Verschlüsselte Datenübertragung im WWW mittels SHTTP / SSL

Thomas Hungenberg und Christoph Neerfeld

Fachhochschule Rhein-Sieg, Wintersemester 1999/2000
Seminar Informationssicherheit
Prof. Dr. Hartmut Pohl

04. Januar 2000

Inhaltsverzeichnis

1 Einleitung **3**

2 SHTTP, SSL und TLS **3**

3 Secure Sockets Layer (SSL) Protokoll **4**
3.1 Aufbau 4
3.2 Implementierung 5
3.3 SSL Sub-Protokolle 6
3.4 Sessions und Datenverbindungen 7

4 mod_ssl - Starke Verschlüsselung für den Apache Webserver **8**
4.1 Was ist mod_ssl? 8
4.2 Welche SSL Verschlüsselungsarten unterstützt mod_ssl? 8
4.3 Praktischer Test von mod_ssl 9
4.3.1 Installation 9
4.3.2 Erstellung eines Test-Zertifikats 9
4.4 Test des Servers mittels Netscape Communicator 9
4.5 Fortify für Netscape 10

5 Schlußbetrachtung **11**

1 Einleitung

Das World Wide Web hat sich in den letzten Jahren zu einem bedeutenden Kommunikationsmedium entwickelt, über das in zunehmenden Maße auch kommerzielle Transaktionen abgewickelt werden. Dies beginnt bei Bestellungen mittels Web-Formularen, bei denen die Zahlung nach Erhalt der Ware per Überweisung geschieht, geht weiter über die direkte Übertragung von Kreditkartennummern über das Internet, bis hin zu Online-Banking, bei dem Überweisungsaufträge vom heimischen PC aus erteilt werden können.

All diese Transaktionen stellen jedoch Anforderungen an die Sicherheit der Übertragung, die bei der Entwicklung des WWW nur eine untergeordnete Rolle gespielt haben. Das WWW war ursprünglich entwickelt worden, damit Wissenschaftler ihre Forschungsergebnisse möglichst schnell und unkompliziert veröffentlichen konnten. Dinge wie Vertraulichkeit, Integrität und Authentizität waren hierbei von geringer Bedeutung und wurden daher vollkommen vernachlässigt.

Das dem WWW zugrundeliegende Hypertext-Transfer-Protocol (HTTP) überträgt sämtliche Daten im Klartext, so daß sie auf einfache Weise von einem Dritten mitgelesen werden können. Ferner können die Daten auf ihrem Weg vom Sender zum Empfänger verändert werden, ohne daß dies von einem der Kommunikationspartner bemerkt wird. Darüberhinaus hat man keine Gewähr, daß sein Kommunikationspartner auch tatsächlich der ist, für den er sich ausgibt.

Um diese Mängel von HTTP auszugleichen, wurden im Laufe der Zeit eine Reihe von Erweiterungen vorgeschlagen: das Secure Hypertext-Transfer-Protocol (SHTTP)[1], Secure Sockets Layer (SSL) Protokoll[2] und das Transport Layer Security (TLS) Protokoll[3].

2 SHTTP, SSL und TLS

Das Secure Hypertext-Transfer-Protocol (SHTTP) geht auf einen Vorschlag der Firma Terisa Systems zurück[4], die es als erste in ihre Produkte integriert hat. SHTTP ist im wesentlichen ein erweitertes HTTP. Der Kommunikationsablauf ist aus diesem Grund fast identisch und lediglich die Nachrichtenformate wurden erweitert.

Um die Integration von SHTTP in bestehende Produkte möglichst einfach zu machen, werden die kompletten HTTP-Nachrichten, die bei einer unverschlüsselten Übertragung verwendet würden, in einer SHTTP-Nachricht verpackt. Die SHTTP-Nachricht als solche ist hierbei unverschlüsselt. Lediglich die in ihr enthaltene HTTP-Nachricht ist verschlüsselt. Abbildung 1 soll dies noch einmal verdeutlichen.

Die Spezifikation von SHTTP wurde mittlerweile im RFC 2260 niedergelegt, welches jedoch bis jetzt nur den Status „experimentell“ besitzt. Die Zukunft von SHTTP ist allerdings fraglich, da seine Entwicklung bisher hauptsächlich von Terisa Systems vorangetrieben wurde, welche mittlerweile von der Spyrus Company[5] aufgekauft worden ist. Spyrus selbst scheint jedoch in seinen Produkten den Konkurrenten SSL zu bevorzugen, was eine Weiterentwicklung von SHTTP zumindest fraglich erscheinen läßt.

Das Secure Sockets Layer (SSL) Protokoll ist eine Entwicklung von Netscape Incorporated[6]. Netscape erkannte bereits früh, daß HTTP den steigenden Sicherheitsanforderungen im WWW nicht genügt und entwarf daher ein Protokoll, mit dem sich die Kommunikation über HTTP, aber auch über andere Protokolle wie z. B. das File Transfer Protocol (FTP) absichern läßt.

