# TP3 – Systems Security
# - IPsec -

Vasile-Alexandru IFRIM
M2 SSI

The purpose of this TP is to set up IPsec between multiple computers with different configs, using the `charon` daemon (under `strongswan`) implementing IKEv2.

## 1 Authentication based on preshared key (PSK)

Exercise 1.

In order to set up the virtual network, we will use LXD as a virtualization environment. Creating 2 "routers" from the Ubuntu;20.04 image, connected by default to the lxdbr0 interface (with internet access) we start by downloading the necessary packages – `strongswan` and `strongswan-pki`.

We configure some network bridges to establish the links between the different networks.

```
# subnet
lxc network create br12 ipv4.address=none ipv6.address=none
lxc network create br1 ipv4.address=none ipv6.address=none
lxc network create br2 ipv4.address=none ipv6.address=none
```

```
# plug devices to R1
lxc config device add R1 eth0 nic nictype=bridged parent=br1
lxc config device add R1 eth1 nic nictype=bridged parent=br12

# plug devices to R2
lxc config device add R2 eth0 nic nictype=bridged parent=br2
lxc config device add R2 eth1 nic nictype=bridged parent=br12
```

The containers use `netplan` for networking – for our network configuration, we have to push 2 yaml files where we assign static IP addresses and define the routes needed for communication between the 2 subnets. Finally, we enable IP packets forwarding on the containers for the ability to transfer packets between their connected interfaces.

```
containers=("R1" "R2")
for i in "${!containers[@]}"; do
  cont=${containers[$i]}
  lxc file push "$(pwd)/$cont/50-cloud-init.yaml" "$cont/etc/netplan/50-cloud-init.yaml"
  lxc exec $cont -- netplan apply
  lxc exec $cont -- echo "net.ipv4.ip_forward=1" >> /etc/sysctl.conf
  lxc exec $cont -- sysctl -p /etc/sysctl.conf
done
```

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      dhcp6: no
      addresses:
        - 192.168.1.1/24
    eth1:
      dhcp4: no
      dhcp6: no
      addresses:
        - 10.0.0.1/30
      routes:
        - to: 192.168.2.0/24
          via: 10.0.0.2
```

*R1 netplan config*

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      dhcp6: no
      addresses:
        - 192.168.2.1/24
    eth1:
      dhcp4: no
      dhcp6: no
      addresses:
        - 10.0.0.2/30
      routes:
        - to: 192.168.1.0/24
          via: 10.0.0.1
```

*R2 netplan config*

We can now move on to configuring IPsec. We start by creating a random key of 256 bits:

```
$ xxd -l 32 -p /dev/random | tr -d "\n"; echo
```

This is used as a preshared key (PSK) by a sender to a receiver, on both ends:

```
src_ip dst_ip : PSK "[…]"
```

```
root@R1:~# cat /etc/ipsec.secrets
# This file holds shared secrets or RSA private keys for authentication.

# RSA private key for this host, authenticating it to any other host
# which knows the public part.

192.168.1.1 192.168.2.1 : PSK "97a7c4ca924ab8642c1d8e94395752e2982ed0a6ffef2d71560f0208b061a3ed"
```

```
root@R2:~# cat /etc/ipsec.secrets
# This file holds shared secrets or RSA private keys for authentication.

# RSA private key for this host, authenticating it to any other host
# which knows the public part.

192.168.2.1 192.168.1.1 : PSK "97a7c4ca924ab8642c1d8e94395752e2982ed0a6ffef2d71560f0208b061a3ed"
```

And continue by configuring IPsec:

```
conn %default
  ikelifetime=60m
  keylife=20m
  rekeymargin=3m
  keyingtries=1
  keyexchange=ikev2
  mobike=no
conn IKEv2-PSK
  authby=psk
  left=192.168.1.1
  right=192.168.2.1
  auto=start
```

*R1 ipsec.conf*

```
conn %default
  ikelifetime=60m
  keylife=20m
  rekeymargin=3m
  keyingtries=1
  keyexchange=ikev2
  mobike=no
conn IKEv2-PSK
  authby=psk
  left=192.168.2.1
  right=192.168.1.1
  auto=start
```

*R2 ipsec.conf*

$ `conn %default` – store default values used in all connections

- `ikelifetime` – life duration of the keying channel of a connection (ISAKMP SA); after 60 minutes, a new key is regenerated; This represents the IKE Phase 1 (ISAKMP) life time which should be greater than the IKE Phase 2 (IPSec) life time.
- `keylife`, or "`lifetime`", or "`salifetime`" – how long a particular SA of a connection should last, from successful negotiation to expiry
- `rekeymargin` - the period before the end of the lifetime during which a new key will be negotiated. Here the key will be renegotiated 3m before its expiration.
- `keyingtries` – the number of attempts allowed to establish a connection
- `keyexchange` – the key exchange protocol
- `mobike` – sets the ability to manage IP changes without interruptions in the connection

$ `conn IKEv2-PSK` – configure block for a IPsec connection between 2 specified hosts, using a PSK

- `authby` – authentication method
- `left` – local IP address
- `right` – the other IP address
- `auto=strat` – specify that the connection must be automatically initiated when strongswan starts

To launch a network capture on R1:

```
$ tcpdump -i eth1 udp port 500 or udp port 4500 or esp -w
  capture.pcap &
```

Then start ipsec on both containers:

```
$ ipsec start --nofork --debug-all
```

```
15[NET] received packet: from 192.168.2.1[500] to 192.168.1.1[500] (208 bytes)
15[ENC] parsed IKE_AUTH response 1 [ IDr AUTH SA TSi TSr N(AUTH_LFT) ]
15[IKE] authentication of '192.168.2.1' with pre-shared key successful
15[IKE] IKE_SA IKEv2-PSK[1] established between 192.168.1.1[192.168.1.1]...192.168.2.1[192.168.2.
1]
15[IKE] scheduling reauthentication in 3402s
15[IKE] maximum IKE_SA lifetime 3582s
15[CFG] selected proposal: ESP:AES_CBC_128/HMAC_SHA2_256_128/NO_EXT_SEQ
15[IKE] CHILD_SA IKEv2-PSK{2} established with SPIs c3cea8df_i c21627c6_o and TS 192.168.1.1/32 =
== 192.168.2.1/32
15[IKE] received AUTH_LIFETIME of 3361s, scheduling reauthentication in 3181s
```

