

5 OpenVPN: Die Open-Source-Alternative

Nachdem in den bisherigen Abschnitten recht allgemein verschiedene Eigenschaften von VPNs erläutert wurden, wird im Rest des Buches ausschließlich OpenVPN behandelt. Die Leserin und der Leser darf auf die folgenden Themen gespannt sein:

- ❑ Der Beginn ist ein kurzer Abstecher in die Geschichte von OpenVPN.
- ❑ Anschließend werden grundlegende Bausteine dieser Software erläutert.
- ❑ Die Installation auf den verschiedenen Betriebssystemen soll natürlich keineswegs »unterschlagen« werden, bevor es an die Details geht
- ❑ und der erste Tunnel aufgebaut wird.
- ❑ Die Konfiguration des Tunnels wird danach durch einige interessante Optionen abgerundet,
- ❑ Fallbeispiele,
- ❑ Details zur Authentisierung,
- ❑ des Managements und
- ❑ der Programmierung
- ❑ und natürlich ein paar Beispiele mit Fehlermeldungen und deren Ursachen werden ebenfalls erläutert.

In diesem Sinne: Viel Vergnügen!

5.1 Ein Blick zurück zur Motivation

In einem Interview mit LinuxSecurity.Com [31] hat James Yonan einige interessante Hintergründe über die Entwicklung von OpenVPN und über seine Motivation berichtet. Die Idee zu OpenVPN kam ihm im Laufe seiner beruflichen Tätigkeit um die Jahrtausendwende, als er damit beauftragt wurde, die Verantwortung für ein System zu übernehmen, welches rund um die Uhr verfügbar

sein sollte. Dabei war es von seinem Auftraggeber nicht gefordert, dass er rund um die Uhr präsent sein musste. Allerdings hatte er sicherzustellen, dass er von jedem beliebigen Ort der Welt seine Arbeit erledigen konnte.

Im Rahmen dieser »Einschränkung« (oder Freiheit, je nach Standpunkt) nutzte James die Möglichkeiten bestehender Internettechnologien, um seine Aufgaben aus der Ferne erfüllen zu können.

Er selbst gibt zu, dass er dabei mehr und mehr Interesse an Remote-Access-Technologien bekam, die nicht nur den geforderten Sicherheitsrichtlinien Genüge taten, sondern auch effizient und effektiv waren (und zwar ohne großen Aufwand!). Dies wurde ihm umso wichtiger, je mehr er bemerkte, dass Verbindungen durchaus »gekapert« werden können.

Die Entwicklung von FreeS/WAN wurde mit Version 2.06 im April 2004 eingestellt. Die Projekte Openswan [19] und strongSwan [25] haben die Nachfolge angetreten.

Zu dieser Zeit war das Thema »VPN mit Linux« in zwei Lager gespalten: Das eine Lager war sehr auf die Sicherheit per se fixiert, woraus sich sehr bekannte Softwarepakete wie beispielsweise FreeS/WAN [9], SSH oder auch SSL/TLS entwickelten. Die Lösungen dieses Lagers hatten naturgemäß einen hohen Anspruch an die Korrektheit ihrer Lösung und erkaufte sich diesen teilweise mit doch recht komplexen Konfigurationsaufgaben, die ein Benutzer oder Administrator zu erledigen hatte. Und wenn alles erfolgreich verlief, dann bestand durchaus die Möglichkeit, mit kommerziellen VPN-Geräten auf der Basis von IPsec einen verschlüsselten Tunnel aufbauen zu können.

Das andere Lager kümmerte sich nicht um den IPsec-Standard und fokussierte sein Augenmerk mehr auf grundlegende Funktionen, die VPNs bereitstellen. Auf leichte Erlern- und Nutzbarkeit wurde hierbei besonders Wert gelegt. Es entwickelten sich, trotz des hohen Preises der Inkompatibilität zu IPsec, zahlreiche Lösungen. In einigen wurden bahnbrechende Entwicklungen geschaffen, ohne die OpenVPN heute nicht funktionieren könnte: Die virtuelle Netzwerkschnittstelle ist hierfür ein gutes Beispiel. Auf dieser Basis war man in der Lage, portable Software zu entwickeln, die VPNs als Benutzerprozesse implementierte und keinen Eingriff in das Betriebssystem erforderte.

James Yonan fragte sich zu der Zeit, ob es nicht möglich sei, Technologien aus beiden Lagern zu verknüpfen, wie beispielsweise SSL/TLS und virtuelle Netzwerkschnittstellen. Und damit war die Idee von OpenVPN geboren.

OpenVPN entwickelte sich in der Folgezeit sehr schnell und es bildete sich eine wachsende Gemeinschaft von Benutzern und Entwicklern. Wer sich heute die Datei `ChangeLog` aus dem OpenVPN-

Paket betrachtet, dem werden sicher einige Meilensteine der Entwicklung auffallen, wie zum Beispiel:

- ❑ 13. Mai 2001 *Version 0.90* als erste veröffentlichte Version arbeitete mit UDP/IP, konnte verschlüsseln und mit SHA-1 HMAC signieren.
- ❑ 23. März 2002 *Version 1.0* unterstützte bereits TLS-basierte Authentisierung und dynamischen Schlüsselaustausch.
- ❑ 9. April 2002 *Version 1.1.0* gab es nun auch für OpenBSD. Ferner wurde nun OpenSSL als Standardbibliothek verwendet.
- ❑ 22. Mai 2002 *Version 1.2* war ebenfalls für Solaris verfügbar und unterstützte den Einsatz von Multithreading, was insbesondere bei der Berechnung neuer Schlüsselpaare im Hintergrund hilfreich ist.
- ❑ 23. Oktober 2002 *Version 1.3.2* unterstützte NetBSD und konnte mit Linux auch IPv6 übertragen.
- ❑ 15. Juli 2003 *Version 1.4.2* startete den Beginn der Unterstützung für Windows-Systeme.
- ❑ 24. Juli 2003 *Version 1.5-beta1* war für Windows XP und Windows 2000 verfügbar.
- ❑ 3. November 2003 *Version 1.5-beta13* hatte u.a. Unterstützung für Tunnel über Proxys (mit und ohne Authentisierung am Proxy).
- ❑ 1. Februar 2004 *Version 1.6-beta5* u.a. mit Socks5-Support.
- ❑ 9. Mai 2004 *Version 1.6.0* u.a. mit DHCP-Support für Windows.
- ❑ 17. April 2005 *Version 2.0* u.a. mit Point-to-Multipoint-Servern, Übergabe von Tunnelparametern an den Client zur Laufzeit, PAM-basierte Authentisierung als Plugin, ...
- ❑ 16. Februar 2006 *Version 2.1-beta9* erlaubt es, zwei verschiedene Prozesse über den gleichen Port erreichbar zu machen, z.B. OpenVPN und Apache (https) auf Port 443.

Mittlerweile hat OpenVPN eine Fülle von Optionen und Eigenschaften, die es problemlos mit kommerziellen Produkten konkurrieren lässt. Und die Entwicklung geht natürlich noch weiter.

