



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

*Implementation of OpenVPN on a 4G Router for site-to-site VPN in
CG-NAT environment*

Relatore:

Prof. Ennio Gambi

Correlatore:

Ing. Adelmo De Santis

Laureando:

Simone Viozzi

Prefazione

Indice

Prefazione	ii
Indice	iii
Elenco delle figure	v
1 Introduzione	1
1.1 TCP/IP e modello a strati	1
1.1.1 Internet Protocol Suite	2
1.1.2 Incapsulamento	3
1.1.3 IP Network Address Translator (NAT)	4
1.2 Openvpn	4
1.2.1 Crittografia e autenticazione	5
1.2.2 Networking	5
1.3 OpenWrt OK	6
1.3.1 LuCI web interface	6
2 Overview dell'architettura e delle componenti utilizzate	8
2.1 Obiettivo da ottenere	8
2.2 Specifiche dei componenti	10
2.2.1 Esse-ti 4G.Router	10
2.2.2 VPS OVHCloud	11
2.2.3 Host domotico	12
2.2.4 Macchina del cliente	12
3 Configurazione del server	13
3.1 Overview della configurazione e prerequisiti	13
3.2 Creazione della Public key infrastructure	14

3.2.1	Creazione della struttura di cartelle necessaria per ospitare la PKI	14
3.2.2	Creazione della CA	15
3.2.3	Configurazione della PKI di OpenVPN	16
3.2.4	Firma del certificato OpenVPN dalla CA	18
3.3	Generazione della <i>tls-crypt pre-shared key</i>	19
3.4	Generazione delle chiavi per i clients	19
3.5	Creazione del file di configurazione del server OpenVPN	21
3.6	Configurazioni sulla network stack del server openvpn	21
3.7	Configurazione del firewall	22
3.7.1	Configurazione del NAT	22
3.7.2	Configurazione del packet forwarding	23
3.7.3	Conclusione della configurazione del firewall	23
3.8	Avvio del server OpenVPN	24
3.9	Script per la creazione delle configurazioni dei client	25
3.10	Test della configurazione	26
4	Configurazione Router	29
4.1	Overview della configurazione	29
4.2	Creazione della configurazione Openvpn	29
4.3	Abilitazione del Client-to-Client nel server OpenVPN	31
4.4	Assegnazione ip statico al Router	32
5	Connessione degli host domotici alla VPN	34
5.1	Overview della configurazione	34
5.2	Creazione della firewall zone per la VPN	35
5.3	Aggiunta dell'interfaccia tun0 alla zona vpn	37
5.4	Modifiche alla configurazione OpenVPN del server	38
5.5	Test della configurazione	39
6	Conclusione	42
	Bibliografia	43

Elenco delle figure

1.1	Rappresentazione degli strati del modello TCP/Ip, con relativo incapsulamento e dispositivo di dominio	2
1.2	Incapsulamento	4
1.3	Incapsulamento	6
1.4	Interfaccia web LuCI	7
2.1	Schema concettuale dell'obiettivo da raggiungere.	8
2.2	Schema concettuale dell'architettura che si dovrà implementare.	9
2.3	Topologia virtuale vista dal cliente.	9
2.4	Esse-ti 4G.Router	10
2.5	Interfaccia web Esse-ti 4G.Router	11
3.1	Configurazione di partenza e di obiettivo per il capitolo 3.	13
3.2	Diagramma per schematizzare la procedura di creazione della CA.	14
3.3	Diagramma per schematizzare la procedura di firma di un certificato client. . . .	19
4.1	Schemi delle configurazioni iniziali e finali per il capitolo 4	29
4.2	Configurazione della VPN tramite LuCI.	31
5.1	Schemi concettuali della configurazione iniziale e finale per il capitolo 5	34
5.2	Configurazione iniziale del firewall	35
5.3	Configurazione del firewall	36
5.4	Configurazione finale del firewall	37
5.5	Assegnazione interfaccia <i>tun0</i> alla zona firewall <i>vpn</i> tramite interfaccia LuCI . .	38

Capitolo 1

Introduzione

1.1 TCP/IP e modello a strati

Internet costituisce la più grande rete di comunicazione al mondo, con più di 5 miliardi di dispositivi connessi nello stesso momento [6]. Eppure la sua architettura di base è relativamente semplice, tutto è incentrato su un set di protocolli e i dispositivi che li implementano.

Il suo obiettivo è quello di collegare utenti attraverso una rete geografica, indipendentemente dalla distanza o da quali mezzi fisici vengano usati per trasferire i dati. Deve quindi essere in grado di gestire una grande varietà di applicazioni, tipi di rete e mezzi di trasmissione. Per garantire la flessibilità necessaria e non aggiungere troppa complessità, l'architettura di Internet è stata suddivisa in alcuni strati, ad esempio il modello TCP/IP è costituito da 4 strati.

1.1.1 Internet Protocol Suite

Per comunicare su Internet, gli Host devono implementare un set di protocolli che costituiscono l'*Internet protocol suite* [1]. I protocolli si suddividono in strati logici che li raggruppano in 4 categorie, ogni Host deve implementare almeno un protocollo per ogni strato.

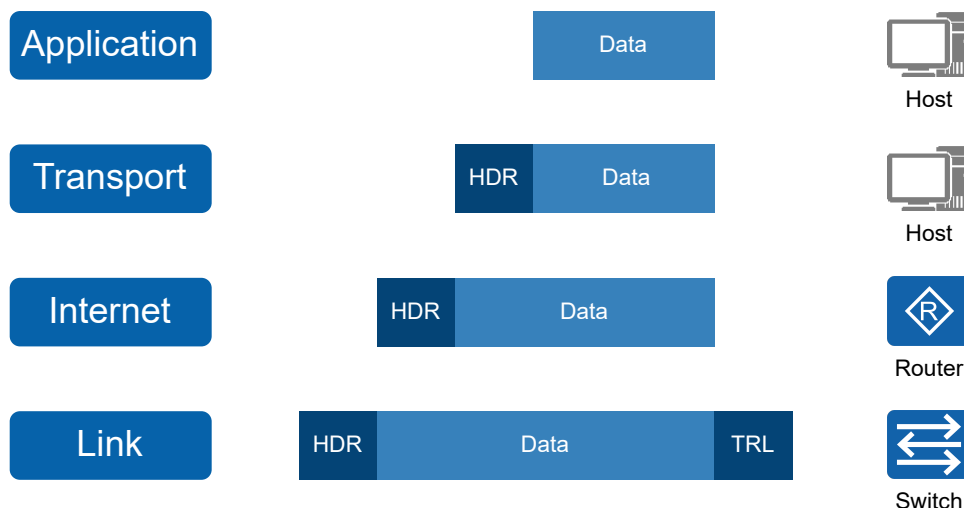


Figura 1.1: Rappresentazione degli strati del modello TCP/Ip, con relativo incapsulamento e dispositivo di dominio

Vediamo una breve descrizione dei layer e delle loro funzioni:

layer 4: **Application:** Il livello applicazione è il layer più alto dell'*Internet protocol suite*, i protocolli di questo livello si suddividono in protocolli utente e di supporto.

I protocolli utente espongono un servizio direttamente all'utente finale, alcuni esempi sono: [http](#), [ftp](#), [ssh](#), etc. Mentre i protocolli di supporto non sono direttamente usati dagli utenti ma sono comunque necessari per il funzionamento della rete, alcuni esempi sono: [DNS](#), [SNMP](#) etc.

layer 3: **Transport:** Il livello di trasporto fornisce una comunicazione end-to-end tra le applicazioni, infatti in generale il campo data del livello di trasporto non viene letto da nessuno se non l'applicazione di sorgente e destinazione. I protocolli principali di questo livello sono TCP e UDP: il [TCP](#) è connection-oriented e fornisce alta affidabilità; mentre l'[UDP](#) è connection-less, quindi ogni inaffidabilità della rete deve essere gestita a livello applicazione.

layer 2: **Internet:** Tutti i protocolli di trasporto usano il protocollo Internet (IP) per portare i dati dall'host sorgente alla destinazione. Al contrario dei protocolli di livello trasporto il protocollo IP non è end-to-end, quindi è intrinsecamente di tipo connection-less. Non fornisce quindi nessuna garanzia che il pacchetto arrivi a destinazione, o arrivi danneggiato o duplicato. I layer sopra al livello IP sono responsabili di mantenere l'affidabilità dei servizi quando essa è richiesta. Di questo layer fanno parte i protocolli [IP](#), [ICMP](#), etc.

layer 1: **Link**: È il layer più vicino al mezzo fisico su cui viaggiano i dati, ogni host deve implementare il protocollo usato per la specifica interfaccia che usa. Ad esempio un'host con un'interfaccia Ethernet deve implementare i protocolli [Ethernet II](#) e [IEEE 802.3](#).

1.1.2 Incapsulamento

Ogni protocollo di ogni layer aggiunge un header e un trailer, incapsulando il prodotto del layer precedente nel suo campo data, possiamo vedere una rappresentazione grafica in fig. 1.1.

Possiamo vedere l'incapsulamento in azione catturando un pacchetto con il programma *Wireshark*, ad esempio in questo caso si tratta di un pacchetto proveniente da una pagina web https:

```
Wireshark code: 1.1.1
1 1 > Frame 43408: 93 bytes on wire (744 bits), 93 bytes captured (744 bits) on interface
   ↳ wlp5s0, id 0
2 1 > Ethernet II, Src: IntelCor_eb:91:5f (cc:d9:ac:eb:91:5f), Dst: HuaweiDe_27:a9:24
   ↳ (0c:e4:a0:27:a9:24)
3 2 > Internet Protocol Version 4, Src: 192.168.8.119, Dst: 142.250.180.174
4 3 > Transmission Control Protocol, Src Port: 36354, Dst Port: 443, Seq: 36720, Ack:
   ↳ 13639, Len: 39
5 4 > Transport Layer Security
6 > TLSv1.3 Record Layer: Application Data Protocol: http-over-tls
7 Opaque Type: Application Data (23)
8 Version: TLS 1.2 (0x0303)
9 Length: 34
10 Encrypted Application Data:
   ↳ 389bf9516cce567d0d90ef62ba2a87376091fedb7f66f3b9e60e45a39376b1ae667a
11 [Application Data Protocol: http-over-tls]
```

TODO: da perfezionare

Quindi andando ad analizzare lo stack di protocolli si vede:

- **Transport Layer Security**[9]: È il dato effettivo che è stato trasmesso in rete. In questo caso consiste in un pacchetto http cifrato con il protocollo TLS.
- **Transmission Control Protocol**[8]: incapsula il layer applicazione in un layer di trasporto usando il protocollo TCP. Le informazioni principali contenute nell'header sono la porta sorgente e destinazione, ciò permette di individuare a quale applicazione è destinato il pacchetto.
- **Internet Protocol Version 4**[7]: incapsula il layer di trasporto in un pacchetto IP. Nell'header IP sono contenuti l'indirizzo IP di sorgente e destinazione, ciò permette di individuare a quale host è destinato il pacchetto.
- **Ethernet II**[10]: in questo caso il mezzo fisico è una porta Ethernet quindi il pacchetto IP viene incapsulato con il protocollo Ethernet II. Nell'header Ethernet II sono contenuti

l'indirizzo MAC di sorgente e destinazione, ciò permette di individuare a quale host è l'interfaccia fisica di destinazione.