SSL verfolgt dabei einen vollkommen anderen Ansatz als SHTTP. Während SHTTP auf der Ebene von Nachrichten (HTTP-Anfrage und HTTP-Antwort) arbeitet und dabei das eigentliche HTTP ersetzt, schiebt sich SSL als zusätzliche Schicht zwischen das Transferprotokoll (z. B. TCP/IP) und HTTP. Die HTTP-Nachrichten werden hierbei als Datenstrom vollkommen transparent durch SSL weitergeleitet. Da SSL demnach keine Kenntnis über den Aufbau von HTTP besitzt, ist es auch

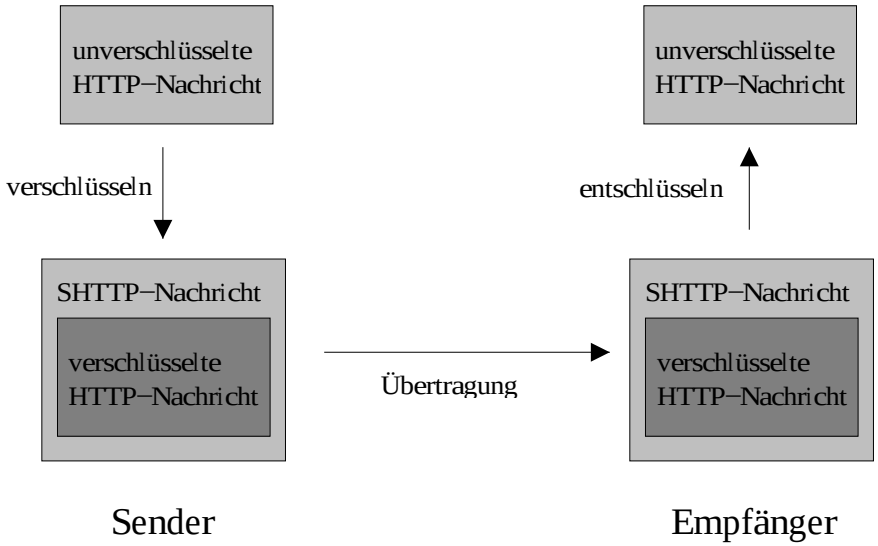

Abbildung 1: Grundprinzip der Nachrichtenübertragung bei SHTTP

möglich, andere Protokolle, wie z. B. FTP, das Post Office Protocol (POP3) oder das Internet Mail Access Protocol (IMAP), über SSL zu übertragen. In Abbildung 2 ist dieser Aufbau noch einmal dargestellt.

SSL ist derzeit der „de-facto-Standard“ für die sichere Datenkommunikation im WWW. Es wird von allen gängigen Internet-Browsern, sowie von einer Reihe von Web-Servern unterstützt. Die Spezifikation zu SSL wurde Ende 1995 der Internet Engineering Task Force (IETF) zur Standardisierung vorgelegt. Aktuell ist die Version 3.02 von November 1996, welche als Internet-Draft vorliegt.

Das Transport Layer Security (TLS) Protokoll ist eigentlich keine eigenständige Entwicklung, da es direkt auf dem SSL Protokoll basiert. Tatsächlich ist es bis auf einige wenige Erweiterungen identisch mit SSL in der Version 3. Da diese Erweiterungen jedoch eine direkte Kommunikation zwischen einer SSL- und einer TLS-Implementierung verhindern, wurde TLS von der IETF als eigenständiges Protokoll standardisiert, welches die SSL-Implementierungen in Zukunft ablösen soll. Um diesen Migrationsprozeß so einfach wie möglich zu gestalten, enthält die TLS-Spezifikation einen Abschnitt zur Backward-Kompatibilität zu SSL Version 3.

Die aktuelle Version 1.0 des TLS Protokolls ist im RFC 2246[3] von Januar 1999 beschrieben und hat den Status eines „Proposed Standard“.

In den nun folgenden Abschnitten wird nur noch auf das Secure Sockets Layer (SSL) Protokoll eingegangen, da es den „de-facto-Standard“ darstellt. Das Secure Hypertext Transfer Protocol (SHTTP) ist demgegenüber wenig verbreitet und seine Zukunft ist obendrein fraglich. Es ist zwar anzunehmen, daß das Transport Layer Security (TLS) Protokoll in Zukunft SSL ablösen wird, da es sich in der grundsätzlichen Arbeitsweise jedoch nicht von SSL unterscheidet, wird hier auf eine genauere Beschreibung verzichtet.

3 Secure Sockets Layer (SSL) Protokoll

3.1 Aufbau

Das Hauptziel des Secure Sockets Layer (SSL) Protokolls besteht darin, eine sichere Kommu-

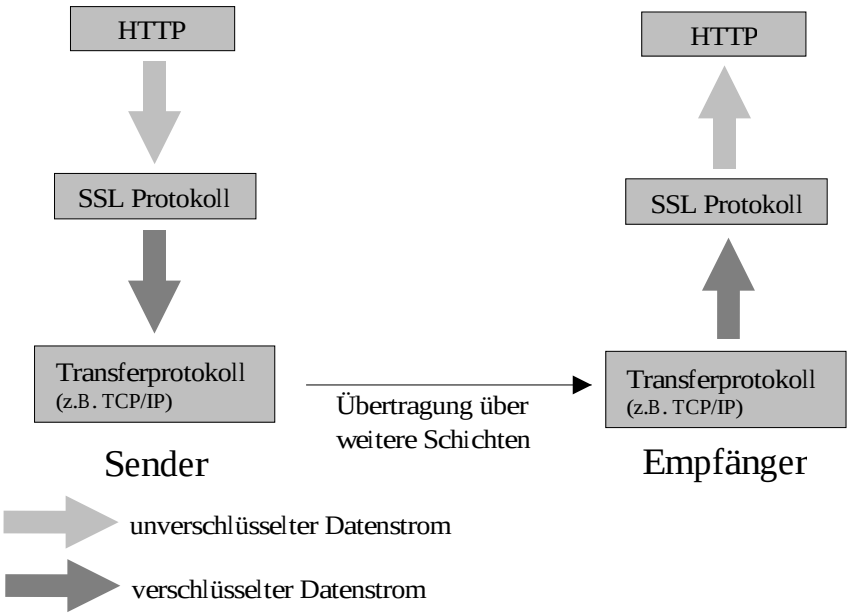

Abbildung 2: Verschlüsselte Übertragung eines Datenstroms mittels SSL

nikation zwischen zwei Anwendungen zu gewährleisten. Diese Sicherheit umfaßt dabei die folgenden drei Teilaspekte:

- Die zu übertragenden Daten werden mit Hilfe eines symmetrischen Verfahrens (z. B. DES oder RC4) verschlüsselt, so daß diese von einem Dritten nicht mitgelesen werden können.
- Die Kommunikationspartner können über ein asymmetrisches Verfahren (z. B. RSA oder DSS) authentifiziert werden, um sicherzustellen, daß der Kommunikationspartner der ist, für den er sich ausgibt.
- Die Datenübertragung ist zuverlässig. Mit Hilfe eines sicheren Hash-Algorithmus (z. B. SHA oder MD5) wird sichergestellt, daß die Daten nicht verändert worden sind. Zusätzlich garantiert die Verwendung einer Sequenznummer, daß die Daten vollständig übertragen worden sind.

