
**UNIVERSITÀ POLITECNICA DELLE MARCHE**

FACOLTÀ DI INGEGNERIA

Corso di Laurea triennale in *Ingegneria Informatica e dell'Automazione*

# Implementazione di OpenVPN su router 4G per site-to-site vpn in ambiente CG-NAT

*Study and configuration of a site-to-site VPN in CG-NAT environment*

**Relatore:**
Prof. Ennio Gambi
**Correlatore:**
Ing. Adelmo De Santis

**Laureando:**
Simone Viozzi

Anno Accademico 2021/2022

# Prefazione

#TODO da riscrivere

Nell'ambito del mio percorso universitario ho avuto modo di approfondire le tematiche relative al mondo delle reti e del networking, a tal proposito grazie alla possibilità offerta dal Dipartimento di Ingegneria dell'Informazione, dal Prof. Ennio Gambi e dall'Ing. Adelmo De Santis ho conseguito con successo la certificazione "*HUAWEI HCIA Routing and Switching*". Successivamente, grazie alle competenze acquisite, ho collaborato con alcuni miei colleghi per progettare e realizzare una implementazione di una VPN site-to-site attraverso una connessione radiomobile per conto dell'azienda Esse-ti S.r.l.
In questo elaborato verranno esposte le principali fasi del progetto realizzato, ponendo un particolare focus sulle problematiche iniziali affrontate e all'architettura di rete nel cui ambito è stata realizzata la comunicazione tramite un canale sicuro.

# Indice

# Elenco delle figure

# Capitolo 1

# Introduzione

#TODO da scrivere da 0

## 1.1 Intro su ip/tcp

## 1.2 openvpn

## 1.3 openwrt

# Capitolo 2

# Overview dell'architettura e delle componenti utilizzate

## 2.1 Obbiettivo da ottenere

In una collaborazione tra il Dipartimento di Ingegneria dell'Informazione e l'azienda **Esse-ti S.R.L.** ci è stato esposto un progetto che consiste nel:

- fornire a dei clienti un router 4G, su cui possono essere connessi vari dispositivi, ad es. di tipo domotico.
- rendere questi dispositivi accessibili ai clienti attraverso internet

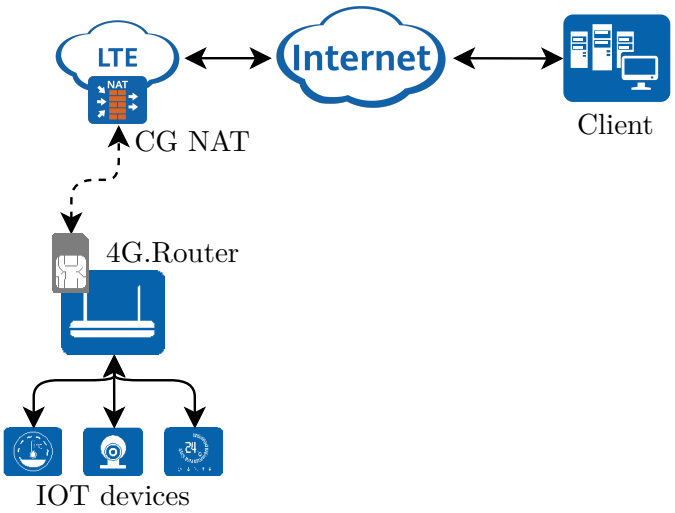



Figura 2.1: Schema concettuale dell'obbiettivo da raggiungere. [1]

Data la presenza del CG-NAT si vede subito che non è realizzabile a meno che il cliente non abbia un'IP pubblico e la sua macchina venga configurata opportunamente. Questo però non è possibile nel caso generale, quindi per risolvere efficacemente questa topologia si deve necessariamente introdurre una terza macchina provvista di IP pubblico e che funga da ponte tra il 4G.Router e il cliente.

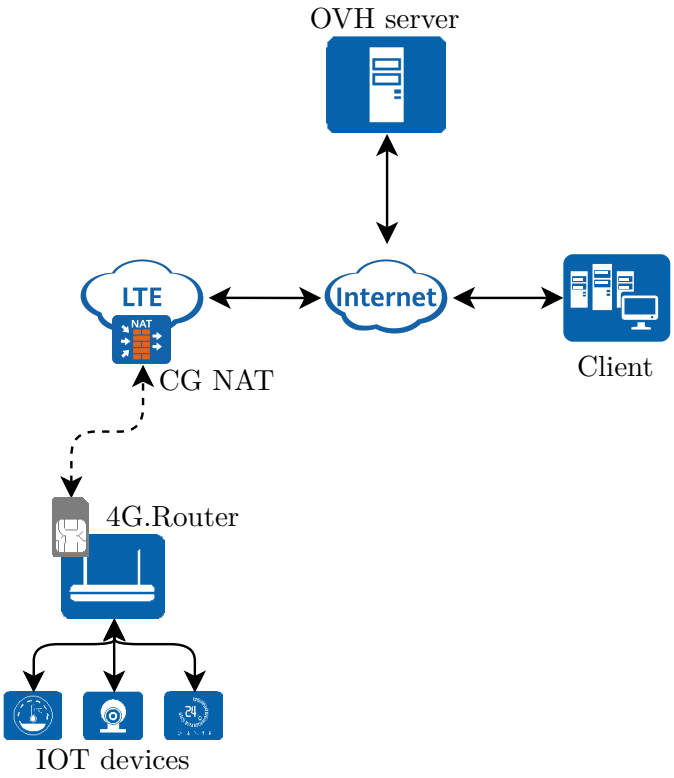



Figura 2.2: Schema concettuale dell'architettura che si dovrà implementare. [1]

