OpenVPN + EasyRSA-3: Montando la VPN

*En esta práctica veremos cómo configurar un túnel **VPN (Virtual Private Network – Red Privada Virtual)** utilizando la **OpenVPN** y la nueva versión de **EasyRSA,** la **v3,** y lo correremos en Arch Linux, aunque el tutorial será válido para cualquier distro **GNU/Linux** en la que tengamos instalado **EasyRSA.***

OpenVPN

OpenVPN es una excelente herramienta para montar **túneles cifrados** en Internet, más conocidos como redes privadas virtuales, o **VPN** por sus siglas en inglés.

Si bien no nos adentraremos en detalles específicos de configuración (openvpn ofrece gran flexibilidad), sí montaremos un túnel VPN utilizando un servidor y cliente GNU/Linux, aunque la configuración es prácticamente la misma que la que haríamos en Windows.

EasyRSA

EasyRSA es una gran herramienta de automatización en la creación de certificados digitales y claves RSA. También nos permite generar módulos *Diffie-Hellman*, necesarios para ejecutar un servidor OpenVPN (en algún artículo específico sobre criptografía hablaremos de D-H y su intercambio de claves).

Nota: OpenSSL front-end

Si no queremos usar EasyRSA, igual podemos correr los comandos de OpenSSL a mano, esta es una forma de facilitar la creación y firmado (easyrsa no es más que un script de la shell, un front-end para openssl).

Instalando OpenVPN + EasyRSA

OpenVPN generalmente se encuentra disponible en los repos oficiales de la mayoría de las distribuciones… versiones más, versiones menos, cualquier versión reciente va a servirnos.

En mi caso particular lo he instalado en ArchLinux y en Debian 9:

```
sudo apt install openvpn   # Debian




sudo pacman -S openvpn # ArchLinux
```

El OpenVPN es el mismo software que puede correr como cliente o como servidor según sean las opciones que pasamos, por lo que lo instalaremos en nuestro cliente de la misma forma.

Por su parte, **EasyRSA** sí puede darnos algunos problemas, ya que en los repos oficiales a veces tenemos hasta la versión v2 (como es el caso de Debian al día de la fecha), y en otros la versión v3 (como en mi Arch… siempre al pie del cañón).

No es necesario instalar EasyRSA en servidor y cliente, con el servidor nos basta, puesto que nos va a servir para generar los certificados digitales y claves necesarias para el túnel OpenVPN, tanto para cliente como para servidor, y luego solo tendremos que transferir al cliente los archivos necesarios. (Ver notas adicionales abajo).

Si bien cualquiera de las versiones va a servirnos, desde la v2 a la v3 cambiaron varias cosas, y la más importante, la sintaxis de los comandos, razón por la cual veremos cómo instalar y usar la v3 en cualquier sistema GNU/Linux.

En ArchLinux, por ejemplo, dispongo de la versión v3 de EasyRSA en el repo oficial, por lo que simplemente lo podemos instalar de esta forma:

```
sudo pacman -S easy-rsa
```

En cambio, en Debian, por el momento, solo disponemos de la v2, por lo que, para que esta práctica sea útil, se ha utilizado el repositorio oficial de OpenVPN/EasyRSA en github.

```
cd /opt


git clone https://github.com/OpenVPN/easy-rsa.git


cd easy-rsa/easyrsa3
```

Otra forma es descargarse el tarball de la release… lo menciono por si sirve a alguien, pero no lo utilizaré:

```
cd /opt


wget  https://github.com/OpenVPN/easy-rsa/releases/download/v3.0.4/EasyRSA-3.0
.4.tgz


# (si no descarga, revisar en el repo la última versión disponible, puede cam
biar)


tar -xvf EasyRSA-3.0.4.tgz


cd EasyRSA-3.0.4/
```

Creando las claves: adaptando el entorno

Para OpenVPN necesitamos crear:

- Una clave privada y un certificado x509 para la **autoridad certificante** que firma (CA)
- Una clave privada y un certificado x509 firmado para el **servidor**.
- Una clave privada y un certificado x509 firmado para **cada cliente**.
- Un grupo **Diffie-Hellman** para el servidor.

Estos datos los crearemos con la EasyRSA-3, en el servidor (más simple) y luego copiaremos (vía **scp**) al cliente la información que necesite, a saber, clave privada y certificado del cliente, y certificado de la CA.

