Studienarbeit
Verschlüsselte und PKI authentifizierte VPN Tunnel

Andreas Hofmeier

Betreuer: Dr. Kai-Oliver Detken
Studiengang: ESTI, 7. Semester
Martrikelnummer: 94453

Zusammenfassung

Im ersten Teil gibt dieses Dokument eine Einführung in die Grundlagen der Verschlüsselung. Dabei wird näher auf die Public Key Encryption (PKE) und die darauf aufsetzende Publik Key Infrastructure (PKI) eingegangen.

Der zweite Teil gibt eine grundlegende Übersicht über verschiedene Virtual Private Network (VPN) Techniken. Im Anschluss daran wird auf die Funktionsweise von OpenVPN, einer speziellen Lösung, eingegangen.

Im letzten Abschnitt wird die versuchsweise Erstellung eines VPN-Tunnels zwischen zwei Computern beschrieben. In diesem Versuch wird ein PKI aufgebaut um die Teilnehmer zu authentifizieren. An dem fertigen VPN wurden einige Versuche vorgenommen.

Der Anhang führt die im dritten Abschnitt verwendeten Skripte und Konfigurationsdateien auf.

Inhaltsverzeichnis

1 Verschlüsselung

Verschlüsselung bezeichnet ein Verfahren, welches entwickelt wurde um Informationen vor unbefugten Zugriff zu schützen.

Am einfachsten sind die Prinzipien der Verschlüsselung an mehr oder weniger konkreten Beispielen zu erklären. Daher wird im Folgenden immer wieder auf das Beispiel dreier Personen zurückgegriffen, welche miteinander Kommunizieren: Alice und Bob versuchen sich zu unterhalten und wollen dies so sicher wie möglich halten. Trudy versucht mit verschiedensten Tricks die Unterhaltung abzuhören oder zu manipulieren. Diese Personen sind nicht nur als reale Personen zu betrachten, sondern auch als Endgeräte (z.B. als handelsübliche PCs).

1.1 Probleme bei unverschlüsselter Übertragung

Im einfachsten Fall sendet Alice eine unverschlüsselte Nachricht an Bob.

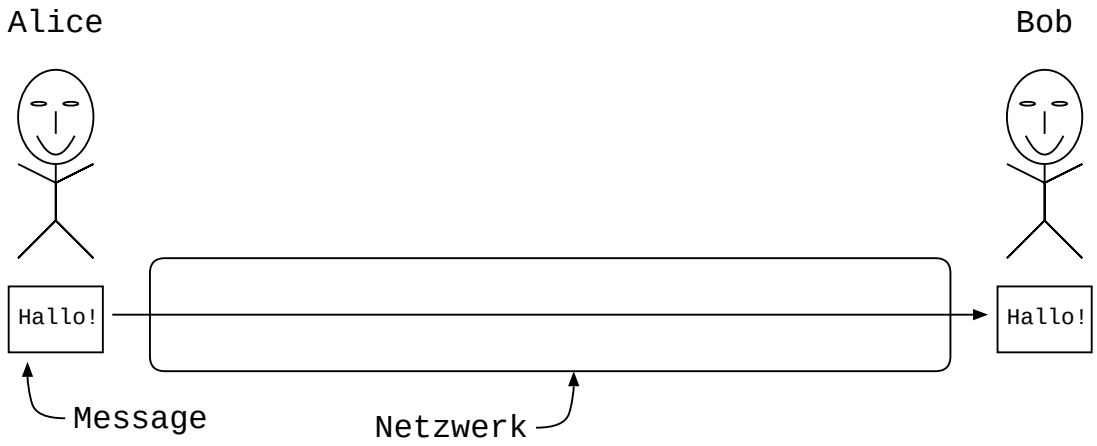

Abbildung 1: Alice sendet eine Nachricht unverschlüsselt über das Netzwerk an Bob.

Wenn sie zu diesem Zweck ein Computernetzwerk verwendet, entspricht diese Nachricht einer Folge von Nullen und Einsen, von Bits und Bytes. Dies zieht einige Konsequenzen nach sich, die berücksichtigt werden müssen:

1. Vertraulichkeit: Es ist nicht ohne weiteres Möglich, die Nachricht in einem Briefumschlag zu stecken. Die herkömmliche Übertragung von Daten in Computernetzwerken kann eher mit Postkarten verglichen werden: Jeder

der sich in den Weg der Postkarte einklinken kann, kann diese Lesen, ohne das dies Spuren hinterlässt.

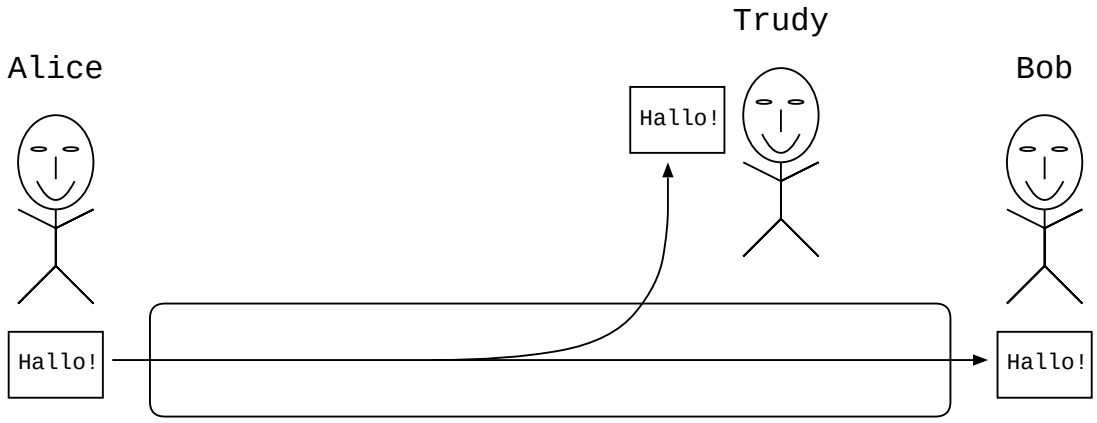

Abbildung 2: Alice sendet eine Nachricht an Bob. Trudy greift die Nachricht auf dem Weg ab und ließt sie.

2. Echtheit: Die gleichen Daten (Buchstaben und Zahlen) sehen mehr oder weniger immer gleich aus. Es kann somit keine Handschrift unterschieden werden. Eine Folge hiervon ist, dass der Sender der Nachricht nicht sicher identifiziert werden kann.

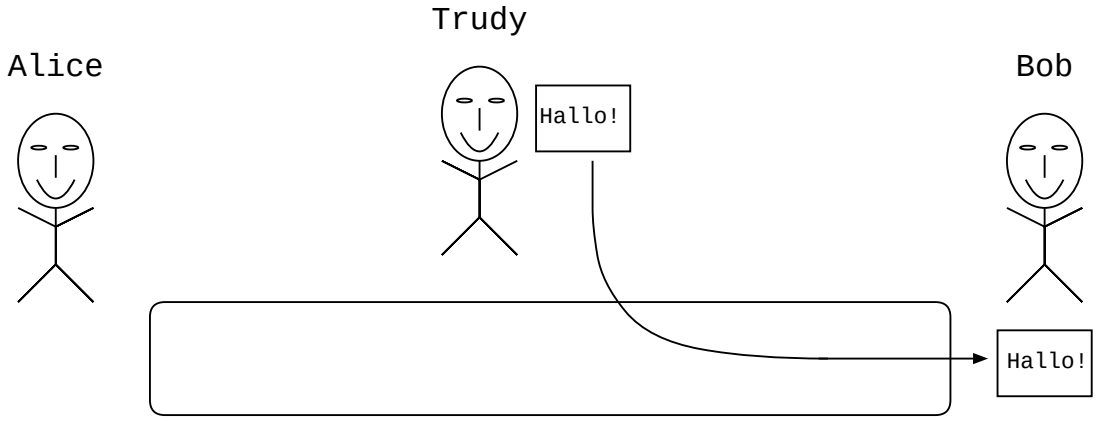

Abbildung 3: Trudy sendet eine Nachricht an Bob und macht ihn glaubend diese stamme von Alice.

3. Unversehrtheit: Die originale Nachricht könnte verändert worden sein, ohne dass dies erkennbare Spuren hinterlässt.

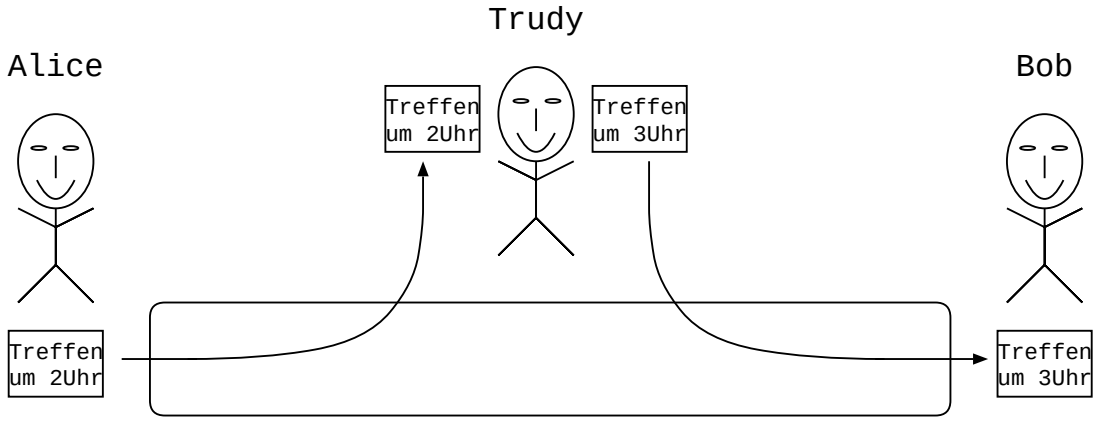

Abbildung 4: Alice sendet eine Nachricht an Bob. Trudy fängt diese ab und sendet sie verändert an Bob weiter.

1.2 Symmetrische Verschlüsselung

Verschlüsselung ist ein Versuch, die Vertraulichkeit einer Nachricht zu wahren. Zu diesem Zweck wird die Nachricht mit Hilfe eines mathematischen Verfahrens in die Chiffre[1] umgewandelt. Wer die Nachricht lesen will, muss über ein Verfahren verfügen, welches diese Umwandlung rückgängig macht.

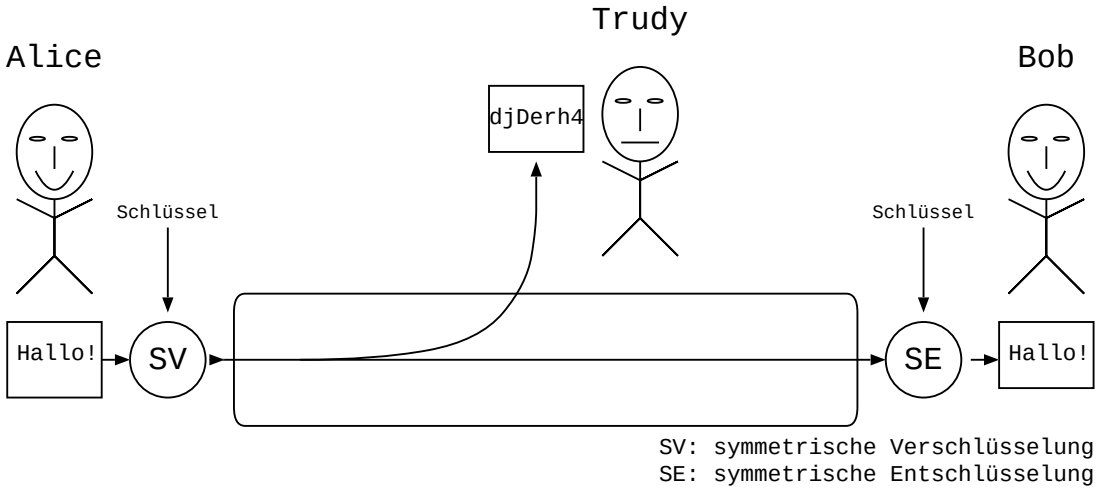

Abbildung 5: Alice sendet eine verschlüsselte Nachricht an Bob. Trudy fängt diese ab kann sie aber nicht verstehen.

An dieser Stelle wird davon ausgegangen, dass dieses Verfahren von einem Parameter abhängig ist: dem Schlüssel. Das Verfahren führt daher bei Verwendung

[1] oder Geheimschrift

gleicher Eingangsdaten und verschiedener Schlüssel zu verschiedenen Chiffren. Wer diese verschlüsselte Nachricht lesen möchte, muss nicht nur das mathematische Verfahren kennen, sondern ebenfalls den Schlüssel.

Diese Form der Verschlüsselung wird als "symmetrische Verschlüsselung" bezeichnet, da der gleiche Schlüssel zum verschlüsseln und entschlüsseln verwendet wird.

Die praktische Umsetzung führt allerdings zu dem Problem der Schlüsselverteilung. Es ist notwendig, dass sich je zwei Personen (hier Alice und Bob) einen einzigartigen Schlüssel teilen. Würden sich mehr als zwei Personen einen Schlüssel Teilen, wäre nicht gewährleistet, dass nur die Zielperson die Nachricht lesen kann. Wird davon ausgegangen, dass Alice den Schlüssel per Zufallsverfahren erzeugt, stellt sich immer noch die Frage, wie wird der Schlüssel von Alice zu Bob übertragen wird.

Übermittelt Alice den Schlüssel über das Netzwerk zu Bob, könnte Trudy den Schlüssel abfangen und damit alle nachfolgenden Nachrichten entschlüsseln und somit lesen. Es könnte darüber Nachgedacht werden, den Schlüssel zu verschlüsseln. Bei Anwendung der symmetrischen Verschlüsselung, verschiebt sich dadurch das Problem der Schlüsselverteilung lediglich. Denn der Schlüssel zur Verschlüsselung des Schlüssels müsste ebenfalls übermittelt werden.

