Virtual Private Network - VPN

Network Infrastructures Lab Sessions

SAPIENZA
UNIVERSITÀ DI ROMA

Instructors:
- Pietro Spadaccino

Today

- VPN introduction
- TLS & PKI
- OpenVPN

Further readings:

- Virtual Private Networks; Charlie Scott et al.; O'Reilly

Private Network

A **private network** is composed of computers owned by a single organization that share information specifically with each other. They're assured that they are going to be the only ones using the network, and that information sent between them will (at worst) only be seen by others in the group.

The typical corporate Local Area Network (LAN) is an example of a private network. The line between a private and public network has always been drawn at the gateway router, where a company will erect a firewall to keep intruders from the public network out of their private network, or to keep their own internal users from perusing the public network

Private Network Issues

However issues may arise with LANs. For example:

- What happens if the organization has multiple geographical locations?
- What happens if you are in a mission abroad and want to access the network?

We have two possible solutions:

1. DON'T USE the public internet, use instead a dedicated secure connection
    - Of course not feasible
2. USE the public internet to connect to the network
    - Security? Managing of IP addresses?

Virtual Private Network (VPN)

Virtual Private Network (VPN)

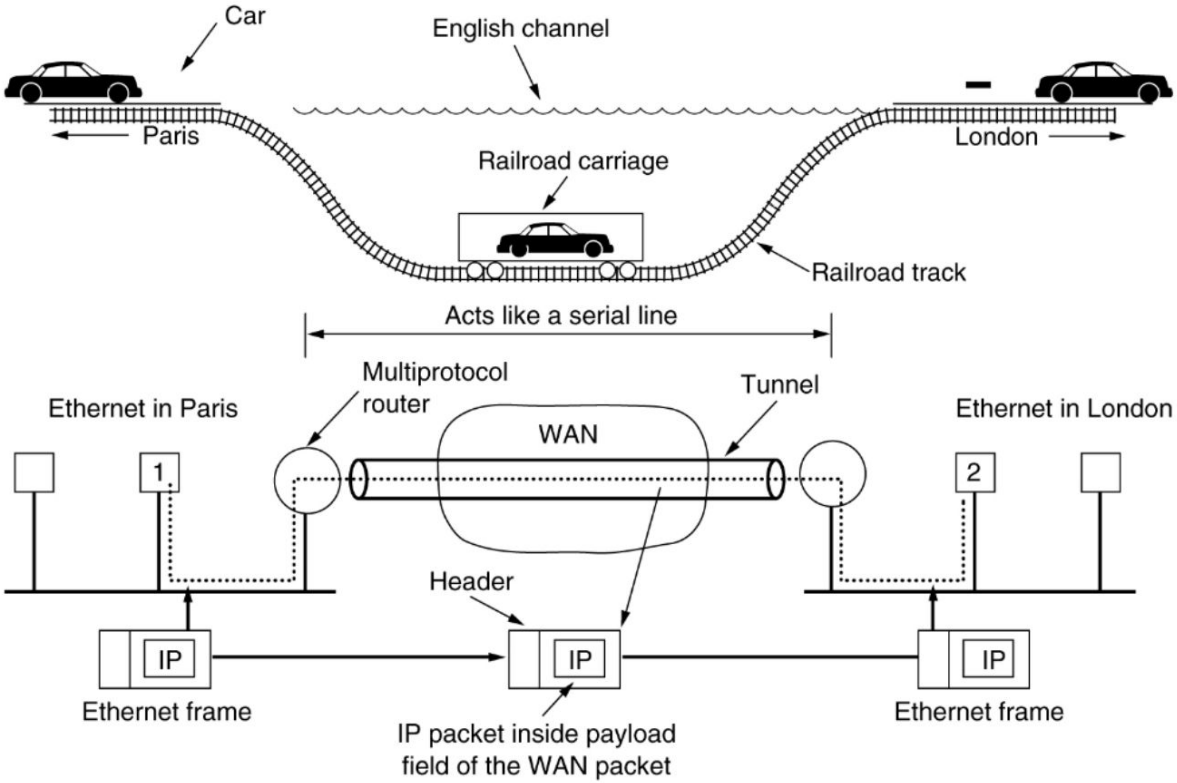
VPNs allow you to create a secure, private network over a public network such as the Internet. This is done through **encryption**, **authentication**, **packet tunneling** and **firewalls**.