In questo modo si può configurare una VPN sul server OVH e connettervi sia il 4G.Router che la macchina del cliente. In questo modo l'unica configurazione che il cliente dovrà fare è l'installazione di un cliente VPN, ciò è il minimo possibile di configurazione.

La configurazione virtuale vista dal 4G.Router e dai clienti sarà quindi:

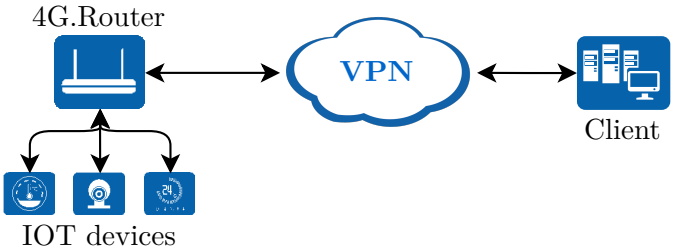



Figura 2.3: Topologia virtuale. [1]

## 2.2 Specifiche dei componenti

i componenti necessari sono:

- Esse-ti 4G.Router
- Server
- Host domotico
- Macchina del cliente

vediamo le caratteristiche minime che i componenti dovranno avere:

### 2.2.1 Esse-ti 4G.Router

Ci è stato fornito dall'azienda Esse-ti, consiste in un gateway 4G con funzionalità di router. Le specifiche complete possono essere trovate sul sito del produttore (link)


Figura 2.4: 4G.Router

Per l'implementazione di questa architettura sono necessarie solo un sub-set delle specifiche:

- Access Point wireless per offrire connettività Internet Wi-Fi a dispositivi wireless
- Client Dynamic DNS per consentire all'utente di raggiungere da remoto, tramite Internet, il router stesso e tutti i dispositivi connessi via Wi-Fi o porta LAN
- Gateway telefonico per consentire l'invio e la ricezione di chiamate attraverso la rete 4G LTE/UMTS/GSM a telefoni fissi, combinatori o altri dispositivi telefonici collegati all'ingresso FXS

Presenta inoltre come sistema operativo una versione personalizzata di OpenWrt.

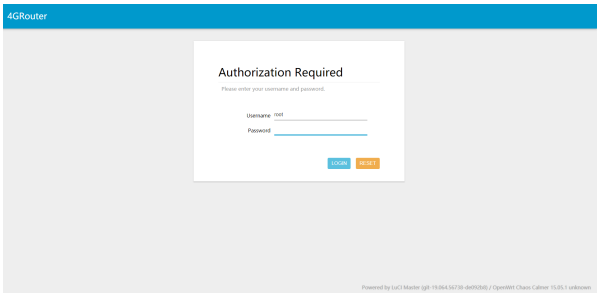
La configurazione del dispositivo puo' essere fatta sia da terminale, entrando in ssh, sia da interfaccia web:

L'interfaccia web e' una versione personalizzata di Luci.

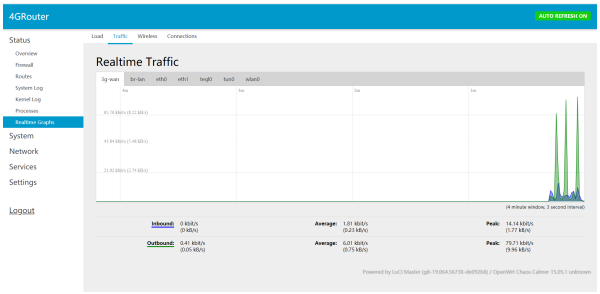
Per semplicita' si fara' riferimento all'*Esse-ti 4G.Router* chiamandolo semplicemente router.