After doing a ping from R1 to R2, we stop and pull the capture to analyze it in Wireshark.

```
$ sudo lxc file pull R1/root/capture.pcap capture.pcap
```

| Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|
| 192.168.1.1 | 192.168.2.1 | ISAKMP | 978 | IKE_SA_INIT MID=00 Initiator Request |
| 192.168.1.1 | 192.168.2.1 | ISAKMP | 978 | IKE_SA_INIT MID=00 Initiator Request |
| 192.168.2.1 | 192.168.1.1 | ISAKMP | 978 | IKE_SA_INIT MID=00 Initiator Request |
| 192.168.1.1 | 192.168.2.1 | ISAKMP | 322 | IKE_SA_INIT MID=00 Responder Response |
| 192.168.2.1 | 192.168.1.1 | ISAKMP | 346 | IKE_AUTH MID=01 Initiator Request |
| 192.168.1.1 | 192.168.2.1 | ISAKMP | 250 | IKE_AUTH MID=01 Responder Response |
| 192.168.1.1 | 192.168.2.1 | ISAKMP | 978 | IKE_SA_INIT MID=00 Initiator Request |
| 192.168.2.1 | 192.168.1.1 | ISAKMP | 322 | IKE_SA_INIT MID=00 Responder Response |
| 192.168.1.1 | 192.168.2.1 | ISAKMP | 330 | IKE_AUTH MID=01 Initiator Request |
| 192.168.2.1 | 192.168.1.1 | ISAKMP | 250 | IKE_AUTH MID=01 Responder Response |
| 192.168.2.1 | 192.168.1.1 | ISAKMP | 122 | INFORMATIONAL MID=02 Initiator Request |
| 192.168.1.1 | 192.168.2.1 | ISAKMP | 122 | INFORMATIONAL MID=02 Responder Response |
| 192.168.1.1 | 192.168.2.1 | ESP | 170 | ESP (SPI=0xc21627c6) |
| 192.168.2.1 | 192.168.1.1 | ESP | 170 | ESP (SPI=0xc3cea8df) |
| 192.168.1.1 | 192.168.2.1 | ESP | 170 | ESP (SPI=0xc21627c6) |
| 192.168.2.1 | 192.168.1.1 | ESP | 170 | ESP (SPI=0xc3cea8df) |
| 192.168.1.1 | 192.168.2.1 | ESP | 170 | ESP (SPI=0xc21627c6) |
| 192.168.2.1 | 192.168.1.1 | ESP | 170 | ESP (SPI=0xc3cea8df) |
| 192.168.1.1 | 192.168.2.1 | ESP | 170 | ESP (SPI=0xc21627c6) |
| 192.168.2.1 | 192.168.1.1 | ESP | 170 | ESP (SPI=0xc3cea8df) |

IKE_SA_INIT in the Phase-1 of IKEv2 – the cryptographic parameters and keys are negociated.
IKE_AUTH is the Phase-2, R1 and R2 authenticated eachother and the SAs have been negociated for the Encapsulating Security Payload protocol (ESP).
Next, we see an ESP packet exchange. The data is encrypted and encapsulated by the IPsec protocol. The  two different SPIs are used to identify an SA and make it possible to differentiate the encrypted flows for each direction.
Thanks to the presence of ESP packets, we can confirm that the IPsec link is activated and that the data is transmitted securely between the two routers.
Each ESP packet contains a unique sequence number that is incremented for each packet sent in the same SA. If a packet with the same number is already received or not in the right order, it is discarded (a security feature). The sequence number helps identify and reassemble packets in the right order at the destination. If packets arrive out of order, the IPsec stack uses the sequence numbers to put the packets back in order before forwarding them to the upper layer (similar to TCP). In addition, the sequence number is included in the HMAC calculation, which ensures data integrity.

Below are captures of the `ipsec` debug messages. We can see that R1 generates an IKE_SA_INIT message to initiate the negotiation with R1. It includes the security parameters (SA), the public key (KE), and other extensions.

```
05[CFG] added configuration 'IKEv2-PSK'
08[CFG] received stroke: initiate 'IKEv2-PSK'
08[IKE] initiating IKE_SA IKEv2-PSK[1] to 192.168.2.1
08[ENC] generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP) N(HASH_
ALG) N(REDIR_SUP) ]
08[NET] sending packet: from 192.168.1.1[500] to 192.168.2.1[500] (936 bytes)
```

Next, R2 accepted the negocation with the corresponding parameters.

```
08[NET] sending packet: from 192.168.1.1[500] to 192.168.2.1[500] (936 bytes)
09[NET] received packet: from 192.168.2.1[500] to 192.168.1.1[500] (280 bytes)
09[ENC] parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP) N(HASH_ALG
) N(CHDLESS_SUP) N(MULT_AUTH) ]
09[CFG] selected proposal: IKE:AES_CBC_128/HMAC_SHA2_256_128/PRF_AES128_XCBC/ECP_256
```

Then R1 sends an IKE_AUTH to authenticate himself to R2 – so, the PSK proves our identity and the address range for the tunnel.

```
09[IKE] authentication of '192.168.1.1' (myself) with pre-shared key
09[IKE] establishing CHILD_SA IKEv2-PSK{1}
09[ENC] generating IKE_AUTH request 1 [ IDi N(INIT_CONTACT) IDr AUTH SA TSi TSr N(MULT_AUTH) N(EA
P_ONLY) N(MSG_ID_SYN_SUP) ]
09[NET] sending packet: from 192.168.1.1[500] to 192.168.2.1[500] (304 bytes)
```

Here we see that in response, R2 also authenticates to R1, who confirms it.

```
09[NET] sending packet: from 192.168.1.1[500] to 192.168.2.1[500] (304 bytes)
10[NET] received packet: from 192.168.2.1[500] to 192.168.1.1[500] (208 bytes)
10[ENC] parsed IKE_AUTH response 1 [ IDr AUTH SA TSi TSr N(AUTH_LFT) ]
10[IKE] authentication of '192.168.2.1' with pre-shared key successful
10[IKE] IKE_SA IKEv2-PSK[1] established between 192.168.1.1[192.168.1.1]...192.168.2.1[192.168.2.
1]
10[IKE] scheduling reauthentication in 3335s
```