Ein weiteres Ziel bei der Entwicklung von SSL liegt in seiner Erweiterungsfähigkeit. Da die Forschung nach besseren Verschlüsselungsalgorithmen stetig fortschreitet, wurde darauf verzichtet, bestimmte Algorithmen fest vorzugeben. Stattdessen bietet SSL einen Rahmen an, der die Verwendung der verschiedensten Algorithmen erlaubt. Dabei ist eine Implementierung nicht auf einen bestimmten Algorithmus festgelegt, sondern kann alternativ eine ganze Reihe von Algorithmen für die gleiche Aufgabe unterstützen. Allerdings führt dies dazu, daß unter Umständen zwei verschiedene SSL Implementierungen nicht miteinander kommunizieren können, sofern sie nicht über einen gemeinsamen Algorithmus verfügen.

3.2 Implementierung

Das SSL Protokoll befindet sich, wie in Abbildung 3 dargestellt, zwischen den Schichten 4 und 5 des OSI-Referenzmodells. Es liegt somit zwischen der Transportschicht des Betriebsystems und der eigentlichen Anwendung. Dabei wird das SSL Protokoll in aller Regel innerhalb der Anwendung implementiert. Allerdings ist es auch möglich das SSL Protokoll in einem externen Proxy zu implementieren wie dies z. B. in [7] beschrieben wird. Auf diese Weise ist es möglich, eine nicht SSL-fähige

Anwendung über SSL mit der Außenwelt kommunizieren zu lassen.

3.3 SSL Sub-Protokolle

SSL selbst ist wiederum in zwei Schichten unterteilt. Die untere Schicht besteht aus dem SSL-Record-Protokoll, welches für die Sicherung der Datenverbindung zuständig ist. Seine Aufgaben bestehen in der Komprimierung und Verschlüsselung der Daten, dem Berechnen und Prüfen der Hash-Werte, sowie der Erzeugung der Sequenznummern. Die folgenden vier Protokolle verwenden das Record-Protokoll für ihre Datenübertragung:

SSL-Handshake-Protokoll Mit Hilfe dieses Protokolls einigen sich die Kommunikationspartner über die gewünschten Sicherungsparameter. Dies beinhaltet die Authentifizierung, die Wahl des Verschlüsselungsalgorithmus, sowie den Austausch eines geheimen Schlüssels für das symmetrische Verschlüsselungsverfahren. Im folgenden wird auf das Handshake-Protokoll noch genauer eingegangen.

SSL-ChangeCipherSpec-Protokoll Die durch das Handshake-Protokoll ausgehandelten Parameter treten nicht automatisch in Kraft. Stattdessen verwendet jeder Partner das ChangeCipherSpec-Protokoll, um dem anderen mitzuteilen, daß alle folgenden Nachrichten gemäß der neuen Parameter verschlüsselt werden.

SSL-Alert-Protokoll Seine Aufgabe besteht unter anderem darin, Fehlernachrichten zwischen den Kommunikationspartnern auszutauschen. Zu diesen Fehlern gehören z. B. die Erkennung eines falschen Hash-Wertes oder die Ablehnung eines Zertifikates. Viele dieser Fehler führen zu einem direkten Abbruch der Verbindung.
Die zweite Aufgabe des Alert-Protokolls besteht darin, die Verbindung regulär zu beenden. SSL verlangt, daß beide Partner die Verbindung ordnungsgemäß beenden, um einen *Truncation* Angriff[1] zu verhindern.

SSL-ApplicationData-Protokoll Dieses Protokoll dient dazu, die eigentlichen Daten der Anwendung wie z. B. HTTP, FTP oder POP3 über SSL zu übertragen. Der Datenstrom wird hierbei für SSL vollkommen transparent übertragen.

Nachdem eine Verbindung zwischen einem Client und einem Server auf der Ebene der Transportschicht hergestellt worden ist, erfolgt der Datenaustausch zunächst einmal unverschlüsselt. Zwar läuft die Übertragung bereits über das Record-Protokoll, welches für die Verschlüsselung zuständig ist, doch anfangs ist der Algorithmus auf „Keine-Verschlüsselung“ eingestellt.

Bevor jetzt Daten der Anwendung übertragen werden können, müssen sich die Kommunikationspartner mit Hilfe des Handshake-Protokolls über die gewünschten Verschlüsselungsparameter einigen. Hierzu sendet der Client eine *ClientHello* Nachricht, in der er mitteilt, welche Kryptoverfahren und Kompressionsalgorithmen von ihm unterstützt werden. Außerdem enthält diese Nachricht eine 28-Byte lange Zufallszahl. Der Server antwortet mit einer *ServerHello* Nachricht, die das von ihm gewählte Kryptoverfahren, den Kompressionsalgorithmus, eine Session-ID und ebenfalls eine 28-Byte lange Zufallszahl enthält. Sofern sich der Server authentifizieren will, sendet er daraufhin eine *Certificate* Nachricht, welche ein Zertifikat nach X.509v3[8] enthält. Falls der Server sich nicht authentifiziert oder sein Zertifikat nur zum digitalen Signieren und nicht zum Schlüsselaustausch geeignet ist, erzeugt er ein temporäres Public/Private-Key-Paar und übermittelt den Public-Key durch eine *ServerKeyExchange* Nachricht. Wenn eine Authentifizierung des Clients erwünscht ist, kann der

[1] Der Angreifer unterbricht an einer bestimmten Stelle gezielt die Verbindung. Aufgrund der fehlenden Information kann der Empfänger zu einem fehlerhaften Verhalten verleitet werden.