Possiamo vedere graficamente il pacchetto descritto in [code:1.1.1](#) e le informazioni principali di ogni header:

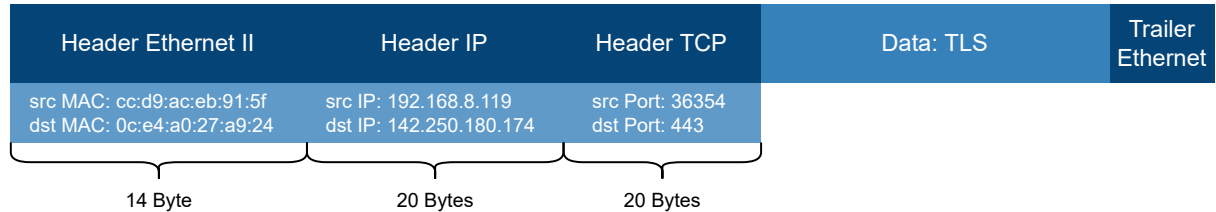


Figura 1.2: Incapsulamento

1.1.3 IP Network Address Translator (NAT)

Per mitigare il problema della saturazione degli indirizzi IPv4 [\[11\]](#), è stato introdotto un dispositivo di rete che consente di riusare spazi di indirizzi privati e ridurre il numero di Ipv4 pubblici necessari.

Il NAT [\[2\]](#) permette di effettuare una mappatura da uno spazio di indirizzi a un altro, modificando così le informazioni di routing nel pacchetto IP. La mappatura può essere sia uno-a-uno, nel caso di *Static NAT* e *Dynamic NAT*; sia molti-a-molti, nel caso di *Network Address and Port Translation* (NAPT), in cui oltre all'IP viene modificata anche la porta del protocollo di trasporto. Ciò consente di mappare un grande numero di indirizzi privati in relativamente pochi indirizzi pubblici.

Una delle applicazioni più comuni è inserirlo nel gateway di una rete privata domestica, ciò permette di mascherare i dispositivi nella rete privata e far sì che la rete esterna veda tutto il traffico attraverso un solo dispositivo, il router.

Carrier Grade NAT (CG-NAT)

I CG-NAT sono una tipologia di NAT che lavorano su larga scala e in generale sono implementati all'interno della rete dell'ISP. Sono usati per mappare un grande numero di utenti in relativi pochi indirizzi pubblici. Dato che sono gestiti dall'ISP, gli utenti non hanno nessun controllo sulla sua configurazione.

1.2 Openvpn

OpenVPN è un applicativo open source che ha l'obiettivo di fornire una VPN che sia semplice da configurare e che funzioni in ogni contesto. Openvpn può incapsulare sia pacchetti IP che frame Ethernet, in un tunnel sicuro che può viaggiare sia su TCP che UDP. Ha molte opzioni di configurazione, come la possibilità di usare qualsiasi porta, oppure l'uso della compressione. Il

tutto è raccolto in un singolo applicativo che può funzionare sia da client che da server, in base alla configurazione fornita.

Possiamo ad esempio vedere una cattura di Wireshark di un pacchetto OpenVPN su UDP e porta 1194:

```
Wireshark code: 1.2.1
1 > Frame 90: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface
  ↳ br-08876ccdf1f5, id 0
2 > Ethernet II, Src: cc:d9:ac:eb:91:5f (cc:d9:ac:eb:91:5f), Dst: 0c:e4:a0:27:a9:24
  ↳ (0c:e4:a0:27:a9:24)
3 > Internet Protocol Version 4, Src: 192.168.1.20, Dst: 51.178.141.119
4 > User Datagram Protocol, Src Port: 47007, Dst Port: 1194
5 > OpenVPN Protocol
6 Type: 0x48 [opcode/key_id]
7 Peer ID: 0
8 Data (36 bytes)
9 Data: 00000019a366196eb2aca181df226faf8514ab73f524f7ef335d55fc57322d032a2095e4
```

1.2.1 Crittografia e autenticazione

Per la cifratura e autenticazione viene usata la libreria [OpenSSL](#), open source e ampiamente usata dalla maggior parte dei servizi su internet, come ad esempio l'https. Ciò fornisce ad OpenVPN la flessibilità di poter usare tutti i cifrari forniti da questa libreria.

L'autenticazione può essere eseguita usando una pre-shared key, un sistema basato sull'utilizzo dei certificati, una semplice password o una combinazione dei precedenti. Il metodo più sicuro è quello basato sui certificati, che sfrutta una *Public key infrastructure* [12] per autenticare che i certificati forniti dai client siano effettivamente autentici. Con questo metodo si crea un certificato per ogni utente che, se opportunamente firmato, permette a quello specifico utente di autenticarsi al server VPN. Questo metodo ha inoltre il vantaggio che un certificato può essere revocato in ogni momento, facendo così perdere l'accesso all'utente che lo stava usando.

Maggiori informazioni possono essere trovate sulla [wiki di OpenVPN](#).

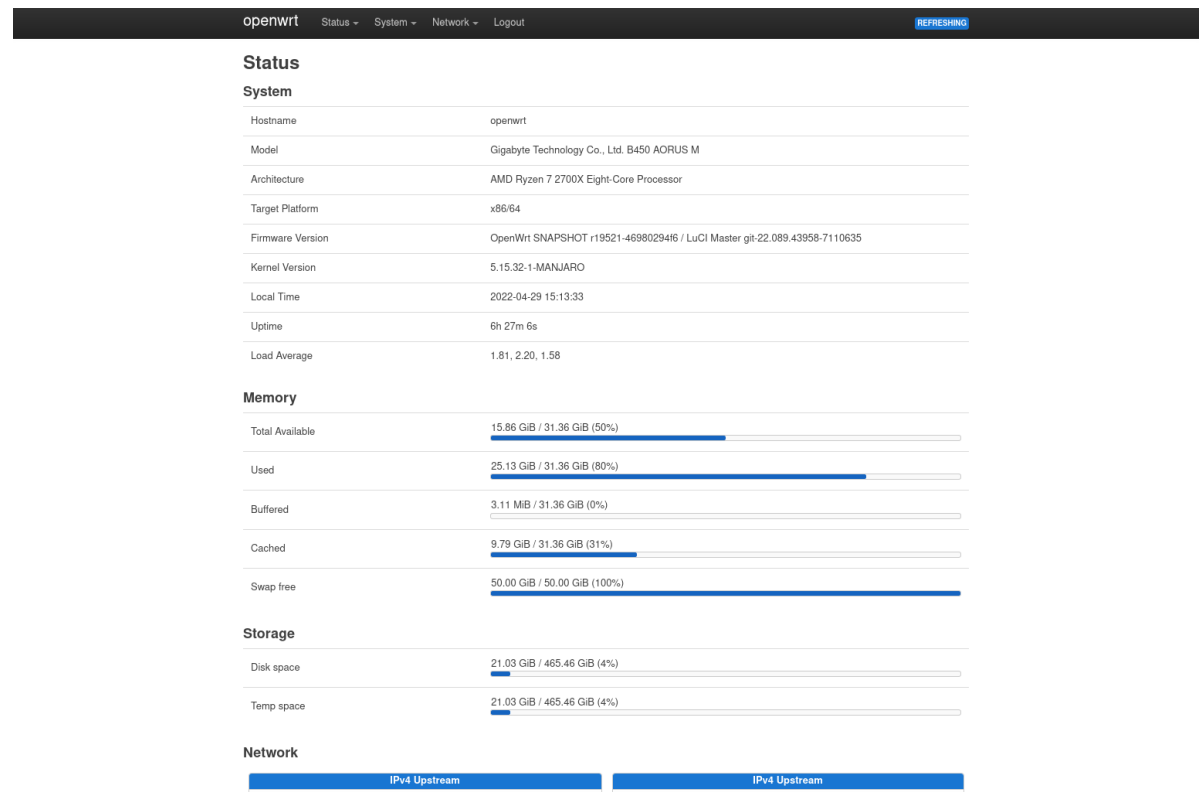
1.2.2 Networking

OpenVPN viene incapsulato dai più comuni protocolli di trasporto (TCP e UDP), ciò lo rende adatto in caso l'ips blocchi VPN di livello più basso, es. [ipsec](#). Può inoltre funzionare attraverso la maggior parte dei server proxy, firewalls e NAT.

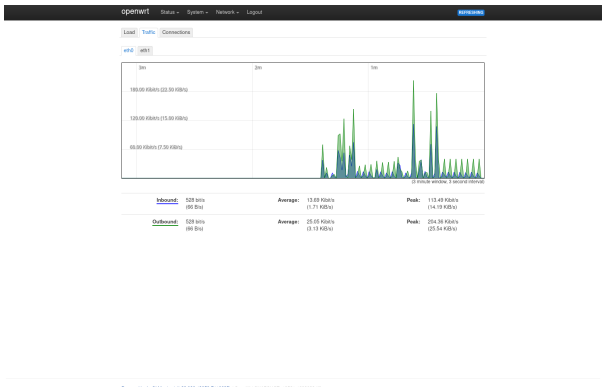
La configurazione del server permette di impostare opzioni che modificano la configurazione di rete del server o dei client, ciò permette ad esempio di aggiungere una rotta alla tabella di routing dei client nel momento in cui si connettono alla VPN.