Since the Internet is a public network, you always risk having someone access any system you connect to it.

Also VPNs have issues:

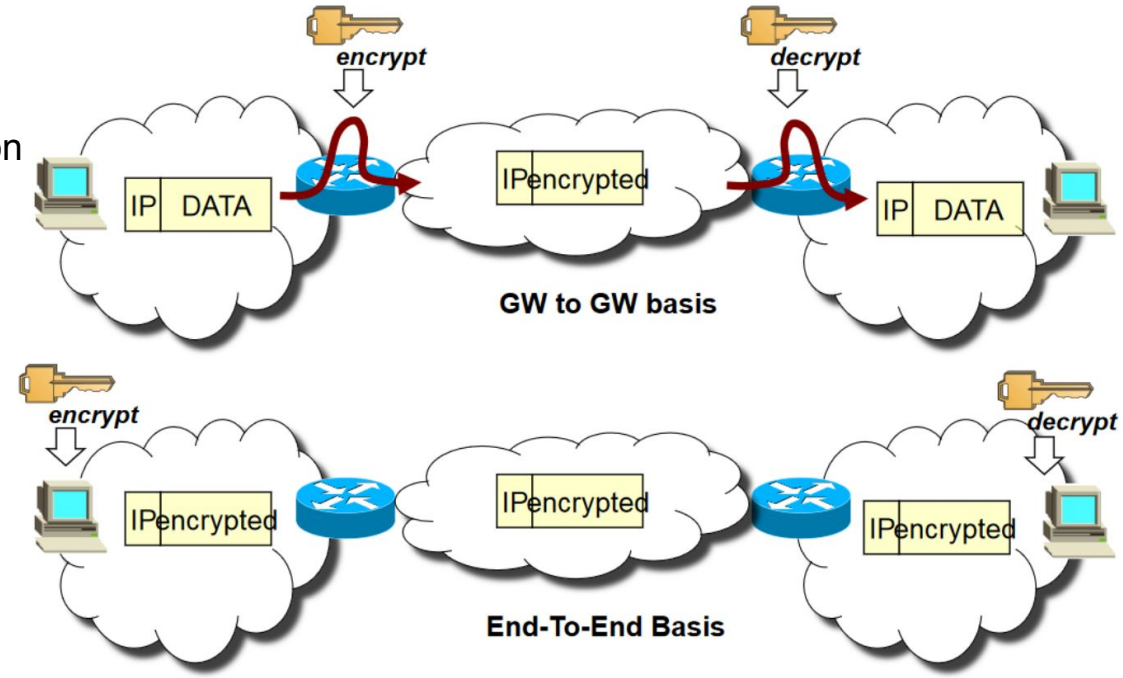
- Reliability concerns: by using the public internet you may pass through several different networks of varying speeds, reliability, and utilization, each run by a different company.
- Security: VPNs should not be used as a single mechanism of defense

Tunnels


Car
English channel
Paris
London
Railroad carriage
Railroad track
Acts like a serial line
Multiprotocol router
Tunnel
Ethernet in Paris
WAN
Ethernet in London
1
2
Header
IP
IP
IP
Ethernet frame
Ethernet frame
IP packet inside payload field of the WAN packet

VPN Modes

VPN clients on the gateways



VPN clients on the clients

Examples of VPN configuration.

Depending on the configuration, you can adopt a GW to GW, End to End or a mixture of the two

OpenVPN

We will use OpenVPN as VPN implementation on Linux. Some features are:

- Based on **TLS** (same transport layer as **HTTPS** and many more)
- Tunnel any IP subnetwork or virtual ethernet adapter over a single UDP or TCP connection
- Use any cipher, key size, or HMAC digest (for datagram integrity checking) supported by the OpenSSL library
- Choose between static-key based conventional encryption or certificate-based public key encryption
- Tunnel networks over NAT
- Use static, pre-shared keys or TLS-based dynamic key exchange

Public Key Infrastructure (PKI)

A PKI consists of the protocols, the policies and the cryptography mechanism used to manage public key certificate. A PKI requires the definition of:

- Certificate format
- Relationship among CAs
- Mechanisms and policies for issuing and revoking certificate
- Storage services

Public Key Certificate

A public key **certificate** is a data structure that binds a public key <PUB KEY> to the the identity *id* of the legitimate owner:

$$\text{CERT}_{id} \longleftarrow \langle id, \text{PUB KEY } id \rangle$$

- In reality certificates hold also other informations, such as common name (CN), date of expiration, …

The binding between $\langle id, \text{PUB KEY } id \rangle$ is granted by a trusted **Certification Authority** (**CA**) that **signs** CERT_{id}.

