Beginning OpenVPN 2.0.9

Build and integrate Virtual Private Networks using OpenVPN

Markus Feilner   Norbert Graf

Beginning OpenVPN 2.0.9

Build and integrate Virtual Private Networks using OpenVPN

Markus Feilner

Norbert Graf

Beginning OpenVPN 2.0.9

Credits

Author
Markus Feilner

Co-author
Norbert Graf

Reviewers
Chris Buechler
Ralf Hildebrandt

Acquisition Editor
Louay Fatoohi

Development Editor
Swapna Verlekar

Technical Editor
Akash Johari

Copy Editor
Leonard D'silva

Indexer
Hemangini Bari

Editorial Team Leader
Akshara Aware

Project Team Leader
Priya Mukherji

Project Coordinator
Zainab Bagasrawala

Proofreaders
Kevin McGowan
Chris Smith

Graphics
Nilesh R. Mohite

Production Coordinator
Dolly Dasilva

Cover Work
Dolly Dasilva

About the Author

Markus Feilner is a Linux professional from Regensburg, Germany and has been working with open source software since the mid 1990s. His first contact with Unix was with a SUN cluster and with SPARC workstations at Regensburg University during his studies of geography, computer science, and GIS. Since the year 2000, he has published several documents used in Linux training all over Germany. In 2001, he founded his own Linux consulting and training company, Feilner IT (`http://www.feilner-it.net`). Here, and as trainer, consultant, and systems engineer at Millenux, Munich, he focused on groupware, collaboration, and virtualization with Linux-based systems and networks.

He is working as Stellvertretender Chefredakteur at German Linux-Magazine, where he writes about open source software for both printed and online magazines, including the *Linux Technical Review* and the *Linux Magazine International* (`http://www.linux-magazine.com`). He regularly gives speeches and lectures at conferences in Germany. Security and VPN have never left his focus in his publications and articles. Together with Packt, he published *OpenVPN: Building and Integrating Virtual Private Networks* in 2006 and *Scalix: Linux Administrator's Guide* in 2008.

He is interested in anything concerning geography, traveling, photography, philosophy (especially that of open source software), global politics, soccer, and literature, but always has too little time for these hobbies.

Markus Feilner supports Linux4afrika—a project bringing Linux computers into African schools.

For more information, please visit `http://www.linux4afrika.de`.

Acknowledgement

About the Co-author

Norbert Graf is a professional IT specialist from Munich with many years of experience in network security and server virtualization. His special fields of interest are Linux-based firewalls, VMware, and XEN virtualization.

Since 2002, he has been working as a consultant for an IT company near Munich, for customers from the healthcare sector like hospitals or pharmaceutical concerns to small companies.

He made his first experiences with computers with the Commodore C64 learning to program in basic, followed by an x86 processor PC with DOS and Windows. He is still working with Windows and Linux networks every day. His field of work especially includes integrating Linux servers like Proxies or OpenVPN servers in Microsoft Active Directory infrastructures.

Since 2007, he has published several articles (mostly about Windows and Linux cooperation) together with Markus Feilner in the German and International *Linux Magazine*.

In November 2007, his son Moritz was born and made the whole family very happy.

About the Reviewers

Chris Buechler is the co-founder and Chief Technology Officer of BSD Perimeter LLC, the corporate arm of the pfSense open source firewall distribution. He has more than a decade of IT experience and holds numerous industry certifications including CISSP, SSCP, MCSE, and CCNA among others. He served as the contributing author on security for the book *SharePoint 2007: The Definitive Guide* from O'Reilly and is the primary author of a book on pfSense to be published by Reed Media in 2009. He has presented on security topics at more than a dozen conferences in the US and Canada. He can be reached at `cmb@chrisbuechler.com`.

Ralf Hildebrandt holds a degree in computer science and has been working with Unix since 1994. His experience with computers dates back to 1984 and a sturdy old C64. Recently, he changed employer from T-Systems to Charite and became `postmaster@python.org`, thus gaining experience in running large listservers.

Ralf is the co-author of *The Book of Postfix*.

Table of Contents

Preface

OpenVPN is an outstanding piece of software that was invented by James Yonan in the year 2001 and has steadily been improved since then. No other VPN solution offers a comparable mixture of enterprise-level security, usability, and feature richness. We have been working with OpenVPN for many years now, and it has always proven to be the best solution. This book is intended to introduce OpenVPN software to network specialists and VPN newbies alike. OpenVPN works where most other solutions fail and exists on almost any platform. Thus, it is an ideal solution for problematic setups and an easy approach for the inexperienced.

On the other hand, the complexity of classic VPN solutions, especially IPsec, gives the impression that VPN technology in general is difficult and a topic only for very experienced (network and security) specialists. OpenVPN proves that this can be different, and this book aims to document that.

I want to provide both a concise description of OpenVPN's features and an easy-to-understand introduction for the inexperienced. Though there may be many other possible ways to success in the scenarios described, the ones presented have been tested in many setups and have been selected for simplicity reasons.

What this book covers

Chapter 1, *VPN – Virtual Private Network*, gives a brief overview about what VPNs are, what security means here, and similar important basics.

Chapter 2, *VPN Security*, introduces basic security concepts necessary to understand VPNs and OpenVPN in particular. We will have a look at encryption matters, symmetric and asymmetric keying, and certificates.

Chapter 3, *OpenVPN*, discusses OpenVPN, its development, features, resources, advantages, and disadvantages compared to other VPN solutions, especially IPsec.

Chapter 4, *Installing OpenVPN on Windows and Mac*, shows step-by-step how to install OpenVPN on clients using Apple or Microsoft products.

Chapter 5, *Installing OpenVPN on Linux and Unix Systems*, deals with simple installation on Linux and Unix.

Chapter 6, *Advanced OpenVPN Installation*, shows you how to get OpenVPN up and running even when it gets difficult or non-standard.

Chapter 7, *Configuring an OpenVPN Server — The First Tunnel*, introduces the use of OpenVPN to build a first tunnel.

Chapter 8, *Setting Up OpenVPN with X.509 Certificates*, explains us how to use OpenVPN to build a tunnel using the safe and easily manageable certificates.

Chapter 9, *The Command openvpn and Its Configuration File*, groups an abundance of command-line options that OpenVPN has to offer into several tables, which enable you to search and find the relevant once far more easily.

Chapter 10, *Securing OpenVPN Tunnels and Servers*, shows how to use several Firewalls (Windows and Linux) and security-relevant extensions like Authentication for OpenVPN.

Chapter 11, *Advanced Certificate Management*, deals with security issues, and advanced certificate management tools, such as TinyCA or xca, help us understand and manage a PKI thoroughly.

Chapter 12, *OpenVPN GUI Tools*, shows you how to choose a suitable client out of three GUI-tools for OpenVPN for your setup.

Chapter 13, *Advanced OpenVPN Configuration*, discusses tunneling proxies, pushing configurations from the server to the client, and many other examples up to clusters and redundancy.

Chapter 14, *Mobile Security with OpenVPN*, teaches us how to connect our mobile device, be it Windows Mobile, an embedded Linux device, or a laptop, to our VPN and start communicating privately.

Chapter 15, *Troubleshooting and Monitoring*, will help you in many cases when you run into network problems, or if anything doesn't work.

Appendix, *Internet Resources and More*, holds all abbreviations used and all weblinks found throughout the whole book.

What you need for this book

For learning VPN technologies, it may be helpful to have at least two or four PCs. Virtualization tools like KVM, XEN, or VMware are very helpful here, especially if you want to test with different operating systems and switch between varying configurations easily. However, one PC is completely enough to follow the course of this book.

Two separate networks (connected by the Internet) can provide a useful setup if you want to test firewall and advanced OpenVPN setup.

Who this book is for

This book is for Newbies and Admins alike. Anybody interested in security and privacy in the internet, and anybody who wants to have his or her notebook or mobile phone connect safely to the Internet will learn how to connect to and how to set up the server in the main branch of his or her company or at home. You will learn how to build your own VPN, surf anonymously and without censorship, connect branches over the Internet in a safe way, and learn all the basics on how to administer and build Virtual Private Networks.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code will be set as follows:

```
remote xxx.dyndns.org
(...)
tls-remote "/C=DE/ST=BY/O=Feilner-IT/CN=VPN-Server/
emailAddress=security@feilner-it.net"
(...)
resolv-retry 86400
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items will be shown in bold:

```
suse01:/var/log # ldapwhoami -x -h 10.10.10.1 -D
uid=mfeilner,ou=Feilner-it_Users,dc=feilner-it,dc=home -w correct_
password
dn:uid=mfeilner,ou=Feilner-it_Users,dc=feilner-it,dc=home

suse01: # ldapwhoami -x -h 10.10.10.1 -D uid=mfeilner,ou=Feilner-it_
Users,dc=feilner-it,dc=home -w wrong_password
ldap_bind: Invalid credentials (49)
```

Any command-line input or output is written as follows:

```
opensuse01:~ # echo "1" > /proc/sys/net/ipv4/ip_forward
opensuse01:~ #
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "Start YaST on your SuSE Linux system and change to the **Firewall** module, which can be found in **Security and Users**".

> Warnings or important notes appear in a box like this.

> Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an email to feedback@packtpub.com, and mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or email suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration, and help us to improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **let us know** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata added to any list of existing errata. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or web site name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
VPN—Virtual Private Network

This chapter will start with networking solutions that were used in the past for connecting several branches of a company. Technological advances, such as broadband Internet access, brought about new possibilities and new concepts for this issue, one of them being the **Virtual Private Network** (**VPN**). In this chapter, you will learn what the term VPN means, how it evolved during the last few decades, why it is a necessity for modern enterprises, and how typical VPNs work. Basic networking concepts are necessary to understand the variety of possibilities that VPNs offer.

Historical: In former times, information exchange between branches of a company was mainly done by mail, telephone, and later by fax. But today there are five main challenges for modern VPN solutions that are discussed in this chapter. The challenges faced by companies are as follows:

- The general acceleration of business processes and the rising need for fast, flexible information exchange between all branches of a company have made 'old-fashioned' mail and even fax services appear to be too slow for modern requirements.

- Technologies, such as Groupware, **Customer Relationship Management** (**CRM**), and **Enterprise Resource Planning** (**ERP**) are used to ensure productive teamwork, and every employee is expected to cooperate.

- Almost every enterprise has several branches in different locations and often has field and home workers. All of these must be enabled to participate in internal information exchange without delays.

- All computer networks have to fulfill security standards to high levels to ensure data integrity, authenticity, and stability.

- Secure and flexible access for mobile devices has to be implemented, including new strategies for laptops and modern smartphones.

These factors have led to the need for sophisticated networking solutions between companies' offices all over the world. With computer networks connecting all desktops within a single location, the need for connections between sites has become more and more urgent.

Many years ago you could only rent dedicated lines between your sites. These lines were expensive, thus only large companies could afford to connect their branches to enable worldwide team working. To achieve this fast and expensive connections had to be installed at every site, costing much more than normal enterprise Internet access.

The concept behind this network design was based on a real network between the branches of the company. A provider was needed to connect every location and a physical cable connection between all branches was established. Like the telephone network, a single dedicated line connecting two partners was used for communication.

Security for this line was achieved by providing a dedicated network—every connection between branches had to be installed with a leased line. For a company with four branches (A, B, C, and D), six dedicated lines would then become necessary.
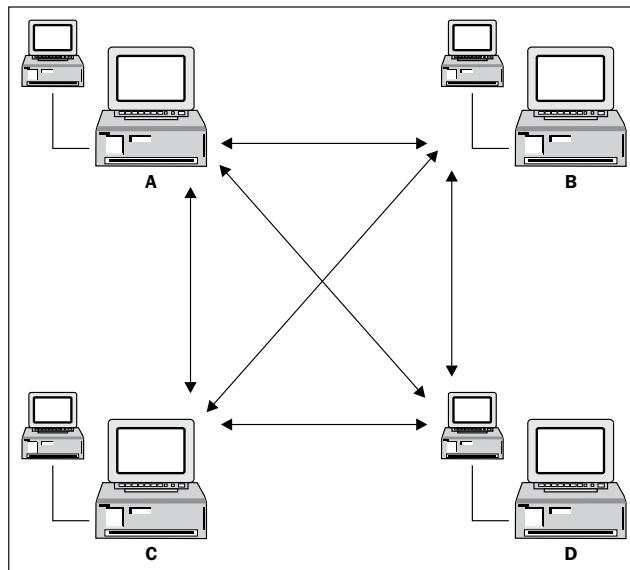
Furthermore, **Remote Access Servers** (**RAS**) were used for field or home workers, who would only connect temporarily to the company's network. These people had to use special dial-in connections (with a modem or ISDN line) and the company acted as an Internet provider. For every remote worker, a dial-in account had to be configured and field workers could only connect over this line. The telephone company provided one dedicated line for every dial-up and the central branch had to make sure that enough telephone lines were always available.

By protecting the cables and the dial-in server, a real private network was installed at very high cost. Privacy within the company's network spanning multiple branches was achieved by securing the lines and providing services only to hard-wired connection points. Almost all security and availability tasks were handed over to the service provider at very high cost. But by connecting sites directly, a higher data transfer speed could be achieved than with 'normal' Internet connections at that time.

Until the middle of the 1990s, expensive dedicated lines and dial-in access servers were used to enable team work between different branches and field workers of large companies.

Broadband Internet access and VPNs

In the mid 1990s, the rise of the Internet and the increase in speed of cheap Internet connections paved the way for new technologies. Many developers, administrators, and last but not least, managers, had discovered that there might be better solutions than spending several hundreds of dollars, if not thousands of dollars, on dedicated and dial-up access lines.

The idea was to use the Internet for communication between branches and at the same time ensure the safety and secrecy of the data transferred. In short, to provide secure connections between enterprise branches through low-cost lines using the Internet. This is a very basic description of what **VPNs** are all about.

A VPN is:

- **Virtual**: This is because there is no real direct network connection between the two (or more) communication partners, only a virtual connection provided by **VPN software**, realized normally over public Internet connections.
- **Private**: This is because only the members of the company connected by the **VPN software** are allowed to read the data that has been transferred.

With a VPN, your staff in Sydney can work with the London office as if both were in the same location. The VPN software provides a virtual network between those sites using a low-cost Internet connection. This network is called **virtual** because no real, dedicated network connection to the partner is being established.
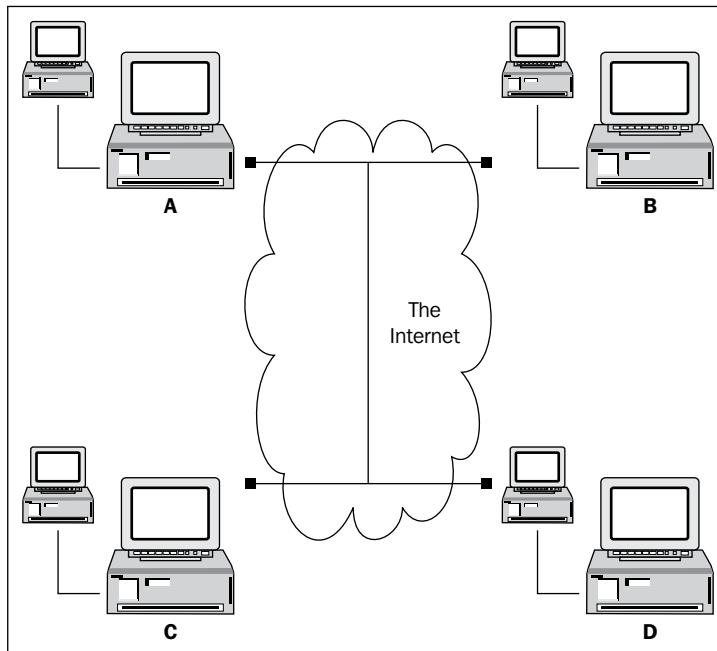
A VPN can also be described as a set of logical connections secured by special software that establishes privacy by safeguarding the connection endpoints. Today the Internet is the network medium used, and privacy is achieved by modern cryptographic methods.

How does a VPN work?

Let's use an example to explain how VPNs work. The **Virtual Entity Networks Inc.** (**VEN Inc.**) has two branches, London and Sydney. If the Australian branch in Sydney decides to contract a supplier, then the London office might need to know that immediately. The main part of the IT infrastructure is set up in London. In Sydney there are twenty people whose work depends on the availability of the data hosted on London servers.

-------- encrypted connection tunnel
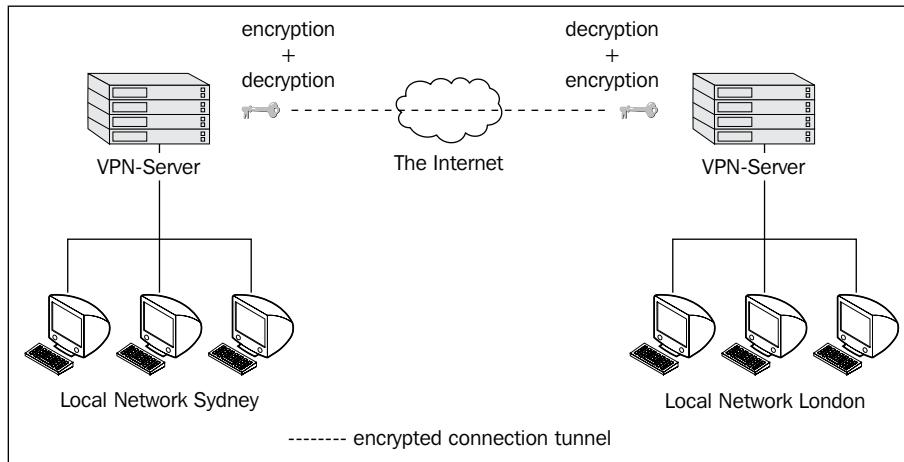
Both sites are equipped with a permanent Internet line. An Internet gateway router is set up to provide Internet access for the staff. This router is configured to protect the local network of the site from unauthorized access from the other side—the 'evil' Internet. Such a router set up to block special traffic can be called a **firewall** and must be installed and configured in every branch that is supposed to take part in the VPN.

The VPN software must be installed on this firewall (or a device or server protected by it). Every modern firewall appliance includes this feature, and there is VPN software for all hardware and software platforms.

In the next step, the VPN software has to be configured to establish the connection to the other side. For example, the London VPN server has to accept connections from the Sydney server, and the Sydney server must connect to London (or vice versa).

If this step is completed successfully, then the company has a working virtual network. The two branches are connected through the Internet and can work together as in a real network. Here, we have a VPN without privacy, because any Internet router between London and Sydney can read the exchanged data. A competitor gaining control over an Internet router could read all the relevant business data that is going through the virtual network.

So how do we make this virtual network private? The solution is encryption. The VPN traffic between the two branches is locked with special keys, and only computers or persons owning this key can open this lock and look at the data that has been sent.
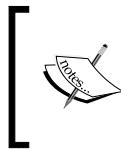
In fact all encryption technology can be hacked. Decrypting information without the right key is merely a question of time, force, and resources. A very good explanation of this is in the book, *Time Based Security* by Winn Schwartau.
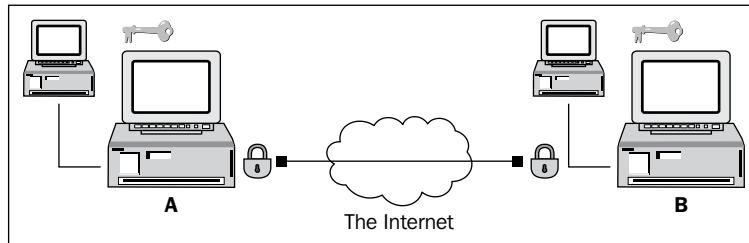
All data that has been sent from Sydney to London or from London to Sydney must be encrypted before and decrypted after transmission. The encryption safeguards the data in the connection in the same way the walls of a tunnel protect a train from the mountain around it. This explains why Virtual Private Networks are often simply known as tunnels or VPN tunnels, and the technology is often called **tunneling** — even if there is no quantum mechanics or other magic involved.

The exact method of encryption and providing the keys to all parties that are involved makes one of the main distinguishing factors between different VPN solutions.

A VPN connection is normally built between two Internet access routers that are equipped with a firewall and VPN software. The software must be set up to connect to the VPN partner, the firewall must be set up to allow access, and the data that is exchanged between VPN partners must be secured (by encryption). The encryption key must be provided to all VPN partners so that the data exchanged can only be read by authorized VPN partners.

What are VPNs used for?

In the earlier examples, we discussed several possible scenarios for the use of VPN technology. But one typical VPN solution must be added here. More and more enterprises offer their customers or business partners a protected access to relevant data for their business relations such as ordering formulae or stock data. Thus, we have three typical scenarios for VPN solutions in modern enterprises as follows:

- An intranet spanning over several locations of a company
- A dial-up access for home or field workers with changing IPs, mobile devices, and centralized protection
- An extranet for customers or business partners

Each of these typical scenarios requires special security considerations and setups. The external home workers will need different access to servers in the company than the customers and business partners. In fact, access for business partners and customers must be restricted severely.

Now that we have seen how a VPN can securely interconnect a company in different ways, we will have a closer look at the way VPNs work. To understand the functionality, some basic network concepts need to be understood.

All data exchange in computer networks is based on protocols. Protocols are like languages or rituals that must be used between communication partners in networks. Without the correct use of the correct protocol, communication fails.

Networking concepts—protocols and layers

There are a large number of protocols involved in any action you take when you access the Internet or a PC in your local network. Your **Network Interface Card** (**NIC**) will communicate with a hub, a switch, or a router. Your application will communicate with its partner on a server on another PC, and many more protocol-based communication procedures are necessary to exchange data.

Because of this, the **Open Systems Interconnection** (**OSI**) specification was created. Every protocol used in today's networks can be classified by this scheme.

The OSI specification defines seven numbered layers of data exchange which start at layer 1 (the physical layer) of the underlying network media (electrical, optical, or radio signals) and span up to layer 7 (the application layer), where applications on PCs communicate with each other.

The layers of the OSI model are as follows:

- **Physical layer**: Sending and receiving through the hardware
- **Data link layer**: Encoding and decoding data packets into bits
- **Network layer**: Switching, routing, addressing, error handling, and so on
- **Transport layer**: End-to-end error recovery and flow control

- **Session layer**: Establishing connections and sessions between applications
- **Presentation layer**: Translating between application data formats and network formats
- **Application layer**: Application-specific protocols

This set of layers is hierarchical and every layer serves the layer above and the layer below. If the protocols of the physical layer could communicate successfully, then the control is handed to the next layer, the data link layer. Only if all layers, 1 through 6, can communicate successfully, can data exchange between applications (on layer 7) achieved. A good introductory read to the OSI model can be found in Wikipedia: `http://en.wikipedia.org/wiki/OSI_model` and a list of OSI protocols at `http://en.wikipedia.org/wiki/OSI_protocols`.

In the Internet, however, a slightly different approach is used.

The Internet is mainly based on the **Internet Protocol** (**IP**).

The layers of the IP model are as follows:

- **Link layer**: A concatenation of OSI layers 1 and 2 (the physical and data link layers).
- **Network layer**: Comprising the network layer of the OSI model.
- **Transport layer**: Comprising protocols, such as **Transmission Control Protocol** (**TCP**) and **User Datagram Protocol** (**UDP**), which are the basis for protocols of the application layer.
- **Application layer**: Concatenation of OSI layers 5 through 7 (the session, presentation, and application layers). The protocols in the transport layer are the basis for protocols of the application layer (layer 5 through layer 7) such as HTTP, FTP, or others.

A TCP/IP network packet consists of two parts — header and data. The header is a sort of label containing metadata on sender, recipient, and administrative information for the transfer. On the networking level of an Ethernet network these packets are called **frames**. In the context of the Internet Protocol these packets are called **datagrams**, **Internet datagrams**, **IP datagrams**, or simply **packets**. Again, a very good introductory article can be found in Wikipedia: `http://en.wikipedia.org/wiki/Internet_Protocol`.

So what do VPNs do? VPN software takes IP packets or Ethernet frames and wraps them into another packet. This may sound complicated, but it is a very simple trick, as the following examples will show:

Example 1: Sending a (not really) anonymous parcel.

You want to send a parcel to a friend who lives in a community with strange people whom you don't trust. Your parcel has the address label with sender and recipient data (like an IP packet). If you do not want the community to know that you sent your friend a parcel, but at the same time you want your friend to realize this before he opens it, what would you do? Just wrap the whole parcel in another packet with a different address label (without your sender information) and no one in the community will know that this parcel is from you. But your friend will unpack the first layer and see a parcel still unpacked with an address label from you.

Example 2: Sending a locked parcel.

Let's distrust the community still more. Somebody might want to open the parcel in order to find out what's inside. To prevent this we will use a locked case. There are only two keys to the lock, one for us and one for our friend. Only we and our friend can unlock the case and look inside the packet.

VPN software uses a combination of the earlier two examples:

- Whole network packets (frames, datagrams) consisting of header and data are **wrapped** into new packets
- All data, including metadata, such as recipient and sender, are encrypted
- The new packets are labeled with new headers containing meta-information about the VPN and are addressed to the VPN partner

All VPN software systems differ only in the special way of *wrapping* and *locking* the data.
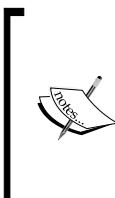
> Protocols define the method of data exchange in computer networks. The OSI model classifies protocols in seven layers, spanning from network layers to application layers. IP packets consist of headers with meta-information and data. VPNs wrap and encrypt whole network packets in new network packets, adding new headers including address data.

Tunneling and overhead

We have already learned that VPN technology is often called tunneling because the data in a VPN connection is protected from the Internet, as the walls of a road or rail tunnel protect the traffic in the tunnel from the weight of stone of the mountain above. Let's now have a closer look at how the VPN software does this.
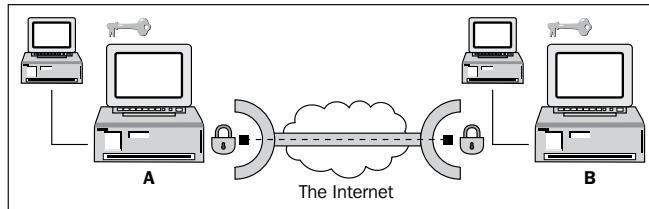
The VPN software in the locations **A** and **B** encrypts (locks) and decrypts (unlocks) the data and sends it through the tunnel. Like cars or trains in a tunnel, the data cannot go anywhere else but to the other tunnel endpoint (if they are properly routed).

The following are put together and wrapped into one new package:

- Tunnel information (such as the address of the other endpoint)
- Encryption data and methods
- The original IP packet (or network frame)

The new package is then sent to the other tunnel endpoint. The payload of this package now holds the complete IP packet (or network frame), but in an encrypted form. Therefore it is not readable to anyone who does not possess the right key. The new header of the packet simply contains the addresses of the sender, recipient, and other metadata that is necessary for and provided by the VPN software that is used.

Perhaps you have noticed that the amount of data that is sent grows during the process of 'wrapping'. Depending on the VPN software used, this so-called overhead can become a very important factor. The **overhead** is the difference between the net data that is sent to the tunnel software and the gross data that is sent through the tunnel by the VPN software. If a file of 1MB is sent from user A to user B, and this file causes 1.5MB traffic in the tunnel, then the overhead would be 50%, a very high level indeed (note that every protocol that is used causes overhead, so not all of that 50% might be the fault of the VPN solution.). The overhead caused by the VPN software depends on the amount of organizational (meta-) data and the encryption used. Whereas the first depends only on the VPN software used, the latter is simply a matter of choice between security and speed. In other words, the better the cipher you use for encryption, the more overhead you will produce. Speed versus security is your choice.
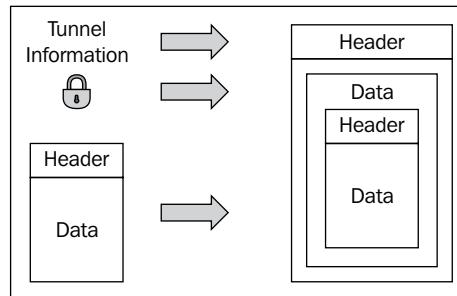
VPN concepts—overview

During the last ten years, many different VPN concepts have evolved. You may have noticed that I added 'network frames' in parenthesis when I spoke of tunneling IP packets. This was necessary because, in principle, tunneling can be done on almost all layers of the OSI model.

A proposed standard for tunneling

The **General Routing Encapsulation** (**GRE**) provides a standard for tunneling data, which was defined in 1994 in **Request for Comments** (**RFCs**) 1701 and 1702, and later in RFCs 2784 and 2890. Perhaps because this definition is not a protocol definition, but more or less a standard proposal on how to tunnel data, this implementation has found its way into many devices and has become the basis for other protocols.

The concept of GRE is pretty simple. A protocol header and a delivery header are added to the original packet, and its payload is encapsulated in the new packet. If no encryption is done, then GRE offers no security. The advantages of this model are obvious—the simplicity offers many possibilities: the transparency enables administrators and routers to look inside the packets and pass decisions based on the type of payload that has been sent. By doing so, special applications can receive privileged treatment by traffic shaping or similar methods.

There are many implementations for GRE tunneling software under Linux. Only kernel support is necessary, which is fulfilled by most modern distributions. Due to its flexibility, GRE can also be used in scenarios where IPv4- and IPv6-networks collide, or for tunneling Netware's or Apple's protocols. GRE is assigned the IP protocol number 47.

Protocols implemented on OSI layer 2

Encapsulating packages on the OSI layer 2 has a significant advantage — the tunnel is able to transfer non-IP protocols. IP is a standard that is widely used in the Internet and in Ethernet networks. However there are different standards in use. Netware Systems, for example, uses the **Internetwork Packet Exchange (IPX**) protocol to communicate. VPN technologies residing in layer 2 can theoretically tunnel any kind of packet. In most cases a virtual **Point-to-Point Protocol** (**PPP**) device is established, which is used to connect to the other tunnel endpoint. A PPP device is normally used for modem or DSL connections.

Four well known layer 2 VPN technologies, which are defined by RFCs, use encryption methods and provide user authentication, as follows:

1. The **Point to Point Tunneling Protocol** (**PPTP),** RFC 2637, which was developed with the help of Microsoft, is an expansion of the PPP. It is integrated in all newer Microsoft operating systems. PPTP uses GRE for encapsulation and can tunnel IP, IPX, and other protocols over the Internet. The main disadvantage is the restriction that there can only be one tunnel at a time between communication partners.

2. The **Layer 2 Forwarding** (**L2F),** RFC 2341, was developed almost at the same time by other companies, including Cisco, and offers more possibilities than PPTP, especially regarding tunneling of network frames and multiple simultaneous tunnels.

3. The **Layer 2 Tunneling Protocol** (**L2TP),** RFC 2661, is accepted as an industry standard and is being widely used by Cisco and other manufacturers. Its success is based on the fact that it combines the advantages of L2F and PPTP without suffering their drawbacks. Even though it does not provide its own security mechanisms, it can be combined with technologies offering such mechanisms, such as **IPsec** (see the section *Protocols Implemented on OSI layer 3*).

4. The **Layer 2 Security Protocol** (**L2Sec),** RFC 2716, was developed to provide a solution to the security flaws of IPsec. Even though its overhead is rather big, the security mechanisms that are used are secure, because mainly SSL/TLS is used.

Other distinguishing factors between the mentioned systems and protocols are as follows:

- Availability of authentication mechanisms
- Simple and complete support for advanced networking features such as **Network Address Translation (NAT**)
- Dynamic allocation of IP addresses for tunnel partners in dial-up mode
- Support for **Public Key Infrastructures** (**PKI)**

These features will be discussed in later chapters.

Protocols implemented on OSI layer 3

IPsec (**Internet Protocol Security**) is the most widespread tunneling technology. In fact it is a more complex set of protocols, standards, and mechanisms than a single technology. The wide range of definitions, specifications, and protocols is the main problem with IPsec. It is a complicated technology with many different implementations and many security loopholes. IPsec was a compromise accepted by a commission, and therefore is something as a least common denominator that has been agreed upon. This means that IPsec can be used in many different setups and environments, ensuring compatibility, but almost no aspect of it offers the best possible solution.

IPsec was developed as an Internet Security Standard on layer 3 and has been standardized by the **Internet Engineering Task Force** (**IETF**) since 1995. IPsec can be used to encapsulate any traffic of application layers, but no traffic of lower network layers. Network frames, IPX packets, and broadcast messages cannot be transferred, and network address translation is only possible with restrictions.

Nevertheless IPsec can use a variety of encryption mechanisms, authentication protocols, and other security associations. IPsec software exists for almost every platform. Compatibility with the implementation of other manufacturers' software is secured in most cases, even though there can be significant problems resulting from proprietary extensions.

The main advantage of IPsec is the fact that it is being used everywhere. An administrator can choose from a large number of hardware devices, software implementations, and administration frontends to provide networks with a secure tunnel.

Basically there are two methods that IPsec uses. They are as follows:

- **Tunnel mode**: The tunnel mode works like the examples listed above. All the IP packets are encapsulated in a new packet and sent to the other tunnel endpoint, where the VPN software unpacks them and forwards them to the recipient. In this way the IP addresses of sender and recipient and all other metadata are protected.

- **Transport mode**: In transport mode, only the payload of the data section is encrypted and encapsulated. In this way the overhead becomes significantly smaller than in tunnel mode, but an attacker can easily read the metadata and find out who is communicating with whom. However the data is encrypted and therefore protected, which makes IPsec a real 'private' VPN solution.

IPsec's security model is probably the most complex of all existing VPN solutions and will be discussed in brief in the next chapter. It has been specified in several RFCs. A long list of these together with a good introduction can be found in Wikipedia: `http://en.wikipedia.org/wiki/IPsec`.

Protocols implemented on OSI layer 4

It is also possible to establish VPN tunnels using only the application layer. **Secure Sockets Layer** (**SSL**) and **Transport Layer Security** (**TLS**) solutions follow this approach. **Secure Shell** (**SSH**) tunnels are a typical example of that, and they are widespread among Linux/Unix users. Consider the following command:

```
ssh mfeilner@ssh-server -L 1143:mailserver:143
```

The user `mfeilner` has opened a tunnel through the company's firewall to the remote `mailserver` to his local port 1143. The only prerequisite is an SSH server with an appropriate account. More details on this so-called SSH forwarding can be found here: `http://www.ssh.com/support/documentation/online/ssh/winhelp/32/Tunneling_Explained.html`.

A field worker can access a SSL-VPN network using a simple browser connection between his or her client and the VPN server in the enterprise. This is simply started by logging into an HTTPS-secured web site with a browser. Meanwhile, there are several promising products available, such as SSL-Explorer from `http://3sp.com/showSslExplorer.do`, and software like this can offer great flexibility when combined with strong security and easy setup. Using the secure connection that the browser offers, users can connect network drives and access services in the remote network. Security is achieved by encrypting traffic using SSL/TLS mechanisms, which have proven to be very reliable and are permanently being improved and tested.

Recently many hardware vendors have developed and integrated such SSL-VPNs, but none of them are compatible with other vendor versions, and the security aspect is a matter of trust in the vendor. In most cases it's better to stick to a standard implementation.

OpenVPN—a SSL/TLS-based solution

OpenVPN is a newer and an outstanding newer VPN solution that combines several advantages of the previously described technologies. It implements layer 2 or layer 3 connections, uses the industry standard SSL/TLS for encryption, and combines almost all features of the mentioned VPN solutions. Its main disadvantage is the fact that there are currently very few hardware manufacturers that are integrating it in their products but it is becoming more and more interesting for industry grade products such as MoRoS (`http://www.insys-tec.de/moros`), which is carrying an embedded Linux with an OpenVPN solution as a central component for remote access.

Summary

In this chapter, you have learned about techniques that have been, and are, used in companies that have computer networks spanning over several branches. You have learned network basics, such as protocols, networking layers, the OSI reference model, and which VPN solutions work on which layer. You have read what tunneling is, how it works, and how different VPN solutions implement it.

Furthermore, you have received a first glimpse of where OpenVPN has its strengths and weaknesses. We will now dive in deeper into OpenVPN in the next chapter.

2
VPN Security

In this chapter, we will discuss goals and techniques concerning VPN security. These two terms are linked together very closely. Without security, a VPN is not private anymore.

Therefore, we will first have a look at basic security issues and guiding measures to be taken in a company. Information on symmetric and asymmetric keying methods, key exchange techniques, and the problem of security versus simplicity pave the way for SSL/TLS security and a closer look at SSL certificates. After having read this chapter, you will be ready to understand the underlying security concerns of OpenVPN (and any other VPN solution).

VPN security

IT security, and therefore VPN security, is best described by the three goals that have to be attained. They are as follows:

- **Privacy** (**Confidentiality**): The data transferred should only be available to the authorized
- **Reliability** (**Integrity**): The data transferred must not be changed between sender and receiver
- **Availability**: The data transferred must be available when needed

Furthermore, a VPN solution must offer secure authentication and non-repudiation. All of these goals have to be achieved using reliable software, hardware, Internet service providers, and security policies. A security policy defines responsibilities, standard procedures, and disaster management and recovery scenarios to be prepared for the worst. Understanding maximum damage and the costs of the worst possible catastrophe can give an idea of how much effort should be expended on security issues. Security policies should also define organizational questions such as:

- Who has the key to the server room when the administrator is on holiday?
- Who is allowed to bring in a private laptop?
- How are the cables protected?
- How is a **wireless LAN (WLAN**) protected?

However, discussing all these questions would go far beyond the scope of this book. There are a number of excellent documents online, where you can read more about basic security issues that should also be discussed in your company. I only want to mention two of them here — the *IT Baseline Protection* (`http://www.bsi.bund.de/ english/gshb/index.htm` and `http://www.cccure.org/Documents/HISM/ ewtoc.html`) as published by the German BSI and the *IT-Sec Handbook* (`http://www.cccure.org/Documents/HISM/ewtoc.html`) containing concise security hints. They are often quoted as the reference material for all security issues in modern enterprises. The same applies to the Handbook of *Information Security Management* (`http://www.cccure.org/Documents/HISM/ewtoc.html`).

VPN security itself is achieved by protecting traffic with modern, strong encryption methods, secure authentication techniques, and firewalls controlling traffic into and out of the tunnels. Simply encrypting traffic is not enough as there are huge differences in security depending on the methods used. The following sections will deal with issues concerning confidentiality and integrity, whereas the approach to ensuring availability is discussed in the following chapter.

Privacy—encrypting traffic

Often passwords or encryption keys are used to encrypt data. If both sides use the same key to encrypt and decrypt data, it is called **symmetric encryption**. The encryption key has to be put on all machines that are supposed to take part in the VPN connection.

Symmetric encryption and pre-shared keys

Anybody who has this key can decrypt the traffic. If an attacker gets hold of this key, he can decrypt all traffic and compromise all systems that are taking part in the VPN until all systems are supplied with another key. Furthermore, such a static, pre-shared key can be guessed, deciphered, or hacked by brute-force attacks. It is merely a matter of time for an attacker to find out the key to read, or even worse, change the data.

Encryption and Decryption
with pre-shared key

Message

Message

asd 234H
FKNYX

transport through tunnel

asd 234H
FKNYX

Sydney

London

encrypted message
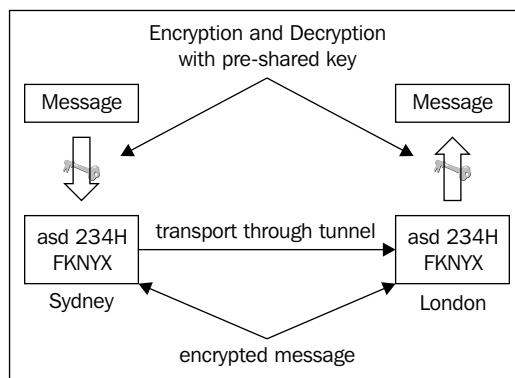
Therefore, VPN software, like IPsec, changes keys at defined intervals. Every key is only valid for a certain period of time called **key lifetime**. A good combination of key lifetime and key length ensures that an attacker cannot decrypt the key while it is still valid. If the VPN software is changing keys, then the attacker must be quick, or the acquired key is worthless.

Nevertheless, if the VPN software is permanently changing keys, a method of key exchange between the communication partners has to be used to ensure that both sides use the same encryption key at the same time. This key exchange must also be secured again, following the same principles mentioned earlier. During the last decade, many key exchange methods have been invented, some very sophisticated, and lots of them have proven insecure since. Basically, this key exchange adds a layer of complexity to the VPN software, which is prone to failure or being compromised.

IPsec, the most frequently used VPN technology, brings its own protocol for exchanging the encryption keys. This protocol is called **Internet Key Exchange** (**IKE**) protocol, and has been in development since the mid-nineties and is still not finished. Many discussions about the security of this protocol can be found on the Internet and even though IKE seems to have some security issues, it is used (with IPsec) in many companies.

Reliability and authentication

Another danger is the so-called **man-in-the-middle (MITM)** attack (`http://en.wikipedia.org/wiki/Man-in-the-middle`), also known as **eavesdropping**. In this scenario, a hacker intercepts all data traffic between sender and receiver, copies it and forwards it to its true destination. Neither sender nor receiver would notice that the data is being intercepted. The man-in-the-middle can store, copy, analyze, and perhaps even modify the captured traffic. This is possible if the attacker can intercept and decrypt the keys while they are being used for encryption.

The problem of complexity in classic VPNs

With classical VPNs that use symmetric keying, there are several layers of authentication, exchange of encryption keys, and encryption/decryption. The following are the first three steps of VPNs with symmetric encryption:

1. The partners have to authenticate each other.
2. They have to agree on encryption methods.
3. They have to agree on the key exchange methods to be used.
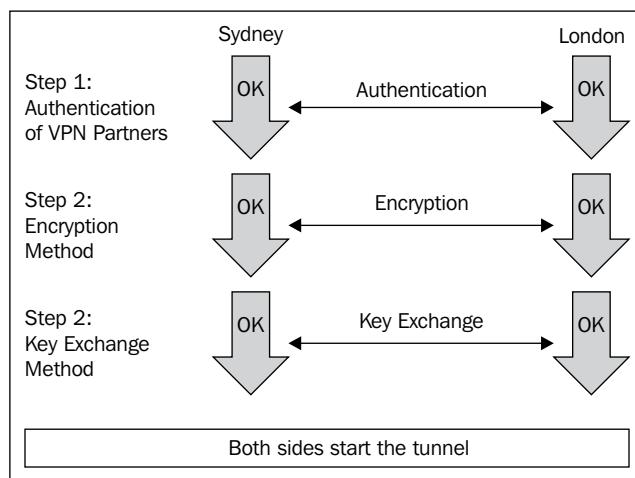
This is why VPN technology is often considered complex and difficult. Previous paragraphs have described more or less the basic way in which many modern VPN solutions work. In a nutshell, the different approaches to keying, key exchange, and authentication of VPN partners make the main part of the differences between the VPN solutions.

Asymmetric encryption with SSL/TLS

SSL/TLS uses one of the best encryption technologies, called **asymmetric encryption**, to ensure the identity of the VPN partner. Both encryption partners own two keys each—one public and the other private. The public key is handed over to the communication partners who encrypt the data with it. Because of the selected mathematical algorithm used to create the public/private key pair, only the recipient's private key can decrypt data encoded by his public key.
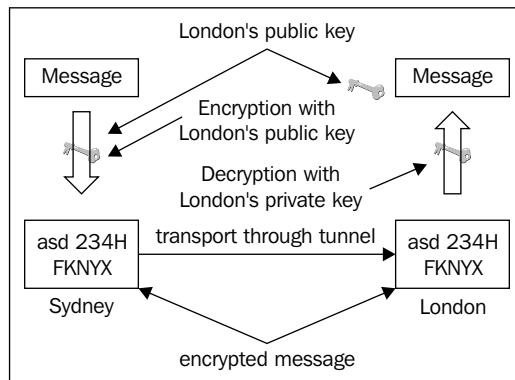
The private keys have to be kept secret and the public keys have to be exchanged. In the previous example, a text message is encrypted in Sydney with the public key of London. The scrambled code is sent to London, where it can be deciphered using London's private key. This can be done vice versa for data from London to Sydney, which is encrypted by the Sydney public key in London and can only be decrypted by the Sydney private key in Sydney.

A similar procedure can also be used for authentication purposes. London sends a large random number to Sydney, where this number is encoded with the private key and sent back. In London, the Sydney public key can decode the number. If the numbers sent and decrypted match, then the sender must be the holder of the Sydney private key. This is called a **digital signature**.

If you want to delve deeper into how OpenVPN and OpenSSL work, then here are some good reads:

- *OpenVPN and the SSL Revolution* (`http://www.sans.org/reading_room/whitepapers/vpns/1459.php`) explains in detail how the keying and rekeying is done

- The cryptographic layer introduced by OpenVPN is explained on the project web site's security pages, including its reliability layer (`http://openvpn.net/index.php/documentation/security-overview.html`)

> The book, *VPNs Illustrated: Tunnels, VPNs, and IPsec* by Jon C. Snader has a concise and illustrated chapter (8.5) on OpenVPN. It is available online here: `http://fengnet.com/book/VPNs%20Illustrated%20Tunnels%20%20VPNsand%20IPsec/ch08lev1sec5.html`. Snader studies the security model and shows packet headers, operation codes, and message formats for almost any datagram that OpenVPN can send.

SSL/TLS security

The SSL/TLS library can be used for authentication and encryption purposes. This library is part of the **OpenSSL software** that is installed on any modern operating system. If available, SSL/TLS certificate-based authentication and encryption should always be the first choice for any tunnel that you create. The following part of this chapter takes the user's perspective as the starting point for understanding SSL/TLS certificate issues.

SSL, also known as TLS, is a protocol originally designed by Netscape Communications Corporation to ensure easy-to-use data integrity and authenticity for the fast growing Internet in the 1990s. Anybody using a modern browser can participate in encrypted communication. SSL/TLS is an outstanding technology that is being used all over the Web for banking, e-commerce, or any other application where privacy and security are needed. It is being steadily controlled, debugged, tested, and improved by both open source and proprietary developers and many corporations. RFC 2246 specifies SSL, and with regards to Windows security, there is a good explanation here: `http://www.windowsecurity.com/articles/Secure_Socket_Layer.html`. The home of the OpenSSL project is `http://www.openssl.org`.

HTTPS

As SSL/TLS resides beneath application protocols, it can be used for almost any application. Every surfer has noticed URLs beginning with `https://`, instead of `http://`. This signifies an encrypted connection. Point your browser to a web site encrypted with `https://`, for example, `https://packtpub.com`. Consider the following screenshot:
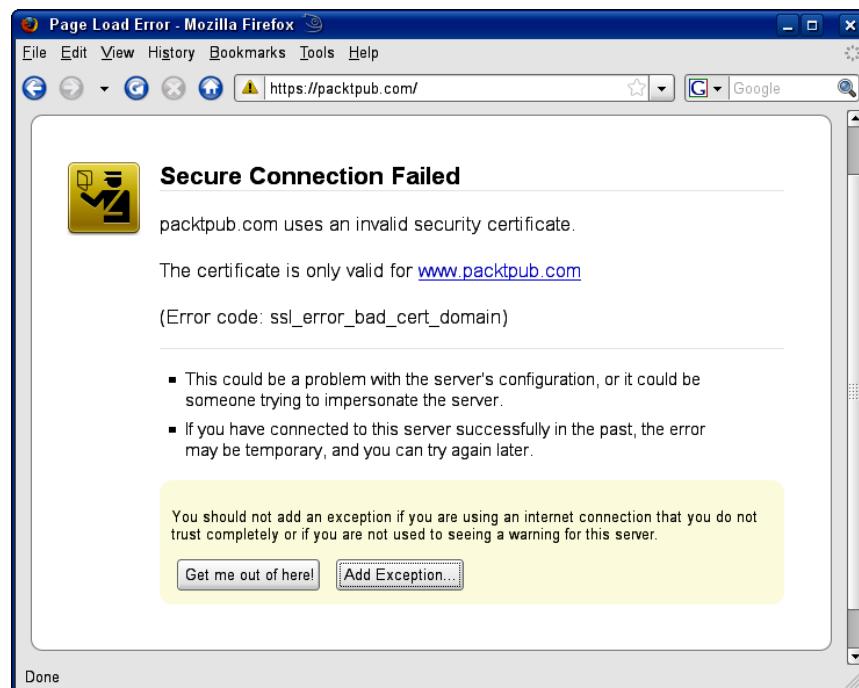
Whenever you point your browser to a page like this for the first time, you have to validate an SSL certificate. Usually your browser does this for you when the certificate is trustworthy. The screenshot shows a Firefox 3 warning, which you receive when there are errors in validating the certificate. Often, with older browsers, this was a problem. People mostly press one of the available buttons (shown in the screenshot) while browsing, without further attention.

Understanding SSL/TLS certificates

By accepting a certificate, that is, by clicking the **Confirm Security Exception** button in Firefox, the browser is told to trust the issuer (the web site that provided the certificate), and you agree to use this certificate for encryption of communication with this server. When you're using Mozilla, Firefox, or Konqueror, you are prompted to accept the certificate.
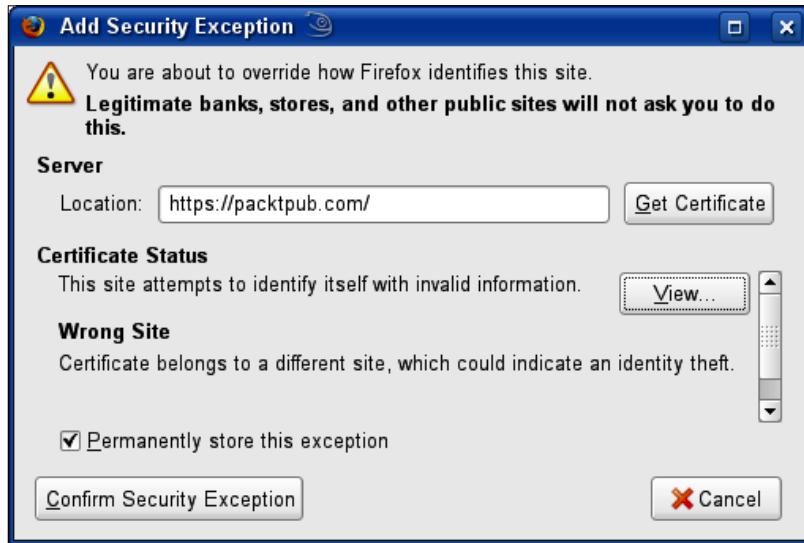
Click on the **View...** button, and you will see a screen like the one that is shown in the next screenshot in the section on *Trusted certificates*.

Trusted certificates

The following window shows the information contained in the SSL certificate. The information in the fields **Issued To** and **Issued By** is probably the most important. If you find the entries here that you were expecting, then it can be safe to trust this certificate. Trustworthy means one of several organizations who *sign certificates*, thereby guaranteeing the identity of the owner of the certificate.

With a signed certificate, the owner of the certificate can prove that he or she is who he or she claims to be to anybody who trusts the certificate authority. Every TLS-enabled browser contains a list of trustworthy organizations that are entitled to sign certificates and the keys necessary to confirm this.
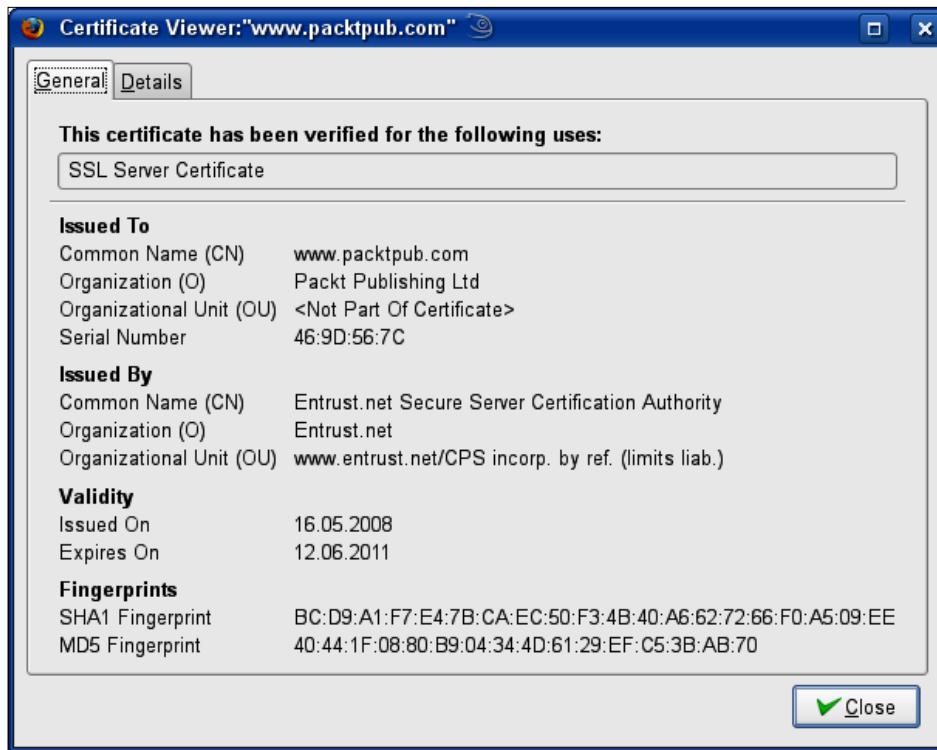
Click on the **Close** button and have another look at the first window, **Secure Connection Failed**. It is in fact a warning. The certificate was originally issued for `www.packtpub.com` and not for `packtpub.com`, from where it was received, and the Firefox SSL client simply warns about the fact. `www.packtpub.com` is a sub domain of `packtpub.com`, so this difference should not be crucial. However, if you receive a warning that the certificate for domain A was originally issued for domain B, then you should become suspicious.

This so-called third party authentication scheme, where a certificate authority guarantees identity, is pretty common today. The ID cards and passports that we use work the same way. The government of the state you live in guarantees that you are who you claim to be. This information is only valid for a certain time and could be traced back to the issuer. Almost every other person, company, or organization relies on this information. These principles are also implemented in many modern authentication mechanisms such as Kerberos or SSL/TLS.

Self-signed certificates

It is also possible to use certificates that are not signed by the mentioned authorities, but by a local **Certificate Authority** (**CA**).

In real life, if a good friend introduces us to a reliable friend of his, then we tend to trust him too, simply because of the recommendation. But we would not trust somebody we do not know. If you point Firefox to a site with a certificate that is signed *only* by a local CA, you will receive the following warning:
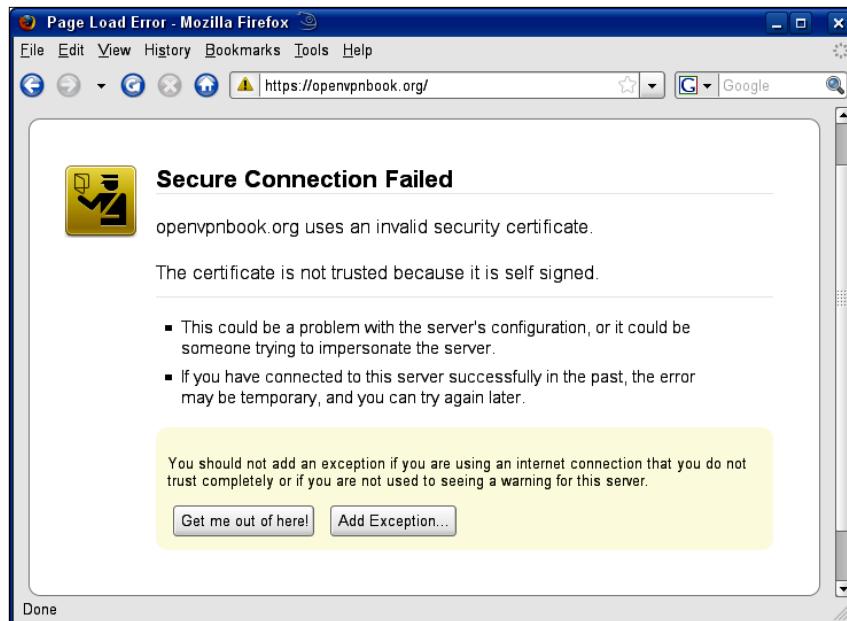
Firefox reports: **The certificate is not trusted because it is self signed.** The warning means, 'Watch out, I do not know the issuer of this certificate, nor do I know someone who guarantees the identity of the issuer'. Every SSL/TLS client gives you a warning when a client wants to establish an encrypted connection with an unsigned private certificate. But where does this certificate come from?

The solution is simple. The OpenSSL software package, which contains the encryption software, also provides programs to create certificates and to sign them. Such certificates are called **self-signed certificates** and can only be considered trustworthy when the issuer or the CA is known to and trusted by the client. Later in this book, you will learn how to create, sign, and manage such certificates.

Self-signed certificates are often used for testing purposes or in local networks because registering (signing) certificates at certificate authorities is expensive and not necessary in many scenarios. However, the security policy of a company should contain procedures for the use of signed and unsigned certificates on servers. Web sites, such as `http://www.pki-page.org`, present long lists of certification authorities all over the world.

SSL/TLS certificates and VPNs

SSL/TLS certificates work exactly the same way with VPNs—a certificate authority is defined or created, and all valid certificates issued by this authority are accepted by the VPN. Every client must have a valid certificate issued by this CA and is therefore allowed to establish a connection to the VPN.

A **Certificate Revocation List** (**CRL**) can be used to revoke certificates that belong to clients who must not be allowed to connect to the VPN any longer. This can be done without configuration on any client, simply by creating an appropriate revocation list on the server. This is very useful when a laptop is stolen or compromised.

An organization using a pre-shared key must put this key on every system that connects to the VPN server. The key must be changed on all systems if just one single system or key is lost. But if you are using certificates with revocation lists, you only have to put the certificate of the stolen laptop on the server's CRL. When this client tries to connect to the server, access will be denied. There is no need for interaction with any client.

Connections are refused if:

- No certificate is presented
- A certificate from an incorrect CA is presented
- A revoked certificate is presented

Such certificates can be used for many purposes. HTTPS and OpenVPN are only two applications of a large variety of possibilities. Other VPN systems (like IPsec), web servers, mail servers, and almost every other server application can use these certificates to authenticate clients. If you have understood and applied this technology correctly, then you have achieved a very high degree of security.

Generating certificates and keys

Several steps have to be accomplished to create a working setup with certificates for any kind of VPN. These steps are as follows:

1. Create a Certification Authority certificate for your CA, which will sign and revoke client certificates.
2. Create a key and a certificate signing request for the clients (or users), or let the users create them.
3. Sign the requests using the CA certificate, thereby making them valid.
4. Provide keys and certificates to the VPN partners.

As you can see, certificate handling can be pretty complex. There are a number of ways to accomplish these steps, and different partners are involved with different actions. There are special software packages such as the ones OpenSSL provides, some of these are really powerful, though they only deal with the topic of handling certificates and keys in medium and large size companies.

The certificate authorities can or should be organized in chains and organizational units, which are allowed to sign certificates and keys only for their organization. For example, in VEN Inc., the administrator of the Sydney branch should be allowed to produce certificates and keys for the Australian field workers. But these should not automatically have access to the Munich network. Thus, access to Sydney's VPN is restricted to certificates of the organizational unit, 'Sydney Branch', and in Germany, to 'Munich Branch'. If there are some people regularly travelling between the two cities then they may need VPN-access on both continents, which could be achieved by having top-level or second-level CA certificates.

Chapters 8 and 11 deal with certificate management in more detail.

Summary

In this chapter, you have learned basic security concepts that are necessary for VPN technologies. There are several web sites with excellent material on IT security issues. You have received an overview of basic security and encryption issues and learnt why complexity is always an enemy of security. With symmetric keying, both encryption partners use the same key, but when asymmetric keying is used, the encryption key is different from the one used for decrypting the data. The SSL/TLS library uses asymmetric keying and provides certificates that are used by millions of web sites running on `https://`. The certificates can be signed by official authorities, in the same way as our passports or ID cards, or self-signed by the local authority that created them. This is called third-party authentication because a certificate signed by that third party is trusted.

3
OpenVPN

In this chapter we will discuss the nature of OpenVPN. We will start with its features and its release history, followed by its basic networking concepts, and a first brief look at the configuration. At the end of the chapter, OpenVPN is compared to IPsec, the quasi-standard in VPN technology.

This chapter will cover the following:

- Advantages of OpenVPN
- History of OpenVPN
- Networking with OpenVPN
- OpenVPN and firewalls
- Configuring OpenVPN
- OpenVPN versus IpSec
- Source for documentation

Advantages of OpenVPN

With the advent of OpenVPN a new generation of VPN entered the scene. While other VPN solutions often use proprietary or non-standard mechanisms, OpenVPN has a modular concept, both for underlying security and for networking. OpenVPN uses the secure, stable, and lauded SSL/TLS mechanisms and combines them in its own reliability layer. It does not suffer from the complexity that characterizes other VPN implementations like the market leader IPsec. At the same time, it offers possibilities that go beyond every other VPN implementation's scope.

- **Layer 2 and Layer 3 VPN**: OpenVPN offers two basic modes, which run either as Layer 2 or Layer 3 VPN. Thus, OpenVPN tunnels on Layer 2 can also transport Ethernet frames, IPX packets, and Windows Network Browsing packets (NETBIOS), all of which are problems in most other VPN solutions.

- **Protecting field workers with the internal firewall**: A field worker connected to the central branch of their company with a VPN tunnel can change the network setup on their laptop so that all of their network traffic is sent through the tunnel. Once OpenVPN has established a tunnel, the central firewall in the company's central branch can protect the laptop, even though it is not a local machine. Only one network port must be opened to the local (customers') network by the field worker. The employee is protected by the central firewall whenever he is connected to the VPN. Even better, the administrator of the central VPN server can force the client to use the central firewall by imposing configuration options on the clients.

- **OpenVPN connections can be tunneled through almost every firewall and proxy**: If you have Internet access and can access HTTPS web sites, then OpenVPN tunnels should work. Setups where OpenVPN tunnels are banned are very rare. OpenVPN has full proxy support including authentication.

- **Server and client mode**, **UDP and TCP support**: OpenVPN can be configured to run as a TCP or UDP service and as a server or client. As a server, OpenVPN simply waits until a client requests a connection, whereas a client establishes a connection according to its configuration. A server on the Internet can be completely shut down from any other machine except the ones in its virtual private network, which extends the security level of such systems enormously.

- **Only one port in the firewall must be opened to allow incoming connections**: Since **OpenVPN 2.0,** the special server mode allows multiple incoming connections on the same TCP or UDP port, while still using different configurations for every single connection.

- **No problems with NAT**: Both OpenVPN server and clients can be within a network using only private IP addresses. Every firewall can be used to send the tunnel traffic to the other tunnel endpoint.

- **Virtual interfaces allow flexible very specific networking and almost every imaginable firewall rule**: All restrictions, mechanisms like forwarding, and concepts like **NAT** (**Network Address Translation**) or package mangling (changing the metadata of network datagrams, like some firewalls do) can be used with and within OpenVPN tunnels. Any IP Protocol is possible. Yes, you can tunnel VPNs, like IPsec, inside an OpenVPN tunnel.

- **High flexibility with extensive scripting possibilities**: OpenVPN offers numerous points during connection setup to start individual scripts. These scripts can be used for a great variety of purposes from authentication to failover and more.

- **Transparent**, **high-performance support for dynamic IPs**: By using OpenVPN, there is no longer a need to use expensive, static IPs on either side of the tunnel. Both tunnel endpoints can have cheap DSL access with dynamic IPs. The users will rarely notice a change of IP on either side, Windows Terminal Server and **Secure Shell (SSH)** sessions will only seem to *hang* for few seconds, but they will not terminate and will carry on with the action requested after a short pause. All traffic can be compressed through the LZO library and OpenVPN continuously checks if the compression has been successful. So-called adaptive compression merely 'zips' the uncompressed data to avoid unnecessary overhead.
- **Simple installation on any platform**: Both installation and use are incredibly simple. Especially, if you have tried to set up IPsec connections with different implementations, you will find OpenVPN appealing.
- **Modular Design**: The modular design with a high degree of simplicity both in security and networking is outstanding. No other VPN solution can offer the same options at this level of security.
- **Support for mobile and embedded**: More and more mobile devices are supported. Packages for Windows Mobile and Nokia's Maemo platform, and embedded operating systems like OpenWrt/FreeWrt have all been provided for recently, and there are many others in development.
- **Very active community**: OpenVPN has acquired a huge amount of fans in the last few years. There are installations with high volume users with high availability.

History of OpenVPN

According to an interview on `http://linuxsecurity.com` published in 2003, James Yonan was traveling in Central Asia in the days prior to September 11, 2001 and connecting to his office over Asian or Russian Internet Providers.

The fact that these connections were established over servers in countries with very dubious security made him more and more aware of and concerned about security issues. His research revealed that there were two main streams in VPN technology, one promoting security, and the other usability. None of the solutions available at that time offered an ideal blend of both objectives. IPsec and all of its implementations were difficult to set up, but offered acceptable security. However, its complex structure made it vulnerable to attacks, bugs, and security flaws. Therefore, the networking approach Yonan found in some of the usability camp's solutions seemed to make more sense to him, leading him to a modular networking model using the TUN/TAP virtual networking devices that are provided by the Linux kernel.

> *After some study of the open source VPN field, my conclusion was that the*
> *'usability first' camp had the right ideas about networking and inter-network*
> *tunneling, and the SSH, SSL/TLS, and IPSec camps had the appropriate level of*
> *seriousness toward the deep crypto issues. This was the basic conceptual starting*
> *point for my work on OpenVPN.*

James Yonan in a `LinuxSecurity.com` interview on November 10, 2003.
(`http://www.linuxsecurity.com/content/view/117363/49/`)

Choosing the TUN/TAP devices as a networking model immediately offered a
flexibility that other VPN solutions could not offer. While other SSL/TLS-based
VPN solutions needed a browser to establish connections, OpenVPN would prepare
almost real (but still virtual) network devices, on which almost all networking
activities can be carried out.

Yonan then chose the name OpenVPN with respect to the libraries and programs
of the **OpenSSL** project and because of the clear message that *this is open source and*
free software.

OpenVPN Version 1

OpenVPN entered the scene of VPN solutions on May 13, 2001 with an initial release
that could barely tunnel IP packets over UDP, and could only encrypt with Blowfish
cipher and SHA HMAC signatures (secure encryption and signing methods). This
version was already numbered 0.90, which seemed ambitious, as only one version
(0.91) followed in 2001, offering extended encryption support. For SSL/TLS support,
users would have to wait for almost one year after the first release. Version 1.0
was released in March 2002 and provided SSL/TLS-based authentication and key
exchange. This version was also the first to contain documentation in the form of
a manpage.

Then, OpenVPN development picked up speed. Only five days later, version 1.0.2
was released, which was the first version with added adaptations for RPM-based
systems. From this version onwards, releases were published almost regularly every
four to eight weeks.

The following table gives an overview of the releases and lists the dates and
versions when certain selected features were added to the 1.x version of OpenVPN.
More details can be found in the **Change Log** sections of the OpenVPN website at
`http://openvpn.net/changelog.html` and release notes at `http://openvpn.net/`
`relnotes.html`.

Date	Version	Important features/changes
May 13, 2001	0.90	The initial release, with only a few functions such as IP over UDP, and only one encryption mechanism.
December 26, 2001	0.91	More encryption mechanisms were added.
March 23, 2001	1.0	TLS-based authentication and key exchange were added.
		First manual page was included.
March 28, 2001	1.0.2	Bug fixes and improvements, especially for RPM-based systems like Red Hat, were introduced.
April 9, 2002	1.1.0	Extended support for TLS/SSL.
		Traffic shaping was added.
		First OpenBSD port was included.
		Extended replay protection made OpenVPN more secure.
		Further improvement of documentation (manpage).
April 22, 2002	1.1.1	Options for automatic configuration of an OpenVPN network.
		Inactivity control features were introduced.
May 22, 2002	1.2.0	Configuration file support was added.
		SSL/TLS as background process—longer keys were now possible.
		Various ports were added/improved (Solaris, OpenBSD, Mac OSX, x64).
		Website was improved, 'howto' was included.
		Installation without automake was now possible.
June 12, 2002	1.2.1	Binary RPM files for installation on Red Hat-based systems were provided.
		Major improvements on signal handling and key management on restart.
		Support for dynamical changes in incoming packages (such as dynamic IPs).
		Added support for identity downgrade after installation—OpenVPN can be run as non-privileged user.
July 10, 2002	1.3.0	
		'Housekeeping Releases'—bug fixes, minor improvements, and new features. Works now with OpenSSL 0.9.7 Beta 2.
July 10, 2002	1.3.1	

Date	Version	Important features/changes
October 23,2002	1.3.2	NetBSD port was added
		Support for inetd/xinetd instantiation under Linux.
		Simple building of SSL/TLS certificates was added (easy-rsa script).
		Support for IPv6 over TUN was added.
May 7,2003	1.4.0	Improvement of replay protection (security).
		Numerous bug fixes, improvements, and additions.
May 15,2003	1.4.1	Improved support for kernel 2.4.
July 15,2003	1.4.2	First beginnings of Windows port (but still missing Windows kernel driver).
		Gentoo init script.
August 4,2003	1.4.3	Bug fix was released.
November 20,2003	1.5.0 (and 14 beta versions before that)	Certificate revocation lists.
		TCP support.
		Port to Windows 2000 and XP was added, also included Win32 installer.
		Increased sanity checks in configuration parameters.
		Proxy support was added.
		Extended routing functions (such as redirect gateway).
		Improved TLS support, extended key and cipher features.
May 9,2004	1.6.0 (including 4 release candidates and 7 beta versions)	SOCKS proxy support was added.
		Various improvements on Windows networking behavior—**Dynamic Host Configuration Protocol** (**DHCP**)
		Various bug fixes were introduced.

OpenVPN Version 2

Parallel to the improvement and development of OpenVPN version 1, the test bed for OpenVPN version 2 was created in November 2003. In February 2004, version 2.0-test3 initially prepared the goal for a multi-client server for OpenVPN. This multi-client server is one of the most outstanding features of OpenVPN today. Several clients can connect to the VPN server on the same port. On February 22, 2004, the two development branches, 1.6-beta7 and 2.0-test3, were merged and further development was continued in the branch of version 2.

There were fewer than 29 versions labeled as 'test' versions, 20 beta versions, and 21 release candidates, until on April 17, 2005, OpenVPN version 2.0 was released. This was only possible because of the great number of developers who were contributing to the project, fixing bugs, and improving performance and stability permanently.

The following list will give a brief overview of the new features that were added to OpenVPN version 2:

- **Multi-client support**: OpenVPN offers a special connection mode, where TLS-authenticated clients (that are not blacklisted on the CRL) are provided in DHCP-style with IPs and networking (tunnel) data. This way, several tunnels (up to 128) can communicate over the same TCP or UDP port. Obviously, a mode control switch for activating the server mode became necessary.
- **Push/pul**l **options**: The Network setup of clients can be controlled by the server. After the successful setup of a tunnel, the server can tell the client (both Windows and Linux) to use a different network setup instantaneously.
- A management interface (Telnet) is added.
- The Windows driver and software have been improved extensively.

The current stable version of OpenVPN is version 2.0.9 released on October 1, 2006. There are many reports on the mailing lists that the release candidates of version 2.1 are very stable and usable in enterprise environments also. Use them at your own risk.

The road to version 2.1

Since the middle of 2005, the developers of OpenVPN have been continuously working towards the newest version of OpenVPN, that is, 2.1. At the time of writing (late 2008), the fifteenth release candidate is the most up-to-date version of OpenVPN. The following table shows the improvements that the programmers added along the way:

Date	Version	Important features/changes
June 12, 2005	2.0.1-rc3	Client-side push and pull scripts now support also the dhcp-options 'DOMAIN' and 'DNS'
June 15, 2005	2.0.1-rc4	LZO2 support
July 15, 2005	2.0.1-rc6	The '@'-character can now be used in client-config-directory file names
July 21, 2005	2.0.1-rc7	LZO 2.1 support
August 15, 2005	2.0.1	Various security fixes (DOS attacks against error queues and others), the `auth-retry` option is added to the management interface
August 25, 2005	2.0.2	Source code is removed from the Windows installer, bugs, and security fixes (no changes from 2.0.2-rc1)
September 7, 2005	2.0.2-TO1	New features: `--topology`, `--redirect-gateway bypass-dhcp` for non-local DHCP servers, DHCP client service becomes a dependency on Windows, Plugin interface is extended.
September 23, 2005	2.0.2-TO4	Windows-TAP-Adapter can be opened in non-admin-mode, `--redirect-gateway bypass-dns` when non-local DNS servers option is added.
October 1, 2005	2.1.beta1	Compression (LZO) directive can be pushed to the clients, version merge with 2.0.3-rc1, renaming of several directories (easy-rsa).
October 16, 2005	2.1.beta3	Support for PKCS#11 key format, several patches around certificate handling. --bind option is added.
November 12, 2005	2.1.beta7	Certificates and keys can now be specified inline in the configuration files.
		Multiline parameter listings for plugins. Many fixes and patches.
January 3, 2006	2.1.beta8	`--topology` subnet also for Macintosh systems, automatic proxy detection (Windows only), improved network handling for Windows (both `--ip-win32` and `--route` now have adaptive as standard, on Windows OpenVPN tries to use the IP helper API first, then falls back to the `route` command.

