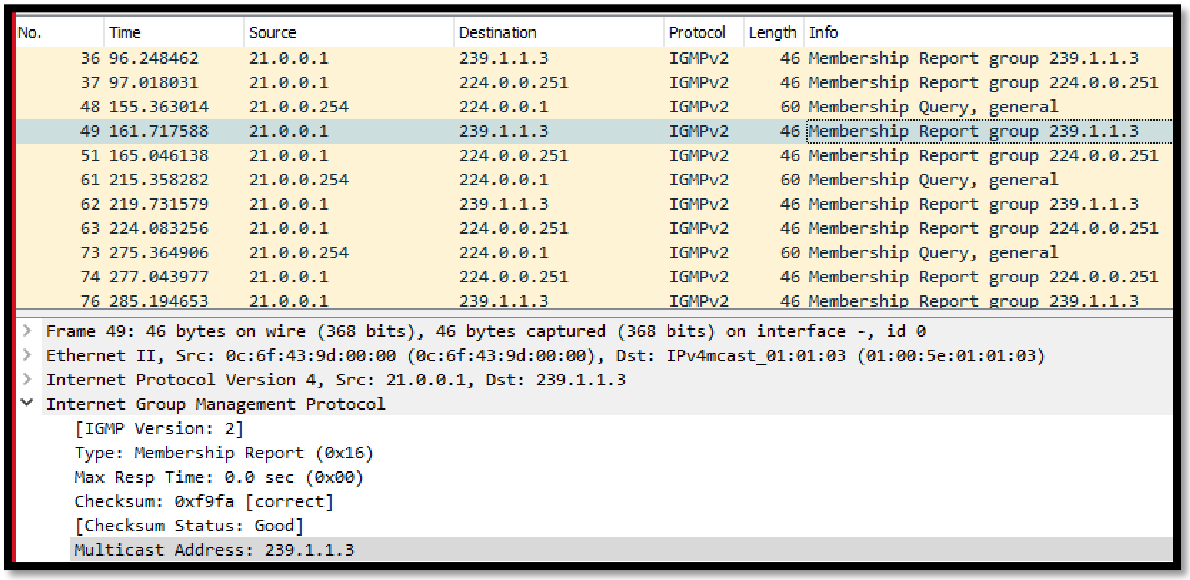
Figure 75 - IGMP Groups R1

2. Both UB1 and UB2 have containers attached to two different subnets: **10.0.0.0/24** and **11.0.0.0/24**. In mind, UB1 and UB2 must be part of the two multicast groups. To better explore the IGMP process, we configured a Wireshark capture between the Switch and R1 and another between R1 and UB3. The IGMP messages regarding the membership request for both UB1 and UB2 can be found in Figure 76. Here we can identify the messages with source UB1 and UB2 to join the multicast groups **230.1.1.3** attached to the VLAN blue and 239.1.1.6 connected to VLAN red.

```
igmp
No.   Time        Source       Destination   Protocol  Length Info
  115 333.698818  20.0.0.1     224.0.0.251   IGMPv2    46 Membership Report group 224.0.0.251
  117 334.701937  20.0.0.254   224.0.1.40    IGMPv2    60 Membership Report group 224.0.1.40
  118 334.848562  20.0.0.2     239.1.1.3     IGMPv2    46 Membership Report group 239.1.1.3
  128 389.697655  20.0.0.254   224.0.0.1     IGMPv2    60 Membership Query, general
  131 391.243255  20.0.0.2     224.0.0.251   IGMPv2    46 Membership Report group 224.0.0.251
  133 394.346443  20.0.0.1     239.1.1.3     IGMPv2    46 Membership Report group 239.1.1.3
  134 395.755352  20.0.0.2     239.1.1.6     IGMPv2    46 Membership Report group 239.1.1.6
  135 399.705385  20.0.0.254   224.0.1.40    IGMPv2    60 Membership Report group 224.0.1.40

> Frame 134: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface -, id 0
> Ethernet II, Src: 0c:ee:a5:40:00:00 (0c:ee:a5:40:00:00), Dst: IPv4mcast_01:01:06 (01:00:5e:01:01:06)
> Internet Protocol Version 4, Src: 20.0.0.2, Dst: 239.1.1.6
v Internet Group Management Protocol
    [IGMP Version: 2]
    Type: Membership Report (0x16)
    Max Resp Time: 0.0 sec (0x00)
    Checksum: 0xf9f7 [correct]
    [Checksum Status: Good]
    Multicast Address: 239.1.1.6
```

Figure 76 - Membership request IGMP messages of UB1 and UB2

3. Furthermore, ub3 has containers only attached to the subnet **10.0.0.0/24**. This way, UB3 only needs to be in the multicast group **239.1.1.3**. We can confirm such in Figure 77.



No.	Time	Source	Destination	Protocol	Length	Info
36	96.248462	21.0.0.1	239.1.1.3	IGMPv2	46	Membership Report group 239.1.1.3
37	97.018031	21.0.0.1	224.0.0.251	IGMPv2	46	Membership Report group 224.0.0.251
48	155.363014	21.0.0.254	224.0.0.1	IGMPv2	60	Membership Query, general
49	161.717588	21.0.0.1	239.1.1.3	IGMPv2	46	Membership Report group 239.1.1.3
51	165.046138	21.0.0.1	224.0.0.251	IGMPv2	46	Membership Report group 224.0.0.251
61	215.358282	21.0.0.254	224.0.0.1	IGMPv2	60	Membership Query, general
62	219.731579	21.0.0.1	239.1.1.3	IGMPv2	46	Membership Report group 239.1.1.3
63	224.083256	21.0.0.1	224.0.0.251	IGMPv2	46	Membership Report group 224.0.0.251
73	275.364906	21.0.0.254	224.0.0.1	IGMPv2	60	Membership Query, general
74	277.043977	21.0.0.1	224.0.0.251	IGMPv2	46	Membership Report group 224.0.0.251
76	285.194653	21.0.0.1	239.1.1.3	IGMPv2	46	Membership Report group 239.1.1.3

```
> Frame 49: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface -, id 0
> Ethernet II, Src: 0c:6f:43:9d:00:00 (0c:6f:43:9d:00:00), Dst: IPv4mcast_01:01:03 (01:00:5e:01:01:03)
> Internet Protocol Version 4, Src: 21.0.0.1, Dst: 239.1.1.3
v Internet Group Management Protocol
    [IGMP Version: 2]
    Type: Membership Report (0x16)
    Max Resp Time: 0.0 sec (0x00)
    Checksum: 0xf9fa [correct]
    [Checksum Status: Good]
    Multicast Address: 239.1.1.3
```

Figure 77 - Membership request IGMP messages of UB3

4. In this exercise, we configured multicast routing in dense-mode. Dense mode multicast routing protocols are used for networks where most subnets in the network should receive the multicast traffic. When a router receives the multicast traffic, it will flood it on all of its interfaces except the interface where it received the multicast traffic. We can see the multicast routing table of R1 in Figure 78.

```
(*, 239.1.1.3), 01:08:24/stopped, RP 0.0.0.0, flags: DC
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet1/0, Forward/Dense, 00:52:48/00:00:00
    FastEthernet0/0, Forward/Dense, 01:08:24/00:00:00

(20.0.0.1, 239.1.1.3), 00:01:21/00:02:52, flags: T
  Incoming interface: FastEthernet0/0, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet1/0, Forward/Dense, 00:01:23/00:00:00

(20.0.0.2, 239.1.1.3), 00:00:13/00:02:51, flags: T
  Incoming interface: FastEthernet0/0, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet1/0, Forward/Dense, 00:00:13/00:00:00

(*, 239.1.1.6), 01:08:20/stopped, RP 0.0.0.0, flags: DC
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet0/0, Forward/Dense, 01:08:20/00:00:00

(20.0.0.1, 239.1.1.6), 00:00:03/00:02:55, flags: PT
  Incoming interface: FastEthernet0/0, RPF nbr 0.0.0.0
  Outgoing interface list: Null

(*, 224.0.1.40), 01:11:08/00:01:54, RP 0.0.0.0, flags: DCL
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet0/0, Forward/Dense, 01:11:08/00:00:00

R1#
```

