Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik IV
Prof. Dr. rer. nat. Otto Spaniol

S – HTTP
Secure Hypertext Transfer Protocol

Seminar: Internetprotokolle für die Multimediakommunikation
WS 02 - 03

Hanno Wirtz
Matrikelnummer: 238008

Betreuung: Stefan Penz
Lehrstuhl für Informatik IV, RWTH Aachen

Inhalt

1 Motivation und Geschichte

S-HTTP (Secure Hypertext Transfer Protocol) [1] wurde ursprünglich von Eric Rescorla und Allan Schiffman im Auftrag des CommerceNet – Konsortiums entworfen, welches sich zur Aufgabe gemacht hat, den Electronic Commerce im Internet zu fördern.

Während im Internet, beziehungsweise im World Wide Web, anfangs eher der Gedanke der kompletten Offenheit von Daten bzw. Informationen vorherrschte, so stieg mit dem rapiden Wachstum des Internets auch das Bedürfnis nach Sicherheit. Gründe hierfür sind die Einführung und Verbreitung von E-Commerce, die steigende Verwendung des Internets als Geschäftsbasis jeglicher Art sowie die Speicherung relevanter Daten in ans Internet angeschlossenen Computern. Generell gesagt also die sichere Bereitstellung zu schützender Daten. Im WWW bildet HTTP hierbei das Transferprotokoll auf der Anwendungsebene, nach dessen Standard die Daten übertragen werden.

Während viele HTTP-Server passwort- bzw. adressbasierte Authentisierung bieten, sind diese Vorkehrungen doch eher einfach zu umgehen. Passwörter können entweder abgefangen oder mittels Brute-Force, also durch simples Ausprobieren, geknackt werden und IP-Adressen können mit relativ einfachen Programmen vorgetäuscht werden. Um also die Sicherheit einer Übertragung von Daten mittels HTTP zu erhöhen, ist der Einsatz kryptographischer Verfahren nötig. Zum einen ist es möglich, die zu versendenden Daten per **Verschlüsselung** so zu modifizieren, dass auch abgefangene Daten Dritten nicht von Nutzen sind. Weiterhin ist die eindeutige Zugehörigkeit der Daten zum Sender, also die **Authentizität** der Daten, und die Korrektheit der übertragenen Daten (**Integrität**) vom Empfänger festzustellen.

Um diese Anforderungen zu gewährleisten, gibt es verschiedene Ansätze, die man grob in zwei Gruppen unterteilen kann. Zum einen die **Absicherung der Verbindung** auf der Transportschicht. Ein bekannter Vertreter dieser Gruppe ist **SSL (Secure Sockets Layer)[8]**. Die andere Art der Absicherung arbeitet auf der **Anwendungsebene**. Secure HTTP gehört zu dieser zweiten Gruppe und erweitert HTTP um einige kryptographische Möglichkeiten. Dabei wird besonderer Wert auf die Flexibilität des Protokolls gelegt, die Vorgehensweise ist auf die Verbreitung und den Einsatz von vielfältigen Sicherheitsmechanismen und –verfahren ausgelegt und so gestaltet, dass keine Kombination derselben ungenutzt bleiben muss, wenn sich Server und Client in dieser Hinsicht einig werden. Herzstück dieser Vorgehensweise ist dabei die Aushandlung der Sicherheitsmassnahmen, mit denen die Übertragung geschützt werden soll (Option Negotiation). Dabei einigen sich Server und Client auf eine Konfiguration von Sicherheitsvorkehrungen, unter denen danach die Übertragung durchgeführt wird.

Die Einsatzgebiete von S-HTTP sind sicherlich weitreichend, da jede Art der Absicherung unterstützt wird, so dass sich jede Sparte von Anwendung aus den generell bestehenden Möglichkeiten die für sie am besten geeignete(n) Möglichkeite(n) heraussuchen kann. Anwendungsbeispiele sind Homebanking, hierbei ist die Unversehrtheit der Nachricht wichtig, und Behördengänge, bei denen natürlich die Authentizität des Antragstellers und der zugeordneten Übermittlung feststehen muss.