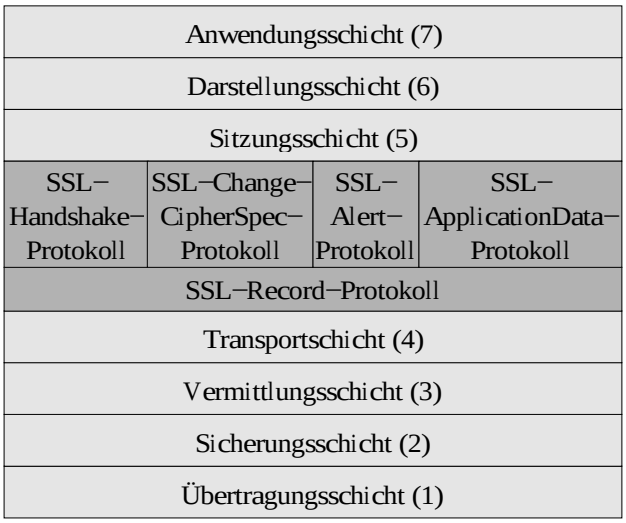

Abbildung 3: Einordnung von SSL in das OSI-Referenzmodell

Server diese mittels einer *CertificateRequest* Nachricht anfordern. Anschließend sendet er eine *ServerHelloDone* Nachricht, um anzuzeigen, daß er jetzt eine Antwort des Clients erwartet.

Der Client antwortet zunächst mit einer *Certificate* Nachricht, sofern diese angefordert worden war. Daraufhin erzeugt er den sogenannten PreMaster-Key aus dem später die geheimen Schlüssel für die Datenübertragung berechnet werden und übermittelt ihn in einer *ClientKeyExchange* Nachricht. Der Inhalt dieser Nachricht wird entweder mit dem Public-Key des Server-Zertifikats oder mit dem temporären Public-Key der *ServerKeyExchange* Nachricht verschlüsselt. Sofern sich der Client authentifizieren sollte, leistet er daraufhin mittels der *CertificateVerify* Nachricht eine digitale Signatur, mit der er beweist, daß er im Besitz des zugehörigen Private-Key zu seiner *Certificate* Nachricht ist.

Zu diesem Zeitpunkt sind sowohl Client als auch Server im Besitz des PreMaster-Key, der geschützt durch den Public-Key des Servers übertragen wurde. Aus dem PreMaster-Key und den beiden Zufallszahlen aus den *Hello* Nachrichten berechnen jetzt Client und Server nach einem vorgegebenem Algorithmus den Master-Key und aus diesem wiederum eine Reihe von geheimen Schlüsseln. Diese Schlüssel dienen zur Verschlüsselung der Nachrichten und zur Generierung der Hash-Werte. Client und Server senden daraufhin *ChangeCipherSpec* Nachrichten und verwenden von diesem Zeitpunkt an die soeben ausgehandelten Verschlüsselungsparameter. Abschließend senden beide noch eine *Finished* Nachricht, um sich zu vergewissern, daß sie im Besitz der gleichen geheimen Schlüssel sind.

3.4 Sessions und Datenverbindungen

Wie man sehen kann, ist dieser Handshake sehr aufwendig und führt gerade bei HTTP zu einer spürbaren Verzögerung, da hier jede Anfrage und Antwort eine eigene Verbindung darstellt. Aus diesem Grund unterscheidet SSL zwischen einer sogenannten *Session* und einer *Datenverbindung*.

Eine Session ist eine logische Verbindung zwischen einem Client und einem Server, die auch dann noch besteht, wenn zur Zeit keine Daten übertragen werden. Die Dauer einer

Session ist abhängig von der jeweiligen Implementierung, und darf maximal 24 Stunden betragen. Während dieser Zeit merken sich Client und Server eine gemeinsame eindeutige Session-ID und den dazugehörigen Master-Key.

Will der Client eine Datenverbindung öffnen, die an eine bestehende Session anknüpft, so übermittelt er in seiner *ClientHello* Nachricht die entsprechende Session-ID. Der Server überprüft diese Session-ID und verwendet die gleiche ID in seiner ServerHello Nachricht, sofern er bereit ist, an die alte Session anzuknüpfen. Ist er dazu nicht bereit, so generiert er einfach eine neue Session-ID.

Da Client und Server nach wie vor über den gemeinsamen Master-Key verfügen, ist es nicht notwendig weitere Nachrichten auszutauschen. Sie können direkt aus dem Master-Key neue geheime Schlüssel generieren, wobei sie die Zufallszahlen aus den beiden *Hello* Nachrichten verwenden. Auf diese Weise erhalten sie für diese Datenverbindung andere geheime Schlüssel als beim ersten Mal. Anschließen senden sie wieder *ChangeCipherSpec* und *Finished* Nachrichten, womit der verkürzte Handshake abgeschlossen ist.

4 mod_ssl - Starke Verschlüsselung für den Apache Webserver

4.1 Was ist mod_ssl?

mod_ssl[10] ist ein Modul für den Apache[11] Webserver (ab Version 1.3), welches eine starke Verschlüsselung der Übertragung mittels des Secure Socket Layer (SSL v2/v3) oder des Transport Layer Security (TLS v1) Protokolls ermöglicht. Es benutzt dazu die Open Source SSL/TLS Implementierung OpenSSL[12] von Eric A. Young und Tim Hudson.