Figure 78 - R1's Multicast routing table

5. We then proceed to test the connection between containers. As expected, we were able to ping between containers of the same network even if located in different hosts. Pings to containers attached to different subnets were not successful. We can see evidence of both cases in Figure 79 and Figure 80.

```
root@gns3:/home/gns3# docker exec b1 ping -c 4 10.0.0.2
PING 10.0.0.2 (10.0.0.2): 56 data bytes
64 bytes from 10.0.0.2: seq=0 ttl=64 time=5.425 ms
64 bytes from 10.0.0.2: seq=1 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: seq=2 ttl=64 time=0.057 ms
64 bytes from 10.0.0.2: seq=3 ttl=64 time=0.057 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.055/1.398/5.425 ms
```

Figure 79 - ICMP ping from b1 to b2

```
root@gns3:/home/gns3# docker exec r1 ping -c 4 10.0.0.3
PING 10.0.0.3 (10.0.0.3): 56 data bytes

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 0 packets received, 100% packet loss
```

Figure 80 - ICMP ping from r1 to b3

6. After issuing the ping, we analysed a Wireshark capture for more information. We discovered ARP and ICMP packets generated by a ping from r1 to r4. We understood that when pinging for the first time, ARP messages are issued to map the IP associated with the destination host MAC address or, if the destination host is in a different subnet, with the default gateway. ARP and ICMP packets can be seen in Figure 81 and Figure 82 respectively.

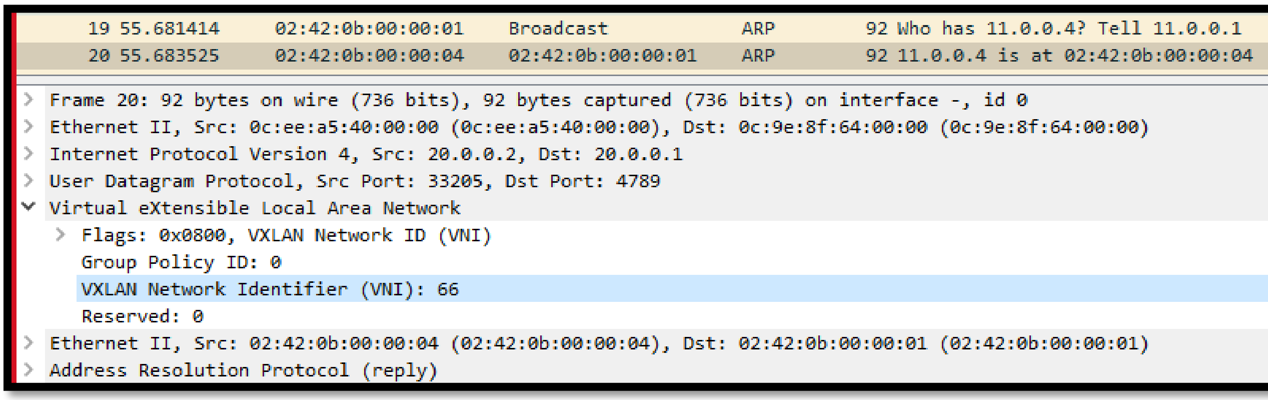


Figure 81 – ARP reply message from UB2 to UB1

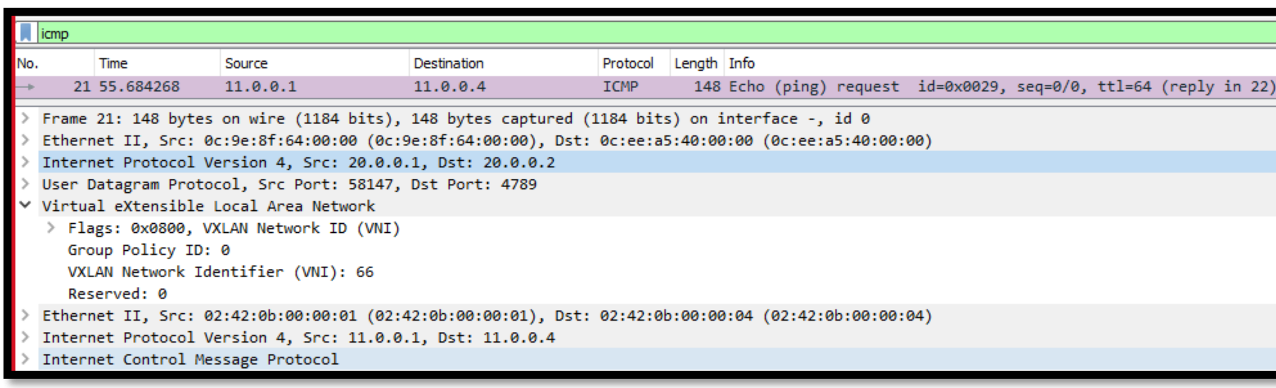


Figure 82 - ICMP packet from r1 to r4

7. We can see that both packets have a Virtual eXtensible Local Area Network (VxLAN) header. This field contains the VXLAN Network identifier (VNI). The VNI is used to identify the source VLAN of the packet sent. Furthermore, we can identify two IP headers in the ICMP packets. The outer header contains the IP of the source and destination hosts, while the inner header contains the IP of the source and destination containers.

8. After the ping we can verify that the ARP cache at r1 has been updated with the IP-MAC address mapping of r4, in Figure 83.

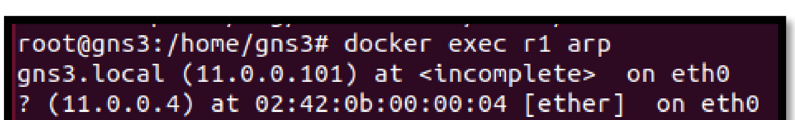


Figure 83 - ARP cache of r1

2.4. Docker Swarm without data encryption

Docker swarm is a container orchestration tool, allowing the user to manage multiple containers deployed across multiple machines, providing high availability for applications.

In a Docker swarm implementation, there are several **worker** nodes and at least one **manager** node responsible for efficiently handling the worker node's resources and ensuring that the cluster operates efficiently. (Sumo Logic, 2022)

All swarm service management traffic is encrypted by default, using the AES algorithm in GCM mode. Manager nodes in the swarm rotate the key used to encrypt secret data every 12 hours. A Docker swarm generates two different kinds of traffic:

- **Control and management plane traffic:** This includes swarm management messages, such as requests to join or leave the swarm. This traffic is always encrypted.
- **Application data plane traffic:** This includes container traffic and traffic to and from external clients. (Docker, 2022)

Network Architecture:

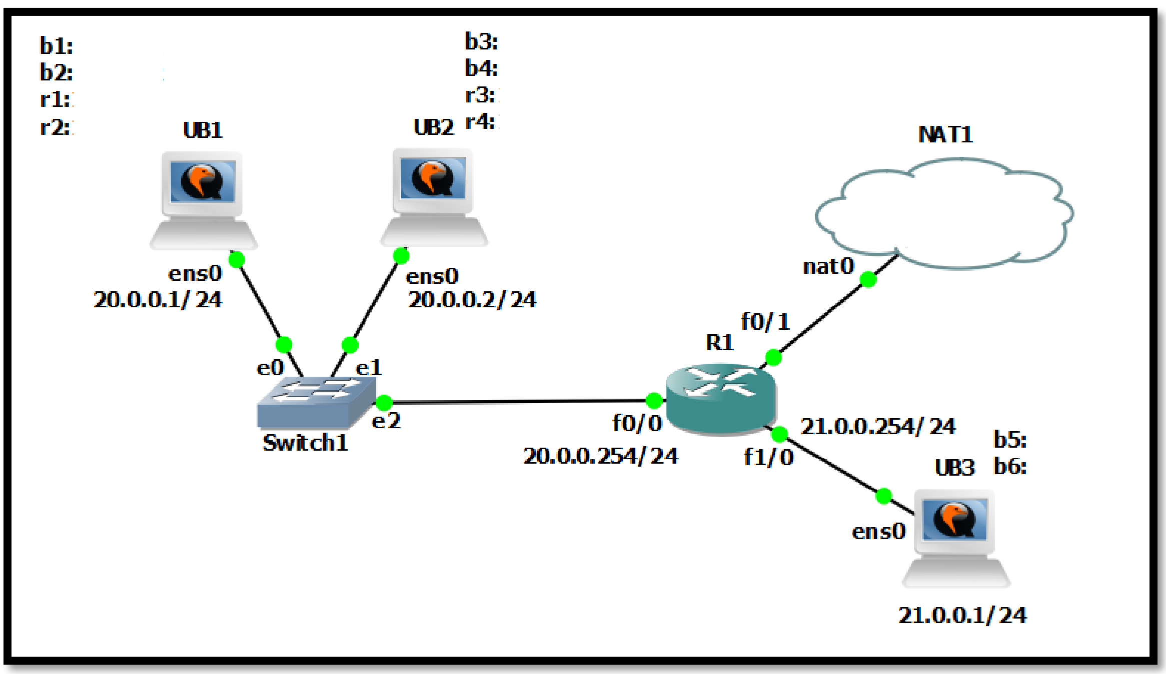


Figure 84 - Docker Swarm without data encryption network architecture

This exercise replicates the network architecture of exercise 2.3, but with Docker Swarm. Where UB1 is the manager and UB2 and UB3 are the workers. The complete configuration can be found in Annex B.