The CHILD_SA, used to encrypt traffic through ESP, was established for the inbound traffic to R1

```
10[CFG] selected proposal: ESP:AES_CBC_128/HMAC_SHA2_256_128/NO_EXT_SEQ
10[IKE] CHILD_SA IKEv2-PSK{1} established with SPIs ce1472cd_i c340bafa_o and TS 192.168.1.1/32 =
== 192.168.2.1/32
10[IKE] received AUTH_LIFETIME of 3406s, scheduling reauthentication in 3226s
```

A CHILD_SA has been successfully established to handle encrypted traffic between the two peers. This is the 2nd CHILD_SA created for this connection. The SPIs of the old SA are no longer valid and will be deleted once the new tunnel is active.

```
15[CFG] selected proposal: ESP:AES_CBC_128/HMAC_SHA2_256_128/NO_EXT_SEQ
15[IKE] CHILD_SA IKEv2-PSK{2} established with SPIs c6b3d571_i ca8dce99_o and TS 192.168.1.1/32 =
== 192.168.2.1/32
```

Exercise 2.

```
root@R1:~# ipsec statusall                                                          17:48:23 [0/11]
Status of IKE charon daemon (strongSwan 5.8.2, Linux 6.8.0-52-generic, x86_64):
  uptime: 19 minutes, since Feb 12 16:28:31 2025
  malloc: sbrk 2744320, mmap 0, used 668016, free 2076304
  worker threads: 11 of 16 idle, 5/0/0/0 working, job queue: 0/0/0/0, scheduled: 5
  loaded plugins: charon aesni aes rc2 sha2 sha1 md5 mgf1 random nonce x509 revocation constraint
s pubkey pkcs1 pkcs7 pkcs8 pkcs12 pgp dnskey sshkey pem openssl fips-prf gmp agent xcbc hmac gcm
drbg attr kernel-netlink resolve socket-default connmark stroke updown eap-mschapv2 xauth-generic
 counters
Listening IP addresses:
  192.168.1.1
  10.0.0.1
Connections:
   IKEv2-PSK:  192.168.1.1...192.168.2.1  IKEv2
   IKEv2-PSK:   local:  [192.168.1.1] uses pre-shared key authentication
   IKEv2-PSK:   remote: [192.168.2.1] uses pre-shared key authentication
   IKEv2-PSK:   child:  dynamic === dynamic TUNNEL
Security Associations (1 up, 0 connecting):
   IKEv2-PSK[2]: ESTABLISHED 19 minutes ago, 192.168.1.1[192.168.1.1]...192.168.2.1[192.168.2.1]
   IKEv2-PSK[2]: IKEv2 SPIs: 7b90cd287f70c613_i 5f978542b6d3cabc_r*, pre-shared key reauthenticat
ion in 36 minutes
   IKEv2-PSK[2]: IKE proposal: AES_CBC_128/HMAC_SHA2_256_128/PRF_AES128_XCBC/ECP_256
   IKEv2-PSK{3}:  INSTALLED, TUNNEL, reqid 1, ESP SPIs: c8df9e28_i c19fd452_o
   IKEv2-PSK{3}:  AES_CBC_128/HMAC_SHA2_256_128, 0 bytes_i, 0 bytes_o, rekeying in 9 minutes
   IKEv2-PSK{3}:   192.168.1.1/32 === 192.168.2.1/32
```

| IKE | | ESP | |
|---|---|---|---|
| Encryption | AES_CBC_128 | Encryption | AES_CBC_128 |
| Hash | HMAC_SHA2_256_128 | | |
| | | | |
| PRF | PRF_AES128_XCBC | Authentication | HMAC_SHA2_256_128 |
| Key exchange | ECP_256 (elliptic curve DH) | | |

Tunnel mode is used – the entire IP packet is encapsulated and encrypted.

```
INSTALLED, TUNNEL, reqid 1, ESP SPIs: […]
```

```
root@R1:~# ipsec listalgs

List of registered IKE algorithms:

  encryption: AES_CBC[aesni] AES_ECB[aesni] AES_CTR[aesni] RC2_CBC[rc2] 3DES_CBC[openssl] CAMELLIA_CBC[openssl]
              CAST_CBC[openssl] BLOWFISH_CBC[openssl] DES_CBC[openssl] DES_ECB[openssl] NULL[openssl]
  integrity:  AES_XCBC_96[aesni] AES_CMAC_96[aesni] HMAC_MD5_96[openssl] HMAC_MD5_128[openssl] HMAC_SHA1_96[openssl]
              HMAC_SHA1_128[openssl] HMAC_SHA1_160[openssl] HMAC_SHA2_256_128[openssl] HMAC_SHA2_256_256[openssl]
              HMAC_SHA2_384_192[openssl] HMAC_SHA2_384_384[openssl] HMAC_SHA2_512_256[openssl]
              HMAC_SHA2_512_512[openssl] CAMELLIA_XCBC_96[xcbc]
  aead:       AES_CCM_8[aesni] AES_CCM_12[aesni] AES_CCM_16[aesni] AES_GCM_8[aesni] AES_GCM_12[aesni] AES_GCM_16[aesni
]
              CHACHA20_POLY1305[openssl]
  hasher:     HASH_SHA1[sha1] HASH_SHA2_224[sha2] HASH_SHA2_256[sha2] HASH_SHA2_384[sha2] HASH_SHA2_512[sha2]
              HASH_MD5[md5] HASH_MD4[openssl] HASH_IDENTITY[openssl]
  prf:        PRF_AES128_XCBC[aesni] PRF_AES128_CMAC[aesni] PRF_KEYED_SHA1[sha1] PRF_HMAC_MD5[openssl]
              PRF_HMAC_SHA1[openssl] PRF_HMAC_SHA2_256[openssl] PRF_HMAC_SHA2_384[openssl] PRF_HMAC_SHA2_512[openssl]
              PRF_FIPS_SHA1_160[fips-prf] PRF_CAMELLIA128_XCBC[xcbc]
  xof:        XOF_MGF1_SHA1[mgf1] XOF_MGF1_SHA224[mgf1] XOF_MGF1_SHA256[mgf1] XOF_MGF1_SHA384[mgf1]
              XOF_MGF1_SHA512[mgf1]
  drbg:       DRBG_CTR_AES128[drbg] DRBG_CTR_AES192[drbg] DRBG_CTR_AES256[drbg] DRBG_HMAC_SHA1[drbg]
              DRBG_HMAC_SHA256[drbg] DRBG_HMAC_SHA384[drbg] DRBG_HMAC_SHA512[drbg]
  dh-group:   ECP_256[openssl] ECP_384[openssl] ECP_521[openssl] ECP_224[openssl] ECP_192[openssl] ECP_256_BP[openssl]
              ECP_384_BP[openssl] ECP_512_BP[openssl] ECP_224_BP[openssl] MODP_3072[openssl] MODP_4096[openssl]
              MODP_6144[openssl] MODP_8192[openssl] MODP_2048[openssl] MODP_2048_224[openssl] MODP_2048_256[openssl]
              MODP_1536[openssl] MODP_1024[openssl] MODP_1024_160[openssl] MODP_768[openssl] MODP_CUSTOM[openssl]
              CURVE_25519[openssl] CURVE_448[openssl]
  random-gen: RNG_WEAK[openssl] RNG_STRONG[random] RNG_TRUE[random]
  nonce-gen:  [nonce]
```