La generación de los certificados requiere que especifiquemos información de propiedad (ORG, CN, País, provincia, email, etc). Una forma de establecer esta información y evitar que nos la pida con cada

certificado creado, es copiar el archivo vars.example como vars, y luego editar algunas líneas:

```
cp vars.example vars

vim vars  # sí, uso Vim... pero pueden usar el editor de texto que quieran :)
```

De aquí solamente deberemos descomentar (eliminar el «#» del principio de la línea) a las siguientes líneas:

```
#set_var EASYRSA_REQ_COUNTRY "US"

#set_var EASYRSA_REQ_PROVINCE "California"

#set_var  EASYRSA_REQ_CITY  "San  Francisco"

#set_var EASYRSA_REQ_ORG "Copyleft Certificate Co"

#set_var EASYRSA_REQ_EMAIL "me@example.net"

#set_var EASYRSA_REQ_OU "My Organizational Unit"
```

Y adaptarlas con la info que quisiéramos... algo así:

```
set_var EASYRSA_REQ_COUNTRY "ES"

set_var EASYRSA_REQ_PROVINCE "Madrid"

set_var EASYRSA_REQ_CITY "Torrejon"

set_var EASYRSA_REQ_ORG "Informatica"

set_var EASYRSA_REQ_EMAIL "admin@asir.com"

set_var EASYRSA_REQ_OU "Ejemplo OpenVPN"
```

Creando las claves:

Ahora sí crearemos las claves. Cabe notar antes que los comandos que voy a colocar en los ejemplos son del tipo:

./easyrsa blablabla

Pero si el EasyRSA-3 fue instalado desde los repositorios, podremos simplemente ejecutarlos así:

easyrsa blablabla

Debido a que el binario está en el PATH del sistema… simplemente para tener en cuenta.

1. Iniciando el entorno EasyRSA PKI

Antes que nada, es necesario iniciar el entorno de **PKI** (Public Key Infrastructure / Infraestructura de clave pública) de EasyRSA (datos necesarios en memoria para comenzar a trabajar):

./easyrsa init-pki

La salida del comando será algo similar a esto:

```
init-pki complete; you may now create a CA or requests.


Your newly created PKI dir is: /opt/easy-rsa/easyrsa3/pki
```

Esto significa se creó en mi directorio local de easyrsa el subdirectorio **«pki»**. Esto es importante porque ahí dentro iremos generando las claves y certificados x509 neceasrios, y luego los copiaremos a las ubicaciones correspondientes.

Cabe aclarar también que, si ya disponemos del directorio «pki» y las claves, al ejecutar el «init-pki» borraremos lo que ya tenemos… **cuidado** con eso.

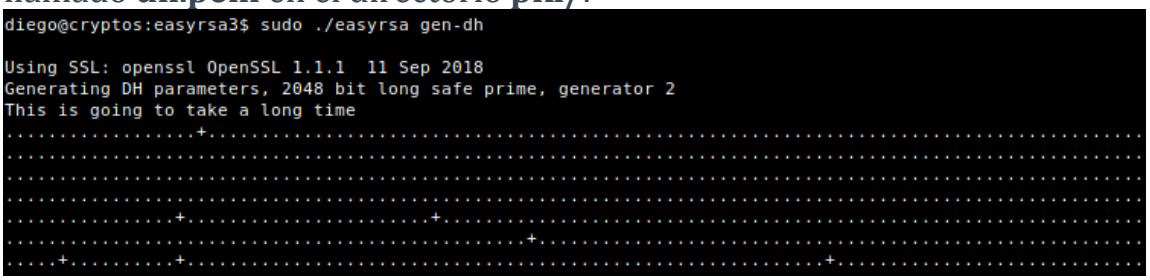
2. Generación de los parámetros Diffie-Hellman

En este punto crearemos la clave o módulos **Diffie-Hellman** utilizados por OpenVPN al establecer el primer contacto entre los nodos de la VPN.

Este paso generará un archivo que luego deberá ser configurado en el servidor de OpenVPN.

```
./easyrsa gen-dh
```

Esto demorará algunos segundos o minutos, y creará un archivo llamado **dh.pem** en el directorio **pki/**.



