tinc Manual

Setting up a Virtual Private Network with tinc

Ivo Timmermans and Guus Sliepen

This is the info manual for tinc version 1.0.8, a Virtual Private Network daemon.

$Id: tinc.texi 1467 2006-11-11 20:37:58Z guus $

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 singe 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: http://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 2.1.60 up to 2.4.0

For kernels up to 2.4.0, you need a kernel that supports the ethertap device. Most distributions come with kernels that already support this. If not, 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
Networking options
[*] Kernel/User netlink socket
<M> Netlink device emulation
Network device support
<M> Ethertap network tap
```

If you want to run more than one instance of tinc or other programs that use the ethertap, you have to compile the ethertap driver as a module, otherwise you can also choose to compile it directly into the kernel.

If you decide to build any of these as dynamic kernel modules, it's a good idea to add these lines to '`/etc/modules.conf`':

```
alias char-major-36 netlink_dev
alias tap0 ethertap
options tap0 -o tap0 unit=0
alias tap1 ethertap
options tap1 -o tap1 unit=1
...
alias tapN ethertap
options tapN -o tapN unit=N
```

Add as much alias/options lines as necessary.

2.1.2 Configuration of Linux kernels 2.4.0 and higher

For kernels 2.4.0 and higher, 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 have an early 2.4 kernel, you can choose both the tun/tap driver and the 'Ethertap network tap' device. This latter is marked obsolete, and chances are that it won't even function correctly anymore. Make sure you select the universal tun/tap driver.

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
```

2.1.3 Configuration of FreeBSD kernels

For FreeBSD version 4.1 and higher, tun and tap drivers are included in the default kernel configuration. Using tap devices is recommended.

2.1.4 Configuration of OpenBSD kernels

For OpenBSD version 2.9 and higher, the tun driver is included in the default kernel configuration. There is also a kernel patch from http://diehard.n-r-g.com/stuff/openbsd/ which adds a tap device to OpenBSD. This should work with tinc.

2.1.5 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.

2.1.6 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 http://www.monkey.org/~dugsong/fragroute/. If the 'net/if_tun.h' header file is missing, install it from the source package.

2.1.7 Configuration of Darwin (MacOS/X) kernels

Tinc on Darwin relies on a tunnel driver for its data acquisition from the kernel. Tinc supports either the driver from http://www-user.rhrk.uni-kl.de/~nissler/tuntap/, which supports both tun and tap style devices, and also the driver from from http://chrisp.de/en/projects/tunnel.html. The former driver is recommended. The tunnel driver must be loaded before starting tinc with the following command:

```
kmodload tunnel
```

2.1.8 Configuration of Windows

You will need to install the latest TAP-Win32 driver from OpenVPN. You can download it from http://openvpn.sourceforge.net. 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 OpenSSL, zlib and lzo 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 OpenSSL

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

If this library is not installed, you wil get an error when configuring tinc for build. Support for running tinc without having OpenSSL 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 you have to install OpenSSL manually, you can get the source code from http://www.openssl.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 OpenSSL libraries from source, it may be necessary to let configure know where they are, by passing configure one of the –with-openssl-* parameters.

```
--with-openssl=DIR      OpenSSL library and headers prefix
--with-openssl-include=DIR OpenSSL headers directory
                        (Default is OPENSSL_DIR/include)
--with-openssl-lib=DIR  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 http://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 (http://www.tinc-vpn.org/) to link the LZO library with the OpenSSL library (http://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 configuring tinc for build. Support for running tinc without having zlib 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 you have to install zlib manually, you can get the source code from http://www.gzip.org/zlib/. 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 configuring tinc for build. Support for running tinc without having lzo 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 you have to install lzo manually, you can get the source code from http://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).

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, which has the checksums of these files listed; you may wish to check these with md5sum before continuing.

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 the MacOS/X Developer Tools from `http://developer.apple.com/tools/macosxtools.html` and a recent version of Fink from `http://fink.sourceforge.net/`.

After installation use fink to download and install the following packages: autoconf25, automake, dlcompat, m4, openssl, zlib and lzo.

3.1.2 Cygwin (Windows) build environment

If Cygwin hasn't already been installed, install it directly from `http://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`.

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

First, you'll need the special device file(s) that form the interface between the kernel and the daemon.

The permissions for these files have to be such that only the super user may read/write to this file. You'd want this, because otherwise eavesdropping would become a bit too easy. This does, however, imply that you'd have to run tincd as root.

If you use Linux and have a kernel version prior to 2.4.0, you have to make the ethertap devices:

```
mknod -m 600 /dev/tap0 c 36 16
mknod -m 600 /dev/tap1 c 36 17
...
mknod -m 600 /dev/tapN c 36 N+16
```

There is a maximum of 16 ethertap devices.

If you use the universal tun/tap driver, you have to create the following device file (unless it already exist):

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

If you use Linux, and you run the new 2.4 kernel using the devfs filesystem, then the tun/tap device will probably be automatically generated as '`/dev/net/tun`'.

Unlike the ethertap device, you do not need multiple device files if you are planning to run multiple tinc daemons.

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, generate the configuration files (‘`tinc.conf`’, your host configuration file, ‘`tinc-up`’ and perhaps ‘`tinc-down`’). Then generate the keypairs. Finally, distribute the host configuration files. 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 sites 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 tincd with the -n argument, which will assign a netname to this daemon.

The effect of this is that the daemon will set its configuration root to ‘`/etc/tinc/`*`netname`*`/`’, where *netname* is your argument to the -n option. You'll notice that it appears in syslog as ‘`tinc.`*`netname`*’.

However, it is not strictly necessary that you call tinc with the -n option. In this case, the network name would just be empty, and it will be used as such. tinc now looks for files in ‘`/etc/tinc/`’, instead of ‘`/etc/tinc/`*`netname`*`/`’; the configuration file should be ‘`/etc/tinc/tinc.conf`’, and the host configuration files are now expected to be in ‘`/etc/tinc/hosts/`’.

But it is highly recommended that you use this feature of tinc, because it will be so much clearer whom your daemon talks to. Hence, we will assume that you use it.

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.

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/`'.

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.

In this section 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.

BindToAddress = <*address*> [experimental]
: If your computer has more than one IPv4 or IPv6 address, tinc will by default listen on all of them for incoming connections. It is possible to bind only to a single address with this variable.

  This option may not work on all platforms.

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.

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, tinc won't try to connect to other daemons at all, and will instead just listen for incoming connections.

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 8.

GraphDumpFile = <*filename*> [experimental]
: If this option is present, tinc will dump the current network graph to the file *filename* every minute, unless there were no changes to the graph. The file is in a format that can be read by graphviz tools. If *filename* starts with a pipe symbol `|`, then the rest of the filename is interpreted as a shell command that is executed, the graph is then sent to stdin.

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.

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.

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 unicast 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".

Name = <*name*> [required]
: This is a symbolic name for this connection. It can be anything

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 '`tincd --generate-keys`'. It must be a full path, not a relative directory.

  Note that there must be exactly one of PrivateKey or PrivateKeyFile specified in the configuration file.

TunnelServer = <yes|no> (no) [experimental]
: When this option is enabled tinc will no longer forward information between other tinc daemons, and will only allow nodes and subnets on the VPN which are present in the '`/etc/tinc/`*`netname`*`/hosts/`' directory.

4.4.2 Host configuration variables

Address = <*IP address*|*hostname*> [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.

Cipher = <*cipher*> (blowfish)
: The symmetric cipher algorithm used to encrypt UDP packets. Any cipher supported by OpenSSL is recognized. Furthermore, specifying "none" will turn off packet encryption. It is best to use only those ciphers which support CBC mode.

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. Any digest supported by OpenSSL is recognized. Furthermore, specifying "none" will turn off packet authentication.

IndirectData = <yes|no> (no)
: This option specifies whether other tinc daemons besides the one you specified with ConnectTo can make a direct connection to you. This is especially useful if you are behind a firewall and it is impossible to make a connection from the outside to your tinc daemon. Otherwise, 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. Can be anything from 0 up to the length of the digest produced by the digest algorithm.

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 '`tincd --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*]>
: 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. Shorthand notations are not supported. 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 fec0:0:0:1:0:0:0:0/64. 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

TCPonly = <yes|no> (no) [experimental]
: 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.

4.4.3 Scripts

Apart from reading the server and host configuration files, tinc can also run scripts at certain moments. Under Windows (not Cygwin), the scripts should have the extension .bat.

'`/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.

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`' 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.

4.4.4 How to configure

Step 1. Creating the main configuration file

The main configuration file will be called '`/etc/tinc/`*`netname`*`/tinc.conf`'. Adapt the following example to create a basic configuration file:

```
Name = yourname
Device = '/dev/tap0'
```

Then, if you know to which other tinc daemon(s) yours is going to connect, add 'ConnectTo' values.

Step 2. Creating your host configuration file

If you added a line containing 'Name = yourname' in the main configuarion file, you will need to create a host configuration file '`/etc/tinc/`*`netname`*`/hosts/yourname`'. Adapt the following example to create a host configuration file:

```
Address = your.real.hostname.org
Subnet = 192.168.1.0/24
```

You can also use an IP address instead of a hostname. The 'Subnet' specifies the address range that is local for *your part of the VPN only*. If you have multiple address ranges you can specify more than one 'Subnet'. You might also need to add a 'Port' if you want your tinc daemon to run on a different port number than the default (655).

4.5 Generating keypairs

Now that you have already created the main configuration file and your host configuration file, you can easily create a public/private keypair by entering the following command:

```
tincd -n netname -K
```

Tinc will generate a public and a private key and ask you where to put them. Just press enter to accept the defaults.

4.6 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.

An example ‘`tinc-up`’ script:

```
#!/bin/sh
ifconfig $INTERFACE 192.168.1.1 netmask 255.255.0.0
```

This script gives the interface an IP address and a netmask. The kernel will also automatically add a 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 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 7 [Platform specific information], page 35, but it is best to consult the manpages of those utilities on your platform.

4.7 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.

For Branch A

BranchA would be configured like this:

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

```
# 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
Device = /dev/tap0
```

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 tap0 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 give both real internal network interfaces and tap 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':

```
# 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 tap0 device. Also, ConnectTo is given so that no-one can connect to this node.

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':

```
# 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
Device = /dev/tap1
```

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':

```
# 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
Device = /dev/net/tun
```

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. Also note that since D uses the tun/tap driver, the network interface will not be called 'tun' or 'tap0' or something like that, but will have the same name as netname.

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 generated a public/private keypair with the following command:

```
tincd -n company -K
```

The private key is stored in '`/etc/tinc/company/rsa_key.priv`', the public key is put into the host configuration file in the '`/etc/tinc/company/hosts/`' directory. During key generation, tinc automatically guesses the right filenames based on the -n option and the Name directive in the '`tinc.conf`' file (if it is available).

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:

```
tincd -n netname
```

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.

'`-k, --kill[=signal]`'
: Attempt to kill a running tincd (optionally with the specified *signal* instead of SIGTERM) and exit. Use it in conjunction with the -n option to make sure you kill the right tinc daemon. Under native Windows the optional argument is ignored, the service will always be stopped and removed.

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

'`-K, --generate-keys[=bits]`'
: Generate public/private keypair of *bits* length. If *bits* is not specified, 1024 is the default. 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 in combination with -K). After that, tinc will quit.

'`-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.

'`--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`'.

'`--pidfile=file`'
: Write PID to *file* instead of '`/var/run/tinc.netname.pid`'.

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

'`--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.

'`INT`' Temporarily increases debug level to 5. Send this signal again to revert to the original level.

'`USR1`' Dumps the connection list to syslog.

'`USR2`' Dumps virtual network device statistics, all known nodes, edges and subnets to syslog.

'`WINCH`' Purges all information remembered about unreachable nodes.

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 8.1 [Contact information], page 37. 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 Technical information

6.1 The connection

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

6.1.1 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.

6.1.2 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.

6.2 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 6.3.1 [Authentication protocol], page 30. 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.