### 2.2.2 VPS OVHCloud

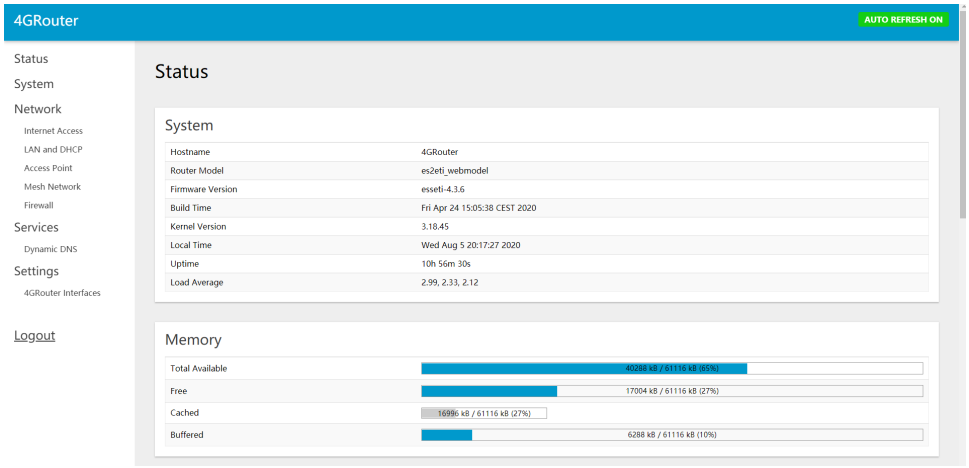
La VPS ha il solo vincolo di dover avere un'ip pubblico e una connessione a internet abbastanza veloce. Dovra' infatti sopportare un traffico simmetrico in upload / download.




(a) Schermata di autenticazione




(b) Grafico del traffico




(c) Schermata con stato riassuntivo

Per la realizzazione della topologia e' stata selezionata una macchina una VPS del provider OVHCloud, con le seguenti caratteristiche:

- 2 core virtuali
- 4Gb di memoria ram
- 80Gb di storage NVMe
- 500Mbps simmetrici di banda
- ipv4 pubblico
- Ubuntu 16.04

Per semplicita' si fara' riferimento alla *VPS OVHCloud* come server.

### 2.2.3 Host domotico

Per effettuare le varie operazioni di testing e' stato aggiunta *raspberry pi* che ha svolto il ruolo di "host domotico". Sono state fatti test con ping e iperf per testare che tutta la topologia sia stata configurata correttamente.

### 2.2.4 Macchina del cliente

Deve poter essere una qualunque macchina, non ha vincoli di sistema operativo

Necessita di avere il client openvpn installato:

- con sistema operativo Windows si deve scaricare l'eseguibile dal sito ufficiale
- su linux e' sufficiente cercare nei repository ufficiali della distribuzione che si sta usando.

# Capitolo 3

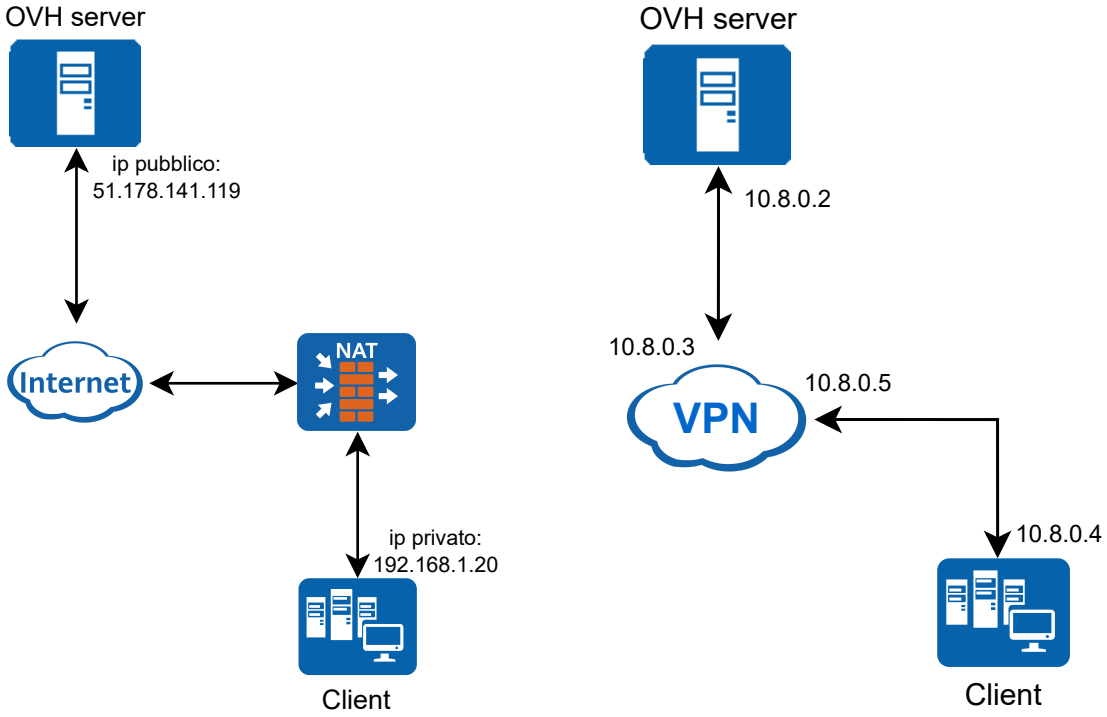
# Configurazione del server

## 3.1 Overview della configurazione e prerequisiti

In questa sezione andremo a installare e configurare OpenVPN server sulla VPS di OVHCloud.