5.2 Schlüsselaustausch mit *Transport Layer Security (TLS)*

*SSL: Secure
Socket Layer
TLS: Transport
Layer Security*

Um eine gesicherte Verbindung mittels OpenVPN zu erstellen, müssen selbstverständlich alle Daten, die über den Primärkanal des VPN laufen, verschlüsselt werden. Hierfür verwendet OpenVPN SSL beziehungsweise den Nachfolger TLS. SSL ist ein ursprünglich von Netscape beschriebenes Protokoll, von dem es drei verschiedene Versionen von SSLv1 über SSLv2 bis hin zu SSLv3 gibt. Es wurde zunächst entwickelt, um HTTP-Verbindungen sicher übertragen zu können. Wenn in einem Browser Internetadressen eingegeben werden, die statt dem üblichen »http« mit »https« beginnen, so bedeutet dies, dass eine in SSL gekapselte HTTP-Verbindung benutzt wird. Allerdings ist SSL nicht an HTTP gebunden – im Gegenteil: Es ist viel allgemeiner und abstrakter, so dass sich eine Reihe vielfältiger Benutzungsmöglichkeiten ergeben.

Die früheren Protokollversionen wurden durch ihre Nachfolger unter anderem deshalb abgelöst, weil (vor allem in SSLv1) signifikante Sicherheitsrisiken festgestellt wurden, welche unter ungünstigen Umständen eine Verbindung kompromittieren konnten. Wenn zwei Hosts eine verschlüsselte Verbindung mittels SSL initiieren und verschiedene Versionen von SSL verwenden, wird jeweils die niedere der beiden Versionen selektiert. Dieses Fallback ist zwar aus Interoperabilitätsgründen komfortabel, da ein älterer SSL-Client beispielsweise nicht aktualisiert werden muss; aus der Sicherheitsperspektive heraus, ist es allerdings gefährlich. Deshalb bieten verschiedene Programme, unter anderem OpenVPN, die Möglichkeit an, ein Fallback auf ältere Protokollversionen zu verhindern, ganz nach dem Motto: »Lieber gar keine Verbindung als trügerische Scheinsicherheit.«

Die Option, ein Fallback zu verhindern, sollte dringend wahrgenommen werden. Zumindest eine Verwendung von SSLv1 ist aus heutiger Sicht mehr als bedenklich und für die Verschlüsselung von möglicherweise sensiblen Daten absolut inakzeptabel.

SSL setzt auf einem verlässlichen Transportprotokoll (normalerweise TCP) auf, also über ISO/OSI-Layer 3. Das darunterliegende Transportprotokoll garantiert der Verbindung, dass kein Paketverlust auftritt, indem verloren gegangene Pakete erneut gesendet werden. Weiterhin wird dadurch sichergestellt, dass Pakete beim Empfänger wieder in der richtigen Reihenfolge zusammengesetzt werden und die Bandbreite der Verbindung optimal genutzt wird.

Das SSL-Protokoll selbst besteht aus zwei Verbindungskanälen: dem *SSL Record Protocol* und dem darin geschachtelten *SSL Handshake Protocol*. Durch das *SSL Handshake Protocol* wird es dem Client und Server einerseits ermöglicht, sich gegenseitig zu authentifizieren, und andererseits einen Verschlüsselungsalgorithmus auszuhandeln und zugehörige kryptografische Schlüssel auszutauschen. Dies geschieht noch, bevor ein einziges Zeichen über den Kanal übertragen wird. Hierzu wird der *Diffie-Hellman Key Exchange* benutzt, der in Kapitel 11.3 näher erläutert wird. Die so erstellte Verbindung erfüllt also von Anfang an folgende Kriterien:

- ❑ Die Verbindung ist sicher; nach dem anfänglichen Handshake wird sämtliche Nutzlast mit dem ausgehandelten, symmetrischen Schlüsselalgorithmus verschlüsselt (beispielsweise DES oder RC4).
- ❑ Beide Seiten können die Identität des jeweiligen Partners überprüfen. Dies wird durch einen asymmetrischen Schlüsselalgorithmus (beispielsweise RSA oder DSS) bewerkstelligt.
- ❑ Die Verbindung ist verlässlich. Dies schließt unter anderem ein, dass die gesamte, in Paketen übertragene Nutzlast mit einem HMAC versehen wird. Von der Nutzlast wird also vor dem Senden ein »Fingerabdruck« genommen und dieser mit übertragen. Der Empfänger überprüft dann, ob die empfangenen Daten den korrekten Fingerabdruck haben, und verwirft die Daten, falls dies nicht der Fall sein sollte. Für die Erstellung des Fingerabdrucks werden sichere Hashfunktionen (wie SHA oder MD5) verwendet.

*HMAC: Hashed
Message
Authentication Code*

Die Ziele einer solchen SSL-verschlüsselten Verbindung sind – in der Reihenfolge absteigender Wichtigkeit:

1. *Kryptografische Sicherheit*: Ein potenzieller Angreifer könnte mitgelesene Daten nicht entschlüsseln und kann sie so nicht verstehen.
2. *Interoperabilität*: SSL stellt Programmierern eine uniforme API bereit. So wird es möglich, dass verschiedene Programmierer Applikationen entwickeln, ohne dass der eine Einsicht in den Code des anderen nehmen muss.
3. *Erweiterbarkeit*: Wie schon angedeutet, können in dem SSL-Standard beliebige symmetrische und asymmetrische Verschlüsselungsmethoden sowie Hashalgorithmen verwendet

*API: Application
Programming
Interface*

werden. SSL bietet also eine abstrakte Schnittstelle für die jeweiligen Funktionstypen an und die Algorithmen werden in diese eingesetzt wie Bits in einen Schraubendreher. So können einerseits bereits existierende Verschlüsselungsbibliotheken benutzt und andererseits muss kein komplett neues Protokoll entwickelt werden, falls sich ein bestimmter Algorithmus als fehlerhaft erweisen sollte. Dadurch wird verhindert, dass sich der Verschlüsselungsstandard häufig ändert und so möglicherweise neue Risiken in einen neuem Standard eingebracht werden.

4. *Effizienz*: Da sämtliche kryptografische Funktionen (besonders asymmetrische) dazu neigen, eine hohe CPU-Last zu verursachen, kann SSL durch eine optionale, intelligente Sitzungsverwaltung die Anzahl der Verbindungen, die komplett neu aufgebaut werden müssen, reduzieren. Zusätzlich wurde darauf Wert gelegt, dass Verbindungen so wenig Netzwerkaktivität wie möglich verursachen.