Proof-of-Concept:

1. We started the configuration of this technology by assigning the roles of each node, whereas, in UB1, we generated an advertisement token and used it at the worker nodes to join them to the advertised manager. We can see all the hosts in Figure 85.

```
root@gns3:/home/gns3# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
vtmm5812pe43jcwpxvn87izx1 *   UB1        Ready     Active         Leader           20.10.7
uj4kgknxpfqxoydjg73d3meal     UB2        Ready     Active                          20.10.7
c3nphp90jmbutzac2jwqsz9fm     UB3        Ready     Active                          20.10.7
```

Figure 85 - Docker smarm node's roles

2. Then we created two overlay networks named blue and red, with all the required containers in Figure 84. We finalised the configuration of the network with a connectivity test of all the containers, and we can ensure the network's success in Figure 86.

```
root@gns3:/home/gns3# docker exec b1 ping -c 4 b3
PING b3 (10.0.1.7): 56 data bytes
64 bytes from 10.0.1.7: seq=0 ttl=64 time=1.336 ms
64 bytes from 10.0.1.7: seq=1 ttl=64 time=0.590 ms
64 bytes from 10.0.1.7: seq=2 ttl=64 time=0.555 ms
64 bytes from 10.0.1.7: seq=3 ttl=64 time=0.557 ms

--- b3 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.555/0.759/1.336 ms
root@gns3:/home/gns3# docker exec b1 ping -c 4 b4
PING b4 (10.0.1.9): 56 data bytes
64 bytes from 10.0.1.9: seq=0 ttl=64 time=9.234 ms
64 bytes from 10.0.1.9: seq=1 ttl=64 time=0.862 ms
64 bytes from 10.0.1.9: seq=2 ttl=64 time=0.906 ms
64 bytes from 10.0.1.9: seq=3 ttl=64 time=0.702 ms

--- b4 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.702/2.926/9.234 ms
root@gns3:/home/gns3# docker exec b2 ping -c 4 b5
PING b5 (10.0.1.10): 56 data bytes
64 bytes from 10.0.1.10: seq=0 ttl=64 time=16.287 ms
64 bytes from 10.0.1.10: seq=1 ttl=64 time=13.777 ms
64 bytes from 10.0.1.10: seq=2 ttl=64 time=16.438 ms
64 bytes from 10.0.1.10: seq=3 ttl=64 time=19.707 ms

--- b5 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 13.777/16.552/19.707 ms
root@gns3:/home/gns3# docker exec r1 ping -c 4 r4
PING r4 (10.0.2.7): 56 data bytes
64 bytes from 10.0.2.7: seq=0 ttl=64 time=14.764 ms
64 bytes from 10.0.2.7: seq=1 ttl=64 time=0.576 ms
64 bytes from 10.0.2.7: seq=2 ttl=64 time=1.237 ms
64 bytes from 10.0.2.7: seq=3 ttl=64 time=1.122 ms

--- r4 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.576/4.424/14.764 ms
root@gns3:/home/gns3#
```

Figure 86 - Connectivity test between all the containers

3. Comparing the configuration of Docker Swarm with the previous lab exercise, we can conclude that Docker Swarm has a higher level of automation and simplicity. For instance, we did not have to assign IP addresses to each container or set up the VxLAN interfaces. All these configurations are done automatically. In Figure 87, we can see the details of the containers b1 and b2 at UB1, including their automatically assigned IP and MAC addresses and in Figure 88, the respective VxLAN ID of the blue network.

```
"Containers": {
    "045be051a6b30c6df3737150d19ca4eb20379e009887c6a0e01ba47f2d68edee": {
        "Name": "b1",
        "EndpointID": "ed531cfefe930041a56b74a6b1867e9af64b6a13d41a39e5515df1d5962a1c32",
        "MacAddress": "02:42:0a:00:01:02",
        "IPv4Address": "10.0.1.2/24",
        "IPv6Address": ""
    },
    "7679a6ada50d56f730b1eecba70df1e52ca8875654cd354731dd8b0aeee71700": {
        "Name": "b2",
        "EndpointID": "8da23f40bc740f91806b40d62dd34c536b720d4442c5cbeeac6e1184cdb0b780",
        "MacAddress": "02:42:0a:00:01:04",
        "IPv4Address": "10.0.1.4/24",
        "IPv6Address": ""
    },
```

Figure 87 - Blue network containers of UB1

```
"Options": {
    "com.docker.network.driver.overlay.vxlanid_list": "4097"
```

Figure 88 - Blue network VxLAN ID

Using Wireshark, we analysed the encapsulation of ICMP packets when pinging between containers located in different nodes. In Figure 89 we can see an ICMP packet from b1 to b5 where we can verify that the communication uses VxLAN tunnelling and identify its VxLAN VNI as 4097. In this packet we can also confirm that the outer header contains the IP and MAC of the source and destination hosts, while the inner header contains the IP and MAC of the source and destination containers.

```
→ 141 18.231640   10.0.1.2   10.0.1.12   ICMP   148 Echo (ping) request  id=0x0012, seq=3/768, ttl=64 (reply in 142)
> Frame 141: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface -, id 0
> Ethernet II, Src: 0c:9e:8f:64:00:00 (0c:9e:8f:64:00:00), Dst: c2:01:06:3f:00:00 (c2:01:06:3f:00:00)
> Internet Protocol Version 4, Src: 20.0.0.1, Dst: 21.0.0.1
> User Datagram Protocol, Src Port: 55355, Dst Port: 4789
v Virtual eXtensible Local Area Network
  > Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 4097
    Reserved: 0
> Ethernet II, Src: 02:42:0a:00:01:02 (02:42:0a:00:01:02), Dst: 02:42:0a:00:01:0c (02:42:0a:00:01:0c)
> Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.1.12
> Internet Control Message Protocol
```

Figure 89 - ICMP packet from b1 to b5

4. Also, while analysing the communication in the network via Wireshark we noticed that although we were pinging this container for the first time and using the container's domain names to ping, no ARP or DNS messages were being exchanged. This happens

because of the Docker Swarm control and management plane that exchanges messages, via TLS 1.2, between the containers containing the required information for communication between the different hosts to be possible.

5. We also noticed that it is impossible to see ICMP packets sent to a container of the same node because the ICMP packet reaches the destination container without exiting the node.

2.5. Docker Swarm with data encryption

In this lab exercise, we repeat the last configuration of exercise 2.4 but now with configured encryption for application data.

We add **--opt encrypted** when creating the overlay network to encrypt application data. This extra flag enables IPsec encryption at the level of the VxLAN. This encryption imposes a non-negligible performance penalty.

When overlay encryption is enabled, Docker creates IPSEC tunnels between all the nodes where tasks are scheduled for services attached to the overlay network. These tunnels also use the AES algorithm in GCM mode, and manager nodes automatically rotate the keys every 12 hours. (Docker, 2022)

Proof-of-Concept:

We used the same topology and devices in this network as exercise 2.4. The devices configuration and container creation were also the same, except for the network creation commands at the manager node:

```
#Create two overlay networks named blue and red
docker network create --opt encrypted --driver=overlay --attachable blue
docker network create --opt encrypted --driver=overlay --attachable red
```

1. In the above commands, we create the same overlay networks but now with the specified **--opt encrypted** flag. We can confirm that the encrypted tunnel was built by pinging b5 from b1 (UB1 to UB3) and analysing, in Wireshark, that the ICMP packets are indeed cyphered. In Figure 90, we exhibit one of those packets where we can see the ESP protocol being applied.