Supponiamo di partire da una configurazione di base, che contiene solo il server openvpn ed un generico client, che supponiamo sia sotto un NAT.

Supponiamo inoltre che che l'ip pubblico del *server* sia `51.178.141.119`, si avra' quindi una configurazione come in figura 3.1a.




(a) Configurazione di partenza per questo capitolo. (b) Configurazione virtuale da raggiungere.

Figura 3.1: Configurazione di partenza e di obbiettivo per questo capitolo. [1]

Per instaurare una comunicazione bidirezionale tra il server e il client, si dovra' configurare oppornunamente una rete vpn la cui configurazione e' rappresentata in figura 3.1b.

I pacchetti necessari sono `openvpn` ed `easy-rsa`, che possono essere installati con:

*Server*

```
$ sudo apt-get update
$ sudo apt-get install -y openvpn easy-rsa
```

E' inoltre necessario avere un editor di testo, ad es. `nano` o `vim`

## 3.2 Creazione della Public key infrastructure Certificate Authority (PKI CA)

La CA puo' essere configurata sulla stessa macchina dove e' stato installato opnevpn, ma cio' e' sconsigliato per motivi di sicurezza, supponiamo quindi di usare un secondo server chiamato *server CA*

La utility `easy-rsa` mette a disposizione il comando `make-cadir`, che permette di creare una cartella pronta ad ospitare la Certificate Authority.

Andiamo quindi a crearla, nella home ad esempio:

*Server CA*

```
$ mkdir ~/openvpn-ca
$ ln -s /usr/share/easy-rsa/* ~/openvpn-ca/
$ chmod 700 /home/ubuntu/openvpn-ca/
$ cd openvpn-ca/
$ ./easyrsa init-pki

init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /home/ubuntu/openvpn-ca/pki

$ la
easyrsa  openssl-easyrsa.cnf  pki  vars.example  x509-types
```

Ora si devono personalizzare le variabili `vars`, si puo' sia partire da un file vuoto oppure modificare `vars.example` per poi rinominarlo `vars`. Andiamo quindi a creare un nuovo file vars:

*Server CA*

```
$ vim vars
set_var EASYRSA_REQ_COUNTRY  "IT"
set_var EASYRSA_REQ_PROVINCE "MC"
set_var EASYRSA_REQ_CITY     "Recanati"
set_var EASYRSA_REQ_ORG      "Esse-ti"
set_var EASYRSA_REQ_EMAIL    "s.gasparrini@esse-ti.it"
set_var EASYRSA_REQ_OU       "Esse-ti"

set_var EASYRSA_ALGO         "ec"
set_var EASYRSA_DIGEST       "sha512"
```

Le variabili nel primo blocco determinano i dati che poi verranno registrati nei certificati.

Le ultime 2 sono opzioni di sicurezza, in particolare si setta l'algoritmo di cifratura

A questo punti si deve laciare il comando `build-ca` per costruire la CA:

*Server CA*

```
$ ./easyrsa build-ca

Note: using Easy-RSA configuration from: ./vars

Using SSL: openssl OpenSSL 1.1.1f  31 Mar 2020

Enter New CA Key Passphrase:
Re-Enter New CA Key Passphrase:
read EC key
writing EC key

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:

CA creation complete and you may now import and sign cert requests.
Your new CA certificate file for publishing is at:
/home/ubuntu/openvpn-ca/pki/ca.crt

```

Eseguendo il comando verra' chiesto di inserire una passshare, che verra' usata per criptare la chiave privata appena generata. Il secondo prompt e' relativo al nome da dare alla certificazione, in questo caso e' stato lasciato il valore di default `Easy-RSA CA`.

## 3.3 Configurazione della PKI di OpenVPN

Il procedimento e' simile al precedente, ma questa volta va eseguito sul *server*.

Creiamo quindi una cartella per ospitare la PKI, es `/openvpn-pki`, e linkiamo `easy-rsa`. Inoltre limitiamo i permessi all'utente non root che stimao usando, in questo caso "ubuntu".

*Server*

```
$ mkdir ~/openvpn-pki
$ ln -s /usr/share/easy-rsa/* ~/openvpn-pki/
$ sudo chown ubuntu ~/openvpn-pki/
$ chmod 700 ~/openvpn-pki/
$ cd ~/openvpn-pki/
```

Andiamo a creare un file `vars`:

*Server*

```
$ vim vars
set_var EASYRSA_ALGO    "ec"
set_var EASYRSA_DIGEST  "sha512"
```

Concludiamo la creazione della PKI con il comando:

*Server*

```
$ ./easyrsa init-pki

Note: using Easy-RSA configuration from: ./vars

init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /home/ubuntu/openvpn-pki/pki

```

A questo punto il server opnevpn ha tutti i prerequisiti per creare una sua chiave privata e relativa *Certificate Signing Request*.

