ISS CW2 - Group nobs

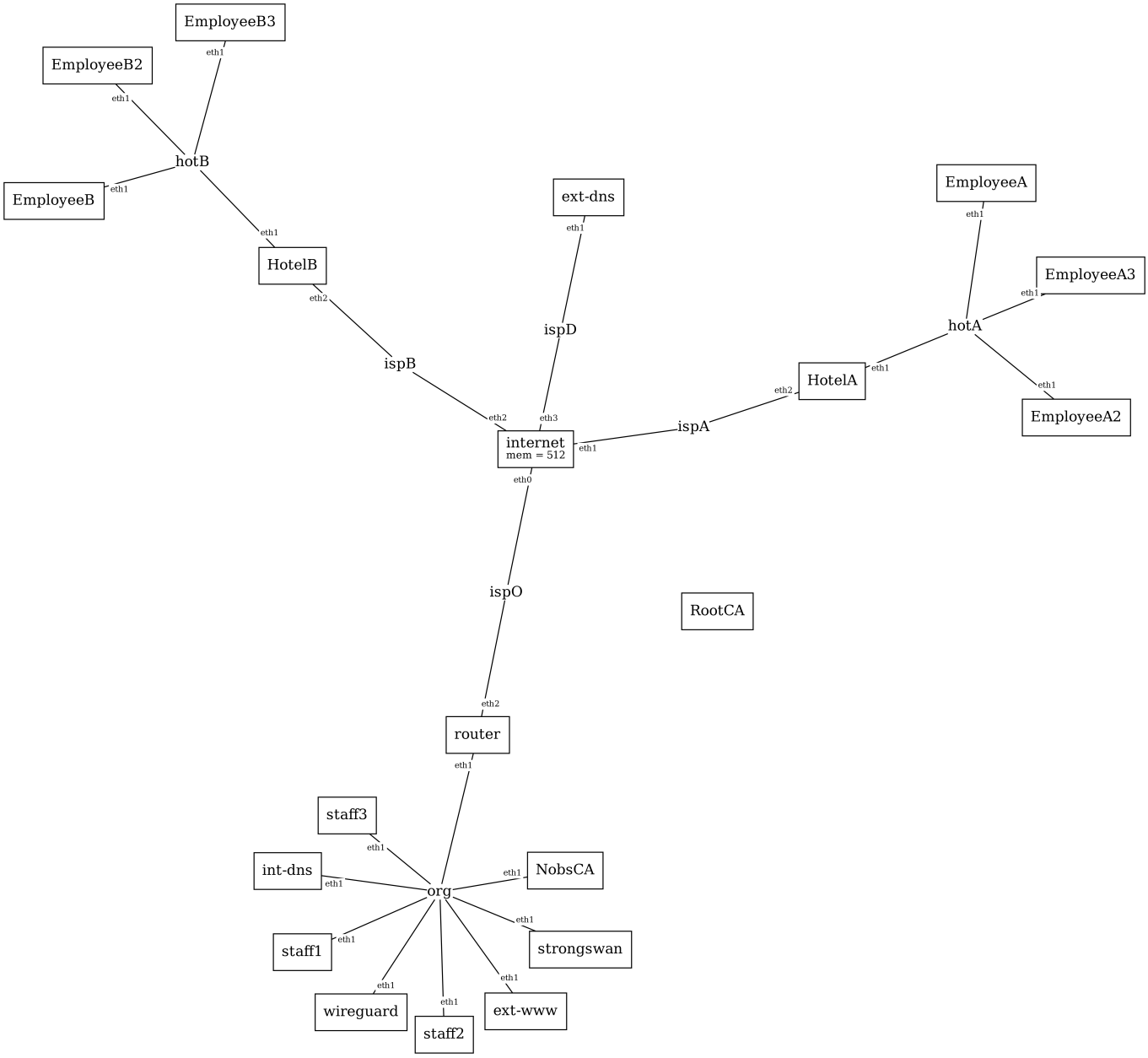
Members:

- Nathan Chamberlain - *u2043025* - `3D7CD14D03DA7867FB097434C332ADB38F4300BC`
- Oscar Cornish - *u2053390* - `AE4D5827E507CCD6BC12051B1202F8B8CBD803A0`
- Ben Smith - *u2027437* - `C7DFCEEDD43774D05F6FF812205C19426EFBFDD8`

Table of contents:

Number after refers to the point in the brief

Network diagram:



- HotelA has three Wireguard clients *(EmployeeA, EmployeeA2, EmployeeA3)*
- HotelB has three Strongswan clients *(EmployeeB, EmployeeB2, EmployeeB3)*

---

x509 Certificate hierarchy:

Certificate structure:

- RootCA
  - NobsCA
    - Strongswan vpn cert
    - EmployeeB cert
    - EmployeeB2 cert
    - EmployeeB3 cert
    - www.nobs.cyber.test cert

Justification:

I decided to go for a structure with a depth of 3 for simplicity, while still demonstrating a realistic scenario. The root certificate authority (`RootCA`) is an offline machine to limit exposure, It has a cert on it that is signed by itself that it uses to sign the certificates of the layer below, in our implementation it is just the `NobsCA` but in reality there would be many more.

