tinc Manual

Setting up a Virtual Private Network with tinc

Ivo Timmermans and Guus Sliepen

This is the info manual for tinc version 1.1pre13, a Virtual Private Network daemon.

1 Introduction

Tinc is a Virtual Private Network (VPN) daemon that uses tunneling and encryption to create a secure private network between hosts on the Internet.

Because the tunnel appears to the IP level network code as a normal network device, there is no need to adapt any existing software. The encrypted tunnels allows VPN sites to share information with each other over the Internet without exposing any information to others.

This document is the manual for tinc. Included are chapters on how to configure your computer to use tinc, as well as the configuration process of tinc itself.

1.1 Virtual Private Networks

A Virtual Private Network or VPN is a network that can only be accessed by a few elected computers that participate. This goal is achievable in more than just one way.

Private networks can consist of a single stand-alone Ethernet LAN. Or even two computers hooked up using a null-modem cable. In these cases, it is obvious that the network is *private*, no one can access it from the outside. But if your computers are linked to the Internet, the network is not private anymore, unless one uses firewalls to block all private traffic. But then, there is no way to send private data to trusted computers on the other end of the Internet.

This problem can be solved by using *virtual* networks. Virtual networks can live on top of other networks, but they use encapsulation to keep using their private address space so they do not interfere with the Internet. Mostly, virtual networks appear like a single LAN, even though they can span the entire world. But virtual networks can't be secured by using firewalls, because the traffic that flows through it has to go through the Internet, where other people can look at it.

As is the case with either type of VPN, anybody could eavesdrop. Or worse, alter data. Hence it's probably advisable to encrypt the data that flows over the network.

When one introduces encryption, we can form a true VPN. Other people may see encrypted traffic, but if they don't know how to decipher it (they need to know the key for that), they cannot read the information that flows through the VPN. This is what tinc was made for.

1.2 tinc

I really don't quite remember what got us started, but it must have been Guus' idea. He wrote a simple implementation (about 50 lines of C) that used the ethertap device that Linux knows of since somewhere about kernel 2.1.60. It didn't work immediately and he improved it a bit. At this stage, the project was still simply called "vpnd".

Since then, a lot has changed—to say the least.

Tinc now supports encryption, it consists of a single daemon (tincd) for both the receiving and sending end, it has become largely runtime-configurable—in short, it has become a full-fledged professional package.

Tinc also allows more than two sites to connect to eachother and form a single VPN. Traditionally VPNs are created by making tunnels, which only have two endpoints. Larger VPNs with more sites are created by adding more tunnels. Tinc takes another approach: only endpoints are specified, the software itself will take care of creating the tunnels. This allows for easier configuration and improved scalability.

A lot can—and will be—changed. We have a number of things that we would like to see in the future releases of tinc. Not everything will be available in the near future. Our first objective is to make tinc work perfectly as it stands, and then add more advanced features.

Meanwhile, we're always open-minded towards new ideas. And we're available too.

1.3 Supported platforms

Tinc has been verified to work under Linux, FreeBSD, OpenBSD, NetBSD, MacOS/X (Darwin), Solaris, and Windows (both natively and in a Cygwin environment), with various hardware architectures. These are some of the platforms that are supported by the universal tun/tap device driver or other virtual network device drivers. Without such a driver, tinc will most likely compile and run, but it will not be able to send or receive data packets.

For an up to date list of supported platforms, please check the list on our website: https://www.tinc-vpn.org/platforms/.

2 Preparations

This chapter contains information on how to prepare your system to support tinc.

2.1 Configuring the kernel

2.1.1 Configuration of Linux kernels

For tinc to work, you need a kernel that supports the Universal tun/tap device. Most distributions come with kernels that already support this. Here are the options you have to turn on when configuring a new kernel:

```
Code maturity level options
[*] Prompt for development and/or incomplete code/drivers
Network device support
<M> Universal tun/tap device driver support
```

It's not necessary to compile this driver as a module, even if you are going to run more than one instance of tinc.

If you decide to build the tun/tap driver as a kernel module, add these lines to `/etc/modules.conf`:

```
alias char-major-10-200 tun
```

2.1.2 Configuration of FreeBSD kernels

For FreeBSD version 4.1 and higher, tun and tap drivers are included in the default kernel configuration. The tap driver can be loaded with `kldload if_tap`, or by adding `if_tap_load="YES"` to `/boot/loader.conf`.

2.1.3 Configuration of OpenBSD kernels

Recent versions of OpenBSD come with both tun and tap devices enabled in the default kernel configuration.

2.1.4 Configuration of NetBSD kernels

For NetBSD version 1.5.2 and higher, the tun driver is included in the default kernel configuration.

Tunneling IPv6 may not work on NetBSD's tun device.

2.1.5 Configuration of Solaris kernels

For Solaris 8 (SunOS 5.8) and higher, the tun driver may or may not be included in the default kernel configuration. If it isn't, the source can be downloaded from http://vtun.sourceforge.net/tun/. For x86 and sparc64 architectures, precompiled versions can be found at https://www.monkey.org/~dugsong/fragroute/. If the `net/if_tun.h` header file is missing, install it from the source package.

2.1.6 Configuration of Darwin (MacOS/X) kernels

Tinc on Darwin relies on a tunnel driver for its data acquisition from the kernel. OS X version 10.6.8 and later have a built-in tun driver called "utun". Tinc also supports the driver from http://tuntaposx.sourceforge.net/, which supports both tun and tap style devices,

By default, tinc expects the tuntaposx driver to be installed. To use the utun driver, set add `Device = utunX` to `tinc.conf`, where X is the desired number for the utun interface. You can also omit the number, in which case the first free number will be chosen.

2.1.7 Configuration of Windows

You will need to install the latest TAP-Win32 driver from OpenVPN. You can download it from https://openvpn.net/index.php/open-source/downloads.html. Using the Network Connections control panel, configure the TAP-Win32 network interface in the same way as you would do from the tinc-up script, as explained in the rest of the documentation.

2.2 Libraries

Before you can configure or build tinc, you need to have the LibreSSL or OpenSSL, zlib, lzo, curses and readline libraries installed on your system. If you try to configure tinc without having them installed, configure will give you an error message, and stop.

2.2.1 LibreSSL/OpenSSL

For all cryptography-related functions, tinc uses the functions provided by the LibreSSL or the OpenSSL library.

If this library is not installed, you wil get an error when configuring tinc for build. Support for running tinc with other cryptographic libraries installed *may* be added in the future.

You can use your operating system's package manager to install this if available. Make sure you install the development AND runtime versions of this package.

If your operating system comes neither with LibreSSL or OpenSSL, you have to install one manually. It is recommended that you get the latest version of LibreSSL from http://www.libressl.org/. Instructions on how to configure, build and install this package are included within the package. Please make sure you build development and runtime libraries (which is the default).

If you installed the LibreSSL or OpenSSL libraries from source, it may be necessary to let configure know where they are, by passing configure one of the –with-openssl-* parameters. Note that you even have to use –with-openssl-* if you are using LibreSSL.

```
--with-openssl=DIR      LibreSSL/OpenSSL library and headers prefix
--with-openssl-include=DIR LibreSSL/OpenSSL headers directory
                        (Default is OPENSSL_DIR/include)
--with-openssl-lib=DIR  LibreSSL/OpenSSL library directory
                        (Default is OPENSSL_DIR/lib)
```

License

The complete source code of tinc is covered by the GNU GPL version 2. Since the license under which OpenSSL is distributed is not directly compatible with the terms of the GNU GPL https://www.openssl.org/support/faq.html#LEGAL2, we include an exemption to the GPL (see also the file COPYING.README) to allow everyone to create a statically or dynamically linked executable:

> This program is released under the GPL with the additional exemption that compiling, linking, and/or using OpenSSL is allowed. You may provide binary packages linked to the OpenSSL libraries, provided that all other requirements of the GPL are met.

Since the LZO library used by tinc is also covered by the GPL, we also present the following exemption:

> Hereby I grant a special exception to the tinc VPN project (https://www.tinc-vpn.org/) to link the LZO library with the OpenSSL library (https://www.openssl.org).
>
> Markus F.X.J. Oberhumer

2.2.2 zlib

For the optional compression of UDP packets, tinc uses the functions provided by the zlib library.

If this library is not installed, you wil get an error when running the configure script. You can either install the zlib library, or disable support for zlib compression by using the "–disable-zlib" option when running the configure script. Note that if you disable support for zlib, the resulting binary will not work correctly on VPNs where zlib compression is used.

You can use your operating system's package manager to install this if available. Make sure you install the development AND runtime versions of this package.

If you have to install zlib manually, you can get the source code from http://www.zlib.net/. Instructions on how to configure, build and install this package are included within the package. Please make sure you build development and runtime libraries (which is the default).

2.2.3 lzo

Another form of compression is offered using the LZO library.

If this library is not installed, you wil get an error when running the configure script. You can either install the LZO library, or disable support for LZO compression by using the "–disable-lzo" option when running the configure script. Note that if you disable support for LZO, the resulting binary will not work correctly on VPNs where LZO compression is used.

You can use your operating system's package manager to install this if available. Make sure you install the development AND runtime versions of this package.

If you have to install lzo manually, you can get the source code from https://www.oberhumer.com/opensource/lzo/. Instructions on how to configure, build and install this package are included within the package. Please make sure you build development and runtime libraries (which is the default).

2.2.4 libcurses

For the "tinc top" command, tinc requires a curses library.

If this library is not installed, you wil get an error when running the configure script. You can either install a suitable curses library, or disable all functionality that depends on a curses library by using the "–disable-curses" option when running the configure script.

There are several curses libraries. It is recommended that you install "ncurses" (http://invisible-island.net/ncurses/), however other curses libraries should also work. In particular, "PDCurses" (http://pdcurses.sourceforge.net/) is recommended if you want to compile tinc for Windows.

You can use your operating system's package manager to install this if available. Make sure you install the development AND runtime versions of this package.

2.2.5 libreadline

For the "tinc" command's shell functionality, tinc uses the readline library.

If this library is not installed, you wil get an error when running the configure script. You can either install a suitable readline library, or disable all functionality that depends on a readline library by using the "–disable-readline" option when running the configure script.

You can use your operating system's package manager to install this if available. Make sure you install the development AND runtime versions of this package.

If you have to install libreadline manually, you can get the source code from http://www.gnu.org/software/readline/. Instructions on how to configure, build and install this package are included within the package. Please make sure you build development and runtime libraries (which is the default).

3 Installation

If you use Debian, you may want to install one of the precompiled packages for your system. These packages are equipped with system startup scripts and sample configurations.

If you cannot use one of the precompiled packages, or you want to compile tinc for yourself, you can use the source. The source is distributed under the GNU General Public License (GPL). Download the source from the download page.

Tinc comes in a convenient autoconf/automake package, which you can just treat the same as any other package. Which is just untar it, type './configure' and then 'make'. More detailed instructions are in the file `INSTALL`, which is included in the source distribution.

3.1 Building and installing tinc

Detailed instructions on configuring the source, building tinc and installing tinc can be found in the file called `INSTALL`.

If you happen to have a binary package for tinc for your distribution, you can use the package management tools of that distribution to install tinc. The documentation that comes along with your distribution will tell you how to do that.

3.1.1 Darwin (MacOS/X) build environment

In order to build tinc on Darwin, you need to install Xcode from `https://developer.apple.com/xcode/`. It might also help to install a recent version of Fink from `http://www.finkproject.org/`.

You need to download and install LibreSSL (or OpenSSL) and LZO, either directly from their websites (see Section 2.2 [Libraries], page 4) or using Fink.

3.1.2 Cygwin (Windows) build environment

If Cygwin hasn't already been installed, install it directly from `https://www.cygwin.com/`.

When tinc is compiled in a Cygwin environment, it can only be run in this environment, but all programs, including those started outside the Cygwin environment, will be able to use the VPN. It will also support all features.

3.1.3 MinGW (Windows) build environment

You will need to install the MinGW environment from `http://www.mingw.org`. You also need to download and install LibreSSL (or OpenSSL) and LZO.

When tinc is compiled using MinGW it runs natively under Windows, it is not necessary to keep MinGW installed.

When detaching, tinc will install itself as a service, which will be restarted automatically after reboots.

3.2 System files

Before you can run tinc, you must make sure you have all the needed files on your system.

3.2.1 Device files

Most operating systems nowadays come with the necessary device files by default, or they have a mechanism to create them on demand.

If you use Linux and do not have udev installed, you may need to create the following device file if it does not exist:

```
mknod -m 600 /dev/net/tun c 10 200
```

3.2.2 Other files

/etc/networks

You may add a line to `/etc/networks` so that your VPN will get a symbolic name. For example:

```
myvpn 10.0.0.0
```

/etc/services

You may add this line to `/etc/services`. The effect is that you may supply a '`tinc`' as a valid port number to some programs. The number 655 is registered with the IANA.

```
tinc            655/tcp    TINC
tinc            655/udp    TINC
#                          Ivo Timmermans <ivo@tinc-vpn.org>
```

4 Configuration

4.1 Configuration introduction

Before actually starting to configure tinc and editing files, make sure you have read this entire section so you know what to expect. Then, make it clear to yourself how you want to organize your VPN: What are the nodes (computers running tinc)? What IP addresses/subnets do they have? What is the network mask of the entire VPN? Do you need special firewall rules? Do you have to set up masquerading or forwarding rules? Do you want to run tinc in router mode or switch mode? These questions can only be answered by yourself, you will not find the answers in this documentation. Make sure you have an adequate understanding of networks in general. A good resource on networking is the Linux Network Administrators Guide.

