

[Home](#)[VPN Service](#)[VPN Solution](#)[Community](#)[Downloads](#)[Overview](#)[Downloads](#)[Source Code](#)[Documentation](#)[HOWTO](#)[Security Overview](#)[Examples](#)[Graphical User Interface](#)[Manuals](#)[Change Log](#)[Installation Notes](#)[Release Notes](#)[Miscellaneous](#)[Non-English](#)[File Signatures](#)[Articles](#)[FAQ](#)[General](#)[Client](#)[Server](#)[Books](#)[Wiki/Tracker](#)[Forums](#)[Contributing](#)

HOWTO

Introduction

OpenVPN is a full-featured SSL VPN which implements OSI layer 2 or 3 secure network extension using the industry standard SSL/TLS protocol, supports flexible client authentication methods based on certificates, smart cards, and/or username/password credentials, and allows user or group-specific access control policies using firewall rules applied to the VPN virtual interface. OpenVPN is not a web application proxy and does not operate through a web browser.

OpenVPN 2.0 expands on the capabilities of [OpenVPN 1.x](#) by offering a scalable client/server mode, allowing multiple clients to connect to a single OpenVPN server process over a single TCP or UDP port. OpenVPN 2.3 includes [a large number of improvements](#), including full IPv6 support and PolarSSL support.

This document provides step-by-step instructions for configuring an OpenVPN 2.x client/server VPN, including:

- [OpenVPN Quickstart](#).
- [Installing OpenVPN](#).
- [Determining whether to use a routed or bridged VPN](#).
- [Numbering private subnets](#).
- [Setting up your own Certificate Authority \(CA\) and generating certificates and keys for an OpenVPN server and multiple clients](#).
- [Creating configuration files for server and clients](#).
- [Starting up the VPN and testing for initial connectivity](#).
- [Configuring OpenVPN to run automatically on system startup](#).
- [Controlling a running OpenVPN process](#).
- [Expanding the scope of the VPN to include additional machines on either the client or server subnet](#).
- [Pushing DHCP options to clients](#).
- [Configuring client-specific rules and access policies](#).
- [Using alternative authentication methods](#).
- [How to add dual-factor authentication to an OpenVPN configuration using client-side smart cards](#).
- [Routing all client traffic \(including web-traffic\) through the VPN](#).
- [Running an OpenVPN server on a dynamic IP address](#).
- [Connecting to an OpenVPN server via an HTTP proxy](#).
- [Connecting to a Samba share over OpenVPN](#).
- [Implementing a load-balancing/failover configuration](#).
- [Hardening OpenVPN Security](#).
- [Revoking Certificates](#).
- [Additional Security Notes](#).

The impatient may wish to jump straight to the sample configuration files:

- [Server configuration file](#).
- [Client configuration file](#).

Intended Audience

This HOWTO assumes that readers possess a prior understanding of basic networking concepts such as IP addresses, DNS names, netmasks, subnets, IP routing, routers, network interfaces, LANs, gateways, and firewall rules.

Additional Documentation

OpenVPN Books

Please take a look at the [OpenVPN books page](#).

OpenVPN 1.x HOWTO

The original [OpenVPN 1.x HOWTO](#) is still available, and remains relevant for point-to-point or static-key configurations.

OpenVPN Articles

For additional documentation, see the [articles page](#) and the [OpenVPN wiki](#).

OpenVPN Quickstart

While this HOWTO will guide you in setting up a scalable client/server VPN using an X509 PKI (public key infrastructure using certificates and private keys), this might be overkill if you are only looking for a simple VPN setup with a server that can handle a single client.

If you would like to get a VPN running quickly with minimal configuration, you might check out the [Static Key Mini-HOWTO](#).

Static Key advantages

- Simple Setup
- No X509 PKI (Public Key Infrastructure) to maintain

Static Key disadvantages

- Limited scalability -- one client, one server
- Lack of *perfect forward secrecy* -- key compromise results in total disclosure of previous sessions
- Secret key must exist in plaintext form on each VPN peer
- Secret key must be exchanged using a pre-existing secure channel

Installing OpenVPN

OpenVPN source code and Windows installers can be [downloaded here](#). Recent releases (2.2 and later) are also available as Debian and RPM packages; see the [OpenVPN wiki](#) for details.

For security, it's a good idea to check the [file release signature](#) after downloading.

The OpenVPN executable should be installed on both server and client machines, since the single executable provides both client and server functions.

Linux Notes (using RPM package)

If you are using a Linux distribution which supports RPM packages (SuSE, Fedora, Redhat, etc.), it's best to install using this mechanism. The easiest method is to find an existing binary RPM file for your distribution. You can also build your own binary RPM file:

```
rpmbuild -tb openvpn-[version].tar.gz
```

Once you have the .rpm file, you can install it with the usual

```
rpm -ivh openvpn-[details].rpm
```

or upgrade an existing installation with

```
rpm -Uvh openvpn-[details].rpm
```

Installing OpenVPN from a binary RPM package has these dependencies:

- openssl
- lzo
- pam

Furthermore, if you are building your own binary RPM package, there are several additional dependencies:

- openssl-devel
- lzo-devel
- pam-devel

See the [openvpn.spec](#) file for additional notes on building an RPM package for Red Hat Linux 9 or building with reduced dependencies.

Linux Notes (without RPM)

If you are using Debian, Gentoo, or a non-RPM-based Linux distribution, use your distro-specific packaging mechanism such as **apt-get** on Debian or **emerge** on Gentoo.

It is also possible to install OpenVPN on Linux using the universal **.configure** method. First expand the .tar.gz file:

```
tar xzf openvpn-[version].tar.gz
```

Then cd to the top-level directory and type:

```
./configure
make
make install
```

Windows Notes

OpenVPN for Windows can be installed from the self-installing exe file on the [OpenVPN download page](#). Remember that OpenVPN will only run on Windows XP or later. Also note that OpenVPN must be installed and run by a user who has administrative privileges (this restriction is imposed by Windows, not OpenVPN). The restriction can be sidestepped by running OpenVPN in the background as a service, in which case even non-admin users will be able to access the VPN, once it is installed. [More discussion on OpenVPN + Windows privilege issues.](#)

Official OpenVPN Windows installers include [OpenVPN-GUI](#), which allows managing OpenVPN connections from a system tray applet. Other [GUI](#) applications are also available.

After you've run the Windows installer, OpenVPN is ready for use and will associate itself with files having the **.ovpn** extension. To run OpenVPN, you can:

- Right click on an OpenVPN configuration file (.ovpn) and select **Start OpenVPN on this configuration file**. Once running, you can use the **F4** key to exit.

- Run OpenVPN from a command prompt Window with a command such as:

```
openvpn myconfig.ovpn
```

Once running in a command prompt window, OpenVPN can be stopped by the **F4** key.

- Run OpenVPN as a service by putting one or more .ovpn configuration files in **\Program Files\OpenVPN\config** and starting the OpenVPN Service, which can be controlled from Start Menu -> Control Panel -> Administrative Tools -> Services.

[Additional Windows install notes.](#)

Mac OS X Notes

Angelo Laub and Dirk Theisen have developed an [OpenVPN GUI for OS X](#).

Other OSes

Some notes are available in the [INSTALL](#) file for specific OSes. In general, the

```
./configure
make
make install
```

method can be used, or you can search for an OpenVPN port or package which is specific to your OS/distribution.

Determining whether to use a routed or bridged VPN

See [FAQ](#) for an overview of Routing vs. Ethernet Bridging. See also the OpenVPN [Ethernet Bridging](#) page for more notes and details on bridging.

Overall, routing is probably a better choice for most people, as it is more efficient and easier to set up (as far as the OpenVPN configuration itself) than bridging. Routing also provides a greater ability to selectively control access rights on a client-specific basis.

I would recommend using routing unless you need a specific feature which requires bridging, such as:

- the VPN needs to be able to handle non-IP protocols such as IPX,
- you are running applications over the VPN which rely on network broadcasts (such as LAN games), or
- you would like to allow browsing of Windows file shares across the VPN without setting up a Samba or WINS server.

Numbering private subnets

Setting up a VPN often entails linking together private subnets from different locations.

The Internet Assigned Numbers Authority (IANA) has reserved the following three blocks of the IP address space for private internets (codified in RFC 1918):

10.0.0.0	10.255.255.255	(10/8 prefix)
172.16.0.0	172.31.255.255	(172.16/12 prefix)
192.168.0.0	192.168.255.255	(192.168/16 prefix)

While addresses from these netblocks should normally be used in VPN configurations, it's important to select addresses that minimize the probability of IP address or subnet conflicts. The types of conflicts that need to be avoided are:

- conflicts from different sites on the VPN using the same LAN subnet numbering, or
- remote access connections from sites which are using private subnets which conflict with your VPN subnets.

