NAME

openvpn − Secure IP tunnel daemon

SYNOPSIS

openvpn [options ...]
openvpn −−help

INTRODUCTION

OpenVPN is an open source VPN daemon by James Yonan. Because OpenVPN tries to be a universal VPN tool offering a great deal of flexibility, there are a lot of options on this manual page. If you're new to OpenVPN, you might want to skip ahead to the examples section where you will see how to construct simple VPNs on the command line without even needing a configuration file.

Also note that there's more documentation and examples on the OpenVPN web site: *https://openvpn.net/*

And if you would like to see a shorter version of this manual, see the openvpn usage message which can be obtained by running **openvpn** without any parameters.

DESCRIPTION

OpenVPN is a robust and highly flexible VPN daemon. OpenVPN supports SSL/TLS security, ethernet bridging, TCP or UDP tunnel transport through proxies or NAT, support for dynamic IP addresses and DHCP, scalability to hundreds or thousands of users, and portability to most major OS platforms.

OpenVPN is tightly bound to the OpenSSL library, and derives much of its crypto capabilities from it.

OpenVPN supports conventional encryption using a pre−shared secret key **(Static Key mode)** or public key security **(SSL/TLS mode)** using client & server certificates. OpenVPN also supports non−encrypted TCP/UDP tunnels.

OpenVPN is designed to work with the **TUN/TAP** virtual networking interface that exists on most platforms.

Overall, OpenVPN aims to offer many of the key features of IPSec but with a relatively lightweight footprint.

OPTIONS

OpenVPN allows any option to be placed either on the command line or in a configuration file. Though all command line options are preceded by a double−leading−dash ("−−"), this prefix can be removed when an option is placed in a configuration file.

Generic Options

This section covers generic options which are accessible regardless of which mode OpenVPN is configured as.

−−help Show options.

−−auth−nocache

Don't cache **−−askpass** or **−−auth−user−pass** username/passwords in virtual memory.

If specified, this directive will cause OpenVPN to immediately forget username/password inputs after they are used. As a result, when OpenVPN needs a username/password, it will prompt for input from stdin, which may be multiple times during the duration of an OpenVPN session.

When using **−−auth−nocache** in combination with a user/password file and **−−chroot** or **−−daemon**, make sure to use an absolute path.

This directive does not affect the **−−http−proxy** username/password. It is always cached.

−−cd *dir*

Change directory to **dir** prior to reading any files such as configuration files, key files, scripts, etc. **dir** should be an absolute path, with a leading "/", and without any references to the current directory such as **.** or **..**.

This option is useful when you are running OpenVPN in **−−daemon** mode, and you want to consolidate all of your OpenVPN control files in one location.

−−chroot *dir*

Chroot to **dir** after initialization. **−−chroot** essentially redefines **dir** as being the top level directory tree (/). OpenVPN will therefore be unable to access any files outside this tree. This can be desirable from a security standpoint.

Since the chroot operation is delayed until after initialization, most OpenVPN options that reference files will operate in a pre−chroot context.

In many cases, the **dir** parameter can point to an empty directory, however complications can result when scripts or restarts are executed after the chroot operation.

Note: The SSL library will probably need /dev/urandom to be available inside the chroot directory **dir**. This is because SSL libraries occasionally need to collect fresh random. Newer linux kernels and some BSDs implement a getrandom() or getentropy() syscall that removes the need for /dev/urandom to be available.

−−config *file*

Load additional config options from **file** where each line corresponds to one command line option, but with the leading '−−' removed.

If **−−config file** is the only option to the openvpn command, the **−−config** can be removed, and the command can be given as **openvpn file**

Note that configuration files can be nested to a reasonable depth.

Double quotation or single quotation characters ("", ') can be used to enclose single parameters containing whitespace, and "#" or ";" characters in the first column can be used to denote comments.

Note that OpenVPN 2.0 and higher performs backslash−based shell escaping for characters not in single quotations, so the following mappings should be observed:

```
\\       Maps to a single backslash character (\).
\"       Pass a literal doublequote character ("), don't
         interpret it as enclosing a parameter.
\[SPACE] Pass a literal space or tab character, don't
         interpret it as a parameter delimiter.
```

For example on Windows, use double backslashes to represent pathnames:

```
secret "c:\\OpenVPN\\secret.key"
```

For examples of configuration files, see *https://openvpn.net/community−resources/how−to/*

Here is an example configuration file:

```
#
# Sample OpenVPN configuration file for
```

```
# using a pre-shared static key.
#
# '#' or ';' may be used to delimit comments.

# Use a dynamic tun device.
dev tun

# Our remote peer
remote mypeer.mydomain

# 10.1.0.1 is our local VPN endpoint
# 10.1.0.2 is our remote VPN endpoint
ifconfig 10.1.0.1 10.1.0.2

# Our pre-shared static key
secret static.key
```

--daemon *progname*

Become a daemon after all initialization functions are completed. This option will cause all message and error output to be sent to the syslog file (such as **/var/log/messages**), except for the output of scripts and ifconfig commands, which will go to **/dev/null** unless otherwise redirected. The syslog redirection occurs immediately at the point that **--daemon** is parsed on the command line even though the daemonization point occurs later. If one of the **--log** options is present, it will supersede syslog redirection.

The optional **progname** parameter will cause OpenVPN to report its program name to the system logger as **progname**. This can be useful in linking OpenVPN messages in the syslog file with specific tunnels. When unspecified, **progname** defaults to "openvpn".

When OpenVPN is run with the **--daemon** option, it will try to delay daemonization until the majority of initialization functions which are capable of generating fatal errors are complete. This means that initialization scripts can test the return status of the openvpn command for a fairly reliable indication of whether the command has correctly initialized and entered the packet forwarding event loop.

In OpenVPN, the vast majority of errors which occur after initialization are non-fatal.

Note: as soon as OpenVPN has daemonized, it can not ask for usernames, passwords, or key pass phrases anymore. This has certain consequences, namely that using a password-protected private key will fail unless the **--askpass** option is used to tell OpenVPN to ask for the pass phrase (this requirement is new in v2.3.7, and is a consequence of calling daemon() before initializing the crypto layer).

Further, using **--daemon** together with **--auth-user-pass** (entered on console) and **--auth-nocache** will fail as soon as key renegotiation (and reauthentication) occurs.

--disable-occ

Don't output a warning message if option inconsistencies are detected between peers. An example of an option inconsistency would be where one peer uses **--dev tun** while the other peer uses **--dev tap**.

Use of this option is discouraged, but is provided as a temporary fix in situations where a recent version of OpenVPN must connect to an old version.

−−engine *engine−name*
Enable OpenSSL hardware−based crypto engine functionality.

If **engine−name** is specified, use a specific crypto engine. Use the **−−show−engines** standalone option to list the crypto engines which are supported by OpenSSL.

−−fast−io
(Experimental) Optimize TUN/TAP/UDP I/O writes by avoiding a call to poll/epoll/select prior to the write operation. The purpose of such a call would normally be to block until the device or socket is ready to accept the write. Such blocking is unnecessary on some platforms which don't support write blocking on UDP sockets or TUN/TAP devices. In such cases, one can optimize the event loop by avoiding the poll/epoll/select call, improving CPU efficiency by 5% to 10%.

This option can only be used on non−Windows systems, when **−−proto udp** is specified, and when **−−shaper** is NOT specified.

−−group *group*
Similar to the **−−user** option, this option changes the group ID of the OpenVPN process to **group** after initialization.

−−ignore−unknown−option *args*
Valid syntax:

```
ignore−unknown−options opt1 opt2 opt3 ... optN
```

When one of options **opt1 ... optN** is encountered in the configuration file the configuration file parsing does not fail if this OpenVPN version does not support the option. Multiple **−−ignore−unknown−option** options can be given to support a larger number of options to ignore.

This option should be used with caution, as there are good security reasons for having OpenVPN fail if it detects problems in a config file. Having said that, there are valid reasons for wanting new software features to gracefully degrade when encountered by older software versions.

−−ignore−unknown−option is available since OpenVPN 2.3.3.

−−iproute *cmd*
Set alternate command to execute instead of default **iproute2** command. May be used in order to execute OpenVPN in unprivileged environment.

−−keying−material−exporter *args*
Save Exported Keying Material [RFC5705] of len bytes (must be between 16 and 4095 bytes) using **label** in environment (**exported_keying_material**) for use by plugins in **OPENVPN_PLUGIN_TLS_FINAL** callback.

Valid syntax:

```
keying−material−exporter label len
```

Note that exporter **labels** have the potential to collide with existing PRF labels. In order to prevent this, labels *MUST* begin with **EXPORTER**.

−−mlock
Disable paging by calling the POSIX mlockall function. Requires that OpenVPN be initially run as root (though OpenVPN can subsequently downgrade its UID using the **−−user** option).

Using this option ensures that key material and tunnel data are never written to disk due to virtual memory paging operations which occur under most modern operating systems. It ensures that even if an attacker was able to crack the box running OpenVPN, he would not be able to scan the

system swap file to recover previously used ephemeral keys, which are used for a period of time governed by the **−−reneg** options (see below), then are discarded.

The downside of using **−−mlock** is that it will reduce the amount of physical memory available to other applications.

The limit on how much memory can be locked and how that limit is enforced are OS−dependent. On Linux the default limit that an unprivileged process may lock (RLIMIT_MEMLOCK) is low, and if privileges are dropped later, future memory allocations will very likely fail. The limit can be increased using ulimit or systemd directives depending on how OpenVPN is started.

−−nice *n*
Change process priority after initialization (**n** greater than 0 is lower priority, **n** less than zero is higher priority).

−−persist−key
Don't re−read key files across **SIGUSR1** or **−−ping−restart**.

This option can be combined with **−−user nobody** to allow restarts triggered by the **SIGUSR1** signal. Normally if you drop root privileges in OpenVPN, the daemon cannot be restarted since it will now be unable to re−read protected key files.

This option solves the problem by persisting keys across **SIGUSR1** resets, so they don't need to be re−read.

−−remap−usr1 *signal*
Control whether internally or externally generated **SIGUSR1** signals are remapped to **SIGHUP** (restart without persisting state) or SIGTERM (exit).

signal can be set to **SIGHUP** or **SIGTERM**. By default, no remapping occurs.

−−script−security *level*
This directive offers policy−level control over OpenVPN's usage of external programs and scripts. Lower **level** values are more restrictive, higher values are more permissive. Settings for **level**:

0 Strictly no calling of external programs.

1 (Default) Only call built−in executables such as ifconfig, ip, route, or netsh.

2 Allow calling of built−in executables and user−defined scripts.

3 Allow passwords to be passed to scripts via environmental variables (potentially unsafe).

OpenVPN releases before v2.3 also supported a **method** flag which indicated how OpenVPN should call external commands and scripts. This could be either **execve** or **system**. As of OpenVPN 2.3, this flag is no longer accepted. In most *nix environments the execve() approach has been used without any issues.

Some directives such as **−−up** allow options to be passed to the external script. In these cases make sure the script name does not contain any spaces or the configuration parser will choke because it can't determine where the script name ends and script options start.

To run scripts in Windows in earlier OpenVPN versions you needed to either add a full path to the script interpreter which can parse the script or use the **system** flag to run these scripts. As of OpenVPN 2.3 it is now a strict requirement to have full path to the script interpreter when running non−executables files. This is not needed for executable files, such as .exe, .com, .bat or .cmd files. For example, if you have a Visual Basic script, you must use this syntax now:

```
−−up 'C:\\Windows\\System32\\wscript.exe C:\\Program\ Files\\OpenVPN\\cor
```

Please note the single quote marks and the escaping of the backslashes (\) and the space character.

The reason the support for the **system** flag was removed is due to the security implications with shell expansions when executing scripts via the **system()** call.

−−setcon *context*
Apply SELinux **context** after initialization. This essentially provides the ability to restrict OpenVPN's rights to only network I/O operations, thanks to SELinux. This goes further than **−−user** and **−−chroot** in that those two, while being great security features, unfortunately do not protect against privilege escalation by exploitation of a vulnerable system call. You can of course combine all three, but please note that since setcon requires access to /proc you will have to provide it inside the chroot directory (e.g. with mount −−bind).

Since the setcon operation is delayed until after initialization, OpenVPN can be restricted to just network−related system calls, whereas by applying the context before startup (such as the OpenVPN one provided in the SELinux Reference Policies) you will have to allow many things required only during initialization.

Like with chroot, complications can result when scripts or restarts are executed after the setcon operation, which is why you should really consider using the **−−persist−key** and **−−persist−tun** options.

−−status *args*
Write operational status to **file** every **n** seconds.

Valid syntaxes:

```
status file
status file n
```

Status can also be written to the syslog by sending a **SIGUSR2** signal.

With multi−client capability enabled on a server, the status file includes a list of clients and a routing table. The output format can be controlled by the **−−status−version** option in that case.

For clients or instances running in point−to−point mode, it will contain the traffic statistics.

−−status−version *n*
Set the status file format version number to **n**.

This only affects the status file on servers with multi−client capability enabled. Valid status version values:

1 Traditional format (default). The client list contains the following fields comma−separated: Common Name, Real Address, Bytes Received, Bytes Sent, Connected Since.

2 A more reliable format for external processing. Compared to version **1**, the client list contains some additional fields: Virtual Address, Virtual IPv6 Address, Username, Client ID, Peer ID, Data Channel Cipher. Future versions may extend the number of fields.

3 Identical to **2**, but fields are tab−separated.

−−test−crypto
Do a self−test of OpenVPN's crypto options by encrypting and decrypting test packets using the data channel encryption options specified above. This option does not require a peer to function, and therefore can be specified without **−−dev** or **−−remote**.

The typical usage of **−−test−crypto** would be something like this:

```
openvpn --test-crypto --secret key
```

or

```
openvpn --test-crypto --secret key --verb 9
```

This option is very useful to test OpenVPN after it has been ported to a new platform, or to isolate problems in the compiler, OpenSSL crypto library, or OpenVPN's crypto code. Since it is a self−test mode, problems with encryption and authentication can be debugged independently of network and tunnel issues.

−−tmp−dir *dir*
Specify a directory **dir** for temporary files. This directory will be used by openvpn processes and script to communicate temporary data with openvpn main process. Note that the directory must be writable by the OpenVPN process after it has dropped it's root privileges.

This directory will be used by in the following cases:

- **−−client−connect** scripts and **OPENVPN_PLUGIN_CLIENT_CONNECT** plug−in hook to dynamically generate client−specific configuration **client_connect_config_file** and return success/failure via **client_connect_deferred_file** when using deferred client connect method
- **OPENVPN_PLUGIN_AUTH_USER_PASS_VERIFY** plug−in hooks returns success/failure via **auth_control_file** when using deferred auth method
- **OPENVPN_PLUGIN_ENABLE_PF** plugin hook to pass filtering rules via **pf_file**

−−use−prediction−resistance
Enable prediction resistance on mbed TLS's RNG.

Enabling prediction resistance causes the RNG to reseed in each call for random. Reseeding this often can quickly deplete the kernel entropy pool.

If you need this option, please consider running a daemon that adds entropy to the kernel pool.

−−user *user*
Change the user ID of the OpenVPN process to **user** after initialization, dropping privileges in the process. This option is useful to protect the system in the event that some hostile party was able to gain control of an OpenVPN session. Though OpenVPN's security features make this unlikely, it is provided as a second line of defense.

By setting **user** to **nobody** or somebody similarly unprivileged, the hostile party would be limited in what damage they could cause. Of course once you take away privileges, you cannot return them to an OpenVPN session. This means, for example, that if you want to reset an OpenVPN daemon with a **SIGUSR1** signal (for example in response to a DHCP reset), you should make use of one or more of the **−−persist** options to ensure that OpenVPN doesn't need to execute any privileged operations in order to restart (such as re−reading key files or running **ifconfig** on the TUN device).

−−writepid *file*
Write OpenVPN's main process ID to **file**.

Log options

−−echo *parms*
Echo **parms** to log output.

Designed to be used to send messages to a controlling application which is receiving the OpenVPN log output.

−−errors−to−stderr
Output errors to stderr instead of stdout unless log output is redirected by one of the **−−log** options.

−−log *file*
Output logging messages to **file**, including output to stdout/stderr which is generated by called scripts. If **file** already exists it will be truncated. This option takes effect immediately when it is parsed in the command line and will supersede syslog output if **−−daemon** or **−−inetd** is also specified. This option is persistent over the entire course of an OpenVPN instantiation and will not be reset by **SIGHUP**, **SIGUSR1**, or **−−ping−restart**.

Note that on Windows, when OpenVPN is started as a service, logging occurs by default without the need to specify this option.

−−log−append *file*
Append logging messages to **file**. If**file** does not e xist, it will be created. This option behaves exactly like **−−log** except that it appends to rather than truncating the log file.

−−machine−readable−output
Always write timestamps and message flags to log messages, even when they otherwise would not be prefixed. In particular, this applies to log messages sent to stdout.

−−mute *n*
Log at most **n** consecutive messages in the same category. This is useful to limit repetitive logging of similar message types.

−−mute−replay−warnings
Silence the output of replay warnings, which are a common false alarm on WiFi networks. This option preserves the security of the replay protection code without the verbosity associated with warnings about duplicate packets.

−−suppress−timestamps
Avoid writing timestamps to log messages, even when they otherwise would be prepended. In particular, this applies to log messages sent to stdout.

−−syslog *progname*
Direct log output to system logger, but do not become a daemon. See **−−daemon** directive above for description of **progname** parameter.

−−verb *n*
Set output verbosity to **n** (default **1**). Each level shows all info from the previous levels. Level **3** is recommended if you want a good summary of what's happening without being swamped by output.

0 No output except fatal errors.

1 to 4 Normal usage range.

5 Outputs **R** and **W** characters to the console for each packet read and write, uppercase is used for TCP/UDP packets and lowercase is used for TUN/TAP packets.

6 to 11 Debug info range (see **errlevel.h** in the source code for additional information on debug levels).

Protocol options
Options in this section affect features available in the OpenVPN wire protocol. Many of these options also define the encryption options of the data channel in the OpenVPN wire protocol. These options must be configured in a compatible way between both the local and remote side.

−−allow−compression *mode*
As described in the **−−compress** option, compression is a potentially dangerous option. This option allows controlling the behaviour of OpenVPN when compression is used and allowed.

Valid syntaxes:

```
allow-compression
allow-compression mode
```

The **mode** argument can be one of the following values:

asym (default)
OpenVPN will only *decompress downlink packets* but *not compress uplink packets*. This also allows migrating to disable compression when changing both server and client configurations to remove compression at the same time is not a feasible option.

no OpenVPN will refuse any non-stub compression.

yes OpenVPN will send and receive compressed packets.

--auth *alg*
Authenticate data channel packets and (if enabled) **tls-auth** control channel packets with HMAC using message digest algorithm **alg**. (The default is **SHA1**). HMAC is a commonly used message authentication algorithm (MAC) that uses a data string, a secure hash algorithm and a key to produce a digital signature.

The OpenVPN data channel protocol uses encrypt-then-mac (i.e. first encrypt a packet then HMAC the resulting ciphertext), which prevents padding oracle attacks.

If an AEAD cipher mode (e.g. GCM) is chosen then the specified **--auth** algorithm is ignored for the data channel and the authentication method of the AEAD cipher is used instead. Note that **alg** still specifies the digest used for **tls-auth**.

In static-key encryption mode, the HMAC key is included in the key file generated by **--genkey**. In TLS mode, the HMAC key is dynamically generated and shared between peers via the TLS control channel. If OpenVPN receives a packet with a bad HMAC it will drop the packet. HMAC usually adds 16 or 20 bytes per packet. Set **alg=none** to disable authentication.

For more information on HMAC see *http://www.cs.ucsd.edu/users/mihir/papers/hmac.html*

--cipher *alg*
This option is deprecated for server-client mode. **--data-ciphers** or possibly *--data-ciphers-fallback*` should be used instead.

Encrypt data channel packets with cipher algorithm **alg**.

The default is **BF-CBC**, an abbreviation for Blowfish in Cipher Block Chaining mode. When cipher negotiation (NCP) is allowed, OpenVPN 2.4 and newer on both client and server side will automatically upgrade to **AES-256-GCM**. See**--data-ciphers** and **--ncp-disable** for more details on NCP.

Using **BF-CBC** is no longer recommended, because of its 64-bit block size. This small block size allows attacks based on collisions, as demonstrated by SWEET32. See *https://community.openvpn.net/openvpn/wiki/SWEET32* for details. Due to this, support for **BF-CBC**, **DES**, **CAST5**, **IDEA** and **RC2** ciphers will be removed in OpenVPN 2.6.

To see other ciphers that are available with OpenVPN, use the **--show-ciphers** option.

Set **alg** to **none** to disable encryption.

−−compress *algorithm*

DEPRECATED Enable a compression algorithm. Compression is generally not recommended. VPN tunnels which use compression are susceptible to the VORALCE attack vector.

The **algorithm** parameter may be **lzo**, **lz4**, **lz4−v2**, **stub**, **stub−v2** or empty. LZO and LZ4 are different compression algorithms, with LZ4 generally offering the best performance with least CPU usage.

The **lz4−v2** and **stub−v2** variants implement a better framing that does not add overhead when packets cannot be compressed. All other variants always add one extra framing byte compared to no compression framing.

If the **algorithm** parameter is **stub**, **stub−v2** or empty, compression will be turned off, but the packet framing for compression will still be enabled, allowing a different setting to be pushed later. Additionally, **stub** and **stub−v2** wil disable announcing **lzo** and **lz4** compression support via *IV_* variables to the server.

Note: the **stub** (or empty) option is NOT compatible with the older option **−−comp−lzo no**.

Security Considerations

Compression and encryption is a tricky combination. If an attacker knows or is able to control (parts of) the plain−text of packets that contain secrets, the attacker might be able to extract the secret if compression is enabled. See e.g. the *CRIME* and *BREACH* attacks on TLS and *VORACLE* on VPNs which also leverage to break encryption. If you are not entirely sure that the above does not apply to your traffic, you are advised to *not* enable compression.

−−comp−lzo *mode*

DEPRECATED Enable LZO compression algorithm. Compression is generally not recommended. VPN tunnels which uses compression are suspectible to the VORALCE attack vector.

Use LZO compression −− may add up to 1 byte per packet for incompressible data. **mode** may be **yes**, **no**, or **adaptive** (default).

In a server mode setup, it is possible to selectively turn compression on or off for individual clients.

First, make sure the client−side config file enables selective compression by having at least one **−−comp−lzo** directive, such as **−−comp−lzo no**. This will turn off compression by default, but allow a future directive push from the server to dynamically change the **on**/**off**/**adaptive** setting.

Next in a **−−client−config−dir** file, specify the compression setting for the client, for example:

```
comp−lzo yes
push "comp−lzo yes"
```

The first line sets the **comp−lzo** setting for the server side of the link, the second sets the client side.

−−comp−noadapt

DEPRECATED When used in conjunction with **−−comp−lzo**, this option will disable OpenVPN's adaptive compression algorithm. Normally, adaptive compression is enabled with **−−comp−lzo**.

Adaptive compression tries to optimize the case where you have compression enabled, but you are sending predominantly incompressible (or pre−compressed) packets over the tunnel, such as an

FTP or rsync transfer of a large, compressed file. With adaptive compression, OpenVPN will periodically sample the compression process to measure its efficiency. If the data being sent over the tunnel is already compressed, the compression efficiency will be very low, triggering openvpn to disable compression for a period of time until the next re−sample test.