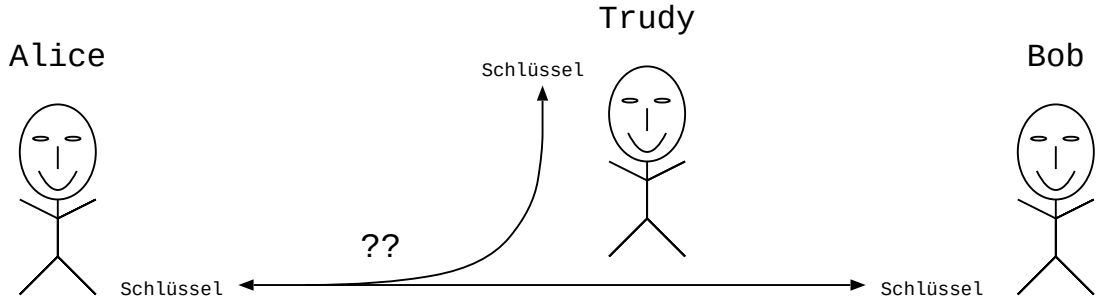

Abbildung 6: Wie einigen sich Alice und Bob auf einen Schlüssel?

Diese Problem könnte auf der menschlichen Ebene gelöst werden:

- Alice trifft Bob persönlich zum Austausch des Schlüssels.
- Alice greift zum Telefon, ruft Bob an und gibt den Schlüssel durch. Bob

erkennt Alices Stimme. In diesem Fall wird auf ein zweites Netzwerk zurückgegriffen, welches als sicherer erachtet wird.

Beide Varianten setzen voraus, dass sich Alice und Bob persönlich kennen. Aber wie oft kommt es vor, dass wir den Betreiber einer Webseite persönlich kennen? Selbst wenn wir jede Person (Betreiber eines Services) kennen sollten, wäre der Aufwand immens, die Schlüssel von Hand (z.B. per Telefon) auszutauschen oder sich gar zu treffen.

1.3 Publik-Key (unsymmetrische) Verschlüsselung

Etwas neues muss her: Publik Key Encryption (PKE). Dieses Verfahren wird des öfteren als unsymmetrische Verschlüsselung bezeichnet, da verschiedene Schlüssel zum ver- und entschlüsseln verwendet werden. Auch bei Verwendung mittels eines PKEs ist es notwendig, dass sich alle Beteiligten auf einen konkreten Algorithmus verständigen.

Das Grundprinzip des PKE-Verfahrens besteht darin, dass Schlüssel immer Paarweise verwendet werden müssen. Statt eines einzelnen Schlüssels, wird ein Schlüsselpaar erzeugt. Wird mit einem dieser zwei Schlüssel verschlüsselt, muss der andere zum entschlüsseln verwendet werden. Des weiteren sollte es praktisch[2] unmöglich oder sehr aufwändig (= unbezahlbar) sein, bei Kenntnis eines Schlüssels, den korrespondierenden Schlüssel zu errechnen.

Wenn Alice eine verschlüsselte Nachricht an Bob senden möchte, ist es Notwendig, dass Bob ein Schlüsselpaar erzeugt hat. Einer der beiden Schlüssel wird von da an als öffentlichen Schlüssel (Publik Key) verwendet und öffentlich zugänglich gemacht. Der andere wird als privater Schlüssel (Private Key) verwendet und geheimgehalten. Alice wendet sich entweder direkt an Bob, oder fragt bei einem Publik Key Server nach, um Bobs öffentlicher Schlüssel zu erhalten. Der so erfragte Schlüssel wird verwendet um die Nachricht zu verschlüsseln. Die verschlüsselte Nachricht kann nur von jemandem gelesen werden, der über Bobs privaten Schlüssel verfügt. Bleibt nur zu hoffen, das Bob der einzige ist, der Zugang zu diesem Schlüssel hat. Der Besitz von Bobs öffentlichem Schlüssel ist nicht ausreichend um die Nachricht zu entschlüsseln.

Allerdings hat auch diese Sache einen Haken: Wer garantiert Alice, dass das Schlüssel mit der Aufschrift “Bob” wirklich von Bob stammt? Trudy könnte ein Schlüsselpaar erzeugt haben und behaupten, sie sei Bob. Wie schon erwähnt,

[2] d.h. rechnertechnisch

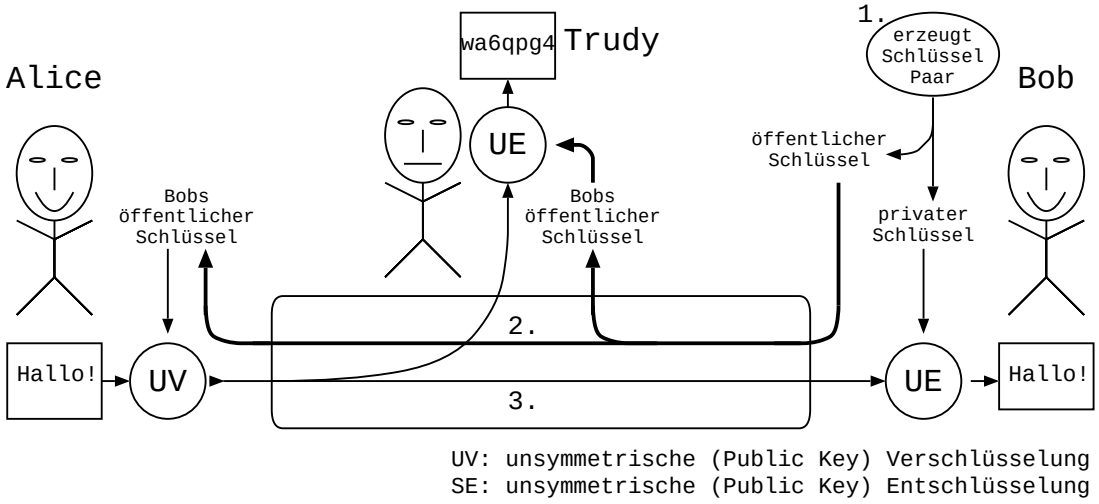

Abbildung 7: Alice sendet Bob eine Nachricht unter Verwendung eines PKE-Verfahrens.

können sich Alice und Bob persönlich treffen oder sich gegenseitig anrufen um die Schlüssel auszutauschen. Dies setzt aber wieder voraus, dass sich beide kennen.

Ein weiteres Problem ist, dass es für Bob nicht Möglich ist, festzustellen, ob Alice die Nachricht gesendet hat. Jemand anders könnte sich als Alice ausgeben.

1.4 Unterschreiben mit Hashing

Um die beiden eben genannten Probleme zu beseitigen, kommt ein weiteres mathematisches Verfahren zum Einsatz: das Hashing.

Dieses mathematische Verfahren wird durch sechs Bedingungen definiert, welche an es gestellt werden:

1. Das Verfahren berechnet aus einer Eingabe mit beliebiger Länge, eine Ausgabe mit fester Länge. Identische Eingaben führen immer zu der gleichen Ausgabe.
2. Bei Kenntnis der Ausgabe ist es rechnerisch unmöglich auf die Eingabe zu schließen.
3. Bei vorgegebener Ausgabe, ist es rechnerisch unmöglich eine Eingabe zu finden, welche bei Anwendung des Hash-Verfahrens zur gewünschten Ausgabe führt.

4. Es ist rechnerisch unmöglich bei vorgegebener Eingabe eine weitere Eingabe zu finden, welche bei Anwendung des Hash-Verfahrens zu der gleichen Ausgabe führt.

5. Es ist rechnerisch unmöglich zwei Eingaben zu erzeugen, welche verschieden sind und die gleiche Ausgabe hervorrufen, wenn das Hash-Verfahrens auf sie angewendet wird.

6. Es ist sehr unwahrscheinlich, dass Zufällig zwei Eingaben entdeckt werden, die bei Anwendung des Hash-Verfahrens die gleiche Ausgabe hervorrufen.

Beispiele für solche Hash-Verfahren sind:

1. Secure-Hash-Algorithmus (SHA-1)

2. Message-Digest-Algorithmus (MD5)

Zurück zum Beispiel: Alice sendet eine verschlüsselte Nachricht an Bob. Zusätzlich hängt sie eine Signatur (Unterschrift) an ihre Nachricht an. Mit Hilfe von dieser kann Bob überprüfen, ob die empfangene Nachricht wirklich von Alice gesendet wurde. Um die Signatur zu errechnen, wendet Alice zuerst das Hash-Verfahren auf ihre Nachricht[3] an und verschlüsselt dann die Ausgabe des Verfahrens mit ihrem privatem Schlüssel.

Bob wendet seinerseits das Hash-Verfahren auf die empfangene Nachricht an. Die empfangene Signatur entschlüsselt er mit Alices öffentlicher Schlüssel. Stimmt beides (Ausgabe des Hash-Verfahrens und entschlüsselte Signatur) überein, ist bewiesen, dass die Nachricht mit Alices privatem Schlüssel unterschrieben wurde. Dies ist durch die paarweise Anwendung der Schlüssel zu erklären: Es muss immer mit dem korrespondierenden Schlüssel entschlüsselt werden, egal, ob es sich um den öffentlichen oder privaten Schlüssel handelt. Jemand, der nicht über Alices privaten Schlüssel verfügt, kann daher keine gültige Unterschrift erzeugen. Zusätzlich ist bewiesen, dass die Nachricht nicht verändert wurde. Denn eine Veränderung der Nachricht würde zu einer veränderten Hash-Ausgabe führen.

An dieser Stelle tritt wieder Hacken 1 auf: Woher weiß Bob, dass Alices öffentlicher Schlüssel wirklich zu Alice gehört?

[3] Ob das Hash-Verfahren auf die originale Nachricht oder auf die bereits verschlüsselte Nachricht angewendet wird, spielt hierbei keine Rolle. Es muss allerdings beim Sender und Empfänger auf die gleiche Weise gehandhabt werden.

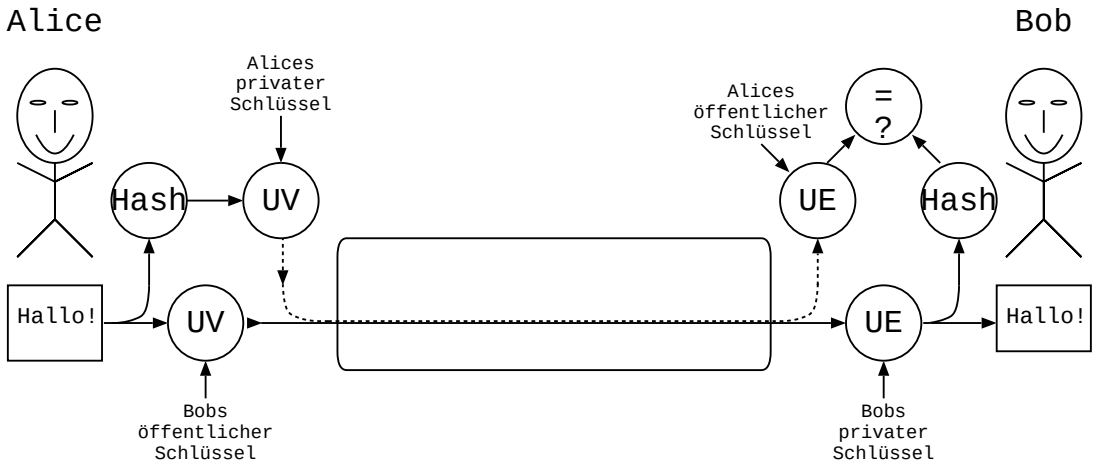

Abbildung 8: Alice sendet Bob eine Nachricht unter Verwendung eines PKE-Verfahrens. Zusätzlich unterschreibt sie ihre Nachricht. Bob prüft die Unterschrift.

1.5 Public Key Infrastructure (PKI)

Dieser Abschnitt beschäftigt sich mit PKI – einem System um zu zertifizieren, dass ein öffentlicher Schlüssel (bzw. ein Schlüsselpaar) zu einer bestimmten Person gehört. Eine PKI verwendet PKE und Hashing um Zertifikate über diesen Sachverhalt auszustellen.

Solch ein Zertifikat besteht im allgemeinen aus diesen Teilen:

1. Angaben zum Schlüsselhalter: Name, Organisation, Adresse, etc
2. Organisatorische Angaben: Z.B. Zeitraum der Gültigkeit und Aussteller des Zertifikates
3. Schlüssel des Halters, welcher zertifiziert wird
4. Unterschrift

1.5.1 Erstellung eines Zertifikates

Zur Erstellung eines Zertifikates werden alle Angaben, außer der Unterschrift, aneinander gehängt. Auf das Ergebnis dieser Verkettung wird das Hash-Verfahren angewendet. Die Ausgabe des Hash-Verfahrens wird zur Unterschrift indem sie mit dem privatem Schlüssel der unterschreibenden Person verschlüsselt wird. Das

fertige Zertifikat ist eine Verkettung der bereits erwähnten Verkettung und der Unterschrift.

Sendet Alice eine Nachricht an Bob, kann sie anhand des Zertifikates überprüfen, ob der ihr vorliegende Schlüssel zu Bob gehört. In diesem Fall kann sie ihre Nachricht mit diesem Schlüssel verschlüsseln. Hat sie ihre Nachricht mit dem weiter oben genannten Verfahren unterschrieben, kann Bob die Echtheit (Nachricht ist von Alice) und Unversehrtheit (Nachricht ist unverändert) der selbigen überprüfen. Dazu überprüft er die Nachricht mit Hilfe des ihm vorliegenden Schlüssels (Alices öffentlicher Schlüssel) anhand der Unterschrift und den Schlüssels anhand des Zertifikates.

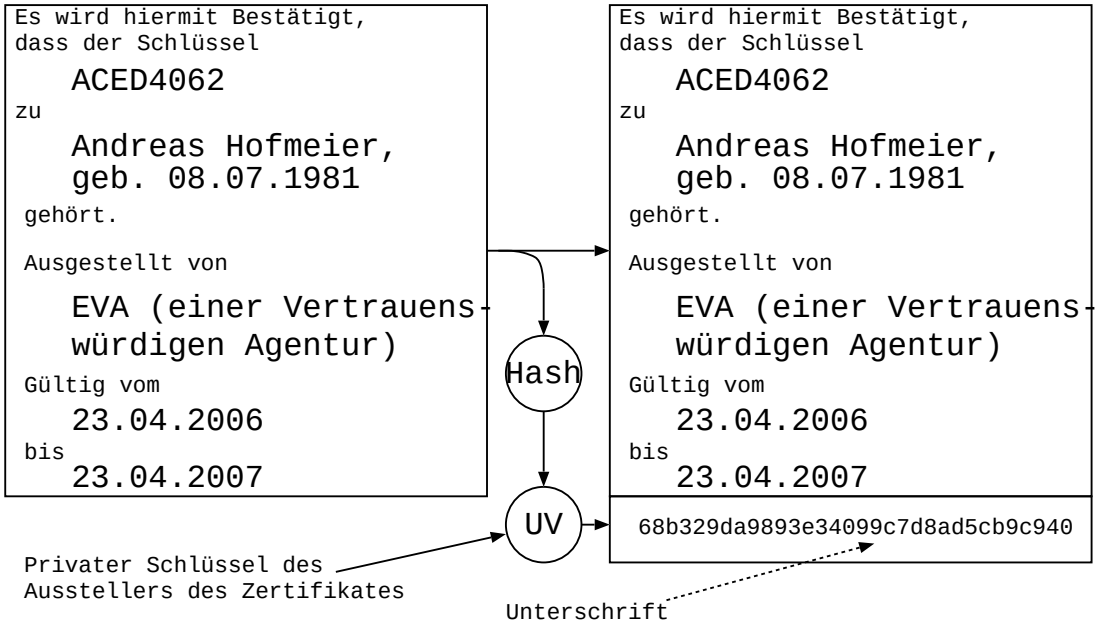

Abbildung 9: Beispiel eines digitalen Zertifikates und wie es erstellt wird.