Das TLS-Protokoll, beschrieben in RFC 2246, basiert auf SSLv3. Momentan ist es in den zwei Versionen 1.0 und 1.1 verfügbar. Die beiden Protokolle sind sich im Aspekt der Verbindungskriterien und gesteckten Ziele so ähnlich, dass normalerweise über beide Protokolle beinahe synonym gesprochen wird: Das Kürzel SSL/TLS ist sehr gebräuchlich. TLS wurde primär deshalb entwickelt, damit die IETF über ein offenes, von der Community unterstütztes Protokoll verfügt. TLS bietet normalerweise auch einen Fallback auf SSLv3 an, so dass, obwohl die Protokolle nicht syntaktisch kompatibel sind, SSL-Hosts mit TLS-Clients und umgekehrt kommunizieren können. Die Hauptunterschiede zwischen den beiden Protokollen liegen in folgenden Details:

- Die Authentifizierung ist bei TLS optional: Das Protokoll unterstützt einen völlig anonymen Modus, in dem keine Seite überprüft, ob die jeweils andere der »richtige« Partner ist. Dies birgt zwangsläufig die Gefahr einer »Man-in-the-Middle«-Attacke. Für den Einsatz bei OpenVPN ist der anonyme Modus von keinerlei Bedeutung, da es üblicherweise keinen Sinn macht, ein VPN ohne Überprüfung der Kommunikationspartner zu betreiben.
- Client- und Serverauthentifizierung wird in TLS mittels X.509-konformer Zertifikate bewerkstelligt. Im Gegensatz zu SSL sind also Zertifikate bei TLS Pflicht. Dadurch muss eine PKI gepflegt werden, was für Einsteiger etwas umständ-

*IETF: Internet
Engineering Task
Force*

*PKI: Public Key
Infrastructure*

lich erscheinen mag. Einmal eingerichtet wird allerdings jeden Anwender die Flexibilität und Erweiterbarkeit einer PKI überzeugen.

5.2.1 OpenSSL

Überblick

OpenSSL ist ein quelloffener, kostenloser SSL-Toolkit. Das Ziel, das sich die Entwickler von OpenSSL gesteckt haben, ist die Implementierung der Standards SSLv2/v3 sowie TLSv1. OpenSSL ist unter anderem deshalb ein so robustes und »mächtiges« Programm geworden, weil den Entwicklern der Sicherheitsaspekt besonders am Herzen liegt: Innerhalb von 3 Jahren gab es lediglich 15 bekannt gewordene Sicherheitsrisiken. Von diesen 15 sind 4 Stück als definitives Sicherheitsproblem einzustufen, wobei tatsächlich nur eine Sicherheitslücke (behooben im Juli 2002) als praktisch realisierbar gilt. Die anderen drei sind eher theoretischer Natur und nur unter exakt definierten Laborbedingungen durchführbar. Auf der OpenSSL-Webseite, www.openssl.org, wurden sämtliche Probleme gut dokumentiert und schnell behoben. Diese Zahl erscheint einem erst recht gering, wenn man den enormen Umfang des OpenSSL-Projekts ansieht: In der aktuellen Version besteht OpenSSL aus knapp 350.000 Zeilen Quelltext.

Zertifikate

In einer PKI, wie sie auch von OpenVPN benutzt wird, hat jeder Teilnehmer einen öffentlichen und einen privaten Schlüssel (Public beziehungsweise Private Key). Die gesamte Verschlüsselung wird deshalb als asymmetrisch bezeichnet, weil zum Ver- beziehungsweise Entschlüsseln verschiedene Schlüssel benötigt werden.

Will Teilnehmer A (Andreas) an B (Barbara) eine verschlüsselte Nachricht senden, so benutzt er zum Verschlüsseln Barbaras öffentlichen Schlüssel. Nur Barbara kann die Nachricht jedoch, unter Benutzung ihres privaten Schlüssels, entschlüsseln. Andreas kann die von ihm verschlüsselte Nachricht also nicht einmal selbst mehr entschlüsseln. Von einem solchen Schlüsselpaar ist der öffentliche Schlüssel also für jeden zugänglich; jeder Teilnehmer hat nur jeweils seinen eigenen privaten Schlüssel. Wichtig hierbei ist die Voraussetzung, dass aus dem öffentlichen Schlüssel keinerlei Rückschlüsse auf den privaten Schlüssel vorgenommen

werden können. Dies wird bei allen gängigen asymmetrischen Verschlüsselungsverfahren gewährleistet.

*Brute-Force-Attacken
(»mit roher Gewalt«)
bezeichnen
üblicherweise den
einfachsten, aber
langwierigsten Weg,
ein Passwort zu
entschlüsseln.*

Theoretisch besteht die Möglichkeit, einen privaten Schlüssel herauszufinden, indem man einfach alle möglichen Schlüssel durchprobiert (Brute-Force-Attacke). Dass diese Attacke allerdings rein theoretischer Natur ist, wird schnell klar, wenn man sich ausrechnet, wie lange so etwas dauert: Selbst wenn alle Menschen auf der Welt einen Computer hätten und diesen 24 Stunden am Tag dafür einsetzen würden, einen 1024-Bit-Schlüssel zu knacken (wobei jeder Computer pro Sekunde eine Million Schlüssel testen könnte), würde das Vorhaben knappe 10^{285} Jahre dauern. Heutzutage sind übliche Schlüssellängen 2048 sowie 4096 Bit, was den Vorgang noch einmal auf über 10^{593} beziehungsweise 10^{1209} Jahre verlängern würde.

Da in einem OpenVPN-System möglicherweise sehr viele Teilnehmer vorhanden sind, ist das Management der verschiedenen Schlüsselpaare sehr aufwändig: Der Server muss jeden öffentlichen Schlüssel aller Teilnehmer verwalten und diese natürlich updaten, wenn sie sich ändern sollten. Deshalb gibt es einen Zertifizierungsprozess für Schlüsselpaare, der auf digitalen Signaturen basiert.

Digitale Signatur

Eine digitale Signatur funktioniert folgendermaßen: Will Andreas an Barbara eine Nachricht schreiben, die er signiert hat, so macht er zunächst von der Nachricht einen so genannten »Message Digest«, generiert also einen Hashwert. Die Hashfunktion muss dabei die Eigenschaft besitzen, dass schon winzige Änderungen an Zeichen der originalen Nachricht (also beispielsweise Änderung eines einzelnen Bits) einen komplett anderen Hashwert erzeugen. Es darf also keine leicht berechenbare Korrelation zwischen der Nachricht und dem Hashwert bestehen.

privater Schlüssel

Zur Illustration kann man sich eine sehr leichte Hashfunktion ausdenken: In einem beliebigen Text wird jedem Buchstaben die zugehörige Zahl von 1-26 des Alphabets zugeordnet. Diese so entstandenen Zahlen werden aufaddiert und am Ende werden die letzten 3 Stellen tatsächlich verwendet (Berechnung durch modulo 1000). Der Text »OpenVPN« hätte also beispielsweise den Hashwert 102. Diese Zahl verschlüsselt nun Andreas mit seinem *privaten* Schlüssel und hängt sie der Nachricht an. Barbara bekommt die Nachricht mit angehängter Signatur und erstellt zunächst denselben Hashwert über den Text, den sie empfangen hat. Dann entschlüsselt sie die Signatur mit Hilfe von Andreas *öffentlichem* Schlüssel. Stimmen beide Werte überein, gilt die Signatur als korrekt und Barbara kann annehmen, dass die Nachricht tat-

öffentlicher Schlüssel

sächlich von Andreas stammt: Schließlich hat nur Andreas selbst seinen privaten Schlüssel.