If you have everything clearly pictured in your mind, proceed in the following order: First, create the initial configuration files and public/private keypairs using the following command:

```
tinc -n NETNAME init NAME
```

Second, use '`tinc -n NETNAME add ...`' to further configure tinc. Finally, export your host configuration file using '`tinc -n NETNAME export`' and send it to those people or computers you want tinc to connect to. They should send you their host configuration file back, which you can import using '`tinc -n NETNAME import`'.

These steps are described in the subsections below.

4.2 Multiple networks

In order to allow you to run more than one tinc daemon on one computer, for instance if your computer is part of more than one VPN, you can assign a *netname* to your VPN. It is not required if you only run one tinc daemon, it doesn't even have to be the same on all the nodes of your VPN, but it is recommended that you choose one anyway.

We will asume you use a netname throughout this document. This means that you call tinc with the -n argument, which will specify the netname.

The effect of this option is that tinc will set its configuration root to `/etc/tinc/netname/`, where *netname* is your argument to the -n option. You will also notice that log messages it appears in syslog as coming from `tinc.netname`, and on Linux, unless specified otherwise, the name of the virtual network interface will be the same as the network name.

However, it is not strictly necessary that you call tinc with the -n option. If you do not use it, the network name will just be empty, and tinc will look for files in `/etc/tinc/` instead of `/etc/tinc/netname/`; the configuration file will then be `/etc/tinc/tinc.conf`, and the host configuration files are expected to be in `/etc/tinc/hosts/`.

4.3 How connections work

When tinc starts up, it parses the command-line options and then reads in the configuration file tinc.conf. If it sees one or more 'ConnectTo' values pointing to other tinc daemons in that file, it will try to connect to those other daemons. Whether this succeeds or not and whether 'ConnectTo' is specified or not, tinc will listen for incoming connection from other deamons. If you did specify a 'ConnectTo' value and the other side is not responding, tinc will keep retrying. This means that once started, tinc will stay running until you tell it to stop, and failures to connect to other tinc daemons will not stop your tinc daemon for trying again later. This means you don't have to intervene if there are temporary network problems.

There is no real distinction between a server and a client in tinc. If you wish, you can view a tinc daemon without a 'ConnectTo' value as a server, and one which does specify such a value

as a client. It does not matter if two tinc daemons have a 'ConnectTo' value pointing to each other however.

Connections specified using 'ConnectTo' are so-called meta-connections. Tinc daemons exchange information about all other daemon they know about via these meta-connections. After learning about all the daemons in the VPN, tinc will create other connections as necessary in order to communicate with them. For example, if there are three daemons named A, B and C, and A has '`ConnectTo = B`' in its tinc.conf file, and C has '`ConnectTo = B`' in its tinc.conf file, then A will learn about C from B, and will be able to exchange VPN packets with C without the need to have '`ConnectTo = C`' in its tinc.conf file.

It could be that some daemons are located behind a Network Address Translation (NAT) device, or behind a firewall. In the above scenario with three daemons, if A and C are behind a NAT, B will automatically help A and C punch holes through their NAT, in a way similar to the STUN protocol, so that A and C can still communicate with each other directly. It is not always possible to do this however, and firewalls might also prevent direct communication. In that case, VPN packets between A and C will be forwarded by B.

In effect, all nodes in the VPN will be able to talk to each other, as long as their is a path of meta-connections between them, and whenever possible, two nodes will communicate with each other directly.

4.4 Configuration files

The actual configuration of the daemon is done in the file `/etc/tinc/`*`netname`*`/tinc.conf` and at least one other file in the directory `/etc/tinc/`*`netname`*`/hosts/`.

An optionnal directory `/etc/tinc/`*`netname`*`/conf.d` can be added from which any .conf file will be read.

These file consists of comments (lines started with a #) or assignments in the form of

```
Variable = Value.
```

The variable names are case insensitive, and any spaces, tabs, newlines and carriage returns are ignored. Note: it is not required that you put in the '=' sign, but doing so improves readability. If you leave it out, remember to replace it with at least one space character.

The server configuration is complemented with host specific configuration (see the next section). Although all host configuration options for the local node listed in this document can also be put in `/etc/tinc/`*`netname`*`/tinc.conf`, it is recommended to put host specific configuration options in the host configuration file, as this makes it easy to exchange with other nodes.

You can edit the config file manually, but it is recommended that you use the tinc command to change configuration variables for you.

In the following two subsections all valid variables are listed in alphabetical order. The default value is given between parentheses, other comments are between square brackets.

4.4.1 Main configuration variables

AddressFamily = <ipv4|ipv6|any> (any)
: This option affects the address family of listening and outgoing sockets. If any is selected, then depending on the operating system both IPv4 and IPv6 or just IPv6 listening sockets will be created.

AutoConnect = <yes|no> (no) [experimental]
: If set to yes, tinc will automatically set up meta connections to other nodes, without requiring *ConnectTo* variables.

BindToAddress = <*address*> [<*port*>]
: This is the same as ListenAddress, however the address given with the BindToAddress option will also be used for outgoing connections. This is useful if your computer has more than one IPv4 or IPv6 address, and you want tinc to only use a specific one for outgoing packets.

BindToInterface = <*interface*> [experimental]
: If you have more than one network interface in your computer, tinc will by default listen on all of them for incoming connections. It is possible to bind tinc to a single interface like eth0 or ppp0 with this variable.

  This option may not work on all platforms. Also, on some platforms it will not actually bind to an interface, but rather to the address that the interface has at the moment a socket is created.

Broadcast = <no | mst | direct> (mst) [experimental]
: This option selects the way broadcast packets are sent to other daemons. *NOTE: all nodes in a VPN must use the same Broadcast mode, otherwise routing loops can form.*

  no — Broadcast packets are never sent to other nodes.

  mst — Broadcast packets are sent and forwarded via the VPN's Minimum Spanning Tree. This ensures broadcast packets reach all nodes.

  direct — Broadcast packets are sent directly to all nodes that can be reached directly. Broadcast packets received from other nodes are never forwarded. If the IndirectData option is also set, broadcast packets will only be sent to nodes which we have a meta connection to.

BroadcastSubnet = *address*[/*prefixlength*]
: Declares a broadcast subnet. Any packet with a destination address falling into such a subnet will be routed as a broadcast (provided all nodes have it declared). This is most useful to declare subnet broadcast addresses (e.g. 10.42.255.255), otherwise tinc won't know what to do with them.

  Note that global broadcast addresses (MAC ff:ff:ff:ff:ff:ff, IPv4 255.255.255.255), as well as multicast space (IPv4 224.0.0.0/4, IPv6 ff00::/8) are always considered broadcast addresses and don't need to be declared.

ConnectTo = <*name*>
: Specifies which other tinc daemon to connect to on startup. Multiple ConnectTo variables may be specified, in which case outgoing connections to each specified tinc daemon are made. The names should be known to this tinc daemon (i.e., there should be a host configuration file for the name on the ConnectTo line).

  If you don't specify a host with ConnectTo and don't enable AutoConnect, tinc won't try to connect to other daemons at all, and will instead just listen for incoming connections.

DecrementTTL = <yes | no> (no) [experimental]
: When enabled, tinc will decrement the Time To Live field in IPv4 packets, or the Hop Limit field in IPv6 packets, before forwarding a received packet to the virtual network device or to another node, and will drop packets that have a TTL value of zero, in which case it will send an ICMP Time Exceeded packet back.

  Do not use this option if you use switch mode and want to use IPv6.

Device = <*device*> (`/dev/tap0`, `/dev/net/tun` or other depending on platform)
: The virtual network device to use. Tinc will automatically detect what kind of device it is. Note that you can only use one device per daemon. Under Windows,

use *Interface* instead of *Device*. Note that you can only use one device per daemon. See also Section 3.2.1 [Device files], page 7.

DeviceStandby = <yes | no> (no)
: When disabled, tinc calls `tinc-up` on startup, and `tinc-down` on shutdown. When enabled, tinc will only call `tinc-up` when at least one node is reachable, and will call `tinc-down` as soon as no nodes are reachable. On Windows, this also determines when the virtual network interface "cable" is "plugged".

DeviceType = <*type*> (platform dependent)
: The type of the virtual network device. Tinc will normally automatically select the right type of tun/tap interface, and this option should not be used. However, this option can be used to select one of the special interface types, if support for them is compiled in.

  dummy
  : Use a dummy interface. No packets are ever read or written to a virtual network device. Useful for testing, or when setting up a node that only forwards packets for other nodes.

  raw_socket
  : Open a raw socket, and bind it to a pre-existing *Interface* (eth0 by default). All packets are read from this interface. Packets received for the local node are written to the raw socket. However, at least on Linux, the operating system does not process IP packets destined for the local host.

  multicast
  : Open a multicast UDP socket and bind it to the address and port (separated by spaces) and optionally a TTL value specified using *Device*. Packets are read from and written to this multicast socket. This can be used to connect to UML, QEMU or KVM instances listening on the same multicast address. Do NOT connect multiple tinc daemons to the same multicast address, this will very likely cause routing loops. Also note that this can cause decrypted VPN packets to be sent out on a real network if misconfigured.

  uml (not compiled in by default)
  : Create a UNIX socket with the filename specified by *Device*, or `/var/run/`*`netname`*`.umlsocket` if not specified. Tinc will wait for a User Mode Linux instance to connect to this socket.

  vde (not compiled in by default)
  : Uses the libvdeplug library to connect to a Virtual Distributed Ethernet switch, using the UNIX socket specified by *Device*, or `/var/run/vde.ctl` if not specified.

  Also, in case tinc does not seem to correctly interpret packets received from the virtual network device, it can be used to change the way packets are interpreted:

  tun (BSD and Linux)
  : Set type to tun. Depending on the platform, this can either be with or without an address family header (see below).

  tunnohead (BSD)
  : Set type to tun without an address family header. Tinc will expect packets read from the virtual network device to start with an IP header. On some platforms IPv6 packets cannot be read from or written to the device in this mode.

tunifhead (BSD)
: Set type to tun with an address family header. Tinc will expect packets read from the virtual network device to start with a four byte header containing the address family, followed by an IP header. This mode should support both IPv4 and IPv6 packets.

utun (OS X)
: Set type to utun. This is only supported on OS X version 10.6.8 and higher, but doesn't require the tuntaposx module. This mode should support both IPv4 and IPv6 packets.

tap (BSD and Linux)
: Set type to tap. Tinc will expect packets read from the virtual network device to start with an Ethernet header.

DirectOnly = <yes|no> (no) [experimental]
: When this option is enabled, packets that cannot be sent directly to the destination node, but which would have to be forwarded by an intermediate node, are dropped instead. When combined with the IndirectData option, packets for nodes for which we do not have a meta connection with are also dropped.

Ed25519PrivateKeyFile = <*path*> (`/etc/tinc/`*`netname`*`/ed25519_key.priv`)
: The file in which the private Ed25519 key of this tinc daemon resides. This is only used if ExperimentalProtocol is enabled.

ExperimentalProtocol = <yes|no> (yes)
: When this option is enabled, the SPTPS protocol will be used when connecting to nodes that also support it. Ephemeral ECDH will be used for key exchanges, and Ed25519 will be used instead of RSA for authentication. When enabled, an Ed25519 key must have been generated before with '`tinc generate-ed25519-keys`'.

Forwarding = <off|internal|kernel> (internal) [experimental]
: This option selects the way indirect packets are forwarded.

  off
  : Incoming packets that are not meant for the local node, but which should be forwarded to another node, are dropped.

  internal
  : Incoming packets that are meant for another node are forwarded by tinc internally.

    This is the default mode, and unless you really know you need another forwarding mode, don't change it.

  kernel
  : Incoming packets are always sent to the TUN/TAP device, even if the packets are not for the local node. This is less efficient, but allows the kernel to apply its routing and firewall rules on them, and can also help debugging.

Hostnames = <yes|no> (no)
: This option selects whether IP addresses (both real and on the VPN) should be resolved. Since DNS lookups are blocking, it might affect tinc's efficiency, even stopping the daemon for a few seconds everytime it does a lookup if your DNS server is not responding.

  This does not affect resolving hostnames to IP addresses from the configuration file, but whether hostnames should be resolved while logging.

Interface = <*interface*>
: Defines the name of the interface corresponding to the virtual network device. Depending on the operating system and the type of device this may or may not actually

set the name of the interface. Under Windows, this variable is used to select which network interface will be used. If you specified a Device, this variable is almost always already correctly set.

ListenAddress = <*address*> [<*port*>]
: If your computer has more than one IPv4 or IPv6 address, tinc will by default listen on all of them for incoming connections. This option can be used to restrict which addresses tinc listens on. Multiple ListenAddress variables may be specified, in which case listening sockets for each specified address are made.

  If no *port* is specified, the socket will listen on the port specified by the Port option, or to port 655 if neither is given. To only listen on a specific port but not to a specific address, use "*" for the *address*.