−−key−direction
Alternative way of specifying the optional direction parameter for the **−−tls−auth** and **−−secret** options. Useful when using inline files (See section on inline files).

−−keysize *n*
DEPRECATED This option will be removed in OpenVPN 2.6.

Size of cipher key in bits (optional). If unspecified, defaults to cipher−specific default. The **−−show−ciphers** option (see below) shows all available OpenSSL ciphers, their default key sizes, and whether the key size can be changed. Use care in changing a cipher's default key size. Many ciphers have not been extensively cryptanalyzed with non−standard key lengths, and a larger key may offer no real guarantee of greater security, or may even reduce security.

−−data−ciphers *cipher−list*
Restrict the allowed ciphers to be negotiated to the ciphers in **cipher−list**. **cipher−list** is a colon−separated list of ciphers, and defaults to **AES−256−GCM:AES−128−GCM**.

For servers, the first cipher from **cipher−list** that is also supported by the client will be pushed to clients that support cipher negotiation.

Cipher negotiation is enabled in client−server mode only. I.e. if **−−mode** is set to 'server' (server−side, implied by setting **−−server**), or if **−−pull** is specified (client−side, implied by setting −−client).

If no common cipher is found during cipher negotiation, the connection is terminated. To support old clients/old servers that do not provide any cipher negotiation support see **−−data−ciphers−fallback**.

Additionally, to allow for more smooth transition, if NCP is enabled, OpenVPN will inherit the cipher of the peer if that cipher is different from the local **−−cipher** setting, but the peer cipher is one of the ciphers specified in **−−data−ciphers**. E.g. a non−NCP client (<=v2.3, or with −−ncp−disabled set) connecting to a NCP server (v2.4+) with **−−cipher BF−CBC** and **−−data−ciphers AES−256−GCM:AES−256−CBC** set can either specify **−−cipher BF−CBC** or **−−cipher AES−256−CBC** and both will work.

Note for using NCP with an OpenVPN 2.4 peer: This list must include the **AES−256−GCM** and **AES−128−GCM** ciphers.

This list is restricted to be 127 chars long after conversion to OpenVPN ciphers.

This option was called **−−ncp−ciphers** in OpenVPN 2.4 but has been renamed to **−−data−ciphers** in OpenVPN 2.5 to more accurately reflect its meaning.

−−data−ciphers−fallback *alg*
Configure a cipher that is used to fall back to if we could not determine which cipher the peer is willing to use.

This option should only be needed to connect to peers that are running OpenVPN 2.3 and older version, and have been configured with *−−enable−small* (typically used on routers or other embedded devices).

−−ncp−disable
DEPRECATED Disable "Negotiable Crypto Parameters". This completely disables cipher negotiation.

−−secret *args*
Enable Static Key encryption mode (non−TLS). Use pre−shared secret **file** which was generated with **−−genkey**.

Valid syntaxes:

```
secret file
secret file direction
```

The optional **direction** parameter enables the use of 4 distinct keys (HMAC−send, cipher−encrypt, HMAC−receive, cipher−decrypt), so that each data flow direction has a different set of HMAC and cipher keys. This has a number of desirable security properties including eliminating certain kinds of DoS and message replay attacks.

When the **direction** parameter is omitted, 2 keys are used bidirectionally, one for HMAC and the other for encryption/decryption.

The **direction** parameter should always be complementary on either side of the connection, i.e. one side should use **0** and the other should use **1**, or both sides should omit it altogether.

The **direction** parameter requires that **file** contains a 2048 bit key. While pre−1.5 versions of OpenVPN generate 1024 bit key files, any version of OpenVPN which supports the **direction** parameter, will also support 2048 bit key file generation using the **−−genkey** option.

Static key encryption mode has certain advantages, the primary being ease of configuration.

There are no certificates or certificate authorities or complicated negotiation handshakes and protocols. The only requirement is that you have a pre−existing secure channel with your peer (such as **ssh**) to initially copy the key. This requirement, along with the fact that your key never changes unless you manually generate a new one, makes it somewhat less secure than TLS mode (see below). If an attacker manages to steal your key, everything that was ever encrypted with it is compromised. Contrast that to the perfect forward secrecy features of TLS mode (using Diffie Hellman key exchange), where even if an attacker was able to steal your private key, he would gain no information to help him decrypt past sessions.

Another advantageous aspect of Static Key encryption mode is that it is a handshake−free protocol without any distinguishing signature or feature (such as a header or protocol handshake sequence) that would mark the ciphertext packets as being generated by OpenVPN. Anyone eavesdropping on the wire would see nothing but random−looking data.

−−tran−window *n*
Transition window −− our old key can live this many seconds after a new a key renegotiation begins (default **3600** seconds). This feature allows for a graceful transition from old to new key, and removes the key renegotiation sequence from the critical path of tunnel data forwarding.

Client Options
The client options are used when connecting to an OpenVPN server configured to use **−−server**, **−−server−bridge**, or **−−mode server** in its configuration.

−−allow−pull−fqdn
Allow client to pull DNS names from server (rather than being limited to IP address) for **−−ifconfig**, **−−route**, and **−−route−gateway**.

−−allow−recursive−routing
When this option is set, OpenVPN will not drop incoming tun packets with same destination as host.

−−auth−token *token*
This is not an option to be used directly in any configuration files, but rather push this option from a **−−client−connect** script or a **−−plugin** which hooks into the **OPENVPN_PLUGIN_CLIENT_CONNECT** or **OPENVPN_PLUGIN_CLIENT_CONNECT_V2** calls. This option provides a possibility to replace the clients password with an authentication token during the lifetime of the OpenVPN client.

Whenever the connection is renegotiated and the **−−auth−user−pass−verify** script or **−−plugin** making use of the **OPENVPN_PLUGIN_AUTH_USER_PASS_VERIFY** hook is triggered, it will pass over this token as the password instead of the password the user provided. The authentication token can only be reset by a full reconnect where the server can push new options to the client. The password the user entered is never preserved once an authentication token has been set. If the OpenVPN server side rejects the authentication token then the client will receive an **AUTH_FAILED** and disconnect.

The purpose of this is to enable two factor authentication methods, such as HOTP or TOTP, to be used without needing to retrieve a new OTP code each time the connection is renegotiated. Another use case is to cache authentication data on the client without needing to have the users password cached in memory during the life time of the session.

To make use of this feature, the **−−client−connect** script or **−−plugin** needs to put

```
push "auth-token UNIQUE_TOKEN_VALUE"
```

into the file/buffer for dynamic configuration data. This will then make the OpenVPN server to push this value to the client, which replaces the local password with the **UNIQUE_TOKEN_VALUE**.

Newer clients (2.4.7+) will fall back to the original password method after a failed auth. Older clients will keep using the token value and react according to **−−auth−retry**

−−auth−token−user *base64username*
Companion option to **−−auth−token**. This options allows to override the username used by the client when reauthenticating with the **auth−token**. It also allows to use **−−auth−token** in setups that normally do not use username and password.

The username has to be base64 encoded.

−−auth−user−pass
Authenticate with server using username/password.

Valid syntaxes:

```
auth-user-pass
auth-user-pass up
```

If **up** is present, it must be a file containing username/password on 2 lines. If the password line is missing, OpenVPN will prompt for one.

If **up** is omitted, username/password will be prompted from the console.

The server configuration must specify an **−−auth−user−pass−verify** script to verify the

username/password provided by the client.

−−auth−retry *type*
Controls how OpenVPN responds to username/password verification errors such as the client−side response to an **AUTH_FAILED** message from the server or verification failure of the private key password.

Normally used to prevent auth errors from being fatal on the client side, and to permit username/password requeries in case of error.

An **AUTH_FAILED** message is generated by the server if the client fails **−−auth−user−pass** authentication, or if the server−side **−−client−connect** script returns an error status when the client tries to connect.

type can be one of:

none Client will exit with a fatal error (this is the default).

nointeract
Client will retry the connection without requerying for an **−−auth−user−pass** username/password. Use this option for unattended clients.

interact
Client will requery for an **−−auth−user−pass** username/password and/or private key password before attempting a reconnection.

Note that while this option cannot be pushed, it can be controlled from the management interface.

−−client
A helper directive designed to simplify the configuration of OpenVPN's client mode. This directive is equivalent to:

```
pull
tls-client
```

−−client−nat *args*
This pushable client option sets up a stateless one−to−one NAT rule on packet addresses (not ports), and is useful in cases where routes or ifconfig settings pushed to the client would create an IP numbering conflict.

Examples:

```
client-nat snat 192.168.0.0/255.255.0.0
client-nat dnat 10.64.0.0/255.255.0.0
```

network/netmask (for example **192.168.0.0/255.255.0.0**) defines the local view of a resource from the client perspective, while **alias/netmask** (for example **10.64.0.0/255.255.0.0**) defines the remote view from the server perspective.

Use **snat** (source NAT) for resources owned by the client and **dnat** (destination NAT) for remote resources.

Set **−−verb 6** for debugging info showing the transformation of src/dest addresses in packets.

−−connect−retry *n*
Wait **n** seconds between connection attempts (default **5**). Repeated reconnection attempts are slowed down after 5 retries per remote by doubling the wait time after each unsuccessful attempt. An optional argument **max** specifies the maximum value of wait time in seconds at which it gets capped (default **300**).

−−connect−retry−max *n*
n specifies the number of times each **−−remote** or **<connection>** entry is tried. Specifying **n** as **1** would try each entry exactly once. A successful connection resets the counter. (default *unlimited*).

−−connect−timeout *n*
See **−−server−poll−timeout**.

−−explicit−exit−notify *n*
In UDP client mode or point−to−point mode, send server/peer an exit notification if tunnel is restarted or OpenVPN process is exited. In client mode, on exit/restart, this option will tell the server to immediately close its client instance object rather than waiting for a timeout.

The **n** parameter (default **1** if not present) controls the maximum number of attempts that the client will try to resend the exit notification message.

In UDP server mode, send **RESTART** control channel command to connected clients. The **n** parameter (default **1** if not present) controls client behavior. With **n** = **1** client will attempt to reconnect to the same server, with **n** = **2** client will advance to the next server.

OpenVPN will not send any exit notifications unless this option is enabled.

−−inactive *args*
Causes OpenVPN to exit after **n** seconds of inactivity on the TUN/TAP device. The time length of inactivity is measured since the last incoming or outgoing tunnel packet. The default value is 0 seconds, which disables this feature.

Valid syntaxes:

```
inactive n
inactive n bytes
```

If the optional **bytes** parameter is included, exit if less than **bytes** of combined in/out traffic are produced on the tun/tap device in **n** seconds.

In any case, OpenVPN's internal ping packets (which are just keepalives) and TLS control packets are not considered "activity", nor are they counted as traffic, as they are used internally by OpenVPN and are not an indication of actual user activity.

−−proto−force *p*
When iterating through connection profiles, only consider profiles using protocol **p** (**tcp** | **udp**).

−−pull This option must be used on a client which is connecting to a multi−client server. It indicates to OpenVPN that it should accept options pushed by the server, provided they are part of the legal set of pushable options (note that the **−−pull** option is implied by **−−client**).

In particular, **−−pull** allows the server to push routes to the client, so you should not use **−−pull** or **−−client** in situations where you don't trust the server to have control over the client's routing table.

−−pull−filter *args*
Filter options on the client pushed by the server to the client.

Valid syntaxes:

```
pull−filter accept text
pull−filter ignore text
pull−filter reject text
```

Filter options received from the server if the option starts with **text**. The action flag**accept** allo ws the option, **ignore** removes it and **reject** flags an error and triggers a **SIGUSR1** restart. The filters may be specified multiple times, and each filter is applied in the order it is specified. The filtering of each option stops as soon as a match is found. Unmatched options are accepted by default.

Prefix comparison is used to match **text** against the received option so that

```
pull-filter ignore "route"
```

would remove all pushed options starting with **route** which would include, for example, **route-gateway**. Enclose *text* in quotes to embed spaces.

```
pull-filter accept "route 192.168.1."
pull-filter ignore "route "
```

would remove all routes that do not start with **192.168.1**.

Note that **reject** may result in a repeated cycle of failure and reconnect, unless multiple remotes are specified and connection to the next remote succeeds. To silently ignore an option pushed by the server, use **ignore**.

--push-peer-info

Push additional information about the client to server. The following data is always pushed to the server:

IV_VER=<version>
The client OpenVPN version

IV_PLAT=[linux|solaris|openbsd|mac|netbsd|freebsd|win]
The client OS platform

IV_LZO_STUB=1
If client was built with LZO stub capability

IV_LZ4=1
If the client supports LZ4 compressions.

IV_PROTO
Details about protocol extensions that the peer supports. The variable is a bitfield and the bits are defined as follows (starting a bit 0 for the first (unused) bit:

- bit 1: The peer supports peer-id floating mechanism
- bit 2: The client expects a push-reply and the server may send this reply without waiting for a push-request first.
- bit 3: The client is capable of doing key derivation using RFC5705 key material exporter.
- bit 4: The client is capable of accepting additional arguments to the *AUTH_PENDING* message.

IV_NCP=2
Negotiable ciphers, client supports **--cipher** pushed by the server, a value of 2 or greater indicates client supports *AES-GCM-128* and *AES-GCM-256*.

IV_CIPHERS=<ncp-ciphers>
The client announces the list of supported ciphers configured with the **--data-ciphers** option to the server.

IV_GUI_VER=<gui_id> <version>
The UI version of a UI if one is running, for example **de.blinkt.openvpn 0.5.47** for the Android app.

IV_SSO=[crtext,][openurl,][proxy_url]
Additional authentication methods supported by the client. This may be set by the client UI/GUI using **−−setenv**

When **−−push−peer−info** is enabled the additional information consists of the following data:

IV_HWADDR=<string>
This is intended to be a unique and persistent ID of the client. The string value can be any readable ASCII string up to 64 bytes. OpenVPN 2.x and some other implementations use the MAC address of the client's interface used to reach the default gateway. If this string is generated by the client, it should be consistent and preserved across independent session and preferably re−installations and upgrades.

IV_SSL=<version string>
The ssl version used by the client, e.g. **OpenSSL 1.0.2f 28 Jan 2016**.

IV_PLAT_VER=x.y
The version of the operating system, e.g. 6.1 for Windows 7.

UV_<name>=<value>
Client environment variables whose names start with **UV_**

−−remote *args*
Remote host name or IP address, port and protocol.

Valid syntaxes:

```
remote host
remote host port
remote host port proto
```

The **port** and **proto** arguments are optional. The OpenVPN client will try to connect to a server at **host:port**. The**pr oto** argument indicates the protocol to use when connecting with the remote, and may be **tcp** or **udp**. To enforce IPv4 or IPv6 connections add a **4** or **6** suffix; like **udp4** / **udp6** / **tcp4** / **tcp6**.

On the client, multiple **−−remote** options may be specified for redundancy, each referring to a different OpenVPN server, in the order specified by the list of **−−remote** options. Specifying multiple **−−remote** options for this purpose is a special case of the more general connection−profile feature. See the **<connection>** documentation below.

The client will move on to the next host in the list, in the event of connection failure. Note that at any given time, the OpenVPN client will at most be connected to one server.

Examples:

```
remote server1.example.net
remote server1.example.net 1194
remote server2.example.net 1194 tcp
```

Note: Since UDP is connectionless, connection failure is defined by the **−−ping** and **−−ping−restart** options.

Also, if you use multiple **−−remote** options, AND you are dropping root privileges on the client with **−−user** and/or **−−group** AND the client is running a non−Windows OS, if the

client needs to switch to a different server, and that server pushes back different TUN/TAP or route settings, the client may lack the necessary privileges to close and reopen the TUN/TAP interface. This could cause the client to exit with a fatal error.

If **−−remote** is unspecified, OpenVPN will listen for packets from any IP address, but will not act on those packets unless they pass all authentication tests. This requirement for authentication is binding on all potential peers, even those from known and supposedly trusted IP addresses (it is very easy to forge a source IP address on a UDP packet).

When used in TCP mode, **−−remote** will act as a filter, rejecting connections from any host which does not match **host**.

If **host** is a DNS name which resolves to multiple IP addresses, OpenVPN will try them in the order that the system getaddrinfo() presents them, so priorization and DNS randomization is done by the system library. Unless an IP version is forced by the protocol specification (4/6 suffix), OpenVPN will try both IPv4 and IPv6 addresses, in the order getaddrinfo() returns them.

−−remote−random
When multiple **−−remote** address/ports are specified, or if connection profiles are being used, initially randomize the order of the list as a kind of basic load−balancing measure.

−−remote−random−hostname
Prepend a random string (6 bytes, 12 hex characters) to hostname to prevent DNS caching. For example, "foo.bar.gov" would be modified to "<random−chars>.foo.bar.gov".

−−resolv−retry *n*
If hostname resolve fails for **−−remote**, retry resolve for **n** seconds before failing.

Set **n** to "infinite" to retry indefinitely.

By default, **−−resolv−retry infinite** is enabled. You can disable by setting n=0.

−−single−session
After initially connecting to a remote peer, disallow any new connections. Using this option means that a remote peer cannot connect, disconnect, and then reconnect.

If the daemon is reset by a signal or **−−ping−restart**, it will allow one new connection.

−−single−session can be used with **−−ping−exit** or **−−inactive** to create a single dynamic session that will exit when finished.

−−server−poll−timeout *n*
When connecting to a remote server do not wait for more than **n** seconds for a response before trying the next server. The default value is 120s. This timeout includes proxy and TCP connect timeouts.

−−static−challenge *args*
Enable static challenge/response protocol

Valid syntax:

```
static−challenge text echo
```

The **text** challenge text is presented to the user which describes what information is requested. The **echo** flag indicates if the user's input should be echoed on the screen. Valid **echo** values are **0** or **1**.

See management−notes.txt in the OpenVPN distribution for a description of the OpenVPN

challenge/response protocol.

−−show−proxy−settings
Show sensed HTTP or SOCKS proxy settings. Currently, only Windows clients support this option.

−−http−proxy *args*
Connect to remote host through an HTTP proxy. This requires at least an address**ser ver** and **port** argument. If HTTP Proxy−Authenticate is required, a file name to an**authfile** file containing a username and password on 2 lines can be given, or **stdin** to prompt from console. Its content can also be specified in the config file with the **−−http−proxy−user−pass** option. (See section on inline files)

The last optional argument is an **auth−method** which should be one of **none**, **basic**, or **ntlm**.

HTTP Digest authentication is supported as well, but only via the **auto** or **auto−nct** flags (below). This must replace the **authfile** argument.

The **auto** flag causes OpenVPN to automatically determine the **auth−method** and query stdin or the management interface for username/password credentials, if required. This flag exists on OpenVPN 2.1 or higher.

The **auto−nct** flag (no clear−text auth) instructs OpenVPN to automatically determine the authentication method, but to reject weak authentication protocols such as HTTP Basic Authentication.

Examples:

```
http−proxy proxy.example.net 3128
http−proxy proxy.example.net 3128 authfile.txt
http−proxy proxy.example.net 3128 stdin
http−proxy proxy.example.net 3128 auto basic
http−proxy proxy.example.net 3128 auto−nct ntlm
```

−−http−proxy−option *args*
Set extended HTTP proxy options. Requires an option **type** as argument and an optional **parameter** to the type. Repeat to set multiple options.

VERSION version
Set HTTP version number to **version** (default **1.0**).

AGENT user−agent
Set HTTP "User−Agent" string to **user−agent**.

CUSTOM−HEADER name content
Adds the custom Header with **name** as name and **content** as the content of the custom HTTP header.

Examples:

```
http−proxy−option VERSION 1.1
http−proxy−option AGENT OpenVPN/2.4
http−proxy−option X−Proxy−Flag some−flags
```

−−socks−proxy *args*
Connect to remote host through a Socks5 proxy. A required**ser ver** argument is needed. Optionally a **port** (default **1080**) and **authfile** can be given. The**authfile** is a file containing a username and password on 2 lines, or **stdin** can be used to prompt from console.

Server Options

Starting with OpenVPN 2.0, a multi−client TCP/UDP server mode is supported, and can be enabled with the **−−mode server** option. In server mode, OpenVPN will listen on a single port for incoming client connections. All client connections will be routed through a single tun or tap interface. This mode is designed for scalability and should be able to support hundreds or even thousands of clients on sufficiently fast hardware. SSL/TLS authentication must be used in this mode.

−−auth−gen−token *args*

Returns an authentication token to successfully authenticated clients.

Valid syntax:

```
auth−gen−token [lifetime] [external−auth]
```

After successful user/password authentication, the OpenVPN server will with this option generate a temporary authentication token and push that to the client. On the following renegotiations, the OpenVPN client will pass this token instead of the users password. On the server side the server will do the token authentication internally and it will NOT do any additional authentications against configured external user/password authentication mechanisms.

The tokens implemented by this mechanism include an initial timestamp and a renew timestamp and are secured by HMAC.

The **lifetime** argument defines how long the generated token is valid. The lifetime is defined in seconds. If lifetime is not set or it is set to **0**, the token will never expire.

The token will expire either after the configured **lifetime** of the token is reached or after not being renewed for more than 2 * **reneg−sec** seconds. Clients will be sent renewed tokens on every TLS renogiation to keep the client's token updated. This is done to invalidate a token if a client is disconnected for a sufficently long time, while at the same time permitting much longer token lifetimes for active clients.

This feature is useful for environments which are configured to use One Time Passwords (OTP) as part of the user/password authentications and that authentication mechanism does not implement any auth−token support.

When the **external−auth** keyword is present the normal authentication method will always be called even if auth−token succeeds. Normally other authentications method are skipped if auth−token verification suceeds or fails.

This option postpones this decision to the external authentication methods and checks the validity of the account and do other checks.

In this mode the environment will have a **session_id** variable that holds the session id from auth−gen−token. Also an environment variable **session_state** is present. This variable indicates whether the auth−token has succeeded or not. It can have the following values:

Initial No token from client.

Authenticated
Token is valid and not expired.

Expired
Token is valid but has expired.

Invalid Token is invalid (failed HMAC or wrong length)

AuthenticatedEmptyUser / ExpiredEmptyUser
The token is not valid with the username sent from the client but would be valid (or expired) if we assume an empty username was used instead. These two cases are a workaround for behaviour in OpenVPN 3. If this workaround is not needed these two cases should be handled in the same way as **Invalid**.

Warning: Use this feature only if you want your authentication method called on every verification. Since the external authentication is called it needs to also indicate a success or failure of the authentication. It is strongly recommended to return an authentication failure in the case of the Invalid/Expired auth−token with the external−auth option unless the client could authenticate in another acceptable way (e.g. client certificate), otherwise returning success will lead to authentication bypass (as does returning success on a wrong password from a script).

−−auth−gen−token−secret *file*
Specifies a file that holds a secret for the HMAC used in **−−auth−gen−token** If **file** is not present OpenVPN will generate a random secret on startup. This file should be used if auth−token should validate after restarting a server or if client should be able to roam between multiple OpenVPN servers with their auth−token.