The `NobsCA` is refered to as a subordinate CA (As it is signed by another CA) in this scenario the organisation has control over their own CA but it is signed by a public `RootCA` (I'm not sure if this really happens in th real world, I couldn't find anything to suggest it would or would not) therefore they can issue certificates without having to get the Certificate Authority to sign their request, thus they can easily issue and revoke certificates for roadwarrior employees.

Another benefit to the `NobsCA` being signed by the `RootCA` is that (Assuming `RootCA` is deemed a 'trust anchor' by their device) they will implicitly trust the certs issued by `NobsCA`, including the one for the https webserver, avoiding the untrusted warning they would otherwise get.

IP addresses and DNS Records:

External DNS records point to the gateways and the webserver:

```
; <<>> DiG 9.16.15-Debian <<>> gw1.nobs.cyber.test
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6827
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;gw1.nobs.cyber.test.           IN      A

;; ANSWER SECTION:
gw1.nobs.cyber.test.    0       IN      A       213.1.133.98

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Feb 22 16:32:40 UTC 2022
;; MSG SIZE  rcvd: 64

root@ext-dns:~#
; <<>> DiG 9.16.15-Debian <<>> gw2.nobs.cyber.test
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10859
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;gw2.nobs.cyber.test.           IN      A

;; ANSWER SECTION:
gw2.nobs.cyber.test.    0       IN      A       213.1.133.99

;; Query time: 10 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Feb 22 16:32:50 UTC 2022
;; MSG SIZE  rcvd: 64

root@ext-dns:~#
```

```
; <<>> DiG 9.16.15-Debian <<>> www.nobs.cyber.test
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26354
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.nobs.cyber.test.           IN      A

;; ANSWER SECTION:
```

```
www.nobs.cyber.test.    0       IN      A       213.1.133.100

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Feb 22 16:33:58 UTC 2022
;; MSG SIZE  rcvd: 64
```

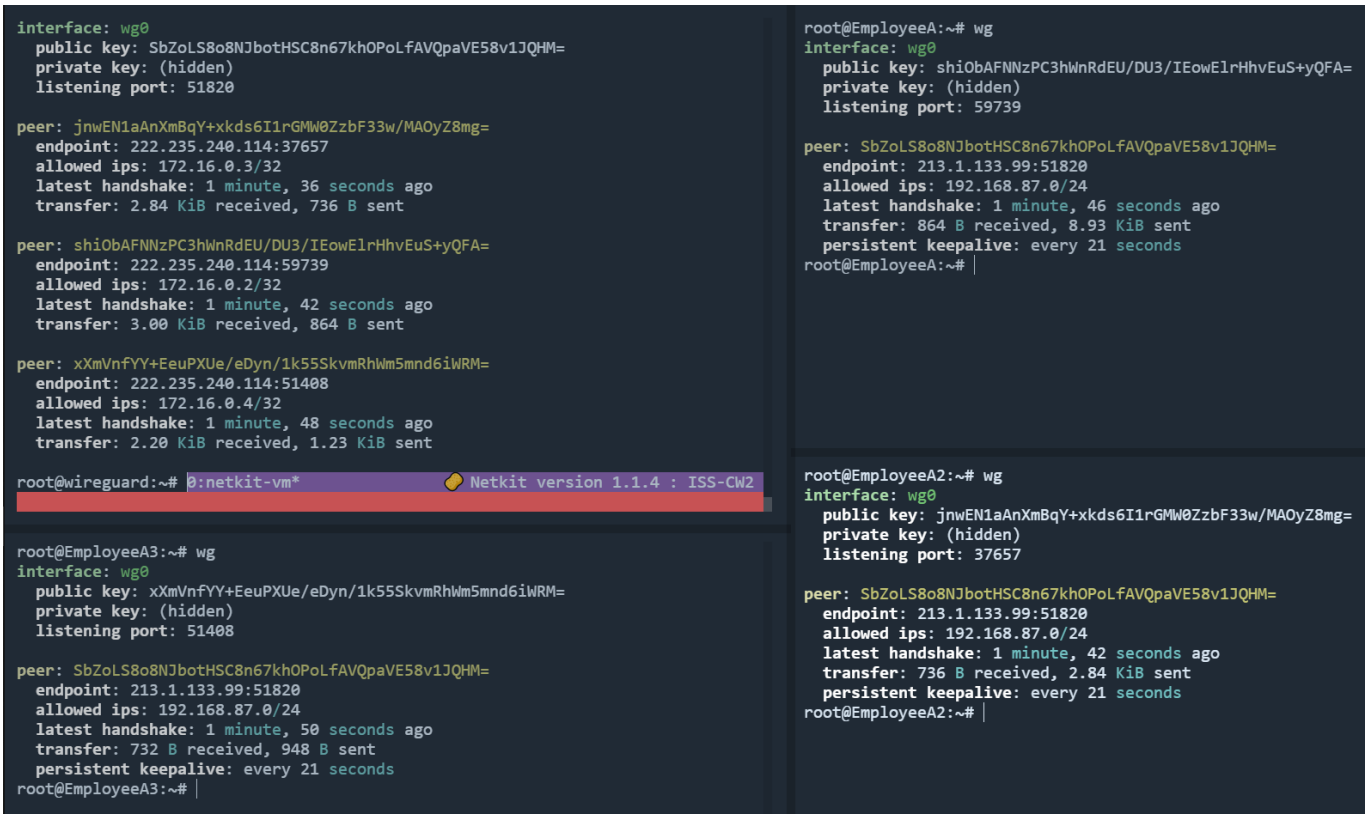
This means the webserver can be accessed externally:

```
root@ext-dns:~# curl www.nobs.cyber.test
Example website for the nobs group
```