1.5.2 Die Grundstruktur der PKI

Bleibt nur eine Frage: Wer Zertifiziert die Schlüssel von Alice und Bob?

Das könnte zum Beispiel eine von beiden gekannte und als vertrauenswürdig eingestufte dritte Person sein. Soll dies in großen Maßstäben funktionieren, muss hier eine Struktur hineingebracht werden. Wir nähern uns der Public Key Infrastructure.

Die PKI geht von einer absolut vertrauenswürdigen Person (oder Organisation)

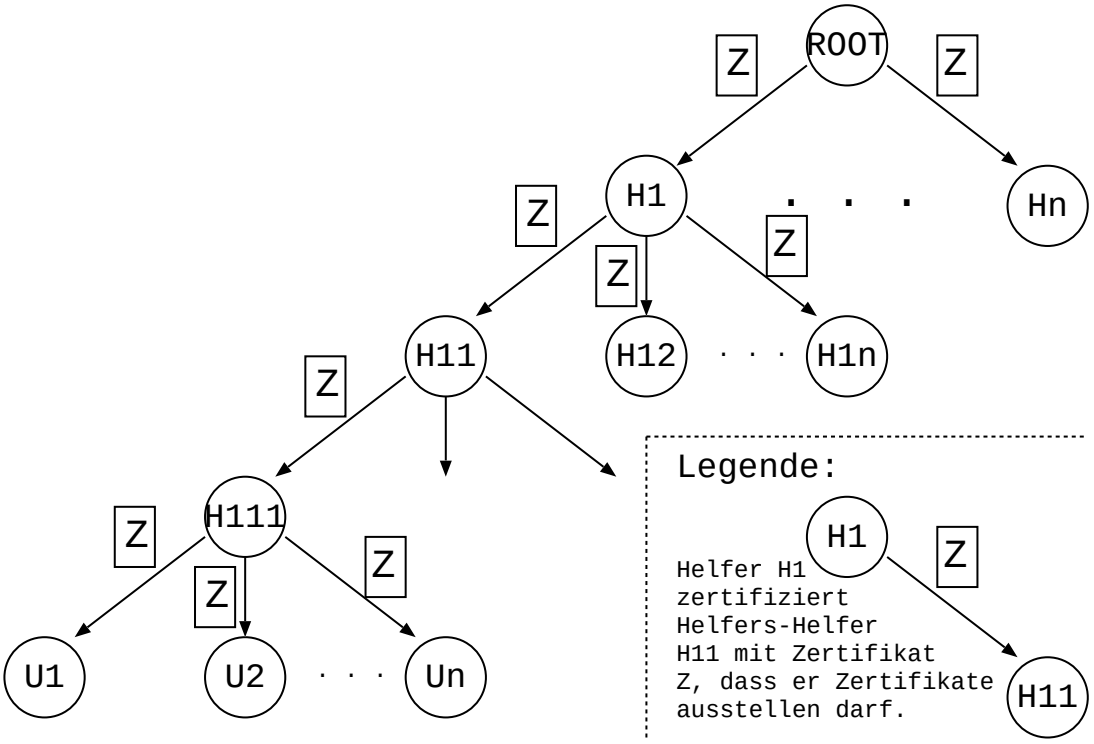

Abbildung 10: Beispiel für eine PKI (Baum zertifiziertem Helfern)

aus. Diese wird in diesem Zusammenhang als Root[4] bezeichnet. Diese Person wäre völlig überfordert Millionen von Schlüsseln zu zertifizieren. Aus diesem Grund braucht diese Person Helfer. Es wäre sehr gefährlich für die Root-Person ihren privaten Schlüssel an ihre Helfer weiter zu geben. Wenn einer der Helfer eine undichte Stelle darstellte, wären alle Zertifikate unbrauchbar geworden. Hat die PKI einen großen Maßstab erreicht, sagen wir weltweit, wäre dieser Fall eine Katastrophe. Um dies zu verhindern, verwenden die Helfer nicht den privaten Schlüssel des Roots, sondern ihren eigenen. Der Root stellt ein Zertifikat aus, welches dem Helfer bescheinigt, dass er im Namen des Roots Zertifikate ausstellen darf. Das ausgestellte Zertifikat folgt den bereits beschriebenen Muster. Zusätzlich wird unter "Organisatorisches" ein Vermerk eingefügt, dass diese Person berechtigt ist Zertifikate auszustellen. Wenn die PKI weltweit ausgedehnt ist, wäre es für den Root immer noch zu aufwändig jeden einzelnen Helfer zu Zertifizieren. Auf der anderen Seite können Helfer auch keine unendliche Mengen an Zertifikaten ausstellen, aber ihren privaten Schlüssel auch nicht weitergeben. Das führt dazu, dass die Helfer ihrerseits wieder Helfer einstellen und zertifizieren müssen. Das ganze läuft auf eine Baumstruktur hinaus. Der Root zertifiziert seine Helfer, seine Helfer zertifizieren ihre Helfer, und so weiter. Dies wird so lange durchgeführt, bis

[4]eng. Wurzel, hier: ranghöchte Person

die Menge an Arbeit und Verantwortung auf ein gesundes Maß aufgeteilt wurde. Die Helfer mit dem niedrigsten Rang stellen im allgemeinen die Zertifikate für die User aus. Helfer höherer Ebenen können ebenfalls User zertifizieren. Die User ihrerseits sind nicht berechtigt im Namen des Roots Zertifikate auszustellen. Eine solche Organisation (Root + alle Helfer) wird im als Certificate Authority (CA) bezeichnet.

Sollte einer der Helfers-Helfers-...Helfer undicht werden, müssten lediglich die vom ihm unterschriebenen Zertifikate für ungültig erklärt werden. Wie sollte es anders sein: Dies wird erwartungsgemäß über ein Revoke (Widerruf) Zertifikat gemacht. Dieses Zertifikat enthält das widerrufene Zertifikat, einen Vermerk "widerrufen" und eine Unterschrift eines in der Rangordnung höhergestellten Helfers. Mit dieser Methode lassen sich auch einzelne Zertifikate widerrufen, wenn sich herausstellt, dass bei ihrer Ausstellung ein Fehler unterlaufen ist.

1.5.3 Prüfung eines Schlüssels anhand eines Zertifikates

Sinn dieser Prüfung ist es, festzustellen, ob der vorliegende Schlüssel tatsächlich zu der genannten Person gehört. Dazu wird das von einem Helfer ausgestellte Zertifikat – mit Hilfe des öffentlichen Schlüssels des Helfers – überprüft. Dies geschieht indem der Inhalt des Zertifikates "gehashed" wird und die Unterschrift mit dem öffentlichen Schlüssel entschlüsselt wird. Stimmt beides überein, wird das Zertifikat als gültig anerkannt. Daraufhin wird ein Zertifikat für den Helfer gesucht, welches diesem gestattet seinerseits Zertifikate auszustellen. Das gefundene Zertifikat wird mit Hilfe des öffentlichen Schlüssels des Ausstellers überprüft. Für den Aussteller wird seinerseits wieder ein Zertifikat gesucht, usw. Dies wird solange wiederholt, bis die Root erreicht wurde. Gleichzeitig wird immer nach Revoke-Zertifikaten gesucht. Suchstädten für diese Zertifikate sind öffentliche Schlüssel-Server. Voraussetzung für die Anwendung dieses Verfahrens ist, dass der öffentliche Schlüssel der Roots bekannt ist (d.h. lokal vorliegt). Also das Ziel der Suche definiert ist. Ist dies nicht der Fall, könnte sich jeder als CA ausgeben, indem er einen vom ihm erzeugten Root vorgibt.

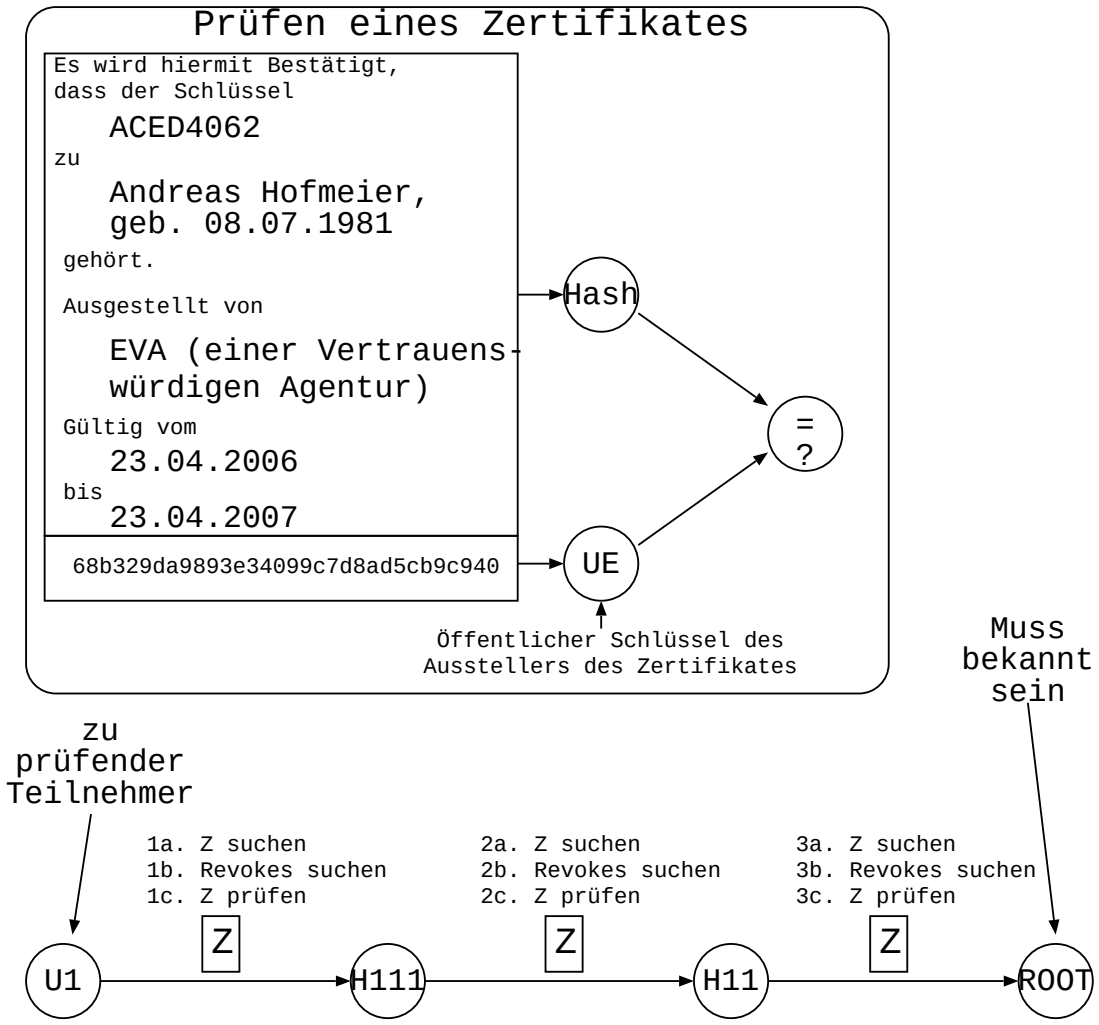

Abbildung 11: Prüfen eines Zertifikates. Bezug auf vorhergehendes PKI-(Baum-)Beispiel.

1.5.4 Konkrete Umsetzung

In der Realität stellt sich das Problem, einen Root zu finden, dem alle vertrauen. Bisher wurde keine Lösung für dieses Problem gefunden. Aus diesem Grund existieren viele Roots. Viele Firmen haben die Zertifizierung zu ihrem Geschäft gemacht. Des weiteren stellen viele Organisationen einfach ihren eigenen Root. Soll das Zertifikat eines Teilnehmers überprüft werden, muss dieser von mindestens einem Helfer eines bekannten Roots zertifiziert sein. Am Beispiel des Web-Browsers lässt sich dies gut erkennen: Die meisten Web-Browser bringen eine Liste mit vom Hersteller anerkannten Roots mit. Reicht diese Liste nicht aus, da zum Beispiel der Root-Schlüssel der Hochschule Bremen nicht in die Liste aufge-

nommen wurde, kann dieser manuell hinzugefügt werden. Von da an werden alle von der Hochschule Bremen zertifizierte Server vom Web-Browser anerkannt.

Beispiel für ein konkretes Format für ein Zertifikat ist das `X.509`-Format, welches breite Anwendung findet und durch RFCs standardisiert ist.

Auswahl an von Mozilla anerkannten CAs: AOL, GlabalSign, VISA, VeriSign.

Diese Organisationen verlangen viel Geld für das Zertifizieren. Daher hat sich eine private Organisation gebildet, die eine Form von kostenlosem Vertrauensnetzwerk aufbaut, um zu zertifizieren: `www.CAcert.org`. Jeder der von drei zertifizierten Personen zertifiziert wurde, kann selbst Zertifikate (u.a. für andere Personen und für WEB-Server) ausstellen. Dieser Ansatz ist allerdings noch nicht von den gängigen Web-Browsern-Herstellern akzeptiert worden.

1.6 Vor- und Nachteile

Unsymmetrische Verschlüsselung ist langsamer, aufwendiger und unsicherer im Vergleich zu der Symmetrischen. Um den Sicherheitsverlust zu kompensieren, werden längere Schlüssel verwendet. Dadurch wird die unsymmetrische Verschlüsselung noch langsamer und aufwendiger.