- This signature is contained into the certificate CERT_{id}

Provided that we have the CA's *public* key, we can verify the CA signature and therefore verify the public key authenticity

TLS certificates - chain of trust

A client has its certificate issued (signed) by a CA *x*.

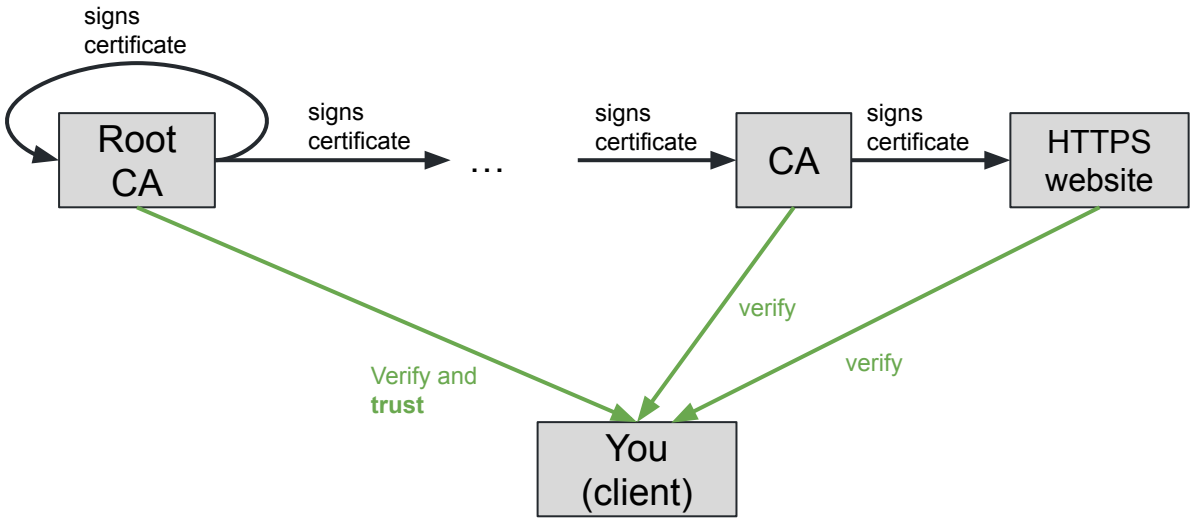
What ensures me that about the identity of the CA *x*?

- The **certificate** of *x*, issued by **another CA** *y.*

This creates a chain of signatures, also known as **chain of trust**

- The CA that “starts” the chain is called **Root CA**
- A Root CA **self-signs** its own certificate

TLS certificates - verifying the chain of trust



Key concept 1: clients have to trust root CAs

Key concept 2: signature **verification** can be done **by the client** just with the certificates