−−auth−user−pass−optional
Allow connections by clients that do not specify a username/password. Normally, when **−−auth−user−pass−verify** or **−−management−client−auth** are specified (or an authentication plugin module), the OpenVPN server daemon will require connecting clients to specify a username and password. This option makes the submission of a username/password by clients optional, passing the responsibility to the user−defined authentication module/script to accept or deny the client based on other factors (such as the setting of X509 certificate fields). When this option is used, and a connecting client does not submit a username/password, the user−defined authentication module/script will see the username and password as being set to empty strings (""). The authentication module/script MUST have logic to detect this condition and respond accordingly.

−−ccd−exclusive
Require, as a condition of authentication, that a connecting client has a **−−client−config−dir** file.

−−client−config−dir *dir*
Specify a directory **dir** for custom client config files. After a connecting client has been authenticated, OpenVPN will look in this directory for a file having the same name as the client's X509 common name. If a matching file exists, it will be opened and parsed for client−specific configuration options. If no matching file is found, OpenVPN will instead try to open and parse a default file called "DEFAULT", which may be provided but is not required. Note that the configuration files must be readable by the OpenVPN process after it has dropped it's root privileges.

This file can specify a fixed IP address for a given client using **−−ifconfig−push**, as well as fixed subnets owned by the client using **−−iroute**.

One of the useful properties of this option is that it allows client configuration files to be conveniently created, edited, or removed while the server is live, without needing to restart the server.

The following options are legal in a client−specific context: **−−push**, **−−push−reset**, **−−push−remove**, **−−iroute**, **−−ifconfig−push**, **−−vlan−pvid** and **−−config**.

−−client−to−client
Because the OpenVPN server mode handles multiple clients through a single tun or tap interface, it is effectively a router. The **−−client−to−client** flag tells OpenVPN to internally route client−to−client traffic rather than pushing all client−originating traffic to the TUN/TAP interface.

When this option is used, each client will "see" the other clients which are currently connected.

Otherwise, each client will only see the server. Don't use this option if you want to firewall tunnel traffic using custom, per−client rules.

−−disable
Disable a particular client (based on the common name) from connecting. Don't use this option to disable a client due to key or password compromise. Use a CRL (certificate revocation list) instead (see the **−−crl−verify** option).

This option must be associated with a specific client instance, which means that it must be specified either in a client instance config file using **−−client−config−dir** or dynamically generated using a **−−client−connect** script.

−−connect−freq *args*
Allow a maximum of **n** new connections per **sec** seconds from clients.

Valid syntax:

```
connect−freq n sec
```

This is designed to contain DoS attacks which flood the server with connection requests using certificates which will ultimately fail to authenticate.

This is an imperfect solution however, because in a real DoS scenario, legitimate connections might also be refused.

For the best protection against DoS attacks in server mode, use **−−proto udp** and either **−−tls−auth** or **−−tls−crypt**.

−−duplicate−cn
Allow multiple clients with the same common name to concurrently connect. In the absence of this option, OpenVPN will disconnect a client instance upon connection of a new client having the same common name.

−−ifconfig−pool *args*
Set aside a pool of subnets to be dynamically allocated to connecting clients, similar to a DHCP server.

Valid syntax:

```
ifconfig−pool start−IP end−IP [netmask]
```

For tun−style tunnels, each client will be given a /30 subnet (for interoperability with Windows clients). For tap−style tunnels, individual addresses will be allocated, and the optional **netmask** parameter will also be pushed to clients.

−−ifconfig−ipv6−pool *args*
Specify an IPv6 address pool for dynamic assignment to clients.

Valid args:

```
ifconfig−ipv6−pool ipv6addr/bits
```

The pool starts at **ipv6addr** and matches the offset determined from the start of the IPv4 pool. If the host part of the given IPv6 address is **0**, the pool starts at **ipv6addr** +1.

−−ifconfig−pool−persist *args*
Persist/unpersist ifconfig−pool data to **file**, at **seconds** intervals (default **600**), as well as on program startup and shutdown.

Valid syntax:

```
ifconfig-pool-persist file [seconds]
```

The goal of this option is to provide a long−term association between clients (denoted by their common name) and the virtual IP address assigned to them from the ifconfig−pool. Maintaining a long−term association is good for clients because it allows them to effectively use the **−−persist−tun** option.

file is a comma−delimited ASCII file, formatted as **<Common−Name>,<IP−address>**.

If **seconds** = **0**, **file** will be treated as read−only. This is useful if you would like to treat **file** as a configuration file.

Note that the entries in this file are treated by OpenVPN as *suggestions* only, based on past associations between a common name and IP address. They do not guarantee that the given common name will always receive the given IP address. If you want guaranteed assignment, use **−−ifconfig−push**

−−ifconfig−push *args*

Push virtual IP endpoints for client tunnel, overriding the **−−ifconfig−pool** dynamic allocation.

Valid syntax:

```
ifconfig-push local remote-netmask [alias]
```

The parameters **local** and **remote−netmask** are set according to the **−−ifconfig** directive which you want to execute on the client machine to configure the remote end of the tunnel. Note that the parameters **local** and **remote−netmask** are from the perspective of the client, not the server. They may be DNS names rather than IP addresses, in which case they will be resolved on the server at the time of client connection.

The optional **alias** parameter may be used in cases where NAT causes the client view of its local endpoint to differ from the server view. In this case **local/remote−netmask** will refer to the server view while **alias/remote−netmask** will refer to the client view.

This option must be associated with a specific client instance, which means that it must be specified either in a client instance config file using **−−client−config−dir** or dynamically generated using a **−−client−connect** script.

Remember also to include a **−−route** directive in the main OpenVPN config file which encloses **local**, so that the kernel will know to route it to the server's TUN/TAP interface.

OpenVPN's internal client IP address selection algorithm works as follows:

1. Use **−−client−connect script** generated file for static IP (first choice).
2. Use **−−client−config−dir** file for static IP (next choice).
3. Use **−−ifconfig−pool** allocation for dynamic IP (last choice).

−−ifconfig−ipv6−push *args*

for **−−client−config−dir** per−client static IPv6 interface configuration, see **−−client−config−dir** and **−−ifconfig−push** for more details.

Valid syntax:

```
ifconfig-ipv6-push ipv6addr/bits ipv6remote
```

--inetd *args*

Valid syntaxes:

```
inetd
inetd wait
inetd nowait
inetd wait progname
```

Use this option when OpenVPN is being run from the inetd or **xinetd**(8) server.

The **wait** and **nowait** option must match what is specified in the inetd/xinetd config file. The **nowait** mode can only be used with **--proto tcp-server** The default is **wait**. The**no wait** mode can be used to instantiate the OpenVPN daemon as a classic TCP server, where client connection requests are serviced on a single port number. For additional information on this kind of configuration, see the OpenVPN FAQ: *https://community.openvpn.net/openvpn/wiki/325-openvpn-as-a--forking-tcp-server-which-can-service-*

This option precludes the use of **--daemon**, **--local** or **--remote**. Note that this option causes message and error output to be handled in the same way as the **--daemon** option. The optional **progname** parameter is also handled exactly as in **--daemon**.

Also note that in **wait** mode, each OpenVPN tunnel requires a separate TCP/UDP port and a separate inetd or xinetd entry. See the OpenVPN 1.x HOWTO for an example on using OpenVPN with xinetd: *https://openvpn.net/community-resources/1xhowto/*

--multihome

Configure a multi-homed UDP server. This option needs to be used when a server has more than one IP address (e.g. multiple interfaces, or secondary IP addresses), and is not using **--local** to force binding to one specific address only. This option will add some extra lookups to the packet path to ensure that the UDP reply packets are always sent from the address that the client is talking to. This is not supported on all platforms, and it adds more processing, so it's not enabled by default.

Notes:

- This option is only relevant for UDP servers.
- If you do an IPv6+IPv4 dual-stack bind on a Linux machine with multiple IPv4 address, connections to IPv4 addresses will not work right on kernels before 3.15, due to missing kernel support for the IPv4-mapped case (some distributions have ported this to earlier kernel versions, though).

--iroute *args*

Generate an internal route to a specific client. The **netmask** parameter, if omitted, defaults to **255.255.255.255**.

Valid syntax:

```
iroute network [netmask]
```

This directive can be used to route a fixed subnet from the server to a particular client, regardless of where the client is connecting from. Remember that you must also add the route to the system routing table as well (such as by using the **--route** directive). The reason why two routes are needed is that the **--route** directive routes the packet from the kernel to OpenVPN. Once in OpenVPN, the **--iroute** directive routes to the specific client.

This option must be specified either in a client instance config file using **−−client−config−dir** or dynamically generated using a **−−client−connect** script.

The **−−iroute** directive also has an important interaction with **−−push "route ..."**. **−−iroute** essentially defines a subnet which is owned by a particular client (we will call this client *A*). If you would like other clients to be able to reach *A*'s subnet, you can use **−−push "route ..."** together with **−−client−to−client** to effect this. In order for all clients to see *A*'s subnet, OpenVPN must push this route to all clients EXCEPT for *A*, since the subnet is already owned by *A*. OpenVPN accomplishes this by not not pushing a route to a client if it matches one of the client's iroutes.

−−iroute−ipv6 *args*
for **−−client−config−dir** per−client static IPv6 route configuration, see **−−iroute** for more details how to setup and use this, and how **−−iroute** and **−−route** interact.

Valid syntax:

```
iroute−ipv6 ipv6addr/bits
```

−−max−clients *n*
Limit server to a maximum of **n** concurrent clients.

−−max−routes−per−client *n*
Allow a maximum of **n** internal routes per client (default **256**). This is designed to help contain DoS attacks where an authenticated client floods the server with packets appearing to come from many unique MAC addresses, forcing the server to deplete virtual memory as its internal routing table expands. This directive can be used in a **−−client−config−dir** file or auto−generated by a **−−client−connect** script to override the global value for a particular client.

Note that this directive affects OpenVPN's internal routing table, not the kernel routing table.

−−opt−verify
Clients that connect with options that are incompatible with those of the server will be disconnected.

Options that will be compared for compatibility include **dev−type**, **link−mtu**, **tun−mtu**, **proto**, **ifconfig**, **comp−lzo**, **fragment**, **keydir**, **cipher**, **auth**, **keysize**, **secret**, **no−replay**, **tls−auth**, **key−method**, **tls−server** and **tls−client**.

This option requires that **−−disable−occ** NOT be used.

−−port−share *args*
Share OpenVPN TCP with another service

Valid syntax:

```
port−share host port [dir]
```

When run in TCP server mode, share the OpenVPN port with another application, such as an HTTPS server. If OpenVPN senses a connection to its port which is using a non−OpenVPN protocol, it will proxy the connection to the server at **host**:**port**. Currently only designed to work with HTTP/HTTPS, though it would be theoretically possible to extend to other protocols such as ssh.

dir specifies an optional directory where a temporary file with name N containing content C will be dynamically generated for each proxy connection, where N is the source IP:port of the client connection and C is the source IP:port of the connection to the proxy receiver. This directory can be used as a dictionary by the proxy receiver to determine the origin of the connection. Each generated file will be automatically deleted when the proxied connection is torn down.

Not implemented on Windows.

−−push *option*

Push a config file option back to the client for remote execution. Note that **option** must be enclosed in double quotes (""). The client must specify **−−pull** in its config file. The set of options which can be pushed is limited by both feasibility and security. Some options such as those which would execute scripts are banned, since they would effectively allow a compromised server to execute arbitrary code on the client. Other options such as TLS or MTU parameters cannot be pushed because the client needs to know them before the connection to the server can be initiated.

This is a partial list of options which can currently be pushed: **−−route**, **−−route−gateway**, **−−route−delay**, **−−redirect−gateway**, **−−ip−win32**, **−−dhcp−option**, **−−inactive**, **−−ping**, **−−ping−exit**, **−−ping−restart**, **−−setenv**, **−−auth−token**, **−−persist−key**, **−−persist−tun**, **−−echo**, **−−comp−lzo**, **−−socket−flags**, **−−sndbuf**, **−−rcvbuf**

−−push−remove *opt*

Selectively remove all **−−push** options matching "opt" from the option list for a client. **opt** is matched as a substring against the whole option string to−be−pushed to the client, so **−−push−remove route** would remove all **−−push route ...** and **−−push route−ipv6 ...** statements, while **−−push−remove "route−ipv6 2001:"** would only remove IPv6 routes for **2001:...** networks.

−−push−remove can only be used in a client−specific context, like in a **−−client−config−dir** file, or **−−client−connect** script or plugin −− similar to **−−push−reset**, just more selective.

NOTE: to *change* an option, **−−push−remove** can be used to first remove the old value, and then add a new **−−push** option with the new value.

NOTE 2: due to implementation details, 'ifconfig' and 'ifconfig−ipv6' can only be removed with an exact match on the option (**push−remove ifconfig**), no substring matching and no matching on the IPv4/IPv6 address argument is possible.

−−push−reset

Don't inherit the global push list for a specific client instance. Specify this option in a client−specific context such as with a **−−client−config−dir** configuration file. This option will ignore **−−push** options at the global config file level.

NOTE: **−−push−reset** is very thorough: it will remove almost all options from the list of to−be−pushed options. In many cases, some of these options will need to be re−configured afterwards − specifically, **−−topology subnet** and **−−route−gateway** will get lost and this will break client configs in many cases. Thus, for most purposes, **−−push−remove** is better suited to selectively remove push options for individual clients.

−−server *args*

A helper directive designed to simplify the configuration of OpenVPN's server mode. This directive will set up an OpenVPN server which will allocate addresses to clients out of the given network/netmask. The server itself will take the **.1** address of the given network for use as the server−side endpoint of the local TUN/TAP interface. If the optional **nopool** flag is given, no dynamic IP address pool will prepared for VPN clients.

Valid syntax:

```
server network netmask [nopool]
```

For example, **−−server 10.8.0.0 255.255.255.0** expands as follows:

```
mode server
tls−server
```

```
push "topology [topology]"

if dev tun AND (topology == net30 OR topology == p2p):
  ifconfig 10.8.0.1 10.8.0.2
  if !nopool:
    ifconfig-pool 10.8.0.4 10.8.0.251
  route 10.8.0.0 255.255.255.0
  if client-to-client:
    push "route 10.8.0.0 255.255.255.0"
  else if topology == net30:
    push "route 10.8.0.1"

if dev tap OR (dev tun AND topology == subnet):
  ifconfig 10.8.0.1 255.255.255.0
  if !nopool:
    ifconfig-pool 10.8.0.2 10.8.0.253 255.255.255.0
  push "route-gateway 10.8.0.1"
  if route-gateway unset:
    route-gateway 10.8.0.2
```

Don't use **--server** if you are ethernet bridging. Use **--server-bridge** instead.

--server-bridge *args*

A helper directive similar to **--server** which is designed to simplify the configuration of OpenVPN's server mode in ethernet bridging configurations.

Valid syntaxes:

```
server-bridge gateway netmask pool-start-IP pool-end-IP
server-bridge [nogw]
```

If **--server-bridge** is used without any parameters, it will enable a DHCP-proxy mode, where connecting OpenVPN clients will receive an IP address for their TAP adapter from the DHCP server running on the OpenVPN server-side LAN. Note that only clients that support the binding of a DHCP client with the TAP adapter (such as Windows) can support this mode. The optional **nogw** flag (advanced) indicates that gateway information should not be pushed to the client.

To configure ethernet bridging, you must first use your OS's bridging capability to bridge the TAP interface with the ethernet NIC interface. For example, on Linux this is done with the **brctl** tool, and with Windows XP it is done in the Network Connections Panel by selecting the ethernet and TAP adapters and right-clicking on "Bridge Connections".

Next you you must manually set the IP/netmask on the bridge interface. The**gateway** and **netmask** parameters to **--server-bridge** can be set to either the IP/netmask of the bridge interface, or the IP/netmask of the default gateway/router on the bridged subnet.

Finally, set aside a IP range in the bridged subnet, denoted by **pool-start-IP** and **pool-end-IP**, for OpenVPN to allocate to connecting clients.

For example, **server-bridge 10.8.0.4 255.255.255.0 10.8.0.128 10.8.0.254** expands as follows:

```
mode server
tls-server

ifconfig-pool 10.8.0.128 10.8.0.254 255.255.255.0
```

```
push "route-gateway 10.8.0.4"
```

In another example, **--server-bridge** (without parameters) expands as follows:

```
mode server
tls-server

push "route-gateway dhcp"
```

Or **--server-bridge nogw** expands as follows:

```
mode server
tls-server
```

--server-ipv6 *args*
Convenience-function to enable a number of IPv6 related options at once, namely **--ifconfig-ipv6**, **--ifconfig-ipv6-pool** and **--push tun-ipv6**.

Valid syntax:

```
server-ipv6 ipv6addr/bits
```

Pushing of the **--tun-ipv6** directive is done for older clients which require an explicit **--tun-ipv6** in their configuration.

--stale-routes-check *args*
Remove routes which haven't had activity for **n** seconds (i.e. the ageing time). This check is run every **t** seconds (i.e. check interval).

Valid syntax:

```
stale-routes-check n [t]
```

If **t** is not present it defaults to **n**.

This option helps to keep the dynamic routing table small. See also **--max-routes-per-client**

--username-as-common-name
Use the authenticated username as the common-name, rather than the common-name from the client certificate. Requires that some form of **--auth-user-pass** verification is in effect. As the replacement happens after **--auth-user-pass** verification, the verification script or plugin will still receive the common-name from the certificate.

The common_name environment variable passed to scripts and plugins invoked after authentication (e.g, client-connect script) and file names parsed in client-config directory will match the username.

--verify-client-cert *mode*
Specify whether the client is required to supply a valid certificate.

Possible **mode** options are:

none A client certificate is not required. the client needs to authenticate using username/password only. Be aware that using this directive is less secure than requiring certificates from all clients.

If you use this directive, the entire responsibility of authentication will rest on your

−−auth−user−pass−verify script, so keep in mind that bugs in your script could potentially compromise the security of your VPN.

−−verify−client−cert none is functionally equivalent to **−−client−cert−not−required**.

optional
A client may present a certificate but it is not required to do so. When using this directive, you should also use a **−−auth−user−pass−verify** script to ensure that clients are authenticated using a certificate, a username and password, or possibly even both.

Again, the entire responsibility of authentication will rest on your **−−auth−user−pass−verify** script, so keep in mind that bugs in your script could potentially compromise the security of your VPN.

require
This is the default option. A client is required to present a certificate, otherwise VPN access is refused.

If you don't use this directive (or use **−−verify−client−cert require**) but you also specify an **−−auth−user−pass−verify** script, then OpenVPN will perform double authentication. The client certificate verification AND the **−−auth−user−pass−verify** script will need to succeed in order for a client to be authenticated and accepted onto the VPN.

−−vlan−tagging
Server−only option. Turns the OpenVPN server instance into a switch that understands VLAN−tagging, based on IEEE 802.1Q.

The server TAP device and each of the connecting clients is seen as a port of the switch. All client ports are in untagged mode and the server TAP device is VLAN−tagged, untagged or accepts both, depending on the **−−vlan−accept** setting.

Ethernet frames with a prepended 802.1Q tag are called "tagged". If the VLAN Identifier (VID) field in such a tag is non−zero, the frame is called "VLAN−tagged". If the VID is zero, but the Priority Control Point (PCP) field is non−zero, the frame is called "prio−tagged". If there is no 802.1Q tag, the frame is "untagged".

Using the **−−vlan−pvid v** option once per client (see −−client−config−dir), each port can be associated with a certain VID. Packets can only be forwarded between ports having the same VID. Therefore, clients with differing VIDs are completely separated from one−another, even if **−−client−to−client** is activated.

The packet filtering takes place in the OpenVPN server. Clients should not have any VLAN tagging configuration applied.

The **−−vlan−tagging** option is off by default. While turned off, OpenVPN accepts any Ethernet frame and does not perform any special processing for VLAN−tagged packets.

This option can only be activated in **−−dev tap mode**.

−−vlan−accept *args*
Configure the VLAN tagging policy for the server TAP device.

Valid syntax:

```
vlan-accept  all|tagged|untagged
```

The following modes are available:

tagged Admit only VLAN−tagged frames. Only VLAN−tagged packets are accepted, while untagged or priority−tagged packets are dropped when entering the server TAP device.

untagged
Admit only untagged and prio−tagged frames. VLAN−tagged packets are not accepted, while untagged or priority−tagged packets entering the server TAP device are tagged with the value configured for the global **−−vlan−pvid** setting.

all (default)
Admit all frames. All packets are admitted and then treated like untagged or tagged mode respectively.

*Note***:** Some vendors refer to switch ports running in **tagged** mode as "trunk ports" and switch ports running in **untagged** mode as "access ports".

Packets forwarded from clients to the server are VLAN−tagged with the originating client's PVID, unless the VID matches the global **−−vlan−pvid**, in which case the tag is removed.

If no *PVID* is configured for a given client (see −−vlan−pvid) packets are tagged with 1 by default.

−−vlan−pvid *v*
Specifies which VLAN identifier a "port" is associated with. Only valid when **−−vlan−tagging** is speficied.

In the client context, the setting specifies which VLAN ID a client is associated with. In the global context, the VLAN ID of the server TAP device is set. The latter only makes sense for **−−vlan−accept untagged** and **−−vlan−accept all** modes.

Valid values for **v** go from **1** through to **4094**. The global value defaults to **1**. If no **−−vlan−pvid** is specified in the client context, the global value is inherited.

In some switch implementations, the *PVID* is also referred to as "Native VLAN".

ENCRYPTION OPTIONS

SSL Library information

−−show−ciphers
(Standalone) Show all cipher algorithms to use with the **−−cipher** option.

−−show−digests
(Standalone) Show all message digest algorithms to use with the **−−auth** option.

−−show−tls
(Standalone) Show all TLS ciphers supported by the crypto library. OpenVPN uses TLS to secure the control channel, over which the keys that are used to protect the actual VPN traffic are exchanged. The TLS ciphers will be sorted from highest preference (most secure) to lowest.

Be aware that whether a cipher suite in this list can actually work depends on the specific setup of both peers (e.g. both peers must support the cipher, and an ECDSA cipher suite will not work if you are using an RSA certificate, etc.).

−−show−engines
(Standalone) Show currently available hardware−based crypto acceleration engines supported by the OpenSSL library.

−−show−groups
(Standalone) Show all available elliptic curves/groups to use with the **−−ecdh−curve** and **tls−groups** options.

Generating key material

−−genkey *args*

(Standalone) Generate a key to be used of the type keytype. if keyfile is left out or empty the key will be output on stdout. See the following sections for the different keytypes.

Valid syntax:

```
--genkey keytype keyfile
```

Valid keytype arguments are:

secret Standard OpenVPN shared secret keys

tls−crypt Alias for **secret**

tls−auth Alias for **secret**

auth−token Key used for **−−auth−gen−token−key**

tls−crypt−v2−server TLS Crypt v2 server key

tls−crypt−v2−client TLS Crypt v2 client key

Examples:

```
$ openvpn --genkey secret shared.key
$ openvpn --genkey tls-crypt shared.key
$ openvpn --genkey tls-auth shared.key
$ openvpn --genkey tls-crypt-v2-server v2crypt-server.key
$ openvpn --tls-crypt-v2 v2crypt-server.key --genkey tls-crypt-v2-client
```

- Generating *Shared Secret Keys* Generate a shared secret, for use with the **−−secret**, **−−tls−auth** or **−−tls−crypt** options.

  Syntax:

  ```
  $ openvpn --genkey secret|tls-crypt|tls-auth keyfile
  ```

  The key is saved in **keyfile**. All three variants (**−−secret**, **tls−crypt** and **tls−auth**) generate the same type of key. The aliases are added for convenience.