Come nome e' stato scelto "server":

*Server*

```
$ ./easyrsa gen-req server nopass

Note: using Easy-RSA configuration from: ./vars

Using SSL: openssl OpenSSL 1.1.1f  31 Mar 2020
Generating an EC private key
writing new private key to '/home/ubuntu/openvpn-pki/pki/private/server.key.438W2xM0g9'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [server]:

Keypair and certificate request completed. Your files are:
req: /home/ubuntu/openvpn-pki/pki/reqs/server.req
key: /home/ubuntu/openvpn-pki/pki/private/server.key

```

La chiave `server.key` va copiata nell'apposita cartella.

*Server*

```
$ sudo cp /home/ubuntu/openvpn-pki/pki/private/server.key /etc/openvpn/server/
```

Il secondo file creato, `server.req`, corrisponde ad una *Certificate Signing Request (CSR)* che va firmata e validata dalla CA. In questo modo ogni client che si fida della CA si fidera' di conseguenza del server OpenVPN

## 3.4 Firma del certificato opnevpn dalla CA

Dobbiamo quindi copiare il file `server.req` nel *server CA*, possiamo qualunque metodo purche' sia sicuro, ad esempio con `scp`:

*Server*

```
$ scp -3 ubuntu@openvpn_server:/home/ubuntu/openvpn-pki/pki/reqs/server.req
↪  ubuntu@ca_server:/tmp
```

Dobbiamo qundi spostarci sul server CA e importare la *certificate request* e firmarlo:

*Server CA*

```
$ cd ~/openvpn-ca
$ ./easyrsa import-req /tmp/server.req server
$ ./easyrsa sign-req server server
Using configuration from /home/ubuntu/openvpn-ca/pki/safessl-easyrsa.cnf
Check that the request matches the signature
Signature ok
The Subject\'s Distinguished Name is as follows
commonName            :ASN.1 12:'ChangeMe'
Certificate is to be certified until Mar 11 15:50:45 2025 GMT (1080 days)

Write out database with 1 new entries
Data Base Updated
```

Verra' creato un file in `/openvpn-ca/pki/issued` chiamato `server.crt` che conterra' la chiave pubblica che verra' usata dal server openvpn e inoltre la firma della CA.

Ora si devono copiare i file `ca.crt` e `server.crt` dal *server CA* al *server OpnenVPN*:

*Server*

```
$ scp -3 ubuntu@ca_server:/home/ubuntu/openvpn-ca/pki/issued/server.crt
↪  ubuntu@openvpn_server:/tmp
$ scp -3 ubuntu@ca_server:/home/ubuntu/openvpn-ca/pki/ca.crt ubuntu@openvpn_server:/tmp
```

Possiamo quindi tornare sul *server OpenVPN* e copiare i 2 file da `/tmp` a `/etc/openvpn/server`:

*Server*

```
$ sudo cp /tmp/server.crt /etc/openvpn/server
$ sudo cp /tmp/ca.crt /etc/openvpn/server
```

## 3.5 Generazione della *tls-crypt pre-shared key*

Per aumentare ulteriormente la sicurezza del nostro *server OpenVPN* possiamo creare un'ulteriore chiave, che consiste un una chiave preshared che verra' inserita in tutte le configurazioni e serve ad offuscare il certificato in fase di validazione. Quindi in caso di attacco si dovra' conoscere anche questa chiave.

La creazione va fatta sul *server OpenVPN*:

*Server*

```
$ cd ~/openvpn-pki/
$ openvpn --genkey --secret ta.key
```

il file generato `ta.key` dovra' essere copiato nella directory del server openvpn:

*Server*

```
$ sudo cp ta.key /etc/openvpn/server
```

## 3.6 Generazione delle chiavi per i clients

Creiamo una cartella nella home che ospitera' le chiavi dei client e le configurazioni openvpn:

*Server*

```
$ mkdir -p ~/client-configs/keys
$ chmod -R 700 ~/client-configs
```

Creiamo quindi un certificato per un *client*:

*Server*

```
$ cd ~/openvpn-pki/
$ ./easyrsa gen-req client1 nopass
```

Ora dobbiamo copiare `client1.key` nella directory appena creata, e `client1.req` va copiato nel server CA per essere firmato:

*Server*

```
$ cp pki/private/client1.key ~/client-configs/keys/
$ scp -3 ubuntu@openvpn_server:/home/ubuntu/openvpn-pki/pki/reqs/client1.req
↪  ubuntu@ca_server:/tmp
```

Dobbiamo quindi spostarci sul server CA e importare la *certificate request* e firmarla:

*Server CA*

```
$ cd ~/openvpn-ca
$ ./easyrsa import-req /tmp/client1.req client1
$ ./easyrsa sign-req client client1
Using configuration from /home/ubuntu/openvpn-ca/pki/safessl-easyrsa.cnf
Check that the request matches the signature
Signature ok
The Subject\'s Distinguished Name is as follows
commonName            :ASN.1 12:'ChangeMe'
Certificate is to be certified until Mar 16 13:15:09 2025 GMT (1080 days)

Write out database with 1 new entries
Data Base Updated
```