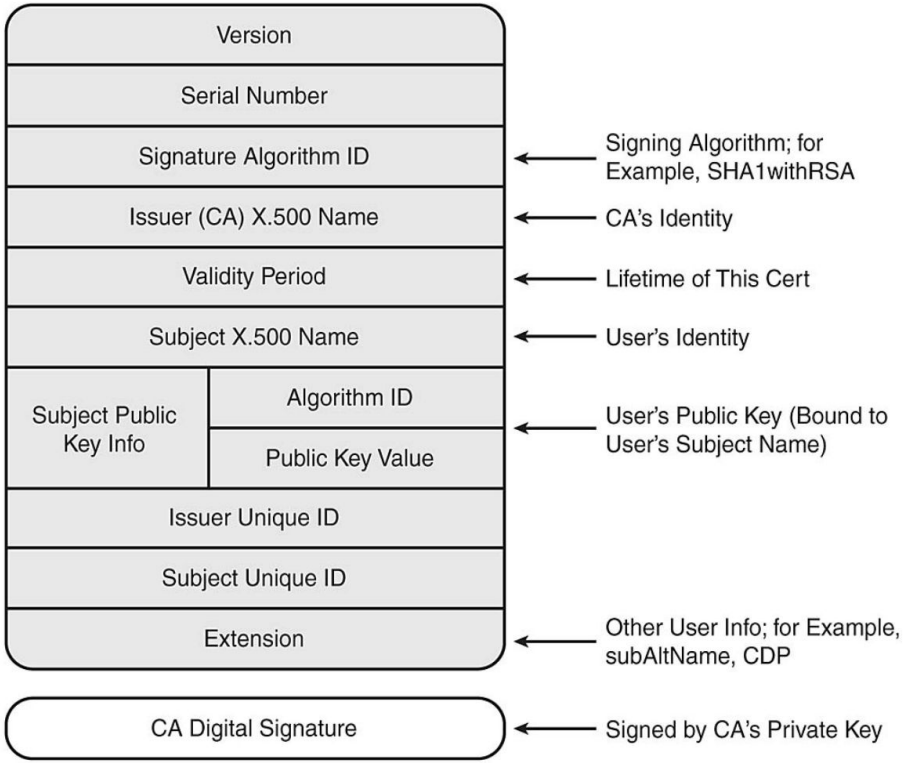
- There is **no need** to use the **private key** to verify the signature
  - The servers holding the private key are **not required** to perform **anything** during verification, they may also be **disconnected** from the internet

X.509 certificate format

The x509 standard is used in TLS, which is the security transport layer on top of which is based OpenVPN, HTTPS and many more

To inspect a x509 certificate using OpenSSL:

```
openssl x509 -in cert -inform [der, pem] -text
```



OpenVPN Security Model

The first step in building an OpenVPN configuration is to establish a PKI which consists of:

- a master Certificate Authority (CA) certificate and key which is used to sign each of the server and client certificates
- a separate certificate and private key for the server and each client

Authentication: both server and client will verify that the presented certificate was signed by the master certificate authority (CA). Then they may also check for fields in the header such as cert. type (client or server), common name, etc.

OpenVPN requires bidirectional authentication: the client is authenticated to the server and the server is authenticated to the client

OpenVPN Security Model

This security model has several desirable features from the VPN perspective:

- The server only needs its **own** certificate/key -- it doesn't need to know the individual certificates of every client which might possibly connect to it.
- The server can perform CA signature verification **without accessing** to the CA private key itself. The CA key (the most sensitive key in the entire PKI) may reside on a completely different machine (even one without a network connection).
- If a private key is compromised, it can be **disabled** by adding its certificate to a CRL (certificate revocation list). The CRL allows compromised certificates to be selectively rejected without requiring that the entire PKI be rebuilt.

Building PKI - Certification Authority

We will use the `easy-rsa` library which is bundled with OpenVPN (they are wrappers on the `openssl` library)

- In Kathara `cd` into `/usr/share/easy-rsa`
- Initialize the PKI with:

```
$ ./easyrsa init-pki        #initialized PKI folder structures
$ ./easyrsa build-ca        #build CA
```

The last command will build the CA certificate and key with interactive openssl commands. Most of the parameters can be left as default. You may enter the Common Name (CN) parameter

Building PKI - Server and client

Generate a certificate and private key for the server

```
$ ./easyrsa build-server-full SERVER_NAME
```

Generate client certificate and keys

```
$ ./easyrsa build-client-full CLIENT_NAME
```

Generate Diffie Hellman parameters for the OpenVPN server (can take some time)

```
$ openssl dhparam -out dh2048.pem 2048
```