Unsymmetrische Verschlüsselung hat den Vorteil, dass Schlüssel immer paarweise verwendet werden müssen. Dies begünstigt und vereinfacht den Schlüsselaustausch und ist notwendig zum Aufbau einer PKI.

Aufgrund des hohen Aufwands des unsymmetrischen Verfahrens, wird es in der Praxis nur zum verschlüsseln von Schlüsseln eingesetzt. Das heißt, dass eine zu sendende Nachricht mit einem symmetrischen Verfahren verschlüsselt wird, wobei der Schlüssel in diesem Fall ein Zufallswert ist. Dieser Schlüssel wird mit einem unsymmetrischen Verfahren verschlüsselt und zusammen mit der verschlüsselten Nachricht übermittelt. Der Empfänger braucht das unsymmetrische Verfahren um den Schlüssel zu entschlüsseln. Erst im nächsten Schritt kann er mit diesem Schlüssel die Nachricht entschlüsseln. Dieses Vorgehen kombiniert die Stärken beider Verfahren.

1.7 Verschiedene Ebenen der Verschlüsselung

Verschlüsselung kann auf verschiedenen Ebenen angesetzt werden:

1.7.1 Application Layer

Die Anwendung übernimmt das Verschlüsseln der Daten. (Siehe Abbildung 12, Punkt 1) Ein Beispiele hierfür ist PGP/GnuPG[5], ein Programm zum Verschlüsseln von eMails und Dateien. Bei dieser Form der Verschlüsselung handelt es sich fast immer um eine End-zu-End Verschlüsselung.

Vorteil dieser Lösung ist, dass nicht in die Netzwerkkonfiguration des Computers eingegriffen werden muss. Allerdings muss die Anwendung in der Lage sein zu verschlüsseln. Diese Lösung ist von Vorteil, wenn die Verschlüsselung Anwendungsspezifisch ist.

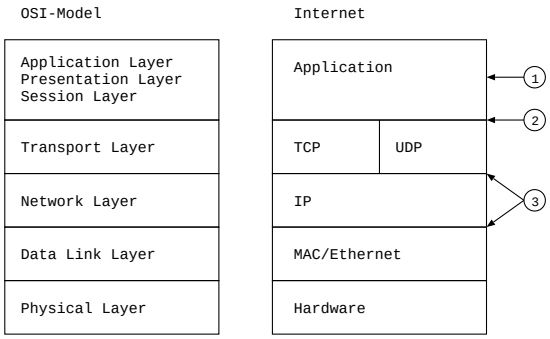

Abbildung 12: Ebenen der Verschlüsselung

1.7.2 Zusätzlicher Layer zwischen Anwendung und Transport Layer

Die Anwendung setzt auf eine Bibliothek auf um die Verschlüsselung zu realisieren. (ehe Abbildung 12, Punkt 2) Eine weit verbreitete Bibliothek hierfür ist OpenSSL, welche den SSL/TLS-Standard implementiert. Die am weit verbreitetste Anwendung hierfür ist HTTPS, welches die SSL-verschlüsselte Variante des HTTP[6]-Protokolls darstellt.

[5]Siehe `http://de.wikipedia.org/wiki/GnuPG`

[6]Protokoll welches vom Web-Browser verwendet wird um Web-Seiten anzufordern.

Bei dieser Herangehensweise muss das Anwenderprogramm neu kompiliert werden. Dies macht oftmals kleinere Änderungen am Programmcode erforderlich, um die Bibliothek einzubinden. Ein Alternative besteht darin, das Programm so zu konfigurieren, dass es alle Netzwerkzugriffe über einen speziellen Proxy (z.B. sslwrap[7]) durchführt. Dieser ist dann für die Verschlüsselung zuständig.

1.7.3 Network Layer – Tunneln

Bei dieser Methode werden alle Pakete in neue Pakete eingepackt oder mit zusätzlichen Headern versehen. (ehe Abbildung 12, Punkt 3) Die Verschlüsselung ist vollkommen transparent. Weder die Anwendung noch der User bemerken sie. Die Verschlüsselung wird vom Netzwerk-Stack des Betriebssystem realisiert oder durch diesen an eine User-Level-Anwendung weitergereicht. Beispiele hierfür sind OpenVPN (auf welches in den folgenden Kapiteln eingegangen wird), IPsec und PPTP.

Vorteil ist hierbei, dass keinerlei Eingriffe in das Anwenderprogramm notwendig sind. Allerdings sind Eingriffe in die Netzwerkkonfiguration des Computers notwendig.

[7] `http://www.rickk.com/sslwrap/`

2 VPN – Virtual Private Network

2.1 Übersicht über Verschiedene Techniken

Es folgt eine Übersicht über die am häufigsten verwendeten VPN-Systeme. Zudem werden Vor- und Nachteile der Lösungen angeschnitten.

2.1.1 OpenVPN

- Weiterführende Informationen: www.OpenVPN.net
- Protokoll: Nutzt den Transport Layer. SSL/TLS über TCP oder UDP.
- Vorteile:
  - + Einfache Konfiguration
  - + Einfache Installation
  - + Verwendet TUN/TAP (virtueller Netzwerk Treiber). Kein Low-Level-Kernel-Treiber erforderlich. Läuft im User-Mode.
  - + Ein einziger erreichbarer UDP oder TCP-Port auf dem Server ist ausreichend. Daher kann der Klient eine dynamische IP haben und/oder hinter einer Firewall angesiedelt sein.
  - + Kann auch über einen HTTP-Proxy getunnelt werden. (TCP Mode erforderlich)
  - + Betriebssysteme: Linux, Windows 2000/XP oder höher, OpenBSD, FreeBSD, NetBSD, Mac OS X, und Solaris.
  - + Password (preshared key) oder PKI Authentifizierung
  - + Vollständig Frei: GPL
  - + Modularer Aufbau. Verwendet OpenSSL-Bibliothek. Daher können neue Algorithmen durch ein Update dieser Bibliothek auch in OpenVPN verwendet werden.
- Nachteile:
  - - Eingeschränkte Anwendungsmöglichkeiten
  - - Läuft nicht auf Windows 98 und darunter
  - - Vergrößerter Overhead gegenüber IPsec

2.1.2 IPsec

- Weiterführende Informationen: `http://www.google.com/search?q=IPSec`
- Protokoll: Nutzt den Network Layer. Ein Paket kann übertragen werden, indem es in einem neuen Paket komplett eingeschlossen wird (Neuer IP-Header: Tunnel Mode) oder indem ein zusätzlicher Header angefügt wird (Transport Mode).
- Vorteile:
  + Weitläufig anerkannter Standard
  + Sehr weites Spektrum der Einsatz- und Konfigurationsmöglichkeiten
  + Hardware-Implementationen (z.B. Router) verfügbar
  + Treiber für fast jedes Betriebssystem
- Nachteile:
  - Aufwendige und langwierige Konfiguration
  - Es ist oftmals schwierig verschiedene IPsec-Implementationen zusammenarbeiten zu lassen (trotz Standard)

2.1.3 PPTP: MPPE

Dies steht für: Point-to-Point Tunneling Protokoll: Microsoft^{TM} Point-to-Point Encryption Protokoll. Leitet eine Point-to-Point-Verbindung über einen TCP Stream.

- Weiterführende Informationen: `http://pptpclient.sourceforge.net/`, `http://www.networksorcery.com/enp/protocol/mppe.htm`
- Protokoll: Transport Layer. Leitet eine PPP-Verbindung über einen TCP Stream.
- Vorteile:
  + Klient wird als Bestandteil von Microsoft Windows ausgeliefert
  + Einfache Konfiguration unter Microsoft Windows.
  + Ein einziger erreichbarer TCP-Port auf dem Server ist ausreichend. Daher kann der Klient eine dynamische IP haben und/oder hinter einer Firewall angesiedelt sein.

- Nachteile:
  - Instabile Verbindung
  - Sehr aufwendige Installation auf vielen nicht MS Betriebssystemen.
  - Vergrößerter Overhead gegenüber IPsec

2.2 Funktionsweise von OpenVPN

Beim Aufbauen eines virtuellen Tunnels wird zuerst eine Verbindung zum Server mittels TCP oder UDP hergestellt. Über diese Verbindung wird dannach eine SSL-Verschlüsselungschicht gelegt. (Siehe Abb. OpenVPN Network Stack) Die SSL Schicht wird wird mittels OpenSSL[8], einer Freien SSL-Bibliothek, realisiert. Ist die Verbindung hergestellt und der Klient bzw. der Server authentifiziert, wird das Virtuelle Netzwerkinterface angelegt und mit dem OpenVPN-Treiber verbunden. Dannach werden Netzwerkeinstellungen (IP-Adressen, Routing, etc) ausgetauscht und am Netzwerkinterface eingestellt.

Für das Anwenderprogramm erscheint das virtuelle Netzwerkinterface (mittlere Abbildung von Graphik OpenVPN Network Stack) wie ein normales Netzwerkinterface. Mit dem kleinen Unterschied, dass unter der IP-Schicht (Network-Layer) nicht das erwartete Ethernet folgt, sondern OpenVPN. Alle Pakete, welche an diese Interface gesendet werden, gehen an den OpenVPN-Treiber und nicht wie erwartet über das Ethernet-Protokoll an eine Hardware.

OpenVPN seinerseits nimmt diese Pakete entgegen. Leitet diese dann zwecks Verschlüsselung durch OpenSSL und reicht sie schließlich an das reale Netzwerk (rechte Abbildung von Graphik OpenVPN Network Stack) weiter. Dieser übermittelt das Packe unter Verwendung von Ethernet an die Hardware. OpenVPN kann also als Anwendung im normalen Netzwerkstack verstanden werden. Wobei es gleichzeitig Netzwerktreiber für das virtuelle Interface ist.

Es ist auch möglich OpenVPN seinerseits wieder durch ein virtuelles Netzwerkinterface hindurch arbeiten zu lassen. (Aufgrund der Schichtenteilung macht dies für die jeweilige Anwendung keinen Unterschied.) Hierbei handelt es sich um einen Tunnel über einen anderen Tunnel.

Wird dieser Vorgang von der Sicht der Endanwendung her betrachtet, ergibt sich folgendes Bild:

[8] `http://www.openssl.org/`

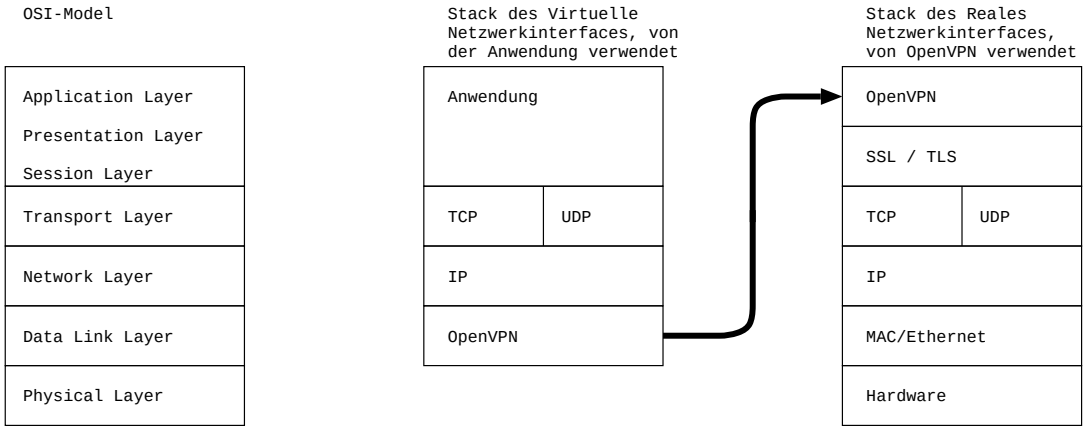

Abbildung 13: OpenVPN Network Stack

Bei dem Durchlaufen jeder Schicht im Netzwerkstack wird ein Header angefügt. Die Anwendung überträgt ihre Daten (z.B. ein Webbrowser eine HTTP-Anfrage) an das Virtuelle Netzwerkinterface, welches einen TCP- (oder andere, z.B. UDP-) und einen IP-Header anfügt. OpenVPN nimmt dieses Paket entgegen und fügt einen eigenden Header an. Dann leitet OpenVPN es durch die SSL-Schicht, welche das bisherige Paket verschlüsselt und ebenfalls einen Header anfügt. Dannach durchläuft das Paket wieder den normalen Netzwerkstack mit TCP (oder UDP) und IP. Alle diese Schichten fügen jeweils einen neuen Header an. Ein 'OpenVPN-Paket' enthält also einen neuen IP-, TCP- sowie einen SSL-Header. Der Payload (Nutzlast) besteht aus dem eingeschlossenen Paket, welches verschlüsselt worden ist. Die folgenden Graphik soll diesen Zusammenhang verdeutlichen. In ihr werden die Pakete von rechts nach links aufgebaut.

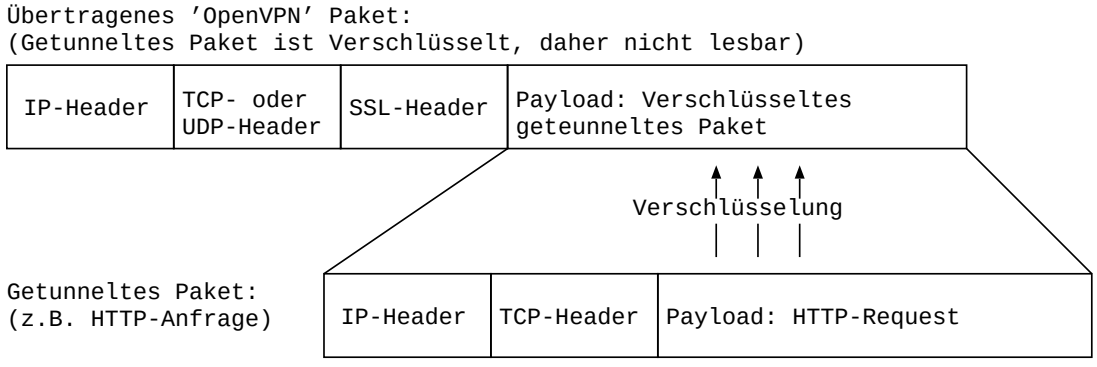

Abbildung 14: Beispiel eines Paketes einer HTTP-Anfrage, durch OpenVPN getunnelt

3 Beispiel eines VPNs mittels OpenVPN

In diesem Abschnitt werden der Aufbau eines VPN zwischen zwei Computern mittels OpenVPN und einige Experimente mit diesem Tunnel beschrieben.