LocalDiscovery = <yes | no> (no)
: When enabled, tinc will try to detect peers that are on the same local network. This will allow direct communication using LAN addresses, even if both peers are behind a NAT and they only ConnectTo a third node outside the NAT, which normally would prevent the peers from learning each other's LAN address.

  Currently, local discovery is implemented by sending some packets to the local address of the node during UDP discovery. This will not work with old nodes that don't transmit their local address.

LocalDiscoveryAddress <*address*>
: If this variable is specified, local discovery packets are sent to the given *address*.

Mode = <router|switch|hub> (router)
: This option selects the way packets are routed to other daemons.

  router
  : In this mode Subnet variables in the host configuration files will be used to form a routing table. Only packets of routable protocols (IPv4 and IPv6) are supported in this mode.

    This is the default mode, and unless you really know you need another mode, don't change it.

  switch
  : In this mode the MAC addresses of the packets on the VPN will be used to dynamically create a routing table just like an Ethernet switch does. Unicast, multicast and broadcast packets of every protocol that runs over Ethernet are supported in this mode at the cost of frequent broadcast ARP requests and routing table updates.

    This mode is primarily useful if you want to bridge Ethernet segments.

  hub
  : This mode is almost the same as the switch mode, but instead every packet will be broadcast to the other daemons while no routing table is managed.

KeyExpire = <*seconds*> (3600)
: This option controls the time the encryption keys used to encrypt the data are valid. It is common practice to change keys at regular intervals to make it even harder for crackers, even though it is thought to be nearly impossible to crack a single key.

MACExpire = <*seconds*> (600)
: This option controls the amount of time MAC addresses are kept before they are removed. This only has effect when Mode is set to "switch".

MaxConnectionBurst = <*count*> (100)
: This option controls how many connections tinc accepts in quick succession. If there are more connections than the given number in a short time interval, tinc will

reduce the number of accepted connections to only one per second, until the burst has passed.

Name = <*name*> [required]
: This is a symbolic name for this connection. The name must consist only of alfanumeric and underscore characters (a-z, A-Z, 0-9 and _), and is case sensitive.

  If Name starts with a $, then the contents of the environment variable that follows will be used. In that case, invalid characters will be converted to underscores. If Name is $HOST, but no such environment variable exist, the hostname will be read using the gethostname() system call.

PingInterval = <*seconds*> (60)
: The number of seconds of inactivity that tinc will wait before sending a probe to the other end.

PingTimeout = <*seconds*> (5)
: The number of seconds to wait for a response to pings or to allow meta connections to block. If the other end doesn't respond within this time, the connection is terminated, and the others will be notified of this.

PriorityInheritance = <yes|no> (no) [experimental]
: When this option is enabled the value of the TOS field of tunneled IPv4 packets will be inherited by the UDP packets that are sent out.

PrivateKey = <*key*> [obsolete]
: This is the RSA private key for tinc. However, for safety reasons it is advised to store private keys of any kind in separate files. This prevents accidental eavesdropping if you are editting the configuration file.

PrivateKeyFile = <*path*> (`/etc/tinc/`*`netname`*`/rsa_key.priv`)
: This is the full path name of the RSA private key file that was generated by '`tinc generate-keys`'. It must be a full path, not a relative directory.

ProcessPriority = <low|normal|high>
: When this option is used the priority of the tincd process will be adjusted. Increasing the priority may help to reduce latency and packet loss on the VPN.

Proxy = socks4 | socks5 | http | exec ... [experimental]
: Use a proxy when making outgoing connections. The following proxy types are currently supported:

  socks4 <*address*> <*port*> [<*username*>]
  : Connects to the proxy using the SOCKS version 4 protocol. Optionally, a *username* can be supplied which will be passed on to the proxy server.

  socks5 <*address*> <*port*> [<*username*> <*password*>]
  : Connect to the proxy using the SOCKS version 5 protocol. If a *username* and *password* are given, basic username/password authentication will be used, otherwise no authentication will be used.

  http <*address*> <*port*>
  : Connects to the proxy and sends a HTTP CONNECT request.

  exec <*command*>
  : Executes the given command which should set up the outgoing connection. The environment variables `NAME`, `NODE`, `REMOTEADDRES` and `REMOTEPORT` are available.

ReplayWindow = <bytes> (32)
: This is the size of the replay tracking window for each remote node, in bytes. The window is a bitfield which tracks 1 packet per bit, so for example the default setting of 32 will track up to 256 packets in the window. In high bandwidth scenarios, setting this to a higher value can reduce packet loss from the interaction of replay tracking with underlying real packet loss and/or reordering. Setting this to zero will disable replay tracking completely and pass all traffic, but leaves tinc vulnerable to replay-based attacks on your traffic.

StrictSubnets = <yes|no> (no) [experimental]
: When this option is enabled tinc will only use Subnet statements which are present in the host config files in the local `/etc/tinc/`*`netname`*`/hosts/` directory. Subnets learned via connections to other nodes and which are not present in the local host config files are ignored.

TunnelServer = <yes|no> (no) [experimental]
: When this option is enabled tinc will no longer forward information between other tinc daemons, and will only allow connections with nodes for which host config files are present in the local `/etc/tinc/`*`netname`*`/hosts/` directory. Setting this options also implicitly sets StrictSubnets.

UDPDiscovery = <yes|no> (yes)
: When this option is enabled tinc will try to establish UDP connectivity to nodes, using TCP while it determines if a node is reachable over UDP. If it is disabled, tinc always assumes a node is reachable over UDP. Note that tinc will never use UDP with nodes that have TCPOnly enabled.

UDPDiscoveryKeepaliveInterval = <seconds> (9)
: The minimum amount of time between sending UDP ping datagrams to check UDP connectivity once it has been established. Note that these pings are large, since they are used to verify link MTU as well.

UDPDiscoveryInterval = <seconds> (2)
: The minimum amount of time between sending UDP ping datagrams to try to establish UDP connectivity.

UDPDiscoveryTimeout = <seconds> (30)
: If tinc doesn't receive any UDP ping replies over the specified interval, it will assume UDP communication is broken and will fall back to TCP.

UDPInfoInterval = <seconds> (5)
: The minimum amount of time between sending periodic updates about UDP addresses, which are mostly useful for UDP hole punching.

UDPRcvBuf = <bytes> (1048576)
: Sets the socket receive buffer size for the UDP socket, in bytes. If set to zero, the default buffer size will be used by the operating system. Note: this setting can have a significant impact on performance, especially raw throughput.

UDPSndBuf = <bytes> (1048576)
: Sets the socket send buffer size for the UDP socket, in bytes. If set to zero, the default buffer size will be used by the operating system. Note: this setting can have a significant impact on performance, especially raw throughput.

UPnP = <yes|udponly|no> (no)
: If this option is enabled then tinc will search for UPnP-IGD devices on the local network. It will then create and maintain port mappings for tinc's listening TCP

and UDP ports. If set to "udponly", tinc will only create a mapping for its UDP (data) port, not for its TCP (metaconnection) port. Note that tinc must have been built with miniupnpc support for this feature to be available. Furthermore, be advised that enabling this can have security implications, because the miniupnpc library that tinc uses might not be well-hardened with regard to malicious UPnP replies.

UPnPDiscoverWait = <seconds> (5)
The amount of time to wait for replies when probing the local network for UPnP devices.

UPnPRefreshPeriod = <seconds> (5)
How often tinc will re-add the port mapping, in case it gets reset on the UPnP device. This also controls the duration of the port mapping itself, which will be set to twice that duration.

4.4.2 Host configuration variables

Address = <*IP address*|*hostname*> [<port>] [recommended]
This variable is only required if you want to connect to this host. It must resolve to the external IP address where the host can be reached, not the one that is internal to the VPN. If no port is specified, the default Port is used. Multiple Address variables can be specified, in which case each address will be tried until a working connection has been established.

Cipher = <*cipher*> (blowfish)
The symmetric cipher algorithm used to encrypt UDP packets using the legacy protocol. Any cipher supported by LibreSSL or OpenSSL is recognized. Furthermore, specifying "none" will turn off packet encryption. It is best to use only those ciphers which support CBC mode. This option has no effect for connections using the SPTPS protocol, which always use AES-256-CTR.

ClampMSS = <yes|no> (yes)
This option specifies whether tinc should clamp the maximum segment size (MSS) of TCP packets to the path MTU. This helps in situations where ICMP Fragmentation Needed or Packet too Big messages are dropped by firewalls.

Compression = <*level*> (0)
This option sets the level of compression used for UDP packets. Possible values are 0 (off), 1 (fast zlib) and any integer up to 9 (best zlib), 10 (fast lzo) and 11 (best lzo).

Digest = <*digest*> (sha1)
The digest algorithm used to authenticate UDP packets using the legacy protocol. Any digest supported by LibreSSL or OpenSSL is recognized. Furthermore, specifying "none" will turn off packet authentication. This option has no effect for connections using the SPTPS protocol, which always use HMAC-SHA-256.

IndirectData = <yes|no> (no)
When set to yes, other nodes which do not already have a meta connection to you will not try to establish direct communication with you. It is best to leave this option out or set it to no.

MACLength = <*bytes*> (4)
The length of the message authentication code used to authenticate UDP packets using the legacy protocol. Can be anything from 0 up to the length of the digest produced by the digest algorithm. This option has no effect for connections using the SPTPS protocol, which never truncate MACs.

PMTU = <*mtu*> (1514)
: This option controls the initial path MTU to this node.

PMTUDiscovery = <yes|no> (yes)
: When this option is enabled, tinc will try to discover the path MTU to this node. After the path MTU has been discovered, it will be enforced on the VPN.

MTUInfoInterval = <seconds> (5)
: The minimum amount of time between sending periodic updates about relay path MTU. Useful for quickly determining MTU to indirect nodes.

Port = <*port*> (655)
: This is the port this tinc daemon listens on. You can use decimal portnumbers or symbolic names (as listed in `/etc/services`).

PublicKey = <*key*> [obsolete]
: This is the RSA public key for this host.

PublicKeyFile = <*path*> [obsolete]
: This is the full path name of the RSA public key file that was generated by '`tinc generate-keys`'. It must be a full path, not a relative directory.

  From version 1.0pre4 on tinc will store the public key directly into the host configuration file in PEM format, the above two options then are not necessary. Either the PEM format is used, or exactly **one of the above two options** must be specified in each host configuration file, if you want to be able to establish a connection with that host.

Subnet = <*address*[/*prefixlength*[#*weight*]]>
: The subnet which this tinc daemon will serve. Tinc tries to look up which other daemon it should send a packet to by searching the appropiate subnet. If the packet matches a subnet, it will be sent to the daemon who has this subnet in his host configuration file. Multiple subnet lines can be specified for each daemon.

  Subnets can either be single MAC, IPv4 or IPv6 addresses, in which case a subnet consisting of only that single address is assumed, or they can be a IPv4 or IPv6 network address with a prefixlength. For example, IPv4 subnets must be in a form like 192.168.1.0/24, where 192.168.1.0 is the network address and 24 is the number of bits set in the netmask. Note that subnets like 192.168.1.1/24 are invalid! Read a networking HOWTO/FAQ/guide if you don't understand this. IPv6 subnets are notated like fec0:0:0:1::/64. MAC addresses are notated like 0:1a:2b:3c:4d:5e.

  Prefixlength is the number of bits set to 1 in the netmask part; for example: netmask 255.255.255.0 would become /24, 255.255.252.0 becomes /22. This conforms to standard CIDR notation as described in RFC1519

  A Subnet can be given a weight to indicate its priority over identical Subnets owned by different nodes. The default weight is 10. Lower values indicate higher priority. Packets will be sent to the node with the highest priority, unless that node is not reachable, in which case the node with the next highest priority will be tried, and so on.

TCPonly = <yes|no> (no)
: If this variable is set to yes, then the packets are tunnelled over a TCP connection instead of a UDP connection. This is especially useful for those who want to run a tinc daemon from behind a masquerading firewall, or if UDP packet routing is disabled somehow. Setting this options also implicitly sets IndirectData.

Weight = <weight>
: If this variable is set, it overrides the weight given to connections made with another host. A higher weight means a lower priority is given to this connection when broadcasting or forwarding packets.

4.4.3 Scripts

Apart from reading the server and host configuration files, tinc can also run scripts at certain moments. Below is a list of filenames of scripts and a description of when they are run. A script is only run if it exists and if it is executable.

Scripts are run synchronously; this means that tinc will temporarily stop processing packets until the called script finishes executing. This guarantees that scripts will execute in the exact same order as the events that trigger them. If you need to run commands asynchronously, you have to ensure yourself that they are being run in the background.

Under Windows (not Cygwin), the scripts should have the extension `.bat` or `.cmd`.

`/etc/tinc/`*`netname`*`/tinc-up`
: This is the most important script. If it is present it will be executed right after the tinc daemon has been started and has connected to the virtual network device. It should be used to set up the corresponding network interface, but can also be used to start other things.

  Under Windows you can use the Network Connections control panel instead of creating this script.

`/etc/tinc/`*`netname`*`/tinc-down`
: This script is started right before the tinc daemon quits.