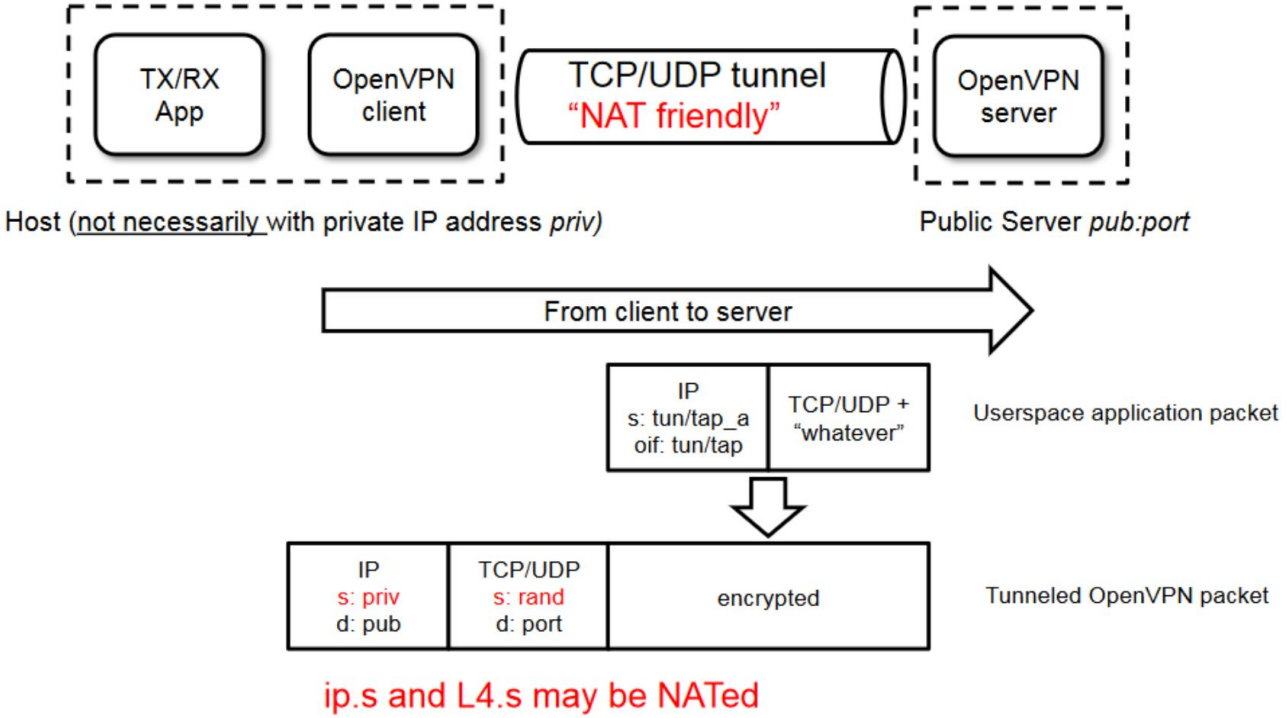
- Alternatively one can use the sample dh parameters located in `/usr/share/doc/openvpn/examples/sample-keys/dh2048.pem`

Building PKI - Files generated

Inside `/usr/share/easy-rsa/pki` folder are stored the generated certificates and keys (the client and server files are named as their Common Name):

Filename	Needed By	Purpose	Secret
ca.crt	server + all clients	Root CA certificate	NO
ca.key	key signing machine only	Root CA key	YES
dh{n}.pem	server only	Diffie Hellman parameters	NO
server.crt	server only	Server Certificate	NO
server.key	server only	Server Key	YES
client1.crt	client1 only	Client1 Certificate	NO
client1.key	client1 only	Client1 Key	YES

OpenVPN Architecture

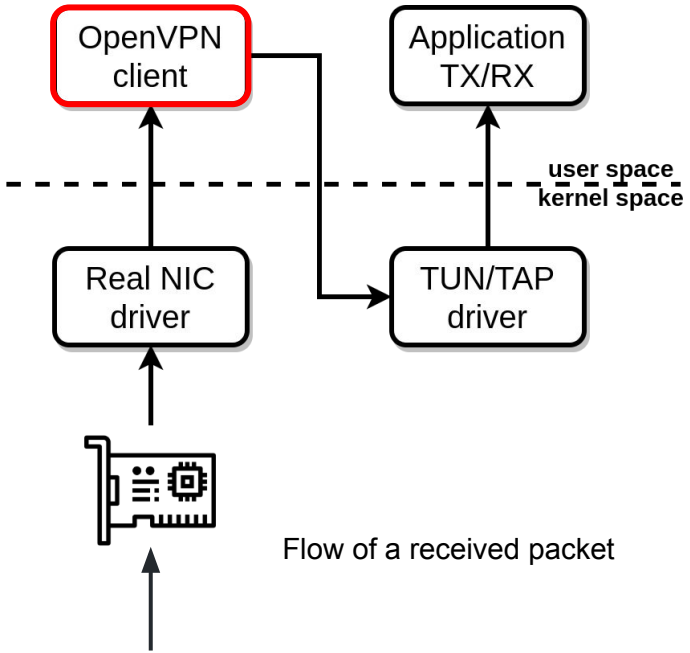


OpenVPN Architecture

OpenVPN creates a virtual network interface on which runs the connection to the VPN