Per depositare il traffico nella network stack dei client, OpenVPN usa i [driver universali TUN/TAP](#). Può quindi creare un tunnel IP di livello 3 (TUN), o Ethernet livello 2 (TAP).

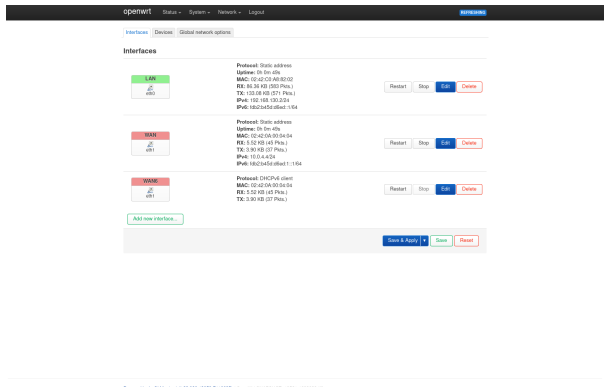
e configurazione di LuCI è molto semplice, si può infatti seguire la [guida ufficiale](#).



(a) Status page



(b) Graphs page



(c) Interfaces page

Figura 1.4: Interfaccia web LuCI

L'home page, fig. 1.4a, mostra un riepilogo dello stato del router, ad esempio sono presenti: informazioni sull'hardware, informazioni sulla memoria e storage, sono presenti inoltre informazioni riassuntive sulle interfacce di rete e sul *DHCP*. L'interfaccia LuCI è estensiva e permette di configurare quasi ogni aspetto del funzionamento del router, compreso il firewall, il *DHCP*, i processi in esecuzione, etc. Presenta inoltre la possibilità di installare plugins che modificano e/o aggiungono funzionalità non presenti di default nell'interfaccia.

Capitolo 2

Overview dell'architettura e delle componenti utilizzate

2.1 Obbiettivo da ottenere

TODO la sezione è da espandere! in qualche modo

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

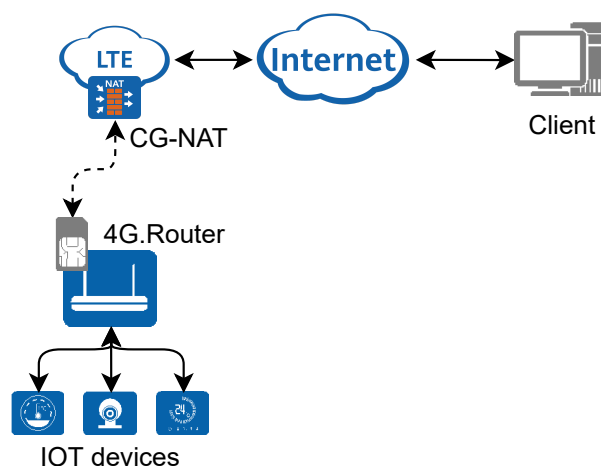


Figura 2.1: Schema concettuale dell'obiettivo da raggiungere.

Data la presenza del 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.

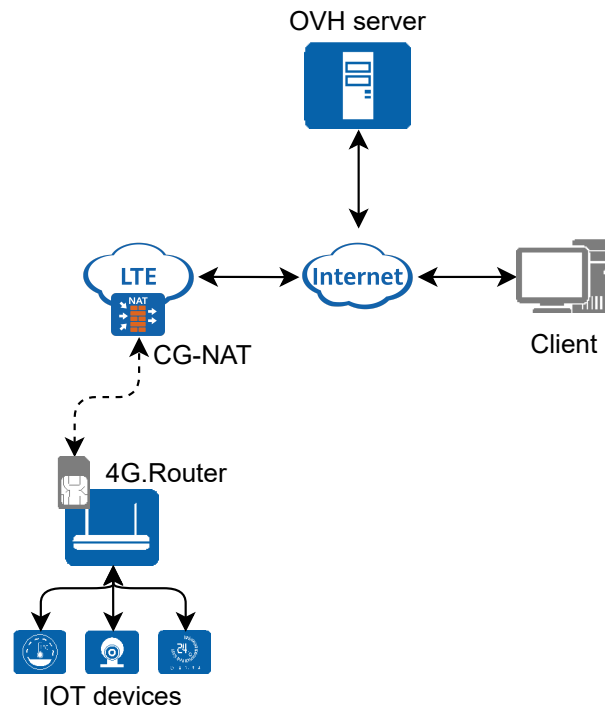


Figura 2.2: Schema concettuale dell'architettura che si dovrà implementare.

In questo modo si può usare il *Server* per configurare una *Virtual Private Network* (VPN), a cui saranno connessi sia il *4G.Router* che il cliente. Ciò consente la creazione di una topologia virtuale in cui tutti i dispositivi connessi alla VPN sono nella stessa rete locale, quindi possono comunicare tra loro.

Inoltre in questo modo viene minimizzata la configurazione da effettuare sulle macchine dei clienti, infatti sarà sufficiente avere un client OpenVPN.

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

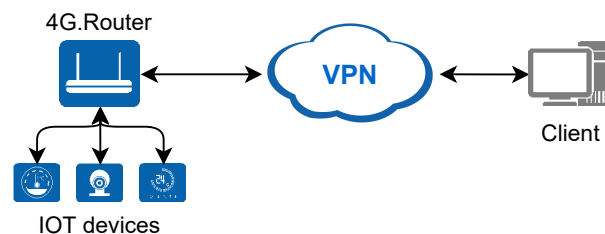


Figura 2.3: Topologia virtuale vista dal cliente.

2.2 Specifiche dei componenti

Per la realizzazione di questa topologia sono necessari i seguenti componenti:

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

Alcuni di questi componenti necessiteranno di delle caratteristiche specifiche.

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: Esse-ti 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
- 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

Ha il sistema operativo *Draghino*, una versione personalizzata di OpenWRT, in cui è stata modificata l'interfaccia web... mh no

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



(a) Schermata di autenticazione



(b) Grafico del traffico



(c) Schermata con stato riassuntivo

Figura 2.5: Interfaccia web Esse-ti 4G.Router

Per semplicità si farà 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. Dovrà infatti supportare un traffico simmetrico in upload / download.

Per la realizzazione della topologia è stata selezionata una macchina VPS del provider *OVH-Cloud*, 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 semplicità si farà riferimento alla *VPS OVHCloud* come *Server*.

2.2.3 Host domotico

Per simulare un'host domotico ed effettuare le varie operazioni di testing è stata usata una *raspberry pi*, con le utility *ping* e *tracert*.

2.2.4 Macchina del cliente

Per avere la massima flessibilità la macchina del cliente deve essere generica e non deve necessitare di nessuna configurazione specifica. Data la scelta di usare OpenVPN come provider VPN, e dato che OpenVPN è cross-platform, la macchina del cliente non ha specifiche di sistema operativo. L'unica necessità è di avere il client OpenVPN installato sul sistema, ad esempio:

- con sistema operativo Windows si deve scaricare l'eseguibile dal [sito ufficiale](#)
- su linux è sufficiente cercare nei repository ufficiali della distribuzione che si sta usando, es. Ubuntu: `apt-get install openvpn`.

Capitolo 3

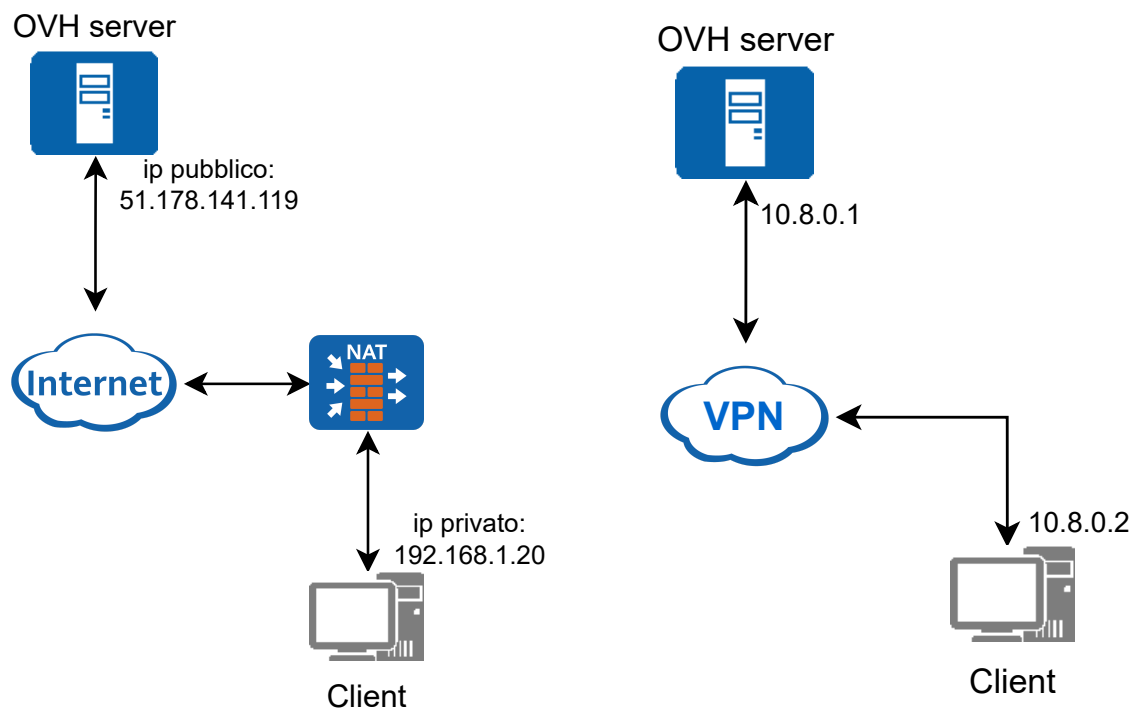
Configurazione del server

3.1 Overview della configurazione e prerequisiti

In questo capitolo andremo a installare e configurare *OpenVPN server* sulla VPS di OVHCloud.