3. Clave RSA y certificado de la CA

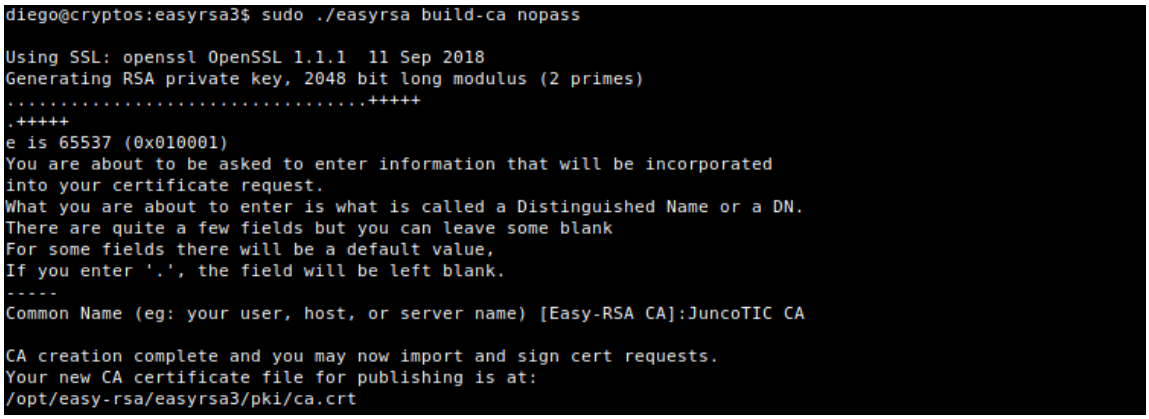
Para generar estos archivos utilizaremos el siguiente comando:

./easyrsa build-ca nopass

Este comando generará dos archivos:

- **ca.crt:** Es la clave pública de la autoridad certificante encapsulada en un formato de certificado digital x509 que tanto cliente como servidor OpenVPN utilizarán para identificarse entre si con confianza mutua. Por esto es necesario que este archivo esté disponible tanto en el servidor como en todos los clientes.
- **ca.key:** Es la clave privada RSA de la autoridad certificante, y es con la que se firman las claves y certificados tanto del servidor como de los clientes. Este archivo sólo debe estar disponible en la máquina que genere los certificados y claves, llamada «CA machine», que, en nuestro caso, será el mismo equipo server, pero podría ser cualquier otro. ***(Ver notas adicionales abajo)***.

OpenSSL da la posibilidad de crear una password o contraseña de acceso a las claves privadas. En nuestro caso sería un contratiempo puesto que cada vez que se vaya a utilizar la clave privada (para firmar, por ejemplo) deberíamos escribir la contraseña. Para evitar este paso es que especificamos el modificador **«nopass»** en el comando.



4. Clave RSA y certificado del servidor

En este caso crearemos la clave privada y pública (encapsulada en un certificado x509) para el servidor. Para ellos solo necesitamos correr este comando:

```
./easyrsa gen-req servidor nopass
```

El parámetro «servidor» es el nombre que le damos al servidor para identificarlo criptográficamente. Por su parte, **«nopass»** tiene el mismo efecto que el especificado en el comando anterior.

En este punto ya tendremos una **clave privada** para el servidor, y una petición de firma de certificado digital x509 (**sign request**) que incluye su **clave pública**. Estos archivos los encontraremos en el directorio **pki/** local, como bien nos lo dice la ejecución del comando:

```
...

Keypair and certificate request completed. Your files are:

req: /opt/easy-rsa/easyrsa3/pki/reqs/servidor.req

key: /opt/easy-rsa/easyrsa3/pki/private/servidor.key

...
```

```
diego@cryptos:easyrsa3$ sudo ./easyrsa gen-req servidor nopass

Using SSL: openssl OpenSSL 1.1.1  11 Sep 2018
Generating a RSA private key
.....................................................................................+++++
.........+++++
writing new private key to '/opt/easy-rsa/easyrsa3/pki/private/servidor.key.W8CUNJRSyM'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [servidor]:

Keypair and certificate request completed. Your files are:
req: /opt/easy-rsa/easyrsa3/pki/reqs/servidor.req
key: /opt/easy-rsa/easyrsa3/pki/private/servidor.key
```

El archivo **.req** representa la petición de firma, mientras que el archivo **.key** representa la clave privada, y se almacenan respectivamente en los directorios **reqs/** y **private/** respectivamente.