Ganz ähnlich wird ein Zertifikat erstellt: Über allen Schlüssel-paaren steht die so genannte CA, die Certificate Authority, welche auch ein Schlüsselpaar besitzt. Der öffentliche Schlüssel der CA muss sowohl dem OpenVPN-Server als auch allen OpenVPN-Clients vorliegen. Wenn nun Andreas sich zu unserem OpenVPN-Server verbinden will, erstellt er zunächst einmal ein Schlüssel-paar, also einen öffentlichen und dazu passenden privaten Schlüssel. Der öffentliche Teil dies Paares wird dann »Certificate Signing Request«, also die Anfrage auf Ausstellung eines Zertifikats, genannt. Andreas legt seinen öffentlichen Schlüssel dann der CA vor. Der Manager der CA überprüft, ob Andreas tatsächlich der ist, der er vorgibt zu sein (zum Beispiel durch Prüfung des Personalausweises), und ob es ihm überhaupt erlaubt sein soll, sich zu dem OpenVPN-Server zu verbinden.

Wenn alles in Ordnung ist, unterschreibt die CA mit ihrem privaten Schlüssel den öffentlichen Schlüssel von Andreas. Zusätzlich werden der Unterschrift weitere Informationen beigelegt, wie etwa Name und Kontaktadresse des Ausstellers und des Signierenden. Schließlich wird alles mit einem Verfallsdatum versehen, an dem die Unterschrift ihre Gültigkeit verliert. Die Kombination aus öffentlichem Schlüssel zusammen mit der digitalen Signatur einer CA nennt man nun »Zertifikat«.

Digitales Zertifikat

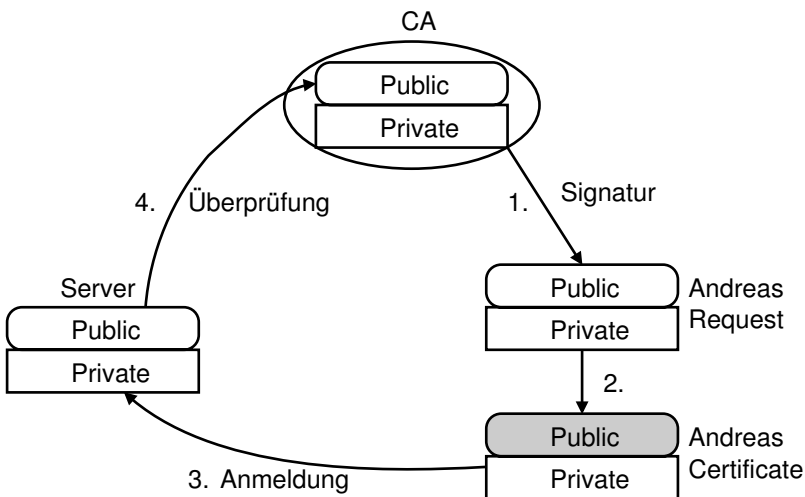


Abbildung 5.1
Ausstellung und
Überprüfung eines
Zertifikats

Zunächst scheint die CA also den Vorgang der Authentifikation zu verkomplizieren – aber der Eindruck täuscht. Die CA ist

lediglich bei einer Kommunikation von *zwei* (oder wenig mehr) Partnern ein Hindernis: Denn würde ein Server eine auf asymmetrischer Kryptografie basierende Verbindung mit sehr vielen Clients aufbauen wollen, so müsste er von *jedem einzelnen* Client jeweils den öffentlichen Schlüssel kennen! Dies wird mit einer zunehmenden Anzahl von Kommunikationsteilnehmern also ziemlich unübersichtlich.

Man stelle sich vor, es gäbe drei verschiedene Server, die alle jeweils einhundert Clients bedienen sollen: Jeder der drei Server bräuchte also alle öffentlichen Schlüssel der Clients. Die Datenbank der öffentlichen Schlüssel muss jeweils bei allen Servern synchron sein: ein nicht zu unterschätzender Wartungsaufwand.

Der Verwaltungsaufwand beim Schlüsselmanagement kann mit einer CA beträchtlich verringert werden!

Mit einer CA ist das Problem elegant gelöst: Jeder der drei Server kennt nur den öffentlichen Schlüssel der vertrauten CA. Wenn nun ein Client mit einem von dieser CA signierten, gültigen Zertifikat eine Anmeldung wünscht, so kann dies ganz einfach überprüft werden, ohne dass der Server den Client überhaupt vorher kennen muss. Das Motto lautet hier also: Die Freunde meiner CA sind auch meine Freunde!

Wenn sich nun Andreas mit dem OpenVPN-Server verbinden will, sendet er dem Server einfach sein von der CA unterschriebenes Zertifikat. Der Server kann die Signatur leicht überprüfen, da er ja auch über den öffentlichen Schlüssel der CA verfügt. Wenn die Signatur der CA stimmt, wird die Verbindung akzeptiert. Falls das Zertifikat von irgendeiner anderen CA unterschrieben wurde, das Verfallsdatum überschritten ist oder die Signatur nicht stimmt, wird ein Verbindungsaufbau abgelehnt.

Die CA ist also die kritische Instanz bei der Vergabe von Zertifikaten: Mit ihr steht oder fällt die Sicherheit eines OpenVPN-gesicherten Netzwerkes. Wenn der private Schlüssel der CA kompromittiert wird, so kann ein Angreifer beliebige neue Zertifikate erstellen und so unbemerkt weitere Teilnehmer für das OpenVPN-Netzwerk akkreditieren.

Um diese Situation zu vermeiden, gibt es mehrere Sicherheitsmaßnahmen: Zunächst kann (und sollte!) der private Schlüssel der CA zusätzlich durch ein symmetrisches Verschlüsselungsverfahren gesichert werden. Dadurch wird es notwendig, dass der private Schlüssel vor dem Signiervorgang zunächst selbst einmal durch Eingabe eines Passwortes entschlüsselt werden muss. Sollte es also tatsächlich einem Angreifer gelingen, den privaten Schlüssel zu stehlen, so ist dieser für ihn wertlos, da ihm das Passwort, mit dem der Schlüssel verschlüsselt wurde, nicht bekannt ist. Diese Sicherheitsmaßnahme sollte durch die Verwendung eines Passwortes für

den privaten Schlüssel der CA unterstützt werden, welches *nirgendwo* anders in Gebrauch ist. Sonst könnte es einem Angreifer gelingen, den privaten Schlüssel der CA doch zu benutzen.

Über allem steht allerdings die physikalische Sicherheit: Es darf keinem Angreifer gelingen, in irgendeiner Weise physikalischen Zugriff auf die CA zu bekommen. Am einfachsten und effektivsten ist dies zu realisieren, indem man die CA auf einem separaten Rechner aufsetzt, der über *keinerlei* Netzwerkverbindung verfügt. Wenn Schlüssel unterschrieben werden sollen, können diese per Diskette oder Memorystick auf den CA-Rechner übertragen und dort signiert werden. Wenn auch etwas unbequem, so ist diese Methode für kritische Systeme allerdings ein unverzichtbarer Gewinn an Sicherheit.

Hierzu sei auch auf den RFC 3647 [32] verwiesen, in dem allgemeine Ratschläge gegeben werden, wie solche kritischen Systeme gehandhabt werden sollten. Unter anderem setzt auch die DFN-CERT, die Zertifizierungsinstanz des Deutschen Forschungsnetzes, diese Richtlinien bei der Signierung von Zertifikaten um.