3.1 Verwendete Computer und Software

Bei dem experimentellen Aufbau wurde auf die im folgenden beschriebene Software und Hardware zurückgegriffen:

3.1.1 Software

- Kernel: Linux 2.4.27
- Betriebsystem: Debian GNU/Linux, Version 3.1 (Sarge). Neben einer netzwerkfähigen Basisinstallation wurden folgende Pakete verwendet:

  `openvpn` in Version 2.0. OpenVPN ist abhängig von `debconf` (Konfigurations-Management-System von Debian) `libc6` (Grundlegende Bibliothek, für C-Programme), `liblzo1` (Bibliothek für Echtzeit Kompression) und `libssl0.9.7` (Implementation des Verschlüsselungslayers SSL/TLS als Shared-Library).

3.1.2 LBlacky (Server)

- Laptop: HP omnibook 6000
- Pentium III 850MHz, L1-Cache 16KB, L2-Cache: 256KB, 512MB RAM
- IP-Adresse: `192.168.1.3`

3.1.3 Blacky (Client)

- Laptop: Siemens Scenic Mobile 510 AGP
- Pentium II 333MHz, L1-Cache 16KB, L2-Cache: 256KB, 128MB RAM
- IP-Adresse: `192.168.1.2`

3.2 Erstellen der PKI

Im folgenden wird das Erstellen der Zertifikate beschrieben. Das Vorgehen und die verwendeten Scripte wurden von `http://openvpn.net/easyrsa.html` adaptiert. Unter der Adresse `http://docs.vernstok.nl/openvpn/examples/easy-rsa/` können die Skripte direkt heruntergeladen werden. Alle hier verwendeten Skripte sind im Anhang A zu finden.

3.2.1 Setzten von Umgebungsvariablen

Als erstes müssen Umgebungsvariablen zur Steuerung der Schlüsselerzeugung gesetzt werden:

```
export KEY_CONFIG=$HOME/.vpn_keys/openssl.cnf
export KEY_DIR=$HOME/.vpn_keys
export KEY_SIZE=2048
export KEY_COUNTRY=DE
export KEY_PROVINCE=NA
export KEY_CITY=Bremen
export KEY_ORG="ABM-Hofmeier-VPN"
export KEY_EMAIL="andreas@abmh.de"
```

Dies wird mit Hilfe des `00-init-vars`-Skriptes erledigt. Diese Skript muss für den jeweiligen Zweck angepasst werden.

```
andreas@LBlacky:~/Projects/DerTunnel/easyrsa > . 00-init-vars
NOTE: when you run ./clean-all, I will be doing a rm -rf on /home/andr\
eas/.vpn_keys
```

3.2.2 Erstellung des Roots der CA

Der nächste Schritt ist das Erzeugen einer Certificate Authority (CA). Dazu muss ein Schlüsselpaar für die Root dieser CA erzeugt werden.

Erzeugte Dateien:

- `ca.key`: Privater Schlüssel des CA-Root
- `ca.crt`: Öffentlicher Schlüssel des CA-Root

```
andreas@LBlacky:~/Projects/DerTunnel/easyrsa > . 01-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) [NA]:
Locality Name (eg, city) [Bremen]:
Organization Name (eg, company) [ABM-Hofmeier-VPN]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:Andreas Hofmeier
Email Address [andreas@abmh.de]:
```

3.2.3 Erstellung der Diffie-Hellman-Parameter

Zum Betrieb des Servers sind Diffie-Hellman-Parameter notwendig. Diese werden in diesem Schritt erzeugt und in der Datei dh2048.pem abgelegt. Das Diffie-Hellman-Verfahren ermöglicht es Schlüssel sicher durch ein unsicheres Medium hindurch auszutauschen, ohne die Voraussetzung eines vorherigen Austausches. Nachteil des Verfahrens ist, dass die Gegenseite nicht authentifiziert werden kann. Aus diesem Grund wird dieses Verfahren nur als erste Stufe zum Aufbau einer sicheren Verbindung verwendet. Die Authentifizierung erfolgt erst später über die PKI. Der folgende Befehl produzierte jede Menge Punkte und brauchte 34 Minuten auf LBlacky (850MHz) bei voller CPU Auslastung.

Erzeugte Dateien:

- `dh2048.pem`: Diffie-Hellman-Parameter in Datei.

```
andreas@LBlacky:~/Projects/DerTunnel/easyrsa > . 02-build-dh-helpc
Generating DH parameters, 2048 bit long safe prime, generator 2
This is going to take a long time
...........+.....
[...]
```

3.2.4 Erstellung von Schlüsselpaaren für Server und Klienten

Während der erste Schritt (`01-build-ca`) normalerweise nur einmal für ein Netzwerk durchgeführt wird, müssen die anderen unter Umständen häufiger ausgeführt werden.

Für jeden Server muss Schritt zweit (`02-build-dh-helpc`), Schritt vier (`04-build- req-server`) und Schritt fünf (`05-sign-key`) ausgeführt werden.

Für jeden Klienten müssen die folgenden Schritte durchlaufen werden: drei (`03-build-req`) und fünf (`05-sign-key`).

Alle neu erzeugten oder geänderten Dateien sind an den Ort zu kopieren, wo sie von OpenVPN gefunden werden. Bzw. wo OpenVPN konfiguriert ist sie zu suchen.

Hoffen wir, dass kein unterschriebendes Zertifikat mit samt des zugehörigen privaten Schlüssels den rechtmäßigen Eigentümer verlässt. In diesem glücklichen Fall ist der Schritt sieben und das dazugehörige Skript `07-revoke` überflüssig.

Erzeugung das Schlüssel-Paares für den Klienten `Blacky`:

Erzeugte Dateien:

- `blacky.key`: Blackys privater Schlüssel
- `blacky.csr`: Blackys öffentlicher Schlüssel als Certificate Request (Zertifikat Anfrage)

```
andreas@LBlacky:~/Projects/DerTunnel/easyrsa > . 03-build-req blacky
Generating a 2048 bit RSA private key
..............................+++
.....................................................................
.....................................................+++
writing new private key to 'blacky.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) [NA]:
```

```
Locality Name (eg, city) [Bremen]:
Organization Name (eg, company) [ABM-Hofmeier-VPN]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:Blacky
Email Address [andreas@abmh.de]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Erzeugung des Schlüssel-Paares für den Server `LBlacky`:

Erzeugte Dateien:

- `lblacky.key`: LBlackys privater Schlüssel
- `lblacky.csr`: LBlackys öffentlicher Schlüssel als Certificate Request (Zertifikat Anfrage)

```
andreas@LBlacky:~/Projects/DerTunnel/easyrsa > . 04-build-req-server lblacky
Generating a 2048 bit RSA private key
..............+++
.....................................................................+++
writing new private key to 'lblacky.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) [NA]:
Locality Name (eg, city) [Bremen]:
Organization Name (eg, company) [ABM-Hofmeier-VPN]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:LBlacky
Email Address [andreas@abmh.de]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

3.2.5 Ausstellung der Zertifikate

Um die Übersichtlichkeit zu erhalten kommen in diesem Beispiel keine Helfer zum Einsatz. Alle Zertifikate werden direkt mit dem Root-Schlüssel unterschrieben. Unter der weiter oben genannten URL ist ein Skript zum erzeugen von Helfern (Intermediate) zu finden: `build-inter`

Bevor die Zertifikatanfrage unterschrieben und in ein Zertifikat umgewandelt werden kann, ist es notwendig die index-Datei und die Seriennummern-Datei anzulegen. In der Index-Datei wird eine Liste mit alle Schlüsseln abgelegt, welche zertifiziert wurden. Als Anfangswert reicht eine leere Datei, welche hier mit `touch` erzeugt wird. In der Datei `serial` wird eine Nummer abgelegt, welche bei jedem ausgestellten Zertifikat um eins erhöht wird. Anfangswert ist hier 0.

```
andreas@LBlacky:~/.vpn_keys > touch index.txt
andreas@LBlacky:~/.vpn_keys > echo 00 > serial
```

Ausstellung des Zertifikates für Blacky (Client)

Verwendete Dateien:

- `ca.key`: Privater Schlüssel des CA-Root. Dieser wird verwendet um die Zertifikate der anderen Schlüssel zu unterschreiben.
- `blacky.csr`: Blackys öffentlicher Schlüssel als Certificate Request (Zertifikat Anfrage)

Erzeugte Dateien:

- `blacky.crt`: Blackys öffentlicher Schlüssel als von CA-Root unterschriebendes Zertifikat

```
andreas@LBlacky:~/Projects/DerTunnel/easyrsa > . 05-sign-key blacky
Using configuration from /home/andreas/.vpn_keys/openssl.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'DE'
stateOrProvinceName   :PRINTABLE:'NA'
localityName          :PRINTABLE:'Bremen'
organizationName      :PRINTABLE:'ABM-Hofmeier-VPN'
commonName            :PRINTABLE:'Blacky'
emailAddress          :IA5STRING:'andreas@abmh.de'
```

```
Certificate is to be certified until Dec  1 20:05:11 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

andreas@LBlacky:~/.vpn_keys > cat serial
01
```

Ausstellung des Zertifikates für LBlacky (Server)

Verwendete Dateien:

- `ca.key`: Privater Schlüssel des CA-Root. Dieser wird verwendet um die Zertifikate der anderen Schlüssel zu unterschreiben.
- `lblacky.csr`: LBlackys öffentlicher Schlüssel als Certificate Request (Zertifikat Anfrage)

Erzeugte Dateien:

- `lblacky.crt`: LBlackys öffentlicher Schlüssel als von CA-Root unterschriebendes Zertifikat

```
andreas@LBlacky:~/Projects/DerTunnel/easyrsa > . 05-sign-key lblacky
Using configuration from /home/andreas/.vpn_keys/openssl.cnf
DEBUG[load_index]: unique_subject = "yes"
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'DE'
stateOrProvinceName   :PRINTABLE:'NA'
localityName          :PRINTABLE:'Bremen'
organizationName      :PRINTABLE:'ABM-Hofmeier-VPN'
commonName            :PRINTABLE:'LBlacky'
emailAddress          :IA5STRING:'andreas@abmh.de'
Certificate is to be certified until Dec  1 20:06:55 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
```

Wie zu erkennen ist, wird die Nummer in der Datei `serial` tatsächlich mit jedem unterschriebenen Zertifikat um eins erhöht.

```
andreas@LBlacky:~/.vpn_keys > cat serial
02
```

Die Datei `index.txt` enthält eine Liste mit bis zu diesem Zeitpunkt zertifizierten Schlüsseln:

```
andreas@LBlacky:~/.vpn_keys > cat index.txt
V       151201200511Z           00      unknown /C=DE/ST=NA/O=ABM-Hofm\
eier-VPN/CN=Blacky/emailAddress=andreas@abmh.de
V       151201200655Z           01      unknown /C=DE/ST=NA/O=ABM-Hofm\
eier-VPN/CN=LBlacky/emailAddress=andreas@abmh.de
```

3.2.6 Kopieren der Schlüssel

Nach Erstellung der Schlüssel sind diese an die entsprechenden Stellen zu kopieren, damit OpenVPN mit ihnen arbeiten kann.

Die Dateien `ca.crt`, `dh2048.pem`, `lblacky.crt` und `lblacky.key` werden auf den Computer LBlacky (Server) ins Verzeichnis `/etc/openvpn/` kopiert.

Die Dateien `ca.crt`, `blacky.crt` und `blacky.key` werden auf den Computer Blacky (Client) ins Verzeichnis `/etc/openvpn/` kopiert.

Es ist darauf zu achten, dass die privaten Schlüssel nicht in falsche Hände fallen. `ca.key` sollte extrem sicher aufbewahrt werden, damit nur die "guten Jungs" neue Zertifikate ausstellen können. Die für jeden Server erzeugte `dh2048.pem` sollte ebenfalls nur dem entsprechendem Server zugänglich sein. Dasselbe gilt auch für die Dateien `lblacky.key` und `blacky.key`, wobei die erste nur dem Server LBlacky und die zweite nur dem Klienten Blacky zugänglich sein sollte. Alle anderen Dateien sind sicherheitstechnisch unkritisch. Es muss trotzdem sichergestellt sein, dass niemand die Konfigurationsdateien, die Schlüssel und die Zertifikate auf dem Server austauschen oder verändern kann.

3.3 Aufbauen des VPNs

Im folgenden wird davon ausgegangen, dass die im Anhang B.1 aufgeführte `server.conf` im Verzeichnis `/etc/openvpn` vorhanden ist.

OpenVPN wurde manuell aufgerufen, was dazu führt, dass die Ausgaben auf den Bildschirm und nicht in der Logdatei erscheinen. Die Unterscheidung, ob OpenVPN als Server oder Client fungiert, wird durch die Konfigurationsdatei getroffen, nicht durch ihren Namen. Dieser kann beliebig gewählt werden.