Per poi ricopiare dal server CA al server openvpn il certificato firmato:

*Server*

```
$ scp -3 ubuntu@ca_server:/home/ubuntu/openvpn-ca/pki/issued/client1.crt
↪  ubuntu@openvpn_server:/tmp
```

Quindi ci dobbiamo spostare sul server opnevpn e copiare le chiavi nella cartella `client-configs/keys`, in modo da prepararla per la crazione delle configurazioni opnevpn. E' necessario inoltre cambiare i permessi dei file rendendoli accessibili all'utente ubuntu:

*Server*

```
$ cp /tmp/client1.crt ~/client-configs/keys/
$ cp ~/openvpn-pki/ta.key ~/client-configs/keys/
$ sudo cp /etc/openvpn/server/ca.crt ~/client-configs/keys/
$ sudo chown ubuntu:ubuntu ~/client-configs/keys/*
```

## 3.7 Creazione del file di configurazione del server OpenVPN

Il server openvpn viene configurato attraverso `/etc/openvpn/server/server.conf`, per non partire da una configurazione vuota si puo' copiare la configurazione di esempio:

*Server*

```
$ cd /etc/openvpn/server/
$ sudo wget "https://raw.githubusercontent.com/OpenVPN/openvpn/\
                master/sample/sample-config-files/server.conf"
```

Dobbiamo quindi modificare il file e cambiare alcune configurazioni, per facilitare la lettura sara' incluso il numero riga modificato:

*Server*

```
$ sudo vim server.conf
85  dh none             # non sonos stati usati i parametri Diffie-Hellman
244 ;tls-auth ta.key 0 # This file is secret
245 tls-crypt ta.key    # selezione della preshared key
253 cipher AES-256-GCM  # selezione della cifratura scelta
275 user nobody         # utente che eseguira' il server openvpn, in modo da restringere
↪  i permessi
276 group nogroup       # stassa cosa per il gruppo
318 auth sha256         # selezione del metodo di autenticazione
```

## 3.8 Configurazioni sulla network stack del server openvpn

Per abilitare l'*ip forwarding* si dovra' modificare il file `/etc/sysctl.conf`, il comando successivo serve a ricaricare le configurazioni dai file:

*Server*

```
$ sudo vim /etc/sysctl.conf
69 net.ipv4.ip_forward = 1
$ sudo sysctl -p
net.ipv4.ip_forward = 1
```

## 3.9 Configurazione del firewall

Sulla VPS scelta e' presente il firewall *firewalld*, ma per una piu' semplice configurazione e' consigliato di disattivarlo e installare *ufw*:

*Server*

```
$ sudo systemctl mask firewalld
$ sudo systemctl stop firewalld
$ sudo apt-get install ufw
$ sudo ufw allow ssh
Rule added
Rule added (v6)
$ sudo ufw enable
```

E' importantissimo ricordarsi di consentire l'ssh prima di abilitare il firewall, altrimenti si perdera' l'accesso alla VPS.

### 3.9.1 Configurazione del NAT

Per far si che i pacchetti provenienti dalla *VPN* entrino nella network stack del *server* si deve aggiungere una regola di *NAT* nel firewall. Per farlo si deve conoscere quale e' l'interfaccia di rete del *server*, cioe' quella che ha come ip il suo ip pubblico:

*Server*

```
$ ip addr
[...]
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen
↪  1000
    link/ether a6:23:5f:48:ba:de brd ff:ff:ff:ff:ff:ff
    inet 51.178.141.119/20 brd 51.178.141.255 scope global dynamic ens3
       valid_lft 1857sec preferred_lft 1857sec
    inet6 fe80::23:bfff:ac24:aace/64 scope link
       valid_lft forever preferred_lft forever
[...]
```

In questo caso il nome dell'interfaccia di rete e' *ens3*, possiamo quindi procedere con la configurazione del firewall, si andra' a modificare il file `/etc/ufw/before.rules` e aggiungere la regola di NAT:

*Server*

```
$ sudo vim /etc/ufw/before.rules
# ## rules.before
# ## Rules that should be run before the ufw command line added rules. Custom
# rules should be added to one of these chains:
# ufw-before-input
# ufw-before-output
# ufw-before-forward
#

# START OPENVPN RULES
# NAT table rules
*nat
:POSTROUTING ACCEPT [0:0]
# Allow traffic from OpenVPN client to ens3
-A POSTROUTING -s 10.8.0.0/24 -o ens3 -j MASQUERADE
COMMIT
# END OPENVPN RULES


# Don't delete these required lines, otherwise there will be errors
*filter
. . .
```

Nella modifica del file si deve stare attenti ad inserire la nuova regola in cima al file e sotto i commenti iniziali, e' inoltre importante inserire i commenti nella regola.

