



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

TODO: da riscrivere

Indice

Prefazione	<i>TODO: da riscrivere</i>	ii
Indice		iii
Elenco delle figure		vi
1	Introduzione	1
1.1	TCP/IP e modello a strati OK <i>TODO: non ho minimamente messo la distinzione ip pubblico / privato</i>	1
1.1.1	Internet Protocol Suite OK	2
1.1.2	Incapsulamento work-in-progress	2
1.1.3	IP Network Address Translator (NAT) work-in-progress	4
1.2	Openvpn OK	4
1.2.1	Crittografia e autenticazione OK	5
1.2.2	Networking work-in-progress	5
1.3	OpenWrt OK	6
1.3.1	LuCI web interface OK	6
2	Overview dell'architettura e delle componenti utilizzate <i>TODO: da rivedere è un disastro</i>	8
2.1	Obbiettivo da ottenere work-in-progress	8
2.2	Specifiche dei componenti	10
2.2.1	Esse-ti 4G.Router work-in-progress	10
2.2.2	VPS OVHCloud	11
2.2.3	Host domotico work-in-progress	12
2.2.4	Macchina del cliente work-in-progress	12
3	Configurazione del <i>Server</i> OK	13
3.1	Overview della configurazione e prerequisiti OK	13

3.2	Creazione della <i>Public key infrastructure</i> OK	14
3.2.1	Creazione della struttura di cartelle necessaria per ospitare la PKI OK	15
3.2.2	Creazione della <i>Certificate Authority</i> CA OK	16
3.2.3	Configurazione della PKI di OpenVPN OK	17
3.2.4	Firma del certificato OpenVPN dalla CA OK	19
3.3	Generazione della <i>tls-crypt pre-shared key</i> OK	19
3.4	Generazione dei certificati per i <i>clients</i> OK	20
3.4.1	Script per la creazione delle configurazioni dei client OK	21
3.5	Creazione del file di configurazione del Server OpenVPN OK	23
3.6	Configurazioni sulla network stack del Server OK	23
3.7	Configurazione del firewall	24
3.7.1	Configurazione del NAT OK	24
3.7.2	Configurazione del packet forwarding OK	25
3.7.3	Conclusione della configurazione del firewall OK	25
3.8	Avvio del Server OpenVPN OK	26
3.9	Test della configurazione OK <i>TODO: si può espandere in qualche modo?</i>	27
4	Configurazione del <i>Router</i>	30
4.1	Overview della configurazione OK	30
4.2	Prerequisiti per la configurazione del <i>Router</i> <i>TODO: molto male da sistemare</i>	31
4.3	Creazione della configurazione OpenVPN per il <i>Router</i> OK	31
4.4	Connessione del <i>Router</i> alla VPN OK	31
4.5	Auto start del <i>Client VPN</i> nel <i>Router</i> OK	32
4.6	Abilitazione del Client-to-Client nel server OpenVPN <i>TODO: da rileggere</i>	33
4.7	Assegnazione IP statico al <i>Router</i> OK	35
5	Connessione degli Host domotici alla VPN	36
5.1	Overview della configurazione	36
5.2	Creazione della zona firewall per la VPN	37
5.3	Aggiunta dell'interfaccia <i>tun0</i> alla zona firewall <i>vpn</i>	40
5.4	Modifiche alla configurazione OpenVPN del <i>Server</i>	42
5.5	Test e analisi della configurazione work-in-progress	43
6	Conclusione work-in-progress	47

Elenco delle figure

1.1	Rappresentazione degli strati del modello TCP/Ip, con relativo incapsulamento e dispositivo di dominio	1
1.2	Incapsulamento	3
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.	14
3.2	Diagramma per schematizzare la procedura di creazione della CA e della PKI del server OpenVPN.	15
3.3	Diagramma per schematizzare la procedura di firma di un certificato client.	20
4.1	Schemi delle configurazioni iniziali e finali per il capitolo 4.	30
4.2	Configurazione della VPN tramite LuCI.	32
4.3	Effetto del <code>client-to-client</code> sulla network stack del <i>Server</i> [15].	34
5.1	Schemi concettuali della configurazione iniziale e finale per il capitolo 5	37
5.2	Configurazione di default delle zone del firewall.	38
5.3	Schermata di aggiunta di una nuova zona firewall.	39
5.4	Configurazione delle zone firewall dopo l'aggiunta della zona <i>vpn</i>	40
5.5	Assegnazione interfaccia <i>tun0</i> alla zona firewall <i>vpn</i> tramite interfaccia LuCI	41
5.6	Diagramma di test realizzato con il <i>Router</i> e il <i>Client</i>	44
5.7	Diagramma di test virtuale	45

6.1	multi-istanza real	48
6.2	multi-istanza virtual	48

Tutti i diagrammi sono stati realizzati con il tool *diagrams.net* [5], usando le icone fornite da Huawei [3].

Capitolo 1

Introduzione

1.1 TCP/IP e modello a strati OK TODO: non ho minimamente messo la distinzione ip pubblico / privato

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, il modello TCP/IP è costituito da 4 strati.

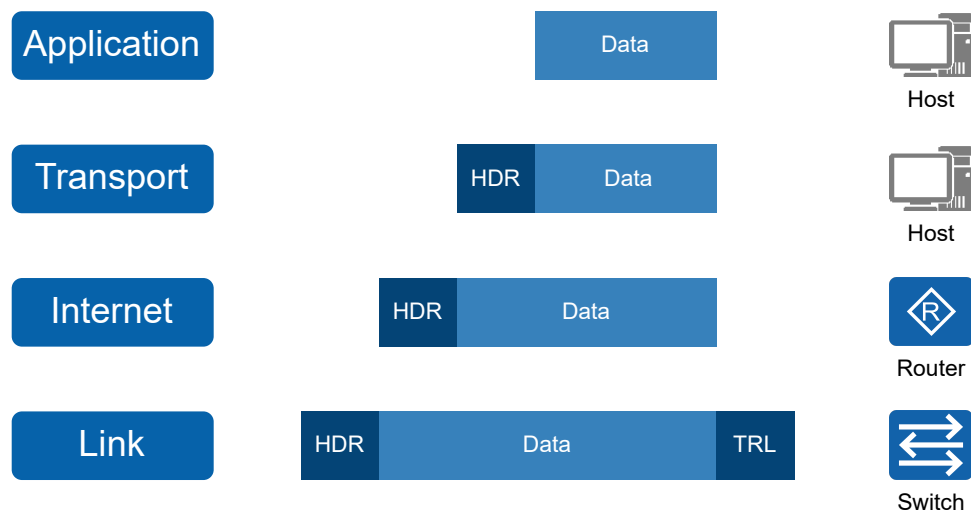


Figura 1.1: Rappresentazione degli strati del modello TCP/Ip, con relativo incapsulamento e dispositivo di dominio

1.1.1 Internet Protocol Suite OK

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.

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](#). 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](#).

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](#) e [ICMP](#) ad esempio.

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 work-in-progress

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.

TODO: togli ripetizione di possiamo vedere

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  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**[12]: 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 qual è l’interfaccia fisica a cui è destinato il pacchetto.

Possiamo vedere graficamente il pacchetto descritto in code:1.1.1 e le informazioni principali di ogni header:

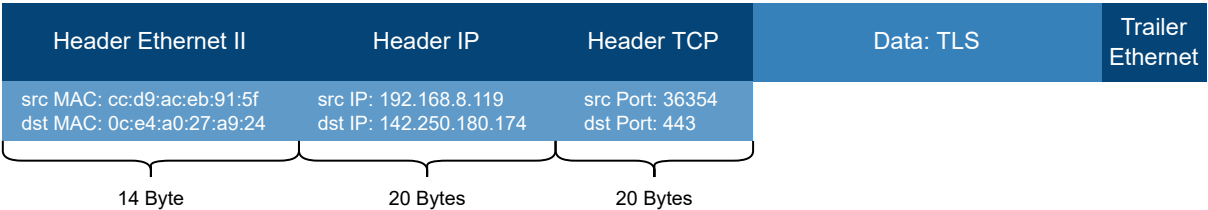


Figura 1.2: Incapsulamento

1.1.3 IP Network Address Translator (NAT) work-in-progress

Per mitigare il problema della saturazione degli indirizzi IPv4 [13], 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 multi-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.

TODO: da qua in poi è da rieleggere sta molto work in progress

Dato il funzionamento del NAT è intrinsecamente impossibile comunicare dall'esterno verso un dispositivo al di sotto del NAT a meno che non venga opportunamente configurato per consentirlo.

Prendiamo in esempio un NAT di tipo *easyip*, in questo caso un'intera sottorete viene mappata su un singolo ip esterno. Ogni volta che un'host della rete tenta di comunicare con un server esterno, nel NAT viene salvata la relazione tra chi ha inviato la richiesta e su quale porta esterna è stata inviata verso il server. Data questa relazione il NAT è in grado di re-inoltrare all'host corretto la risposta del server. Si vede quindi che la comunicazione attraverso un NAT è possibile solo se viene iniziata da un'host della rete privata, poichè in caso contrario il NAT non potrebbe conoscere a quale host stiamo tentando di comunicare e quindi semplicemente scarterebbe il pacchetto.

Carrier Grade NAT (CG-NAT)

I CG-NAT [11] 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 OK

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 OK

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* [14] 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 allo 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 work-in-progress

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).

TODO: da rivedere!

In figura 1.3 possiamo vedere l'incapsulamento del pacchetto OpenVPN descritto in code: 1.2.1. Si vede inoltre come il contenuto del campo data del pacchetto OpenVPN contenga al suo interno un pacchetto IP, ciò non è visibile dall'esterno perchè il tutto è opportunamente cifrato.