Aufruf von OpenVPN. Ein Virtuelle Netzwerkinterface wird erstellt, dannach wartet der Server auf eingehende Verbindungen:

```
root@LBlacky:/etc/openvpn >  openvpn --config server.conf
Sat Dec  3 21:30:57 2005 OpenVPN 2.0 i386-pc-linux [SSL] [LZO] [EPOLL]\
 built on Nov  3 2005
Sat Dec  3 21:30:57 2005 Diffie-Hellman initialized with 2048 bit key
Sat Dec  3 21:30:57 2005 TLS-Auth MTU parms [ L:1560 D:140 EF:40 EB:0 \
ET:0 EL:0 ]
Sat Dec  3 21:30:57 2005 TUN/TAP device tun0 opened
Sat Dec  3 21:30:57 2005 ifconfig tun0 10.8.0.1 pointopoint 10.8.0.2 m\
tu 1500
Sat Dec  3 21:30:57 2005 route add -net 10.8.0.0 netmask 255.255.255.0\
 gw 10.8.0.2
Sat Dec  3 21:30:57 2005 Data Channel MTU parms [ L:1560 D:1450 EF:60 \
EB:23 ET:0 EL:0 AF:3/1 ]
Sat Dec  3 21:30:57 2005 Listening for incoming TCP connection on [und\
ef]:1194
Sat Dec  3 21:30:57 2005 TCPv4_SERVER link local (bound): [undef]:1194
Sat Dec  3 21:30:57 2005 TCPv4_SERVER link remote: [undef]
Sat Dec  3 21:30:57 2005 MULTI: multi_init called, r=256 v=256
Sat Dec  3 21:30:57 2005 IFCONFIG POOL: base=10.8.0.4 size=62
Sat Dec  3 21:30:57 2005 IFCONFIG POOL LIST
Sat Dec  3 21:30:57 2005 Blacky,10.8.0.4
Sat Dec  3 21:30:57 2005 Note: sys_epoll API is unavailable, falling b\
ack to poll/select API
Sat Dec  3 21:30:57 2005 MULTI: TCP INIT maxclients=1024 maxevents=102\
8
Sat Dec  3 21:30:57 2005 Initialization Sequence Completed
```

Der Server ist jetzt bereit einen Tunnel aufzubauen. Die folgenden Logs entstanden beim Server, als der Klient eine Verbindung aufbaute:

```
Sat Dec  3 21:31:04 2005 MULTI: multi_create_instance called
Sat Dec  3 21:31:04 2005 Re-using SSL/TLS context
Sat Dec  3 21:31:04 2005 LZO compression initialized
Sat Dec  3 21:31:04 2005 Control Channel MTU parms [ L:1560 D:140 EF:4\
0 EB:0 ET:0 EL:0 ]
Sat Dec  3 21:31:04 2005 Data Channel MTU parms [ L:1560 D:1450 EF:60 \
EB:23 ET:0 EL:0 AF:3/1 ]
Sat Dec  3 21:31:04 2005 Local Options hash (VER=V4): 'b695cb4a'
Sat Dec  3 21:31:04 2005 Expected Remote Options hash (VER=V4): 'bc077\
30e'
Sat Dec  3 21:31:04 2005 TCP connection established with 192.168.1.2:1\
067
```

```
Sat Dec  3 21:31:04 2005 TCPv4_SERVER link local: [undef]
Sat Dec  3 21:31:04 2005 TCPv4_SERVER link remote: 192.168.1.2:1067
Sat Dec  3 21:31:04 2005 192.168.1.2:1067 TLS: Initial packet from 192\
.168.1.2:1067, sid=8d73ef15 13c6e1ed
Sat Dec  3 21:31:06 2005 192.168.1.2:1067 VERIFY OK: depth=1, /C=DE/ST\
=NA/L=Bremen/O=ABM-Hofmeier-VPN/CN=Andreas_Hofmeier/emailAddress=andre\
as@abmh.de
Sat Dec  3 21:31:06 2005 192.168.1.2:1067 VERIFY OK: depth=0, /C=DE/ST\
=NA/O=ABM-Hofmeier-VPN/CN=Blacky/emailAddress=andreas@abmh.de
Sat Dec  3 21:31:06 2005 192.168.1.2:1067 Data Channel Encrypt: Cipher\
 'AES-128-CBC' initialized with 128 bit key
Sat Dec  3 21:31:06 2005 192.168.1.2:1067 Data Channel Encrypt: Using \
160 bit message hash 'SHA1' for HMAC authentication
Sat Dec  3 21:31:06 2005 192.168.1.2:1067 Data Channel Decrypt: Cipher\
 'AES-128-CBC' initialized with 128 bit key
Sat Dec  3 21:31:06 2005 192.168.1.2:1067 Data Channel Decrypt: Using \
160 bit message hash 'SHA1' for HMAC authentication
Sat Dec  3 21:31:06 2005 192.168.1.2:1067 Control Channel: TLSv1, ciph\
er TLSv1/SSLv3 DHE-RSA-AES256-SHA, 2048 bit RSA
Sat Dec  3 21:31:06 2005 192.168.1.2:1067 [Blacky] Peer Connection Ini\
tiated with 192.168.1.2:1067
Sat Dec  3 21:31:06 2005 Blacky/192.168.1.2:1067 MULTI: Learn: 10.8.0.\
6 -> Blacky/192.168.1.2:1067
Sat Dec  3 21:31:06 2005 Blacky/192.168.1.2:1067 MULTI: primary virtua\
l IP for Blacky/192.168.1.2:1067: 10.8.0.6
Sat Dec  3 21:31:07 2005 Blacky/192.168.1.2:1067 PUSH: Received contro\
l message: 'PUSH_REQUEST'
Sat Dec  3 21:31:07 2005 Blacky/192.168.1.2:1067 SENT CONTROL [Blacky]\
: 'PUSH_REPLY,route 10.8.0.1,ping 10,ping-restart 120,ifconfig 10.8.0.\
6 10.8.0.5' (status=1)
```

Es folgen die Ausgaben vom OpenVPN Klient, welche entstanden als der Klient manuell aufgerufen wurde und eine Verbindung aufbaute. Als Konfigurationsdatei wurde hier die `client.conf` verwendet, welche im Anhang B.2 aufgeführt wurde.

```
root@Blacky:/etc/openvpn > openvpn --config client.conf
Sat Dec  3 21:31:04 2005 OpenVPN 2.0 i386-pc-linux [SSL] [LZO] [EPOLL]\
 built on Nov  3 2005
Sat Dec  3 21:31:04 2005 IMPORTANT: OpenVPN's default port number is n\
ow 1194, based on an official port number assignment by IANA.  OpenVPN\
 2.0-beta16 and earlier used 5000 as the default port.
Sat Dec  3 21:31:04 2005 WARNING: No server certificate verification m\
ethod has been enabled.  See http://openvpn.net/howto.html#mitm for mo\
re info.
Sat Dec  3 21:31:04 2005 LZO compression initialized
Sat Dec  3 21:31:04 2005 Control Channel MTU parms [ L:1560 D:140 EF:4\
0 EB:0 ET:0 EL:0 ]
Sat Dec  3 21:31:04 2005 Data Channel MTU parms [ L:1560 D:1450 EF:60 \
EB:23 ET:0 EL:0 AF:3/1 ]
Sat Dec  3 21:31:04 2005 Local Options hash (VER=V4): 'bc07730e'
Sat Dec  3 21:31:04 2005 Expected Remote Options hash (VER=V4): 'b695c\
```

```
b4a'
Sat Dec  3 21:31:04 2005 Attempting to establish TCP connection with 1\
92.168.1.3:1194
Sat Dec  3 21:31:04 2005 TCP connection established with 192.168.1.3:1\
194
Sat Dec  3 21:31:04 2005 TCPv4_CLIENT link local: [undef]
Sat Dec  3 21:31:04 2005 TCPv4_CLIENT link remote: 192.168.1.3:1194
Sat Dec  3 21:31:04 2005 TLS: Initial packet from 192.168.1.3:1194, si\
d=19eb938d cd36ffd8
Sat Dec  3 21:31:05 2005 VERIFY OK: depth=1, /C=DE/ST=NA/L=Bremen/O=AB\
M-Hofmeier-VPN/CN=Andreas_Hofmeier/emailAddress=andreas@abmh.de
Sat Dec  3 21:31:05 2005 VERIFY OK: depth=0, /C=DE/ST=NA/O=ABM-Hofmeie\
r-VPN/CN=LBlacky/emailAddress=andreas@abmh.de
Sat Dec  3 21:31:06 2005 Data Channel Encrypt: Cipher 'AES-128-CBC' in\
itialized with 128 bit key
Sat Dec  3 21:31:06 2005 Data Channel Encrypt: Using 160 bit message h\
ash 'SHA1' for HMAC authentication
Sat Dec  3 21:31:06 2005 Data Channel Decrypt: Cipher 'AES-128-CBC' in\
itialized with 128 bit key
Sat Dec  3 21:31:06 2005 Data Channel Decrypt: Using 160 bit message h\
ash 'SHA1' for HMAC authentication
Sat Dec  3 21:31:06 2005 Control Channel: TLSv1, cipher TLSv1/SSLv3 DH\
E-RSA-AES256-SHA, 2048 bit RSA
Sat Dec  3 21:31:06 2005 [LBlacky] Peer Connection Initiated with 192.\
168.1.3:1194
Sat Dec  3 21:31:07 2005 SENT CONTROL [LBlacky]: 'PUSH_REQUEST' (statu\
s=1)
Sat Dec  3 21:31:07 2005 PUSH: Received control message: 'PUSH_REPLY,r\
oute 10.8.0.1,ping 10,ping-restart 120,ifconfig 10.8.0.6 10.8.0.5'
Sat Dec  3 21:31:07 2005 OPTIONS IMPORT: timers and/or timeouts modifi\
ed
Sat Dec  3 21:31:07 2005 OPTIONS IMPORT: --ifconfig/up options modifie\
d
Sat Dec  3 21:31:07 2005 OPTIONS IMPORT: route options modified
Sat Dec  3 21:31:07 2005 TUN/TAP device tun0 opened
Sat Dec  3 21:31:07 2005 ifconfig tun0 10.8.0.6 pointopoint 10.8.0.5 m\
tu 1500
Sat Dec  3 21:31:07 2005 route add -net 10.8.0.1 netmask 255.255.255.2\
55 gw 10.8.0.5
Sat Dec  3 21:31:07 2005 Initialization Sequence Completed
```

Auf LBlacky (Server) wurde `ifconfig` ausgeführt um das von OpenVPN erzeugte Virtuelle Netzwerkinterface anzuzeigen. Dannach wurde das Routing mit `route` untersucht. Das hier angezeigt Interface und Routing ist bereits nach der Startphase (bevor ein Klient eine Verbindung aufgebaut hat) sichtbar. Durch Aufbau einer Verbindung ändert sich an beidem nichts mehr. Es wird also auf der Server Seite nur ein virtuelle Interface für alle Klienten benutzt.

```
andreas@LBlacky:~/.vpn_keys > ifconfig
[...]
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-0\
0-00-00-00-00
          inet addr:10.8.0.1  P-t-P:10.8.0.2  Mask:255.255.255.255
          UP POINTOPOINT 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
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

andreas@LBlacky:~/.vpn_keys > route -n
Kernel IP routing table
Destination   Gateway     Genmask         Flags Metric Ref Use Iface
[...]
10.8.0.2      0.0.0.0     255.255.255.255 UH    0      0      0 tun0
10.8.0.0      10.8.0.2    255.255.255.0   UG    0      0      0 tun0
[...]
```

Auf der Seite von Blacky (Client) wurde ebenfalls das virtuelle Interface und das damit im Zusammenhang stehende Routing angezeigt:

```
andreas@Blacky:~ > ifconfig tun0
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-0\
0-00-00-00-00
          inet addr:10.8.0.6  P-t-P:10.8.0.5  Mask:255.255.255.255
          UP POINTOPOINT 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
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

andreas@Blacky:~ > route -n
Kernel IP routing table
Destination   Gateway     Genmask         Flags Metric Ref Use Iface
10.8.0.5      0.0.0.0     255.255.255.255 UH    0      0      0 tun0
10.8.0.1      10.8.0.5    255.255.255.255 UGH   0      0      0 tun0
[...]
```

3.4 Tests an dem erstellten VPN-Tunnel

3.4.1 Geschwindigkeitstest

Um die Übertragungsgeschwindigkeit zu messen, wurden jeweils die ersten 50 Megabyte von der Festplatte gelesen und an den anderen Computer (an Port 1234) übermittelt:

```
dd if=/dev/hda count=50 bs=1M | nc <target-ip> 1234
```

Die empfangende Seite nahm die Daten entgegen und leitete sie nach `/dev/null` um. Alles was an `/dev/null` gesendet wird, wird nicht weiter verarbeitet, es hört schlicht und einfach auf zu existieren.

```
nc -l -p 1234 | dd of=/dev/null
```

Nachdem die Daten übermittelt wurden, beendete `dd` und meldete die Menge der übermittelten Daten, die benötigte Zeit und die Datentransferrate. Diese Werte wurden in der folgenden Tabelle für verschiedene Fälle gegenübergestellt:

				OpenVPN					
	Direkt			TCP			UDP		
Richtung	DTR	LS	LE	DTR	LS	LE	DTR	LS	LE
Blacky -> LBlacky (Client -> Server)	2.709	25	25	2.078 (77%)	95	50	1.979 (73%)	98	50
LBlacky -> Blacky (Server -> Client)	10.483	25	50	1.748 (17%)	50	90	1.932 (18%)	50	100

Legende:

- DTR: Daten-Transfer-Rate in MByte pro Sekunde
- LS: CPU Nutzung auf der Sender-Seite in Prozent
- LE: CPU Nutzung auf der Empfänger-Seite in Prozent
- Werte in Klammern: Prozent der aktuellen DTR, bezogen auf die maximale (unverschlüsselte) DTR in der aktuellen Richtung.

Die Werte für die CPU Nutzung wurden am CPU-Monitor von Afterstep mit Augenmaß abgelesen und sind daher nur Schätzwerte.

Der Computer LBlacky ist neuer und verfügt daher über ein wesentlich schnelleres Bussystem. Damit kann erklärt werden, dass die Datenübertragungsrate bei unverschlüsselter Übertragung von LBlacky zu Blacky fünf mal größer war als umgekehrt. Denn mit diesem Vorteil ist das gleichzeitige Lesen von der Festplatte und Senden an die Netzwerkkarte wesentlich schneller möglich.

Erstaunlich ist, dass TCP in einer Richtung deutlich besser abschneidet als UDP. Theoretisch sollte UDP weniger Protokoll-Overhead erzeugen, da es Verbindungslos arbeitet.

Wie erwartet zweigte der Test, dass eine verschlüsselte Verbindung weniger Durchsatz bei höherer CPU-Verwendung aufweist.

3.4.2 Darf der Computer Server sein?

Es ist theoretisch möglich, dass ein normaler Klient sich als Server ausgibt. Ein sich verbindender Klient würde den anderen Klient als Server akzeptieren. Um dies zu verhindern, ist es möglich, zwei verschiedene CA-Roots zu erzeugen. Einen, mit dem die Klienten zertifiziert werden und einen anderen um die Server zu zertifizieren. Da dies etwas Aufwendig ist, gibt es die Möglichkeit, ein Feld im Zertifikat zu verwenden, um zwischen Servern und Klienten zu unterscheiden. Bei einem Server-Zertifikat wird das Feld `nsCertType` auf "Server" gesetzt. Aus diesem Grund gibt es zwei Skripte zum Erzeugen von Schlüsselpaaren: `04-build-req-server` und `build-req`, mal abgesehen von `01-build-ca`. `04-build-req-server` fügt das Feld `nsCertType` hinzu, `03-build-req` tut dies nicht.