For example, suppose you use the popular 192.168.0.0/24 subnet as your private LAN subnet. Now you are trying to connect to the VPN from an internet cafe which is using the same subnet for its WiFi LAN. You will have a routing conflict because your machine won't know if 192.168.0.1 refers to the local WiFi gateway or to the same address on the VPN.

As another example, suppose you want to link together multiple sites by VPN, but each site is using 192.168.0.0/24 as its LAN subnet. This won't work without adding a complexifying layer of NAT translation, because the VPN won't know how to route packets between multiple sites if those sites don't use a subnet which uniquely identifies them.

The best solution is to avoid using 10.0.0.0/24 or 192.168.0.0/24 as private LAN network addresses. Instead, use something that has a lower probability of being used in a WiFi cafe, airport, or hotel where you might expect to connect from remotely. The best candidates are subnets in the middle of the vast 10.0.0.0/8 netblock (for example 10.66.77.0/24).

And to avoid cross-site IP numbering conflicts, always use unique numbering for your LAN subnets.

Setting up your own Certificate Authority (CA) and generating certificates and keys for an OpenVPN server and multiple clients

Overview

The first step in building an OpenVPN 2.x configuration is to establish a PKI (public key infrastructure). The PKI consists of:

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

OpenVPN supports bidirectional authentication based on certificates, meaning that the client must authenticate the server certificate and the server must authenticate the client certificate before mutual trust is established.

Both server and client will authenticate the other by first verifying that the presented certificate was signed by the master certificate authority (CA), and then by testing information in the now-authenticated certificate header, such as the certificate common name or certificate type (client or server).

This security model has a number of 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 will only accept clients whose certificates were signed by the master CA certificate (which we will generate below). And because the server can perform this signature verification without needing access to the CA private key itself, it is possible for the CA key (the most sensitive key in the entire PKI) to 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.
- The server can enforce client-specific access rights based on embedded certificate fields, such as the Common Name.

Note that the server and client clocks need to be roughly in sync or certificates might not work properly.

Generate the master Certificate Authority (CA) certificate & key

In this section we will generate a master CA certificate/key, a server certificate/key, and certificates/keys for 3 separate clients.

For PKI management, we will use *easy-rsa 2*, a set of scripts which is bundled with OpenVPN 2.2.x and earlier. If you're using OpenVPN 2.3.x, you need to download *easy-rsa 2* separately from [here](#).

For PKI management, we will use [easy-rsa 2](#), a set of scripts which is bundled with OpenVPN 2.2.x and earlier. If you're using OpenVPN 2.3.x, you may need to download *easy-rsa 2* separately from the [easy-rsa-old project page](#). An *easy-rsa 2* package is also available for Debian and Ubuntu in the [OpenVPN software repos](#). On *NIX platforms you should look into using [easy-rsa 3](#) instead; refer to its own documentation for details.

If you are using Linux, BSD, or a unix-like OS, open a shell and cd to the **easy-rsa** subdirectory. If you installed OpenVPN from a RPM or DEB file, the *easy-rsa* directory can usually be found in `/usr/share/doc/packages/openvpn` or `/usr/share/doc/openvpn` (it's best to copy this directory to another location such as `/etc/openvpn`, before any edits, so that future OpenVPN package upgrades won't overwrite your modifications). If you installed from a .tar.gz file, the *easy-rsa* directory will be in the top level directory of the expanded source tree.

If you are using Windows, open up a Command Prompt window and cd to `\Program Files\OpenVPN\easy-rsa`. Run the following batch file to copy configuration files into place (this will overwrite any preexisting `vars.bat` and `openssl.cnf` files):

```
init-config
```

Now edit the **vars** file (called **vars.bat** on Windows) and set the `KEY_COUNTRY`, `KEY_PROVINCE`, `KEY_CITY`, `KEY_ORG`, and `KEY_EMAIL` parameters. Don't leave any of these parameters blank.

Next, initialize the PKI. On Linux/BSD/Unix:

```
./vars
./clean-all
./build-ca
```

On Windows:

```
vars
clean-all
build-ca
```

The final command (**build-ca**) will build the certificate authority (CA) certificate and key by invoking the interactive **openssl** command:

```
ai:easy-rsa # ./build-ca
Generating a 1024 bit RSA private key
.....++++++
.....++++++
writing new private key to 'ca.key'
-----
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.
-----
Country Name (2 letter code) [KG]:
State or Province Name (full name) [NA]:
Locality Name (eg, city) [BISHKEK]:
Organization Name (eg, company) [OpenVPN-TEST]:
Organizational Unit Name (eg, section) []:
```

```
Common Name (eg, your name or your server's hostname) []:OpenVPN-CA
Email Address [me@myhost.mydomain]:
```

Note that in the above sequence, most queried parameters were defaulted to the values set in the **vars** or **vars.bat** files. The only parameter which must be explicitly entered is the **Common Name**. In the example above, I used "OpenVPN-CA".

Generate certificate & key for server

Next, we will generate a certificate and private key for the server. On Linux/BSD/Unix:

```
./build-key-server server
```

On Windows:

```
build-key-server server
```

As in the previous step, most parameters can be defaulted. When the **Common Name** is queried, enter "server". Two other queries require positive responses, "Sign the certificate? [y/n]" and "1 out of 1 certificate requests certified, commit? [y/n]".

Generate certificates & keys for 3 clients

Generating client certificates is very similar to the previous step. On Linux/BSD/Unix:

```
./build-key client1
./build-key client2
./build-key client3
```

On Windows:

```
build-key client1
build-key client2
build-key client3
```

If you would like to password-protect your client keys, substitute the **build-key-pass** script.

Remember that for each client, make sure to type the appropriate **Common Name** when prompted, i.e. "client1", "client2", or "client3". Always use a unique common name for each client.

Generate Diffie Hellman parameters

[Diffie Hellman](#) parameters must be generated for the OpenVPN server. On Linux/BSD/Unix:

```
./build-dh
```

On Windows:

```
build-dh
```

Output:

```
ai:easy-rsa # ./build-dh
Generating DH parameters, 1024 bit long safe prime, generator 2
This is going to take a long time
.....+.....
.....+.....+.....+.....
.....
```

Key Files

Now we will find our newly-generated keys and certificates in the **keys** subdirectory. Here is an explanation of the relevant files:

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
client2.crt	client2 only	Client2 Certificate	NO

client2.key	client2 only	Client2 Key	YES
client3.crt	client3 only	Client3 Certificate	NO
client3.key	client3 only	Client3 Key	YES

The final step in the key generation process is to copy all files to the machines which need them, taking care to copy secret files over a secure channel.

Now wait, you may say. Shouldn't it be possible to set up the PKI without a pre-existing secure channel?

The answer is ostensibly yes. In the example above, for the sake of brevity, we generated all private keys in the same place. With a bit more effort, we could have done this differently. For example, instead of generating the client certificate and keys on the server, we could have had the client generate its own private key locally, and then submit a Certificate Signing Request (CSR) to the key-signing machine. In turn, the key-signing machine could have processed the CSR and returned a signed certificate to the client. This could have been done without ever requiring that a secret **.key** file leave the hard drive of the machine on which it was generated.

Creating configuration files for server and clients

Getting the sample config files

It's best to use the OpenVPN [sample configuration files](#) as a starting point for your own configuration. These files can also be found in

- the **sample-config-files** directory of the OpenVPN source distribution
- the **sample-config-files** directory in `/usr/share/doc/packages/openvpn` or `/usr/share/doc/openvpn` if you installed from an RPM or DEB package
- **Start Menu -> All Programs -> OpenVPN -> OpenVPN Sample Configuration Files** on Windows

Note that on Linux, BSD, or unix-like OSes, the sample configuration files are named **server.conf** and **client.conf**. On Windows they are named **server.ovpn** and **client.ovpn**.

Editing the server configuration file

The sample server configuration file is an ideal starting point for an OpenVPN server configuration. It will create a VPN using a virtual **TUN** network interface (for routing), will listen for client connections on **UDP port 1194** (OpenVPN's official port number), and distribute virtual addresses to connecting clients from the **10.8.0.0/24** subnet.

Before you use the sample configuration file, you should first edit the **ca**, **cert**, **key**, and **dh** parameters to point to the files you generated in the [PKI](#) section above.

At this point, the server configuration file is usable, however you still might want to customize it further:

- If you are using [Ethernet bridging](#), you must use **server-bridge** and **dev tap** instead of **server** and **dev tun**.
- If you want your OpenVPN server to listen on a TCP port instead of a UDP port, use **proto tcp** instead of **proto udp** (If you want OpenVPN to listen on both a UDP and TCP port, you must run two separate OpenVPN instances).
- If you want to use a virtual IP address range other than **10.8.0.0/24**, you should modify the **server** directive. Remember that this virtual IP address range should be a private range which is currently unused on your network.
- Uncomment out the **client-to-client** directive if you would like connecting clients to be able to reach each other over the VPN. By default, clients will only be able to reach the server.
- If you are using Linux, BSD, or a Unix-like OS, you can improve security by uncommenting out the **user nobody** and **group nobody** directives.