2 HTTP – Kurzbeschreibung

S-HTTP stellt eine Erweiterung des HTTP-Standards dar, daher sollen die grundlegenden Funktionen und Eigenschaften von HTTP im Folgenden anhand eines typischen Kommunikationsablaufs kurz vorgestellt werden:

Der Client fordert beim Server an einem bestimmten Port (S-HTTP verwendet wie HTTP Port 80) ein Dokument an und schickt eine Anfrage, die aus drei Teilen besteht :

- Request Line
- Header section
- Entity body

Die **Request Line** besteht aus einer Methode, dem Namen des angeforderten Dokuments und einer HTTP-Versionsnummer. Eine solche Anfrage könnte wie folgt aussehen :

Get / index.html http/1.0

Mit dieser Request Line fordert ein Client mittels der „Get"-Methode das Dokument „/index.html" vom Server an. Weiterhin teilt er diesem die HTTP-Version mit, die zur Übertragung benutzt werden soll.

Die anschliessend folgende **Header Section** teilt dem Server die Konfiguration des Clients und die akzeptierten Dokumententypen mit. Jede Zeile dieses Abschnitts enthält einen Header-Namen und einen passenden Wert. Abgeschlossen wird die Header Section mit einer leeren Zeile. Beispielsweise teilt die Header Section

User-Agent: Mozilla/2.02Gold (WinNT; I)
*Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */**

dem Server mit, dass als Browser Mozilla 2.02 und als Betriebssystem Windows NT verwendet werden. In der Accept-Zeile sind die Dokumententypen mit ihren genauen Ausprägungen aufgeführt, die der Client als Reaktion auf seine Anfrage erwartet und akzeptieren wird.

Anschliessend sendet der Client gegebenenfalls noch zusätzliche Daten im **Entity Body**,
die der Server verarbeiten soll, beispielsweise ein Dokument oder Daten, die zur Bearbeitung seiner Anfrage nötig sind.

Die Antwort des Servers auf diese Anfrage stellt sich ebenfalls aus drei Teilen zusammen :

- Status Line
- Header Section
- Daten

In der **Status Line** befindet sich die HTTP-Version, die der Server zum Beantworten der Anfrage nutzt, der Status Code, der das Resultat der Anfrage beschreibt und die Status Description, der sprachliche Ausdruck des Status Codes. Bei Erfolg würde zum Beispiel

HTTP/1.0 200 OK
zurückgeliefert werden.

Wie auch bei der Anfrage des Clients, macht der Server in der **Header Section** Angaben über sich selbst und das gesendete Dokument.

Date: Fri, 20 Sep 1996 08:17:58 GMT
Server: NCSA/1.5.2
Last-modified: Mon, 17 Jun 1996 21:53:08 GMT
Content-type: text/html
Content-length: 2482

Ist die Anfrage des Clients erfolgreich, so sendet der Server im Anschluss die angeforderten **Daten,** andernfalls eine lesbare Beschreibung, warum der Server die Anfrage nicht erfüllen konnte.

Eine Anfrage bei www.heise.de verläuft beispielsweise wie folgt :

Client-Anfrage :

GET / HTTP/1.1
*Accept: text/html, image/jpeg, image/gif, */**
Accept-Charset: ISO-8859-1
Referrer: http://www.bolege.de/whoiam/
User-Agent: whoiam [http://www.bolege.de/whoiam]
Original-Request-IP: 217.82.197.30
Host: www.heise.de

Server-Antwort:

HTTP/1.1 200 OK
Date: Mon, 28 Oct 2002 22:57:12 GMT
Server: Apache/1.3.26
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html

Im weiteren Verlauf der Server-Antwort wird noch der HTML-Code für die angeforderte Seite aufgeführt.