In `ipsec.conf`, in both containers, we will add the following to the `conn IKEv2-PSK` category:

```
  ike=aes256ctr-sha2_512-modp3072
  esp=aes256ctr-sha2_256
```

Which sets for the key exchange phase AES-256-CTR for encryption, SHA-512 for integrity and MODP-3072 for DH. In the next phase, for ESP, encryption is assured through AES-256-CTR and integrity through SHA-256.

```
root@R1:~# ipsec statusall
Status of IKE charon daemon (strongSwan 5.8.2, Linux 6.8.0-52-generic, x86_64):
  uptime: 84 seconds, since Feb 12 17:10:41 2025
  malloc: sbrk 2744320, mmap 0, used 658784, free 2085536
  worker threads: 10 of 16 idle, 6/0/0/0 working, job queue: 0/0/0/0, scheduled: 5
  loaded plugins: charon aesni aes rc2 sha2 sha1 md5 mgf1 random nonce x509 revocation constraints pubkey pkcs1 pkcs7
pkcs8 pkcs12 pgp dnskey sshkey pem openssl fips-prf gmp agent xcbc hmac gcm drbg attr kernel-netlink resolve socket-de
fault connmark stroke updown eap-mschapv2 xauth-generic counters
Listening IP addresses:
  192.168.1.1
  10.0.0.1
Connections:
   IKEv2-PSK:  192.168.1.1...192.168.2.1  IKEv2
   IKEv2-PSK:   local:  [192.168.1.1] uses pre-shared key authentication
   IKEv2-PSK:   remote: [192.168.2.1] uses pre-shared key authentication
   IKEv2-PSK:   child:  dynamic === dynamic TUNNEL
Security Associations (1 up, 0 connecting):
   IKEv2-PSK[2]: ESTABLISHED 78 seconds ago, 192.168.1.1[192.168.1.1]...192.168.2.1[192.168.2.1]
   IKEv2-PSK[2]: IKEv2 SPIs: d2dbab3605b8458c_i c2292b8193ac39b2_r*, pre-shared key reauthentication in 52 minutes
   IKEv2-PSK[2]: IKE proposal: AES_CTR_256/HMAC_SHA2_512_256/PRF_HMAC_SHA2_512/MODP_3072
   IKEv2-PSK{2}:  INSTALLED, TUNNEL, reqid 1, ESP SPIs: c41f1d3b_i c34e7adb_o
   IKEv2-PSK{2}:  AES_CTR_256/HMAC_SHA2_256_128, 168 bytes_i (2 pkts, 49s ago), 168 bytes_o (2 pkts, 49s ago), rekeyin
g in 13 minutes
   IKEv2-PSK{2}:   192.168.1.1/32 === 192.168.2.1/32
```

We can switch from ESP to Authentication Header AH mode, where only integrity and authenticity are checked – data is not encrypted. So, in `ipsec.conf`:

```
conn IKEv2-PSK
  authby=psk
  left=192.168.2.1
  right=192.168.1.1
  auto=start
  ike=aes256ctr-sha2_512-modp3072
  # esp=aes256ctr-sha2_256
  ah=sha2_256
```

If we start monitoring capture, restart the connections, and ping, we will soon find in the packet capture that the ICMP packets are visible.

```
 1 0.000000   192.168.2.1   192.168.1.1   ISAKMP   1342 IKE_SA_INIT MID=00 Initiator Request
 2 4.001780   192.168.2.1   192.168.1.1   ISAKMP   1342 IKE_SA_INIT MID=00 Initiator Request
 3 5.879707   192.168.1.1   192.168.2.1   ISAKMP   1342 IKE_SA_INIT MID=00 Initiator Request
 4 5.891868   192.168.2.1   192.168.1.1   ISAKMP    642 IKE_SA_INIT MID=00 Responder Response
 5 5.900740   192.168.1.1   192.168.2.1   ISAKMP    383 IKE_AUTH MID=01 Initiator Request
 6 5.911463   192.168.2.1   192.168.1.1   ISAKMP    291 IKE_AUTH MID=01 Responder Response
 7 11.204513  192.168.2.1   192.168.1.1   ISAKMP   1342 IKE_SA_INIT MID=00 Initiator Request
 8 11.224611  192.168.1.1   192.168.2.1   ISAKMP    642 IKE_SA_INIT MID=00 Responder Response
 9 11.233012  192.168.2.1   192.168.1.1   ISAKMP    375 IKE_AUTH MID=01 Initiator Request
10 11.234116  192.168.1.1   192.168.2.1   ISAKMP    291 IKE_AUTH MID=01 Responder Response
11 21.238537  192.168.1.1   192.168.2.1   ISAKMP    123 INFORMATIONAL MID=02 Initiator Request
12 21.241611  192.168.2.1   192.168.1.1   ISAKMP    115 INFORMATIONAL MID=02 Responder Response
13 30.311787  192.168.1.1   192.168.2.1   ICMP      146 Echo (ping) request  id=0x0a6f, seq=1/25
14 30.311910  192.168.2.1   192.168.1.1   ICMP      146 Echo (ping) reply    id=0x0a6f, seq=1/25
15 31.322982  192.168.1.1   192.168.2.1   ICMP      146 Echo (ping) request  id=0x0a6f, seq=2/51
16 31.323255  192.168.2.1   192.168.1.1   ICMP      146 Echo (ping) reply    id=0x0a6f, seq=2/51
17 32.347090  192.168.1.1   192.168.2.1   ICMP      146 Echo (ping) request  id=0x0a6f, seq=3/76
18 32.347264  192.168.2.1   192.168.1.1   ICMP      146 Echo (ping) reply    id=0x0a6f, seq=3/76
19 33.370730  192.168.1.1   192.168.2.1   ICMP      146 Echo (ping) request  id=0x0a6f, seq=4/10
20 33.370899  192.168.2.1   192.168.1.1   ICMP      146 Echo (ping) reply    id=0x0a6f, seq=4/10
21 34.395316  192.168.1.1   192.168.2.1   ICMP      146 Echo (ping) request  id=0x0a6f, seq=5/12
```