If you want to run multiple OpenVPN instances on the same machine, each using a different configuration file, it is possible if you:

- Use a different **port** number for each instance (the UDP and TCP protocols use different port spaces so you can run one daemon listening on UDP-1194 and another on TCP-1194).
- If you are using Windows, each OpenVPN configuration file needs to have its own TAP-Windows adapter. You can add additional adapters by going to **Start Menu -> All Programs -> TAP-Windows -> Add a new TAP-Windows virtual ethernet adapter**.
- If you are running multiple OpenVPN instances out of the same directory, make sure to edit directives which create output files so that multiple instances do not overwrite each other's output files. These directives include **log**, **log-append**, **status**, and **ifconfig-pool-persist**.

Editing the client configuration files

The sample client configuration file (**client.conf** on Linux/BSD/Unix or **client.ovpn** on Windows) mirrors the default directives set in the sample server configuration file.

- Like the server configuration file, first edit the **ca**, **cert**, and **key** parameters to point to the files you generated in the [PKI](#) section above. Note that each client should have its own **cert/key** pair. Only the **cafile** is universal across the OpenVPN server and all clients.

- Next, edit the **remote** directive to point to the hostname/IP address and port number of the OpenVPN server (if your OpenVPN server will be running on a single-NIC machine behind a firewall/NAT-gateway, use the public IP address of the gateway, and a port number which you have configured the gateway to forward to the OpenVPN server).
- Finally, ensure that the client configuration file is consistent with the directives used in the server configuration. The major thing to check for is that the **dev** (tun or tap) and **proto** (udp or tcp) directives are consistent. Also make sure that **comp-lzo** and **fragment**, if used, are present in both client and server config files.

Starting up the VPN and testing for initial connectivity

Starting the server

First, make sure the OpenVPN server will be accessible from the internet. That means:

- opening up UDP port 1194 on the firewall (or whatever TCP/UDP port you've configured), or
- setting up a port forward rule to forward UDP port 1194 from the firewall/gateway to the machine running the OpenVPN server.

Next, [make sure that the TUN/TAP interface is not firewalled](#).

To simplify troubleshooting, it's best to initially start the OpenVPN server from the command line (or right-click on the **.ovpn** file on Windows), rather than start it as a daemon or service:

```
openvpn [server config file]
```

A normal server startup should look like this (output will vary across platforms):

```
Sun Feb 6 20:46:38 2005 OpenVPN 2.0_rc12 i686-suse-linux [SSL] [LZO] [EPOLL] built on Feb
Sun Feb 6 20:46:38 2005 Diffie-Hellman initialized with 1024 bit key
Sun Feb 6 20:46:38 2005 TLS-Auth MTU parms [ L:1542 D:138 EF:38 EB:0 ET:0 EL:0 ]
Sun Feb 6 20:46:38 2005 TUN/TAP device tun1 opened
Sun Feb 6 20:46:38 2005 /sbin/ifconfig tun1 10.8.0.1 pointopoint 10.8.0.2 mtu 1500
Sun Feb 6 20:46:38 2005 /sbin/route add -net 10.8.0.0 netmask 255.255.255.0 gw 10.8.0.2
Sun Feb 6 20:46:38 2005 Data Channel MTU parms [ L:1542 D:1450 EF:42 EB:23 ET:0 EL:0 AF:0
Sun Feb 6 20:46:38 2005 UDPv4 link local (bound): [undef]:1194
Sun Feb 6 20:46:38 2005 UDPv4 link remote: [undef]
Sun Feb 6 20:46:38 2005 MULTI: multi_init called, r=256 v=256
Sun Feb 6 20:46:38 2005 IFCONFIG POOL: base=10.8.0.4 size=62
Sun Feb 6 20:46:38 2005 IFCONFIG POOL LIST
Sun Feb 6 20:46:38 2005 Initialization Sequence Completed
```

Starting the client

As in the server configuration, it's best to initially start the OpenVPN server from the command line (or on Windows, by right-clicking on the **client.ovpn** file), rather than start it as a daemon or service:

```
openvpn [client config file]
```

A normal client startup on Windows will look similar to the server output above, and should end with the **Initialization Sequence Completed** message.

Now, try a ping across the VPN from the client. If you are using routing (i.e. **dev tun** in the server config file), try:

```
ping 10.8.0.1
```

If you are using bridging (i.e. **dev tap** in the server config file), try to ping the IP address of a machine on the server's ethernet subnet.

If the ping succeeds, congratulations! You now have a functioning VPN.

Troubleshooting

If the ping failed or the OpenVPN client initialization failed to complete, here is a checklist of common symptoms and their solutions:

- You get the error message: **TLS Error: TLS key negotiation failed to occur within 60 seconds (check your network connectivity)**. This error indicates that the client was unable to establish a network connection with the server.

Solutions:

- Make sure the client is using the correct hostname/IP address and port number which will allow it to reach the OpenVPN server.
- If the OpenVPN server machine is a single-NIC box inside a protected LAN, make sure you are using a correct port forward rule on the server's gateway firewall. For example, suppose your OpenVPN box is at 192.168.4.4 inside the firewall, listening for client connections on UDP port 1194. The NAT gateway servicing the 192.168.4.x subnet should have a port forward rule that says **forward UDP port 1194 from my public IP address to 192.168.4.4**.
- Open up the server's firewall to allow incoming connections to UDP port 1194 (or whatever TCP/UDP port you have configured in the server config file).
- You get the error message: **Initialization Sequence Completed with errors**-- This error can occur on Windows if (a) You don't have the DHCP client service running, or (b) You are using certain third-party personal firewalls on XP SP2.

Solution: Start the DHCP client service and make sure that you are using a personal firewall which is known to work correctly on XP SP2.

- You get the **Initialization Sequence Completed** message but the ping test fails -- This usually indicates that a firewall on either server or client is blocking VPN network traffic by filtering on the TUN/TAP interface.

Solution: Disable the client firewall (if one exists) from filtering the TUN/TAP interface on the client. For example on Windows XP SP2, you can do this by going to **Windows Security Center -> Windows Firewall -> Advanced** and unchecking the box which corresponds to the TAP-Windows adapter (disabling the client firewall from filtering the TUN/TAP adapter is generally reasonable from a security perspective, as you are essentially telling the firewall not to block authenticated VPN traffic). Also make sure that the TUN/TAP interface on the server is not being filtered by a firewall (having said that, note that selective firewalling of the TUN/TAP interface on the server side can confer certain security benefits. See the [access policies](#) section below).

- The connection stalls on startup when using a **proto udp** configuration, the server log file shows this line:

```
TLS: Initial packet from x.x.x.x:x, sid=xxxxxxxx xxxxxxxx
```

however the client log does not show an equivalent line.

Solution: You have a one-way connection from client to server. The server to client direction is blocked by a firewall, usually on the client side. The firewall can either be (a) a personal software firewall running on the client, or (b) the NAT router gateway for the client. Modify the firewall to allow returning UDP packets from the server to reach the client.

See the [FAQ](#) for additional troubleshooting information.

Configuring OpenVPN to run automatically on system startup

The lack of standards in this area means that most OSes have a different way of configuring daemons/services for autostart on boot. The best way to have this functionality configured by default is to install OpenVPN as a package, such as via RPM on Linux or using the Windows installer.

Linux

If you install OpenVPN via an RPM or DEB package on Linux, the installer will set up an **initscript**. When executed, the initscript will scan for **.conf** configuration files in **/etc/openvpn**, and if found, will start up a separate OpenVPN daemon for each file.

Windows

The Windows installer will set up a Service Wrapper, but leave it turned off by default. To activate it, go to Control Panel / Administrative Tools / Services, select the OpenVPN service, right-click on properties, and set the Startup Type to Automatic. This will configure the service for automatic start on the next reboot.

When started, the OpenVPN Service Wrapper will scan the **\Program Files\OpenVPN\config** folder for **.ovpn** configuration files, starting a separate OpenVPN process on each file.

Controlling a running OpenVPN process

Running on Linux/BSD/Unix

OpenVPN accepts several signals:

- **SIGUSR1** -- Conditional restart, designed to restart without root privileges
- **SIGHUP** -- Hard restart
- **SIGUSR2** -- Output connection statistics to log file or syslog
- **SIGTERM, SIGINT** -- Exit

Use the **writepid** directive to write the OpenVPN daemon's PID to a file, so that you know where to send the signal (if you are starting openvpn with an **initscript**, the script may already be passing a **--writepid** directive on the **openvpn** command line).

Running on Windows as a GUI

See the [OpenVPN GUI page](#).

Running in a Windows command prompt window

On Windows, you can start OpenVPN by right clicking on an OpenVPN configuration file (**.ovpn** file) and selecting "Start OpenVPN on this config file".