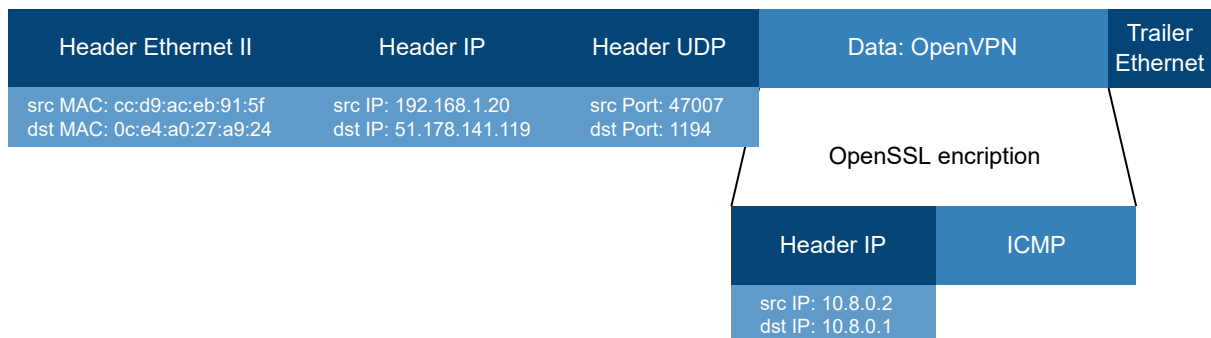


Figura 1.3: Incapsulamento

1.3 OpenWrt OK

```
Login banner di Openwrt                                     code: 1.3.1
1 BusyBox v1.35.0 (2022-04-24 21:09:51 UTC) built-in shell (ash)
2 -----
3 |      |.-----|.-----|. | | |.-----| |
4 | -  | | _ | -__| | | | | | _| | _|
5 |_____| | _|_____| | | |_____| | _| |_____|
6 |__| W I R E L E S S   F R E E D O M
7 -----
8 OpenWrt SNAPSHOT, r19521-46980294f6
9 -----
```

OpenWRT è un sistema operativo open-source per sistemi embedded, principalmente usato come firmware alternativo per router domestici. Si basa su kernel linux, con specifica attenzione all'ottimizzazione per renderlo adatto a dispositivi con risorse estremamente limitate.

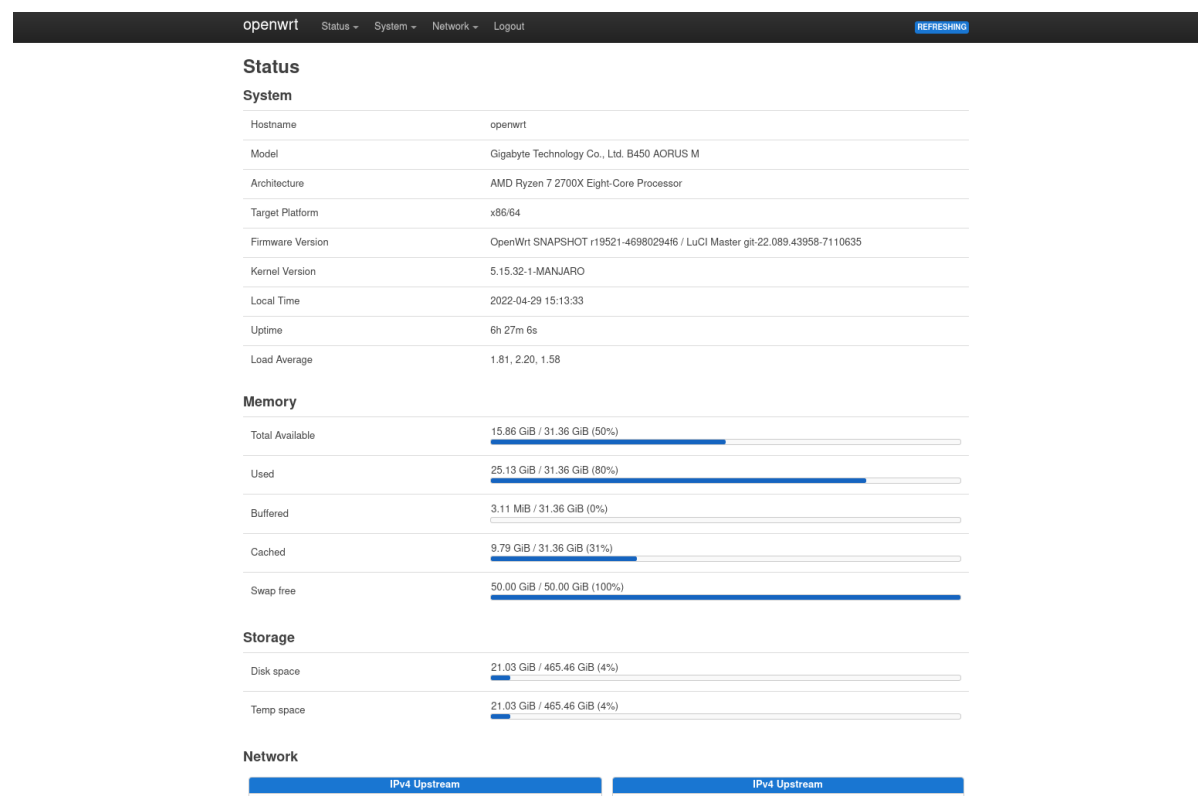
Al contrario di altri sistemi operativi per dispositivi embedded, OpenWRT presenta un filesystem con permessi di scrittura. Questo permette di modificare il funzionamento del sistema senza dover reinstallare l'intero firmware a ogni modifica. Per facilitare l'installazione delle funzionalità aggiuntive ha il suo gestore pacchetti ([opkg](#)), ciò lo rende estremamente estensibile e configurabile.

OpenWRT può essere configurato sia tramite shell ([ash](#)) che tramite interfaccia web ([LuCI](#)).

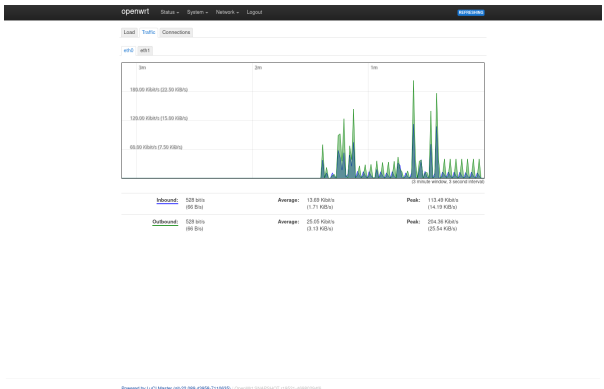
1.3.1 LuCI web interface OK

LuCI è l'interfaccia web ufficiale di *OpenWrt*, è un progetto open-source nato dalla necessità di un'interfaccia web per sistemi embedded che sia gratuita, completa ed estensibile. L'installazione

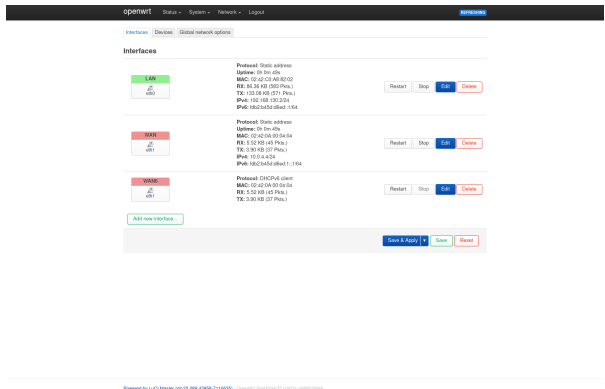
e configurazione di LuCI è molto semplice, si può infatti seguire la [guida ufficiale](#).



(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

TODO: da rivedere è un disastro

2.1 Obiettivo da ottenere *work-in-progress*

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

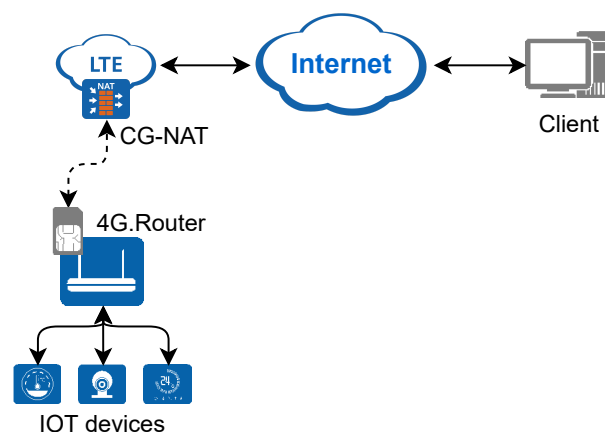


Figura 2.1: Schema concettuale dell'obiettivo da raggiungere.

TODO: espandere menzionando la sim

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.

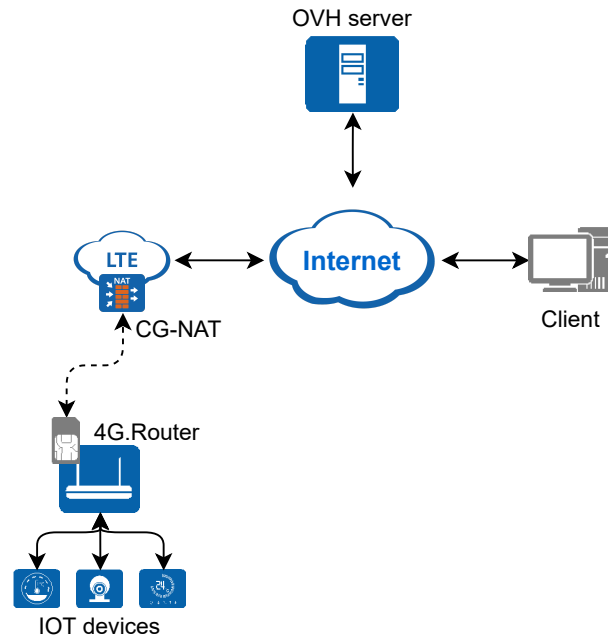


Figura 2.2: Schema concettuale dell'architettura che si dovrà implementare.