Die erste Version des mod_ssl Pakets wurde im April 1998 von Ralf S. Engelschall als Ableitung des Apache-SSL[13] Pakets von Ben Laurie geschaffen. Die erste „public release"-Version war mod_ssl 2.0.0 und wurde am 10. August 1988 veröffentlicht.

mod_ssl unterliegt einer BSD-ähnlichen Lizenz, die äquivalent der Lizenz für den Apache Webserver selbst ist. Dies bedeutet, kurz gesagt, daß sowohl der kommerzielle als auch der nicht-kommerzielle Einsatz der Software frei ist, solange die Copyright-Anmerkungen der Autoren beibehalten werden und ein entsprechender Hinweis gegeben wird.

4.2 Welche SSL Verschlüsselungsarten unterstützt mod_ssl?

Im allgemeinen werden alle von der benutzten OpenSSL Version unterstützten Verschlüsselungsarten[2] auch von mod_ssl unterstützt. Dies sind typischerweise *mindestens*:

- RC4 mit MD5
- RC4 mit MD5 (Export-Version mit 40 Bit Schlüssellänge)
- RC2 mit MD5
- RC2 mit MD5 (Export-Version mit 40 Bit Schlüssellänge)
- IDEA mit MD5
- DES mit MD5
- Triple-DES mit MD5

Eine komplette Liste der von der installierten OpenSSL Version unterstützten Verschlüsselungsarten erhält man mittels des Befehls

```
$ openssl ciphers -v
```

[2]Mit „Verschlüsselungsart" ist hier die Kombination eines Verschlüsselungsverfahrens, wie z. B. DES, und eines Message Digest-Verfahrens, wie z. B. MD5, gemeint.

4.3 Praktischer Test von mod_ssl

4.3.1 Installation

Als Testumgebung wurde das Unix-ähnliche Betriebssystem Debian GNU/Linux[14] in der Version 2.2 („potato“) gewählt. Nach der Installation des Betriebssystems wurden die für diese Distribution derzeit aktuellen Pakete von Apache (1.3.9), OpenSSL (0.9.4) und mod_ssl (2.4.2) installiert. Danach wurde die Konfigurationsdatei `/etc/apache/httpd.conf`[3] des Apache Servers entsprechend der im mod_ssl-Paket von Debian enthaltenen Beispielkonfiguration modifiziert. Für eine komplette Übersicht der möglichen Konfigurationsparameter, wie z. B. die gezielte (De-)Aktivierung spezieller Verschlüsselungsarten, siehe [10].

Anschließend wurde in dem in der Konfiguration als `DocumentRoot` angegebenen Verzeichnis eine Test-HTML-Datei angelegt.

4.3.2 Erstellung eines Test-Zertifikats

Nach der Installation wurde mittels des Debian-Skripts `mod-ssl-makecert`, welches auf dem SSL Certificate Generation Utility (`mkcert.sh`) aus dem mod_ssl-Paket basiert, ein X.509v3 Test-Zertifikat erstellt. Das Test-Zertifikat enthält standardmäßig einen RSA-Public-Key mit einer Schlüssellänge von 1024 Bit[4] und wird in der Datei `/etc/apache/ssl.crt/server.crt` gespeichert. Es wird automatisch mit dem im gleichen Verzeichnis installierten Dummy-Zertifikat der „Snake Oil CA“ signiert (wobei der Signatur-Algorithmus „md5WithRSAEncryption“ verwendet wird) und hat eine Gültigkeitsdauer von einem Jahr.

Diese Zertifikate sind ausschließlich zu Test-Zwecken und nicht für den realen Einsatz (besonders im Internet) gedacht!

Die genauen Daten, die in den Zertifikaten gespeichert sind, erhält man mittels:

```
$ openssl x509 -noout -text
-in <zertifikatname>.crt
```

Der zugehörige RSA-Private-Key wird standardmäßig mittels Triple-DES mit einem Paßwort, welches bei der Zertifikaterzeugung eingegeben werden muß, verschlüsselt in der Datei `/etc/apache/ssl.key/server.key` abgelegt.

Die Parameter des privaten RSA-Schlüssel lassen sich mit folgendem Befehl anzeigen:

```
$ openssl rsa -noout -text
-in <dateiname>.key
```

Beim Starten des Apache Webservers müssen die Paßworte aller verschlüsselt gespeicherten privaten Schlüssel eingegeben werden, was z. B. das automatische Rebooten eines Servers erschwert. Falls davon ausgegangen werden kann, daß der Server genügend abgesichert ist und das automatische Neustarten des Servers ohne manuelle Eingabe der Paßwörter nötig ist, können die privaten Schlüssel bei der Generierung alternativ unverschlüsselt gespeichert werden. Nachträglich kann die Verschlüsselung wie folgt entfernt werden:

```
$ openssl rsa -in <verschluesselt>.key
-out <entschluesselt>.key
```

Dabei sollte jedoch unbedingt sichergestellt werden, daß die unverschlüsselten privaten Schlüsseldateien nur von berechtigten Usern gelesen werden können (unter Unix üblicherweise nur `root`).

4.4 Test des Servers mittels Netscape Communicator

Als Test-Client wurde der Web-Browser Netscape Communicator[6] in der Version 4.61 (englisch) gewählt, welcher ebenfalls als Debian-Paket installiert wurde.

Nach dem Starten von Apache wurde die URL des Webservers im Browser im Format `https://www.server.name/` eingegeben.

[3] Alle Pfadangaben sind Debian 2.2-spezifisch und können bei anderen Installationen abweichen.

[4] Die Schlüssellänge muß entweder 512 oder 1024 Bit betragen, um kompatibel zu den gängigen Web-Browsern zu sein.