  If using this for **−−secret**, this file must be shared with the peer over a pre−existing secure channel such as **scp**(1).

- Generating *TLS Crypt v2 Server key* Generate a **−−tls−crypt−v2** key to be used by an OpenVPN server. The key is stored in **keyfile**.

  Syntax:

  ```
  --genkey tls-crypt-v2-server keyfile
  ```

- Generating *TLS Crypt v2 Client key* Generate a −−tls−crypt−v2 key to be used by OpenVPN clients. The key is stored in **keyfile**.

  Syntax

```
--genkey tls-crypt-v2-client keyfile [metadata]
```

If supplied, include the supplied **metadata** in the wrapped client key. This metadata must be supplied in base64-encoded form. The metadata must be at most 735 bytes long (980 bytes in base64).

If no metadata is supplied, OpenVPN will use a 64-bit unix timestamp representing the current time in UTC, encoded in network order, as metadata for the generated key.

A tls-crypt-v2 client key is wrapped using a server key. To generate a client key, the user must therefore supply the server key using the **--tls-crypt-v2** option.

Servers can use **--tls-crypt-v2-verify** to specify a metadata verification command.

- Generate *Authentication Token key* Generate a new secret that can be used with **--auth-gen-token-secret**

  Syntax:

  ```
  --genkey auth-token [keyfile]
  ```

  Note: This file should be kept secret to the server as anyone that has access to this file will be able to generate auth tokens that the OpenVPN server will accept as valid.

Data Channel Renegotiation

When running OpenVPN in client/server mode, the data channel will use a separate ephemeral encryption key which is rotated at regular intervals.

--reneg-bytes *n*

Renegotiate data channel key after **n** bytes sent or received (disabled by default with an exception, see below). OpenVPN allows the lifetime of a key to be expressed as a number of bytes encrypted/decrypted, a number of packets, or a number of seconds. A key renegotiation will be forced if any of these three criteria are met by either peer.

If using ciphers with cipher block sizes less than 128-bits, **--reneg-bytes** is set to 64MB by default, unless it is explicitly disabled by setting the value to **0**, but this is **HIGHLY DISCOURAGED** as this is designed to add some protection against the SWEET32 attack vector. For more information see the **--cipher** option.

--reneg-pkts *n*

Renegotiate data channel key after **n** packets sent and received (disabled by default).

--reneg-sec *args*

Renegotiate data channel key after at most **max** seconds (default **3600**) and at least **min** seconds (default is 90% of **max** for servers, and equal to **max** for clients).

```
reneg-sec max [min]
```

The effective **--reneg-sec** value used is per session pseudo-uniform-randomized between **min** and **max**.

With the default value of **3600** this results in an effective per session value in the range of **3240** .. **3600** seconds for servers, or just 3600 for clients.

When using dual-factor authentication, note that this default value may cause the end user to be challenged to reauthorize once per hour.

Also, keep in mind that this option can be used on both the client and server, and whichever uses

the lower value will be the one to trigger the renegotiation. A common mistake is to set **−−reneg−sec** to a higher value on either the client or server, while the other side of the connection is still using the default value of **3600** seconds, meaning that the renegotiation will still occur once per **3600** seconds. The solution is to increase −−reneg−sec on both the client and server, or set it to **0** on one side of the connection (to disable), and to your chosen value on the other side.

TLS Mode Options

TLS mode is the most powerful crypto mode of OpenVPN in both security and flexibility. TLS mode works by establishing control and data channels which are multiplexed over a single TCP/UDP port. OpenVPN initiates a TLS session over the control channel and uses it to exchange cipher and HMAC keys to protect the data channel. TLS mode uses a robust reliability layer over the UDP connection for all control channel communication, while the data channel, over which encrypted tunnel data passes, is forwarded without any mediation. The result is the best of both worlds: a fast data channel that forwards over UDP with only the overhead of encrypt, decrypt, and HMAC functions, and a control channel that provides all of the security features of TLS, including certificate−based authentication and Diffie Hellman forward secrecy.

To use TLS mode, each peer that runs OpenVPN should have its own local certificate/key pair (**−−cert** and **−−key**), signed by the root certificate which is specified in **−−ca**.

When two OpenVPN peers connect, each presents its local certificate to the other. Each peer will then check that its partner peer presented a certificate which was signed by the master root certificate as specified in **−−ca**.

If that check on both peers succeeds, then the TLS negotiation will succeed, both OpenVPN peers will exchange temporary session keys, and the tunnel will begin passing data.

The OpenVPN project provides a set of scripts for managing RSA certificates and keys: *https://github.com/OpenVPN/easy−rsa*

−−askpass *file*

Get certificate password from console or **file** before we daemonize.

Valid syntaxes:

```
askpass
askpass file
```

For the extremely security conscious, it is possible to protect your private key with a password. Of course this means that every time the OpenVPN daemon is started you must be there to type the password. The **−−askpass** option allows you to start OpenVPN from the command line. It will query you for a password before it daemonizes. To protect a private key with a password you should omit the **−nodes** option when you use the **openssl** command line tool to manage certificates and private keys.

If **file** is specified, read the password from the first line of **file**. Keep in mind that storing your password in a file to a certain extent invalidates the extra security provided by using an encrypted key.

−−ca *file*

Certificate authority (CA) file in .pem format, also referred to as the *root* certificate. This file can have multiple certificates in .pem format, concatenated together. You can construct your own certificate authority certificate and private key by using a command such as:

```
openssl req −nodes −new −x509 −keyout ca.key −out ca.crt
```

Then edit your openssl.cnf file and edit the **certificate** variable to point to your new root certificate **ca.crt**.

For testing purposes only, the OpenVPN distribution includes a sample CA certificate (ca.crt). Of course you should never use the test certificates and test keys distributed with OpenVPN in a production environment, since by virtue of the fact that they are distributed with OpenVPN, they are totally insecure.

−−capath *dir*
Directory containing trusted certificates (CAs and CRLs). Not available with mbed TLS.

CAs in the capath directory are expected to be named <hash>.<n>. CRLs are expected to be named <hash>.r<n>. See the **−CApath** option of **openssl verify**, and the **−hash** option of **openssl x509**, **openssl crl** and **X509_LOOKUP_hash_dir()**(3) for more information.

Similar to the **−−crl−verify** option, CRLs are not mandatory − OpenVPN will log the usual warning in the logs if the relevant CRL is missing, but the connection will be allowed.

−−cert *file*
Local peer's signed certificate in .pem format −− must be signed by a certificate authority whose certificate is in **−−ca file**. Each peer in an OpenVPN link running in TLS mode should have its own certificate and private key file. In addition, each certificate should have been signed by the key of a certificate authority whose public key resides in the **−−ca** certificate authority file. You can easily make your own certificate authority (see above) or pay money to use a commercial service such as thawte.com (in which case you will be helping to finance the world's second space tourist :). To generate a certificate, you can use a command such as:

```
openssl req −nodes −new −keyout mycert.key −out mycert.csr
```

If your certificate authority private key lives on another machine, copy the certificate signing request (mycert.csr) to this other machine (this can be done over an insecure channel such as email). Now sign the certificate with a command such as:

```
openssl ca −out mycert.crt −in mycert.csr
```

Now copy the certificate (mycert.crt) back to the peer which initially generated the .csr file (this can be over a public medium). Note that the **openssl ca** command reads the location of the certificate authority key from its configuration file such as **/usr/share/ssl/openssl.cnf** −− note also that for certificate authority functions, you must set up the files **index.txt** (may be empty) and **serial** (initialize to **01**).

−−crl−verify *args*
Check peer certificate against a Certificate Revocation List.

Valid syntax:

```
crl−verify file/directory flag
```

Examples:

```
crl−verify crl−file.pem
crl−verify /etc/openvpn/crls dir
```

A CRL (certificate revocation list) is used when a particular key is compromised but when the overall PKI is still intact.

Suppose you had a PKI consisting of a CA, root certificate, and a number of client certificates. Suppose a laptop computer containing a client key and certificate was stolen. By adding the stolen certificate to the CRL file, you could reject any connection which attempts to use it, while

preserving the overall integrity of the PKI.

The only time when it would be necessary to rebuild the entire PKI from scratch would be if the root certificate key itself was compromised.

The option is not mandatory − if the relevant CRL is missing, OpenVPN will log a warning in the logs − e.g.

```
VERIFY WARNING: depth=0, unable to get certificate CRL
```

but the connection will be allowed. If the optional**dir** flag is specified, enable a dif ferent mode where the **crl−verify** is pointed at a directory containing files named as revoked serial numbers (the files may be empty, the contents are never read). If a client requests a connection, where the client certificate serial number (decimal string) is the name of a file present in the directory, it will be rejected.

Note: As the crl file (or directory) is read every time a peer connects, if you are dropping root privileges with **−−user**, make sure that this user has sufficient privileges to read the file.

−−dh *file*
File containing Diffie Hellman parameters in .pem format (required for **−−tls−server** only).

Set **file** to **none** to disable Diffie Hellman key exchange (and use ECDH only). Note that this requires peers to be using an SSL library that supports ECDH TLS cipher suites (e.g. OpenSSL 1.0.1+, or mbed TLS 2.0+).

Use **openssl dhparam −out dh2048.pem 2048** to generate 2048−bit DH parameters. Diffie Hellman parameters may be considered public.

−−ecdh−curve *name*
Specify the curve to use for elliptic curve Diffie Hellman. Available curves can be listed with **−−show−curves**. The specified curve will only be used for ECDH TLS−ciphers.

This option is not supported in mbed TLS builds of OpenVPN.

−−extra−certs *file*
Specify a **file** containing one or more PEM certs (concatenated together) that complete the local certificate chain.

This option is useful for "split" CAs, where the CA for server certs is different than the CA for client certs. Putting certs in this file allows them to be used to complete the local certificate chain without trusting them to verify the peer−submitted certificate, as would be the case if the certs were placed in the **ca** file.

−−hand−window *n*
Handshake Window −− the TLS−based key exchange must finalize within **n** seconds of handshake initiation by any peer (default **60** seconds). If the handshake fails we will attempt to reset our connection with our peer and try again. Even in the event of handshake failure we will still use our expiring key for up to **−−tran−window** seconds to maintain continuity of transmission of tunnel data.

−−key *file*
Local peer's private key in .pem format. Use the private key which was generated when you built your peer's certificate (see **−−cert file** above).

−−pkcs12 *file*
Specify a PKCS #12 file containing local private key, local certificate, and root CA certificate. This option can be used instead of **−−ca**, **−−cert**, and **−−key**. Not available with mbed TLS.

−−remote−cert−eku *oid*
Require that peer certificate was signed with an explicit *extended key usage*.

This is a useful security option for clients, to ensure that the host they connect to is a designated server.

The extended key usage should be encoded in *oid notation*, or *OpenSSL symbolic representation*.

−−remote−cert−ku *key−usage*
Require that peer certificate was signed with an explicit **key−usage**.

If present in the certificate, the **keyUsage** value is validated by the TLS library during the TLS handshake. Specifying this option without arguments requires this extension to be present (so the TLS library will verify it).

If **key−usage** is a list of usage bits, the **keyUsage** field must have *at least* the same bits set as the bits in *one of* the values supplied in the **key−usage** list.

The **key−usage** values in the list must be encoded in hex, e.g.

```
remote−cert−ku a0
```

−−remote−cert−tls *type*
Require that peer certificate was signed with an explicit *key usage* and *extended key usage* based on RFC3280 TLS rules.

Valid syntaxes:

```
remote−cert−tls server
remote−cert−tls client
```

This is a useful security option for clients, to ensure that the host they connect to is a designated server. Or the other way around; for a server to verify that only hosts with a client certificate can connect.

The **−−remote−cert−tls client** option is equivalent to

```
remote−cert−ku
remote−cert−eku "TLS Web Client Authentication"
```

The **−−remote−cert−tls server** option is equivalent to

```
remote−cert−ku
remote−cert−eku "TLS Web Server Authentication"
```

This is an important security precaution to protect against a man−in−the−middle attack where an authorized client attempts to connect to another client by impersonating the server. The attack is easily prevented by having clients verify the server certificate using any one of **−−remote−cert−tls**, **−−verify−x509−name**, or **−−tls−verify**.

−−tls−auth *args*
Add an additional layer of HMAC authentication on top of the TLS control channel to mitigate DoS attacks and attacks on the TLS stack.

Valid syntaxes:

```
tls-auth file
tls-auth file 0
tls-auth file 1
```

In a nutshell, **--tls-auth** enables a kind of "HMAC firewall" on OpenVPN's TCP/UDP port, where TLS control channel packets bearing an incorrect HMAC signature can be dropped immediately without response.

file (required) is a file in OpenVPN static key format which can be generated by **--genkey**.

Older versions (up to OpenVPN 2.3) supported a freeform passphrase file. This is no longer supported in newer versions (v2.4+).

See the **--secret** option for more information on the optional **direction** parameter.

--tls-auth is recommended when you are running OpenVPN in a mode where it is listening for packets from any IP address, such as when **--remote** is not specified, or **--remote** is specified with **--float**.

The rationale for this feature is as follows. TLS requires a multi-packet exchange before it is able to authenticate a peer. During this time before authentication, OpenVPN is allocating resources (memory and CPU) to this potential peer. The potential peer is also exposing many parts of OpenVPN and the OpenSSL library to the packets it is sending. Most successful network attacks today seek to either exploit bugs in programs (such as buffer overflow attacks) or force a program to consume so many resources that it becomes unusable. Of course the first line of defense is always to produce clean, well-audited code. OpenVPN has been written with buffer overflow attack prevention as a top priority. But as history has shown, many of the most widely used network applications have, from time to time, fallen to buffer overflow attacks.

So as a second line of defense, OpenVPN offers this special layer of authentication on top of the TLS control channel so that every packet on the control channel is authenticated by an HMAC signature and a unique ID for replay protection. This signature will also help protect against DoS (Denial of Service) attacks. An important rule of thumb in reducing vulnerability to DoS attacks is to minimize the amount of resources a potential, but as yet unauthenticated, client is able to consume.

--tls-auth does this by signing every TLS control channel packet with an HMAC signature, including packets which are sent before the TLS level has had a chance to authenticate the peer. The result is that packets without the correct signature can be dropped immediately upon reception, before they have a chance to consume additional system resources such as by initiating a TLS handshake. **--tls-auth** can be strengthened by adding the **--replay-persist** option which will keep OpenVPN's replay protection state in a file so that it is not lost across restarts.

It should be emphasized that this feature is optional and that the key file used with **--tls-auth** gives a peer nothing more than the power to initiate a TLS handshake. It is not used to encrypt or authenticate any tunnel data.

Use **--tls-crypt** instead if you want to use the key file to not only authenticate, but also encrypt the TLS control channel.

--tls-groups *list*

A list of allowable groups/curves in order of preference.

Set the allowed elliptic curves/groups for the TLS session. These groups are allowed to be used in signatures and key exchange.

mbedTLS currently allows all known curves per default.

OpenSSL 1.1+ restricts the list per default to

```
"X25519:secp256r1:X448:secp521r1:secp384r1".
```

If you use certificates that use non−standard curves, you might need to add them here. If you do not force the ecdh curve by using **−−ecdh−curve**, the groups for ecdh will also be picked from this list.

OpenVPN maps the curve name *secp256r1* to *prime256v1* to allow specifying the same tls−groups option for mbedTLS and OpenSSL.

Warning: this option not only affects elliptic curve certificates but also the key exchange in TLS 1.3 and using this option improperly will disable TLS 1.3.

−−tls−cert−profile *profile*
Set the allowed cryptographic algorithms for certificates according to **profile**.

The following profiles are supported:

legacy (default)
SHA1 and newer, RSA 2048−bit+, any elliptic curve.

preferred
SHA2 and newer, RSA 2048−bit+, any elliptic curve.

suiteb SHA256/SHA384, ECDSA with P−256 or P−384.

This option is only fully supported for mbed TLS builds. OpenSSL builds use the following approximation:

legacy (default)
sets "security level 1"

preferred
sets "security level 2"

suiteb sets "security level 3" and **−−tls−cipher "SUITEB128"**.

OpenVPN will migrate to 'preferred' as default in the future. Please ensure that your keys already comply.

WARNING: **−−tls−ciphers, −−tls−ciphersuites and tls−groups**
These options are expert features, which − if used correctly − can improve the security of your VPN connection. But it is also easy to unwittingly use them to carefully align a gun with your foot, or just break your connection. Use with care!

−−tls−cipher *l*
A list **l** of allowable TLS ciphers delimited by a colon ("**:**").

These setting can be used to ensure that certain cipher suites are used (or not used) for the TLS connection. OpenVPN uses TLS to secure the control channel, over which the keys that are used to protect the actual VPN traffic are exchanged.

The supplied list of ciphers is (after potential OpenSSL/IANA name translation) simply supplied to the crypto library. Please see the OpenSSL and/or mbed TLS documentation for details on the cipher list interpretation.

For OpenSSL, the **−−tls−cipher** is used for TLS 1.2 and below.

Use **−−show−tls** to see a list of TLS ciphers supported by your crypto library.

The default for **−−tls−cipher** is to use mbed TLS's default cipher list when using mbed TLS or **DEFAULT:!EXP:!LOW:!MEDIUM:!kDH:!kECDH:!DSS:!PSK:!SRP:!kRSA** when using OpenSSL.

−−tls−ciphersuites *l*
Same as **−−tls−cipher** but for TLS 1.3 and up. mbed TLS has no TLS 1.3 support yet and only the **−−tls−cipher** setting is used.

The default for *−−tls−ciphersuites* is to use the crypto library's default.

−−tls−client
Enable TLS and assume client role during TLS handshake.

−−tls−crypt *keyfile*
Encrypt and authenticate all control channel packets with the key from **keyfile**. (See **−−tls−auth** for more background.)

Encrypting (and authenticating) control channel packets:

- provides more privacy by hiding the certificate used for the TLS connection,
- makes it harder to identify OpenVPN traffic as such,
- provides "poor−man's" post−quantum security, against attackers who will never know the pre−shared key (i.e. no forward secrecy).

In contrast to **−−tls−auth**, **−−tls−crypt** does *not* require the user to set **−−key−direction**.

Security Considerations

All peers use the same **−−tls−crypt** pre−shared group key to authenticate and encrypt control channel messages. To ensure that IV collisions remain unlikely, this key should not be used to encrypt more than 2^48 client−to−server or 2^48 server−to−client control channel messages. A typical initial negotiation is about 10 packets in each direction. Assuming both initial negotiation and renegotiations are at most 2^16 (65536) packets (to be conservative), and (re)negotiations happen each minute for each user (24/7), this limits the tls−crypt key lifetime to 8171 years divided by the number of users. So a setup with 1000 users should rotate the key at least once each eight years. (And a setup with 8000 users each year.)

If IV collisions were to occur, this could result in the security of **−−tls−crypt** degrading to the same security as using **−−tls−auth**. That is, the control channel still benefits from the extra protection against active man−in−the−middle−attacks and DoS attacks, but may no longer offer extra privacy and post−quantum security on top of what TLS itself offers.

For large setups or setups where clients are not trusted, consider using **−−tls−crypt−v2** instead. That uses per−client unique keys, and thereby improves the bounds to 'rotate a client key at least once per 8000 years'.

−−tls−crypt−v2 *keyfile*
Use client−specific tls−crypt keys.

For clients, **keyfile** is a client−specific tls−crypt key. Such a key can be generated using the **−−genkey tls−crypt−v2−client** option.

For servers, **keyfile** is used to unwrap client−specific keys supplied by the client during connection setup. This key must be the same as the key used to generate the client−specific key (see **−−genkey tls−crypt−v2−client**).

On servers, this option can be used together with the **−−tls−auth** or **−−tls−crypt** option. In that case, the server will detect whether the client is using client−specific keys, and automatically select the right mode.

−−tls−crypt−v2−verify *cmd*
Run command **cmd** to verify the metadata of the client−specific tls−crypt−v2 key of a connecting client. This allows server administrators to reject client connections, before exposing the TLS stack (including the notoriously dangerous X.509 and ASN.1 stacks) to the connecting client.

OpenVPN supplies the following environment variables to the command:

- **script_type** is set to **tls−crypt−v2−verify**
- **metadata_type** is set to **0** if the metadata was user supplied, or **1** if it's a 64−bit unix timestamp representing the key creation time.
- **metadata_file** contains the filename of a temporary file that contains the client metadata.

The command can reject the connection by exiting with a non−zero exit code.

−−tls−exit
Exit on TLS negotiation failure.

−−tls−export−cert *directory*
Store the certificates the clients use upon connection to this directory. This will be done before **−−tls−verify** is called. The certificates will use a temporary name and will be deleted when the tls−verify script returns. The file name used for the certificate is available via the **peer_cert** environment variable.

−−tls−server
Enable TLS and assume server role during TLS handshake. Note that OpenVPN is designed as a peer−to−peer application. The designation of client or server is only for the purpose of negotiating the TLS control channel.

−−tls−timeout *n*
Packet retransmit timeout on TLS control channel if no acknowledgment from remote within **n** seconds (default **2**). When OpenVPN sends a control packet to its peer, it will expect to receive an acknowledgement within **n** seconds or it will retransmit the packet, subject to a TCP−like exponential backoff algorithm. This parameter only applies to control channel packets. Data channel packets (which carry encrypted tunnel data) are never acknowledged, sequenced, or retransmitted by OpenVPN because the higher level network protocols running on top of the tunnel such as TCP expect this role to be left to them.

−−tls−version−min *args*
Sets the minimum TLS version we will accept from the peer (default is "1.0").

Valid syntax:

```
tls−version−min version ['or−highest']
```

Examples for version include **1.0**, **1.1**, or **1.2**. If **or−highest** is specified and version is not recognized, we will only accept the highest TLS version supported by the local SSL implementation.

−−tls−version−max *version*
Set the maximum TLS version we will use (default is the highest version supported). Examples for version include **1.0**, **1.1**, or **1.2**.

−−verify−hash *args*
Specify SHA1 or SHA256 fingerprint for level−1 cert.

Valid syntax:

```
verify-hash hash [algo]
```

The level-1 cert is the CA (or intermediate cert) that signs the leaf certificate, and is one removed from the leaf certificate in the direction of the root. When accepting a connection from a peer, the level-1 cert fingerprint must match **hash** or certificate verification will fail. Hash is specified as XX:XX:... For example:

```
AD:B0:95:D8:09:C8:36:45:12:A9:89:C8:90:09:CB:13:72:A6:AD:16
```

The **algo** flag can be either **SHA1** or **SHA256**. If not provided, it defaults to **SHA1**.

--verify-x509-name *args*

Accept connections only if a host's X.509 name is equal to **name.** The remote host must also pass all other tests of verification.

Valid syntax:

```
verify-x509 name type
```

Which X.509 name is compared to **name** depends on the setting of type. **type** can be **subject** to match the complete subject DN (default), **name** to match a subject RDN or **name-prefix** to match a subject RDN prefix. Which RDN is verified as name depends on the **--x509-username-field** option. But it defaults to the common name (CN), e.g. a certificate with a subject DN

```
C=KG, ST=NA, L=Bishkek, CN=Server-1
```

would be matched by:

```
verify-x509-name 'C=KG, ST=NA, L=Bishkek, CN=Server-1'
verify-x509-name Server-1 name
verify-x509-name Server- name-prefix
```

The last example is useful if you want a client to only accept connections to **Server-1**, **Server-2**, etc.