Para configurar el OpenVPN necesitamos el certificado del servidor firmado digitalmente por la CA. Para ello podemos ejecutar este comando:

```
./easyrsa sign-req server servidor
```

Este comando firmará el certificado de la entidad «servidor» como tipo «server». Los tipos válidos se encuentran en el directorio **x509-types/** dentro de nuestro **easyrsa/**. El certificado firmado se almacenará en el directorio **pki/issued/** y será de tipo CRT.

```
diego@cryptos:easyrsa3$ sudo ./easyrsa sign-req server servidor

Using SSL: openssl OpenSSL 1.1.1  11 Sep 2018


You are about to sign the following certificate.
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
source or that you have verified the request checksum with the sender.

Request subject, to be signed as a server certificate for 1080 days:

subject=
    commonName                = servidor


Type the word 'yes' to continue, or any other input to abort.
  Confirm request details: yes
Using configuration from /opt/easy-rsa/easyrsa3/pki/safessl-easyrsa.cnf
Can't open /opt/easy-rsa/easyrsa3/pki/index.txt.attr for reading, No such file or directory
140088781849088:error:02001002:system library:fopen:No such file or directory:crypto/bio/bss_file.c:72:fope
n('/opt/easy-rsa/easyrsa3/pki/index.txt.attr','r')
140088781849088:error:2006D080:BIO routines:BIO_new_file:no such file:crypto/bio/bss_file.c:79:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'servidor'
Certificate is to be certified until Oct 21 21:51:04 2021 GMT (1080 days)

Write out database with 1 new entries
Data Base Updated

Certificate created at: /opt/easy-rsa/easyrsa3/pki/issued/servidor.crt
```

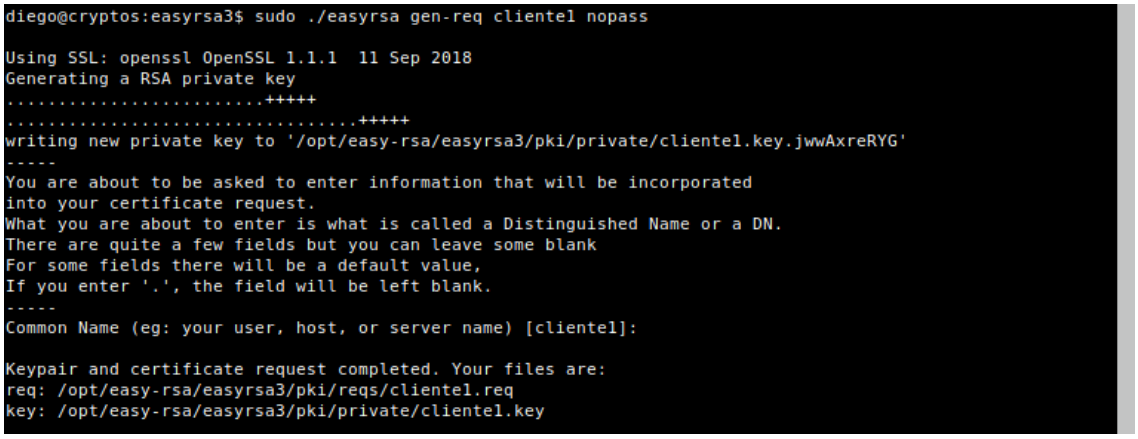
5. Clave RSA y certificado del cliente

Ahora vamos a crear la clave privada y certificado digital x509 de un cliente, y los firmaremos con la CA creada anteriormente.

Podemos hacerlo con el siguiente comando:

```
./easyrsa gen-req cliente1 nopass
```

