## NAME

openvpn examples − Secure IP tunnel daemon

## INTRODUCTION

This man page gives a few simple examples to create OpenVPN setups and configuration files.

## EXAMPLES

Prior to running these examples, you should have OpenVPN installed on two machines with network connectivity between them. If you have not yet installed OpenVPN, consult the INSTALL file included in the OpenVPN distribution.

### Firewall Setup:

If firewalls exist between the two machines, they should be set to forward the port OpenVPN is configured to use, in both directions. The default for OpenVPN is 1194/udp. If you do not have control over the firewalls between the two machines, you may still be able to use OpenVPN by adding **−−ping 15** to each of the **openvpn** commands used below in the examples (this will cause each peer to send out a UDP ping to its remote peer once every 15 seconds which will cause many stateful firewalls to forward packets in both directions without an explicit firewall rule).

Please see your operating system guides for how to configure the firewall on your systems.

### VPN Address Setup:

For purposes of our example, our two machines will be called **bob.example.com** and **alice.example.com**. If you are constructing a VPN over the internet, then replace **bob.example.com** and **alice.example.com** with the internet hostname or IP address that each machine will use to contact the other over the internet.

Now we will choose the tunnel endpoints. Tunnel endpoints are private IP addresses that only have meaning in the context of the VPN. Each machine will use the tunnel endpoint of the other machine to access it over the VPN. In our example, the tunnel endpoint for bob.example.com will be 10.4.0.1 and for alice.example.com, 10.4.0.2.

Once the VPN is established, you have essentially created a secure alternate path between the two hosts which is addressed by using the tunnel endpoints. You can control which network traffic passes between the hosts (a) over the VPN or (b) independently of the VPN, by choosing whether to use (a) the VPN endpoint address or (b) the public internet address, to access the remote host. For example if you are on bob.example.com and you wish to connect to **alice.example.com** via **ssh** without using the VPN (since **ssh** has its own built−in security) you would use the command **ssh alice.example.com**. However in the same scenario, you could also use the command **telnet 10.4.0.2** to create a telnet session with alice.example.com over the VPN, that would use the VPN to secure the session rather than **ssh**.

You can use any address you wish for the tunnel endpoints but make sure that they are private addresses (such as those that begin with 10 or 192.168) and that they are not part of any existing subnet on the networks of either peer, unless you are bridging. If you use an address that is part of your local subnet for either of the tunnel endpoints, you will get a weird feedback loop.

### Example 1: A simple tunnel without security

On bob:

```
openvpn −−remote alice.example.com −−dev tun1 \
         −−ifconfig 10.4.0.1 10.4.0.2 −−verb 9
```

On alice:

```
openvpn −−remote bob.example.com −−dev tun1 \
         −−ifconfig 10.4.0.2 10.4.0.1 −−verb 9
```

Now verify the tunnel is working by pinging across the tunnel.

On bob:

```
ping 10.4.0.2
```

On alice:

```
ping 10.4.0.1
```

The **−−verb 9** option will produce verbose output, similar to the **tcpdump**(8) program. Omit the **−−verb 9** option to have OpenVPN run quietly.

### Example 2: A tunnel with static−key security (i.e. using a pre−shared secret)

First build a static key on bob.

```
openvpn --genkey --secret key
```

This command will build a key file called **key** (in ascii format). Now copy **key** to **alice.example.com** over a secure medium such as by using the **scp**(1) program.

On bob:

```
openvpn --remote alice.example.com --dev tun1  \
         --ifconfig 10.4.0.1 10.4.0.2 --verb 5 \
         --secret key
```

On alice:

```
openvpn --remote bob.example.com --dev tun1   \
        --ifconfig 10.4.0.2 10.4.0.1 --verb 5 \
        --secret key
```

Now verify the tunnel is working by pinging across the tunnel.

On bob:

```
ping 10.4.0.2
```

On alice:

```
ping 10.4.0.1
```

### Example 3: A tunnel with full TLS−based security

For this test, we will designate **bob** as the TLS client and **alice** as the TLS server.

*Note:* The client or server designation only has meaning for the TLS subsystem. It has no bearing on OpenVPN's peer−to−peer, UDP−based communication model.*

First, build a separate certificate/key pair for both bob and alice (see above where **−−cert** is discussed for more info). Then construct Diffie Hellman parameters (see above where **−−dh** is discussed for more info). You can also use the included test files **client.crt**, **client.key**, **server.crt**, **server.key** and **ca.crt**. The **.crt** files are certificates/public−keys, the **.key** files are private keys, and **ca.crt** is a certification authority who has signed both **client.crt** and **server.crt**. For Diffie Hellman parameters you can use the included file **dh2048.pem**.

*WARNING:*

All client, server, and certificate authority certificates and keys included in the OpenVPN distribution are totally insecure and should be used for testing only.

On bob:

```
openvpn --remote alice.example.com --dev tun1    \
        --ifconfig 10.4.0.1 10.4.0.2             \
        --tls-client --ca ca.crt                 \
        --cert client.crt --key client.key       \
        --reneg-sec 60 --verb 5
```

On alice:

```
openvpn --remote bob.example.com --dev tun1      \
        --ifconfig 10.4.0.2 10.4.0.1             \
        --tls-server --dh dh1024.pem --ca ca.crt \
        --cert server.crt --key server.key       \
        --reneg-sec 60 --verb 5
```

Now verify the tunnel is working by pinging across the tunnel.

On bob:

```
ping 10.4.0.2
```

On alice:

```
ping 10.4.0.1
```

Notice the **--reneg-sec 60** option we used above. That tells OpenVPN to renegotiate the data channel keys every minute. Since we used **--verb 5** above, you will see status information on each new key negotiation.

For production operations, a key renegotiation interval of 60 seconds is probably too frequent. Omit the **--reneg-sec 60** option to use OpenVPN's default key renegotiation interval of one hour.

**Routing:**

Assuming you can ping across the tunnel, the next step is to route a real subnet over the secure tunnel. Suppose that bob and alice have two network interfaces each, one connected to the internet, and the other to a private network. Our goal is to securely connect both private networks. We will assume that bob's private subnet is *10.0.0.0/24* and alice's is *10.0.1.0/24*.

First, ensure that IP forwarding is enabled on both peers. On Linux, enable routing:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

This setting is not persistent. Please see your operating systems documentation how to properly configure IP forwarding, which is also persistent through system boots.

If your system is configured with a firewall. Please see your operating systems guide on how to configure the firewall. You typically want to allow traffic coming from and going to the tun/tap adapter OpenVPN is configured to use.

On bob:

```
route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.4.0.2
```

On alice:

```
route add −net 10.0.0.0 netmask 255.255.255.0 gw 10.4.0.1
```

Now any machine on the *10.0.0.0/24* subnet can access any machine on the *10.0.1.0/24* subnet over the secure tunnel (or vice versa).

In a production environment, you could put the route command(s) in a script and execute with the **−−up** option.