Der Verein zur Förderung eines Deutschen Forschungsnetzes e.V. – kurz DFN – betreibt nicht nur das DFN-Cert, sondern mit dem X-WiN auch eines der größten deutschen Internet-Backbones.

5.2.2 Erwerbbares Zertifikate

Ein Zertifikat ist natürlich nur immer so viel wert wie die CA, die es signiert hat. Deshalb gibt es bestimmte Unternehmen, die als Dienstleistung anbieten, einen vorgelegten Public Key zu signieren, nachdem sie die Identität des Anfragenden überprüft haben. Zu diesen Institutionen zählt beispielsweise VeriSign oder Trustcenter. Die öffentlichen Schlüssel dieser akkreditierten Firmen werden nun ihrerseits von einer obersten Instanz signiert. In Deutschland macht dies die »Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen« (ehemals Regulationsbehörde für Telekommunikation und Post, kurz RegTP). Eine solche oberste Instanz nennt man »Root CA«. So kann mittels des öffentlichen Schlüssels der Bundesnetzagentur überprüft werden, ob das Zertifikat einer hierarchisch darunterliegenden CA (wie beispielsweise von VeriSign) gültig ist. Mit dem öffentlichen Schlüssel von VeriSign wiederum kann der Benutzer prüfen, ob ein gekauftes Zertifikat wirklich von VeriSign unterschrieben wurde.

Die Bundesnetzagentur signiert freilich nicht die Root-Zertifikate aller kommerziellen Zertifikatvertreiber. Lediglich jene Zertifikate, die nach der Zertifikatverordnung erstellt und gehandhabt werden, werden von ihr signiert. Dies trifft jedoch nur auf die Minderheit der Anbieter zu: Der Großteil benutzt selbstsignierte

Root-Zertifikate. Der Kunde ist deshalb bei diesen Firmen auf den »guten Namen« angewiesen – eine zweifelhafte Vertrauensgrundlage also. Für OpenVPN-Zertifikate jedoch sind auch selbstsignierte Zertifikate völlig ausreichend, da in der Regel derjenige, der den Server administriert, gleichzeitig auch derjenige ist, der die CA erstellt hat.

https: »Hypertext Transfer Protocol Secure« ist HTTP in Kombination mit SSL-Verschlüsselung und bietet daher die Möglichkeit, dass der Client den Server authentifizieren kann.

Solche erwerbbaaren Zertifikate sind beispielsweise bei Webseiten, die mit https verschlüsselt werden, von Nutzen: Da viele Webbrowser, wie beispielsweise der Mozilla Firefox oder Opera, schon viele öffentliche Schlüssel der Zertifizierungsunternehmen mitbringen, wird der Benutzer nicht extra aufgefordert, das Zertifikat selbst zu kontrollieren, wenn er eine mit einem solchen Zertifikat verschlüsselte Webseite betritt. Der Browser prüft die Signatur des Zertifikats eigenständig anhand des mitgelieferten öffentlichen Schlüssels und setzt die Übertragung unterbrechungsfrei fort, wenn alles in Ordnung ist. Trotzdem ist es theoretisch möglich, dass Malware oder Trojanische Pferde den Pool der »vertrauten Schlüssel« einfach um eigene Schlüssel ergänzen: Wird der Schlüsselpool so verseucht, ist eine automatische Überprüfung des Webbrowsers natürlich wirkungslos und nicht aussagekräftig. Letztendlich bleibt hier also die endgültige Überprüfung, von wem Zertifikate stammen, wieder dem Benutzer überlassen.

Generell sind erwerbbaare Zertifikate für ein VPN nicht wirklich sinnvoll, da alle Parteien, die an dem VPN teilnehmen (Server und Clients), sich jeweils persönlich authentifizieren sollten, bevor irgendwelche Schlüssel oder Rechte vergeben werden.

5.3 Verschlüsselungsverfahren

OpenVPN bietet grundsätzlich zwei verschiedene Verschlüsselungsverfahren an: eine Verschlüsselung mittels statischen Schlüssels (static key) oder eine Verschlüsselung auf der Basis der SSL/TLS-Kryptografiebibliothek *OpenSSL* im Zusammenhang mit Zertifikaten. Grundsätzlich ist die Verschlüsselung unter Benutzung eines statischen Schlüssels problematisch, denn sie erreicht nicht das Sicherheitsniveau einer zertifikatbasierten Verschlüsselung: Da immer derselbe statische Schlüssel für alle Pakete verwendet wird, muss ein Angreifer nur ein einziges Mal den Schlüssel »knacken« und hat sofort alle bisher möglicherweise mitgeschnittenen Verbindungen im Klartext vorliegen.

Es fehlt also jegliche »Forward Secrecy« in diesem Schema: Damit wird die Eigenschaft von Kryptosystemen bezeichnet, dass mit

dem Aufbrechen eines einzelnen Pakets eben *nicht* alle früher mitgeschnittenen Verbindungen sofort auch gebrochen sind. Dies ist ein grundlegender Bestandteil der Sicherheit, die unter anderem OpenVPN bietet.

Dies ist bei einer Verschlüsselung durch SSL/TLS nicht der Fall. Da bei diesem Verfahren in regelmäßigen Abständen temporäre Sitzungsschlüssel erzeugt werden, würde das Brechen des Hauptschlüssels dem Angreifer nur einen geringen Teil an kompromittierenden Daten liefern.

Statischer Schlüssel	SSL/TLS
<ul style="list-style-type: none"> + Einfache Konfiguration + Keine Zertifikatverwaltung notwendig 	<ul style="list-style-type: none"> + Ständig wechselnde Sitzungsschlüssel + Mehrere Clients möglich
<ul style="list-style-type: none"> - Ein Client, ein Server - Im Fall einer Kompromittierung sind alle früheren Sitzungen lesbar - Geheimer Schlüssel muss auf beiden Rechnern im Klartext vorhanden sein - Geheimer Schlüssel muss vor Start des VPN getauscht werden 	<ul style="list-style-type: none"> - Relativ komplizierter Konfigurationsprozess

Table 5.1

Statische Schlüssel und SSL/TLS-Zertifikate haben unterschiedliche Vor- und Nachteile.

Auch die Skalierbarkeit einer statischen Verschlüsselung ist beschränkt: Im Gegensatz zu einer PKI, also einer zertifikatbasierten Verschlüsselung, kann hier jeweils ein Server nur mit einem Client verbunden werden. Bei Verwendung einer PKI hingegen kann ein Server eine Verbindung an eine beliebige Anzahl an Clients zur Verfügung stellen.

Zuletzt haben statische Schlüssel noch den Nachteil, dass der geheime Schlüssel auf beiden Rechnern, dem Server und dem Client, im Klartext existieren muss. Es gibt also zwei Angriffsstellen, deren Kompromittierung in einem Verlust der Sicherheit des VPN resultieren würde. Diese Klartextschlüssel müssen vorher über einen sicheren Kanal ausgetauscht werden. Aus diesem Grund muss beispielsweise eine auf dem Basis von SSH bzw. OpenSSH aufsetzende Managementstruktur bestehen oder ein sicheres Medium (USB-Stick, Floppy-Disk) gewählt werden.