Netscape zeigte daraufhin den Empfang eines neuen Server-Zertifikats („New Site Certificate“) und die darin enthaltenen Daten an (Detailansicht siehe Abbildung 4). Als Verschlüsselung wurde „Export Grade (RC4-40 with 40-bit secret key)“ angezeigt, obwohl der Server deutlich stärkere Verschlüsselungsarten beherrscht. Dies liegt daran, daß Netscape den US-amerikanischen Exportbeschränkungen unterliegt und somit auf eine maximale Schlüssellänge von 40 Bit (bei symmetrischer Verschlüsselung) beschränkt ist.

Wenn der Unterzeichner eines neu empfangenen Zertifikats in Netscape registriert ist (also z. B. eine bekannte Zertifizierungsstelle (Certification Authority, CA) wie VeriSign[9] ist), so wird das Zertifikat i. a. nicht angezeigt. Dies läßt sich jedoch in den Sicherheitseinstellungen von Netscape unter Security/Certificates/Signers/Edit für jeden registrierten Unterzeichner getrennt einstellen.

Als nächstes kann man die lokale Gültigkeitsdauer des empfangenen Zertifikats auswählen und hat dabei die Optionen, das Zertifikat entweder nicht zu akzeptieren und die Verbindung zu trennen, das Zertifikat für die aktuelle Session zu akzeptieren oder das Zertifikat bis zum Ende seiner Gültigkeitsdauer zu akzeptieren. Anschließend kann noch eingestellt werden, ob man jederzeit von Netscape gewarnt werden möchte, bevor Daten mittels des zu akzeptierenden Zertifikats übertragen werden.

Nachdem das Test-Zertifikat akzeptiert worden war, wurde eine gesicherte Verbindung zum Webserver hergestellt. Dies wird bei Netscape durch das gelb hinterlegte Symbol eines geschlossenen Schlosses in der linken unteren Ecke symbolisiert. Das Mitschneiden des Datenstroms mittels eines Netzwerk-Sniffers bestätigte, daß die Daten nicht im Klartext übertragen wurden.

4.5 Fortify für Netscape

Die aktuellen Export-Versionen des Netscape Communicators (Version 4.x) enthalten bereits Routinen zur starken Verschlüsselung mittels SSL (RC4 mit 128 Bit und Triple-DES mit 168 Bit). Diese starken Verschlüsselungsarten werden jedoch ausschließlich bei Verbindungen zu spezifischen, speziell zugelassenen Webservern (mit entsprechenden Zertifikaten) aktiviert. VeriSign Inc.[9] erteilt diese Genehmigung im Rahmen seines „Global Server ID certificate program“. Außerhalb der USA ist eine solche Genehmigung jedoch nur für einige spezielle Kategorien von Organisationen erhältlich.

Bei einer Verbindung zu einem nichtzugelassenen SSL Webserver ist die stärkste erlaubte Verschlüsselungsart 56-Bit DES, welche durch die US-Exportbeschränkungen außerdem auf max. 40 wirklich *geheime* Bits beschränkt ist („die restlichen Bits werden im Klartext übertragen“[15]).

Seit einiger Zeit ist für alle gängigen Netscape Communicator-Versionen die Software Fortify[15] erhältlich. Dieses kostenlose Programm installiert sich direkt in den Browser und modifiziert diesen so, daß eine starke Verschlüsselung aktiviert wird, wann immer dies möglich ist (d. h., wenn der Server dies unterstützt). Es werden dann keine speziellen Zertifikate mehr benötigt, um eine stark verschlüsselte Datenverbindung aufzubauen. Da sich Fortify direkt in den Browser installiert werden zur Laufzeit außerdem keine weiteren Hilfsprogramme oder ein spezieller SSL-Proxy benötigt. Die mittels Fortify „erweiterte“ Version des Communicators ist somit genauso „stark“ wie die „U.S. Domestic Version“.

Nachdem der Netscape Communicator in unserer Testumgebung mittels Fortify erweitert worden war, zeigte er bei einem Verbindungsaufbau zu unserem Testserver im „Neues Zertifikat“-Dialog als Verschlüsselung „Highest Grade (RC4 with 128-bit secret key)“ an.

Durch die Veränderung des Konfigurationsparameters `SSLCipherSuite` in der Apache-Konfiguration ist auch die gezielte Verwendung anderer starker Verschlüsselungsarten, wie z. B. Triple-DES, möglich. Bei einer Aktivierung aller verfügbaren starken Verfahren wird jedoch von Netscape Communicator immer 128-Bit RC4 gewählt.