`/etc/tinc/`*`netname`*`/hosts/`*`host`*`-up`
: This script is started when the tinc daemon with name *host* becomes reachable.

`/etc/tinc/`*`netname`*`/hosts/`*`host`*`-down`
: This script is started when the tinc daemon with name *host* becomes unreachable.

`/etc/tinc/`*`netname`*`/host-up`
: This script is started when any host becomes reachable.

`/etc/tinc/`*`netname`*`/host-down`
: This script is started when any host becomes unreachable.

`/etc/tinc/`*`netname`*`/subnet-up`
: This script is started when a Subnet becomes reachable. The Subnet and the node it belongs to are passed in environment variables.

`/etc/tinc/`*`netname`*`/subnet-down`
: This script is started when a Subnet becomes unreachable.

`/etc/tinc/`*`netname`*`/invitation-created`
: This script is started when a new invitation has been created.

`/etc/tinc/`*`netname`*`/invitation-accepted`
: This script is started when an invitation has been used.

The scripts are started without command line arguments, but can make use of certain environment variables. Under UNIX like operating systems the names of environment variables must be preceded by a $ in scripts. Under Windows, in `.bat` or `.cmd` files, they have to be put between % signs.

`NETNAME` If a netname was specified, this environment variable contains it.

`NAME` Contains the name of this tinc daemon.

`DEVICE` Contains the name of the virtual network device that tinc uses.

`INTERFACE`
Contains the name of the virtual network interface that tinc uses. This should be used for commands like ifconfig.

`NODE` When a host becomes (un)reachable, this is set to its name. If a subnet becomes (un)reachable, this is set to the owner of that subnet.

`REMOTEADDRESS`
When a host becomes (un)reachable, this is set to its real address.

`REMOTEPORT`
When a host becomes (un)reachable, this is set to the port number it uses for communication with other tinc daemons.

`SUBNET` When a subnet becomes (un)reachable, this is set to the subnet.

`WEIGHT` When a subnet becomes (un)reachable, this is set to the subnet weight.

`INVITATION_FILE`
When the `invitation-created` script is called, this is set to the file where the invitation details will be stored.

`INVITATION_URL`
When the `invitation-created` script is called, this is set to the invitation URL that has been created.

Do not forget that under UNIX operating systems, you have to make the scripts executable, using the command '`chmod a+x script`'.

4.4.4 How to configure

Step 1. Creating initial configuration files.

The initial directory structure, configuration files and public/private keypairs are created using the following command:

```
tinc -n netname init name
```

(You will need to run this as root, or use "sudo".) This will create the configuration directory `/etc/tinc/`*`netname`*`.`, and inside it will create another directory named `hosts/`. In the configuration directory, it will create the file `tinc.conf` with the following contents:

```
Name = name
```

It will also create private RSA and Ed25519 keys, which will be stored in the files `rsa_key.priv` and `ed25519_key.priv`. It will also create a host configuration file `hosts/`*`name`*, which will contain the corresponding public RSA and Ed25519 keys.

Finally, on UNIX operating systems, it will create an executable script `tinc-up`, which will initially not do anything except warning that you should edit it.

Step 2. Modifying the initial configuration.

Unless you want to use tinc in switch mode, you should now configure which range of addresses you will use on the VPN. Let's assume you will be part of a VPN which uses the address range 192.168.0.0/16, and you yourself have a smaller portion of that range: 192.168.2.0/24. Then you should run the following command:

```
tinc -n netname add subnet 192.168.2.0/24
```

This will add a Subnet statement to your host configuration file. Try opening the file `/etc/tinc/netname/hosts/name` in an editor. You should now see a file containing the public RSA and Ed25519 keys (which looks like a bunch of random characters), and the following line at the bottom:

```
Subnet = 192.168.2.0/24
```

If you will use more than one address range, you can add more Subnets. For example, if you also use the IPv6 subnet fec0:0:0:2::/64, you can add it as well:

```
tinc -n netname add subnet fec0:0:0:2::/24
```

This will add another line to the file `hosts/name`. If you make a mistake, you can undo it by simply using '`del`' instead of '`add`'.

If you want other tinc daemons to create meta-connections to your daemon, you should add your public IP address or hostname to your host configuration file. For example, if your hostname is foo.example.org, run:

```
tinc -n netname add address foo.example.org
```

If you already know to which daemons your daemon should make meta-connections, you should configure that now as well. Suppose you want to connect to a daemon named "bar", run:

```
tinc -n netname add connectto bar
```

Note that you specify the Name of the other daemon here, not an IP address or hostname! When you start tinc, and it tries to make a connection to "bar", it will look for a host configuration file named `hosts/bar`, and will read Address statements and public keys from that file.

Step 2. Exchanging configuration files.

If your daemon has a ConnectTo = bar statement in its `tinc.conf` file, or if bar has a ConnectTo your daemon, then you both need each other's host configuration files. You should send `hosts/name` to bar, and bar should send you his file which you should move to `hosts/bar`. If you are on a UNIX platform, you can easily send an email containing the necessary information using the following command (assuming the owner of bar has the email address bar@example.org):

```
tinc -n netname export | mail -s "My config file" bar@example.org
```

If the owner of bar does the same to send his host configuration file to you, you can probably pipe his email through the following command, or you can just start this command in a terminal and copy&paste the email:

```
tinc -n netname import
```

If you are the owner of bar yourself, and you have SSH access to that computer, you can also swap the host configuration files using the following command:

```
tinc -n netname export \
    | ssh bar.example.org tinc -n netname exchange \
    | tinc -n netname import
```

You should repeat this for all nodes you ConnectTo, or which ConnectTo you. However, remember that you do not need to ConnectTo all nodes in the VPN; it is only necessary to create one or a few meta-connections, after the connections are made tinc will learn about all the other nodes in the VPN, and will automatically make other connections as necessary.

4.5 Network interfaces

Before tinc can start transmitting data over the tunnel, it must set up the virtual network interface.

First, decide which IP addresses you want to have associated with these devices, and what network mask they must have.

Tinc will open a virtual network device (`/dev/tun`, `/dev/tap0` or similar), which will also create a network interface called something like '`tun0`', '`tap0`'. If you are using the Linux tun/tap driver, the network interface will by default have the same name as the *netname*. Under Windows you can change the name of the network interface from the Network Connections control panel.

You can configure the network interface by putting ordinary ifconfig, route, and other commands to a script named `/etc/tinc/`*`netname`*`/tinc-up`. When tinc starts, this script will be executed. When tinc exits, it will execute the script named `/etc/tinc/`*`netname`*`/tinc-down`, but normally you don't need to create that script. You can manually open the script in an editor, or use the following command:

```
tinc -n netname edit tinc-up
```

An example `tinc-up` script, that would be appropriate for the scenario in the previous section, is:

```
#!/bin/sh
ifconfig $INTERFACE 192.168.2.1 netmask 255.255.0.0
ip addr add fec0:0:0:2::/48 dev $INTERFACE
```

The first command gives the interface an IPv4 address and a netmask. The kernel will also automatically add an IPv4 route to this interface, so normally you don't need to add route commands to the `tinc-up` script. The kernel will also bring the interface up after this command. The netmask is the mask of the *entire* VPN network, not just your own subnet. The second command gives the interface an IPv6 address and netmask, which will also automatically add an IPv6 route. If you only want to use "ip addr" commands on Linux, don't forget that it doesn't bring the interface up, unlike ifconfig, so you need to add '`ip link set $INTERFACE up`' in that case.

The exact syntax of the ifconfig and route commands differs from platform to platform. You can look up the commands for setting addresses and adding routes in Chapter 9 [Platform specific information], page 45, but it is best to consult the manpages of those utilities on your platform.

4.6 Example configuration

Imagine the following situation. Branch A of our example 'company' wants to connect three branch offices in B, C and D using the Internet. All four offices have a 24/7 connection to the Internet.

A is going to serve as the center of the network. B and C will connect to A, and D will connect to C. Each office will be assigned their own IP network, 10.x.0.0.

```
A: net 10.1.0.0 mask 255.255.0.0 gateway 10.1.54.1 internet IP 1.2.3.4
B: net 10.2.0.0 mask 255.255.0.0 gateway 10.2.1.12 internet IP 2.3.4.5
C: net 10.3.0.0 mask 255.255.0.0 gateway 10.3.69.254 internet IP 3.4.5.6
D: net 10.4.0.0 mask 255.255.0.0 gateway 10.4.3.32 internet IP 4.5.6.7
```

Here, "gateway" is the VPN IP address of the machine that is running the tincd, and "internet IP" is the IP address of the firewall, which does not need to run tincd, but it must do a port forwarding of TCP and UDP on port 655 (unless otherwise configured).

In this example, it is assumed that eth0 is the interface that points to the inner (physical) LAN of the office, although this could also be the same as the interface that leads to the Internet. The configuration of the real interface is also shown as a comment, to give you an idea of how these example host is set up. All branches use the netname 'company' for this particular VPN.

Each branch is set up using the '`tinc init`' and '`tinc config`' commands, here we just show the end results:

For Branch A

BranchA would be configured like this:

In `/etc/tinc/company/tinc-up`:

```
#!/bin/sh

# Real interface of internal network:
# ifconfig eth0 10.1.54.1 netmask 255.255.0.0

ifconfig $INTERFACE 10.1.54.1 netmask 255.0.0.0
```

and in `/etc/tinc/company/tinc.conf`:

```
Name = BranchA
```

On all hosts, `/etc/tinc/company/hosts/BranchA` contains:

```
Subnet = 10.1.0.0/16
Address = 1.2.3.4

-----BEGIN RSA PUBLIC KEY-----
...
-----END RSA PUBLIC KEY-----
```

Note that the IP addresses of eth0 and the VPN interface are the same. This is quite possible, if you make sure that the netmasks of the interfaces are different. It is in fact recommended to give both real internal network interfaces and VPN interfaces the same IP address, since that will make things a lot easier to remember and set up.

For Branch B

In `/etc/tinc/company/tinc-up`:

```
#!/bin/sh

# Real interface of internal network:
# ifconfig eth0 10.2.43.8 netmask 255.255.0.0

ifconfig $INTERFACE 10.2.1.12 netmask 255.0.0.0
```

and in `/etc/tinc/company/tinc.conf`:

```
Name = BranchB
ConnectTo = BranchA
```

Note here that the internal address (on eth0) doesn't have to be the same as on the VPN interface. Also, ConnectTo is given so that this node will always try to connect to BranchA.

On all hosts, in `/etc/tinc/company/hosts/BranchB`:

```
Subnet = 10.2.0.0/16
Address = 2.3.4.5

-----BEGIN RSA PUBLIC KEY-----
...
-----END RSA PUBLIC KEY-----
```

For Branch C

In `/etc/tinc/company/tinc-up`:

```
#!/bin/sh
```

```
# Real interface of internal network:
# ifconfig eth0 10.3.69.254 netmask 255.255.0.0

ifconfig $INTERFACE 10.3.69.254 netmask 255.0.0.0
```

and in `/etc/tinc/company/tinc.conf`:

```
Name = BranchC
ConnectTo = BranchA
```

C already has another daemon that runs on port 655, so they have to reserve another port for tinc. It knows the portnumber it has to listen on from it's own host configuration file.

On all hosts, in `/etc/tinc/company/hosts/BranchC`:

```
Address = 3.4.5.6
Subnet = 10.3.0.0/16
Port = 2000

-----BEGIN RSA PUBLIC KEY-----
...
-----END RSA PUBLIC KEY-----
```

For Branch D

In `/etc/tinc/company/tinc-up`:

```
#!/bin/sh

# Real interface of internal network:
# ifconfig eth0 10.4.3.32 netmask 255.255.0.0

ifconfig $INTERFACE 10.4.3.32 netmask 255.0.0.0
```

and in `/etc/tinc/company/tinc.conf`:

```
Name = BranchD
ConnectTo = BranchC
```

D will be connecting to C, which has a tincd running for this network on port 2000. It knows the port number from the host configuration file.

On all hosts, in `/etc/tinc/company/hosts/BranchD`:

```
Subnet = 10.4.0.0/16
Address = 4.5.6.7

-----BEGIN RSA PUBLIC KEY-----
...
-----END RSA PUBLIC KEY-----
```

Key files

A, B, C and D all have their own public/private keypairs:

The private RSA key is stored in `/etc/tinc/company/rsa_key.priv`, the private Ed25519 key is stored in `/etc/tinc/company/ed25519_key.priv`, and the public RSA and Ed25519 keys are put into the host configuration file in the `/etc/tinc/company/hosts/` directory.

Starting

After each branch has finished configuration and they have distributed the host configuration files amongst them, they can start their tinc daemons. They don't necessarily have to wait for the other branches to have started their daemons, tinc will try connecting until they are available.

5 Running tinc

If everything else is done, you can start tinc by typing the following command:

```
tinc -n netname start
```

Tinc will detach from the terminal and continue to run in the background like a good daemon. If there are any problems however you can try to increase the debug level and look in the syslog to find out what the problems are.

5.1 Runtime options

Besides the settings in the configuration file, tinc also accepts some command line options.

`-c, --config=path`
: Read configuration options from the directory *path*. The default is `/etc/tinc/netname/`.