VPN Implementation:

Wireguard:

All three wireguard clients can connect to the vpn simultaneously:



These clients can ping the internal network via the wg0 interface:

```
root@EmployeeA:~# ip r get 192.168.87.101
192.168.87.101 dev wg0 src 172.16.0.2 uid 0
    cache
root@EmployeeA:~# ping -c2 192.168.87.101
PING 192.168.87.101 (192.168.87.101) 56(84) bytes of data.
64 bytes from 192.168.87.101: icmp_seq=1 ttl=63 time=1.93 ms
64 bytes from 192.168.87.101: icmp_seq=2 ttl=63 time=3.55 ms

--- 192.168.87.101 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 1.930/2.741/3.553/0.811 ms
root@EmployeeA:~#
```

```
root@EmployeeA2:~# ip r get 192.168.87.102
192.168.87.102 dev wg0 src 172.16.0.3 uid 0
    cache
root@EmployeeA2:~# ping -c2 192.168.87.101
PING 192.168.87.101 (192.168.87.101) 56(84) bytes of data.
64 bytes from 192.168.87.101: icmp_seq=1 ttl=63 time=2.01 ms
64 bytes from 192.168.87.101: icmp_seq=2 ttl=63 time=3.17 ms

--- 192.168.87.101 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 2.012/2.593/3.174/0.581 ms
root@EmployeeA2:~#
```

```
root@EmployeeA3:~# ip r get 192.168.87.103
192.168.87.103 dev wg0 src 172.16.0.4 uid 0
    cache
root@EmployeeA3:~# ping -c2 192.168.87.103
PING 192.168.87.103 (192.168.87.103) 56(84) bytes of data.
64 bytes from 192.168.87.103: icmp_seq=1 ttl=63 time=1.89 ms
64 bytes from 192.168.87.103: icmp_seq=2 ttl=63 time=2.88 ms

--- 192.168.87.103 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 1.893/2.386/2.879/0.493 ms
root@EmployeeA3:~#
```

The traffic between these is encrypted, and the contents cannot be seen by somebody sniffing packets:

1	0.000000	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0x0F15F869, counter=6
2	0.000008	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0x0F15F869, counter=6
3	0.000243	222.235.240.114	213.1.133.99	WireGu…	196	Handshake Initiation, sender=0x85C48B72
4	0.000252	222.235.240.114	213.1.133.99	WireGu…	196	Handshake Initiation, sender=0x85C48B72
5	0.001306	213.1.133.99	222.235.240.114	WireGu…	140	Handshake Response, sender=0x6CCF4C33, receiver=0x85C48B72
6	0.001311	213.1.133.99	222.235.240.114	WireGu…	140	Handshake Response, sender=0x6CCF4C33, receiver=0x85C48B72
7	0.002081	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0x6CCF4C33, counter=0
8	0.002085	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0x6CCF4C33, counter=0
9	0.551579	Schneide_00:00:02		ARP	48	Who has 213.1.133.97? Tell 213.1.133.100
10	0.551615	Schneide_11:00:01		ARP	48	213.1.133.97 is at 00:11:00:11:00:01
11	0.556623	00:aa:aa:11:00:01		ARP	48	Who has 222.235.240.114? Tell 222.235.240.113
12	0.557149	00:aa:aa:10:00:01		ARP	48	222.235.240.114 is at 00:aa:aa:10:00:01
13	16.644863	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xE4923378, counter=1
14	16.644873	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xE4923378, counter=1
15	17.669690	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xC2FE02B8, counter=6
16	17.669703	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xC2FE02B8, counter=6
17	17.669994	222.235.240.114	213.1.133.99	WireGu…	196	Handshake Initiation, sender=0x09634FF3
18	17.670002	222.235.240.114	213.1.133.99	WireGu…	196	Handshake Initiation, sender=0x09634FF3
19	17.671461	213.1.133.99	222.235.240.114	WireGu…	140	Handshake Response, sender=0xC90B2476, receiver=0x09634FF3
20	17.671469	213.1.133.99	222.235.240.114	WireGu…	140	Handshake Response, sender=0xC90B2476, receiver=0x09634FF3
21	17.672583	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xC90B2476, counter=0
22	17.672590	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xC90B2476, counter=0
23	21.120171	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0x6CCF4C33, counter=1
24	21.120181	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0x6CCF4C33, counter=1
25	21.688006	00:aa:aa:10:00:01		ARP	48	Who has 222.235.240.113? Tell 222.235.240.114
26	21.688019	00:aa:aa:11:00:01		ARP	48	222.235.240.113 is at 00:aa:aa:11:00:01
27	37.765337	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xE4923378, counter=2
28	37.765350	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xE4923378, counter=2
29	38.788860	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xC90B2476, counter=1
30	38.788870	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xC90B2476, counter=1
31	42.240144	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0x6CCF4C33, counter=2
32	42.240153	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0x6CCF4C33, counter=2
33	42.796615	Schneide_11:00:01		ARP	48	Who has 213.1.133.99? Tell 213.1.133.97
34	42.797040	Schneide_00:00:02		ARP	48	213.1.133.99 is at 00:11:00:00:00:02
35	52.176427	222.235.240.114	213.1.133.99	WireGu…	176	Transport Data, receiver=0xC90B2476, counter=2, datalen=96
36	52.176435	222.235.240.114	213.1.133.99	WireGu…	176	Transport Data, receiver=0xC90B2476, counter=2, datalen=96
37	52.177639	213.1.133.99	222.235.240.114	WireGu…	176	Transport Data, receiver=0x09634FF3, counter=0, datalen=96
38	52.177646	213.1.133.99	222.235.240.114	WireGu…	176	Transport Data, receiver=0x09634FF3, counter=0, datalen=96
39	53.179068	222.235.240.114	213.1.133.99	WireGu…	176	Transport Data, receiver=0xC90B2476, counter=3, datalen=96
40	53.179079	222.235.240.114	213.1.133.99	WireGu…	176	Transport Data, receiver=0xC90B2476, counter=3, datalen=96
41	53.180837	213.1.133.99	222.235.240.114	WireGu…	176	Transport Data, receiver=0x09634FF3, counter=1, datalen=96
42	53.180845	213.1.133.99	222.235.240.114	WireGu…	176	Transport Data, receiver=0x09634FF3, counter=1, datalen=96
43	57.191477	Schneide_00:00:02		ARP	48	Who has 213.1.133.97? Tell 213.1.133.100
44	57.191489	Schneide_11:00:01		ARP	48	213.1.133.97 is at 00:11:00:11:00:01
45	57.196529	00:aa:aa:11:00:01		ARP	48	Who has 222.235.240.114? Tell 222.235.240.113
46	57.196891	00:aa:aa:10:00:01		ARP	48	222.235.240.114 is at 00:aa:aa:10:00:01
47	57.208481	00:aa:aa:10:00:01		ARP	48	Who has 222.235.240.113? Tell 222.235.240.114
48	57.208490	00:aa:aa:11:00:01		ARP	48	222.235.240.113 is at 00:aa:aa:11:00:01
49	58.885286	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xE4923378, counter=3
50	58.885297	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0xE4923378, counter=3
51	63.360240	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0x6CCF4C33, counter=3
52	63.360250	222.235.240.114	213.1.133.99	WireGu…	80	Keepalive, receiver=0x6CCF4C33, counter=3