--verify-x509-name is a useful replacement for the **--tls-verify** option to verify the remote host, because **--verify-x509-name** works in a **--chroot** environment without any dependencies.

Using a name prefix is a useful alternative to managing a CRL (Certificate Revocation List) on the client, since it allows the client to refuse all certificates except for those associated with designated servers.

NOTE: Test against a name prefix only when you are using OpenVPN with a custom CA certificate that is under your control. Never use this option with type **name-prefix** when your client certificates are signed by a third party, such as a commercial web CA.

--x509-track *attribute*

Save peer X509 **attribute** value in environment for use by plugins and management interface. Prepend a **+** to **attribute** to save values from full cert chain. Values will be encoded as **X509_<depth>_<attribute>=<value>**. Multiple **--x509-track** options can be defined to track multiple attributes.

--x509-username-field *args*

Field in the X.509 certificate subject to be used as the username (default **CN**).

Valid syntax:

```
x509-username-field [ext:]fieldname
```

Typically, this option is specified with **fieldname** as either of the following:

```
x509-username-field emailAddress
x509-username-field ext:subjectAltName
```

The first example uses the value of the **emailAddress** attribute in the certificate's Subject field as the username. The second example uses the **ext:** prefix to signify that the X.509 extension **fieldname subjectAltName** be searched for an rfc822Name (email) field to be used as the username. In cases where there are multiple email addresses in **ext:fieldname**, the last occurrence is chosen.

When this option is used, the **--verify-x509-name** option will match against the chosen **fieldname** instead of the Common Name.

Only the **subjectAltName** and **issuerAltName** X.509 extensions are supported.

Please note: This option has a feature which will convert an all-lowercase **fieldname** to uppercase characters, e.g., **ou** -> **OU**. A mixed-case **fieldname** or one having the **ext:** prefix will be left as-is. This automatic upcasing feature is deprecated and will be removed in a future release.

PKCS#11 / SmartCard options

--pkcs11-cert-private *args*

Set if access to certificate object should be performed after login. Every provider has its own setting.

Valid syntaxes:

```
pkcs11-cert-private 0
pkcs11-cert-private 1
```

--pkcs11-id *name*

Specify the serialized certificate id to be used. The id can be gotten by the standalone **--show-pkcs11-ids** option.

--pkcs11-id-management

Acquire PKCS#11 id from management interface. In this case a **NEED-STR 'pkcs11-id-request'** real-time message will be triggered, application may use pkcs11-id-count command to retrieve available number of certificates, and pkcs11-id-get command to retrieve certificate id and certificate body.

--pkcs11-pin-cache *seconds*

Specify how many seconds the PIN can be cached, the default is until the token is removed.

--pkcs11-private-mode *mode*

Specify which method to use in order to perform private key operations. A different mode can be specified for each provider. Mode is encoded as hex number, and can be a mask one of the following:

0 (default) Try to determine automatically.

1 Use sign.

2 Use sign recover.

4 Use decrypt.

8 Use unwrap.

−−pkcs11−protected−authentication *args*
Use PKCS#11 protected authentication path, useful for biometric and external keypad devices. Every provider has its own setting.

Valid syntaxes:

```
pkcs11−protected−authentication 0
pkcs11−protected−authentication 1
```

−−pkcs11−providers *provider*
Specify an RSA Security Inc. PKCS #11 Cryptographic Token Interface (Cryptoki) providers to load. This option can be used instead of **−−cert**, **−−key** and **−−pkcs12**.

If p11−kit is present on the system, its **p11−kit−proxy.so** module will be loaded by default if either the **−−pkcs11−id** or **−−pkcs11−id−management** options are specified without **−−pkcs11−provider** being given.

−−show−pkcs11−ids *args*
(Standalone) Show PKCS#11 token object list.

Valid syntax:

```
show−pkcs11 [provider] [cert_private]
```

Specify **cert_private** as **1** if certificates are stored as private objects.

If *p11−kit* is present on the system, the **provider** argument is optional; if omitted the default **p11−kit−proxy.so** module will be queried.

−−verb option can be used BEFORE this option to produce debugging information.

DATA CHANNEL CIPHER NEGOTIATION

OpenVPN 2.4 and higher have the capability to negotiate the data cipher that is used to encrypt data packets. This section describes the mechanism in more detail and the different backwards compatibility mechanism with older server and clients.

OpenVPN 2.5 and higher behaviour

When both client and server are at least running OpenVPN 2.5, that the order of the ciphers of the server's **−−data−ciphers** is used to pick the the data cipher. That means that the first cipher in that list that is also in the client's **−−data−ciphers** list is chosen. If no common cipher is found the client is rejected with a AUTH_FAILED message (as seen in client log):

AUTH: Received control message: AUTH_FAILED,Data channel cipher negotiation failed (no shared cipher)

OpenVPN 2.5 will only allow the ciphers specified in **−−data−ciphers**. To ensure backwards compatibility also if a cipher is specified using the **−−cipher** option it is automatically added to this list. If both options are unset the default is **AES−256−GCM:AES−128−GCM**.

OpenVPN 2.4 clients

The negotiation support in OpenVPN 2.4 was the first iteration of the implementation and still had some quirks. Its main goal was "upgrade to AES−256−GCM when possible". An OpenVPN 2.4 client that is built against a crypto library that supports AES in GCM mode and does not have **−−ncp−disable** will always announce support for *AES−256−GCM* and *AES−128−GCM* to a server by sending **IV_NCP=2**.

This only causes a problem if **−−ncp−ciphers** option has been changed from the default of **AES−256−GCM:AES−128−GCM** to a value that does not include these two ciphers. When a OpenVPN

servers try to use *AES−256−GCM* or *AES−128−GCM* the connection will then fail. It is therefore recommended to always have the *AES−256−GCM* and *AES−128−GCM* ciphers to the **−−ncp−ciphers** options to avoid this behaviour.

OpenVPN 3 clients

Clients based on the OpenVPN 3.x library (*https://github.com/openvpn/openvpn3/*) do not have a configurable **−−ncp−ciphers** or **−−data−ciphers** option. Instead these clients will announce support for all their supported AEAD ciphers (*AES−256−GCM*, *AES−128−GCM* and in newer versions also *Chacha20−Poly1305*).

To support OpenVPN 3.x based clients at least one of these ciphers needs to be included in the server's **−−data−ciphers** option.

OpenVPN 2.3 and older clients (and clients with −−ncp−disable)

When a client without cipher negotiation support connects to a server the cipher specified with the **−−cipher** option in the client configuration must be included in the **−−data−ciphers** option of the server to allow the client to connect. Otherwise the client will be sent the **AUTH_FAILED** message that indicates no shared cipher.

If the client is 2.3 or older and has been configured with the **−−enable−small ./configure** argument, using **data−ciphers−fallback cipher** in the server config file with the explicit cipher used by the client is necessary.

OpenVPN 2.4 server

When a client indicates support for *AES−128−GCM* and *AES−256−GCM* (with **IV_NCP=2**) an OpenVPN 2.4 server will send the first cipher of the **−−ncp−ciphers** to the OpenVPN client regardless of what the cipher is. To emulate the behaviour of an OpenVPN 2.4 client as close as possible and have compatibility to a setup that depends on this quirk, adding *AES−128−GCM* and *AES−256−GCM* to the client's **−−data−ciphers** option is required. OpenVPN 2.5+ will only announce the **IV_NCP=2** flag if those ciphers are present.

OpenVPN 2.3 and older servers (and servers with −−ncp−disable)

The cipher used by the server must be included in **−−data−ciphers** to allow the client connecting to a server without cipher negotiation support. (For compatibility OpenVPN 2.5 will also accept the cipher set with **−−cipher**)

If the server is 2.3 or older and has been configured with the **−−enable−small ./configure** argument, adding **data−ciphers−fallback cipher** to the client config with the explicit cipher used by the server is necessary.

Blowfish in CBC mode (BF−CBC) deprecation

The **−−cipher** option defaulted to **BF−CBC** in OpenVPN 2.4 and older version. The default was never changed to ensure backwards compatibility. In OpenVPN 2.5 this behaviour has now been changed so that if the **−−cipher** is not explicitly set it does not allow the weak **BF−CBC** cipher any more and needs to explicitly added as **−−cipher BFC−CBC** or added to **−−data−ciphers**.

We strongly recommend to switching away from BF−CBC to a more secure cipher as soon as possible instead.

NETWORK CONFIGURATION

OpenVPN consists of two sides of network configuration. One side is the *link* between the local and remote side, the other side is the *virtual network adapter* (tun/tap device).

Link Options

This link options section covers options related to the connection between the local and the remote host.

−−bind *keywords*

Bind to local address and port. This is the default unless any of **−−proto tcp−client** , **−−http−proxy** or **−−socks−proxy** are used.

If the optional **ipv6only** keyword is present OpenVPN will bind only to IPv6 (as opposed to IPv6 and IPv4) when a IPv6 socket is opened.

−−float Allow remote peer to change its IP address and/or port number, such as due to DHCP (this is the default if **−−remote** is not used). **−−float** when specified with **−−remote** allows an OpenVPN session to initially connect to a peer at a known address, however if packets arrive from a new address and pass all authentication tests, the new address will take control of the session. This is useful when you are connecting to a peer which holds a dynamic address such as a dial−in user or DHCP client.

Essentially, **−−float** tells OpenVPN to accept authenticated packets from any address, not only the address which was specified in the **−−remote** option.

−−fragment *max*
Enable internal datagram fragmentation so that no UDP datagrams are sent which are larger than **max** bytes.

The **max** parameter is interpreted in the same way as the **−−link−mtu** parameter, i.e. the UDP packet size after encapsulation overhead has been added in, but not including the UDP header itself.

The **−−fragment** option only makes sense when you are using the UDP protocol (**−−proto udp**).

−−fragment adds 4 bytes of overhead per datagram.

See the **−−mssfix** option below for an important related option to **−−fragment**.

It should also be noted that this option is not meant to replace UDP fragmentation at the IP stack level. It is only meant as a last resort when path MTU discovery is broken. Using this option is less efficient than fixing path MTU discovery for your IP link and using native IP fragmentation instead.

Having said that, there are circumstances where using OpenVPN's internal fragmentation capability may be your only option, such as tunneling a UDP multicast stream which requires fragmentation.

−−keepalive *args*
A helper directive designed to simplify the expression of **−−ping** and **−−ping−restart**.

Valid syntax:

```
keepalive interval timeout
```

This option can be used on both client and server side, but it is enough to add this on the server side as it will push appropriate **−−ping** and **−−ping−restart** options to the client. If used on both server and client, the values pushed from server will override the client local values.

The **timeout** argument will be twice as long on the server side. This ensures that a timeout is detected on client side before the server side drops the connection.

For example, **−−keepalive 10 60** expands as follows:

```
if mode server:
    ping 10                  # Argument: interval
    ping-restart 120         # Argument: timeout*2
    push "ping 10"           # Argument: interval
```

```
        push "ping-restart 60"     # Argument: timeout
    else
        ping 10                    # Argument: interval
        ping-restart 60            # Argument: timeout
```

--link-mtu *n*
Sets an upper bound on the size of UDP packets which are sent between OpenVPN peers. *It's best not to set this parameter unless you know what you're doing.*

--local *host*
Local host name or IP address for bind. If specified, OpenVPN will bind to this address only. If unspecified, OpenVPN will bind to all interfaces.

--lport *port*
Set local TCP/UDP port number or name. Cannot be used together with **--nobind** option.

--mark *value*
Mark encrypted packets being sent with value. The mark value can be matched in policy routing and packetfilter rules. This option is only supported in Linux and does nothing on other operating systems.

--mode *m*
Set OpenVPN major mode. By default, OpenVPN runs in point-to-point mode (**p2p**). OpenVPN 2.0 introduces a new mode (**server**) which implements a multi-client server capability.

--mssfix *max*
Announce to TCP sessions running over the tunnel that they should limit their send packet sizes such that after OpenVPN has encapsulated them, the resulting UDP packet size that OpenVPN sends to its peer will not exceed **max** bytes. The default value is **1450**.

The **max** parameter is interpreted in the same way as the **--link-mtu** parameter, i.e. the UDP packet size after encapsulation overhead has been added in, but not including the UDP header itself. Resulting packet would be at most 28 bytes larger for IPv4 and 48 bytes for IPv6 (20/40 bytes for IP header and 8 bytes for UDP header). Default value of 1450 allows IPv4 packets to be transmitted over a link with MTU 1473 or higher without IP level fragmentation.

The **--mssfix** option only makes sense when you are using the UDP protocol for OpenVPN peer-to-peer communication, i.e. **--proto udp**.

--mssfix and **--fragment** can be ideally used together, where **--mssfix** will try to keep TCP from needing packet fragmentation in the first place, and if big packets come through anyhow (from protocols other than TCP), **--fragment** will internally fragment them.

Both **--fragment** and **--mssfix** are designed to work around cases where Path MTU discovery is broken on the network path between OpenVPN peers.

The usual symptom of such a breakdown is an OpenVPN connection which successfully starts, but then stalls during active usage.

If **--fragment** and **--mssfix** are used together, **--mssfix** will take its default **max** parameter from the **--fragment max** option.

Therefore, one could lower the maximum UDP packet size to 1300 (a good first try for solving MTU-related connection problems) with the following options:

```
--tun-mtu 1500 --fragment 1300 --mssfix
```

−−mtu−disc *type*
Should we do Path MTU discovery on TCP/UDP channel? Only supported on OSes such as Linux that supports the necessary system call to set.

Valid types:

no Never send DF (Don't Fragment) frames

maybe Use per−route hints

yes Always DF (Don't Fragment)

−−mtu−test
To empirically measure MTU on connection startup, add the **−−mtu−test** option to your configuration. OpenVPN will send ping packets of various sizes to the remote peer and measure the largest packets which were successfully received. The **−−mtu−test** process normally takes about 3 minutes to complete.

−−nobind
Do not bind to local address and port. The IP stack will allocate a dynamic port for returning packets. Since the value of the dynamic port could not be known in advance by a peer, this option is only suitable for peers which will be initiating connections by using the −−remote option.

−−passtos
Set the TOS field of the tunnel packet to what the payload's TOS is.

−−ping *n*
Ping remote over the TCP/UDP control channel if no packets have been sent for at least **n** seconds (specify **−−ping** on both peers to cause ping packets to be sent in both directions since OpenVPN ping packets are not echoed like IP ping packets). When used in one of OpenVPN's secure modes (where **−−secret**, **−−tls−server** or **−−tls−client** is specified), the ping packet will be cryptographically secure.

This option has two intended uses:

1. Compatibility with stateful firewalls. The periodic ping will ensure that a stateful firewall rule which allows OpenVPN UDP packets to pass will not time out.
2. To provide a basis for the remote to test the existence of its peer using the **−−ping−exit** option.

−−ping−exit *n*
Causes OpenVPN to exit after **n** seconds pass without reception of a ping or other packet from remote. This option can be combined with **−−inactive**, **−−ping** and **−−ping−exit** to create a two−tiered inactivity disconnect.

For example,

```
openvpn [options...] −−inactive 3600 −−ping 10 −−ping−exit 60
```

when used on both peers will cause OpenVPN to exit within 60 seconds if its peer disconnects, but will exit after one hour if no actual tunnel data is exchanged.

−−ping−restart *n*
Similar to **−−ping−exit**, but trigger a **SIGUSR1** restart after **n** seconds pass without reception of a ping or other packet from remote.

This option is useful in cases where the remote peer has a dynamic IP address and a low−TTL DNS name is used to track the IP address using a service such as *https://www.nsupdate.info/* + a dynamic DNS client such as **ddclient**.

If the peer cannot be reached, a restart will be triggered, causing the hostname used with **−−remote** to be re−resolved (if **−−resolv−retry** is also specified).

In server mode, **−−ping−restart**, **−−inactive** or any other type of internally generated signal will always be applied to individual client instance objects, never to whole server itself. Note also in server mode that any internally generated signal which would normally cause a restart, will cause the deletion of the client instance object instead.

In client mode, the **−−ping−restart** parameter is set to 120 seconds by default. This default will hold until the client pulls a replacement value from the server, based on the **−−keepalive** setting in the server configuration. To disable the 120 second default, set **−−ping−restart 0** on the client.

See the signals section below for more information on **SIGUSR1**.

Note that the behavior of **SIGUSR1** can be modified by the **−−persist−tun**, **−−persist−key**, **−−persist−local−ip** and **−−persist−remote−ip** options.

Also note that **−−ping−exit** and **−−ping−restart** are mutually exclusive and cannot be used together.

−−ping−timer−rem
Run the **−−ping−exit** / **−−ping−restart** timer only if we have a remote address. Use this option if you are starting the daemon in listen mode (i.e. without an explicit **−−remote** peer), and you don't want to start clocking timeouts until a remote peer connects.

−−proto *p*
Use protocol **p** for communicating with remote host. **p** can be **udp**, **tcp−client**, or **tcp−server**.

The default protocol is **udp** when **−−proto** is not specified.

For UDP operation, **−−proto udp** should be specified on both peers.

For TCP operation, one peer must use **−−proto tcp−server** and the other must use **−−proto tcp−client**. A peer started with **tcp−server** will wait indefinitely for an incoming connection. A peer started with **tcp−client** will attempt to connect, and if that fails, will sleep for 5 seconds (adjustable via the **−−connect−retry** option) and try again infinite or up to N retries (adjustable via the **−−connect−retry−max** option). Both TCP client and server will simulate a SIGUSR1 restart signal if either side resets the connection.

OpenVPN is designed to operate optimally over UDP, but TCP capability is provided for situations where UDP cannot be used. In comparison with UDP, TCP will usually be somewhat less efficient and less robust when used over unreliable or congested networks.

This article outlines some of problems with tunneling IP over TCP: *http://sites.inka.de/sites/bigred/devel/tcp−tcp.html*

There are certain cases, however, where using TCP may be advantageous from a security and robustness perspective, such as tunneling non−IP or application−level UDP protocols, or tunneling protocols which don't possess a built−in reliability layer.

−−port *port*
TCP/UDP port number or port name for both local and remote (sets both **−−lport** and **−−rport** options to given port). The current default of 1194 represents the official IANA port number assignment for OpenVPN and has been used since version 2.0−beta17. Previous versions used port 5000 as the default.

−−rport *port*
Set TCP/UDP port number or name used by the **−−remote** option. The port can also be set directly using the **−−remote** option.

−−replay−window *args*
Modify the replay protection sliding−window size and time window.

Valid syntax:

```
replay-window n [t]
```

Use a replay protection sliding−window of size **n** and a time window of **t** seconds.

By default **n** is 64 (the IPSec default) and **t** is 15 seconds.

This option is only relevant in UDP mode, i.e. when either **−−proto udp** is specified, or no **−−proto** option is specified.

When OpenVPN tunnels IP packets over UDP, there is the possibility that packets might be dropped or delivered out of order. Because OpenVPN, like IPSec, is emulating the physical network layer, it will accept an out−of−order packet sequence, and will deliver such packets in the same order they were received to the TCP/IP protocol stack, provided they satisfy several constraints.

a. The packet cannot be a replay (unless **−−no−replay** is specified, which disables replay protection altogether).

b. If a packet arrives out of order, it will only be accepted if the difference between its sequence number and the highest sequence number received so far is less than **n**.

c. If a packet arrives out of order, it will only be accepted if it arrives no later than **t** seconds after any packet containing a higher sequence number.

If you are using a network link with a large pipeline (meaning that the product of bandwidth and latency is high), you may want to use a larger value for **n**. Satellite links in particular often require this.

If you run OpenVPN at **−−verb 4**, you will see the message "Replay−window backtrack occurred [x]" every time the maximum sequence number backtrack seen thus far increases. This can be used to calibrate **n**.

There is some controversy on the appropriate method of handling packet reordering at the security layer.

Namely, to what extent should the security layer protect the encapsulated protocol from attacks which masquerade as the kinds of normal packet loss and reordering that occur over IP networks?

The IPSec and OpenVPN approach is to allow packet reordering within a certain fixed sequence number window.

OpenVPN adds to the IPSec model by limiting the window size in time as well as sequence space.

OpenVPN also adds TCP transport as an option (not offered by IPSec) in which case OpenVPN can adopt a very strict attitude towards message deletion and reordering: Don't allow it. Since TCP guarantees reliability, any packet loss or reordering event can be assumed to be an attack.

In this sense, it could be argued that TCP tunnel transport is preferred when tunneling non−IP or

UDP application protocols which might be vulnerable to a message deletion or reordering attack which falls within the normal operational parameters of IP networks.

So I would make the statement that one should never tunnel a non−IP protocol or UDP application protocol over UDP, if the protocol might be vulnerable to a message deletion or reordering attack that falls within the normal operating parameters of what is to be expected from the physical IP layer. The problem is easily fixed by simply using TCP as the VPN transport layer.

−−replay−persist *file*
Persist replay−protection state across sessions using **file** to save and reload the state.

This option will strengthen protection against replay attacks, especially when you are using OpenVPN in a dynamic context (such as with **−−inetd**) when OpenVPN sessions are frequently started and stopped.

This option will keep a disk copy of the current replay protection state (i.e. the most recent packet timestamp and sequence number received from the remote peer), so that if an OpenVPN session is stopped and restarted, it will reject any replays of packets which were already received by the prior session.

This option only makes sense when replay protection is enabled (the default) and you are using either **−−secret** (shared−secret key mode) or TLS mode with **−−tls−auth**.

−−socket−flags *flags*
Apply the given flags to the OpenVPN transport socket. Currently, only **TCP_NODELAY** is supported.

The **TCP_NODELAY** socket flag is useful in TCP mode, and causes the kernel to send tunnel packets immediately over the TCP connection without trying to group several smaller packets into a larger packet. This can result in a considerably improvement in latency.

This option is pushable from server to client, and should be used on both client and server for maximum effect.

−−tcp−nodelay
This macro sets the **TCP_NODELAY** socket flag on the server as well as pushes it to connecting clients. The **TCP_NODELAY** flag disables the Nagle algorithm on TCP sockets causing packets to be transmitted immediately with low latency, rather than waiting a short period of time in order to aggregate several packets into a larger containing packet. In VPN applications over TCP, **TCP_NODELAY** is generally a good latency optimization.

The macro expands as follows:

```
if mode server:
    socket−flags TCP_NODELAY
    push "socket−flags TCP_NODELAY"
```

Virtual Network Adapter (VPN interface)
Options in this section relates to configuration of the virtual tun/tap network interface, including setting the VPN IP address and network routing.

−−bind−dev *device*
(Linux only) Set **device** to bind the server socket to a *Virtual Routing and Forwarding* device

−−block−ipv6
On the client, instead of sending IPv6 packets over the VPN tunnel, all IPv6 packets are answered with an ICMPv6 no route host message. On the server, all IPv6 packets from clients are answered with an ICMPv6 no route to host message. This options is intended for cases when IPv6 should be

blocked and other options are not available. **−−block−ipv6** will use the remote IPv6 as source address of the ICMPv6 packets if set, otherwise will use **fe80::7** as source address.

For this option to make sense you actually have to route traffic to the tun interface. The following example config block would send all IPv6 traffic to OpenVPN and answer all requests with no route to host, effectively blocking IPv6 (to avoid IPv6 connections from dual−stacked clients leaking around IPv4−only VPN services).