`-D, --no-detach`
: Don't fork and detach. This will also disable the automatic restart mechanism for fatal errors.

`-d, --debug=level`
: Set debug level to *level*. The higher the debug level, the more gets logged. Everything goes via syslog.

`-n, --net=netname`
: Use configuration for net *netname*. This will let tinc read all configuration files from `/etc/tinc/netname/`. Specifying . for *netname* is the same as not specifying any *netname*. See Section 4.2 [Multiple networks], page 9.

`--pidfile=filename`
: Store a cookie in *filename* which allows tinc to authenticate. If unspecified, the default is `/var/run/tinc.netname.pid`.

`-o, --option=[HOST.]KEY=VALUE`
: Without specifying a *HOST*, this will set server configuration variable *KEY* to *VALUE*. If specified as *HOST.KEY=VALUE*, this will set the host configuration variable *KEY* of the host named *HOST* to *VALUE*. This option can be used more than once to specify multiple configuration variables.

`-L, --mlock`
: Lock tinc into main memory. This will prevent sensitive data like shared private keys to be written to the system swap files/partitions.

  This option is not supported on all platforms.

`--logfile[=file]`
: Write log entries to a file instead of to the system logging facility. If *file* is omitted, the default is `/var/log/tinc.netname.log`.

`--bypass-security`
: Disables encryption and authentication. Only useful for debugging.

`-R, --chroot`
: Change process root directory to the directory where the config file is located (`/etc/tinc/netname/` as determined by -n/–net option or as given by -c/–config option), for added security. The chroot is performed after all the initialization is done, after writing pid files and opening network sockets.

  Note that this option alone does not do any good without -U/–user, below.

Note also that tinc can't run scripts anymore (such as tinc-down or host-up), unless it's setup to be runnable inside chroot environment.

This option is not supported on all platforms.

`-U, --user=`*user*
: Switch to the given *user* after initialization, at the same time as chroot is performed (see –chroot above). With this option tinc drops privileges, for added security.

  This option is not supported on all platforms.

`--help`
: Display a short reminder of these runtime options and terminate.

`--version`
: Output version information and exit.

5.2 Signals

You can also send the following signals to a running tincd process:

'`ALRM`'
: Forces tinc to try to connect to all uplinks immediately. Usually tinc attempts to do this itself, but increases the time it waits between the attempts each time it failed, and if tinc didn't succeed to connect to an uplink the first time after it started, it defaults to the maximum time of 15 minutes.

'`HUP`'
: Partially rereads configuration files. Connections to hosts whose host config file are removed are closed. New outgoing connections specified in `tinc.conf` will be made. If the –logfile option is used, this will also close and reopen the log file, useful when log rotation is used.

5.3 Debug levels

The tinc daemon can send a lot of messages to the syslog. The higher the debug level, the more messages it will log. Each level inherits all messages of the previous level:

'`0`'
: This will log a message indicating tinc has started along with a version number. It will also log any serious error.

'`1`'
: This will log all connections that are made with other tinc daemons.

'`2`'
: This will log status and error messages from scripts and other tinc daemons.

'`3`'
: This will log all requests that are exchanged with other tinc daemons. These include authentication, key exchange and connection list updates.

'`4`'
: This will log a copy of everything received on the meta socket.

'`5`'
: This will log all network traffic over the virtual private network.

5.4 Solving problems

If tinc starts without problems, but if the VPN doesn't work, you will have to find the cause of the problem. The first thing to do is to start tinc with a high debug level in the foreground, so you can directly see everything tinc logs:

```
tincd -n netname -d5 -D
```

If tinc does not log any error messages, then you might want to check the following things:

- `tinc-up` script Does this script contain the right commands? Normally you must give the interface the address of this host on the VPN, and the netmask must be big enough so that the entire VPN is covered.

- Subnet Does the Subnet (or Subnets) in the host configuration file of this host match the portion of the VPN that belongs to this host?
- Firewalls and NATs Do you have a firewall or a NAT device (a masquerading firewall or perhaps an ADSL router that performs masquerading)? If so, check that it allows TCP and UDP traffic on port 655. If it masquerades and the host running tinc is behind it, make sure that it forwards TCP and UDP traffic to port 655 to the host running tinc. You can add '`TCPOnly = yes`' to your host config file to force tinc to only use a single TCP connection, this works through most firewalls and NATs.

5.5 Error messages

What follows is a list of the most common error messages you might find in the logs. Some of them will only be visible if the debug level is high enough.

'`Could not open /dev/tap0: No such device`'

- You forgot to 'modprobe netlink_dev' or 'modprobe ethertap'.
- You forgot to compile 'Netlink device emulation' in the kernel.

'`Can't write to /dev/net/tun: No such device`'

- You forgot to 'modprobe tun'.
- You forgot to compile 'Universal TUN/TAP driver' in the kernel.
- The tun device is located somewhere else in `/dev/`.

'`Network address and prefix length do not match!`'

- The Subnet field must contain a *network* address, trailing bits should be 0.
- If you only want to use one IP address, set the netmask to /32.

'`Error reading RSA key file 'rsa_key.priv': No such file or directory`'

- You forgot to create a public/private keypair.
- Specify the complete pathname to the private key file with the '`PrivateKeyFile`' option.

'`Warning: insecure file permissions for RSA private key file 'rsa_key.priv'!`'

- The private key file is readable by users other than root. Use chmod to correct the file permissions.

'`Creating metasocket failed: Address family not supported`'

- By default tinc tries to create both IPv4 and IPv6 sockets. On some platforms this might not be implemented. If the logs show '`Ready`' later on, then at least one metasocket was created, and you can ignore this message. You can add '`AddressFamily = ipv4`' to `tinc.conf` to prevent this from happening.

'`Cannot route packet: unknown IPv4 destination 1.2.3.4`'

- You try to send traffic to a host on the VPN for which no Subnet is known.
- If it is a broadcast address (ending in .255), it probably is a samba server or a Windows host sending broadcast packets. You can ignore it.

'`Cannot route packet: ARP request for unknown address 1.2.3.4`'

- You try to send traffic to a host on the VPN for which no Subnet is known.

'`Packet with destination 1.2.3.4 is looping back to us!`'

- Something is not configured right. Packets are being sent out to the virtual network device, but according to the Subnet directives in your host configuration file, those packets should go to your own host. Most common mistake is that you have a Subnet line in your host configuration file with a prefix length which is just as large as the prefix of the virtual network interface. The latter

should in almost all cases be larger. Rethink your configuration. Note that you will only see this message if you specified a debug level of 5 or higher!

- Chances are that a '`Subnet = ...`' line in the host configuration file of this tinc daemon is wrong. Change it to a subnet that is accepted locally by another interface, or if that is not the case, try changing the prefix length into /32.

'`Node foo (1.2.3.4) is not reachable`'

- Node foo does not have a connection anymore, its tinc daemon is not running or its connection to the Internet is broken.

'`Received UDP packet from unknown source 1.2.3.4 (port 12345)`'

- If you see this only sporadically, it is harmless and caused by a node sending packets using an old key.
- If you see this often and another node is not reachable anymore, then a NAT (masquerading firewall) is changing the source address of UDP packets. You can add '`TCPOnly = yes`' to host configuration files to force all VPN traffic to go over a TCP connection.

'`Got bad/bogus/unauthorized REQUEST from foo (1.2.3.4 port 12345)`'

- Node foo does not have the right public/private keypair. Generate new keypairs and distribute them again.
- An attacker tries to gain access to your VPN.
- A network error caused corruption of metadata sent from foo.

5.6 Sending bug reports

If you really can't find the cause of a problem, or if you suspect tinc is not working right, you can send us a bugreport, see Section 10.1 [Contact information], page 47. Be sure to include the following information in your bugreport:

- A clear description of what you are trying to achieve and what the problem is.
- What platform (operating system, version, hardware architecture) and which version of tinc you use.
- If compiling tinc fails, a copy of `config.log` and the error messages you get.
- Otherwise, a copy of `tinc.conf`, `tinc-up` and all files in the `hosts/` directory.
- The output of the commands '`ifconfig -a`' and '`route -n`' (or '`netstat -rn`' if that doesn't work).
- The output of any command that fails to work as it should (like ping or traceroute).

6 Controlling tinc

You can start, stop, control and inspect a running tincd through the tinc command. A quick example:

```
tinc -n netname reload
```

If tinc is started without a command, it will act as a shell; it will display a prompt, and commands can be entered on the prompt. If tinc is compiled with libreadline, history and command completion are available on the prompt. One can also pipe a script containing commands through tinc. In that case, lines starting with a # symbol will be ignored.

6.1 tinc runtime options

`-c, --config=path`
: Read configuration options from the directory *path*. The default is `/etc/tinc/netname/`.

`-n, --net=netname`
: Use configuration for net *netname*. See Section 4.2 [Multiple networks], page 9.

`--pidfile=filename`
: Use the cookie from *filename* to authenticate with a running tinc daemon. If unspecified, the default is `/var/run/tinc.netname.pid`.

`--force` Force some commands to work despite warnings.

`--help` Display a short reminder of runtime options and commands, then terminate.

`--version`
: Output version information and exit.

6.2 tinc environment variables

`NETNAME` If no netname is specified on the command line with the `-n` option, the value of this environment variable is used.

6.3 tinc commands

`init [name]`
: Create initial configuration files and RSA and Ed25519 keypairs with default length. If no *name* for this node is given, it will be asked for.

`get variable`
: Print the current value of configuration variable *variable*. If more than one variable with the same name exists, the value of each of them will be printed on a separate line.

`set variable value`
: Set configuration variable *variable* to the given *value*. All previously existing configuration variables with the same name are removed. To set a variable for a specific host, use the notation *host.variable*.

`add variable value`
: As above, but without removing any previously existing configuration variables. If the variable already exists with the given value, nothing happens.

`del variable [value]`
: Remove configuration variables with the same name and *value*. If no *value* is given, all configuration variables with the same name will be removed.

`edit` *`filename`*
: Start an editor for the given configuration file. You do not need to specify the full path to the file.

`export`
: Export the host configuration file of the local node to standard output.

`export-all`
: Export all host configuration files to standard output.

`import`
: Import host configuration file(s) generated by the tinc export command from standard input. Already existing host configuration files are not overwritten unless the option –force is used.

`exchange`
: The same as export followed by import.

`exchange-all`
: The same as export-all followed by import.

`invite` *`name`*
: Prepares an invitation for a new node with the given *name*, and prints a short invitation URL that can be used with the join command.

`join [`*`URL`*`]`
: Join an existing VPN using an invitation URL created using the invite command. If no *URL* is given, it will be read from standard input.

`start [tincd options]`
: Start '`tincd`', optionally with the given extra options.

`stop`
: Stop '`tincd`'.

`restart [tincd options]`
: Restart '`tincd`', optionally with the given extra options.

`reload`
: Partially rereads configuration files. Connections to hosts whose host config files are removed are closed. New outgoing connections specified in `tinc.conf` will be made.

`pid`
: Shows the PID of the currently running '`tincd`'.

`generate-keys [`*`bits`*`]`
: Generate both RSA and Ed25519 keypairs (see below) and exit. tinc will ask where you want to store the files, but will default to the configuration directory (you can use the -c or -n option).

`generate-ed25519-keys`
: Generate public/private Ed25519 keypair and exit.

`generate-rsa-keys [`*`bits`*`]`
: Generate public/private RSA keypair and exit. If *bits* is omitted, the default length will be 2048 bits. When saving keys to existing files, tinc will not delete the old keys; you have to remove them manually.

`dump [reachable] nodes`
: Dump a list of all known nodes in the VPN. If the reachable keyword is used, only lists reachable nodes.

`dump edges`
: Dump a list of all known connections in the VPN.

`dump subnets`
: Dump a list of all known subnets in the VPN.

`dump connections`
: Dump a list of all meta connections with ourself.

`dump graph | digraph`
: Dump a graph of the VPN in dotty format. Nodes are colored according to their reachability: red nodes are unreachable, orange nodes are indirectly reachable, green nodes are directly reachable. Black nodes are either directly or indirectly reachable, but direct reachability has not been tried yet.

`dump invitations`
: Dump a list of outstanding invitations. The filename of the invitation, as well as the name of the node that is being invited is shown for each invitation.

`info` *`node | subnet | address`*
: Show information about a particular *node*, *subnet* or *address*. If an *address* is given, any matching subnet will be shown.

`purge`
: Purges all information remembered about unreachable nodes.

`debug` *`level`*
: Sets debug level to *level*.

`log [`*`level`*`]`
: Capture log messages from a running tinc daemon. An optional debug level can be given that will be applied only for log messages sent to tinc.

`retry`
: Forces tinc to try to connect to all uplinks immediately. Usually tinc attempts to do this itself, but increases the time it waits between the attempts each time it failed, and if tinc didn't succeed to connect to an uplink the first time after it started, it defaults to the maximum time of 15 minutes.

`disconnect` *`node`*
: Closes the meta connection with the given *node*.

`top`
: If tinc is compiled with libcurses support, this will display live traffic statistics for all the known nodes, similar to the UNIX top command. See below for more information.

`pcap`
: Dump VPN traffic going through the local tinc node in pcap-savefile format to standard output, from where it can be redirected to a file or piped through a program that can parse it directly, such as tcpdump.