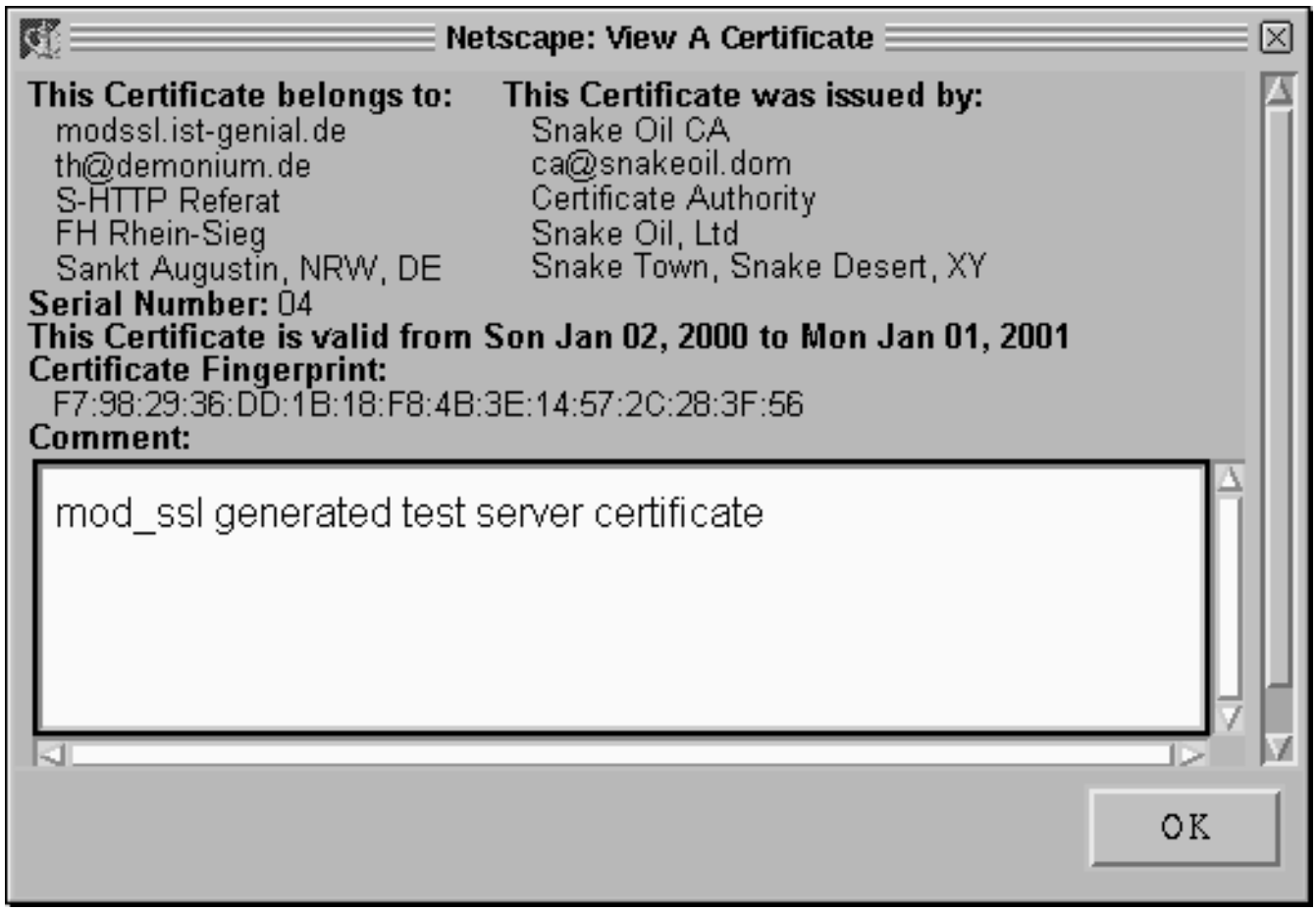

Abbildung 4: Detailansicht eines neues Zertifikats in Netscape Communicator

Negativ fällt auf, daß in der Konfiguration der zu verwendenden Verschlüsselungsarten in Netscape Communicator unter Security/Navigator/Configure SSL v3 (bzw. v2) nach der Installation von Fortify die schwachen Verschlüsselungen standardmäßig weiterhin aktiviert sind (siehe Abbildung 5). Der Grund dafür ist wahrscheinlich die Erhaltung der Kompatiblität zu „Export Grade“-Servern, die nur schwach verschlüsseln können und zu denen ansonsten kein Verbindungsaufbau mehr möglich wäre. Dies führt jedoch zu dem Problem, daß sich der Benutzer evtl. in falscher Sicherheit wiegt, wenn er davon ausgeht, daß alle SSL Verbindungen stark verschlüsselt sind. Beim Empfang eines neuen Zertifikats, welches von einem in Netscape Communicator registrierten Unterzeichner signiert ist, wird die verwendete Verschlüsselung nämlich, wie oben erwähnt, standardmäßig nicht automatisch angezeigt. Bei einer Verbindung z. B. zu einem „Export Grade“-Server würde die Kommunikation dann trotz Fortify nur mit einer Schlüssellänge von 40 Bit verschlüsselt, was als schwach angesehen werden muß.

5 Schlußbetrachtung

Das Secure Sockets Layer Protokoll wird seit der Version 3.0 als sicher angesehen[7]. Zwar gab es verschiedentlich einige spezielle Implementierungen, die Sicherheitsmängel aufwiesen, aber davon abgesehen hat sich SSL in den letzten Jahren auch in der Praxis bewährt. Über den Einsatz im WWW hinaus gibt es mittlerweile außerdem eine Reihe weiterer Anwendung, wie z. B. Telnet-Server und -Clients, deren Kommunikation mit Hilfe von SSL verschlüsselt wird. Diese Tatsache dürfte die weitere Verbreitung von SSL vorantreiben.