Dennoch hat eine Verschlüsselung mittels statischer Schlüssel einen unschlagbaren Vorteil gegenüber einer SSL/TLS-basierten

PKI: Public Key Infrastructure

Authentifizierung: Sie ist sehr einfach zu konfigurieren. Sämtliche Schritte zum Erstellen eines statischen Schlüssels können von OpenVPN selbst übernommen werden, eine Benutzung der Programme der OpenSSL-Bibliothek entfällt hierbei vollständig.

5.4 Virtuelle Netzwerkschnittstellen

Auf der Netzwerkebene sind virtuelle Netzwerkschnittstellen für OpenVPN der zentrale Begriff. Die folgenden Abschnitte werden diesen nun näher erläutern.

5.4.1 Das Tunnelprinzip

VPN-Lösungen, welche die Verschlüsselung und somit die Modifikation der Datenpakete im Betriebssystemkern durchführen, haben ein besonderes Problem: Die Portierung auf andere Systeme ist schwierig. Alternativ könnte man sich an Standards wie IPsec halten, aber auch dort tauchen immer wieder Inkompatibilitätsprobleme auf, da manche Hersteller nicht alle genormten Erweiterungen unterstützen. Für den Anwender unterschiedlicher Systeme bedeutet dies häufig, dass in solchen Fällen nur der kleinste gemeinsame Nenner aller Lösungen genutzt werden kann.

OpenVPN allerdings basiert auf so genannten *virtuellen Netzwerkschnittstellen* und verknüpft diese mit einer Verschlüsselung im so genannten *Userspace*. Diese Schnittstellen bilden die Verknüpfung zwischen den über einen Tunnel übertragenen Daten, dem OpenVPN-Prozess und dem Betriebssystem. Durch diese Kombination entfällt ein Eingriff in den Betriebssystemkern. Zusätzlich erlangt OpenVPN durch den Einsatz solcher virtuellen Schnittstellen auch noch einen hohen Grad an Portabilität.

Protokollverschachtelung: Geschickt verpackte Daten

Jedes IP-Paket besteht im Wesentlichen aus zwei Teilen: einem so genannten Kopf (oder *Header*) aus mindestens 20 Bytes und dem Datenanteil (auch *Payload* genannt). Der Header enthält alle notwendigen Informationen, die ein Router im Internet benötigt, um ein Paket in die richtige Richtung weiterzuleiten. Hierunter fallen beispielsweise die Adresse des Zielsystems, die Länge des Pakets und auch die TTL.

TTL: Time to Live

Bei der Protokollverschachtelung in Tunneln wird ein kleiner Trick verwendet: Ein Paket wird schlichtweg als Payload eines anderen Pakets transportiert. Man stelle sich hierzu einfach ein Ge-

schenk vor, das in einem Versandkarton einer Versandfirma verpackt ist. Dieser Versandkarton wird üblicherweise mit Absender- und Empfängerinformationen versehen, bevor er auf die Reise geschickt wird. Die Versandfirma wiederum sammelt mehrere dieser Kartons und befördert diese während des Transports in Containern (welche sinnbildlich auch wieder Versandkartons sind), bis diese an einem Ort nahe des Ziels ankommen. Dort wird der Versandkarton wieder dem Container entnommen und dem Ziel zuge stellt. Analog zu diesem Beispiel werden in Tunneln Anwendungsdaten (Geschenk) über ein IP-Paket (Versandkarton) in einem weiteren IP-Paket (Container) transportiert.

Es gibt übrigens unterschiedliche Implementationen solcher *IP-in-IP*-Tunnel. Der Standard *GRE (Generic Routing Encapsulation)* [35] dürfte davon sicher einer der meistverwendeten sein.

Die von OpenVPN benutzten virtuellen Netzwerkschnittstellen *tun* und *tap* stellen gewissermaßen den Endpunkt eines solchen Tunnels dar.

GRE: Generic Routing Encapsulation

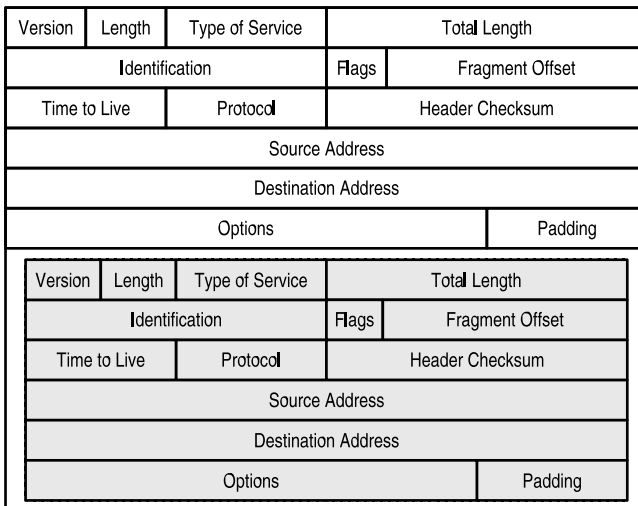


Abbildung 5.2
IP-Header verpackt in IP

5.4.2 Hilfsmittel der Abstraktion: *tun*- und *tap*-Geräte

Virtuelle Netzwerkschnittstellen sind die Grundlage für einen Verbindungsaufbau von OpenVPN. Das Betriebssystem behandelt die Tunnelschnittstellen wie eigenständige Netzwerkkarten mit einem eigenen Interface-Namen. Unter Linux werden diese Inter-

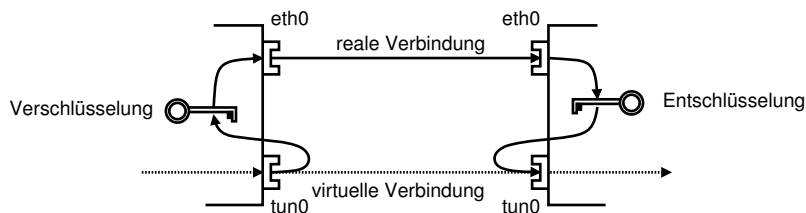
face `tunX` beziehungsweise `tapX` genannt. Sie stellen eine Punkt-zu-Punkt-Verbindung dar.

Eine Verbindung, die über virtuelle Netzwerkschnittstellen aufgebaut wird, läuft über eine »echte« Netzwerkschnittstelle, zum Beispiel `eth0`. Diese »echten« Daten nennt man *Primärpakete*, die »getunnelten« *Sekundärpakete*. Jedes Paket, das über ein virtuelles Interface gesendet wird, wird in TCP/IP- oder UDP/IP-Pakete verpackt und über die »echte« Netzwerkschnittstelle gesendet. Mit einer realen Netzwerkschnittstelle ist ein Ethernet-Adapter, ein *loopback device* oder eine andere Punkt-zu-Punkt-Verbindung gemeint, die TCP/IP und UDP/IP unterstützt.

tap-Schnittstellen werden verwendet, um Ethernet-Netzwerke miteinander zu verbinden. Wenn ein Rechner als Switch eingesetzt werden soll, um Netzwerk A mit Netzwerk C über Netzwerk B zu verbinden, wird zwischen Rechner A in Netzwerk A und Rechner C in Netzwerk C ein Punkt-zu-Punkt-Tunnel aufgebaut. Alle Pakete in Netzwerk A, die einen Empfänger aus Netzwerk C adressieren, werden so von Rechner A an Rechner C über Netzwerk B geleitet.

tun-Schnittstellen sind ebenfalls Netzwerkschnittstellen, dienen jedoch dazu, Rechner A direkt an das Netzwerk C über Netzwerk B zu verbinden. Das bedeutet, dass diese virtuelle Netzwerkschnittstelle nur Pakete des Rechners A, aber nicht alle Pakete aus Netzwerk A, die für Netzwerk C bestimmt sind, übermittelt.