Per facilitare la configurazione e il testing, supponiamo di partire da una topologia che contiene solo il server openvpn e un generico client. Come da specifiche il Server deve avere a disposizione un'ip pubblico e il client deve essere il più generico possibile, lo supponiamo quindi sotto a un NAT.

Supponiamo inoltre che l'ip pubblico del Server sia `51.178.141.119`, si avrà quindi una configurazione iniziale come in figura 3.1a.



(a) Configurazione di partenza per questo capitolo.

(b) Configurazione virtuale da raggiungere per questo capitolo.

Figura 3.1: Configurazione di partenza e di obiettivo per il capitolo 3.

Per instaurare una comunicazione bidirezionale tra il server e il client, si dovrà configurare opportunamente una rete VPN, che risulterà nella configurazione rappresentata in figura 3.1b.

3.2 Creazione della Public key infrastructure

work-in-progress

Per la gestione dell'autenticazione dei client alla VPN è necessario creare una *Public key infrastructure* (PKI), come descritto nella sezione 1.2.1. Inoltre, per una maggiore sicurezza è indicato separare la *Certificate Authority* dal *Server OpenVPN* [4], supponiamo quindi di usare un secondo server chiamato *Server CA*.

Il *Server CA* verrà usato in fase di configurazione del server e di creazione dei certificati per i client, dopodiché non sarà più necessario.

TODO: va bene senza mettere nessuna spiegazione per lo schema??

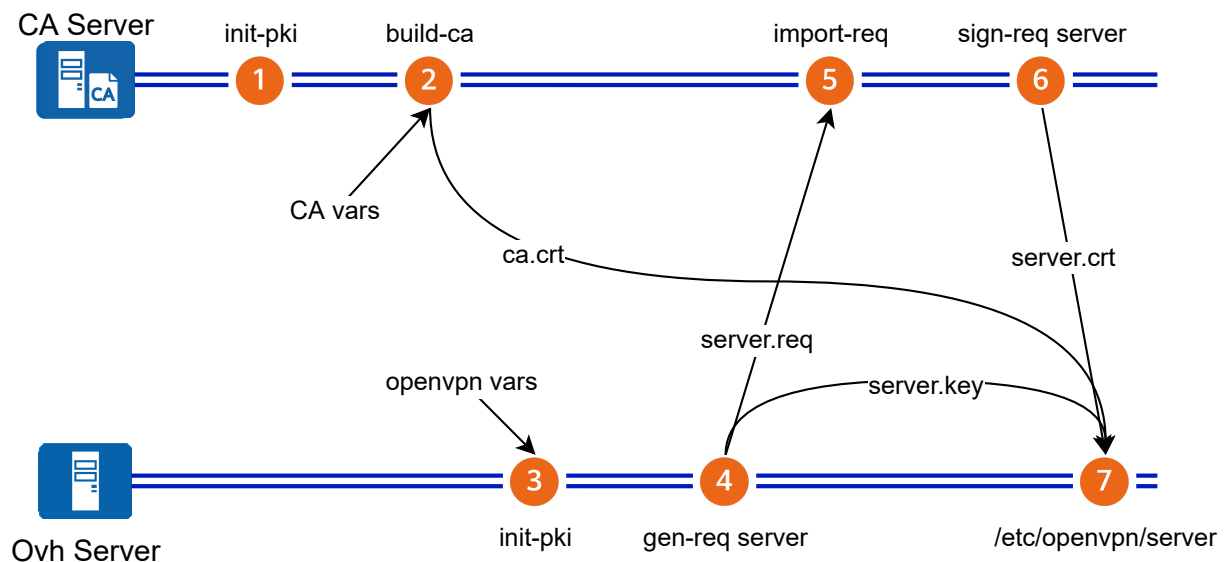


Figura 3.2: Diagramma per schematizzare la procedura di creazione della CA.

3.2.1 Creazione della struttura di cartelle necessaria per ospitare la PKI

Per la gestione della PKI verrà usato il tool *easy-rsa*, che fornisce un wrapper intorno alle funzionalità di *OpenSSL* facilitandone l'utilizzo.

Il pacchetto *easy-rsa* è presente nei repository ufficiali di ubuntu e può essere installato con il comando `sudo apt-get install easy-rsa`.

Dopo l'installazione verrà creata una struttura di cartelle pronta a ospitare la PKI nel percorso `/usr/share/easy-rsa/`, ma è sconsigliato usarlo per motivi di sicurezza. Quindi si deve spostare/collegare questo percorso in una cartella di un'utente non privilegiato, andiamo quindi a crearne un collegamento nella home:

Server CA

code: 3.2.1

```
1 $ mkdir ~/openvpn-ca
2 $ ln -s /usr/share/easy-rsa/* ~/openvpn-ca/ # soft link di easy-rsa nella home
3 $ chmod 700 /home/ubuntu/openvpn-ca/      # cambio i permessi in modo che solo l'utente
4                                           # corrente possa leggere il contenuto
```

A questo punto possiamo usare lo script `easyrsa` presente nella cartella per creare la PKI.

Server CA

code: 3.2.2

```
1 $ tree
2 .
3 |-- easyrsa -> /usr/share/easy-rsa/easyrsa
4 |-- openssl-easyrsa.cnf -> /usr/share/easy-rsa/openssl-easyrsa.cnf
5 |-- vars.example -> /usr/share/easy-rsa/vars.example
6 |-- x509-types -> /usr/share/easy-rsa/x509-types
```

3.2.2 Creazione della CA

Il primo step è la creazione della *Certificate Authority*, passaggio 1 fig.3.2.

Andiamo quindi a usare l'albero di cartelle creato in code: 3.2.1 e diamo il comando `easyrsa init-pki`, ciò creerà i file e le cartelle necessarie alla gestione della PKI.

Server CA

code: 3.2.3

```
1 $ cd openvpn-ca/
2 $ ./easyrsa init-pki
3 init-pki complete; you may now create a CA or requests.
4 Your newly created PKI dir is: /home/ubuntu/openvpn-ca/pki
```

Ora si devono personalizzare le variabili `vars`, si può sia partire da un file vuoto oppure modificare `vars.example` per poi rinominarlo `vars`.

Andiamo quindi a creare un nuovo file `vars`:

Server CA

code: 3.2.4

```
1 $ vim vars
2 set_var EASYRSA_REQ_COUNTRY "IT"
3 set_var EASYRSA_REQ_PROVINCE "MC"
4 set_var EASYRSA_REQ_CITY "Recanati"
5 set_var EASYRSA_REQ_ORG "Esse-ti"
6 set_var EASYRSA_REQ_EMAIL "s.gasparrini@esse-ti.it"
7 set_var EASYRSA_REQ_OU "Esse-ti"
8 set_var EASYRSA_REQ_CN "openvpn-ca"
9
10 set_var EASYRSA_ALGO "ec"
11 set_var EASYRSA_DIGEST "sha512"
```

work-in-progress

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 il tipo di algoritmo di cifratura ad *Elliptic Curve*

A questo punto si deve lanciare il comando `build-ca` per costruire la CA (passaggio 2 fig.3.2):

```
Server CA code: 3.2.5
1 $ ./easyrsa build-ca
2
3 Note: using Easy-RSA configuration from: ./vars
4
5 Using SSL: openssl OpenSSL 1.1.1f 31 Mar 2020
6
7 Enter New CA Key Passphrase:
8 Re-Enter New CA Key Passphrase:
9 read EC key
10 writing EC key
11
12 You are about to be asked to enter information that will be incorporated
13 into your certificate request.
14 What you are about to enter is what is called a Distinguished Name or a DN.
15 There are quite a few fields but you can leave some blank
16 For some fields there will be a default value,
17 If you enter '.', the field will be left blank.
18 -----
19 Common Name (eg: your user, host, or server name) [Easy-RSA CA]:
20
21 CA creation complete and you may now import and sign cert requests.
22 Your new CA certificate file for publishing is at:
23 /home/ubuntu/openvpn-ca/pki/ca.crt
24
```

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

3.2.3 Configurazione della PKI di OpenVPN

Il procedimento è 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 stiamo usando, in questo caso "ubuntu".

```
1 $ mkdir ~/openvpn-pki
2 $ ln -s /usr/share/easy-rsa/* ~/openvpn-pki/
3 $ sudo chown ubuntu ~/openvpn-pki/
4 $ chmod 700 ~/openvpn-pki/
5 $ cd ~/openvpn-pki/
```

Andiamo a creare un file `vars` e dare il comando `init-pki` :

```
1 $ vim vars
2 set_var EASYRSA_ALGO      "ec"
3 set_var EASYRSA_DIGEST   "sha512"
4 $ ./easyrsa init-pki
5 Note: using Easy-RSA configuration from: ./vars
6
7 init-pki complete; you may now create a CA or requests.
8 Your newly created PKI dir is: /home/ubuntu/openvpn-pki/pki
```

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

Come nome è stato scelto "server":

```
1 $ ./easyrsa gen-req server nopass
2
3 Note: using Easy-RSA configuration from: ./vars
4
5 Using SSL: openssl OpenSSL 1.1.1f 31 Mar 2020
6 Generating an EC private key
7 writing new private key to '/home/ubuntu/openvpn-pki/pki/private/server.key.438W2xM0g9'
8 -----
9 You are about to be asked to enter information that will be incorporated
10 into your certificate request.
11 What you are about to enter is what is called a Distinguished Name or a DN.
12 There are quite a few fields but you can leave some blank
13 For some fields there will be a default value,
14 If you enter '.', the field will be left blank.
15 -----
16 Common Name (eg: your user, host, or server name) [server]:
17
18 Keypair and certificate request completed. Your files are:
19 req: /home/ubuntu/openvpn-pki/pki/reqs/server.req
20 key: /home/ubuntu/openvpn-pki/pki/private/server.key
```

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

```
1 $ sudo cp /home/ubuntu/openvpn-pki/pki/private/server.key /etc/openvpn/server/
```

Il secondo file creato, `server.req`, corrisponde a una *Certificate Signing Request (CSR)* che va firmata e validata dalla CA.