Date	Version	Important features/changes
February 16, 2006	2.1.beta9	Port-sharing allows simultaneous use of a HTTPS-Server and OpenVPN on the same Port. Adds a management client option and a bytecount to the management interface.
1899-12-30	2.1.beta10-16	Numerous Fixes and patches, Topology mode also for BSD, --route-metric option, Mac address for the virtual interfaces can now be specified. Windows installer updates, Easy-RSA-update, PKCS#11 handling updated.
October 31, 2006	2.1_rc1	First release candidate of version 2.1, only minor changes.
February 27, 2007 to April 25, 2007	2.1_rc2-4	First workarounds for common Vista and Windows 64-bit problems, several fixes and patches. Signatures for Installer, OpenVPN GUI as installation option on Windows.
January 23, 2008	2.1_rc5	Improved Windows build system including a custom installer and OpenSSL 0.9.7m. Unprivileged mode for Linux.
June 11, 2008	2.1_rc8	Client package filtering and authentication capability in the management interface. Client-side connection profiles. Windows version contains OpenSSL 0.9.8h. NTLMv2 proxy authentication support. Numerous fixes, patches, and minor feature extensions.
September 10, 2008	2.1_rc10	New server-bridge mode that works like a DHCP-proxy, `-route-gateway dhcp` option makes it possible to use the former default gateway as provided by DHCP. OpenVPN warns in case of address conflicts, and the `--allow-pull-fqdn` directive allows the use of DNS names instead of IPs for several network options. Numerous fixes and patches, extensive security hardening. Scripts exist because rc8 was handled by `execve` and `CreateProcess` instead of `system()` calls. Rc10 provides backwards compatibility.
October 7, 2008	2.1.rc13	Openssl 0.9.8i for the Windows package, Domain sockets for the management interface, Copyright Change to OpenVPN Technologies, Inc.
May 30, 2009	2.1.rc14-18	Several bugfixes, especially for Windows Clients.

In addition to the stable version of 2.1, a commercial version 3.0 is in progress. Perhaps you have noticed that from the beginning of October 2008, the copyright of OpenVPN has changed to OpenVPN Technologies, INC., a company founded by James Yonan. This company is developing several products based on OpenVPN for business setups. Both a version with commercial support and a hardware appliance, services and support, and a web-based management interface shall be available soon. Since 2008, the OpenVPN website's redesign has reflected a professional approach towards business customers.

Networking with OpenVPN

The modular structure of OpenVPN can not only be found in its security model, but also in the networking scheme. James Yonan chose the Universal TUN/TAP driver for the networking layer of OpenVPN.

The TUN/TAP driver is an open source project that is included in all modern Linux/Unix distributions, as well as Windows, Solaris, and Mac OS X. Like SSL/TLS, it is used in many projects, and therefore it is steadily being improved, and new features are being added. Using the TUN/TAP devices takes away a lot of complexity from the structure of OpenVPN. Its simple structure brings increased security when compared to other VPN solutions. Complexity is always the main enemy of security. For example, IPsec has a complex structure with complex modifications in the kernel and the IP stack, thereby creating many possible security loopholes.

The Universal TUN/TAP driver was developed to provide Linux kernel support for tunneling IP traffic. It is a virtual network interface, which appears as authentic to all applications and users. Only the name tunX or tapX distinguishes it from other devices. Every application that is capable of using a network interface can use the tunnel interface. Every technology that you are running in your network can be run on a TUN or TAP interface too.

This driver is one of the main factors that makes OpenVPN very easy to understand, easy to configure, and at the same time, very secure.

The following figure depicts OpenVPN using standard interfaces:
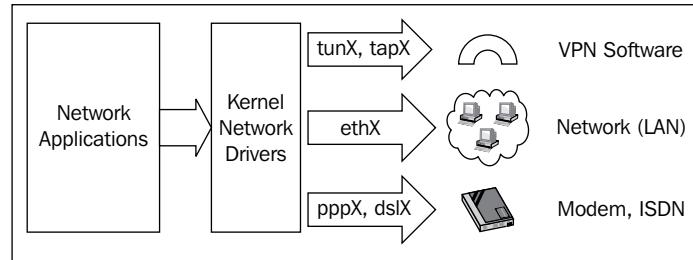
A TUN device can be used like a virtual point-to-point interface, like a modem or DSL link. This is called **routed mode** because routes are set up to the VPN partner.

However, a TAP device can be used like a virtual Ethernet adapter. This enables the daemon listening on the interface to capture Ethernet frames, which is not possible with TUN devices. This mode is called **bridging mode** because the networks are connected as if over a hardware bridge. Applications can read/write to this interface. Software (the tunnel driver) will take all the data and use the cryptographic libraries of SSL/TLS to encrypt them. The data is packaged and sent to the other end of the tunnel. This packaging is done with standardized UDP or optional TCP packets. UDP should be the first choice, but TCP can be helpful in some cases. You are almost completely free to choose the configuration parameters such as protocol or port numbers, as long as both tunnel ends agree on the same figures.
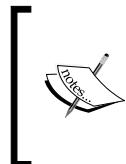
> OpenVPN listens on TUN/TAP devices, takes the traffic, encrypts it, and sends it to the other VPN partner, where another OpenVPN process receives the data, decrypts it, and hands it over to the virtual network device, where the application might already be waiting for the data.

But there's more. As the network interface is a standardized Linux network interface (either TUN or TAP), anything possible on an Ethernet NIC can also be done on VPN tunnels. Consider the following:

- Firewalls can restrict and control traffic
- Traffic shaping is not only possible, but it is also a feature incorporated in OpenVPN

Also, if you want to use DSL lines with frequent reconnects and dynamically assigned IPs, OpenVPN will be your first choice. The reconnect is much faster than that of any other VPN software that we have tested. A Windows terminal server or SSH session does not terminate when one of the VPN partners changes its IP. The session just freezes for a few seconds and then you can continue. Can your VPN accomplish that?

OpenVPN and firewalls

OpenVPN works perfectly with firewalls. There are a few VPN solutions that can claim to have similar firewall support, but none can offer the same level of security.

What is a firewall? There is a famous and simple definition. A firewall is a router that does not route. If you consider this to be not very helpful, then here is a more refined definition:

A **firewall** is a router that routes only selected Internet data. Firewall rules define how to handle specific data and traffic.

Firewalls can be devices or software on PCs, servers, or on other devices. A firewall takes care of the data that has been received and has a closer look at it. Modern firewalls are so-called packet filtering, stateful inspection firewalls. Depending on the OSI layer it is operating in, the firewall can pass decisions based on the data that is found in the headers of the packets or application data. Packet filtering firewalls usually operate by reading the IP data header. Stateful inspection is a mechanism to remember the connection states. In this way, internal networks can be protected from external networks. While Internet connections initiated from the inside can be allowed, all unwanted unauthorized connections from the outside can be rejected. At the same time, incoming data requested by a member of the local net is passed through (because the firewall remembers the state of the request).

Under Linux, most firewalls are based on the program **iptables**. This is a user-space interface to the Linux kernel's netfilter firewall functionality, and offers everything that modern firewalls should. Probably the best way to protect your LAN is by writing a set of iptables rules with a shell script. However, the usability of such a script is not perfect. Most administrators want a **Graphical User Interface** (**GUI**) for firewall control and all the hardware firewalls offer this. Enterprise Distributions, such as RHEL or SLES come with sophisticated firewall tools, but there are also several open source projects. Outstanding tools for this purpose and Linux (iptables) firewalls are as follows:

- The **Shorewall** (**Shoreline Firewall**) project that integrates into the Webmin suite—a web-based frontend to administer Linux systems from a browser. People from the Shorewall project, namely, Simon Matter and Tom Eastep, have written a very useful guideline for the integration of OpenVPN tunnels into Shorewall and more at `http://www.shorewall.net/OPENVPN.html`.

- **IPCop** (`http://www.ipcop.org`) is a promising standalone, easy-to-configure Linux firewall system that is also equipped with a professional GUI. It has had great success in third-world projects like Linux4africa (`http://www.linux4africa.de`) and in other medium-size professional setups. Standardized installation, simple structures, and modular add-ons make this a fast-growing project, and with the help of OpenVPN, the IPCop firewall becomes a true VPN server.

- Tools like Fwbuilder (`http://www.fwbuilder.org`) help you build, manage, and distribute your iptables scripts on your own. Fwbuilder does even more. It can work independently from your platform and is able to translate Linux rules into Cisco, BSD, or other firewall languages. This is really worth a look.

Configuring OpenVPN

Up till now, you have seen that OpenVPN has a secure and easy-to-use security approach and a flexible networking model. Consequently, very simple configuration syntax and good documentation characterize the user interface of OpenVPN. Configuration is done by editing a simple text file. The syntax is the same on every operating system. Here is an example of a simple configuration file with 13 lines.

```
remote feilner-it.dynalias.net
float
dev tun
tun-mtu 1500
ifconfig 10.79.10.1 10.79.10.2
```

```
secret my_secret_key.txt
port 5050
route 10.94.0.0 255.255.0.0 10.79.10.2
comp-lzo
keepalive 120 600
resolv-retry 86400
route-up "/sbin/firewall restart"
log-append /var/log/openvpn/ultrino.log
```

A command-line interface allows you to start temporary tunnels at will, which is very useful when testing setups. The same parameters as in the configuration file are added to the command line, and the tunnels are started.

In the so called server mode, OpenVPN can push various configuration data to the clients through the tunnel. Multiple tunnels can be run on one singular port, either UDP or TCP. OpenVPN can be tunneled through firewalls and proxies, if they allow HTTPS connections, and the server can tell the client to use the tunnel as the default route to the Internet.

This offers a huge variety of possibilities. You can have your field workers open only one port to whatever network they are connected to. This is the port OpenVPN uses to connect to your company's VPN server. Once connected, all Internet traffic from this laptop is routed through the network of the company to which the VPN tunnel is connected. In this way, your company's firewall can also protect the road warriors. A road warrior is a member of a company (or a company's network), who is working outside the company's walls and connects to the network frequently through different connections. A typical road warrior may be a salesman with his or her laptop, who needs to access the company's resources using his or her customer's network.

Problems with OpenVPN

OpenVPN has a few weaknesses:

- It is not IPsec compatible, and IPsec is the standard VPN solution. Lots of devices such as Cisco or Bintec routers use IPsec and can connect to applications of other manufacturers or software IPsec clients. At least they should be able to, because in practice many manufacturers tend to develop their own proprietary extensions to IPsec, which make their implementations practically incompatible with other IPsec devices.

- OpenVPN is not defined by any RFC. But for the future, Yonan has posted several times that RFC 4347 (**DTLS—Datagram Transport Layer Security**) offers a very promising specification with compatible modules taken into account.

- There are still relatively few people who know how to use OpenVPN, especially in difficult scenarios (though those tend to be rare). So if you read on, you can acquire a valuable qualification.
- There is no enterprise class GUI for administration, but there are some promising projects.
- Today you can only connect to other computers. But this is changing, there are companies working on devices with integrated OpenVPN clients.
- OpenVPN runs in user space and all network traffic needs to go from kernel space to user space and back.

As you can see, the main weaknesses of OpenVPN are incompatibility to IPsec and lack of public knowledge about its features and hardware manufacturers. The first will probably never change because the architectures differ too much, but the latter is already changing.

OpenVPN compared to IPsec VPN

Even though IPsec is the de facto standard, there are many arguments for using OpenVPN. If you want to convince your management about why your branches should be connected through OpenVPN instead of IPsec VPN, then the following table can help your argument (points that are preceded by '+' are advantages and points that are preceded by '-' are disadvantages):

IPsec VPN	OpenVPN
+ The standard VPN technology	- Still rather unknown, not compatible with IPsec, perhaps will soon be standardized in part by usage of DTLS.
+ Hardware platforms (devices, appliances)	- Only on computers, but on all operating systems. Exceptions are devices, where embedded Unixs are running such as OpenWrt and similar.
+ Well-known technology	- New technology, still growing and rising.
+ Many GUIs for administration	- No professional GUI, however, there are some interesting and promising projects.
- Complex modification of IP stack	+ Simple technology.
- Critical modification of kernel necessary	+ Standardized network interfaces and packets.
- Administrator privileges are necessary	+ OpenVPN Software can run in user space, and can be chroot-ed.

IPsec VPN	OpenVPN
Different IPsec implementations from different manufacturers can be incompatible	+ Standardized encryption technologies.
- Complex configuration, complex technology	+ Easy well-structured modular technology and easy configuration.
- Steep learning curve for newbies	+ Easy to learn, fast success for newbies.
- Several ports and protocols in firewall is necessary	+ Only one port in firewall is necessary.
- Complex modification of IP stack	+ Simple technology.
- Problems with dynamic addresses on both sides	+ DynDNS works flawlessly, and reconnects faster.
- Security problems with IPsec technologies	+SSL/TLS as industry-standard cryptographic layer.
	+ Traffic shaping.
	+ Speed (up to 20 Mbps on a 1Ghz machine).
	+ Compatibility with firewalls and proxies.
	+ No problems with NAT (both sides can be in NATed networks).
	+ Possibilities for road warriors**.**

Probably the best argument is that you can use both VPN solutions in parallel, as long as you're using Linux or a Linux-based application. Due to the different approaches to networking, there are no conflicts between the two systems. Moreover, you can tunnel IPsec over OpenVPN.

User space versus kernel space

How do the two solutions compare when it comes to speed and latency? **Latency** is a parameter that defines the responsiveness of a line. The less latency, the faster the roundtrip for IP packets, and communication gets faster. A very interesting study by Nejc Skoberne from Ljubliana University (`http://stuff.skoberne.net/ IPSec_and_OpenVPN_Performance.pdf`) demonstrates the expected. OpenVPN runs in user space, whereas IPsec runs in kernel space. That's why many Linux systems need specially patched kernels or add-ons for IPsec. As the technology of this VPN is very close to the network stack and is implemented directly in or at least close to the kernel, it will be faster than OpenVPN in most setups. With any VPN solution running in user space, the operating system's kernel has to perform significantly more context switches than with a technology running in kernel space, which results in a higher latency of the connection. As the OpenVPN server has to do slightly more work than the IPsec variant, its latency will always be a little higher.

Here comes the interesting part. OpenVPN beats IPsec when the number of connected clients rises. Although the author has no explanation for this phenomenon, he documents very well how ten OpenVPN clients are able to draw more bandwidth from the server than a single client or ten IPsec clients. Thus, from the perspective of performance, IPsec may be the better choice, if there are protocols involved that need small latencies such as Samba. However, as the number of clients rises, OpenVPN becomes better and better.

Sources for help and documentation

If you want to learn more about OpenVPN, there are numerous resources on the Internet. Web sites, mailing lists, forums, and private pages of OpenVPN fans can be found in abundance. Google finds more than seven million hits for 'open vpn'. This list of course cannot be complete, but here you will find links to web sites that were helpful to me when I started using OpenVPN and where I still look for help today.

The project community

OpenVPN project has its own web site, including downloads of new versions and updates, documentation, how-to's, mailing lists, and links to various VPN-related pages. The OpenVPN project page could not be bettered. You'll find it at `http://www.openvpn.net`.

The most important source of help is the mailing list: `http://www.openvpn.net/index.php/documentation/miscellaneous/mailing-lists.html`.

As we are using SSL/TLS for encryption purposes, you will certainly want to understand this toolkit. The SSL/TLS Cryptographic library's web site provides detailed documentation and mailing lists, which can be found at `http://www.openssl.org`.

The website of the TLS Charter by the TLS Working Group provides a list with many related RFCs and Internet drafts you might consider helpful at `http://www.ietf.org/html.charters/tls-charter.html`. The promising DTLS RFC 4347 can be found at `http://www.ietf.org/rfc/rfc4347.txt`

The Universal TUN/TAP driver can be downloaded from the following page: `http://vtun.sourceforge.net/tun`. Nevertheless, this should not be necessary, as every modern distribution (and kernel) should have this feature built-in. However, the FAQ of this project may be helpful for various questions.

Documentation in the software packages

If you install OpenVPN from the binary packages for your distribution, then you will have the standard documentation in the following directories:

Distribution	Path to documentation
Debian	`/usr/share/doc/openvpn`
SuSE	`/usr/share/doc/packages/openvpn`
Red Hat	`/usr/share/doc/openvpn-2.x`
Windows	online documentation only

Other distributions may have different locations. Check your package management system for details. RPM-based systems give a list of all the files belonging to a specific package when you type `rpm -ql openvpn` as the super user. Debian-based systems (like Ubuntu) should give the same information when root enters `dpkg -L openvpn`. Other systems that use ipkg as a manager will respond to `ipkg files openvpn`. Replace `openvpn` with the name of the package you had installed.

The source code package (tarball) contains several READMEs and documentation files. Just browse through the directories from which you had extracted OpenVPN. If you're interested, have a look at some of the source code files. The developer's comments can be of great help to understand the depths of the software.

Summary

OpenVPN offers great possibilities. The networking concept allows very transparent setups with firewalls or in road warrior configurations. James Yonan, the founder, has made very good decisions when trusting the TUN/TAP network drivers and the SSL/TLS libraries. OpenVPN was first published in 2001, version 2 came out in 2005 and offers many more advanced features than the versions before. The current stable version is 2.0.9, but there are already 15 release candidates for 2.1, and a commercial version 3 is on the way. Multi-client support, Vista support, the push/pull options, a management interface, and the advanced port sharing are only some of the features. OpenVPN is easy to configure and has only a few weaknesses, the most serious of which is its incompatibility with IPsec by design. But to name this as a weakness is harsh if it is compared to IPsec, as was done earlier in this chapter. IPsec is still the standard, but OpenVPN has many more features at a much better security level. So now let's install our first tunnel.

4

Installing OpenVPN on Windows and Mac

Installing OpenVPN is easy and platform independent. In this chapter, we will see how to install it on Windows Server, Vista, and Mac OS X. For both operating systems, there are software packages available that can be downloaded very easily from the relevant web sites.

Obtaining the software

There are only a few prerequisites that have to be met if you want to install OpenVPN on Windows, if you are running versions later than Windows 2000. Mac OS X is required on Apple platforms. Installation of OpenVPN can be done in one of the following ways:

- For Microsoft Windows operating systems, you have to download the binary `.exe` file from `http://openvpn.net/index.php/open-source/downloads.html` or the package containing a graphical user interface from `http://openvpn.se/`. Those who dare to use the release candidate of version 2.1, or a forthcoming version 2.1 will find that the Windows GUI is already integrated (since OpenVPN 2.1rc13 from October 2008).

- On Macintosh systems running Mac OS X, there is a graphical installation wizard and management tool called **Tunnelblick**.

Note that OpenVPN versions that are not tagged as *stable* should never be used in the production environment. There may be security issues and bugs that cause the code to crash or open your complete network to intruders. The stable versions have been tested for stability and security flaws, and will not be published as stable until they meet the developer team's requirements.

Installing OpenVPN on Windows

If you want to install OpenVPN on Windows, you have to make a choice before downloading. You can install the original OpenVPN software from a link such as `http://www.openvpn.net/release/openvpn-2.0.9-install.exe` (this is still my preferred suggestion) or install the OpenVPN GUI from `http://openvpn.se/files/install_packages/openvpn-2.0.9-gui-1.0.3-install.exe`. This package contains the OpenVPN software plus a GUI to bring up or close down tunnels. Especially, if you set up an OpenVPN client—be it a laptop or desktop PC for a home worker, which is only connecting temporarily to your VPN—the Windows user will want to have an easy-to-use, clickable interface. However, if you do not want the users to interact with the VPN tunnels, then the original OpenVPN software will do, and, as mentioned, beginning with release candidate 13 of version 2.1, the GUI is integrated.

OpenVPN can be made to run as a service on the Windows PC, which means it is started automatically on startup. It can be configured to enable the tunnel automatically or forced by a click of a mouse. The installation is pretty straightforward and should not pose any problem to the experienced Windows user. The following sections show you a guided installation process. If you are prompted that the driver has not passed Windows Logo testing, click on **Install**.


Downloading and starting installation

Download the newest version of the OpenVPN GUI from `http://openvpn.se/` to your local drive. Log in as the administrator or a privileged user, and double-click on the downloaded file to start the **Setup Wizard**. If you are using a desktop firewall, you will be prompted to allow OpenVPN to be installed and connected to the Internet later.

The OpenVPN GUI installation wizard, probably the most convenient way to install OpenVPN on Windows, is started. Click on **Next** to proceed and agree to the terms of the license agreement (**I Agree**). Even though OpenVPN and the OpenVPN GUI are freely available under the open source **General Public License** (**GPL**), you still have to accept a license agreement. You should read the license to make sure that your planned use of OpenVPN conforms to it. Click on **I Agree** to proceed.

Selecting the components and location

The next dialog window offers a choice on the top of OpenVPN components that you may want to install. The standard selection of components change makes sense to is suitable for most cases.
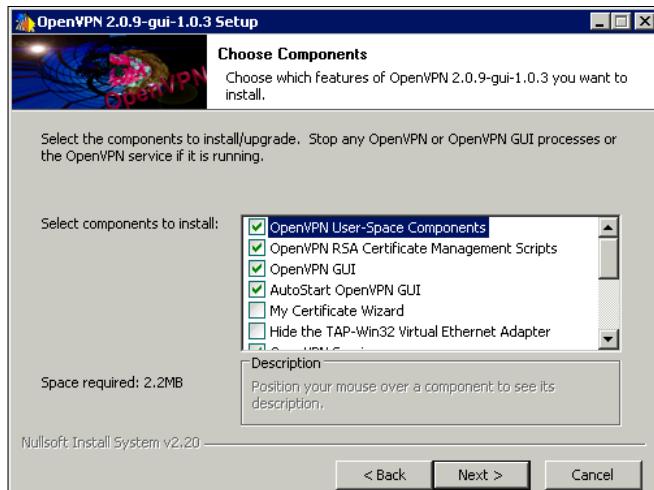
In this dialog, you have several options to choose from. Even if you normally don't need to make changes here, the following table gives you an overview of the entries and when you should install which feature. The **Client Install** is a system that only connects to another OpenVPN system, whereas the **Server Install** is an OpenVPN system that allows incoming connections.

Option	Feature	Client Install	Server Install
OpenVPN User-Space Components	The OpenVPN program	x	x
OpenVPN RSA Certificate Management Scripts	easy-rsa for Windows		x
OpenVPN GUI	The graphical user interface	x	
AutoStart OpenVPN GUI	Link for auto start	x	
My Certificate Wizard	Certificate requests for a certificate authority	x	
Hide the TAP-Win32 VEA	Interface is not shown in network setup		
OpenVPN Service	Configure OpenVPN as a service		x
OpenVPN File Associations	Configuration files (`*.ovpn`) are associated with OpenVPN	x	x
OpenSSL DLLs	Dynamic link libraries	x	x
OpenSSL Utilities	Various Programs for OpenSSL	x	x
TAP-WIN32 Virtual Ethernet Adapter	Virtual network interface	x	x
Add OpenVPN to PATH	`Openvpn.exe` is in the path of every user's command line	x	x
Add Shortcuts to Start Menu	Shortcut to the start menu	x	x

As you can see, the only differences are the RSA management and the option to run OpenVPN as a service. Both can be configured using different methods later, such as the configuration file, the Windows system management, or software like **xca** that we will use to generate and administer certificates.

Press **Next** to continue installation and choose the path that you want to install OpenVPN to. This normally defaults to `C:\Program Files\OpenVPN`, and there are usually very few reasons to change that. Click on **Install** to confirm.

Finishing installation

While OpenVPN is installing, you can read its output in the installation window and follow the creation of folders, files, and shortcuts and the installation of drivers (TAP) for networking.
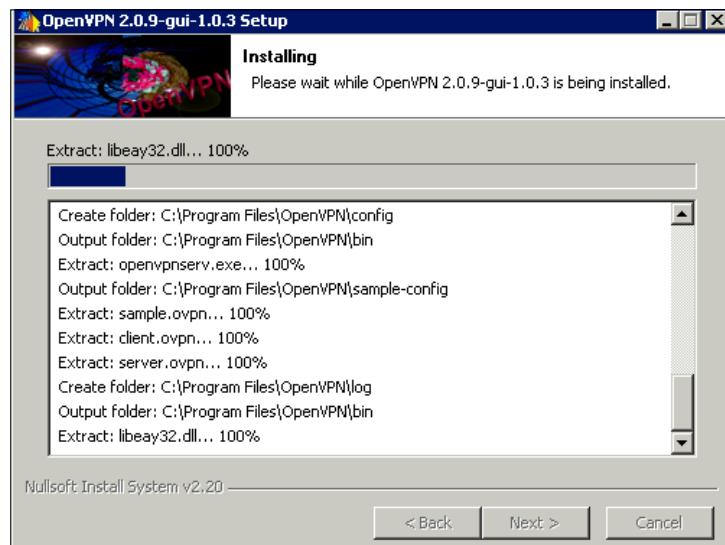
Recent Windows systems will warn you about the TUN/TAP driver that is about to be installed. As Microsoft can't validate the origin of the driver, its security subsystem warns you with the following dialog (Windows Server 2008):
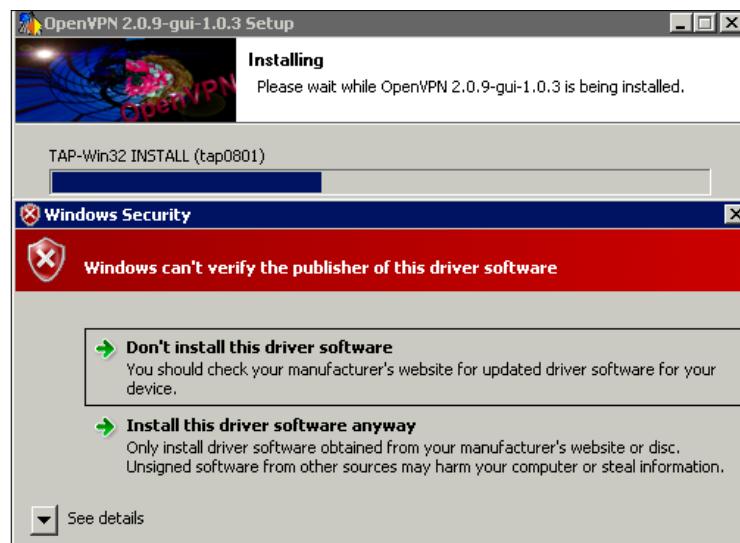
Click on **Install this driver software anyway** and see the OpenVPN installer complete the installation. If you've made it so far, you have successfully installed OpenVPN on your Windows system. If you want to read the Readme file, then activate the checkbox **Show Readme** before you click on **Finish**.
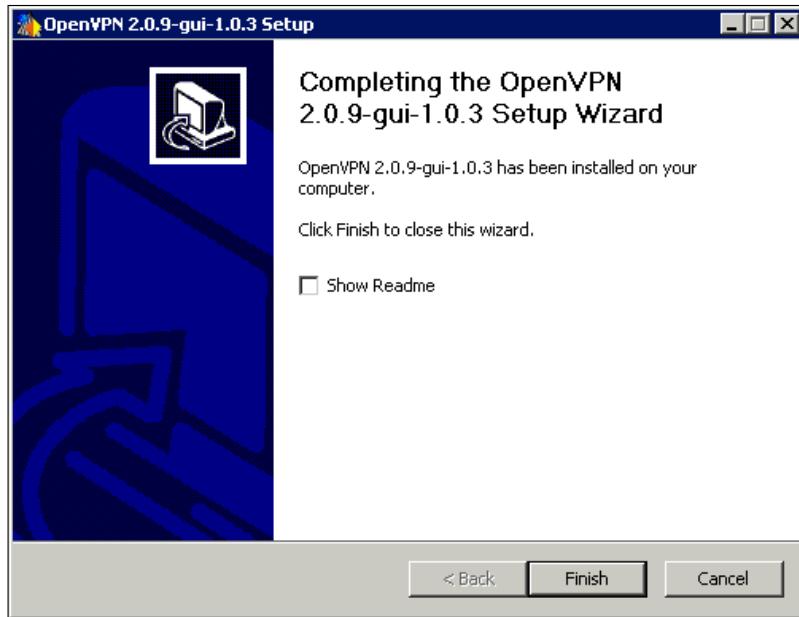

# Testing the installation—a first look at the panel applet

After the installation of OpenVPN GUI, OpenVPN is started and a panel applet is created. In the following screenshot, it is the icon close to the left, with the two red computer screens connected with a globe. Although this image is from Windows Server 2008, it looks pretty much the same on other Windows systems:

The applet provides a convenient method for Windows users to control and configure (partly) OpenVPN. However, as there is no interface for configuration yet, the configuration file can only be edited by using an editor, and until a first configuration is created, the context menu may look rather poor. Right-click on the panel applet as shown in the following screenshot:
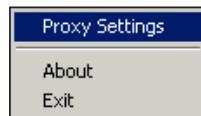
Later, when we have configured the first connection, this menu will be populated with new entries. With entries showing, such as **Connect** and **Disconnect**, you can start and stop the configured tunnels. At the same time, the start menu is populated with new entries. Consider the following screenshot:
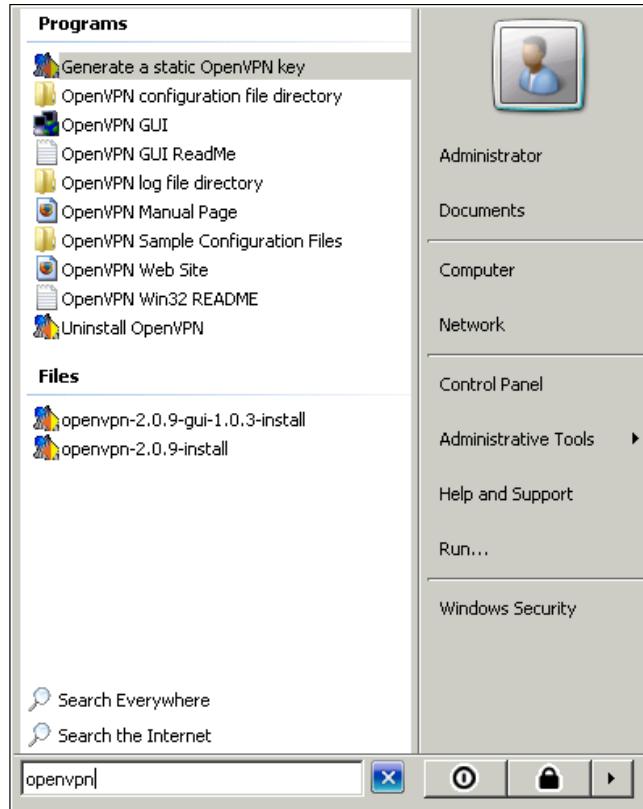
The rest of the configuration setup is done within a text editor like Notepad. The following image shows the file `C:\Windows\Program Files\openVPN\ sample-config\client`:

Installing OpenVPN on Mac OS X (Tunnelblick)

Of course there is also OpenVPN software for Mac OS X. Its name is Tunnelblick and it is free open source software, released under the GPLv2 license, and it contains a graphical installation wizard. You can download it from `http://code.google.com/p/tunnelblick/`. It comes as a disk image file (`.dmg`), including the command-line application (by the OpenVPN project) and the Tunnelblick GUI for Macintosh computers. It works on all Mac OS X later than Tiger (10.4).

If you need more detailed information on installing and uninstalling Tunnelblick, the online readme `http://www.tunnelblick.net/README.txt` file is the best place to look. It contains a full list of the files that are installed on your system. For version 3.0, these files are as follows:

```
/System/Library/Extensions/tap.kext
/System/Library/Extensions/tun.kext
/System/Library/StartupItems/tap
/System/Library/StartupItems/tun
/usr/local/sbin/openvpn
/usr/local/sbin/openvpnstop
/usr/local/sbin/openvpnstart
/Applications/Tunnelblick.app
```

To uninstall Tunnelblick from your system, you just need to delete these files and reboot your machine.

But before that, let's install Tunnelblick. The installation is started simply by double-clicking on the file `Tunnelblick-Complete.mpkg` to start the installation wizard.
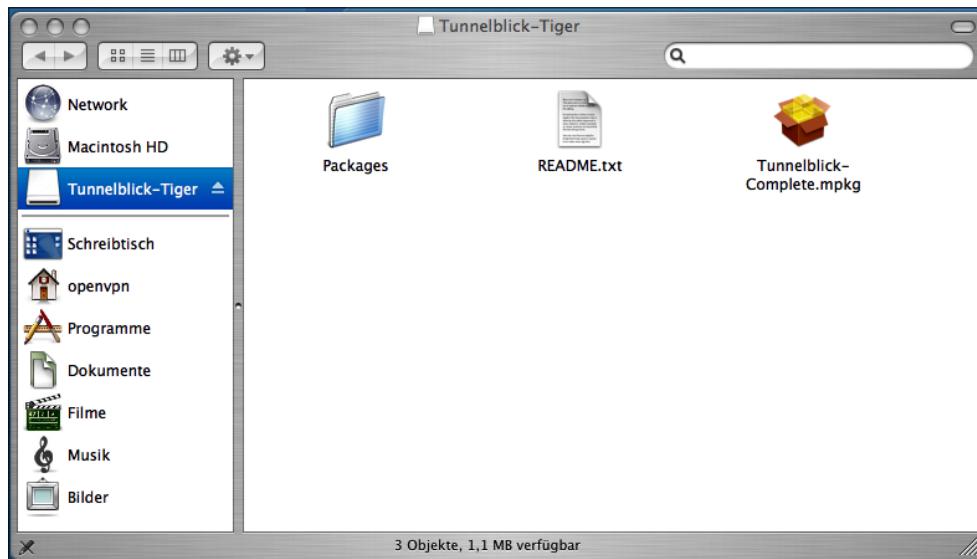
An installation wizard will guide you through five steps. Simply choose the installation location and type and the wizard will solve all your questions. The file `README.txt` contains information on installing, uninstalling, and the configuration of OpenVPN with special regards to Macintosh and OS X 10.3 or later.

Testing the installation—the Tunnelblick panel applet

After installation, you will find the Tunnelblick icon in the system tray of your panel.

If you select the menu entry **Edit Config File...**, then you will be presented with the standard configuration file in a text editor, as shown in the following screenshot:

If you need more information on OpenVPN on Macintosh, the following links are good places to visit:

- Detailed installation instructions for Mac OS X 10.3: `http://www.helsinki.fi/atk/english/hy-ppp/hy-vpn/hy-vpn-mac.html`

- Homepage of the Tunnelblick OpenVPN wrtGUI for Macintosh: `http://www.tunnelblick.net/`

Summary

This chapter showed that installation of OpenVPN is very easy on both Windows and Mac. There are working and very reliable software packages , and installation GUIs for both platforms.

5

Installing OpenVPN on Linux and Unix Systems

Installing OpenVPN is easy and platform independent. In this chapter, we will install it on different Linux versions and FreeBSD.

Prerequisites

All Linux/Unix systems must meet the following requirements to install OpenVPN successfully:

- Your system must provide support for the Universal TUN/TAP driver. The kernels newer than version 2.4 of almost all modern Linux distributions provide support for TUN/TAP devices. Only if you are using an old distribution or if you have built your own kernel, will you have to add this support to your configuration. The section of this chapter, *Enabling Linux kernel support for TUN/TAP devices*, deals with this problem. This project's web site can be found at `http://vtun.sourceforge.net/tun/`.

- OpenSSL libraries have to be installed on your system. I have never encountered any modern Linux/Unix system that does not meet this requirement. However, if you want to compile OpenVPN from source code, the SSL development package may be necessary. The web site is: `http://www.openssl.org/`.

- The **Lempel-Ziv-Oberhumer** (**LZO**) Compression library has to be installed. Again, most modern Linux/Unix systems provide these packages, so there shouldn't be any problem. LZO is a real-time compression library that is used by OpenVPN to compress data before sending. Packages can be found on `http://openvpn.net/download.html`, and the web site of this project is `http://www.oberhumer.com/opensource/lzo/`.

- Most Linux/Unix systems' installation tools are able to resolve these so-called dependencies on their own, but it might be helpful to know where to get the required software.

- Most commercial Linux systems, like SuSE, provide installation tools, like **Yet another Setup Tool** (**YaST**), and contain up-to-date versions of OpenVPN on their installation media (CD or DVD). Furthermore, systems based on RPM software can also install and manage OpenVPN software at the command line.

- Linux systems, like Debian, use sophisticated package management tools that can install software that is provided by repositories on web servers. No local media is needed, the package management will resolve potential dependencies by itself, and install the newest and safest possible version of OpenVPN.

- FreeBSD and other BSD-style systems use their package management tools such as `pkg_add` or the ports system.

- Like all open source projects, OpenVPN source code is available for download. These compressed `tar.gz` or `tar.bz2` archives can be downloaded from `http://openvpn.net/download.html` and unpacked to a local directory. This source code has to be configured and translated (compiled) for your operating system.

- You can also install unstable, developer, or older versions of OpenVPN from `http://openvpn.net/download.html`. This may be interesting if you want to test new features of forthcoming versions.

- Daily (unstable!) OpenVPN source code extracts can be obtained from `http://sourceforge.net/cvs/?group_id=48978`. Here you find the **Concurrent Versions System** (**CVS**) repository, where all OpenVPN developers post their changes to the project files.

Installing OpenVPN on SuSE Linux

Installing OpenVPN on SuSE Linux is almost as easy as installing under Windows or Mac OS X. Linux users may consider it even easier. On SuSE Linux almost all administrative tasks can be carried out using the administration interface YaST. OpenVPN can be installed completely using this. The people distributing SuSE have always tried to include up-to-date software in their distribution. Thus, the installation media of OpenSuSE 11 already contains version 2.0.9 of OpenVPN, and both the Enterprise editions SLES 10 and the forthcoming SLES 11 that offer five years of support. Updates include up-to-date versions of OpenVPN. Both OpenSuSE and SLES use YaST for installing software.

Using YaST to install software

Start YaST. Under both GNOME and the **K Desktop Environment** (**KDE**—the standard desktop under SuSE Linux), you will find YaST in the main menu under **System | YaST**, or as an icon on the Desktop. If you are logged in as a normal user, you will be prompted to enter your root password and confirm the same. The YaST control center is started.

This administration interface consists of many different modules, which are represented by symbols in the right half of the window and grouped by the labels on the left.
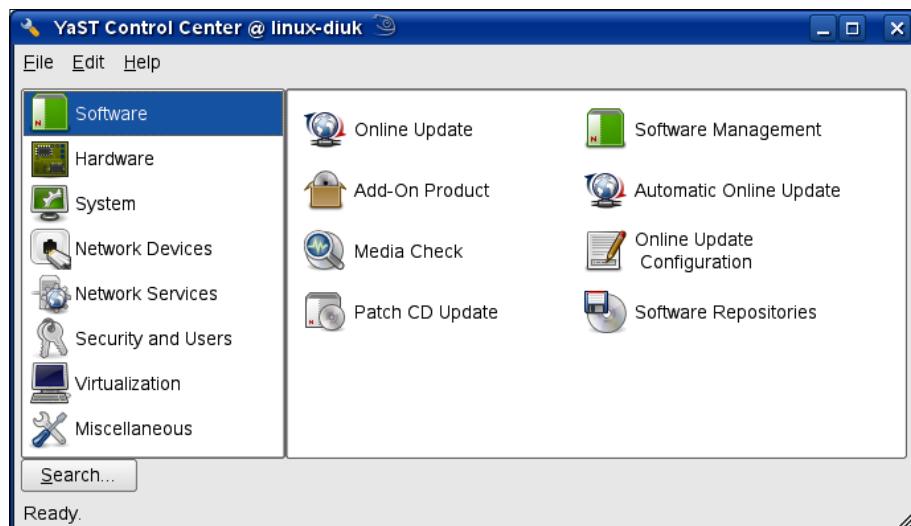
After starting YaST, click on the symbol labeled **Software Management** in the right column to start the software management interface of YaST.

The software management tool in YaST is very powerful. Under SuSE, data about the installed and installable software is kept in a database, which can be searched very easily. Select the entry **Search** in the drop-down list **Filter:** and enter **openvpn** in the **Search** field.

YaST will find at least one entry that matches your search value **openvpn.** Depending on the (online) installation sources that you have configured, various add-ons and tools for OpenVPN will be found. If you chose to add the community repositories like I did on this system, then OpenSuSE will list more than 10 hits.
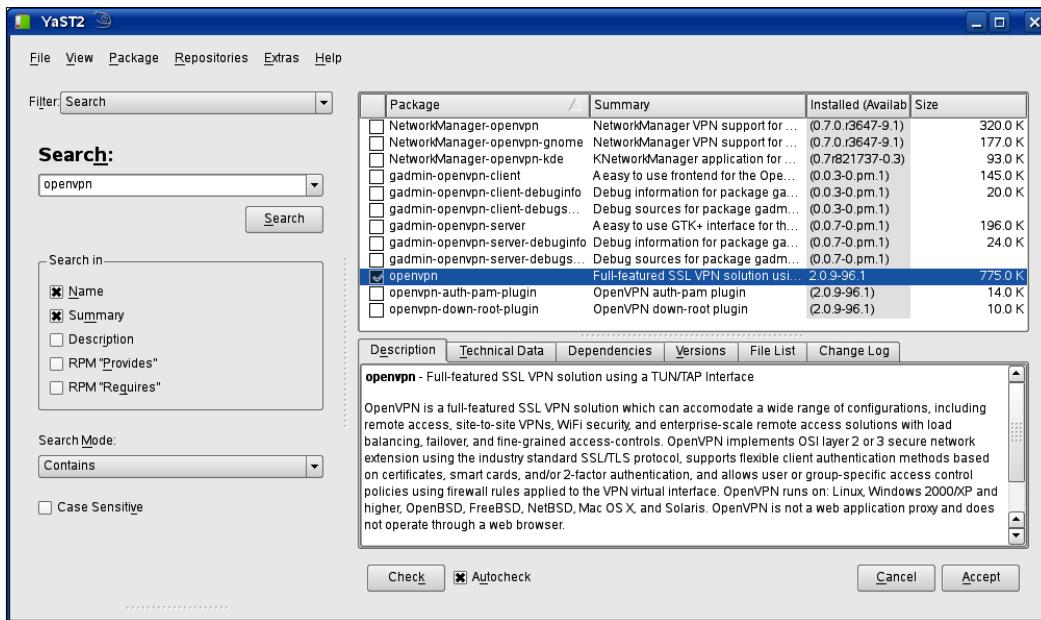
Select the entry **openvpn** by checking the box besides the entry in the first column. If you want to obtain information about the OpenVPN package, have a look at the lower half of the right side—here you will find the software **Description**, **Technical Data**, **Dependencies**, and more information about the package that you have selected. Click on the **Accept** button to start the OpenVPN installation.

If you installed from a local medium, then put your CD or DVD in your local drive now. YaST will retrieve the OpenVPN files from your installation media. If you have configured your system to use one of the web/FTP servers of SuSE for installation, then this might take a while. The files are unpacked and installed on your system, and YaST updates the configuration. This is managed by the script SuSEconfig and other scripts that are called by it.

SuSEconfig and YaST were once very infamous for deleting local configuration created by the local administrator or omitting relevant changes. This problem only occurred when updating and re-installing software that was previously installed. However, the latest SuSE versions have proven very reliable, and the system configuration tools never delete configuration files that you have added manually. Instead, the standard configuration files installed with the new software package may be renamed to `<file>.rpmnew` or similar, and your configuration is loaded.

During installation, SuSEconfig calls several helper scripts and updates your configuration, and informs you of the progress in a separate window. After successful software installation, you are prompted if you want to install more packages or exit the installation. Click on the **Finish** button.

The Novell/OpenSuSE teams have added a very handy tool called `zypper` to their package management. From version 10.1 onwards, you can simply install software from a root console by typing `zypper in openvpn`. Of course this only works if you know the exact name of the package that you want to install. If not, then you will have to search for it, for example, by using `zypper search vpn`.

```
opensuse11:~ # zypper search openvpn
Reading installed packages...

S | Name                          | Summary                        | Type
--+-------------------------------+--------------------------------+--------
  | NetworkManager-openvpn        | NetworkManager VPN support f-> | package
  | NetworkManager-openvpn-gnome  | NetworkManager VPN support f-> | package
  | NetworkManager-openvpn-kde    | KNetworkManager application -> | package
  | gadmin-openvpn-client         | A easy to use frontend for t-> | package
  | gadmin-openvpn-client-debuginfo | Debug information for packag-> | package
  | gadmin-openvpn-client-debugsource | Debug sources for package ga-> | package
  | gadmin-openvpn-server         | A easy to use GTK+ interface-> | package
  | gadmin-openvpn-server-debuginfo | Debug information for packag-> | package
  | gadmin-openvpn-server-debugsource | Debug sources for package ga-> | package
i | openvpn                       | Full-featured SSL VPN soluti-> | package
  | openvpn-auth-pam-plugin       | OpenVPN auth-pam plugin         | package
  | openvpn-down-root-plugin      | OpenVPN down-root plugin        | package
opensuse11:~ #
```

Installing OpenVPN on Red Hat Fedora using yum

If you are using Red Hat Fedora, the **Yellow dog Updater, Modified** (**yum**) is probably the easiest way to install software. It can be found on `http://linux.duke.edu/projects/yum/`, and provides many interesting features, such as automatic updates, solving dependency problems, and managing installation of software packages.

Even though OpenVPN installation on Fedora can only be done on the command line, it still is a very easy task. The installation makes use of the commands `wget`, `rpm`, and `yum`.

- `wget`: A command-line download manager suitable for `ftp` or `http` downloads.
- `rpm`: The Red Hat Package Manager is a software management system used by distributions like SuSE or Red Hat. It keeps track of changes and can solve dependencies between programs.
- `yum`: This provides a simple installation program for RPM-based software.

To use `yum`, you have to adapt its configuration file as follows:

- Log in as administrator (root).
- Change to Fedora's configuration directory `/etc`.
- Save the old, probably original, configurations file `yum.conf` by renaming or moving it. You can use commands such as `mv yum.conf yum.conf_fedora_org` to accomplish this.
- The web site `http://www.fedorafaq.org/` provides a suitable configuration file for `yum`. Download the file `http://www.fedorafaq.org/samples/yum.conf` using `wget`. The command-line syntax is:

  `wget http://www.fedorafaq.org/samples/yum.conf`

- At the same web site a sophisticated `yum` configuration is available for downloading. Install this as well:

  `rpm -Uvh http://www.fedorafaq.org/yum`

The following excerpt shows the output of these five steps on the system:

```
[root@fedora ~]# cd /etc
[root@fedora etc]# mv yum.conf yum.conf.org
[root@fedora etc]# wget http://www.fedorafaq.org/samples/yum.conf
--11:33:25--  http://www.fedorafaq.org/samples/yum.conf
```

```
            => `yum.conf'
Resolving www.fedorafaq.org... 70.84.209.18
Connecting to www.fedorafaq.org[70.84.209.18]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 595 [text/plain]

100%[================================================================
===============================================>] 595         --.-
-K/s

11:33:25 (405.20 KB/s) - `yum.conf' saved [595/595]

[root@fedora etc]# rpm -Uvh http://www.fedorafaq.org/yum
Retrieving http://www.fedorafaq.org/yum
Preparing...                   #######################################
## [100%]
   1:yum-fedorafaq             #######################################
## [100%]
[root@fedora etc]#
```

The rest of the OpenVPN installation is very simple. Just enter yum install openvpn
in your root shell. Now yum will start and give you a lot of output. We will have a
short look at the things yum does.

```
[root@fedora ~]#yum install openvpn
Setting up Install Process
Setting up repositories
livna                    100% |=========================|  951 B
00:00
updates-released         100% |=========================|  951 B
00:00
base                     100% |=========================| 1.1 kB
00:00
extras                   100% |=========================| 1.1 kB
00:00
Reading repository metadata in from local files
primary.xml.gz           100% |=========================| 127 kB
00:00
livna     : ############################################# 380/380
Added 380 new packages, deleted 0 old in 1.36 seconds
primary.xml.gz           100% |=========================| 371 kB
00:00
updates-re: #############################################
1053/1053
Added 0 new packages, deleted 13 old in 0.93 seconds
```

`yum` has set up the installation process and integrated online repositories for the installation of software. This feature is the reason why Fedora does not need a URL source for installing OpenVPN. The repository metadata contains information about location, availability, and dependencies between packages. Resolving the dependencies is the next step.

```
Parsing package install arguments
Resolving Dependencies
--> Populating transaction set with selected packages. Please wait.
---> Downloading header for openvpn to pack into transaction set.
openvpn-2.0.9-1.fc5.i386. 100% |=========================|  18 kB
00:00
---> Package openvpn.i386 0:2.0.9-1.fc5 set to be updated
--> Running transaction check
--> Processing Dependency: liblzo.so.1 for package: openvpn
--> Restarting Dependency Resolution with new changes.
--> Populating transaction set with selected packages. Please wait.
---> Downloading header for lzo to pack into transaction set.
lzo-1.08-4.i386.rpm         100% |=========================| 3.2 kB
00:00
---> Package lzo.i386 0:1.08-4 set to be updated
--> Running transaction check

Dependencies Resolved
```

OpenVPN needs the LZO library for installation, and `yum` is about to resolve this dependency. As a next step, `yum` tests whether this library has unresolved dependencies. If this is not the case, we are presented with an overview of the packages to be installed. Confirm by entering **y** and press the *Enter* key. `yum` will start downloading the required packages.

If the RPM process that `yum` is using to install the software packages encounters a missing encryption key, then confirm the import of this key from `http://www.fedoraproject.org` by entering **y** and pressing the *Enter* key. This GPG key is used to control the authenticity of the packages selected for installation.

```
Key imported successfully
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing: lzo                          #########################
[1/2]
  Installing: openvpn                      #########################
[2/2]

Installed: openvpn.i386 0:2.0.9-1.fc5
```

```
Dependency Installed: lzo.i386 0:1.08-4
Complete!
[root@fedora etc]#
```

That's all! `yum` has been downloaded, checked, and has installed OpenVPN and the LZO libraries.

Installing OpenVPN on Red Hat Enterprise Linux

SuSE Linux Enterprise Server's competitor, **Red Hat Enterprise Linux** (**RHEL**), comes without a central management tool like YaST. Nevertheless, Red Hat has proven to be the system of choice for many setups with high expectations. Due to its more conservative approach, the OpenVPN package for RHEL 5 is not in the standard repositories. As administrator, you have to download the descriptions for the Fedora **EPEL** (**Extra Packages for Enterprise Linux**) repositories before you type `yum install openvpn`.

```
[root@rhel ~]# rpm -ivh http://download.fedora.redhat.com/pub/epel/5/
x86_64/epel-release-5-3.noarch.rpm
Empfange http://download.fedora.redhat.com/pub/epel/5/x86_64/epel-
release-5-3.noarch.rpm
Warnung: /var/tmp/rpm-xfer.IFMGoW: Header V3 DSA-Signatur: NOKEY, key
ID 217521f6
Vorbereiten...                 #########################################
## [100%]
   1:epel-release               #########################################
## [100%]
[root@rhel ~]# yum install openvpn
Loading "rhnplugin" plugin
Loading "security" plugin
epel                      100% |=========================| 2.1 kB
00:00
primary.sqlite.bz2        100% |=========================| 1.9 MB
00:02
rhel-i386-server-vt-5     100% |=========================| 1.4 kB
00:00
rhel-i386-server-5        100% |=========================| 1.4 kB
00:00
rhn-tools-rhel-i386-serve 100% |=========================| 1.2 kB
00:00
Setting up Install Process
```

```
Parsing package install arguments
Resolving Dependencies
--> Running transaction check
---> Package openvpn.i386 0:2.1-0.27.rc9.el5 set to be updated
--> Processing Dependency: liblzo2.so.2 for package: openvpn
--> Running transaction check
---> Package lzo.i386 0:2.02-2.el5.1 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

========================================================================
========
 Package                 Arch        Version          Repository
Size
========================================================================
========
Installing:
 openvpn                 i386        2.1-0.27.rc9.el5  epel
353 k
Installing for dependencies:
 lzo                     i386        2.02-2.el5.1     epel
63 k

Transaction Summary
========================================================================
========
Install     2 Package(s)
Update      0 Package(s)
Remove      0 Package(s)

Total download size: 416 k
Is this ok [y/N]:y
Downloading Packages:
(1/2): openvpn-2.1-0.27.r 100% |=========================| 353 kB
00:00
(2/2): lzo-2.02-2.el5.1.i 100% |=========================|  63 kB
00:00
warning: rpmts_HdrFromFdno: Header V3 DSA signature: NOKEY, key ID
217521f6
Importing GPG key 0x217521F6 "Fedora EPEL <epel@fedoraproject.org>"
from
/etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL
Is this ok [y/N]: y
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
```

```
Running Transaction
  Installing: lzo                          #########################
[1/2]
  Installing: openvpn                      #########################
[2/2]

Installed: openvpn.i386 0:2.1-0.27.rc9.el5
Dependency Installed: lzo.i386 0:2.02-2.el5.1
Complete!
[root@rhel ~]#
```

As you can see, the EPEL repositories provided the current (but officially still
unstable) Release Candidate 9 of OpenVPN (Version 2.1). yum also installed the
necessary LZO libraries.

Installing OpenVPN on RPM-based systems

On both SuSE and Fedora, there is another possible way to install OpenVPN.
The command-line interface rpm is available on all systems that are using the
Red Hat package management system. rpm is a very powerful command that can
install, remove, update, test, and query software packages. Installing software with
rpm is carried out using the following three steps:

1. Downloading the software.
2. Testing installation and resolving dependencies.
3. Installing the RPM files with the appropriate rpm command.

Whenever you run into problems with RPM, its man page is the best reference for
all of its abundant options.

The best place to look for the right version of OpenVPN under SuSE is
ftp://ftp.suse.com/. Fedora RPMs can be obtained from Dag Wieers' web site
http://dag.wieers.com/packages/openvpn/. The command-line extract in the
following section shows the typical process of obtaining and installing OpenVPN
on OpenSuSE 11, but this procedure will work in exactly the same way on Fedora
or any other RPM-based system.

Using wget to download OpenVPN RPMs

Enter `wget 'ftp://ftp.suse.com/pub/suse/i386/9.3/suse/i586/openvpn-2.0-5.i586.rpm'` on your SuSE system to download OpenVPN in version 2.0.5.

One of the most interesting features of the Red Hat Package Manager, `rpm`, is that it allows for a dry run to test the installation. This is done with the following command:

`rpm -ivh --test openvpn-2.0.9-96.1.i586.rpm`

The options are simple:

- `-i` stands for install
- `-v` means verbose output
- `-h` prints a progress bar
- `--test` lets `rpm` do a dry run to test the installation of the package

In almost all the cases, you will receive the following output:

```
opensuse11:~ # rpm -ivh --test openvpn-2.0-9.-96.1.i586.rpm
Preparing...                ###########################################
## [100%]
opensuse11:~ #
```

OK, `rpm` reports no errors, so we can install OpenVPN without the following test switch:

`opensuse11:~ # rpm -ivh --test openvpn-2.0-9.-96.1.i586.rpm`

Installing OpenVPN and the LZO library with wget and RPM

If your system is still missing the LZO library, our test installation will fail. `rpm` reports an error, already pointing you to the solution. We have to download the RPM and install it. Again, `wget` is a good choice for this issue.

```
opensuse11:~ # wget 'ftp://sunsite.informatik.rwth-aachen.de/pub/
linux/opensuse/distribution/11.0/repo/oss/suse/i586/lzo-2.02-
113.1.i586.rpm'
```

A good idea may be creating a local directory and downloading both RPM files to this directory and install them both in one go:

```
opensuse11:~ # mkdir openvpn-rpms
opensuse11:~ # cd openvpn-rpms
opensuse11:~ # wget 'ftp://sunsite.informatik.rwth-aachen.de/pub/
linux/opensuse/distribution/11.0/repo/oss/suse/i586/lzo-2.02-
113.1.i586.rpm'
(...)
opensuse11:~/openvpn-rpms # wget 'ftp://sunsite.informatik.rwth-
aachen.de/pub/linux/opensuse/distribution/11.0/repo/oss/suse/i586/
openvpn-2.0.9-96.1.i586.rpm'
(...)
opensuse11:~/openvpn-rpms # rpm -ivh *rpm
Preparing...                  #########################################
## [100%]
   1:openvpn                  #########################################
## [ 50%]
   2:lzo                      #########################################
## [100%]
opensuse11:~/openvpn-rpms #
```

As the last command shows, you can call RPM with wildcards and instruct it to install all the RPM files it finds in this directory at once.

RPM can also use a remote location for the package to be installed, but this only works if there are no dependencies. As this can only be checked after download, you may have to try several times. This is why `wget` is the better choice in most cases.

```
opensuse11:~ # rpm -Uvh 'ftp://sunsite.informatik.rwth-aachen.de/pub/
linux/opensuse/distribution/11.0/repo/oss/suse/i586/openvpn-2.0.9-
96.1.i586.rpm'
```

Using rpm to obtain information on the installed OpenVPN version

You can use `rpm` to query the software database by adding options that begin with `-q` to the command.

Whereas `rpm -qi` provides information about the installed version, `rpm -qli` will print all files that have been installed by this software package, including their full path.

```
[root@fedora ~]# rpm -ql openvpn
/etc/openvpn
/etc/rc.d/init.d/openvpn
/usr/lib/openvpn
/usr/lib/openvpn/plugin
/usr/lib/openvpn/plugin/lib
/usr/lib/openvpn/plugin/lib/openvpn-auth-pam.so
```

```
/usr/lib/openvpn/plugin/lib/openvpn-down-root.so
/usr/sbin/openvpn
/usr/share/doc/openvpn-2.0.9
/usr/share/doc/openvpn-2.0.9/AUTHORS
/usr/share/doc/openvpn-2.0.9/COPYING
/usr/share/doc/openvpn-2.0.9/COPYRIGHT.GPL
/usr/share/doc/openvpn-2.0.9/INSTALL
```

The following table shows the function of the most important directories and files in this list:

Full path and files Installed by OpenVPN	Function
`/etc/openvpn`	Directory containing configuration files
`/etc/init.d/openvpn/usr/sbin/` `rcopenvpn`	Start/stop script for services
`/usr/sbin/openvpn`	The binary
`/usr/share/doc/openvpn`	Documentation files
`/usr/share/man/man8/openvpn.8.gz`	Manual page
`/usr/share/doc/openvpn/examples/` `sample-config-files`	Example configuration files
`/usr/share/doc/openvpn/examples/` `sample-keys`	Example keys and certificates
`/usr/share/doc/openvpn/examples/` `easy-rsa`	`easy-rsa`—a collection of scripts useful for creating tunnels
`/usr/share/doc/openvpn/` `changelog.Debian.gz`	Version history
`/usr/share/doc/openvpn/changelog.gz`	
`/usr/share/openvpn/verify-cn`	`verify-cn` function (revoke command)
`/usr/lib/openvpn/openvpn-auth-pam.` `so`	Libraries for PAM-Authentication and chroot mode
`/usr/lib/openvpn/openvpn-down-root.` `so`	
`/usr/share/doc/packages/openvpn/` `suse`	

`/usr/share/doc/packages/openvpn/`
`suse/`
`openvpn.init`
	SuSE-specific start/stop scripts

`/var/run/openvpn`	Process ID of the running OpenVPN process

Installing OpenVPN on Debian and Ubuntu

Probably the easiest distribution on which to install OpenVPN is Debian and its derivates like Ubuntu. Just type `apt-get install openvpn`, answer two questions, and OpenVPN is installed and ready to be used.

The Debian package management system is capable of solving all the issues that might occur during the installation. If your system is configured correctly, then the automatic installation will cover the following steps:

1. The installation helper `apt-get` will find the software on the installation servers.

2. The helper will then download the chosen package and unpack it to your local system.

3. An interactive configuration script is executed, which configures your system and the newly installed software for later use with the parameters that you enter.

The following code extract is the standard output of `apt-get install openvpn` on a Debian system. This output may vary depending on your previous software selection, and in many cases the LZO compression library will have to be installed. On some systems `apt` will install OpenSSL libraries, but in most cases, `apt-get` is able to solve all problems for you.

```
debian01:~# apt-get install openvpn
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  openvpn
0 upgraded, 1 newly installed, 0 to remove and 7 not upgraded.
Need to get 293kB of archives.
After unpacking 762kB of additional disk space will be used.
Get:1 http://ftp.uni-erlangen.de testing/main openvpn 2.0.9 [293kB]
Fetched 298kB in 1s (247kB/s)
Preconfiguring packages ...
Selecting previously deselected package openvpn.
(Reading database ... 9727 files and directories currently installed.)
Unpacking openvpn (from .../openvpn_2.0-9_i386.deb) ...
Setting up openvpn (2.0-9) ...
Restarting virtual private network daemon:.
debian01:~#
```

During this process, you will be prompted to answer the following two questions:

- You have to allow `apt` to create a TUN/TAP device for use by OpenVPN software. If you select **No**, your tunnels will not be created and your tunnel software won't work.

```
Debian Configuration

                          ┤ Configuring openvpn ├
  If you accept here, the package will make a special device called /dev/net/tun for
  openvpn's use. If you refuse, the device won't be made now. Read README.Debian for details
  on how to make it. If you are using devfs refuse here.

  Would you like a TUN/TAP device to be created?

               <Yes>                                    <No>
```
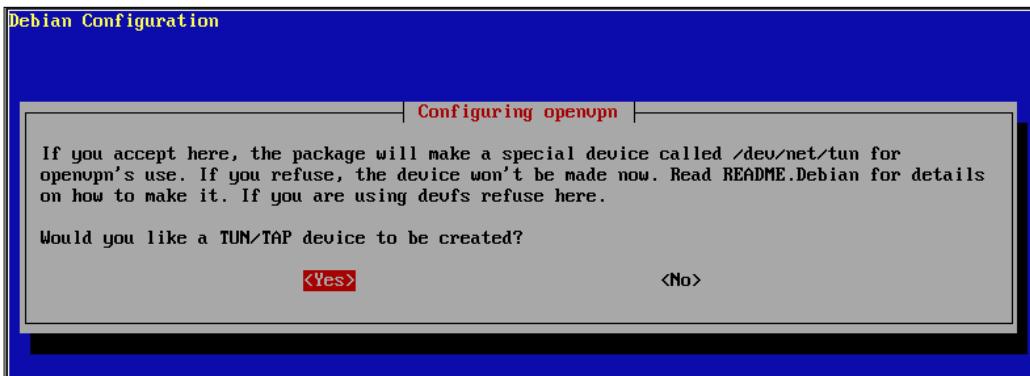
- The second question raises a security issue. OpenVPN software should be stopped during an update, so you have to select **YES** and hit return.

```
Debian Configuration

                          ┤ Configuring openvpn ├
  In some cases you may be upgrading openvpn in a remote server using a VPN to do so. The
  upgrade process stops the running daemon before installing the new version, in that case
  you may lose your connection, the upgrade may be interrupted, and you may not be able to
  reconnect to the remote host.

  Unless you do your upgrades locally, it is advised NOT to stop openvpn before it gets
  upgraded. The installation process will restart it once it's done.

  This option will take effect in your next upgrade.

  Would you like to stop openvpn before it gets upgraded?

               <Yes>                                    <No>
```

You have to stop the *old* tunnel software when an update is running. All tunneling will be stopped, and your users will not be able to connect to your system during this time. From then on, all tunnels are created by the new OpenVPN software, including patches and bug fixes. This is the safe way to go.

However, if you choose **No**, you risk that the old software and libraries are still running, even after the installation of new OpenVPN software. Bug fixes and patches of the new version may not apply to existing tunnels until they are started again. You may run into serious inconsistencies in your system, if you have several tunnels and they are running different versions of your software. Thus, it is safer to have a short time when users will not be able to connect.

Installing Debian packages

Software packages for Debian systems are provided in the so-called `.deb` file format. DEB files are usually stored in online repositories on FTP or web servers, and every Debian system holds a list of repositories that can be used for installation. You will find this list in `/etc/apt/sources.list`. The setup program `base-config` provides a menu-based configuration interface for `apt`.

```
Debian Configuration

                        ┤ Apt-Konfiguration ├
    Select the Debian mirror apt should use. You should select a mirror that is close
    to you on the net.

    Choose the Debian mirror to use:

                           mirrors.kernel.org
                           ftp.debian.org
                           debian.oregonstate.edu
                           ftp.us.debian.org
                           archive.progeny.com
                           linux.csua.berkeley.edu
                           ftp.ndlug.nd.edu
                           sluglug.ucsc.edu
                           debian.crosslink.net
                           ftp.egr.msu.edu
                           ftp.rutgers.edu
                           debian.secsup.org
                           debian.uchicago.edu
                           debian.lcs.mit.edu
                           debian.mirrors.pair.com
                           cudlug.cudenver.edu
                           mirrors.xmission.com


              <Ok>                          <Cancel>
```

If you want to add source repositories to your Debian installation, type
`base-config` and change to the menu `configure apt`. Select the country
you live in and the repository of your choice. Select **Ok**. Now all the software
packages of this server can automatically be installed on your system, simply by
typing `apt-get install <package>`.

A Debian package contains the software and information about it, such as name,
version, description, contents, prerequisites, dependencies, and configuration scripts
that are to be started after installation.

Debian systems offer some very powerful programs with which you can control
software installation very specifically. Listing all programs and options would go far
beyond the scope of this book, but here is a short overview of some handy package
management commands.

| Command	Function
`apt-get remove <package>`	Removes the selected package from your system
`apt-get update`	Updates the list of packages available on the repositories listed in `/etc/apt/sources.list`
`apt-get upgrade`	Installs the latest available versions of all your installed software
`apt-get dist-upgrade`	Installs the latest available software related to your configuration
`dpkg-reconfigure`	Restarts/Starts the configuration script inside the package, which will bring up the menu-based dialogs in the same way as after installation
`apt-cache show <package>`	Prints detailed information about the software package
`dpkg -l <package>`	Prints information on the installed software package
`dpkg -L <package>`	Lists all files installed by the software package
`dpkg -i <file>`	Installs a local (`.deb`) file to your system
`dpkg -S <file>`	Prints information about the software package owning `<file>`
`apt-cache search <string>`	Searches apt database for packages containing `<string>` in their name and description

These programs should solve all possible questions, issues, and problems concerning the installation of software on Debian systems. Just try these commands with the freshly installed OpenVPN package on your system. Type the command `apt-cache show openvpn` to receive information about the installed package.

Using Aptitude to search and install packages

Although the Debian command-line tools are very powerful, there are more programs that help you to retrieve and install software. Probably the most common software for this purpose is Aptitude. Type `aptitude` in a command line in order to start the menu-based installation interface. If Aptitude is not installed on your system, type `apt-get install aptitude`. If you prefer aptitude, you can use it at the command line in the same way as `apt-get`.
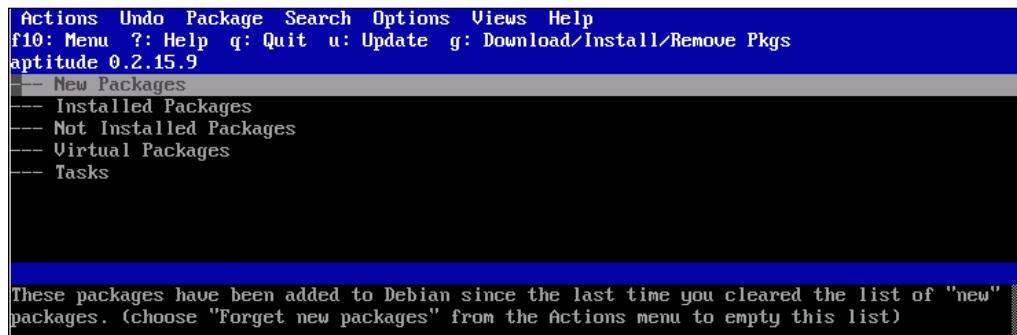
Aptitude consists of a menu at the top of the screen, a list of packages, and a window showing details on the software selected in the package list. If you have console mouse support, you can click on menu entries.

Click on the menu entry **Search**, or hit the *F10* key and navigate through the **Search** menu. Select the entry **Find**. You will be prompted with a search mask. Enter **openvpn**. While you are typing, Aptitude is steadily updating the main window. Click on **OK** and have a look at the output.

```
Actions  Undo  Package  Search  Options  Views  Help
f10: Menu  ?: Help  q: Quit  u: Update  g: Download/Install/Remove Pkgs
aptitude 0.2.15.9
i      openvpn                                   2.0-4       2.0-4
i      pidentd                                   3.0.18-3    3.0.18-3
i      portmap                                   5-14        5-14
i      pppoe                                     3.5-4       3.5-4
i      pppoeconf                                 1.7         1.7
i      shorewall                                 2.4.1-2     2.4.1-2
i      ssh                                       1:3.8.1p1-  1:3.8.1p1-
i      telnet                                    0.17-29     0.17-29
i      traceroute                                1.4a12-19   1.4a12-19
i      whois                                     4.7.5       4.7.5
  --- oldlibs - Obsolete libraries
  --- otherosfs - Emulators and software to read foreign filesystems
  --- perl - Perl interpreter and libraries
  --- python - Python interpreter and libraries
  --- shells - Command shells and alternative console environments
  --- text - Text processing utilities
Virtual Private Network daemon
An application to securely tunnel IP networks over a single UDP or TCP port. It can be used
to access remote sites, make secure point to point connections, enhance WiFi security,
etc.

OpenVPN uses all of the encryption, authentication, and certification features of the
OpenSSL library (any cipher, key size, or HMAC digest).

OpenVPN may use static, pre-shared keys or TLS-based dynamic key exchange.  It also
supports VPNs with dynamic endpoints (DHCP or dial-up clients), tunnels over NAT or
connection-oriented stateful firewalls (like Linux's iptables).
```

Aptitude will find the OpenVPN version that you had installed previously, and the entries in the menus **Actions** and **Package** help you to select and install software. Depending on the selection of repositories that you have added to your `sources.list` during installation, Aptitude can also help you to choose different versions of OpenVPN.

OpenVPN—the files installed on Debian

The following table gives an overview of the files that were installed by the Debian package management system. Some of these files will be used in later chapters.

Full path and file Installed by OpenVPN	Function
`/etc/openvpn`	Directory containing configuration files
`/etc/network/if-up.d/openvpn`	Start/stop `openvpn` when the network goes up/down
`/etc/network/if-down.d`	
`/etc/network/if-down.d/openvpn`	
`/etc/init.d/openvpn`	Start/stop script for services
`/sbin/openvpn`	The binary
`/usr/share/doc/openvpn`	Documentation files
`/usr/share/man/man8/openvpn.8.gz`	Manual page
`/usr/share/doc/openvpn/examples/sample-config-files`	Example configuration files
`/usr/share/doc/openvpn/examples/sample-keys`	Example keys
`/usr/share/doc/openvpn/examples/easy-rsa`	`easy-rsa`—a collection of scripts useful for creating tunnels
`/usr/share/doc/openvpn/changelog.Debian.gz`	Version history
`/usr/share/doc/openvpn/changelog.gz`	
`/usr/share/openvpn/verify-cn`	`verify-cn` function (revoke command)
`/usr/lib/openvpn/openvpn-auth-pam.so`	Libraries for PAM-Authentication and chroot mode
`/usr/lib/openvpn/openvpn-down-root.so`	

Installing OpenVPN on FreeBSD

FreeBSD and BSD in general are Unix systems of outstanding stability and security, and are therefore very popular among network administrators. In practice, with FreeBSD, you do not have to worry much about security issues of the software that you install, but you may not always get up-to-date versions.

FreeBSD also has a modern software management system. Simply type `pkg_add -vr openvpn` and OpenVPN software is installed on your system. Calling `pkg_add` with the parameter `-r` installs software from remote servers, similar to `apt-get` or `rpm`. If you run into problems, increasing verbosity with the parameter `-v` can be helpful.