3 Übersicht über einige kryptographische Verfahren

Die verwendete Kryptographie unter S-HTTP setzt sich aus drei Komponenten zusammen, **Digitale Signatur, Verschlüsselung** und **Authentisierung**. Die Kombination dieser Komponenten ist unter S-HTTP freigestellt und wird bei einer Anfrage zwischen Client und Server ausgehandelt. Im Anschluss sollen anhand von unterstützten Verfahren diese drei Komponenten vorgestellt werden. Die folgende Tabelle zeigt diese Verfahren, von denen die meisten in [7] erklärt werden.

Kryptographisches Verfahren	**Algorithmus**
Hash – Funktionen (Authentizität)	MD2
	MD5
	SHA-1
Verschlüsselungsalgorithmen	DES-CBC
	3DES-CBC
	DESX-CBC
	IDEA-CFB
	RC2-CBC
	RC4
	CDMF-CBC
Digitale Signatur (-Algorithmen)	RSA
	DSS

Von S-HTTP unterstütze kryptographische Verfahren

3.1 Digitale Signatur (anhand von RSA)

Der bekannteste Vertreter dieser kryptographischen Technik ist wahrscheinlich RSA[3] von Rivest, Shamir und Adleman. Dieser Algorithmus wird unter anderem im Internet Explorer von Microsoft und im Netscape Navigator verwendet. RSA verwendet eine **asymmetrische Verschlüsselung** unter Verwendung von **Private Key-Verschlüsselung** und **Public Key-Entschlüsselung.**

Der **Private Key** und der **Public Key** werden beide mittels des RSA – Algorithmus bereitgestellt. Dieser funktioniert folgendermaßen:

- Wahl zweier grosser Primzahlen *p* und *q* (jeweils 256 Bit lang)
- Berechnung von $n = p*q$
- Berechnung der Eulerschen Phi-Funktion $\varphi(n)=(p-1)(q-1)$, d.h. Berechnung der Anzahl der Zahlen kleiner n, die zu n relativ prim sind (p und q bleiben bei Schritt 2 und 3 geheim und werden nicht weiter benötigt)
- Finden einer Zahl *e*, welche relativ prim zu $\varphi(n)$ ist
- Finden des multiplikativen Inversen *d* zu *e* , also $d*e = 1$
- Als Public Key ergibt sich nun (e,n)
- Als Private Key bleiben (d,n)
- Verschlüsselung der Nachricht m über : $c = m^e \bmod n$

- Entschlüsselung der Nachricht m : $m = c^d \bmod n$

Die Sicherheit dieses Verfahrens beruht auf der Tatsache, dass grosse Zahlen nicht in vertretbarer Zeit in ihre Primfaktoren zerlegt werden können, was nötig wäre, um die Sicherheit von RSA aufzuheben. Diese Sicherheit wird allerdings durch einige Experimente mit Quantencomputern gefährdet, bei denen nachgewiesen wurde, dass diese Systeme in der Lage sind, Primzahlen in polynomineller Zeit in ihre Primfaktoren zu zerlegen. Da der Einsatz und erst recht die Verbreitung von Quantencomputern allerdings bisher praktisch gleich null ist, gelten solche Verfahren heutzutage als sicher.

Der Public Key des Empfängers wird nun zur Verschlüsselung einer Nachricht verwendet und diese anschließend über das Internet versendet. Zur Entschlüsselung dieser Nachricht ist der Private Key des Empfängers nötig, den nur dieser selbst besitzt. Um nun sicherzustellen, dass kein Dritter die Nachricht mittels des öffentlich verfügbaren Public Key verschlüsselt und übersendet hat, fügt der Absender seine **Digitale Signatur** bei. Erstellt wird diese mittels einer mathematischen Zusammenfassung der zu signierenden Nachricht, gefertigt mittels einer speziellen Hash-Funktion nach dem Secure Hash Standard (SHS). Hierbei wird der zu versendende Text „mathematisch zusammengefasst“ und später mit dem Private Key des Absenders mit RSA verschlüsselt und bildet nun eine einzigartige Signatur für diese Nachricht. Der Empfänger erstellt nun selber ein Hash der erhaltenen Nachricht und entschlüsselt mittels des Public Keys des Absenders die Signatur. Wenn die Ergebnisse übereinstimmen, kann er zweifelsfrei die Echtheit des Dokuments und des Absenders feststellen. Aufgrund der Tatsache, dass zur Verschlüsselung ein anderer Key benutzt wird als zur Entschlüsselung, gehört dieses Verfahren zur Gruppe der „**asymmetrischen Verschlüsselungs**“-Verfahren. Die hierbei verwendeten Schlüssel haben in der Regel eine Länge von 1024 Bit.