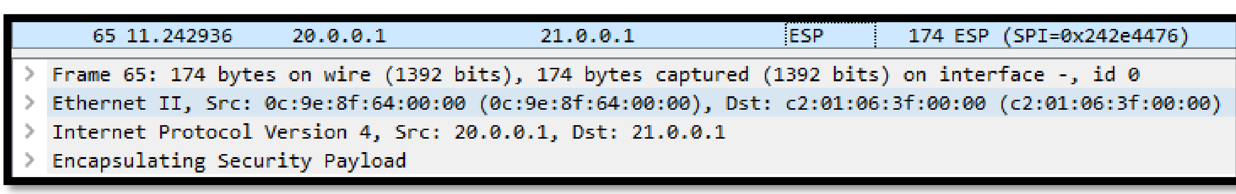


Figure 90 - Cyphered ICMP packet from b1 to b5

2. In the network configuration files we were also able to identify a new entry called encrypted bellow the VxLAN ID identifier, presented in Figure 91.

```
},
"Options": {
    "com.docker.network.driver.overlay.vxlanid_list": "4097",
    "encrypted": ""
},
```

Figure 91 - Encrypted entry at blue network UB1

References:

Adams, D. (2021). *How to Use Linux Network Namespace*. Retrieved from Linux Hint: https://linuxhint.com/use-linux-network-namespace/#:~:text=Linux%20network%20namespaces%20are%20a,the%20system%20and%20operate%20independently.

Clercq, J. D. (2012, May 25). *Q: What firewall ports should we open to make IPSec work through our firewalls?* Retrieved from IT Pro Today: https://www.itprotoday.com/strategy/q-what-firewall-ports-should-we-open-make-ipsec-work-through-our-firewalls

cloudflare. (2022). *What is IGMP? | Internet Group Management Protocol*. Retrieved from cloudflare: https://www.cloudflare.com/learning/network-layer/what-is-igmp/

Docker. (2022). *Manage swarm service networks*. Retrieved from Docker Docs: https://docs.docker.com/engine/swarm/networking/

fryadmin. (2011, August 05). *DMVPN and Routing Protocols – EIGRP*. Retrieved from Fryguy.net: https://fryguy.net/2011/08/05/dmvpn-and-routing-protocols-eigrp/#:~:text=One%20of%20the%20joys%20of,uses%20to%20reach%20the%20destination.

Hi Cube. (2022). *Bridge vs Macvlan*. Retrieved from Hi Cube: https://hicu.be/bridge-vs-macvlan

Network Lessons. (2022). *Introduction to DMVPN*. Retrieved from Network Lessons: https://networklessons.com/cisco/ccie-routing-switching/introduction-to-dmvpn

Sumo Logic. (2022). *DevOps and Security Glossary Terms: Docker Swarm*. Retrieved from Sumo Logic: https://www.sumologic.com/glossary/docker-swarm/#:~:text=Docker%20swarm%20is%20a%20container,of%20availability%20offered%20for%20applications.

Trenner, B. (2021, October 29). *Understanding Next Hop Resolution Protocol Commands*. Retrieved from Global Knowledge: https://www.globalknowledge.com/us-en/resources/resource-library/articles/understanding-next-hop-resolution-protocol-commands/#:~:text=Next%20Hop%20Resolution%20Protocol%20(NHRP,NHS%20is%20the%20hub%20router.

Annex A: Configurations

IPSec using ESP in tunnel mode

```
R1:
#Configuration
enable
conf t
int f0/1
ip address 192.168.1.1 255.255.255.0
no shut

int f0/0
ip address 200.1.1.1 255.255.255.0
no shut
end

#OSPF Configuration
conf t
router ospf 1
network 200.1.1.0 0.0.0.255 area 0
end

conf t
ip route 192.168.2.0 255.255.255.0 200.2.2.2
end

#Define interesting traffic
access-list 110 permit ip 192.168.1.0 0.0.0.255 192.168.2.0 0.0.0.255

#IKE Phase 1 - Configure ISAKMP policy
crypto isakmp policy 10
hash sha
authentication pre-share
group 5
lifetime 86400
encryption aes 256

#IKE Phase 1 - Configure pre-shared key
crypto isakmp key saar address 200.2.2.2

#IKE Phase 2 - Configure IPsec transform set
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac

#Configure crypto map
crypto map myCMap 10 ipsec-isakmp
set peer 200.2.2.2
set transform-set myTSet
match address 110

#Apply crypto map to interface
interface f0/0
crypto map myCMap

R2:
#Configuration
enable
conf t
int f0/1
ip address 192.168.2.2 255.255.255.0
no shut

```

```
int f0/0
ip address 200.2.2.2 255.255.255.0
no shut


#OSPF Configuration
conf t
router ospf 1
network 200.2.2.0 0.0.0.255 area 0
end

conf t
ip route 192.168.1.0 255.255.255.0 200.1.1.1
end

#Define interesting traffic
access-list 110 permit ip 192.168.2.0 0.0.0.255 192.168.1.0 0.0.0.255

#IKE Phase 1 - Configure ISAKMP policy
crypto isakmp policy 10
hash sha
authentication pre-share
group 5
lifetime 86400
encryption aes 256

#IKE Phase 1 - Configure pre-shared key
crypto isakmp key saar address 200.1.1.1

#IKE Phase 2 - Configure IPsec transform set
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac

#Configure crypto map
crypto map myCMap 10 ipsec-isakmp
set peer 200.1.1.1
set transform-set myTSet
match address 110

#Apply crypto map to interface
interface f0/0
crypto map myCMap

#Different Policy
#IKE Phase 1 - Configure ISAKMP policy
crypto isakmp policy 10
hash md5
authentication pre-share
group 5
lifetime 86400
encryption des

RA:
#Configuration
enable
conf t
int f0/0
ip address 200.1.1.10 255.255.255.0
no shut

int f0/1
ip address 200.2.2.10 255.255.255.0
no shut
end

#OSPF Configuration
```

```
conf t
router ospf 1
network 200.1.1.0 0.0.0.255 area 0
network 200.2.2.0 0.0.0.255 area 0
end

PC1:
ip 192.168.1.100 255.255.255.0 192.168.1.1


PC2:
ip 192.168.2.100 255.255.255.0 192.168.2.2
```

IPSec with digital certificates and certificate authority

```
R1:
#Configuration
enable
conf t
int g0/1
ip address 192.168.1.1 255.255.255.0
no shut

int g0/0
ip address 200.1.1.1 255.255.255.0
no shut
end

#OSPF Configuration
conf t
router ospf 1
network 200.1.1.0 0.0.0.255 area 0
end

conf t
ip route 192.168.2.0 255.255.255.0 200.2.2.2
end

#CA Client Configuration
crypto pki trustpoint myTrustpoint
enrollment url http://200.1.1.10:80
exit

crypto ca authenticate myTrustpoint

crypto ca enroll myTrustpoint
show crypto pki certificates
show crypto key mypub rsa

#Define interesting traffic
access-list 110 permit ip 192.168.1.0 0.0.0.255 192.168.2.0 0.0.0.255

#IKE Phase 1 - Configure ISAKMP policy
crypto isakmp policy 10
hash sha
authentication rsa-sig
group 5
lifetime 86400
encryption aes 256

#IKE Phase 2 - Configure IPsec transform set
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac
```

```

#Configure crypto map
crypto map myCMap 10 ipsec-isakmp
set peer 200.2.2.2
set transform-set myTSet
match address 110

#Apply crypto map to interface
interface g0/0
crypto map myCMap

R2:
#Configuration
enable
conf t
int g0/1
ip address 192.168.2.2 255.255.255.0
no shut

int g0/0
ip address 200.2.2.2 255.255.255.0
no shut
end

#OSPF Configuration
conf t
router ospf 1
network 200.2.2.0 0.0.0.255 area 0
end

conf t
ip route 192.168.1.0 255.255.255.0 200.1.1.1
end

#CA Client Configuration
conf t
crypto pki trustpoint myTrustpoint
enrollment url http://200.1.1.10:80
exit

crypto ca authenticate myTrustpoint

crypto ca enroll myTrustpoint

#Define interesting traffic
access-list 110 permit ip 192.168.2.0 0.0.0.255 192.168.1.0 0.0.0.255

#IKE Phase 1 - Configure ISAKMP policy
crypto isakmp policy 10
hash sha
authentication rsa-sig
group 5
lifetime 86400
encryption aes 256

#IKE Phase 2 - Configure IPsec transform set
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac

#Configure crypto map
crypto map myCMap 10 ipsec-isakmp
set peer 200.1.1.1
set transform-set myTSet
match address 110

#Apply crypto map to interface
```

```
interface g0/0
crypto map myCMap

RA:
#Configuration
enable
conf t
int f0/0
ip address 200.1.1.10 255.255.255.0
no shut

int f0/1
ip address 200.2.2.10 255.255.255.0
no shut
end

#OSPF Configuration
conf t
router ospf 1
network 200.1.1.0 0.0.0.255 area 0
network 200.2.2.0 0.0.0.255 area 0
end

#CA Configuration
conf t
ip http server
crypto pki server myCA
issuer-name cn=saarCA
lifetime certificate 365
grant auto
no shutdown
saarlab2

show crypto pki server
show crypto ca certificate
show crypto key mypub rsa


PC1:
ip 192.168.1.100 255.255.255.0 192.168.1.1


PC2:
ip 192.168.2.100 255.255.255.0 192.168.2.2
```