The following excerpt shows the output of pkg_add:

```
freebsd# pkg_add -vr openvpn
looking up ftp.freebsd.org
connecting to ftp.freebsd.org:21
setting passive mode
opening data connection
initiating transfer
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/packages-5.4-
release/Latest/openvpn.tbz...x +CONTENTS
x +COMMENT
(...)
x share/doc/openvpn/sample-scripts/verify-cn
tar command returns 0 status
 Done.
Package 'openvpn-1.6.0' depends on 'lzo-1.08_1' with 'archivers/lzo'
origin.
setting passive mode
opening data connection
initiating transfer
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/packages-5.4-
release/All/lzo-1.08_1.tbz...x +CONTENTS
(...)
tar command returns 0 status
 Done.
Finished loading lzo-1.08_1 over FTP.
extract: Package name is lzo-1.08_1
(...)
        'lzo-1.08_1' loaded successfully.
(...)
extract: Package name is openvpn-1.6.0
(...)
Package openvpn-1.6.0 registered in /var/db/pkg/openvpn-1.6.0
### -----------------------------------------------------------------
----- ###
###  To use the tap driver, you may need to do: kldload if_tap
###
###  See ${PREFIX}/etc/rc.d/openvpn.sh.sample for how to do this
###
###  automatically at system boot-up time.
###
### -----------------------------------------------------------------
----- ###
###  To retain backwards compatibility of OpenVPN 1.3.0 with OpenVPN
peers ###
###  that run older versions (back to 1.1.0), you will have to set the
MTU ###
###  explicitly by command line options since OpenVPN 1.3.0.
###
```

```
###
###
###   When connecting to 1.4.X or older peers with a TAP-style tunnel,
set   ###
###   --tun-mtu 1500 --tun-mtu-extra 32 on the peer.
###
###
###
###   When using TLS security and your peer runs OpenVPN 1.3.X, the
PEER      ###
###   must use --disable-occ.  This version of OpenVPN cannot use TLS
mode  ###
###   to peers running OpenVPN 1.2.x or older.
###
###
###
###   Note: use at most --verb 4 for regular use, --verb 5 is for
debugging ###
### -------------------------------------------------------------
----- ###
freebsd#
```

The `pkg_add` looks for an appropriate installation candidate, downloads it, and checks for dependencies. As LZO is required, but not installed, `pkg_add` starts by downloading this package first. After successful installation of LZO, OpenVPN is installed. When called with the parameter `-v`, `pkg_add` also gives you a list of all the installed files.

After this installation, there are four issues to be considered.

- The OpenVPN binary may not be found in the standard path. Call OpenVPN with the full path, or add its path to your startup file.

- In our example OpenVPN version 1.6.0 was installed. There are some features of version 2.0 that cannot be used. The section that follows shows how you can install a newer version on your system.

- The standard configuration file path is `/usr/local/etc/openvpn/`.

- The `init` script that is used to start OpenVPN and its tunnels at system boot must be edited before we can use it.

The OpenVPN installation on FreeBSD provides a sample startup script (normally in `/etc/rc.d`, but in some installations it may be found in `/usr/local/etc/rc.d/`) that needs a little editing after which it can be used at system boot. To start OpenVPN at boot time, we have to change three entries in the file `/etc/rc.conf`, containing the startup configuration for the services.

Simply add or edit the following lines in your `/etc/rc.conf` to these values:

```
openvpn_enable="YES"
openvpn_if=tun
openvpn-dir=/etc/openvpn
```

If you have set the correct paths in your `init` script, OpenVPN will be started the next time you boot your system.

# Installing a newer version of OpenVPN on FreeBSD—the ports system

If you want to install OpenVPN version 2.0 on FreeBSD, you can install a FreeBSD port of OpenVPN. But before that, we should uninstall the version of OpenVPN that we have just installed. Just type `pkg_delete openvpn-1.6.0`.

`freebsd# pkg_delete openvpn-1.6.0`

Then browse to the FreeBSD web site `http://www.freebsd.org`, which is the best place to look for documentation, help, and software for FreeBSD. Click on the **Ports** under the **SHORTCUTS** section, which will take you to `http://www.freebsd.org/ports/index.html`. The **ports** are patches to the original source code of applications, as well as download routines and information for the software installation management.

Installing the port system with sysinstall

To make use of these ports, the so-called port system has to be installed on your machine. This can easily be done with the setup tool for FreeBSD called **sysinstall**. Start by typing `sysinstall`.

```
                     sysinstall Main Menu
   Welcome to the FreeBSD installation and configuration tool.  Please
   select one of the options below by using the arrow keys or typing the
   first character of the option name you're interested in.  Invoke an
   option with [SPACE] or [ENTER].  To exit, use [TAB] to move to Exit.

        Usage        Quick start - How to use this menu system
        Standard     Begin a standard installation (recommended)
        Express      Begin a quick installation (for experts)
         Custom      Begin a custom installation (for experts)
        Configure    Do post-install configuration of FreeBSD
        Doc          Installation instructions, README, etc.
        Keymap       Select keyboard type
        Options      View/Set various installation options
        Fixit        Repair mode with CDROM/DVD/floppy or start shell
        Upgrade      Upgrade an existing system
        Load Config  Load default install configuration
        Index        Glossary of functions


                    [ Select ]    X Exit Install
                 [ Press F1 for Installation Guide ]
```

Use the up/down arrow keys to select the entry **Configure** and press *Enter*.
In the following window called, 'FreeBSD Configuration Menu', change to the
module **Distributions**:

```
────── Select the distributions you wish to install. ──────
Please check off the distributions you wish to install.  At the
very minimum, this should be "base".

 <<< X Exit       Exit this menu (returning to previous)
       All        All system sources, binaries and X Window System
       Reset      Reset all of the below
 [ ]  base        Binary base distribution (required)
 [ ]  compat1x    FreeBSD 1.x binary compatibility
 [ ]  compat20    FreeBSD 2.0 binary compatibility
 [ ]  compat21    FreeBSD 2.1 binary compatibility
 [ ]  compat22    FreeBSD 2.2.x and 3.0 a.out binary compatibility
 [ ]  compat3x    FreeBSD 3.x binary compatibility
 [ ]  compat4x    FreeBSD 4.x binary compatibility
 [ ]  dict        Spelling checker dictionary files
 [ ]  doc         Miscellaneous FreeBSD online docs
 [ ]  games       Games (non-commercial)
 [ ]  info        GNU info files
 [ ]  man         System manual pages - recommended
 [ ]  catman      Preformatted system manual pages
 [ ]  proflibs    Profiled versions of the libraries
 [ ]  src         Sources for everything
 [X]  ports       The FreeBSD Ports collection
 [ ]  local       Local additions collection
 [ ]  perl        The Perl distribution
 [ ]  X.Org       The X.Org distribution


              [  OK  ]        Cancel
```

The distributions dialog contains many different distributions to install, but only
The FreeBSD Ports collection is relevant for our purpose. Activate this entry with
your *spacebar* and hit *Enter*. You will be asked to choose a source from which you
want to install these ports. Just confirm with *Enter* three times. The port system is
then downloaded and installed.

Downloading and installing a BSD port

Now we must download the port package from the BSD web site and extract it to a
local folder. Point your browser to `http://www.freebsd.org/ports/index.html`,
enter **openvpn** in the search field, and click on the **Submit** button.

As result from this search, you will be presented with OpenVPN in version 2.0.2
or newer, in the security section. Click on the download link and save the tarball
(`.tar` file) to a local directory.

Enter this directory and type `make`. The port system will fetch the appropriate sources for this port, patch them, and start the compilation process. When `make` is ready, type `make install` to install the binaries in your system.

```
freebsd# make install
===>  Installing for openvpn-2.0.2
===>   openvpn-2.0.2 depends on shared library: lzo.1 - found
===>   Generating temporary packing list
===>  Checking if security/openvpn already installed
test -z "/usr/local/sbin" || /root/openvpn/work/openvpn-2.0.2/install-
sh -d "/usr/local/sbin"
  install  -s -o root -g wheel -m 555 'openvpn' '/usr/local/sbin/
openvpn'
(...)
      This port has installed the following files which may act as
network
      servers and may therefore pose a remote security risk to the
system.
/usr/local/sbin/openvpn

      This port has installed the following startup scripts which may
cause
      these network services to be started at boot time.
/usr/local/etc/rc.d/openvpn.sh

      If there are vulnerabilities in these programs there may be a
security
      risk to the system. FreeBSD makes no guarantee about the
security of
      ports included in the Ports Collection. Please type 'make
deinstall'
      to deinstall the port if this is a concern.

      For more information, and contact details about the security
      status of this software, see the following webpage:
http://openvpn.sourceforge.net/
freebsd#
```

That's it! A new version of OpenVPN has successfully been installed on your system. You can test it with `/usr/local/sbin/openvpn –version`.

Summary

In this chapter, we have seen with numerous installations on different Linux systems that installing OpenVPN is very easy. Modern Linux systems, such as SuSE, Red Hat, Debian, Ubuntu, or FreeBSD, provide sophisticated installation and package management systems, and still offer other ways to install the software.

6
Advanced OpenVPN Installation

In most cases installing OpenVPN is easy. In this chapter, however, we will discuss non-standard and advanced methods of installing OpenVPN by compiling the source code that is provided by the OpenVPN project. When building OpenVPN from sources, it is also possible to produce RPM files for your SuSE or Red Hat systems or Debian packages, or for example, Ubuntu machines. The last troubleshooting hint may be useful for anybody who is running self-compiled kernels, and also for those who need to activate the TUN/TAP driver in the kernel, which should seldom be necessary.

Troubleshooting—advanced installation methods

Our next installation example—installing from source code—will cover a procedure that is possible on every platform and enables the administrator to change the basic behavior of OpenVPN. Many developers and administrators consider that this should be the standard installation procedure for all systems. There are some advantages regarding stability and performance that can only be optimized for your individual system by compiling as much relevant software as possible, by yourself (the so-called Gentoo approach). In most cases, however, the installation tools provided with the systems are much easier to use, but if you are looking for detailed debugging information, the source code will be the first choice.

Installing OpenVPN from source code

Your system needs several basic development tools such as make and a C compiler on a typical Debian system. You can find most of them in the package build-essential, which installs this on an Intrepid Ibex (Ubuntu 8.10) command line as follows:

```
root@ubuntu810:/home/mfeilner# aptitude install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
The following NEW packages will be installed:
  build-essential dpkg-dev{a} g++{a} g++-4.3{a} libstdc++6-4.3-dev{a}
patch{a}
0 packages upgraded, 6 newly installed, 0 to remove and 0 not
upgraded.
Need to get 6203kB of archives. After unpacking 21.3MB will be used.
Do you want to continue? [Y/n/?] Y
(...)
Setting up patch (2.5.9-5) ...
Setting up dpkg-dev (1.14.20ubuntu6) ...
Setting up libstdc++6-4.3-dev (4.3.2-1ubuntu11) ...
Setting up g++-4.3 (4.3.2-1ubuntu11) ...
Setting up g++ (4:4.3.1-1ubuntu2) ...
Setting up build-essential (11.4) ...
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
Writing extended state information... Done

root@ubuntu810:/home/mfeilner#
```

Although you can now can compile most open source software packages ('tarballs'), OpenVPN development has some more prerequisites. You have to install the compression library `liblzo1`, the corresponding development package `liblzo-devel`, and the headers of OpenSSL, `libssl-devel`. On Debian with kernel 2.6, simply type `aptitude install liblzo-dev libssl-dev`, and the package management will solve the necessary dependencies for you.

```
root@ubuntu810:/home/mfeilner# aptitude install liblzo-dev libssl-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

```
Reading extended state information
Initializing package states... Done
The following NEW packages will be installed:
  liblzo-dev liblzo1{a} libssl-dev zlib1g-dev{a}
0 packages upgraded, 4 newly installed, 0 to remove and 0 not
upgraded.
Need to get 2308kB of archives. After unpacking 6808kB will be used.
Do you want to continue? [Y/n/?] Y
(...)
Setting up liblzo1 (1.08-3) ...
Setting up liblzo-dev (1.08-3) ...
Setting up zlib1g-dev (1:1.2.3.3.dfsg-12ubuntu1) ...
Setting up libssl-dev (0.9.8g-10.1ubuntu2) ...
Processing triggers for libc6 ...
ldconfig deferred processing now taking place
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
Writing extended state information... Done
root@ubuntu810:/home/mfeilner#
```

As the next step we have to download the OpenVPN source code, in this example from a release candidate of version 2.1.

```
root@ubuntu810:/home/mfeilner# wget http://openvpn.net/release/
openvpn-2.1_rc15.tar.gz
--2008-11-23 00:04:14--  http://openvpn.net/release/openvpn-2.1_rc15.
tar.gz
Resolving openvpn.net... 74.54.73.229
Connecting to openvpn.net|74.54.73.229|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 833429 (814K) [application/x-gzip]
Saving to: `openvpn-2.1_rc15.tar.gz'

100%[================================>] 833,429      182K/s   in
5.1s

2008-11-23 00:04:19 (159 KB/s) - `openvpn-2.1_rc15.tar.gz' saved
[833429/833429]

root@ubuntu810:/home/mfeilner#
```

We have to untar the `tar.gz` archive to a local directory.

```
root@ubuntu810:/home/mfeilner# tar -xzf openvpn-2.1_rc15.tar.gz
root@ubuntu810:/home/mfeilner# ls -l
total 852
drwxr-xr-x  2 mfeilner mfeilner   4096 2008-11-22 22:35 Desktop
drwxr-xr-x  2 mfeilner mfeilner   4096 2008-11-22 22:33 Documents
drwxrwxrwx 14 mfeilner mfeilner   4096 2008-11-19 19:19 openvpn-2.1_
rc15
-rw-r--r--  1 root     root     833429 2008-11-19 18:19 openvpn-2.1_
rc15.tar.gz
(...)
root@ubuntu810:/home/mfeilner#
```

A directory called `openvpn-x` is created, with 'x' as the version number of the tarball that you chose to download. Move to this directory and type: `/configure`.

```
root@ubuntu810:/home/mfeilner/openvpn-2.1_rc15# ./configure
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking for ifconfig... /sbin/ifconfig
checking for ip... /sbin/ip
checking for route... /sbin/route
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking for a BSD-compatible install... /usr/bin/install -c
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking whether gcc needs -traditional... no
checking for ANSI C header files... yes
(...)
pkcs11-helper headers not found.
configure: creating ./config.status
```

```
config.status: creating Makefile
config.status: creating openvpn.spec
config.status: creating config-win32.h
config.status: creating images/Makefile
config.status: creating service-win32/Makefile
config.status: creating install-win32/Makefile
config.status: creating install-win32/settings
config.status: creating config.h
config.status: executing depfiles commands
root@ubuntu810:/home/mfeilner/openvpn-2.1_rc15#
```

You will receive few screens full of output. The configure script checks for software dependencies and the compatibility of the source code with your system and then creates a so-called `makefile`, which is used as a sort of guideline for later compilation. In the example above, `configure` found no `pkcs11-helper` headers on your system, so you won't be able to use those in your later configuration. But `configure` can do much more for you, you can specify paths, options, and many parameters for OpenVPN at this command line. If you don't want the `openvpn` binary to reside in `/usr/local/sbin/` or `/usr/sbin/`, here is the place to change that. The following screenshot shows the optional features and packages that can be used with the `configure` command of OpenVPN:

In the next step, the command `make` interprets the `makefile`, and compiles the program and all the necessary libraries. Type `make` to start this process.

```
root@ubuntu810:/home/mfeilner/openvpn-2.1_rc15# make
make  all-recursive
make[1]: Entering directory `/home/mfeilner/openvpn-2.1_rc15'
(...)
if gcc -DHAVE_CONFIG_H -I. -I. -I.   -I.    -g -O2 -MT manage.o -MD
-MP -MF ".deps/manage.Tpo" -c -o manage.o manage.c; \
        then mv -f ".deps/manage.Tpo" ".deps/manage.Po"; else rm -f
".deps/manage.Tpo"; exit 1; fi
if gcc -DHAVE_CONFIG_H -I. -I. -I.   -I.    -g -O2 -MT mbuf.o -MD -MP
-MF ".deps/mbuf.Tpo" -c -o mbuf.o mbuf.c; \
        then mv -f ".deps/mbuf.Tpo" ".deps/mbuf.Po"; else rm -f
".deps/mbuf.Tpo"; exit 1; fi
if gcc -DHAVE_CONFIG_H -I. -I. -I.   -I.    -g -O2 -MT misc.o -MD -MP
-MF ".deps/misc.Tpo" -c -o misc.o misc.c; \
(…)
if gcc -DHAVE_CONFIG_H -I. -I. -I.   -I.    -g -O2 -MT cryptoapi.o -MD
-MP -MF ".deps/cryptoapi.Tpo" -c -o cryptoapi.o cryptoapi.c; \
        then mv -f ".deps/cryptoapi.Tpo" ".deps/cryptoapi.Po"; else rm
-f ".deps/cryptoapi.Tpo"; exit 1; fi
gcc  -g -O2   -o openvpn  base64.o buffer.o crypto.o dhcp.o error.o
event.o fdmisc.o forward.o fragment.o gremlin.o helper.o lladdr.o
init.o interval.o list.o lzo.o manage.o mbuf.o misc.o mroute.o mss.o
mtcp.o mtu.o mudp.o multi.o ntlm.o occ.o pkcs11.o openvpn.o options.o
otime.o packet_id.o perf.o pf.o ping.o plugin.o pool.o proto.o proxy.o
ieproxy.o ps.o push.o reliable.o route.o schedule.o session_id.o
shaper.o sig.o socket.o socks.o ssl.o status.o thread.o tun.o win32.o
cryptoapi.o  -lssl -lcrypto -llzo -ldl
make[2]: Leaving directory `/home/mfeilner/openvpn-2.1_rc15'
make[1]: Leaving directory `/home/mfeilner/openvpn-2.1_rc15'
root@ubuntu810:/home/mfeilner/openvpn-2.1_rc15#
```

On a modern PC, this will take a few minutes, on slow systems, you can have a coffee now. OpenVPN and its components are compiled. Make calls `gcc` with parameters according to the `makefile` that `configure` has created. `gcc` compiles the source code files to binary files that you (or your operating system) can execute.

```
root@ubuntu810:/home/mfeilner/openvpn-2.1_rc15# ls -l openvpn
-rwxr-xr-x 1 root root 1805104 2008-11-23 00:23 openvpn
root@ubuntu810:/home/mfeilner/openvpn-2.1_rc15#
```

These binary files have to be installed to the proper places in your system. Type `make install` to accomplish that.
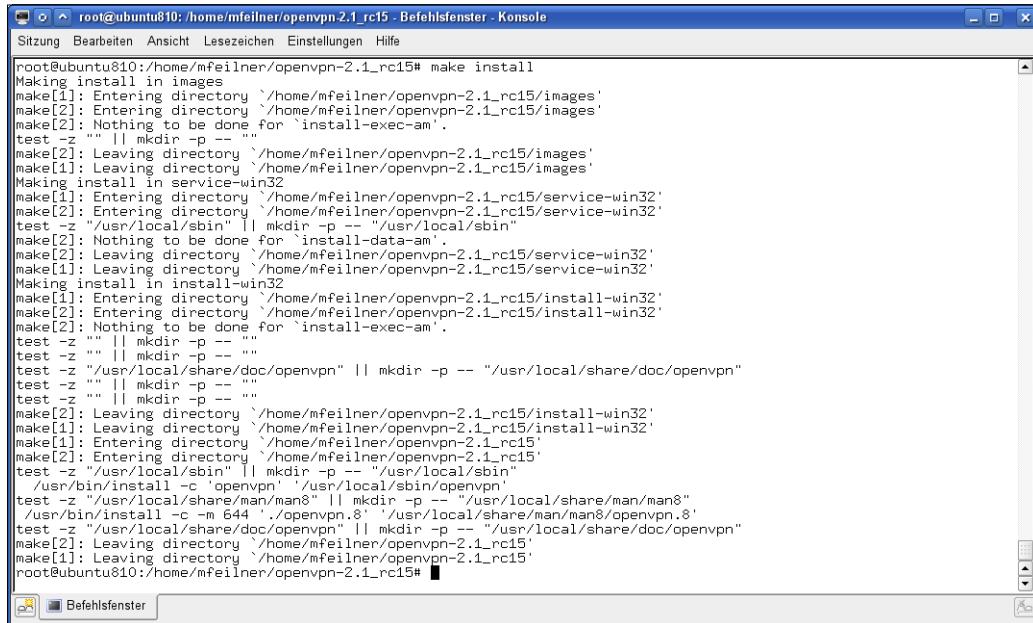
```
root@ubuntu810: /home/mfeilner/openvpn-2.1_rc15 - Befehlsfenster - Konsole
Sitzung  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
root@ubuntu810:/home/mfeilner/openvpn-2.1_rc15# make install
Making install in images
make[1]: Entering directory `/home/mfeilner/openvpn-2.1_rc15/images'
make[2]: Entering directory `/home/mfeilner/openvpn-2.1_rc15/images'
make[2]: Nothing to be done for `install-exec-am'.
test -z "" || mkdir -p -- ""
make[2]: Leaving directory `/home/mfeilner/openvpn-2.1_rc15/images'
make[1]: Leaving directory `/home/mfeilner/openvpn-2.1_rc15/images'
Making install in service-win32
make[1]: Entering directory `/home/mfeilner/openvpn-2.1_rc15/service-win32'
make[2]: Entering directory `/home/mfeilner/openvpn-2.1_rc15/service-win32'
test -z "/usr/local/sbin" || mkdir -p -- "/usr/local/sbin"
make[2]: Nothing to be done for `install-data-am'.
make[2]: Leaving directory `/home/mfeilner/openvpn-2.1_rc15/service-win32'
make[1]: Leaving directory `/home/mfeilner/openvpn-2.1_rc15/service-win32'
Making install in install-win32
make[1]: Entering directory `/home/mfeilner/openvpn-2.1_rc15/install-win32'
make[2]: Entering directory `/home/mfeilner/openvpn-2.1_rc15/install-win32'
make[2]: Nothing to be done for `install-exec-am'.
test -z "" || mkdir -p -- ""
test -z "" || mkdir -p -- ""
test -z "/usr/local/share/doc/openvpn" || mkdir -p -- "/usr/local/share/doc/openvpn"
test -z "" || mkdir -p -- ""
test -z "" || mkdir -p -- ""
make[2]: Leaving directory `/home/mfeilner/openvpn-2.1_rc15/install-win32'
make[1]: Leaving directory `/home/mfeilner/openvpn-2.1_rc15/install-win32'
make[1]: Entering directory `/home/mfeilner/openvpn-2.1_rc15'
make[2]: Entering directory `/home/mfeilner/openvpn-2.1_rc15'
test -z "/usr/local/sbin" || mkdir -p -- "/usr/local/sbin"
  /usr/bin/install -c 'openvpn' '/usr/local/sbin/openvpn'
test -z "/usr/local/share/man/man8" || mkdir -p -- "/usr/local/share/man/man8"
 /usr/bin/install -c -m 644 './openvpn.8' '/usr/local/share/man/man8/openvpn.8'
test -z "/usr/local/share/doc/openvpn" || mkdir -p -- "/usr/local/share/doc/openvpn"
make[2]: Leaving directory `/home/mfeilner/openvpn-2.1_rc15'
make[1]: Leaving directory `/home/mfeilner/openvpn-2.1_rc15'
root@ubuntu810:/home/mfeilner/openvpn-2.1_rc15# 
Befehlsfenster
```

The `make install` is the only stage of this process (besides `aptitude`) that requires root privileges. Many admins recommend not to invoke `configure` and `make` as root, and I guess as many others say this is paranoid. However, if you want to work on a secure basis, open two shell windows in parallel, one for root and one for a 'normal' user.

We see that only a few files are installed: `/usr/local/sbin/openvpn`, and two manual pages. OpenVPN is now ready to be used on your system. If you don't believe, just type `openvpn –version`.

```
root@ubuntu810:/home/mfeilner/openvpn-2.1_rc15# openvpn --version
OpenVPN 2.1_rc15 i686-pc-linux-gnu [SSL] [LZO1] [EPOLL] built on Nov
23 2008
Developed by James Yonan
Copyright (C) 2002-2008 OpenVPN Technologies, Inc. <sales@openvpn.net>
root@ubuntu810:/home/mfeilner/openvpn-2.1_rc15#
```

The OpenVPN binary that was used was compiled (built) on November 23, 2008, and is available in your path.

Building and distributing .deb packages

One great feature of the Debian package management is automatic installation and update of software packages. If you have built your own Debian package from the source code, you can easily update all your servers automatically with an individually configured, improved, and tested OpenVPN version, simply by placing a file in your own repository.

There are several ways to build a `.deb` package. The following is the shortest one that I found reasonable, and the example works on Ubuntu. Needless to say, these instructions will work on any Debian machine. If you already have the build-essential package installed, you need to start by adding three packages to your system. As root, type the following:

`aptitude install autotools-dev fakeroot dh-make`

Then get the source code, extract it, and move the directory to one named `openvpn-2.1.15` because `dh_make` is somewhat sensitive to the name of the directory.

`cd openvpn-2.1.15`

Now, type `dh_make` to generate a so-called control file for the Debian package that is going to be built. Use the option `-f` to specify the tarball that you want to use.

```
root@ubuntu810:/home/mfeilner/openvpn-2.1.15# dh_make -f ../
openvpn-2.1_rc15.tar.gz

Type of package: single binary, multiple binary, library, kernel
module or cdbs?
 [s/m/l/k/b] s

Maintainer name : root
Email-Address   : root@unknown
Date            : Sun, 23 Nov 2008 01:36:47 +0100
Package Name    : openvpn
Version         : 2.1.15
License         : blank
Using dpatch    : no
Type of Package : Single
Hit <enter> to confirm:
Done. Please edit the files in the debian/ subdirectory now. openvpn
uses a configure script, so you probably don't have to edit the
Makefiles.
root@ubuntu810:/home/mfeilner/openvpn-2.1.15#
```

You can edit the data in the file `debian/control` below your working directory.
Now you can build the package by invoking `dpkg-buildpackage`.

```
root@ubuntu810:~/openvpn-2.1.15$ dpkg-buildpackage
dpkg-buildpackage: set CFLAGS to default value: -g -O2
dpkg-buildpackage: set CPPFLAGS to default value:
(...)
if gcc -DHAVE_CONFIG_H -I. -I. -I.   -I.    -g -O2 -MT ssl.o -MD -MP
-MF ".deps/ssl.Tpo" -c -o ssl.o ssl.c; \
        then mv -f ".deps/ssl.Tpo" ".deps/ssl.Po"; else rm -f ".deps/
ssl.Tpo"; exit 1; fi
if gcc -DHAVE_CONFIG_H -I. -I. -I.   -I.    -g -O2 -MT status.o -MD
-MP -MF ".deps/status.Tpo" -c -o status.o status.c; \
        then mv -f ".deps/status.Tpo" ".deps/status.Po"; else rm -f
".deps/status.Tpo"; exit 1; fi
(...)
make[1]: Leaving directory `/home/mfeilner/openvpn-2.1.15'
(...)
dh_md5sums
dh_builddeb
dpkg-deb: building package `openvpn' in `../openvpn_2.1.15-1_i386.
deb'.
 dpkg-genchanges  >../openvpn_2.1.15-1_i386.changes
dpkg-genchanges: including full source code in upload
dpkg-buildpackage: full upload (original source is included)
root@ubuntu810:/home/mfeilner/openvpn-2.1.15#
```

`Dpkg-buildpackage` runs the usual `configure` and `make` processes. If it runs
without errors, you will find the appropriate `.deb` file in a directory that is above
the source code directory. As a final test, you should install it, and call the binary
to control the following version:

```
root@ubuntu810:/home/mfeilner# dpkg -i openvpn_2.1.15-1_i386.deb
Selecting previously deselected package openvpn.
(Reading database ... 102636 files and directories currently
installed.)
Unpacking openvpn (from openvpn_2.1.15-1_i386.deb) ...
Setting up openvpn (2.1.15-1) ...
Processing triggers for man-db ...
root@ubuntu810:/home/mfeilner# openvpn --version
OpenVPN 2.1_rc15 i686-pc-linux-gnu [SSL] [LZO1] [EPOLL] built on Nov
23 2008
Developed by James Yonan
Copyright (C) 2002-2008 OpenVPN Technologies, Inc. <sales@openvpn.net>
root@ubuntu810:/home/mfeilner#
```

Building your own RPM file

As you may have seen in the section on Red Hat and SuSE, RPM files are quite handy. You can copy them to any other system of the same type and have them installed automatically. If you need to use a specific version of OpenVPN, you may want to create your own RPM files from a source code file, and distribute them to your servers. This may sound complicated, but it is done with one single command, if your system meets some prerequisites. The first step is downloading the source code of OpenVPN. The second step is installing some dependencies such as the mentioned `lzo-devel` packages and the `pkcs-helper` package.

On most standard SuSE systems, only these two libraries are missing for building this RPM. On OpenSuSE, you simply install them with YaST, or with a simple `zypper in lzo-devel pkcs11-helper-devel`. On Red Hat systems you can use `yum`. After installing them, start `rpmbuild`.

```
opensuse110:/home/mfeilner # rpmbuild -tb openvpn-2.0.9.tar.gz
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.53657
+ umask 022
+ cd /usr/src/packages/BUILD
+ cd /usr/src/packages/BUILD
+ rm -rf openvpn-2.0.9
+ /usr/bin/gzip -dc /home/mfeilner/openvpn-2.0.9.tar.gz
+ tar -xf -
```

```
+ STATUS=0
(…)
checking for ifconfig... /sbin/ifconfig
checking for ip... /sbin/ip
checking for route... /sbin/route
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking target system type... i586-suse-linux
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane...
(...)
Wrote: /usr/src/packages/RPMS/i586/openvpn-2.0.9-1.i586.rpm
Executing(%clean): /bin/sh -e /var/tmp/rpm-tmp.31583
+ umask 022
+ cd /usr/src/packages/BUILD
+ cd openvpn-2.0.9
+ '[' /var/tmp/openvpn-root '!=' / ']'
+ rm -rf /var/tmp/openvpn-root
+ rm -rf filelists
opensuse110:/home/mfeilner #
```

While you receive several screens of output, the OpenVPN source code is configured and compiled automatically. At the end, the RPM file is placed in /usr/src/packages/RPMS/i586/, and can be installed with RPM from the following location:

```
opensuse110:/home/mfeilner # rpm -Uvh /usr/src/packages/RPMS/i586/
openvpn-2.0.9-1.i586.rpm
Preparing...                #########################################
## [100%]
   1:openvpn                #########################################
## [100%]
openvpn                 0:off  1:off  2:off  3:on   4:off  5:on
6:off
Shutting down openvpn:
done
Starting openvpn:
done
opensuse110:/home/mfeilner #
```

No matter whether you have a RPM or Debian-based distribution, for a working repository server, there is still some organizational work that is left to be done:

- Configure one of your HTTP or FTP servers to act as a Debian or a SuSE repository. Howtos for both cases can be found here: `http://www.debian.org/doc/manuals/repository-howto/repository-howto` (Debian) and `http://www.charlescurley.com/yum/repository.html` (Red Hat), or simply mirror an online repository (SuSE).
- Add your repository to the list of installation sources of all the systems in which you want to automatically install your software.
- Add a cronjob to your systems that runs `aptitude update` and `aptitude upgrade`, `yum update` or `zypper up` on a regular basis.
- Place your binaries on the server.

The next time your servers run the software update, it will automatically download the new OpenVPN software.

Enabling Linux kernel TUN/TAP support

If your kernel does not support TUN/TAP devices, you have to enable it in the kernel configuration. All modern Linux/UNIX distributions support TUN/TAP devices, so it is very unlikely for you to run into this problem. Probably, this will only usually happen if you have built your own kernel. In this case, you will already guess how to enable TUN/TAP support.

If you are not running your own kernel, but your system does not support TUN/TAP devices, you have to build a kernel of your own. Even though this process is not that complicated, the documentation would go beyond the scope of this book.

> The process of kernel compilation is documented at `http://tldp.org/LDP/tlk/tlk.html` and the Linux kernel source code can be obtained from `http://www.kernel.org/`.

In short you have to do the following:

- Install the sources of the kernel of your choice.
- Change to the directory where you installed the sources. In most cases they can be found in `/usr/src/linux`.
- Configure the kernel with one of the appropriate configuration tools such as `menuconfig` or `Xconfig`.
- Compile the kernel and the modules using `make` and `make modules`.
- Install the kernel and configure your boot manager's settings.

If you want TUN/TAP device support, you have to select the driver during the process of kernel configuration. This can be done with various tools such as `Xconfig` or `menuconfig`. The `Xconfig` tool is probably best if you have a workstation with a running X-Server, whereas `menuconfig` is best on a simple command line.

Using menuconfig

The following three steps show you how to enable module support for your Linux kernel before building it. Type `make menuconfig` to configure the sources of your kernel, on an Ubuntu standard system with `build-essential` installed, you will still need the package `ncurses-dev`. You can navigate through `menuconfig` using the up/down and *Tab* keys. Select an entry by highlighting it with your cursor and pressing *Enter*.

- Select the entry **Device Drivers** and press *Enter* to receive the list of available devices that the kernel source code supports.
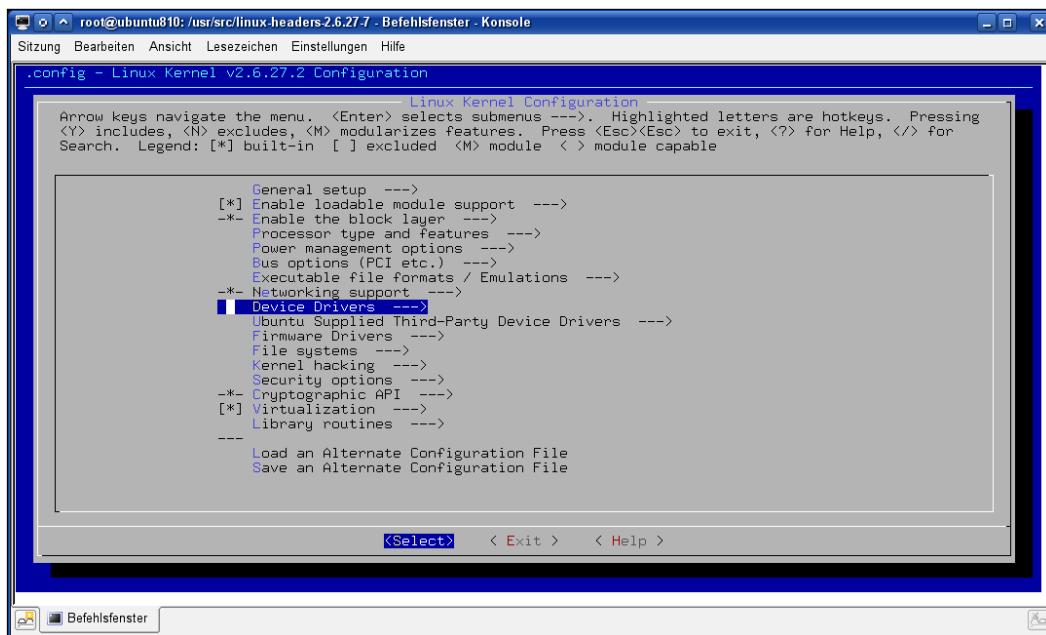
- Select **Network device support** and press *Enter*.
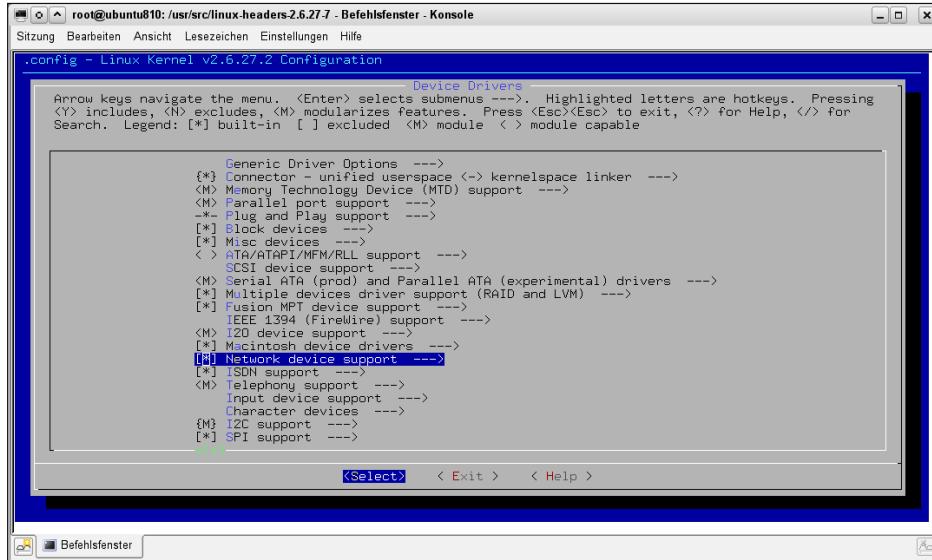
- In the list of available network drivers, you will see the entry **Universal TUN/TAP driver support**. By pressing the *spacebar*, you can select if the driver is loaded permanently, as a module, or not at all. In the first column, a letter will show your selection. (**M** is for 'module', ***** is for 'permanent', empty is for 'not to be installed').
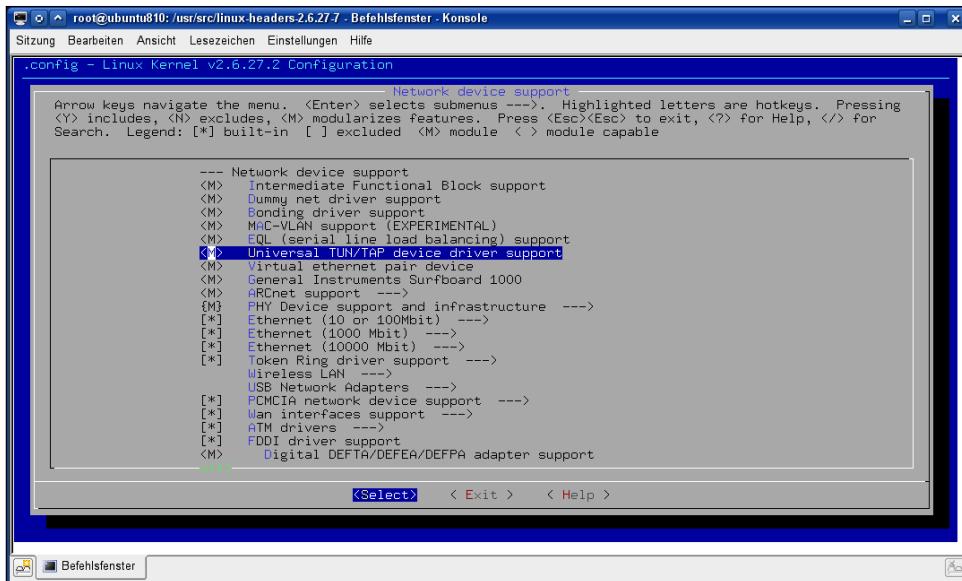
In the previous screenshot the driver is selected as a module, which means the driver is only loaded when needed. This is probably the best selection because the tunnel driver is unloaded when it is not needed, and system's resources are set free.

Now you can continue your kernel configuration. After compilation, installation, and reboot, your system should be able to provide TUN/TAP devices.

Summary

In this chapter, we learned about the advanced options that an installation from source code offers. OpenVPN can be configured in detail using this method, and by building Debian or RPM binary files the administrator does not lose the flexibility that modern package management has to offer.

7
Configuring an OpenVPN Server—The First Tunnel

In this chapter, we will create an encryption key for OpenVPN and use it to set up our first OpenVPN tunnel between two Windows systems in the same network. By doing so, we have a test-bed environment where no problems with firewalls or routers will interfere with our OpenVPN setup, and we can concentrate on learning how to create tunnels.

A little work on the configuration file needs to be done and the key has to be exchanged between these systems. After this, the tunnel will be started and tested with the `ping` command. We will then copy the key on a Linux system and connect this system with a tunnel to the first Windows machine. Finally, we will ensure that OpenVPN is run automatically on both systems and have a look at the Service Manager on Windows and the `init` system on Linux.

OpenVPN on Microsoft Windows

During the installation process, OpenVPN creates the following entries in Windows' main menu, as shown on Windows XP:
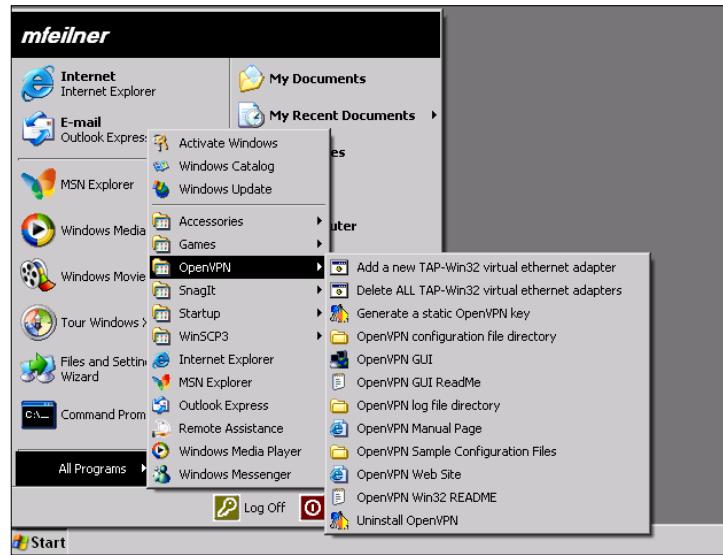
On Windows Server 2008 with OpenVPN 2.1_rc15, the following entries are created:
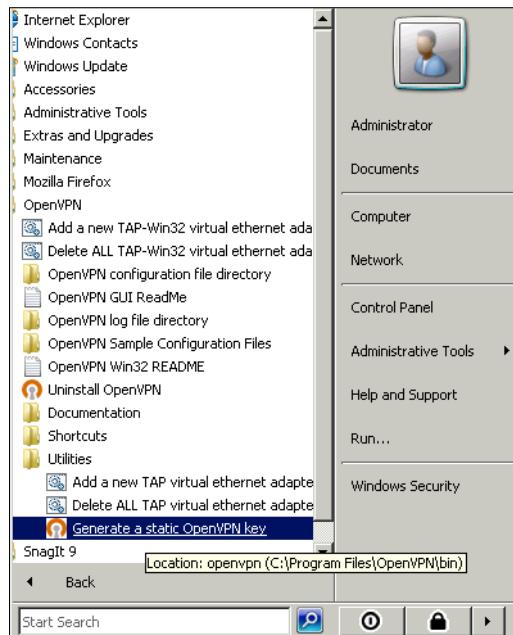
As you can see, depending on your Windows and OpenVPN version, the exact location in the menu can vary.

At this point, only the following five entries in this menu are relevant:

Title	Function
Generate a static OpenVPN key	Creates a static encryption key that can be used for creating tunnels
OpenVPN configuration file directory	Opens an Explorer window in the directory `C:\Program Files\OpenVPN\config`, where the configuration data for OpenVPN is stored
OpenVPN GUI	Starts the **OpenVPN GUI** that plugs in the system tray of the taskbar
OpenVPN log file directory	Opens an Explorer window in the directory `C:\Program Files\OpenVPN\log`, where the log files for OpenVPN are kept
OpenVPN Sample Configuration Files	Opens an Explorer window in the directory `C:\Program Files\OpenVPN\sample-config`, where example configuration files for OpenVPN can be found

Apart from these entries, you will find information on OpenVPN in the online manual page, a readme file, a link to the web site, and some entries helping you manage the network interfaces that OpenVPN creates. In OpenVPN 2.1, the entry for quick key generation can be found under `Utilities`.

Generating a static OpenVPN key

Before we can connect two systems with an OpenVPN tunnel, we have to create a static key that will be used for encryption of the traffic. This key must be provided to both systems because in this case of symmetric encryption, both sides will use the same key.

Select the entry **Generate a static OpenVPN key** in Windows' **OpenVPN** menu:
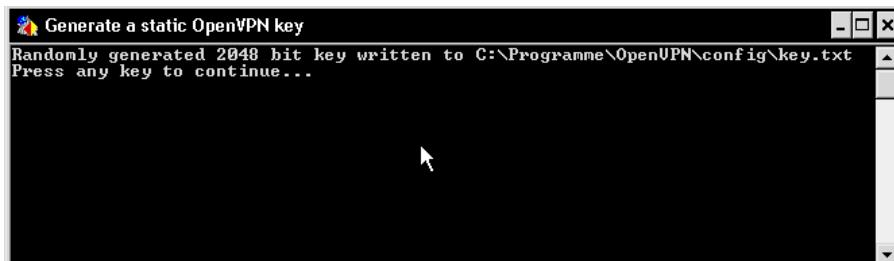
OpenVPN will open a command-line window and generate a 2048 bit encryption key. This key is saved in the standard configuration directory with the name `key.txt`. This key should only be used for testing and learning purposes. However, for our small test setup it is necessary.

This process is also executed by the `openvpn.exe` program. The shortcut calls the following:

```
"C:\Program Files\OpenVPN\bin\openvpn.exe" --pause-exit --verb 3
--genkey --secret "C:\Program Files\OpenVPN\config\key.txt"
```
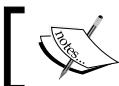
> Do not use this key for anything but testing OpenVPN connections.

In the next chapter, we will explain the use of the OpenVPN command-line interface.

The menu entry **OpenVPN GUI** starts the OpenVPN panel applet. If there is no such entry on your system, then you can find the program in `C:\Program Files\OpenVPN\bin\openvpn-gui.exe`. After the installation, this applet runs in the background, so clicking this menu entry will only bring up the window stating, **OpenVPN GUI is already running**. If you stop the GUI, then this entry will restart the panel applet.

The other three menu entries open Explorer windows in three different directories:

- The directory `C:\Program Files\OpenVPN\config\` is the default place where OpenVPN will look for configuration and key files. Have a look at the screenshot of the key generation, and you will see that the key that we have generated is written to `C:\Program Files\OpenVPN\config\key.txt`.

- In the directory `C:\Program Files\OpenVPN\sample-config\`, we find configuration files for standard setup. These files need to be changed slightly and can be then used to test VPN functionality.

- The output of the tunnel software is written to text files in the directory `C:\Program Files\OpenVPN\log\`.

The following screenshot shows an arrangement of Explorer windows of the four most important OpenVPN directories:
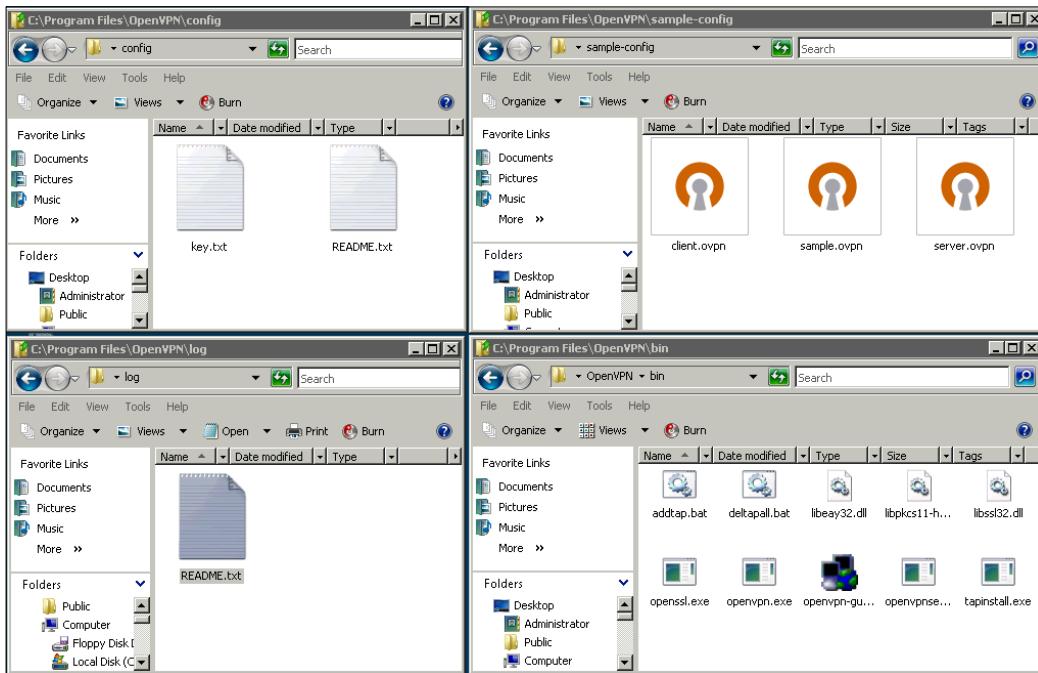

Creating a sample connection

We will now create a sample VPN connection to see how the **OpenVPN GUI** works. Open all three directories by clicking on their entries in the main menu. Copy the sample configuration file `sample.ovpn` from the sample configuration directory into the configuration directory. On Windows, the standard filename extension for OpenVPN configuration files is `.ovpn`, whereas on Linux it is `.conf`.

You can use drag-and-drop to accomplish that. That's all! Your new OpenVPN configuration can be started through the panel applet—if your network matches the needs of the sample configuration.

Right-click on the panel applet. You will see the context menu has more entries now. Select the entry `Edit Config` to view the sample file in `notepad.exe`, and click on **Connect** to start the sample configuration.
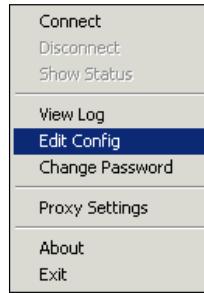

The window **OpenVPN Connection (sample)** is opened. In this window, the protocol output of the sample connection, which is also written to a logfile in the log directory, is shown. You can see that there is still some configuration work to be done. In the sample configuration, OpenVPN is advised to connect to a remote server called **myremote**. If you don't happen to have an OpenVPN server with this name in your local network, then you should see a window exactly like the one that follows. This means that your Windows OpenVPN software is up and running, but it cannot create a tunnel.
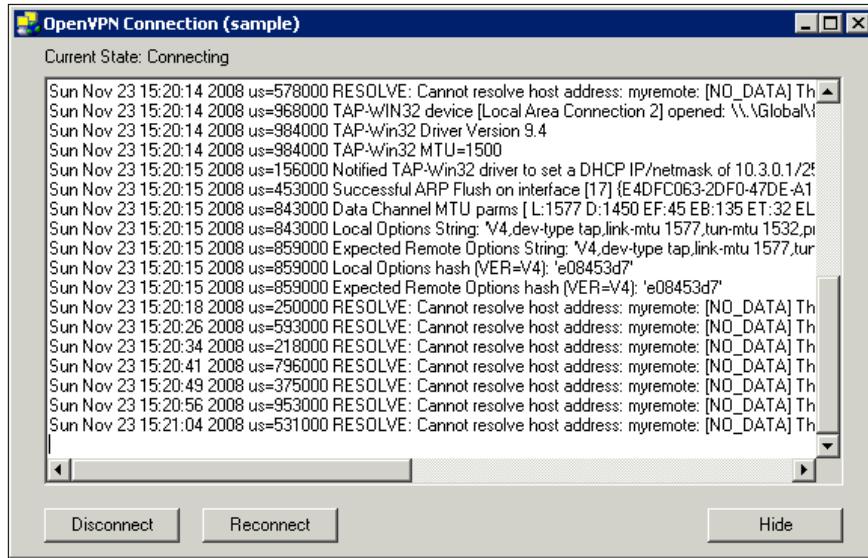
Adapting the sample configuration file provided by OpenVPN

Obviously, we have to change our configuration a little. Select the entry **OpenVPN configuration file directory** from the Windows main menu, and double-click on the sample configuration file we copied here. Notepad starts up and shows us the sample configuration file.

In this file we have to change or enter the following three settings:

- The name or IP address of the other VPN host
- The name of the key file
- The IP addresses for the VPN and the host

OpenVPN needs the IP address of the other tunnel endpoint in order to know where to connect to. To make sure both sides are using the same encryption key, we must specify the file in which the key is kept. Last but not least, the tunnel net itself must be equipped with IP addresses. These IP addresses are the ones assigned to the virtual network adapter. Every tunnel has one virtual network adapter on either side, and both sides can only communicate with each other if they are in the same network segment. Thus, we have to choose an IP address for each host. In my example I use the IP addresses provided by the sample file set, namely, **10.3.0.1** and **10.3.0.2**.

Once you have chosen the appropriate parameters for these settings, you can easily connect two systems. Just keep in mind that you need to have the settings for the IP addresses mirrored.

The following table shows my OpenVPN configuration file entries for two hosts connected through OpenVPN that are in the same subnet, namely 10.10.10.0:

Host A (10.10.10.103)	Host B (10.10.10.104)
remote 10.10.10.104	**remote 10.10.10.103**
ifconfig 10.3.0.1 255.255.255.0	**ifconfig 10.3.0.2 255.255.255.0**
secret `key.txt`	**secret** `key.txt`

These are the only configuration parameters in OpenVPN configuration files which are important when setting up our example tunnel:

- `remote`: It defines the other end of the tunnel. Here you can use IP addresses or DNS entries.
- `ifconfig`: It sets the local IP and netmask for the tunnel interface. **secret** tells OpenVPN which key file to use, relative to the configuration directory.

The following graphic should help to clarify this a little:
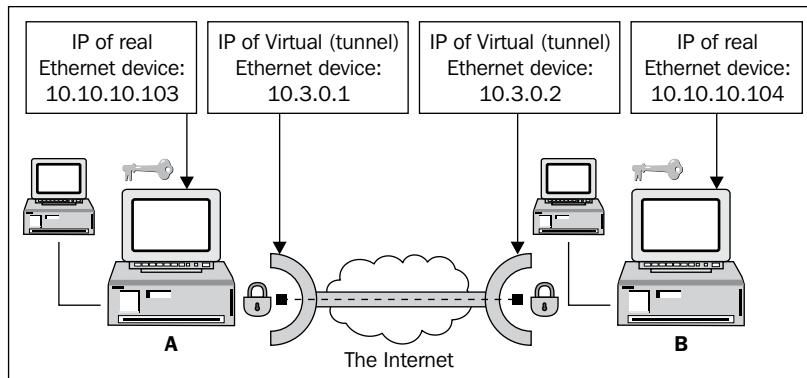
For an OpenVPN tunnel, there are four network devices involved. Two of them are real Ethernet cards and the other two are virtual tunnel devices (TUN or TAP). The real network devices have IP addresses assigned to them under which the system is reachable in its local net. The virtual network devices have IP addresses assigned to them that are used to set up the tunnel.

In our little example, Host A with the LAN IP **10.10.10.103** tries to connect to Host B with the LAN IP **10.10.10.104**. The IP of the virtual network interface (in the tunnel network) for Host A is **10.3.0.1**, and for Host B is **10.3.0.2**. The name of the key file is `key.txt` on both systems.
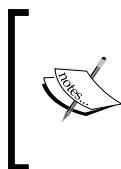
> OpenVPN can have either IP addresses or DNS names as options to the configuration parameter `remote`. If you use DNS names, you have to make sure that domain name resolution on your system is configured properly. In any case, you must make sure the other host is reachable — check DNS, routing, and firewall configuration.

You may have noticed that the two hosts in our example are in the same subnet. This is a simple setup where no routing, DNS, or firewall issues will interfere with our tunnels. All we need are two PCs running OpenVPN. The option `remote` is the only option that needs to be changed later when we set up a tunnel between two Internet sites. OpenVPN allows both DNS names and IP addresses here.

Now, copy the key file `key.txt` to the second system and edit this system's configuration file. An easy way to do this is by creating a shared folder on one system and mapping it as a network drive on the other system.

Starting and testing the tunnel

When both systems are prepared, start the **OpenVPN GUI** (or make sure it is running) and select the entry **Connect** from its context menu on both systems.

If everything has worked out fine, the OpenVPN icon on both systems will change to green like the one here:

If you see a red light then no OpenVPN tunnel is connected. Yellow is shown while a connection is being set up. Once this process is successful, the icon switches to green.

However, if you are using a local firewall on either system, be sure that it is not blocking these packets. The Windows XP firewall, like most firewall systems, is per default not blocking outgoing packets, which means that an OpenVPN connection should always be established. If you run into connection problems, then check the section, *Troubleshooting Firewall Issues*, at the end of this chapter.

Select the entry **Show Status** from the **OpenVPN GUI** context menu to receive more detailed information about the process of connecting, as shown in the following screenshot:
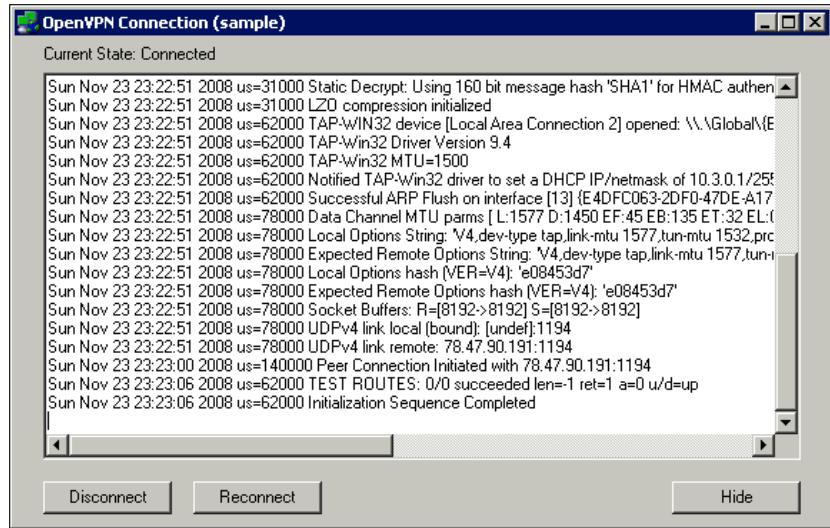
For now, only the last line of this output is important: **Initialization Sequence Completed** is OpenVPN's message of success. Your tunnel is up and running and both systems should show this message in the status log.

Let's now test the tunnel with the `ping` command. Start a DOS shell by selecting the Windows main menu **Run** and entering **cmd.exe**. You will be presented with a command-line interface, as shown in the following screenshot. Type `ping 10.3.0.2` on Host A to check if the ping packets are correctly transferred to Host B. On Host B, you will have to enter `ping 10.3.0.1` if you used the same network addresses as in the previously mentioned example.

If you receive output as in the following screenshot, then the `ping` command has been successful, and the OpenVPN tunnel is working.
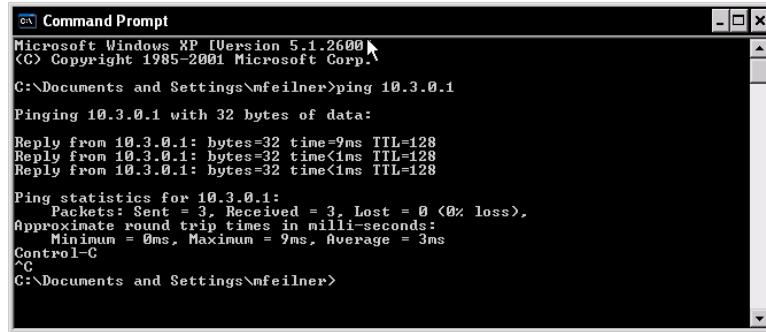
A brief look at Windows OpenVPN network interfaces

On your Windows system, open the **Control Panel** and change to **Network Connections**. As you can see, for every OpenVPN tunnel you configure, a virtual network interface is added. The following screenshot shows the active interface—the default when the tunnel is up. This appears like a real network interface and can be used like any other interface. To verify this have a look at the properties dialog in the context menu of the interface's icon. Apart from the fact that this interface is presented as a **TAP-Win32 Adapter V8**, every setting possible on real network adapters can be chosen here too. This is what this looks like on Windows XP:
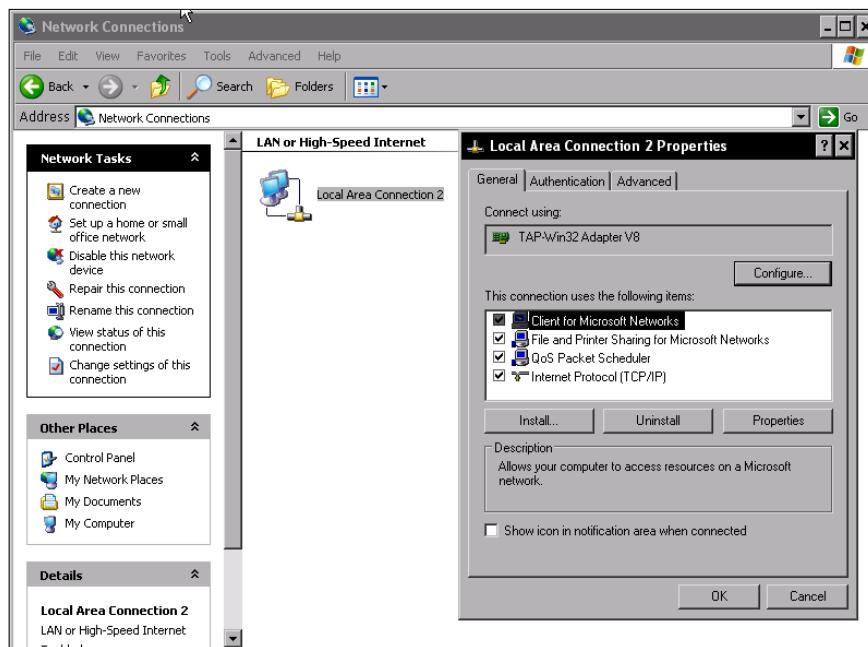
On newer systems the TAP-Adapter will be version 9 (`V9`). You can disable this interface by simply double-clicking on its icon. But keep in mind that the tunnel won't be connected automatically when you enable the interface again. You must reconnect manually by selecting the entry in OpenVPN's context menu.

If you need detailed information on network interfaces, the command `ipconfig /all` is very helpful. Open a DOS Shell under Windows and enter `ipconfig /all`. Windows will list all available network interfaces, the IPs, and routing data.
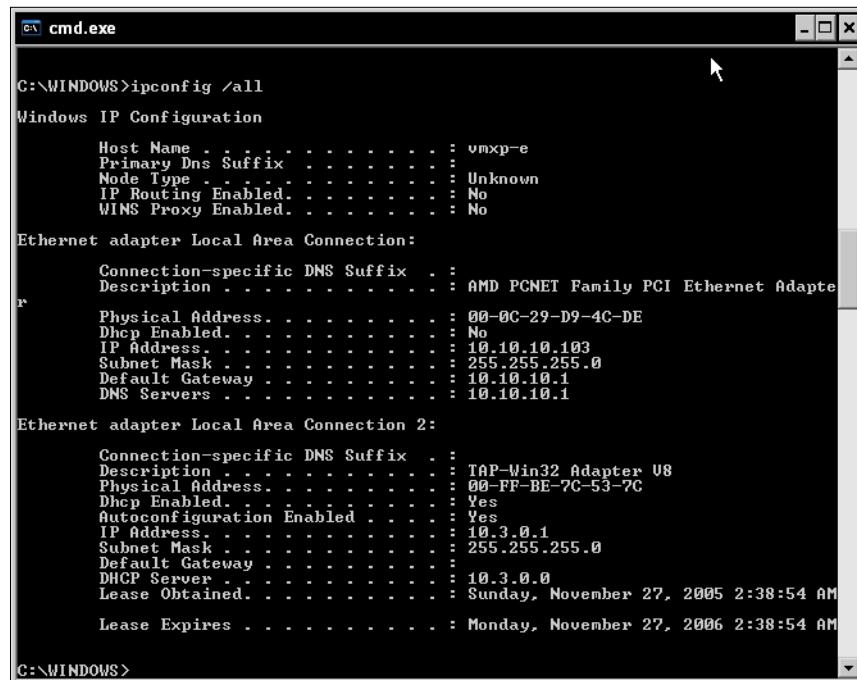
```
cmd.exe

C:\WINDOWS>ipconfig /all

Windows IP Configuration

        Host Name . . . . . . . . . . . . : vmxp-e
        Primary Dns Suffix  . . . . . . . :
        Node Type . . . . . . . . . . . . : Unknown
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        Description . . . . . . . . . . . : AMD PCNET Family PCI Ethernet Adapte
r
        Physical Address. . . . . . . . . : 00-0C-29-D9-4C-DE
        Dhcp Enabled. . . . . . . . . . . : No
        IP Address. . . . . . . . . . . . : 10.10.10.103
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 10.10.10.1
        DNS Servers . . . . . . . . . . . : 10.10.10.1

Ethernet adapter Local Area Connection 2:

        Connection-specific DNS Suffix  . :
        Description . . . . . . . . . . . : TAP-Win32 Adapter V8
        Physical Address. . . . . . . . . : 00-FF-BE-7C-53-7C
        Dhcp Enabled. . . . . . . . . . . : Yes
        Autoconfiguration Enabled . . . . : Yes
        IP Address. . . . . . . . . . . . : 10.3.0.1
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . :
        DHCP Server . . . . . . . . . . . : 10.3.0.0
        Lease Obtained. . . . . . . . . . : Sunday, November 27, 2005 2:38:54 AM

        Lease Expires . . . . . . . . . . : Monday, November 27, 2006 2:38:54 AM

C:\WINDOWS>
```

Connecting Windows and Linux

Connections between these two operating systems are almost as simple as those described in the previous section. The steps that need to be taken are exactly the same. However, there are two pitfalls that you must avoid, and both of the pitfalls are related to transferring files from Windows to Linux and vice versa.

File exchange between Windows and Linux

Windows systems use the **Server Message Block** (**SMB**) protocol to communicate and exchange data. Linux has no native support for this, but there is a powerful server suite called **Samba**, which can be used to make Linux machines appear like Windows PCs (and even integrate them into Active Directory domains). With a little Linux know-how and the right tools, access to such a samba drive is easy. Just install the Linux samba client and point a tool such as KDE's konqueror to your Windows machine, as shown in the following screenshot:
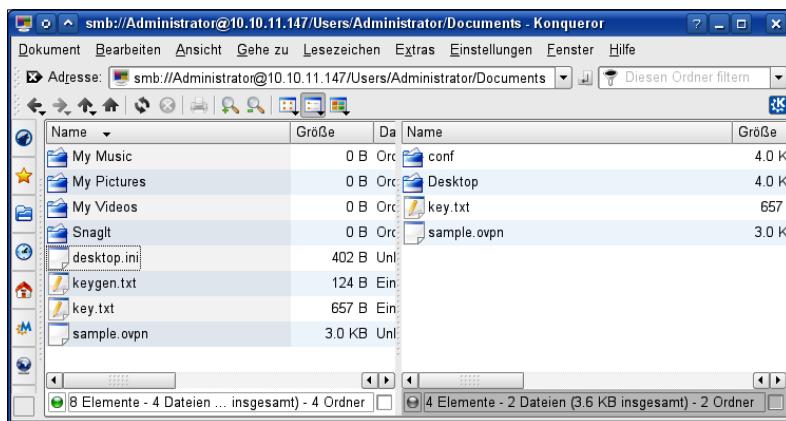
On SuSE systems, for example, only two packages are required, namely, `kdebase3-samba` and `libsmbclient`. After installing them with `zypper install kdebase3-samba libsmbclient`, you can enter a URL like `smb://User@Windowshost:/Path/To/Files` in Konqueror and easily access and drag-and-drop the relevant files, after you have allowed remote access to the relevant folder on the Windows system.

WinSCP

But how do we copy the `key` file from a Windows machine to a Linux server? On Linux, remote command execution and data exchange through the Secure Shell SSH is the standard. SSH also uses OpenSSL for encryption in the same way as OpenVPN. However, Windows has no built-in support for such encrypted data exchange. We have two possibilities. Either we set up Samba on Linux to act as a Windows client or server or (this is by far the better choice) we install SSH client software on Windows. A very simple tool for this purpose is **WinSCP**, which can be downloaded freely from `http://winscp.net/`. WinSCP is an Explorer-style application that provides drag-and-drop copying over secure connections.

- Download WinSCP and double-click on the EXE file.
- Select your preferred language, and click on **OK**.

- After the welcome, you are asked to accept the free GPL license. Click on **Next** twice.

- If you want a different location for this program, then enter its path in the third dialog.

- Select a preferred user interface. Probably the best for beginners is the default one—the **Norton Commander Interface**.

This dialog lets you choose the default look of WinSCP. If you choose the **Norton Commander interface**, you will be presented with a file manager window split into two parts, namely, a local and a remote directory. This is the default selection and might be the most useful one. However, if you prefer the Windows Explorer style, then select the button **Explorer-like interface**, which presents the remote directory in one single window.

Finish the installation by clicking on **Next** and then on **Install** in the following dialog. The setup program then extracts and sets up WinSCP. After clicking on **Finish**, WinSCP will start automatically.

Transferring the key file from Windows to Linux with WinSCP

After WinSCP is started, you have to tell it where to connect to. Enter the IP address or DNS name of your Linux system in the field **Host name**, the name of the Linux user (the administrator 'root') in the field **User name**, and the password in the field **Password**. Other options are not necessary at this point. Click on **Login** to start the connection. If you are connecting for the first time, WinSCP will ask you if you are sure of the authenticity of the host you want to connect to. If you click **OK** here, then WinSCP will remember this host's signature next time.
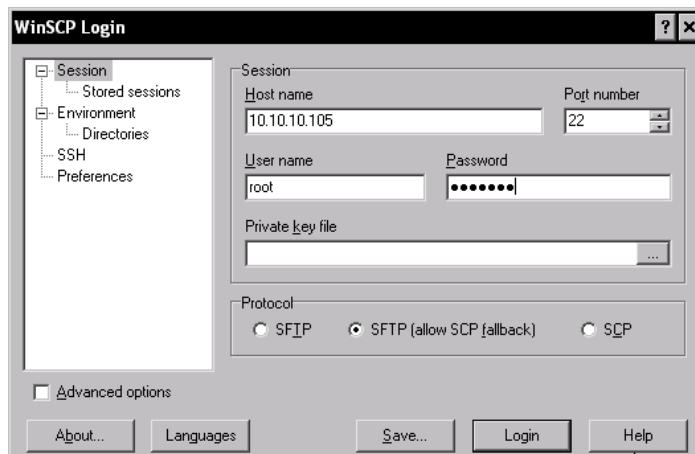
If you want to save your profile for later use, then click on **Save…**. Click on **Login** to start the connection, and accept the **Warning** window about the unknown host key. At first connect, this is normal. But if you receive such a warning later on, then you should check if you have made a mistake or why your server has a new key. Has an attacker had the chance to change it?
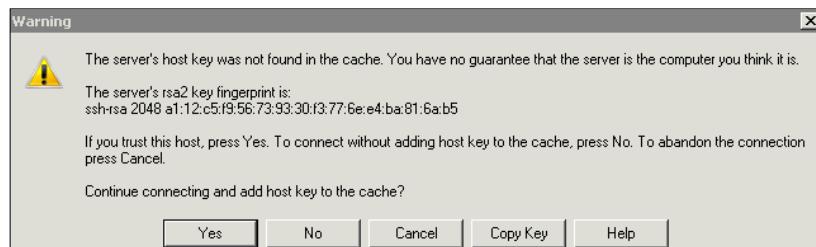
WinSCP presents a window similar to the following screenshot. On the left-hand side of the window there should be a local directory listing, while the right-hand side shows a directory on the remote server. The small drop-down menus above the listings allow fast selection and change of working directories.
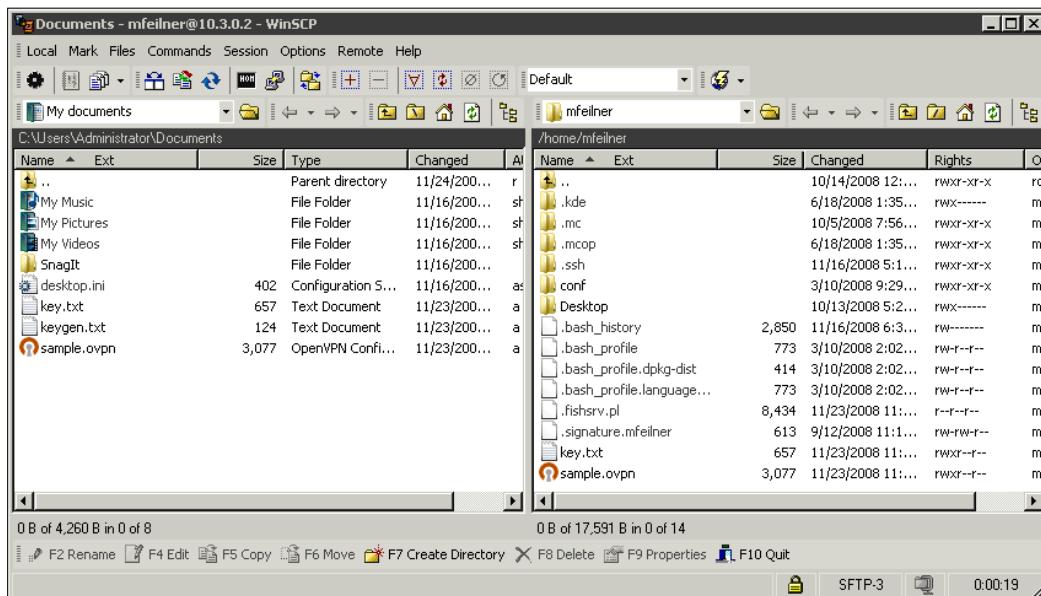
Now let's copy the `key` file and the configuration file from Windows to the Linux system. On the Windows machine, change to the directory `C:\Program Files\OpenVPN\config`, on the Linux system, change to `/etc/openvpn`. Drag-and-drop the `key.txt` file and the configuration file `sample.ovpn` to the Linux system.
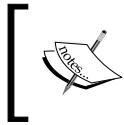
> .ovpn is the standard extension for OpenVPN's Windows configuration files. .conf is the OpenVPN standard extension on Linux.

The second pitfall—carriage return/end of line

Exchanging text files between Linux and Windows always produces another problem. On Unix systems, the *new line* character signifies the end of a line. On DOS/Windows, the characters *return* and *new line* are always used together to signify this. In our tests with Windows Server 2008, this problem seemed to be solved.

Thus text files copied from a DOS system to a Unix system always have superfluous characters at the end of the lines, and files that are copied the other way always miss line feeds. Because this problem is very common, the Linux community has developed the **dos2unix** and **unix2dos** utilities. The dos2unix converts text files from the DOS/Windows format into correct Unix format, and the unix2dos does it the other way.

In our example, we have to convert both the key file and the configuration file into a Unix format.