Once running in this fashion, several keyboard commands are available:

- **F1** -- Conditional restart (doesn't close/reopen TAP adapter)
- **F2** -- Show connection statistics
- **F3** -- Hard restart
- **F4** -- Exit

Running as a Windows Service

When OpenVPN is started as a service on Windows, the only way to control it is:

- Via the service control manager (Control Panel / Administrative Tools / Services) which gives start/stop control.
- Via the management interface (see below).

Modifying a live server configuration

While most configuration changes require you to restart the server, there are two directives in particular which refer to files which can be dynamically updated on-the-fly, and which will take immediate effect on the server without needing to restart the server process.

client-config-dir -- This directive sets a client configuration directory, which the OpenVPN server will scan on every incoming connection, searching for a client-specific configuration file (see the [the manual page](#) for more information). Files in this directory can be updated on-the-fly, without restarting the server. Note that changes in this directory will only take effect for new connections, not existing connections. If you would like a client-specific configuration file change to take immediate effect on a currently connected client (or one which has disconnected, but where the server has not timed-out its instance object), kill the client instance object by using the management interface (described below). This will cause the client to reconnect and use the new **client-config-dir** file.

crl-verify -- This directive names a *Certificate Revocation List* file, described below in the [Revoking Certificates](#) section. The CRL file can be modified on the fly, and changes will take effect immediately for new connections, or existing connections which are renegotiating their SSL/TLS channel (occurs once per hour by default). If you would like to kill a currently connected client whose certificate has just been added to the CRL, use the management interface (described below).

Status File

The default [server.conf](#) file has a line

```
status openvpn-status.log
```

which will output a list of current client connections to the file **openvpn-status.log** once per minute.

Using the management interface

The [OpenVPN management interface](#) allows a great deal of control over a running OpenVPN process. You can use the management interface directly, by telnetting to the management interface port, or indirectly by using an [OpenVPN GUI](#) which itself connects to the management interface.

To enable the management interface on either an OpenVPN server or client, add this to the configuration file:

```
management localhost 7505
```

This tells OpenVPN to listen on TCP port 7505 for management interface clients (port 7505 is an arbitrary choice -- you can use any free port).

Once OpenVPN is running, you can connect to the management interface using a **telnet** client. For example:

```
ai:~ # telnet localhost 7505
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^'.
>INFO:OpenVPN Management Interface Version 1 -- type 'help' for more info
help
Management Interface for OpenVPN 2.0_rc14 i686-suse-linux [SSL] [LZO] [EPOLL] built on Feb
Commands:
echo [on|off] [N|all] : Like log, but only show messages in echo buffer.
```

```

exit|quit          : Close management session.
help              : Print this message.
hold [on|off|release] : Set/show hold flag to on/off state, or
                    release current hold and start tunnel.

kill cn           : Kill the client instance(s) having common name cn.
kill IP:port      : Kill the client instance connecting from IP:port.
log [on|off] [N|all] : Turn on/off realtime log display
                    + show last N lines or 'all' for entire history.

mute [n]          : Set log mute level to n, or show level if n is absent.
net              : (Windows only) Show network info and routing table.
password type p    : Enter password p for a queried OpenVPN password.
signal s          : Send signal s to daemon,
                    s = SIGHUP|SIGTERM|SIGUSR1|SIGUSR2.

state [on|off] [N|all] : Like log, but show state history.
status [n]         : Show current daemon status info using format #n.
test n            : Produce n lines of output for testing/debugging.
username type u    : Enter username u for a queried OpenVPN username.
verb [n]          : Set log verbosity level to n, or show if n is absent.
version           : Show current version number.
END
exit
Connection closed by foreign host.
ai:~ #

```

For more information, see the [OpenVPN Management Interface Documentation](#).

Expanding the scope of the VPN to include additional machines on either the client or server subnet.

Including multiple machines on the server side when using a routed VPN (dev tun)

Once the VPN is operational in a point-to-point capacity between client and server, it may be desirable to expand the scope of the VPN so that clients can reach multiple machines on the server network, rather than only the server machine itself.

For the purpose of this example, we will assume that the server-side LAN uses a subnet of **10.66.0.0/24** and the VPN IP address pool uses **10.8.0.0/24** as cited in the **server** directive in the OpenVPN server configuration file.

First, you must *advertise* the **10.66.0.0/24** subnet to VPN clients as being accessible through the VPN. This can easily be done with the following server-side config file directive:

```
push "route 10.66.0.0 255.255.255.0"
```

Next, you must set up a route on the server-side LAN gateway to route the VPN client subnet (**10.8.0.0/24**) to the OpenVPN server (this is only necessary if the OpenVPN server and the LAN gateway are different machines).

Make sure that you've enabled [IP](#) and [TUN/TAP](#) forwarding on the OpenVPN server machine.

Including multiple machines on the server side when using a bridged VPN (dev tap)

One of the benefits of using [ethernet bridging](#) is that you get this for free without needing any additional configuration.

Including multiple machines on the client side when using a routed VPN (dev tun)

In a typical road-warrior or remote access scenario, the client machine connects to the VPN as a single machine. But suppose the client machine is a gateway for a local LAN (such as a home office), and you would like each machine on the client LAN to be able to route through the VPN.

For this example, we will assume that the client LAN is using the **192.168.4.0/24** subnet, and that the VPN client is using a certificate with a common name of **client2**. Our goal is to set up the VPN so that any machine on the client LAN can communicate with any machine on the server LAN through the VPN.

Before setup, there are some basic prerequisites which must be followed:

- The client LAN subnet (192.168.4.0/24 in our example) must not be exported to the VPN by the server or any other client sites which are using the same subnet. Every subnet which is joined to the VPN via routing must be unique.
- The client must have a unique Common Name in its certificate ("client2" in our example), and the **duplicate-cn** flag must not be used in the OpenVPN server configuration file.

First, make sure that [IP](#) and [TUN/TAP](#) forwarding is enabled on the client machine.

Next, we will deal with the necessary configuration changes on the server side. If the server configuration file does not currently reference a client configuration directory, add one now:

```
client-config-dir ccd
```

In the above directive, **ccd** should be the name of a directory which has been pre-created in the default directory where the OpenVPN server daemon runs. On Linux this tends to be **/etc/openvpn** and on

Windows it is usually `\Program Files\OpenVPN\config`. When a new client connects to the OpenVPN server, the daemon will check this directory for a file which matches the common name of the connecting client. If a matching file is found, it will be read and processed for additional configuration file directives to be applied to the named client.

The next step is to create a file called **client2** in the **ccd** directory. This file should contain the line:

```
iroute 192.168.4.0 255.255.255.0
```

This will tell the OpenVPN server that the 192.168.4.0/24 subnet should be routed to **client2**.

Next, add the following line to the main server config file (not the **ccd/client2** file):

```
route 192.168.4.0 255.255.255.0
```

Why the redundant **route** and **iroute** statements, you might ask? The reason is that **route** controls the routing from the kernel to the OpenVPN server (via the TUN interface) while **iroute** controls the routing from the OpenVPN server to the remote clients. Both are necessary.

Next, ask yourself if you would like to allow network traffic between client2's subnet (192.168.4.0/24) and other clients of the OpenVPN server. If so, add the following to the server config file.

```
client-to-client
push "route 192.168.4.0 255.255.255.0"
```

This will cause the OpenVPN server to *advertise* client2's subnet to other connecting clients.

The last step, and one that is often forgotten, is to add a route to the server's LAN gateway which directs 192.168.4.0/24 to the OpenVPN server box (you won't need this if the OpenVPN server box *is* the gateway for the server LAN). Suppose you were missing this step and you tried to ping a machine (not the OpenVPN server itself) on the server LAN from 192.168.4.8? The outgoing ping would probably reach the machine, but then it wouldn't know how to route the ping reply, because it would have no idea how to reach 192.168.4.0/24. The rule of thumb to use is that when routing entire LANs through the VPN (when the VPN server is not the same machine as the LAN gateway), make sure that the gateway for the LAN routes all VPN subnets to the VPN server machine.

Similarly, if the client machine running OpenVPN is not also the gateway for the client LAN, then the gateway for the client LAN must have a route which directs all subnets which should be reachable through the VPN to the OpenVPN client machine.

Including multiple machines on the client side when using a bridged VPN (dev tap)

This requires a more complex setup (maybe not more complex in practice, but more complicated to explain in detail):

- You must bridge the client TAP interface with the LAN-connected NIC on the client.
- You must manually set the IP/netmask of the TAP interface on the client.
- You must configure client-side machines to use an IP/netmask that is inside of the bridged subnet, possibly by [querying a DHCP server on the OpenVPN server side of the VPN](#).

Pushing DHCP options to clients

The OpenVPN server can push DHCP options such as DNS and WINS server addresses to clients (some [caveats](#) to be aware of). Windows clients can accept pushed DHCP options natively, while non-Windows clients can accept them by using a client-side **up** script which parses the **foreign_option_n** environmental variable list. See the [man page](#) or [openvpn-users mailing list archive](#) for non-Windows **foreign_option_n** documentation and script examples.