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:

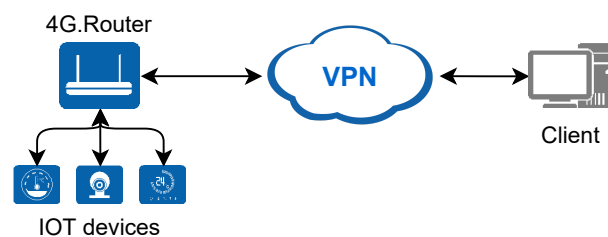


Figura 2.3: Topologia virtuale vista dal cliente.

2.2 Specifiche dei componenti

TODO: declassare da subsection a qualco di più basso

2.2.1 Esse-ti 4G.Router [work-in-progress](#)

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

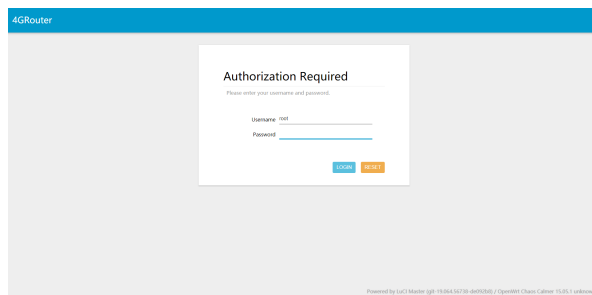
Per l'implementazione di questa architettura sono necessarie solo un sub-set delle specifiche:

- 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

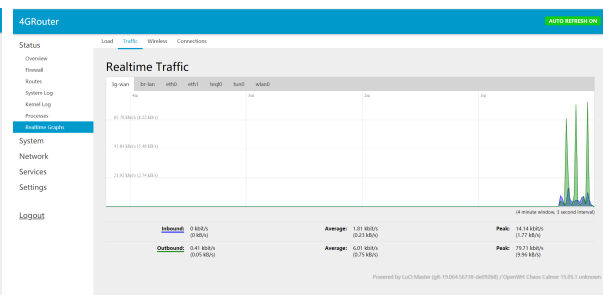
TODO: meglio ma si può fare meglio

Il sistema operativo del router è una versione personalizzata di OpenWRT, le funzionalità sono le stesse della versione OpenSource ma la grafica è leggermente differente ed è preconfigurato con le impostazioni per la gestione del 4G.

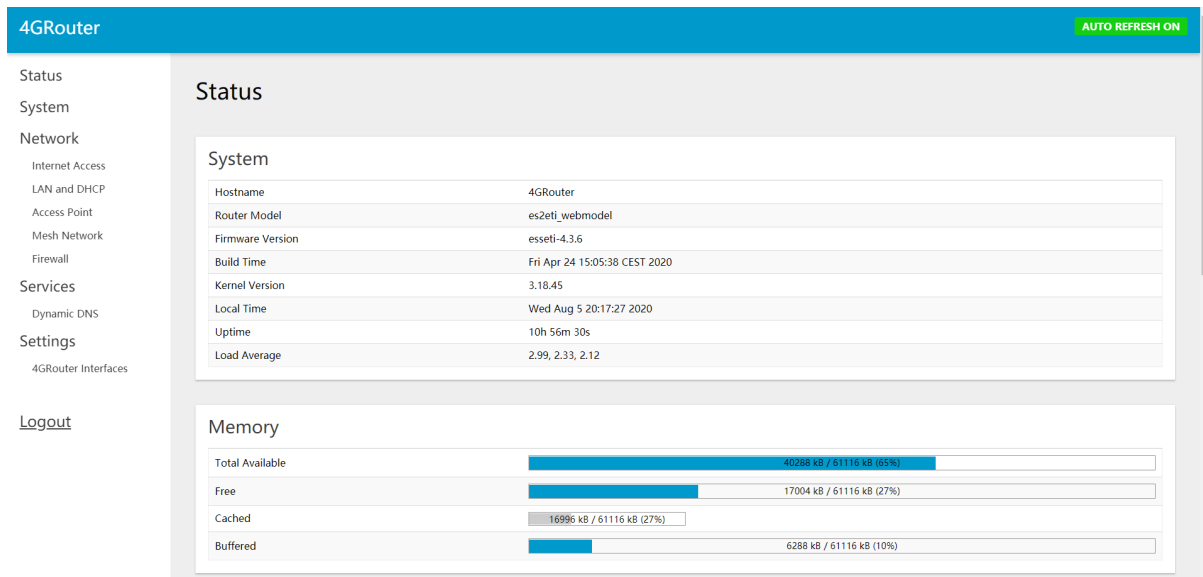
La configurazione del dispositivo può essere fatta sia da terminale, entrando in ssh, sia da interfaccia web:



(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 *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 semplicità si farà riferimento alla *VPS OVHCloud* come *Server*.

2.2.3 Host domotico [work-in-progress](#)

L'obiettivo dell'azienda *Esse-ti* è quello di fornire ai clienti una connessione diretta verso dispositivi domotici posti in una locazione remota. In questa topologia un'host domotico potrebbe essere un qualunque dispositivo di rete, una telecamera di videosorveglianza, un termostato digitale o un dispositivo più complesso come una raspberry pi.

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 [work-in-progress](#)

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:

TODO: don't like forse lo tolgo

- 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* OK

3.1 Overview della configurazione e prerequisiti OK

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 (sezione 2.2.2); e il client deve essere il più generico possibile, lo supponiamo quindi sotto a un NAT (sezione 2.2.4).

TODO: non supponiamo, da cambiare

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

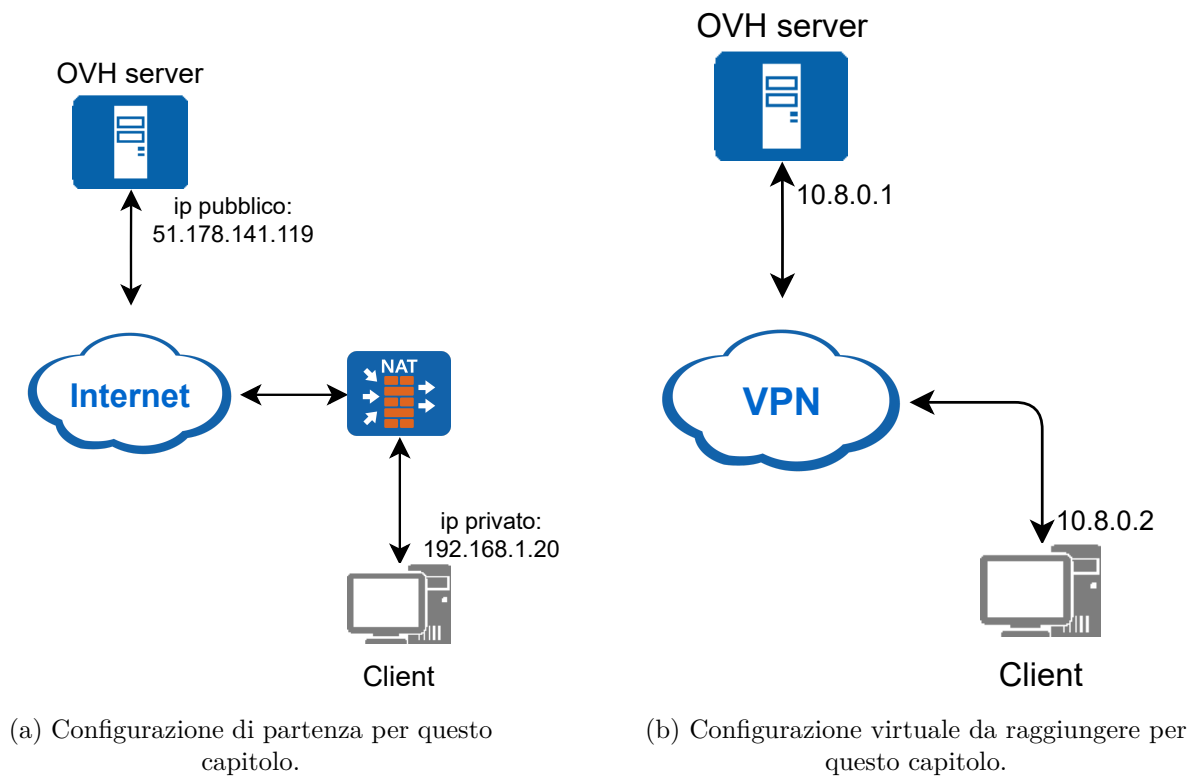


Figura 3.1: Configurazione di partenza e di obiettivo per il capitolo 3.

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

3.2 Creazione della *Public key infrastructure* OK

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??