In questo modo ogni client che si fida della CA si fiderà di conseguenza del server OpenVPN

3.2.4 Firma del certificato OpenVPN dalla CA

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

```
1 $ scp ubuntu@openvpn_server:/home/ubuntu/openvpn-pki/pki/reqs/server.req /tmp
```

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

```
1 $ cd ~/openvpn-ca
2 $ ./easyrsa import-req /tmp/server.req server
3 $ ./easyrsa sign-req server server
4 Using configuration from /home/ubuntu/openvpn-ca/pki/safessl-easyrsa.cnf
5 Check that the request matches the signature
6 Signature ok
7 The Subject's Distinguished Name is as follows
8 commonName          :ASN.1 12:'server'
9 Certificate is to be certified until Mar 11 15:50:45 2025 GMT (1080 days)
10
11 Write out database with 1 new entries
12 Data Base Updated
```

Verrà creato un file in `/openvpn-ca/pki/issued` chiamato `server.crt` che conterrà la chiave pubblica che verrà 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 OpenVPN*:

```
1 $ scp ubuntu@ca_server:/home/ubuntu/openvpn-ca/pki/issued/server.crt /tmp
2 $ scp ubuntu@ca_server:/home/ubuntu/openvpn-ca/pki/ca.crt /tmp
```

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

Server

code: 3.2.13

```
1 $ sudo cp /tmp/server.crt /etc/openvpn/server
2 $ sudo cp /tmp/ca.crt /etc/openvpn/server
```

3.3 Generazione della *tls-crypt pre-shared key*

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

La creazione va fatta sul *server OpenVPN*:

Server

code: 3.3.1

```
1 $ cd ~/openvpn-pki/
2 $ openvpn --genkey --secret ta.key
```

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

Server

code: 3.3.2

```
1 $ sudo cp ta.key /etc/openvpn/server
```

3.4 Generazione delle chiavi per i clients

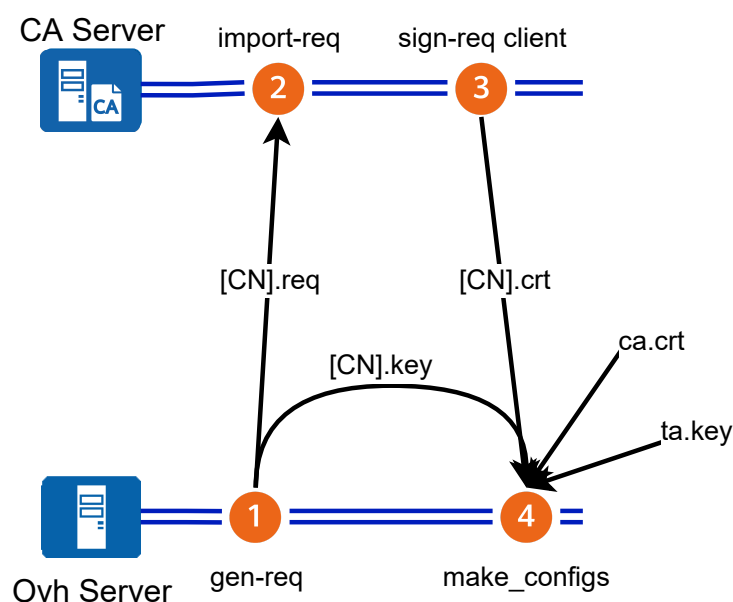


Figura 3.3: Diagramma per schematizzare la procedura di firma di un certificato client.

Creiamo una cartella nella *home* che ospiterà le chiavi dei *client* e le configurazioni openvpn:

Server	code: 3.4.1
<pre>1 \$ mkdir -p ~/client-configs/keys 2 \$ chmod -R 700 ~/client-configs</pre>	

Creiamo quindi un certificato per un *client*:

Server	code: 3.4.2
<pre>1 \$ cd ~/openvpn-pki/ 2 \$./easysrsa gen-req client1 nopass</pre>	

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

Server	code: 3.4.3
<pre>1 \$ cp pki/private/client1.key ~/client-configs/keys/ 2 \$ scp /home/ubuntu/openvpn-pki/pki/reqs/client1.req ubuntu@ca_server:/tmp</pre>	

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

Server CA	code: 3.4.4
<pre>1 \$ cd ~/openvpn-ca 2 \$./easysrsa import-req /tmp/client1.req client1 3 \$./easysrsa sign-req client client1 4 Using configuration from /home/ubuntu/openvpn-ca/pki/safessl-easysrsa.cnf 5 Check that the request matches the signature 6 Signature ok 7 The Subject's Distinguished Name is as follows 8 commonName :ASN.1 12:'client1' 9 Certificate is to be certified until Mar 16 13:15:09 2025 GMT (1080 days) 10 11 Write out database with 1 new entries 12 Data Base Updated</pre>	

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

Server	code: 3.4.5
<pre>1 \$ scp ubuntu@ca_server:/home/ubuntu/openvpn-ca/pki/issued/client1.crt /tmp</pre>	

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

```
1 $ cp /tmp/client1.crt ~/client-configs/keys/
2 $ cp ~/openvpn-pki/ta.key ~/client-configs/keys/
3 $ sudo cp /etc/openvpn/server/ca.crt ~/client-configs/keys/
4 $ sudo chown ubuntu:ubuntu ~/client-configs/keys/*
```

3.5 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 può usare la configurazione di esempio offerta da openvpn:

```
1 $ cd /etc/openvpn/server/
2 $ sudo wget "https://raw.githubusercontent.com/OpenVPN/openvpn/\
3     master/sample/sample-config-files/server.conf"
```

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

```
1 $ sudo vim server.conf
2 85 dh none           # non sono stati usati i parametri Diffie-Hellman
3 92 topology subnet   # topologia raccomandata
4 244 ;tls-auth ta.key 0 # This file is secret
5 245 tls-crypt ta.key  # selezione della preshared key
6 253 cipher AES-256-GCM # selezione della cifratura scelta
7 275 user nobody       # utente che eseguirà il server openvpn, in modo da
8                        # restringere i permessi
9 276 group nogroup     # stacca cosa per il gruppo
10 318 auth sha256       # selezione del metodo di autenticazione
```

3.6 Configurazioni sulla network stack del server openvpn

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

```
1 $ sudo vim /etc/sysctl.conf
2 69 net.ipv4.ip_forward = 1
3 $ sudo sysctl -p
4 net.ipv4.ip_forward = 1
```

3.7 Configurazione del firewall

Sulla VPS scelta è presente il firewall `firewalld`, ma per una più semplice configurazione è consigliato di disattivarlo e installare `ufw`:

Server	code: 3.7.1
<pre>1 \$ sudo systemctl mask firewalld 2 \$ sudo systemctl stop firewalld 3 \$ sudo apt-get install ufw 4 \$ sudo ufw allow ssh 5 Rule added 6 Rule added (v6) 7 \$ sudo ufw enable</pre>	

È importantissimo ricordarsi di consentire l'SSH prima di abilitare il firewall, altrimenti si perderà l'accesso alla VPS.

3.7.1 Configurazione del NAT

Per far sì 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 è l'interfaccia di rete del *server*, cioè quella che ha come ip il suo ip pubblico:

Server	code: 3.7.2
<pre>1 \$ ip addr 2 [...] 3 2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen ↪ 1000 4 link/ether a6:23:5f:48:ba:de brd ff:ff:ff:ff:ff:ff 5 inet 51.178.141.119/20 brd 51.178.141.255 scope global dynamic ens3 6 valid_lft 1857sec preferred_lft 1857sec 7 inet6 fe80::23:bfff:ac24:aace/64 scope link 8 valid_lft forever preferred_lft forever 9 [...]</pre>	

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

```
1 $ sudo vim /etc/ufw/before.rules
2 # ## rules.before
3 # ## Rules that should be run before the ufw command line added rules. Custom
4 # rules should be added to one of these chains:
5 # ufw-before-input
6 # ufw-before-output
7 # ufw-before-forward
8 #
9
10 # START OPENVPN RULES
11 # NAT table rules
12 *nat
13 :POSTROUTING ACCEPT [0:0]
14 # Allow traffic from OpenVPN client to ens3
15 -A POSTROUTING -s 10.8.0.0/24 -o ens3 -j MASQUERADE
16 COMMIT
17 # END OPENVPN RULES
18
19
20 # Don't delete these required lines, otherwise there will be errors
21 *filter
22 . . .
```

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

3.7.2 Configurazione del packet forwarding

Precedentemente abbiamo abilitato il forwarding nella network stack del server, ora si deve abilitare la corrispondente opzione nel firewall. Si deve quindi cambiare la regola di default per i pacchetti inoltrati da `DROP` ad `ACCEPT`.

Per farlo si deve modificare il file `/etc/default/ufw`:

```
1 $ sudo vim /etc/default/ufw
2 DEFAULT_FORWARD_POLICY="ACCEPT"
```

3.7.3 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:

```

1 $ sudo ufw allow 1194/udp
2 $ sudo ufw reload
3 $ sudo ufw status
4 Status: active
5 To Action From
6 --
7 22 ALLOW Anywhere
8 1194/udp ALLOW Anywhere
9 22 (v6) ALLOW Anywhere (v6)
10 1194/udp (v6) ALLOW Anywhere (v6)

```

3.8 Avvio del server OpenVPN

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

```

1 $ sudo systemctl enable openvpn-server@server.service
2 $ sudo systemctl start openvpn-server@server.service
3 $ sudo systemctl status openvpn-server@server.service
4 • openvpn-server@server.service - OpenVPN service for server
5   Loaded: loaded (/usr/lib/systemd/system/openvpn-server@.service; enabled; vendor
6   ↳ preset: disabled)
7   Active: active (running) since Mon 2022-04-18 13:08:44 CEST; 4h 22min ago
8   Docs: man:openvpn(8)
9         https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
10        https://community.openvpn.net/openvpn/wiki/HOWTO
11   Main PID: 436 (openvpn)
12   Status: "Initialization Sequence Completed"
13   Tasks: 1 (limit: 9488)
14   Memory: 4.8M
15   CPU: 199ms
16   CGroup: /system.slice/system-openvpn\x2dservice.slice/openvpn-server@server.service
17           └─436 /usr/bin/openvpn --status /run/openvpn-server/status-server.log
18           ↳ --status-version 2 --suppress-timestamps --config server.conf
19
20 Apr 18 13:08:44 server openvpn[436]: TUN/TAP device tun0 opened
21 Apr 18 13:08:44 server openvpn[436]: Incoming Control Channel Encryption: Cipher
22 ↳ 'AES-256-CTR' initialized with 256 bit key
23 Apr 18 13:08:44 server openvpn[436]: Incoming Control Channel Encryption: Using 256 bit
24 ↳ message hash 'SHA256' for HMAC authentication
25 Apr 18 13:08:44 server openvpn[436]: net_addr_v4_add: 10.8.0.1/24 dev tun0
26 Apr 18 13:08:44 server openvpn[436]: UDPv4 link local (bound): [AF_INET][undef]:1194
27 Apr 18 13:08:44 server openvpn[436]: UDPv4 link remote: [AF_UNSPEC]
28 Apr 18 13:08:44 server openvpn[436]: MULTI: multi_init called, r=256 v=256
29 Apr 18 13:08:44 server openvpn[436]: IFCONFIG POOL IPv4: base=10.8.0.2 size=253
30 Apr 18 13:08:44 server openvpn[436]: IFCONFIG POOL LIST
31 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 può verificare lo stato del servizio, si vede che il servizio è *active (running)*.

3.9 Script per la creazione delle configurazioni dei client

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

Server

code: 3.9.1