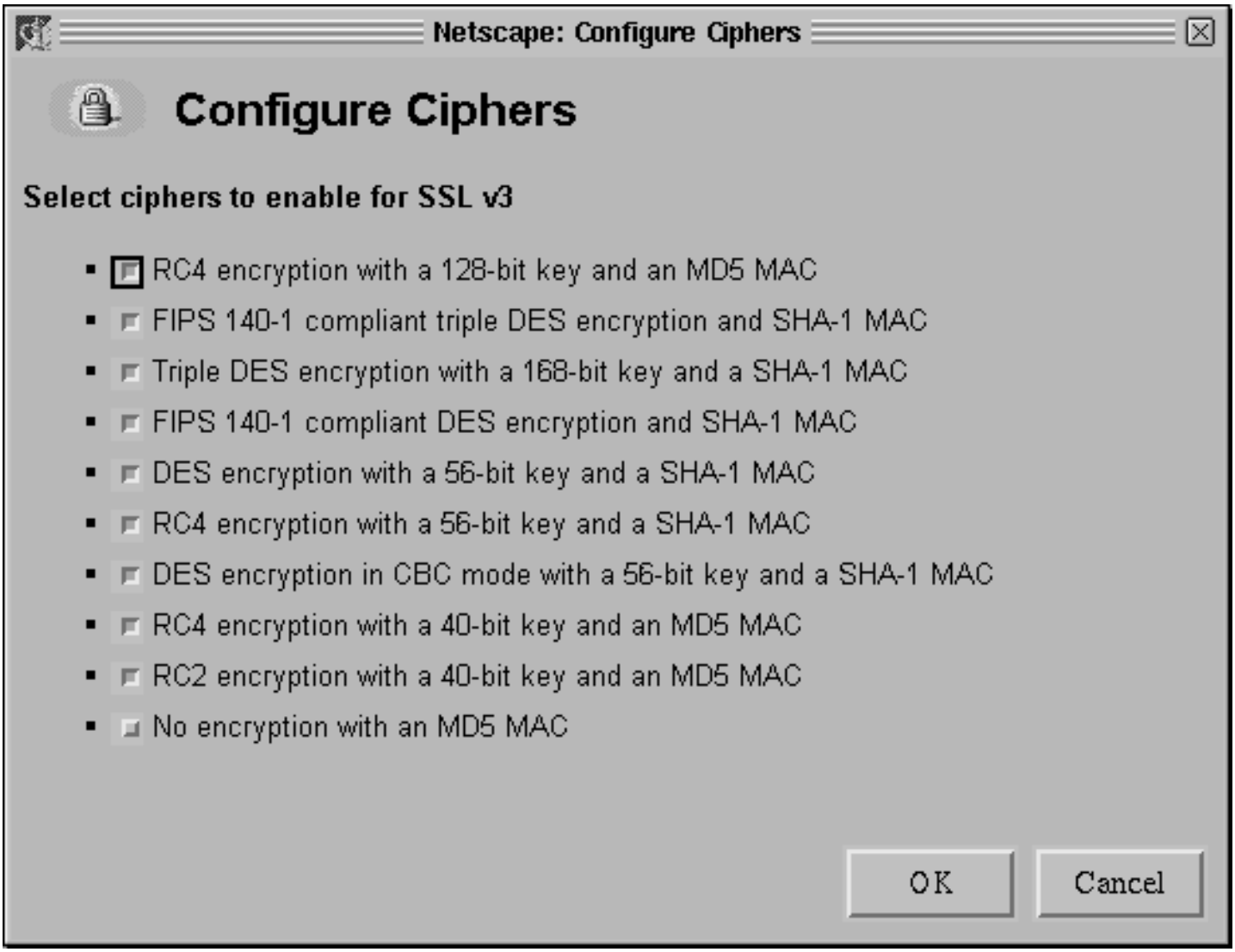

Abbildung 5: Konfiguration der SSLv3-Verschlüsselungsarten in Netscape Communicator

Problematisch ist allerdings, daß viele SSL-Implementierungen aufgrund der US-Exportbeschränkungen mehrere Verschlüsselungsverfahren, wie z. B. DES, nur mit einer Schlüssellänge von 40 Bit unterstützen, wodurch diese als unsicher betrachtet werden müssen. Hierbei handelt es sich jedoch um ein Problem, welches mit allen US-amerikanischen Krypto-Produkten verbunden ist und welches damit genauso auf SHTTP und andere Protokolle zutrifft. Aus diesem Grund kann man daher nur vom Einsatz US-amerikanischer Produkte in diesem Bereich abraten. Beim Anwender kann durch diese Problematik leicht ein Gefühl der Sicherheit entstehen, welches aufgrund der kurzen Schlüssellänge nicht gerechtfertigt ist. Dieses Problem tritt auch, wie oben bereits erwähnt, bei der Verwendung von Fortify auf, da der Netscape Communicator standardmäßig alle Verschlüsselungen akzeptiert. Besser wäre es, wenn bereits ohne eine manuelle Konfiguration die schwachen Verschlüsselungen deaktiviert werden würden.

Insgesamt gesehen bietet die Kombination eines SSL-fähigen Apache Webservers mit einem durch Fortify aufgerüsteten Netscape Communicator jedoch ein hohes Maß an Sicherheit im Internet.

Literatur

[1] RFC 2260: *The Secure HyperText Transfer Protocol.*
`http://www.FreeSoft.org/CIE/RFC/Orig/rfc2660.txt`

[2] Netscape: *SSL 3.0 Specification.*
`http://home.netscape.com/eng/ssl3/index.html`

[3] RFC 2246: *The TLS Protocol Version 1.0.*
`http://www.FreeSoft.org/CIE/RFC/Orig/rfc2246.txt`

[4] Othmar Kyas: *Sicherheit im Internet.* Datacom, S. 207-209.

[5] Spyrus Company.
`http://www.spyrus.com`

[6] Netscape Incorporated.
`http://www.netscape.com`

[7] Bernhard Esslinger, Maik Müller: *Secure sockets Layer (SSL) Protokoll.*
DuD 12/1997, S. 691-697.

[8] ITU Recommendation X.509v3: *The Directory: Authentication Framework.*
`ftp://ftp.cert.dfn.de/pub/tools/crypt/secude/Security/x509v3/`

[9] VeriSign, Inc.
`http://www.verisign.com`

[10] *mod_ssl - The Apache Interface To OpenSSL.*
`http://www.modssl.org`

[11] *The Apache HTTP Server Project.*
`http://www.apache.org`

[12] *The OpenSSL Project.*
`http://www.openssl.org`

[13] *Apache SSL.*
`http://www.apache-ssl.org`

[14] *Debian GNU/Linux - The Universal Operating System.*
`http://www.debian.org`

[15] *Fortify for Netscape.*
`http://www.fortify.net`