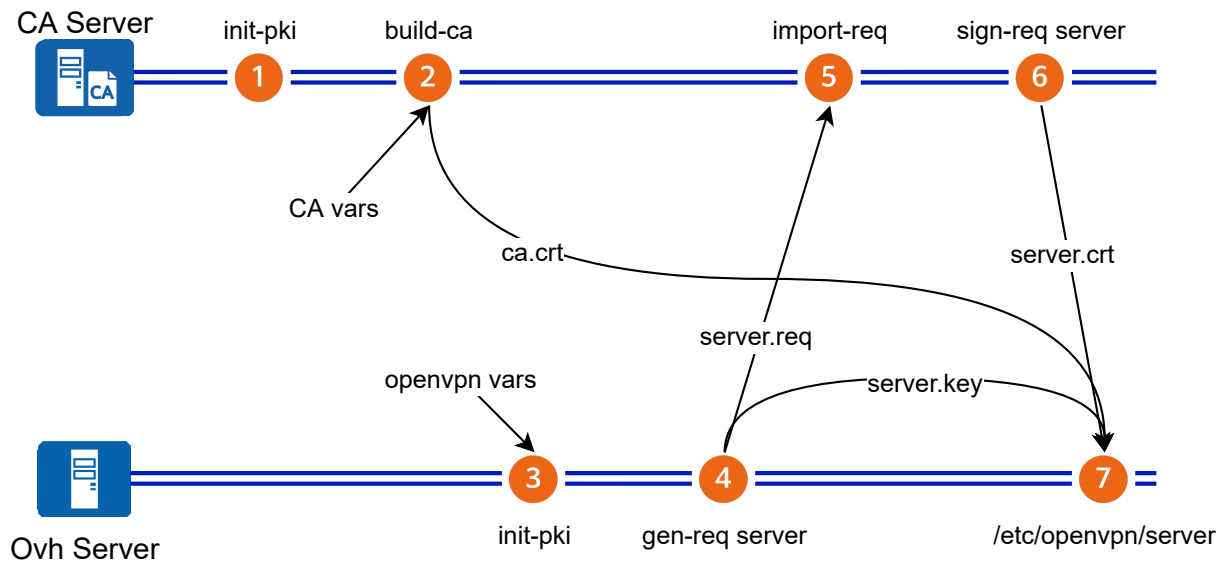


Figura 3.2: Diagramma per schematizzare la procedura di creazione della CA e della PKI del server OpenVPN.

3.2.1 Creazione della struttura di cartelle necessaria per ospitare la PKI OK

TODO: va bene wrapper??? :joy:

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`.

TODO: va bene sta frase???

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 dell'utente `ubuntu`:

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.

```
1 $ tree ~/openvpn-ca/  
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 *Certificate Authority* CA OK

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:

```
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`:

TODO: è ok lasciare queste?

```
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"
```

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 per usare la crittografia a chiave ellittica ([Elliptic-Curve Cryptography](#)); L'ultima opzione setta l'algoritmo di hashing da usare nella firma dei certificati.

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


```
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
```

TODO: criptare o cifrare?

Eseguendo il comando verrà chiesto di inserire una passphrase, che verrà usata per criptare la chiave privata appena generata. Se non si vuole criptare la chiave privata, basta lasciare il campo vuoto. Il secondo prompt è relativo al *Common Name* 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 OK

Il procedimento è simile al precedente, ma questa volta va eseguito sul *Server*.

Quindi per prima cosa si devono ripetere i passaggi fatti in sezione 3.2.1, questa volta però chiamiamo la cartella `openvpn-pki` in modo da distinguerle:

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

Andiamo a creare un file `vars`, che questa volta contiene solo le informazioni essenziali, e poi dare il comando `init-pki` (step 3 fig.3.2):

Server

code: 3.2.7

```
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* (step 4 fig.3.2).

Come *Common Name* è stato scelto `server`:

Server

code: 3.2.8

```
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 nella cartella del server OpenVPN:

Server

code: 3.2.9

```
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.

3.2.4 Firma del certificato OpenVPN dalla CA OK

Per firmare la *Certificate Signing Request* si deve copiare il file `.req` nel *Server CA*, supponiamo quindi di averlo copiato nella cartella `/tmp`.

Ora lo si deve importare nella CA e firmarlo, ciò corrisponde ai passaggi 5 e 6 fig.3.2:

Server CA

code: 3.2.10

```
1 $ cd ~/openvpn-ca
2 $ ./easymca import-req /tmp/server.req server # passaggio 5
3 $ ./easymca sign-req server server # passaggio 6
4 Using configuration from /home/ubuntu/openvpn-ca/pki/safessl-easymca.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 contiene il certificato firmato dalla CA.

Per concludere la procedura (step 7 fig.3.2), si devono copiare i file `ca.crt` e `server.crt` dal *Server CA* al *Server OpenVPN*, spostandoli nella cartella del *Server OpenVPN*: `/etc/openvpn/server`.

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

Per aumentare ulteriormente la sicurezza del nostro *Server OpenVPN* possiamo creare un'ulteriore chiave, che permette di offuscare il certificato in fase di validazione. In questo caso la chiave è di tipo *preshaed* ed è comune per utti gli utenti, serve principalmente per aggiungere un ulteriore livello di sicurezza.

La creazione va fatta sul *Server OpenVPN* e il file risultante va copiato nella cartella del server OpenVPN:

Server

code: 3.3.1

```
1 $ cd ~/openvpn-pki/
2 $ openvpn --genkey --secret ta.key
3 $ sudo cp ta.key /etc/openvpn/server
```

3.4 Generazione dei certificati per i *clients* OK

La generazione dei certificati per i client consiste in una procedura molto simile alla creazione del certificato del *Server*, sezione 3.2.3 e 3.2.4.



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

Per ospitare i certificati dei client e le loro chiavi, creiamo una cartella nella home. Successivamente verrà usata per automatizzare la creazione dei file di configurazione OpenVPN che verranno usati dai client per connettersi alla VPN.

Server

code: 3.4.1

```
1 $ mkdir -p ~/client-configs/keys
2 $ chmod -R 700 ~/client-configs
```

Creiamo quindi un certificato per un *Client*, è importante che il *Common Name* dato al certificato sia unico ed è consigliabile usare un nome che permetta di collegare il certificato con il cliente che ha richiesto il servizio. Ciò permette di conoscere quale certificato revocare in caso sia necessario.

In questo caso è stato usato il *Common Name* `client1` (step 1 fig.3.3):

Server

code: 3.4.2

```
1 $ cd ~/openvpn-pki/
2 $ ./easysrsa gen-req client1 nopass
```

Con questo comando verranno creati 2 file: `pki/private/client1.key` costituisce la chiave privata del certificato e va copiata nella directory creata in code: 3.4.1;

TODO: serve sto cp?

Server

code: 3.4.3

```
1 $ cp pki/private/client1.key ~/client-configs/keys/
```

`pki/reqs/client1.req` è la *Certificate Signing Request* che deve essere copiata nel *Server CA* per essere firmata.

Supponiamo di aver copiato il file `client1.req` nella cartella `tmp` del *Server CA*, possiamo quindi importarla e firmarla (passaggi 2 e 3 fig. 3.3):

Server CA

code: 3.4.4

```
1 $ cd ~/openvpn-ca
2 $ ./easyrsa import-req /tmp/client1.req client1      # passaggio 2
3 $ ./easyrsa sign-req client client1                 # passaggio 3
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:'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
```

Ciò produrrà il file di certificato `openvpn-ca/pki/issued/client1.crt` che deve essere copiato nel *Server OpenVpn*, supponiamo di copiarlo nella cartella `~/client-configs/keys/`.

3.4.1 Script per la creazione delle configurazioni dei client OK

Per facilitare la creazione dei file di configurazione dei client, `Common-Name.ovpn`, andremo a creare un apposito script bash che userà la cartella creata in code: 3.4.1.

Per concludere la preparazione si devono copiare i file `ta.key` (creato nella sezione 3.3) e `ca.crt` (creato nella sezione 3.2.4), sistemando inoltre l'*owner* in modo che l'utente non privilegiato vi possa accedere:

Server

code: 3.4.5

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

Ora andiamo a scaricare e personalizzare la configurazione di esempio per i client:

```

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 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

```

Questo script permette di aggiungere in modo *inline* i file necessari all'autenticazione di un client alla VPN, ha il vantaggio di poter inviare al cliente un solo file invece di 5.

Quindi, in base al *Common Name* passato come argomento, lo script userà: il certificato della CA, `ca.crt` ; il certificato e chiave relativi al *Client* per cui si sta creando la configurazione, usando il *Common Name* passato come argomento; e la *presared key*, `ta.key` .

Il tutto viene scritto in un file che ha lo stesso nome del *Common Name* del *Client* per cui si

sta creando la configurazione ma con estensione `.ovpn`.

Quindi per creare la configurazione di *client1* (passaggio 4 fig.3.3):

```
Server code: 3.4.8
1 $ ./make_config.sh client1
```

Ciò creerà il file `client-configs/files/client1.ovpn`, successivamente sarà necessario copiare questo file nella macchina del *Client*.

3.5 Creazione del file di configurazione del Server OpenVPN OK

Il server openvpn viene configurato attraverso il file di configurazione `/etc/openvpn/server/server.conf`, per non partire da una configurazione vuota si può usare la configurazione di esempio offerta da OpenVPN:

```
Server code: 3.5.1
1 $ cd /etc/openvpn/server/
2 $ sudo wget "https://raw.githubusercontent.com/OpenVPN/openvpn/\
3     master/sample/sample-config-files/server.conf"
```

A questo punto andiamo a modificare il file di esempio personalizzandolo per le nostre necessità, per facilitare la riproduzione di questa configurazione è stato aggiunto il numero della riga che è stata modificata:

```
Server code: 3.5.2
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 # questa riga va commentata
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     # stassa cosa per il gruppo
10 318 auth sha256       # selezione del metodo di autenticazione
```

3.6 Configurazioni sulla network stack del Server OK

OpenVPN per funzionare necessita che il server abbia l'*ip forwarding* abilitato. Ciò è necessario per permettere al server OpenVPN di inoltrare il traffico proveniente dalla VPN nella sua network stack, e viceversa.