IPSec with NAT traversal

```
R1:
#Configuration
enable
conf t
int f0/1
ip address 192.168.1.1 255.255.255.0
no shut

int f0/0
ip address 192.168.3.1 255.255.255.0
no shut
end

#OSPF R1 CONFIGURATION
conf t
router ospf 1
network 192.168.1.0 0.0.0.255 area 0
network 192.168.3.0 0.0.0.255 area 0
end

#Define interesting traffic
conf t
access-list 110 permit ip 192.168.1.0 0.0.0.255 192.168.2.0 0.0.0.255

#IKE Phase 1 - Configure ISAKMP policy
crypto isakmp policy 10
encryption aes 256
authentication pre-share
group 5
exit

#IKE Phase 1 - Configure pre-shared key
crypto isakmp key saar address 200.2.2.2

#IKE Phase 2 - Configure IPsec transform set
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac

#Configure crypto map
crypto map myCMap 10 ipsec-isakmp
set peer 200.2.2.2
set transform-set myTSet
match address 110

#Apply crypto map to interface
interface f0/0
crypto map myCMap

R2:
#Configuration
enable
conf t
int f0/1
ip address 192.168.2.2 255.255.255.0
no shut

int f0/0
ip address 200.2.2.2 255.255.255.0
no shut
end

#Default route
```

```
conf t
ip route 0.0.0.0 0.0.0.0 200.2.2.10
end

#Define interesting traffic
conf t
access-list 110 permit ip 192.168.2.0 0.0.0.255 192.168.1.0 0.0.0.255

#IKE Phase 1 - Configure ISAKMP policy
crypto isakmp policy 10
encryption aes 256
authentication pre-share
group 5
exit

#IKE Phase 1 - Configure pre-shared key
crypto isakmp key saar address 200.2.2.10

#IKE Phase 2 - Configure IPsec transform set
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac

#Configure crypto map
crypto map myCMap 10 ipsec-isakmp
set peer 200.2.2.10
set transform-set myTSet
match address 110

#Apply crypto map to interface
interface f0/0
crypto map myCMap
end

RA:
#Configuration
enable
conf t
int f0/1
ip address 200.2.2.10 255.255.255.0
no shut

int f0/0
ip address 192.168.3.10 255.255.255.0
no shut
end

#OSPF AND DEFAULT ROUTER CONFIGURATION
conf t
router ospf 1
network 192.168.3.0 0.0.0.255 area 0
default-information originate
ip route 0.0.0.0 0.0.0.0 200.2.2.2
end

#NAT CONFIGURATION
conf t
access-list 1 permit 192.168.1.0 0.255.255.255
access-list 1 permit 192.168.3.0 0.255.255.255
ip nat inside source list 1 interface f0/1 overload
int f0/0
ip nat inside
int f0/1
ip nat outside
end
```

```
PC1:
ip 192.168.1.100 255.255.255.0 192.168.1.1


PC2:
ip 192.168.2.100 255.255.255.0 192.168.2.2
```

GRE over IPSec

```
R1:
#Configuration
enable
conf t
int f0/1
ip address 192.168.1.1 255.255.255.0
no shut

int f0/0
ip address 200.1.1.1 255.255.255.0
no shut

int loopback0
ip address 1.1.1.1 0.0.0.0
no shut
end

#OSPF Configuration
conf t
router ospf 1
network 1.1.1.1 0.0.0.0 area 0
network 200.1.1.0 0.0.0.255 area 0
end

router ospf 2
router-id 1.1.1.1

#IKE Phase 1 - Configure ISAKMP policy
conf t
crypto isakmp policy 10
encryption aes 256
authentication pre-share
group 5

#Configure pre-shared key
crypto isakmp key saar address 2.2.2.2

#IKE Phase 2 - Configure IPsec transform set - AH
crypto ipsec transform-set myTSet ah-sha-hmac

#IKE Phase 2 - Configure IPsec transform set - ESP
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac

#IKE Phase 2 - Configure IPsec transform set - AH Transport
crypto ipsec transform-set myTSet ah-sha-hmac
mode transport

#Configure ipsec profile
crypto ipsec profile myIPSecProfile
set transform-set myTSet

#Configure tunnel interface
interface tunnel 0
```

```
ip unnumbered f0/1
tunnel source Loopback0
tunnel destination 2.2.2.2
tunnel mode gre ip
tunnel protection ipsec profile myIPSecProfile
ip ospf 2 area 0
#Assign OSPF process 2 to interface
int f0/1
ip ospf 2 area 0
R2:
#Configuration
enable
conf t
int f0/1
ip address 192.168.2.2 255.255.255.0
no shut
int f0/0
ip address 200.2.2.2 255.255.255.0
no shut
int loopback0
ip address 2.2.2.2 0.0.0.0
no shut
end
#OSPF Configuration
conf t
router ospf 1
network 2.2.2.2 0.0.0.0 area 0
network 200.2.2.0 0.0.0.255 area 0
end
router ospf 2
router-id 2.2.2.2
#IKE Phase 1 - Configure ISAKMP policy
conf t
crypto isakmp policy 10
encryption aes 256
authentication pre-share
group 5
#Configure pre-shared key
crypto isakmp key saar address 1.1.1.1
#IKE Phase 2 - Configure IPsec transform set - AH
crypto ipsec transform-set myTSet ah-sha-hmac
#IKE Phase 2 - Configure IPsec transform set - ESP
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac
#IKE Phase 2 - Configure IPsec transform set - AH Transport
crypto ipsec transform-set myTSet ah-sha-hmac
mode transport
#Configure ipsec profile
crypto ipsec profile myIPSecProfile
set transform-set myTSet
#Configure tunnel interface
interface tunnel 0
```

```
ip unnumbered f0/1
tunnel source Loopback0
tunnel destination 1.1.1.1
tunnel mode gre ip
tunnel protection ipsec profile myIPSecProfile
ip ospf 2 area 0

#Assign OSPF process 2 to interface
int f0/1
ip ospf 2 area 0

RA:
#Configuration
enable
conf t
int f0/0
ip address 200.1.1.10 255.255.255.0
no shut

int f1/1
ip address 200.2.2.10 255.255.255.0
no shut
end

#OSPF Configuration
conf t
router ospf 1
network 200.1.1.0 0.0.0.255 area 0
network 200.2.2.0 0.0.0.255 area 0
end

PC1:
ip 192.168.1.100 255.255.255.0 192.168.1.1


PC2:
ip 192.168.2.100 255.255.255.0 192.168.2.2

