

# Meet OpenVPN

By Hans-Cees Speel ([hanscees@hanscees.com](mailto:hanscees@hanscees.com))

If your company has people on the road, such as sales or technical people, a VPN is a good method for letting them access data on the company network. Many different VPN solutions can be bought, but many are free. Here, I discuss only solutions you can set up without buying a commercial VPN product.

The main VPN solution used for more complex tasks is IPsec; some people use PPTP. Although PPTP is usable, [security flaws](#) have occurred in its past, and it simply does not match up to IPsec.

IPsec in tunnel mode would be a much better solution, were it not for the crippled Windows-client implementation: Windows XP/2000 clients can't use IPsec in tunnel mode without using L2TP. There is nothing wrong with L2TP security-wise, but it increases latency--through the need for both PPP and L2TP processes--and increases packet-overhead, slowing down connections. Open-source servers have not had much experience with L2TP yet, so using open source for it is problematic at this time.

A disadvantage of plain IPsec is its notorious complexity: many, many things can and do go wrong. To the rescue, then, comes [OpenVPN](#), a full-blown open-source VPN solution based on SSL. OpenVPN offers the same functionality as IPsec in tunnel mode; you can tunnel entire networks through it. In this article, I focus on using OpenVPN as a road warrior's VPN solution.

Every VPN approach has its list of pros and cons. The pros of OpenVPN are:

- Same functionality as IPsec in tunnel mode: you can tunnel entire networks (IP tunnel or bridging tunnel).
- A Windows XP/2000 install.exe file with a GUI is available for starting the tunnel. The config files are text based.
- The OpenVPN server can push routes, DNS server IP addresses and other configuration details to the clients. This makes OpenVPN well suited for road-warrior setups, because you can modify the setup without touching far-away laptops.
- You can use a bridging or routing setup.
- The server/client code is the same: the config determines the role.
- SSL is as solidly proven as security protocols get, using RSA public key cryptography if you want. See [this paper](#) for more information on its security setup.
- OpenVPN costs you nothing in terms of money--a server, an Internet connection and know-how is all you need).
- Plenty of [man page](#) and [HOWTOs](#) are available to get you going.
- All encryption processes are handled in userland, meaning it is easy to install--much less complicated than IPsec.

The list of cons includes:

- The setup uses TUN/TAP devices. This can make things complicated to figure out when things go wrong. If Microsoft changes its code, it also might just break.
- The OpenVPN process is executed in userland and, thus, is relatively slow. TUN/TAP devices combine together with a userland-process to create a setup in which traffic has to cross userland/kernel borders relatively often. This setup might create rather high latency on connections.
- A packet overhead is present because IP/Ethernet is encapsulated in SSL and SSL in UDP/TCP.
- The latest version OpenVPN is beta; earlier versions have further drawbacks.
- Who can you call when things go wrong? Some companies want to pay to get support.

Considering these arguments, OpenVPN should be a serious option if you are setting up a VPN. The days when only money could get you a decent VPN definitely are over.

### Setting Up an OpenVPN (Routing) Scenario

The rest of this article is a guide to setting up a road-warrior scenario using routing, not bridging, with TUN devices. Its aim is to make sure laptops on the Internet can connect safely to companies' networks, using internal servers and data.

The basic HOWTO I drew on when writing this article can be found [here](#). It is a HOWTO for setting up OpenVPN in bridging mode on a Linux [SME-server](#). My setup is slightly different, because I do not use a bridging setup. Another good source is the [OpenVPN HOWTO](#).

## The Security Setup

Anyone setting up a VPN without considering the different kinds of security risks one faces is a fool. Therefore, you should start any VPN setup doing exactly that--considering security.

## Connection Security

OpenVPN traffic flowing over the Internet is protected by [TLS](#). The setup here uses public key exchange; computer authentication is done by RSA-based public/private key-pairs (public keys also are called certificates). In this setup we make our own root certificate; that is, for our VPN scheme, we are our own Verisign, so to speak. We are the root of the Web of trust here. We make a server key pair and multiple client-key pairs. We sign those with our own root certificate. This setup is this basic cryptography design of OpenVPN.