Abbildung 5.3
Die über virtuelle Schnittstellen transportierten Pakete werden wiederum in Pakete der tatsächlich vorhandenen Schnittstellen gewandelt.



In Abbildung 5.3 bindet sich OpenVPN an das virtuelle Interface `tun0` und schickt alle Daten der Netzwerkschicht, welche auf die virtuelle Schnittstelle `tun0` geschrieben werden, über die Anwendungsschicht von `eth0` als Nutzdaten. Somit kann jedes Protokoll in TCP/IP- oder UDP/IP-Pakete verpackt und übermittelt werden.

Konfiguration virtueller Netzwerkinterfaces

Für den testweisen Aufbau einer Verbindung zwischen zwei virtuellen Interfaces kann man mittels OpenVPN einen unverschlüs-


```

UP PUNKTZUPUNKT RUNNING NOARP MULTICAST MTU:1500 Metric
:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
Kollisionen:0 Sendewarteschlangenlänge:100
RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)

```

Bei deutschen Umgebungen werden Punkt-zu-Punkt-Verbindungen mit dem Kürzel P-z-P ausgezeichnet, englische Umgebungen verwenden P-t-P.

Sowohl beim Server als auch beim Client lässt sich erkennen, dass die virtuellen Schnittstellen *tun0* als Punkt-zu-Punkt-Verbindungen (P-z-P) mit einer Netzmaske 255.255.255.255 konfiguriert sind.

5.5 Kontroll- und Datenkanäle

Wie bereits in Kapitel 5.2 erwähnt, setzt SSL/TLS eine zuverlässige Datenübertragung zwischen den beiden beteiligten Partnern voraus. Üblicherweise wird deshalb für Applikationen, die SSL/TLS verwenden, TCP/IP als Transportprotokoll eingesetzt. Der Vorteil von TCP/IP ist hierbei, dass die erforderliche Sicherungsschicht (im Sinne von Zuverlässigkeit) vom Betriebssystem bereitgestellt wird.

OpenVPN bietet allerdings auch die Möglichkeit, Tunnel über UDP/IP aufzubauen. Da UDP aber kein zuverlässiges Protokoll ist, müssen zusätzliche Vorkehrungen für eine reibungslose Zusammenarbeit von SSL/TLS mit OpenVPN getroffen werden. Die Lösung ist recht einfach: OpenVPN verwendet einen alten Trick aus der Netzwerktechnik und teilt eine Verbindung in zwei logische Kanäle auf:

- Alle Aktivitäten im Zusammenhang mit SSL/TLS werden (unabhängig vom verwendeten Transportprotokoll!) über einen speziellen Kanal transportiert, der eine zuverlässige Datenübertragung garantiert. Dadurch bekommt dieser Kanal wesentliche Eigenschaften einer TCP-Verbindung und ist für den Transport von SSL/TLS vorbereitet. Die einzelnen Nachrichten dieses Kanals sind durch HMAC gegen Integritätsverletzungen geschützt.
- Nutzdaten werden über einen separaten Kanal transportiert, der keine zuverlässige Datenübertragung garantiert. Diese Nachrichten sind ebenfalls mit HMAC gegen Integritätsverletzungen geschützt.
- Die Pakete beider Kanäle werden durch einen Multiplexer getrennt voneinander übertragen.

HMAC: Hashed Message Authentication Code

Mit dieser Aufteilung stehen für die SSL/TLS-bezogenen Verfahren und den eigentlichen verschlüsselten Datentransport die jeweils erforderlichen beziehungsweise erwünschten Mechanismen zur Verfügung. Zusätzlich kann es durch den Einsatz von UDP nicht zu der typischen Kollisionsgefahr aufgrund der Verwendung mehrfach ineinander verpackter zuverlässiger Transportprotokolle kommen, wie in Kapitel 13.1 erläutert.

5.6 Betriebsmodi: Server, Client, Point-to-Point und Server Mode

Nach den bisherigen technischen Grundlagen von OpenVPN werden in diesem Kapitel Aspekte des Betriebs und der Architektur betrachtet.

Wichtig ist: Man sollte sich bei der Planung von VPNs darüber im Klaren sein, für welchen Zweck es bereitgestellt werden soll, denn im Rahmen von OpenVPN lassen sich generell zwei Varianten unterscheiden:

- Point-to-Point*-Verbindungen oder
- Server-Mode*-Verbindungen

5.6.1 Point-to-Point

Wenn die Aufgabe darin besteht, eine Verbindung ausschließlich zwischen zwei Hosts herzustellen, wird das üblicherweise im *Point-to-Point*-Modus durchgeführt. Dieser war bis OpenVPN Version 1.6 der einzig verfügbare Modus und erfordert jeweils eine spezifische Konfigurationsdatei pro VPN-Verbindung. Neben der separat notwendigen Konfigurationsdatei erfordert diese Konfigurationsvariante auch pro VPN-Tunnel eine separate Kommunikationsbeziehung, was letztlich einen TCP- oder UDP-Port pro verwendetem Tunnel bedeutet.

In Abbildung 5.4 ist dargestellt, wie die drei Hosts `host_gelb`, `host_gruen` und `host_blaue` eine VPN-Verbindung zum Host `server_a` aufgebaut haben. Auf `server_a` sind für diesen Zweck drei OpenVPN-Instanzen aktiv, die jeweils mit einer eigenen Konfigurationsdatei gestartet wurden und die Kommunikation über unterschiedliche Ports (in diesem Fall 1194, 1195 und 1196) durchführen.

Sofern es nicht anderweitig konfiguriert ist, benutzt OpenVPN diesen Modus als Standard für den Betrieb. Um Unklarheiten zu

OpenVPN konnte bis einschliesslich Version 1.6 nur Point-to-Point-Verbindungen!