- **TUN** is a virtual network iface for IP tunnelling (layer 3)
- **TAP** is a virtual network iface for Ethernet tunnelling, a.k.a. bridging (layer 2)

The user application can send IP or Ethernet frames to tun/tap interfaces. The driver will give the frames to the OpenVPN client. The client will ultimately encapsulate the packets and send them using the physical network device



TUN / TAP Interfaces

TUN is a virtual network iface for IP tunnelling (layer 3)

- Easier to set up with respect to TAP interfaces
- Less data overhead w.r.t. TAP interfaces

TAP is a virtual network iface for Ethernet tunnelling, a.k.a. bridging (layer 2)

- Your host behaves as it was directly attached via Ethernet to the LAN. It can use non-IP protocols e.g. ARP

We will only cover OpenVPN with TUN interfaces.

Client and Server configuration

Example of client and server configuration files are available in the shared drive, commenting many available options

To launch an OpenVPN instance use:

```
$ openvpn CONF_FILE
```

Where `CONF_FILE` is the configuration file.

Sample client and server configuration files are provided in the drive, and are available here and here.

In Kathara you can place the previous command in a .startup file. Also, on this current image we use for the course, we have to manually create the tun interface (see the .startup files of the vpn laboratories)

Caveat:

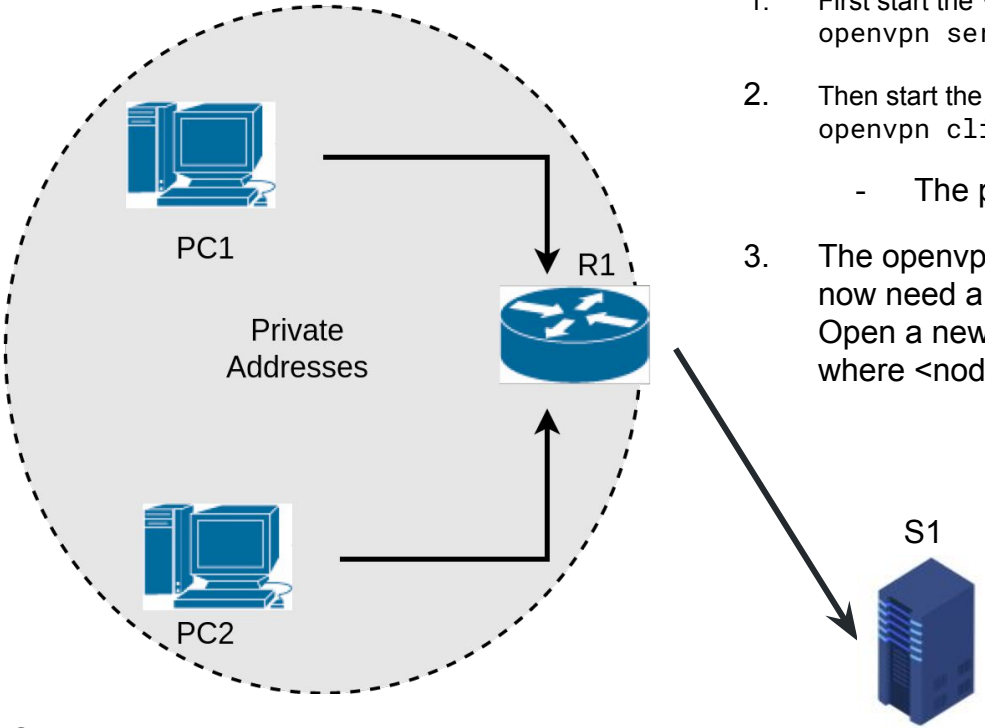
Due to kathara limitations, when you are using OpenVPN insert these lines into the startup files:

```
mkdir -p /dev/net

mknod /dev/net/tun c 10 200

chmod 600 /dev/net/tun
```