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

```
Server code: 3.6.1
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 (sezione 2.2.2) è presente il firewall `firewalld`, ma per una più semplice configurazione è consigliato di disattivarlo e installare `ufw`:

```
Server code: 3.7.1
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
```

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

3.7.1 Configurazione del NAT OK

TODO: va bene corsivo per vpn e 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
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 [...]
```


In questo caso il nome dell'interfaccia di rete è `ens3`. Possiamo quindi procedere con la configurazione del NAT:

Server

code: 3.7.3

```
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 OK

Precedentemente abbiamo abilitato il forwarding nella network stack del server (sezione 3.6), 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`:

Server

code: 3.7.4

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

3.7.3 Conclusione della configurazione del firewall OK

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

Servercode: 3.7.5

```
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)
```

Se la configurazione è stata effettuata correttamente di avrà un output simile al precedente, in caso contrario si dovranno valutare gli errori dati da ufw.

3.8 Avvio del Server OpenVPN OK

Ora che la configurazione del Server è in una situazione stabile possiamo tentate di 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
        ↳ preset: disabled)
6   Active: active (running) since Mon 2022-04-18 13:08:44 CEST; 4h 22min ago
7     Docs: man:openvpn(8)
8           https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
9           https://community.openvpn.net/openvpn/wiki/HOWTO
10  Main PID: 436 (openvpn)
11    Status: "Initialization Sequence Completed"
12     Tasks: 1 (limit: 9488)
13    Memory: 4.8M
14     CPU: 199ms
15    CGroup: /system.slice/system-openvpn\x2dserver.slice/openvpn-server@server.service
16            └─436 /usr/bin/openvpn --status /run/openvpn-server/status-server.log
17            ↳ --status-version 2 --suppress-timestamps --config server.conf
18 Apr 18 13:08:44 server openvpn[436]: TUN/TAP device tun0 opened
19 Apr 18 13:08:44 server openvpn[436]: Incoming Control Channel Encryption: Cipher
        ↳ 'AES-256-CTR' initialized with 256 bit key
20 Apr 18 13:08:44 server openvpn[436]: Incoming Control Channel Encryption: Using 256 bit
        ↳ message hash 'SHA256' for HMAC authentication
21 Apr 18 13:08:44 server openvpn[436]: net_addr_v4_add: 10.8.0.1/24 dev tun0
22 Apr 18 13:08:44 server openvpn[436]: UDPv4 link local (bound): [AF_INET][undef]:1194
23 Apr 18 13:08:44 server openvpn[436]: UDPv4 link remote: [AF_UNSPEC]
24 Apr 18 13:08:44 server openvpn[436]: MULTI: multi_init called, r=256 v=256
25 Apr 18 13:08:44 server openvpn[436]: IFCONFIG POOL IPv4: base=10.8.0.2 size=253
26 Apr 18 13:08:44 server openvpn[436]: IFCONFIG POOL LIST
27 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)*.

In caso il servizio fallisse ad avviarsi si avrà un breve log con `systemctl status`, oppure si possono leggere i log completi usando `journalctl -u openvpn-server@server.service`.

3.9 Test della configurazione OK TODO: si può espandere in qualche modo?

Ora che la configurazione per questo capitolo è conclusa possiamo testare che tutto funzioni correttamente.

Per farlo ci spostiamo su una macchina *Client*, con sistema operativo Linux ad esempio. Usando la configurazione creata in code:3.4.8 possiamo connettere il *Client* alla VPN:

Clientcode: 3.9.1

```
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:

Clientcode: 3.9.2

```
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*:

Clientcode: 3.9.3

```
1 $ ping -c1 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
```

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

Servercode: 3.9.4

```
1 $ sudo tcpdump -i tun0
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
```

Si vede quindi che è possibile instaurare una comunicazione bidirezionale tra *Client*, `10.8.0.2`,

e *Server*, 10.8.0.1 .

Abbiamo quindi realizzato correttamente la topologia di obbiettivo per questo capitolo, descritta in fig.3.1b.

Capitolo 4

Configurazione del *Router*

4.1 Overview della configurazione OK

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

Continuando a seguire un approccio incrementale, ignoriamo per ora gli *host-domotici* e aggiungiamo alla topologia solo il *Router 4G*. Quindi, la configurazione di partenza sarà descritta in fig. 4.1a.

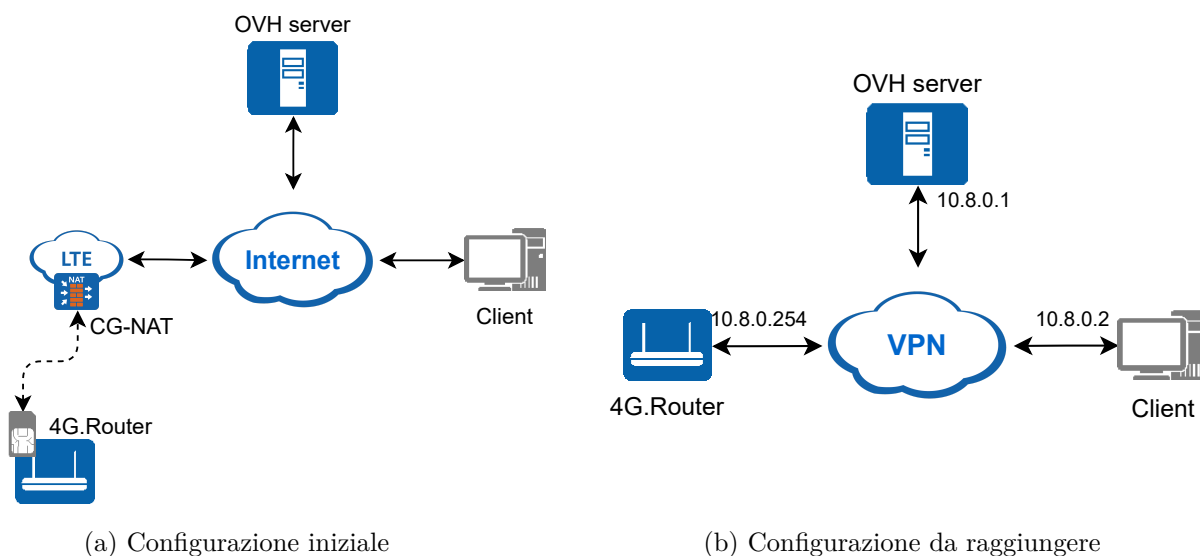


Figura 4.1: Schemi delle configurazioni iniziali e finali per il capitolo 4.

L'obiettivo di questo capitolo è di configurare il *Router 4G* e il *Server* in modo da poter instaurare un canale di comunicazione tra *Router* e *Client*, raggiungendo una topologia virtuale descritta in fig. 4.1b.

4.2 Prerequisiti per la configurazione del *Router* TODO: molto male da sistemare

La configurazione del *Router* verrà effettuata sia via shell sia via interfaccia grafica, ci si deve assicurare quindi che sia installata (sezione 1.3.1).

Di default OpenVPN non è presente, lo si può installare con:

<i>Router 4g</i>	code: 4.2.1
<pre>1 \$ opkg update 2 \$ opkg install openvpn luci-app-openvpn</pre>	

4.3 Creazione della configurazione OpenVPN per il *Router* OK

Per creare il file di configurazione OpenVPN per il *Router* si devono seguire gli step descritti in fig. 3.3. Supponiamo di usare come *Common Name* `router`, quindi dopo aver firmato il certificato e ottenuto il file `router.crt` dal *Server CA* possiamo procedere con la creazione della configurazione OpenVPN usando l'apposito script:

<i>Server</i>	code: 4.3.1
<pre>1 \$./make_config.sh router</pre>	

Dopodiché si deve copiare il file `router.ovpn` dal *Server* al *Router*, supponiamo di averlo copiato nella cartella `/configs` del *Router*.

4.4 Connessione del *Router* alla VPN OK

Possiamo quindi spostarci nel *Router* e tentare di connetterlo alla VPN:

<i>Router 4g</i>	code: 4.4.1
<pre>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</pre>	

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

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

4.5 Auto start del *Client VPN* nel *Router* OK

Per abilitare l'avvio automatico del *Client VPN* all'accensione del router si deve, per prima cosa, modificare il file `/etc/config/openvpn` in modo che faccia riferimento alla configurazione corretta:

```
Router 4g code: 4.5.1
1 $ vim /etc/config/openvpn
2 20 option config /configs/router.ovpn
```

Ora possiamo abilitarla usando luci:

The screenshot shows the OpenVPN configuration page in the LuCI web interface. At the top, there is a navigation bar with links for Status, System, Network, VPN, and Logout. The main heading is "OpenVPN". Below it, the section "OpenVPN instances" is displayed, with a note stating "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	start	1194	udp	Edit Delete
sample_server	<input type="checkbox"/>	no	start	1194	udp	Edit Delete
sample_client	<input type="checkbox"/>	no	start	-	udp	Edit Delete

Below the table, there is a section titled "Template based configuration" with a form containing an "Instance name" input field, a "Select template ..." dropdown menu, and an "Add" button.

Next is the "OVPN configuration file upload" section, which also has an "Instance name" input field, a "Browse..." button, and a "No file selected." message, followed by an "Upload" button.

At the bottom of the page, there are three buttons: "Save & Apply", "Save", and "Reset".

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.

4.6 Abilitazione del Client-to-Client nel server OpenVPN *TODO:* da rileggere

In questo momento i client della VPN, `client1` e `router`, possono comunicare tra loro. Ma lo fanno passando per la network stack del *Server*, possiamo verificarlo con le utility `ping` e `tcpdump`.

Eseguiamo ad esempio un test in cui:

- `router` effettua il `ping` verso `client1`
- `server` è in ascolto sull'interfaccia `tun0` usando `tcpdump`

```
Router 4g code: 4.6.1
1 $ ping -c1 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
```

Il `ping` ha successo, quindi è possibile una comunicazione bidirezionale tra `router` e `client1`.

Dal server possiamo vedere un'analisi dei pacchetti che passano attraverso l'interfaccia `tun0` usando la utility `tcpdump`:

```
Server code: 4.6.2
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
```

Si vede che ogni richiesta viene duplicata, la prima è in entrata sulla network stack del *Server* e la seconda in uscita. Ciò perchè il forwarding del pacchetto viene effettuato dal layer IP del *Server*, e dato che `tcpdump` è in ascolto sull'interfaccia `tun0`, il pacchetto viene letto 2 volte. Si può vedere una rappresentazione del percorso del pacchetto nella network stack del *Server* in fig. 4.3a, si vede come il pacchetto passa 2 volte per l'interfaccia `tun0`, punto 2 e 5.

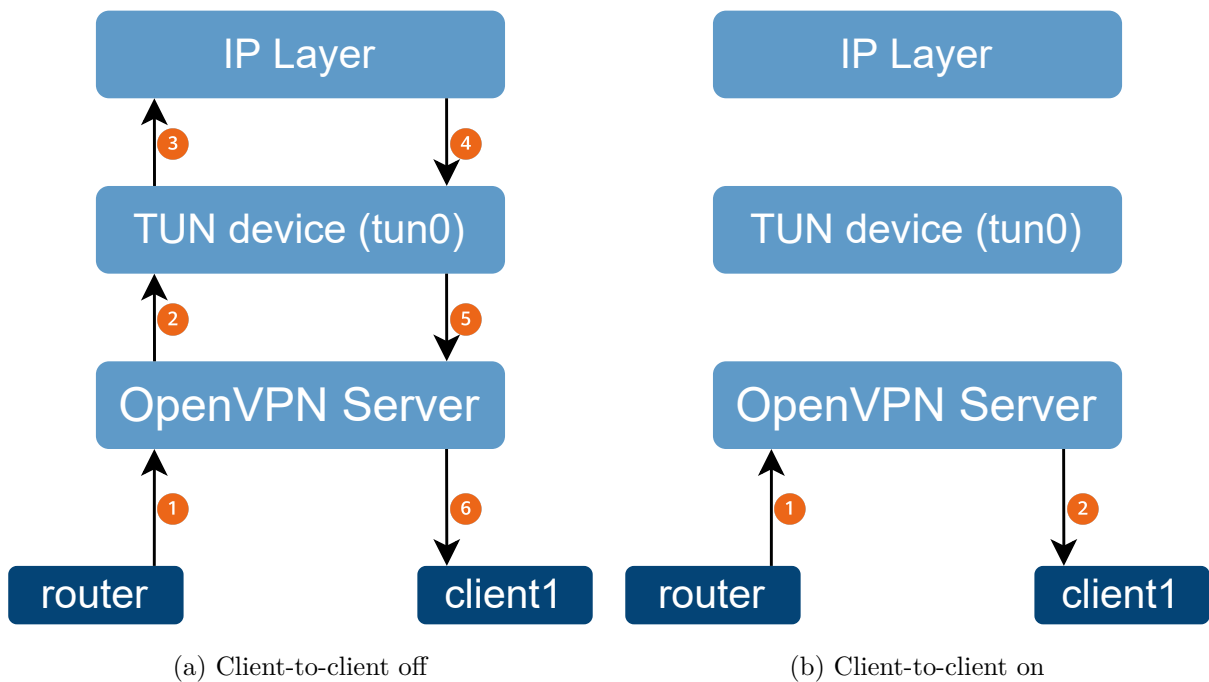


Figura 4.3: Effetto del `client-to-client` sulla network stack del *Server* [15].

Per evitare questo traffico possiamo abilitare l'opzione `client-to-client` nel *Server OpenVPN*. In questo modo il layer OpenVPN effettuerà direttamente il forwarding tra i client della VPN. Graficamente si vede in fig. 4.3b come il pacchetto viene subito inoltrato e non entra affatto nella network stack del server. Ciò comporta, come effetto collaterale, che i pacchetti tra client della VPN non sono soggetti alle regole di firewall imposte nel *Server*.

Andiamo quindi a modificare la configurazione del *Server OpenVPN*:

```

Server code: 4.6.3
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: 4.6.4
1 $ ping -c1 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=64 time=0.351 ms

```

Ma questa volta la network stack del *Server* non vede nessun pacchetto, dato che i pacchetti non passano attraverso l'interfaccia `tun0` (fig. 4.3b):

```

Server code: 4.6.5
1 $ sudo tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes

```

4.7 Assegnazione IP statico al *Router* OK

Dato che non possiamo sapere a priori quanti client si connetteranno contemporaneamente alla VPN, e l'assegnazione degli IP da parte del *Server OpenVPN* è dinamico, per conoscere sempre quale IP viene assegnato al *Router* è necessario assegnargliene uno statico attraverso un'opportuna configurazione.

Questo viene fatto usando la `client-config-dir`, che consiste in una cartella dove l'amministratore può creare dei file di configurazione specifici per determinati utenti. Ciò è utile in caso alcune configurazioni non debbano essere applicate a tutti gli utenti, ma solo a uno. Il nome del file nella cartella deve essere lo stesso del *Common Name* del certificato del client per cui si sta creando la configurazione.

Nel nostro caso siamo interessati a impostare un IP statico per il *Router*, quindi si deve creare nella cartella `ccd` un file chiamato `router`.

L'opzione che permette di impostare un IP statico attraverso la `client-config-dir` è `ifconfig-push` [10]:

Server	code: 4.7.1
1	<code>\$ sudo mkdir /etc/openvpn/server/ccd</code> # creo la client-config-dir
2	<code>\$ sudo vim /etc/openvpn/server/ccd/router</code>
3	<code>ifconfig-push 10.8.0.254 255.255.255.0</code> # impongo l'ip per questo common name
4	<code>\$ sudo vim /etc/openvpn/server/server.conf</code> # abilito l'opzione nella config del server
5	<code>167 client-config-dir ccd</code>
6	<code>\$ sudo systemctl restart openvpn-server@server.service</code>

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 fare ciò si dovrà configurare il *Router* in modo che effettui il forwarding verso la VPN, e il *Server* in modo che annunci la sottorete del *Router* a tutti gli Host della VPN.

TODO: non sta nel capitolo 2 sta cosa? va bene ripetere?

Per simulare un host domotico è stata usata una *raspberry pi*, ciò ci consente di poter accedere alla sua shell per effettuare i vari test necessari per confermare che la topologia funzioni correttamente.

In fig. 5.1a possiamo vedere la configurazione iniziale per questo capitolo, cioè la configurazione finale del capitolo 4 con l'aggiunta dell'*host domotico* connesso al *Router*.

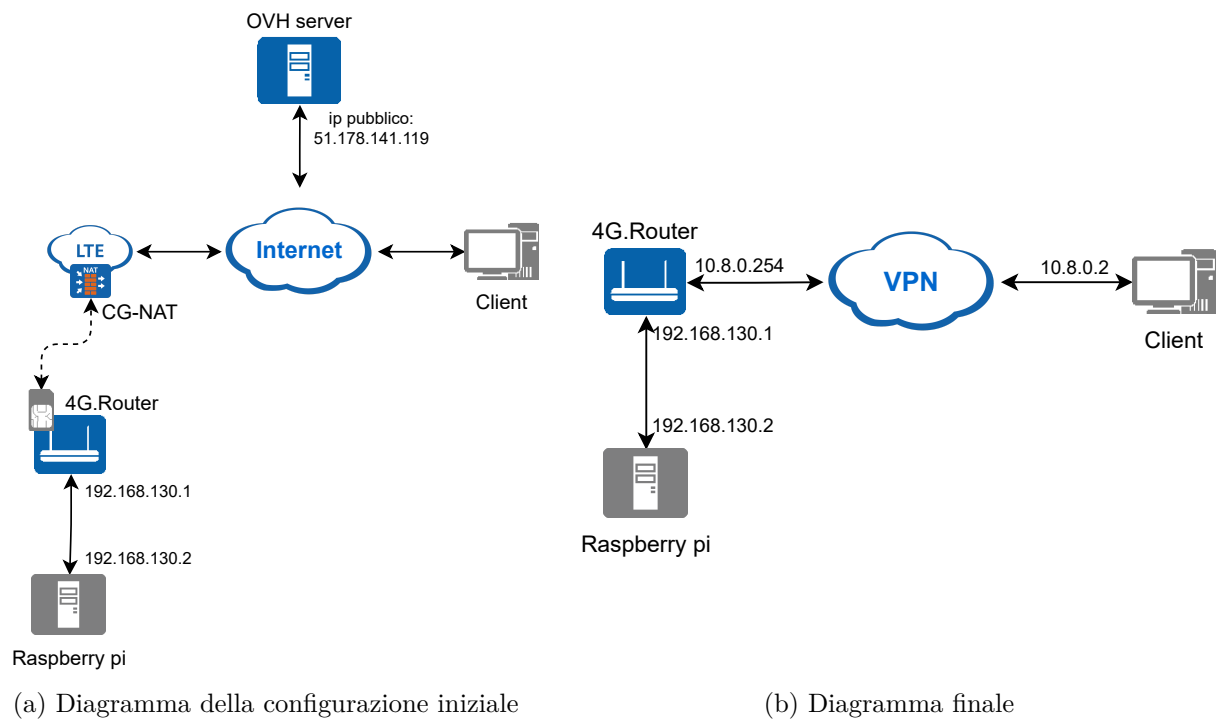


Figura 5.1: Schemi concettuali della configurazione iniziale e finale per il capitolo 5

Una volta conclusa la configurazione si arriverà a una topologia virtuale descritta in fig. 5.1b, che costituisce inoltre la topologia di obbiettivo per questo elaborato (fig. 2.3).

5.2 Creazione della zona firewall per la VPN

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

Ciò viene fatto direttamente da LuCI, nella sezione firewall si avranno preconfigurate delle zone firewall, in questo caso sono presenti di default 2 zone: *lan* e *wan*:

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

Output

Forward

Zones

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

Add

Save & Apply
Save
Reset

Figura 5.2: Configurazione di default delle zone del firewall.

Le 2 zone sono collegate alle relative interfacce: la zona *lan* è collegata all'interfaccia relativa alla rete interna del *Router*; la zona *wan* è collegata all'interfaccia esterna del *Router*.

Andiamo quindi ad aggiungere una nuova zona, chiamata *vpn*, che successivamente sarà collegata all'interfaccia virtuale della VPN (*tun0*). La zona *vpn* dovrà avere le seguenti opzioni:

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

Possiamo vedere la pagina di configurazione con le opzioni inserite:

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: Schermata di aggiunta di una nuova zona 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

Output

Forward

Zones

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

Add

Save & Apply
Save
Reset

Figura 5.4: Configurazione delle zone firewall dopo l'aggiunta della zona *vpn*.

5.3 Aggiunta dell'interfaccia *tun0* alla zona firewall *vpn*

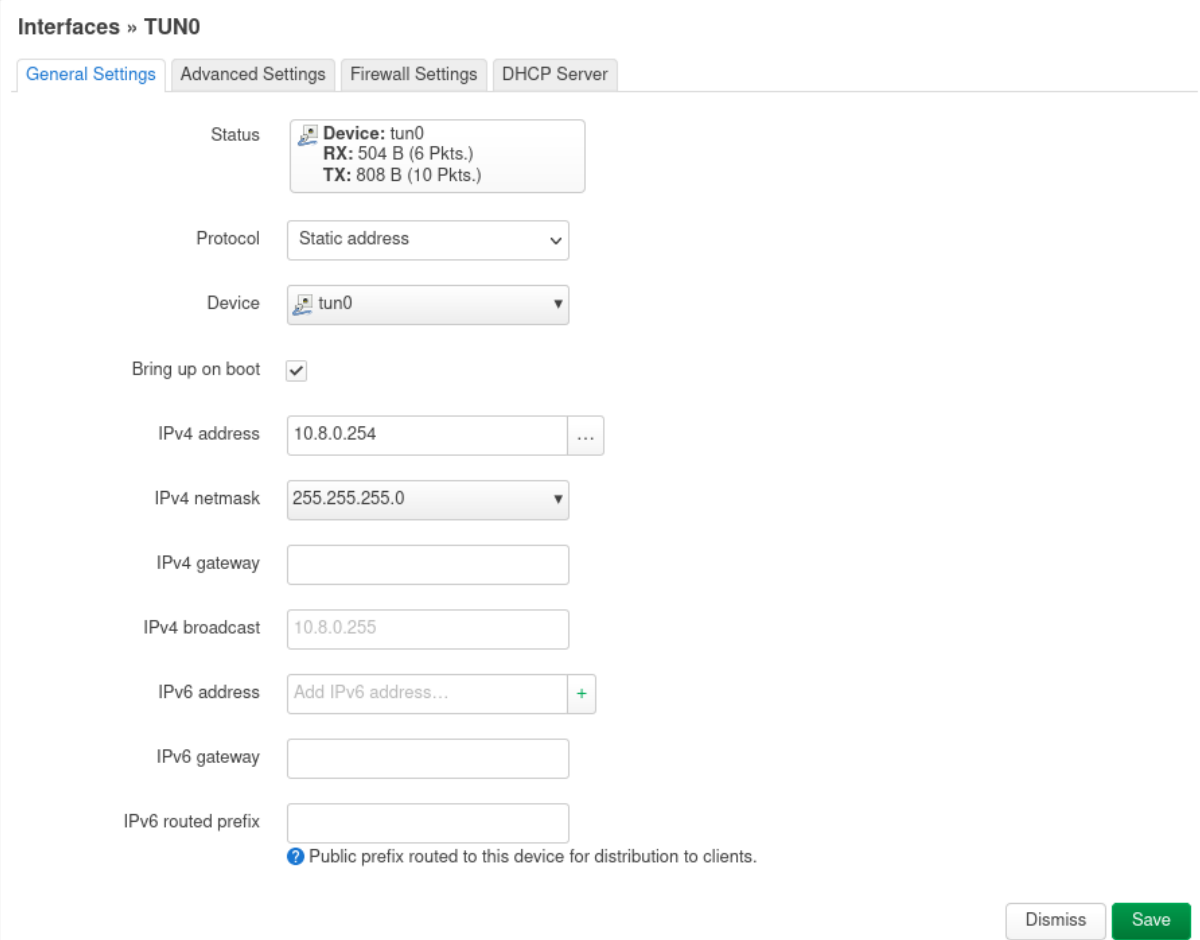
Per far effettivamente funzionare la zona firewall aggiunta nella sezione precedente, si deve aggiungere l'interfaccia relativa alla VPN (*tun0*) alla zona *vpn*.

Dato che l'interfaccia *tun0* non è presente in LuCI, la si deve creare usando le opzioni che già conosciamo. Durante la creazione, nella sezione *Firewall Settings*, sarà possibile assegnare l'interfaccia a una zona firewall.

Quindi nella sezione *interfaces*, si deve 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... +

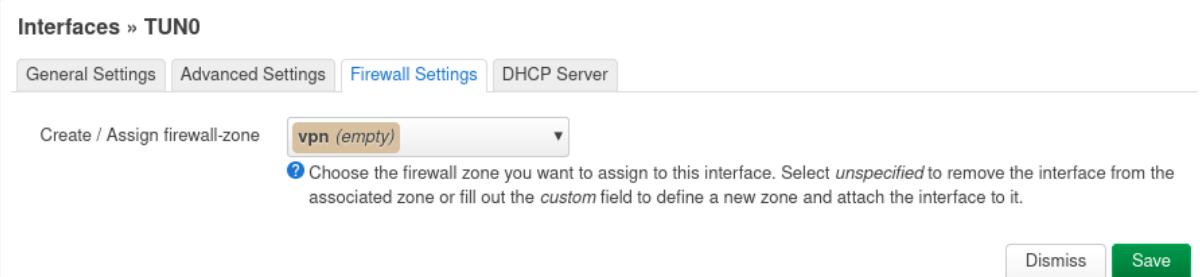
IPv6 gateway

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

A questo punto i pacchetti provenienti dalla lan del *Router* vengono inoltrati correttamente nella VPN, ma i device nella VPN non hanno una rotta verso la rete interna del router. Quindi in questo momento la comunicazione è monodirezionale.

Possiamo vederlo praticamente con un test:

- l'host domotico effettua il ping verso il client
- il *Router* è in ascolto sull'interfaccia *tun0* con `tcpdump`

Host-domotico

code: 5.3.1