Strongswan:

The strongswan clients can also all connect at the same time:

```
Security Associations (3 up, 0 connecting):
   ikev2-vpn[6]: ESTABLISHED 29 seconds ago, 192.168.87.3[C=UK, ST=Coventry, L=Coventry, O=Nobs, CN=Nobs Strongswan]..
.104.212.7.162[C=UK, ST=Coventry, L=Coventry, O=Nobs, CN=EmployeeB3]
   ikev2-vpn[6]: IKEv2 SPIs: 475fc63e173295c0_i e890dfebc1491b8a_r*, public key reauthentication in 2 hours
   ikev2-vpn[6]: IKE proposal: AES_CBC_128/HMAC_SHA2_256_128/PRF_AES128_XCBC/ECP_256
   ikev2-vpn{3}:  INSTALLED, TUNNEL, reqid 3, ESP in UDP SPIs: c45673fe_i c83961c1_o
   ikev2-vpn{3}:  AES_CBC_128/HMAC_SHA2_256_128, 0 bytes_i, 0 bytes_o, rekeying in 45 minutes
   ikev2-vpn{3}:   192.168.87.0/24 === 172.16.1.3/32
   ikev2-vpn[3]: ESTABLISHED 10 minutes ago, 192.168.87.3[C=UK, ST=Coventry, L=Coventry, O=Nobs, CN=Nobs Strongswan]..
.104.212.7.162[C=UK, ST=Coventry, L=Coventry, O=Nobs, CN=EmployeeB2]
   ikev2-vpn[3]: IKEv2 SPIs: 824ae231a171ca6d_i 74c09d4b109e3884_r*, public key reauthentication in 2 hours
   ikev2-vpn[3]: IKE proposal: AES_CBC_128/HMAC_SHA2_256_128/PRF_AES128_XCBC/ECP_256
   ikev2-vpn{2}:  INSTALLED, TUNNEL, reqid 2, ESP in UDP SPIs: cb0bfc7a_i c1ba7346_o
   ikev2-vpn{2}:  AES_CBC_128/HMAC_SHA2_256_128, 0 bytes_i, 0 bytes_o, rekeying in 35 minutes
   ikev2-vpn{2}:   192.168.87.0/24 === 172.16.1.2/32
   ikev2-vpn[2]: ESTABLISHED 10 minutes ago, 192.168.87.3[C=UK, ST=Coventry, L=Coventry, O=Nobs, CN=Nobs Strongswan]..
.104.212.7.162[C=UK, ST=Coventry, L=Coventry, O=Nobs, CN=EmployeeB]
   ikev2-vpn[2]: IKEv2 SPIs: d02ca420f59aa5ea_i 90b31ff742d53d49_r*, public key reauthentication in 2 hours
   ikev2-vpn[2]: IKE proposal: AES_CBC_128/HMAC_SHA2_256_128/PRF_AES128_XCBC/ECP_256
   ikev2-vpn{1}:  INSTALLED, TUNNEL, reqid 1, ESP in UDP SPIs: c35a9b34_i c4bdf07b_o
   ikev2-vpn{1}:  AES_CBC_128/HMAC_SHA2_256_128, 0 bytes_i, 0 bytes_o, rekeying in 36 minutes
   ikev2-vpn{1}:   192.168.87.0/24 === 172.16.1.1/32
root@strongswan:~#
```

This allows the remote employees to ping the internal network, note that unlike wireguard that uses a network device `wg0` strongswan instead uses policy based routing, on table 220:

```
root@EmployeeB3:~# ip r get 192.168.87.101
192.168.87.101 via 192.168.1.1 dev eth1 table 220 src 172.16.1.3 uid 0
    cache
root@EmployeeB3:~# ping -c1 192.168.87.101
PING 192.168.87.101 (192.168.87.101) 56(84) bytes of data.
64 bytes from 192.168.87.101: icmp_seq=1 ttl=63 time=5.96 ms

--- 192.168.87.101 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 5.955/5.955/5.955/0.000 ms
root@EmployeeB3:~#
```

Packet sniffers cannot see the content or type of traffic:

9	2.017560	104.212.7.162	213.1.133.98	UDPENC…	49	NAT-keepalive
10	2.017584	104.212.7.162	213.1.133.98	UDPENC…	49	NAT-keepalive
11	2.373945	104.212.7.162	213.1.133.98	ESP	184	ESP (SPI=0xc45673fe)
12	2.373952	104.212.7.162	213.1.133.98	ESP	184	ESP (SPI=0xc45673fe)
13	2.376039	213.1.133.98	104.212.7.162	ESP	184	ESP (SPI=0xc83961c1)
14	2.376069	213.1.133.98	104.212.7.162	ESP	184	ESP (SPI=0xc83961c1)
15	3.071587	213.1.133.98	104.212.7.162	UDPENC…	49	NAT-keepalive
16	3.071622	213.1.133.98	104.212.7.162	UDPENC…	49	NAT-keepalive

VPN Comparison:

Wireguard (wg) VPN Clients have the downside that they use a pre-shared key, at some point wg clients must exchange public keys with the server, wg itself has no key exchange ability, additionaly the roadwarrior users must have static IP addresses which can cause the configuration to become bloated with large numbers of clients.

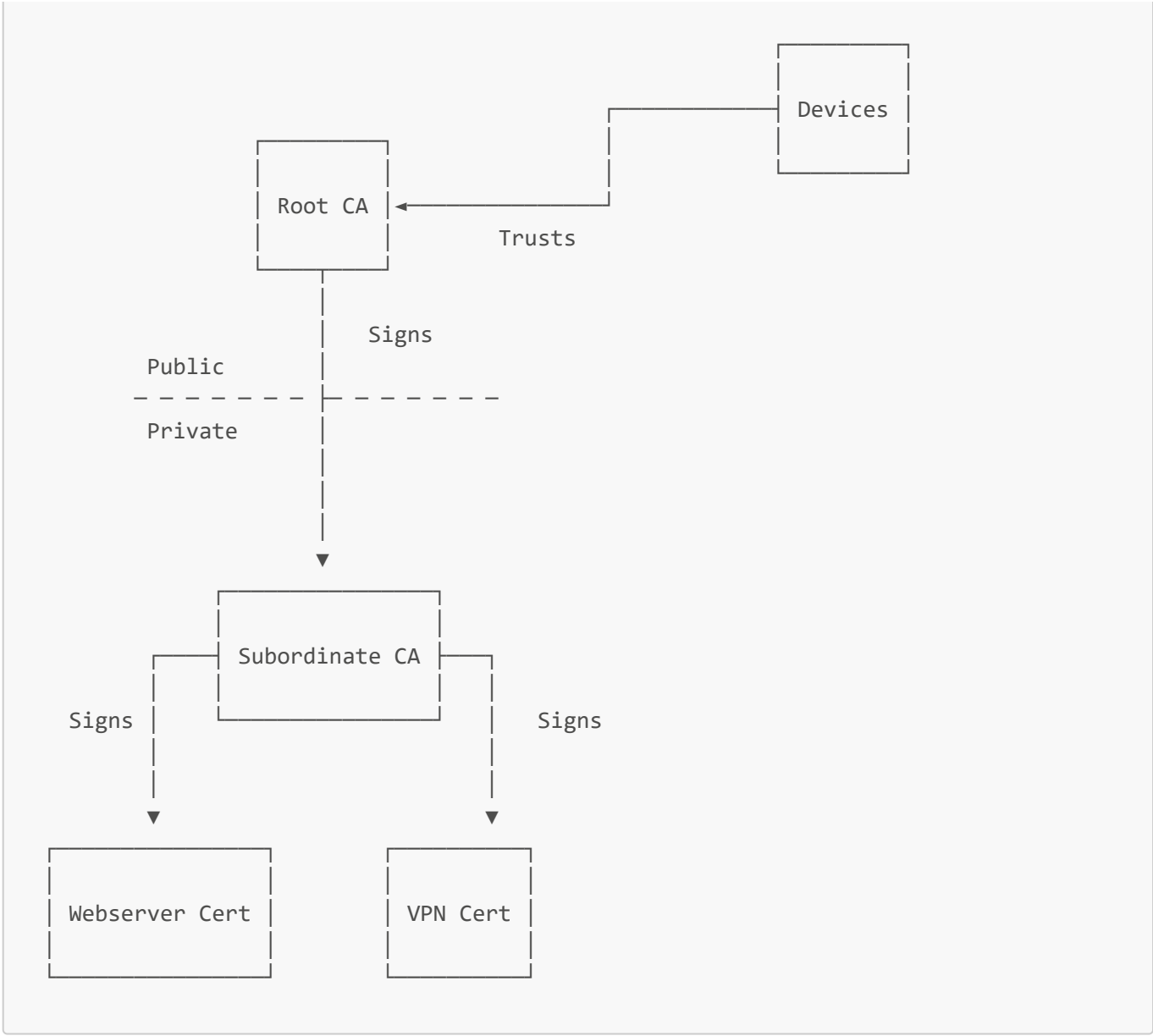
The advantage to Wireguard is that it is simpler to setup, it does not require the implementation of x509 certificates and has a relatively simple configuration, this makes it a very suitable VPN choice for small organistaions with limited infrastructure and limited roadwarrrior clients.