6.3 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. Tinc by default uses blowfish encryption with 128 bit keys in CBC mode, 32 bit sequence numbers and 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 OpenSSL.

6.3.1 Authentication protocol

A new scheme for authentication in tinc has been devised, which offers some improvements over the protocol used in 1.0pre2 and 1.0pre3. Explanation is below.

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

server  <accepts connection>

client  ID client 12
              |   +---> version
              +-------> name of tinc daemon

server  ID server 12
              |   +---> version
              +-------> name of tinc daemon

client  META_KEY 5f0823a93e35b69e...7086ec7866ce582b
                  \_________________________________/
                                    +-> RSAKEYLEN bits totally random string S1,
                                        encrypted with server's public RSA key
```

```
server  META_KEY 6ab9c1640388f8f0...45d1a07f8a672630
                 \_________________________________/
                                   +-> RSAKEYLEN bits totally random string S2,
                                       encrypted with client's public RSA key

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 new scheme has several improvements, both in efficiency and security.

First of all, the server sends exactly the same kind of messages over the wire as the client. The previous versions of tinc first authenticated the client, and then the server. This scheme even allows both sides to send their messages simultaneously, there is no need to wait for the other to send something first. This means that any calculations that need to be done upon sending or receiving a message can also be done in parallel. This is especially important when doing RSA encryption/decryption. Given that these calculations are the main part of the CPU time spent for the authentication, speed is improved by a factor 2.

Second, only one RSA encrypted message is sent instead of two. This reduces the amount of information attackers can see (and thus use for a cryptographic attack). It also improves speed by a factor two, making the total speedup a factor 4.

Third, and most important: The symmetric cipher keys are exchanged first, the challenge is done afterwards. In the previous authentication scheme, because a man-in-the-middle could pass the challenge/chal_reply phase (by just copying the messages between the two real tinc daemons), but no information was exchanged that was really needed to read the rest of the messages, the challenge/chal_reply phase was of no real use. The man-in-the-middle was only stopped by the fact that only after the ACK messages were encrypted with the symmetric cipher. Potentially, it could even send it's own symmetric key to the server (if it knew the server's public key) and read some of the metadata the server would send it (it was impossible for the mitm to read actual network packets though). The new scheme however prevents this.

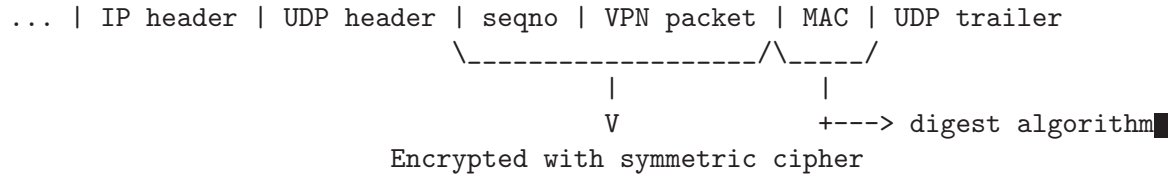
This new scheme makes sure that first of all, symmetric keys are exchanged. The rest of the messages are then encrypted with the symmetric cipher. Then, each side can only read received messages if they have their private key. The challenge is there to let the other side know that the private key is really known, because a challenge reply can only be sent back if the challenge is decrypted correctly, and that can only be done with knowledge of the private key.

Fourth: the first thing that is sent via the symmetric cipher encrypted connection is a totally random string, so that there is no known plaintext (for an attacker) in the beginning of the encrypted stream.

6.3.2 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 packet is stored in a queue while waiting for the key to arrive.

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. By default the first 4 bytes of the digest are used for this, but this can be changed using the MACLength configuration variable.

6.3.3 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 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 this version of tinc, but tinc's security is not as strong as TLS or IPsec. We will address these issues in tinc 2.0.

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.

7 Platform specific information

7.1 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*

7.2 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*

8 About us

8.1 Contact information

Tinc's website is at http://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.

8.2 Authors

Ivo Timmermans (zarq) (ivo@tinc-vpn.org)
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

P

R

S

T

U

V

W

Z

Table of Contents