For example, suppose you would like connecting clients to use an internal DNS server at 10.66.0.4 or 10.66.0.5 and a WINS server at 10.66.0.8. Add this to the OpenVPN server configuration:

```
push "dhcp-option DNS 10.66.0.4"
push "dhcp-option DNS 10.66.0.5"
push "dhcp-option WINS 10.66.0.8"
```

To test this feature on Windows, run the following from a command prompt window after the machine has connected to an OpenVPN server:

```
ipconfig /all
```

The entry for the TAP-Windows adapter should show the DHCP options which were pushed by the server.

Configuring client-specific rules and access policies

Suppose we are setting up a company VPN, and we would like to establish separate access policies for 3 different classes of users:

- **System administrators** -- full access to all machines on the network
- **Employees** -- access only to Samba/email server
- **Contractors** -- access to a special server only

The basic approach we will take is (a) segregate each user class into its own virtual IP address range,

and (b) control access to machines by setting up firewall rules which key off the client's virtual IP address.

In our example, suppose that we have a variable number of employees, but only one system administrator, and two contractors. Our IP allocation approach will be to put all employees into an IP address pool, and then allocate fixed IP addresses for the system administrator and contractors.

Note that one of the prerequisites of this example is that you have a software firewall running on the OpenVPN server machine which gives you the ability to define specific firewall rules. For our example, we will assume the firewall is Linux **iptables**.

First, let's create a virtual IP address map according to user class:

Class	Virtual IP Range	Allowed LAN Access	Common Names
Employees	10.8.0.0/24	Samba/email server at 10.66.4.4	[variable]
System Administrators	10.8.1.0/24	Entire 10.66.4.0/24 subnet	sysadmin1
Contractors	10.8.2.0/24	Contractor server at 10.66.4.12	contractor1, contractor2

Next, let's translate this map into an OpenVPN server configuration. First of all, make sure you've followed the steps [above](#) for making the 10.66.4.0/24 subnet available to all clients (while we will configure routing to allow client access to the entire 10.66.4.0/24 subnet, we will then impose access restrictions using firewall rules to implement the above policy table).

First, define a static unit number for our **tun** interface, so that we will be able to refer to it later in our firewall rules:

```
dev tun0
```

In the server configuration file, define the Employee IP address pool:

```
server 10.8.0.0 255.255.255.0
```

Add routes for the System Administrator and Contractor IP ranges:

```
route 10.8.1.0 255.255.255.0
route 10.8.2.0 255.255.255.0
```

Because we will be assigning fixed IP addresses for specific System Administrators and Contractors, we will use a client configuration directory:

```
client-config-dir ccd
```

Now place special configuration files in the **ccd** subdirectory to define the fixed IP address for each non-Employee VPN client.

ccd/sysadmin1

```
ifconfig-push 10.8.1.1 10.8.1.2
```

ccd/contractor1

```
ifconfig-push 10.8.2.1 10.8.2.2
```

ccd/contractor2

```
ifconfig-push 10.8.2.5 10.8.2.6
```

Each pair of **ifconfig-push** addresses represent the virtual client and server IP endpoints. They must be taken from successive /30 subnets in order to be compatible with Windows clients and the TAP-Windows driver. Specifically, the last octet in the IP address of each endpoint pair must be taken from this set:

```
[ 1, 2] [ 5, 6] [ 9, 10] [ 13, 14] [ 17, 18]
[ 21, 22] [ 25, 26] [ 29, 30] [ 33, 34] [ 37, 38]
[ 41, 42] [ 45, 46] [ 49, 50] [ 53, 54] [ 57, 58]
[ 61, 62] [ 65, 66] [ 69, 70] [ 73, 74] [ 77, 78]
[ 81, 82] [ 85, 86] [ 89, 90] [ 93, 94] [ 97, 98]
[101,102] [105,106] [109,110] [113,114] [117,118]
[121,122] [125,126] [129,130] [133,134] [137,138]
[141,142] [145,146] [149,150] [153,154] [157,158]
[161,162] [165,166] [169,170] [173,174] [177,178]
[181,182] [185,186] [189,190] [193,194] [197,198]
[201,202] [205,206] [209,210] [213,214] [217,218]
[221,222] [225,226] [229,230] [233,234] [237,238]
[241,242] [245,246] [249,250] [253,254]
```

This completes the OpenVPN configuration. The final step is to add firewall rules to finalize the access policy. For this example, we will use firewall rules in the Linux **iptables** syntax:

```
# Employee rule
iptables -A FORWARD -i tun0 -s 10.8.0.0/24 -d 10.66.4.4 -j ACCEPT

# Sysadmin rule
iptables -A FORWARD -i tun0 -s 10.8.1.0/24 -d 10.66.4.0/24 -j ACCEPT

# Contractor rule
iptables -A FORWARD -i tun0 -s 10.8.2.0/24 -d 10.66.4.12 -j ACCEPT
```

Using alternative authentication methods

OpenVPN 2.0 and later include a feature that allows the OpenVPN server to securely obtain a username and password from a connecting client, and to use that information as a basis for authenticating the client.

To use this authentication method, first add the **auth-user-pass** directive to the client configuration. It will direct the OpenVPN client to query the user for a username/password, passing it on to the server over the secure TLS channel.

Next, configure the server to use an authentication plugin, which may be a script, shared object, or DLL. The OpenVPN server will call the plugin every time a VPN client tries to connect, passing it the username/password entered on the client. The authentication plugin can control whether or not the OpenVPN server allows the client to connect by returning a failure (1) or success (0) value.

Using Script Plugins

Script plugins can be used by adding the **auth-user-pass-verify** directive to the server-side configuration file. For example:

```
auth-user-pass-verify auth-pam.pl via-file
```

will use the **auth-pam.pl** perl script to authenticate the username/password of connecting clients. See the description of **auth-user-pass-verify** in the [manual page](#) for more information.

The **auth-pam.pl** script is included in the OpenVPN source file distribution in the **sample-scripts** subdirectory. It will authenticate users on a Linux server using a PAM authentication module, which could in turn implement shadow password, RADIUS, or LDAP authentication. **auth-pam.pl** is primarily intended for demonstration purposes. For real-world PAM authentication, use the **openvpn-auth-pam** shared object plugin described below.

Using Shared Object or DLL Plugins

Shared object or DLL plugins are usually compiled C modules which are loaded by the OpenVPN server at run time. For example if you are using an RPM-based OpenVPN package on Linux, the **openvpn-auth-pam** plugin should be already built. To use it, add this to the server-side config file:

```
plugin /usr/share/openvpn/plugin/lib/openvpn-auth-pam.so login
```

This will tell the OpenVPN server to validate the username/password entered by clients using the **login** PAM module.

For real-world production use, it's better to use the **openvpn-auth-pam** plugin, because it has several advantages over the **auth-pam.pl** script:

- The shared object **openvpn-auth-pam** plugin uses a split-privilege execution model for better security. This means that the OpenVPN server can run with reduced privileges by using the directives **user nobody**, **group nobody**, and **chroot**, and will still be able to authenticate against the root-readable-only shadow password file.
- OpenVPN can pass the username/password to a plugin via virtual memory, rather than via a file or the environment, which is better for local security on the server machine.
- C-compiled plugin modules generally run faster than scripts.

If you would like more information on developing your own plugins for use with OpenVPN, see the **README** files in the **plugin** subdirectory of the OpenVPN source distribution.

To build the **openvpn-auth-pam** plugin on Linux, cd to the **plugin/auth-pam** directory in the OpenVPN source distribution and run **make**.

Using username/password authentication as the only form of client authentication

By default, using **auth-user-pass-verify** or a username/password-checking **plugin** on the server will enable dual authentication, requiring that both client-certificate and username/password authentication succeed in order for the client to be authenticated.

While it is discouraged from a security perspective, it is also possible to disable the use of client certificates, and force username/password authentication only. On the server:

```
client-cert-not-required
```

Such configurations should usually also set:

```
username-as-common-name
```

which will tell the server to use the username for indexing purposes as it would use the Common Name of a client which was authenticating via a client certificate.

Note that **client-cert-not-required** will not obviate the need for a server certificate, so a client connecting to a server which uses **client-cert-not-required** may remove the **cert** and **key** directives from the client configuration file, but not the **ca** directive, because it is necessary for the client to verify the server certificate.

How to add dual-factor authentication to an OpenVPN configuration using client-side smart cards

Also see Article: [The OpenVPN Smartcard HOWTO](#)

- [About dual-factor authentication](#)
- [What is PKCS#11?](#)
- [Finding PKCS#11 provider library.](#)
- [How to configure a cryptographic token](#)
- [How to modify an OpenVPN configuration to make use of cryptographic tokens](#)
 - [Determine the correct object.](#)
 - [Using OpenVPN with PKCS#11.](#)
 - [PKCS#11 implementation considerations.](#)
 - [OpenSC PKCS#11 provider.](#)
- [Difference between PKCS#11 and Microsoft Cryptographic API \(CryptoAPI\).](#)

About dual-factor authentication

Dual-factor authentication is a method of authentication that combines two elements: something you have and something you know.