Client config

```
--ifconfig-ipv6 fd15:53b6:dead::2/64 fd15:53b6:dead::1
--redirect-gateway ipv6
--block-ipv6
```

Server config

Push a "valid" ipv6 config to the client and block on the server

```
--push "ifconfig-ipv6 fd15:53b6:dead::2/64 fd15:53b6:dead::1"
--push "redirect-gateway ipv6"
--block-ipv6
```

Note: this option does not influence traffic sent from the server towards the client (neither on the server nor on the client side). This is not seen as necessary, as such traffic can be most easily avoided by not configuring IPv6 on the server tun, or setting up a server−side firewall rule.

−−dev *device*

TUN/TAP virtual network device which can be **tunX**, **tapX**, **null** or an arbitrary name string (**X** can be omitted for a dynamic device.)

See examples section below for an example on setting up a TUN device.

You must use either tun devices on both ends of the connection or tap devices on both ends. You cannot mix them, as they represent different underlying network layers:

tun devices encapsulate IPv4 or IPv6 (OSI Layer 3)

tap devices encapsulate Ethernet 802.3 (OSI Layer 2).

Valid syntaxes:

```
dev tun2
dev tap4
dev ovpn
```

When the device name starts with **tun** or **tap**, the device type is extracted automatically. Otherwise the **−−dev−type** option needs to be added as well.

−−dev−node *node*

Explicitly set the device node rather than using **/dev/net/tun**, **/dev/tun**, **/dev/tap**, etc. If OpenVPN cannot figure out whether **node** is a TUN or TAP device based on the name, you should also specify **−−dev−type tun** or **−−dev−type tap**.

Under Mac OS X this option can be used to specify the default tun implementation. Using **−−dev−node utun** forces usage of the native Darwin tun kernel support. Use **−−dev−node utunN** to select a specific utun instance. To force using the **tun.kext** (**/dev/tunX**) use **−−dev−node tun**. When not specifying a **−−dev−node** option openvpn will first try to open utun, and fall back to tun.kext.

On Windows systems, select the TAP−Win32 adapter which is named **node** in the Network Connections Control Panel or the raw GUID of the adapter enclosed by braces. The **−−show−adapters** option under Windows can also be used to enumerate all available TAP−Win32 adapters and will show both the network connections control panel name and the GUID for each TAP−Win32 adapter.

−−dev−type *device−type*
Which device type are we using? **device−type** should be **tun** (OSI Layer 3) or **tap** (OSI Layer 2). Use this option only if the TUN/TAP device used with **−−dev** does not begin with **tun** or **tap**.

−−dhcp−option *args*
Set additional network parameters on supported platforms. May be specified on the client or pushed from the server. On Windows these options are handled by the **tap−windows6** driver by default or directly by OpenVPN if dhcp is disabled or the **wintun** driver is in use. The **OpenVPN for Android** client also handles them internally.

On all other platforms these options are only saved in the client's environment under the name **foreign_option_{n}** before the **−−up** script is called. A plugin or an **−−up** script must be used to pick up and interpret these as required. Many Linux distributions include such scripts and some third−party user interfaces such as tunnelblick also come with scripts that process these options.

Valid syntax:

```
dhcp-options type [parm]
```

DOMAIN name
Set Connection−specific DNS Suffix to **name**.

ADAPTER_DOMAIN_SUFFIX name
Alias to **DOMAIN**. This is a compatibility option, it should not be used in new deployments.

DOMAIN−SEARCH name
Add **name** to the domain search list. Repeat this option to add more entries. Up to 10 domains are supported.

DNS address
Set primary domain name server IPv4 or IPv6 address. Repeat this option to set secondary DNS server addresses.

Note: DNS IPv6 servers are currently set using netsh (the existing DHCP code can only do IPv4 DHCP, and that protocol only permits IPv4 addresses anywhere). The option will be put into the environment, so an **−−up** script could act upon it if needed.

WINS address
Set primary WINS server address (NetBIOS over TCP/IP Name Server). Repeat this option to set secondary WINS server addresses.

NBDD address
Set primary NBDD server address (NetBIOS over TCP/IP Datagram Distribution Server). Repeat this option to set secondary NBDD server addresses.

NTP address
Set primary NTP server address (Network Time Protocol). Repeat this option to set secondary NTP server addresses.

NBT type
Set NetBIOS over TCP/IP Node type. Possible options:

1 b−node (broadcasts)

2 p−node (point−to−point name queries to a WINS server)

4 m−node (broadcast then query name server)

8 h−node (query name server, then broadcast).

NBS scope−id

Set NetBIOS over TCP/IP Scope. A NetBIOS Scope ID provides an extended naming service for the NetBIOS over TCP/IP (Known as NBT) module. The primary purpose of a NetBIOS scope ID is to isolate NetBIOS traffic on a single network to only those nodes with the same NetBIOS scope ID. The NetBIOS scope ID is a character string that is appended to the NetBIOS name. The NetBIOS scope ID on two hosts must match, or the two hosts will not be able to communicate. The NetBIOS Scope ID also allows computers to use the same computer name, as they have different scope IDs. The Scope ID becomes a part of the NetBIOS name, making the name unique. (This description of NetBIOS scopes courtesy of *NeonSurge@abyss.com*)

DISABLE−NBT

Disable Netbios−over−TCP/IP.

−−ifconfig *args*

Set TUN/TAP adapter parameters. It requires the *IP address* of the local VPN endpoint. For TUN devices in point−to−point mode, the next argument must be the VPN IP address of the remote VPN endpoint. For TAP devices, or TUN devices used with **−−topology subnet**, the second argument is the subnet mask of the virtual network segment which is being created or connected to.

For TUN devices, which facilitate virtual point−to−point IP connections (when used in **−−topology net30** or **p2p** mode), the proper usage of **−−ifconfig** is to use two private IP addresses which are not a member of any existing subnet which is in use. The IP addresses may be consecutive and should have their order reversed on the remote peer. After the VPN is established, by pinging**r n**, you will be pinging across the VPN.

For TAP devices, which provide the ability to create virtual ethernet segments, or TUN devices in **−−topology subnet** mode (which create virtual "multipoint networks"), **−−ifconfig** is used to set an IP address and subnet mask just as a physical ethernet adapter would be similarly configured. If you are attempting to connect to a remote ethernet bridge, the IP address and subnet should be set to values which would be valid on the the bridged ethernet segment (note also that DHCP can be used for the same purpose).

This option, while primarily a proxy for the **ifconfig**(8) command, is designed to simplify TUN/TAP tunnel configuration by providing a standard interface to the different ifconfig implementations on different platforms.

−−ifconfig parameters which are IP addresses can also be specified as a DNS or /etc/hosts file resolvable name.

For TAP devices, **−−ifconfig** should not be used if the TAP interface will be getting an IP address lease from a DHCP server.

Examples:

```
# tun device in net30/p2p mode
ifconfig 10.8.0.2 10.8.0.1

# tun/tap device in subnet mode
ifconfig 10.8.0.2 255.255.255.0
```

−−ifconfig−ipv6 *args*
Configure an IPv6 address on the *tun* device.

Valid syntax:

```
ifconfig−ipv6 ipv6addr/bits [ipv6remote]
```

The **ipv6addr/bits** argument is the IPv6 address to use. The second parameter is used as route target for **−−route−ipv6** if no gateway is specified.

The **−−topology** option has no influence with **−−ifconfig−ipv6**

−−ifconfig−noexec
Don't actually execute ifconfig/netsh commands, instead pass **−−ifconfig** parameters to scripts using environmental variables.

−−ifconfig−nowarn
Don't output an options consistency check warning if the **−−ifconfig** option on this side of the connection doesn't match the remote side. This is useful when you want to retain the overall benefits of the options consistency check (also see **−−disable−occ** option) while only disabling the ifconfig component of the check.

For example, if you have a configuration where the local host uses **−−ifconfig** but the remote host does not, use **−−ifconfig−nowarn** on the local host.

This option will also silence warnings about potential address conflicts which occasionally annoy more experienced users by triggering "false positive" warnings.

−−lladdr *address*
Specify the link layer address, more commonly known as the MAC address. Only applied to TAP devices.

−−persist−tun
Don't close and reopen TUN/TAP device or run up/down scripts across **SIGUSR1** or **−−ping−restart** restarts.

SIGUSR1 is a restart signal similar to **SIGHUP**, but which offers finer−grained control over reset options.

−−redirect−gateway *flags*
Automatically execute routing commands to cause all outgoing IP traffic to be redirected over the VPN. This is a client−side option.

This option performs three steps:

1. Create a static route for the **−−remote** address which forwards to the pre−existing default gateway. This is done so that **(3)** will not create a routing loop.
2. Delete the default gateway route.
3. Set the new default gateway to be the VPN endpoint address (derived either from **−−route−gateway** or the second parameter to **−−ifconfig** when **−−dev tun** is specified).

When the tunnel is torn down, all of the above steps are reversed so that the original default route is restored.

Option flags:

local Add the **local** flag if both OpenVPN peers are directly connected via a common subnet, such as with wireless. The **local** flag will cause step **(1)** above to be omitted.

autolocal
Try to automatically determine whether to enable **local** flag above.

def1 Use this flag to override the default gateway by using **0.0.0.0/1** and **128.0.0.0/1** rather than **0.0.0.0/0**. This has the benefit of overriding but not wiping out the original default gateway.

bypass−dhcp
Add a direct route to the DHCP server (if it is non−local) which bypasses the tunnel (Available on Windows clients, may not be available on non−Windows clients).

bypass−dns
Add a direct route to the DNS server(s) (if they are non−local) which bypasses the tunnel (Available on Windows clients, may not be available on non−Windows clients).

block−local
Block access to local LAN when the tunnel is active, except for the LAN gateway itself. This is accomplished by routing the local LAN (except for the LAN gateway address) into the tunnel.

ipv6 Redirect IPv6 routing into the tunnel. This works similar to the **def1** flag, that is, more specific IPv6 routes are added (**2000::/4**, **3000::/4**), covering the whole IPv6 unicast space.

!ipv4 Do not redirect IPv4 traffic − typically used in the flag pair **ipv6 !ipv4** to redirect IPv6−only.

−−**redirect−private** *flags*
Like −−**redirect−gateway**, but omit actually changing the default gateway. Useful when pushing private subnets.

−−**route** *args*
Add route to routing table after connection is established. Multiple routes can be specified. Routes will be automatically torn down in reverse order prior to TUN/TAP device close.

Valid syntaxes:

```
route network/IP
route network/IP netmask
route network/IP netmask gateway
route network/IP netmask gateway metric
```

This option is intended as a convenience proxy for the **route**(8) shell command, while at the same time providing portable semantics across OpenVPN's platform space.

netmask
defaults to **255.255.255.255** when not given

gateway
default taken from −−**route−gateway** or the second parameter to −−**ifconfig** when −−**dev tun** is specified.

metric default taken from −−**route−metric** if set, otherwise **0**.

The default can be specified by leaving an option blank or setting it to **default**.

The **network** and **gateway** parameters can also be specified as a DNS or **/etc/hosts** file resolvable name, or as one of three special keywords:

vpn_gateway
The remote VPN endpoint address (derived either from −−**route−gateway** or the second parameter to −−**ifconfig** when −−**dev tun** is specified).

net_gateway
The pre−existing IP default gateway, read from the routing table (not supported on all OSes).

remote_host
The **−−remote** address if OpenVPN is being run in client mode, and is undefined in server mode.

−−route−delay *args*
Valid syntaxes:

```
route−delay
route−delay n
route−delay n m
```

Delay **n** seconds (default **0**) after connection establishment, before adding routes. If **n** is **0**, routes will be added immediately upon connection establishment. If **−−route−delay** is omitted, routes will be added immediately after TUN/TAP device open and **−−up** script execution, before any **−−user** or **−−group** privilege downgrade (or **−−chroot** execution.)

This option is designed to be useful in scenarios where DHCP is used to set tap adapter addresses. The delay will give the DHCP handshake time to complete before routes are added.

On Windows, **−−route−delay** tries to be more intelligent by waiting **w** seconds (default **30** by default) for the TAP−Win32 adapter to come up before adding routes.

−−route−ipv6 *args*
Setup IPv6 routing in the system to send the specified IPv6 network into OpenVPN's *tun*.

Valid syntax:

```
route−ipv6 ipv6addr/bits [gateway] [metric]
```

The gateway parameter is only used for IPv6 routes across *tap* devices, and if missing, the **ipv6remote** field from **−−ifconfig−ipv6** or **−−route−ipv6−gateway** is used.

−−route−gateway *arg*
Specify a default *gateway* for use with **−−route**.

If **dhcp** is specified as the parameter, the gateway address will be extracted from a DHCP negotiation with the OpenVPN server−side LAN.

Valid syntaxes:

```
route−gateway gateway
route−gateway dhcp
```

−−route−ipv6−gateway *gw*
Specify a default gateway **gw** for use with **−−route−ipv6**.

−−route−metric *m*
Specify a default metric **m** for use with **−−route**.

−−route−noexec
Don't add or remove routes automatically. Instead pass routes to **−−route−up** script using environmental variables.

−−route−nopull
When used with **−−client** or **−−pull**, accept options pushed by server EXCEPT for routes, block−outside−dns and dhcp options like DNS servers.

When used on the client, this option effectively bars the server from adding routes to the client's routing table, however note that this option still allows the server to set the TCP/IP properties of the client's TUN/TAP interface.

−−topology *mode*
Configure virtual addressing topology when running in **−−dev tun** mode. This directive has no meaning in **−−dev tap** mode, which always uses a **subnet** topology.

If you set this directive on the server, the **−−server** and **−−server−bridge** directives will automatically push your chosen topology setting to clients as well. This directive can also be manually pushed to clients. Like the **−−dev** directive, this directive must always be compatible between client and server.

mode can be one of:

net30 Use a point−to−point topology, by allocating one /30 subnet per client. This is designed to allow point−to−point semantics when some or all of the connecting clients might be Windows systems. This is the default on OpenVPN 2.0.

p2p Use a point−to−point topology where the remote endpoint of the client's tun interface always points to the local endpoint of the server's tun interface. This mode allocates a single IP address per connecting client. Only use when none of the connecting clients are Windows systems.

subnet Use a subnet rather than a point−to−point topology by configuring the tun interface with a local IP address and subnet mask, similar to the topology used in **−−dev tap** and ethernet bridging mode. This mode allocates a single IP address per connecting client and works on Windows as well. Only available when server and clients are OpenVPN 2.1 or higher, or OpenVPN 2.0.x which has been manually patched with the **−−topology** directive code. When used on Windows, requires version 8.2 or higher of the TAP−Win32 driver. When used on *nix, requires that the tun driver supports an **ifconfig**(8) command which sets a subnet instead of a remote endpoint IP address.

Note: Using **−−topology subnet** changes the interpretation of the arguments of **−−ifconfig** to mean "address netmask", no longer "local remote".

−−tun−mtu *n*
Take the TUN device MTU to be **n** and derive the link MTU from it (default **1500**). In most cases, you will probably want to leave this parameter set to its default value.

The MTU (Maximum Transmission Units) is the maximum datagram size in bytes that can be sent unfragmented over a particular network path. OpenVPN requires that packets on the control and data channels be sent unfragmented.

MTU problems often manifest themselves as connections which hang during periods of active usage.

It's best to use the **−−fragment** and/or **−−mssfix** options to deal with MTU sizing issues.

−−tun−mtu−extra *n*
Assume that the TUN/TAP device might return as many as **n** bytes more than the **−−tun−mtu** size on read. This parameter defaults to 0, which is sufficient for most TUN devices. TAP devices may introduce additional overhead in excess of the MTU size, and a setting of 32 is the default when TAP devices are used. This parameter only controls internal OpenVPN buffer sizing, so there is no

transmission overhead associated with using a larger value.

TUN/TAP standalone operations

These two standalone operations will require **−−dev** and optionally **−−user** and/or **−−group**.

−−mktun

(Standalone) Create a persistent tunnel on platforms which support them such as Linux. Normally TUN/TAP tunnels exist only for the period of time that an application has them open. This option takes advantage of the TUN/TAP driver's ability to build persistent tunnels that live through multiple instantiations of OpenVPN and die only when they are deleted or the machine is rebooted.

One of the advantages of persistent tunnels is that they eliminate the need for separate **−−up** and **−−down** scripts to run the appropriate **ifconfig**(8) and **route**(8) commands. These commands can be placed in the the same shell script which starts or terminates an OpenVPN session.

Another advantage is that open connections through the TUN/TAP−based tunnel will not be reset if the OpenVPN peer restarts. This can be useful to provide uninterrupted connectivity through the tunnel in the event of a DHCP reset of the peer's public IP address (see the **−−ipchange** option above).

One disadvantage of persistent tunnels is that it is harder to automatically configure their MTU value (see **−−link−mtu** and **−−tun−mtu** above).

On some platforms such as Windows, TAP−Win32 tunnels are persistent by default.

−−rmtun

(Standalone) Remove a persistent tunnel.

Virtual Routing and Forwarding

Options in this section relates to configuration of virtual routing and forwarding in combination with the underlying operating system.

As of today this is only supported on Linux, a kernel >= 4.9 is recommended.

This could come in handy when for example the external network should be only used as a means to connect to some VPN endpoints and all regular traffic should only be routed through any tunnel(s). This could be achieved by setting up a VRF and configuring the interface connected to the external network to be part of the VRF. The examples below will cover this setup.

Another option would be to put the tun/tap interface into a VRF. This could be done by an up−script which uses the **ip link set** command shown below.

VRF setup with iproute2

Create VRF **vrf_external** and map it to routing table **1023**

```
ip link add vrf_external type vrf table 1023
```

Move **eth0** into **vrf_external**

```
ip link set master vrf_external dev eth0
```

Any prefixes configured on **eth0** will be moved from the :code`main` routing table into routing table *1023*

VRF setup with ifupdown

For Debian based Distributions **ifupdown2** provides an almost drop−in replacement for **ifupdown** including VRFs and other features. A configuration for an interface **eth0** being part of VRF code:*vrf_external* could look like this:

```
auto eth0
iface eth0
    address 192.0.2.42/24
    address 2001:db8:08:15::42/64
    gateway 192.0.2.1
    gateway 2001:db8:08:15::1
    vrf vrf_external

auto vrf_external
iface vrf_external
    vrf-table 1023
```

OpenVPN configuration

The OpenVPN configuration needs to contain this line:

```
bind-dev vrf_external
```

Further reading

Wikipedia has nice page one VRFs: *https://en.wikipedia.org/wiki/Virtual_routing_and_forwarding*

This talk from the Network Track of FrOSCon 2018 provides an overview about advanced layer 2 and layer 3 features of Linux

- Slides: *https://www.slideshare.net/BarbarossaTM/l2l3-fr-fortgeschrittene-helle-und-dunkle-magie-im-linuxnetzwerk*
- Video (german): *https://media.ccc.de/v/froscon2018-2247-l2_l3_fur_fortgeschrittene_-_helle_und_dunkle_magie_im_linux-netz*

SCRIPTING INTEGRATION

OpenVPN can execute external scripts in various phases of the lifetime of the OpenVPN process.

Script Order of Execution

1. **--up**

   Executed after TCP/UDP socket bind and TUN/TAP open.

2. **--tls-verify**

   Executed when we have a still untrusted remote peer.

3. **--ipchange**

   Executed after connection authentication, or remote IP address change.

4. **--client-connect**

   Executed in **--mode server** mode immediately after client authentication.

5. **--route-up**

   Executed after connection authentication, either immediately after, or some number of seconds after as defined by the **--route-delay** option.

6. **--route-pre-down**

   Executed right before the routes are removed.

7. **--client-disconnect**

   Executed in **--mode server** mode on client instance shutdown.

8. **−−down**

   Executed after TCP/UDP and TUN/TAP close.

9. **−−learn−address**

   Executed in **−−mode server** mode whenever an IPv4 address/route or MAC address is added to OpenVPN's internal routing table.

10. **−−auth−user−pass−verify**

    Executed in **−−mode server** mode on new client connections, when the client is still untrusted.

SCRIPT HOOKS

−−auth−user−pass−verify *args*

Require the client to provide a username/password (possibly in addition to a client certificate) for authentication.

Valid syntax:

```
auth−user−pass−verify cmd method
```

OpenVPN will run command **cmd** to validate the username/password provided by the client.

cmd consists of a path to a script (or executable program), optionally followed by arguments. The path and arguments may be single− or double−quoted and/or escaped using a backslash, and should be separated by one or more spaces.

If **method** is set to **via−env**, OpenVPN will call **script** with the environmental variables **username** and **password** set to the username/password strings provided by the client. *Beware* that this method is insecure on some platforms which make the environment of a process publicly visible to other unprivileged processes.

If **method** is set to **via−file**, OpenVPN will write the username and password to the first two lines of a temporary file. The filename will be passed as an argument to **script**, and the file will be automatically deleted by OpenVPN after the script returns. The location of the temporary file is controlled by the **−−tmp−dir** option, and will default to the current directory if unspecified. For security, consider setting **−−tmp−dir** to a volatile storage medium such as **/dev/shm** (if available) to prevent the username/password file from touching the hard drive.

The script should examine the username and password, returning a success exit code (**0**) if the client's authentication request is to be accepted, or a failure code (**1**) to reject the client.

This directive is designed to enable a plugin−style interface for extending OpenVPN's authentication capabilities.

To protect against a client passing a maliciously formed username or password string, the username string must consist only of these characters: alphanumeric, underbar ('_'), dash ('−'), dot ('.'), or at ('@'). The password string can consist of any printable characters except for CR or LF. Any illegal characters in either the username or password string will be converted to underbar ('_').

Care must be taken by any user−defined scripts to avoid creating a security vulnerability in the way that these strings are handled. Never use these strings in such a way that they might be escaped or evaluated by a shell interpreter.

For a sample script that performs PAM authentication, see **sample−scripts/auth−pam.pl** in the

OpenVPN source distribution.

−−client−connect *cmd*
Run command **cmd** on client connection.

cmd consists of a path to a script (or executable program), optionally followed by arguments. The path and arguments may be single− or double−quoted and/or escaped using a backslash, and should be separated by one or more spaces.

The command is passed the common name and IP address of the just−authenticated client as environmental variables (see environmental variable section below). The command is also passed the pathname of a freshly created temporary file as the last argument (after any arguments specified in **cmd**), to be used by the command to pass dynamically generated config file directives back to OpenVPN.

If the script wants to generate a dynamic config file to be applied on the server when the client connects, it should write it to the file named by the last argument.

See the **−−client−config−dir** option below for options which can be legally used in a dynamically generated config file.

Note that the return value of **script** is significant. If **script** returns a non−zero error status, it will cause the client to be disconnected.

If a **−−client−connect** wants to defer the generating of the configuration then the script needs to use the **client_connect_deferred_file** and **client_connect_config_file** environment variables, and write status accordingly into these files. See the *Environmental Variables* section for more details.

−−client−disconnect *cmd*
Like **−−client−connect** but called on client instance shutdown. Will not be called unless the **−−client−connect** script and plugins (if defined) were previously called on this instance with successful (0) status returns.

The exception to this rule is if the **−−client−disconnect** command or plugins are cascaded, and at least one client−connect function succeeded, then ALL of the client−disconnect functions for scripts and plugins will be called on client instance object deletion, even in cases where some of the related client−connect functions returned an error status.