The SSL/TLS connection is set up up with those keys. After authentication is done, [Diffie-Hellmann encryption](#) is used to exchange keys to set up the connection. New keys are negotiated every hour using perfect forward secrecy, or PFS--the next key used is not derived by using the former key. By default, the connection uses 128-bit Blowfish in Cipher Block Chaining mode, with SHA1 message digests.

## Server Security

The OpenVPN server itself, of course, could be attacked. You can minimize that risk by:

- Using shared keys with the `tls-auth` option before public key exchange occurs. Doing so keeps people from exploiting the SSL setup, should this be possible.
- Setting options `user nobody` and `group nobody`. This makes sure the server does not run as root. You also can use a `chroot-jail`.
- Using a separate box in a DMZ. This way a successful hack is slowed down by the firewall protecting the internal network from the DMZ. Strange connects can be noticed in the firewall logging.
- By using `iptables` firewall rules on the OpenVPN server that prevent traffic from tunnel hosts entering the server, as well as all traffic from the Internet except for the need UDP traffic.

## Authentication of Users

The security setup of your client laptops is critical. If your road warriors are using laptops and can access your company's network, your data may become public in the future. No matter how good the SSL crypto, this is a separate risk. If a laptop can connect through an OpenVPN tunnel directly into your networks, you have a problem. To avoid this, you need to establish authentication of the user to the laptop or to the SSL keys.

Many ways exist to do this authentication. You can password-protect the SSL keys of the client, which is recommended. But if workers have the habit of writing down passwords near their laptops, password protection is not sufficient. An option is to get USB-based iKeys with a pincode that holds the client keys. Pincodes are easier to remember, so the need to write them down is smaller. Of course, the iKey should be carried on a keychain and not with the laptop itself. You should establish an AUP (acceptable user policy) to make sure all users understand this. Doing so may prevent a stolen laptop from becoming a disaster. In addition, you might use encrypted filesystems on laptops.

Another option is to set up your own custom authentication scheme. For instance, you can use strong authentication with hardware tokens, coupled with a Kerberos server. OpenVPN has the script hooks to do that. You also can use the server password file.

## Network Setup

The network setup my configuration files is aiming for is this:

- The OpenVPN server at 65.66.45.x.
- The client is somewhere on the Internet.
- The client/server P2P network is 192.168.100.0/24 or, rather, a /32 network in that network.
- The company-network behind the OpenVPN server is 172.16.1.0/24.

So, the internal mailserver of this company might be at 172.16.1.3, the DC at 172.16.1.5 and the fileserver at 172.16.1.6. Schematically, this setup looks like this:

```
CLIENT -> [modem/adsl-router] -> Internet <-UDP-> OpenVPNserver
CLIENT - TUNInterFace <=tunnel=> TUNInterFace ==> Internal network
CLIENT - 192.168.100.6 <=====> 192.168.100.5 <==> 172.16.1.0
```

I am using a **Linux SME-server**, which basically is a Red Hat system stripped down to what a file/printer/firewall/e-mail server needs, with a Perl/HTTP-based config panel. After being a problematic open-source project for a while, Linux SME-server is being developed further by Lycoris. I have used Linux SME-server for years and will migrate only if forced to--it is extremely easy to use.

### OpenVPN Server Install

Installing OpenVPN is easy to do. On the Linux server side, you must install one or two RPMs. On SME these RPMs are `lzo.xxx.rpm` and `openvpn-2.0_beta17-1.i386.rpm`. Most systems already include lzo. Your kernel should include TUN devices, most kernels do. If you run `openvpn` from `/usr/sbin/openvpn`, you should find a TUN device. With the settings we are going to use, it has a P2P connection.