Something you have should be a device that cannot be duplicated; such a device can be a cryptographic token that contains a private secret key. This private key is generated inside the device and never leaves it. If a user possessing this token attempts to access protected services on a remote network, the authorization process which grants or denies network access can establish, with a high degree of certainty, that the user seeking access is in physical possession of a known, certified token.

Something you know can be a password presented to the cryptographic device. Without presenting the proper password you cannot access the private secret key. Another feature of cryptographic devices is to prohibit the use of the private secret key if the wrong password had been presented more than an allowed number of times. This behavior ensures that if a user lost his device, it would be infeasible for another person to use it.

Cryptographic devices are commonly called "smart cards" or "tokens", and are used in conjunction with a PKI (Public Key Infrastructure). The VPN server can examine a X.509 certificate and verify that the user holds the corresponding private secret key. Since the device cannot be duplicated and requires a valid password, the server is able to authenticate the user with a high degree of confidence.

Dual-factor authentication is much stronger than password-based authentication, because in the worst-case scenario, only one person at a time can use the cryptographic token. Passwords can be guessed and can be exposed to other users, so in the worst-case scenario an infinite number of people could attempt to gain unauthorized access when resources are protected using password-only authentication.

If you store the secret private key in a file, the key is usually encrypted by a password. The problem with this approach is that the encrypted key is exposed to decryption attacks or spyware/malware running on the client machine. Unlike when using a cryptographic device, the file cannot erase itself automatically after several failed decryption attempts.

What is PKCS#11?

This standard specifies an API, called Cryptoki, to devices which hold cryptographic information and perform cryptographic functions. Cryptoki, pronounced "crypto-key" and short for cryptographic token interface, follows a simple object-based approach, addressing the goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting to applications a common, logical view of the device called a cryptographic token.

Source: RSA Security Inc. <https://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm>.

To summarize, PKCS#11 is a standard that can be used by application software to access cryptographic tokens such as smart cards and other devices. Most device vendors provide a library that implements the PKCS#11 provider interface -- this library can be used by applications in order to access these devices. PKCS#11 is a cross-platform, vendor-independent free standard.

Finding PKCS#11 provider library

The first thing you need to do is to find the provider library, it should be installed with the device drivers. Each vendor has its own library. For example, the OpenSC PKCS#11 provider is located at /usr/lib

/pkcs11/opensc-pkcs11.so on Unix or at opensc-pkcs11.dll on Windows.

How to configure cryptographic token

You should follow an enrollment procedure:

- Initialize the PKCS#11 token.
- Generate RSA key pair on the PKCS#11 token.
- Create a certificate request based on the key pair, you can use [OpenSC and OpenSSL](#) in order to do that.
- Submit the certificate request to a certificate authority, and receive a certificate.
- Load the certificate onto the token, while noting that the id and label attributes of the certificate must match those of the private key.

A configured token is a token that has a private key object and a certificate object, where both share the same id and label attributes.

A simple enrollment utility is Easy-RSA 2.0 which is part of OpenVPN 2.1 series. Follow the instructions specified in the README file, and then use the pktool in order to enroll.

Initialize a token using the following command:

```
$ ./pktool --pkcs11-slots /usr/lib/pkcs11/
$ ./pktool --pkcs11-init /usr/lib/pkcs11/
```

Enroll a certificate using the following command:

```
$ ./pktool --pkcs11 /usr/lib/pkcs11/ client1
```

How to modify an OpenVPN configuration to make use of cryptographic tokens

You should have OpenVPN 2.1 or above in order to use the PKCS#11 features.

Determine the correct object

Each PKCS#11 provider can support multiple devices. In order to view the available object list you can use the following command:

```
$ openvpn --show-pkcs11-ids /usr/lib/pkcs11/
```

The following objects are available for use.
Each object shown below may be used as parameter to
--pkcs11-id option please remember to use single quote mark.

```
Certificate
      DN: /CN=User1
      Serial: 490B82C4000000000075
      Serialized id: aaaa/bbb/41545F5349474E415455524581D2A1A1B23C4AA4CB17FAF7A4600
```

Each certificate/private key pair have unique "Serialized id" string. The serialized id string of the requested certificate should be specified to the **pkcs11-id** option using single quote marks.

```
pkcs11-id 'aaa/bbb/41545F5349474E415455524581D2A1A1B23C4AA4CB17FAF7A4600'
```

Using OpenVPN with PKCS#11

A typical set of OpenVPN options for PKCS#11

```
pkcs11-providers /usr/lib/pkcs11/
pkcs11-id 'aaa/bbb/41545F5349474E415455524581D2A1A1B23C4AA4CB17FAF7A4600'
```

This will select the object which matches the pkcs11-id string.

Advanced OpenVPN options for PKCS#11

```
pkcs11-providers /usr/lib/pkcs11/provider1.so /usr/lib/pkcs11/provider2.so
pkcs11-id 'aaa/bbb/41545F5349474E415455524581D2A1A1B23C4AA4CB17FAF7A4600'
pkcs11-pin-cache 300
daemon
auth-retry nointeract
management-hold
management-signal
management 127.0.0.1 8888
management-query-passwords
```

This will load two providers into OpenVPN, use the certificate specified on **pkcs11-id** option, and use the

management interface in order to query passwords. The daemon will resume into hold state on the event when token cannot be accessed. The token will be used for 300 seconds after which the password will be re-queried, session will disconnect if management session disconnects.

PKCS#11 implementation considerations

Many PKCS#11 providers make use of threads, in order to avoid problems caused by implementation of LinuxThreads (setuid, chroot), it is highly recommend to upgrade to Native POSIX Thread Library (NPTL) enabled glibc if you intend to use PKCS#11.

OpenSC PKCS#11 provider

OpenSC PKCS#11 provider is located at /usr/lib/pkcs11/opensc-pkcs11.so on Unix or at opensc-pkcs11.dll on Windows.

Difference between PKCS#11 and Microsoft Cryptographic API (CryptoAPI)

PKCS#11 is a free, cross-platform vendor independent standard. CryptoAPI is a Microsoft specific API. Most smart card vendors provide support for both interfaces. In the Windows environment, the user should select which interface to use.

The current implementation of OpenVPN that uses the MS CryptoAPI (**cryptoapicert** option) works well as long as you don't run OpenVPN as a service. If you wish to run OpenVPN in an administrative environment using a service, the implementation will not work with most smart cards because of the following reasons:

- Most smart card providers do not load certificates into the local machine store, so the implementation will be unable to access the user certificate.
- If the OpenVPN client is running as a service without direct interaction with the end-user, the service cannot query the user to provide a password for the smart card, causing the password-verification process on the smart card to fail.

Using the PKCS#11 interface, you can use smart cards with OpenVPN in any implementation, since PKCS#11 does not access Microsoft stores and does not necessarily require direct interaction with the end-user.

Routing all client traffic (including web-traffic) through the VPN

Overview

By default, when an OpenVPN client is active, only network traffic to and from the OpenVPN server site will pass over the VPN. General web browsing, for example, will be accomplished with direct connections that bypass the VPN.

In certain cases this behavior might not be desirable -- you might want a VPN client to tunnel all network traffic through the VPN, including general internet web browsing. While this type of VPN configuration will exact a performance penalty on the client, it gives the VPN administrator more control over security policies when a client is simultaneously connected to both the public internet and the VPN at the same time.

Implementation

Add the following directive to the server configuration file:

```
push "redirect-gateway def1"
```

If your VPN setup is over a wireless network, where all clients and the server are on the same wireless subnet, add the **local** flag:

```
push "redirect-gateway local def1"
```

Pushing the **redirect-gateway** option to clients will cause all IP network traffic originating on client machines to pass through the OpenVPN server. The server will need to be configured to deal with this traffic somehow, such as by NATing it to the internet, or routing it through the server site's HTTP proxy.

On Linux, you could use a command such as this to NAT the VPN client traffic to the internet:

```
iptables -t nat -A POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE
```

This command assumes that the VPN subnet is **10.8.0.0/24** (taken from the **server** directive in the OpenVPN server configuration) and that the local ethernet interface is **eth0**.

When **redirect-gateway** is used, OpenVPN clients will route DNS queries through the VPN, and the VPN server will need handle them. This can be accomplished by pushing a DNS server address to connecting clients which will replace their normal DNS server settings during the time that the VPN is active. For example:

```
push "dhcp-option DNS 10.8.0.1"
```

will configure Windows clients (or non-Windows clients with some extra server-side scripting) to use 10.8.0.1 as their DNS server. Any address which is reachable from clients may be used as the DNS server address.