Der Klient prüft, ob der Server den er gerade anwählt über ein Zertifikat verfügt, welches das Fels `nsCertType=server` enthält, wenn die folgende Zeile in der Klient-Konfigurationsdatei enthalten ist:

```
ns-cert-type server
```

Die folgenden Log-Files geben den Fall wieder, dass das Zertifikat des Servers nicht über das `nsCertType=server`-Feld verfügte und der Check am Klienten eingeschaltet war. Ausgaben vom Klienten:

```
Sat Dec  3 21:41:38 2005 TLS: Initial packet from 192.168.1.3:1194, si\
d=1b00ddb9 b80c0c51
Sat Dec  3 21:41:38 2005 VERIFY OK: depth=1, /C=DE/ST=NA/L=Bremen/O=AB\
M-Hofmeier-VPN/CN=Andreas_Hofmeier/emailAddress=andreas@abmh.de
Sat Dec  3 21:41:38 2005 VERIFY nsCertType ERROR: /C=DE/ST=NA/O=ABM-Ho\
fmeier-VPN/CN=Whitey/emailAddress=andreas@abmh.de, require nsCertType=\
SERVER
Sat Dec  3 21:41:38 2005 TLS_ERROR: BIO read tls_read_plaintext error:\
 error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate v\
erify failed
Sat Dec  3 21:41:38 2005 TLS Error: TLS object -> incoming plaintext r\
ead error
Sat Dec  3 21:41:38 2005 TLS Error: TLS handshake failed
Sat Dec  3 21:41:38 2005 Fatal TLS error (check_tls_errors_co), restar\
ting
Sat Dec  3 21:41:38 2005 TCP/UDP: Closing socket
Sat Dec  3 21:41:38 2005 SIGUSR1[soft,tls-error] received, process res\
tarting
Sat Dec  3 21:41:38 2005 Restart pause, 5 second(s)
```

Hier die andere Seite. Der Klient bricht die Verbindung ab, bevor der Server den Klienten authentifizieren kann:

```
Sat Dec  3 21:41:38 2005 192.168.1.2:1071 TLS: Initial packet from 192\
.168.1.2:1071, sid=41a70297 1bb82597
Sat Dec  3 21:41:38 2005 192.168.1.2:1071 Connection reset, restarting\
 [0]
Sat Dec  3 21:41:38 2005 192.168.1.2:1071 SIGUSR1[soft,connection-rese\
t] received, client-instance restarting
Sat Dec  3 21:41:38 2005 TCP/UDP: Closing socket
```

3.4.3 Neuer CA mit gleichen Angaben

Im folgenden Beispiel wurde ein komplett neuer CA angelegt, welcher allerdings die gleichen Angaben für Country Name, State or Province Name, Locality Name, Organization Name, Organizational Unit Name, Common Name und Email Address aufwies. Mit diesem CA-Root wurde ein ebenfalls neu angelegter Schlüssel unterschrieben. Dieser Schlüssel wurde vom Klienten verwendet, um sich beim Server anzumelden.

Der Klient authentifiziert den Server korrekt. (Das Root-CA-Zertifikat wurde weder beim Klienten noch beim Server ersetzt.) Allerdings wird die Verbindung vom Server abgebrochen.

```
Sat Dec  3 21:51:33 2005 TLS: Initial packet from 192.168.1.3:1194, si\
d=61cc91cd 6794b8d5
Sat Dec  3 21:51:34 2005 VERIFY OK: depth=1, /C=DE/ST=NA/L=Bremen/O=AB\
M-Hofmeier-VPN/CN=Andreas_Hofmeier/emailAddress=andreas@abmh.de
Sat Dec  3 21:51:34 2005 VERIFY OK: depth=0, /C=DE/ST=NA/O=ABM-Hofmeie\
r-VPN/CN=LBlacky/emailAddress=andreas@abmh.de
Sat Dec  3 21:51:35 2005 Connection reset, restarting [-1]
Sat Dec  3 21:51:35 2005 TCP/UDP: Closing socket
Sat Dec  3 21:51:35 2005 SIGUSR1[soft,connection-reset] received, proc\
ess restarting
Sat Dec  3 21:51:35 2005 Restart pause, 5 second(s)
```

Der Server versucht den Klienten dessen Schlüsselpaar mit dem gefälschten Root-CA zertifiziert wurde zu authentifizieren. Der versuch schlägt fehl und die Verbindung wird getrennt.

```
Sat Dec  3 21:51:33 2005 192.168.1.2:1079 TLS: Initial packet from 192\
.168.1.2:1079, sid=5fbf1194 cbfef6e4
Sat Dec  3 21:51:35 2005 192.168.1.2:1079 VERIFY ERROR: depth=0, error\
=unable to get local issuer certificate: /C=DE/ST=NA/O=ABM-Hofmeier-VP\
N/CN=Blacky/emailAddress=andreas@abmh.de
Sat Dec  3 21:51:35 2005 192.168.1.2:1079 TLS_ERROR: BIO read tls_read\
```

```
_plaintext error: error:140890B2:SSL routines:SSL3_GET_CLIENT_CERTIFIC\
ATE:no certificate returned
Sat Dec  3 21:51:35 2005 192.168.1.2:1079 TLS Error: TLS object -> inc\
oming plaintext read error
Sat Dec  3 21:51:35 2005 192.168.1.2:1079 TLS Error: TLS handshake fai\
led
Sat Dec  3 21:51:35 2005 192.168.1.2:1079 Fatal TLS error (check_tls_e\
rrors_co), restarting
Sat Dec  3 21:51:35 2005 192.168.1.2:1079 SIGUSR1[soft,tls-error] rece\
ived, client-instance restarting
Sat Dec  3 21:51:35 2005 TCP/UDP: Closing socket
```

3.4.4 Revoke

Es kann passieren, dass ein zertifizierter Schlüssel in die falschen Hände gelangt. Für diesen Fall gibt es die Möglichkeit eine Revoke-Liste anzulegen, welche dem Server oder dem Klienten mitteilt, welche Schlüssel ungültig geworden sind. Es ist überraschend, dass dies nicht in der Standard-Konfigurationsdatei vorgesehen ist. Und das obwohl die Einstellungen für PKI vorhanden sind und erklärt werden.

Der Server erkennt die Revoke liste nicht automatisch und lässt einen Klienten mit widerrufenem Zertifikat eine Verbindung aufbauen.

```
[...]
Sun Dec  4 12:34:09 2005 192.168.1.2:1084 TLS: Initial packet from 192\
.168.1.2:1084, sid=8bd8918c 8bdac88a
Sun Dec  4 12:34:11 2005 192.168.1.2:1084 VERIFY OK: depth=1, /C=DE/ST\
=NA/L=Bremen/O=ABM-Hofmeier-VPN/CN=Andreas_Hofmeier/emailAddress=andre\
as@abmh.de
Sun Dec  4 12:34:11 2005 192.168.1.2:1084 VERIFY OK: depth=0, /C=DE/ST\
=NA/O=ABM-Hofmeier-VPN/CN=Trudy/emailAddress=andreas@abmh.de
[...]
```

Nachdem in der Konfigurationsdatei des Server die Revoke-Liste durch `crl-verify crl.pem` bekannt gegeben wurde, sieht dies ganz anders aus:

```
Sun Dec  4 12:49:05 2005 192.168.1.2:1188 TLS: Initial packet from 192\
.168.1.2:1188, sid=6763722e af126845
Sun Dec  4 12:49:06 2005 192.168.1.2:1188 CRL CHECK OK: /C=DE/ST=NA/L=\
Bremen/O=ABM-Hofmeier-VPN/CN=Andreas_Hofmeier/emailAddress=andreas@abm\
h.de
Sun Dec  4 12:49:06 2005 192.168.1.2:1188 VERIFY OK: depth=1, /C=DE/ST\
=NA/L=Bremen/O=ABM-Hofmeier-VPN/CN=Andreas_Hofmeier/emailAddress=andre\
as@abmh.de
Sun Dec  4 12:49:06 2005 192.168.1.2:1188 CRL CHECK FAILED: /C=DE/ST=N\
A/O=ABM-Hofmeier-VPN/CN=Trudy/emailAddress=andreas@abmh.de is REVOKED
```

Und so wurde das Revoke-Zertifikat erstellt:

Verwendete Dateien:

- `crl.pem`: Liste mit widerrufenen Schlüsseln bzw. Zertifikaten. (Wird erweitert, falls vorhanden.)
- `ca.key`: Privater Schlüssel des CA-Root. Dieser wird verwendet um das Revoke-Zertifikat zu unterschreiben.
- `trudy.csr`: Trudy öffentlicher Schlüssel als unterschriebenes Zertifikat

Erzeugte Dateien:

- `crl.pem`: Liste mit widerrufenen Schlüsseln bzw. Zertifikaten.

```
andreas@LBlacky:~/Projects/DerTunnel/easyrsa > . 07-revoke trudy
Using configuration from /home/andreas/.vpn_keys/openssl.cnf
DEBUG[load_index]: unique_subject = "yes"
Revoking Certificate 03.
Data Base Updated
Using configuration from /home/andreas/.vpn_keys/openssl.cnf
DEBUG[load_index]: unique_subject = "yes"
trudy.crt: /C=DE/ST=NA/O=ABM-Hofmeier-VPN/CN=Trudy/emailAddress=andrea\
s@abmh.de
error 23 at 0 depth lookup:certificate revoked
```

Anhang A: Verwendete Easy-RSA-Skripte

A.1 00-init-vars

```
# easy-rsa parameter settings

# NOTE: If you installed from an RPM,
# don't edit this file in place in
# /usr/share/openvpn/easy-rsa --
# instead, you should copy the whole
# easy-rsa directory to another location
# (such as /etc/openvpn) so that your
# edits will not be wiped out by a future
# OpenVPN package upgrade.

# This variable should point to
# the top level of the easy-rsa
# tree.
#export D=`pwd`

# This variable should point to
# the openssl.cnf file included
# with easy-rsa.
#export KEY_CONFIG=$D/openssl.cnf
export KEY_CONFIG=$HOME/.vpn_keys/openssl.cnf

# Edit this variable to point to
# your soon-to-be-created key
# directory.
#
# WARNING: clean-all will do
# a rm -rf on this directory
# so make sure you define
# it correctly!
#export KEY_DIR=$D/keys
export KEY_DIR=$HOME/.vpn_keys

# Issue rm -rf warning
echo NOTE: when you run ./clean-all, I will be doing a rm -rf on $KEY_DIR

# Increase this to 2048 if you
# are paranoid.  This will slow
# down TLS negotiation performance
# as well as the one-time DH parms
# generation process.
export KEY_SIZE=2048

# These are the default values for fields
# which will be placed in the certificate.
# Don't leave any of these fields blank.
```

```
export KEY_COUNTRY=DE
export KEY_PROVINCE=NA
export KEY_CITY=Bremen
export KEY_ORG="ABM-Hofmeier-VPN"
export KEY_EMAIL="andreas@abmh.de"
```

A.2 01-build-ca

```
#!/bin/sh

#
# Build a root certificate
#

if test $KEY_DIR; then
  cd $KEY_DIR && \
  openssl req -days 3650 -nodes -new -x509 -keyout ca.key -out ca.crt \
          -config $KEY_CONFIG && \
  chmod 0600 ca.key
else
  echo you must define KEY_DIR
fi
```

A.3 01-build-ca

```
#!/bin/sh

#
# Build a root certificate
#

if test $KEY_DIR; then
  cd $KEY_DIR && \
  openssl req -days 3650 -nodes -new -x509 -keyout ca.key -out ca.crt \
          -config $KEY_CONFIG && \
  chmod 0600 ca.key
else
  echo you must define KEY_DIR
fi

andreas@LBlacky:~/Projects/DerTunnel/easyrsa > cat 02-build-dh-helpc
#!/bin/sh

#
# Build Diffie-Hellman parameters for the server side
# of an SSL/TLS connection.
#

if test $KEY_DIR; then
```

```
  openssl dhparam -out ${KEY_DIR}/dh${KEY_SIZE}.pem ${KEY_SIZE}
else
  echo you must define KEY_DIR
fi
```

A.4 03-build-req

```
#!/bin/sh

#
# Build a certificate signing request and private key.  Use this
# when your root certificate and key is not available locally.
#

if test $# -ne 1; then
  echo "usage: build-req <name>";
  exit 1
fi

if test $KEY_DIR; then
  cd $KEY_DIR && \
  openssl req -days 3650 -nodes -new -keyout $1.key -out $1.csr \
          -config $KEY_CONFIG
else
  echo you must define KEY_DIR
fi
```

A.5 04-build-req-server

```
#!/bin/sh

#
# Make a certificate/private key pair using a locally generated
# root certificate.
#
# Explicitly set nsCertType to server using the "server"
# extension in the openssl.cnf file.

if test $# -ne 1; then
  echo "usage: build-key-server <name>";
  exit 1
fi

if test $KEY_DIR; then
  cd $KEY_DIR && \
  openssl req -days 3650 -nodes -new -keyout $1.key -out $1.csr \
          -extensions server -config $KEY_CONFIG && \
  chmod 0600 $1.key
else
```

```
  echo you must define KEY_DIR
fi
```

A.6 05-sign-key

```
#!/bin/sh

#
# Sign a certificate signing request (a .csr file)
# with a local root certificate and key.
#

if test $# -ne 1; then
  echo "usage: sign-req <name>";
  exit 1
fi

if test $KEY_DIR; then
  cd $KEY_DIR && \
  openssl ca -days 3650 -out $1.crt -in $1.csr -config $KEY_CONFIG \
          -keyfile ca.key -cert ca.crt -outdir .
else
  echo you must define KEY_DIR
fi
```

A.7 07-revoke

```
#!/bin/sh

# revoke a certificate, regenerate CRL,
# and verify revocation

CRL=crl.pem
RT=revoke-test.pem

if test $# -ne 1; then
        echo "usage: revoke-full <name>";
        exit 1
fi

if test $KEY_DIR; then
  cd $KEY_DIR
  rm -f $RT

  # revoke key and generate a new CRL
  openssl ca -revoke $1.crt -config $KEY_CONFIG

  # generate a new CRL
  openssl ca -gencrl -out $CRL -config $KEY_CONFIG
```

```
  cat ca.crt $CRL >$RT

  # verify the revocation
  openssl verify -CAfile $RT -crl_check $1.crt
else
  echo you must define KEY_DIR
fi
```