Authentication is done by AH - an extra header. Below is the content of the packet highlighted above, from R1 to R2; here we can identify the SPI and see that it coincides with R2's inbound SPI; same applies for the rest.

```
▸ Frame 13: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits
▸ Ethernet II, Src: Xensource_24:6a:fa (00:16:3e:24:6a:fa), Dst: Xensour
▸ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.2.1
▾ Authentication Header
    Next header: IPIP (4)
    Length: 5 (28 bytes)
    Reserved: 0000
    AH SPI: 0xc478cd8a
    AH Sequence: 1
    AH ICV: 3f8ba78c5875bf32ba0ec30474ccf3fa
▸ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.2.1
▸ Internet Control Message Protocol
```

```
KE] IKE_SA IKEv2-PSK[1] established between 192.168.2.1[192.168.2.1]...192.168.1.1[192.168.1.1]
KE] scheduling reauthentication in 3299s
KE] maximum IKE_SA lifetime 3479s
FG] selected proposal: AH:HMAC_SHA2_256_128/NO_EXT_SEQ
KE] CHILD_SA IKEv2-PSK{2} established with SPIs c478cd8a_i c5be9471_o and TS 192.168.2.1/32 === 192.168.1.1/32
KE] received AUTH_LIFETIME of 3318s, scheduling reauthentication in 3138s
ET] received packet: from 192.168.1.1[500] to 192.168.2.1[500] (81 bytes)
```

We can also see that the protocol in IPv4 is 51, therefore AH.

```
Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.2.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 132
    Identification: 0xf254 (62036)
    010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: Authentication Header (51)
    Header Checksum: 0xc39f [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.1
    Destination Address: 192.168.2.1
Authentication Header
Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.2.1
Internet Control Message Protocol
```

AH guarantees that the packet has not been modified in transit, so it includes a cryptographic hash of the IP headers and contents of the packet, but has no encryption.

## Exercise 3.

We will now add a client to each subnet and configure their network interfaces.

```
containers=("C1" "C2")
bridges=("br1" "br2")
for i in "${!containers[@]}"; do
  cont=${containers[$i]}
  br=${bridges[$i]}
  lxc config device add C1 eth0 nic nictype=bridged parent=$br
  lxc file push "$(pwd)/$cont/50-cloud-init.yaml" "$cont/etc/netplan/50-cloud-init.yaml"
  lxc exec $cont -- netplan apply
done
```

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      dhcp6: no
      addresses:
        - 192.168.1.10/24
      gateway4: 192.168.1.1
```

*C1 netplan*

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      dhcp6: no
      addresses:
        - 192.168.2.10/24
      gateway4: 192.168.2.1
```

*C2 netplan*

```
root Feb 13 10:53 .../alex/SystemsSecurity/TP3-IPSec > lxc ls
+------+---------+-----------------------+------+-----------+-----------+
| NAME |  STATE  |         IPV4          | IPV6 |   TYPE    | SNAPSHOTS |
+------+---------+-----------------------+------+-----------+-----------+
| C1   | RUNNING | 192.168.1.10 (eth0)   |      | CONTAINER | 0         |
+------+---------+-----------------------+------+-----------+-----------+
| C2   | RUNNING | 192.168.2.10 (eth0)   |      | CONTAINER | 0         |
+------+---------+-----------------------+------+-----------+-----------+
| R1   | RUNNING | 192.168.1.1 (eth0)    |      | CONTAINER | 0         |
|      |         | 10.0.0.1 (eth1)       |      |           |           |
+------+---------+-----------------------+------+-----------+-----------+
| R2   | RUNNING | 192.168.2.1 (eth0)    |      | CONTAINER | 0         |
|      |         | 10.0.0.2 (eth1)       |      |           |           |
+------+---------+-----------------------+------+-----------+-----------+
```

Next, modifications on the routers are needed for their IPsec configs. Previously we simply set the 2 sides, left and right, to the internal IP addresses and the tunneling simply worked – that's because we had IP forwarding enabled. Packets coming from internal subnets are redirected via external interfaces, so `strongswan` can still establish an IPsec tunnel from these addresses.
So, initially, strongswan used the internal addresses as endpoints for the tunnel. To further isolate IPSEC connections and clarify the role of the interfaces, we modify the configs by putting the IPs of the external interfaces. (`10.0.0.x`) for left and right, respectively, then defined the corresponding subnets with leftsubnet and rightsubnet (`192.168.x.0/24`). Next, because we change the endpoints definitions, we need to change/add a new (or reuse the previous) PSK for the external interface IPs.

```
# which knows the public part.

192.168.1.1 192.168.2.1 : PSK "97a7c4ca924ab8642c1d8e94395752e2982ed0a6ffef2d71560f0208b061a3ed"
10.0.0.1 10.0.0.2 : PSK "97a7c4ca924ab8642c1d8e94395752e2982ed0a6ffef2d71560f0208b061a3ed"
~
```

```

conn %default
  ikelifetime=60m
  keylife=20m
  rekeymargin=3m
  keyingtries=1
  keyexchange=ikev2
  mobike=no
conn IKEv2-PSK
  authby=psk
  left=10.0.0.1
  leftsubnet=192.168.1.0/24
  right=10.0.0.2
  rightsubnet=192.168.2.0/24
  auto=start
  ike=aes256ctr-sha2_512-modp3072
  esp=aes256ctr-sha2_256
```