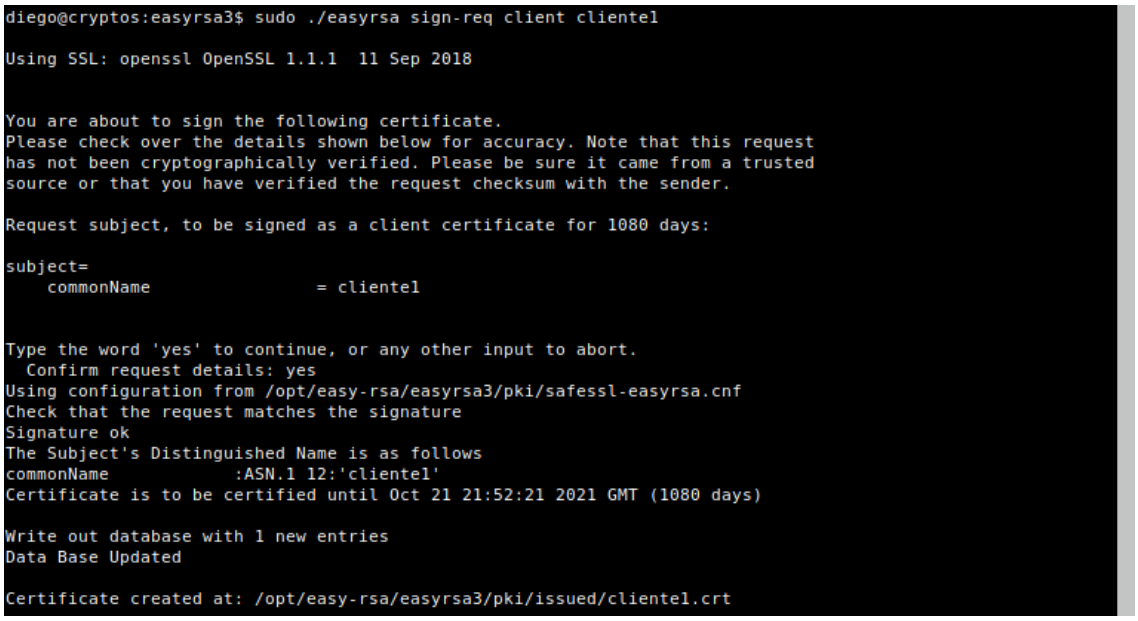
Esto nos generará, al igual que en el caso del servidor, la clave privada del cliente (ubicada en **pki/private)** y la petición de firma de certificado para este cliente (ubicada en **pki/reqs**).



El siguiente paso es firmar con la CA el certificado, y obtener el **.crt:**

```
./easyrsa sign-req client cliente1
```

Ahora sí, tendremos nuestro certificado crt firmado en el directorio **pki/issued** y listos para utilizar.



Este paso deberá ejecutarse para cada cliente que deseemos conectar.

6. Generando la clave TLS

Si bien este paso es opcional, resulta conveniente llevarlo a cabo, puesto que maximiza la seguridad del servidor, y reduce la posibilidad de ataques de DDoS.

Generaremos una clave simétrica para TLS desde la suite de OpenVPN, y luego deberemos copiarla a los equipos cliente (ver más adelante).

```
openvpn --genkey secret ta.key
```

Esto nos generará un archivo llamado ta.key dentro del directorio local.

Moviendo las claves

A este punto ya tenemos estos archivos generados y listos para usar:

- **pki/dh.pem** (para el servidor)
- **pki/ca.crt** (para el servidor y clientes)
- **pki/private/ca.key** (para firmar en la CA machine)
- **pki/private/servidor.key** (para el servidor)
- **pki/issued/servidor.crt** (para el servidor)
- **pki/private/cliente1.key** (para el cliente)
- **pki/issued/cliente1.crt** (para el cliente)
- **ta.key** (para el servidor y clientes)

Como todos estos archivos los creamos en nuestro equipo servidor, podemos copiar los necesarios al directorio del servidor OpenVPN.

Claves en el servidor OpenVPN

El servidor de OpenVPN necesita el certificado de la CA, certificado y clave del servidor, y el archivo de Diffie-Hellman:

```
sudo mkdir /etc/openvpn/keys

sudo cp pki/dh.pem /etc/openvpn/keys

sudo cp pki/ca.crt /etc/openvpn/keys

sudo cp pki/private/servidor.key /etc/openvpn/keys

sudo cp pki/issued/servidor.crt /etc/openvpn/keys

sudo cp ta.key /etc/openvpn/keys
```

Claves en el cliente OpenVPN

El cliente de OpenVPN necesita el certificado de la CA, y el certificado y clave del cliente, por lo que podemos copiar los archivos desde el servidor a un directorio temporal en el cliente:

```
scp pki/private/cliente1.key usuario@ip_cliente1_openvpn:/tmp

scp pki/issued/cliente1.crt usuario@ip_cliente1_openvpn:/tmp

scp pki/ca.crt usuario@ip_cliente1_openvpn:/tmp

scp ta.key usuario@ip_cliente1_openvpn:/tmp
```

Y luego, en el cliente, moverlos a su ubicación definitiva:

```
sudo mkdir /etc/openvpn/keys

sudo mv /tmp/cliente1.key /etc/openvpn/keys
```

```
sudo mv /tmp/cliente1.crt /etc/openvpn/keys

sudo mv /tmp/ca.crt /etc/openvpn/keys

sudo mv /tmp/ta.key /etc/openvpn/keys
```