```
(...)
# Change 'myremote' to be your remote host,^M
# or comment out to enter a listening^M
# server mode.^M
remote 10.10.10.104^M
^M
# Uncomment this line to use a different^M
# port number than the default of 1194.^M
; port 1194^M
^M
# Choose one of three protocols supported by^M
# OpenVPN.  If left commented out, defaults^M
# to udp.^M
(...)
```

If your sample.ovpn looks like this (as my vi shows it), then you have copied the file from Windows to Unix. To convert it to Unix format, simply type the following command:

debian01:~# dos2unix sample.ovpn

Have a look at this file again. Everything should be alright now. Repeat this step for the key file (`key.txt`). If you forget this step, OpenVPN will find different keys on both systems and will therefore deny setting up the tunnel.

> The `dos2unix` is contained in the `sysutils` package of Debian systems. Run `apt-get install sysutils` to install these tools.

Configuring the Linux system

In our next step, we have to adapt the Linux configuration, just as we did on the Windows systems before. We will use exactly the same configuration as in our first example. Only three lines need to be changed. The following figure gives an overview on how the interfaces will be set up:
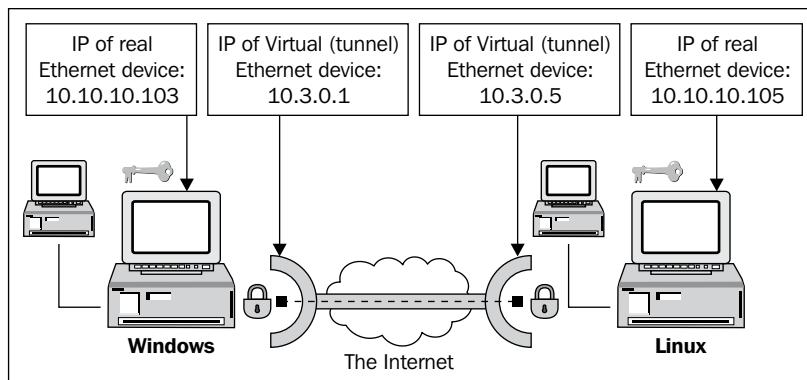
The Linux OpenVPN configuration is as simple as its Windows counterpart. Just modify the following lines in your `sample.ovpn`:

- `remote 10.10.10.103`
- `ifconfig 10.3.0.5 255.255.255.0`
- `secret key.txt`

After modifying the previously mentioned lines, adapt them to your needs. The IP address specified in the line `remote 10.10.10.103` must be replaced with that of your Windows server. The IP address specified in the line `ifconfig 10.3.0.5 255.255.255.0` defines the IP address of the virtual tunnel network interface. You may have noticed that this IP address can be chosen freely in this network segment.

Then fire up the tunnel by typing the following command in the configuration directory:

```
openvpn --config sample.ovpn
```

This command is the main part of OpenVPN. No matter what operating system you are using, it is the way to start tunnels for testing purposes and is also called by the scripts that provide OpenVPN services.

You will receive output similar to the following:

```
Wed Oct 19 00:23:01 2005 us=318267 TUN/TAP device tap0 opened
Wed Oct 19 00:23:01 2005 us=318335 TUN/TAP TX queue length set to 100
Wed Oct 19 00:23:01 2005 us=318372 /sbin/ifconfig tap0 10.3.0.5
netmask 255.255.255.0 mtu 1500 broadcast 10.3.0.255
Wed Oct 19 00:23:01 2005 us=334639 Data Channel MTU parms [ L:1577
D:1450 EF:45 EB:135 ET:32 EL:0 AF:3/1 ]
Wed Oct 19 00:23:01 2005 us=334726 Local Options String: 'V4,dev-
type tap,link-mtu 1577,tun-mtu 1532,proto UDPv4,ifconfig 10.3.0.0
255.255.255.0,comp-lzo,cipher BF-CBC,auth SHA1,keysize 128,secret'
Wed Oct 19 00:23:01 2005 us=334740 Expected Remote Options String:
'V4,dev-type tap,link-mtu 1577,tun-mtu 1532,proto UDPv4,ifconfig
10.3.0.0 255.255.255.0,comp-lzo,cipher BF-CBC,auth SHA1,keysize
128,secret'
Wed Oct 19 00:23:01 2005 us=334806 Local Options hash (VER=V4):
'e08453d7'
Wed Oct 19 00:23:01 2005 us=334831 Expected Remote Options hash
(VER=V4): 'e08453d7'
Wed Oct 19 00:23:01 2005 us=334886 Socket Buffers: R=[109568->131072]
S=[109568->131072]
Wed Oct 19 00:23:01 2005 us=334961 UDPv4 link local (bound):
[undef]:1194
Wed Oct 19 00:23:01 2005 us=334975 UDPv4 link remote:
10.10.10.103:1194
Wed Oct 19 00:23:03 2005 us=513994 Peer Connection Initiated with
10.10.10.103:1194
Wed Oct 19 00:23:03 2005 us=514046 Initialization Sequence Completed
```

This program is also part of the Windows installation. Start a command line and change to the directory containing the configuration file. Type the command `openvpn --config sample.ovpn` and press *Enter*. You will receive to the output shown in the following screenshot. As you can see, OpenVPN's behavior is almost identical to Linux version.

Unfortunately, if you start a tunnel manually like this, then the **OpenVPN GUI** will not be able to notice it.
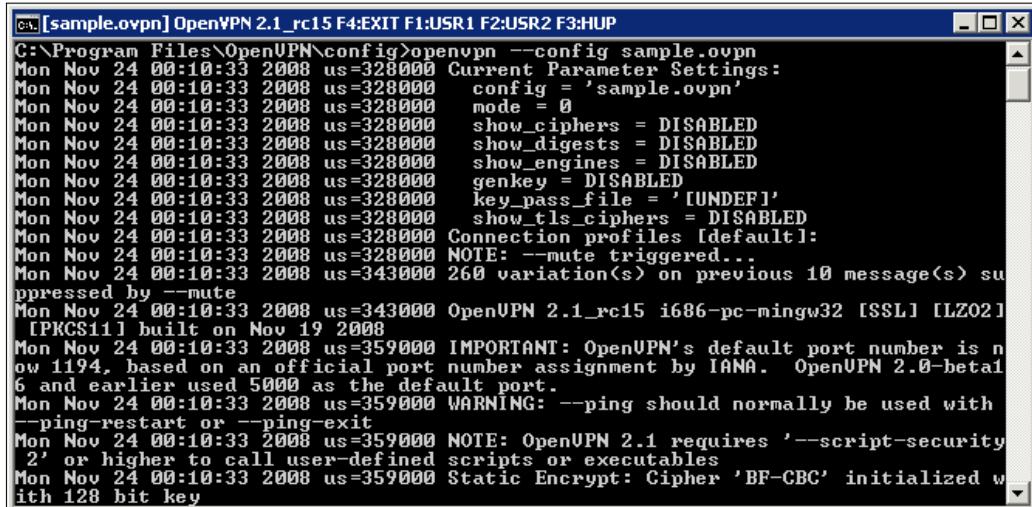
```
[sample.ovpn] OpenVPN 2.1_rc15 F4:EXIT F1:USR1 F2:USR2 F3:HUP        _ □ X
C:\Program Files\OpenVPN\config>openvpn --config sample.ovpn
Mon Nov 24 00:10:33 2008 us=328000 Current Parameter Settings:
Mon Nov 24 00:10:33 2008 us=328000   config = 'sample.ovpn'
Mon Nov 24 00:10:33 2008 us=328000   mode = 0
Mon Nov 24 00:10:33 2008 us=328000   show_ciphers = DISABLED
Mon Nov 24 00:10:33 2008 us=328000   show_digests = DISABLED
Mon Nov 24 00:10:33 2008 us=328000   show_engines = DISABLED
Mon Nov 24 00:10:33 2008 us=328000   genkey = DISABLED
Mon Nov 24 00:10:33 2008 us=328000   key_pass_file = '[UNDEF]'
Mon Nov 24 00:10:33 2008 us=328000   show_tls_ciphers = DISABLED
Mon Nov 24 00:10:33 2008 us=328000 Connection profiles [default]:
Mon Nov 24 00:10:33 2008 us=328000 NOTE: --mute triggered...
Mon Nov 24 00:10:33 2008 us=343000 260 variation(s) on previous 10 message(s) su
ppressed by --mute
Mon Nov 24 00:10:33 2008 us=343000 OpenVPN 2.1_rc15 i686-pc-mingw32 [SSL] [LZO2]
 [PKCS11] built on Nov 19 2008
Mon Nov 24 00:10:33 2008 us=359000 IMPORTANT: OpenVPN's default port number is n
ow 1194, based on an official port number assignment by IANA.  OpenVPN 2.0-beta1
6 and earlier used 5000 as the default port.
Mon Nov 24 00:10:33 2008 us=359000 WARNING: --ping should normally be used with
--ping-restart or --ping-exit
Mon Nov 24 00:10:33 2008 us=359000 NOTE: OpenVPN 2.1 requires '--script-security
 2' or higher to call user-defined scripts or executables
Mon Nov 24 00:10:33 2008 us=359000 Static Encrypt: Cipher 'BF-CBC' initialized w
ith 128 bit key
```

Testing the tunnel

Now it's time to test the tunnel. Simply use `ping` again to test the reachability of the other tunnel endpoint. On our Linux system, the following will be shown:

```
vpnserver:~# ping 10.3.0.1
PING 10.3.0.1 (10.3.0.1) 56(84) bytes of data.
64 bytes from 10.3.0.1: icmp_seq=1 ttl=128 time=77.7 ms
64 bytes from 10.3.0.1: icmp_seq=2 ttl=128 time=23.2 ms
64 bytes from 10.3.0.1: icmp_seq=3 ttl=128 time=23.5 ms
64 bytes from 10.3.0.1: icmp_seq=4 ttl=128 time=23.4 ms
64 bytes from 10.3.0.1: icmp_seq=5 ttl=128 time=23.5 ms

--- 10.3.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 23.242/34.310/77.730/21.710 ms
vpnserver:~#
```

Both tunnel endpoints are reachable. Our Windows-Linux tunnel is working!

A look at the Linux network interfaces

As we did on Windows, we will have a short look at the Linux network interfaces. Type `ifconfig` and Linux will show you all the available interfaces, as shown in the following block of code:

```
vpnserver:~# ifconfig
eth1      Protokoll:Ethernet  Hardware Adresse 00:16:00:00:00:16
          inet Adresse:WW.XX.YY.ZZ  Bcast:WW.XX.YY.ZZ
Maske:255.255.255.248
          inet6 Adresse: fe80::216:ff:fe00:16/64
Gültigkeitsbereich:Verbindung
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:112620 errors:0 dropped:0 overruns:0 frame:0
          TX packets:64662 errors:0 dropped:0 overruns:0 carrier:0
          Kollisionen:0 Sendewarteschlangenlänge:1000
          RX bytes:141835529 (135.2 MiB)  TX bytes:4943699 (4.7 MiB)
lo        Protokoll:Lokale Schleife
          inet Adresse:127.0.0.1  Maske:255.0.0.0
          inet6 Adresse: ::1/128 Gültigkeitsbereich:Maschine
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:62 errors:0 dropped:0 overruns:0 frame:0
          TX packets:62 errors:0 dropped:0 overruns:0 carrier:0
          Kollisionen:0 Sendewarteschlangenlänge:0
          RX bytes:9432 (9.2 KiB)  TX bytes:9432 (9.2 KiB)
tap0      Protokoll:Ethernet  Hardware Adresse E2:C4:62:25:03:E5
          inet Adresse:10.3.0.2  Bcast:10.3.0.255  Maske:255.255.255.0
          inet6 Adresse: fe80::e0c4:62ff:fe25:3e5/64
Gültigkeitsbereich:Verbindung
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:308 errors:0 dropped:0 overruns:0 frame:0
          TX packets:72 errors:0 dropped:0 overruns:0 carrier:0
          Kollisionen:0 Sendewarteschlangenlänge:100
          RX bytes:48798 (47.6 KiB)  TX bytes:10224 (9.9 KiB)
vpnserver:~#
```

On this system, there is an Ethernet card `eth0` configured with a 'real' IP and a loopback interface `lo`. The device `tap0` is the TAP device used by OpenVPN and has the IP `10.3.0.2` assigned. This TAP device is a virtual Ethernet device that runs OpenVPN's bridging mode. On Unix systems, you can choose between bridging mode with TAP devices and routing mode with TUN devices, but for Windows systems only the TAP driver is available.

Running OpenVPN automatically

If you want your OpenVPN machine to provide remote access and therefore act like a VPN server, you simply need to start the OpenVPN process (task) and have it run permanently. Once a client like those we configured before connects, the tunnel is up. On Windows, this task is done with the **Services** module of the **Control Panel**.

OpenVPN as a server on Windows

From Windows XP main menu, select the entry **Control Panel | Administrative Tools | Services** to start the service manager (On Windows Server 2008, you will find that in **Administrative Tools | Services**).
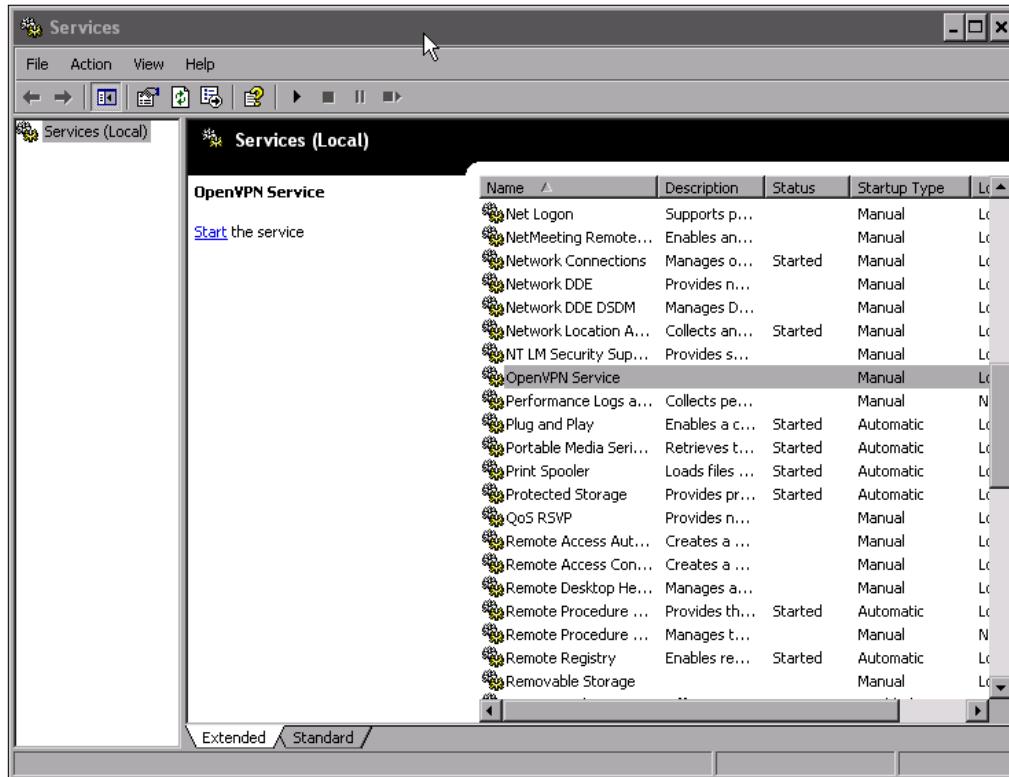
Scroll down this list until you find the entry **OpenVPN Service**. The fourth column shows the **Startup Type** for OpenVPN and is set to **Manual** by default. Double-click on this entry and you will see the following properties window:
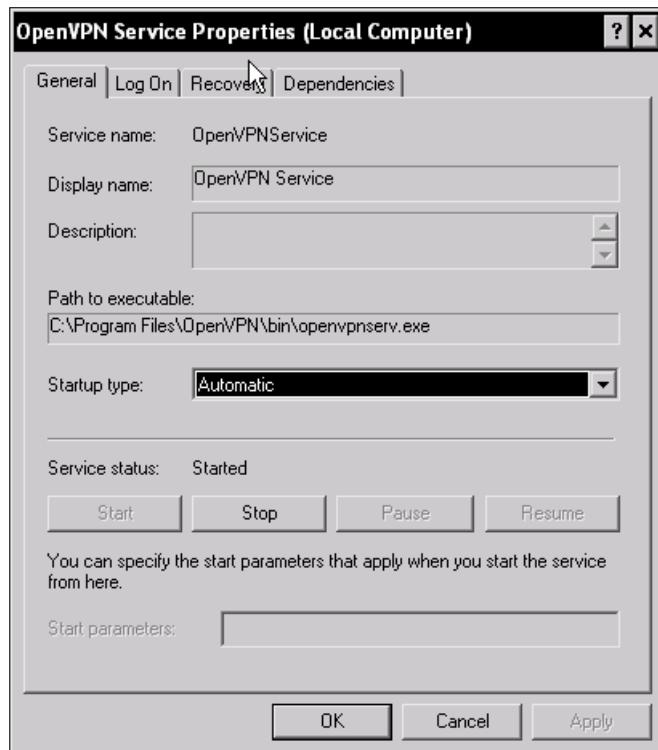
Select the entry **Automatic** from the **Startup type** drop-down menu to provide tunnel access from boot time. Confirm by clicking on **OK** and closing the services dialog. You have successfully turned your system into a simple VPN server.

To test this, simply reboot your system and have a look at the remote system's log file. You will find entries saying **Connection refused** or **No route to host**, but after the restart the tunnel will be started automatically, the logfile will show **Connection initiated**, and pings will be successful.

> OpenVPN will try to start a tunnel for every `.ovpn` file it finds
> in the `config` directory, if it is called through the service manager
> (for example, on reboot).

OpenVPN as a server on Linux

During the Linux installation on Debian-based systems, you would have been asked whether you wanted OpenVPN to be started automatically. This is the standard if you accepted all the defaults during installation by pressing *Enter* all the time. On Windows you have the services dialog, and on Linux there is the directory `/etc/init.d` containing start scripts for an abundance of server processes. A typical script in this directory can be called with the options `start` and `stop` (among others) and therefore starts or stops the server process described in its code. After you have installed OpenVPN there is a script `/etc/init.d/openvpn` on your system that you can use to stop and start your server.

Some examples of calling the OpenVPN script on Linux are as follows:

Script Syntax	Function
`/etc/init.d/openvpn start`	Starts the OpenVPN server
`/etc/init.d/openvpn stop`	Stops the OpenVPN server
`/etc/init.d/openvpn restart`	Stops and then restarts the OpenVPN server
`/etc/init.d/openvpn reload`	Forces the OpenVPN server to reload its configuration, applying changes

Runlevels and init scripts on Linux

Every Linux system can be run at different runlevels. Like the gears of a car that offer different combinations of speed and power, every runlevel on a Linux system provides different server processes and possibilities. Runlevel 1, for example, is normally used for maintenance mode and provides only single-user access, no networking, and no GUI. Runlevel 5 is usually used for a full-featured desktop system with network access. Most servers run in runlevel 3, where no graphical interface is started, but both networking and multi-user support are available.

Of course you can configure exactly which service is to be run on which runlevel. The following description explains how:

A tree of directories with start/stop scripts is used to configure the starting and stopping of services during boot time or runlevel change. On Debian systems you find this tree under `/etc` in the directories `rc0.d` through `rc6.d`. On SuSE and Red Hat these directories can be found under `/etc/init.d`. Each of these directories contains links to the `/etc/init.d/` service files. The links have names starting with `K` or `S` indicating that this service is to be stopped (`K`—killed) or started (`S`) for this runlevel, while the number after the `K` or `S` is used to order the services. Thus, all necessary processes for a server can be started in the correct order before the server process starts itself. For example, OpenVPN needs `network` and `syslog` support to work correctly. Therefore, the link has a number higher than the link files of the `network` and `syslog` daemons. On a SuSE Linux system, for example, network services are started through `S05network`, then `S06syslog` starts the logging facilities, and OpenVPN is started with `S12openvpn`.

For each runlevel, a directory exists containing a collection of links following the scheme explained before. The links in the directory `/etc/rc3.d`, for example, on a Debian system start and stop the services for runlevel 3. An OpenVPN starts script that has been called through the link `S20openvpn` in the directory `/etc/rc3.d` will be started on entering runlevel 3 after all scripts with names from `S1` through `S19` are started.

Three command line programs are relevant for management of system services on Linux—`init`, `runlevel`, and `update-rc.d`. The following table gives us an overview:

The Program	Used For
`init <runlevel>`	Change to runlevel number `<runlevel>`
`runlevel`	Lists the active (and the last) runlevel
`update-rc.d <options>`	Helps you arrange the processes automatically

Using runlevel and init to change and check runlevels

Both `runlevel` and `init` are very easy-to-use programs. The `init 1` switches your system to runlevel 1—mostly configured as single user mode for maintenance. The `init 5` switches to runlevel 5, which is the desktop user mode.

In the following example, we will first find out at which runlevel our system is at, and as the next step, switch to runlevel 5. Again, we check if the runlevel was changed successfully and then change back to runlevel 3, where we were before.

```
vpnserver:~# runlevel
N 2
vpnserver:~#init 5
INIT: Switching to runlevel: 5
(...)
vpnserver:~# runlevel
2 5
vpnserver:~#init 3
INIT: Switching to runlevel: 3
(...)
vpnserver:~# runlevel
5 3
vpnserver:~#
```

The system control for runlevels

The configuration file `/etc/inittab` contains the information the program `init` uses to determine:

- The standard runlevel (the runlevel in which the system after boot)
- Which directories are to be used for which runlevel
- Many other useful options (for example, what happens when you press *Ctrl*+*Alt*+*Delete*)

Here is an extract from the `inittab` file on Debian systems:

```
# /etc/inittab: init(8) configuration.
# $Id: inittab,v 1.91 2002/01/25 13:35:21 miquels Exp $
# The default runlevel.
id:2:initdefault:
(...)
```

The last line defines the standard runlevel after reboot—on this system it is runlevel 2, and the following comments indicate where `init` shows how the runlevels on this Debian system are supposed to work:

```
(...)
# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.
(...)
```

Managing init scripts

The third important tool for managing server processes on Debian Linux is `update-rc.d`. This Perl script can check, create, and delete `init` scripts that are suitable for your system configuration.

Options for update-rc.d	Explanation
`update-rc.d <service> <options> <action>`	Configures the links in your `init` directories to your needs (according to the options passed)
`update-rc.d -n <options>`	Dry-run mode, only shows what it would do
`update-rc.d <options> remove`	Removes the start/stop scripts listed in `options`
`update-rc.d -f <options>`	Ignore warnings

Let's do some examples. The command `update-rc.d -n openvpn remove` removes all links to OpenVPN, but not really, only in a dry run to test if there would be problems. After this command, OpenVPN would not be started again in any runlevel. In our example, we encounter a little problem, which can easily be fixed by the 'force' switch `-f`. The `update-rc.d -n -f openvpn remove` gives us a list of files that would be deleted.

```
vpnserver:/etc/rc3.d# update-rc.d -n openvpn remove
update-rc.d: /etc/init.d/openvpn exists during rc.d purge (use -f to
force)
vpnserver:/etc/rc3.d# update-rc.d -n -f openvpn remove
update-rc.d: /etc/init.d/openvpn exists during rc.d purge (continuing)
 Removing any system startup links for /etc/init.d/openvpn ...
   /etc/rc0.d/K20openvpn
   /etc/rc1.d/K20openvpn
   /etc/rc2.d/S16openvpn
   /etc/rc3.d/S16openvpn
   /etc/rc4.d/S16openvpn
   /etc/rc5.d/S16openvpn
   /etc/rc6.d/K20openvpn
vpnserver:/etc/rc3.d# ls -l /etc/rc2.d/S16openvpn

lrwxrwxrwx  1 root root 17 2005-09-04 16:23 /etc/rc2.d/S16openvpn ->
../init.d/openvpn

vpnserver:/etc/rc3.d#
```

As you can see in the last line, the files are still there. Repeat these steps without the option `-n`, and the links will be deleted permanently.

`update-rc.d` can also create the links for you. Its syntax is easy, as shown:

```
update-rc.d <options> <service name><start/stop><service
number><runlevel>
```

Thus the following command is supposed to start `openvpn` with service number `16` in runlevel `3`:

```
vpnserver:/etc/rc3.d# update-rc.d -f openvpn start 16 3 .
 Adding system startup for /etc/init.d/openvpn ...
   /etc/rc3.d/S16openvpn -> ../init.d/openvpn
vpnserver:/etc/rc3.d# ls -l /etc/rc3.d/S16openvpn
lrwxrwxrwx  1 root root 17 2005-10-21 12:37 /etc/rc3.d/S16openvpn ->
../init.d/openvpn
vpnserver:/etc/rc3.d#
```

Now try to create the links that you have deleted above.
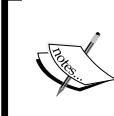
> If you want to revert to the default configuration of OpenVPN—like that after installation—simply enter `dpkg-reconfigure openvpn`. This program starts the post-install configuration dialog and process again and installs the default links to your runlevel directories.

Using SuSE's YaST module system services (runlevel)

SuSE systems have a sophisticated tool within YaST for the maintenance of your server's processes. The **System Services** editor can be found in the YaST module **System | System Services (Runlevel)**, here on OpenSuSE 11:
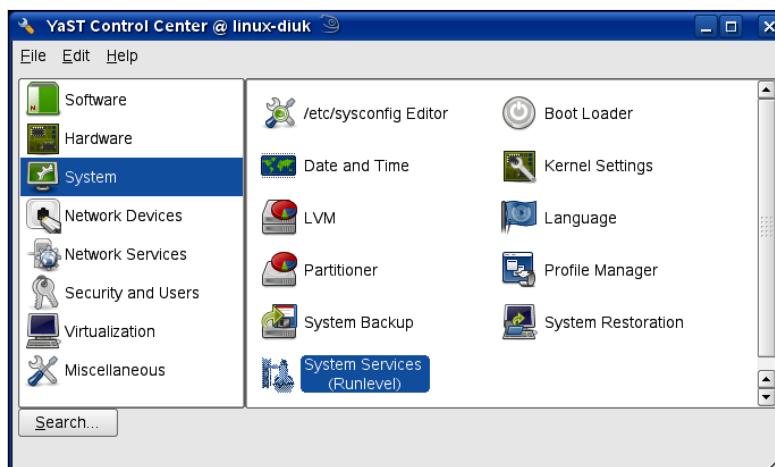
This runlevel editor can be run in two modes: **Simple Mode** and **Expert Mode**. In the standard **Simple Mode** you can simply switch services on or off, and YaST takes care of all considerations necessary for you. You are presented with a list of all available services, and two buttons, **Enable** and **Disable**.
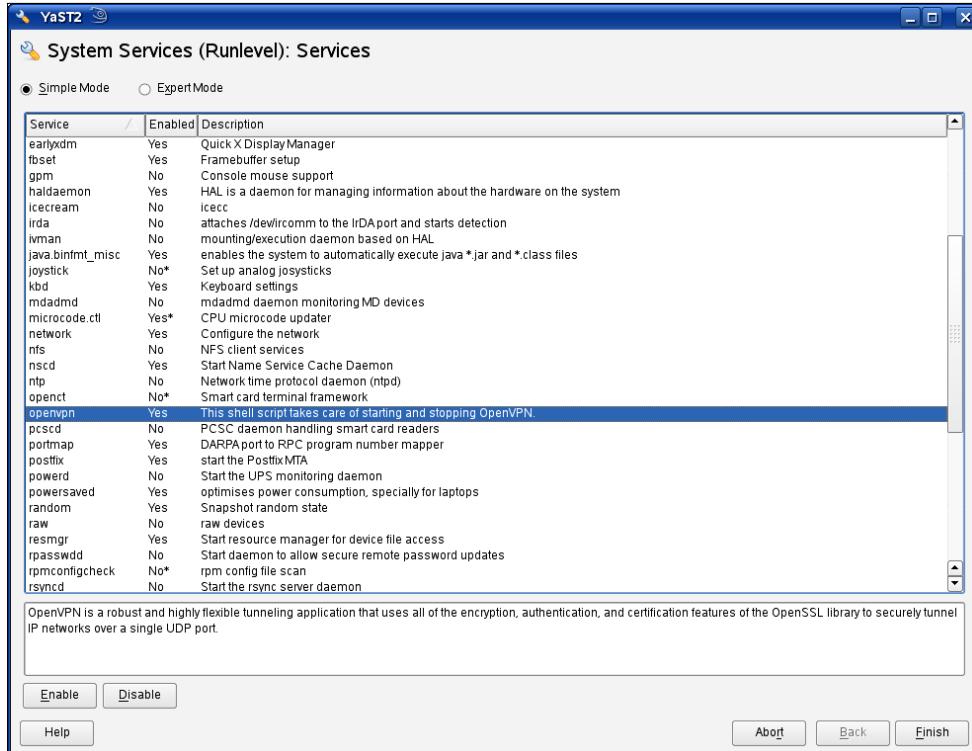
If you want to enable **openvpn**, then simply highlight it in the list and press **Enable**. The entry in the second column of the line **openvpn** in the list will change to **Yes**, and a status window reports **OpenVPN started**. Try to activate or deactivate the service **openvpn** several times.

Even though the **Simple Mode** is a convenient and fast method to retrieve an overview of the running services, there may be some disadvantages caused by the standard settings. In **Expert Mode** you can explicitly define the runlevels in which the different services will be started. You will see a separate column for each runlevel and a list of checkboxes with which you can easily activate the service in a single runlevel. Select **openvpn** in the list and activate it in runlevel 3 by activating the checkbox. Do this with the mouse or by simply entering *Alt+3*.

In either mode, click on **Finish** to activate your changes.

Troubleshooting firewall issues

Windows XP and SuSE Linux have firewall systems installed that are activated automatically after installation. Like most (personal or desktop) firewalls, these are configured to allow traffic originating from the local system and destined for the Internet or the local network. This configuration is sufficient for OpenVPN in almost every case. However, if your tunnels won't start and you receive messages announcing connection problems, it may be the fault of a misconfigured desktop firewall. As only SuSE Linux and Windows XP come with preinstalled firewalls which may conflict with OpenVPN, we will learn how to quickly deactivate them for testing purposes.

Deactivating the Windows XP service pack 2 firewall

On Windows XP with service pack 2, you will find the firewall configuration as an entry in the **Control Panel**. If you have service pack 2 installed, you will find an icon **Windows Firewall** in the list of available control panel modules, as shown in the following screenshot:
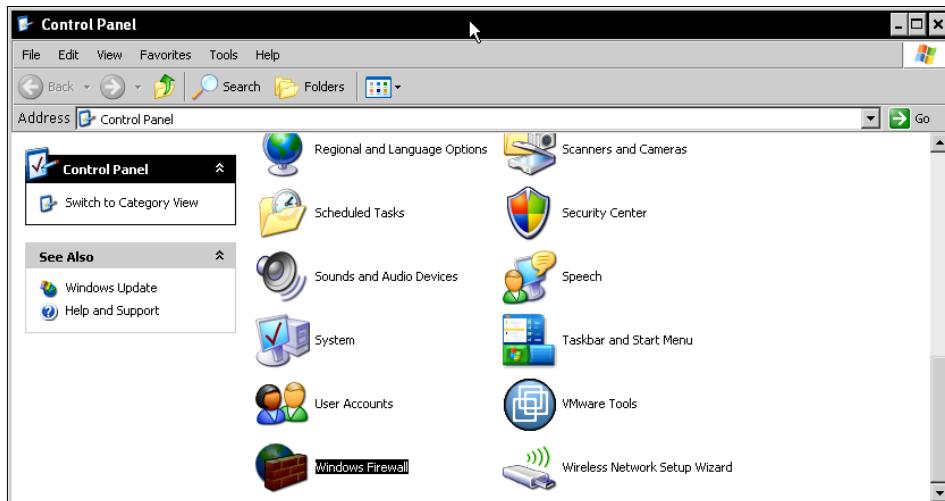
Double-click on the **Windows Firewall** icon to start the firewall configuration dialog. A window like this will appear:
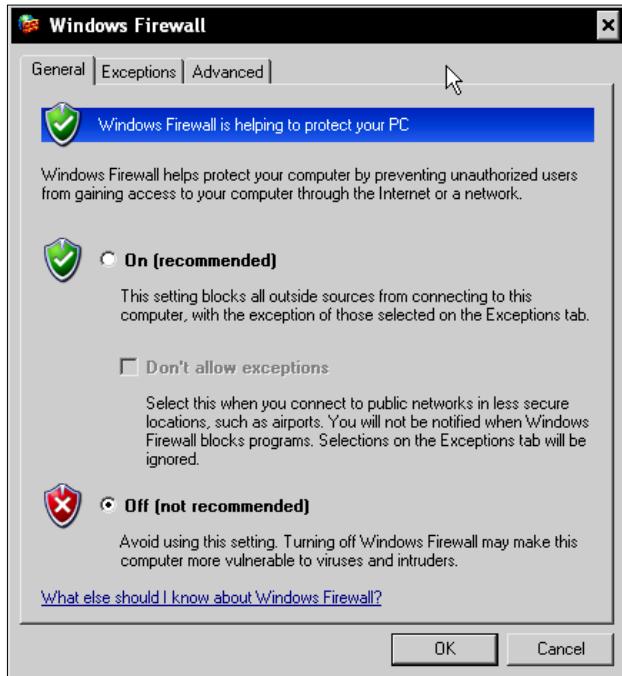
Activate the button **Off (not recommended)** to deactivate the **Windows Firewall**. Click on **OK** to finish the setup. Your Windows system is unprotected now.

> It is considered unwise to have a running Windows system without a firewall. But for our OpenVPN test-bed, this is acceptable. Please do not use this in production environments. In Chapter 8, we will deal with the correct firewall setup for an OpenVPN host.

If you do not want to deactivate your Windows firewall, then you can explicitly allow OpenVPN access to the Internet. If you start an OpenVPN connection, then you may be asked by your firewall software:

This **Windows Security Alert** dialog informs you that a local program called **openvpn** (strange, isn't it?) wants to accept connections from the Internet. Click on **Unblock** here and OpenVPN should work fine with the Windows firewall.

Stopping the SuSE firewall

On SuSE Linux you can use YaST to deactivate your desktop firewall. Start YaST from the main menu and enter your root password. Change to the **Security and Users** module, and left-click on the **Firewall** icon. The following dialog is opened:
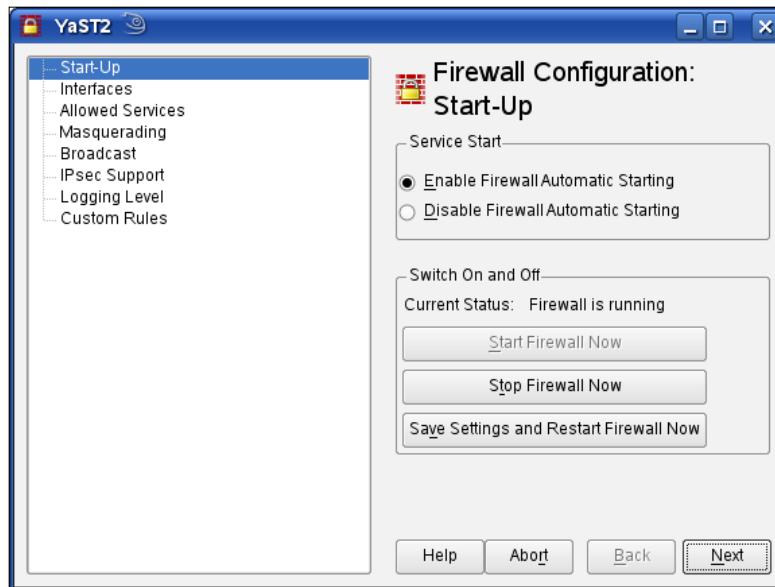
This window will show the current state of your firewall. Depending on your settings (and your selections during installation), it may be active or inactive and can be started manually or automatically. Set your firewall configuration as in the previous screenshot, which means that it is not started automatically, and it is not running:

- Click on the **Stop Firewall Now** button to stop the firewall on your SuSE system.

- Activate the button **Disable Firewall Automatic Starting** to prevent the firewall from being started at boot time.

> Even though there are no viruses and fewer security issues related to Linux systems, you should always protect your systems with a firewall. Consider the deactivation of the firewall only reasonable for testing purposes.

If you have a different firewall system running on your OpenVPN host, you will have to check the software documentation. The following hint may be helpful.

> Standard OpenVPN configuration initializes connections on UDP port 1194. If you want your system to answer OpenVPN connection requests, then you have to allow this port.

Summary

In this chapter we have successfully configured our first tunnel. We have connected Windows and Linux systems and safely transferred the necessary encryption keys using WinSCP. We have learnt to use the tool `dos2unix` to correct the plaintext files exchanged. After that, we tested the tunnels and activated them at boot time on both systems, including a short introduction to the Linux `init` system and runlevels. The last topic we discussed was Windows and SuSE Linux firewall issues, including stopping and deactivating these firewalls.

8

Setting Up OpenVPN with X.509 Certificates

In this chapter, we will create X.509 server and client certificates for use with OpenVPN. We will create a certificate authority, and sign and distribute new certificates. We will use `easy-rsa`, which comes with OpenVPN and exists for both Windows and Linux. This tool allows creation and administration of certificates that have to be transferred to the machines that are supposed to take part in the VPN.

Creating certificates

In the last chapter we successfully set up our first tunnels using pre-shared keys with static encryption, but in the initial chapters we learned why X.509 certificates provide a much better level of security than pre-shared keys do. There is, however, slightly more work to be done to set up and connect two systems with certificate-based authentication. The following five steps have to be accomplished:

1. Create a CA certificate for your CA with which we will sign and revoke client certificates.

2. Create a key and a certificate request for the clients.

3. Sign the certificate request using the CA certificate thereby making it valid.

4. Provide keys and certificates to the VPN partners.

5. Change the OpenVPN configuration so that OpenVPN will use the certificates and keys, and restart OpenVPN.

There are several ways to accomplish these steps. The `easy-rsa` is a command-line tool that comes with OpenVPN, and exists on both Linux and Windows. On Windows systems you could create certificates by clicking on the batch files in the Windows Explorer, but starting the batch files at the command-line prompt should be the better solution. On Linux you type the full path of the scripts, which share the same name as on Windows, simply without the extension `.bat`.

Certificate generation on Windows Server 2008 with easy-rsa

The following description has been tested on Windows 2000, Windows Vista, and Windows Server 2008. Open the Windows Explorer and change to the directory `C:\Program Files\OpenVPN\easy-rsa\`. The Windows version of `easy-rsa` consists of thirteen files. On Linux systems, you will have to check your package management tools to find the right path to the `easy-rsa` scripts. On Debian Linux, you will find them in `/usr/share/doc/openvpn/examples/easy-rsa/`, SuSE users may want to have a look in `/usr/share/openvpn/easy-rsa`.
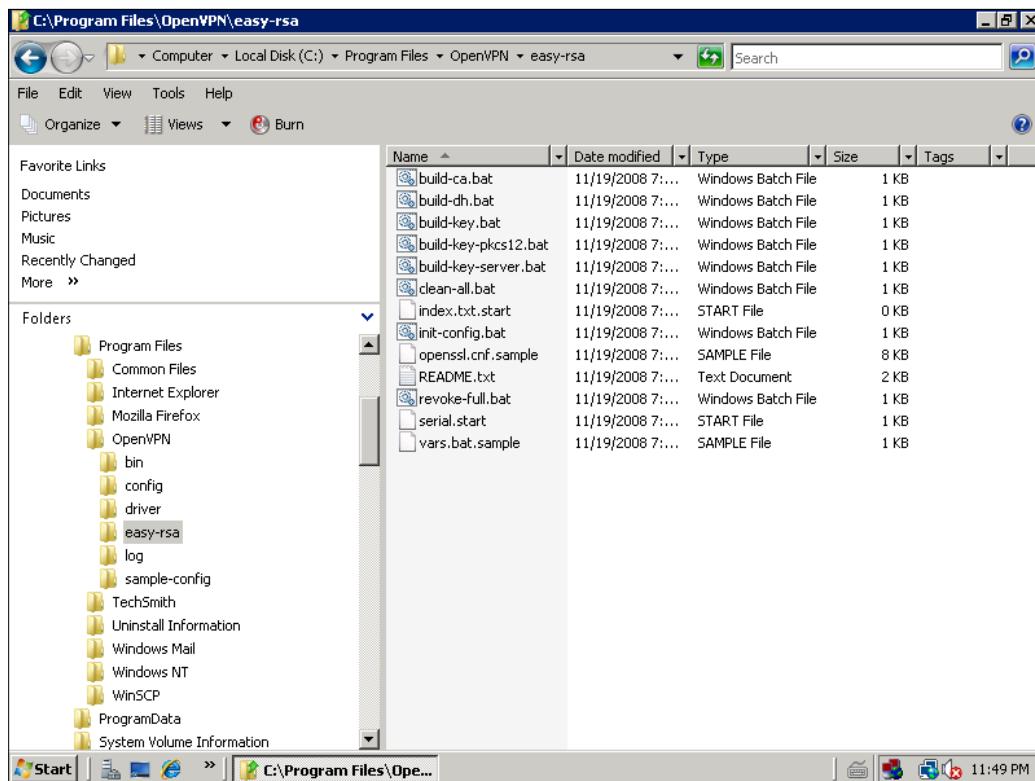
You find there are eight batch files, four configuration files, and a **README** (which is actually not that helpful). However, we must now create a directory called `keys`, copy the files `serial.start` and `index.txt.start` into it, and rename them to `serial` and `index.txt` respectively. The keys and certificates created by `easy-rsa` will be stored in this directory. These files are used as a database for certificate generation.

Now we let `easy-rsa` prepare the standard configuration for our certificates. Double-click on the file `C:\Program Files\ OpenVPN\easy-rsa\init-config.bat` or start this batch file at a command-line prompt. It simply copies the template files `vars.bat.sample` to `vars.bat` and `openssl.cnf.sample` to `openvpn.ssl`. While the file `openssl` is a standard OpenSSL configuration, the file `vars.bat` contains variables used by OpenVPN's scripts to create our certificates, and needs some editing in the next step.

Setting variables—editing vars.bat

Right-click on the `vars.bat` file's icon and select **Edit** from the menu.
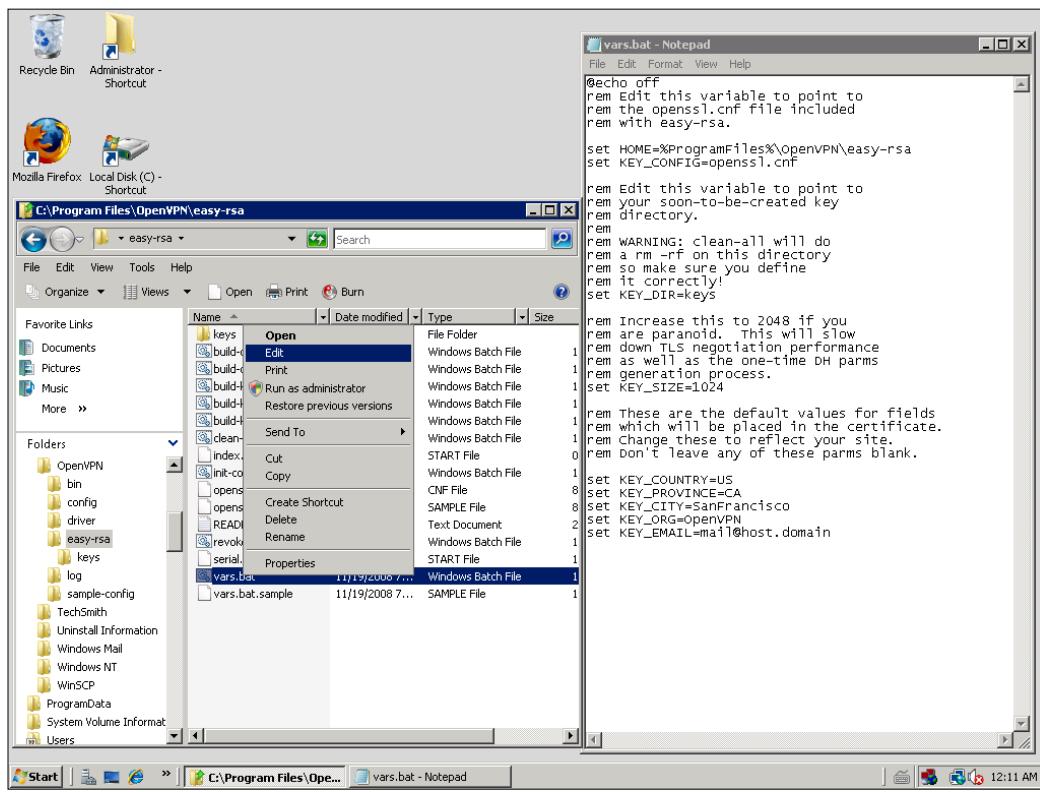
In this file, several parameters are set that are used by the certificate generation scripts later. The following table gives a quick overview of the entries in the file:

Entry in vars.bat	Function
`set HOME=%ProgramFiles%\OpenVPN\easy-rsa`	The path to the directory where `easy-rsa` resides.
`set KEY_CONFIG=openssl.cnf`	The name of the OpenSSL configuration file.
`set KEY_DIR=keys`	The path to the directory where the newly generated keys are stored—relative to `$HOME` as set above.
`set KEY_SIZE=1024`	The length of the SSL key. It often makes sense to increase this to 2048.

Setting the last five entries to your needs might be very helpful later. Every time we generate a certificate, `easy-rsa` will ask (among others) for these five parameters, and give a suggestion that could be accepted simply by pressing *Enter*. The better the default values set here in `vars.bat` fit our needs, the less typing work we will have later. I leave it up to you to change these settings here.

The next step is easy. Run `vars.bat` to set the variables. Even though you could simply double-click on its explorer icon, I recommend that you run it in a shell window. Select the entry **Run** from Windows' main menu, type `cmd.exe`, and change to the `easy-rsa` directory by typing `cd "C:\Program Files\OpenVPN\easy-rsa\"` and pressing *Enter*. As next step, proceed in exactly the same way as we would do on a Linux system (except for the `.bat` extensions).

Creating the Diffie-Hellman key

Now it is time to create the keys that will be used for encryption, authentication, and key exchange. For the latter, a Diffie-Hellman key is used by OpenVPN. The Diffie-Hellman key agreement protocol enables two communication partners to exchange a secret key safely. No prior secrets or safe lines are needed. A special mathematical algorithm guarantees that only the two partners know the used shared key. If you would like to know exactly what this algebra is about, have a look at this web site: `http://www.rsasecurity.com/rsalabs/node.asp?id=2248`.

The `easy-rsa` provides a script (batch) file that generates the key for you: `C:\Program Files\OpenVPN\easy-rsa\build-dh.bat`. Start it by typing `build-dh.bat`. A Diffie-Hellman key is being generated. The batch file tells you, **This is going to take a long time**, which is only true if your system is really old or if you are not patient enough. However, on modern systems some minutes may be a horribly long time span!
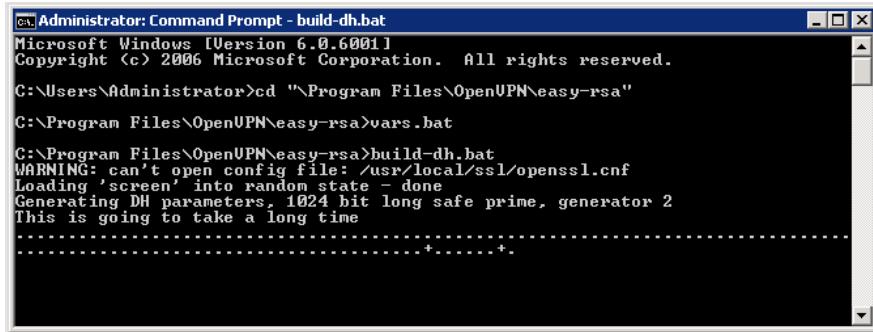
Building the certificate authority

OK, now it's time to generate our first CA.

Enter `build-ca.bat`. This script generates a self-signed certificate for a CA. Such a certificate can be used to create and sign client certificates and thereby authenticate other machines. Depending on the data you entered in your `vars.bat` file, `build-ca.bat` will suggest different default parameters during the process of generating this certificate. Five of the seven listed lines are taken from the variables set in `vars.bat`. If you edited these parameters, a simple return will do here and the certificate for the CA is generated in the `keys` directory. Let's now have a look there. Point your Windows Explorer to it and you will see that the following files have been created:
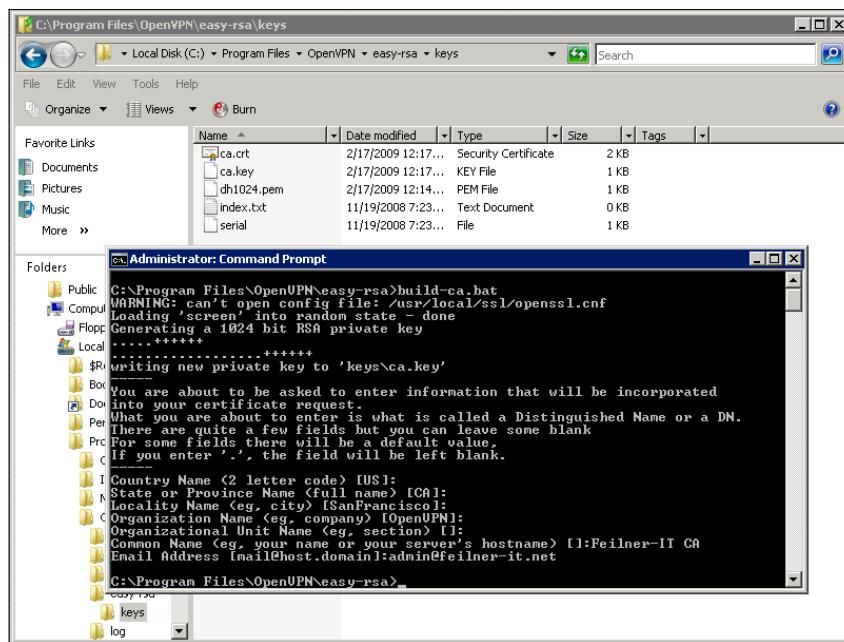
The `build-ca.bat` script of `easy-rsa` has created a certificate file `ca.crt` and a CA key file `ca.key`. The `build-dh.bat` script has built a `dh1024.pem` Diffie-Hellman key file, where the length of this key is part of the filename—if you use 2048-bit keys, this file will be named `dh2048.pem`. Really paranoid (but patient) readers may find a `dh4096.pem` file.

The file `ca.crt` is needed by all machines that are supposed to connect to your server, whereas the `dh2048.pem` file must only be available on the server.

> Please note that whoever owns the file `ca.key` (and `ca.crt`) is able to sign requests for your CA. Therefore, this file must be kept absolutely secret and should never leave the CA server. This file is essential and is the central key to your VPN. It should be kept protected on one computer strictly. Many experts advise you to use a dedicated machine without network connection (local login only) and strict access rules for this purpose.

Generating server and client keys

Our next step is to provide a VPN server certificate and a key, and have it signed from the CA. Or, to be more precise, we will create a certificate request that will be signed by the CA. An unsigned request cannot be used. Like a passport not stamped or unsigned by your local authority, there is no use for an unsigned certificate request. Again, batch files are provided to fulfill this task. `Start build-key-server.bat VPN-Server` at your command-line prompt. The parameter you give to this script is the template name used for the files. In this example, we will use `VPN-Server` as an example.

A 2048-bit private RSA key is generated. Again, the values derived from the parameters in your `vars.bat` are provided as default and can be accepted by simply pressing *Enter*. However, in the field `Common Name`, you should be very specific and enter a distinguished name for your VPN server. Every time you generate a certificate/key pair, you should enter the name for the machine you want to use this certificate/key pair on. It makes sense to use the same name you chose as command-line argument. As we will see later in this book, OpenVPN can have different configurations based on and distinguished by the value that you enter here, and choosing names skillfully here can save a lot of work later.

If you want, you can also enter some extra attributes, like a password that needs to be entered every time the certificate is used or an optional company name. However, if you enter a password here, no one (including no service) can set up a connection without this password. I leave it up to you to decide if this makes sense, if you are a little inclined to paranoia it will.

```
Administrator: Command Prompt                                      _ □ ×

C:\Program Files\OpenVPN\easy-rsa>build-key-server.bat UPN-Server
WARNING: can't open config file: /usr/local/ssl/openssl.cnf
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
.++++++
...................++++++
writing new private key to 'keys\UPN-Server.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [CA]:
Locality Name (eg, city) [SanFrancisco]:
Organization Name (eg, company) [OpenVPN]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:UPN-Server
Email Address [mail@host.domain]:ca-admin@feilner-it.net

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
WARNING: can't open config file: /usr/local/ssl/openssl.cnf
Using configuration from openssl.cnf
Loading 'screen' into random state - done
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'US'
stateOrProvinceName   :PRINTABLE:'CA'
localityName          :PRINTABLE:'SanFrancisco'
organizationName      :PRINTABLE:'OpenVPN'
commonName            :PRINTABLE:'UPN-Server'
emailAddress          :IA5STRING:'ca-admin@feilner-it.net'
Certificate is to be certified until Feb 14 23:24:00 2019 GMT (3650 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

C:\Program Files\OpenVPN\easy-rsa>_
```

Even though on Windows Server 2008 the Release Candidate of OpenVPN reports an error (`/usr/local/ssl/openssl.conf` not found), it seems to work without error. After the certificate request is generated, the batch script asks you if you want to have it signed by the CA. Simply enter *Y* twice, and the request is signed.

Let's again have a look at the `keys` directory. Three files whose name starts with `VPN-Server` have been generated: `VPN-Server.key`, `VPN-Server.crt`, and `VPN-Server.csr`. The file with the extension `.key` is the server key, the file with the extension `.crt` contains the server certificate, and the file `VPN-Server.csr` holds the certificate created in the step before.
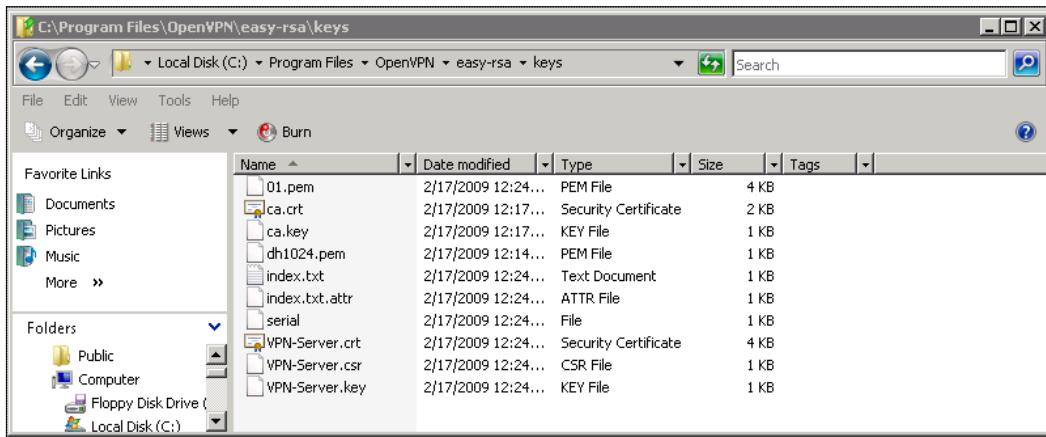
What does that mean now? Right, we have a certificate/key pair for our VPN server that tells everybody that the machine owning and using this pair is (or was) trusted by the CA we created before. With newer Windows systems a user can open the `.crt` file and have the information displayed in a nice dialogue:
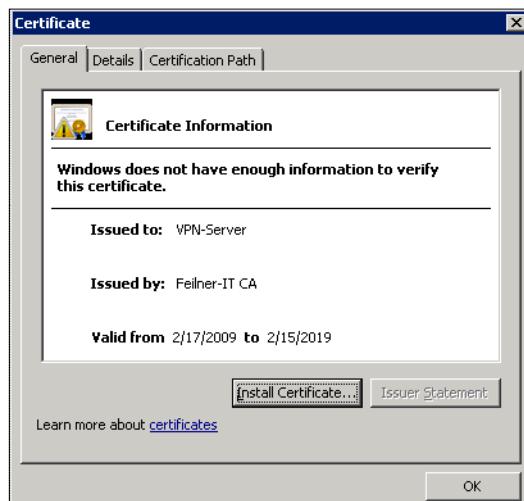
What a pity that nobody else knows about this authority up till now. Let's hurry to change this and create a certificate for the client.

Not very surprisingly, another batch file will help us here. It's called `build-key.bat` and you should give the name of the VPN client as a command-line parameter. I chose `VPN-Client` just to have a simple, recognizable name.
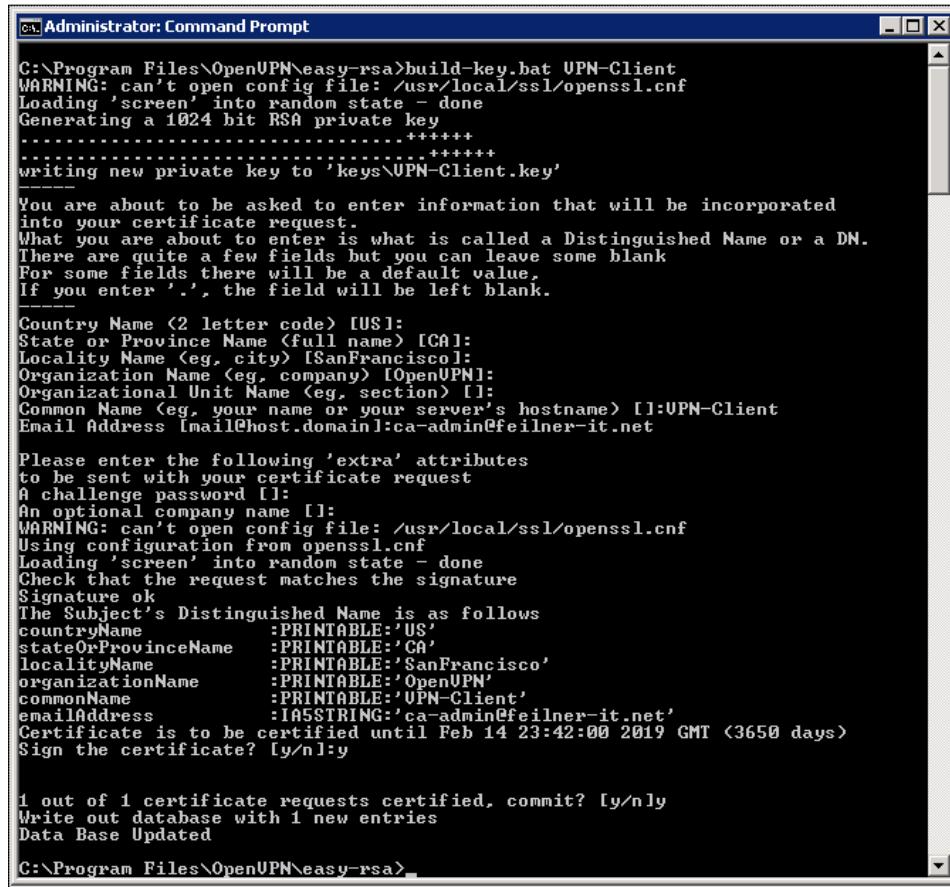
```
C:\Program Files\OpenVPN\easy-rsa>build-key.bat VPN-Client
WARNING: can't open config file: /usr/local/ssl/openssl.cnf
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
...................................++++++
.................................++++++
writing new private key to 'keys\VPN-Client.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [CA]:
Locality Name (eg, city) [SanFrancisco]:
Organization Name (eg, company) [OpenVPN]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:VPN-Client
Email Address [mail@host.domain]:ca-admin@feilner-it.net

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
WARNING: can't open config file: /usr/local/ssl/openssl.cnf
Using configuration from openssl.cnf
Loading 'screen' into random state - done
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName            :PRINTABLE:'US'
stateOrProvinceName    :PRINTABLE:'CA'
localityName           :PRINTABLE:'SanFrancisco'
organizationName       :PRINTABLE:'OpenVPN'
commonName             :PRINTABLE:'VPN-Client'
emailAddress           :IA5STRING:'ca-admin@feilner-it.net'
Certificate is to be certified until Feb 14 23:42:00 2019 GMT (3650 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

C:\Program Files\OpenVPN\easy-rsa>
```

> Create another (or many) signed certificate(s) for the other tunnel partner(s) with the batch script `build-key.bat`.

Distributing the files to the VPN partners

Again, in your `keys` directory you will find three new files `VPN-Client.csr`, `VPN-Client.key`, and `VPN-Client.crt`, two of which need to be transferred to the VPN partner. Do you know which ones, already? The following table gives an overview of the files we have created up to now and the ones that have to be transferred to our client.
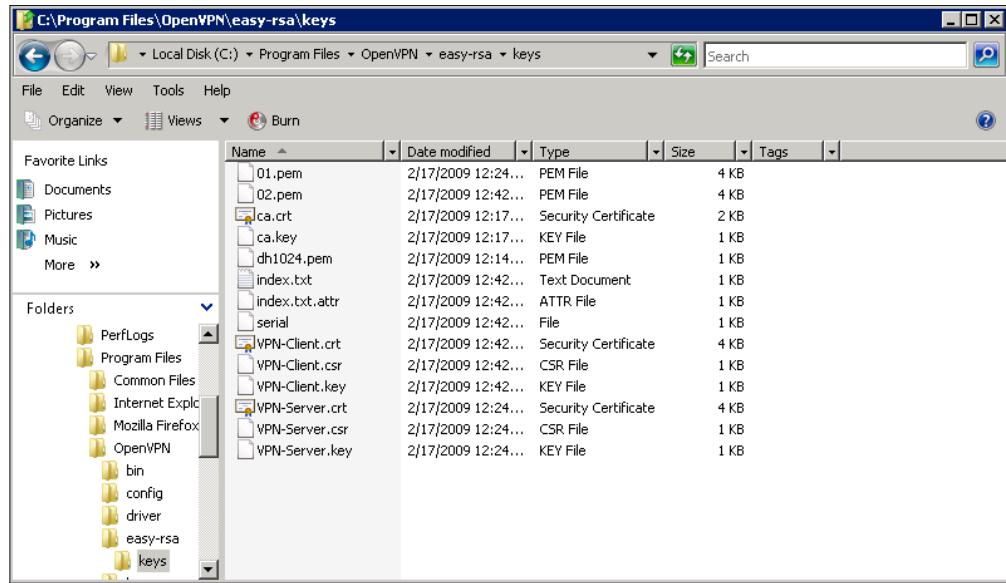

File	Location and purpose
`VPN-Server.crt`	Signed certificate of the `VPN-Server`, must be on `VPN-Server`.
`VPN-Server.key`	Private RSA key of the `VPN-Server`, must be on `VPN-Server`.
`VPN-Server.csr`	Certificate signing request of `VPN-Server`, can be deleted.
`VPN-Client.crt`	Signed certificate of the `VPN-client`, must be on `VPN-Client`.
`VPN-Client.key`	Private RSA key of the `VPN-client`, must be on `VPN-Client`.
`VPN-Client.csr`	Certificate signing request of `VPN-Client`, can be deleted.
`ca.crt`	CA certificate, must be available on both machines.
`ca.key`	The key to the CA, must be kept only on CA. Must be kept very secret, as it can be used to sign valid certificates that allow access to your services and networks.

OK, we must transfer three files, `VPN-Client.crt`, `VPN-Client.key`, and `ca.crt` to our VPN client. Maybe you want to use the PKCS12 file format, which bundles all three files into one, but the standard consists of three separate files. In any case, we have to use a secure transfer method to do so. If the client is a Linux machine, we will use WinSCP to accomplish that. Start WinSCP and change to the remote directory `/etc/openvpn` on the Linux machine. Create a directory `/etc/openvpn/keys`. Although this is not really necessary, a reasonable directory structure is very helpful and makes administration much easier.

Copy the three files by drag-and-drop to the remote directory. Then create a local directory called `keys` under `C:\Program Files\OpenVPN\config\` and copy the three files `VPN-Server.crt`, `VPN-Server.key`, `ca.crt`, and `dh1024.pem` into this directory. These are the files needed on the VPN server.

The following image shows the server's key directory below `easy-rsa` on the left and the Linux client's configuration directory `/etc/openvpn` on the right.
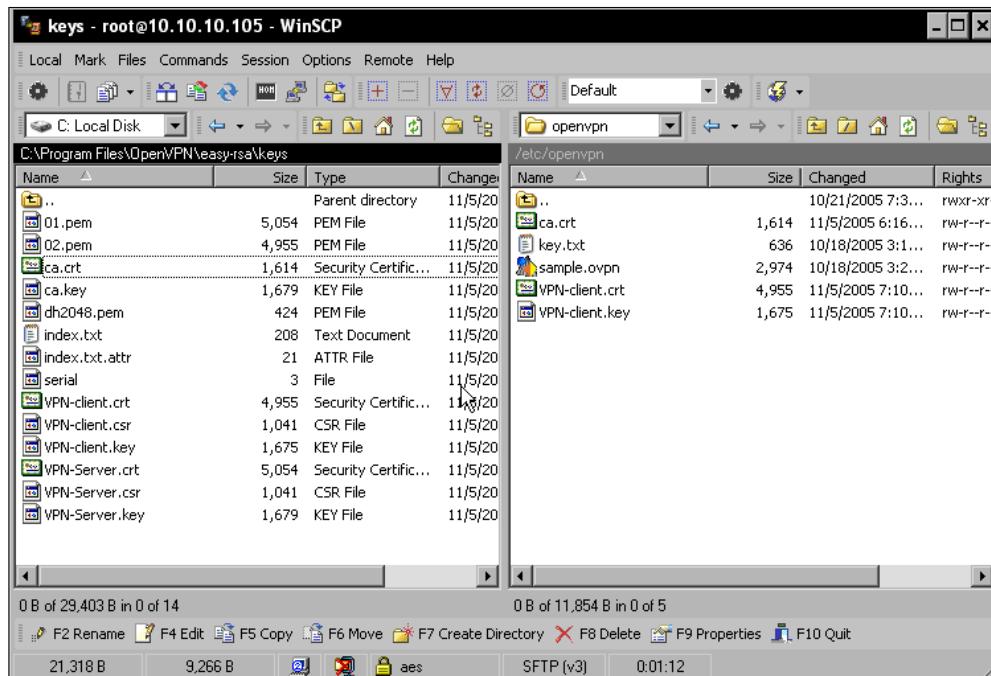
As a last step, we must adapt our configuration files so that OpenVPN uses X.509 certificates and knows where to find them.

Configuring OpenVPN to use certificates

Open the configuration file in your favorite editor, of course you may also use Notepad.
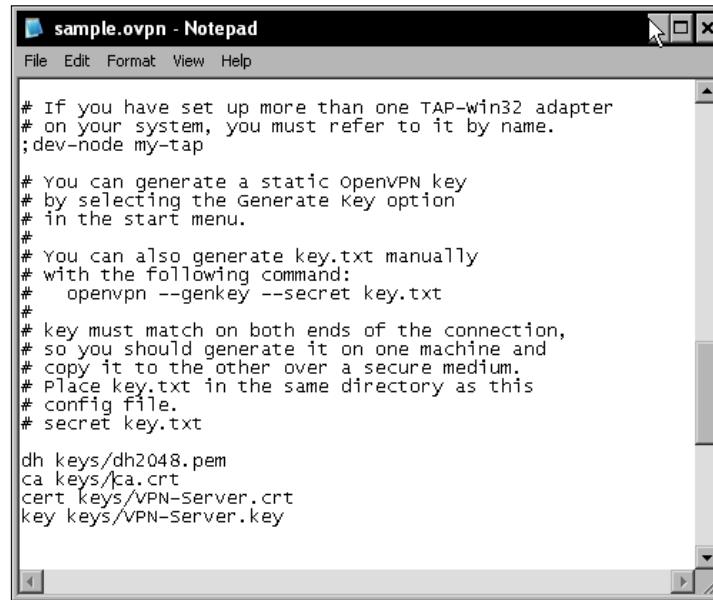
All you have to do here is put # in front of the entry **secret key.txt**, which we adapted in our last chapter, and add the following five entries:

Entry in config file	Function
`tls-server`	OpenVPN will run in TLS-server mode (on a client you will have to add TLS-client)
`dh keys/dh2048.pem`	Use the Diffie-Hellman key stored in `keys/dh2048.pem`
`ca keys/ca.crt`	Use the CA certificate in `keys/ca.crt`
`cert keys/VPN-Server.crt`	Use my certificate in `keys/VPN-Server.crt`

In my test-bed network, where the local net is 10.10.10.0/24 and the tunnel network is 10.3.0.0/24, the simplest possible configuration file (`C:\Program Files\OpenVPN\config\sample.ovpn` on Windows) for an X.509-enabled OpenVPN server is:

```
dev tap
ifconfig 10.3.0.1 255.255.255.0
tls-server
dh keys/dh1024.pem
ca keys/ca.crt
cert keys/VPN-Server.crt
key keys/VPN-Server.key
```

If you are using a Linux machine as a server, you should adjust the file permissions with the command `chmod go-x /etc/openvpn/keys/*`. This command makes the keys and certificates only readable for Root. Now let's test this configuration:

```
vpnserver:/etc/openvpn# openvpn --config cert-server.conf
Tue Feb 17 00:00:00 2009 OpenVPN 2.0.9 i486-pc-linux-gnu [SSL] [LZO]
[EPOLL] built on Sep 20 2007
Tue Feb 17 00:00:00 2009 IMPORTANT: OpenVPN's default port number
is now 1194, based on an official port number assignment by IANA.
OpenVPN 2.0-beta16 and earlier used 5000 as the default port.
Tue Feb 17 00:00:00 2009 TUN/TAP device tap0 opened
Tue Feb 17 00:00:00 2009 ifconfig tap0 10.3.0.1 netmask 255.255.255.0
mtu 1500 broadcast 10.3.0.255
Tue Feb 17 00:00:00 2009 UDPv4 link local (bound): [undef]:1194
Tue Feb 17 00:00:00 2009 UDPv4 link remote: [undef]
```

Everything is fine, the server is running and waiting for clients to connect.

And the simplest possible configuration file for a client is:

```
remote 10.10.10.103
dev tap
tls-client
ifconfig 10.3.0.2 255.255.255.0
dh keys/dh2048.pem
ca keys/ca.crt
cert keys/VPN-Client.crt
key keys/VPN-Client.key
```

> Change the OpenVPN configuration on the two systems to the values above. Rename the Linux file's extension to `.conf` and make sure the Windows file's name ends with `.ovpn`.

It's as simple as that. And the best thing is that this configuration is the same on all platforms. Simply edit your openvpn configuration file on the Linux machine as in the previous example, restart your openvpn services, and the tunnels will come up, but this time safe and secure with X.509 certificates.