```
1 $ ping -c1 10.8.0.2      # client 1
2 PING 10.8.0.2 (10.8.0.2) 56(84) bytes of data.
3
4 --- 10.8.0.2 ping statistics ---
5 1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

Router

code: 5.3.2

```
1 $ tcpdump -i tun0
2 15:33:38.363082 IP 192.168.130.2 > 10.8.0.2: ICMP echo request, id 17, seq 1, length 64
```

Si vede come il pacchetto ICMP *echo request* venga correttamente inoltrato nella VPN dal *Router*, ma il `ping` fallisce poiché che non vi è nessun *echo reply* dal *Client*. Questi problemi verranno risolti prossima sezione.

5.4 Modifiche alla configurazione OpenVPN del *Server*

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

1. il *Server OpenVPN* sia consapevole che il *Router* vuole esporre una sua sottorete verso la VPN
2. i client della VPN abbiano l'opportuna rotta per raggiungere la sottorete dove si trova l'*host-domotico*

Per il punto 1 è necessario aggiungere l'opzione `iroute` nel file `/etc/openvpn/server/ccd/router`, creato in sezione 4.7:

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      # net e netmask della rete lan del router
```

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* è necessario 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 e analisi della configurazione [work-in-progress](#)

Ora che la configurazione della topologia è conclusa, raggiungendo l'obiettivo posto in sezione [2.1](#), dobbiamo testarla approfonditamente per confermare che tutto funzioni come previsto.

TODO: da rivedere

Nei vari test verranno usate le utility `ping`, `tracert` e `tcpdump`, poste in vari punti della topologia, in modo da valutare il percorso reale e virtuale dei pacchetti.

Supponiamo che il *Client* voglia comunicare con un *host-domotico*, in questo caso la *raspberrypi* che ha IP `192.168.130.2`, e che il dato da trasmettere sia un ICMP *echo request*, facente parte di un `ping`.

Quindi il *Client* fa partire il ping:

*Client*code: 5.5.1