Configurando el servidor

Para configurar el servidor crearemos un archivo llamado **servidor.conf** dentro del directorio **/etc/openvpn/**, y pondremos ahí las siguientes líneas:

```
port 1194

proto udp

#poner la red que estéis usando
server 192.168.10.0 255.255.255.0

client-to-client

persist-key

persist-tun

ca /etc/openvpn/ssl/ca.crt

cert /etc/openvpn/ssl/Servidor.crt

dh /etc/openvpn/ssl/dh.pem

key /etc/openvpn/ssl/Servidor.key

tls-auth /etc/openvpn/ssl/ta.key 0

crl-verify /etc/openvpn/ssl/crl.pem

comp-lzo adaptive

dev tun
```

```
ifconfig-pool-persist server-ipp.txt 0

keepalive 10 120

cipher AES-256-CBC

auth SHA512

tls-version-min 1.2

tls-cipher  TLS-DHE-RSA-WITH-AES-256-GCM-SHA384

log /var/log/openvpn/server.log

verb 3
```

Cabe resaltar, no obstante, las opciones **«ca», «cert», «key»,** y **«dh»** en las que hemos configurado, respectivamente, el certificado de la CA, el certificado del servidor, la clave del servidor, y los módulos Diffie-Hellman.

A esto podemos sumarle la opción **«tls-auth»** en la que especificamos la clave ta.key, y que en este caso hemos acompañado por un «**0**» (cero) al final para indicar que se trata del servidor.

Otra opción interesante para nuestro caso es la opción **«server»,** que especifica el pool de direcciones IP a entregar a los clientes cuando se conecten a la VPN.

Luego de esto, podemos correr nuestro servidor con el comando:

sudo openvpn --config /etc/openvpn/servidor.conf

Aunque debería iniciarse también con el gestor de servicios que utilice la distro (systemd, sysvinit, etc). Esto es porque el servicio de inicio busca un archivo **«.conf»** dentro del directorio **/etc/openvpn…** así que podríamos usar algo así para correr el daemon en segundo plano:

sudo systemctl start openvpn.service

Configurando el cliente

En el lado del cliente hacemos algo similar, creamos un archivo de configuración, en nuestro caso, **cliente1.conf,** dentro de **/etc/openvpn,** con un contenido similar a este:

```
client

dev tun

proto udp

port 1194

remote 192.168.0.10 1194

remote-cert-tls server

resolv-retry infinite

nobind

persist-key

persist-tun

comp-lzo

verb 3

cipher AES-256-CBC

auth SHA512

tls-version-min 1.2

tls-cipher TLS-DHE-RSA-WITH-AES-256-GCM-SHA384

ca /etc/openvpn/keys/ca.crt

key /etc/openvpn/keys/cliente.key

cert /etc/openvpn/keys/cliente.crt
```

```
key-direction 1


tls-auth /etc/openvpn/keys/ta.key 1
```

En este caso, las líneas interesantes son las que corresponden con las opciones **«ca», «cert»** y **«key»,** el certificado de la CA, el certificado del cliente, y su clave privada. Al igual que del lado del servidor, la línea **«tls-auth»** permite especificar la clave ta.key generada y transferida a este cliente, con la diferencia de que en este caso el parámetro es **«1»** (uno), indicando cliente.

La opción **«remote»** permite al cliente especificar la IP pública o nombre de dominio del servidor, y el puerto en el que está atendiendo el daemon.

El cliente puede lanzarse del mismo modo que el servidor, utilizando el gestor de servicios, o con el comando **«openvpn»** especificando el archivo **.conf** a utilizar (el cliente.conf en este caso).

Convenientemente podremos lanzarlo la primera vez desde línea de comandos, y verificar que no tengamos errores en la ejecución, y que el túnel se creó correctamente.

Probando las configuraciones!!