```
vpnserver:/etc/openvpn# openvpn --config cert-server.conf
Tue Feb 17 00:18:24 2009 OpenVPN 2.0.9 i486-pc-linux-gnu [SSL] [LZO]
[EPOLL] built on Sep 20 2007
Tue Feb 17 00:18:24 2009 IMPORTANT: OpenVPN's default port number
is now 1194, based on an official port number assignment by IANA.
OpenVPN 2.0-beta16 and earlier used 5000 as the default port.
Tue Feb 17 00:18:24 2009 TUN/TAP device tap0 opened
Tue Feb 17 00:18:24 2009 ifconfig tap0 10.3.0.1 netmask 255.255.255.0
mtu 1500 broadcast 10.3.0.255
Tue Feb 17 00:18:24 2009 UDPv4 link local (bound): [undef]:1194
Tue Feb 17 00:18:24 2009 UDPv4 link remote: [undef]
Tue Feb 17 00:18:26 2009 [VPN-Client] Peer Connection Initiated with
10.10.10.111:1194
Tue Feb 17 00:18:27 2009 Initialization Sequence Completed
```

If you do not believe, check it by the command ping on either side of the tunnel.

```
mfeilner@vpnserver:~$ ping 10.3.0.2
PING 10.3.0.2 (10.3.0.2) 56(84) bytes of data.
64 bytes from 10.3.0.2: icmp_seq=1 ttl=128 time=27.0 ms
64 bytes from 10.3.0.2: icmp_seq=2 ttl=128 time=26.7 ms
64 bytes from 10.3.0.2: icmp_seq=3 ttl=128 time=26.9 ms
64 bytes from 10.3.0.2: icmp_seq=4 ttl=128 time=25.7 ms
64 bytes from 10.3.0.2: icmp_seq=5 ttl=128 time=29.2 ms

--- 10.3.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 25.701/27.151/29.253/1.164 ms
mfeilner@vpnserver:~$
```

> Use ping to test the tunnel once OpenVPN reports 'Peer Connection Initiated'.

Using easy-rsa on Linux

We have learned earlier that `easy-rsa` is a part of OpenVPN and available on all platforms. Because we have worked through the generation of certificates on Windows, we will now have a look at the same process on a Linux system. On Debian Linux, `easy-rsa` can be found in the directory `/usr/share/doc/openvpn/examples/easy-rsa`. Start a root shell and change to this directory:

```
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa# ls -l
insgesamt 80
drwxr-xr-x 2 root root 4096 2008-03-10 00:37 2.0
-rwxr-xr-x 1 root root  242 2005-11-01 11:06 build-ca
-rwxr-xr-x 1 root root  228 2005-11-01 11:06 build-dh
-rwxr-xr-x 1 root root  529 2005-11-01 11:06 build-inter
-rwxr-xr-x 1 root root  516 2005-11-01 11:06 build-key
-rwxr-xr-x 1 root root  424 2005-11-01 11:06 build-key-pass
-rwxr-xr-x 1 root root  695 2005-11-01 11:06 build-key-pkcs12
-rwxr-xr-x 1 root root  662 2005-11-01 11:06 build-key-server
-rwxr-xr-x 1 root root  466 2005-11-01 11:06 build-req
-rwxr-xr-x 1 root root  402 2005-11-01 11:06 build-req-pass
-rwxr-xr-x 1 root root  280 2005-11-01 11:06 clean-all
-rw-r--r-- 1 root root  264 2005-11-01 11:06 list-crl
-rw-r--r-- 1 root root  268 2005-11-01 11:06 make-crl
-rw-r--r-- 1 root root 7487 2005-11-01 11:06 openssl.cnf
-rw-r--r-- 1 root root 2619 2005-11-01 11:06 README.gz
-rw-r--r-- 1 root root  268 2005-11-01 11:06 revoke-crt
-rwxr-xr-x 1 root root  593 2005-11-01 11:06 revoke-full
-rwxr-xr-x 1 root root  411 2005-11-01 11:06 sign-req
-rw-r--r-- 1 root root 1266 2005-11-01 11:06 vars
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa#
```

As you can see, this directory contains all the scripts that we have used on Windows, and some more too. On Linux, there is a file called `vars`, which is a shell script that contains all the information and variables like its Windows counterpart, `vars.bat`.

> On Debian Linux, `easy-rsa` is located in `/usr/share/doc/openvpn/examples/easy-rsa`. Start a root shell and change to this directory.

Preparing variables in vars

Open `vars` with your favorite editor and change the certificate values to fit your needs. Don't forget to point the entry **export KEY_DIR** to an existing directory or create the directory `/usr/share/doc/openvpn/examples/easy-rsa/keys`. Create the two files `index.txt` and `serial` in this directory before proceeding.

On Windows, `vars.bat` is a batch file that simply is executed, on Linux it is **sourced**, which means that the shell reads this file and sets the environment variables you defined in it—a very common way to read configuration files on Linux. The command for this purpose is called `source`, and its abbreviation is simply a dot.

Now type `source vars` or simply `.vars` to have your shell read the configuration variables you edited.

```
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa# . vars
NOTE: when you run ./clean-all, I will be doing a rm -rf on /usr/
share/doc/openvpn/examples/easy-rsa/keys
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa#
```

The note you receive is important. In this directory, there is a script called `clean-all`, which removes all old configurations and keys you created previously from the `keys` directory you enter in `vars`. If you want to execute `clean-all`, be sure to back up all files you might need later on. Normally there should be no need to run `clean-all`.

Creating the Diffie-Hellman key and the certificate authority

As our next step we will create a Diffie-Hellman key with the script `build-dh`. On most Linux systems, the working directory is not in the path of the user root, so you have to invoke it with `./build-dh`.

```
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa# ./build-dh
Generating DH parameters, 1024 bit long safe prime, generator 2
This is going to take a long time
....+........+.+......+......................+......................
....................+.
(...)
```

Now your system might be occupied for some time, busily calculating a 1024-bit prime number. If you want to set the key size to 2048, have a look in `/usr/share/doc/openvpn/examples/easy-rsa/vars`, as we did on Windows. And once we're ready again, create the certificate for the CA.

```
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa# ./build-ca
Generating a 2048 bit RSA private key
.......................................++++++
.............++++++
writing new private key to 'ca.key'
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or
a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:
State or Province Name (full name) [BY]:
Locality Name (eg, city) [Regensburg]:
Organization Name (eg, company) [Feilner-IT]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:CA-Server
Email Address [security@feilner-it.net]:
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa#  ls -l keys
total 12
-rw-r--r--  1 root root 1245 2005-11-20 00:17 ca.crt
-rw-------  1 root root  887 2005-11-20 00:17 ca.key
-rw-r--r--  1 root root  245 2005-11-20 00:14 dh1024.pem
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa#
```

Certificate and key have been created in the directory `/usr/share/doc/openvpn/examples/easy-rsa/keys`.

Creating the first server certificate/key pair

Now we can create the first certificate/key pair for our first VPN server. Remember, that the `Common Name` can be used to recognize a client authenticating with this certificate, so choose a distinguishing name here. After generation of the certificate, we are asked if we want to sign the certificate using the CA's certificate.

Start creation of a certificate/key pair called `VPN-Server` with the command `/build-key VPN-Server`.

```
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa# ./build-key VPN-
Server
Generating a 1024 bit RSA private key
......++++++
...++++++
writing new private key to 'VPN-Server.key'
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or
a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:
State or Province Name (full name) [BY]:
Locality Name (eg, city) [Regensburg]:
Organization Name (eg, company) [Feilner-IT]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:VPN-Server
Email Address [security@feilner-it.net]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /usr/share/doc/openvpn/examples/easy-rsa/
openssl.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'DE'
stateOrProvinceName   :PRINTABLE:'BY'
localityName          :PRINTABLE:'Regensburg'
organizationName      :PRINTABLE:'Feilner-IT'
commonName            :PRINTABLE:'VPN-Server'
emailAddress          :IA5STRING:'security@feilner-it.net'
Certificate is to be certified until Nov 17 23:40:04 2015 GMT (3650
days)
Sign the certificate? [y/n]:y
```

```
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa#
```

Enter a distinguishing `Common Name` and enter *Y* twice to have the certificate signed. The certificate and key file are created in `/usr/share/doc/openvpn/examples/easy-rsa/keys`.

```
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa# ls -l keys/
total 44
-rw-r--r--  1 root root 3653 2005-11-20 00:40 01.pem
-rw-r--r--  1 root root 1233 2005-11-20 00:39 ca.crt
-rw-------  1 root root  887 2005-11-20 00:39 ca.key
-rw-r--r--  1 root root  245 2005-11-20 00:37 dh1024.pem
-rw-r--r--  1 root root  104 2005-11-20 00:40 index.txt
-rw-r--r--  1 root root   21 2005-11-20 00:40 index.txt.attr
-rw-r--r--  1 root root    0 2005-11-20 00:31 index.txt.old
-rw-r--r--  1 root root    3 2005-11-20 00:40 serial
-rw-r--r--  1 root root    3 2005-11-20 00:31 serial.old
-rw-r--r--  1 root root 3653 2005-11-20 00:40 VPN-Server.crt
-rw-r--r--  1 root root  688 2005-11-20 00:40 VPN-Server.csr
-rw-------  1 root root  887 2005-11-20 00:40 VPN-Server.key
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa#
```

Now we have the certificate for the CA and a certificate and key for the first OpenVPN machine.

Creating further certificates and keys

Let's repeat the last step for a second machine, which is called `VPN-client`:

```
vpnserver:/usr/share/doc/openvpn/examples/easy-rsa# ./build-key-server
VPN-Client
Generating a 1024 bit RSA private key
.........................++++++
(...)
```

That's it! Repeat the last command for every machine you want to equip with a certificate. You will find the certificate, key, and CA certificate in `/usr/share/doc/openvpn/examples/easy-rsa/keys` (or the path you specified in the file `vars`). Transfer these files to the machines involved in your VPN using a secure method. WinSCP works perfectly here, if you have Windows clients, the command-line tool `scp` (from the **sshd** package) is the best choice for data exchange between systems with SSH servers (most Linux/Unix systems).

Troubleshooting

If you run into problems, check the following:

- Ensure basic network connectivity between the two systems. Can they `ping` each other without problems? Are there firewalls involved between them?

- Disable all firewalls on both systems during testing the tunnels. We will later set them up properly. Remember that both Windows XP and SuSE activate their firewall solutions by default.

- OpenVPN and X.509 certificates need synchronized time on both systems for reasonable validity checks. For testing purposes you can set the time by hand. On Linux, the commands `date` and `hwclock` will help you. For the production environment a time server client should be set up. On Linux, `ntp` is probably the most common one, its homepage offers documentation: `http://www.eecis.udel.edu/~ntp/`.

- If you copy the files from a Windows machine to a Linux machine, remember to have `dos2unix` run and convert the end-of-line characters. The same applies to configuration files, certificates, and keys created on Linux and transferred to Windows—apply `unix2dos` before transfer. Depending on your Linux system and OpenVPN version, it may be necessary to change the file access permissions in the `keys` directory as follows:

```
vpnserver:~# cd /etc/openvpn/keys
vpnserver:/etc/openvpn/keys# ls -l
total 16
-rw-------  1 root root 1606 2005-11-05 09:54 ca.crt
-rw-------  1 root root 4948 2005-11-05 09:55 VPN-Client.crt
-rw-------  1 root root 1679 2005-11-05 09:55 VPN-Client.key
vpnserver:/etc/openvpn/keys#
```

- If file permissions are set less restrictively, some OpenVPN versions may refuse to start.

- Check the data you enter during the process of creating the certificates. Ensure that you have not misspelled anything and that there are no typos. Any character different in the certificates can cause the process of connecting the systems to fail.

If you have checked this, repeat the process of certificate generation with `easy-rsa` and enter your data carefully. Analyze the logfile entries in the Windows main menu and context menu of the OpenVPN GUI or have a look at the output of `openvpn` at the command line when invoked manually.

Summary

In this chapter we have used the scripts in the `easy-rsa` directory, provided with OpenVPN, to create a CA, a Diffie-Hellman key, and both keys, certificate requests, and keys for the two VPN partners. The client and server certificates were automatically signed during creation. After having them transferred to the VPN partner (Windows or Linux), we started the new, secure tunnel.

9

The Command openvpn and Its Configuration File

In this chapter we will have a look at the syntax of the command-line tool `openvpn`, which enables us to build tunnels quickly. By analyzing the standard configuration file we used to set up a tunnel with a pre-shared key, we will now dive into the depths of the configuration options of `openvpn`. This way, we will learn about basic tunnel network setup and control, compression, and debug output.

As a next step, the configuration file containing the certificate-based tunnel created in Chapter 8 will be in our focus. From then on we will go through several groups of parameters that can be given to `openvpn` (be it in a configuration file or at the command-line prompt). We will deal with examples for many of these parameters and look at scenarios where they might prove helpful. Parameters available in server and client mode, encryption, and Windows-specific options are explained.

Many of the following options are explained in detail on the manual page of OpenVPN. The explanations (especially in the tables) are close to the explanations in the man page, some details and examples have been added, some removed. If you feel unsure about some options, have a closer look at the man page, which is updated regularly on the web site.

A subsection at the end of this chapter deals with new features of version 2.1 of OpenVPN.

Syntax of openvpn

In the previous chapters we have invoked `openvpn` at the command line several times. On Windows, this is an easy way to get more detailed output during troubleshooting, on Linux it is the normal way to set up a tunnel quickly. And on both systems this is what lies beneath the services layer or the GUI tools.

OpenVPN on both Windows and Linux is called by start scripts that add special parameters to the command `openvpn`. Normally, there is (among others) the parameter `--config` (followed by a filename), which lets `openvpn` read a configuration file, on Linux normally a file in `/etc/openvpn/`. On Windows configuration files have the extension `.ovpn` and on Linux, `.conf`. The start scripts will read all configuration files in the configuration file directory and start the tunnels described in them. If you have three `.conf` files in your Linux configuration directory, `openvpn` will try to start three tunnels. The same applies for `.ovpn` files on Windows and if you double-click such a file on Windows, a tunnel should be started.

OpenVPN command-line parameters

Our first tunnel from the Linux system was configured in a configuration file transferred from the Windows VPN partner. OpenVPN had to be told where this configuration file is to be found, so we started it with `openvpn --config sample.ovpn`. We now know that the extension `.ovpn` is typical for the Windows version of OpenVPN. Basically, you could use any extension you like, but only tunnels described in `.ovpn` and `.conf` files will be started automatically. The Linux system would not start a tunnel described in this file automatically until you rename the file to a `.conf` extension (and restart the service).

Although, this was the first time we called `openvpn`, it already shows its syntax:

```
openvpn <option1> <parameter(s)> ... <optionn> <parameter(s)>
```

Parameters and options for OpenVPN are either stored in a configuration file or entered at the command-line prompt. Normally there is no difference between the name of the command-line option and the configuration file parameter, of course with the exception of the following parameters:

- `--config <file>`: Directs to the location of the configuration file
- `--help`: Gives you a brief introduction to the syntax of `openvpn`
- `--version`: Prints the installed version and copyright information

Parameter	Options	Function	Usage
config	<file>	Directs openvpn to the location of the configuration file	Command line only
help	-	Prints help and a list of options	Command line only
version	-	Prints the version of OpenVPN	Command line only

The following code extract shows the first lines of the output of openvpn --help:

```
vpnserver:/root# openvpn --help
OpenVPN 2.0.9 i486-pc-linux-gnu [SSL] [LZO] [EPOLL] built on Sep 20 20
07

General Options:
--config file   : Read configuration options from file.
--help          : Show options.
--version       : Show copyright and version information.
(...)
```

Using OpenVPN at the command line

In the course of this book we have already invoked openvpn several times from a command line. As a first example, we built a tunnel with a pre-shared key and a rather simple configuration file. Even though there are some other parameters set in the standard configuration file we used, the easiest command to start a tunnel with a static key is:

```
vpnserver:/etc/openvpn# openvpn --remote <IP of System B> --dev tun1
--ifconfig 10.3.0.1 10.3.0.2 --secret /etc/openvpn/key.txt
```

You see, it's very easy to connect two systems with an openvpn tunnel, when we know their IPs. All we need is a pre-shared key, a tunnel IP, and a decision on which device type to use.

If the second tunnel endpoint is a Linux system already provided with the pre-shared key /etc/openvpn/key.txt, then all we need to do to start our tunnel is enter the aforementioned command on system A, and enter the following command on system B:

```
/etc/openvpn# openvpn --remote <IP of System A> --dev tun1 --ifconfig
10.3.0.2 10.3.0.1 --secret /etc/openvpn/key.txt
```

That's all. Your tunnel is up and running. However, this tunnel is rather temporary and will be closed when you exit the shell around it. Nevertheless, you may consider it a convenient method to start and stop quick tunnels, especially for testing purposes.

The following table gives an overview on the parameters used here:

Parameter	Options	Function	Usage	Example
`remote`	`<hostname>` `<IP>`	Points to the other tunnel endpoint	Command line and config file	`--remote vpn. dyndns.org`
`dev`	`<device>`	Tells `openvpn` which network device (type) to use	Command line and config file	`--dev tun` `--dev tap`
`ifconfig`	For TUN devices: `<local IP>` `<remote IP>` For TAP devices: `<local IP>` `<subnet mask>`	Sets tunnel endpoints' virtual IPs and netmasks in the tunnel	Command line and config file	`--ifconfig 10.3.0.2 10.3.0.1` `--ifconfig 10.3.0.2 255.255.255.0`
`secret`	File containing the pre-shared key	Tells `openvpn` the location of the pre-shared key	Command line and config file	`--secret key.txt`
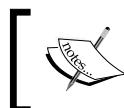
> The parameter `remote` specifies the machine on which the OpenVPN software is running and takes IPs or DNS entries as parameters.

In combination with DynDNS entries, we can build VPNs between dial-up network lines based on cheap DSL lines, on both sides of the tunnel!

Depending on the device type we select, `ifconfig` must set the IP/netmask combination differently. TUN devices are virtual point-to-point devices, and therefore `ifconfig` must be provided with the virtual IP of the other point-to-point partner. TAP devices, however, are virtual network devices and thus `ifconfig` needs a netmask for this virtual network segment.

In our above example, `openvpn` is called in `tun` mode and the parameter `ifconfig` is used with the options `10.3.0.2 10.3.0.1`. This means that a virtual point-to-point network is created between the two OpenVPN servers, with `10.3.0.1` and `10.3.0.2` as virtual endpoints.

The example below shows the correct `ifconfig` syntax for a `tap` device: `--ifconfig 10.3.0.2 255.255.255.0`. Since TAP devices provide virtual Ethernet segments, a netmask is needed.

> TUN devices provide routing mode and start a virtual point-to-point connection, TAP devices provide bridging mode and start a virtual network segment. The parameter `ifconfig` needs the two tunnel IPs when we are using `tun` devices, and the local IP along with netmask, when we are using `tap` devices.

Parameters used in the standard configuration file for a static key client

When we want to connect a Linux system to a Windows XP system with the standard configuration file that we used (and adapted slightly) in Chapter 5, we have to change this command a little bit:

```
vpnserver:/etc/openvpn# openvpn --remote 10.10.10.103 --dev tap
--ifconfig 10.3.0.2 255.255.255.0 --secret key.txt --comp-lzo
Fri Nov 18 22:35:15 2005 OpenVPN 2.0.9 i486-pc-linux-gnu [SSL] [LZO]
[EPOLL] built on Sep 20 2007
Fri Nov 18 22:35:15 2005 LZO compression initialized
Fri Nov 18 22:35:15 2005 TUN/TAP device tap0 opened
Fri Nov 18 22:35:15 2005 /sbin/ifconfig tap0 10.3.0.3 netmask
255.255.255.0 mtu 1500 broadcast 10.3.0.255
Fri Nov 18 22:35:15 2005 UDPv4 link local (bound): [undef]:1194
Fri Nov 18 22:35:15 2005 UDPv4 link remote: 10.10.10.103:1194
```

Compressing the data

Until OpenVPN 1.5, Windows needed TAP devices, thus the option `--dev` has the parameter `tap` instead of `tun`, and in the standard configuration file the `lzo` compression is activated, that's why I typed `–comp-lzo` at the end of my command line.

Parameter	Options	Function	Usage	Example
comp-lzo	\<yes\> \<no\> \<adaptive\> (default)	openvpn uses lzo library to compress tunnel traffic	Command line and configuration file	--comp-lzo

Normally, we do not need any option to this parameter, unless you want to control compression of tunnel data more specifically (such as switching compression on/off on the fly in server mode), the manual page will provide detailed information here too.

> The parameter `comp-lzo` activates compression of tunnel data.

Without further options, `comp-lzo` will use its adaptive algorithm. OpenVPN checks compression efficiency regularly and adapts it to the results. By doing so, compressed data will very likely not be compressed again, but other data have a high probability of being compressed.

Now let's have a look at the other parameters used in the OpenVPN standard configuration files. In Chapter 5, we adapted the configuration file for a client that uses static keys.

On Windows, open the file `C:\Program Files\OpenVPN\sample-config\sample.ovpn` in Notepad. On Linux, open the configuration file copied from the Windows system.

In this file, there are some more parameters that we did not talk about. Some of them are commented, either by a semi-colon or by a hash mark at the beginning of a line. The following table gives an overview of ports, protocols, and network devices:

Parameter	Options	Function	Usage	Example
`port`	`<port number>`	Specifies the port (both local and remote) which OpenVPN will use.	Command line and configuration file	`--port 5001`
`proto`	`<udp>` `<tcp-client>` `<tcp-server>`	Sets the protocol OpenVPN uses. A TCP client will try to start connections, while a TCP server only waits for clients.	Command line and configuration file	`--proto udp` `--proto tcp-client` `--proto tcp-server`
`tun-mtu`	`<mtu size>`	Sets the maximum transmission units.	Command line and configuration file	`--tun-mtu 1200`
`dev`	`<interface name>`	Specifies the name of the interface to be used. (Linux)	Command line and configuration file	`--dev openvpn1`
`dev-node`	`<node name>`	Specifies the name of an existing device node to use. (Windows)	Command line and configuration file	`--dev node tap9`
`dev-type`	`tun or tap`	Specifies the type of devices explicitly.	Command line and configuration file	`--dev-type tap`

You may have noticed that I left out two parameters: `fragment` and `mssfix`. These two are relevant if you run into problems with **Maximum Transmission Units** (**MTUs**) and datagram sizes when you are using UDP. I never ran into such problems, but if you need more information, the online man page is very detailed, and there are many reports of people who could solve speed issues by cutting down the MTU.

With the parameter `dev-node`, you can tell `openvpn` to use a specific network device. In the aforementioned example, I have entered `openvpn1` as name of the device. This is the name I gave the network adapter in the **Network Connections** module of the Windows **Control Panel**. On Linux you can also simply set the name of the device as an option to the parameter `dev`:

```
vpnserver:/etc/openvpn# openvpn --remote 10.10.10.103 –dev-type tap
--dev openvpn0 --ifconfig 10.3.0.2 255.255.255.0 --secret key.txt
--verb 1 --comp-lzo
```

A brief check with `ifconfig` shows that this command will have the tunnel created over the network device `openvpn0`:

```
mfeilner@vpnserver:~$ /sbin/ifconfig
eth1      Protocol:Ethernet  Hardware Address 06:1C:ED:9D:12:56
(...)
openvpn0  Protocol:Ethernet  Hardware Address 06:1C:ED:9D:12:51
          inet Adresse:10.3.0.2  Bcast:10.3.0.255
Maske:255.255.255.0
```

On Windows you would need to add the parameter `--dev-node` followed by the name of the network device you want to use.

Controlling and restarting the tunnel

The following parameters from our standard file can be used by OpenVPN to determine whether a tunnel is still up or already down.

Parameter	Options	Function	Usage	Example
`ping`	`<seconds>`	Sends a ping to the other tunnel partner through the tunnel after `<seconds>` without traffic	Command line and configuration file	`--ping 10`
`ping-restart`	`<seconds>`	After `<seconds>` without receiving any packet from remote, the tunnel will be restarted	Command line and configuration file	`--ping-restart 60`
`ping-timer-rem`	-	`ping-restart` runs only when a remote address is given	Command line and configuration file	`--ping-timer-rem`
`persist-tun`	-	Keeps `tun`/`tap` devices up when `openvpn` is restarted	Command line and configuration file	`--persist-tun`
`persist-key`	-	`openvpn` will not re-read the keys on a restart	Command line and configuration file	`--persist-key`
`resolv-retry`	`<seconds>`	This sets the time for which `openvpn` will try to resolve a hostname before giving up	Command line and configuration file	`--resolv-retry 86400`

OpenVPN brings some sophisticated tools to check tunnels and restart them, if they are not working anymore.

- `ping`: This parameter is used to send ping packages through the tunnel to the tunnel partner on a regular schedule.

- `ping-restart`: If the sender does not receive any traffic for the time span defined by the parameter, `openvpn` assumes that this tunnel is dead and will try to establish it again by restarting it.

- `ping-timer-rem`: If you add the parameter `ping-timer-rem`, `openvpn` will only start a tunnel if a remote address for the tunnel is given—thus a server only listening for clients will not try to reconnect. If the option `persist-tun` is set, `openvpn` will keep up the network devices used.

- `persist-key`: This parameter will prevent `openvpn` from re-reading the key files on a restart. This should only be necessary when `openvpn` runs as a non-privileged user without access to the key files.

Debugging output—troubleshooting

And last but not least, the parameters that define the verbosity and debugging output of OpenVPN:

Parameter	Options	Function	Usage	Example
`verb`	`<verbosity level>`	Sets level of verbosity, `0` is lowest, `11` is maximum detail level	Command line and configuration file	`--verb 4`
`mute`	`<number of messages>`	openvpn will print only `10` consecutive messages from the same category	Command line and configuration file	`--mute 10`

The parameter `verb` offers a range from `0` to `11` for the verbosity of the output `openvpn` provides. Default for this parameter is `1`, which should provide enough output in most cases. Selecting `0` here will make `openvpn` provide messages only when fatal errors occur. While levels `1-4` provide an increasing level of verbosity, which is useful for administration, levels `5` and above are ideal only for debugging. Following is an example for the output of `openvpn` concerning the successful initialization of our sample connection:

```
vpnserver:/etc/openvpn# openvpn --remote 10.10.10.103 --dev tap
--ifconfig 10.3.0.2 255.255.255.0 --secret key.txt --verb 1 --comp-lzo
--dev tap1 --verb 11
(...)
t Nov 19 01:07:21 2005 us=949416 UDPv4 read returned 60
Sat Nov 19 01:07:21 2005 us=949642 UDPv4 READ [60] from
10.10.10.103:1194:  DATA 01edfefe f6ed7f34 019f0f09 9c560481 084241cc
63d35cfd 71f001d8 d640fbe[more...]
Sat Nov 19 01:07:21 2005 us=949815 DECRYPT IV: 63d35cfd 71f001d8
```

```
Sat Nov 19 01:07:21 2005 us=950033 DECRYPT TO: 00000220 43844441
fa2a187b f3641eb4 cb07ed2d 0a981fc7 48
Sat Nov 19 01:07:21 2005 us=950197 PID TEST 0:0 1132741697:544
Sat Nov 19 01:07:21 2005 us=950378 Peer Connection Initiated with
10.10.10.103:1194
Sat Nov 19 01:07:21 2005 us=950687 RECEIVED PING PACKET
Sat Nov 19 01:07:21 2005 us=950709 Initialization Sequence Completed
Sat Nov 19 01:07:21 2005 us=950724 TIMER: coarse timer wakeup 1
seconds
Sat Nov 19 01:07:21 2005 us=950741 PO_CTL rwflags=0x0001 ev=3
arg=0x08090424
Sat Nov 19 01:07:21 2005 us=950768 PO_CTL rwflags=0x0001 ev=4
arg=0x08090420
Sat Nov 19 01:07:21 2005 us=950788 I/O WAIT TR|Tw|SR|Sw [1/185372]
Sat Nov 19 01:07:23 2005 us=150719  event_wait returned 0
Sat Nov 19 01:07:23 2005 us=150773 I/O WAIT status=0x0020
Sat Nov 19 01:07:23 2005 us=150791 TIMER: coarse timer wakeup 5
seconds
Sat Nov 19 01:07:23 2005 us=150813 PO_CTL rwflags=0x0001 ev=3
arg=0x08090424
Sat Nov 19 01:07:23 2005 us=150851 PO_CTL rwflags=0x0001 ev=4
arg=0x08090420
Sat Nov 19 01:07:23 2005 us=150870 I/O WAIT TR|Tw|SR|Sw [5/185372]
A very helpful level of verbosity can be set by using --verb 5:
vpnserver:/etc/openvpn# openvpn --remote 10.10.10.103 --dev tap
--ifconfig 10.3.0.2 255.255.255.0 --secret key.txt --verb 1 --comp-lzo
--dev tap1 --verb 5
(...)
Sat Nov 19 01:38:53 2005 us=827058 UDPv4 link local (bound):
[undef]:1194
Sat Nov 19 01:38:53 2005 us=827200 UDPv4 link remote:
10.10.10.103:1194
RSat Nov 19 01:39:01 2005 us=970557 Peer Connection Initiated with
10.10.10.103:1194
Sat Nov 19 01:39:01 2005 us=970938 Initialization Sequence Completed
WRRwrWRwrWRwrWrWRwRw
```

As you can see in the last line, OpenVPN prints **w**'s and **r**'s for each packet travelling through the tunnel. A capital letter stands for a packet read or written to the TUN/TAP adapter, a small letter stands for a packet written or read in the tunnel. This is really very useful, because you can easily track packets (like pings) and find out how far they come. Set up your tunnel with verb 5 on both sides, ping the other host from either side, and watch the debug output—there are four letters for each ping: **RwrW**.

Configuring OpenVPN with certificates—simple TLS mode

In Chapter 8, we worked with a configuration file like the following:

```
remote 10.10.10.103
dev tap
tls-client
ifconfig 10.3.0.2 255.255.255.0
dh keys/dh2048.pem
ca keys/ca.crt
cert keys/VPN-Client.crt
key keys/VPN-Client.key
```

In the third line of our little configuration file, we find the parameter `tls-client`, on our VPN-server we entered `tls-server` here. These entries cause `openvpn` to start TLS to protect the data transferred. All machines involved in the VPN need the same CA certificate and a local certificate and key pair issued by this CA. On connection, the two partners exchange their local certificates and validate the partner's certificate by checking if it was signed by the common CA. OpenVPN must know which files contain the CA and local certificate and key.

The following table shows the main parameters that we need to adapt for use with certificates:

Parameter	Options	Function	Usage	Example
dh	<file>	Defines the Diffie-Hellmann key	Command line and configuration file	--dh keys/ dh2048.pem
ca	<file>	Defines the certificate file of the CA	Command line and configuration file	--ca keys/ ca.crt
cert	<file>	Defines the local machine's certificate file	Command line and configuration file	--cert keys/ VPN-Client. crt

Parameter	Options	Function	Usage	Example
`key`	`<file>`	Defines the local machine's key file	Command line and configuration file	`--key keys/ VPN-Client. key`
`tls-server`	-	Local machine acts as TLS server	Command line and configuration file	`--tls-server`
`tls-client`	-	Local machine acts as TLS client	Command line and configuration file	`--tls-client`

The options `tls-server` and `tls-client` affect only the way in which the TLS handshake is dealt with and have no further consequences for OpenVPN. The section below called *Encryption parameters* lists further options, for example how to use Keys stored in PKCS formats.

Overview of OpenVPN parameters

The table in the following section is a detailed list of all parameters OpenVPN offers concerning basic tunnel options. They can be used both at the command line and in configuration files.

General tunnel options

Most of these options are used to determine the way in which `openvpn` connects to the tunnel partner and how it deals with connections not responding or changing.

Parameter	Options	Function	Usage
`local`	`<host>`	Binds local service to the address of `<host>`. Useful if you want `openvpn` to run only on one interface of a host, with multiple home sites.	`--local 192.168.0.50`
`remote`	`<host>`	Connects to the host. IP or DNS are equivalent, DynDNS is possible.	`--remote feilner-it.net`
`remote- random`		Simple load balancing, specify multiple `--remote` addresses and `openvpn` will randomly connect to one of them.	`--remote- random`
`float`		Allows the remote VPN partner to change the remote IP address (for example with DynDNS).	`--float`

Parameter	Options	Function	Usage
`ipchange`	`<cmd>`	Calls the program `<cmd>` if the IP address has changed.	`--ipchange / script-ip.sh`
`connect-retry`	`<seconds>`	Retries to connect for `<seconds>` if connection fails.	`--connect-retry 60`
`connect-retry-max`	`<n>`	n is the maximum number of retries that can be done if the connection can't be established.	`--connect-retry-max`
`resolv-retry`	`<seconds>`	If `openvpn` can't resolve the hostname of the tunnel partner, it will try to reconnect after n seconds.	`--resolv-retry 86400`
`proto`	`<tcp/udp>`	Protocol to use.	`--proto udp`
`port`	`<port>`	Uses this port for connections (both local and remote).	`--port 5493`
`lport`	`<port>`	Uses this local port to bind OpenVPN.	`--lport 1194`
`rport`	`<port>`	Uses this remote port to bind OpenVPN.	`--rport 5000`
`nobind`		Uses dynamic port to connect (only client).	`--nobind`
`shaper`	`<Bytes>`	Throttles the outgoing data bandwidth of your tunnel (only client; only outgoing bandwidth).	`--shaper 10000`
`ip-win32`	`<method>`	Sets the Windows network adapter's IP and netmask using `<method>`.	`--ip-win32 ipapi`
`tun-ipv6`	-	Sets up a tunnel (TUN or TAP) capable of IPv6 traffic.	`--tun-ipv6`

Unfortunately, it's impossible to deal with all options in detail within the scope of this book. Nevertheless, we will have a close look at various parameters that have proven useful.

- If your system has several NICs or several IP addresses, you may want OpenVPN to run only on one of them. This can easily be done with the parameter `--local` followed by the IP you want to bind OpenVPN to. This option might be very interesting for routers or firewalls providing VPN services, too.

- We have learned about the option `remote`, and that it supports DNS entries (and therefore DynDNS), but we need to set the `float` parameter to allow the other tunnel endpoint change its IP without needing to restart the tunnel. The parameter `ipchange` specifies a command that can be executed on such an event.

> With the option `float`, OpenVPN does not need to restart tunnels when the IP of a partner changes.

- If you specify multiple `--remote` addresses, the parameter `remote-random` enables automatic load balancing between the hosts by choosing randomly which to connect to.

> Multiple entries in the remote addresses' field enable simple load balancing or redundancy.

- The options `connect-retry`, `connect-retry-max`, and `resolv-retry` define how (often and long) OpenVPN will try to establish a connection when errors occur. (86400 seconds are one day).

- The parameter `proto` switches `udp` and `tcp` mode within OpenVPN. UDP should always be preferred, as there are some problems with TCP.

- Furthermore, the options `port`, `lport`, `rport`, and `nobind` give us the possibility to define exactly from which local port to which remote port our tunnel shall be connected. And if we like, `--nobind` will use dynamically assigned ports—almost randomly.

- Probably the handiest parameter in this section is `--shaper`. Using `--shaper 10000` will limit outgoing bandwidth of the `openvpn` tunnel to `10000` byte/sec. Only outgoing traffic can be shaped (do you know why?), so if you want your tunnel to use 10k of bandwidth as a maximum, you have to set this on both sides!

> The option `shaper` offers simple bandwidth management, also known as **traffic shaping**.

And the last parameter is the one that I have not used till today: `--ip-win32` lets you decide the method with which the Windows network adapter receives its IP and netmask. This method may be one of adaptive, IPAPI, Netsh, Dynamic, or Manual. More information on this can be found in the manual page.

The following example shows an excerpt from a configuration file. Can you explain what `openvpn` is supposed to do according to this file?

```
(...)
local 192.168.0.150
remote feilner-it.net
remote openvpn.dyndns.org
```

```
remote-random
float
resolv-retry 86400
proto tcp-client
lport 22222
rport 22223
connect-retry 86400
shaper 10000
(...)
```

These lines make `openvpn` set up a tunnel:

- Listening only on the local IP `192.168.0.150`
- Trying to connect randomly to `feilner-it.net` and `openvpn.dyndns.org`
- Ignoring changing IP of the other tunnel partner, as long as encryption is OK
- Running as `tcp-client` on local port `22222`
- Trying to connect to remote port `22223`
- If the connection fails, retrying for a day
- Outgoing traffic is limited to `10000` bytes/sec

Routing

The parameters in this section deal with routing of the traffic inside, to, and from the tunnel. We have already learned about the parameter `ifconfig` and that it needs different parameters for TAP or TUN devices. A second important point in this section is the parameter `route`. Many people seem to have difficulties with connecting networks over OpenVPN, but it's really easy.

Parameter	Options	Function	Usage
`ifconfig`	`<local remote>`	Sets the IP address and netmask for the tunnel on TAP devices, sets the local and remote IP address for the tunnel on TUN devices	`--ifconfig 10.1.0.1 10.1.0.2` `--ifconfig 10.1.0.1 255.255.255.0`
`route`	`<network>`	Sets a specific route on the VPN host when `openvpn` has successfully started the tunnel	`--route 10.0.10.0 255.255.255.252`

Parameter	Options	Function	Usage
route-gateway	`<IP>`	Sets the gateway on the VPN host	`--route-gateway 192.168.0.22`
route-delay	`<seconds>`	Waits n seconds before setting the routes	`--route-delay 5`
route-up	`<cmd>`	Calls a program if the routes are up	`--route-up / script.sh`
redirect-gateway		Sets default route through the tunnel	`--redirect-gateway`

- `--ifconfig`: Sets the IPs of the tunnel. Here you need to give the two IPs of a point-to-point VPN, based on TUN devices, or the IP and netmask of a TAP-based VPN-bridge.

- `--route`, `--route-gateway`, and `--redirect-gateway`: Affect the routing of packets on the VPN host. After our tunnel is set up correctly, we have to make sure that both VPN servers are forwarding traffic (perhaps we need a firewall?), and that the connected networks are routed correctly on the other side. A later example will deal with this setup. `--redirect-gateway` is an excellent feature, for example, for notebooks of road warriors.

- `--route-up`: Enables us to run scripts when the routes are set up.

- `--route-delay`: Tells `openvpn` to wait a little before setting the routes when the tunnel is set up.

There are many possibilities to use `ifconfig`, `route`, and `route-up` commands for an `openvpn` tunnel:

```
(...)
ifconfig 10.3.0.1 10.3.0.2
route 192.168.0.0 255.255.255.0 10.3.0.2
route-up "/sbin/firewall_openvpn_1 start"
route-up "route add -net 192.168.1.0 netmask 255.255.255.0 gw
192.168.0.1"
route-delay 2
(...)
```

This example provides a tunnel, where a firewall script (`/sbin/firewall_openvpn_1`) is started after routing is set up. A `route` is defined into a subnet `192.168.0.0` behind the other tunnel endpoint. Another `route` is defined into a third subnet on the other side of the tunnel using the `route-up` parameter and the Linux system tool `route`. And last but not least, `openvpn` waits 2 seconds between setting up the tunnel and configuring routing.

Controlling the tunnel

Parameter	Options	Function	Usage
`inactive`	`<seconds>`	The TUN/TAP device is closed after `<seconds>` of inactivity	`--inactive 120`
`ping-exit`	`<seconds>`	After `<seconds>` with no packet received, shutdown OpenVPN	`--ping-exit 120`
`keepalive`	`<seconds>`	Simply `ping` and `ping-restart`	`--keep-alive 10 60`
`persist-local-ip`	`<IP>`	Keeps local IP over restarts	
`persist-remote-ip`	`<IP>`	You can't restart the tunnel if the IP was changed	`--persist-remote-ip 62.184.232.1`

In this context the parameters `--ping`, `--ping-restart`, `--ping-timer-rem`, `--persist-tun`, and `--persist-key` should also be mentioned. We met them as part of the standard configuration.

All these parameters influence the behavior of `openvpn` concerning testing and restarting a tunnel. If there is no traffic in the tunnel for the number of seconds specified by the `ping` parameter, `openvpn` will send a ping packet through the tunnel. If no packet is received for the amount of seconds defined with `--ping-restart`, the tunnel is started over.

The parameter `--keepalive` is a shortcut for a combination of `ping` and `ping-restart`, you can express:

```
ping 100
ping-restart 200
```

by the simple directive:

```
keepalive 100 200
```

The parameters `--ping`, `--ping-restart`, `--ping-exit`, and `--inactive` can be combined in many ways, depending on your setup and goals. Can you imagine what the following example does?

```
(...)
ping 20
ping-restart 120
inactive 3600
(...)
```

These directives cause openvpn to send pings after 20 seconds of inactivity. After two minutes of inactivity, openvpn will restart the tunnel. After an hour without tunnel data being exchanged, openvpn will exit.

Scripting

OpenVPN has several points of time when scripts can be executed. We have already learned about one of them, `--route-up <command>`. Here is a list of more parameters that allow scripts to be run:

Parameter	Options	Function	Usage
up	<command>	Calls program when the TUN/TAP device is up	--up script-up.sh
up-delay	<seconds>	Waits n seconds after connect for the up-script	--up-delay 5
down	<command>	Calls program when the TUN/TAP device is down	--down script-down.sh
down-pre	<command>	Calls script before TUN/TAP shuts down	--down-pre
up-restart	<command>	Calls script after every reconnect	--up-restart
route-up	<command>	Calls a program when the routes are up	--route-up script.sh
ipchange	<command>	Calls script when the IP changes	-- ipchange script.sh
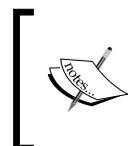
> With openvpn we can have our own scripts executed before and after the interface is brought up or down, when we are reconnected, when the routes are set up, and when our IP changes.

Modules

Apart from the flexible scripting hooks, OpenVPN also brings a handy plugin architecture. Invoked with the parameter `--plugin`, an administrator can use an abundance of precompiled modules that the community offers. The plugin architecture follows the following syntax:

```
--plugin module-pathname [init-string]
```

`module-pathname` should be replaced with the full path to the module program, and `init-string` can be an argument to initialize the module. On SuSE systems, a good read to start with is `/usr/share/doc/packages/openvpn/README.plugins`, a Readme that is not present on Debian systems, where you will have to refer to `/usr/include/openvpn/openvpn-plugin.h` for details.

You can use several modules in a row and combine them with the scripting hooks. OpenVPN can call modules at 9 stages of the tunnel creation:

- `up`: The tunnel is started
- `down`: The tunnel is stopped
- `route-up`: Routing is set up
- `ipchange`: The IP address of the system has changed
- `tls-verify`: The certificate is being verified
- `auth-user-pass-verify`: User Authentication is in progress
- `client-connect`: A client connects
- `client-disconnect`: A client disconnects
- `learn-address`: OpenVPN learns the address from a VPN server

OpenVPN comes with three plugins that prove very helpful:

- `auth-pam` can only be used on Linux and enables OpenVPN to use the underlying **Pluggable Authentication Modules** (**PAM**) as source for user credential verification.
- `down-root` allows down-scripts to be run with root privileges, even when `--user` or `--group` has been activated for the tunnel. Also not for Windows.
- An example plugin (`example`) that you can use to learn and try editing your own plugins.

An example on using modules for flexible authentication can be found in Chapter 13—*Advanced Configuration*.

Logging

Besides the debugging parameters `--verb` and `--mute` that we learned about when dealing with our standard configuration file, there are several parameters useful for directing the output of `openvpn`:

Parameter	Options	Function	Usage
`log`	`<file>`	Defines the log file which the output of messages for this tunnel is supposed to be written	`--log /var/ log/vpn.log`
`log-append`	`<file>`	Appends messages to the log file—does not overwrite it	`--log-append / var/log/ openvpn/ messages.log`
`status`	`<file>`	Writes a status file of the connections to `<file>`	`--status /var/ log/openvpn/ status.log`

You should add the following two lines to every tunnel you configure:

```
log-append /var/log/openvpn/packt.log
status /var/log/openvpn/packt.status
```

The first entry in a configuration file will cause `openvpn` to write debug information and messages in the specified file. The latter will print status information like the following in a status logfile:

```
vpnserver:/etc/openvpn # cat /var/log/openvpn/packt.status
OpenVPN STATISTICS
Updated,Thu Nov 24 09:11:02 2005
TUN/TAP read bytes,3189334
TUN/TAP write bytes,3783482
TCP/UDP read bytes,4847840
TCP/UDP write bytes,4248748
Auth read bytes,3801636
pre-compress bytes,579459
post-compress bytes,546430
pre-decompress bytes,489729
post-decompress bytes,678607
END
```

These data are updated automatically and can be very helpful for statistic programs such as Nagios, Munin, or Cacti. Depending on the function of your VPN machine, the statistics will look different. A server in TLS mode will collect the data grouped by client:

```
vpnserver:/var/log # cat openvpn/openvpn-stats.log
OpenVPN CLIENT LIST
Updated,Mon Mar 30 18:47:07 2009
Common Name,Real Address,Bytes Received,Bytes Sent,Connected Since
Client1,78.47.XX.XX:54723,3240337,3354489,Tue Mar 24 21:45:12 2009
Client2,84.57.XX.XX:25694,741856,587295,Mon Mar 30 14:58:03 2009
Client3,78.47.XX.XX:58053,342703146,27997912,Tue Mar 24 21:45:12 2009
WRT,84.57.XX.XX:1026,3199634,3345205,Tue Mar 24 21:45:12 2009
ROUTING TABLE
Virtual Address,Common Name,Real Address,Last Ref
10.10.11.12,WRT,84.57.XX.XX:1026,Tue Mar 24 21:45:14 2009
10.10.11.10,Client1,78.47.XX.XX:54723,Tue Mar 24 21:45:12 2009
10.10.11.8,Client3,78.47.XX.XX:58053,Mon Mar 30 17:25:49 2009
10.10.11.6,Client2,84.57.XX.XX:25694,Mon Mar 30 18:47:06 2009
GLOBAL STATS
Max bcast/mcast queue length,0
END
```

Specifying a user and group

On Linux, we can specify a certain user and group under whose privileges `openvpn` shall run—a good idea to reduce the number of processes running with root privileges and increase security:

Parameter	Options	Function	Usage
user	Unix Account	For more security	--user nobody
group	Unix Account	For more security	--group nogroup

Please note that `openvpn` will be started with root privileges, but once a tunnel configured with `--user nobody` is started, it switches to the environment of this user. This may lead to problems, when key or certificate files are not readable to the user you defined in your configuration, as root `openvpn` can read the key files and start the tunnel. Later, this tunnel is restarted due to some parameter passed (like `--ping-restart`), and now, `openvpn` will try to re-read the key files. If the unprivileged user (`nobody`) has no right to read these files, this will fail and the tunnel won't be set up. You can avoid this by using the parameter `--persist-key`. The same applies to the network devices—you can avoid this problem with the parameter `--persist-tun`.

The management interface

OpenVPN provides a management interface available through Telnet. This interface is designed for use by management tools like **OpenVPN-Admin** that allow GUI management of tunnels.

Parameter	Options	Function	Usage
`management`	`<IP> <port>` `<pw-file>`	Management interface of OpenVPN	`--management` `127.0.0.1 5702`
`--management-hold`	-	The tunnel will not be set up until the command `hold release` is entered in the management console	`--management-hold`
`--management-log-cache`	`<number>`	Caches the number of lines for use with the management interface	`--management-log-cache 10`

If you want to activate the management interface, you simply need to add a line like the following to your configuration file:

```
management 10.10.10.105 5702
```

Or invoke OpenVPN with a command like:

```
vpnserver:/etc/openvpn# openvpn --remote 10.10.10.103 --dev-type tap
--dev openvpn0 --ifconfig 10.3.0.2 255.255.255.0 --secret key.txt
--verb 1 --comp-lzo --management 127.0.0.1 5702
```

To connect to the management interface is easy—just type `telnet <IP> <Port>` and replace `IP` and `Port` with the values you placed in the configuration file. After you have connected, type `help` to get a list of available commands:

```
vpnserver:/home/mfeilner# telnet localhost 5702
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
>INFO:OpenVPN Management Interface Version 1 -- type 'help' for more
info
help
Management Interface for OpenVPN 2.0.9 i486-pc-linux-gnu [SSL] [LZO]
[EPOLL] built on Sep 20 2007
Commands:
auth-retry t           : Auth failure retry mode
(none,interact,nointeract).
echo [on|off] [N|all]  : Like log, but only show messages in echo
buffer.
```

```
exit|quit               : Close management session.
help                    : Print this message.
hold [on|off|release]   : Set/show hold flag to on/off state, or
                          release current hold and start tunnel.
kill cn                 : Kill the client instance(s) having common
name cn.
kill IP:port            : Kill the client instance connecting from
IP:port.
log [on|off] [N|all]    : Turn on/off realtime log display
                            + show last N lines or 'all' for entire
history.
mute [n]                : Set log mute level to n, or show level if n
is absent.
net                     : (Windows only) Show network info and routing
table.
password type p         : Enter password p for a queried OpenVPN
password.
signal s                : Send signal s to daemon,
                            s = SIGHUP|SIGTERM|SIGUSR1|SIGUSR2.
state [on|off] [N|all] : Like log, but show state history.
status [n]              : Show current daemon status info using format
#n.
test n                  : Produce n lines of output for testing/
debugging.
username type u         : Enter username u for a queried OpenVPN
username.
verb [n]                : Set log verbosity level to n, or show if n is
absent.
version                 : Show current version number.
END
status
OpenVPN STATISTICS
Updated,Mon Mar 30 16:55:01 2009
TUN/TAP read bytes,468
TUN/TAP write bytes,0
TCP/UDP read bytes,0
TCP/UDP write bytes,1004
Auth read bytes,0
pre-compress bytes,0
post-compress bytes,0
pre-decompress bytes,0
post-decompress bytes,0
END
```

Every output of the management console ends with the string END, in this example the user entered status as a second command. If you have set the verbosity level to any level higher than 2, you will receive entries in your log file like the following every time a client connects.

```
Mar 30 16:54:06 2009 us=571960 MANAGEMENT: Client connected from
127.0.0.1:5702
Mar 30 16:54:28 2009 us=521860 MANAGEMENT: Client disconnected
```

- The management console can be password-protected, simply put your password in a file and add the path to this file in your configuration file.
- Tunnels can be started in suspended mode, which means that they are only started after a command sent from the management console. Just add --management-hold to the configuration. The tunnel will not be started until you log in to this tunnel's management interface and type hold release.

Proxies

Because OpenVPN uses SSL/TLS for encryption, and UDP or TCP as transport protocol, it can be easily tunneled through an HTTP-proxy. Similarly, we can have our tunnels proxied over a **SOCKetS** (**SOCKS**) proxy server. The following parameters are available for proxy support:

Parameter	Options	Function	Usage
http-proxy	<server port [auth]>	OpenVPN can tunnel through proxies. Specify the proxy and the port here. Optionally, authentication is supported.	--http-proxy 192.168.0.12 8080
http-proxy-retry	-	Retries indefinitely if connection fails.	--http-proxy-retry
http-proxy-timeout	<seconds>	Considers connection to proxy as failed after <seconds> inactivity.	--http-proxy-timeout 5
socks-proxy	<server port>	Tunneling through a socks5 gateway.	--socks-proxy 192.168.0.12 8080
socks-proxy-retry	-	Retries indefinitely if connection fails.	--socks-proxy-retry
auto-proxy		Tries to determine the proxy automatically, needs OpenVPN 2.1 or higher.	--auto-proxy

OpenVPN tunnels can be tunneled through both HTTP and SOCKS proxies.

Encryption parameters

Chapter 10 in this book deals with security options for OpenVPN, but we will have a short (introductory) look at the parameters OpenVPN's cryptographic layer provides. The most important ones are here in the following table, and we already know many of them:

Parameter	Options	Function	Usage
secret	`<file>`	Points to the file with the static key	`--secret /kex.txt`
cipher	`<alg>`	Specifies the algorithm to use for encryption of packets	`--cipher AES-256-CBC`
keysize	`<n>`	Specifies the size of the cipher key in bits	`--keysize 128`
auth	`<alg>`	Defines the message digest algorithm `<alg>` used by the HMAC authentication algorithm	`--auth SHA1`
tls-server		Uses SSL certificates and acts as TLS server during TLS handshake	`--tls-server`
tls-client		Uses SSL certificates and act as TLS client during TLS handshake	`--tls-client`
ca	`<file>`	Your generated CA file	`--ca /CA.crt`
dh	`<file>`	Your generated Diffie-Hellman key	`--dh /DH.pem`
cert	`<file>`	Your server's local certificate file	`--cert /SERVER.crt`
key	`<file>`	Your server's local key file	`--key /SERVER/key.pem`
pkcs12	`<file>`	PKCS12 file (containing certificate, key, and CA in one file)	`--pkcs12 /file`
crl-verify	`<file>`	Certificate revocation list	`--tls-verify /revoke.crl`
no-replay		Disables OpenVPN's protection against replay attacks	`--no-replay`
no-iv		Disables OpenVPN's use of Cipher Initialization Vector (IV)	`--no-iv`

The following parameters may be new to you. In most cases you do not need to make any changes here:

- `cipher`: Here you can specify a different algorithm for transport encryption. Have a look at the option `--show-ciphers` later to receive a list of available algorithms.

- `keysize`: You can specify a different key size for the cipher algorithm that you chose with the `--cipher` parameter. The option `--show-ciphers` (later) shows the default key sizes.

- `auth`: OpenVPN uses SHA1 with HMAC to authenticate packets. No changes should be necessary here, but with the option `auth none` you could disable authentication.

- `pkcs12`: This is a file format in which CA certificate, server certificate, and local key are packed together. Using such a file would replace the directives `--ca`, `--cert`, and `--key`.

- `no-replay` and `no-iv`: These disable basic security mechanisms that OpenVPN provides. Do not deactivate these unless you know what you are doing. These parameters switch off basic security functions and will leave your system insecure.

- `crl-verify`: This defines the file in which a certificate revocation list is stored. Such a list contains certificates that are no longer valid for use with our OpenVPN tunnels.

> The parameter `crl-revoke <file>` specifies the file containing the certificate revocation list.

Testing the crypto system with --test-crypto

With the command-line parameter `--test-crypto` we will now test the cryptographic system of our VPN server with a static key.

Parameter	Options	Function	Usage
test-crypto		Command line only. Do a self-test of OpenVPN's crypto options by encrypting and decrypting test packets using the data channel encryption options specified above.	--test-crypto

```
vpnserver:/etc/openvpn# openvpn --test-crypto --secret key.txt
Mon Mar 30 16:59:39 2009 OpenVPN 2.0.9 i486-pc-linux-gnu [SSL] [LZO]
[EPOLL] built on Sep 20 2007
Mon Mar 30 16:59:39 2009 OpenVPN 2.0.9 i486-pc-linux-gnu [SSL] [LZO]
[EPOLL] built on Sep 20 2007
Mon Mar 30 16:59:39 2009 Entering OpenVPN crypto self-test mode.
Mon Mar 30 16:59:39 2009 TESTING ENCRYPT/DECRYPT of packet length=1
Mon Mar 30 16:59:39 2009 TESTING ENCRYPT/DECRYPT of packet length=2
Mon Mar 30 16:59:39 2009 TESTING ENCRYPT/DECRYPT of packet length=3
Mon Mar 30 16:59:39 2009 TESTING ENCRYPT/DECRYPT of packet length=4
Mon Mar 30 16:59:39 2009 TESTING ENCRYPT/DECRYPT of packet length=5

(..)

Mon Mar 30 17:00:33 2009 TESTING ENCRYPT/DECRYPT of packet length=1495
Mon Mar 30 17:00:33 2009 TESTING ENCRYPT/DECRYPT of packet length=1496
Mon Mar 30 17:00:33 2009 TESTING ENCRYPT/DECRYPT of packet length=1497
Mon Mar 30 17:00:33 2009 TESTING ENCRYPT/DECRYPT of packet length=1498
Mon Mar 30 17:00:34 2009 TESTING ENCRYPT/DECRYPT of packet length=1499
Mon Mar 30 17:00:34 2009 TESTING ENCRYPT/DECRYPT of packet length=1500
Mon Mar 30 17:00:34 2009 OpenVPN crypto self-test mode SUCCEEDED.
```

Everything looks fine, the crypto system is working well. It has successfully encrypted and decrypted `1500` packets with our pre-shared key without any errors.

SSL information—command line

Parameter	Function
openvpn --show-ciphers	Shows all available cipher algorithms for use with the --cipher option
openvpn --show-digests	Shows all available message digest algorithms to use with the --auth option
openvpn --show-tls	Shows the available TLS ciphers in a list sorted from highest preference and security to lowest
openvpn --engine	Uses a specific SSL-based hardware encryption engine
openvpn --show-engines	Shows available hardware-based crypto engines

The following examples give an overview of the standard output of OpenVPN's cryptographic engines. First, we will ask for a list of the cipher algorithms that can be used for transport encryption, which can be set using the `--cipher` parameter:

```
vpnserver:/etc/openvpn# openvpn --show-ciphers
(...)

DES-CBC 64 bit default key (fixed)
RC2-CBC 128 bit default key (variable)
DES-EDE-CBC 128 bit default key (fixed)
DES-EDE3-CBC 192 bit default key (fixed)
DESX-CBC 192 bit default key (fixed)
BF-CBC 128 bit default key (variable)
RC2-40-CBC 40 bit default key (variable)
CAST5-CBC 128 bit default key (variable)
RC2-64-CBC 64 bit default key (variable)
AES-128-CBC 128 bit default key (fixed)
AES-192-CBC 192 bit default key (fixed)
AES-256-CBC 256 bit default key (fixed)
```

The last entry, `AES-256-CBC 256`, is the safest one, `BF-CBC 128` is the default. Remember that using safer algorithms causes more traffic overhead—maybe a price to pay. It may be interesting for the paranoid or cautious user that the differences between the underlying operating systems and SSL-libraries are very big here. Whereas the previous example is from a Debian (Lenny) system, this is the output from a more up-to-date machine running OpenSuSE 11.0:

```
opensuse11:/home/mfeilner # openvpn --show-ciphers
(...)

DES-CBC 64 bit default key (fixed)
RC2-CBC 128 bit default key (variable)
DES-EDE-CBC 128 bit default key (fixed)
DES-EDE3-CBC 192 bit default key (fixed)
DESX-CBC 192 bit default key (fixed)
BF-CBC 128 bit default key (variable)
RC2-40-CBC 40 bit default key (variable)
CAST5-CBC 128 bit default key (variable)
RC2-64-CBC 64 bit default key (variable)
AES-128-CBC 128 bit default key (fixed)
AES-192-CBC 192 bit default key (fixed)
AES-256-CBC 256 bit default key (fixed)
CAMELLIA-128-CBC 128 bit default key (fixed)
CAMELLIA-192-CBC 192 bit default key (fixed)
CAMELLIA-256-CBC 256 bit default key (fixed)
```

The SuSE system offers also the Camellia algorithms. However, this could not be used together with the Debian server mentioned before.

The same applies for the parameter `--show-digests` that lists all available digest methods for use with the `--auth` parameter in the configuration file. However, the differences here are not that big.

```
vpnserver:/etc/openvpn# openvpn --show-digests
(...)
MD2 128 bit digest size
MD5 128 bit digest size
RSA-MD2 128 bit digest size
RSA-MD5 128 bit digest size
SHA 160 bit digest size
RSA-SHA 160 bit digest size
SHA1 160 bit digest size
RSA-SHA1 160 bit digest size
DSA-SHA 160 bit digest size
DSA-SHA1-old 160 bit digest size
DSA-SHA1 160 bit digest size
RSA-SHA1-2 160 bit digest size
DSA 160 bit digest size
RIPEMD160 160 bit digest size
RSA-RIPEMD160 160 bit digest size
MD4 128 bit digest size
RSA-MD4 128 bit digest size
ecdsa-with-SHA1 160 bit digest size
RSA-SHA256 256 bit digest size
RSA-SHA384 384 bit digest size
RSA-SHA512 512 bit digest size
RSA-SHA224 224 bit digest size
SHA256 256 bit digest size
SHA384 384 bit digest size
SHA512 512 bit digest size
SHA224 224 bit digest size
```

The standard is SHA 160. The entries in this list rank from insecure (but fast) to safe and slow.

This does not apply for the list of TLS methods available. This list is in `order of preference`, which means the first method is the safest (and slowest) one.

Available TLS-methods of Debian Lenny...	... and of Opensuse 11.0
`vpnserver:/etc/openvpn# openvpn --show-tls`	`opensuse11:/home/mfeilner # openvpn --show-tls`
Available TLS Ciphers, listed in order of preference:	Available TLS Ciphers, listed in order of preference:
DHE-RSA-AES256-SHA	DHE-RSA-AES256-SHA
DHE-DSS-AES256-SHA	DHE-DSS-AES256-SHA
AES256-SHA	AES256-SHA
EDH-RSA-DES-CBC3-SHA	DHE-RSA-CAMELLIA256-SHA
EDH-DSS-DES-CBC3-SHA	DHE-DSS-CAMELLIA256-SHA
DES-CBC3-SHA	CAMELLIA256-SHA
DHE-RSA-AES128-SHA	EDH-RSA-DES-CBC3-SHA
DHE-DSS-AES128-SHA	EDH-DSS-DES-CBC3-SHA
AES128-SHA	DES-CBC3-SHA
RC4-SHA	DHE-RSA-AES128-SHA
RC4-MD5	DHE-DSS-AES128-SHA
EDH-RSA-DES-CBC-SHA	AES128-SHA
EDH-DSS-DES-CBC-SHA	DHE-RSA-CAMELLIA128-SHA
DES-CBC-SHA	DHE-DSS-CAMELLIA128-SHA
EXP-EDH-RSA-DES-CBC-SHA	CAMELLIA128-SHA
EXP-EDH-DSS-DES-CBC-SHA	RC4-SHA
EXP-DES-CBC-SHA	RC4-MD5
EXP-RC2-CBC-MD5	EDH-RSA-DES-CBC-SHA
EXP-RC4-MD5	EDH-DSS-DES-CBC-SHA
	DES-CBC-SHA
	EXP-EDH-RSA-DES-CBC-SHA
	EXP-EDH-DSS-DES-CBC-SHA
	EXP-DES-CBC-SHA
	EXP-RC2-CBC-MD5
	EXP-RC4-MD5

And last, but not least, OpenVPN (and SSL/TLS in particular) can support hardware encryption devices. The parameter `--show-engines` lists available engines for such devices.

```
opensuse11:/home/mfeilner # openvpn --show-engines
OpenSSL Crypto Engines

VIA PadLock (no-RNG, no-ACE) [padlock]
Dynamic engine loading support [dynamic]
```

In our configuration file or at the command line, such an engine can be specified for usage with the `--engine` parameter.
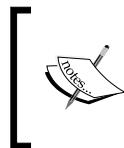
> OpenVPN provides several tools that list available cryptographic algorithms: `--show-tls`, `--show-ciphers`, and `--show-digests`. OpenVPN can be instructed to use a specific mechanism in the configuration file or at the command line.

Server mode

A very powerful parameter has been available since OpenVPN version 2: `--server`. This parameter can replace the `ifconfig` directive that is used to set up networking over TUN/TAP devices, and provide IP addresses and network configuration dynamically for clients.

Parameter	Options	Function	Usage
server	\<network\> \<mask\>	Sets the network addresses that are assigned to clients	`--server` `10.3.0.0` `255.255.255.0`

We must notice that `--server` implies TLS mode automatically, thus a directive like `--server 10.3.0.0 255.255.255.0` implies the following:

- The VPN software on this machine acts as a server for the tunnel described in this configuration (or in this command)
- This tunnel will be run in TLS-server mode— certificates are required
- Clients logging into this tunnel will be provided with an IP address from the network mask specified as option

In our previous example, TLS-certified clients will receive IP addresses between `10.3.0.1` and `10.3.0.254`. With TUN devices (running a virtual point-to-point connection), a /30 subnet is necessary for every connection, thus 128 clients can connect to this server. If we need a bridged network, the directive `server-bridge` is very helpful:

Parameter	Options	Function	Usage
server-bridge	gateway mask pool	Server mode for bridging devices (TAP)	--server bridge 10.3.0.1 255.255.255.0 10.3.0.128 10.3.0.254

The example would provide addresses from `10.3.0.128` to `10.3.0.254` and tell the (TLS-authenticated only) clients to use `10.3.0.1` as gateway in a bridged tunnel setup.

To be honest, `--server` is only a sort of *shortcut* for the directives setting server mode, TLS server, and network addresses. We will deal with these parameters in the next section, just note that there is a parameter called `--mode` that can be called with an option `server`.

Parameter	Options	Function	Usage
<mode>	<server>	Switches on openvpn server mode (since version 2)	--mode server
	<p2p> (default)	mode p2p is not necessary	--mode p2p
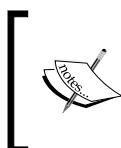
> `--mode server` switches on server mode in an OpenVPN tunnel. The directives `--server` and `--server-bridge` are handier, since they allow setting relevant data easily and switch on TLS automatically.

Server mode parameters

You may have noticed that there are several functions included in the parameter `--server` that we have not dealt with, like defining an IP range for clients logging into the VPN. The following table gives an overview of parameters useful for such issues:

Parameter	Options	Function	Usage
push	<options>	Allows *pushing* of configuration data to the client. (See later section for further options)	--push route 192.168.0.0 255.255.255.0
ifconfig-pool	<start-IP> <end-IP> <mask>	Defines a range of IP addresses to be used for the tunnel subnet.	--ifconfig-pool 10.1.0.1 10.1.0.10 255.255.255.0
ifconfig-pool-persist	<file> <seconds>	Ensures IP associations for clients—so that clients will always (hopefully) be assigned the same IP. IP-to-client associations will be written to <file> every <seconds>.	--ifconfig-pool-persist /etc/openvpn/IPs 100
client-to-client	-	All clients are allowed to connect to each other.	--client-to-client
tmp-dir	<directory>	Specifies a directory for temporary files.	--tmp-dir /etc/openvpn/tmp
max-clients	<number>	Maximum number of clients allowed to connect.	--max-clients 5
max-routes-per-client	<number>	Maximum number of routes possible for a single client.	--max-routes-per-client 5
connect-freq	<number> <seconds>	A client is allowed to connect this <number> of connections per specified <seconds> as maximum.	--connect-freq 5 120
learn-address	<cmd>	Shell script command <script> to validate client virtual addresses or routes.	--learn-address /etc/openvpn/script.sh
auth-user-pass-verify	<script> <method>	OpenVPN will execute script as a shell command to validate the username/password provided by the client.	--auth-user-pass
client-cert-not-required		Doesn't require client certificate, client will authenticate using username/password only.	--client-cert-not-required
duplicate-cn	-	Uses one client certificate for several (or all) clients.	--duplicate.cn

The following parameters are my favorites in server mode:

- `--push`: A complete new scope of VPN functionality is opened up here. The table in the section *Push options* shows how you can *push* configuration options to clients connecting to the VPN on initialization of the connection.

- `--ifconfig-pool` and `--ifconfig-pool-persist`: Define in detail how and which IP addresses the server is supposed to provide to clients connecting.

- `--client-to-client`: If your VPN clients need connections between them, this option will help.