Strongswan (ss) on the other hand uses x509 certificates for authentication (in our implementation), this requires some infrastructure to exist - although it should be noted that a lightweight private CA is not as hard to manage and setup. The configuration for strongswan is much more extensive, and thus harder to setup, although this extended configuration does allow for more flexible use of the VPN.

Strongswans greatest advantage over wireguard is that IP addresses are dynamically assigned to remote users, this means they can all share the same config and also do not require any setup on the server side (wireguard clients must be predefined on the server side), this gives the organisation the ability to quickly and simply add new remote clients, all they need is a certificate signed by the `NobsCA`.

Building the x509 Certificate structure:

Structure:



The benefits to using a subordinate CA is that the organisation can issue and revoke new certificates quickly and easily, they do not rely on the

Creating the certificates:

> *An issue I ran into here was that I used a small key for all of this, merely to save time when they are created - unfortunately apache does not like a key of this size as it is insecure, therefore to set up https we'd have to increase the key size from the root down to atleast 2048 bits.*

Root CA: (`RootCA/cert_script.sh`)

This script creates a simple directory structure, then creates the private key and cert.

> The RootCA machine sits offline, certificate requests to be signed are taken to it via removable media, this air-gap is a good way of keeping it safe.

we generate the certificate with numerous flags:

- *-newkey rsa:1024* : Create a new key for this certificate (1024 bits is low for a root CA, this is for demonstration only)
- *-nodes* : Do not encrypt the output key (again, just for simplification)
- *-x509* : output x509 certificate (self-signed)
- *-days 3650* : Certificate is valid for 10 years (almost...)

Then make the key read only (to our user) since we dont intend on changing the file.

Then just verify that the key is valid.

The line: "`keyUsage = critical, digitalSignature, cRLSign, keyCertSign`" Allows this certficate to later sign other certificates, and also revoke certificates, this is important functionality for a RootCA

```
## Create and verify root certificates

# Prep directories
mkdir -p x509/private
mkdir -p x509/certs
chmod 700 x509/private

# Write to config
cat > "openssl.cnf" << EOF
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
C  = UK

[v3_req]
basicConstraints     = critical, CA:TRUE
subjectKeyIdentifier = hash
keyUsage             = critical, digitalSignature, cRLSign, keyCertSign
EOF

# Create cert + key
openssl req \
    -newkey rsa:1024 -nodes -keyout x509/private/RootCA.key \
    -subj "/C=UK/ST=Coventry/L=Coventry/O=Nobs Root CA/CN=(Untrustworthy) Example
Certificate Authority" \
    -extensions v3_req \
    -sha256 -x509 -days 3650 -config "openssl.cnf" -out x509/certs/RootCA_Cert.crt

# Make key read only
chmod 400 x509/private/RootCA.key

# Verify key
openssl x509 -noout -text -in x509/certs/RootCA_Cert.crt
```

Subordinate Certificate Request: (`NobsCA/create_csr.sh`)

Works in the same way as the previous example, except this time dropping the *-x509* tag, since we would like a certificate request instead of a self-signed certificate.

The config here will mean that the CSR contains the usages we have given, however these have to be specified when the CSR is signed still - I have left them here as a way of indicating to the RootCA what usages are desired.

```
## Create a certificate request for the NOBS subordinate CA

# Prep dirs
mkdir -p x509/private
mkdir x509/certs
mkdir x509/unsigned

chmod 700 x509/private

cat > "openssl.cnf" << EOF
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
C  = UK

[ v3_req ]
basicConstraints        = critical, CA:TRUE, pathlen:1
subjectKeyIdentifier    = hash
keyUsage                = critical, cRLSign, digitalSignature, keyCertSign
EOF

# Create certificate signing request + key
openssl req \
        -newkey rsa:1024 -nodes -keyout x509/private/nobsCA.key \
        -subj "/C=UK/ST=Coventry/L=Coventry/O=Nobs/CN=Nobs Subordinate CA" \
        -config "openssl.cnf" -extensions v3_req \
        -sha256 -out x509/unsigned/nobsCA.csr

# Make key read only
chmod 400 x509/private/nobsCA.key
```