Sin más, corremos el servidor, y deberíamos ver algo similar a esto:

```
diego@cryptos:easyrsa3$ sudo openvpn --config /etc/openvpn/server.conf
Wed Nov  7 20:57:20 2018 Warning: Error redirecting stdout/stderr to --log file: /var/log/openvpn/server.log: No such file or directory
(errno=2)
Wed Nov  7 20:57:20 2018 OpenVPN 2.4.6 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] built on Apr
 24 2018
Wed Nov  7 20:57:20 2018 library versions: OpenSSL 1.1.1  11 Sep 2018, LZO 2.10
Wed Nov  7 20:57:20 2018 NOTE: your local LAN uses the extremely common subnet address 192.168.0.x or 192.168.1.x.  Be aware that this m
ight create routing conflicts if you connect to the VPN server from public locations such as internet cafes that use the same subnet.
Wed Nov  7 20:57:20 2018 Note: cannot open server-ipp.txt for READ
Wed Nov  7 20:57:20 2018 Diffie-Hellman initialized with 2048 bit key
Wed Nov  7 20:57:20 2018 Outgoing Control Channel Authentication: Using 512 bit message hash 'SHA512' for HMAC authentication
Wed Nov  7 20:57:20 2018 Incoming Control Channel Authentication: Using 512 bit message hash 'SHA512' for HMAC authentication
Wed Nov  7 20:57:20 2018 ROUTE_GATEWAY 192.168.0.1/255.255.255.0 IFACE=eth0 HWADDR=bc:5f:f4:d9:aa:5e
Wed Nov  7 20:57:20 2018 TUN/TAP device tun0 opened
Wed Nov  7 20:57:20 2018 TUN/TAP TX queue length set to 100
Wed Nov  7 20:57:20 2018 do_ifconfig, tt->did_ifconfig_ipv6_setup=0
Wed Nov  7 20:57:20 2018 /usr/bin/ip link set dev tun0 up mtu 1500
Wed Nov  7 20:57:20 2018 /usr/bin/ip addr add dev tun0 local 192.168.10.1 peer 192.168.10.2
Wed Nov  7 20:57:20 2018 /usr/bin/ip route add 192.168.10.0/24 via 192.168.10.2
Wed Nov  7 20:57:20 2018 Could not determine IPv4/IPv6 protocol. Using AF_INET
Wed Nov  7 20:57:20 2018 Socket Buffers: R=[212992->212992] S=[212992->212992]
Wed Nov  7 20:57:20 2018 UDPv4 link local (bound): [AF_INET][undef]:1194
Wed Nov  7 20:57:20 2018 UDPv4 link remote: [AF_UNSPEC]
Wed Nov  7 20:57:20 2018 MULTI: multi_init called, r=256 v=256
Wed Nov  7 20:57:20 2018 IFCONFIG POOL: base=192.168.10.4 size=62, ipv6=0
Wed Nov  7 20:57:20 2018 IFCONFIG POOL LIST
Wed Nov  7 20:57:20 2018 Initialization Sequence Completed
```