```

DMVPN Phase 3

```
R1:
#Configuration
enable
conf t
int gi0/1
ip address 192.168.1.1 255.255.255.0
no shut

int gi0/0
ip address 200.1.1.1 255.255.255.0
no shut

int loopback0
ip address 1.1.1.1 255.255.255.255
no shut
end

#OSPF Configuration
conf t
router ospf 1
router-id 1.1.1.1
network 200.1.1.0 0.0.0.255 area 0
network 1.1.1.1 0.0.0.0 area 0
end

#RIPv2 configuration
conf t
router rip
version 2
no auto-summary
network 192.168.1.0
network 10.10.10.0
end

#Configuration of the tunnel interface
conf t
int Tunnel0
ip add 10.10.10.1 255.255.255.0
tunnel source Lo0
tunnel mode gre multipoint
ip nhrp network-id 1
ip nhrp map multicast dynamic
ip nhrp redirect
ip summary-address rip 0.0.0.0 0.0.0.0
end

no ip summary-address rip 0.0.0.0 0.0.0.0
no ip split-horizon
show dmvpn

R2:
#Configuration
enable
conf t
int g0/1
ip address 192.168.2.2 255.255.255.0
no shut

int g0/0
ip address 200.2.2.2 255.255.255.0
no shut
```

```
int loopback0
ip address 2.2.2.2 255.255.255.255
no shut
end
#OSPF Configuration
conf t
router ospf 1
router-id 2.2.2.2
network 200.2.2.0 0.0.0.255 area 0
network 2.2.2.2 0.0.0.0 area 0
end
#RIPv2 configuration
conf t
router rip
version 2
no auto-summary
network 192.168.2.0
network 10.10.10.0
end
#Configuration of the tunnel interface
conf t
int Tunnel0
ip add 10.10.10.2 255.255.255.0
tunnel source Lo0
tunnel mode gre multipoint
ip nhrp map 10.10.10.1 1.1.1.1
ip nhrp map multicast 1.1.1.1
ip nhrp nhs 10.10.10.1
ip nhrp network-id 1
ip nhrp shortcut
exit
R3:
#Configuration
enable
conf t
int g0/1
ip address 192.168.3.3 255.255.255.0
no shut
int g0/0
ip address 200.3.3.3 255.255.255.0
no shut
int loopback0
ip address 3.3.3.3 255.255.255.255
no shut
end
#OSPF Configuration
conf t
router ospf 1
router-id 3.3.3.3
network 200.3.3.0 0.0.0.255 area 0
network 3.3.3.3 0.0.0.0 area 0
end
#RIPv2 configuration
conf t
router rip
version 2
```

```
no auto-summary
network 192.168.3.0
network 10.10.10.0
end

#Configuration of the tunnel interface
conf t
int Tunnel0
ip add 10.10.10.3 255.255.255.0
tunnel source Lo0
tunnel mode gre multipoint
tunnel protection ipsec profile myIPSecProfile
ip nhrp map 10.10.10.1 1.1.1.1
ip nhrp map multicast 1.1.1.1
ip nhrp nhs 10.10.10.1
ip nhrp network-id 1
ip nhrp shortcut
end

RA:
#Configuration
enable
conf t
int f0/0
ip address 200.1.1.10 255.255.255.0
no shut

int f0/1
ip address 200.2.2.10 255.255.255.0
no shut

int f1/0
ip address 200.3.3.10 255.255.255.0
no shut
end

#OSPF Configuration
conf t
router ospf 1
network 200.1.1.0 0.0.0.255 area 0
network 200.2.2.0 0.0.0.255 area 0
network 200.3.3.0 0.0.0.255 area 0
end

PC1:
ip 192.168.1.100 255.255.255.0 192.168.1.1


PC2:
ip 192.168.2.100 255.255.255.0 192.168.2.2


PC3:
ip 192.168.3.100 255.255.255.0 192.168.3.3

```

DMVPN over IPSec

```
R1:
#Configuration
enable
conf t
int gi0/1
ip address 192.168.1.1 255.255.255.0
no shut

int gi0/0
ip address 200.1.1.1 255.255.255.0
no shut

int loopback0
ip address 1.1.1.1 255.255.255.255
no shut
end

#RIPv2 configuration
conf t
router rip
version 2
no auto-summary
network 192.168.1.0
network 10.10.10.0
end

#OSPF Configuration
conf t
router ospf 1
router-id 1.1.1.1
network 200.1.1.0 0.0.0.255 area 0
network 1.1.1.1 0.0.0.0 area 0
end

#IKE Phase 1 - Configure ISAKMP policy
conf t
crypto isakmp policy 10
encryption aes 256
authentication pre-share
group 5

#Configure pre-shared key
crypto isakmp key saar address 0.0.0.0

#IKE Phase 2 - Configure IPsec transform set - AH Transport
crypto ipsec transform-set myTSet ah-sha-hmac
mode transport

#IKE Phase 2 - Configure IPSec transform set
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac
mode transport

#Configure ipsec profile
crypto ipsec profile myIPSecProfile
set transform-set myTSet
exit

#Configuration of the tunnel interface
conf t
int Tunnel0
ip add 10.10.10.1 255.255.255.0
```

```
tunnel source Lo0
tunnel mode gre multipoint
tunnel protection ipsec profile myIPSecProfile
ip nhrp network-id 1
ip nhrp map multicast dynamic
ip nhrp redirect
ip summary-address rip 0.0.0.0 0.0.0.0
end

R2:
#Configuration
enable
conf t
int g0/1
ip address 192.168.2.2 255.255.255.0
no shut

int g0/0
ip address 200.2.2.2 255.255.255.0
no shut

int loopback0
ip address 2.2.2.2 255.255.255.255
no shut
end

#RIPv2 configuration
conf t
router rip
version 2
no auto-summary
network 192.168.2.0
network 10.10.10.0
end

#OSPF Configuration
conf t
router ospf 1
 router-id 2.2.2.2
 network 200.2.2.0 0.0.0.255 area 0
 network 2.2.2.2 0.0.0.0 area 0
 end

 #IKE Phase 1 - Configure ISAKMP policy
 conf t
 crypto isakmp policy 10
 encryption aes 256
 authentication pre-share
 group 5

 #Configure pre-shared key
 crypto isakmp key saar address 0.0.0.0

 #IKE Phase 2 - Configure IPsec transform set - AH Transport
 crypto ipsec transform-set myTSet ah-sha-hmac
 mode transport

 #IKE Phase 2 - Configure IPSec transform set
 crypto ipsec transform-set myTSet esp-aes esp-sha-hmac
 mode transport

 #Configure ipsec profile
 crypto ipsec profile myIPSecProfile
 set transform-set myTSet

```

```
#Configuration of the tunnel interface
conf t
int Tunnel0
ip add 10.10.10.2 255.255.255.0
tunnel source Lo0
tunnel mode gre multipoint
tunnel protection ipsec profile myIPSecProfile
ip nhrp map 10.10.10.1 1.1.1.1
ip nhrp map multicast 1.1.1.1
ip nhrp nhs 10.10.10.1
ip nhrp network-id 1
ip nhrp shortcut
exit

R3:
#Configuration
enable
conf t
int g0/1
ip address 192.168.3.3 255.255.255.0
no shut

int g0/0
ip address 200.3.3.3 255.255.255.0
no shut

int loopback0
ip address 3.3.3.3 255.255.255.255
no shut
end

#RIPv2 configuration
conf t
router rip
version 2
no auto-summary
network 192.168.3.0
network 10.10.10.0
end

#OSPF Configuration
conf t
router ospf 1
router-id 3.3.3.3
network 200.3.3.0 0.0.0.255 area 0
network 3.3.3.3 0.0.0.0 area 0
end

#IKE Phase 1 - Configure ISAKMP policy
conf t
crypto isakmp policy 10
encryption aes 256
authentication pre-share
group 5

#Configure pre-shared key
crypto isakmp key saar address 0.0.0.0

#IKE Phase 2 - Configure IPsec transform set - AH Transport
crypto ipsec transform-set myTSet ah-sha-hmac
mode transport

#IKE Phase 2 - Configure IPSec transform set
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac
mode transport
```

```

#Configure ipsec profile
crypto ipsec profile myIPSecProfile
set transform-set myTSet

#Configuration of the tunnel interface
conf t
int Tunnel0
ip add 10.10.10.3 255.255.255.0
tunnel source Lo0
tunnel mode gre multipoint
tunnel protection ipsec profile myIPSecProfile
ip nhrp map 10.10.10.1 1.1.1.1
ip nhrp map multicast 1.1.1.1
ip nhrp nhs 10.10.10.1
ip nhrp network-id 1
ip nhrp shortcut

RA:
#Configuration
enable
conf t
int f0/0
ip address 200.1.1.10 255.255.255.0
no shut

int f0/1
ip address 200.2.2.10 255.255.255.0
no shut

int f1/0
ip address 200.3.3.10 255.255.255.0
no shut
end

#OSPF Configuration
conf t
router ospf 1
network 200.1.1.0 0.0.0.255 area 0
network 200.2.2.0 0.0.0.255 area 0
network 200.3.3.0 0.0.0.255 area 0
end

PC1:
ip 192.168.1.100 255.255.255.0 192.168.1.1


PC2:
ip 192.168.2.100 255.255.255.0 192.168.2.2


PC3:
ip 192.168.3.100 255.255.255.0 192.168.3.3