Weiterhin sei gesagt, dass die Verwaltung der Public Keys einer **Certification Authority (CA)** obliegt, bei der die Authentizität des Public Keys überprüft werden kann. Hierzu verschlüsselt die CA auf Anfrage den Namen und den Public Key des Servers mit ihrem Private Key. Bei Erhalt durch den Client wird der Public Key des Servers dann mit dem Public Key der CA entschlüsselt, so dass der Client von der Authentizität des Servers überzeugt sein kann. Diese Zertifikate finden allerdings nicht in allen Verfahren Anwendung.

3.2 Authentizität anhand von HMAC-MD5

Um die Authentizität einer Nachricht zu überprüfen, wird die Methode der **Prüfsummenbildung** benutzt. Hierbei wird ein **Message Authentication Code (MAC)** hergestellt. Dazu werden so genannte One–Way–Hash–Functions benutzt, bei denen es nicht möglich ist, eine Nachricht so zu manipulieren, dass ein gewünschter MAC generiert wird, oder einen MAC mit unterschiedlichen Nachrichten zweimal zu generieren. Im Falle von **MD5 [6]** wird nach Eingabe einer beliebig langen Nachricht eine 128-Bit lange Prüfsumme generiert, welche anschliessend natürlich noch beliebig weiterverschlüsselt werden kann (Public-Key-Verfahren o.ä.). Hierzu wird aus einem **Authentication Header Key** (AH), einem Secret Key der beiden Parteien, der möglichst oft gewechselt werden sollte, und dem Inhalt der Nachricht eine MD5-Prüfsumme erstellt. Diese wird zwischen den beiden Parteien ausgetauscht und gilt als „Digitaler Fingerabdruck“. Der Empfänger bildet bei Erhalt nun

selbst eine Prüfsumme und vergleicht diese mit der erhaltenen. Bei Gleichheit ist sicher, dass die Nachricht vom Absender stammt, da nur dieser den benutzten AH haben kann.

3.3 Verschlüsselung (anhand von IDEA)

IDEA (**International Data Encryption Algorithm**) [5] ist ein symmetrisches Verfahren, das heisst, derselbe Schlüssel wird zum Ver- und Entschlüsseln der Nachricht benutzt. Entwickelt wurde dieser Algorithmus im Jahre 1990 von der Schweizer Bundesanstalt für Technologie. Es wird ein 128 Bit langer Schlüssel generiert. Zunächst wird der Ausgangstext in Blöcke zu jeweils 64 Bit unterteilt, diese wiederum werden in **vier Teilblöcke** gesplittet. IDEA wendet drei Grundoperationen auf den zu verschlüsselnden Text an: Addition, Multiplikation und Exclusive Or (XOR). Unter Verwendung von 12 Teilschlüsseln werden die Teilblöcke 8 Iterationen lang „behandelt“, wobei die vier Teilblöcke vom Anfang der Input für die erste Runde sind. In jeder Runde werden nun diese 4 Teilblöcke miteinander und mit den Teilschlüsseln addiert, multipliziert und XORed, wobei zwischen jeder Runde die Reihenfolge der Teilschlüssel verändert wird. Die, nach der achten Runde, resultierende Zeichenkette ist der Krypt-Text und kann nach dem gleichen Schema wieder entschlüsselt werden, nur dass die Reihenfolge der Teilschlüssel umgekehrt angewandt wird.