lab_vpn



Consider the README included in the lab

To start the laboratory

1. First start the VPN server. On s1 go into `/root` and:
   `openvpn server.conf`
2. Then start the VPN clients. On pc1, pc2 go into `/root` and:
   `openvpn client.conf`
   - The password of the all keys is “password”
3. The openvpn command should be kept running. You now need a new terminal to type commands.
   Open a new terminal with `kathara connect <node>`
   where <node> is pc1, pc2 or s1

Push routes and client config - lab_vpn_pushroute

Up until now we have seen **end-to-end** OpenVPN configurations: you are connecting two nodes running an OpenVPN instance.

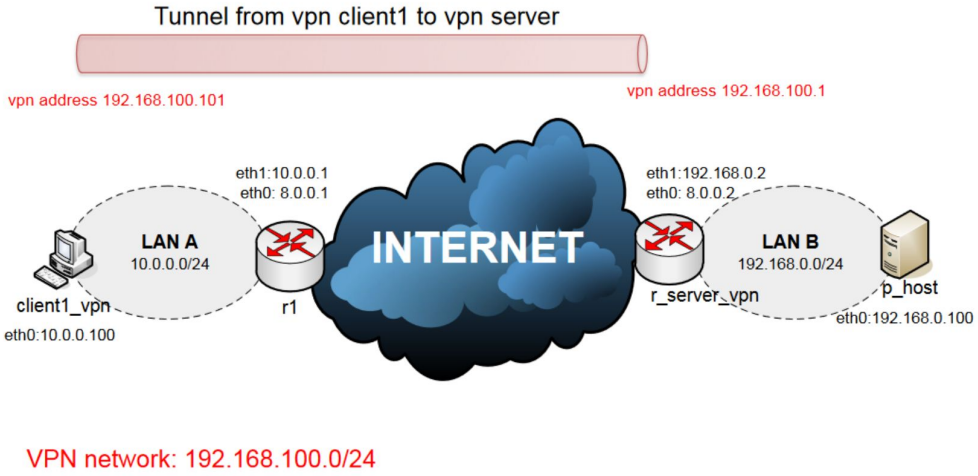
Now, you can configure also an OpenVPN server running on a gateway. This enables a VPN client to connect to a host inside private network of the server, without requiring the host itself to run an OpenVPN instance

Push routes and client config - lab_vpn_pushroute

OpenVPN is running only on `client1_vpn` and `r_server_vpn`
The node `p_host`:

1. does not run a VPN client
2. does not know about the VPN running on r_server_vpn.
3. Cannot reach `client1_vpn` through the LAN A address.

Nonetheless, it could be accessed by client1 **through** the VPN



Push routes and client config - lab_vpn_pushroute

In the server configuration, we added the following line

```
push "route 192.168.0.0 255.255.255.0"
```

When client1_vpn connects to the VPN, it automatically adds to its routing table an entry for the private network 192.168.0.0/24, reachable via the OpenVPN server

- Without the previous configuration line, any packet from client1_vpn to 192.168.0.0/24 would be delivered to its default gateway r1. Then r1 would discard them as it does not know the route

Security Considerations

- We have seen that a **non-member** of a VPN can reach other VPN clients!
  - `p_host` could access client1_vpn even if `p_host` was not in the VPN, and potentially not trusted
- Most VPNs, by default, **do not** provide isolation between the not trusted internet and the VPN itself
  - This must be done by hand using **firewalls**

Client-specific configuration

All of the following is already configured in `lab_vpn_pushroute`

In OpenVPN, if you want, you can specify client-specific configuration. In the server configuration add the following line

```
client-config-dir /root/ccd
```

Now in the server you can create a directory `ccd`. Inside it, you can create a file for each client you want to configure.

Here, we want to give a fixed address 192.168.100.101 to client1. Create a file `ccd/client1` and insert the following content

```
ifconfig-push 192.168.100.101 255.255.255.0
```

The name of the file depends on the client **Common Name** (CN) given when creating its key and certificates