```

Annex B: Configurations

Connecting network namespaces to external networks

```
R1 Configuration:

conf t
int f0/0
ip add 22.0.0.254 255.255.255.0
ip nat inside
no shut

int f0/1
ip add dhcp
ip nat outside
no shut
exit
ip nat inside source list 1 interface FastEthernet0/1 overload

conf t
access-list 1 permit 22.0.0.0 0.0.0.255
end

Ubuntu 1 Configuration:
sudo ifconfig ens3 22.0.0.1/24
sudo ip route add default via 22.0.0.254

#Two Network Namespaces
ip netns add red
ip netns add blue
ip netns

#Create network bridge interface
ip link add v-net-0 type bridge
ip link set dev v-net-0 up
ip link

#Create cables
ip link add veth-red type veth peer name veth-red-br
ip link add veth-blue type veth peer name veth-blue-br

#Assign the “cables” to the interfaces
ip link set veth-red netns red
ip link set veth-red-br master v-net-0

ip link set veth-blue netns blue
ip link set veth-blue-br master v-net-0
ip link

#Assign IP addresses to interfaces
ip -n red addr add 11.0.0.1/24 dev veth-red
ip -n blue addr add 11.0.0.2/24 dev veth-blue

#Bring interfaces up
ip -n red link set veth-red up
ip -n blue link set veth-blue up
ip link set veth-red-br up
ip link set veth-blue-br up

#Check connection
ip netns exec red ping 11.0.0.2
ping 11.0.0.1
```

```

#Assign an IP address to the bridge interface
ip addr add 11.0.0.25/24 dev v-net-0
ip add
ping 11.0.0.1

#Configure a default route at red and blue
ip netns exec red ip route add default via 11.0.0.25
ip netns exec blue ip route add default via 11.0.0.25

#IPtables NAT implementation
iptables -t nat -A POSTROUTING -s 11.0.0.0/24 -o ens3 -j MASQUERADE
sudo sysctl -w net.ipv4.ip_forward=1

iptables -L
iptables -L -v -n
iptables -t nat -L
iptables -t nat -v -x -n -L
iptables -t nat -F

#Communicate with external networks
ip netns exec blue ping 8.8.8.8

```

Macvlan networks with VLANs

```
R1 Configuration:

conf t
int f0/0
ip add 20.0.0.254 255.255.255.0
ip nat inside
no shut

int f0/1
ip add dhcp
ip nat outside
no shut
exit
ip nat inside source list 1 interface FastEthernet0/1 overload
end

conf t
access-list 1 permit 20.0.0.0 0.0.0.255
end

Ubuntu 1 Configuration:
sudo ifconfig ens3 20.0.0.1/24
sudo ip route add default via 20.0.0.254

#Docker Installation
apt-get update
apt install docker.io
reboot
systemctl status docker
ip link

#Create 802.1q macvlan networks
docker network create -d macvlan --subnet=11.0.0.0/24 -o parent=ens3.11 my_8021q_net_vlan1
docker network create -d macvlan --subnet=12.0.0.0/24 -o parent=ens3.12 my_8021q_net_vlan2
docker network ls
ip addr show
```

```

#Create and run two alpine containers
docker run --net=my_8021q_net_vlan1 --ip=11.0.0.10 --name=cont0 -itd alpine /bin/sh
docker run --net=my_8021q_net_vlan2 --ip=12.0.0.11 --name=cont1 -itd alpine /bin/sh
docker container ls
docker ps

docker exec cont0 ping 20.0.0.13

Ubuntu 2 Configuration:
sudo ifconfig ens3 20.0.0.2/24
sudo ip route add default via 20.0.0.254

#Docker Installation
apt-get update
apt install docker.io
reboot
systemctl status docker
ip link

#Create a 802.1q macvlan network
docker network create -d macvlan --subnet=11.0.0.0/24 -o parent=ens3.11 my_8021q_net_vlan1
docker network create -d macvlan --subnet=12.0.0.0/24 -o parent=ens3.12 my_8021q_net_vlan2
docker network ls
ip addr show

#Create and run two alpine containers
docker run --net=my_8021q_net_vlan1 --ip=11.0.0.12 --name=cont2 -itd alpine /bin/sh
docker run --net=my_8021q_net_vlan2 --ip=12.0.0.13 --name=cont3 -itd alpine /bin/sh
docker container ls
docker ps

#Remove Macvlans
docker network rm my_8021q_net_vlan1
docker network rm my_8021q_net_vlan2

```

Linux VxLAN with multicast routing

```
R1 Configuration:

conf t
ip multicast-routing
int f0/0
ip add 20.0.0.254 255.255.255.0
ip nat inside
ip pim dense-mode
no shut

conf t
int f1/0
ip add 21.0.0.254 255.255.255.0
ip nat inside
ip pim dense-mode
no shut

int f0/1
ip add dhcp
ip nat outside
no shut
exit
ip nat inside source list 1 interface FastEthernet0/1 overload
end

conf t
access-list 1 permit 20.0.0.0 0.0.0.255
access-list 1 permit 21.0.0.0 0.0.0.255
end

#Test
show ip igmp groups
show ip mroute

Ubuntu 1 Configuration:
sudo ifconfig ens3 20.0.0.1/24
sudo ip route add default via 20.0.0.254

#Create blue and red VxLAN interfaces
ip link add blue type vxlan id 33 group 239.1.1.3 ttl 5 dev ens3 dstport 4789
ip link add red type vxlan id 66 group 239.1.1.6 ttl 5 dev ens3 dstport 4789

#Activate the VxLAN interfaces
ip link set blue up
ip link set red up

#Assign IP addresses to the VxLAN interfaces
ip addr add 10.0.0.101/24 dev blue
ip addr add 11.0.0.101/24 dev red

#Create two macvlan networks called bluenet and rednet
docker network create -d macvlan --subnet=10.0.0.0/24 --gateway=10.0.0.101 -o parent=blue bluenet
docker network create -d macvlan --subnet=11.0.0.0/24 --gateway=11.0.0.101 -o parent=red rednet

#Create two alpine containers and attach them to bluenet
docker run --privileged --net=bluenet --name=b1 --ip=10.0.0.1 -itd alpine /bin/sh
docker run --privileged --net=bluenet --name=b2 --ip=10.0.0.2 -itd alpine /bin/sh

#Create two alpine containers and attach them to rednet
```

```
docker run --privileged --net=rednet --name=r1 --ip=11.0.0.1 -itd alpine /bin/sh
docker run --privileged --net=rednet --name=r2 --ip=11.0.0.2 -itd alpine /bin/sh

#Test
docker exec b1 ping -c 4 10.0.0.2
docker exec b1 ping -c 4 10.0.0.3
docker exec b2 ping -c 4 10.0.0.6
docker exec r1 ping -c 4 11.0.0.3

docker exec b1 arp -d 10.0.0.4
docker exec b1 ping -c 4 10.0.0.4

Ubuntu 2 Configuration:
sudo ifconfig ens3 20.0.0.2/24
sudo ip route add default via 20.0.0.254

#Create blue and red VxLAN interfaces
ip link add blue type vxlan id 33 group 239.1.1.3 ttl 5 dev ens3 dstport 4789
ip link add red type vxlan id 66 group 239.1.1.6 ttl 5 dev ens3 dstport 4789

#Activate the VxLAN interfaces
ip link set blue up
ip link set red up

#Assign IP addresses to the VxLAN interfaces
ip addr add 10.0.0.102/24 dev blue
ip addr add 11.0.0.102/24 dev red

#Create two macvlan networks called bluenet and rednet
docker network create -d macvlan --subnet=10.0.0.0/24 --gateway=10.0.0.102 -o parent=blue bluenet
docker network create -d macvlan --subnet=11.0.0.0/24 --gateway=11.0.0.102 -o parent=red rednet

#Create two alpine containers and attach them to bluenet
docker run --privileged --net=bluenet --name=b3 --ip=10.0.0.3 -itd alpine /bin/sh
docker run --privileged --net=bluenet --name=b4 --ip=10.0.0.4 -itd alpine /bin/sh

#Create two alpine containers and attach them to rednet
docker run --privileged --net=rednet --name=r3 --ip=11.0.0.3 -itd alpine /bin/sh
docker run --privileged --net=rednet --name=r4 --ip=11.0.0.4 -itd alpine /bin/sh

docker container ls

Ubuntu 3 Configuration:
sudo ifconfig ens3 21.0.0.1/24
sudo ip route add default via 21.0.0.254

#Docker Installation
apt-get update
apt install docker.io
reboot
systemctl status docker
ip link

#Create blue and red VxLAN interfaces
ip link add blue type vxlan id 33 group 239.1.1.3 ttl 5 dev ens3 dstport 4789

#Activate the VxLAN interfaces
ip link set blue up

#Assign IP addresses to the VxLAN interfaces
ip addr add 10.0.0.103/24 dev blue

#Create two macvlan networks called bluenet and rednet
```

```
docker network create -d macvlan --subnet=10.0.0.0/24 --gateway=10.0.0.103 -o parent=blue bluenet

#Create two alpine containers and attach them to bluenet
docker run --privileged --net=bluenet --name=b5 --ip=10.0.0.5 -itd alpine /bin/sh
docker run --privileged --net=bluenet --name=b6 --ip=10.0.0.6 -itd alpine /bin/sh

docker container ls
```