*R1 ipsec.conf*

```
# ipsec.conf - strongSwan IPsec con

conn %default
  ikelifetime=60m
  keylife=20m
  rekeymargin=3m
  keyingtries=1
  keyexchange=ikev2
  mobike=no
conn IKEv2-PSK
  authby=psk
  left=10.0.0.2
  leftsubnet=192.168.2.0/24
  right=10.0.0.1
  rightsubnet=192.168.1.0/24
  auto=start
  ike=aes256ctr-sha2_512-modp3072
  esp=aes256ctr-sha2_256
```

*R2 ipsec.conf*

If we start a new network capture, creating the tunnel and then pinging C2 from C1 will result in the following:

| Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|
| 10.0.0.2 | 10.0.0.1 | ISAKMP | 1342 | IKE_SA_INIT MID=00 Initiator Request |
| 10.0.0.1 | 10.0.0.2 | ISAKMP | 1342 | IKE_SA_INIT MID=00 Initiator Request |
| 10.0.0.2 | 10.0.0.1 | ISAKMP | 642 | IKE_SA_INIT MID=00 Responder Response |
| 10.0.0.1 | 10.0.0.2 | ISAKMP | 431 | IKE_AUTH MID=01 Initiator Request |
| 10.0.0.2 | 10.0.0.1 | ISAKMP | 303 | IKE_AUTH MID=01 Responder Response |
| 10.0.0.2 | 10.0.0.1 | ISAKMP | 1342 | IKE_SA_INIT MID=00 Initiator Request |
| 10.0.0.1 | 10.0.0.2 | ISAKMP | 642 | IKE_SA_INIT MID=00 Responder Response |
| 10.0.0.2 | 10.0.0.1 | ISAKMP | 423 | IKE_AUTH MID=01 Initiator Request |
| 10.0.0.1 | 10.0.0.2 | ISAKMP | 303 | IKE_AUTH MID=01 Responder Response |
| 10.0.0.1 | 10.0.0.2 | ISAKMP | 123 | INFORMATIONAL MID=02 Initiator Reques |
| 10.0.0.2 | 10.0.0.1 | ISAKMP | 115 | INFORMATIONAL MID=02 Responder Respon |
| 10.0.0.1 | 10.0.0.2 | ESP | 154 | ESP (SPI=0xc2c5c3a8) |
| 10.0.0.2 | 10.0.0.1 | ESP | 154 | ESP (SPI=0xc6314678) |
| 10.0.0.1 | 10.0.0.2 | ESP | 154 | ESP (SPI=0xc2c5c3a8) |
| 10.0.0.2 | 10.0.0.1 | ESP | 154 | ESP (SPI=0xc6314678) |
| 10.0.0.1 | 10.0.0.2 | ESP | 154 | ESP (SPI=0xc2c5c3a8) |
| 10.0.0.2 | 10.0.0.1 | ESP | 154 | ESP (SPI=0xc6314678) |
| 10.0.0.1 | 10.0.0.2 | ESP | 154 | ESP (SPI=0xc2c5c3a8) |
| 10.0.0.2 | 10.0.0.1 | ESP | 154 | ESP (SPI=0xc6314678) |
| 10.0.0.1 | 10.0.0.2 | ISAKMP | 123 | INFORMATIONAL MID=00 Responder Reques |
| 10.0.0.2 | 10.0.0.1 | ISAKMP | 115 | INFORMATIONAL MID=00 Initiator Respon |

We only see the IPs of the routers. In fact, the internal headers are encapsulated and encrypted in the ESP. In tunnel mode, the entire packet (IP header + data) is encapsulated and protected in a new packet with a new IP header.

# 2 Authentication based on certificates

*Prerequisites.*

Configuration file – `openca.cnf` for creating the root CA:

```
[ req ]
distinguished_name = req_distinguished_name
x509_extensions = v3_ca
prompt = no

[ req_distinguished_name ]
C = FR
O = ipsec
CN = IPsecCA

[ v3_ca ]
keyUsage = critical, digitalSignature, keyCertSign, cRLSign
basicConstraints = critical,CA:true
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
```

*openca.cnf*

Configuration file - `openssl.cnf` for the R1 / R2 certificates (the only difference is the CN, Common Name):

```
[ req ]
distinguished_name = req_distinguished_name
x509_extensions = v3_ca
prompt = no

[ req_distinguished_name ]
C = FR
O = ipsec
CN = 192.168.1.1

[ v3_ca ]
keyUsage = digitalSignature, nonRepudiation, keyEncipherment
basicConstraints = critical,CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
```

*openssl.cnf*

The process of enabling certificate authentication consists in reating a root Certificate Authority: a private-public key pair and a self signed x509 certificate for it. We continue by having R1 and R2 each create their own key pair and send us a a certificate signing request (CSR) that the root CA will sign and hand them back their certificates. We didn't go exactly on this route – we are going to be the ones that distribute everything (specifically the private keys).