Caveats

Redirecting all network traffic through the VPN is not entirely a problem-free proposition. Here are some typical gotchas to be aware of:

- Many OpenVPN client machines connecting to the internet will periodically interact with a DHCP server to renew their IP address leases. The **redirect-gateway** option might prevent the client from reaching the local DHCP server (because DHCP messages would be routed over the VPN), causing it to lose its IP address lease.
- [Issues exist](#) with respect to pushing DNS addresses to Windows clients.
- Web browsing performance on the client will be noticeably slower.

For more information on the mechanics of the **redirect-gateway** directive, see the [manual page](#).

Running an OpenVPN server on a dynamic IP address

While OpenVPN clients can easily access the server via a dynamic IP address without any special configuration, things get more interesting when the server itself is on a dynamic address. While OpenVPN has no trouble handling the situation of a dynamic server, some extra configuration is required.

The first step is to get a dynamic DNS address which can be configured to "follow" the server every time the server's IP address changes. There are several dynamic DNS service providers available, such as [dyndns.org](#).

The next step is to set up a mechanism so that every time the server's IP address changes, the dynamic DNS name will be quickly updated with the new IP address, allowing clients to find the server at its new IP address. There are two basic ways to accomplish this:

- Use a NAT router appliance with dynamic DNS support (such as the **Linksys BEFSR41**). Most of the inexpensive NAT router appliances that are widely available have the capability to update a dynamic DNS name every time a new DHCP lease is obtained from the ISP. This setup is ideal when the OpenVPN server box is a single-NIC machine inside the firewall.
- Use a dynamic DNS client application such as [ddclient](#) to update the dynamic DNS address whenever the server IP address changes. This setup is ideal when the machine running OpenVPN has multiple NICs and is acting as a site-wide firewall/gateway. To implement this setup, you need to set up a script to be run by your DHCP client software every time an IP address change occurs. This script should (a) run **ddclient** to notify your dynamic DNS provider of your new IP address and (b) restart the OpenVPN server daemon.

The OpenVPN client by default will sense when the server's IP address has changed, if the client configuration is using a **remote** directive which references a dynamic DNS name. The usual chain of events is that (a) the OpenVPN client fails to receive timely keepalive messages from the server's old IP address, triggering a restart, and (b) the restart causes the DNS name in the **remote** directive to be re-resolved, allowing the client to reconnect to the server at its new IP address.

More information can be found in the [FAQ](#).

Connecting to an OpenVPN server via an HTTP proxy.

OpenVPN supports connections through an HTTP proxy, with the following authentication modes:

- No proxy authentication
- Basic proxy authentication
- NTLM proxy authentication

First of all, HTTP proxy usage requires that you use TCP as the tunnel carrier protocol. So add the following to both client and server configurations:

```
proto tcp
```

Make sure that any **proto udp** lines in the config files are deleted.

Next, add the **http-proxy** directive to the client configuration file (see the [manual page](#) for a full description of this directive).

For example, suppose you have an HTTP proxy server on the client LAN at **192.168.4.1**, which is listening for connections on port **1080**. Add this to the client config:

```
http-proxy 192.168.4.1 1080
```

Suppose the HTTP proxy requires Basic authentication:

```
http-proxy 192.168.4.1 1080 stdin basic
```

Suppose the HTTP proxy requires NTLM authentication:

```
http-proxy 192.168.4.1 1080 stdin ntlm
```

The two authentication examples above will cause OpenVPN to prompt for a username/password from standard input. If you would instead like to place these credentials in a file, replace **stdin** with a filename, and place the username on line 1 of this file and the password on line 2.

Connecting to a Samba share over OpenVPN

This example is intended show how OpenVPN clients can connect to a Samba share over a routed **dev tun** tunnel. If you are ethernet bridging (**dev tap**), you probably don't need to follow these instructions, as OpenVPN clients should see server-side machines in their network neighborhood.

For this example, we will assume that:

- the server-side LAN uses a subnet of **10.66.0.0/24**,
- the VPN IP address pool uses **10.8.0.0/24** (as cited in the **server** directive in the OpenVPN server configuration file),
- the Samba server has an IP address of **10.66.0.4**, and
- the Samba server has already been configured and is reachable from the local LAN.

If the Samba and OpenVPN servers are running on different machines, make sure you've followed the section on [expanding the scope of the VPN to include additional machines](#).

Next, edit your Samba configuration file (**smb.conf**). Make sure the **hosts allow** directive will permit OpenVPN clients coming from the **10.8.0.0/24** subnet to connect. For example:

```
hosts allow = 10.66.0.0/24 10.8.0.0/24 127.0.0.1
```

If you are running the Samba and OpenVPN servers on the same machine, you may want to edit the **interfaces** directive in the **smb.conf** file to also listen on the TUN interface subnet of **10.8.0.0/24**:

```
interfaces = 10.66.0.0/24 10.8.0.0/24
```

If you are running the Samba and OpenVPN servers on the same machine, connect from an OpenVPN client to a Samba share using the folder name:

```
\\10.8.0.1\\sharename
```

If the Samba and OpenVPN servers are on different machines, use folder name:

```
\\10.66.0.4\\sharename
```

For example, from a command prompt window:

```
net use z: \\10.66.0.4\\sharename /USER:myusername
```

Implementing a load-balancing/failover configuration

Client

The OpenVPN client configuration can refer to multiple servers for load balancing and failover. For example:

```
remote server1.mydomain
remote server2.mydomain
remote server3.mydomain
```

will direct the OpenVPN client to attempt a connection with server1, server2, and server3 in that order. If an existing connection is broken, the OpenVPN client will retry the most recently connected server, and if that fails, will move on to the next server in the list. You can also direct the OpenVPN client to randomize its server list on startup, so that the client load will be probabilistically spread across the server pool.

```
remote-random
```

If you would also like DNS resolution failures to cause the OpenVPN client to move to the next server in the list, add the following:

```
resolv-retry 60
```

The **60** parameter tells the OpenVPN client to try resolving each **remote** DNS name for 60 seconds before moving on to the next server in the list.

The server list can also refer to multiple OpenVPN server daemons running on the same machine, each listening for connections on a different port, for example:

```
remote smp-server1.mydomain 8000
remote smp-server1.mydomain 8001
remote smp-server2.mydomain 8000
remote smp-server2.mydomain 8001
```

If your servers are multi-processor machines, running multiple OpenVPN daemons on each server can be advantageous from a performance standpoint.

OpenVPN also supports the **remote** directive referring to a DNS name which has multiple **A** records in

the zone configuration for the domain. In this case, the OpenVPN client will randomly choose one of the **A** records every time the domain is resolved.

Server

The simplest approach to a load-balanced/failover configuration on the server is to use equivalent configuration files on each server in the cluster, except use a different virtual IP address pool for each server. For example:

server1

```
server 10.8.0.0 255.255.255.0
```

server2

```
server 10.8.1.0 255.255.255.0
```

server3

```
server 10.8.2.0 255.255.255.0
```

Hardening OpenVPN Security

One of the often-repeated maxims of network security is that one should never place so much trust in a single security component that its failure causes a catastrophic security breach. OpenVPN provides several mechanisms to add additional security layers to hedge against such an outcome.

tls-auth

The **tls-auth** directive adds an additional HMAC signature to all SSL/TLS handshake packets for integrity verification. Any UDP packet not bearing the correct HMAC signature can be dropped without further processing. The **tls-auth** HMAC signature provides an additional level of security above and beyond that provided by SSL/TLS. It can protect against:

- DoS attacks or port flooding on the OpenVPN UDP port.
- Port scanning to determine which server UDP ports are in a listening state.
- Buffer overflow vulnerabilities in the SSL/TLS implementation.
- SSL/TLS handshake initiations from unauthorized machines (while such handshakes would ultimately fail to authenticate, **tls-auth** can cut them off at a much earlier point).

Using **tls-auth** requires that you generate a shared-secret key that is used in addition to the standard RSA certificate/key:

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

This command will generate an OpenVPN static key and write it to the file **ta.key**. This key should be copied over a pre-existing secure channel to the server and all client machines. It can be placed in the same directory as the RSA **.key** and **.crt** files.

In the server configuration, add:

```
tls-auth ta.key 0
```

In the client configuration, add:

```
tls-auth ta.key 1
```

proto udp

While OpenVPN allows either the TCP or UDP protocol to be used as the VPN carrier connection, the UDP protocol will provide better protection against DoS attacks and port scanning than TCP:

```
proto udp
```

user/group (non-Windows only)

OpenVPN has been very carefully designed to allow root privileges to be dropped after initialization, and this feature should always be used on Linux/BSD/Solaris. Without root privileges, a running OpenVPN server daemon provides a far less enticing target to an attacker.

```
user nobody
group nobody
```

Unprivileged mode (Linux only)

On Linux OpenVPN can be run completely unprivileged. This configuration is a little more complex, but provides best security.