Docker Swarm without data encryption

```
R1 Configuration:

conf t
int f0/0
ip add 20.0.0.254 255.255.255.0
ip nat inside
no shut

conf t
int f1/0
ip add 21.0.0.254 255.255.255.0
ip nat inside
no shut

int f0/1
ip add dhcp
ip nat outside
no shut
exit
ip nat inside source list 1 interface FastEthernet0/1 overload
end

conf t
access-list 1 permit 20.0.0.0 0.0.0.255
access-list 1 permit 21.0.0.0 0.0.0.255
end

Ubuntu 1 Configuration:(Master)
nano /etc/netplan/50-cloud-init.yaml

network:
  version: 2
  renderer: networkd
  ethernets:
    ens3:
      addresses:
        - 20.0.0.1/24
      gateway4: 20.0.0.254
      nameservers:
        addresses:
          - 8.8.8.8
netplan apply
reboot

#Master advertise
docker swarm init --advertise-addr=20.0.0.1
docker node ls

#Create two overlay networks named blue and red
docker network create --driver=overlay --attachable blue
docker network create --driver=overlay --attachable red
```

```

#Create containers
docker run --privileged --net=blue --name=b1 -itd alpine /bin/sh
docker run --privileged --net=blue --name=b2 -itd alpine /bin/sh

docker run --privileged --net=red --name=r1 -itd alpine /bin/sh
docker run --privileged --net=red --name=r2 -itd alpine /bin/sh

#Test
docker exec b1 ping -c 4 b3
docker exec b1 ping -c 4 b4
docker exec b2 ping -c 4 b5
docker exec r1 ping -c 4 r4

docker exec b1 arp -d 10.0.1.9

#Identify containers
docker inspect blue
docker inspect red

Ubuntu 2 Configuration:
nano /etc/netplan/50-cloud-init.yaml

network:
  version: 2
  renderer: networkd
  ethernets:
    ens3:
      addresses:
        - 20.0.0.2/24
      gateway4: 20.0.0.254
      nameservers:
        addresses:
          - 8.8.8.8
netplan apply
reboot

#Worker join
docker swarm join
docker node ls

#Create containers
docker run --privileged --net=blue --name=b3 -itd alpine /bin/sh
docker run --privileged --net=blue --name=b4 -itd alpine /bin/sh

docker run --privileged --net=red --name=r3 -itd alpine /bin/sh
docker run --privileged --net=red --name=r4 -itd alpine /bin/sh

#Identify containers
docker inspect blue
docker inspect red

Ubuntu 3 Configuration:
nano /etc/netplan/50-cloud-init.yaml

network:
  version: 2
  renderer: networkd
  ethernets:
    ens3:
      addresses:
        - 21.0.0.1/24
      gateway4: 21.0.0.254
      nameservers:
        addresses:
```

```
            - 8.8.8.8

netplan apply
reboot

#Worker join
docker swarm join
docker node ls

#Create containers
docker run --privileged --net=blue --name=b5 -itd alpine /bin/sh
docker run --privileged --net=blue --name=b6 -itd alpine /bin/sh

#Identify containers
docker inspect blue
docker inspect red

```

Docker Swarm with data encryption

```
R1 Configuration:

conf t
int f0/0
ip add 20.0.0.254 255.255.255.0
ip nat inside
no shut

conf t
int f1/0
ip add 21.0.0.254 255.255.255.0
ip nat inside
no shut

int f0/1
ip add dhcp
ip nat outside
no shut
exit
ip nat inside source list 1 interface FastEthernet0/1 overload
end

conf t
access-list 1 permit 20.0.0.0 0.0.0.255
access-list 1 permit 21.0.0.0 0.0.0.255
end

Ubuntu 1 Configuration:(Master)
nano /etc/netplan/50-cloud-init.yaml

network:
  version: 2
  renderer: networkd
  ethernets:
    ens3:
      addresses:
        - 20.0.0.1/24
      gateway4: 20.0.0.254
      nameservers:
        addresses:
          - 8.8.8.8
netplan apply
reboot

#Master advertise
docker swarm init --advertise-addr=20.0.0.1
docker node ls

#Create two overlay networks named blue and red
docker network create --opt encrypted --driver=overlay --attachable blue
docker network create --opt encrypted --driver=overlay --attachable red

#Create containers
docker run --privileged --net=blue --name=b1 -itd alpine /bin/sh
docker run --privileged --net=blue --name=b2 -itd alpine /bin/sh

docker run --privileged --net=red --name=r1 -itd alpine /bin/sh
docker run --privileged --net=red --name=r2 -itd alpine /bin/sh

#Test
docker exec b1 ping -c 4 b3
```

```
docker exec b1 ping -c 4 b4
docker exec b2 ping -c 4 b5
docker exec r1 ping -c 4 r4

docker exec b1 arp -d 10.0.1.9

#Identify containers
docker inspect blue
docker inspect red

Ubuntu 2 Configuration:
nano /etc/netplan/50-cloud-init.yaml

network:
  version: 2
  renderer: networkd
  ethernets:
    ens3:
      addresses:
        - 20.0.0.2/24
      gateway4: 20.0.0.254
      nameservers:
        addresses:
          - 8.8.8.8
netplan apply
reboot

#Worker join
docker swarm join
docker node ls

#Create containers
docker run --privileged --net=blue --name=b3 -itd alpine /bin/sh
docker run --privileged --net=blue --name=b4 -itd alpine /bin/sh

docker run --privileged --net=red --name=r3 -itd alpine /bin/sh
docker run --privileged --net=red --name=r4 -itd alpine /bin/sh

#Identify containers
docker inspect blue
docker inspect red

Ubuntu 3 Configuration:
nano /etc/netplan/50-cloud-init.yaml

network:
  version: 2
  renderer: networkd
  ethernets:
    ens3:
      addresses:
        - 21.0.0.1/24
      gateway4: 21.0.0.254
      nameservers:
        addresses:
          - 8.8.8.8

netplan apply
reboot

#Worker join
docker swarm join
docker node ls

#Create containers
```

```
docker run --privileged --net=blue --name=b5 -itd alpine /bin/sh
docker run --privileged --net=blue --name=b6 -itd alpine /bin/sh

#Identify containers
docker inspect blue

```