```
1 $ cd ~/client-configs/
2 $ wget "https://raw.githubusercontent.com/OpenVPN/openvpn\
3     /master/sample/sample-config-files/client.conf" \
4     -O base.conf
5 $ vim base.conf
6 42  remote 51.178.141.119 1194      # va messo l'ip e la porta del server OpenVPN
7 88  ;ca ca.crt                    # non useremo i file esterni ma ingloberemo
8 89  ;cert client.crt              # questi file in un file direttamente nella
9 90  ;key client.key               # configurazione del client
10 108 ;tls-auth ta.key 1           #
11 116 cipher AES-256-GCM          # cifratura usata
12 117 auth SHA256                 # autenticazione usata
13 118 key-direction 1             # indica che è un client
```

Ora creiamo lo script bash `make_config.sh`:

```
1 $ vim make_config.sh
2 #!/bin/bash
3
4 # usage:
5 # $ make_config.sh client1
6 # will use [ca.crt, client1.crt, client1.key, ta.key] to create client1.ovpn
7
8 KEY_DIR=~/.client-configs/keys
9 OUTPUT_DIR=~/.client-configs/files
10 BASE_CONFIG=~/.client-configs/base.conf
11
12 cat ${BASE_CONFIG} \
13     <(echo -e '<ca>' \
14     ${KEY_DIR}/ca.crt \
15     <(echo -e '</ca>\n<cert>' \
16     ${KEY_DIR}/${1}.crt \
17     <(echo -e '</cert>\n<key>' \
18     ${KEY_DIR}/${1}.key \
19     <(echo -e '</key>\n<tls-crypt>' \
20     ${KEY_DIR}/ta.key \
21     <(echo -e '</tls-crypt>' \
22     > ${OUTPUT_DIR}/${1}.ovpn
23 $ chmod 700 make_config.sh
```

Lo scopo di questo script è 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 *presheared key*.

Il tutto viene scritto in un file che ha lo stesso nome del *client* per cui si sta creando la configurazione ma `.ovpn`.

Quindi per creare la configurazione di *client 1*:

```
1 $ ./make_config.sh client1
```

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

3.10 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 spostiamo su una macchina client, con SO Linux ad esempio, e si può connettere il *client* alla vpn con la configurazione creata al passo precedente:


```
Client code: 3.10.1
1 $ sudo openvpn --config client1.ovpn
2 [...]
3 Thu Apr 21 12:53:04 2022 Outgoing Data Channel: Cipher 'AES-256-GCM' initialized with 256
  ↳ bit key
4 Thu Apr 21 12:53:04 2022 Incoming Data Channel: Cipher 'AES-256-GCM' initialized with 256
  ↳ bit key
5 Thu Apr 21 12:53:04 2022 ROUTE_GATEWAY 192.168.1.20/255.255.255.0 IFACE=eth0
  ↳ HWADDR=02:42:0a:00:04:03
6 Thu Apr 21 12:53:04 2022 /sbin/ip route add 10.8.0.1/32 via 10.8.0.2
7 Thu Apr 21 12:53:04 2022 WARNING: this configuration may cache passwords in memory -- use
  ↳ the auth-nocache option to prevent this
8 Thu Apr 21 12:53:04 2022 Initialization Sequence Completed
```

Se la configurazione fino a questo punto è corretta si avrà il messaggio

Initialization Sequence Completed.

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

```
Client code: 3.10.2
1 $ ip addr
2 2: tun0: <MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group
  ↳ default qlen 500
3     link/none
4     inet 10.8.0.2/24 scope global tun0
5         valid_lft forever preferred_lft forever
```

Si può vedere come l'ip assegnato al *client* dalla vpn è **10.8.0.2**.

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

```
Client code: 3.10.3
1 $ ping -c2 10.8.0.1
2 PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
3 64 bytes from 10.8.0.1: icmp_seq=1 ttl=64 time=0.250 ms
4 64 bytes from 10.8.0.1: icmp_seq=2 ttl=64 time=0.220 ms
```

Se nel frattempo si esegue la utility **tcpdump** sul server si potranno vedere i pacchetti *echo request* ed *echo reply*:

```
1 $ sudo tcpdump
2 listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
3 13:11:12.018615 IP 10.8.0.2 > 10.8.0.1: ICMP echo request, id 11, seq 1, length 64
4 13:11:12.018640 IP 10.8.0.1 > 10.8.0.2: ICMP echo reply, id 11, seq 1, length 64
5 13:11:13.039993 IP 10.8.0.2 > 10.8.0.1: ICMP echo request, id 11, seq 2, length 64
6 13:11:13.040018 IP 10.8.0.1 > 10.8.0.2: ICMP echo reply, id 11, seq 2, length 64
```

Si vede quindi che è possibile una comunicazione bidirezionale tra *client*, 10.8.0.2, e *server*, 10.8.0.1.

Capitolo 4

Configurazione Router

4.1 Overview della configurazione

In questo capitolo andremo a configurare il *router 4g* e connetterlo alla VPN.

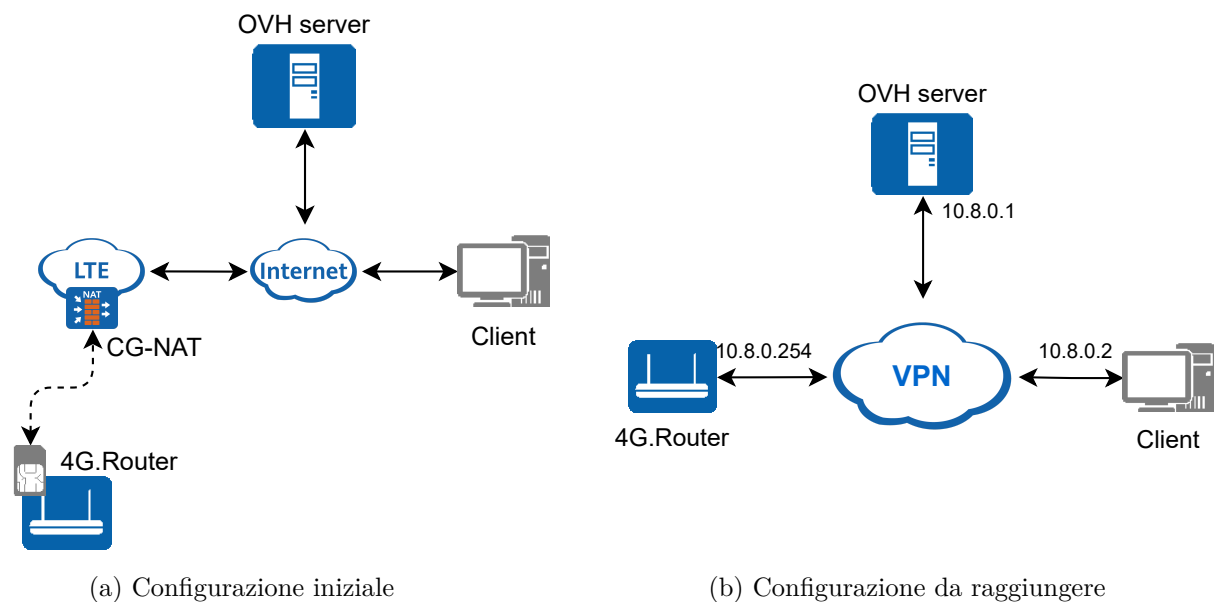


Figura 4.1: Schemi delle configurazioni iniziali e finali per il capitolo 4

4.2 Creazione della configurazione Openvpn

Si devono seguire gli step descritti in sezione 3.4, quindi creare la certificate request e firmarla nel *server CA*. Per poi usare lo script creato in sezione 3.9 per costruire il file di configurazione:

```
Server code: 4.2.1
1 $ ./make_config.sh router
```

Dopodiché si deve spostare il file `router.ovpn` dal *Server* al *Router 4g*, supponiamo di averlo

copiato nella cartella `/configs`.

Di default non è presente *OpenVPN* nel *Router 4g*, lo si può installare con:

```
Router 4g code: 4.2.2
1 $ opkg update
2 $ opkg install openvpn
3 $ opkg install luci-app-openvpn
```

Ora possiamo avviare il client openvpn:

```
Router 4g code: 4.2.3
1 $ openvpn --config /configs/router.ovpn
2 2022-04-29 17:26:37 OpenVPN 2.5.6 x86_64-openwrt-linux-gnu [SSL (mbed TLS)] [LZ4] [EPOLL]
   ↪ [MH/PKTINFO] [AEAD]
3 [...]
4 2022-04-29 17:26:37 VERIFY ECU OK
5 2022-04-29 17:26:37 VERIFY OK: depth=0, CN=server
6 2022-04-29 17:26:37 Control Channel: TLSv1.2, cipher
   ↪ TLS-ECDHE-RSA-WITH-AES-256-GCM-SHA384, 2048 bit key
7 2022-04-29 17:26:37 [server] Peer Connection Initiated with [AF_INET]10.0.4.2:1194
8 2022-04-29 17:26:37 net_addr_ptp_v4_add: 10.8.0.10 peer 10.8.0.9 dev tun0
9 2022-04-29 17:26:37 Initialization Sequence Completed
```

Se il file di configurazione è stato creato correttamente si vedrà il messaggio `Initialization Sequence Completed`.

Comparirà inoltre l'interfaccia `tun0` a cui è stato assegnato l'indirizzo `10.8.0.3`.

Per abilitare l'autostart di openvpn per il router si deve, per prima cosa, modificare il file `/etc/config/openvpn` in modo che faccia riferimento alla config corretta:

```
Router 4g code: 4.2.4
1 $ vim /etc/config/openvpn
2 20 option config /configs/router.ovpn
```

Ora possiamo abilitarla usando luci:

openwrt
Status
System
Network
VPN
Logout

OpenVPN

OpenVPN instances

Below is a list of configured OpenVPN instances and their current state

Name	Enabled	Started	Start/Stop	Port	Protocol	
custom_config	<input type="checkbox"/>	no	<button>start</button>	1194	udp	<button>Edit</button> <button>Delete</button>
sample_server	<input type="checkbox"/>	no	<button>start</button>	1194	udp	<button>Edit</button> <button>Delete</button>
sample_client	<input type="checkbox"/>	no	<button>start</button>	-	udp	<button>Edit</button> <button>Delete</button>

Template based configuration

OVPN configuration file upload

No file selected.

Figura 4.2: Configurazione della VPN tramite LuCI.

Alla riga "custom_config" Si deve mettere il check su *enabled* e premere start, per poi salvare le modifiche. In questo modo il router si conatterà automaticamente alla VPN anche se venisse riavviato.

4.3 Abilitazione del Client-to-Client nel server OpenVPN

In questo momento i client della VPN, *client1* e *router*, possono comunicare tra loro, ma lo fanno passando per la network stack del *server*. Infatti:

Router 4g
code: 4.3.1

```

1 $ ping -c2 10.8.0.2 # client1
2 PING 10.8.0.2 (10.8.0.2): 56 data bytes
3 64 bytes from 10.8.0.2: seq=0 ttl=63 time=0.519 ms
4 64 bytes from 10.8.0.2: seq=1 ttl=63 time=0.501 ms
5
6 --- 10.8.0.2 ping statistics ---
7 2 packets transmitted, 2 packets received, 0% packet loss
8 round-trip min/avg/max = 0.501/0.510/0.519 ms

```

Dal server possiamo vedere i pacchetti con *tcpdump*:

Server
code: 4.3.2

```

1 $ sudo tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
3 16:20:50.791063 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 0, length 64
4 16:20:50.791098 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 0, length 64
5 16:20:50.791273 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 0, length 64
6 16:20:50.791285 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 0, length 64
7 16:20:51.791153 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 1, length 64
8 16:20:51.791174 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 1, length 64
9 16:20:51.791365 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 1, length 64
10 16:20:51.791374 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 1, length 64

```

Si vede che ogni richiesta viene duplicata, la prima è in entrata sulla network stack del *server* e la seconda in uscita. Per evitare questo traffico possiamo abilitare l'opzione `client-to-client` nel file di configurazione del server. In questo modo il layer *openvpn* effettuerà direttamente il forwarding tra i client della vpn [13].

Server
code: 4.3.3

```

1 $ vim /etc/openvpn/server/server.conf
2 209 client-to-client
3 $ sudo systemctl restart openvpn-server@server.service

```

Possiamo quindi rieseguire gli stessi test fatti sopra:

Router 4g
code: 4.3.4

```

1 $ ping -c2 10.8.0.2
2 PING 10.8.0.2 (10.8.0.2): 56 data bytes
3 64 bytes from 10.8.0.2: seq=0 ttl=64 time=0.351 ms
4 64 bytes from 10.8.0.2: seq=1 ttl=64 time=0.307 ms
5
6 --- 10.8.0.2 ping statistics ---
7 2 packets transmitted, 2 packets received, 0% packet loss
8 round-trip min/avg/max = 0.307/0.329/0.351 ms

```

Ma questa volta la network stack del *server* non vede nessun pacchetto:

Server
code: 4.3.5

```

1 $ sudo tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes

```

4.4 Assegnazione ip statico al Router

Dato che non possiamo sapere a priori quanti client si connetteranno contemporaneamente alla VPN, per sapere sempre quale ip viene assegnato al router è necessario assegnargliene uno statico.

Questo viene fatto usando la `cleint-config-dir`.

```
Server code: 4.4.1
1 $ sudo mkdir /etc/openvpn/server/ccd # creo la client-config-dir
2 $ sudo vim /etc/openvpn/server/server.conf # abilito l'opzione nella config del server
3 167 client-config-dir ccd
4 $ sudo vim /etc/openvpn/server/ccd/router
5 ifconfig-push 10.8.0.254 255.255.255.0 # impongo l'ip per questo common name
6 $ sudo systemctl restart openvpn-server@server.service
```

Così facendo al router gli verrà sempre assegnato l'ip `10.8.0.254`, indipendentemente dall'ordine in cui gli host si connettono alla vpn. Ciò ci permette di sapere sempre e a priori qual è l'ip del router.

Capitolo 5

Connessione degli host domotici alla VPN

5.1 Overview della configurazione

L'ultima parte della configurazione consiste nel rendere disponibile nella VPN la rete locale del router 4g, consentendo quindi lo scambio di dati tra gli host domotici e i client della vpn

Per simulare gli host domotici è stata usata una *raspberry pi*.

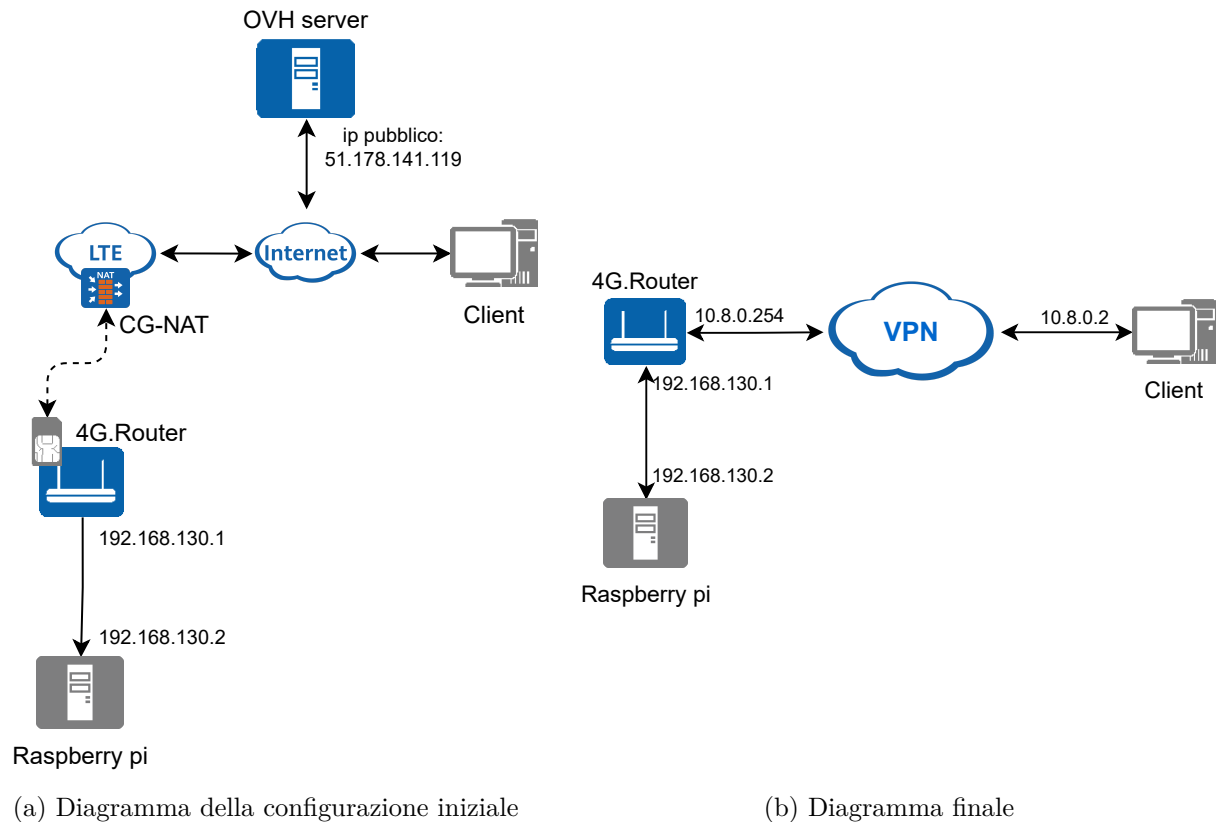


Figura 5.1: Schemi concettuali della configurazione iniziale e finale per il capitolo 5

5.2 Creazione della firewall zone per la VPN

Per rendere possibile la comunicazione tra la rete *lan* del *Router* e la VPN è necessario configurare opportunamente il firewall.

Ciò viene fatto direttamente da LuCI, nella sezione firewall si avranno preconfigurate 2 zone, lan e wan, come si vede figura 5.2.

openwrt Status System Network VPN Logout

General Settings Port Forwards Traffic Rules NAT Rules

Firewall - Zone Settings

The firewall creates zones over your network interfaces to control network traffic flow.

General Settings

Enable SYN-flood protection ☒

Drop invalid packets ☐

Input: accept

Output: accept

Forward: reject

Zones

Zone ⇒ Forwardings	Input	Output	Forward	Masquerading	
lan ⇒ wan	accept	accept	accept	<input type="checkbox"/>	≡ Edit Delete
wan ⇒ REJECT	reject	accept	reject	<input checked="" type="checkbox"/>	≡ Edit Delete

Add

Save & Apply Save Reset

Figura 5.2: Configurazione iniziale del firewall

Si deve quindi aggiungere una nuova zona, chiamata *vpn*, con le seguenti opzioni:

- Policy di forward: accept
- Forward consentito verso la zona lan
- Forward consentito dalla zona lan

Possiamo vedere la pagina di configurazione:

Firewall - Zone Settings

[General Settings](#)[Advanced Settings](#)[Conntrack Settings](#)

This section defines common properties of "this new zone". The *input* and *output* options set the default policies for traffic entering and leaving this zone while the *forward* option describes the policy for forwarded traffic between different networks within the zone. *Covered networks* specifies which available networks are members of this zone.

Name

vpn

Input

accept

Output

accept

Forward

accept

Masquerading

☐

MSS clamping

☐

Covered networks

unspecified

The options below control the forwarding policies between this zone (this new zone) and other zones. *Destination zones* cover forwarded traffic **originating from this new zone**. *Source zones* match forwarded traffic from other zones **targeted at this new zone**. The forwarding rule is *unidirectional*, e.g. a forward from lan to wan does *not* imply a permission to forward from wan to lan as well.

Allow forward to *destination zones*:

lan lan: 

Allow forward from *source zones*:

lan lan: 

Dismiss

Save

Figura 5.3: Configurazione del firewall

Dopo aver salvato, la pagina del firewall sar :

openwrt

Status ▾ System ▾ Network ▾ VPN ▾ Logout

UNSAVED CHANGES: 11

General Settings

Port Forwards

Traffic Rules

NAT Rules

Firewall - Zone Settings

The firewall creates zones over your network interfaces to control network traffic flow.

General Settings

Enable SYN-flood protection

☒

Drop invalid packets

☐

Input

accept ▾

Output

accept ▾

Forward

reject ▾

Zones

Zone → Forwardings	Input	Output	Forward	Masquerading	
lan → wan vpn	accept ▾	accept ▾	accept ▾	<input type="checkbox"/>	⋮ Edit Delete
wan → REJECT	reject ▾	accept ▾	reject ▾	<input checked="" type="checkbox"/>	⋮ Edit Delete
vpn → lan	accept ▾	accept ▾	accept ▾	<input type="checkbox"/>	⋮ Edit Delete

Add

Save & Apply ▾

Save

Reset

Figura 5.4: Configurazione finale del firewall

5.3 Aggiunta dell'interfaccia tun0 alla zona vpn

Per rendere attiva la zona firewall appena creata gli si deve assegnare almeno un'interfaccia, in questo caso gli si deve assegnare l'interfaccia tun0.

Ciò va fatto da LuCI, nella sezione *interfaces*. Si deve quindi aggiungere una nuova interfaccia con le seguenti opzioni:

- Name: tun0
- Proto: static
- Device: tun0
- ipv4 address: 10.8.0.254
- ipv4 netmask: 255.255.255.0
- Assign firewall zone: vpn

Interfaces » TUN0

General Settings | Advanced Settings | Firewall Settings | DHCP Server

Status: **Device:** tun0
RX: 504 B (6 Pkts.)
TX: 808 B (10 Pkts.)

Protocol: Static address

Device: tun0

Bring up on boot: ☒

IPv4 address: 10.8.0.254

IPv4 netmask: 255.255.255.0

IPv4 gateway:

IPv4 broadcast: 10.8.0.255

IPv6 address: Add IPv6 address... +

IPv6 gateway:

IPv6 routed prefix:

? Public prefix routed to this device for distribution to clients.

Dismiss Save

(a) General settings

Interfaces » TUN0

General Settings | Advanced Settings | Firewall Settings | DHCP Server

Create / Assign firewall-zone: vpn (empty)

? Choose the firewall zone you want to assign to this interface. Select *unspecified* to remove the interface from the associated zone or fill out the *custom* field to define a new zone and attach the interface to it.

Dismiss Save

(b) Firewall settings

Figura 5.5: Assegnazione interfaccia *tun0* alla zona firewall *vpn* tramite interfaccia LuCI

5.4 Modifiche alla configurazione OpenVPN del server

Per instaurare una comunicazione bidirezionale tra *host-domotico* e *client* è necessario che:

1. il server vpn sia consapevole che il *router* vuole esporre una sua sottorete verso la vpn
2. il client abbia l'opportuna rotta per raggiungere la sottorete dove si trova l'*host-domotico*

Per il punto 1 è necessario aggiungere la seguente riga nel file

`/etc/openvpn/server/ccd/router` :

Server		code: 5.4.1
1	\$ vim /etc/openvpn/server/ccd/router	
2	ifconfig-push 10.8.0.254 255.255.255.0	
3	iroute 192.168.130.0 255.255.255.0	
4	\$ vim /etc/openvpn/server/server.conf	
5	169 route 192.168.130.0 255.255.255.0 # aggiungo la rotta nel server	

Per il punto 2 è necessario aggiungere la seguente riga alla configurazione OpenVPN nel *Server*:

Server		code: 5.4.2
1	\$ vim /etc/openvpn/server/server.conf	
2	168 push "route 192.168.130.0 255.255.255.0"	

Dopo aver modificato la configurazione del server va riavviarlo con `systemctl restart`.

In questo modo nella procedura di connessione alla VPN, i client aggiungeranno la rotta verso la sottorete 192.168.130.0/24 nella loro tabella di routing. Possiamo verificarlo con:

Client		code: 5.4.3
1	\$ ip route	
2	[...]	
3	192.168.130.0/24 via 10.8.0.1 dev tun0	

5.5 Test della configurazione

TODO: da scrivere di più per i test, sono importanti, forse mettere qualcosa sul trouble shooting... mhhh

Per testare la connessione tra *client* e *host-domotico* usiamo sia `ping` che `tracpath`, possiamo inoltre usare `tcpdump` sul *Server* e *Router* per vedere il percorso dei pacchetti.

Vediamo quindi un ping dal client verso l'*host-domotico*, con server e router in ascolto sull'interfaccia tun0:

Client		code: 5.5.1
1	\$ ping -c 1 192.168.130.3	
2	PING 192.168.130.3 (192.168.130.3) 56(84) bytes of data.	
3	64 bytes from 192.168.130.3: icmp_seq=1 ttl=63 time=0.643 ms	
4		
5	--- 192.168.130.3 ping statistics ---	
6	1 packets transmitted, 1 received, 0% packet loss, time 0ms	
7	rtt min/avg/max/mdev = 0.643/0.643/0.643/0.000 ms	

Server

code: 5.5.2