In order to work with this configuration, OpenVPN must be configured to use `iproute` interface, this is done by specifying `--enable-iproute2` to configure script. `sudo` package should also be available on your system.

This configuration uses the Linux ability to change the permission of a `tun` device, so that unprivileged user may access it. It also uses `sudo` in order to execute `iproute` so that interface properties and routing

table may be modified.

OpenVPN configuration:

- Write the following script and place it at: `/usr/local/sbin/unpriv-ip`:

```
#!/bin/sh
sudo /sbin/ip $*
```

- Execute visudo, and add the followings to allow user 'user1' to execute `/sbin/ip`:

```
user1 ALL=(ALL) NOPASSWD: /sbin/ip
```

You can also enable a group of users with the following command:

```
%users ALL=(ALL) NOPASSWD: /sbin/ip
```

- Add the following to your OpenVPN configuration:

```
dev tunX/tapX
iproute /usr/local/sbin/unpriv-ip
```

Please note that you must select constant X and specify tun or tap not both.

- As root add persistent interface, and permit user and/or group to manage it, the following create tunX (replace with your own) and allow user1 and group users to access it.

```
openvpn --mktun --dev tunX --type tun --user user1 --group users
```

- Run OpenVPN in the context of the unprivileged user.

Further security constraints may be added by examining the parameters at the `/usr/local/sbin/unpriv-ip` script.

chroot (non-Windows only)

The **chroot** directive allows you to lock the OpenVPN daemon into a so-called *chroot jail*, where the daemon would not be able to access any part of the host system's filesystem except for the specific directory given as a parameter to the directive. For example,

```
chroot jail
```

would cause the OpenVPN daemon to cd into the **jail** subdirectory on initialization, and would then reorient its root filesystem to this directory so that it would be impossible thereafter for the daemon to access any files outside of **jail** and its subdirectory tree. This is important from a security perspective, because even if an attacker were able to compromise the server with a code insertion exploit, the exploit would be locked out of most of the server's filesystem.

Caveats: because **chroot** reorients the filesystem (from the perspective of the daemon only), it is necessary to place any files which OpenVPN might need after initialization in the **jail** directory, such as:

- the **ctrl-verify** file, or
- the **client-config-dir** directory.

Larger RSA keys

The RSA key size is controlled by the **KEY_SIZE** variable in the **easy-rsa/vars** file, which must be set before any keys are generated. Currently set to 1024 by default, this value can reasonably be increased to 2048 with no negative impact on VPN tunnel performance, except for a slightly slower SSL/TLS renegotiation handshake which occurs once per client per hour, and a much slower one-time Diffie Hellman parameters generation process using the **easy-rsa/build-dh** script.

Larger symmetric keys

By default OpenVPN uses **Blowfish**, a 128 bit symmetrical cipher.

OpenVPN automatically supports any cipher which is supported by the OpenSSL library, and as such can support ciphers which use large key sizes. For example, the 256-bit version of AES (Advanced Encryption Standard) can be used by adding the following to both server and client configuration files:

```
cipher AES-256-CBC
```

Keep the root key (**ca.key**) on a standalone machine without a network connection

One of the security benefits of using an X509 PKI (as OpenVPN does) is that the root CA key (**ca.key**) need not be present on the OpenVPN server machine. In a high security environment, you might want to specially designate a machine for key signing purposes, keep the machine well-protected physically, and disconnect it from all networks. Floppy disks can be used to move key files back and forth, as necessary. Such measures make it extremely difficult for an attacker to steal the root key, short of physical theft of the key signing machine.

Revoking Certificates

Revoking a certificate means to invalidate a previously signed certificate so that it can no longer be used for authentication purposes.

Typical reasons for wanting to revoke a certificate include:

- The private key associated with the certificate is compromised or stolen.
- The user of an encrypted private key forgets the password on the key.
- You want to terminate a VPN user's access.

Example

As an example, we will revoke the **client2** certificate, which we generated above in the "key generation" section of the HOWTO.

First open up a shell or command prompt window and cd to the **easy-rsa** directory as you did in the "key generation" section above. On Linux/BSD/Unix:

```
./vars
./revoke-full client2
```

On Windows:

```
vars
revoke-full client2
```

You should see output similar to this:

```
Using configuration from /root/openvpn/20/openvpn/tmp/easy-rsa/openssl.cnf
DEBUG[load_index]: unique_subject = "yes"
Revoking Certificate 04.
Data Base Updated
Using configuration from /root/openvpn/20/openvpn/tmp/easy-rsa/openssl.cnf
DEBUG[load_index]: unique_subject = "yes"
client2.crt: /C=KG/ST=NA/O=OpenVPN-TEST/CN=client2/emailAddress=me@myhost.mydomain
error 23 at 0 depth lookup:certificate revoked
```

Note the "error 23" in the last line. That is what you want to see, as it indicates that a certificate verification of the revoked certificate failed.

The **revoke-full** script will generate a CRL (certificate revocation list) file called **crl.pem** in the **keys** subdirectory. The file should be copied to a directory where the OpenVPN server can access it, then CRL verification should be enabled in the server configuration:

```
crl-verify crl.pem
```

Now all connecting clients will have their client certificates verified against the CRL, and any positive match will result in the connection being dropped.

CRL Notes

- When the **crl-verify** option is used in OpenVPN, the CRL file will be re-read any time a new client connects or an existing client renegotiates the SSL/TLS connection (by default once per hour). This means that you can update the CRL file while the OpenVPN server daemon is running, and have the new CRL take effect immediately for newly connecting clients. If the client whose certificate you are revoking is already connected, you can restart the server via a signal (SIGUSR1 or SIGHUP) and flush all clients, or you can telnet to the [management interface](#) and explicitly kill the specific client instance object on the server without disturbing other clients.
- While the **crl-verify** directive can be used on both the OpenVPN server and clients, it is generally unnecessary to distribute a CRL file to clients unless a server certificate has been revoked. Clients don't need to know about other client certificates which have been revoked because [clients shouldn't be accepting direct connections from other clients](#) in the first place.
- The CRL file is not secret, and should be made world-readable so that the OpenVPN daemon can read it after root privileges have been dropped.
- If you are using the **chroot** directive, make sure to put a copy of the CRL file in the chroot directory, since unlike most other files which OpenVPN reads, the CRL file will be read after the chroot call is executed, not before.
- A common reason why certificates need to be revoked is that the user encrypts their private key with a password, then forgets the password. By revoking the original certificate, it is possible to generate a new certificate/key pair with the user's original common name.

Important Note on possible "Man-in-the-Middle" attack if clients do not verify

the certificate of the server they are connecting to.

To avoid a possible Man-in-the-Middle attack where an authorized client tries to connect to another client by impersonating the server, make sure to enforce some kind of server certificate verification by clients. There are currently five different ways of accomplishing this, listed in the order of preference:

- *[OpenVPN 2.1 and above]* Build your server certificates with specific key usage and extended key usage. The RFC3280 determine that the following attributes should be provided for TLS connections:

Mode	Key usage	Extended key usage
Client	digitalSignature	TLS Web Client Authentication
	keyAgreement	
	digitalSignature, keyAgreement	
Server	digitalSignature, keyEncipherment	TLS Web Server Authentication
	digitalSignature, keyAgreement	

You can build your server certificates with the **build-key-server** script (see the [easy-rsa](#) documentation for more info). This will designate the certificate as a server-only certificate by setting the right attributes. Now add the following line to your client configuration:

```
remote-cert-tls server
```

- *[OpenVPN 2.0 and below]* Build your server certificates with the **build-key-server** script (see the [easy-rsa](#) documentation for more info). This will designate the certificate as a server-only certificate by setting **nsCertType=server**. Now add the following line to your client configuration:

```
ns-cert-type server
```

This will block clients from connecting to any server which lacks the **nsCertType=server** designation in its certificate, even if the certificate has been signed by the **ca** file in the OpenVPN configuration file.

- Use the **tls-remote** directive on the client to accept/reject the server connection based on the common name of the server certificate.
- Use a **tls-verify** script or plugin to accept/reject the server connection based on a custom test of the server certificate's embedded X509 subject details.
- Sign server certificates with one CA and client certificates with a different CA. The client configuration **ca** directive should reference the server-signing CA file, while the server configuration **ca** directive should reference the client-signing CA file.

Try our new [VPN Service](https://www.privatetunnel.com) <https://www.privatetunnel.com>

Copyright © 2002-2013 by OpenVPN Technologies, Inc. < info@openvpn.net >. OpenVPN is a trademark of OpenVPN Technologies, Inc.

[Privacy Policy](#)[Terms of Use](#)[About](#)[Jobs](#)[News](#)[Contact](#)[Partners/Clients](#)[Support](#)

© 2002-2018 OpenVPN Inc.

OpenVPN is a registered trademark of OpenVPN Inc.