It's important to mention the paths where IPsec manages the needed files on each host:

- `/etc/ipsec.d/private` for the private keys
- `/etc/ipsec.d/certs` for the certificates
- `/etc/ipsec.d/cacerts` for the root Cas

```
# create root CA
openssl genrsa -out "ca.key" 2048
openssl req -x509 -new -nodes -key "ca.key" -sha256 -days 3650 -config "openca.cnf" -extensions v3_c
a -out "ca.crt"

for i in "${!containers[@]}"; do
  cont=${containers[$i]}
  name=${names[$i]}
  # create RSA key pair, make a certificate request and then use the CA to sign it
  openssl genrsa -out "$cont/$name.key" 2048
  openssl req -new -key "$cont/$name.key" -out "$cont/$name.csr" -config "$cont/openssl.cnf"
  openssl x509 -req -in "$cont/$name.csr" -CA ca.crt -CAkey ca.key -CAcreateserial -out "$cont/$name
.crt" -days 365 -sha256 -extensions v3_ca -extfile "$cont/openssl.cnf"

  # push the needed files to the corresponding host
  lxc file push "$(pwd)/$cont/$name.key" "$cont/etc/ipsec.d/private/$name.key"
  lxc file push "$(pwd)/$cont/$name.crt" "$cont/etc/ipsec.d/certs/$name.crt"
  lxc file push "$(pwd)/ca.crt" "$cont/etc/ipsec.d/cacerts/ca.crt"
done
```

Finally, we can modify the `/etc/ipsec.secrets` on each router to use the private keys:

```
# RSA private key for this host, a
# which knows the public part.

# 192.168.1.1 192.168.2.1 : PSK "9
# 10.0.0.1 10.0.0.2 : PSK "97a7c4
10.0.0.1 10.0.0.2 : RSA r1.key
```

```
# RSA private key for this host, authenticating
# which knows the public part.

# 192.168.2.1 192.168.1.1 : PSK "97a7c4ca924ab86
# 10.0.0.2 10.0.0.1 : PSK "97a7c4ca924ab8642c1d8
10.0.0.2 10.0.0.1 : RSA r2.key
```

As well as modify the `ipsec.conf`:

```
# ipsec.conf - strongSwan IPsec config

conn %default
  ikelifetime=60m
  keylife=20m
  rekeymargin=3m
  keyingtries=1
  keyexchange=ikev2
  mobike=no
conn IKEv2-PSK
  #authby=psk
  authby=pubkey
  left=10.0.0.1
  leftsubnet=192.168.1.0/24
  right=10.0.0.2
  rightsubnet=192.168.2.0/24
  auto=start
  leftcert=r1.crt
  leftid="C=FR, O=ipsec, CN=192.168.1.1"
  rightid="C=FR, O=ipsec, CN=192.168.2.1"
  ike=aes256ctr-sha2_512-modp3072
  esp=aes256ctr-sha2_256
```

*R1 ipsec.conf*

```
# ipsec.conf - strongSwan IPsec config

conn %default
  ikelifetime=60m
  keylife=20m
  rekeymargin=3m
  keyingtries=1
  keyexchange=ikev2
  mobike=no
conn IKEv2-PSK
  # authby=psk
  authby=pubkey
  left=10.0.0.2
  leftsubnet=192.168.2.0/24
  right=10.0.0.1
  rightsubnet=192.168.1.0/24
  auto=start
  leftcert=r2.crt
  leftid="C=FR, O=ipsec, CN=192.168.2.1"
  rightid="C=FR, O=ipsec, CN=192.168.1.1"
  ike=aes256ctr-sha2_512-modp3072
  esp=aes256ctr-sha2_256
```

*R2 ipsec.conf*

We note the addition of `letfcert=..` to specify the certificate used for the router. `leftid` specifies the identity of the certificate (Country, Organization and Common Name) and the same for `rightid`, for the one opposite. `authby=pubkey` means that authentication between peers is done via public keys, provided by the X.509 certificates.

```
05[IKE] received end entity cert "C=FR, O=ipsec, CN=192.168.2.1"
05[CFG]   using certificate "C=FR, O=ipsec, CN=192.168.2.1"
05[CFG]   using trusted ca certificate "C=FR, O=ipsec, CN=IPsecCA"
05[CFG]   reached self-signed root ca with a path length of 0
05[CFG] checking certificate status of "C=FR, O=ipsec, CN=192.168.2.1"
05[CFG] certificate status is not available
05[IKE] authentication of 'C=FR, O=ipsec, CN=192.168.2.1' with RSA_EMSA_PKCS1_SHA2_256 successful
05[IKE] IKE_SA IKEv2-PSK[1] established between 10.0.0.1[C=FR, O=ipsec, CN=192.168.1.1]...10.0.0.2[C=FR,
 O=ipsec, CN=192.168.2.1]
05[IKE] scheduling reauthentication in 3368s
05[IKE] maximum IKE_SA lifetime 3548s
```

```
$ ipsec statusall
```

```
el-netlink resolve socket-default connmark stroke updown eap-mschapv2 xauth-generic counters
Listening IP addresses:
  192.168.1.1
  10.0.0.1
Connections:
   IKEv2-PSK:  10.0.0.1...10.0.0.2  IKEv2
   IKEv2-PSK:   local:  [C=FR, O=ipsec, CN=192.168.1.1] uses public key authentication
   IKEv2-PSK:    cert:  "C=FR, O=ipsec, CN=192.168.1.1"
   IKEv2-PSK:   remote: [C=FR, O=ipsec, CN=192.168.2.1] uses public key authentication
   IKEv2-PSK:   child:  192.168.1.0/24 === 192.168.2.0/24 TUNNEL
Security Associations (1 up, 0 connecting):
   IKEv2-PSK[1]: ESTABLISHED 89 seconds ago, 10.0.0.1[C=FR, O=ipsec, CN=192.168.1.1]...10.0.0.2[C=FR, O=
ipsec, CN=192.168.2.1]
   IKEv2-PSK[1]: IKEv2 SPIs: 547c223a587579b7_i* ca30ea3446dfaeaf_r, public key reauthentication in 50 m
inutes
   IKEv2-PSK[1]: IKE proposal: AES_CTR_256/HMAC_SHA2_512_256/PRF_HMAC_SHA2_512/MODP_3072
   IKEv2-PSK{2}:  INSTALLED, TUNNEL, reqid 1, ESP SPIs: c6880121_i c6abe474_o
   IKEv2-PSK{2}:  AES_CTR_256/HMAC_SHA2_256_128, 0 bytes_i, 0 bytes_o, rekeying in 14 minutes
   IKEv2-PSK{2}:   192.168.1.0/24 === 192.168.2.0/24
```

Exercise 4.