```
1 $ tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
```

Il server non ha visto traffico poiché ha l'opzione `client-to-client` abilitata, ciò comporta che il layer vpn effettua il routing senza passare per l'interfaccia di rete del router.

Router

code: 5.5.3

```
1 $ tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
3 10:44:07.228387 IP 10.8.0.2 > host-domotico: ICMP echo request, id 12, seq 1, length 64
4 10:44:07.228550 IP host-domotico > 10.8.0.2: ICMP echo reply, id 12, seq 1, length 64
```

Il router vede il pacchetto e lo instrada verso l'host corretto.

Provando nella direzione inversa si ottiene lo stesso risultato

Host-domotico

code: 5.5.4

```
1 $ ping -c 1 10.8.0.2
2 PING 10.8.0.2 (10.8.0.2) 56(84) bytes of data.
3 64 bytes from 10.8.0.2: icmp_seq=1 ttl=63 time=0.428 ms
4
5 --- 10.8.0.2 ping statistics ---
6 1 packets transmitted, 1 received, 0% packet loss, time 0ms
7 rtt min/avg/max/mdev = 0.428/0.428/0.428/0.000 ms
```

e il router vede:

Router

code: 5.5.5

```
1 $ tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
3 11:56:10.469901 IP host-domotico > 10.8.0.2: ICMP echo request, id 14, seq 1, length 64
4 11:56:10.470230 IP 10.8.0.2 > host-domotico: ICMP echo reply, id 14, seq 1, length 64
```

Possiamo inoltre usare `tracpath` per vedere gli hop:

Client

code: 5.5.6

```
1 $ tracpath 192.168.130.3
2 1?: [LOCALHOST] pmtu 1500
3 1: 10.8.0.254 0.471ms
4 1: 10.8.0.254 0.473ms
5 2: 192.168.130.3 0.567ms reached
6 Resume: pmtu 1500 hops 2 back 2
```

```
1 $ tracepath 10.8.0.2
2 1?: [LOCALHOST] pmtu 1500
3 1: router-1 0.062ms
4 1: router-1 0.068ms
5 2: 10.8.0.2 0.592ms reached
6 Resume: pmtu 1500 hops 2 back 2
```

Capitolo 6

Conclusione

halo

ciao a tutti!!

Bibliografia

- [1] R. Braden. Requirements for internet hosts - communication layers. STD 3, RFC Editor, October 1989. <http://www.rfc-editor.org/rfc/rfc1122.txt>.
- [2] K. B. Egevang and P. Francis. The ip network address translator (nat). RFC 1631, RFC Editor, May 1994. <http://www.rfc-editor.org/rfc/rfc1631.txt>.
- [3] Huawei. Product image gallery. https://info.support.huawei.com/network/imagelib/getImagePartList?product_family=Router&product_type=Access%20Router%7CIOT%20Gateway&domain=&lang=en.
- [4] jesrush (<https://serverfault.com/users/480025/jesrush>). Openvpn server as it's own ca – security concerns? Server Fault Stack Exchange. URL:<https://serverfault.com/q/923049/558773> (version: 2018-05-22).
- [5] JGraph. draw.io, 10 2021. <https://www.diagrams.net/>.
- [6] H. R. Max Roser and E. Ortiz-Ospina. Internet. *Our World in Data*, 2015. <https://ourworldindata.org/internet>.
- [7] J. Postel. Internet protocol. STD 5, RFC Editor, September 1981. <http://www.rfc-editor.org/rfc/rfc791.txt>.
- [8] J. Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. <http://www.rfc-editor.org/rfc/rfc793.txt>.
- [9] E. Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor, August 2018. <http://www.rfc-editor.org/rfc/rfc8446.txt>.
- [10] Wikipedia. Ethernet II — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Ethernet%5C%20frame&oldid=1080405638#Ethernet_II, 2022. [Online; accessed 04-June-2022].
- [11] Wikipedia. IPv4 address exhaustion — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=IPv4%20address%20exhaustion&oldid=1090374520>, 2022. [Online; accessed 04-June-2022].
- [12] Wikipedia. Public key infrastructure — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Public%20key%20infrastructure&oldid=1085755772>, 2022. [Online; accessed 04-June-2022].
- [13] ysdx (<https://serverfault.com/users/84729/ysdx>). Openvpn client-to-client. Server Fault Stack Exchange. URL:<https://serverfault.com/a/738558/558773> (version: 2018-01-17).