`network [`*`netname`*`]`
: If *netname* is given, switch to that network. Otherwise, display a list of all networks for which configuration files exist.

`fsck`
: This will check the configuration files for possible problems, such as unsafe file permissions, missing executable bit on script, unknown and obsolete configuration variables, wrong public and/or private keys, and so on.

  When problems are found, this will be printed on a line with WARNING or ERROR in front of it. Most problems must be corrected by the user itself, however in some cases (like file permissions and missing public keys), tinc will ask if it should fix the problem.

`sign [`*`filename`*`]`
: Sign a file with the local node's private key. If no *filename* is given, the file is read from standard input. The signed file is written to standard output.

`verify` *`name`* [*`filename`*]
: Check the signature of a file against a node's public key. The *name* of the node must be given, or can be "." to check against the local node's public key, or "*" to allow a signature from any node whose public key is known. If no *filename* is given, the file is read from standard input. If the verification is succesful, a copy of the input with the signature removed is written to standard output, and the exit code will be zero. If the verification failed, nothing will be written to standard output, and the exit code will be non-zero.

6.4 tinc examples

Examples of some commands:

```
tinc -n vpn dump graph | circo -Txlib
tinc -n vpn pcap | tcpdump -r -
tinc -n vpn top
```

Examples of changing the configuration using tinc:

```
tinc -n vpn init foo
tinc -n vpn add Subnet 192.168.1.0/24
tinc -n vpn add bar.Address bar.example.com
tinc -n vpn add ConnectTo bar
tinc -n vpn export | gpg --clearsign | mail -s "My config" vpnmaster@example.com
```

6.5 tinc top

The top command connects to a running tinc daemon and repeatedly queries its per-node traffic counters. It displays a list of all the known nodes in the left-most column, and the amount of bytes and packets read from and sent to each node in the other columns. By default, the information is updated every second. The behaviour of the top command can be changed using the following keys:

`s`
: Change the interval between updates. After pressing the `s` key, enter the desired interval in seconds, followed by enter. Fractional seconds are honored. Intervals lower than 0.1 seconds are not allowed.

`c`
: Toggle between displaying current traffic rates (in packets and bytes per second) and cummulative traffic (total packets and bytes since the tinc daemon started).

`n`
: Sort the list of nodes by name.

`i`
: Sort the list of nodes by incoming amount of bytes.

`I`
: Sort the list of nodes by incoming amount of packets.

`o`
: Sort the list of nodes by outgoing amount of bytes.

`O`
: Sort the list of nodes by outgoing amount of packets.

`t`
: Sort the list of nodes by sum of incoming and outgoing amount of bytes.

`T`
: Sort the list of nodes by sum of incoming and outgoing amount of packets.

`b`
: Show amount of traffic in bytes.

`k`
: Show amount of traffic in kilobytes.

`M`
: Show amount of traffic in megabytes.

`G`
: Show amount of traffic in gigabytes.

`q`
: Quit.

7 Invitations

Invitations are an easy way to add new nodes to an existing VPN. Invitations can be created on an existing node using the `tinc invite` command, which generates a relatively short URL which can be given to someone else, who uses the `tinc join` command to automatically set up tinc so it can connect to the inviting node. The next sections describe how invitations actually work, and how to further automate the invitations.

7.1 How invitations work

When an invitation is created on a node (which from now on we will call the server) using the `tinc invite` command, an invitation file is created that contains all the information necessary for the invitee (which we will call the client) to create its configuration files. The invitation file is stays on the server, but a URL is generated that has enough information for the client to contact the server and to retrieve the invitation file. The whole URL is around 80 characters long and looks like this:

```
server.example.org:12345/cW1NhLHS-1WPFlcFio8ztYHvewTTKYZp8BjEKg3vbMtDz7w4
```

It is composed of four parts:

```
hostname : port / keyhash cookie
```

The hostname and port tell the client how to reach the tinc daemon on the server. The part after the slash looks like one blob, but is composed of two parts. The keyhash is the hash of the public key of the server. The cookie is a shared secret that identifies the client to the server.

When the client connects to the server in order to join the VPN, the client and server will exchange temporary public keys. The client verifies that the hash of the server's public key matches the keyhash from the invitation URL. If not, it will immediately exit with an error. Otherwise, an ECDH exchange will happen so the client and server can communicate privately with each other. The client will then present the cookie to the server. The server uses this to look up the corresponding invitation file it generated earlier. If it exists, it will send the invitation file to the client. The client will also create a permanent public key, and send it to the server. After the exchange is completed, the connection is broken. The server creates a host config file for the client containing the client's permanent public key, and the client creates tinc.conf, host config files and possibly a tinc-up script based on the information in the invitation file.

It is important that the invitation URL is kept secret until it is used; if another person gets a copy of the invitation URL before the real client runs the `tinc join` command, then that other person can try to join the VPN.

7.2 Invitation file format

The contents of an invitation file that is generated by the `tinc invite` command looks like this:

```
Name = client
Netname = vpn
ConnectTo = server
#------------------------------------#
Name = server
Ed25519PublicKey = augbnwegoij123587...
Address = server.example.com
```

The file is basically a concatenation of several host config blocks. Each host config block starts with `Name = ...`. Lines that look like `#---#` are not important, it just makes it easier for humans to read the file.

The first host config block is always the one representing the invitee. So the first Name statement determines the name that the invitee will get. From the first block, the `tinc.conf` and

`hosts/client` files will be generated; the `tinc join` command on the client will automatically separate statements based on whether they should be in `tinc.conf` or in a host config file. Some statements are special and are treated differently:

Netname = <*netname*>
: This is a hint to the invitee which netname to use for the VPN. It is used if the invitee did not already specify a netname, and if there is no pre-existing configuration with the same netname.

Ifconfig = <*address*[/*netmask*] | dhcp | dhcp6 | slaac>
: This is a hint for generating a `tinc-up` script. If an address is specified, a command will be added to `tinc-up` so the VPN interface will be configured to have the given address. If it is the word "dhcp", a command will be added to start a DHCP client on the VPN interface. If it is the word dhcpv6, it will be a DHCPv6 client. If it is "slaac", then it will add commands to enable IPv6 stateless address autoconfiguration. It is also possible to specify a MAC address, in which case a command will be added to set the MAC address of the VPN interface.

  The exact commands added to the `tinc-up` script depends on the operating system the client is using. Multiple Ifconfig statements can be specified, however one should only use one Ifconfig statement per address family.

Route = <*address*[/*netmask*]> [<*gateway*>]
: This is a hint for generating a `tinc-up` script. Route statements are similar to Ifconfig statements, but add routes instead of addresses. These only allow IPv4 and IPv6 routes. If no gateway address is specified, the route is directed to the VPN interface. In general, a gateway is only necessary when running tinc in switch mode.

Subsequent host config blocks are copied verbatim into their respective files in `hosts/`. The invitation file generated by `tinc invite` will normally only contain two blocks; one for the client and one for the server.

7.3 Writing an invitation-created script

When an invitation is generated, the "invitation-created" script is called (if it exists) right after the invitation file is written, but before the URL has been written to stdout. This allows one to change the invitation file automatically before the invitation URL is passed to the invitee. Here is an example shell script that aproximately recreates the default invitation file:

```
#!/bin/sh

cat >$INVITATION_FILE <<EOF
Name = $NODE
Netname = $NETNAME
ConnectTo = $NAME
#---------------#
EOF

tinc export >>$INVITATION_FILE
```

You can add more ConnectTo statements, and change ‘tinc export‘ to ‘tinc export-all‘ for example. But you can also use the script to automatically hand out a Subnet to the invitee. Note that the script doesn't have to be a shell script, you can use any language, it just has to be executable.

8 Technical information

8.1 The connection

Tinc is a daemon that takes VPN data and transmit that to another host computer over the existing Internet infrastructure.

8.1.1 The UDP tunnel

The data itself is read from a character device file, the so-called *virtual network device*. This device is associated with a network interface. Any data sent to this interface can be read from the device, and any data written to the device gets sent from the interface. There are two possible types of virtual network devices: 'tun' style, which are point-to-point devices which can only handle IPv4 and/or IPv6 packets, and 'tap' style, which are Ethernet devices and handle complete Ethernet frames.

So when tinc reads an Ethernet frame from the device, it determines its type. When tinc is in it's default routing mode, it can handle IPv4 and IPv6 packets. Depending on the Subnet lines, it will send the packets off to their destination IP address. In the 'switch' and 'hub' mode, tinc will use broadcasts and MAC address discovery to deduce the destination of the packets. Since the latter modes only depend on the link layer information, any protocol that runs over Ethernet is supported (for instance IPX and Appletalk). However, only 'tap' style devices provide this information.

After the destination has been determined, the packet will be compressed (optionally), a sequence number will be added to the packet, the packet will then be encrypted and a message authentication code will be appended.

When that is done, time has come to actually transport the packet to the destination computer. We do this by sending the packet over an UDP connection to the destination host. This is called *encapsulating*, the VPN packet (though now encrypted) is encapsulated in another IP datagram.

When the destination receives this packet, the same thing happens, only in reverse. So it checks the message authentication code, decrypts the contents of the UDP datagram, checks the sequence number and writes the decrypted information to its own virtual network device.

If the virtual network device is a 'tun' device (a point-to-point tunnel), there is no problem for the kernel to accept a packet. However, if it is a 'tap' device (this is the only available type on FreeBSD), the destination MAC address must match that of the virtual network interface. If tinc is in it's default routing mode, ARP does not work, so the correct destination MAC can not be known by the sending host. Tinc solves this by letting the receiving end detect the MAC address of its own virtual network interface and overwriting the destination MAC address of the received packet.

In switch or hub modes ARP does work so the sender already knows the correct destination MAC address. In those modes every interface should have a unique MAC address, so make sure they are not the same. Because switch and hub modes rely on MAC addresses to function correctly, these modes cannot be used on the following operating systems which don't have a 'tap' style virtual network device: OpenBSD, NetBSD, Darwin and Solaris.

8.1.2 The meta-connection

Having only a UDP connection available is not enough. Though suitable for transmitting data, we want to be able to reliably send other information, such as routing and session key information to somebody.

TCP is a better alternative, because it already contains protection against information being lost, unlike UDP.

So we establish two connections. One for the encrypted VPN data, and one for other information, the meta-data. Hence, we call the second connection the meta-connection. We can now be sure that the meta-information doesn't get lost on the way to another computer.

Like with any communication, we must have a protocol, so that everybody knows what everything stands for, and how she should react. Because we have two connections, we also have two protocols. The protocol used for the UDP data is the "data-protocol," the other one is the "meta-protocol."

The reason we don't use TCP for both protocols is that UDP is much better for encapsulation, even while it is less reliable. The real problem is that when TCP would be used to encapsulate a TCP stream that's on the private network, for every packet sent there would be three ACKs sent instead of just one. Furthermore, if there would be a timeout, both TCP streams would sense the timeout, and both would start re-sending packets.

8.2 The meta-protocol

The meta protocol is used to tie all tinc daemons together, and exchange information about which tinc daemon serves which virtual subnet.

The meta protocol consists of requests that can be sent to the other side. Each request has a unique number and several parameters. All requests are represented in the standard ASCII character set. It is possible to use tools such as telnet or netcat to connect to a tinc daemon started with the –bypass-security option and to read and write requests by hand, provided that one understands the numeric codes sent.

The authentication scheme is described in Section 8.3 [Security], page 37. After a successful authentication, the server and the client will exchange all the information about other tinc daemons and subnets they know of, so that both sides (and all the other tinc daemons behind them) have their information synchronised.

```
message
------------------------------------------------------------------
ADD_EDGE node1 node2 21.32.43.54 655 222 0
          |     |        |       |   |  +-> options
          |     |        |       |   +----> weight
          |     |        |       +--------> UDP port of node2
          |     |        +----------------> real address of node2
          |     +-------------------------> name of destination node
          +-------------------------------> name of source node

ADD_SUBNET node 192.168.1.0/24
            |         |     +--> prefixlength
            |         +--------> network address
            +------------------> owner of this subnet
------------------------------------------------------------------
```

The ADD_EDGE messages are to inform other tinc daemons that a connection between two nodes exist. The address of the destination node is available so that VPN packets can be sent directly to that node.