The **−−client−disconnect** command is not passed any extra arguments (only those arguments specified in cmd, if any).

−−down *cmd*
Run command **cmd** after TUN/TAP device close (post **−−user** UID change and/or **−−chroot**). **cmd** consists of a path to script (or executable program), optionally followed by arguments. The path and arguments may be single− or double−quoted and/or escaped using a backslash, and should be separated by one or more spaces.

Called with the same parameters and environmental variables as the **−−up** option above.

Note that if you reduce privileges by using **−−user** and/or **−−group**, your **−−down** script will also run at reduced privilege.

−−down−pre
Call **−−down** cmd/script before, rather than after, TUN/TAP close.

−−ipchange *cmd*
Run command **cmd** when our remote ip−address is initially authenticated or changes.

cmd consists of a path to a script (or executable program), optionally followed by arguments. The path and arguments may be single− or double−quoted and/or escaped using a backslash, and should be separated by one or more spaces.

When **cmd** is executed two arguments are appended after any arguments specified in **cmd** , as follows:

```
cmd ip address port number
```

Don't use **−−ipchange** in **−−mode server** mode. Use a **−−client−connect** script instead.

See the *Environmental Variables* section below for additional parameters passed as environmental variables.

If you are running in a dynamic IP address environment where the IP addresses of either peer could change without notice, you can use this script, for example, to edit the **/etc/hosts** file with the current address of the peer. The script will be run every time the remote peer changes its IP address.

Similarly if *our* IP address changes due to DHCP, we should configure our IP address change script (see man page for **dhcpcd**(8)) to deliver a **SIGHUP** or **SIGUSR1** signal to OpenVPN. OpenVPN will then re−establish a connection with its most recently authenticated peer on its new IP address.

−−learn−address *cmd*
Run command **cmd** to validate client virtual addresses or routes.

cmd consists of a path to a script (or executable program), optionally followed by arguments. The path and arguments may be single− or double−quoted and/or escaped using a backslash, and should be separated by one or more spaces.

Three arguments will be appended to any arguments in **cmd** as follows:

$1 − [operation]
"add", **"update"**, or **"delete"** based on whether or not the address is being added to, modified, or deleted from OpenVPN's internal routing table.

$2 − [address]
The address being learned or unlearned. This can be an IPv4 address such as **"198.162.10.14"**, an IPv4 subnet such as **"198.162.10.0/24"**, or an ethernet MAC address (when **−−dev tap** is being used) such as **"00:FF:01:02:03:04"**.

$3 − [common name]
The common name on the certificate associated with the client linked to this address. Only present for **"add"** or **"update"** operations, not **"delete"**.

On **"add"** or **"update"** methods, if the script returns a failure code (non−zero), OpenVPN will reject the address and will not modify its internal routing table.

Normally, the **cmd** script will use the information provided above to set appropriate firewall entries on the VPN TUN/TAP interface. Since OpenVPN provides the association between virtual IP or MAC address and the client's authenticated common name, it allows a user−defined script to configure firewall access policies with regard to the client's high−level common name, rather than the low level client virtual addresses.

−−route−up *cmd*
Run command **cmd** after routes are added, subject to **−−route−delay**.

cmd consists of a path to a script (or executable program), optionally followed by arguments. The path and arguments may be single− or double−quoted and/or escaped using a backslash, and should be separated by one or more spaces.

See the *Environmental Variables* section below for additional parameters passed as environmental variables.

−−route−pre−down *cmd*
Run command **cmd** before routes are removed upon disconnection.

cmd consists of a path to a script (or executable program), optionally followed by arguments. The path and arguments may be single− or double−quoted and/or escaped using a backslash, and should be separated by one or more spaces.

See the *Environmental Variables* section below for additional parameters passed as environmental variables.

−−setenv *args*
Set a custom environmental variable **name=value** to pass to script.

Valid syntaxes:

```
setenv name value
setenv FORWARD_COMPATIBLE 1
setenv opt config_option
```

By setting **FORWARD_COMPATIBLE** to **1**, the config file syntax checking is relaxed so that unknown directives will trigger a warning but not a fatal error, on the assumption that a given unknown directive might be valid in future OpenVPN versions.

This option should be used with caution, as there are good security reasons for having OpenVPN fail if it detects problems in a config file. Having said that, there are valid reasons for wanting new software features to gracefully degrade when encountered by older software versions.

It is also possible to tag a single directive so as not to trigger a fatal error if the directive isn't recognized. To do this, prepend the following before the directive: **setenv opt**

Versions prior to OpenVPN 2.3.3 will always ignore options set with the **setenv opt** directive.

See also **−−ignore−unknown−option**

−−setenv−safe *args*
Set a custom environmental variable **OPENVPN_name** to **value** to pass to scripts.

Valid syntaxes:

```
setenv−safe name value
```

This directive is designed to be pushed by the server to clients, and the prepending of **OPENVPN_** to the environmental variable is a safety precaution to prevent a **LD_PRELOAD** style attack from a malicious or compromised server.

−−tls−verify *cmd*
Run command **cmd** to verify the X509 name of a pending TLS connection that has otherwise passed all other tests of certification (except for revocation via **−−crl−verify** directive; the revocation test occurs after the **−−tls−verify** test).

cmd should return **0** to allow the TLS handshake to proceed, or **1** to fail.

cmd consists of a path to a script (or executable program), optionally followed by arguments. The path and arguments may be single− or double−quoted and/or escaped using a backslash, and should be separated by one or more spaces.

When **cmd** is executed two arguments are appended after any arguments specified in **cmd**, as follows:

```
cmd certificate_depth subject
```

These arguments are, respectively, the current certificate depth and the X509 subject distinguished name (dn) of the peer.

This feature is useful if the peer you want to trust has a certificate which was signed by a certificate authority who also signed many other certificates, where you don't necessarily want to trust all of them, but rather be selective about which peer certificate you will accept. This feature allows you to write a script which will test the X509 name on a certificate and decide whether or not it should be accepted. For a simple perl script which will test the common name field on the certificate, see the file **verify−cn** in the OpenVPN distribution.

See the *Environmental Variables* section below for additional parameters passed as environmental variables.

−−up *cmd*
Run command **cmd** after successful TUN/TAP device open (pre **−−user** UID change).

cmd consists of a path to a script (or executable program), optionally followed by arguments. The path and arguments may be single− or double−quoted and/or escaped using a backslash, and should be separated by one or more spaces.

The up command is useful for specifying route commands which route IP traffic destined for private subnets which exist at the other end of the VPN connection into the tunnel.

For **−−dev tun** execute as:

```
cmd tun_dev tun_mtu link_mtu ifconfig_local_ip ifconfig_remote_ip [init
```

For **−−dev tap** execute as:

```
cmd tap_dev tap_mtu link_mtu ifconfig_local_ip ifconfig_netmask [init | r
```

See the *Environmental Variables* section below for additional parameters passed as environmental variables.

Note that if **cmd** includes arguments, all OpenVPN−generated arguments will be appended to them to build an argument list with which the executable will be called.

Typically, **cmd** will run a script to add routes to the tunnel.

Normally the up script is called after the TUN/TAP device is opened. In this context, the last command line parameter passed to the script will be *init*. If the **--up-restart** option is also used, the up script will be called for restarts as well. A restart is considered to be a partial reinitialization of OpenVPN where the TUN/TAP instance is preserved (the **--persist-tun** option will enable such preservation). A restart can be generated by a SIGUSR1 signal, a**--ping-r estart** timeout, or a connection reset when the TCP protocol is enabled with the **--proto** option. If a restart occurs, and **--up-restart** has been specified, the up script will be called with *restart* as the last parameter.

NOTE: On restart, OpenVPN will not pass the full set of environment variables to the script. Namely, everything related to routing and gateways will not be passed, as nothing needs to be done anyway - all the routing setup is already in place. Additionally, the up-restart script will run with the downgraded UID/GID settings (if configured).

The following standalone example shows how the **--up** script can be called in both an initialization and restart context. (*NOTE:* for security reasons, don't run the following example unless UDP port 9999 is blocked by your firewall. Also, the example will run indefinitely, so you should abort with control-c).

```
openvpn --dev tun --port 9999 --verb 4 --ping-restart 10 \
        --up 'echo up' --down 'echo down' --persist-tun  \
        --up-restart
```

Note that OpenVPN also provides the **--ifconfig** option to automatically ifconfig the TUN device, eliminating the need to define an **--up** script, unless you also want to configure routes in the **--up** script.

If **--ifconfig** is also specified, OpenVPN will pass the ifconfig local and remote endpoints on the command line to the **--up** script so that they can be used to configure routes such as:

```
route add -net 10.0.0.0 netmask 255.255.255.0 gw $5
```

--up-delay

Delay TUN/TAP open and possible **--up** script execution until after TCP/UDP connection establishment with peer.

In **--proto udp** mode, this option normally requires the use of **--ping** to allow connection initiation to be sensed in the absence of tunnel data, since UDP is a "connectionless" protocol.

On Windows, this option will delay the TAP-Win32 media state transitioning to "connected" until connection establishment, i.e. the receipt of the first authenticated packet from the peer.

--up-restart

Enable the **--up** and **--down** scripts to be called for restarts as well as initial program start. This option is described more fully above in the **--up** option documentation.

String Types and Remapping

In certain cases, OpenVPN will perform remapping of characters in strings. Essentially, any characters outside the set of permitted characters for each string type will be converted to underbar ('_').

Q: Why is string remapping necessary?

It's an important security feature to prevent the malicious coding of strings from untrusted sources to be passed as parameters to scripts, saved in the environment, used as a common name, translated to a filename, etc.

Q: Can string remapping be disabled?

Yes, by using the **--no-name-remapping** option, however this should be considered an advanced option.

Here is a brief rundown of OpenVPN's current string types and the permitted character class for each string:

X509 Names
: Alphanumeric, underbar ('_'), dash ('−'), dot ('.'), at ('@'), colon (':'), slash ('/'), and equal ('='). Alphanumeric is defined as a character which will cause the C library isalnum() function to return true.

Common Names
: Alphanumeric, underbar ('_'), dash ('−'), dot ('.'), and at ('@').

−−auth−user−pass username
: Same as Common Name, with one exception: starting with OpenVPN 2.0.1, the username is passed to the **OPENVPN_PLUGIN_AUTH_USER_PASS_VERIFY** plugin in its raw form, without string remapping.

−−auth−user−pass password
: Any "printable" character except CR or LF. Printable is defined to be a character which will cause the C library isprint() function to return true.

−−client−config−dir filename as derived from common name or`username
: Alphanumeric, underbar ('_'), dash ('−'), and dot ('.') except for "." or ".." as standalone strings. As of v2.0.1−rc6, the at ('@') character has been added as well for compatibility with the common name character class.

Environmental variable names
: Alphanumeric or underbar ('_').

Environmental variable values
: Any printable character.

For all cases, characters in a string which are not members of the legal character class for that string type will be remapped to underbar ('_').

Environmental Variables

Once set, a variable is persisted indefinitely until it is reset by a new value or a restart,

As of OpenVPN 2.0−beta12, in server mode, environmental variables set by OpenVPN are scoped according to the client objects they are associated with, so there should not be any issues with scripts having access to stale, previously set variables which refer to different client instances.

bytes_received
: Total number of bytes received from client during VPN session. Set prior to execution of the **−−client−disconnect** script.

bytes_sent
: Total number of bytes sent to client during VPN session. Set prior to execution of the **−−client−disconnect** script.

client_connect_config_file
: The path to the configuration file that should be written to by the **−−client−connect** script (optional, if per−session configuration is desired). This is the same file name as passed via command line argument on the call to the **−−client−connect** script.

client_connect_deferred_file
: This file can be optionally written to in order to to communicate a status code of the **−−client−connect** script or plgin. Only the first character in the file is relevant. It must be either**1** to indicate normal script execution, **0** indicates an error (in the same way that a non zero exit status does) or **2** to indicate that the script deferred returning the config file.

  For deferred (background) handling, the script or plugin MUST write **2** to the file to indicate the deferral and then return with exit code **0** to signal **deferred handler started OK**.

A background process or similar must then take care of writing the configuration to the file indicated by the **client_connect_config_file** environment variable and when finished, write the a **1** to this file (or **0** in case of an error).

The absence of any character in the file when the script finishes executing is interpreted the same as **1**. This allows scripts that are not written to support the defer mechanism to be used unmodified.

common_name
The X509 common name of an authenticated client. Set prior to execution of **−−client−connect**, **−−client−disconnect** and **−−auth−user−pass−verify** scripts.

config Name of first **−−config** file. Set on program initiation and reset on SIGHUP.

daemon
Set to "1" if the **−−daemon** directive is specified, or "0" otherwise. Set on program initiation and reset on SIGHUP.

daemon_log_redirect
Set to "1" if the **−−log** or **−−log−append** directives are specified, or "0" otherwise. Set on program initiation and reset on SIGHUP.

dev The actual name of the TUN/TAP device, including a unit number if it exists. Set prior to **−−up** or **−−down** script execution.

dev_idx
On Windows, the device index of the TUN/TAP adapter (to be used in netsh.exe calls which sometimes just do not work right with interface names). Set prior to **−−up** or **−−down** script execution.

foreign_option_{n}
An option pushed via **−−push** to a client which does not natively support it, such as **−−dhcp−option** on a non−Windows system, will be recorded to this environmental variable sequence prior to **−−up** script execution.

ifconfig_broadcast
The broadcast address for the virtual ethernet segment which is derived from the **−−ifconfig** option when **−−dev tap** is used. Set prior to OpenVPN calling the **ifconfig** or **netsh** (windows version of ifconfig) commands which normally occurs prior to **−−up** script execution.

ifconfig_ipv6_local
The local VPN endpoint IPv6 address specified in the **−−ifconfig−ipv6** option (first parameter). Set prior to OpenVPN calling the **ifconfig** or code:*netsh* (windows version of ifconfig) commands which normally occurs prior to **−−up** script execution.

ifconfig_ipv6_netbits
The prefix length of the IPv6 network on the VPN interface. Derived from the /nnn parameter of the IPv6 address in the **−−ifconfig−ipv6** option (first parameter). Set prior to OpenVPN calling the **ifconfig** or **netsh** (windows version of ifconfig) commands which normally occurs prior to **−−up** script execution.

ifconfig_ipv6_remote
The remote VPN endpoint IPv6 address specified in the **−−ifconfig−ipv6** option (second parameter). Set prior to OpenVPN calling the **ifconfig** or **netsh** (windows version of ifconfig) commands which normally occurs prior to **−−up** script execution.

ifconfig_local
The local VPN endpoint IP address specified in the **−−ifconfig** option (first parameter). Set prior to OpenVPN calling the **ifconfig** or **netsh** (windows version of ifconfig) commands which normally occurs prior to **−−up** script execution.

ifconfig_remote
The remote VPN endpoint IP address specified in the **--ifconfig** option (second parameter) when **--dev tun** is used. Set prior to OpenVPN calling the **ifconfig** or **netsh** (windows version of ifconfig) commands which normally occurs prior to **--up** script execution.

ifconfig_netmask
The subnet mask of the virtual ethernet segment that is specified as the second parameter to **--ifconfig** when **--dev tap** is being used. Set prior to OpenVPN calling the **ifconfig** or **netsh** (windows version of ifconfig) commands which normally occurs prior to **--up** script execution.

ifconfig_pool_local_ip
The local virtual IP address for the TUN/TAP tunnel taken from an **--ifconfig-push** directive if specified, or otherwise from the ifconfig pool (controlled by the **--ifconfig-pool** config file directive). Only set for **--dev tun** tunnels. This option is set on the server prior to execution of the **--client-connect** and **--client-disconnect** scripts.

ifconfig_pool_netmask
The virtual IP netmask for the TUN/TAP tunnel taken from an **--ifconfig-push** directive if specified, or otherwise from the ifconfig pool (controlled by the **--ifconfig-pool** config file directive). Only set for **--dev tap** tunnels. This option is set on the server prior to execution of the **--client-connect** and **--client-disconnect** scripts.

ifconfig_pool_remote_ip
The remote virtual IP address for the TUN/TAP tunnel taken from an **--ifconfig-push** directive if specified, or otherwise from the ifconfig pool (controlled by the **--ifconfig-pool** config file directive). This option is set on the server prior to execution of the **--client-connect** and **--client-disconnect** scripts.

link_mtu
The maximum packet size (not including the IP header) of tunnel data in UDP tunnel transport mode. Set prior to **--up** or **--down** script execution.

local The **--local** parameter. Set on program initiation and reset on SIGHUP.

local_port
The local port number or name, specified by **--port** or **--lport**. Set on program initiation and reset on SIGHUP.

password
The password provided by a connecting client. Set prior to **--auth-user-pass-verify** script execution only when the **via-env** modifier is specified, and deleted from the environment after the script returns.

proto The **--proto** parameter. Set on program initiation and reset on SIGHUP.

remote_{n}
The **--remote** parameter. Set on program initiation and reset on SIGHUP.

remote_port_{n}
The remote port number, specified by **--port** or **--rport**. Set on program initiation and reset on SIGHUP.

route_net_gateway
The pre-existing default IP gateway in the system routing table. Set prior to **--up** script execution.

route_vpn_gateway
The default gateway used by **--route** options, as specified in either the **--route-gateway** option or the second parameter to **--ifconfig** when **--dev tun** is specified. Set prior to **--up** script execution.

route_{parm}_{n}
A set of variables which define each route to be added, and are set prior to **−−up** script execution.

parm will be one of **network**, **netmask"**, **gateway**, or **metric**.

n is the OpenVPN route number, starting from 1.

If the network or gateway are resolvable DNS names, their IP address translations will be recorded rather than their names as denoted on the command line or configuration file.

route_ipv6_{parm}_{n}
A set of variables which define each IPv6 route to be added, and are set prior to **−−up** script execution.

parm will be one of **network**, **gateway** or **metric**. **route_ipv6_network_{n}** contains **netmask** as **/nnn**, unlike IPv4 where it is passed in a separate environment variable.

n is the OpenVPN route number, starting from 1.

If the network or gateway are resolvable DNS names, their IP address translations will be recorded rather than their names as denoted on the command line or configuration file.

peer_cert
Temporary file name containing the client certificate upon connection. Useful in conjunction with **−−tls−verify**.

script_context
Set to "init" or "restart" prior to up/down script execution. For more information, see documentation for **−−up**.

script_type
Prior to execution of any script, this variable is set to the type of script being run. It can be one of the following: **up**, **down**, **ipchange**, **route−up**, **tls−verify**, **auth−user−pass−verify**, **client−connect**, **client−disconnect** or **learn−address**. Set prior to execution of any script.

signal The reason for exit or restart. Can be one of **sigusr1**, **sighup**, **sigterm**, **sigint**, **inactive** (controlled by **−−inactive** option), **ping−exit** (controlled by **−−ping−exit** option), **ping−restart** (controlled by **−−ping−restart** option), **connection−reset** (triggered on TCP connection reset), **error** or **unknown** (unknown signal). This variable is set just prior to down script execution.

time_ascii
Client connection timestamp, formatted as a human−readable time string. Set prior to execution of the **−−client−connect** script.

time_duration
The duration (in seconds) of the client session which is now disconnecting. Set prior to execution of the **−−client−disconnect** script.

time_unix
Client connection timestamp, formatted as a unix integer date/time value. Set prior to execution of the **−−client−connect** script.

tls_digest_{n} / tls_digest_sha256_{n}
Contains the certificate SHA1 / SHA256 fingerprint, where **n** is the verification level. Only set for TLS connections. Set prior to execution of **−−tls−verify** script.

tls_id_{n}
A series of certificate fields from the remote peer, where **n** is the verification level. Only set for TLS connections. Set prior to execution of **−−tls−verify** script.

tls_serial_{n}
The serial number of the certificate from the remote peer, where **n** is the verification level. Only set for TLS connections. Set prior to execution of **--tls-verify** script. This is in the form of a decimal string like "933971680", which is suitable for doing serial-based OCSP queries (with OpenSSL, do not prepend "0x" to the string) If something goes wrong while reading the value from the certificate it will be an empty string, so your code should check that. See the **contrib/OCSP_check/OCSP_check.sh** script for an example.

tls_serial_hex_{n}
Like **tls_serial_{n}**, but in hex form (e.g. **12:34:56:78:9A**).

tun_mtu
The MTU of the TUN/TAP device. Set prior to **--up** or **--down** script execution.

trusted_ip / trusted_ip6)
Actual IP address of connecting client or peer which has been authenticated. Set prior to execution of **--ipchange**, **--client-connect** and **--client-disconnect** scripts. If using ipv6 endpoints (udp6, tcp6), **trusted_ip6** will be set instead.

trusted_port
Actual port number of connecting client or peer which has been authenticated. Set prior to execution of **--ipchange**, **--client-connect** and **--client-disconnect** scripts.

untrusted_ip / untrusted_ip6
Actual IP address of connecting client or peer which has not been authenticated yet. Sometimes used to *nmap* the connecting host in a **--tls-verify** script to ensure it is firewalled properly. Set prior to execution of **--tls-verify** and **--auth-user-pass-verify** scripts. If using ipv6 endpoints (udp6, tcp6), **untrusted_ip6** will be set instead.

untrusted_port
Actual port number of connecting client or peer which has not been authenticated yet. Set prior to execution of **--tls-verify** and **--auth-user-pass-verify** scripts.

username
The username provided by a connecting client. Set prior to **--auth-user-pass-verify** script execution only when the **via-env** modifier is specified.

X509_{n}_{subject_field}
An X509 subject field from the remote peer certificate, where **n** is the verification level. Only set for TLS connections. Set prior to execution of **--tls-verify** script. This variable is similar to **tls_id_{n}** except the component X509 subject fields are broken out, and no string remapping occurs on these field values (except for remapping of control characters to "_"). For example, the following variables would be set on the OpenVPN server using the sample client certificate in sample-keys (client.crt). Note that the verification level is 0 for the client certificate and 1 for the CA certificate.

```
X509_0_emailAddress=me@myhost.mydomain
X509_0_CN=Test-Client
X509_0_O=OpenVPN-TEST
X509_0_ST=NA
X509_0_C=KG
X509_1_emailAddress=me@myhost.mydomain
X509_1_O=OpenVPN-TEST
X509_1_L=BISHKEK
X509_1_ST=NA
X509_1_C=KG
```

Management Interface Options
OpenVPN provides a feature rich socket based management interface for both server and client mode operations.

−−management *args*
Enable a management server on a **socket−name** Unix socket on those platforms supporting it, or on a designated TCP port.

Valid syntaxes:

```
management socket−name unix          #
management socket−name unix pw−file  # (recommended)
management IP port                   # (INSECURE)
management IP port pw−file           #
```

pw−file, if specified, is a password file where the password must be on first line. Instead of a filename it can use the keyword stdin which will prompt the user for a password to use when OpenVPN is starting.

For unix sockets, the default behaviour is to create a unix domain socket that may be connected to by any process. Use the **−−management−client−user** and **−−management−client−group** directives to restrict access.

The management interface provides a special mode where the TCP management link can operate over the tunnel itself. To enable this mode, set IP to **tunnel**. Tunnel mode will cause the management interface to listen for a TCP connection on the local VPN address of the TUN/TAP interface.