Y en el cliente debríamos ver algo así (la salida en el server va a modificarse por la negociación del túnel cifrado:

```
user@debian9:~$ sudo openvpn --config /etc/openvpn/cliente.conf
[sudo] password for user:
Wed Nov  7 20:53:33 2018 OpenVPN 2.4.0 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] built on Jul
 18 2017
Wed Nov  7 20:53:33 2018 library versions: OpenSSL 1.0.2l  25 May 2017, LZO 2.08
Wed Nov  7 20:53:33 2018 Outgoing Control Channel Authentication: Using 512 bit message hash 'SHA512' for HMAC authentication
Wed Nov  7 20:53:33 2018 Incoming Control Channel Authentication: Using 512 bit message hash 'SHA512' for HMAC authentication
Wed Nov  7 20:53:33 2018 TCP/UDP: Preserving recently used remote address: [AF_INET]192.168.0.10:1194
Wed Nov  7 20:53:33 2018 Socket Buffers: R=[212992->212992] S=[212992->212992]
Wed Nov  7 20:53:33 2018 UDP link local: (not bound)
Wed Nov  7 20:53:33 2018 UDP link remote: [AF_INET]192.168.0.10:1194
Wed Nov  7 20:53:33 2018 TLS: Initial packet from [AF_INET]192.168.0.10:1194, sid=4ad123c6 de936f98
Wed Nov  7 20:53:33 2018 VERIFY OK: depth=1, CN=Easy-RSA CA
Wed Nov  7 20:53:33 2018 Validating certificate key usage
Wed Nov  7 20:53:33 2018 ++ Certificate has key usage  00a0, expects 00a0
Wed Nov  7 20:53:33 2018 VERIFY KU OK
Wed Nov  7 20:53:33 2018 Validating certificate extended key usage
Wed Nov  7 20:53:33 2018 ++ Certificate has EKU (str) TLS Web Server Authentication, expects TLS Web Server Authentication
Wed Nov  7 20:53:33 2018 VERIFY EKU OK
Wed Nov  7 20:53:33 2018 VERIFY OK: depth=0, CN=Servidor
Wed Nov  7 20:53:33 2018 Control Channel: TLSv1.2, cipher TLSv1/SSLv3 DHE-RSA-AES256-GCM-SHA384, 2048 bit RSA
Wed Nov  7 20:53:33 2018 [Servidor] Peer Connection Initiated with [AF_INET]192.168.0.10:1194
Wed Nov  7 20:53:34 2018 SENT CONTROL [Servidor]: 'PUSH_REQUEST' (status=1)
Wed Nov  7 20:53:34 2018 PUSH: Received control message: 'PUSH_REPLY,route 192.168.10.0 255.255.255.0,topology net30,ping 10,ping-restar
t 120,ifconfig 192.168.10.6 192.168.10.5,peer-id 0,cipher AES-256-GCM'
Wed Nov  7 20:53:34 2018 OPTIONS IMPORT: timers and/or timeouts modified
Wed Nov  7 20:53:34 2018 OPTIONS IMPORT: --ifconfig/up options modified
Wed Nov  7 20:53:34 2018 OPTIONS IMPORT: route options modified
Wed Nov  7 20:53:34 2018 OPTIONS IMPORT: peer-id set
Wed Nov  7 20:53:34 2018 OPTIONS IMPORT: adjusting link_mtu to 1625
Wed Nov  7 20:53:34 2018 OPTIONS IMPORT: data channel crypto options modified
Wed Nov  7 20:53:34 2018 Data Channel Encrypt: Cipher 'AES-256-GCM' initialized with 256 bit key
Wed Nov  7 20:53:34 2018 Data Channel Decrypt: Cipher 'AES-256-GCM' initialized with 256 bit key
Wed Nov  7 20:53:34 2018 ROUTE_GATEWAY 192.168.0.1/255.255.255.0 IFACE=enp0s3 HWADDR=08:00:27:66:af:4a
Wed Nov  7 20:53:34 2018 TUN/TAP device tun0 opened
Wed Nov  7 20:53:34 2018 TUN/TAP TX queue length set to 100
Wed Nov  7 20:53:34 2018 do_ifconfig, tt->did_ifconfig_ipv6_setup=0
Wed Nov  7 20:53:34 2018 /sbin/ip link set dev tun0 up mtu 1500
Wed Nov  7 20:53:34 2018 /sbin/ip addr add dev tun0 local 192.168.10.6 peer 192.168.10.5
Wed Nov  7 20:53:34 2018 /sbin/ip route add 192.168.10.0/24 via 192.168.10.5
Wed Nov  7 20:53:34 2018 Initialization Sequence Completed
```

En este caso todo anduvo bien… y se creó el túnel. Una forma simple de verificarlo es viendo que efectivamente se creó una interfaz «tun» (sí, es por el **dev tun** de la configuración del server), tanto en cliente como servidor… veamos un ejemplo de la salida del comando **ip a** :

```
16: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 1
    link/none
    inet 192.168.10.1 peer 192.168.10.2/32 scope global tun0
       valid_lft forever preferred_lft forever
    inet6 fe80::9129:c90c:e475:925/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
```

El tunel se ha creado, y ahora tenemos comunicación cifrada!

Notas adicionales: una CA externa

En el caso de que la autoridad certificante no esté situada en el servidor, una opción es generar todos los certificados y claves desde la autoridad certificante y luego transferirlos, por ejemplo mediante scp, a servidor y clientes, y la otra opción es que tanto servidor como clientes generen sus claves privadas y sus peticiones de firma de certificado digital **(.req)** y luego firmemos las requests en el equipo de CA utilizando su clave privada.

Para esto, una vez generados los .req como hicimos antes, deberemos transferir al equipo de CA dichos req, y en este equipo importarlos en el easyrsa local de esta forma (suponiendo que en /tmp hemos colocado cliente1.req y servidor.req):

```
./easyrsa import-req /tmp/cliente1.req
```

```
./easyrsa import-req /tmp/servidor.req
```

Y luego firmar como hicimos antes:

```
./easyrsa sign-req client cliente1


./easyrsa sign-req server servidor
```

Esto nos generará los certificados en el directorio pki/issued/ de la CA, que deberemos transferir a cliente y servidor respectivamente.