  ```
  (...)
  client-to-client
  (...)
  ```

  Simply adding the previous code snippet to the configuration file will enable clients to connect to each other through the tunnels.

- `--max-clients` and `--max-routes-per-client`: Restrict the number of clients that are allowed to connect to the VPN server and the number of routes that are allowed to be set to one client.

- `--client-cert-not-required` and `--duplicate-cn`: You may *loosen* certificate restrictions a little, but this may be dangerous!

- `--auth-user-pass-verify <script><method>`: This is really handy. A script is called for authentication, the method specified is used to pass authentication data received from the client to this script.

The method `via-env`, for example, calls a script with the parameters `username` and `password`, expecting a return code of 0 for success, 1 for failure. Can you imagine what a simple Perl script can do here? Authentication against Active Directory, **Lightweight Directory Access Protocol** (**LDAP**), and many more are possible. Sample scripts can be found in the source-code package of OpenVPN, in the file `sample-scripts/auth-pam.pl`.

The file `/usr/share/doc/openvpn/README.auth-pam` holds information on the usage of the Linux Authentication Standard **Pluggable Authentication Modules** (**PAM**) for authentication of VPN clients. PAM itself is built on a modular basis, so that none of your wishes should be unfulfilled.
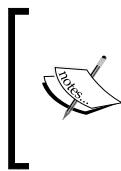
OpenVPN in server mode can assign IP addresses dynamically to clients, but you can specify exceptions. Client-to-client connections are possible, and certificates can be used for multiple clients. The parameter `--auth-user-pass-verify` can be used to verify passwords and usernames against PAM or arbitrary scripts.

--client-config options

We have learned by now that TLS clients can be assigned individual configurations based on the common name of their certificates. For this purpose, we only need to create a client configuration directory, tell `openvpn` where this directory is to be found, and put our client configurations in this directory. The name of the client configuration file must be identical with the common name of the certificate the client uses. This is very important: Only with this field in the certificate can `openvpn` distinguish the clients. A client with the common name `server1` in its certificate must be configured with `server1.conf` in the client configuration directory. The name of this file cannot be chosen.

The parameter `-client-config-dir` is used to tell `openvpn` where to look for the clients' configurations:

Parameter	Options	Function	Usage
client-config-dir	<directory>	The path to our client configuration directory	--client-config-dir / etc/openvpn/ clients
ccd-exclusive		Requires, as a condition of authentication that a connecting client has a --client-config-dir file	--ccd-exclusive

The parameter `ccd-exclusive` allows connections only for clients that have a client configuration file in the client configuration directory.

In a client configuration file almost all parameters and options used in a normal configuration file can be used along with the following parameters, which are only valid in a client configuration file:

Parameter	Options	Function	Usage
`client-connect`	`<script>`	Runs script when a client connects successfully	`--client-connect /file.sh`
`client-disconnect`	`<script>`	Runs script when a client disconnects successfully	`--client-disconnect`
`ifconfig-push`	`<IP> <IP>`	Pushes IP endpoints for client tunnels, overriding the settings from `ifconfig-pool` — useful in client-specific configurations (on the server)	
`iroute`	`<network> <netmask>`	Generates an internal route to a specific network through a VPN client	`--iroute 10.94.0.0 255.255.255.0`

- `--client-connect` and `–client-disconnect`: Allow execution of scripts on connection or disconnection of a client of our VPN—another handy possibility for solving many interesting issues.

- `--ifconfig-push`: Sets the IP of the tunnel endpoints for this connection to different values than specified with `ifconfig-pool`, a convenient method of specifying a fixed IP for a client.

- `--iroute`: Allows setting an internal route to a network *behind* a VPN client, enabling partners on the server side to access the network behind the tunnel (on the client's side). This parameter is very interesting in a scenario like the following: Mr. Smith connects to the VPN server of his company from the LAN at his home with the network address 10.94.0.0/24. He is working on the terminal server in the central branch of his company. Now he wants to print a document, but on a network printer 10.94.0.200 in his home LAN. To fulfill this, the terminal server must have configured this network printer, and therefore it needs to know how to route to 10.94.0.200. Besides setting the route on the company's default gateway pointing to the VPN server, the VPN server itself must also know that this network address is behind the VPN client. All machines that act as routers in this scenario must be configured to do forwarding, including correct firewall setup and access rights to the printer. In our scenario, both VPN partners (also the VPN client machine) must have forwarding enabled!

Client mode parameters

The following table shows parameters that are relevant if your VPN machine acts as a TLS client to a VPN server. `--client` stands here, similarly to `--server`, as a shortcut for two parameters: `--tls-client` and `–pull`. We have talked about `--tls-client`, but not about `–pull`, which simply tells `openvpn` to try to get routes and network configuration from the VPN server.

Parameter	Options	Function	Usage
`client`		Simply pull TLS server option.	`--client`
`pull`		Gets pushed routes and more from the server.	`--pull`
`auth-user-pass`	`<file>`	Authenticates to the server using the username/password pair specified on two lines in `<file>`.	`--auth-user-pass /etc/openvpn/passes`
`auth-retry`	`<interact>` `<noninteract>` `<none>`	Determines the client's behavior on authentication failure. `<interact>`: The client will prompt the user. `<non-interact>`: The client will keep on trying. `<none>`: The client will exit with an error message.	`--auth-retry noninteract`

- `auth-user-pass` and `auth-retry`: These are the client's settings for authentication with a password, where `auth-user-pass` simply wants a file with a username/password pair in it. If called as `–auth-user-pass up`, `openvpn` will prompt for the username/password pair. All of this will only work if the server has `–auth-user-pass-verify` configured properly.

- `auth-retry`: With this parameter, we can specify how OpenVPN clients deal with authorization errors or failures. Unattended systems should be set to `--non-interactive` because otherwise they would stop connecting if a connection error occurs. A road warrior's laptop can be configured to prompt the user because there might be other problems that prevent the tunnel (like a firewall). And `none` is the best solution for the paranoid—If authentication fails just once, no further attempt to set up a tunnel will be made, the `openvpn` process exits.

Push options

Pushing configuration parameters to clients is one of the really great features of `openvpn`.

Parameter	Options	Function	Usage
`push`	`<configuration options>`	Push the `<configuration options>` to the client.	`push "route 192.168.20.0 255.255.255.0"`

An OpenVPN server (running in server mode) can push the following settings to a client (that has the *pull* parameter enabled). You should know most of them by now. Can you imagine how they work without looking in the right column? (Only the last two are new.)

Push Parameter Option	Function
`--route`	The client will set a route.
`--route-gateway`	The client will set its gateway.
`--route-delay`	The client will wait a little before setting its routes.
`--redirect-gateway`	The client will redirect its gateway through the VPN.
`--inactive`	The client will exit after a specified time.
`--ping`, `--ping-exit`, `--ping-restart`	The client will change its ping behavior.
`--persist-key`, `--persist-tun`	The client will change its behavior on restart.
`--comp-lzo`	The client will use compression.
`--dhcp-option`	The client will use specific DHCP options (Windows only, see later).
`--ip-win32`	The client will use the method specified to set IPs and network addresses (Windows only, see later).

It's very important to set the quotation marks correctly. Anything between them will be sent to the client as a configuration directive.

> A VPN server can push routing, network, and DHCP options to a client. Ping behavior and other features can be controlled by the server and set on connection initialization.

Important Windows-specific options

A fast-growing number of options can only be used on Windows clients because other systems can't deal with the methods used. The following table gives an overview of these:

Parameter	Options	Function	Usage
`dhcp-option`	`WINS <IP>` `DNS <IP>` `DOMAIN <name>` `NBDD <IP>` `NTP <IP>` `NBT <type>` `NBS <scope-id>` `DISABLE-NBT`	Sets specific DHCP data over the VPN for Windows clients. Sets a specific DNS or WINS server through DHCP, sets domain name, NetBIOS server address, network time server, and more.	`--push "dhcp-option DNS 10.94.46.11"`
`route-method`	`ipapi` `exe`	Sets the method Windows uses to set routes, either by executing the route command (`exe`) or by using the IPAPI interface.	`--route-method ipapi`
`ip-win32`	`<method>`	Sets the Windows Network adapter's IP and netmask using `<method>`.	`--ip-win32 ipapi`

If you are running Microsoft Vista or Server 2008, OpenVPN versions earlier than 2.1 will need the following two commands to work properly:

```
route-method exe
route-delay 2
```

There are several other options for the Windows user, but only few of them will come in your way. Here is a short glance at them:

- `--win-sys path`: Define a different directory for your Windows' system binaries.
- `--tap-sleep`: Sets the TAP device to sleep for a specified amount of time. Sometimes this solves some problems with the Windows IP helper API.
- `--show-net-up`: Echoes OpenVPN's networking setup to syslog.
- `--dhcp-renew`: Tells Windows to initialize the DHCP client for the TAP device. `--dhcp-release` frees the adapter.

- `--service exit-event 0/1`: Useful if OpenVPN is running in the background. Multiple OpenVPN processes can listen on this Windows event and, for example, shut down simultaneously without user interaction.

- `--show-adapters`: Works on Windows systems similar to the `ifconfig` command on Linux and shows all available network adapters.

- `--allow-nonadmin <device>`: Enable users without administrative privileges to activate the named adapter. Without a device specified, all adapters are accessible for non-administrators.

- `--show-net`: Shows the network from OpenVPN's point of view.

New in Version 2.1

OpenVPN's newest version comes with four new major concepts:

- Connection profiles
- Topology modes
- Script-security
- Port-sharing

Connection profiles

In a client's OpenVPN configuration file, administrators can now specify whole groups of configuration parameters, each of which describes one connection, to a VPN server. The connection profiles can be seen as an extension to several `remote` options with the advantage that you can specify different ports, options, and parameters. Once the client has unsuccessfully tried the first connection with its block of options, and fails, it proceeds to the next block, until a VPN connection can be established.

This can be very handy, for example, when your client notebook is regularly located in unknown networks and you want OpenVPN to try all possible ways of contacting your server. Just have your server listen on the ports 21, 80, and 443 and then create one connection block for each of these ports. Your client will try each of them, until it (hopefully) manages to establish the connection.

The syntax for this example is simple. Each connection is specified by an opening `<connection>` and a final `</connection>`:

```
<connection>
remote myvpnserver.org 21
</connection>
```

```
<connection>
remote myvpnserver.org 80
</connection>
<connection>
remote myvpnserver.org 443
</connection>
<connection>
remote myvpnserver.org 443
http-proxy 192.168.0.1 3128
</connection>
```

As you see in the last profile, other options like those regarding proxies can also be used. According to the man page, allowed options in connection profiles include: `bind, connect-retry, connect-retry-max, connect-timeout, float, http-proxy, http-proxy-option, http-proxy-retry, http-proxy-timeout, local, lport, nobind, port, proto, remote, rport, socks-proxy,` and `socks-proxy-retry. --Remote-Random` can be helpful outside a connection statement to make OpenVPN randomly choose from the list of connection profiles.

Topology mode

If you are using TUN devices, and the point-to-point-topology of the standard TUN-device handling doesn't suit your needs, here is the solution. Especially, administrators having trouble with the fact that in standard TUN mode OpenVPN needs two IP addresses per client, will be happy about `topology subnet`. Starting with OpenVPN 2.1, an admin can choose whether a TUN device will use one of the following network topology modes:

- **net30**: The default behaviour of OpenVPN 2.0, using a /30 subnet, and two IP addresses per client.
- **p2p**: One single IP address is used per client, the remote endpoint always points to the local endpoint. Only works with Windows.
- **subnet**: Use a topology similar to the TAP interfaces. Only one IP address per client is needed, but the client must support a subnet instead of a remote endpoint address.

With `topology subnet` in your `tls-server` configuration file with TUN devices, your clients will get consecutive IP addresses, whereas without the `topology` directive, each client needs two IP addresses.

Script-security

The directive `script-security` is followed by a level of your choice and introduces the beginnings of security policies to OpenVPN. Its syntax is:

```
--script-security level method
```

The level ranges from 0 to 3, a script security level of '0' forbids all calling of external programs, with '1' only built-in executables are allowed. If you want your own scripts to work, you will need at least level '2'. Administrators that need passwords being passed on through environmental variables must use script security level '3', as this is considered extremely unsafe. However, older versions did not have such a control tool, and thus for compatibility reasons '3' is the default.

The method parameter can either be `execve` or `system` and describes how OpenVPN calls external commands. `execve` is the default, and makes use of the `execve()` function on Unix and `CreateProcess()` on Windows. `system` instead calls the `system()` function and is considered less safe.

Port-sharing

Last but not least, a very simple but handy new feature is port sharing. In TCP Mode, OpenVPN can share its port with another program. This can be an almost perfect camouflage, if you are running an HTTPS server like Apache on the same port. OpenVPN automatically proxies all non-OpenVPN traffic to the specified host and port. OpenVPN-traffic is handled as usual. The syntax is:

```
--port-share host port
```

It's as simple as that. Replace `host` with the IP or DNS name of your web server, and `port` with the port to which you want to proxy the connections.

Test

Are you ready for an example? Read the following command line and write down what it does:

```
openvpn --port 5001 --proto udp --dev tun --ca ca1.crt --cert opteron.
crt --key opteron.pem   /
--crl-verify revoke.crl --dh dh2048.pem --server 10.79.2.0
255.255.255.0  /
--push "route 10.19.46.0 255.255.255.0" --push "route 10.18.46.0
255.255.255.0"  /
--push "dhcp-option DNS 10.19.46.15" --push "dhcp-option WINS
10.19.46.12"  /
--client-to-client --keepalive 10 60 --comp-lzo  /
```

```
--status /var/log/openvpn/openvpn-road-status2.log  /
--log-append  /var/log/openvpn/openvpn-road2.log  --verb 4
```

Here is the solution:

This `openvpn` command starts a TLS server listening on port UDP 5001 with the specified certificates, key, and revoked list files. The virtual network has the address 10.79.2.0/24, clients are pushed several routes and DHCP options, (which means they are probably Windows clients), clients are allowed to connect to each other, the traffic is compressed, and log and status messages are written to files in `/varlog/openvpn` at a verbosity of 4.

Summary

In this chapter we started with explaining the syntax of `openvpn` and its configuration file. Parameters that are in our standard configuration file were followed by the ones used during setup of a certificate-based tunnel. From then on we traveled through the basic tunnel parameters, encryption, server mode and client mode. We finished this chapter with parameters that are only available on Windows systems and new features of the forthcoming version 2.1 of OpenVPN.

10
Securing OpenVPN Tunnels and Servers

In this chapter, we will learn how to make the example tunnels we created safer and more persistent by choosing a safe combination of configuration file parameters. We will then discuss authentication and plugins, and how to install and use a firewall with a convenient web-based configuration interface on a standard Linux system, namely Shorewall on a Debian system. After that we will have a look at the **SuSEfirewall 2** that comes with OpenSuSE. A short look will deal with how to configure the Windows XP firewall for use with OpenVPN. Last but not least, we will discuss the possibilities that the Linux command line offers (especially with the examples that come with OpenVPN).

Securing and stabilizing OpenVPN

Up to now, we have built several tunnels and all of them were built with simple mechanisms and focused on simplicity. In this chapter, we will set up an OpenVPN server and tunnels that can be used in a production environment. For this purpose we will use strong encryption layers, which OpenVPN offers, and set some parameters in our configuration file to make sure that OpenVPN keeps running. This will be our first task.

Here is a configuration file for our VPN server for enabling access only for one client. Perhaps it's a good idea that you have a look at the following options and parameters before you read on. This is far from perfect, especially because there is a constant development concerning security going on and hence I do not try to give an example with the highest possible security. Nevertheless, there are some features enabled in this configuration that have proven very helpful.

```
float
dev tunVPN0
tun-mtu 1500
ifconfig 10.179.10.1 10.179.10.2
port 5000
route 10.194.0.0 255.255.0.0 10.179.10.2
comp-lzo
auth SHA512
cipher AES-256-CBC
tls-cipher DHE-RSA-AES256-SHA
tls-auth keys/tls-key.txt
tls-server
tls-remote "/C=DE/ST=BY/O=Feilner-IT/CN=VPN-
Client/emailAddress=security@feilner-it.net"
ca certs/ca.crt
cert certs/server.crt
key certs/server.key
dh dh2048.pem
keepalive 10 60
shaper 20000
route-up "/sbin/firewall restart"
log-append /var/log/openvpn/feilner-it.log
status /var/log/openvpn/feilner-it.status 5
```

An explanation of the options and parameters of the configuration file is as follows:

- `float`: The VPN server accepts connections from clients even if their IP addresses change.

- `dev tunVPN0`: We will use the network device `tunVPN0` for connections. Because the name of the device can be chosen freely, it may be a good idea to use a significant name.

- `ifconfig`: These are the virtual IP addresses of our tunnel network.

- `port`: We will use port 5000 for the VPN communication.

- `route`: This server is told that the subnet 10.194.0.0 is behind the other end of the tunnel.

- `comp-lzo`: All traffic will be compressed before transport.

- We tell OpenVPN to use stronger encryption methods than the standard methods:

```
auth SHA512
cipher AES-256-CBC
tls-cipher DHE-RSA-AES256-SHA
```

Use the commands `openvpn --show-ciphers`, `openvpn --show-digests`, and `openvpn --show-tls` to find out the encryption mechanisms available on both systems. There will be differences depending on the operating systems and software versions used. You must use methods that both systems are capable of.

The values in the file listed are merely examples that will differ from your real setup.

- `tls-auth`: This provides a simple **Denial of Service** (**DOS**) protection. DOS is a kind of attack where somebody tries to *flood* your machine and thereby slow down (or stop) regular connections. An OpenVPN machine with `tls-auth` activated will only accept packets encrypted with the correct HMAC signature generated from the key specified in the file (for example, `tls-key.txt`). The OpenVPN man page speaks of an 'HMAC Firewall'. This option should always be applied when your system is accepting connections from varying IP adresses.

- `tls-server`: This specifies the role that the OpenVPN machine will take for setting up the tunnel and exchanging certificates.

- `tls-remote"/C=DE/ST=BY/O=Feilner-IT/CN=server2/ emailAddress=security@feilner-it.net"`: This specifies the exact subject line of the VPN partner's certificate. This line makes sure that only the VPN partner presenting this certificate is allowed to connect to our VPN. You can extract this line from your certificate file. At a verbosity level of `5` or higher, you will also find this 'subject' line explicitly in the logfile of your VPN machines.

- The following lines specify the location of TLS certificates and keys and the Diffie-Hellman key:
  ```
  ca certs/ca.crt
  cert certs/server.crt
  key certs/server.key
  dh dh2048.pem
  ```

- `keepalive 10 60`: We add these parameters ensuring that the tunnel will be restarted automatically.

- `shaper`: This option must be used on both sides, and limits the traffic through this tunnel to about 20K.

- The last three lines define a firewall script that is run when the tunnel is set up and the location of log and status files.

Our VPN client should receive basically the same configuration, with changes only to the location and names of files and certificates. We will need to type the subject line of the certificate of the server here and we will need a `remote` directive telling our client where to connect to and that our system will be trying to resolve the other hostname for one day before giving up.

```
remote xxx.dyndns.org
(...)
tls-remote "/C=DE/ST=BY/O=Feilner-IT/CN=VPN-Server/
emailAddress=security@feilner-it.net"
(...)
resolv-retry 86400
```

So how can we sum this up in a nutshell?

> With the configuration above:
>
> Our OpenVPN server will only start the connection setup process from an OpenVPN client that authenticates with the correct HMAC signature generated by a static, pre-shared key. The connection process will only be successful if both partners know and can handle the correct ciphers and encryption methods specified. Only the machine offering the X.509 certificate specified in the line starting with `tls-remote` will be accepted.

Some lines of this configuration help re-establishing the tunnel after connection errors and make sure that the systems will try to resolve DNS for one day before giving up.

I guess this configuration is not yet paranoid, but already quite secure, as long as we are careful with our keys and certificates.

Authentication

What we have done until now is securing the tunnel by using strong encryption and client and server certificates. However, these certificates do only secure the machine they were created for. You may argue, what about password protection? That is a fair point, but today central authentication is an important focus of any IT. Thus, a password stored in a certificate created years ago may be difficult to remember. I prefer the two-stage scenario of client and server certificates plus authentication of the client user at logon. All of the GUIs presented in the next chapter support these methods. Let's roll!

Using authentication methods

We have learned before that OpenVPN can be used with authentication based on shared secrets (static keys) and X.509 certificates. Another useful option for authentication is authentication plugins called with the configuration parameter `auth-user-pass-verify`, which can be used together with both methods mentioned before. For example, in a certificate-based VPN, we can use an authentication plugin to make sure that only a user knowing the appropriate username/password combination can start the tunnel. This may be a convenient additional level of security for laptops or other road-warrior machines.

While certificates in this context tend to protect and authenticate machines rather than users, username/password combinations are useful for VPNs that are started by a human. The Windows GUI will pop up a small authentication window where the user must enter a username and password. The VPN client takes these values and sends them to the VPN server, which starts the plugin program (as configured in `auth-user-pass-verify`) to validate the combination. If the authentication program returns an OK, authentication was successful, and the tunnel is created. The tunnel will only be established if the password is correct.

For this purpose, the following configuration parameters must be added. In the server configuration file, add `auth-user-pass-verify /path/to/your/auth/script` to your server configuration and `auth-user-pass` to your client's configuration. The following table shows the usage of these parameters:

Parameter	Allowed options	Usage	Function
`--auth-user-pass-verify`	`<script>` `<method>`	Server configuration	Activates server's authentication and defines the name of the authentication script and the method to use for username/password handling
`--auth-user-pass`	`<file>`	Client configuration	Activates client's authentication and optionally defines a file where username and password are stored

On SuSE systems, there are some example scripts (like `auth_pam.pl`) provided with OpenVPN, which can be found in `/usr/share/doc/packages/openvpn/sample-scripts`. But a typical scenario for such an authentication may be a local LDAP server. LDAP is the system-independent state of the art for all modern directory services, both in open source servers and also in Microsoft's Active Directory Service. The following overview will give you some hints on how to create an authentication plugin using your own LDAP authentication for OpenVPN.

On a Linux system with the LDAP client tools installed, the command `ldapwhoami` can be used for testing username/password pairs against an LDAP server. In the following examples the LDAP server is `10.10.10.1`, the user `mfeilner`, and the password is `correct_password`. The string `uid=mfeilner,ou=Feilner-it_Users,dc=feilner-it,dc=home` must be adapted to the settings on your LDAP server. Here is the output of the `ldapwhoami` command:

```
suse01:/var/log # ldapwhoami -x -h 10.10.10.1 -D
uid=mfeilner,ou=Feilner-it_Users,dc=feilner-it,dc=home -w correct_
password
dn:uid=mfeilner,ou=Feilner-it_Users,dc=feilner-it,dc=home

suse01: # ldapwhoami -x -h 10.10.10.1 -D uid=mfeilner,ou=Feilner-it_
Users,dc=feilner-it,dc=home -w wrong_password
ldap_bind: Invalid credentials (49)
```

The first command will give a return code of '0', whereas the second command, resulting in a failed authentication returns a value of '1'. Creating a little script, that implements the aforementioned LDAP command and returns 0 if authentication was successful and 1 if authentication has failed, is easy and I leave this up to you. An example for such an LDAP authentication plugin script for OpenVPN can be found at `http://www.indato.ch/openvpn/openvpn.html`.

Even though this web site is in German, the LDAP script found here is documented in English. You can find it if you scroll down until the heading **Optionale Authentisierung mit LDAP**. An English web site with an **OpenVPN Auth-LDAP Plugin** can be found at `http://code.google.com/p/openvpn-auth-ldap/`.

The phpLDAPadmin tools is probably one of the best LDAP administration tools. If you are thinking of setting up an LDAP server (which can be used for a variety of purposes), have a look at this screenshot of **phpLDAPadmin** on an LDAP server with the entry `uid=mfeilner,ou=Feilner-it_Users,dc=feilner-it,dc=home`, which was used for authentication in the example earlier.
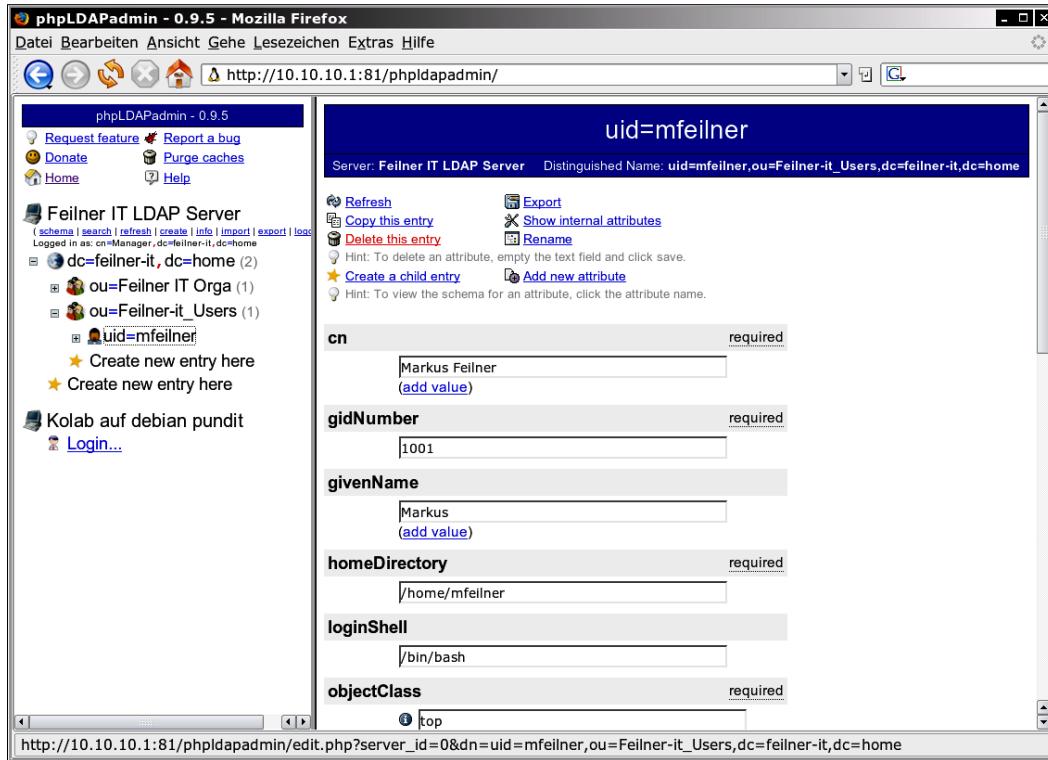
On the left is the LDAP directory tree, on the right the properties of the selected object. Here we can change, for example, the password for the OpenVPN account, create and delete accounts, and thus manage access to our VPN on the basis of the selected authentication plugins.

Authentication plugins overview

A very good read is the web site of the European OpenVPN Club Openvpn e.V., a registered club with the goal of supporting and providing contact between the various OpenVPN users and specialists. The forum of this club holds a great list of plugins for OpenVPN authentication modules. This table can only show an excerpt from that. The maintainers are busily working on translating all content that is not available in English up to now.

Plugin	Code	How-to and documentation
LDAP	`http://code.google.com/p/openvpn-auth-ldap/`	Readme included
Radius	`http://www.nongnu.org/radiusplugin/`	`http://forum.openvpn.eu/viewtopic.php?t=2116` and `http://www.howtoforge.com/openvpn_wikid_strong_authentication`
POP3	`http://popauth.kimballstuff.com/source.html`	`http://www.wenzk.net/bbs/thread-221-1-1.html`
SQLite	`http://www.trooth.cc/projects/authsqlite/`	`http://www.vpnforum.de/openvpn-forum/viewtopic.php?t=3178`
MySQL	Locally through PAM	`http://www.vpnforum.de/openvpn-forum/viewtopic.php?t=3591`
USBAuth	Locally through PAM	`http://usbauth.delta-xi.net/doku.php`
Samba (Windows domains)	Locally through Samba and PAM	`http://www.vpnforum.de/openvpn-forum/viewtopic.php?t=1748`
Local User/Password (PAM)	`http://auth-passwd.sourceforge.net/`	Pam—How-to
Strong auth with WIKID	Through Radius and Wikid Server	`http://www.howtoforge.com/openvpn_wikid_strong_authentication`

Plugin	Code	How-to and documentation
Universal Plugin	`http://frost.ath.cx/ software/openvpn_ auth/`	(Supports almost all existing authentication backends)
openvpn_authd and openvpnClientConnectLDAP	OpenVPN authentication server and client—perhaps a dead link: `http://frost.ath.cx/ software/openvpn_ auth/`	`http://openvpn.net/ archive/openvpn- users/2007-03/ msg00256.html`

Authentication with tokens

If you want to use Hardware tokens such as those of Aladdin Software, here is a brief how-to.

> Thanks to Daniel Salcher of OpenVPN e.V. for this submission!

This setup is running in several offices in Germany, at prosecutors and other legal companies with high security needs. It is used for remote administration and home work for the lawyers, and allows access to FTP, SSH, and VNC. Aladdin's Tokens can be received from Aladdin at `http://www.aladdin.de/how/resellers.aspx`, with the hardware for about 40 Euro and Support for 16 Euro. The working of the Aladdin software is very simple. The USB-token is merely a safe place to store your certificate. So there is no need to make any changes to your server, just generate certificates, export them to your client, and have Aladdin's programs store and encrypt them on the USB stick.

All the client needs is Aladdin's software (or similar) installed, and the location and libraries/programs for accessing the Token. With OpenVPN, this is specified in the configuration file:

This is an example (Windows) client configuration file:

```
tls-client
client
dev tap
proto udp
tun-mtu 1500
ip 1194
ca CA.pem
```

```
cryptoapicert "THUMB:0b 58 43 44 3d 45 xx bb e5 55 22 xx 2b 2f 26 ba
56
89 7c 36"
cipher BF-CBC
comp-lzo
verb 3
ns-cert-type server
pull
```

And this is a working configuration for a Linux client:

```
dev tap
tls-client
client
proto udp
tun-mtu 1500
remote test.org 1194
ca ./CA.pem
pkcs11-providers /usr/lib/libeTPkcs11.so
pkcs11-slot-type label
pkcs11-slot "Daniel Salcher"
pkcs11-id-type subject
pkcs11-id "/C=DE/ST=BY/O=xy/CN=SAVATEC e.K. - Daniel
Salcher/emailAddress=info@xy.de"
cipher BF-CBC
comp-lzo
verb 3
ns-cert-type server
pull
```

If this list still is not enough, then you will be happy with Pam-per-user.

Individual authentication with Pam-per-user

If this long list is not enough for you, the small tool `pam-per-user` will help you. With it you can tell your Linux system which authentication source to use for a specific user. You may, for example, have 10 VPN users authenticating against Active Directory, 10 others against LDAP, and 10 against some proprietary IMAP server, while the rest are POSIX standard Linux users. However, don't try this on Windows, it won't work there, you'll need a Linux server.
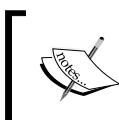
> A big 'Thank you' to Ralf Hildebrandt and the folks from the Charité Hospital of Berlin University for showing me this great tool! Pam-per-user is really great, not just for OpenVPN.

Pam-per-user is a PAM module that simply delegates PAM authentication to other services. In the file `/etc/pam_per_user.map` you may define the service that will be used for a user. But before that an entry in `/etc/pam.d/openvpn` is necessary:

```
auth required pam_per_user.so.1
account required pam_permit.so
```

The next step is an entry in `/etc/pam_per_user.map`:

```
userA : openvpn-ldap
userB : openvpn-krb5
userC : openvpn-winbind
userD : openvpn-imap
```

What happens here is that userA authenticates against LDAP while the server uses a Kerberos server for userB, Winbind for UserC (which is a client for Windows servers, like Active Directory or NT LAN Manager NTLM), and IMAP for UserD.

The file `openvpn-ldap` for userA should reside in the directory `/etc/pam.d` and have a content like:

```
auth required pam_env.so
auth required pam_ldap.so
account required pam_ldap.so
```

The appropriate configuration for userB in `/etc/pam.d/openvpn-krb5` would look like:

```
auth requisite pam_krb5.so no_cache
account required pam_permit.so
```

An interesting thing here is the link to the Kerberos library `pam_krb5.so` whose configuration file `/etc/krb5.conf` will need further work. The same applies to the Samba Winbind configuration, especially if you want to do Active Directory. This is done in four steps very well documented at the Samba project, for example at `http://wiki.samba.org/index.php/Samba_&_Active_Directory`. The four steps are:

- Install Samba, Winbind, and all relevant services including the Samba client package for testing.
- Configure your `samba.conf` for domain membership.
- Enter your domain with your domain administrator's password.

  The command is `net join -U Adminstrator`, if you want to use your admin.
- Create a `pam_winbind.so` for your needs. The Samba packages (especially those of Winbind) bring samples.

More universal and easier to set up is the IMAP trick. UserD has to authenticate against an IMAP server. If you have an Exchange server, take his or her IMAP Login and you have single sign-on against your Active Directory. It won't get simpler!

The file `/etc/pam.d/openvpn-imap` looks like:

```
auth required pam_imap.so conf=/etc/pam.d/pam_imap.conf
account required pam_imap.so conf=/etc/pam.d/pam_imap.conf
```

And in `/etc/pam.d/pam_imap.conf`, the information about the server, his or her SSL certificate, and more.

```
PAM_PasswordString = Password:U
CertificateFile /usr/share/ssl/certs/U
imapd.pem
PAM_Server0 = imapserver.example.com:143
PAM_BlockList = root, admin, Administrator,U
apache
PAM_HashEnable = no
PAM_HashFile = /etc/pam_imap.gdbm
PAM_HashDelta = 20
```

With this, you can use many kinds of authentication for your VPN: Novell Groupwise? Scalix? No Problem. Almost every proprietary solution has an IMAP server authenticating against it. Seems perfect, doesn't it?

However, don't forget the two small steps:

1. On the VPN server, add `plugin/usr/lib/openvpn/ openvpn-auth-pam.so openvpn` to your OpenVPN configuration.
2. On the client, add, `auth-user-pass`.

Linux and Firewalls

Now that OpenVPN is configured safely, how about the system that it runs on? On Linux there are several excellent firewall solutions that can be used with OpenVPN. On the following pages we will deal with two firewalls, which offer graphical interfaces for configuration—Shorewall (with Webmin) and the SuSEfirewall as delivered with OpenSuSE 10.

Debian Linux and Webmin with Shorewall

Webmin is an excellent GUI for Linux system management, if your preference is for
web-based administration. Webmin can be found on `http://www.webmin.com` and
offers almost full control over your Linux systems. It brings a small web server of
its own and supports SSL encryption, user management, and more. However, I do
not want to conceal the fact that there are Perl scripts that set system variables in
files at `/etc`, which is not considered best practice. However, as always, security and
usability are enemies and the compromises may vary. If we use Webmin, we must
secure access to it. A good idea is a separate OpenVPN tunnel for it.

Installing Webmin and Shorewall

Besides Webmin, we will enable SSH access to our Debian system.

```
vpnserver:/home/mfeilner# wget http://switch.dl.sourceforge.net/
sourceforge/webadmin/webmin_1.470_all.deb
--01:22:51--  http://switch.dl.sourceforge.net/sourceforge/webadmin/
webmin_1.470_all.deb
           => `webmin_1.470_all.deb'
(…)
Length: 13.760.140 (13M) [application/x-debian-package]
100%[================================================================
=========>] 13.760.140     5.66M/s

01:22:54 (5.66 MB/s) - »webmin_1.470_all.deb« saved
[13760140/13760140]
```

That should be all! However, on all systems you will need some libraries before
installing Webmin:

```
vpnserver:/home/mfeilner# aptitude install libnet-ssleay-perl
libauthen-pam-perl libio-pty-perl libmd5-perl
(…)
Fetching: 1 http://ftp.de.debian.org lenny/main libnet-ssleay-perl
1.35-1 [206kB]
Fetching: 2 http://ftp.de.debian.org lenny/main libauthen-pam-perl
0.16-1.1+b1 [32,9kB]
Fetching: 3 http://ftp.de.debian.org lenny/main libio-pty-perl 1:1.07-
1+b1 [40,7kB]
Fetching: 4 http://ftp.de.debian.org lenny/main libmd5-perl 2.03-1
[5700B]
vpnserver:/home/mfeilner# dpkg -i webmin_1.470_all.deb
(...)
Webmin install complete. You can now login to https://vpnserver:10000/
as root with your root password, or as any user who can use sudo
to run commands as root.
vpnserver:/home/mfeilner#
```

You are told that Webmin uses a separate password file in `/etc/webmin/miniserv.users`. Confirm this dialog with the **OK** button. This is important, you should never send your root password over a web connection.

The Shorewall firewall is also installed with a simple `aptitude install shorewall`. Webmin comes with a full-featured and intuitive GUI to control your firewall, but the configuration files are probably the faster way to edit your rules.

Looking at Webmin

This is the Webmin login screen:
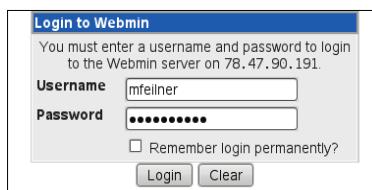
However, there are still some small adjustments that you need to make. Webmin must be secured and configured. This is what the new GUI looks like, everything is done by mouse clicks.

Preparing Webmin and Shorewall for the first start

After installation, you find Webmin installed in `/usr/share/webmin` and the Webmin configuration in `/etc/webmin`. The file `miniserv.conf` contains the basic configuration for access and authentication. The Webmin documentation on the web site is the best place to look for the meaning of these options. At this point you will only need to change one line.

> Change the line `allow=127.0.0.1 /etc/webmin/miniserv.conf` to the address of the client that you want to use for accessing Webmin and type `/etc/init.d/webmin restart`.

Webmin can now be reached from the system you specified with a standard browser (supporting cookies and JavaScript is recommended, but not necessary) on the URL `https://ip-of-our-webmin-server:10000`.

There are only two small changes to configuration files in the Shorewall setup that need editing:

> Set the parameter `startup=0` to `startup=1` in `/etc/default/shorewall`.
>
> Enable forwarding in `/etc/shorewall/shorewall.conf` by changing the line `IP_FORWARDING=Keep` to `IP_FORWARDING=On`.
>
> Create your own certificates for Webmin, either within Webmin or use your PKI from OpenVPN for that.

Of course we can use the certificates generated by OpenVPN's `easy-rsa` for Webmin too, and the best way to do this is generating certificates for Webmin. Perhaps you type something like `webmin-server01` or similar in the **Common Name** field of the certificate and key. If you have certificates, you only have to put them on the server running Webmin and enter the path in the fields in the right dialog. By doing so, you have certificates nobody else is using, which is definitely not true of the original Webmin certificates. In the previous example, the keys are placed in `/etc/openvpn/keys`, but you can choose the location freely.

> Install the Webmin Module for OpenVPN with **Webmin | Webmin Configuration| Webmin Modules | From ftp or http URL** from `http://www.openit.it/index.php/openit/content/download/3566/14482/file/openvpn-2.5.wbm.gz`
>
> Now add a user to use with Webmin only for firewall configuration. Go to **Webmin | Webmin Users** and click on **Create a new Webmin user** hyperlink. Add your username, password, and select the modules **OpenVPN** and **OpenVPN + CA**.

Preparing the Shoreline firewall

This is what Webmin with the Shorewall / Shorewall Firewall GUI looks like:
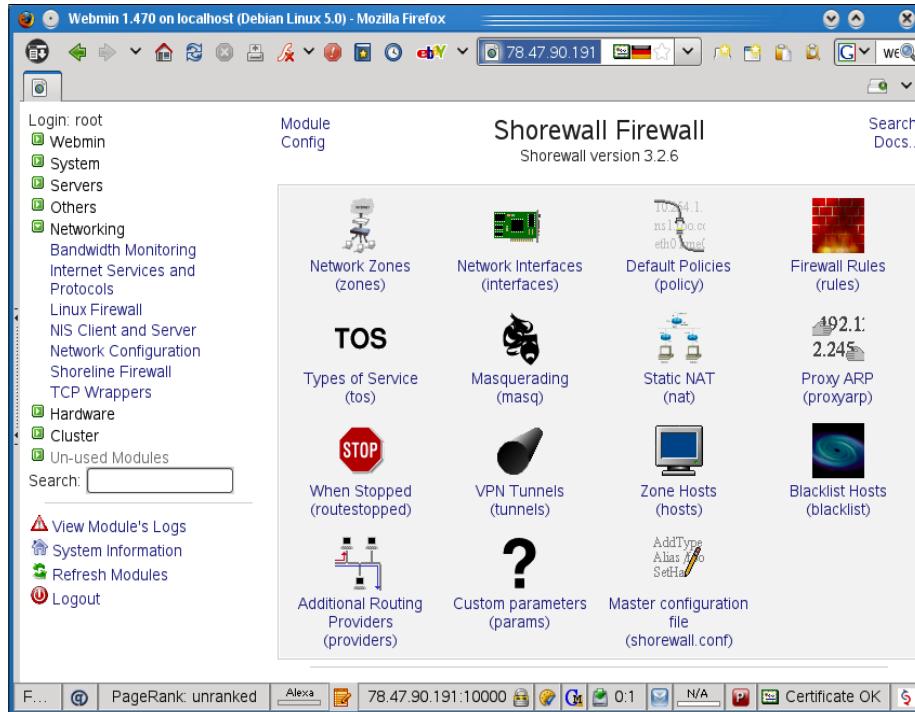
Before we proceed, we need to collect some information:

- What port and protocol is OpenVPN running (by default its UDP port is 1149)?
- What are the names of the network interfaces?
- What is the IP address or DNS name of the VPN partner?

We will now enter this data in our firewall configuration and close all other access except SSH traffic. Thus our firewall will have only two ports open from outside—SSH and OpenVPN. What you want or need to open from your internal network will depend on the other services that you run on this server. I recommend and assume that no other services are running, thus the firewall will be closed to the internal network as well, which gives the following firewall rules. Of course, if your Firewall/VPN server is gateway to the Internet for the local net, there may be some rules to be added.

The Firewall on our OpenVPN server will:

- Allow SSH access from everywhere (remote and local)
- Allow OpenVPN traffic (UDP port 1194, or whatever you opt for)
- Forward traffic between the local network and the remote network (connected by the VPN)

The typical proceeding to set up such rules is as follows:

1. **Add network zones**: Here we define 'what is outside', 'what is inside', and so on.
2. **Define network interfaces and link them to network zones**: We bind the zones 'outside', 'inside', and so on to network cards—real or virtual ones.
3. **Define default policies**: This declares the standard procedure for traffic that is not defined by rules (see next point).
4. **Define firewall rules**: We define exact rules for the traffic based on its IP, port, or protocol.

You can see in the previous screenshot that the four icons in the first line are all that we need, but the Shorewall can do much more. The online help on its homepage is a very concise description of its capabilities.

Troubleshooting Shorewall—editing the configuration files

Shorewall is configured by configuration files that are placed in `/etc/shorewall`. The GUI tool may be the best for the lazy (Windows) administrator, but editing the configuration files is the fastest way to adjust Shorewall behavior. The following table shows the files and the corresponding Webmin modules and functionality of the Shorewall:

Configuration File	Webmin Module	Function
zones	**Network Zones**	Defines the zones (such as external, internal, tunnel) for the firewall
interfaces	**Network Interfaces**	Links zones and network devices
policy	**Default Policies**	How traffic not specified by any firewall rule is treated
rules	**Firewall Rules**	Exact definition of firewall treatment of traffic

If we want to make changes here, we proceed in the same way as we do with Webmin:

- Edit the zones.
- Bind interfaces to zones.
- Define policies for zones.
- Define rules that are different than the policies.

The syntax of these files is simple. The `rules` file created with Webmin looks like this:

```
vpnserver:/etc/shorewall# cat rules
#
#       Accept SSH connections from the local network for
administration
#
SSH/ACCEPT      loc              $FW
SSH/ACCEPT      net              $FW
#
#       Allow Ping from the local network
#
Ping/ACCEPT     loc              $FW
(...)
```

That should allow all SSH traffic from outside and the local net plus pings from inside. The target action is specified in the first column, followed by source zone, destination zone, protocol, and port number. Almost the same system can be used to read the `policy` file:

```
(...)
$FW              net              REJECT         info
$FW              loc              REJECT         info
$FW              all              REJECT         info
#
# Policies for traffic originating from the Internet zone (net)
#
net              $FW              DROP           info
net              loc              DROP           info
net              all              DROP           info
# THE FOLLOWING POLICY MUST BE LAST
all              all              REJECT         info
(...)
```

`$FW` stands for firewall and `all` is a shortcut for all interfaces (like the parameter for **Any** in Webmin). The first column is the source zone, the second shows the destination zone, and the third column, a target action. Optional logging is defined in the fourth column. The policy file shows two new entries at the end, which I have added to allow traffic from the tunnel to access the OpenVPN firewall.

If your `interfaces` file shows this entry at its end, then Webmin and Shorewall have already recognized the tunnel as an internal network interface:

```
#######################################################################
##########
#ZONE    INTERFACE       BROADCAST       OPTIONS
net     eth0            detect          dhcp,tcpflags,norfc1918,routef
ilter,nosmurfs,logmartians
loc     tun0            detect          tcpflags,detectnets,nosmurfs
#LAST LINE -- ADD YOUR ENTRIES BEFORE THIS ONE -- DO NOT REMOVE
```

The first column shows the short name, the second column the real name of the network interface in the system, and optional further columns can define other options.

The last file (which was the first we set up with Webmin) is the one that used to cause problems with some versions of the Shorewall software — the `zones` file:

```
#ZONE   TYPE    OPTIONS             IN                  OUT
#                                   OPTIONS
OPTIONS
fw      firewall
net     ipv4
loc     ipv4
```

This is the content of the file after the editing in Webmin. This configuration works fine with all versions of Shorewall beginning with 3.0.1.

There is also a command `shorewall`, which can be used to start, stop, restart, and check the Shorewall.

Shorewall Command	Function
shorewall check	Checks the Shorewall configuration files
shorewall start	Starts the Shorewall firewall
shorewall stop	Stops the Shorewall firewall
shorewall restart	Stops and then starts the Shorewall firewall
shorewall show	Shows a detailed list of firewall rules, including statistics

> Run a `shorewall check` before you apply any changes to configuration files!

OpenVPN and SuSEfirewall

On SuSE Linux, there is a very sophisticated firewall solution with an administration GUI embedded in YaST. This firewall can also be set up very easily to work with OpenVPN. We will configure the SuSEfirewall for use with the OpenVPN configuration from the beginning of this chapter.

Start YaST on your SuSE Linux system and change to the **Firewall** module, which can be found in **Security and Users.** This is what this looks like on Opensuse 11.1 and 11.0, and it looks pretty similar on SLES 10 and 11:
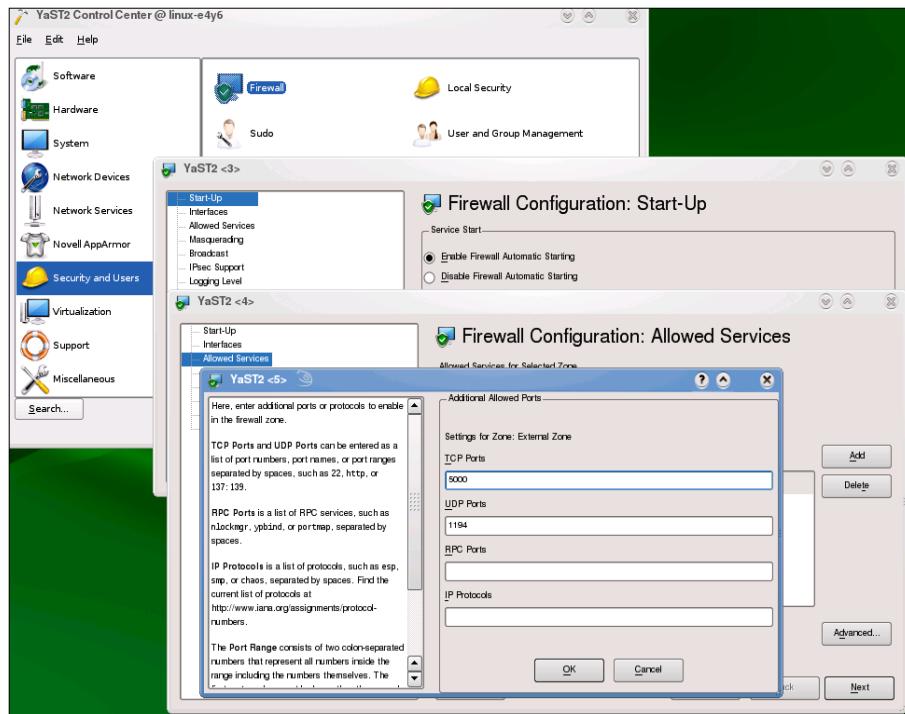
The YaST firewall setup is very straightforward, in the left part of the window we can select the dialogs to be set up for the interfaces, services, and some special features like logging, and so on, and in the right part of the window we enter the parameters and options for these features. The following list will give a step-by-step configuration:

1. Let the SuSEfirewall start at boot time. **Enable Firewall Automatic Start** in Firewall Configuration: **Start-Up**.

2. Change to the entry **Interfaces** in the left part of the window. Look up the MAC addresses of your network cards, double-click on them in the interface list, and select the proper entry from the drop-down menu **Interface Zone**. Here you must define your internal and external devices.

3. Click on the entry **Allowed Services** in the list on the left. Select **External Zone** in the drop-down menu **Allowed Services for Selected Zone** and **SSH** from the **Service to Allow** drop-down menu. Click on the **Add** button to confirm your changes. Now SSH access on the external interface is permitted.

4. Next, click on the button **Advanced** to add our OpenVPN service. The UDP or TCP Port **5000** is not yet part of the standard SuSEfirewall drop-down menu, so we will have to add it using the advanced dialog. Enter **5000** in the field **UDP or TCP Ports** – or **1194** if you decided to keep the standard port.

5. Click on **OK** and on the **Next** button to finish SuSEfirewall setup. Check the settings displayed and click on **Accept**.

6. Now we have the SuSEfirewall configured to deny any access through the external interface except OpenVPN and SSH. What is missing? You may know it by now: forwarding and network traffic from inside the tunnel. These options need to be set up with the **sysconfig Editor** tool of YaST, in the **System** category.

7. Start the YaST module **System | /etc/sysconfig Editor**. The **sysconfig Editor** is a useful tool on SuSE Linux that enables setting of various configuration options that otherwise can only be set on the command line. It consists of a list of variables on the left and fields where our parameters can be entered in the right half of the window.

   We need to enter the following three options:

   ° The OpenVPN interface is an interface that should be treated like the internal network interface.
   ° SuSEfirewall must start routing functionality.
   ° The firewall must route packets between the two networks connected with OpenVPN.

8. Select the entry **Network | Firewall | SuSEfirewall2** in the long list of variables on the left. We will only need to change the values of the following three variables:

Variable	Value
FW_DEV_INT	**eth-id-00:0c:29:88:9c:b0 tunVPN0**
FW_ROUTE	**yes**
FW_FORWARD	**172.16.76.0/24,192.168.250.0/24 192.168.250.0/24,172.16.76.0/24**

9. In my example network the two networks connected are 172.16.76.0/24 and 192.168.250.0/24. The tunnel interface is tunVPN0, and the MAC address of my internal network card is eth-id-00:0c:29:88:9c:b0.

10. Probably the most interesting value in this list is the last line: here we tell the SuSEfirewall that all traffic from 172.16.76.0/24 to 192.168.250.0/24, and from 192.168.250.0/24 to 172.16.76.0/24 shall be allowed.

11. Click on the **Finish** button. You will be asked to confirm a list of the changes you have made. Click on **OK** to commit your changes. Now we must start the YaST firewall module again and restart the SuSEfirewall. Simply start YaST and go to **Security and Users | Firewall** and click on the button **Save settings and restart firewall**.

12. Your SuSEfirewall is up and running.

Routing and firewalls

We have now successfully connected the two networks. Please note that you always need two systems that do routing to connect two networks. If you do not need a firewall on these systems, or if you have problems and do not find the reason for your problems, it may be helpful to enable forwarding without firewall functionality.

Configuring a router without a firewall

The following command activates forwarding of TCP/IP traffic from one network interface to another.

```
opensuse01:~ # echo "1" > /proc/sys/net/ipv4/ip_forward
opensuse01:~ #
```

If your routing setup is correct, then this is absolutely sufficient to make a Linux box a temporary router. Temporary router—because this setting will be gone after a reboot. If you add this command to one of your startup files (or call it from one of the OpenVPN scripts), then your Linux box can act as a router automatically.

iptables—the standard Linux firewall tool

Almost every Linux firewall uses `iptables` as the standard tool. It may be very helpful to know basic features of this tool, not only for debugging, but also to understand what is happening behind firewall GUIs like Shorewall or YaST.

`iptables` is a simple command-line tool that controls the kernel's IP tables. In these tables rules that define how network packets are treated on this system can be stored. As always, the simple commands offer the best solutions when they are combined with an abundance of options. There are a number of options and extensions for `iptables`, so this short description is far from perfect and far from complete. However, I hope that it may help in some cases.

The `iptables` syntax is very simple:

```
iptables <rule command> <chain> <matching extensions><target>
```

A typical rule command is `-A`, which means to 'Add the following rule'. As `iptables` uses different chains (by default, `INPUT`, `FORWARD`, and `OUTPUT`), we must declare the chain that this rule is to be added to. The following table shows three examples:

iptables command	Function
`iptables -A INPUT <rule>`	Adds a rule to the `INPUT` chain, which affects all incoming packets heading for the firewall itself.
`iptables -A OUTPUT <rule>`	Adds a rule to the `FORWARD` chain, which affects all packets that are supposed to be forwarded by the firewall.
`iptables -A FORWARD <rule>`	Adds a rule to the `OUTPUT` chain, which affects all outgoing packets originating from the firewall.

Another typical command is `-P`, which sets the default policy for a chain. This should always be set to `DROP`, because then all packets 'arriving' in this chain are dropped if not specified explicitly by another rule. This is the only way to make sure that only the traffic allowed by us is handled and any unspecified traffic is dropped.

A typical example for this is:

```
opensuse01:~ # iptables -P FORWARD DROP
opensuse01:~ #
```

Then there are the targets of `iptables`. A target can be `DROP`, `REJECT`, or `ACCEPT` (among others), and is invoked by the switch `-j`. Furthermore, so-called 'matching extensions' are like a filter specifying exactly which packet is meant.

Thus a rule such as `iptables -A INPUT <matching extension> -j DROP` means: 'Drop every packet that is headed for my firewall and that matches the `<matching extension>`'.

Matching Extension	Meaning
`-i <interface>`	The incoming interface of the datagram
`-o <interface>`	The outgoing interface of the datagram
`-p <protocol>`	The IP protocol of the datagram
`--dport <destination port>`	The destination port of the datagram
`--sport <source port>`	The source port of the datagram
`-s <source IP>`	The source IP of the sender
`-d <destination IP>`	The destination IP of the recipient

There are many other matching extensions, but these here should be sufficient to understand the basics of `iptables`. Have a look at these lines:

```bash
#!/bin/bash
echo "1" > /proc/sys/net/ipv4/ip_forward

iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -i eth0 -p udp --dport 5000 -j ACCEPT
iptables -A INPUT -i eth0 -j DROP

iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT
iptables -A OUTPUT -o eth0 -p udp --dport 5000 -j ACCEPT
iptables -A OUTPUT -o eth0 -j DROP

iptables -A INPUT -i tun0 -j ACCEPT
iptables -A OUTPUT -o tun0 -j ACCEPT
iptables -A FORWARD -i tun0 -j ACCEPT