BEWARE of enabling the management interface over TCP. In these cases you should *ALWAYS* make use of **pw−file** to password protect the management interface. Any user who can connect to this TCP **IP:port** will be able to manage and control (and interfere with) the OpenVPN process. It is also strongly recommended to set IP to 127.0.0.1 (localhost) to restrict accessibility of the management server to local clients.

While the management port is designed for programmatic control of OpenVPN by other applications, it is possible to telnet to the port, using a telnet client in "raw" mode. Once connected, type **help** for a list of commands.

For detailed documentation on the management interface, see the *management−notes.txt* file in the management folder of the OpenVPN source distribution.

−−management−client
Management interface will connect as a TCP/unix domain client to **IP:port** specified by **−−management** rather than listen as a TCP server or on a unix domain socket.

If the client connection fails to connect or is disconnected, a SIGTERM signal will be generated causing OpenVPN to quit.

−−management−client−auth
Gives management interface client the responsibility to authenticate clients after their client certificate has been verified. See **management−notes.txt** in OpenVPN distribution for detailed notes.

−−management−client−group *g*
When the management interface is listening on a unix domain socket, only allow connections from group **g**.

−−management−client−pf
Management interface clients must specify a packet filter file for each connecting client. See **management−notes.txt** in OpenVPN distribution for detailed notes.

−−management−client−user *u*
When the management interface is listening on a unix domain socket, only allow connections from user **u**.

−−management−external−cert *certificate−hint*
Allows usage for external certificate instead of **−−cert** option (client−only). **certificate−hint** is an arbitrary string which is passed to a management interface client as an argument of *NEED−CERTIFICATE* notification. Requires **−−management−external−key**.

−−management−external−key *args*
Allows usage for external private key file instead of **−−key** option (client−only).

Valid syntaxes:

```
management−external−key
management−external−key nopadding
management−external−key pkcs1
management−external−key nopadding pkcs1
```

The optional parameters **nopadding** and **pkcs1** signal support for different padding algorithms. See **doc/mangement−notes.txt** for a complete description of this feature.

−−management−forget−disconnect
Make OpenVPN forget passwords when management session disconnects.

This directive does not affect the **−−http−proxy** username/password. It is always cached.

−−management−hold
Start OpenVPN in a hibernating state, until a client of the management interface explicitly starts it with the **hold release** command.

−−management−log−cache *n*
Cache the most recent **n** lines of log file history for usage by the management channel.

−−management−query−passwords
Query management channel for private key password and **−−auth−user−pass** username/password. Only query the management channel for inputs which ordinarily would have been queried from the console.

−−management−query−proxy
Query management channel for proxy server information for a specific **−−remote** (client−only).

−−management−query−remote
Allow management interface to override **−−remote** directives (client−only).

−−management−signal
Send SIGUSR1 signal to OpenVPN if management session disconnects. This is useful when you wish to disconnect an OpenVPN session on user logoff. For **−−management−client** this option is not needed since a disconnect will always generate a **SIGTERM**.

−−management−up−down
Report tunnel up/down events to management interface.

Plug−in Interface Options
OpenVPN can be extended by loading external plug−in modules at runtime. These plug−ins must be prebuilt and adhere to the OpenVPN Plug−In API.

−−plugin *args*
Loads an OpenVPN plug−in module.

Valid syntax:

```
plugin module-name
plugin module-name "arguments"
```

The **module-name** needs to be the first argument, indicating the plug-in to load. The second argument is an optional init string which will be passed directly to the plug-in. If the init consists of multiple arguments it must be enclosed in double-quotes ("). Multiple plugin modules may be loaded into one OpenVPN process.

The **module-name** argument can be just a filename or a filename with a relative or absolute path. The format of the filename and path defines if the plug-in will be loaded from a default plug-in directory or outside this directory.

```
--plugin path         Effective directory used
===================== ============================
 myplug.so            DEFAULT_DIR/myplug.so
 subdir/myplug.so     DEFAULT_DIR/subdir/myplug.so
 ./subdir/myplug.so   CWD/subdir/myplug.so
 /usr/lib/my/plug.so  /usr/lib/my/plug.so
```

DEFAULT_DIR is replaced by the default plug-in directory, which is configured at the build time of OpenVPN. **CWD** is the current directory where OpenVPN was started or the directory OpenVPN have switched into via the **--cd** option before the **--plugin** option.

For more information and examples on how to build OpenVPN plug-in modules, see the README file in the **plugin** folder of the OpenVPN source distribution.

If you are using an RPM install of OpenVPN, see **/usr/share/openvpn/plugin**. The documentation is in **doc** and the actual plugin modules are in **lib**.

Multiple plugin modules can be cascaded, and modules can be used in tandem with scripts. The modules will be called by OpenVPN in the order that they are declared in the config file. If both a plugin and script are configured for the same callback, the script will be called last. If the return code of the module/script controls an authentication function (such as tls-verify, auth-user-pass-verify, or client-connect), then every module and script must return success (**0**) in order for the connection to be authenticated.

WARNING:

Plug-ins may do deferred execution, meaning the plug-in will return the control back to the main OpenVPN process and provide the plug-in result later on via a different thread or process. OpenVPN does **NOT** support multiple authentication plug-ins **where more than one plugin** tries to do deferred authentication. If this behaviour is detected, OpenVPN will shut down upon first authentication.

Windows-Specific Options

--allow-nonadmin *TAP-adapter*

(Standalone) Set **TAP-adapter** to allow access from non-administrative accounts. If **TAP-adapter** is omitted, all TAP adapters on the system will be configured to allow non-admin access. The non-admin access setting will only persist for the length of time that the TAP-Win32 device object and driver remain loaded, and will need to be re-enabled after a reboot, or if the driver is unloaded and reloaded. This directive can only be used by an administrator.

--block-outside-dns

Block DNS servers on other network adapters to prevent DNS leaks. This option prevents any application from accessing TCP or UDP port 53 except one inside the tunnel. It uses Windows Filtering Platform (WFP) and works on Windows Vista or later.

This option is considered unknown on non−Windows platforms and unsupported on Windows XP, resulting in fatal error. You may want to use **−−setenv opt** or **−−ignore−unknown−option** (not suitable for Windows XP) to ignore said error. Note that pushing unknown options from server does not trigger fatal errors.

−−cryptoapicert *select−string*
(Windows/OpenSSL Only) Load the certificate and private key from the Windows Certificate System Store.

Use this option instead of **−−cert** and **−−key**.

This makes it possible to use any smart card, supported by Windows, but also any kind of certificate, residing in the Cert Store, where you have access to the private key. This option has been tested with a couple of different smart cards (GemSAFE, Cryptoflex, and Swedish Post Office eID) on the client side, and also an imported PKCS12 software certificate on the server side.

To select a certificate, based on a substring search in the certificate's subject:

```
cryptoapicert "SUBJ:Peter Runestig"
```

To select a certificate, based on certificate's thumbprint:

```
cryptoapicert "THUMB:f6 49 24 41 01 b4 ..."
```

The thumbprint hex string can easily be copy−and−pasted from the Windows Certificate Store GUI.

−−dhcp−release
Ask Windows to release the TAP adapter lease on shutdown. This option has no effect now, as it is enabled by default starting with OpenVPN 2.4.1.

−−dhcp−renew
Ask Windows to renew the TAP adapter lease on startup. This option is normally unnecessary, as Windows automatically triggers a DHCP renegotiation on the TAP adapter when it comes up, however if you set the TAP−Win32 adapter Media Status property to "Always Connected", you may need this flag.

−−ip−win32 *method*
When using **−−ifconfig** on Windows, set the TAP−Win32 adapter IP address and netmask using **method**. Don't use this option unless you are also using **−−ifconfig**.

manual
Don't set the IP address or netmask automatically. Instead output a message to the console telling the user to configure the adapter manually and indicating the IP/netmask which OpenVPN expects the adapter to be set to.

dynamic [offset] [lease−time]
Automatically set the IP address and netmask by replying to DHCP query messages generated by the kernel. This mode is probably the "cleanest" solution for setting the TCP/IP properties since it uses the well−known DHCP protocol. There are, however, two prerequisites for using this mode:

1. The TCP/IP properties for the TAP−Win32 adapter must be set to "Obtain an IP address automatically", and
2. OpenVPN needs to claim an IP address in the subnet for use as the virtual DHCP server address.

By default in **−−dev tap** mode, OpenVPN will take the normally unused first address in

the subnet. For example, if your subnet is **192.168.4.0 netmask 255.255.255.0**, then OpenVPN will take the IP address **192.168.4.0** to use as the virtual DHCP server address. In **−−dev tun** mode, OpenVPN will cause the DHCP server to masquerade as if it were coming from the remote endpoint.

The optional offset parameter is an integer which is > **−256** and < **256** and which defaults to 0. If offset is positive, the DHCP server will masquerade as the IP address at network address + offset. If offset is negative, the DHCP server will masquerade as the IP address at broadcast address + offset.

The Windows **ipconfig /all** command can be used to show what Windows thinks the DHCP server address is. OpenVPN will "claim" this address, so make sure to use a free address. Having said that, different OpenVPN instantiations, including different ends of the same connection, can share the same virtual DHCP server address.

The **lease−time** parameter controls the lease time of the DHCP assignment given to the TAP−Win32 adapter, and is denoted in seconds. Normally a very long lease time is preferred because it prevents routes involving the TAP−Win32 adapter from being lost when the system goes to sleep. The default lease time is one year.

netsh Automatically set the IP address and netmask using the Windows command−line "netsh" command. This method appears to work correctly on Windows XP but not Windows 2000.

ipapi Automatically set the IP address and netmask using the Windows IP Helper API. This approach does not have ideal semantics, though testing has indicated that it works okay in practice. If you use this option, it is best to leave the TCP/IP properties for the TAP−Win32 adapter in their default state, i.e. "Obtain an IP address automatically."

adaptive (Default)
Try **dynamic** method initially and fail over to **netsh** if the DHCP negotiation with the TAP−Win32 adapter does not succeed in 20 seconds. Such failures have been known to occur when certain third−party firewall packages installed on the client machine block the DHCP negotiation used by the TAP−Win32 adapter. Note that if the **netsh** failover occurs, the TAP−Win32 adapter TCP/IP properties will be reset from DHCP to static, and this will cause future OpenVPN startups using the **adaptive** mode to use **netsh** immediately, rather than trying **dynamic** first.

To "unstick" the **adaptive** mode from using **netsh**, run OpenVPN at least once using the **dynamic** mode to restore the TAP−Win32 adapter TCP/IP properties to a DHCP configuration.

−−pause−exit
Put up a "press any key to continue" message on the console prior to OpenVPN program exit. This option is automatically used by the Windows explorer when OpenVPN is run on a configuration file using the right−click explorer menu.

−−register−dns
Run **ipconfig /flushdns** and **ipconfig /registerdns** on connection initiation. This is known to kick Windows into recognizing pushed DNS servers.

−−route−method *m*
Which method **m** to use for adding routes on Windows?

adaptive (default)
Try IP helper API first. If that fails, fall back to the route.exe shell command.

ipapi Use IP helper API.

exe Call the route.exe shell command.

−−service *args*
Should be used when OpenVPN is being automatically executed by another program in such a context that no interaction with the user via display or keyboard is possible.

Valid syntax:

```
service exit−event [0|1]
```

In general, end−users should never need to explicitly use this option, as it is automatically added by the OpenVPN service wrapper when a given OpenVPN configuration is being run as a service.

exit−event is the name of a Windows global event object, and OpenVPN will continuously monitor the state of this event object and exit when it becomes signaled.

The second parameter indicates the initial state of **exit−event** and normally defaults to 0.

Multiple OpenVPN processes can be simultaneously executed with the same **exit−event** parameter. In any case, the controlling process can signal **exit−event**, causing all such OpenVPN processes to exit.

When executing an OpenVPN process using the **−−service** directive, OpenVPN will probably not have a console window to output status/error messages, therefore it is useful to use **−−log** or **−−log−append** to write these messages to a file.

−−show−adapters
(Standalone) Show available TAP−Win32 adapters which can be selected using the **−−dev−node** option. On non−Windows systems, the **ifconfig**(8) command provides similar functionality.

−−show−net
(Standalone) Show OpenVPN's view of the system routing table and network adapter list.

−−show−net−up
Output OpenVPN's view of the system routing table and network adapter list to the syslog or log file after the TUN/TAP adapter has been brought up and any routes have been added.

−−show−valid−subnets
(Standalone) Show valid subnets for **−−dev tun** emulation. Since the TAP−Win32 driver exports an ethernet interface to Windows, and since TUN devices are point−to−point in nature, it is necessary for the TAP−Win32 driver to impose certain constraints on TUN endpoint address selection.

Namely, the point−to−point endpoints used in TUN device emulation must be the middle two addresses of a /30 subnet (netmask 255.255.255.252).

−−tap−sleep *n*
Cause OpenVPN to sleep for **n** seconds immediately after the TAP−Win32 adapter state is set to "connected".

This option is intended to be used to troubleshoot problems with the **−−ifconfig** and **−−ip−win32** options, and is used to give the TAP−Win32 adapter time to come up before Windows IP Helper API operations are applied to it.

−−win−sys *path*
Set the Windows system directory pathname to use when looking for system executables such as **route.exe** and **netsh.exe**. By default, if this directive is not specified, OpenVPN will use the SystemRoot environment variable.

This option has changed behaviour since OpenVPN 2.3. Earlier you had to define **−−win−sys env**

to use the SystemRoot environment variable, otherwise it defaulted to **C:\\WINDOWS**. It is not needed to use the **env** keyword any more, and it will just be ignored. A warning is logged when this is found in the configuration file.

−−windows−driver *drv*
Specifies which tun driver to use. Values are **tap−windows6** (default) and **wintun**. This is a Windows−only option. **wintun**" requires **−−de v tun** and the OpenVPN process to run elevated, or be invoked using the Interactive Service.

Standalone Debug Options

−−show−gateway *args*
(Standalone) Show current IPv4 and IPv6 default gateway and interface towards the gateway (if the protocol in question is enabled).

Valid syntax:

```
−−show−gateway
−−show−gateway IPv6−target
```

For IPv6 this queries the route towards ::/128, or the specified IPv6 target address if passed as argument. For IPv4 on Linux, Windows, MacOS and BSD it looks for a 0.0.0.0/0 route. If there are more specific routes, the result will not always be matching the route of the IPv4 packets to the VPN gateway.

Advanced Expert Options

These are options only required when special tweaking is needed, often used when debugging or testing out special usage scenarios.

−−hash−size *args*
Set the size of the real address hash table to **r** and the virtual address table to **v**.

Valid syntax:

```
hash−size r v
```

By default, both tables are sized at 256 buckets.

−−bcast−buffers *n*
Allocate **n** buffers for broadcast datagrams (default **256**).

−−persist−local−ip
Preserve initially resolved local IP address and port number across **SIGUSR1** or **−−ping−restart** restarts.

−−persist−remote−ip
Preserve most recently authenticated remote IP address and port number across **SIGUSR1** or **−−ping−restart** restarts.

−−prng *args*
(Advanced) Change the PRNG (Pseudo−random number generator) parameters

Valid syntaxes:

```
prng alg
prng alg nsl
```

Changes the PRNG to use digest algorithm **alg** (default **sha1**), and set **nsl** (default **16**) to the size in bytes of the nonce secret length (between 16 and 64).

Set **alg** to **none** to disable the PRNG and use the OpenSSL RAND_bytes function instead for all of OpenVPN's pseudo−random number needs.

−−rcvbuf *size*
Set the TCP/UDP socket receive buffer size. Defaults to operating system default.

−−shaper *n*
Limit bandwidth of outgoing tunnel data to **n** bytes per second on the TCP/UDP port. Note that this will only work if mode is set to **p2p**. If you want to limit the bandwidth in both directions, use this option on both peers.

OpenVPN uses the following algorithm to implement traffic shaping: Given a shaper rate of **n** bytes per second, after a datagram write of **b** bytes is queued on the TCP/UDP port, wait a minimum of **(b / n)** seconds before queuing the next write.

It should be noted that OpenVPN supports multiple tunnels between the same two peers, allowing you to construct full−speed and reduced bandwidth tunnels at the same time, routing low−priority data such as off−site backups over the reduced bandwidth tunnel, and other data over the full−speed tunnel.

Also note that for low bandwidth tunnels (under 1000 bytes per second), you should probably use lower MTU values as well (see above), otherwise the packet latency will grow so large as to trigger timeouts in the TLS layer and TCP connections running over the tunnel.

OpenVPN allows **n** to be between 100 bytes/sec and 100 Mbytes/sec.

−−sndbuf *size*
Set the TCP/UDP socket send buffer size. Defaults to operating system default.

−−tcp−queue−limit *n*
Maximum number of output packets queued before TCP (default **64**).

When OpenVPN is tunneling data from a TUN/TAP device to a remote client over a TCP connection, it is possible that the TUN/TAP device might produce data at a faster rate than the TCP connection can support. When the number of output packets queued before sending to the TCP socket reaches this limit for a given client connection, OpenVPN will start to drop outgoing packets directed at this client.

−−txqueuelen *n*
(Linux only) Set the TX queue length on the TUN/TAP interface. Currently defaults to operating system default.

UNSUPPORTED OPTIONS

Options listed in this section have been removed from OpenVPN and are no longer supported

−−client−cert−not−required
Removed in OpenVPN 2.5. This should be replaxed with **−−verify−client−cert none**.

−−ifconfig−pool−linear
Removed in OpenVPN 2.5. This should be replaced with **−−topology p2p**.

−−key−method
Removed in OpenVPN 2.5. This option should not be used, as using the old **key−method** weakens the VPN tunnel security. The old**k ey−method** was also only needed when the remote side was older than OpenVPN 2.0.

−−no−iv
Removed in OpenVPN 2.5. This option should not be used as it weakens the VPN tunnel security. This has been a NOOP option since OpenVPN 2.4.

--no-replay
Removed in OpenVPN 2.5. This option should not be used as it weakens the VPN tunnel security.

--ns-cert-type
Removed in OpenVPN 2.5. The **nsCertType** field is no longer supported in recent SSL/TLS libraries. If your certificates does not include*y usage* and *extended key usage* fields, they must be upgraded and the **--remote-cert-tls** option should be used instead.

CONNECTION PROFILES

Client configuration files may contain multiple remote servers which it will attempt to connect against. But there are some configuration options which are related to specific **--remote** options. For these use cases, connection profiles are the solution.

By enacpulating the **--remote** option and related options within **<connection>** and **</connection>**, these options are handled as a group.

An OpenVPN client will try each connection profile sequentially until it achieves a successful connection.

--remote-random can be used to initially "scramble" the connection list.

Here is an example of connection profile usage:

```
client
dev tun

<connection>
remote 198.19.34.56 1194 udp
</connection>

<connection>
remote 198.19.34.56 443 tcp
</connection>

<connection>
remote 198.19.34.56 443 tcp
http-proxy 192.168.0.8 8080
</connection>

<connection>
remote 198.19.36.99 443 tcp
http-proxy 192.168.0.8 8080
</connection>

persist-key
persist-tun
pkcs12 client.p12
remote-cert-tls server
verb 3
```

First we try to connect to a server at 198.19.34.56:1194 using UDP. If that fails, we then try to connect to 198.19.34.56:443 using TCP. If that also fails, then try connecting through an HTTP proxy at 192.168.0.8:8080 to 198.19.34.56:443 using TCP. Finally, try to connect through the same proxy to a server at 198.19.36.99:443 using TCP.

The following OpenVPN options may be used inside of a **<connection>** block:

bind, **connect−retry**, **connect−retry−max**, **connect−timeout**, **explicit−exit−notify**, **float**, **fragment**, **http−proxy**, **http−proxy−option**, **key−direction**, **link−mtu**, **local**, **lport**, **mssfix**, **mtu−disc**, **nobind**, **port**, **proto**, **remote**, **rport**, **socks−proxy**, **tls−auth**, **tls−crypt**, **tun−mtu and**, **tun−mtu−extra**.

A defaulting mechanism exists for specifying options to apply to all **<connection>** profiles. If any of the above options (with the exception of **remote**) appear outside of a **<connection>** block, but in a configuration file which has one or more **<connection>** blocks, the option setting will be used as a default for **<connection>** blocks which follow it in the configuration file.

For example, suppose the **nobind** option were placed in the sample configuration file above, near the top of the file, before the first **<connection>** block. The effect would be as if **nobind** were declared in all **<connection>** blocks below it.

INLINE FILE SUPPORT

OpenVPN allows including files in the main configuration for the **−−ca**, **−−cert**, **−−dh**, **−−extra−certs**, **−−key**, **−−pkcs12**, **−−secret**, **−−crl−verify**, **−−http−proxy−user−pass**, **−−tls−auth**, **−−auth−gen−token−secret**, **−−tls−crypt** and **−−tls−crypt−v2** options.

Each inline file started by the line **<option>** and ended by the line **</option>**

Here is an example of an inline file usage

```
<cert>
−−−−−BEGIN CERTIFICATE−−−−−
[...]
−−−−−END CERTIFICATE−−−−−
</cert>
```

When using the inline file feature with **−−pkcs12** the inline file has to be base64 encoded. Encoding of a .p12 file into base64 can be done for example with OpenSSL by running **openssl base64 −in input.p12**

SIGNALS

SIGHUP

Cause OpenVPN to close all TUN/TAP and network connections, restart, re−read the configuration file (if any), and reopen TUN/TAP and network connections.

SIGUSR1

Like **SIGHUP`**, except don't re−read configuration file, and possibly don't close and reopen TUN/TAP device, re−read key files, preserve local IP address/port, or preserve most recently authenticated remote IP address/port based on **−−persist−tun**, **−−persist−key**, **−−persist−local−ip** and **−−persist−remote−ip** options respectively (see above).

This signal may also be internally generated by a timeout condition, governed by the **−−ping−restart** option.

This signal, when combined with **−−persist−remote−ip**, may be sent when the underlying parameters of the host's network interface change such as when the host is a DHCP client and is assigned a new IP address. See **−−ipchange** for more information.

SIGUSR2

Causes OpenVPN to display its current statistics (to the syslog file if **−−daemon** is used, or stdout otherwise).

SIGINT, SIGTERM

Causes OpenVPN to exit gracefully.

FAQ

https://community.openvpn.net/openvpn/wiki/FAQ

HOWTO

For a more comprehensive guide to setting up OpenVPN in a production setting, see the OpenVPN HOWTO at *https://openvpn.net/community−resources/how−to/*

PROTOCOL

For a description of OpenVPN's underlying protocol, see *https://openvpn.net/community−resources/openvpn−protocol/*

WEB

OpenVPN's web site is at *https://openvpn.net/*

Go here to download the latest version of OpenVPN, subscribe to the mailing lists, read the mailing list archives, or browse the SVN repository.

BUGS

Report all bugs to the OpenVPN team *info@openvpn.net*

SEE ALSO

openvpn−examples(5), **dhcpcd**(8), **ifconfig**(8), **openssl**(1), **route**(8), **scp**(1) **ssh**(1)

NOTES

This product includes software developed by the OpenSSL Project (*https://www.openssl.org/*)

For more information on the TLS protocol, see *http://www.ietf.org/rfc/rfc2246.txt*

For more information on the LZO real−time compression library see *https://www.oberhumer.com/opensource/lzo/*

COPYRIGHT

AUTHORS

James Yonan *james@openvpn.net*