```
1 $ ping -c 1 192.168.130.2
```

Supponiamo di poter seguire il percorso del pacchetto analizzando gli step principali, in questo caso consideriamo 6 step:

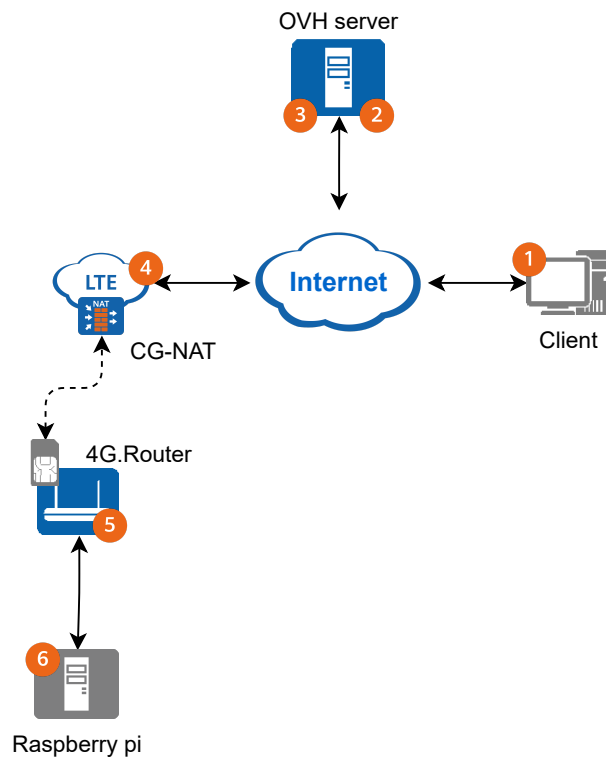


Figura 5.6: Diagramma di test realizzato con il *Router* e il *Client*

TODO: da rileggere mooolto bene

1. Il *Client* (10.8.0.2) effettua il ping verso l'*host-domotico* (192.168.130.2). Dato che il *Client* ha la subnet 192.168.130.0/24 nella sua tabella di routing (sezione 5.4) viene costruito un pacchetto di tipo ICMP *echo request* che ha come IP sorgente 10.8.0.2 e come destinazione 192.168.130.2 . A questo punto il *Client OpenVPN* cifra e incapsula il pacchetto in un ulteriore pacchetto IP (sezione 1.2.2), che questa volta ha come IP di destinazione 51.178.141.119 , cioè l'ip pubblico del *Server*. Il pacchetto può essere quindi trasmesso attraverso internet, esce dal *Client* e raggiunge il *Server*.
2. Il *Server* riceve il pacchetto, vedendo che si tratta di un pacchetto di tipo OpenVPN lo inoltra al *Server OpenVPN* (sezione 3.6 e 3.7). Il contenuto del pacchetto viene quindi decifrato e analizzato, il *Server OpenVPN* vede che il pacchetto è destinato nella subnet 192.168.130.0/24 , data la configurazione fatta in sezione 5.4 il *Server* sa che deve inoltrarlo al *Router* (10.8.0.254).
3. Viene costruito un pacchetto IP destinato al NAT, che ha al suo interno il pacchetto cifrato OpenVPN costruito al punto 2. La comunicazione attraverso il NAT è possibile poichè OpenVPN implementa nel suo protocollo una serie di pacchetti *keepalive* tra ogni host e il server. Ciò permette al nat di sapere a quale host della sua rete privata deve inoltrare i pacchetti.
4. Il NAT effettua la traduzione, modificando l'IP e la porta del pacchetto esterno e lo inoltra all'interfaccia *wan* del *Router*.

5. Il *Router* riceve il pacchetto e lo decifra. Vedendo che il pacchetto è destinato alla subnet `192.168.130.0/24` lo inoltra alla sua *lan* (sezione 5.2 e 5.3). Il pacchetto interno viene quindi inoltrato verso l'*host-domotico* (`192.168.130.2`).
6. Il *host-domotico* riceve il pacchetto e legge il contenuto, vedendo che si tratta di ICMP *echo request* provvede a costruire una risposta ICMP *echo reply* e la invia al *Client*, invertendo sorgente e destinazione del pacchetto che aveva ricevuto.

TODO: da qual in poi è un disastro!

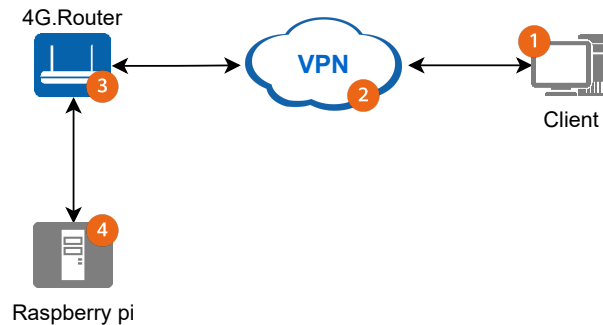


Figura 5.7: Diagramma di test virtuale

Vediamo quindi un ping dal *Client* verso l'*host-domotico*, con *Server* e *Router* in ascolto sull'interfaccia tun0:

Client	code: 5.5.2
<pre> 1 \$ ping -c 1 192.168.130.2 2 PING 192.168.130.2 (192.168.130.2) 56(84) bytes of data. 3 64 bytes from 192.168.130.2: icmp_seq=1 ttl=63 time=0.643 ms 4 5 --- 192.168.130.2 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 </pre>	

Server	code: 5.5.3
<pre> 1 \$ tcpdump -i tun0 2 listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes </pre>	

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: 5.5.4
<pre> 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 > 192.168.130.2: ICMP echo request, id 12, seq 1, length 64 4 10:44:07.228550 IP 192.168.130.2 > 10.8.0.2: ICMP echo reply, id 12, seq 1, length 64 </pre>	

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

Provando nella direzione inversa si ottiene lo stesso risultato

```
Host-domotico code: 5.5.5
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: 5.5.6
1 $ tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
3 11:56:10.469901 IP 192.168.130.2 > 10.8.0.2: ICMP echo request, id 14, seq 1, length 64
4 11:56:10.470230 IP 10.8.0.2 > 192.168.130.2: ICMP echo reply, id 14, seq 1, length 64
```

Possiamo inoltre usare `tracpath` per vedere gli hop:

```
Client code: 5.5.7
1 $ tracpath 192.168.130.2
2 1?: [LOCALHOST] pmtu 1500
3 1: 10.8.0.254 0.471ms
4 1: 10.8.0.254 0.473ms
5 2: 192.168.130.2 0.567ms reached
6 Resume: pmtu 1500 hops 2 back 2
```

```
host-domotico code: 5.5.8
1 $ tracpath 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 work-in-progress

L'architettura proposta in questo elaborato è stata effettivamente implementata su apparati reali durante il tirocinio in collaborazione con l'azienda Esse-ti. Per rendere più completo il prodotto che vogliono vendere è necessario modificare la configurazione per consentire che più clienti, ognuno con il proprio Router 4G, possano usufruire della VPN. Si deve però stare attenti a mantenere l'isolamento tra i clienti, per questo motivo è necessario garantire che un cliente assegnato a una determinata VPN non possa connettersi a quelle degli altri.

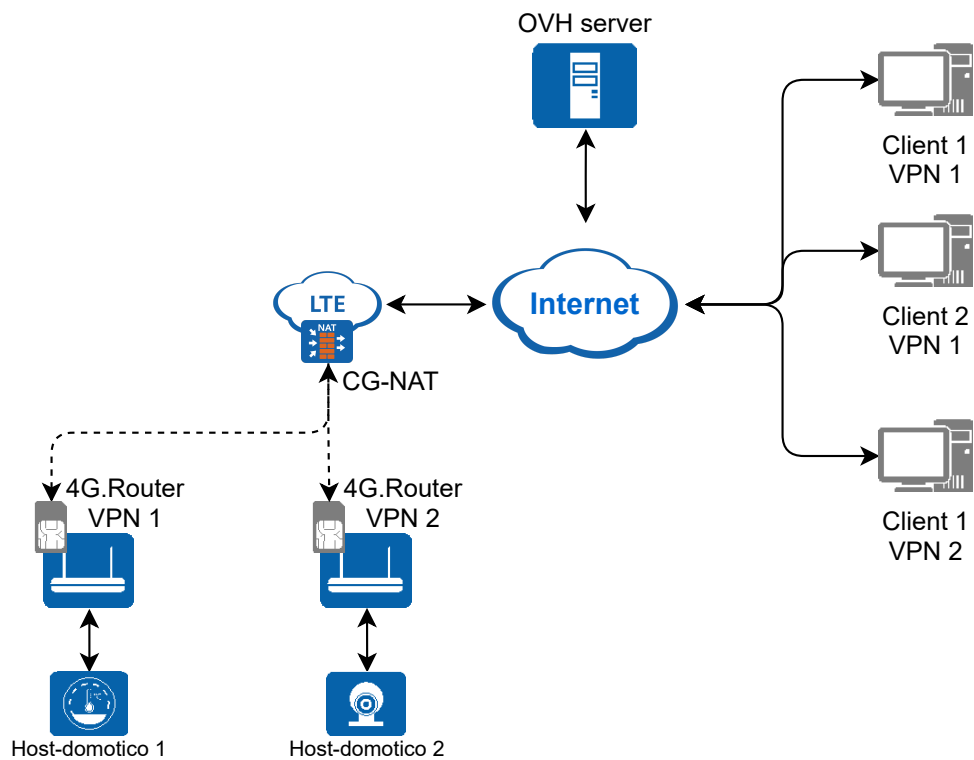


Figura 6.1: multi-istanza real

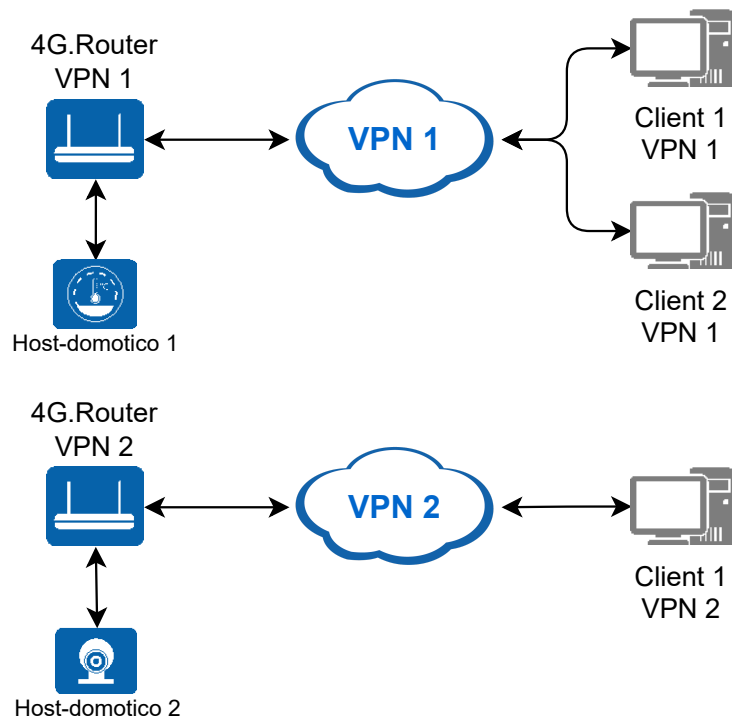


Figura 6.2: multi-istanza virtual

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] L. M. M. (<https://serverfault.com/users/176217/lasse-michael-m%c3%b8lgaard>). openvpn - ifconfig parameter. Server Fault Stack Exchange. URL:<https://serverfault.com/a/1000725/558773> (version: 2020-01-18).
- [11] Wikipedia. Carrier-grade NAT — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Carrier-grade%20NAT&oldid=1080669652>, 2022. [Online; accessed 12-June-2022].
- [12] 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].
- [13] 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].
- [14] 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].

- [15] 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).