Anhang B: Verwendete Konfigurationsdateien

B.1 Angepasste OpenVPN Beispielkonfigurationsdatei: server.conf

```
################################################
# Sample OpenVPN 2.0 config file for           #
# multi-client server.                         #
#                                              #
# Modifiziert                                  #
#                                              #
# Comments are preceded with '#' or ';'        #
################################################

# Which local IP address should OpenVPN
# listen on? (optional)
;local a.b.c.d

# Which TCP/UDP port should OpenVPN listen on?
# If you want to run multiple OpenVPN instances
# on the same machine, use a different port
# number for each one.  You will need to
# open up this port on your firewall.
port 1194

# TCP or UDP server?
proto tcp
;proto udp

# "dev tun" will create a routed IP tunnel,
# "dev tap" will create an ethernet tunnel.
# Use "dev tap" if you are ethernet bridging.
# If you want to control access policies
# over the VPN, you must create firewall
# rules for the the TUN/TAP interface.
# On non-Windows systems, you can give
# an explicit unit number, such as tun0.
# On Windows, use "dev-node" for this.
# On most systems, the VPN will not function
# unless you partially or fully disable
```

```
# the firewall for the TUN/TAP interface.
;dev tap
dev tun

# Windows needs the TAP-Win32 adapter name
# from the Network Connections panel if you
# have more than one.  On XP SP2 or higher,
# you may need to selectively disable the
# Windows firewall for the TAP adapter.
# Non-Windows systems usually don't need this.
;dev-node MyTap

# SSL/TLS root certificate (ca), certificate
# (cert), and private key (key).  Each client
# and the server must have their own cert and
# key file.  The server and all clients will
# use the same ca file.
#
# See the "easy-rsa" directory for a series
# of scripts for generating RSA certificates
# and private keys.  Remember to use
# a unique Common Name for the server
# and each of the client certificates.
#
# Any X509 key management system can be used.
# OpenVPN can also use a PKCS #12 formatted key file
# (see "pkcs12" directive in man page).
ca ca.crt
cert lblacky.crt
key lblacky.key  # This file should be kept secret
crl-verify crl.pem

# Diffie hellman parameters.
# Generate your own with:
#   openssl dhparam -out dh1024.pem 1024
# Substitute 2048 for 1024 if you are using
# 2048 bit keys.
dh dh2048.pem

# Configure server mode and supply a VPN subnet
# for OpenVPN to draw client addresses from.
# The server will take 10.8.0.1 for itself,
# the rest will be made available to clients.
# Each client will be able to reach the server
# on 10.8.0.1. Comment this line out if you are
# ethernet bridging. See the man page for more info.
server 10.8.0.0 255.255.255.0

# Maintain a record of client <-> virtual IP address
# associations in this file.  If OpenVPN goes down or
# is restarted, reconnecting clients can be assigned
```

```
# the same virtual IP address from the pool that was
# previously assigned.
ifconfig-pool-persist /etc/openvpn/ipp.txt

# Configure server mode for ethernet bridging.
# You must first use your OS's bridging capability
# to bridge the TAP interface with the ethernet
# NIC interface.  Then you must manually set the
# IP/netmask on the bridge interface, here we
# assume 10.8.0.4/255.255.255.0.  Finally we
# must set aside an IP range in this subnet
# (start=10.8.0.50 end=10.8.0.100) to allocate
# to connecting clients.  Leave this line commented
# out unless you are ethernet bridging.
;server-bridge 10.8.0.4 255.255.255.0 10.8.0.50 10.8.0.100

# Push routes to the client to allow it
# to reach other private subnets behind
# the server.  Remember that these
# private subnets will also need
# to know to route the OpenVPN client
# address pool (10.8.0.0/255.255.255.0)
# back to the OpenVPN server.
;push "route 192.168.10.0 255.255.255.0"
;push "route 192.168.20.0 255.255.255.0"

# To assign specific IP addresses to specific
# clients or if a connecting client has a private
# subnet behind it that should also have VPN access,
# use the subdirectory "ccd" for client-specific
# configuration files (see man page for more info).

# EXAMPLE: Suppose the client
# having the certificate common name "Thelonious"
# also has a small subnet behind his connecting
# machine, such as 192.168.40.128/255.255.255.248.
# First, uncomment out these lines:
;client-config-dir ccd
;route 192.168.40.128 255.255.255.248
# Then create a file ccd/Thelonious with this line:
#   iroute 192.168.40.128 255.255.255.248
# This will allow Thelonious' private subnet to
# access the VPN.  This example will only work
# if you are routing, not bridging, i.e. you are
# using "dev tun" and "server" directives.

# EXAMPLE: Suppose you want to give
# Thelonious a fixed VPN IP address of 10.9.0.1.
# First uncomment out these lines:
;client-config-dir ccd
;route 10.9.0.0 255.255.255.252
```

```
# Then add this line to ccd/Thelonious:
#   ifconfig-push 10.9.0.1 10.9.0.2

# Suppose that you want to enable different
# firewall access policies for different groups
# of clients.  There are two methods:
# (1) Run multiple OpenVPN daemons, one for each
#     group, and firewall the TUN/TAP interface
#     for each group/daemon appropriately.
# (2) (Advanced) Create a script to dynamically
#     modify the firewall in response to access
#     from different clients.  See man
#     page for more info on learn-address script.
;learn-address ./script

# If enabled, this directive will configure
# all clients to redirect their default
# network gateway through the VPN, causing
# all IP traffic such as web browsing and
# and DNS lookups to go through the VPN
# (The OpenVPN server machine may need to NAT
# the TUN/TAP interface to the internet in
# order for this to work properly).
# CAVEAT: May break client's network config if
# client's local DHCP server packets get routed
# through the tunnel.  Solution: make sure
# client's local DHCP server is reachable via
# a more specific route than the default route
# of 0.0.0.0/0.0.0.0.
;push "redirect-gateway"

# Certain Windows-specific network settings
# can be pushed to clients, such as DNS
# or WINS server addresses.  CAVEAT:
# http://openvpn.net/faq.html#dhcpcaveats
;push "dhcp-option DNS 10.8.0.1"
;push "dhcp-option WINS 10.8.0.1"

# Uncomment this directive to allow different
# clients to be able to "see" each other.
# By default, clients will only see the server.
# To force clients to only see the server, you
# will also need to appropriately firewall the
# server's TUN/TAP interface.
;client-to-client

# Uncomment this directive if multiple clients
# might connect with the same certificate/key
# files or common names.  This is recommended
# only for testing purposes.  For production use,
# each client should have its own certificate/key
```

```
# pair.
#
# IF YOU HAVE NOT GENERATED INDIVIDUAL
# CERTIFICATE/KEY PAIRS FOR EACH CLIENT,
# EACH HAVING ITS OWN UNIQUE "COMMON NAME",
# UNCOMMENT THIS LINE OUT.
;duplicate-cn

# The keepalive directive causes ping-like
# messages to be sent back and forth over
# the link so that each side knows when
# the other side has gone down.
# Ping every 10 seconds, assume that remote
# peer is down if no ping received during
# a 120 second time period.
keepalive 10 120

# For extra security beyond that provided
# by SSL/TLS, create an "HMAC firewall"
# to help block DoS attacks and UDP port flooding.
#
# Generate with:
#   openvpn --genkey --secret ta.key
#
# The server and each client must have
# a copy of this key.
# The second parameter should be '0'
# on the server and '1' on the clients.
;tls-auth ta.key 0 # This file is secret

# Select a cryptographic cipher.
# This config item must be copied to
# the client config file as well.
;cipher BF-CBC        # Blowfish (default)
cipher AES-128-CBC   # AES
;cipher DES-EDE3-CBC  # Triple-DES

# Enable compression on the VPN link.
# If you enable it here, you must also
# enable it in the client config file.
comp-lzo

# The maximum number of concurrently connected
# clients we want to allow.
;max-clients 100

# It's a good idea to reduce the OpenVPN
# daemon's privileges after initialization.
#
# You can uncomment this out on
# non-Windows systems.
```

```
;user nobody
;group nobody

# The persist options will try to avoid
# accessing certain resources on restart
# that may no longer be accessible because
# of the privilege downgrade.
persist-key
persist-tun

# Output a short status file showing
# current connections, truncated
# and rewritten every minute.
status /etc/openvpn/openvpn-status.log

# By default, log messages will go to the syslog (or
# on Windows, if running as a service, they will go to
# the "\Program Files\OpenVPN\log" directory).
# Use log or log-append to override this default.
# "log" will truncate the log file on OpenVPN startup,
# while "log-append" will append to it.  Use one
# or the other (but not both).
;log         openvpn.log
;log-append  openvpn.log

# Set the appropriate level of log
# file verbosity.
#
# 0 is silent, except for fatal errors
# 4 is reasonable for general usage
# 5 and 6 can help to debug connection problems
# 9 is extremely verbose
verb 3

# Silence repeating messages.  At most 20
# sequential messages of the same message
# category will be output to the log.
;mute 20
```

B.2 Angepasste OpenVPN Beispielkonfigurationsdatei: client.conf

```
################################################
# Sample client-side OpenVPN 2.0 config file #
# for connecting to multi-client server.     #
#                                            #
# This configuration can be used by multiple #
# clients, however each client should have   #
# its own cert and key files.                #
#                                            #
```

```
# Modifiziert                                   #
###############################################

# Specify that we are a client and that we
# will be pulling certain config file directives
# from the server.
client

# Use the same setting as you are using on
# the server.
# On most systems, the VPN will not function
# unless you partially or fully disable
# the firewall for the TUN/TAP interface.
;dev tap
dev tun

# Windows needs the TAP-Win32 adapter name
# from the Network Connections panel
# if you have more than one.  On XP SP2,
# you may need to disable the firewall
# for the TAP adapter.
;dev-node MyTap

# Are we connecting to a TCP or
# UDP server?  Use the same setting as
# on the server.
;proto tcp
proto udp

# The hostname/IP and port of the server.
# You can have multiple remote entries
# to load balance between the servers.
remote 192.168.1.3 1194
;remote my-server-2 1194

# Choose a random host from the remote
# list for load-balancing.  Otherwise
# try hosts in the order specified.
;remote-random

# Keep trying indefinitely to resolve the
# host name of the OpenVPN server.  Very useful
# on machines which are not permanently connected
# to the internet such as laptops.
resolv-retry infinite

# Most clients don't need to bind to
# a specific local port number.
nobind

# Downgrade privileges after initialization (non-Windows only)
```

```
;user nobody
;group nobody

# Try to preserve some state across restarts.
persist-key
persist-tun

# If you are connecting through an
# HTTP proxy to reach the actual OpenVPN
# server, put the proxy server/IP and
# port number here.  See the man page
# if your proxy server requires
# authentication.
;http-proxy-retry # retry on connection failures
;http-proxy [proxy server] [proxy port #]

# Wireless networks often produce a lot
# of duplicate packets.  Set this flag
# to silence duplicate packet warnings.
;mute-replay-warnings

# SSL/TLS parms.
# See the server config file for more
# description.  It's best to use
# a separate .crt/.key file pair
# for each client.  A single ca
# file can be used for all clients.
ca ca.crt
cert blacky.crt
key blacky.key

# Verify server certificate by checking
# that the certicate has the nsCertType
# field set to "server".  This is an
# important precaution to protect against
# a potential attack discussed here:
#  http://openvpn.net/howto.html#mitm
#
# To use this feature, you will need to generate
# your server certificates with the nsCertType
# field set to "server".  The build-key-server
# script in the easy-rsa folder will do this.
;ns-cert-type server

# If a tls-auth key is used on the server
# then every client must also have the key.
;tls-auth ta.key 1

# Select a cryptographic cipher.
# If the cipher option is used on the server
# then you must also specify it here.
```

```
cipher AES-128-CBC

# Enable compression on the VPN link.
# Don't enable this unless it is also
# enabled in the server config file.
comp-lzo

# Set log file verbosity.
verb 3

# Silence repeating messages
;mute 20
```

B.3 OpenSSL Konfiguartion von Easy-RSA

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME                    = .
RANDFILE                = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca' and 'req'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

####################################################################
[ ca ]
default_ca      = CA_default            # The default ca section

####################################################################
[ CA_default ]
```

```
dir             = $ENV::KEY_DIR         # Where everything is kept
certs           = $dir                  # Where the issued certs are kept
crl_dir         = $dir                  # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
new_certs_dir   = $dir                  # default place for new certs.

certificate     = $dir/ca.crt           # The CA certificate
serial          = $dir/serial           # The current serial number
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/ca.key           # The private key
RANDFILE        = $dir/.rand            # private random number file

x509_extensions = usr_cert              # The extentions to add to the cert

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crl_extensions        = crl_ext

default_days    = 3650                  # how long to certify for
default_crl_days= 30                    # how long before next CRL
default_md      = md5                   # which md to use.
preserve        = no                    # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy          = policy_match

# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

####################################################################
```

```
[ req ]
default_bits            = $ENV::KEY_SIZE
default_keyfile         = privkey.pem
distinguished_name      = req_distinguished_name
attributes              = req_attributes
x509_extensions = v3_ca # The extentions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix   : PrintableString, BMPString.
# utf8only: only UTF8Strings.
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: current versions of Netscape crash on BMPStrings or UTF8Strings
# so use this option with caution!
string_mask = nombstr

# req_extensions = v3_req # The extensions to add to a certificate request

[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = $ENV::KEY_COUNTRY
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = $ENV::KEY_PROVINCE

localityName                    = Locality Name (eg, city)
localityName_default            = $ENV::KEY_CITY

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = $ENV::KEY_ORG

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, section)
#organizationalUnitName_default =

commonName                      = Common Name (eg, your name or your server\'s hostn
commonName_max                  = 64

emailAddress                    = Email Address
emailAddress_default            = $ENV::KEY_EMAIL
emailAddress_max                = 40
```

```
# SET-ex3                       = SET extension number 3

[ req_attributes ]
challengePassword               = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

unstructuredName                = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                    = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                       = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy

# Copy subject details
# issuerAltName=issuer:copy
```

```
#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

[ server ]

# JY ADDED -- Make a cert with nsCertType set to "server"
basicConstraints=CA:FALSE
nsCertType                      = server
nsComment                       = "OpenSSL Generated Server Certificate"
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]


# Extensions for a typical CA


# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer:always

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
```

```
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always
```