iptables -A INPUT -i eth1 -j ACCEPT
iptables -A OUTPUT -o eth1 -j ACCEPT
iptables -A FORWARD -i eth1 -j ACCEPT
```

Do you already understand them? If you do, congratulations, if not, don't worry, it's easy. These lines represent a simple shell script that can be used to start a very simple firewall example. `iptables` is a command-line tool and therefore is simply called from a script with parameters such as the following:

Command	Meaning
`iptables -P INPUT DROP`	Drop all incoming packets that are not specified by any other rule
`iptables -P OUTPUT DROP`	Drop all outgoing packets that are not specified by any other rule
`iptables -P FORWARD DROP`	Do not forward any packets that are not specified by any other rule
`iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT`	Accept TCP connections for port `22` coming in on network interface `eth0`
`iptables -A INPUT -i eth0 -p udp --dport 5000 -j ACCEPT`	Accept UDP connections for port `5000` coming in on network interface `eth0`
`iptables -A INPUT -i eth0 -j DROP`	Drop everything (else) incoming on interface `eth0`
`iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT`	Accept outgoing TCP connections for port `22` going out on network interface `eth0`

Command	Meaning
`iptables -A OUTPUT -o eth0 -p udp --dport 5000 -j ACCEPT`	Accept outgoing UDP connections for port `5000` going out on network interface `eth0`
`iptables -A OUTPUT -o eth0 -j DROP`	Drop everything (else) going out on interface `eth0`
`iptables -A INPUT -i tun0 -j ACCEPT`	Accept traffic coming from the tunnel headed for the firewall
`iptables -A OUTPUT -o tun0 -j ACCEPT`	Accept traffic headed for the tunnel
`iptables -A FORWARD -i tun0 -j ACCEPT`	Accept traffic to be forwarded coming from the tunnel
`iptables -A INPUT -i eth1 -j ACCEPT`	Allow incoming traffic from the local network interface `eth1`
`iptables -A OUTPUT -o eth1 -j ACCEPT`	Allow outgoing traffic to the local network interface `eth1`
`iptables -A FORWARD -i eth1 -j ACCEPT`	Accept traffic to be forwarded coming from the local network `eth1`

In a nutshell:

- `eth0` is the external interface, where all traffic except SSH and OpenVPN will be dropped
- `tun1` is the tunnel interface, where forwarding to `eth1` is allowed
- `eth1` is the local network, where forwarding into the tunnel is allowed

If you need more information, the manual page of `iptables` is the best place to look for help.

The OpenVPN software package contains a sample script that could be adapted for firewall purposes. The script can be found in `/usr/share/doc/openvpn/examples/sample-config-files/firewall.sh` and can be adapted to your needs. However, this script makes use of some special features of `iptables` that would go beyond the scope of this book.

> Every Linux system (since kernel 2.4) uses `iptables` to set up the rules for its firewall.

Configuring the Windows Firewall for OpenVPN

Microsoft Windows XP with installed service pack 2 offers firewall software too. In the control panel there is an icon called **Windows Firewall**. Double-click on this icon.
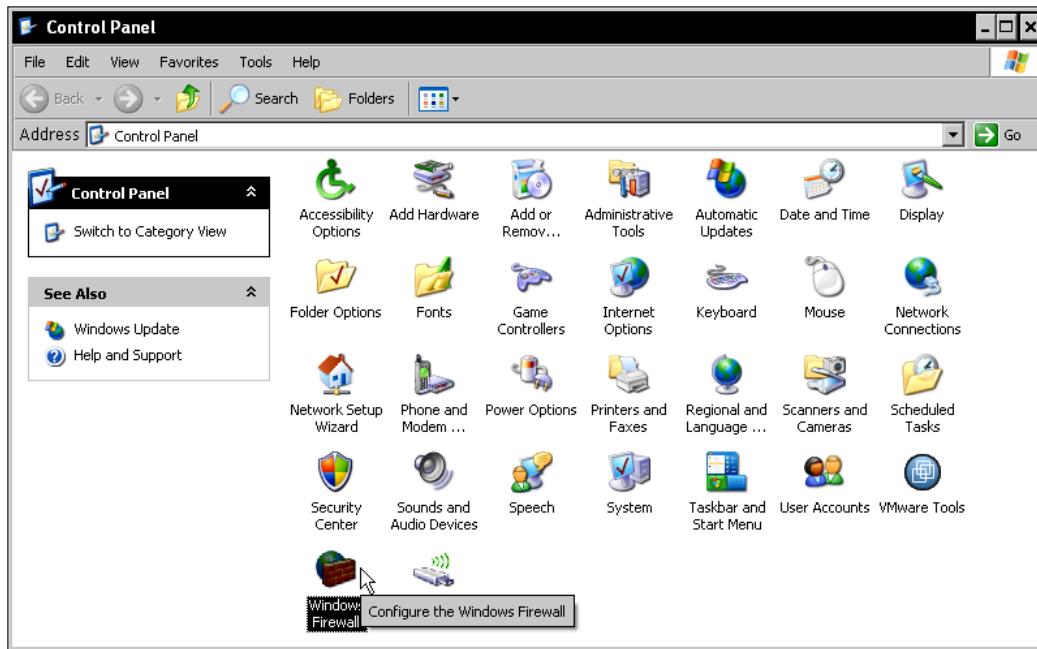
The **Windows Firewall** is activated as default, blocking all connections from outside to the local host. The Windows machine can connect to any host, even OpenVPN as a client can be run without any changes. If you want to connect to this Windows machine with OpenVPN, then some changes have to be made. The **Windows Firewall** offers the possibility to switch off the firewall service completely (which should only be done for testing purposes) and as an alternative to add exceptions to the firewall behavior. This is what we will have a look at later.
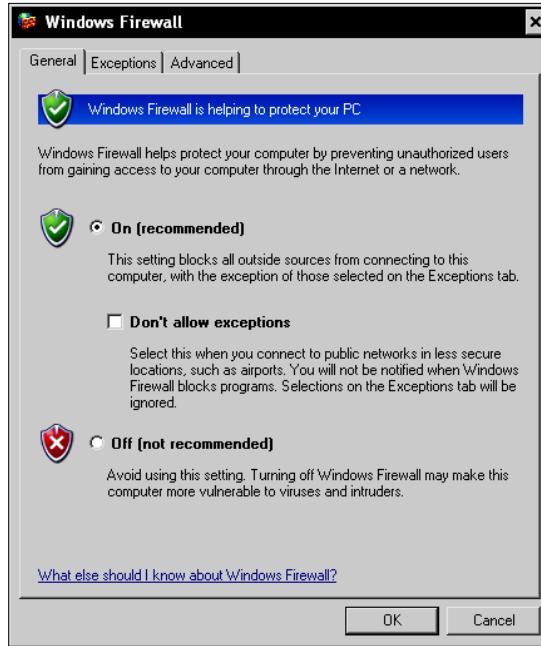
However, if we want to start an OpenVPN server process that binds to a local port and expects other machines to connect, then the **Windows Firewall** causes a security alert with a dialog box like the one that follows. This is probably the easiest way to activate OpenVPN in the **Windows Firewall**:

Click on the **Unblock** button.

As soon as the OpenVPN process is started, another (small) pop-up window will appear and indicate that the OpenVPN process is ready to accept connections.
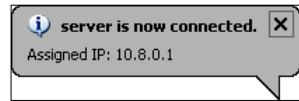
What happened when we clicked **Unblock**? The **Windows Firewall** has automatically created a rule (or so-called exception) that allows incoming connections to the OpenVPN process. Let's click on the **Exceptions** tab in the **Windows Firewall** dialog:

Here is a new rule that was generated when OpenVPN tried to open the port. Click on the **Edit** button, if you want to have a closer look at this rule. With the **Add Port** button we can add any firewall rule to the **Windows Firewall** setup.

Click on **Add Port**. The following dialog shows that we have three options to set up a rule:
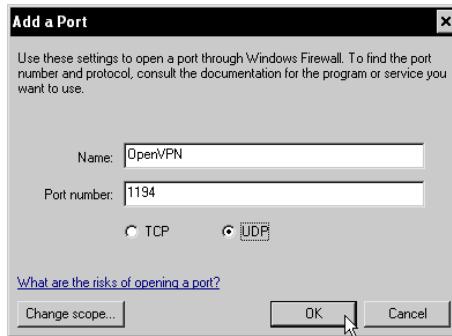
We can enter:

- **Name** for the rule—**Unblock** usually takes the name of the program
- **Port number**
- Protocol (**UDP** or **TCP**) for the connection

In this example the standard port of OpenVPN is entered—port **1194** and protocol **UDP**.

More options can be declared if we click on the **Change Scope** button. Another window pops up, where we can define the source of the connection that is to be allowed.

Three possibilities are offered:

- No restrictions—**Any computer (including those on the Internet)**
- The local subnet—**My network (subnet) only**
- A **Custom list** of IP addresses that are allowed to connect to this process
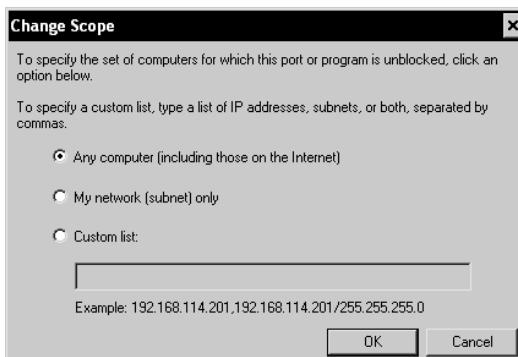
> On Microsoft Windows XP with service pack 2 the firewall can easily be configured with the control panel module **Windows Firewall**. In the **Exceptions** tab, we can enter ports, protocols, and sources for connections.

Summary

In this chapter, we have set up a secure OpenVPN connection between two partners based on certificates and using strong encryption plus some non-standard security features such as sophisticated authentication plugins. In the next step we configured a Debian system with a firewall, which was Shorewall, which offers a nice GUI together with Webmin. A short look at the configuration files of the **Shorewall Firewall** and possible troubleshooting hints followed before we proceeded with the SuSEfirewall of OpenSuSE. After that we configured two different firewall networks that could connect to each other through the secure OpenVPN tunnel. We looked at `iptables`, and finally learned how to configure the **Windows Firewall** on Microsoft Windows XP.

11
Advanced Certificate Management

In this chapter we will learn how to install and use xca, an advanced tool for Windows with which we can easily manage our X509 certificates. We will also learn how to use its Linux alternative, TinyCA2, which can even manage multiple certificate authorities. Both tools can be used to generate certificate revocation lists that are used to block unwanted connections by formerly authorized clients such as with stolen notebooks. A brief selection of other free PKI tools that are available will close this chapter.

Certificate management and security

I think it's quite obvious that a computer that is used to sign certificates and keys, granting or restricting access to a company's network will deserve special attention from everybody interested in accessing this network. My recommendation for a **certificate server** is to disconnect it from the network. Transfer keys and certificates with USB sticks or other non-network media.

This advice has been published before very frequently because it is simply reasonable and true.

However, anybody who really does separate a certificate server computer from the local net and does not control the network of a secret service like a bank or similar infrastructure may send me an email. Most people simply wouldn't. In reality, certificate servers are merely programs running as a background job or as an application that is run by a non-privileged user. They say that there are even Windows machines out there that perform certificate management, and that's why I chose to show two tools in this chapter, one for Windows, and one for Linux machines. My favorites are xca (for Windows) and TinyCA2 (for Linux).

Installing xca

Installing xca is easy. Just download the `.exe` file by searching for it on `http://www.sourceforge.net`. You will find a `.exe` file of about 2MB. As of 2007, the latest version was 0.6.x. Download it and double-click on it to start installation.
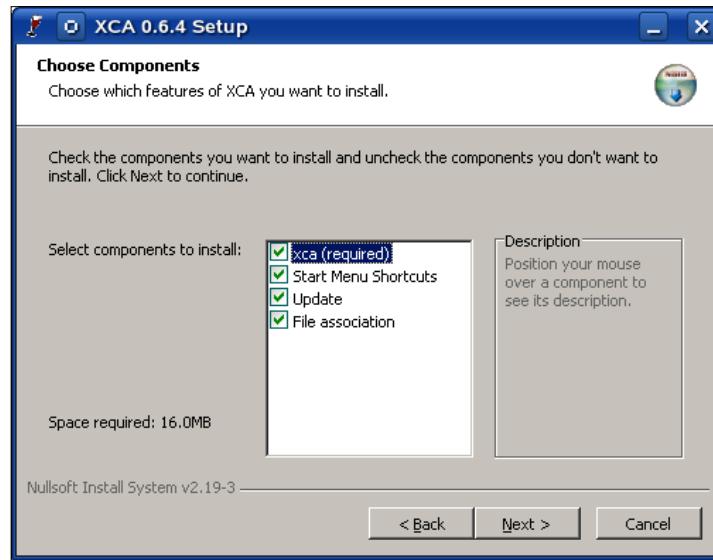
In the first step you are asked to accept the BSD-style license. If you are unsure, read it carefully and then click on the agreement, if it suits your needs. You will then be asked to select the components to be installed. Click on **Next** again and xca will ask you which path to install in.

Using xca

Once you've chosen a directory, xca will install in a few seconds and will automatically start. Later you can select the main menu entry **Start | Programs | xca | xca** to start it. Upon first start, xca may inform you that its data directory (`C:\Documents and Settings\USERNAME\Application Data\xca`) is created. Click on **OK** to close this window and xca is started. As a first step when running xca, we need to create a database where xca stores metadata on the certificates.

Creating a database

Select **New DataBase** from the **File** menu of xca.
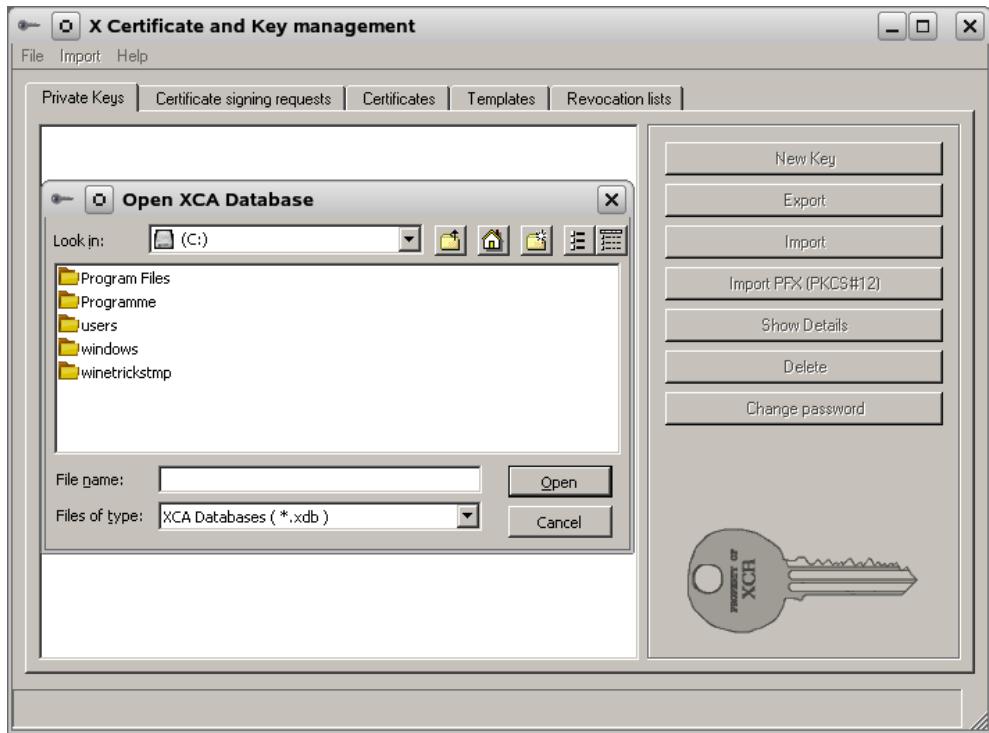
In the **Open XCA Database** dialog, we can select an existing database. However, as we have started xca for the first time, there will not be a database and we will have to create one. For this purpose, we can simply enter a new filename in the field **File name** using the filename extension `.xdb`. This is very important because xca may not recognize the database correctly later if the extension is missing. Click on the **Save** button to confirm the creation of the database.

Now we must define a password for this database. This password will be needed to encrypt the keys in the database file. If you transfer this database to a different machine and want to reopen it, you will have to enter it again.

Importing a CA certificate

The main window of xca offers the following five tabs:

- **RSA Keys**
- **Certificate signing requests**
- **Certificates**
- **Templates**
- **Revocation lists**

Except for the **Templates** tab, we will explain and use all the other tabs.

Let's first import the CA certificate that we had earlier created with `easy-rsa`. Change to the **Certificates** tab and right-click to open the context menu. Select the entry **Import** to have xca import a certificate authority.
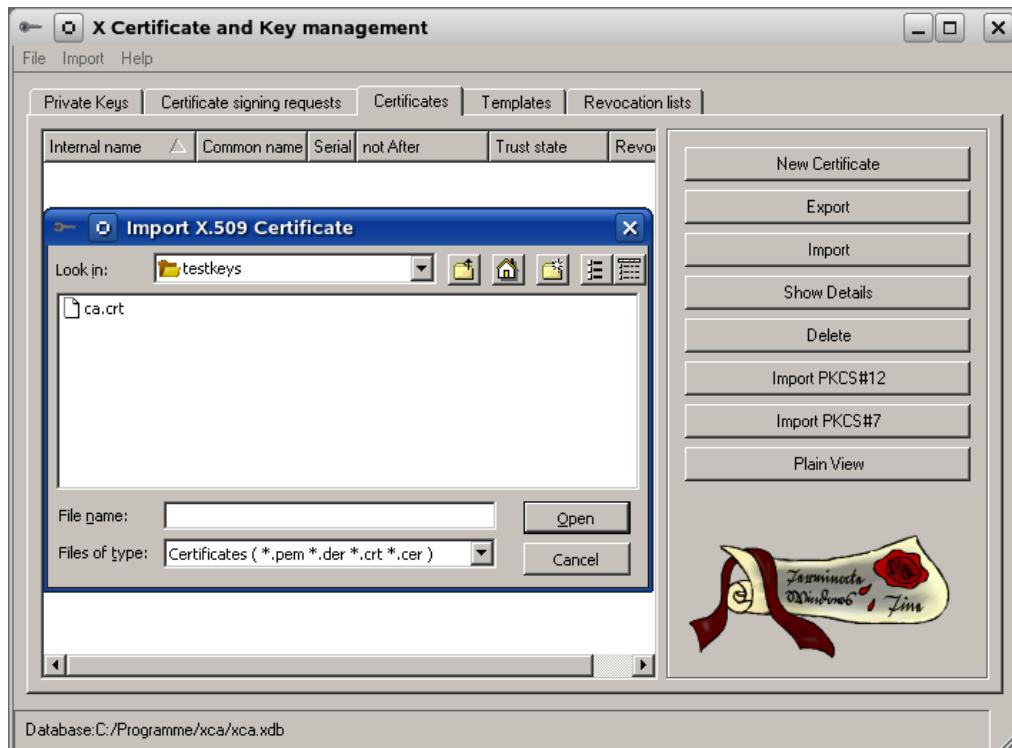
The **Import X.509 Certificate** window is displayed. Change to the directory containing your xca keys. According to our examples this is `C:/Program Files/ OpenVPN/easy-rsa/keys`. Select the `ca.crt` file and confirm by clicking on the **Open** button.

We see a new certificate in our list that is marked with a red question mark. This signifies the fact that the certificate is still unknown and untrustworthy. Right-click on the certificate and select the entry **Trust** to make this certificate a trusted one.
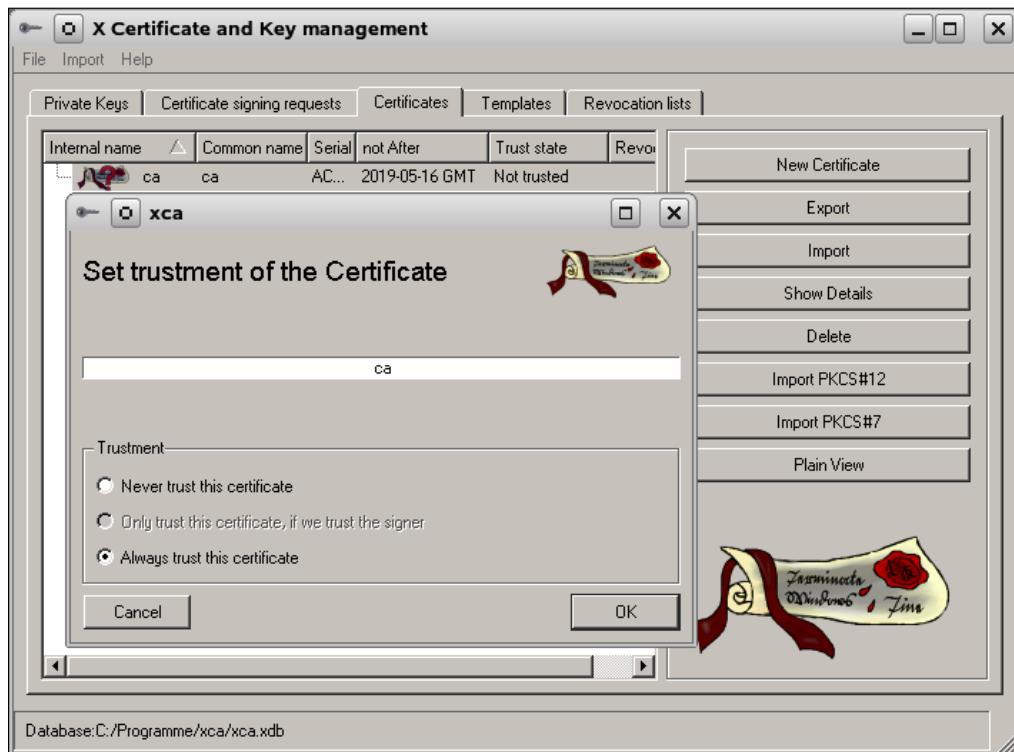
Another pop-up window is displayed, where we have to select **Always trust this certificate** and click on **OK**.

Before we can sign keys and client certificates using this CA certificate, we have to import the CA key. Switch to the **RSA Keys** tab and open the context menu with a right-click of the mouse.
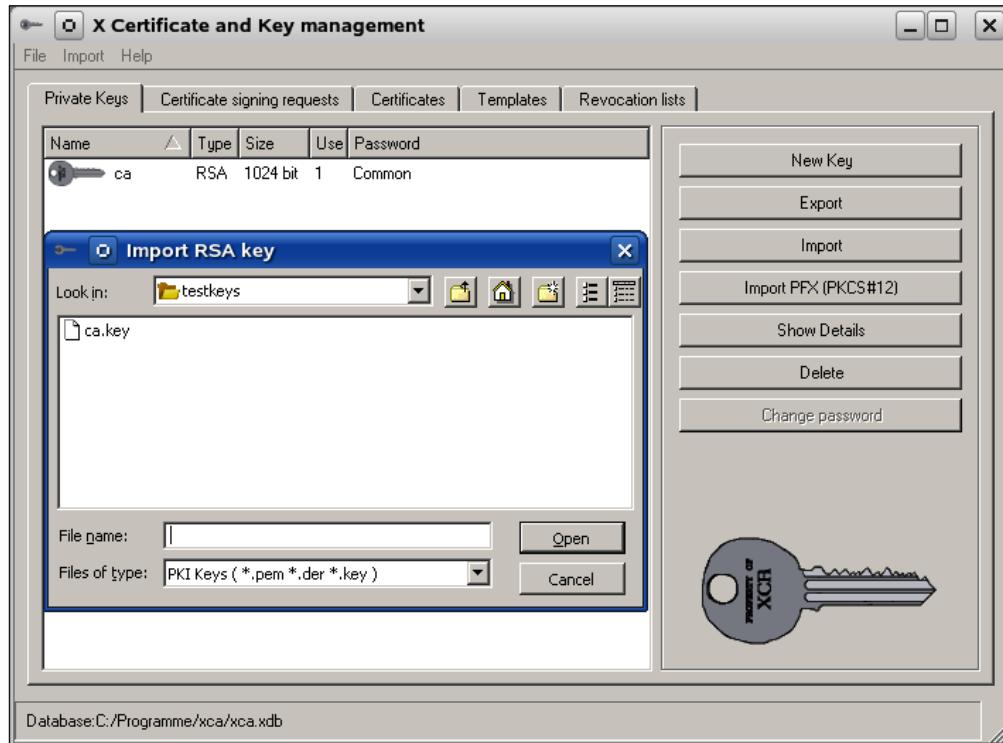
Select **Import** from this list and choose the `ca.key` in the **Import RSA key** dialog. We now see the key that is imported to xca. It is displayed with a key symbol as shown in the previous screenshot.

Creating and signing a new server/client certificate

Now let's create a new certificate for a VPN client. Switch to the **Certificates** tab and select the **New Certificate** entry from the context menu, or click on the button of the same name. In xca versions up to 0.4, a **Certificate Wizard** would be started. Recent versions offer a dialog with six register pages of options and parameters for you.

In the following dialog, you can choose from templates for the certificate that is to be created (you can manage them with the **Template** tab in the main window of xca), but the important selection you have to make is choosing the certificate that you want to use for signing. Select the certificate that you imported in the drop-down menu **Use this Certificate for signing**.

In the **Template** section of this dialog you can choose if your certificate is to be used by a server or a client. In the **Subject** dialog, there is room for a common name, organizational data, generating a key, and more. Click on **Generate a new Key** to have the key calculated. As you can see, without templates, there is a lot of work to be done with Ca-admin when generating new certificates.

Here we can enter the data that `easy-rsa` has also asked us for. The xca templates can make this a lot more comfortable. You know by now that a very important part of this data is the field **commonName**, which can later be used to distinguish VPN clients. You should choose a name that would be useful to distinguish your VPN clients in this field. Click on **Extensions** to enter a range of dates for the validity of the certificate.
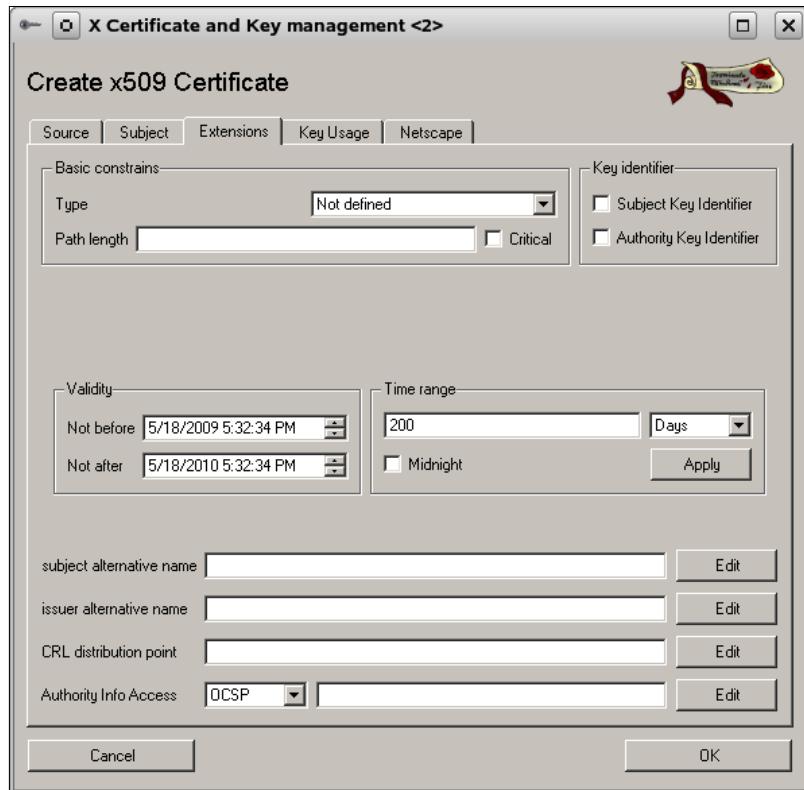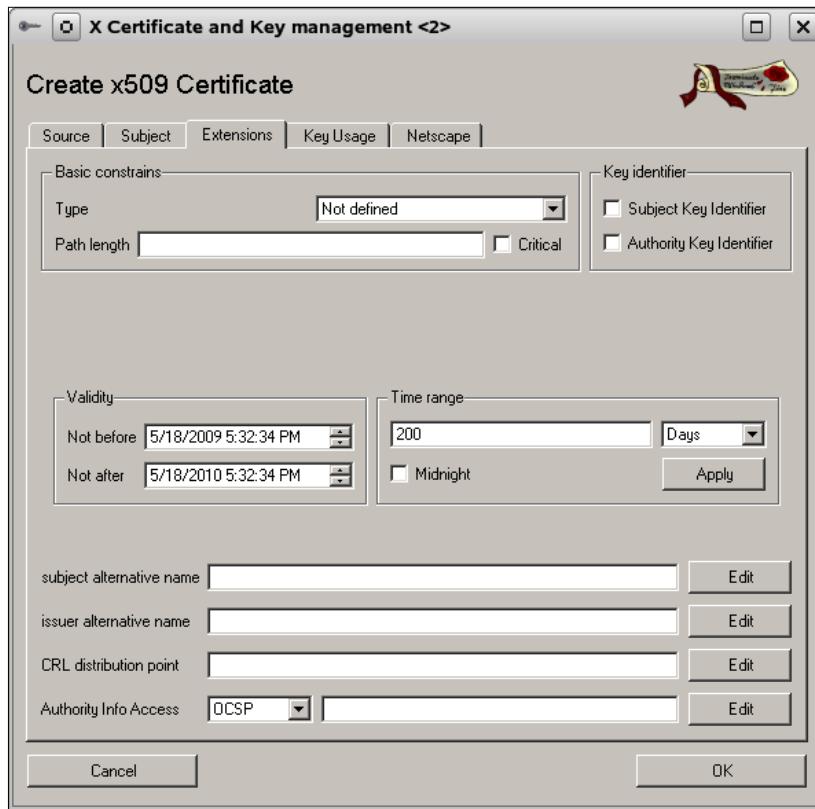
There is a new entry below our CA certificate, with the name of the certificate we created, and statistical data. In the **RSA Keys** section we can find a key for this certificate. The context menus of the **RSA Keys** and **Certificates** sections have entries that allow us to export the keys and certificates to directories from which we can copy them to our VPN servers and clients.

Of course, we have to repeat these steps for every new certificate that we want to create. Again, don't forget to use distinguishing names. That's all! Isn't that easy?

> **PKI** management with xca is easy. Import the CA's CA certificate and declare it as trusted. Then import the CA key and start the certificate generation. Don't forget to use the right CA certificate and an appropriate common name for the certificate. Again, use the context menus to export the keys and certificates.

Revoking certificates with xca

The context menu of a certificate in the **Certificates** tab of xca offers an entry that is called **Revoke**. By clicking on this entry a certificate is immediately made invalid. If we create a revoke list and put this list on our VPN server, then with this list (and a suitable configuration), a client trying to connect with this certificate will not be granted access.

Select a certificate that you have created in xca and click on the entry **Revoke** in its right-click context menu.
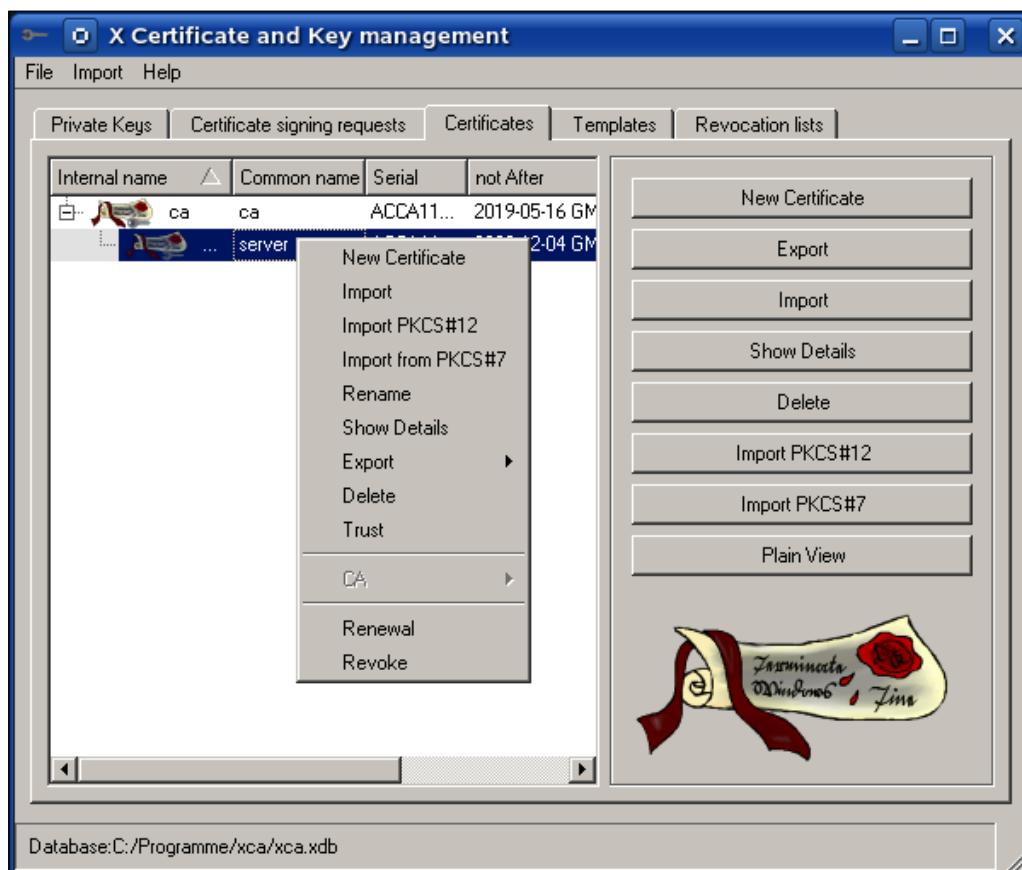
Then right-click on the CA certificate and select the entry **CA | Generate CRL** to create a **Certificate Revocation List** (**CRL**).
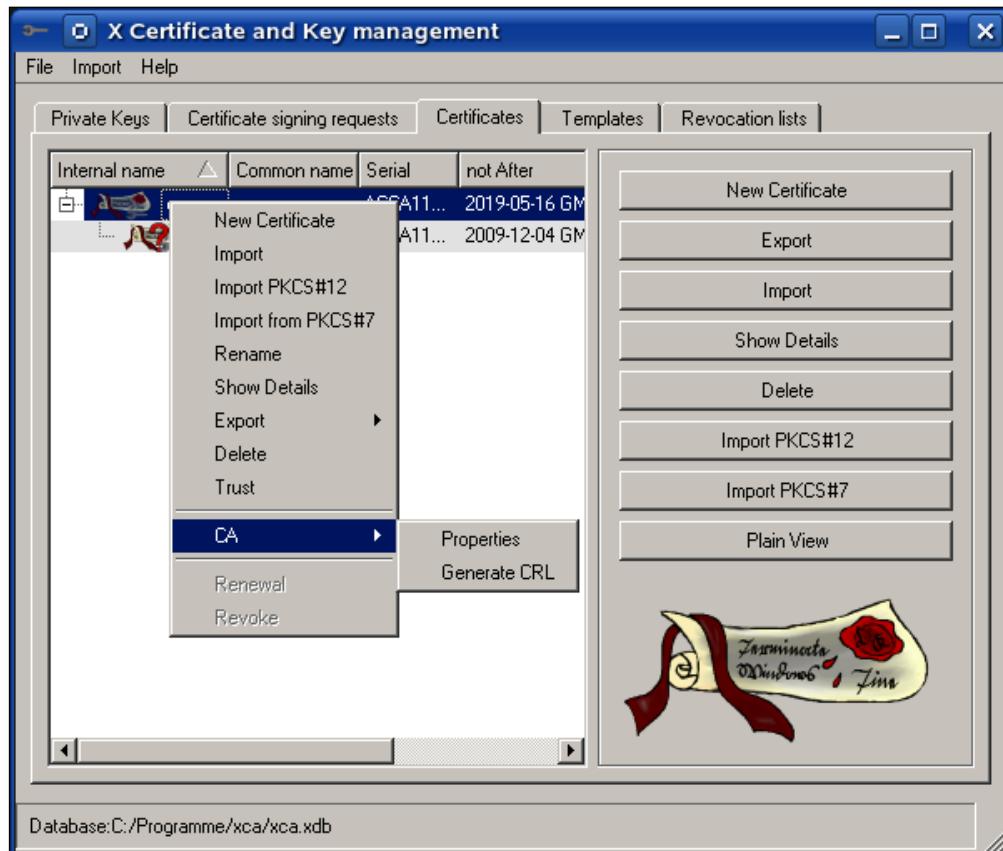
Now switch to the **Revocation lists** tab and double-click on the newly generated revocation list to show the details. Close this dialog by clicking on **OK**. We can now export this list to the VPN server using the context menu entry **Export | PEM,** by right-clicking on the **Revocation List** button. Copy this file to the VPN server and add an appropriate entry such as `crl-verify <filename>` to your configuration.

Create a few certificates and keys, export them to your VPN servers and clients, and revoke them—some hours of training is very helpful to get a good feeling here. Especially when combined with a high level of verbosity in the OpenVPN configuration will help you learn a lot about certificates.

Using TinyCA2 to manage certificates

TinyCA2 is a very handy tool to deal with certificate management. It provides extended functions and the possibility to influence the behavior of OpenSSL itself. TinyCA2 is available for OpenSuSE in online repositories. Other distributions must look on `http://tinyca.sm-zone.net/` for appropriate packages or source code. On OpenSuSE, TinyCA2 can be easily installed using YaST. I also read about a MAC port on `http://tinyca2.darwinports.com/`, so there should be a version for almost every Unix/Linux system, even though the tool has not received much development work during the last few years.

TinyCA2 can be used to create a CA, to import and export CAs, certificates, keys, and revocation lists. It can manage several CAs and will offer the choice of which CA to load on startup, if several CAs are configured.
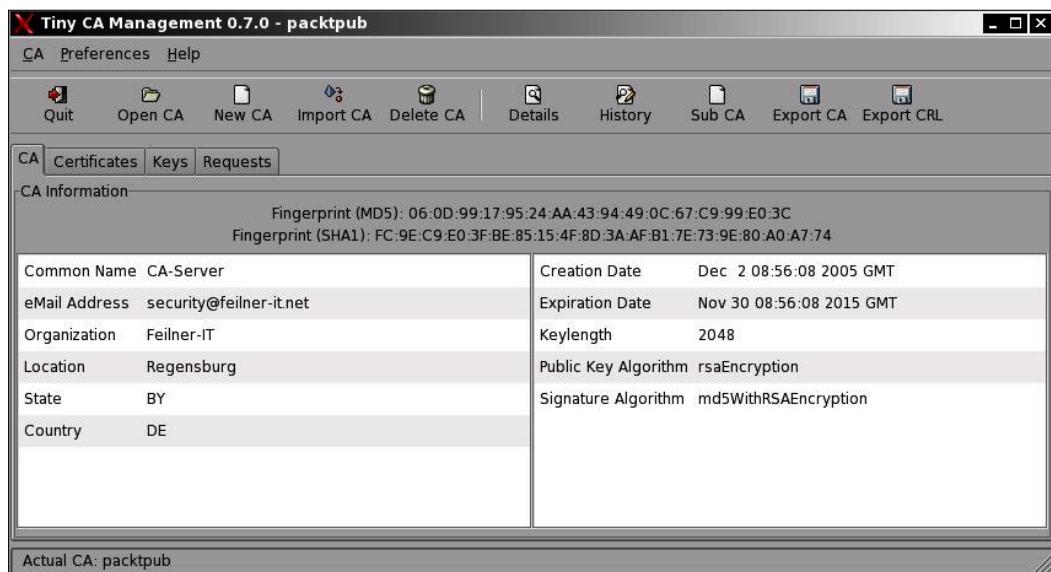
Importing our CA

After installation, start TinyCA2 from SuSE's main menu. Select **Utilities | Security | tool to manage a Certificate Authority (TinyCA2)**. TinyCA2 starts and displays an empty window. The icons in the tool bar offer several options.

- **Open CA**: Open an existing CA—that is a CA that has previously been imported to TinyCA2

- **New CA**: Create a completely new CA

- **Import CA**: Import a CA (like those we created with `easy-rsa`) into TinyCA2

Click on the **Import CA** icon to import a previously created CA. The **Import CA** dialog is displayed as follows:
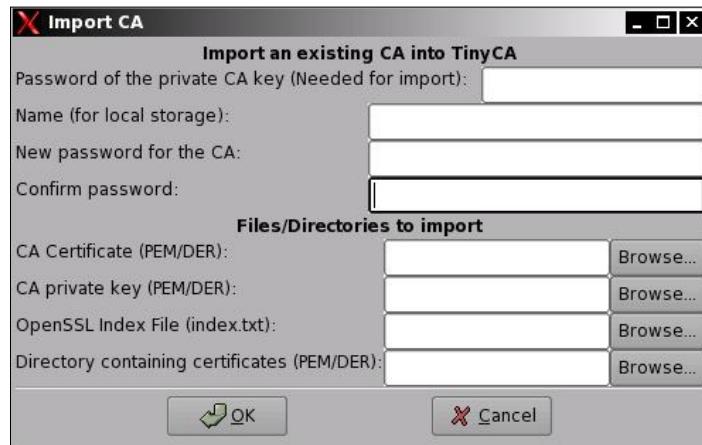
Here we must enter the password, location, and filename of the CA certificate and key file. TinyCA2 offers extended options such as changing the password for the CA right here or importing the SSL index file. However, entering password, certificate file, and key are enough to import the CA. Click on **OK** to start the import.

Using TinyCA2 for CA administration

If you have several CAs to administer, TinyCA2 will present the following window on startup. This window is also displayed when you select the **Open CA** icon.

Once you have loaded, created, or imported a CA, the main window of TinyCA2 will be much richer with icons, menus, and features. TinyCA2 offers a lot of details, information boxes, and history functions that let us manage our certificates and keys in a very reliable and controllable way.

Like xca, TinyCA2 also presents tabs in its main window, and a lot of work is done by selecting entries from context menus. The **Open CA** tab shows some information on the CA itself, and the **Certificates** and **Keys** tabs list the existing certificates and keys for this CA. The **Requests** tab is used to create and sign new certificates and keys.

Creating new certificates and keys

If we want to create and sign a new certificate for our CA with TinyCA2, we have to create a key signing request first. Change to the **Requests** tab, right-click, and select **New Request** from the context menu. The following window appears:

I don't think that you need an explanation for the fields in this window. They are the same as in the information that we had provided for `easy-rsa` and xca on certificate generation. However, we have to make sure that an appropriate key size is selected and that the **Common Name** is distinguishable. Click on **OK** to create the request.
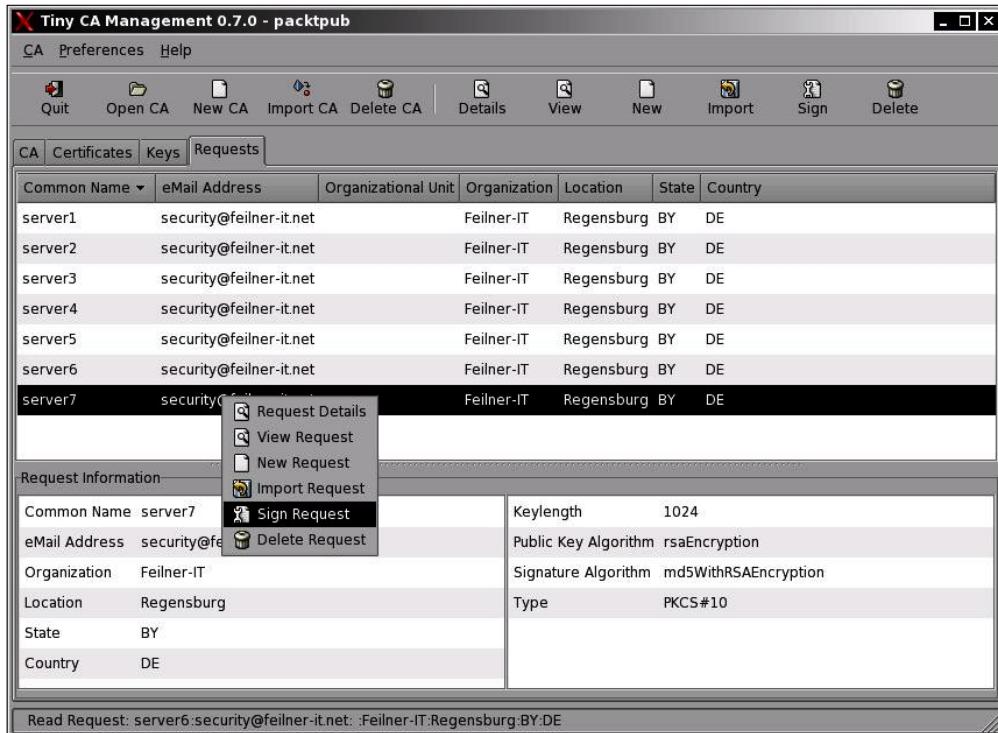
In the previous example we had seen a CA with many certificates and requests. Now, right-click on your newly generated request and select the menu entry **Sign Request** to sign it using the active CA's certificate. Another small menu appears, asking you whether the request will be signed as a server or a client. This is for example purposes that we have talked about on the xca pages. For a TLS server's certificate, choose **Sign Request (Server)**. For all clients choose **Sign Request (Client)**. Consider the following screenshot:
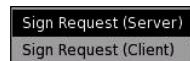
Now we are asked to enter the CA's password to sign the request. Enter your password and check again if the validity is suitable for your purposes, and click on **OK** to confirm. After a few seconds of calculations, your machine will tell you that the certificate has successfully been created. Now switch to the **Keys** section. There is a new entry for the newly created key/certificate pair, and there is also a new entry in the list of the available certificates.

Exporting keys and certificates with TinyCA2
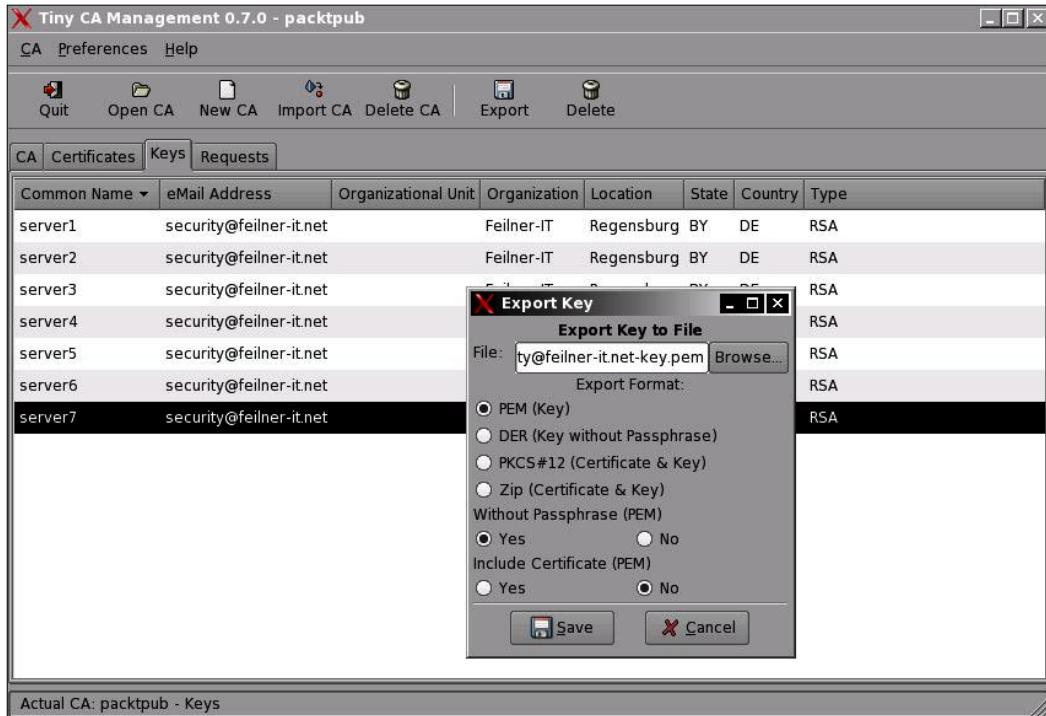
With TinyCA2 we can export the CA, the client certificate, and key to a local file. TinyCA2 recognizes several file formats for the key/certificate pairs. In the previous screenshot you can see the default `.pem` key files. Please note that if you do not want to enter a passphrase every time your OpenVPN tunnel is started, you must activate the button **Without Passphrase (PEM) | Yes**. Otherwise, your key is password-protected, which may be considered as an extra level of security.

Enter a filename or select a directory by clicking on the button **Browse**, and then click on the button **Save**. Repeat these steps for the client certificate (use the standard **PEM Certificate**) and the CA certificate (by clicking on the icon **Export CA** in the toolbar).

Revoking certificates with TinyCA2

Creating and exporting a CRL with TinyCA2 is very easy. From the **Certificate** tab, right-click on the certificate that you want to revoke. You are prompted for the CA password and you are given the option to enter a reason for revocation.

Enter the **CA Password**, select a **Revocation Reason**, and click on **OK** to revoke the selected certificate. Now switch to the **CA** tab and click on the **Export CRL** icon in the toolbar. Again, you have to enter the CA's password and a validity date for this CRL. Enter a filename and click on the **Save** button to export the CRL.
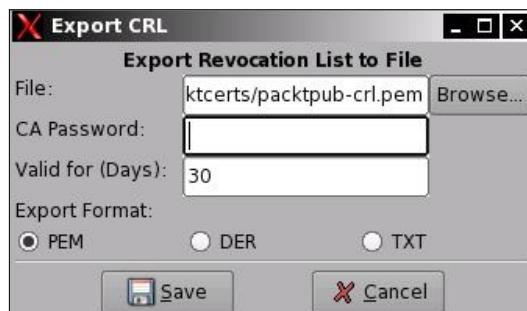
Other tools worth mentioning

There are of course a lot of other tools available for PKI management, especially for X509 certificates.

One of them is OpenSSL itself. If you're not afraid of long command lines, try either `http://www.openssl.org` or the recommended German tutorial at `http://www.online-tutorials.net/security/openssl-tutorial/tutorials-t-69-207.html#beispiel-openvpn`.

SuSE's YaST has a special module for PKI management. Simply install the package `yast2-ca-management` and restart YaST. You'll find it in the section **Security**.

For other distributions, Webmin provides quite suitable modules for PKI management, like the mentioned `OpenVPN + CA` tool.

OpenCA PKI Research Labs are creating OpenCA, a browser based PKI management suite that seems very promising (`https://www.openca.org`).

IDX-PKI is now integrated in a greater structure named OpenTrust and offers an abundance of options for larger companies (`http://www.opentrust.com/content/view/119/111`)

The small Windows tool **My Certificate Wizard** (`http://mycert.sandbox.cz`) is for networks, where users create certificate requests on their own, send them to the admin, and have them signed. This may be of special interest in well-organized high-security environments.

Summary

We have created, imported, and exported CA certificates, client and server certificates and keys, in addition to revocation lists using the tools xca and TinyCA2. We have seen that there are many features that TinyCA2 offers that are neither in the scope of `easy-rsa` nor available in xca. This is the reason why TinyCA2 is my favorite certificate management tool. A small list of other available tools closes the chapter. However, all these tools use only the 'toolbox' OpenSSL. If you want to read more and become a certificate professional, 'man OpenSSL', then the website `http://www.openssl.org` is the place to go.

12
OpenVPN GUI Tools

In this chapter, a lot of images will show several standard GUI tools for OpenVPN. The best one is the Webmin OpenVPN module, probably the best tool for administration of an OpenVPN server. The situation of the GUI clients looks worse. Even though there are at least three interesting projects, none of them seem to work flawlessly at the moment. Most still contain severe bugs, but I assume that the community will make them work in a short time. Perhaps, at the time of reading, everything will be working properly.

We'll start with Webmin's extraordinary frontend that we looked at briefly in the chapter that dealt with security.

KVpnc is next, and the network manager plugin rounds out the chapter.

OpenVPN server administration: Webmin's OpenVPN plugin

In Chapter 10, we introduced Webmin and the Shoreline/Shorewall firewall for securing OpenVPN servers. We added a user that only has permissions to adapt OpenVPN and firewall rules. That is not enough to control the server, but still more secure than the root that is administrating it over a web connection.

The three main blocks that are available in this module are as follows:

- **Certification Authority List**
- **VPN List**
- **Active Connection**

As this screenshot shows, you can even use Webmin to create your own OpenVPN CA. Few admins will need the module configuration (which can be reached by clicking on the text link at the top left)

This is what the OpenVPN plugin looks like:

The VPN list shows the example tunnel that we have created in the course of this book:

Note the links to **Logs**, **Client**, **Status**, **Remove**, and **Actions**. Have a look at the link **Client**, it enables you to get a suitable client configuration at any time—if your paths to Tar, Gzip, and your Webmin OpenVPN configuration is correct.

A click on the name of the configuration (in the previous example, this is still **sample**) allows detailed configuration:

I think this powerful WebGUI is probably the biggest utility for an OpenVPN admin. It allows full control of the configuration, all parameters, starting and stopping of all tunnels, and access to the logfiles. With the client config export, no USB-stick is needed—of course except for certificates.

Client GUIs for Linux

Now let's have a look at how to import and use this configuration into our clients. Today the number one standard tool on any Linux system is the network manager that provides a cool systray applet for both KDE and GNOME. NetworkManager should be installed out of the box on any recent Linux system, and there are suitable packages for OpenVPN. On SUSE 11.1 and similar platforms, they are called `NetworkManager-openvpn`, `NetworkManager-openvpn-kde`, and `NetworkManager-openvpn-gnome`. On Ubuntu, they are called `network-manager-openvpn`, `network-manager-openvpn-gnome`, and `network-manager-openvpn-kde`.

KVpnc

KVpnc seems to be one of the most promising VPN tools for many kinds of VPNs, if you are running KDE. The only problem (apart from some reported instabilities and bugs) that I can see is that it has to be run with root privileges. Whoever wants to use it permanently, for example, on a field worker's laptop, has to be aware of the risks of giving away the root password to a normal user or the risks of a GUI application running as root. I prefer not to use these types of applications.

However, the functions it offers are abundant. Once it reaches a really stable version, it may be a very handy application.

Here are some screenshots and a brief explanation of how it works. After calling `kvpnc` on Ubuntu 9.04 (which has version 0.9 in its repositories), you are asked to enter the root password, then the following window appears:
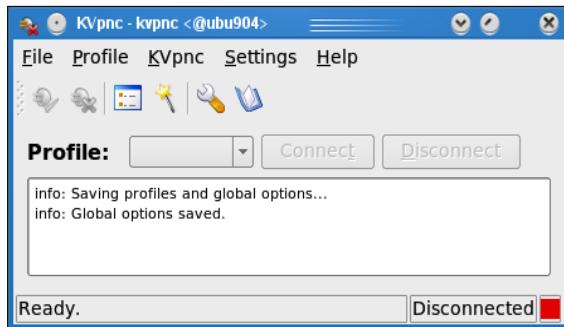
Under the menu entry `Settings - Configure kvpnc`, you'll find an abundance of switches and parameters to enter for several kinds of VPNs, as shown in the following screenshot:
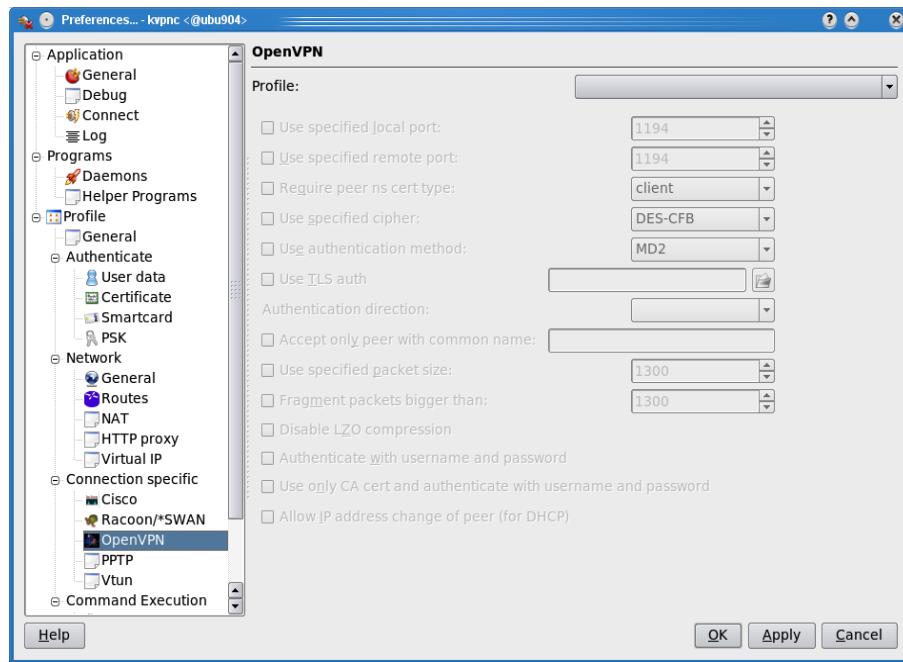
In order to get started, a wizard helps you to generate a new VPN profile, for example, for OpenVPN.

If this is not enough, then kVpnc can do more for you. You can generate a static key for OpenVPN (`Kvpnc – Generate OpenVPN Key`), import config files and certificates, and much more. However, KVpnc seems to ignore some parameters that are set in the imported files, so you have to go and check for yourself if it meets your needs. Good luck!

GAdmin-OpenVPN-Client

Compared to KVpnc, the gadmin-openvpn-client is rather small and only has a few functions. Although it exists only in version 0.0.3, it is already in the standard Ubuntu repositories and seems to work. Furthermore, it may be a better solution for GNOME users than the KDE tool KVpnc.
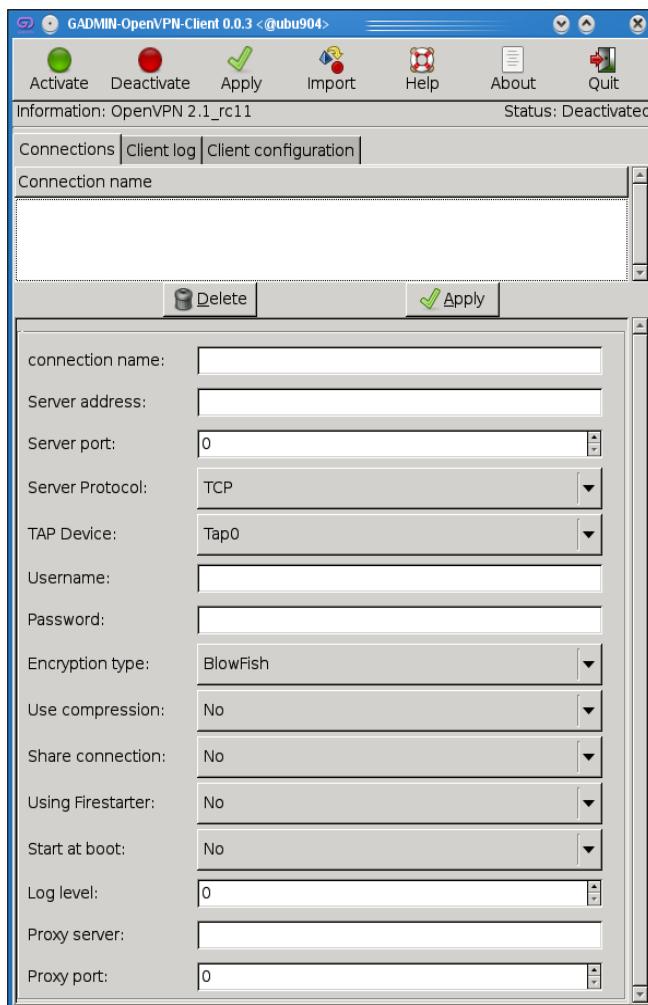
As you can see, you can enter or import your configuration, and activate or stop a tunnel. GAdmin-OpenVPN also allows the monitoring of logfiles or current connections, but it also has to be run with root privileges.

NetworkManager

The OpenVPN NetworkManager applet is available for for both KDE and GNOME. You will need, however, to test it yourself. There are still many instabilities and problems being reported on desktops' bug tracking systems. Both plugins are included in all standard distributions.

This is how you add a new VPN tunnel connection to GNOME's NetworkManager applet on Ubuntu 9.04:

1. Right-click on the NetworkManager symbol in your system tray
2. Choose **Edit Connections** from the context menu
3. Select the register **VPN** in the window which appears
4. In the following dialog, select **OpenVPN** as VPN type

This is what you should see now:

Now perform the following:

- Click on **Create**, and then the **Advanced...** button

Consider the following screenshot:
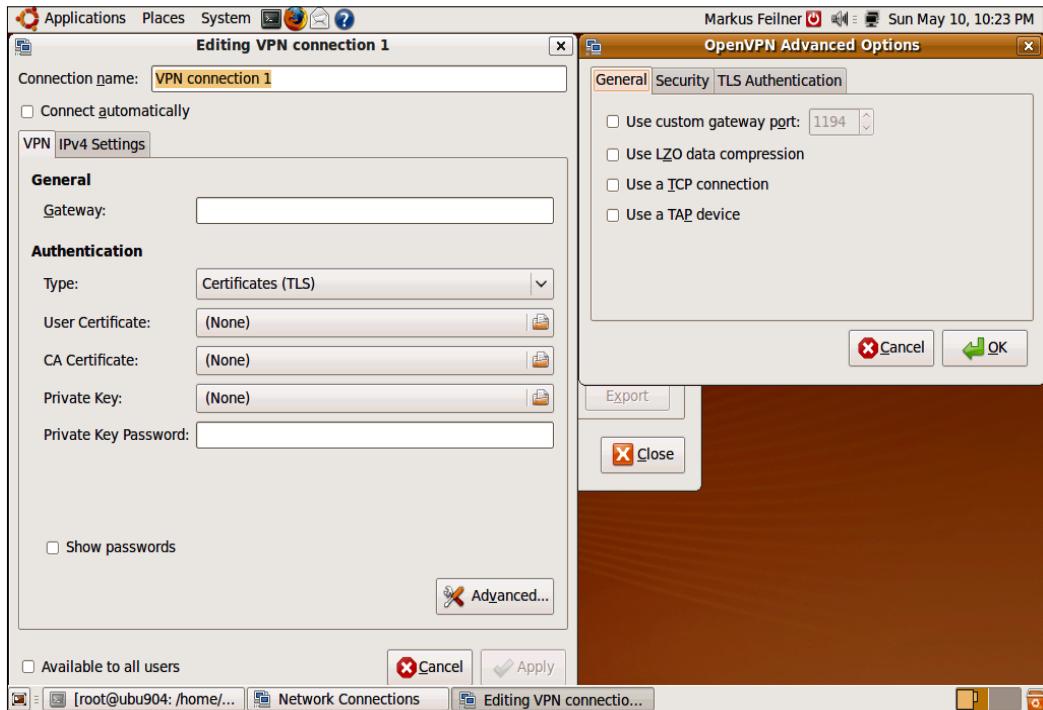
As you can see, the network manager applets offer all the basic functions, with the exception a few key elements. You can't import or export configurations and unfortunately the applet won't work with symmetric static keys. Bugs have been filed, and it would be excellent to see this solution work, as it offers the best means of integration of the two big desktop applications, KDE and GNOME.

Summary

In this chapter, we have discussed some typical GUI-tools for OpenVPN. Beginning with the Webmin module for the server, we also learnt about KVpnc, GAdmin-OpenVPN, and the NetworkManager plugins for OpenVPN.

13
Advanced OpenVPN Configuration

In this chapter, we will deal with several examples of advanced OpenVPN configurations such as:

- Tunneling through a proxy server like **squid**
- Scripting OpenVPN—an overview
- Using a server configuration with specific per-client configurations that are pushed to clients based on their certificates
- Pushing routing commands to clients
- Pushing and setting the default route through a tunnel
- Protecting clients through a firewall behind the tunnel
- Distributed compilation through VPN tunnels with **distcc**
- Automatic installation for Windows clients
- Redundancy with clusters and bonding

Some aspects of these configurations can only be covered at a basic level (like squid proxy or the clustering techniques) because OpenVPN offers an abundance of possibilities. However, there are hints and links to Internet web sites containing detailed information about these setups.

Tunneling a proxy server and protecting the proxy

OpenVPN can use the HTTP method CONNECT to establish a tunnel between the client and its VPN server. As this is a standard method used by most banking web sites or any other security-conscious web sites, most proxies and firewalls are open to such connections.

A simple OpenVPN configuration entry for use with an HTTP proxy may look like the following:

```
(...)
port 443
proto tcp-client
http-proxy proxy 3128
http-proxy-retry
http-proxy-option AGENT Mozilla/4.0 (compatible; MSIE 4.01; Windows NT 5.0)
(...)
```

We are using port 443 TCP, which will make our VPN tunnel almost invisible to local administrators. OpenVPN must also know where to find the proxy server and on which port it is listening. In the aforementioned example, the name of the server is proxy and its port is 3128. In addition to this, OpenVPN will try indefinitely to establish a connection and stealthily pretend to be a Mozilla browser on Windows 2000. Pretty nice, isn't it?

I consider this as one of the main advantages of OpenVPN. There are only a few networks where an OpenVPN tunnel cannot be set up. Don't worry about the frowning local administrators at your side!

The following table shows possible options for the proxy configuration
of OpenVPN:

Parameter	Function
`--auto-proxy`	Tries autodetection of proxy settings.
`--http-proxy <IP> <port>` `<authfile><auth-method>`	IP and port of proxy server, optionally with proxy authentication.
`--http-proxy-retry`	`<authfile>` is a file containing username and password on two separate lines. `<auth-method>` can be `ntlm`, `basic`, or `none`.
`--http-proxy-timeout<n>`	Retries indefinitely to connect to proxy Sets proxy timeout manually to n seconds. The default is 5 (seconds).
`--http-proxy-option type<option>`	Sets user agent (browser version string) or HTTP version that is used.
`--port`	443 (HTTPS) is probably the most inconspicuous selection (remember to set this on both sides), but most proxies also permit port 80 (HTTP) or 21 (FTP).
`--socks-proxy <IP> <port>`	Uses the socks proxy on machine with `<IP> <port>`.
`--socks-proxy-retry`	Retries indefinitely.

However, there are possible solutions to prevent OpenVPN tunnels. A secure squid
proxy server configuration might, for example, look like the following:

```
(...)
acl SSL_ports port 443 563
acl Safe_ports port 80 # http
acl Safe_ports port 21 # ftp
acl Safe_ports port 443 563 # https, snews
(...)
acl CONNECT method CONNECT
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
(...)
acl ADS_WWW_Benutzer external wb_group WWW_User
```

```
acl ADS_WWW_trusted external wb_group WWW_trusted
(...)
http_access allow WWW_User
http_access allow WWW_trusted
http_access allow WWW_trusted !Safe_ports
http_access allow WWW_trusted CONNECT !Safe_ports
http_access deny all
(...)
```

Squid uses access lists (`acl`) and access directives (`http_access`), which can be found in `/etc/squid/squid.conf`, to control Internet access. In the configuration above, access lists are defined for 'SSL Ports' and 'Safe Ports' for HTTPS and FTP. In lines further down in the file there are `http_access` directives, which explicitly allow access to SSL and safe ports for members of the user group `WWW_trusted` only. In this configuration an external authentication program, `wb_group`, is used. `wb_group` is a small Perl script that enables squid to ask user information from a Microsoft Active Directory Server. On this system, Windows administrators can control the usage of HTTPS or other SSL connections through their proxy server by simply adding or removing users from the privileged group. As a side effect, only users in the group `WWW_trusted` can access `https://` web pages. This may be difficult to communicate in a company, but it is definitely more secure. We have been using similar setups in recent years, and (after convincing the administrators) have only had positive experience.
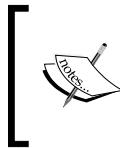
> Combine the proxy settings with the connection profiles of OpenVPN 2.1 and you will almost always have an automatic tunnel provided. I have met only one customer's network in almost ten years where I could not set up an OpenVPN tunnel to my server. With 2.1, this is even easier.

Scripting OpenVPN—an overview

Another striking feature of OpenVPN is its scripting capabilities. We can create our own scripts and have them called on changes to the connection state. This makes it easy to execute a special (for example, Firewall) script any time a client connects, or on similar occasions. There's no limit. I leave it up to you to imagine the possibilities.

The following table gives an overview of the possible interfaces where OpenVPN can be forced to execute arbitrary scripts:

Option	Occurrence
`--learn-address <cmd>`	When the IP address of a VPN partner changes
`--ipchange <cmd>`	When the IP address of the server has changed
`--client-connect <cmd>`	When a client connects
`--client-disconnect <cmd>`	When a client disconnects
`--up <cmd>, down <cmd>`	After configuration (up = starting, down = stopping) of the TUN/TAP device
`--down-pre`	Before shutting down the TUN/TAP device
`--up-restart`	When tunnels are restarted, up/down scripts are also executed

In the man page of OpenVPN, `http://openvpn.net/man.html`, there is a special section, **Environmental Variables**, listing all the variables that are passed to commands. The German website `http://www.pronix.de/pronix-991.html` shows a list of the variables that are passed to the command invoked. For non-German speakers, here is a brief English list of the variables:

- `learn-address`: This option calls a command and hands over three variables.

    - **operation**: It can be one of 'add', 'update', or 'delete', and directly refers to the change of the client's address that has taken place

    - **address**: It contains the IP address which has been set or deleted

    - **common name**: It is the entry from the client's certificate's subject line

- `ipchange`: This refers to the IP address of the VPN server. The command is executed after authentication (or remote IP change).

- `client-connect` and `client-disconnect`: These call commands immediately after connection or disconnection of a VPN client. These options can only be used in OpenVPN server mode.

- `-- up` and `--down`: These are probably the most interesting scripting interface options. The scripts defined here are called immediately after starting or stopping the tunnel interfaces and before an optional `--user` identity change takes place. Thus, here root privileges may be available, which allow, setting routes or similar tasks for example.

Environment Variable	Contents If DEV = TUN	Contents If DEV = TAP
`$1`	Name of (TUN) interface	Name of (TAP) interface
`$2`	MTU	MTU
`$3`	Link-MTU	Link-MTU
`$4`	Local IP of TUN interface	Local IP of TAP interface
`$5`	Remote IP	Netmask of TAP interface
`$6`	init, if called by `--up`, restart if called by `--up-restart`	init, if called by `--up`, restart if called by `--up-restart`

Using a client configuration directory with per-client configurations

Another striking feature of OpenVPN is the fact that we can have client configurations pushed through the tunnel on creation and use client-specific configurations, which are simply set by the subject line of the client's certificate. An appropriate server configuration file may look like the following:

```
port 443
dev tun0FIT
ca /etc/openvpn/certs/ca.crt
cert /etc/openvpn/certs/firewall.crt
key /etc/openvpn/certs/firewall.key
dh /etc/openvpn/certs/dh2048.pem
tls-auth /etc/openvpn/certs/ta.key 0
auth SHA1
cipher AES-256-CBC
tls-cipher DHE-RSA-AES256-SHA
server 10.179.0.0 255.255.0.0
ifconfig-pool-persist /etc/openvpn/ipp.txt
client-config-dir clients
keepalive 10 120
resolv-retry 86400
```

```
comp-lzo
status /var/log/openvpn/status.log
log /var/log/openvpn/main.log
tls-server
verb 3
```

There are three lines that are relevant in this context.

- `server 10.179.0.0 255.255.0.0`: This tells OpenVPN on this machine to act as a server and automatically distribute IP addresses to clients connecting.

- `ifconfig-pool-persist /etc/openvpn/ipp.txt`: This makes OpenVPN keep a list of certificates to IP relationships so that a client connecting will (probably) always have the same IP address.

- `client-config-dir clients`: This has OpenVPN look in the directory 'clients' for a client-specific configuration file when a client connects.

A client configuration file must have a name matching the `CN` in the `Subject` line of the certificate.

```
(...)
Subject: C=DE, ST=Bayern, L=Regensburg, O=Feilner-IT, CN=mfeilner/
emailAddress=mfeilner@feilner-it.net
(...)
```

If a client connects with a certificate containing the previous subject, the server will look if the directory, clients contains a configuration file named `mfeilner`. This file may contain push options like the following:

```
ifconfig-push 10.179.0.3 10.179.0.4
push "route 10.1.0.0 255.255.0.0"
```

In this scenario, the client will always have the IP address `10.179.0.3` and is told about a network (10.1.0.0) behind the tunnel. Thus, if we use different client configurations, we can control the routing and network configuration for every client. But, as we will see later, we can also use the scripting hooks here, thereby generating per-user-scripts and individual environments (see below).

It's simple to grant access to the network by activating or deactivating a client's routing on connecting, but we must always remember that this offers no real protection. This is because every local administrator could also activate this routing on the client.

On the client configuration, the parameter, `client`, must be present. If we want to have the client redirect its default gateway through the tunnel, then we simply need to add the parameter `redirect-gateway`.

The ability to redirect the client's default gateway is another excellent feature of OpenVPN, especially when combined with HTTP-proxy tunneling. The parameter `redirect-gateway` causes the following three actions:

1. A static route to the other tunnel partner is created.
2. The old default gateway is deleted.
3. A new entry for the default gateway is created (pointing to the IP address of the other tunnel endpoint).

Of course we can enter these steps manually if we like. The following `route` command will help us here:

```
vpnserver:~# route add 172.16.103.2 gw 172.16.247.1
vpnserver:~# route del default
vpnserver:~# route add default gw 10.179.10.2
vpnserver:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref
Use Iface
172.16.103.2    172.16.247.1    255.255.255.255 UGH   0      0
0 eth0
10.179.10.2     0.0.0.0         255.255.255.255 UH    0      0
0 tunVPN0
172.16.247.0    0.0.0.0         255.255.255.0   U     0      0
0 eth0
172.16.76.0     10.179.10.2     255.255.255.0   UG    0      0
0 tunVPN0
192.168.250.0   0.0.0.0         255.255.255.0   U     0      0
0 eth1
0.0.0.0         10.179.10.2     0.0.0.0         UG    0      0
0 tunVPN0
vpnserver:~#
```

First, we added a static route to the VPN partner (`route add 172.16.103.2 gw 172.16.247.1`). Then we deleted the old default route (`route del default`), and as a last step we created the new default route with `route add default gw 10.179.10.2`. From this moment on, all traffic that is not destined to the VPN partner's public IP address will be routed through the tunnel, as the output of `route -n` will show. Because the routing entries will be useless when the VPN partner's IP address changes, it is a better idea to have OpenVPN set the routing for us.

The next chapter deals with interpreting routing tables in more detail.

Individual firewall rules for connecting clients

One striking possibility that OpenVPN offers is a setup where:

- An OpenVPN machine acts as a server that protects the company's network, admitting access for OpenVPN clients
- The clients are automatically assigned IP addresses by the server
- The clients are equipped with certificates, and are identified and authorized by these certificates.

The scripting parameter `learn-address` in the server's OpenVPN configuration file will have the server execute a script whenever an authorized client connects to the VPN and is assigned an address. The following parameter takes the full path to a script as an option:

```
learn-address /etc/openvpn/scripts/openvpnFW
```

In this example, the script `openvpnFW` will be executed each time a client is assigned an IP address and will be passed three variables by the OpenVPN server process.

- `$1`: The action taken. This may be one of add, delete, or update.
- `$2`: The IP address assigned to the client connecting.
- `$3`: The common name in the subject line of the client's certificate.

Add the line `learn-address /etc/openvpn/scripts/openvpnFW` to your OpenVPN server configuration file and edit the file `/etc/openvpn/scripts/openvpnFW` to match the following. The following lines will show how to make use of these parameters in a short Linux shell script:

```
#!/bin/sh
LOGFILE= /var/log/openvpn/connections.log
DATE=`/bin/date`
echo $DATE $1 $2 $3 >> $LOGFILE
```

This script will only export the variables passed to the logfile, including a timestamp that is added by the command `date`. Stop and start your tunnel a few times. Now let's have a look at the file `/var/log/openvpn/connections.log`:

```
Mi Feb 1 04:33:53 CET 2006 update 10.99.0.3 mfeilner
Do Feb 2 04:34:33 CET 2006 update 10.99.0.3 mfeilner
Fr Feb 3 04:34:14 CET 2006 update 10.99.0.3 mfeilner
Sa Feb 4 04:34:53 CET 2006 update 10.99.0.3 mfeilner
So Feb 5 04:34:43 CET 2006 update 10.99.0.3 mfeilner
```

The preceding example shows my VPN client reconnecting every day. This alone might be an interesting feature if you want to keep track of your users and their VPN connections. However, we can do more. Let's add some more lines to our `openvpnFW` script.

```
if [ $1 = add ]
then
/etc/openvpn/scripts/$2.FW_connect.sh
fi
if [ $1 = delete ]
then
/etc/openvpn/scripts/$2.FW_disconnect.sh
fi
```

Two simple tests are run, and depending on the content of the variable `$1`, different firewall scripts are executed. Let's express this in brief. If the first variable passed is `add`, then the script `/etc/openvpn/scripts/$2.FW_connect.sh` is run, where `$2` will be replaced by the IP address of the connecting client. If, for example, a client `mfeilner` connects and is assigned the IP `10.99.0.3`, then the variables passed to this script `openvpnFW` will be as follows:

```
add 10.99.0.3 mfeilner
```

And the script run will be called:
`/etc/openvpn/scripts/10.99.0.3.FW_connect.sh`.

Consider the following variable:

```
delete 10.99.0.3
```

If the preceding variable is passed to `openvpnFW`, then the script `/etc/openvpn/scripts/10.99.0.3.FW_disconnect.sh` will be executed.

I think you have already guessed that these two scripts contain firewall rules (like `iptables` statements) for the client with the certificate `mfeilner`. Even though all of this could be done within one single script, I prefer to have the tests and firewall rules split up in several scripts.

This setup can become very powerful and fairly complex. A client that has its default route set through the tunnel can be allowed selective Internet access, simply by enabling or disabling, routing or forwarding. Moreover, access to the local servers can also be easily managed. For example, a SAP server might only be available for road warriors from 7 am to 6 pm, whereas during the night firewall rules protect the server.

Distributed compilation through VPN tunnels with distcc

The `distcc` is a compiler (or a frontend to **GNU Compiler Collection** (**GCC**)) designed to split up compiling processes over many machines, which can speed up the process enormously. The **distcc** daemon has to be run on all of the systems that are to participate, then the system starting the process must be informed about the `distcc` hosts, and then we can start a compiling process.

On Debian systems, installation is as easy as typing `apt-get install distcc`. As a next step, some parameters have to be set in `/etc/default/distcc`:

- Whether `distccd` should be started on boot
- A list of other `distcc` hosts that are allowed to connect
- The interface `distcc` should listen for incoming connections

The following is the file `/etc/default/distcc` on a Debian system:

```
# Defaults for distcc initscript
# sourced by /etc/init.d/distcc

#
# should distcc be started on boot?
#
# STARTDISTCC="true"

STARTDISTCC="false"

#
# Which networks/hosts should be allowed to connect to the daemon?
# You can list multiple hosts/networks separated by spaces.
# Networks have to be in CIDR notation, f.e. 192.168.1.0/24
# Hosts are represented by a single IP address
#
# ALLOWEDNETS="127.0.0.1"

ALLOWEDNETS="127.0.0.1"


#
# Which interface should distccd listen on?
# You can specify a single interface, identified by it's IP address,
here.
#
# LISTENER="127.0.0.1"

LISTENER="127.0.0.1"
```

Here we will have to edit the parameters, ALLOWEDNETS, and LISTENER, to our needs, and repeat this step for every partner that is supposed to take part in the collective compilation. Then, either edit your startup files to include a system variable called DISTCC_HOSTS, or create a configuration file, ./distcc/hosts, in your home directory with a list of the other hosts that are supposed to take part in compiling. The content of this variable or file should simply be a (space-separated) list of hosts such as the following:

```
10.179.0.1 192.168.1.4 10.179.0.3
```

I think you will already know where this is leading to. We will install OpenVPN tunnels on each machine taking part in the distcc network and then we only need to enter the IP address of the tunnel machines in these files.

That's all, now we can use distcc over the tunneled connections. Therefore, the distcc daemon has to be started with /etc/init.d/disstcc start and then we can start a compiling process where we use distcc as compiler. For instance, in the directory /usr/src/linux, simply type make CC=distcc to have the selected machines in your network compile this machine's kernel together. Or have a look at the following example where OpenVPN is compiled through distcc:

```
vpnclient1:~/# make CC=distcc
make  all-am
make[1]: Entering directory `/root/openvpn-3.0'
if distcc -DHAVE_CONFIG_H -I. -I. -I.    -I.    -g -O2 -MT mroute.o -MD
-MP -MF ".deps/mroute.Tpo" -c -o mroute.o mroute.c; \
        then mv -f ".deps/mroute.Tpo" ".deps/mroute.Po"; else rm -f
".deps/mroute.Tpo"; exit 1; fi
if distcc -DHAVE_CONFIG_H -I. -I. -I.    -I.    -g -O2 -MT mss.o -MD -MP
-MF ".deps/mss.Tpo" -c -o mss.o mss.c; \
        then mv -f ".deps/mss.Tpo" ".deps/mss.Po"; else rm -f ".deps/
mss.Tpo"; exit 1; fi
if distcc -DHAVE_CONFIG_H -I. -I. -I.    -I.    -g -O2 -MT mtcp.o -MD
-MP -MF ".deps/mtcp.Tpo" -c -o mtcp.o mtcp.c; \
        then mv -f ".deps/mtcp.Tpo" ".deps/mtcp.Po"; else rm -f
".deps/mtcp.Tpo"; exit 1; fi
(...)
```

Ethernet bridging with OpenVPN

On Linux, Windows XP, and Windows 2003, we can use VPN tunnels as one big logical Ethernet network. By connecting (bridging) a virtual OpenVPN interface and a real Ethernet interface, we connect (bridge) the networks behind these interfaces and provide a virtual Ethernet between the hosts in the real networks, including exchange of Ethernet frames. This feature can be useful for Windows users that need to exchange broadcast packages through the tunnel, for example, for network browsing, LAN parties, and more.

Setting up OpenVPN for bridging mode is simple and the same on all operating systems. We only have to make sure our OpenVPN setup is working and that we are using TAP devices. I recommend the use of TLS-server setup with clients that are automatically assigned addresses and configurations.

On Linux, you will need to install the **bridge-utils** package and follow the instructions on the web site `http://openvpn.net/bridge.html`. It is much easier for Windows users. Just use the network settings of your operating system to activate bridging mode. Open the **Network Connections** window and select (mark) the two network interfaces that you want to bridge. Then select the entry, **Bridge Connections**, from the context menu.
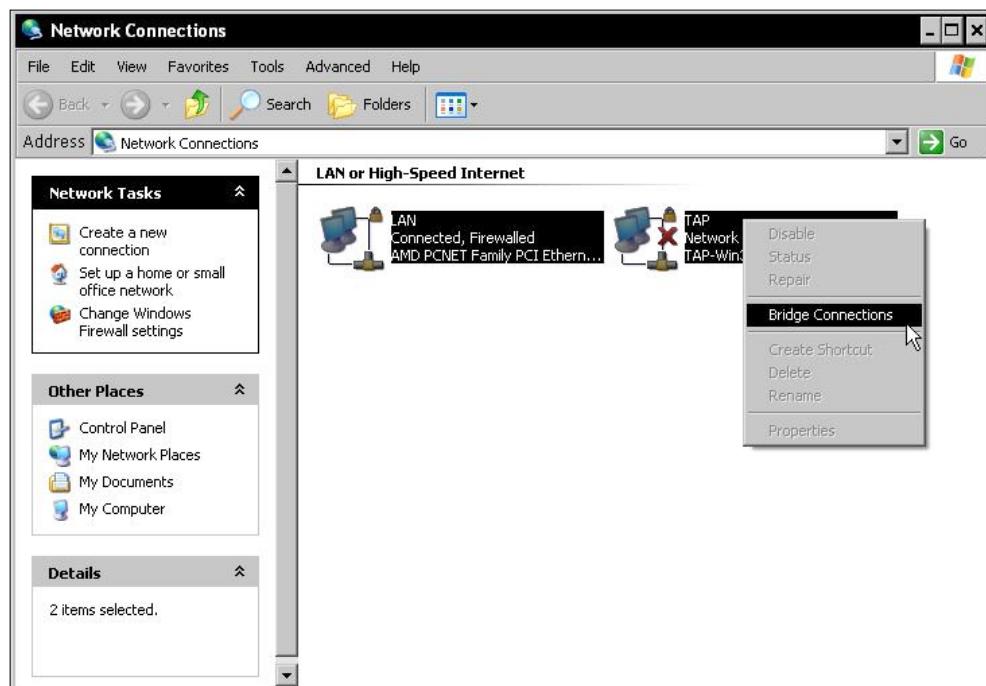
A new icon will appear, called **Network Bridge**, and the LAN interface will show **Bridged** in its name.
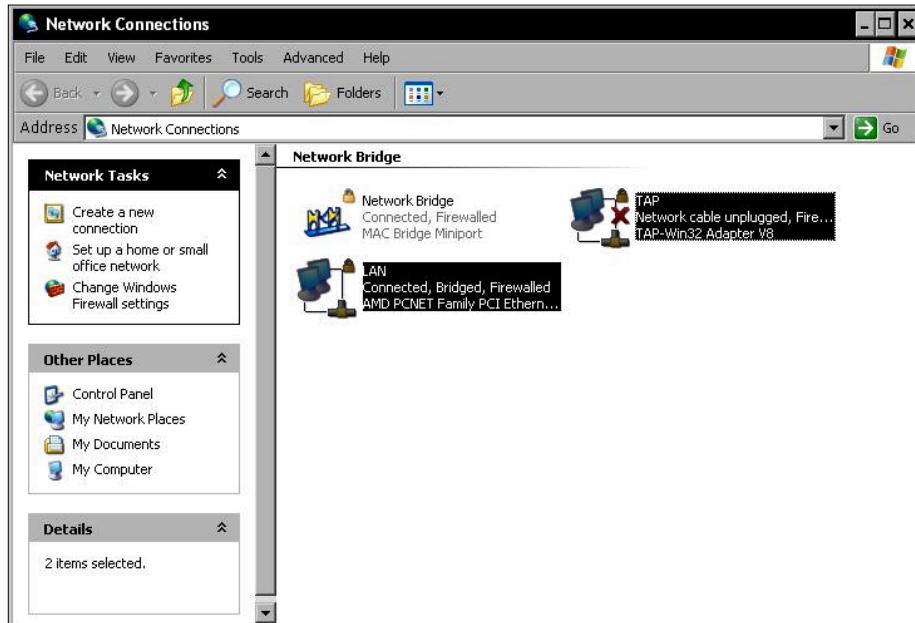
This Ethernet bridge can now be configured in a similar way to any other network device. Select the entry properties from its context menu.
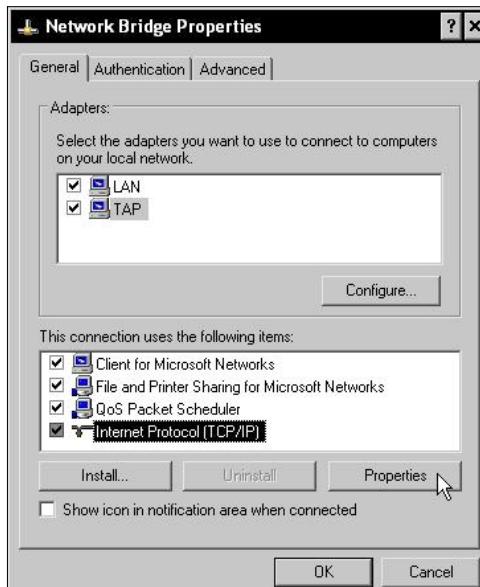
As a last step, we have to assign an IP address to this interface or configure the interface to obtain an IP address automatically, this being the default setting. Select the entry, **Internet Protocol (TCP/IP)**, from the list, **This connection uses the following items:** and click on the button **Properties** to assign an IP address.
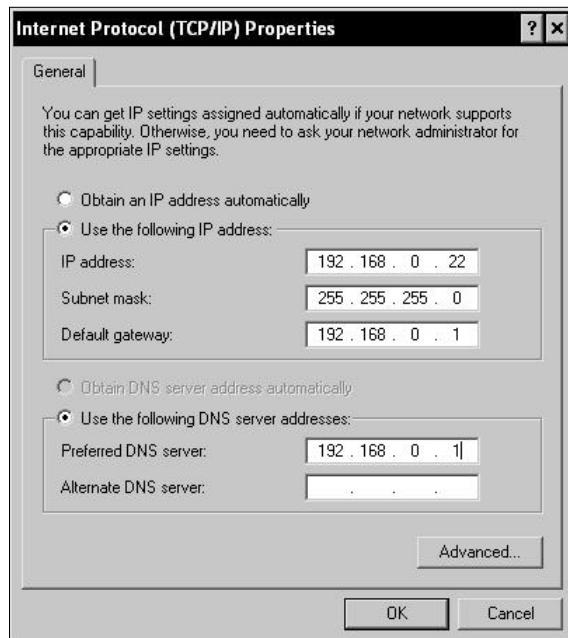
That's it! Your Ethernet bridge is now up and running. If you run into trouble with your OpenVPN configuration, then check these web sites for examples and guidelines:

`http://www.pavelec.net/adam/openvpn/bridge/` and

`http://openvpn.net/bridge.html`

Automatic installation for Windows clients

If you have to administer a large Windows network, then you will probably know the pains of having to install software on several clients. There is a convenient way to install OpenVPN (almost) automatically. The open source Windows software **Nullsoft Scriptable Install System** (**NSIS**) installer is available from `http://www.openvpn.se/files/nsis/nsis205.exe` and documented in `http://openvpn.se/files/howto/openvpn-howto_roll_your_own_installation_package.html`. The installer creates an executable file including configuration and certificate for your client.

Simply download the NSIS installer and execute it. In most cases, you will not need to make any changes to the default values during installation except maybe for the path. Simply click on the **Next** button three times, agree to the license, and NSIS is installed.
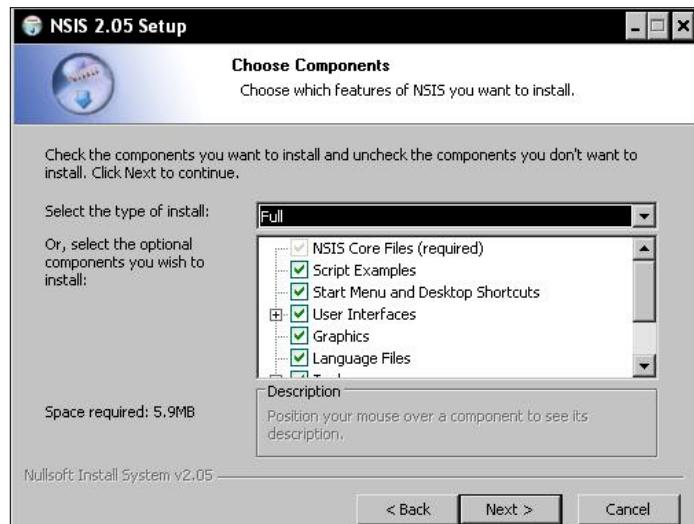
The following window shows the standard dialog of the NSIS installer providing detailed information on this tool:
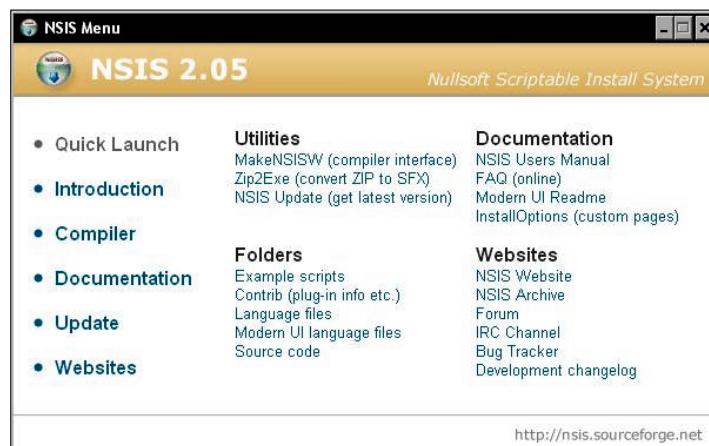
If you are interested in more information on the NSIS installer, have a look at the **Websites** link here. Your next step will be downloading and extracting the OpenVPN-GUI source code from `http://www.openvpn.se/files/ install_packages_source/`.
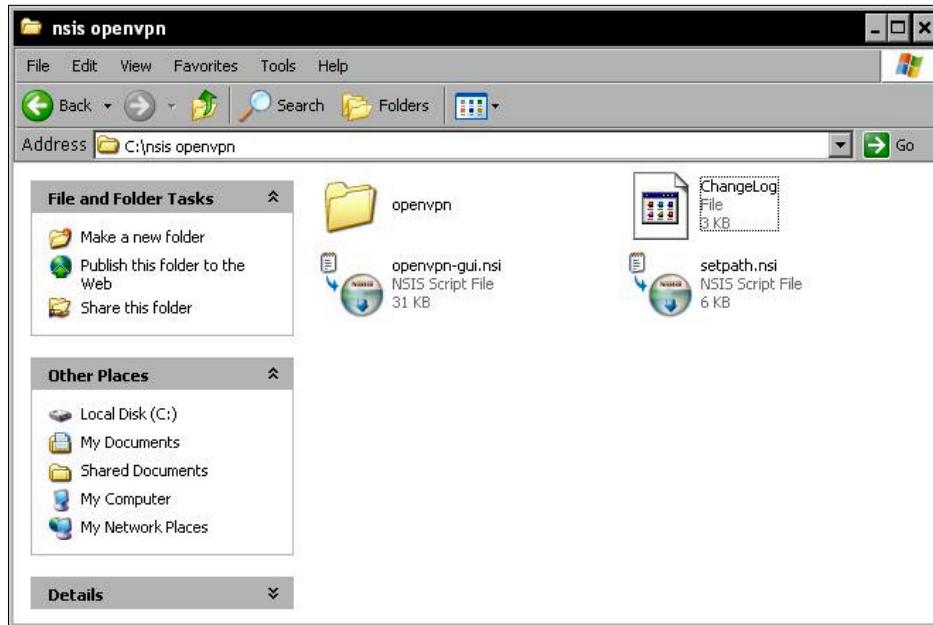
Then copy your OpenVPN configuration and certificates to the directory you extracted the sources to and open the file `openvpn-gui.nsi` with Notepad. Here you only need to enter the name of your files and the path, if it differs from the values in the file. Search for lines containing `<File "${HOME}\config\Office.ovpn">` and change this according to your needs.
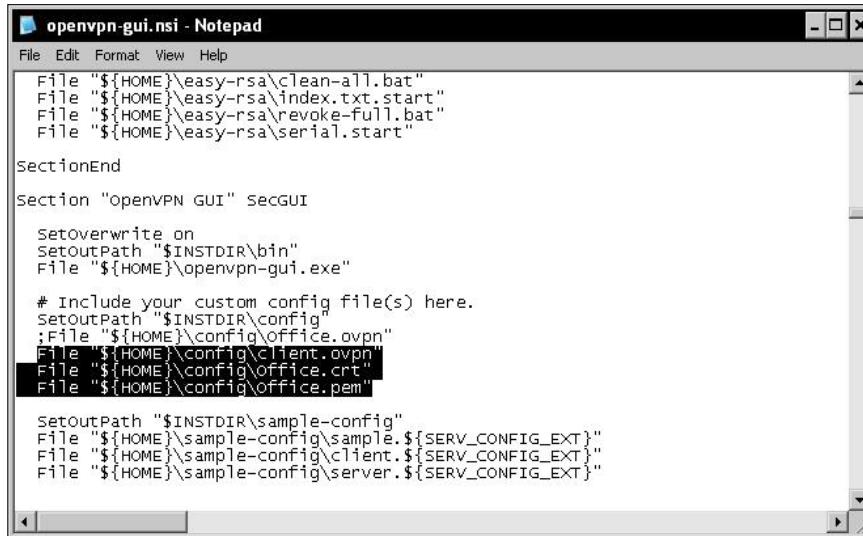
The section **Modifying the script for your own needs** of the web site
`http://openvpn.se/files/howto/openvpn-howto_roll_your_own_`
`installation_package.html` gives detailed information on possible and necessary
changes for different scenarios. If you want to have configuration files deleted when
OpenVPN is uninstalled, then add the lines similar to the following ones:

```
Delete "$INSTDIR\config\client.ovpn"
Delete "$INSTDIR\config\client.crt"
Delete "$INSTDIR\config\client.pem"
```

As a last step, we will now start the compilation progress, which is done with a
simple context menu entry generated by the NSIS installer. Right-click on the file
`openvpn-gui.nsi` and select the menu entry **Compile NSI Script**.
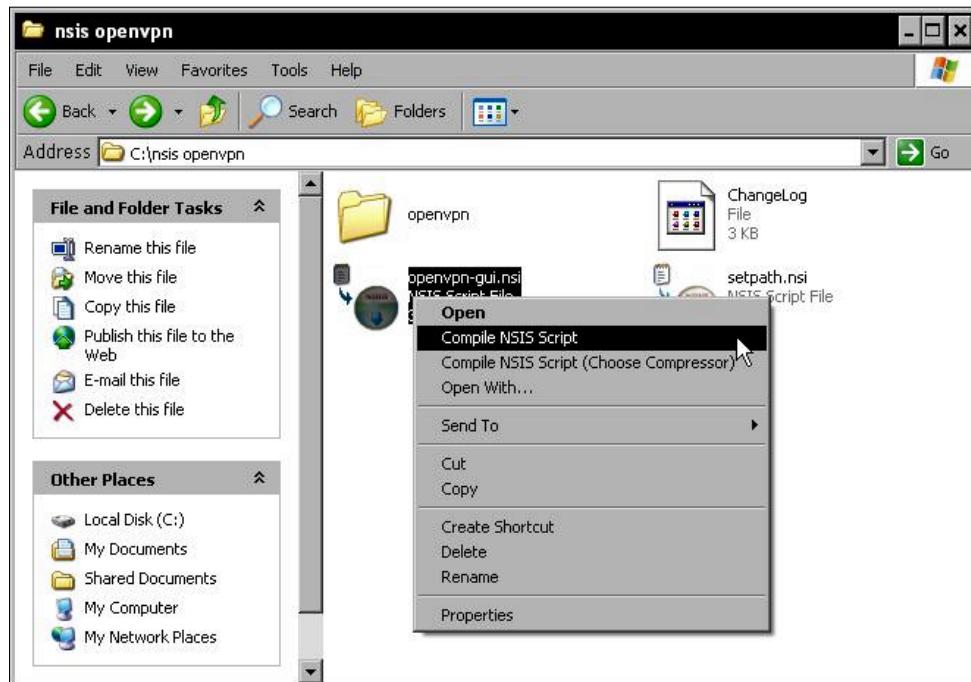
You will receive the following status window telling you about the progress.
In the following example, an installer `.exe` file is created as `C:\nsis openvpn\`
`openvpn-2.0.5-gui-1.0.3-install.exe`.
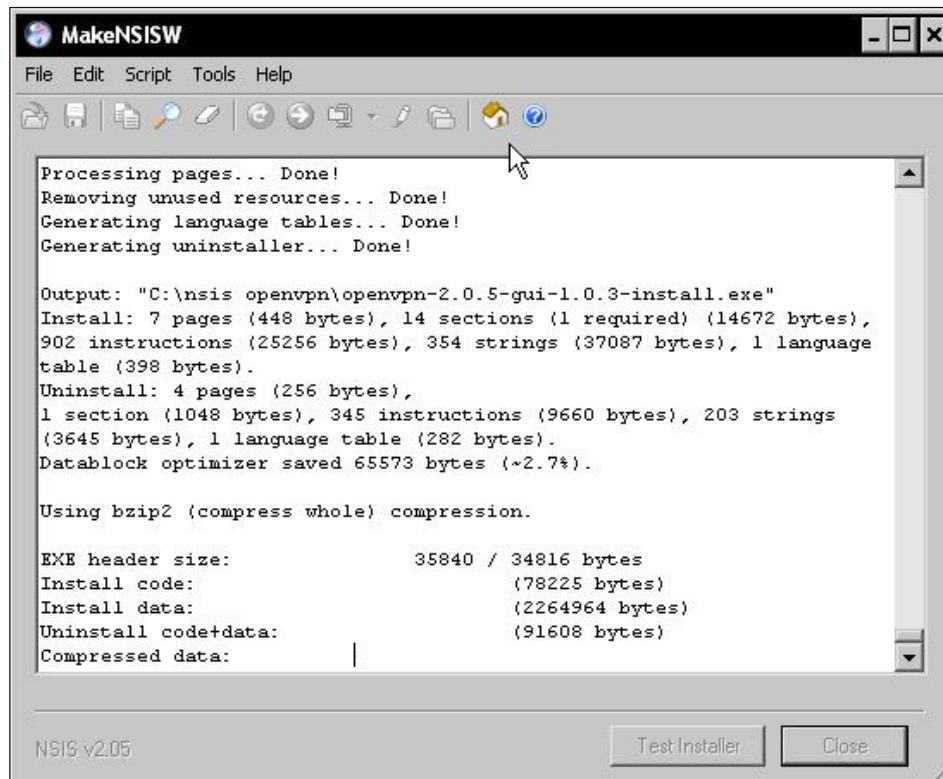
You can now transfer this `.exe` file to all clients and install them automatically with the configuration that you provided earlier. Installation works exactly like the standard installation described before.

However, there is a small problem here. We will need to change every client's certificate once. Otherwise all clients would have the same certificate, which is not a really safe situation. Thus, all we have to do having completed the steps above is as follows:

- Transfer the `.exe` file to a client
- Have it executed as administrator
- Copy the client's certificate to the client

You will need to use the same name for all certificates and configuration files on all clients, but again this is no problem. This is because the common name of the certificate's subject line will distinguish the clients.

Clustering and redundancy

A big topic and a simple solution with OpenVPN. Just set up two or more identical servers and keep the directory `/etc/openvpn` and its subdirectories in sync. Then have your clients use the option remote-random and enter the IP addresses of the servers, done. From that moment on, your clients will almost always get a connection, even if you unplug the electricity from one of the servers. But you still have to keep the servers in sync and will have some administration work. Other options include the following:

- Set up a Heartbeat (`http://www.linux-ha.org/`) cluster with a shared filesystem like DRBD (`http://www.drbd.org/`) or a storage solution behind it and have two or more machines use the same storage. Your clients won't notice a failover. The *Linux Technical Review*, a German high-end Linux magazine has an article in its edition '04 High Availability' of 2007, on pages 114 through 117 that describe a simple setup for this, but only in German. There is always an interesting thread in the OpenVPN club's forum that will surely be translated to English soon: `http://forum.openvpn.eu/viewtopic.php?f=1&t=3183&start=0&hilit=cluster&sid=e238c6598c4a215de2a3885d1a7dcadd`

- Set up two tunnels through two dedicated DSL or Modem lines. Use both of them for normal traffic, for example, one for printing, the other for terminal services. Have Heartbeat control the tunnels, and switch all to one tunnel if one of them fails. What you need for that are the tools of the iproute2 packages (`http://lartc.org/howto/`, `http://www.policyrouting.org/PolicyRoutingBook/ONLINE/TOC.html`), namely programs like `tc` that allow port-based routing and some sophisticated heartbeat setup. And of course two DSL lines (or Modems).

- As in the last example, use two tunnels and bond your interfaces. Bonding is a network-layer failover system integrated in the Linux kernel that can combine two Ethernet devices to one, internally switching in case of errors. Together with techniques like multipathing (routing to the same destination through different routers), you can have a network-level redundancy that other solutions won't achieve. There is an interesting thread in the OpenVPN club's forum that will surely be translated to English soon, `http://forum.openvpn.eu/viewtopic.php?f=1&t=4857&start=0&hilit=bonding&sid=e238c6598c4a215de2a3885d1a7dcadd`

Summary

In this chapter, we have discussed some typical advanced configurations for OpenVPN that demonstrated some of its advantages. We have tunneled OpenVPN through an HTTP proxy and then configured a squid proxy so that we could control who is allowed to do so. Then we had a closer look at the scripting interfaces that OpenVPN offers, including lists of variables that are passed to the scripts by OpenVPN on invocation.

As a next step, we configured OpenVPN to use a per-client configuration based on the client's certificate, which would enable different configurations for different users connecting. This scenario can be made even more complex when combined with per-user firewall rules being activated on the VPN server after a client connects.

The `distcc`, a network-enabled compiler frontend to GCC, can be used together with OpenVPN tunnels to have remote machines work as a team when compiling software. Finally, we looked at automatic installation for Windows machines using the NSIS installer and methods for achieving redundancy.

14

Mobile Security with OpenVPN

OpenVPN is not only a server security tool. In this chapter we will learn how to install and configure it on mobile devices from a standard laptop that is supposed to have secure and uncensored Internet access, using a Windows Machine at home connected through DSL or other subscription lines. I will then show you how to install OpenVPN on Windows Mobile and Smartphones by running embedded Linux such as Nokia's Maemo platform.

Anonymous and uncensored Internet Access

Travelers around the world often visit regions where simple data transfer of files over the web may prove dangerous because some third world countries are known to filter and analyze all data sent from your notebook. And if the local authorities censor web access, then you won't be able to see important web sites for your work. This is where OpenVPN comes into the game. Once a tunnel is set up, no one can read your data. And if you have configured your default route through the tunnel, no local censorship authority will know which page you were looking at. For the following example, a Windows server at home or in your office will do. A PC connected through, for example, DSL receives a new IP every day (at least with most of the providers), so an authority might find it hard to block the IP.

The whole setup is done in a few easy steps.

- Get yourself one or more accounts from services such as `dyndns.org` or `no-ip.com`. Such web sites help accessing, let's say, a PC that is only connected over a leased Internet line, which frequently changes its IP. In Germany, this is normally done by the Provider every 24 hours. Software, like the one provided by `dyndns.org`, checks and updates the DNS-Entry automatically if your IP changes.

- Configure your DSL (or whatever kind) router to forward the OpenVPN port to your Windows machine. Consult your vendor's manual and look for terms like NAT, Masquerading, or Port Forwarding. Don't forget to specify a static IP for your VPN server, either in your router or DHCP-server, or simply in the network configuration of your VPN server on Windows, for example, in the network configuration dialog (**TCP/IP protocol**).

- Install OpenVPN on your server and create certificates for your server, CA, and your client. This is described in detail in previous chapters in this book.

- In the last step, we have to complete the configuration. This implies a working OpenVPN configuration file with at least the option `"push "redirect-gateway" "` with correct routing and certificate setup. Also, you have to bridge the VPN-device and your local LAN device, as described in Chapter 13. Here is a working configuration for your server using the Https port 443:

```
port 443
proto tcp-server
dev tap
ca ca.crt
cert server.crt
key server.key
dh dh1024.pem
server-bridge local_address_of_the_Windows_PC 255.255.255.0
192.168.0.240 192.168.0.250
push "redirect-gateway"
push "route 0.0.0.0 0.0.0.0 IP_of_the DSL_router"
push "dhcp-option DNS IP_of_the_DSL_router"
keepalive 10 120
comp-lzo
persist-key
persist-tun
```

And for your client:

```
client
dev tap
proto tcp-client
remote feilner.dyndns.org 443
resolv-retry infinite
nobind
persist-key
persist-tun
ca ca.crt
cert client.crt
key client.key
comp-lzo
route-up myroute.cmd
```

Don't forget to adapt your IPs and DNS entries to your needs and provide the certificates on either client. The route-up command file `myroute.cmd` should look like this and resides in your configuration directory:

```
route delete 0.0.0.0 MASK 0.0.0.0 \ Local_IP_address_of_the_
Windows_computer_behind_DSL
```

Here you have to enter your server's IP so that the client has the correct routing setup. That's it! Now you can surf without censorship and without the fear of unwanted readers. If you want more, then have a look at the possibility of using several DNS-Names with the `remote` command, the `remote-random` command and especially OpenVPN 2.1 using connection profiles (see Chapter 9). If you want a more detailed description of the setup described here, go to `http://www.linux-magazine.com/online/features/set_up_openvpn_in_four_steps`

OpenVPN on Windows Mobile

The benefits of OpenVPN are not limited to PCs or Macs, many smartphones, especially those running Windows Mobile, can use it because there is an OpenVPN client available here: `http://ovpnppc.ziggurat29.com/ovpnppc-main.htm`. Although this tool dates from 2007, it still works flawlessly and offers all the basic functions that a client needs. The Cab files for installation are best downloaded directly from and to the smartphone.

While installation is simple, it may prove more difficult to transfer your certificates and configuration files from your PC to the phone. Especially finding the right path to enter in the GUI's dialogs may become an interesting job, and although all smart phones seem to store this data in the same place, the path may vary. On the HTC device we tested, the correct path for the smartphone's configuration was:

```
cert "\\Memorycard\\Programs\\OpenVPN\\config\\client.crt"
```

This is a complete example configuration for a German language HTC with a simple Key and no certificates:

```
dev tun
proto udp
remote feilner.dyndns.org 1194
ifconfig 10.3.0.1 10.3.0.2
resolv-retry infinite
nobind
persist-key
persist-tun
secret "\\Speicherkarte\\Programme\\OpenVPN\\config\\key.txt"
#pkcs12 "\\Speicherkarte\\Programme\\OpenVPN\\config\\client.p12"
comp-lzo
verb 3
```

As you can see, there is also an example for `pkcs12` certificate files. After downloading and installing the `.cab` file, you will find a new icon called **OpenVPN Connection Manager** (ovpncmgr) in your `Programs` section:

The rest is simple (assuming that you put your certificates and configuration file in the right directory). Start ovpncmgr and click on the **Menu** button:

On our test device, we have two configs for testing, namely, sample and client. The `.ovpn` extension is automatically stripped by Ovpncmgr. Once you have started one of the available connections, it will be listed in the table above, and a new tab is added to the GUI. In this tab, which is named **[vpn_1]** or something similar, Ovpncmgr offers you detailed information about (from left to right) your VPN, traffic statistics, and your network data:
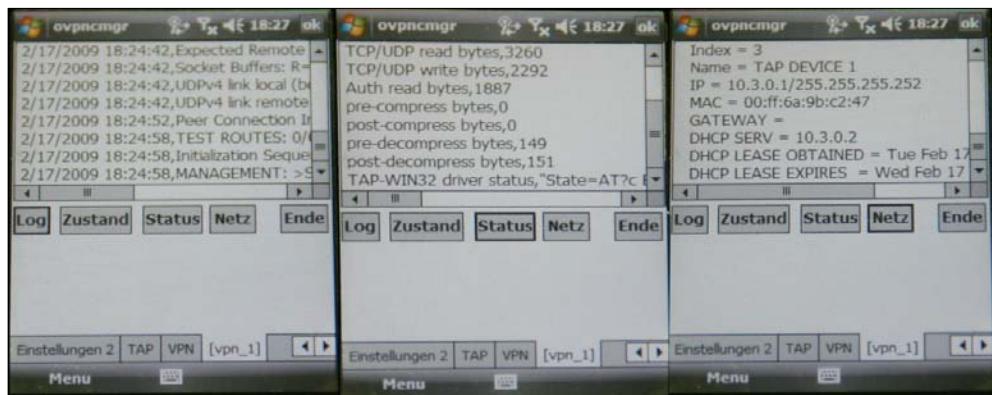
The Windows mobile port of OpenVPN seems to understand all the features of its bigger brother, including authentication and receiving pushed configuration parameters like `redirect-gateway` and `remote-random`. However, some features are more important for mobile phones. For example, the tracking of bytes transferred, which you will find through the button **Status** or **State**, may prove an ideal cost control for expensive wireless connections such as UMTS or GPRS.

Once configured, the OpenVPN client appears as a small icon in Windows Mobile's systray, from which you can start and stop your tunnels at will.

Embedded Linux – Maemo

Another widespread platform for mobile smartphones is embedded Linux variants. Because Nokia's Maemo system has gathered a lot of fans during the last few years, there are many software available for devices like the small touch-screen, web applets called N 800, 810, and N 900.

The screenshot shows the OpenVPN client software in action. As on Windows Mobile, the main problem for the user is in finding out the right path for his/her certificates, keys, and configuration files. But other than on Microsoft's systems, you can easily install an SSH server on these smart phones and correct your configuration from the command line (remotely!).

```
Nokia-N810-36-5:~# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1918 (1.8 KiB)  TX bytes:1918 (1.8 KiB)

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.3.0.1  P-t-P:10.3.0.2  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:93 errors:0 dropped:0 overruns:0 frame:0
          TX packets:59 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:8688 (8.4 KiB)  TX bytes:7416 (7.2 KiB)

wlan0     Link encap:Ethernet  HWaddr 00:1D:6E:9B:C6:42
          inet addr:192.168.0.113  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:253 errors:0 dropped:0 overruns:0 frame:0
          TX packets:205 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:80670 (78.7 KiB)  TX bytes:33315 (32.5 KiB)

Nokia-N810-36-5:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.3.0.2        *               255.255.255.255 UH    0      0        0 tun0
192.168.0.0     *               255.255.255.0   U     0      0        0 wlan0
default         10.3.0.2        0.0.0.0         UG    0      0        0 tun0
```

This is a typical debugging SSH-session on a Nokia N 810 which shows the network setup.

Although you may use the browser to get the OpenVPN-client for Maemo, for example, from the project's website `http://www.maemo.org`, you don't need to. Just open the software management and scroll down until you find the entry **OpenVPN** and confirm that you want to have it installed. Repeat this step for the entry **openvpn-applet**.

You'll receive a few warnings that Nokia has not certified this software and other such similar messages, but be brave and continue because it's worth the price. Once you are through with the installation, a small applet appears on your mini desktop. If no tunnels are configured, then a click on **Open** will bring up an empty window, similar to the following screenshot:
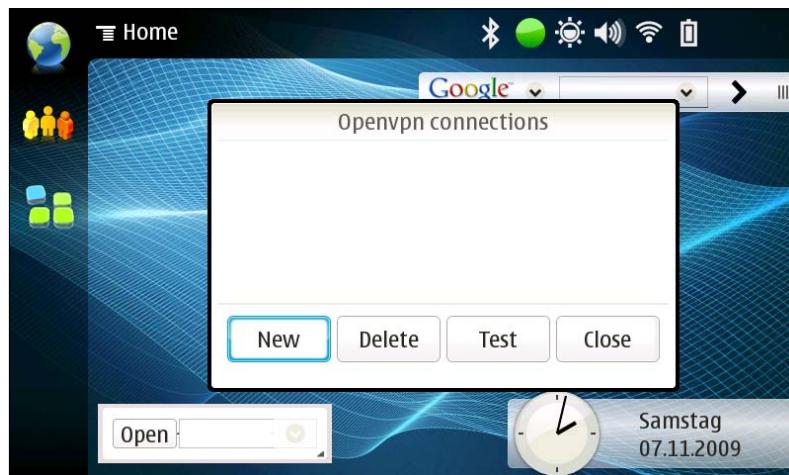
Clicking on **New** makes the dialog shown in the first screen shot of this section appear. All you need to do is enter the correct paths to your configuration file(s), certificates, and keys. If your paths are right, OpenVPN will work like a charm. The smartphone manages the tunnel, no matter whether you're online through Bluetooth, WLAN, UMTS, or any other means.

If you run into trouble, use the local terminal application or the SSH access and look for the logfiles, just like you would do on a big Linux system. Apparently, the openvpn-applet has no such feature built-in.

Summary

In this chapter we have learnt how to configure our notebook's internet access and make it safe and private. In order to avoid unwanted men-in-the-middle, unauthorized readers, and limitations due to censorship by authorities, we used a PC connected to dial-in Internet access, while regularly changing its IP. We adapted firewall and router configurations and bridged the remote laptop to our Home LAN, providing it with uncensored and secure Internet access. We then had a look at the OpenVPN client for Windows Mobile which offers a lot of handy functions. Last but not least, Nokia's Maemo platform stands as an example for embedded Linux variants that can run OpenVPN.

15
Troubleshooting and Monitoring

In this chapter we will learn how to use tools to debug and monitor VPN tunnels. We will also learn how to scan and test the connectivity of a (VPN) server with standard Linux command-line tools. In the second part we will deal with the following graphical monitoring tools:

- Ntop
- Munin
- Nagios
- OpenVPNgraph

Testing network connectivity

In our typical OpenVPN setup, we have connected two networks (192.168.250.0/24 and 172.16.76.0/24) through two Linux servers that are connected to the Internet through a default gateway. Between the two Linux servers is a tunnel that uses virtual interfaces with the IPs 10.179.10.1 and 10.179.10.2.

In the connected local networks there are two Linux machines that we will use to test our tunnels (perhaps by conveniently accessing them remotely with Secure Shell). We will now use the tools `ifconfig`, `route`, and `ping` to show and test the network settings.

In our first step, we will check the local system's network address, default route, and if the default router is pingable. The command `ifconfig` will print statistics of all active network interfaces:

```
root@sydney:~ #ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:AE:8C:D7
          inet addr:192.168.250.128  Bcast:192.168.250.255
Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2640 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2290 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:250738 (244.8 KiB)  TX bytes:273328 (266.9 KiB)
          Interrupt:10 Base address:0x1080

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:57 errors:0 dropped:0 overruns:0 frame:0
          TX packets:57 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:7907 (7.7 KiB)  TX bytes:7907 (7.7 KiB)
root@sydney:~ #
```

This system has the IP address `192.168.250.128` and its network interface is up and running. Obviously this machine is located in Sydney, Australia.

Now let's look at its routing entries. The command `route` prints all routing entries, including the router to the Internet. A **default gateway** is a router that is supposed to handle all traffic not specified by any other routing entries. In our networks, the OpenVPN server is the only router from the internal network and is therefore configured as the default gateway for the local network.

Type the command `route -n` to receive a numeric output of the routing table of your system. Simply typing `route` will work in most cases, but the command will try to resolve the IPs through DNS, which might take a little time.

```
root@sydney:~ #route -n
Kernel IP routing table
Destination     Gateway          Genmask         Flags Metric Ref
Use Iface
192.168.250.0   0.0.0.0          255.255.255.0   U     0      0
0 eth0
0.0.0.0         192.168.250.251 0.0.0.0          UG    0      0
0 eth0
root@sydney:~ #
```

We see a table where destinations, gateways, netmasks, and interfaces are listed. Every line is a routing entry that can be read like a real sentence. An entry `0.0.0.0` simply matches every address (source or destination, depending on the context) and is an example used for the default gateway.

Line three shows that the network `192.168.250.0` is directly connected to the network interface `eth0`, independently from any gateway.

Line four indicates that all the traffic to any destination will be sent over the default gateway `192.168.250.251` through interface `eth0`.

This setup is perfectly OK for a typical network client. Let's now test if the default gateway is reachable by pinging it from the client:

```
root@sydney:~ #ping 192.168.250.251
PING 192.168.250.251 (192.168.250.251): 56 data bytes
64 bytes from 192.168.250.251: icmp_seq=0 ttl=64 time=1.3 ms
64 bytes from 192.168.250.251: icmp_seq=1 ttl=64 time=0.6 msw
64 bytes from 192.168.250.251: icmp_seq=2 ttl=64 time=0.4 ms

--- 192.168.250.251 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.7/1.3 ms
root@sydney:~ #
```

It works! The default gateway (our OpenVPN server) answers the ping requests from our client. If it doesn't answer the ping request in your setup, check the firewall rules on this server to ensure they allow traffic from the internal network to the firewall itself. If you are unsure, it may be a good idea to temporarily stop the firewall services.

Now let's try the same on the client in the other network (obviously in Germany):

```
root@munich:~ #ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:21:07:FC
          inet addr:172.16.76.128  Bcast:172.16.76.255
Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2399 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2715 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:345146 (337.0 KiB)  TX bytes:271839 (265.4 KiB)
          Interrupt:10 Base address:0x1080

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
```

```
                RX packets:8 errors:0 dropped:0 overruns:0 frame:0
                TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
                collisions:0 txqueuelen:0
                RX bytes:772 (772.0 B)  TX bytes:772 (772.0 B)
root@munich:~ #route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref
Use Iface
172.16.76.0     0.0.0.0         255.255.255.0   U     0      0
0 eth0
0.0.0.0         172.16.76.251   0.0.0.0         UG    0      0
0 eth0
root@munich:~ #ping 172.16.76.251
PING 172.16.76.251 (172.16.76.251): 56 data bytes
64 bytes from 172.16.76.251: icmp_seq=0 ttl=64 time=2.0 ms
64 bytes from 172.16.76.251: icmp_seq=1 ttl=64 time=0.5 ms
64 bytes from 172.16.76.251: icmp_seq=2 ttl=64 time=0.5 ms

--- 172.16.76.251 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.5/1.0/2.0 ms
root@munich:~ #
```

The Network configuration and routing are correct, and pinging the VPN server works.

> On Microsoft operating systems you will have to type `ping /t` for persistent pings, `ipconfig /all` for network data, and `route print` to display the routing table.

Checking interfaces, routing, and connectivity on the VPN servers

In our next step, we will have a close look at the network settings on the VPN servers. We will use the same tools, but the output will be a little more complex.

```
opensuse01:~ # ifconfig
eth0      Protokoll:Ethernet  Hardware Adresse 00:0C:29:13:EC:48
          inet Adresse:172.16.103.2  Bcast:172.16.103.255
Maske:255.255.255.0
          inet6 Adresse: fe80::20c:29ff:fe13:ec48/64
Gültigkeitsbereich:Verbindung
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500
Metric:1
```

```
            RX packets:2900 errors:0 dropped:0 overruns:0 frame:0
            TX packets:4790 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 Sendewarteschlangenlänge:1000
            RX bytes:759578 (741.7 Kb)  TX bytes:666545 (650.9 Kb)
            Interrupt:10 Basisadresse:0x1080
eth1      Protokoll:Ethernet  Hardware Adresse 00:0C:29:13:EC:52
            inet Adresse:172.16.76.251  Bcast:172.16.76.255
Maske:255.255.255.0
            inet6 Adresse: fe80::20c:29ff:fe13:ec52/64
Gültigkeitsbereich:Verbindung
            UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500
Metric:1
            RX packets:797 errors:0 dropped:0 overruns:0 frame:0
            TX packets:421 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 Sendewarteschlangenlänge:1000
            RX bytes:77682 (75.8 Kb)  TX bytes:42404 (41.4 Kb)
            Interrupt:9 Basisadresse:0x1400
lo        Protokoll:Lokale Schleife
            inet Adresse:127.0.0.1  Maske:255.0.0.0
            inet6 Adresse: ::1/128 Gültigkeitsbereich:Maschine
            UP LOOPBACK RUNNING  MTU:16436  Metric:1
            RX packets:109 errors:0 dropped:0 overruns:0 frame:0
            TX packets:109 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 Sendewarteschlangenlänge:0
            RX bytes:8380 (8.1 Kb)  TX bytes:8380 (8.1 Kb)
tunVPN0   Protokoll:UNSPEC  Hardware Adresse 00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00
            inet Adresse:10.179.10.2  P-z-P:10.179.10.1
Maske:255.255.255.255
            UP PUNKTZUPUNKT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
            RX packets:1337 errors:0 dropped:0 overruns:0 frame:0
            TX packets:1547 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 Sendewarteschlangenlänge:100
            RX bytes:470725 (459.6 Kb)  TX bytes:181397 (177.1 Kb)
opensuse01:~ #
```

This server has two network interface cards `eth0` and `eth1` (with two networks 172.16.103.0/24 and 172.16.76.0/24) in addition to the OpenVPN tunnel network `tunVPN0` with the network address `10.179.10.2` and the point-to-point partner's IP `10.179.10.1`. How about routing?

```
opensuse01:~ # route -n
Kernel IP routing table
Target          Router        Genmask        Flags  Metric  Ref    Use
Iface
10.179.10.1     0.0.0.0       255.255.255.255 UH     0       0      0
tunVPN0
172.16.103.0    0.0.0.0       255.255.255.0   U      0       0      0      eth0
172.16.76.0     0.0.0.0       255.255.255.0   U      0       0      0      eth1
```

```
192.168.250.0      10.179.10.1 255.255.255.0  UG   0    0    0
tunVPN0
127.0.0.0      0.0.0.0     255.0.0.0     U    0    0    0      lo
0.0.0.0      172.16.103.1  0.0.0.0      UG   0    0    0      eth0
opensuse01:~ #
```

Routing is a little more complicated here. We have two subnets connected to eth0 and eth1, and two entries for our tunnel. Everything to the virtual address 10.179.10.1 is routed through the interface tunVPN0, likewise traffic to the subnet 192.168.250.0/24, but this is routed through the gateway 10.179.10.1. Last but not least, the default gateway of this router has the IP 172.16.103.1. Obviously there is another network between this firewall and the Internet.

Let's now ping the point-to-point partner of this machine. We could see from the previous interface list that this machine has the virtual IP 10.179.10.2, and the VPN partner has the IP 10.179.10.1. If our tunnel is working it should be possible to ping through the tunnel.

```
opensuse01:~ # ping 10.179.10.1
PING 10.179.10.1 (10.179.10.1) 56(84) bytes of data.
64 bytes from 10.179.10.1: icmp_seq=1 ttl=64 time=77 ms
64 bytes from 10.179.10.1: icmp_seq=2 ttl=64 time=50 ms
64 bytes from 10.179.10.1: icmp_seq=3 ttl=64 time=42 ms
64 bytes from 10.179.10.1: icmp_seq=4 ttl=64 time=44 ms

--- 10.179.10.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3013ms
rtt min/avg/max/mdev = 1.425/1.535/1.770/0.141 ms
opensuse01:~ #
```

It's working! Please note that the time taken to answer a ping will be significantly longer through the tunnel than for a local or direct ping.

Now let's do the same tests the other way around. We will analyze the network and routing of the Sydney server and try to ping to Munich through the tunnel.

```
debian01:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:99:7B:CA
          inet addr:172.16.247.2  Bcast:172.16.247.255
Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7735 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11012 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:924335 (902.6 KiB)  TX bytes:1714169 (1.6 MiB)
          Interrupt:18 Base address:0x1080
```

```
eth1      Link encap:Ethernet  HWaddr 00:0C:29:99:7B:D4
          inet addr:192.168.250.251  Bcast:192.168.250.255
Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:490 errors:0 dropped:0 overruns:0 frame:0
          TX packets:468 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:47652 (46.5 KiB)  TX bytes:43728 (42.7 KiB)
          Interrupt:19 Base address:0x1400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

tunVPN0   Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00
          inet addr:10.179.10.1  P-t-P:10.179.10.2
Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:1849 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1489 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:206765 (201.9 KiB)  TX bytes:483493 (472.1 KiB)

debian01:~# route -n
Kernel IP routing table
Destination Gateway    Genmask          Flags   Metric  Ref  Use
Iface
10.179.10.2  0.0.0.0    255.255.255.255  UH    0    0  0  tunVPN0
172.16.247.0  0.0.0.0    255.255.255.0    U    0    0  0  eth0
172.16.76.0  10.179.10.2  255.255.255.0   UG    0    0  0  tunVPN0
192.168.250 .0   0.0.0.0    255.255.255.0    U    0    0  0  eth1
0.0.0.0    172.16.247.1 0.0.0.0      UG    0    0  0  eth0
debian01:~# ping 10.179.10.1
PING 10.179.10.1 (10.179.10.1) 56(84) bytes of data.
64 bytes from 10.179.10.1: icmp_seq=1 ttl=64 time=21 ms
64 bytes from 10.179.10.1: icmp_seq=2 ttl=64 time=69 ms
64 bytes from 10.179.10.1: icmp_seq=3 ttl=64 time=59 ms

--- 10.179.10.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.059/0.116/0.221/0.074 ms
debian01:~#
```

It worked! We have now made sure that:

- The VPN servers are reachable in their local networks
- The OpenVPN tunnel is up and running
- The OpenVPN tunnel is working in both directions

Let's now enter another level of testing. We will now test if the Sydney network is reachable from our VPN server in Munich—still using ICMP packets only. Furthermore, the program `traceroute` will help us follow the route the packets take.

```
opensuse01:~ # ping 192.168.250.128
PING 192.168.250.128 (192.168.250.128) 56(84) bytes of data.
64 bytes from 192.168.250.128: icmp_seq=1 ttl=63 time=19 ms
64 bytes from 192.168.250.128: icmp_seq=2 ttl=63 time=26 ms
64 bytes from 192.168.250.128: icmp_seq=3 ttl=63 time=57 ms

--- 192.168.250.128 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2009ms
rtt min/avg/max/mdev = 1.261/1.577/1.900/0.264 ms
opensuse01:~ # traceroute -n 192.168.250.128
traceroute to 192.168.250.128 (192.168.250.128), 30 hops max, 40 byte
packets
 1  10.179.10.1  1.874 ms   8.949 ms   20.241 ms
 2  192.168.250.128  24.911 ms   35.618 ms   40.988 ms
opensuse01:~ #
```

Again, pinging worked fine. This indicates correct routing on the Sydney side and on the Munich VPN server. The output of the program `traceroute` lists all servers the packets passed on their way to Sydney, they were *thrown* into the tunnel immediately and arrived at the VPN server in Sydney `10.179.10.1`, which passed them on to the local machine. Of course we can also 'traceroute' our packets that go the other way, provided that the administrator of the Debian server has installed `traceroute` (`apt-get install traceroute`).

> On Microsoft operating systems the command `tracert` offers the same functionality as `traceroute` on Linux.

Another very handy tool is 'My traceroute', or `mtr`, called with `mtr -n 192.168.250.128`, `mtr` keeps running `traceroute -n 192.168.250.128` command until you type `q` or *Ctrl+C*. The output is displayed in a clear table. With this command we can easily switch routing entries and view the effect interactively. It is included in all modern distributions, but by default only available for the root user.

```
                        My traceroute  [v0.69]
opensuse01 (0.0.0.0)(tos=0x0 psize=64 bitpattern=0x00) Fri Dec  2 17:33:29 2005
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                           Packets                 Pings
Host                      Loss%  Snt   Last   Avg  Best  Wrst StDev
1. 10.179.10.1             0.0%    9    1.3   1.4   1.2   2.0   0.3
2. 192.168.250.128         0.0%    8    1.8   2.3   1.6   3.2   0.6
```

Debugging with tcpdump and IPTraf

Another very handy tool to control traffic is `tcpdump`. As a network sniffer, `tcpdump` is often used by administrators or hackers to collect data exchanged on a network. `tcpdump` prints all traffic that passes the interface given as a parameter. The following example demonstrates the use of `tcpdump`. When called with the options `-n` and `-i eth1`, `tcpdump` will listen on interface `eth1` and give a numeric output (without resolving DNS):

```
debian01:~# tcpdump -n -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
21:00:16.640142 IP 192.168.250.128 > 172.16.76.128: ICMP echo request,
id
55298, seq 0, length 64
21:00:16.648116 IP 172.16.76.128 > 192.168.250.128: ICMP echo reply,
id 55298, seq 0, length 64
21:00:17.678429 IP 192.168.250.128 > 172.16.76.128: ICMP echo request,
id 55298, seq 256, length 64
21:00:17.680701 IP 172.16.76.128 > 192.168.250.128: ICMP echo reply,
id 55298, seq 256, length 64
21:00:18.668565 IP 192.168.250.128 > 172.16.76.128: ICMP echo request,
id 55298, seq 512, length 64
21:00:18.670722 IP 172.16.76.128 > 192.168.250.128: ICMP echo reply,
id 55298, seq 512, length 64
21:00:19.688618 IP 192.168.250.128 > 172.16.76.128: ICMP echo request,
id 55298, seq 768, length 64
21:00:19.690836 IP 172.16.76.128 > 192.168.250.128: ICMP echo reply,
id 55298, seq 768, length 64
```

As we can see, there were four `ICMP echo` request messages sent from `192.168.250.128` to `172.16.76.128`. All of them were answered by the machine `172.16.76.128` with the appropriate 'echo reply' message.

Now we can use `tcpdump` on every machine in the chain of routers between the two clients in order to track the ICMP packets. For example, if a firewall is blocking the ICMP messages, then no PC *behind* this firewall will receive any of the requests or replies, whereas the machines *before* the firewall will.

```
debian01:~# tcpdump -ni tunVPN0
tcpdump: WARNING: arptype 65534 not supported by libpcap - falling
back to cooked socket
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on tunVPN0, link-type LINUX_SLL (Linux cooked), capture size
96 bytes
21:07:53.800707 IP 172.16.76.128 > 192.168.250.128: ICMP echo request,
id 19971, seq 9472, length 64
21:07:53.801608 IP 192.168.250.128 > 172.16.76.128: ICMP echo reply,
id 19971, seq 9472, length 64
21:07:54.799266 IP 172.16.76.128 > 192.168.250.128: ICMP echo request,
id 19971, seq 9728, length 64
21:07:54.800531 IP 192.168.250.128 > 172.16.76.128: ICMP echo reply,
id 19971, seq 9728, length 64
21:07:55.800302 IP 172.16.76.128 > 192.168.250.128: ICMP echo request,
id 19971, seq 9984, length 64
21:07:55.801296 IP 192.168.250.128 > 172.16.76.128: ICMP echo reply,
id 19971, seq 9984, length 64
21:07:56.752248 IP 172.16.76.128 > 192.168.250.128: ICMP echo request,
id 19971, seq 10240, length 64
21:07:56.752876 IP 192.168.250.128 > 172.16.76.128: ICMP echo reply,
id 19971, seq 10240, length 64

8 packets captured
16 packets received by filter
0 packets dropped by kernel
debian01:~#
```

You see, `tcpdump` also runs on the tunnel interfaces, but some features won't work with TUN or TAP interfaces. Also because the network interface will run in promiscuous mode, `tcpdump` will need root privileges. Furthermore, the information returned will be scarce in most switched networks, where only local packets can be displayed.

Another helpful tool is IPTraf (on Debian it is installed with `apt-get install iptraf`). IPTraf collects and displays packets and statistical data on selected interfaces. IPTraf comes with many options, but we will only focus on its list view.

Enter `iptraf` and hit return four times. You will get a window as depicted in the following screenshot:
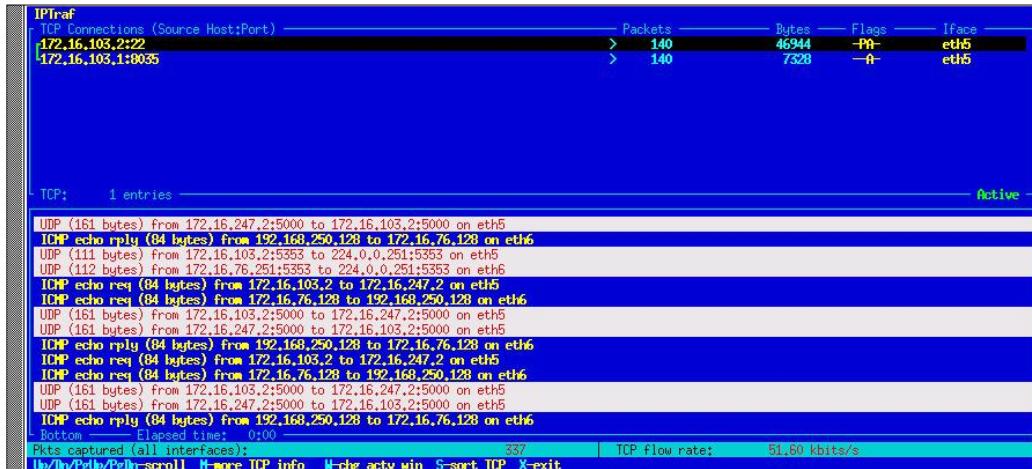
In the upper half of the window, **TCP** connections are displayed. **UDP**, **ICMP**, and other connections can be found in the lower half. In the preceding example, we see an SSH session (from which IPTraf was started), ICMP packages between the Sydney and Munich client PCs, and the UDP packages encapsulating these ICMP packages.

Hit *X* twice and *Enter* once to quit IPTraf.

Using OpenVPN protocol and status files for debugging

A very convenient method to watch tunnel traffic is setting the verbosity of OpenVPN to the fifth level. This is simply done with the entry `verb 5` in its configuration file. The following output shows an excerpt of OpenVPN's protocol file (as specified in the OpenVPN configuration file):

```
Fri Dec  9 21:05:15 2005 us=51912 Data Channel Encrypt: Cipher 'AES-
256-CBC' initialized with 256 bit key
Fri Dec  9 21:05:15 2005 us=51944 Data Channel Encrypt: Using 160 bit
message hash 'SHA1' for HMAC authentication
Fri Dec  9 21:05:15 2005 us=51962 Data Channel Decrypt: Cipher 'AES-
256-CBC' initialized with 256 bit key
Fri Dec  9 21:05:15 2005 us=52033 Data Channel Decrypt: Using 160 bit
message hash 'SHA1' for HMAC authentication
```

```
Fri Dec  9 21:05:15 2005 us=131924 Control Channel: TLSv1, cipher
TLSv1/SSLv3 DHE-RSA-AES256-SHA, 2048 bit RSA
WRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWR
wrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwr
WRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWR
wrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrWRwrW (...)
```

In the last lines, we find the detailed statistics of all tunnel traffic. Upper cased letters stand for TCP or UDP datagrams on the real interface, encapsulating OpenVPN traffic, and lower case letters indicate traffic on the TUN/TAP interface. Unsurprisingly, `r` is for read and `w` is for write. Thus a successful ping command through the tunnel will always cause an entry such as `WRwr` or vice versa.

Another file that our sample setup writes information to is the `status` file. Depending on the time period given as a parameter, OpenVPN will update the information in this file on a regular basis. In the example the file was `/var/log/openvpn/feilner-it.status`, the command `cat` can show us the content of this file.

```
debian01:~# cat /var/log/openvpn/feilner-it.status
OpenVPN STATISTICS
Updated,Fri Dec  9 21:26:53 2005
TUN/TAP read bytes,1102504
TUN/TAP write bytes,806453
TCP/UDP read bytes,1302857
TCP/UDP write bytes,1588558
Auth read bytes,808809
pre-compress bytes,55193
post-compress bytes,53110
pre-decompress bytes,1449
post-decompress bytes,2076
END
debian01:~#
```

This gives us detailed statistical data. If you run into problems with OpenVPN, it may be a good idea to check this file to find out if the values make sense, or if there is either too much or missing traffic on either side, for example, if it gets lost or the routing is wrong.

Depending on your system and logging setup, there may also be entries in your system protocol, like those here on this SuSE system:

```
opensuse01:~ # tail /var/log/messages
Dec  2 17:50:09 opensuse01 openvpn[11661]: Local Options String:
'V4,dev-type tun,link-mtu 1545,tun-mtu 1500,proto UDPv4,ifconfig
10.179.11.1 10.179.11.2,comp-lzo,cipher BF-CBC,auth SHA1,keysize
128,secret'
Dec  2 17:50:09 opensuse01 openvpn[11661]: Expected Remote
```

```
Options String: 'V4,dev-type tun,link-mtu 1545,tun-mtu 1500,proto
UDPv4,ifconfig 10.179.11.2 10.179.11.1,comp-lzo,cipher BF-CBC,auth
SHA1,keysize 128,secret'
Dec  2 17:50:09 opensuse01 openvpn[11661]: Local Options hash
(VER=V4): '59c313f6'
Dec  2 17:50:09 opensuse01 openvpn[11661]: Expected Remote Options
hash (VER=V4): '36b1f115'
Dec  2 17:50:09 opensuse01 openvpn[11661]: Output Traffic Shaping
initialized at 20000 bytes per second
Dec  2 17:50:09 opensuse01 openvpn[11674]: Socket Buffers: R=[113664-
>131072] S=[113664->131072]
Dec  2 17:50:09 opensuse01 openvpn[11674]: UDPv4 link local (bound):
[undef]:5001
Dec  2 17:50:09 opensuse01 openvpn[11674]: UDPv4 link remote:
172.16.247.2:5001
```

This shows that another VPN tunnel has been created, OpenVPN is listening on UDP port 5001.

Scanning servers with Nmap

Nmap is a port scanner that can be used to determine whether a UDP or TCP port on a machine is open and whether there is a server process accepting connections.

> In some countries, including Germany, software like nmap may be considered as hacker software and thereby its use and possession may be regulated by local laws. Some courts in Germany are of the opinion that an administrator using a port scanner to secure his server is violating recent laws. It's a mad mad world, isn't it?

Nmap can also find out if a firewall is protecting the machine scanned, and Nmap can scan whole networks. Let's scan the local client PC (which is obviously not protected by a firewall.).

```
opensuse01:~ # nmap 172.16.76.128

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-12-02
18:02 CET
Interesting ports on localhost (172.16.76.128):
(The 1661 ports scanned but not shown below are in state: closed)
PORT   STATE SERVICE
22/tcp open  ssh
68/tcp open  dhcpclient
MAC Address: 00:0C:29:21:07:FC

Nmap finished: 1 IP address (1 host up) scanned in 1.773 seconds
```

There are two ports open on this system, all the 1661 scanned ports and other scanned ports are closed. If there were a firewall on this system, scanning would not be that easy because most firewalls detect scans and can prevent them. But there are many options to Nmap, including stealth scans, altering sender IPs, and many more—the manual page is really good.

We will now scan one of our OpenVPN servers to find out if our VPN port (5000) can be reached. The command `nmap -sU <IP> -p <Port>` will make Nmap scan only if the UDP port on the machine with the given IP address is open.

```
opensuse01:~ # nmap -sU 172.16.247.2 -p 5000

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-12-02
18:06 CET
Note: Host seems down. If it is really up, but blocking our ping
probes, try -P0
Nmap finished: 1 IP address (0 hosts up) scanned in 2.067 seconds
opensuse01:~ # nmap -P0 -sU 172.16.247.2 -p 5000

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-12-02
18:06 CET
Interesting ports on debian01.feilner-it.home (172.16.247.2):
PORT       STATE          SERVICE
5000/udp open|filtered UPnP

Nmap finished: 1 IP address (1 host up) scanned in 2.039 seconds
opensuse01:~ #
```

You saw how our Shorewall firewall did not reveal information about the port when we scanned it on the first attempt. However, Nmap already gave us a hint—add the parameter `-P0` to act even more stealthily. With this option, Nmap does not ping the hosts it scans before scanning them. Some firewalls recognize this as typical behavior of port scanners and block it. The second try, however, revealed that the UDP port 5000 is filtered (by a firewall). This means that firewall rules may be protecting and limiting access to this port, but it may nevertheless be open.

> On Windows, the program 'Angry IP Scanner' will probably be your first choice for scanning.

Monitoring tools

There are many tools that provide detailed statistics on network interfaces. Two very easily installed monitoring tools with great functions are **ntop** and **Munin**.

ntop

ntop monitors a network and may in some states be illegal because it creates detailed records of connections between IP addresses. Furthermore, it offers a nice browser GUI and does not need a running web server.The `ntop` installs easily on Debian.

Enter `apt-get install ntop` and choose the interface you want to monitor. After software installation, type `ntop -A` and enter an administrator password for ntop's admin account. Now type `/etc/init.d/ntop start` and point a browser to the `http://IP:3000` of this system (ntop is running on port 3000). You will get a feature-rich window with a growing amount of information, especially if ntop has been for running some time:

ntop offers several options. We can save the data to a database, access to a database can be secured and monitored, interfaces can be switched online, and many more possibilities.

Munin

Another helpful statistic tool is Munin. Munin consists of a client and a server process that collect data that is provided from an almost arbitrary source on Linux (or even Windows) systems. The following example shows the standard Munin interface after installation as documented on `http://munin.sf.net`. Unfortunately, Munin needs a web server like Apache, but apart from this, the installation is very easy. Munin is configured from files in `/etc/munin/`, and makes use of a great number of plugins. Even more can be downloaded.
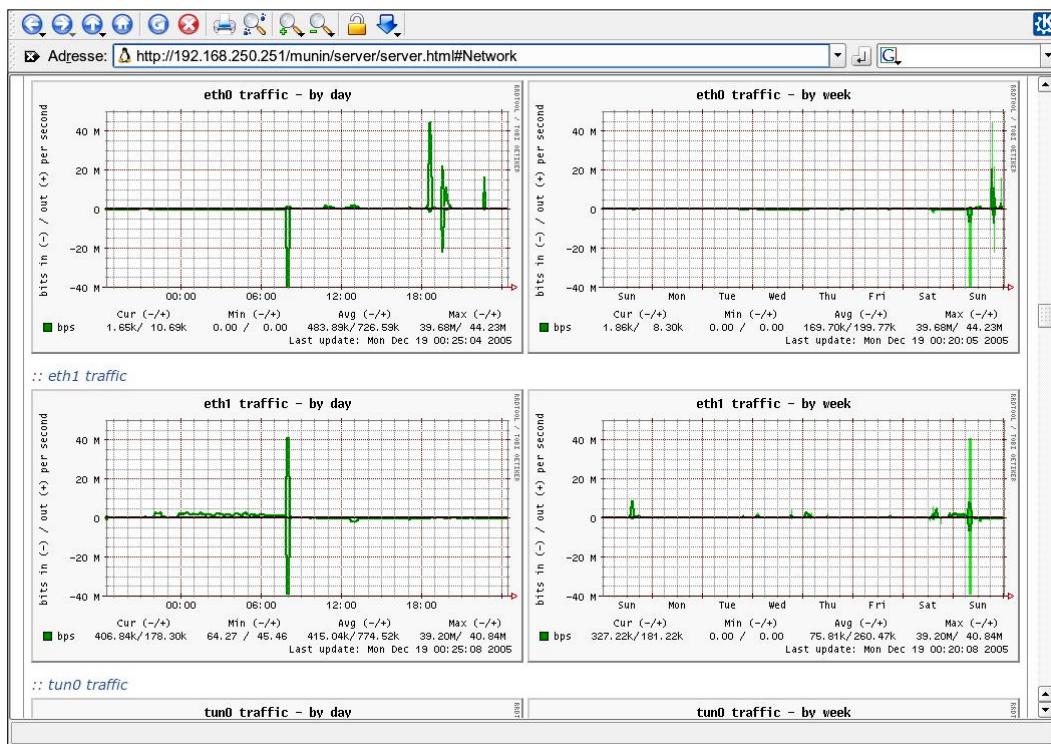
As there are only a few requirements for a Munin plugin, we can easily create our own OpenVPN monitoring plugin. Such a plugin must be executable, and should return data in the format of:

```
router:/usr/share/munin/plugins # /etc/munin/plugins/if_eth0
down.value 1777836059
up.value 94615124
router:/usr/share/munin/plugins #
```

On `http://rodolphe.quiedeville.org/hack/openvpn` there is a simple plugin that reports the number of users connected to an OpenVPN server. I leave it up to you to imagine the possibilities of such plugins when combined with `samba`, `iptables`, OpenVPN, and more. Just think of the OpenVPN `status` file and the information it provides.

Nagios

There is an abundance of networking tools for monitoring, sniffing, and scanning. Two of my favorites are Cacti and Nagios. Cacti is a monitoring tool similar to Munin, but it seems more powerful. Nagios is a tool designed to monitor machines and services.

With Nagios you can not only determine if a server is still answering pings, but can also check for services by accessing them (as an example, the `samba` or HTTP protocols) and trigger actions when the service is not available. You can have your Nagios machine send you an SMS if your OpenVPN tunnel is down, or if the management interface is not responding.

A valid configuration file for this complex piece of software could be:

```
# /usr/local/nagios/etc/vpnendpoint
# Uses Ping check if host is available
# Define the host to check
define host{
        use             generic-switch
        host_name       vpn01
        alias           Netherland - OpenVPN Endpoint
        address         10.99.1.3
        hostgroups      vpns
        }
# Define a hostgroup
define hostgroup{
        hostgroup_name  vpns
        alias           VPN Endpoints
        }
# Define a Nagios service (add your own services like SSH or SMB here)
define service{
        use                     generic-service
        host_name               vpn01
        service_description     PING
        check_command           check_ping!200.0,20%!800.0,80%
        normal_check_interval   5
        retry_check_interval    1
        }
```

Three paragraphs, and that's it. While the first defines a host to check and its name in the Nagios frontend, the second sets up a host group, and the final a service. For a VPN a simple ping check of the VPN endpoints should do, but for checking if the routing to your servers still works, it may make more sense to check the availability of, for example, the samba services of a file server for Windows behind the other VPN endpoint.

This is what the configuration in the Nagios web frontend looks like:

OpenVPNgraph

Finally, one last small but efficient tool to look at. Ralf Hildebrandt has rewritten David Schweickert's `mailgraph` and adapted it to OpenVPN's needs. OpenVPNgraph can be downloaded from `http://computerbeschimpfung.de/openvpngraph.tar.gz`, a `README` file with three simple steps for installation is included. Just make sure that `rrdtools` is installed on your system and after some hours, days, weeks, or months you will have statistics like those in the next image.

The following screenshot shows a big OpenVPN server's daily statistics, the first curve shows a maximum of 75 users connected simultaneously, with only 2 failed and one double login. With this tool the admin can show that most VPN clients connect around noon and in the evening, while at night the numbers go down significantly.

The second diagram reveals the amount of received and sent packages over the VPN interface: More than 7000 KByte/s are possible, as the peak shows, but during the day a maximum of just 200 Kbyte/s or so are needed, the average is about 54 Kbyte/s. Pretty few, isn't it?

By scrolling further down in OpenVPNgraph, this tiny tool (a big thanks to Ralf Hildebrandt!) reveals weekly, monthly, and yearly statistics. The longer it is running, the more interesting the diagrams become.
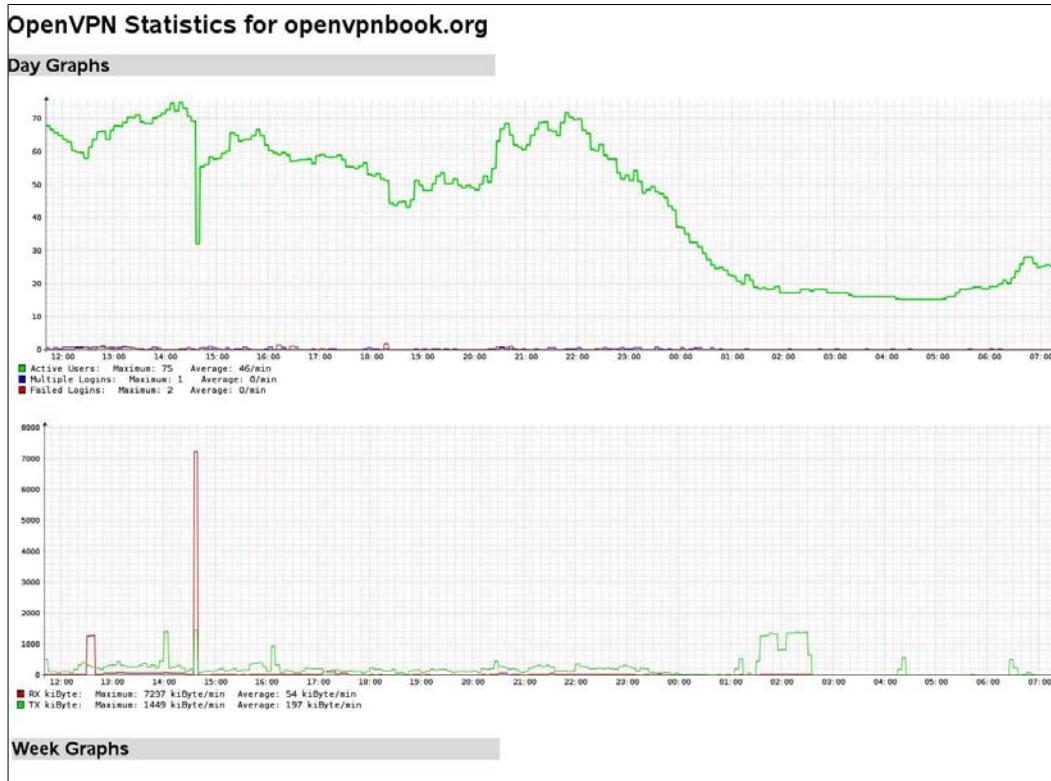
Summary

In this chapter we have learned how to check OpenVPN and networking setup step-by-step using standard Linux tools and evaluating their output. With tools such as `ifconfig`, `ping`, `traceroute`, and `mtr`, we could analyze the flow of datagrams between VPN servers and connected networks. Programs like `tcpdump`, `iptraf`, `ntop`, `munin`, `nagios`, and `OpenVPNgraph` give us detailed information about the current state, traffic, or statistical breakdowns of it. The first place start troubleshooting should always be the `log` or `status` file of OpenVPN itself—especially at a higher level of verbosity.

Internet Resources and More

Chapter 1

A concise but easy explanation of the OSI model can be found in Wikipedia:

`http://en.wikipedia.org/wiki/OSI_model`

Available OSI protocols:

`http://en.wikipedia.org/wiki/OSI_protocols`

The Internet Protocol (IP):

`http://en.wikipedia.org/wiki/Internet_Protocol`

A basic introduction to IPsec:

`http://en.wikipedia.org/wiki/IPsec`

SSH-Tunneling:

`http://www.ssh.com/support/documentation/online/ssh/winhelp/32/`
`Tunneling_Explained.html.`

A very good overview on Layer 2 Forwarding (L2F) can be found here:

`http://www.javvin.com/protocolL2F.html.`

The Internet Engineering Task Force details can be found at:

`http://www.ietf.org.`

Read the IPsec article in Wikipedia:

`http://en.wikipedia.org/wiki/IPsec.`

The Linux IPsec How to:

`http://www.ipsec-howto.org/t1.html.`

An example for a TLS/SSL web-based SSL/TLS VPN solution:

`http://sourceforge.net/projects/sslexplorer/.`

`http://3sp.com/showSslExplorer.do.`

`http://en.wikipedia.org/wiki/Datagram_Transport_Layer_Security.`

Chapter 2

Handbook of Information Security Management:

`http://www.cccure.org/Documents/HISM/ewtoc.html.`

IT Baseline Protection as published by the German BSI (but in English):

`http://www.bsi.bund.de/english/gshb/index.htm.`

`http://www.bsi.bund.de/english/.`

The IT-Sec Handbook—concise security hints:

`http://www.cccure.org/Documents/HISM/ewtoc.html.`

Wikipedia articles are good to start with and contain lots of interesting links:

`http://en.wikipedia.org/wiki/Symmetric_encryption.`

`http://en.wikipedia.org/wiki/Asymmetric_encryption.`

`http://en.wikipedia.org/wiki/Cryptography.`

`http://en.wikipedia.org/wiki/Secure_Sockets_Layer.`

`http://en.wikipedia.org/wiki/Public_key_certificate.`

OpenVPN and the SSL revolution:

`http://www.sans.org/reading_room/whitepapers/vpns/1459.php`

OpenVPN security:

`http://openvpn.net/index.php/documentation/security-overview.html`

VPNs Illustrated: Tunnels, VPNs, and Ipsec:

`http://fengnet.com/book/VPNs%20Illustrated%20Tunnels%20%20VPNsand%20`
`IPsec/ch08lev1sec5.html`

Windows Security and SSL:

`http://www.windowsecurity.com/articles/Secure_Socket_Layer.html.`

The TLS protocol as specified by the IETF:

`http://www.ietf.org/rfc/rfc2246.txt`

Openssl Homepage:

`http://www.openssl.org`

Chapter 3

An interview with James Yonan on Linuxsecurity.com:

`http://www.linuxsecurity.com/content/view/117363/49/.`

Community: The project web site of OpenVPN

`http://openvpn.net/.`

OpenVPN changelog and release notes:

`http://openvpn.net/changelog.html.`

`http://openvpn.net/relnotes.html.`

Shorewall Firewall:

`http://www.shorewall.net/OPENVPN.html.`

`http://home.arcor.de/u.altinkaynak/openvpn.html.`

OpenVPN forum:

`http://www.vpnforum.de/.`

The mailing lists:

`http://openvpn.net/mail.html.`

The SSL/TLS Cryptographic Libraries' web site:

`http://www.openssl.org/.`

The website of the Transport Layer Security Charter by the TLS Working Group:

`http://www.ietf.org/html.charters/tls-charter.html.`

The universal TUN/TAP driver:

`http://vtun.sourceforge.net/tun/.`

Installing the OpenVPN LZO project:

`http://www.oberhumer.com/opensource/lzo/.`

Chapter 4

OpenVPN Windows client:

`http://openvpn.net/index.php/downloads.html`

Graphical User Interface for the Windows client:

`http://openvpn.se/`

Tunnelblick—Client for Mac:

`http://www.tunnelblick.net`

Tunnelblick—Readme:

`http://www.tunnelblick.net/README.txt`

Detailed installation instructions for Mac OS 10.3:

`http://www.helsinki.fi/atk/english/hy-ppp/hy-vpn/hy-vpn-mac.html`

Chapter 5

TUN/TAP devices for Windows:

`http://vtun.sourceforge.net/tun/`

OpenSSL libraries for Linux and Unix systems:

`http://www.openssl.org/`

LZO real-time compression library packages:

`http://openvpn.net/download.html`

LZO real-time compression library—Official web site:

`http://www.oberhumer.com/opensource/lzo`

Source code for OpenVPN, Developer Versions:

`http://openvpn.net/download.html`

Concurrent Versions System (CVS) repository (unstable):

`http://sourceforge.net/cvs/?group_id=48978`

Installing OpenVPN on Red Hat Fedora using yum: Yellow dog Updater, Modified (yum)

`http://linux.duke.edu/projects/yum/`

Fedora web site:

`http://www.fedorafaq.org/`

Yum—suitable configuration:

`http://www.fedorafaq.org/samples/yum.conf`

OpenVPN packages for SuSE:

`ftp://ftp.suse.com/`

Fedora RPMs:

`http://dag.wieers.com/packages/openvpn/`

Installing OpenVPN on FreeBSD:

```
http://www.freebsd.org
```

Ports for FreeBSD:

```
http://www.freebsd.org/ports/index.html
```

Chapter 6

Building Your Own RPM File from the OpenVPN Source Code, configure one of your HTTP or FTP servers to act as a Debian or a SuSE repository:

```
http://www.debian.org/doc/manuals/repository-howto/repository-howto
(Debian)
```

```
http://www.charlescurley.com/yum/repository.html (Red Hat)
```

Enabling Linux Kernel Support for TUN/TAP devices—the process of kernel compilation:

```
http://tldp.org/LDP/tlk/tlk.html
```

Linux kernel source code:

```
http://www.kernel.org
```

Chapter 7

WinSCP website:

```
http://winscp.net/
```

Chapter 8

Creating the Diffie-Hellman Key:

```
http://www.rsasecurity.com/rsalabs/node.asp?id=2248
```

Xntp website:

```
http://www.eecis.udel.edu/~ntp/
```

Chapter 10

OpenVPN Auth-LDAP plugin:

`http://code.google.com/p/openvpn-auth-ldap/`

German OpenVPN Auth-LDAP How to:

`http://www.indato.ch/openvpn/openvpn.html`

Radius How to:

`http://forum.openvpn.eu/viewtopic.php?t=2116`

`http://www.howtoforge.com/openvpn_wikid_strong_authentication`

Pop-Auth How to:

`http://www.wenzk.net/bbs/thread-221-1-1.html`

SQLite How to:

`http://www.vpnforum.de/openvpn-forum/viewtopic.php?t=3178`

MySQL How to:

`http://www.vpnforum.de/openvpn-forum/viewtopic.php?t=3591`

USB How to:

`http://usbauth.delta-xi.net/doku.php`

Samba How to:

`http://www.vpnforum.de/openvpn-forum/viewtopic.php?t=1748`

Via Radius and Wikid-Server How to:

`http://www.howtoforge.com/openvpn_wikid_strong_authentication`

Aladdin tokens:

`http://www.aladdin.de/how/resellers.aspx`

Webmin:

`http://www.webmin.com`

Shorewall:

`http://www.shorewall.net`

Chapter 11

XCA on Sourceforge:

`http://xca.sourceforge.net/`

TinyCA website:

`http://tinyca.sm-zone.net`

TinyCA for a MAC:

`http://tinyca2.darwinports.com/`

OpenSSL web site:

`http://www.openssl.org`

German tutorial for OpenSSL:

`http://www.online-tutorials.net/security/openssl-tutorial/`
`tutorials-t-69-207.html#beispiel-openvpn`

A browser-based PKI management suite from OpenCA PKI Research Labs:

`https://www.openca.org`

OpenTrust PKI:

`http://www.opentrust.com/content/view/119/111`

My Certificate Wizard:

`http://mycert.sandbox.cz`

Chapter 12

OpenVPN Server administration: Webmin and its OpenVPN plugin

`http://www.webmin.com/`

Client GUIs for Linux: KVpnc

`http://home.gna.org/kvpnc/en/index.html`

Gadmin—OpenVPN client:

```
http://gadmintools.flippedweb.com/index.php?option=com_content&task=v
iew&id=58&Itemid=40
```

Chapter 14

Anonymous and uncensored Internet Access: Dynamic DNS Services:

```
http://www.dyndns.org
```

```
http://www.no-ip.com
```

Detailed article on linux-magazine.com:

```
http://www.linux-magazine.com/online/features/
set_up_openvpn_in_four_steps
```

OpenVPN on Windows Mobile (OpenVPN for Pocket PC):

```
http://ovpnppc.ziggurat29.com/ovpnppc-main.htm
```

Embedded Linux—Maemo Nokia N800, 810, N900 Client:

```
http://www.maemo.org
```

Chapter 15

tcpdump:

```
http://www.tcpdump.org/
```

Iptraf:

```
http://iptraf.seul.org/
```

Nmap:

```
http://nmap.org/
```

Angry IP Scanner:

```
http://www.angryziber.com
```

ntop:

```
http://www.ntop.org/
```

munin:

```
http://munin.sf.net
```

OpenVPN Plugin for munin:

```
http://rodolphe.quiedeville.org/hack/openvpn
```

Abbreviations used

Abbreviation	Full Description
VPN	Virtual Private Network
CRM	Customer Relationship Management
ERP	Enterprise Resource Planning
RAS	Remote Access Server
NIC	Network Interface Card
OSI	Open Systems Interconnection
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTP	Hypertext Transfer Protocol
FTP	File Transfer Protocol
GRE	General Routing Encapsulation
RFC	Request For Comments
IPX	Internetwork Packet Exchange
PPP	Point-To-Point Protocol
PPTP	Point-To-Point-Tunneling Protocol
L2F	Layer 2 Forwarding
L2TP	Layer 2 Tunneling Protocol
L2sec	Layer 2 Security Protoocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
IETF	Internet Engineering Task Force
IPsec	Internet Protocol Security

Index

Thank you for buying
Learning OpenVPN 2.0.9

Packt Open Source Project Royalties

When we sell a book written on an Open Source project, we pay a royalty directly to that project. Therefore by purchasing Learning OpenVPN 2.0.9, Packt will have given some of the money received to the OpenVPN project.

In the long term, we see ourselves and you — customers and readers of our books — as part of the Open Source ecosystem, providing sustainable revenue for the projects we publish on. Our aim at Packt is to establish publishing royalties as an essential part of the service and support a business model that sustains Open Source.

If you're working with an Open Source project that you would like us to publish on, and subsequently pay royalties to, please get in touch with us.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

About Packt Publishing

Packt, pronounced 'packed', published its first book "Mastering phpMyAdmin for Effective MySQL Management" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.PacktPub.com.

PACKT PUBLISHING
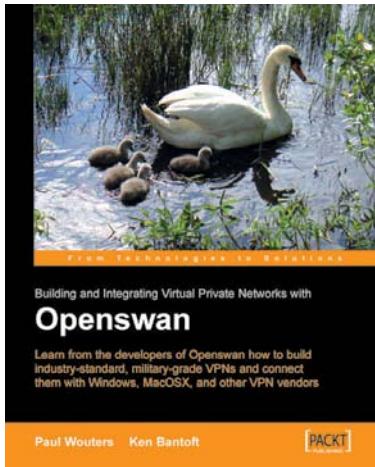
Openswan: Building and Integrating Virtual Private Networks

ISBN: 978-1-904811-25-1   Paperback: 360 pages

Learn from the developers of Openswan how to build industry standard, military grade VPNs and connect them with Windows, MacOSX, and other VPN vendors

1. Learn everything you need to know about Openswan from its core developers

2. Build VPNs that interoperate with Windows, MacOS, and other network vendors

3. Build your own secure hotspots

SSL VPN : Understanding, evaluating and planning secure, web-based remote access

ISBN: 978-1-904811-07-7   Paperback: 212 pages

A comprehensive overview of SSL VPN technologies and design strategies

1. Understand how SSL VPN technology works

2. Evaluate how SSL VPN could fit into your organisation?s security strategy

3. Practical advice on educating users, integrating legacy systems, and eliminating security loopholes

Please check **www.PacktPub.com** for information on our titles
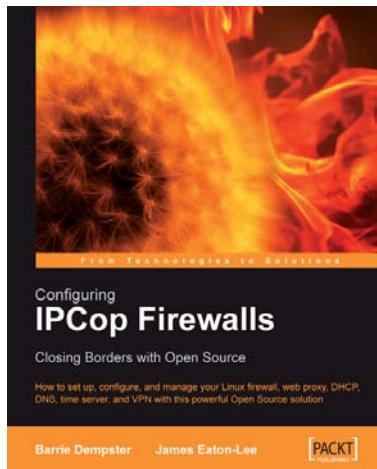
Configuring IPCop Firewalls: Closing Borders with Open Source

ISBN: 978-1-904811-36-7   Paperback: 244 pages

How to setup, configure and manage your Linux firewall, web proxy, DHCP, DNS, time server, and VPN with this powerful Open Source solution

1. Learn how to install, configure, and set up IPCop on your Linux servers

2. Use IPCop as a web proxy, DHCP, DNS, time server, and VPN

3. Advanced add-on management
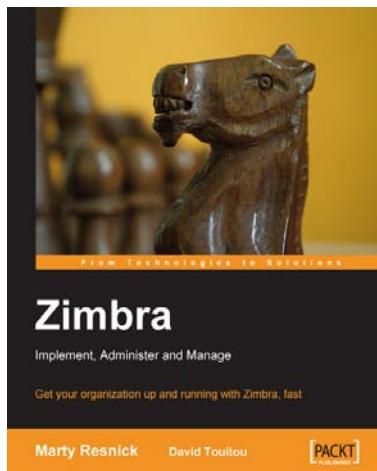
Zimbra: Implement, Administer and Manage

ISBN: 978-1-847192-08-0   Paperback: 220 pages

Get your organization up and running with Zimbra, fast

1. Get your organization up and running with Zimbra, fast

2. Administer the Zimbra server and work with the Zimbra web client

3. Protect your Zimbra installation from hackers, spammers, and viruses

Please check **www.PacktPub.com** for information on our titles