The config file on my box is saved at `/etc/openvpn/server.conf`, but yours may be stored somewhere else. My server configuration file looks like the output shown below; see the [man page](#) to see what all the items reference. they mean):

```
###OpenVPN server config routing TUN setup#####
port 1194
dev tun
tls-server
mode server
dh dh1024.pem
ca ca.crt
cert SERVER.crt
key SERVER.key
duplicate-cn
ifconfig 192.168.100.1 192.168.100.2
ifconfig-pool 192.168.100.5 192.168.100.200 # IP range clients
mtu-test
tun-mtu 1500
tun-mtu-extra 32
mssfix 1450
```

```

#keep tunnel open by ping
push "ping 10"
push "ping-restart 60"
ping 10
ping-restart 120
#route to be established on the server
route-up "route delete -net 192.168.100.0/24"
route-up "route add -net 192.168.100.0/24 tun0"
#route to push to clients
push "route 172.16.1.0 255.255.255.0" #route to company network
push "dhcp-option DOMAIN hansceess.net" #push the DNS domain suffix
push "dhcp-option DNS 172.16.1.7" #push DNS entries to client
push "route 192.168.100.1" # add route to protected network
comp-lzo
status-version 2
status openvpn-status.log
verb 5
##### end server config #####

```

## Client Install

On the Windows client side, you should download the Windows installer and run it. The normal installer is available on the [OpenVPN Web site](#), while and the GUI version can be found [here](#). I recommend using the latter: it gives you a tray-icon with which you can start the OpenVPN service. In the Network connections window under Settings, you should find a tap win32 adapter. You also should see that adapter in your routing table when you type `route print` in a DOSBox.

After the setup has installed everything, you should adjust the config settings in D:/Program Files/OpenVPN/config/\*.ovpn to those you want. You might want to tune your personal firewall as well, if it sees the interface at all--mine didn't. My adjusted settings file looks like this:

```

#####client.ovpn#####
port 1194 #udp by default
dev tun
##remote is the openvpn-server
remote 65.66.45.x
tls-client
ca ca.crt
cert CLIENT.crt

```

```

key CLIENT.key
mtu-test
tun-mtu 1500
tun-mtu-extra 32
mssfix 1450
pull
#ip-win32 ipapi|manual|dynamic|netsh (see man page, use
#when ip address on interface does not appear, but dhcp server
#is visible in ipconfig /all)
#ip-win32 ipapi
comp-lzo
verb 4
#####end#####

```

The OpenVPN process on the client is a Windows service you can start with a script or with the GUI, if you want. The TAP device (in tun-modus) can be tcpdumped, as can any interface, which makes it nice when troubleshooting. The rest of the configuration comes from the server.

## Making RSA Keys

You can make keys for OpenVPN in the same way as you would make them for OpenSSL. But for those who like comfort, OpenVPN has an easy RSA set of scripts to help you out. You first must edit some variables in the vars file for the keys: names for server/clients keys, your company name, e-mail and so on. Next, run `./build-ca` to build your root private key. Then, run `./build-key server` to build the server key pair. Build the client key(s) with `./build-key client`. In this client step, you can add a passphrase in the keys for key-authentication, as discussed above. Finally, you must run `./build-dh` to generate the Diffie-Hellmann .pem file that the server needs. This file holds a large prime number and another parameter (see [this article](#) for details). Using these numbers, the server can generate new keys quickly, which it does every hour by default for standing connections.

You also need to copy the server keys, root certificate and \*.pem file to /openvpn. The client needs the client keys and the root certificate. These should be transported over a secure medium, such as winscp.

There is one thing left to do: get the firewall iptables rules on the server. The rules I added were:

```

# internet interface eth1 let OpenVPN udp port in
    /sbin/iptables --append INPUT -p udp --dport 1194 -m state --state
NEW -i eth1 -j ACCEPT
#block anything into the server from tun interface
    /sbin/iptables --append INPUT -i tun0 -j DROP
#tun0 debugging