The ADD_SUBNET messages inform other tinc daemons that certain subnets belong to certain nodes. tinc will use it to determine to which node a VPN packet has to be sent.

```
message
------------------------------------------------------------------
DEL_EDGE node1 node2
           |     +----> name of destination node
```

```
           +----------> name of source node

DEL_SUBNET node 192.168.1.0/24
             |         |     +--> prefixlength
             |         +--------> network address
             +-------------------> owner of this subnet
-----------------------------------------------------------------
```

In case a connection between two daemons is closed or broken, DEL_EDGE messages are sent to inform the other daemons of that fact. Each daemon will calculate a new route to the the daemons, or mark them unreachable if there isn't any.

```
message
-----------------------------------------------------------------
REQ_KEY origin destination
           |       +--> name of the tinc daemon it wants the key from
           +----------> name of the daemon that wants the key

ANS_KEY origin destination 4ae0b0a82d6e0078 91 64 4
           |       |       \_______________/ |  |  +--> MAC length
           |       |               |        |  +-----> digest algorithm
           |       |               |        +--------> cipher algorithm
           |       |               +--> 128 bits key
           |       +--> name of the daemon that wants the key
           +----------> name of the daemon that uses this key

KEY_CHANGED origin
              +--> daemon that has changed it's packet key
-----------------------------------------------------------------
```

The keys used to encrypt VPN packets are not sent out directly. This is because it would generate a lot of traffic on VPNs with many daemons, and chances are that not every tinc daemon will ever send a packet to every other daemon. Instead, if a daemon needs a key it sends a request for it via the meta connection of the nearest hop in the direction of the destination.

```
daemon  message
-----------------------------------------------------------------
origin  PING
dest.   PONG
-----------------------------------------------------------------
```

There is also a mechanism to check if hosts are still alive. Since network failures or a crash can cause a daemon to be killed without properly shutting down the TCP connection, this is necessary to keep an up to date connection list. PINGs are sent at regular intervals, except when there is also some other traffic. A little bit of salt (random data) is added with each PING and PONG message, to make sure that long sequences of PING/PONG messages without any other traffic won't result in known plaintext.

This basically covers what is sent over the meta connection by tinc.

8.3 Security

Tinc got its name from "TINC," short for *There Is No Cabal*; the alleged Cabal was/is an organisation that was said to keep an eye on the entire Internet. As this is exactly what you *don't* want, we named the tinc project after TINC.

But in order to be "immune" to eavesdropping, you'll have to encrypt your data. Because tinc is a *Secure* VPN (SVPN) daemon, it does exactly that: encrypt. However, encryption in itself does

not prevent an attacker from modifying the encrypted data. Therefore, tinc also authenticates the data. Finally, tinc uses sequence numbers (which themselves are also authenticated) to prevent an attacker from replaying valid packets.

Since version 1.1pre3, tinc has two protocols used to protect your data; the legacy protocol, and the new Simple Peer-to-Peer Security (SPTPS) protocol. The SPTPS protocol is designed to address some weaknesses in the legacy protocol. The new authentication protocol is used when two nodes connect to each other that both have the ExperimentalProtocol option set to yes, otherwise the legacy protocol will be used.

8.3.1 Legacy authentication protocol

```
daemon  message
------------------------------------------------------------------------
client  <attempts connection>

server  <accepts connection>

client  ID client 17.2
              |   |  +-> minor protocol version
              |   +----> major protocol version
              +--------> name of tinc daemon

server  ID server 17.2
              |   |  +-> minor protocol version
              |   +----> major protocol version
              +--------> name of tinc daemon

client  META_KEY 94 64 0 0 5f0823a93e35b69e...7086ec7866ce582b
                  |  |  | | \_________________________________/
                  |  |  | |                 +-> RSAKEYLEN bits totally random string S1
                  |  |  | |                     encrypted with server's public RSA key
                  |  |  | +-> compression level
                  |  |  +---> MAC length
                  |  +------> digest algorithm NID
                  +---------> cipher algorithm NID

server  META_KEY 94 64 0 0 6ab9c1640388f8f0...45d1a07f8a672630
                  |  |  | | \_________________________________/
                  |  |  | |                 +-> RSAKEYLEN bits totally random string S2
                  |  |  | |                     encrypted with client's public RSA key
                  |  |  | +-> compression level
                  |  |  +---> MAC length
                  |  +------> digest algorithm NID
                  +---------> cipher algorithm NID
------------------------------------------------------------------------
```

The protocol allows each side to specify encryption algorithms and parameters, but in practice they are always fixed, since older versions of tinc did not allow them to be different from the default values. The cipher is always Blowfish in OFB mode, the digest is SHA1, but the MAC length is zero and no compression is used.

From now on:

- the client will symmetrically encrypt outgoing traffic using S1
- the server will symmetrically encrypt outgoing traffic using S2

```
-------------------------------------------------------------------------
client  CHALLENGE da02add1817c1920989ba6ae2a49cecbda0
                  \_________________________________/
                                    +-> CHALLEN bits totally random string H1

server  CHALLENGE 57fb4b2ccd70d6bb35a64c142f47e61d57f
                  \_________________________________/
                                    +-> CHALLEN bits totally random string H2

client  CHAL_REPLY 816a86
                       +-> 160 bits SHA1 of H2

server  CHAL_REPLY 928ffe
                       +-> 160 bits SHA1 of H1

After the correct challenge replies are received, both ends have proved
their identity. Further information is exchanged.

client  ACK 655 123 0
            |   |  +-> options
            |   +----> estimated weight
            +--------> listening port of client

server  ACK 655 321 0
            |   |  +-> options
            |   +----> estimated weight
            +--------> listening port of server
-------------------------------------------------------------------------
```

This legacy authentication protocol has several weaknesses, pointed out by security export Peter Gutmann. First, data is encrypted with RSA without padding. Padding schemes are designed to prevent attacks when the size of the plaintext is not equal to the size of the RSA key. Tinc always encrypts random nonces that have the same size as the RSA key, so we do not believe this leads to a break of the security. There might be timing or other side-channel attacks against RSA encryption and decryption, tinc does not employ any protection against those. Furthermore, both sides send identical messages to each other, there is no distinction between server and client, which could make a MITM attack easier. However, no exploit is known in which a third party who is not already trusted by other nodes in the VPN could gain access. Finally, the RSA keys are used to directly encrypt the session keys, which means that if the RSA keys are compromised, it is possible to decrypt all previous VPN traffic. In other words, the legacy protocol does not provide perfect forward secrecy.

8.3.2 Simple Peer-to-Peer Security

The SPTPS protocol is designed to address the weaknesses in the legacy protocol. SPTPS is based on TLS 1.2, but has been simplified: there is no support for exchanging public keys, and there is no cipher suite negotiation. Instead, SPTPS always uses a very strong cipher suite: peers authenticate each other using 521 bits ECC keys, Diffie-Hellman using ephemeral 521 bits ECC keys is used to provide perfect forward secrecy (PFS), AES-256-CTR is used for encryption, and HMAC-SHA-256 for message authentication.

Similar to TLS, messages are split up in records. A complete logical record contains the following information:

- uint32_t seqno (network byte order)

- uint16_t length (network byte order)
- uint8_t type
- opaque data[length]
- opaque hmac[HMAC_SIZE] (HMAC over all preceding fields)

Depending on whether SPTPS records are sent via TCP or UDP, either the seqno or the length field is omitted on the wire (but they are still included in the calculation of the HMAC); for TCP packets are guaranteed to arrive in-order so we can infer the seqno, but packets can be split or merged, so we still need the length field to determine the boundaries between records; for UDP packets we know that there is exactly one record per packet, and we know the length of a packet, but packets can be dropped, duplicated and/or reordered, so we need to include the seqno.

The type field is used to distinguish between application records or handshake records. Types 0 to 127 are application records, type 128 is a handshake record, and types 129 to 255 are reserved.

Before the initial handshake, no fields are encrypted, and the HMAC field is not present. After the authentication handshake, the length (if present), type and data fields are encrypted, and the HMAC field is present. For UDP packets, the seqno field is not encrypted, as it is used to determine the value of the counter used for encryption.

The authentication consists of an exchange of Key EXchange, SIGnature and ACKnowledge messages, transmitted using type 128 records.

Overview:

```
Initiator   Responder
--------------------
KEX ->
            <- KEX
SIG ->
            <- SIG

...encrypt and HMAC using session keys from now on...

App ->
            <- App
...
            ...

...key renegotiation starts here...

KEX ->
            <- KEX
SIG ->
            <- SIG
ACK ->
            <- ACK

...encrypt and HMAC using new session keys from now on...

App ->
            <- App
...
            ...
--------------------
```

Note that the responder does not need to wait before it receives the first KEX message, it can immediately send its own once it has accepted an incoming connection.

Key EXchange message:

- uint8_t kex_version (always 0 in this version of SPTPS)
- opaque nonce[32] (random number)
- opaque ecdh_key[ECDH_SIZE]

SIGnature message:

- opaque ecdsa_signature[ECDSA_SIZE]

ACKnowledge message:

- empty (only sent after key renegotiation)

Remarks:

- At the start, both peers generate a random nonce and an Elliptic Curve public key and send it to the other in the KEX message.
- After receiving the other's KEX message, both KEX messages are concatenated (see below), and the result is signed using ECDSA. The result is sent to the other.
- After receiving the other's SIG message, the signature is verified. If it is correct, the shared secret is calculated from the public keys exchanged in the KEX message using the Elliptic Curve Diffie-Helman algorithm.
- The shared secret key is expanded using a PRF. Both nonces and the application specific label are also used as input for the PRF.
- An ACK message is sent only when doing key renegotiation, and is sent using the old encryption keys.
- The expanded key is used to key the encryption and HMAC algorithms.

The signature is calculated over this string:

- uint8_t initiator (0 = local peer, 1 = remote peer is initiator)
- opaque remote_kex_message[1 + 32 + ECDH_SIZE]
- opaque local_kex_message[1 + 32 + ECDH_SIZE]
- opaque label[label_length]

The PRF is calculated as follows:

- A HMAC using SHA512 is used, the shared secret is used as the key.
- For each block of 64 bytes, a HMAC is calculated. For block n: hmac[n] = HMAC_SHA512(hmac[n - 1] + seed)
- For the first block (n = 1), hmac[0] is given by HMAC_SHA512(zeroes + seed), where zeroes is a block of 64 zero bytes.

The seed is as follows:

- const char[13] "key expansion"
- opaque responder_nonce[32]
- opaque initiator_nonce[32]
- opaque label[label_length]

The expanded key is used as follows:

- opaque responder_cipher_key[CIPHER_KEYSIZE]
- opaque responder_digest_key[DIGEST_KEYSIZE]
- opaque initiator_cipher_key[CIPHER_KEYSIZE]

- opaque initiator_digest_key[DIGEST_KEYSIZE]

Where initiator_cipher_key is the key used by session initiator to encrypt messages sent to the responder.

When using 256 bits Ed25519 keys, the AES-256-CTR cipher and HMAC-SHA-256 digest algorithm, the sizes are as follows:

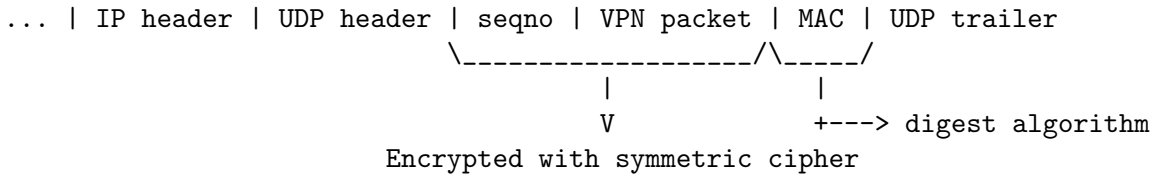
```
ECDH_SIZE:       32 (= 256/8)
ECDSA_SIZE:      64 (= 2 * 256/8)
CIPHER_KEYSIZE:  48 (= 256/8 + 128/8)
DIGEST_KEYSIZE:  32 (= 256/8)
```

Note that the cipher key also includes the initial value for the counter.

8.3.3 Encryption of network packets

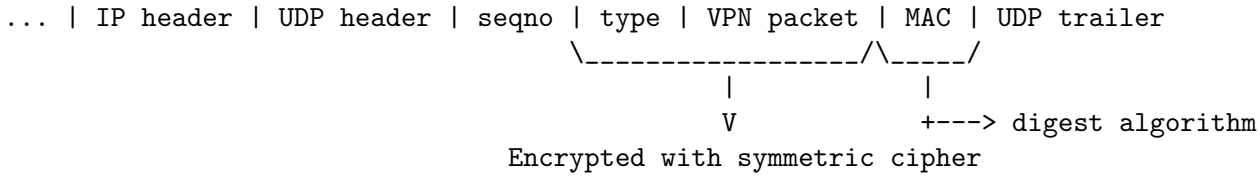
A data packet can only be sent if the encryption key is known to both parties, and the connection is activated. If the encryption key is not known, a request is sent to the destination using the meta connection to retrieve it.

The UDP packets can be either encrypted with the legacy protocol or with SPTPS. In case of the legacy protocol, the UDP packet containing the network packet from the VPN has the following layout:



So, the entire VPN packet is encrypted using a symmetric cipher, including a 32 bits sequence number that is added in front of the actual VPN packet, to act as a unique IV for each packet and to prevent replay attacks. A message authentication code is added to the UDP packet to prevent alteration of packets. Tinc by default encrypts network packets using Blowfish with 128 bit keys in CBC mode and uses 4 byte long message authentication codes to make sure eavesdroppers cannot get and cannot change any information at all from the packets they can intercept. The encryption algorithm and message authentication algorithm can be changed in the configuration. The length of the message authentication codes is also adjustable. The length of the key for the encryption algorithm is always the default length used by LibreSSL/OpenSSL.

The SPTPS protocol is described in Section 8.3.2 [Simple Peer-to-Peer Security], page 39. For comparison, this is how SPTPS UDP packets look:



The difference is that the seqno is not encrypted, since the encryption cipher is used in CTR mode, and therefore the seqno must be known before the packet can be decrypted. Furthermore, the MAC is never truncated. The SPTPS protocol always uses the AES-256-CTR cipher and HMAC-SHA-256 digest, this cannot be changed.