### 3.9.2 Configurazione del packet forwarding

Next, you need to tell UFW to allow forwarded packets by default as well. To do this, open the `/etc/default/ufw` file:

*Server*

```
sudo nano /etc/default/ufw
```

Inside, find the `DEFAULT_FORWARD_POLICY` directive and change the value from DROP to ACCEPT: `/etc/default/ufw`

*Server*

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

Save and close the file when you are finished.

### 3.9.3 Conclusione della configurazione del firewall

Per concludere la configurazione si deve abilitare la porta relativa alla vpn, in questo caso `1194`, e riavviare il firewall:

*Server*

```
$ sudo ufw allow 1194/udp
$ sudo ufw reload
$ sudo ufw status
Status: active
To                         Action      From
--                         ------      ----
22                         ALLOW       Anywhere
1194/udp                   ALLOW       Anywhere
22 (v6)                    ALLOW       Anywhere (v6)
1194/udp (v6)              ALLOW       Anywhere (v6)
```

## 3.10 Avvio del server OpenVPN

Ora che la configurazione del server e' in una situazione stabile possiamo avviarlo:

*Server*

```
$ sudo systemctl enable openvpn-server@server.service
$ sudo systemctl start openvpn-server@server.service
$ sudo systemctl status openvpn-server@server.service
● openvpn-server@server.service - OpenVPN service for server
     Loaded: loaded (/usr/lib/systemd/system/openvpn-server@.service; enabled; vendor
     ↪  preset: disabled)
     Active: active (running) since Mon 2022-04-18 13:08:44 CEST; 4h 22min ago
       Docs: man:openvpn(8)
             https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
             https://community.openvpn.net/openvpn/wiki/HOWTO
   Main PID: 436 (openvpn)
     Status: "Initialization Sequence Completed"
      Tasks: 1 (limit: 9488)
     Memory: 4.8M
        CPU: 199ms
     CGroup: /system.slice/system-openvpn\x2dserver.slice/openvpn-server@server.service
             └─436 /usr/bin/openvpn --status /run/openvpn-server/status-server.log
             ↪  --status-version 2 --suppress-timestamps --config server.conf

Apr 18 13:08:44 server openvpn[436]: /sbin/ip addr add dev tun0 local 10.8.0.1 peer
↪  10.8.0.2
Apr 18 13:08:44 server openvpn[436]: /sbin/ip route add 10.8.0.0/24 via 10.8.0.2
Apr 18 13:08:44 server openvpn[436]: UDPv4 link local (bound): [AF_INET][undef]:1194
Apr 18 13:08:44 server openvpn[436]: UDPv4 link remote: [AF_UNSPEC]
Apr 18 13:08:44 server openvpn[436]: MULTI: multi_init called, r=256 v=256
Apr 18 13:08:44 server openvpn[436]: IFCONFIG POOL: base=10.8.0.4 size=62, ipv6=0
Apr 18 13:08:44 server openvpn[436]: IFCONFIG POOL LIST
Apr 18 13:08:44 server openvpn[436]: Initialization Sequence Completed
```

Il comando `systemctl enable` abilita il servizio per essere avviato all'avvio della macchina, mentre `systemctl start` lo avvia immediatamente. Con il comando `systemctl status` si puo' verificare lo stato del servizio, si vede che il servizio e' *active (running)*.

## 3.11 Script per la creazione delle configurazioni dei client

Per facilitare la creazione dei file di configurazione dei client, `clientX.conf`, andremo a crare un'apposito script bash. Per prima cosa si deve scaricare e personalizzare la configurazione base del client:

*Server*

```
$ cd ~/client-configs/
$ wget "https://raw.githubusercontent.com/OpenVPN/openvpn\
            /master/sample/sample-config-files/client.conf" \
                -O base.conf
$ vim base.conf
42   remote 51.178.141.119 1194     # va messo l'ip e la porta del server OpenVPN
88   ;ca ca.crt                     # non useremo i file esterni ma ingloberemo
89   ;cert client.crt               # questi file in un file direttamente nella
90   ;key client.key                # configurazione del client
108  ;tls-auth ta.key 1             # stessa cosa per la preshared key
116  cipher AES-256-GCM             # cifratura usata
117  auth SHA256                    # autenticazione usata
118  key-direction 1
                                 # le seguenti righe sono delle direttive al dns dell'host
120  ; script-security 2
121  ; up /etc/openvpn/update-resolv-conf
122  ; down /etc/openvpn/update-resolv-conf

125  ; script-security 2
126  ; up /etc/openvpn/update-systemd-resolved
127  ; down /etc/openvpn/update-systemd-resolved
128  ; down-pre
129  ; dhcp-option DOMAIN-ROUTE .
```

Ora creiamo lo script bash `make_config.sh`:

*Server*

```
$ vim make_config.sh
#!/bin/bash

# First argument: Client identifier

KEY_DIR=~/client-configs/keys
OUTPUT_DIR=~/client-configs/files
BASE_CONFIG=~/client-configs/base.conf

cat ${BASE_CONFIG} \
    <(echo -e '<ca>') \
    ${KEY_DIR}/ca.crt \
    <(echo -e '</ca>\n<cert>') \
    ${KEY_DIR}/${1}.crt \
    <(echo -e '</cert>\n<key>') \
    ${KEY_DIR}/${1}.key \
    <(echo -e '</key>\n<tls-crypt>') \
    ${KEY_DIR}/ta.key \
    <(echo -e '</tls-crypt>') \
    > ${OUTPUT_DIR}/${1}.ovpn
$ chmod 700 make_config.sh
```

Lo scopo di questo script e' di aggiungere al file `base.conf` il certificato della CA, `ca.crt`,

il certificato e chiave relativi al client per cui si sta creando la configurazione, passato come argomento allo script, e la *preshared key*. Il tutto viene scritto in un file che ha lo stesso nome del *client* per cui si sta creando la configurazione ma `.conf`.

Quindi per creare la configurazione di *client 1*:

*Server*

```
$ ./make_config.sh client1
```

Nella cartella `client-configs/files/` si trovera' il file di configurazione per il client `client1.ovpn`.

## 3.12 Test della configurazione

Ora che abbiamo un file di configurazione per il client, possiamo testare che la configurazione fino a questo punto sia corretta. Per farlo ci spostamo su una macchina client, con SO linux ad esempio, e si avvia il *client* con la configurazione creata al passo precedente:

*Client*

```
$ sudo openvpn --config client1.ovpn
Thu Apr 21 12:53:04 2022 OPTIONS IMPORT: adjusting link_mtu to 1624
Thu Apr 21 12:53:04 2022 OPTIONS IMPORT: data channel crypto options modified
Thu Apr 21 12:53:04 2022 Outgoing Data Channel: Cipher 'AES-256-GCM' initialized with 256
↪ bit key
Thu Apr 21 12:53:04 2022 Incoming Data Channel: Cipher 'AES-256-GCM' initialized with 256
↪ bit key
Thu Apr 21 12:53:04 2022 ROUTE_GATEWAY 10.0.4.1/255.255.255.0 IFACE=eth0
↪ HWADDR=02:42:0a:00:04:03
Thu Apr 21 12:53:04 2022 TUN/TAP device tun0 opened
Thu Apr 21 12:53:04 2022 TUN/TAP TX queue length set to 100
Thu Apr 21 12:53:04 2022 /sbin/ip link set dev tun0 up mtu 1500
Thu Apr 21 12:53:04 2022 /sbin/ip addr add dev tun0 local 10.8.0.6 peer 10.8.0.5
Thu Apr 21 12:53:04 2022 /sbin/ip route add 10.8.0.1/32 via 10.8.0.5
Thu Apr 21 12:53:04 2022 WARNING: this configuration may cache passwords in memory -- use
↪ the auth-nocache option to prevent this
Thu Apr 21 12:53:04 2022 Initialization Sequence Completed
```

Se la configurazione fino a questo punto e' corretta si avra il messaggio `Initialization Sequence Completed`.

Nel *client* si avra' una nuova interfaccia di rete chiamata `tun0`, questa e' l'interfaccia virtuale creata dalla vpn.

*Client*

```
2: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN
↪  group default qlen 100
    link/none
    inet 10.8.0.6 peer 10.8.0.5/32 scope global tun0
       valid_lft forever preferred_lft forever
```

Si puo' vedere come l'ip assegnato al *client* dalla vpn e' `10.8.0.6`.

Per testare che la connessione sia instaurata correttamente si puo' usare la utility `ping`, ad esempio possiamo fare il ping dal *client* verso l'ip interno alla vpn del *server*:

*Client*

```
$ ping 10.8.0.1
PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
64 bytes from 10.8.0.1: icmp_seq=1 ttl=64 time=0.250 ms
64 bytes from 10.8.0.1: icmp_seq=2 ttl=64 time=0.220 ms
```

Se nel frattempo si esegue la utility `tpcdump` sul server si potranno vedere i pacchetti *echo request* e *echo reply*:

*Server*

```
$ sudo tcpdump
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
13:05:18.423151 IP 10.8.0.6 > 10.8.0.1: ICMP echo request, id 2, seq 1, length 64
13:05:18.423171 IP 10.8.0.1 > 10.8.0.6: ICMP echo reply, id 2, seq 1, length 64
13:05:19.431496 IP 10.8.0.6 > 10.8.0.1: ICMP echo request, id 2, seq 2, length 64
13:05:19.431518 IP 10.8.0.1 > 10.8.0.6: ICMP echo reply, id 2, seq 2, length 64
```

Si vede quindi che e' possibile una comunicazione bidirezionale tra *client*, `10.8.0.6`, e *server*, `10.8.0.1`.

# Bibliografia

[1] Huawei. Product image gallery. https://info.support.huawei.com/network/imagelib/getImagePartList?product_family=Router&product_type=Access%20Router%7CIOT%20Gateway&domain=&lang=en.