Host-to-Net (Roadwarrior, **rw**) connections are being used to connect a host which could be a laptop, smartphone or any other device with an IPsec client to one or more networks. In this exercise we wish to have a “nomad” laptop be able to authenticate upon R1 and access the 2 subnetworks.

```
$ lxc launch ubuntu:20.04 nomad
$ lxc exec nomad -- apt update -y
$ lxc exec nomad -- apt install strongswan strongswan-pki -y
```

Then we will create a new bridge linking the nomad to R1:

```
$ lxc network create brnomad ipv4.address=none ipv6.address=none
$ lxc config device add nomad eth0 nic nictype=bridged
  parent=brnomad
$ lxc config device add R1 eth2 nic nictype=bridged parent=bnomad
```

On the brnomad connection:

- Nomad will be at 192.168.3.1
- R1 will be at 192.168.3.2

For this, we will first configure nomad’s netplan:

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      dhcp6: no
      addresses:
        - 192.168.3.1/24
      routes:
        - to: 192.168.1.0/24
          via: 192.168.3.2
        - to: 192.168.2.0/24
          via: 192.168.3.2
```

To R1 we add the new eth2 interface and route all traffic for the nomad's network to him.

```
      routes:
        - to: 192.168.2.0/24
          via: 10.0.0.2
    eth2:
      dhcp4: no
      dhcp6: no
      addresses:
        - 192.168.3.2
      routes:
        - to: 192.168.3.0/24
          via: 192.168.3.1
```

R2's network config also needs alterations for the eth1 interface to become aware of the new network:

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      dhcp6: no
      addresses:
        - 192.168.2.1/24
    eth1:
      dhcp4: no
      dhcp6: no
      addresses:
        - 10.0.0.2/30
      routes:
        - to: 192.168.1.0/24
          via: 10.0.0.1
        - to: 192.168.3.0/24
          via: 10.0.0.1
```

Now we can move on to configuring IPsec for the nomad:

```
conn %default
  ikelifetime=60m
  keylife=20m
  rekeymargin=3m
  keyingtries=1
  keyexchange=ikev2
# mobike=no
conn rw
  authby=pubkey
  left=192.168.3.1
  right=192.168.3.2
  rightsubnet=192.168.1.0/24,192.168.2.0/24
  auto=start
  leftcert=nomad.crt
  leftid="C=FR, O=ipsec, CN=nomad"
  rightid="C=Fr, O=ipsec, CN=192.168.1.1"
  ike=aes256ctr-sha2_512-modp3072
  esp=aes256ctr-sha2_256
```

Here, we can see that the `rightsubnet` includes the subnets of R1 and R2 - because nomad must be able to transfer encrypted packets via the tunnel to both R1 and R2. Since authentication is done with R1, it only needs the `distinguished_name` of R1's certificate (`rightid`).

Below we further modify the Ipsec config for R1 - the `leftsubnet` of `rw`, in order to add the subnet 192.168.3.0/24 so it becomes accessible via IPsec. The data sent from nomad to R2 is encrypted: this traffic will pass through the tunnel between R1 and R2 and thus will be encrypted.

```
conn %default
  ikelifetime=60m
  keylife=20m
  rekeymargin=3m
  keyingtries=1
  keyexchange=ikev2
  mobike=no
conn rw
  #authby=psk
  authby=pubkey
  left=10.0.0.1
  leftsubnet=192.168.1.0/24,192.168.3.0/24
  right=10.0.0.2
  rightsubnet=192.168.2.0/24
  auto=start
  leftcert=r1.crt
  leftid="C=FR, O=ipsec, CN=192.168.1.1"
  rightid="C=FR, O=ipsec, CN=192.168.2.1"
  ike=aes256ctr-sha2_512-modp3072
  esp=aes256ctr-sha2_256
conn rw_nomad
  authby=pubkey
  left=192.168.3.2
  leftsubnet=192.168.1.0/24,192.168.2.0/24
  right=192.168.3.1
  auto=start
  leftcert=r1.crt
  leftid="C=FR, O=ipsec, CN=192.168.1.1"
  rightid="C=FR, O=ipsec, CN=nomad"
  ike=aes256ctr-sha2_512-modp3072
  esp=aes256ctr-sha2_256
```

We also add a nomad `rw` connection in order to establish a tunnel between nomad and R1. `leftsubnet` contains the subnet of R1 and R2 because both must be able to receive encrypted packets from nomad. `right` is directly the IP address of nomad, `leftid` and `rightid` allow authentication between nomad and R1.
For R2 we add the new subnet to `rightsubnets`:

```
# ipsec.conf - strongSwan IPsec config

conn %default
  ikelifetime=60m
  keylife=20m
  rekeymargin=3m
  keyingtries=1
  keyexchange=ikev2
  mobike=no
conn rw
  # authby=psk
  authby=pubkey
  left=10.0.0.2
  leftsubnet=192.168.2.0/24
  right=10.0.0.1
  rightsubnet=192.168.1.0/24,192.168.3.0/24
  auto=start
  leftcert=r2.crt
  leftid="C=FR, O=ipsec, CN=192.168.2.1"
  rightid="C=FR, O=ipsec, CN=192.168.1.1"
  ike=aes256ctr-sha2_512-modp3072
  esp=aes256ctr-sha2_256
```

In the same manner as for the routers, we will create and sign a certificate for the nomad – necessary for R1 to authenticate it. Don't forget to push them to the nomad container, to their respective paths managed by IPsec, and then add the key to nomad's secrets.

```
$ openssl genrsa -out nomad/nomad.key 2048
$ openssl req -new -key nomad/nomad.key -out nomad/nomad.csr -config
  nomad/openssl.cnf
$ openssl x509 -req -in nomad/nomad.csr -CA ca.crt -CAkey ca.key -
  CAcreateserial -out nomad/nomad.crt -days 365 -sha256 -extensions
  v3_ca -extfile nomad/openssl.cnf
```

```
[ req ]
distinguished_name = req_distinguished_name
x509_extensions = v3_ca
prompt = no

[ req_distinguished_name ]
C = FR
O = ipsec
CN = nomad

[ v3_ca ]
keyUsage = digitalSignature, nonRepudiation, keyEncipherment
basicConstraints = critical,CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
```

*Nomad openssl.cnf*

```
# This file holds shared secrets or RSA private keys for authentication.

# RSA private key for this host, authenticating it to any other host
# which knows the public part.
192.168.3.1 192.168.3.2 : RSA nomad.key
~
```

*Nomad ipsec.secrets*