8.3.4 Security issues

In August 2000, we discovered the existence of a security hole in all versions of tinc up to and including 1.0pre2. This had to do with the way we exchanged keys. Since then, we have been working on a new authentication scheme to make tinc as secure as possible. The current version uses the LibreSSL or OpenSSL library and uses strong authentication with RSA keys.

On the 29th of December 2001, Jerome Etienne posted a security analysis of tinc 1.0pre4. Due to a lack of sequence numbers and a message authentication code for each packet, an attacker could possibly disrupt certain network services or launch a denial of service attack by replaying intercepted packets. The current version adds sequence numbers and message authentication codes to prevent such attacks.

On the 15th of September 2003, Peter Gutmann posted a security analysis of tinc 1.0.1. He argues that the 32 bit sequence number used by tinc is not a good IV, that tinc's default length of 4 bytes for the MAC is too short, and he doesn't like tinc's use of RSA during authentication. We do not know of a security hole in the legacy protocol of tinc, but it is not as strong as TLS or IPsec.

This version of tinc comes with an improved protocol, called Simple Peer-to-Peer Security, which aims to be as strong as TLS with one of the strongest cipher suites.

Cryptography is a hard thing to get right. We cannot make any guarantees. Time, review and feedback are the only things that can prove the security of any cryptographic product. If you wish to review tinc or give us feedback, you are stronly encouraged to do so.

9 Platform specific information

9.1 Interface configuration

When configuring an interface, one normally assigns it an address and a netmask. The address uniquely identifies the host on the network attached to the interface. The netmask, combined with the address, forms a subnet. It is used to add a route to the routing table instructing the kernel to send all packets which fall into that subnet to that interface. Because all packets for the entire VPN should go to the virtual network interface used by tinc, the netmask should be such that it encompasses the entire VPN.

For IPv4 addresses:

Linux	`ifconfig` *interface address* `netmask` *netmask*
Linux iproute2	`ip addr add` *address/prefixlength* `dev` *interface*
FreeBSD	`ifconfig` *interface address* `netmask` *netmask*
OpenBSD	`ifconfig` *interface address* `netmask` *netmask*
NetBSD	`ifconfig` *interface address* `netmask` *netmask*
Solaris	`ifconfig` *interface address* `netmask` *netmask*
Darwin (MacOS/X)	`ifconfig` *interface address* `netmask` *netmask*
Windows	`netsh interface ip set address` *interface* `static` *address netmask*

For IPv6 addresses:

Linux	`ifconfig` *interface* `add` *address/prefixlength*
FreeBSD	`ifconfig` *interface* `inet6` *address* `prefixlen` *prefixlength*
OpenBSD	`ifconfig` *interface* `inet6` *address* `prefixlen` *prefixlength*
NetBSD	`ifconfig` *interface* `inet6` *address* `prefixlen` *prefixlength*
Solaris	`ifconfig` *interface* `inet6 plumb up` `ifconfig` *interface* `inet6 addif` *address address*
Darwin (MacOS/X)	`ifconfig` *interface* `inet6` *address* `prefixlen` *prefixlength*
Windows	`netsh interface ipv6 add address` *interface* `static` *address/prefixlength*

On some platforms, when running tinc in switch mode, the VPN interface must be set to tap mode with an ifconfig command:

OpenBSD	`ifconfig` *interface* `link0`

On Linux, it is possible to create a persistent tun/tap interface which will continue to exist even if tinc quit, although this is normally not required. It can be useful to set up a tun/tap interface owned by a non-root user, so tinc can be started without needing any root privileges at all.

Linux	`ip tuntap add dev` *interface* `mode` *tun|tap* `user` *username*

9.2 Routes

In some cases it might be necessary to add more routes to the virtual network interface. There are two ways to indicate which interface a packet should go to, one is to use the name of the interface itself, another way is to specify the (local) address that is assigned to that interface (*local_address*). The former way is unambiguous and therefore preferable, but not all platforms support this.

Adding routes to IPv4 subnets:

Linux	`route add -net` *network_address* `netmask` *netmask interface*
Linux iproute2	`ip route add` *network_address/prefixlength* `dev` *interface*
FreeBSD	`route add` *network_address/prefixlength local_address*
OpenBSD	`route add` *network_address/prefixlength local_address*
NetBSD	`route add` *network_address/prefixlength local_address*
Solaris	`route add` *network_address/prefixlength local_address* `-interface`

Darwin (MacOS/X)	`route add` *network_address/prefixlength local_address*
Windows	`netsh routing ip add persistentroute` *network_address netmask interface local_address*

Adding routes to IPv6 subnets:

Linux	`route add -A inet6` *network_address/prefixlength interface*
Linux iproute2	`ip route add` *network_address/prefixlength* `dev` *interface*
FreeBSD	`route add -inet6` *network_address/prefixlength local_address*
OpenBSD	`route add -inet6` *network_address local_address* `-prefixlen` *prefixlength*
NetBSD	`route add -inet6` *network_address local_address* `-prefixlen` *prefixlength*
Solaris	`route add -inet6` *network_address/prefixlength local_address* `-interface`
Darwin (MacOS/X)	?
Windows	`netsh interface ipv6 add route` *network address/prefixlength interface*

10 About us

10.1 Contact information

Tinc's website is at https://www.tinc-vpn.org/, this server is located in the Netherlands.

We have an IRC channel on the FreeNode and OFTC IRC networks. Connect to irc.freenode.net or irc.oftc.net and join channel #tinc.

10.2 Authors

Ivo Timmermans (zarq)
Guus Sliepen (guus) (guus@tinc-vpn.org)

We have received a lot of valuable input from users. With their help, tinc has become the flexible and robust tool that it is today. We have composed a list of contributions, in the file called `THANKS` in the source distribution.

Concept Index

A

ACK ... 38
add ... 29
ADD_EDGE ... 36
ADD_SUBNET ... 36
Address ... 17
AddressFamily ... 10
ANS_KEY ... 37
AutoConnect ... 10

B

binary package ... 7
BindToAddress ... 10
BindToInterface ... 11
Broadcast ... 11
BroadcastSubnet ... 11

C

Cabal ... 37
CHAL_REPLY ... 38
CHALLENGE ... 38
CIDR notation ... 18
Cipher ... 17
ClampMSS ... 17
client ... 9
command line ... 25
command line interface ... 29
Compression ... 17
connection ... 35
ConnectTo ... 11

D

daemon ... 25
data-protocol ... 36
debug ... 31
debug level ... 25
debug levels ... 26
DecrementTTL ... 11
del ... 29
DEL_EDGE ... 36
DEL_SUBNET ... 36
Device ... 11
device files ... 7
DeviceStandby ... 12
DeviceType ... 12
DEVICE ... 20
Digest ... 17
DirectOnly ... 13
disconnect ... 31
dummy ... 12
dump ... 30

E

Ed25519PrivateKeyFile ... 13
edit ... 29
encapsulating ... 35
encryption ... 42
environment variables ... 19
example ... 22
exchange ... 30
exchange-all ... 30
exec ... 15
ExperimentalProtocol ... 13
export ... 30
export-all ... 30

F

Forwarding ... 13
frame type ... 35
fsck ... 31

G

generate-ed25519-keys ... 30
generate-keys ... 30
generate-rsa-keys ... 30
get ... 29
graph ... 31

H

Hostnames ... 13
http ... 15
hub ... 14

I

ID ... 38
Ifconfig ... 34
import ... 30
IndirectData ... 17
info ... 31
init ... 29
Interface ... 13
INTERFACE ... 20
INVITATION_FILE ... 20
INVITATION_URL ... 20
invite ... 30
IRC ... 47

J

join ... 30

K

KEY_CHANGED ... 37
KeyExpire ... 14

L

legacy authentication protocol ... 38
libcurses ... 5
libraries ... 4
libreadline ... 5

LibreSSL ... 4
license ... 4
ListenAddress ... 14
LocalDiscovery ... 14
LocalDiscoveryAddress ... 14
log ... 31
lzo ... 5

M

MACExpire ... 14
MACLength ... 17
MaxConnectionBurst ... 14
meta-protocol ... 36
META_KEY ... 38
Mode ... 14
MTUInfoInterval ... 18
multicast ... 12
multiple networks ... 9

N

Name ... 15
NAME ... 19
netmask ... 22
netname ... 9
NETNAME ... 19, 29
network ... 31
Network Administrators Guide ... 9
NODE ... 20

O

OpenSSL ... 4
options ... 25

P

pcap ... 31
PEM format ... 18
pid ... 30
PingInterval ... 15
PingTimeout ... 15
PING ... 37
platforms ... 2
PMTU ... 17
PMTUDiscovery ... 18
PONG ... 37
Port ... 18
port numbers ... 8
PriorityInheritance ... 15
private ... 1
PrivateKey ... 15
PrivateKeyFile ... 15
ProcessPriority ... 15
Proxy ... 15
PublicKey ... 18
PublicKeyFile ... 18
purge ... 31

R

raw_socket ... 12
release ... 2
reload ... 30
REMOTEADDRESS ... 20
REMOTEPORT ... 20
ReplayWindow ... 16
REQ_KEY ... 37
requirements ... 4
restart ... 30
retry ... 31
Route ... 34
router ... 14
runtime options ... 25

S

scalability ... 1
scripts ... 19
server ... 9
set ... 29
shell ... 29
sign ... 31
signals ... 26
socks4 ... 15
socks5 ... 15
SPTPS ... 39
start ... 30
stop ... 30
StrictSubnets ... 16
Subnet ... 18
SUBNET ... 20
SVPN ... 37
switch ... 14

T

TCP ... 35
TCPonly ... 18
tinc ... 1
tinc-down ... 19
tinc-up ... 19, 22
tincd ... 1
TINC ... 37
top ... 31, 32
traditional VPNs ... 1
tunifhead ... 12
TunnelServer ... 16
tunnohead ... 12

U

UDP ... 35, 42
UDPDiscoveryInterval ... 16
UDPDiscoveryKeepaliveInterval ... 16
UDPDiscoveryTimeout ... 16
UDPDiscovey ... 16
UDPInfoInterval ... 16
UDPRcvBuf ... 16
UDPSndBuf ... 16
UML ... 12
Universal tun/tap ... 3
UPnP ... 16
UPnPDiscoverWait ... 17
UPnPRefreshPeriod ... 17
utun ... 13

V

VDE 12
verify 31
virtual 1
virtual network device 35
vpnd 1
VPN 1

W

website 47
Weight 18
WEIGHT 20

Z

zlib 5

Table of Contents

1 Introduction 1
1.1 Virtual Private Networks 1
1.2 tinc 1
1.3 Supported platforms 2

2 Preparations 3
2.1 Configuring the kernel 3
2.1.1 Configuration of Linux kernels 3
2.1.2 Configuration of FreeBSD kernels 3
2.1.3 Configuration of OpenBSD kernels 3
2.1.4 Configuration of NetBSD kernels 3
2.1.5 Configuration of Solaris kernels 3
2.1.6 Configuration of Darwin (MacOS/X) kernels 3
2.1.7 Configuration of Windows 4
2.2 Libraries 4
2.2.1 LibreSSL/OpenSSL 4
2.2.2 zlib 5
2.2.3 lzo 5
2.2.4 libcurses 5
2.2.5 libreadline 5

3 Installation 7
3.1 Building and installing tinc 7
3.1.1 Darwin (MacOS/X) build environment 7
3.1.2 Cygwin (Windows) build environment 7
3.1.3 MinGW (Windows) build environment 7
3.2 System files 7
3.2.1 Device files 7
3.2.2 Other files 8

4 Configuration 9
4.1 Configuration introduction 9
4.2 Multiple networks 9
4.3 How connections work 9
4.4 Configuration files 10
4.4.1 Main configuration variables 10
4.4.2 Host configuration variables 17
4.4.3 Scripts 19
4.4.4 How to configure 20
4.5 Network interfaces 21
4.6 Example configuration 22

5 Running tinc ... 25
5.1 Runtime options ... 25
5.2 Signals ... 26
5.3 Debug levels ... 26
5.4 Solving problems ... 26
5.5 Error messages ... 27
5.6 Sending bug reports ... 28

6 Controlling tinc ... 29
6.1 tinc runtime options ... 29
6.2 tinc environment variables ... 29
6.3 tinc commands ... 29
6.4 tinc examples ... 32
6.5 tinc top ... 32

7 Invitations ... 33
7.1 How invitations work ... 33
7.2 Invitation file format ... 33
7.3 Writing an invitation-created script ... 34

8 Technical information ... 35
8.1 The connection ... 35
8.1.1 The UDP tunnel ... 35
8.1.2 The meta-connection ... 35
8.2 The meta-protocol ... 36
8.3 Security ... 37
8.3.1 Legacy authentication protocol ... 38
8.3.2 Simple Peer-to-Peer Security ... 39
8.3.3 Encryption of network packets ... 42
8.3.4 Security issues ... 42

9 Platform specific information ... 45
9.1 Interface configuration ... 45
9.2 Routes ... 45

10 About us ... 47
10.1 Contact information ... 47
10.2 Authors ... 47

Concept Index ... 49