Signing the Subordinate Certificate Request: (`RootCA/sign_intermediate.sh`)

> Prior to running this script the `NobsCA.csr` on the IntermediateCA machine should be transfered to the `RootCA/x509/NobsCA.csr`

Here we see some new flags:

- *-CAcreateserial* : Creates a serial number file if one doesn't already exist (this is used to keep track of which certificates have been issued and to who).
- *-days 1825* : Approx 5 years, could be anything but certificate will be invalidated when the root expires.

```
## Script for signing the nobs CAs CSR (saved under nobsCA.csr)

cat > "openssl.cnf" << EOF
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
C  = UK

[ v3_req ]
basicConstraints        = critical, CA:TRUE, pathlen:1
subjectKeyIdentifier    = hash
keyUsage                = critical, cRLSign, digitalSignature, keyCertSign
EOF

# Sign the subordinate csr signed by root
openssl x509 -req \
    -CA x509/certs/RootCA_Cert.crt -CAkey x509/private/RootCA.key \
    -CAcreateserial -days 1825 -sha256 \
    -extfile "openssl.cnf" -extensions v3_req \
    -in nobsCA.csr -out nobsCA.crt

# Verify subordinate cert
openssl x509 -noout -text -in nobsCA.crt

# Verify against root cert
openssl verify -CAfile x509/certs/RootCA_Cert.crt nobsCA.crt

# Return cert to nobsCA
```

Now the certificate (`NobsCA.crt`) can be returned to the IntermediateCA machine for use.

Signing the end certificates (`strongswan/create_ss_csr.sh`):

Most of the end certificates are almost carbon copies of one another so I will only show one here, the others are made in exactly the same process.

Firstly the endpoint (We will use strongswan in this case) has to create a CSR for the usages desired:

The difference is that:

- `basicContraints = CA:FALSE` : Means it cannot be sign more certificates
- `extendedKeyUsage = clientAuth, serverAuth`: Used for server or client side authentication
- The alt names are define the SAN, these are the names that the server may have, in this case it is just the FQDN and IP address.

Again these are merely indications to the CA about what we would like the cert to have, the CA has end choice.

```
# Prep dirs
# Strong swan has specific dirs to use
mkdir -p etc/ipsec.d/private
mkdir etc/ipsec.d/certs
chmod 700 etc/ipsec.d/private

# Write to config
cat > "openssl.cnf" << EOF
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
C  = UK
ST = Coventry
L  = Coventry
O  = Nobs
CN = Nobs Strongswan

[v3_req]
basicConstraints     = CA:FALSE
subjectKeyIdentifier = hash
keyUsage             = digitalSignature, keyEncipherment
extendedKeyUsage     = clientAuth, serverAuth
subjectAltName       = @alt_names

[alt_names]
IP.1  = 213.1.133.98
DNS.1 = gw1.nobs.cyber.test
EOF

# Make CSR + key
openssl req \
        -newkey rsa:1024 -nodes -keyout etc/ipsec.d/private/strongswan.key \
        -subj "/C=UK/ST=Coventry/L=Coventry/O=Nobs/CN=Nobs Strongswan" \
        -config "openssl.cnf" -extensions v3_req \
        -sha256 -out strongswan.csr

chmod 400 etc/ipsec.d/private/strongswan.key

# Send to subordinate CA for signing
```

Then we sign the CSR on the subordinate CA, with the attributes it requested:

```
## Script to sign strongswan csr, located at `/unsigned/strongswan.csr`

mkdir signed

# Write to config (Important part here is the alt_names)
cat > "openssl.cnf" << EOF
[v3_req]
basicConstraints     = CA:FALSE
```

```
subjectKeyIdentifier = hash
keyUsage             = digitalSignature, keyEncipherment
extendedKeyUsage     = clientAuth, serverAuth
subjectAltName       = @alt_names

# Alternative names are specified as IP.# and DNS.# for IP addresses and
# DNS accordingly.
[alt_names]
IP.1  = 213.1.133.98
DNS.1 = gw1.nobs.cyber.test
EOF

# Sign CSR using the above config for the alt_names
openssl x509 -req \
        -in "unsigned/strongswan.csr" \
        -CA "x509/certs/nobsCA.crt" \
        -CAkey "x509/private/nobsCA.key" \
        -CAcreateserial -days 365 \
        -extensions v3_req \
        -extfile "openssl.cnf" \
        -out "signed/strongswan.crt"


# Verify cert
openssl x509 -noout -text -in signed/strongswan.crt

# Send back to strongswan machine
```

This can be sent back to the strongswan machine for use.