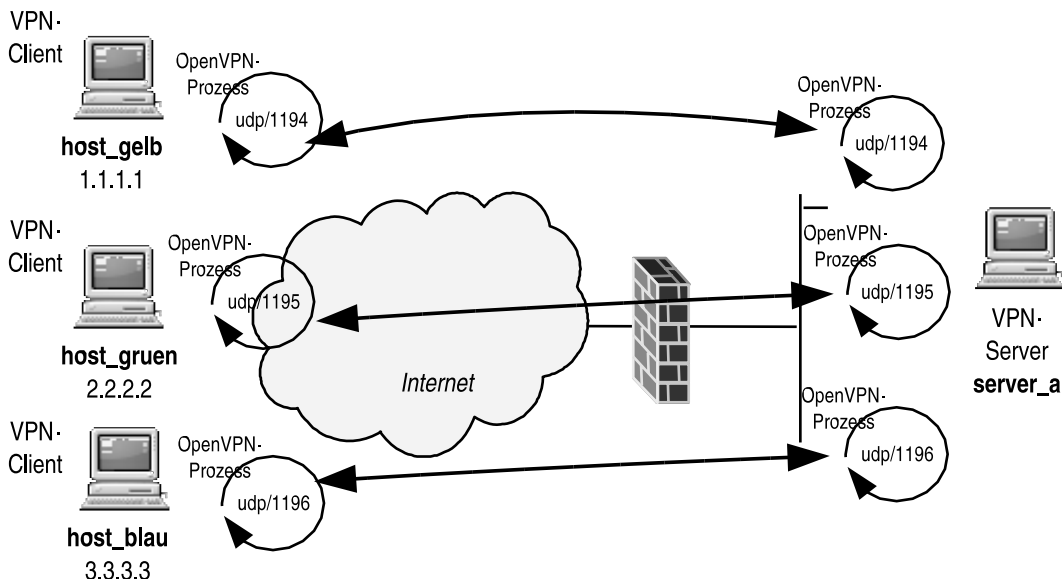


Abbildung 5.4
OpenVPN in einer
Point-to-Point-
Konfiguration

vermeiden, sollte der Modus aber innerhalb der Konfigurationsdatei durch die Option `mode p2p` explizit konfiguriert werden.

Point-to-Point-Konfigurationen sind die bevorzugte Betriebsart, wenn die Tunnelpartner prinzipiell statisch definiert sind, zum Beispiel bei der Verbindung zweier Standorte oder für einen dedizierten Zugang eines speziellen Clients.

5.6.2 Server Mode

Wenn OpenVPN als Ersatz für klassische Remote-Access-Lösungen (wie beispielsweise die Einwahl über ISDN oder analoge Leitungen) verwendet werden soll, dann sind *Point-to-Point*-Konfigurationen wenig sinnvoll. Mit zunehmender Anzahl an Clients steigt auch die Anzahl von Konfigurationsdateien und OpenVPN-Instanzen auf dem Server in entsprechender Weise an.

Wünschenswert wäre es in diesem Fall, eine Umgebung zu haben, in der alle VPN-Tunnel von einem einzigen Prozess verwaltet werden. Dadurch wird die Administration erheblich erleichtert. Der Prozess sollte nur auf einem einzigen Port zur Verfügung gestellt werden, damit der Bereich offener Ports an (vorgelagerten) Firewalls möglichst klein gehalten wird. Firewalladministratoren würden verständlicherweise die »Hände über dem Kopf zusam-

menschlagen«, wenn sie die Anforderung bekämen, nur für VPN-Tunnel 500 oder mehr Ports zu öffnen.

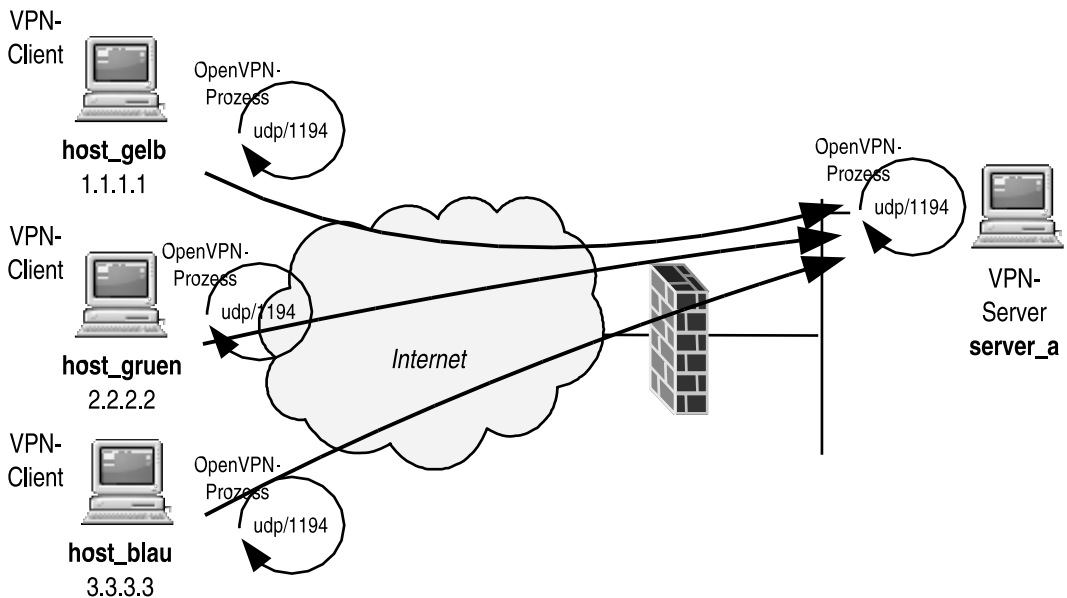


Abbildung 5.5
OpenVPN in einer
Client-Server-
Konfiguration

All diese Wünsche kann OpenVPN erfüllen: Es bedarf nur der Einrichtung des *Point-to-Multipoint*-Modus durch den Parameter `mode server`. Wenn beispielsweise folgende Zeilen für einen OpenVPN-Prozess konfiguriert sind:

```
port 1194
proto udp
dev tun42
mode server
server 192.168.41.0 255.255.255.0
```

dann gilt für diesen Prozess, dass alle Verbindungen (das heißt Anfragen für einen VPN-Tunnel) auf dem Port 1194 über das Protokoll UDP/IP ankommen und entsprechend behandelt werden sollen. Ferner verwendet der Prozess das Interface `tun42` für die Verknüpfung zwischen dem Tunnel und dem Betriebssystem. In Abbildung 5.5 ist dargestellt, wie die drei Hosts `host_gelb`, `host_gruen` und `host_blau` wiederum eine VPN-Verbindung zum Host `server_a` aufgebaut haben. Diesmal verbinden sich allerdings die OpenVPN-Prozesse sämtlicher Clients auf den glei-

chen Prozess des Servers `server_a`, und zwar auf den Port 1194. Sobald ein Verbindungsaufbau authentisiert ist, werden den Clients Adressen aus dem Netz 192.168.41.0/24 für die Konfiguration ihrer VPN-Tunnelinterfaces zugewiesen.

Bei stark frequentierten VPN-Zugängen mit vielen Benutzern ist die Variante *Server Mode* im Konfigurations- und Administrationsaufwand deutlich geringer als im *Point-to-Point*-Modus. Auf dem Server gibt es nur eine Konfigurationsdatei und für die Clients unterscheidet sich die Konfiguration üblicherweise nur durch das verwendete Zertifikat. Sofern es die Sicherheitsvorgaben erlauben, lässt sich das auch noch durch die Option `duplicate-cn` (siehe Kapitel 8.6.3) vereinfachen. Diese gestattet, dass mehrere Clients Verbindungen mit demselben Zertifikat aufbauen dürfen. Aber Achtung: Wenn bei der Verwendung dieser Option ein Zertifikat und der zugehörige Schlüssel in die falschen Hände gelangen, dann muss der Zugang für alle Clients, die dieses Zertifikat verwenden, gesperrt werden, bis auf diesen ein neues Paar installiert ist.