```

```

    #/sbin/iptables --append INPUT -i tun0 -j LOG --log-prefix    tun0-input
    #/sbin/iptables --append OUTPUT -o tun0 -j LOG --log-prefix tun0-
output
    #/sbin/iptables --append FORWARD -i tun0 -j LOG --log-prefix Forward-
ComingFrom-Tunnel
    #/sbin/iptables --append FORWARD -o tun0 -j LOG --log-prefix Forward-
OutTo-Tunnel

#OpenVPN Forward chain: if you have a Tun-device, the forward chain
#screens traffic from networks/hosts outside the tunnel, going to
#internal networks and back. we want this traffic to go through,
#default but first we might want to block some things: remember, the
#other side of the tunnel is not safe by default: he could be routing
#so we should log syns at least coming in
    /sbin/iptables --append FORWARD -i tun0 -m state --state NEW --jump
LOG --log-prefix Tunnel_into_intranet
#you should know the networks allowed in and out through the tunnel
#let client network in.
    /sbin/iptables --append FORWARD -i tun0 --source 192.168.0.0/16 -j
ACCEPT
#let company network out
    /sbin/iptables --append FORWARD -o tun0 --source 172.16.0.0/16
-j ACCEPT
##troubleshoot: let all through
#    /sbin/iptables --append FORWARD -i tun0 -j ACCEPT
#    /sbin/iptables --append FORWARD -o tun0 -j ACCEPT
#you might want to allow some, but not all
#    /sbin/iptables --append FORWARD -i tun0 -p tcp --dport 25 -j ACCEPT
##drop the rest
    /sbin/iptables --append FORWARD -i tun0 -j DROP
    /sbin/iptables --append FORWARD -o tun+ -j DROP

```

## Testing

Once you have installed OpenVPN, it is time to test it. Make sure the server process is started with `service openvpn [re]start`. You should see the TUN device with `ifconfig`. With my config, it shows:

```

Link: encap:Point-to-Point Protocol
Inet addr:192.168.100.1 P-t-P 192.168.100.2.

```

Now, start up the client OpenVPN service. A file found at D:/Program Files/Openvpn/\*.log contains debugging information. With the `verb` setting, you can elaborate the logging. When you start the client service, the icon in your tray shouts it is connected. `Ipconfig /all` in a DOSBox shows an IP address on the tap interface, for instance, 192.168.100.10

Ethernet adapter Local Area Connection 8:

```
Connection-specific DNS Suffix . :
Description . . . . . : TAP-Win32 Adapter V8
Physical Address. . . . . : 00-FF-CF-10-9F-A6
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes
IP Address. . . . . : 192.168.100.10
Subnet Mask . . . . . : 255.255.255.252
Default Gateway . . . . . :
DHCP Server . . . . . : 192.168.100.5
```

`print route` gives you some routes:

```
192.168.100.1 255.255.255.255 192.168.100.9 4 1
192.168.100.8 255.255.255.252 192.168.100.10 4 1
192.168.100.10 255.255.255.255 127.0.0.1 127.0.0.1 1
192.168.100.255 255.255.255.255 192.168.100.10 4
```

Although this all may look quite odd, it works. You now can ping 192.168.100.1; if that succeeds the tunnel is okay. On the server you can see the pings coming in with `tcpdump -nlpi tun0`. Also, `tail -f /var/log/messages` supplies some information.

The routes on the server look something like this (`netstat -rn`) kernel IP routing table:

Destination	Gateway	Genmask	Flags	MSS	Window	irrtt	Iface
192.168.100.2	0.0.0.0	255.255.255.255	UH	0	0	0	tun0
192.168.100.0	0.0.0.0	255.255.255.0	U	0	0	0	tun0
65.66.45.2	0.0.0.0	255.255.255.0	U	0	0	0	eth1
172.16.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0

```
127.0.0.0      0.0.0.0      255.0.0.0      U          0 0          0 lo
0.0.0.0      65.66.45.1  0.0.0.0      UG         0 0          0 eth1
```

If all goes well, your connection should be there. If not, check the server routing table and tcpdump the TUN interfaces. You also can use the iptables debug rules.

## **Conclusion**

In this article I have shown a simple setup for a OpenVPN. In real life, the setup will not be much more complex. Although the security implications of any VPN should be well thought-out, setting up OpenVPN turned out to be rather easy. If you do get into trouble, plenty of helping hands can be found on the mailing lists.

OpenVPN is a serious VPN product. It can contend with IPsec in many ways. It certainly is cheap--try buying a Cisco concentrator--easy to install and, in the open-source tradition, tinkerable.

If OpenVPN has a disadvantage, it might be latency. However, no real-life data exists yet to back up that claim.