Es ist theoretisch möglich, einen Schlüssel unabhängig von seiner Länge mittels Durchprobieren aller Möglichkeiten (**Brute Force**) herauszufinden. Bereits im Falle eines 128-Bit langen Schlüssels würde der Aufwand, alle 2^{128} Möglichkeiten durchzuprobieren, in keinem zeitlich annehmbaren Verhältnis mehr stehen, wenn eine Nachricht entschlüsselt werden soll.

4 Das S-HTTP Verfahren

Von der Reihenfolge und Funktionsweise verläuft jede S-HTTP Anfrage genau wie eine „normale“ HTTP-Anfrage, da S-HTTP eine Erweiterung von HTTP ist. Das bedeutet, dass S-HTTP-Anfragen die gleiche Struktur wie HTTP-Anfragen haben, also zum Beispiel ebenso eine Request Line und eine Header Section besitzen. Der einzige Unterschied ist das so genannte „**Option Negotiation**“–Verfahren, mittels dessen Client und Server die kryptographischen Parameter ihrer Übertragung aushandeln. Hierbei sind S-HTTP-Server konzeptionell so angelegt, dass möglichst große Freiheiten bei der Wahl dieser Optionen gewährleistet werden.

Es kann auch jede Anfrage normaler HTTP-Clients verarbeitet werden, da S-HTTP HTTP nur erweitert, nicht aber ersetzt. Das heißt, so lange keine Sicherheitsbestimmungen des Servers überschritten werden, also zum Beispiel eine Anforderung ein gesichertes Dokument betrifft oder Zugang zu einem Bereich des Servers erbeten wird, der nur mit kryptographischen Absicherungen gewährt wird, ist der Einsatz von Kryptographie nicht von Nöten.

Die kryptographischen Erweiterungen von S-HTTP finden sich hierbei in einer Anzahl zusätzlicher Header, welche die üblichen HTTP-Header erweitern und wie diese über die Sicherheitsvorkehrungen der S-HTTP – Nachricht geschützt werden. Ein Aspekt von S-HTTP ist die Kompatibilität zu sehr vielen kryptographischen Systemen, so dass bei der Wahl derselben so viele Kombinationen wie möglich unterstützt werden. S-HTTP ist damit in der Lage, in jeder Situation beziehungsweise für jede Art von Daten den passenden kryptopgraphischen Schutz zu bieten, immer unter der Prämisse, dass Server und Client gemeinsame kryptographische Verfahren unterstützen. Zu schützende Daten werden natürlich bei einem Scheitern der Verhandlungen nicht übertragen. S-HTTP besteht auf keinem speziellen Verfahren seitens des Clients oder dem Vorhandensein eines Public Keys, wenn auch beispielsweise zur Verschlüsselung standardmäßig, also beim Fehlen entsprechender Angaben seitens des Clients, DES verwendet wird. Die Unabhängigkeit von speziellen kryptographischen Systemen dient der zukünftigen Erweiterbarkeit des Protokolls, also der Möglichkeit, das Protokoll bei Entwicklung neuer kryptographischer Vorkehrungen diesen anzupassen beziehungsweise diese einzufassen.

Ausgehend von einer S-HTTP-fähigen Konfiguration seitens sowohl des Clients als auch des Servers sieht eine Übertragung in der Regel folgendermaßen aus :

- Der Client schickt seine Anfrage unter Angaben der üblichen HTTP-Details und bevorzugter bzw. möglicher kryptographischer Optionen. Hierbei wird zwischen Senden und Empfangen unterschieden, das bedeutet, es werden explizite Angaben gemacht, welche Optionen der Client beim Senden nutzt und welche Optionen beim Empfang von Daten von Nöten sind. Bereits vorhandene Schlüssel, wie ein vom Client generierter Session Key, oder Signaturen können bereits mitgeschickt werden.

- Der Server wählt nun aus diesen Angaben für ihn passende Optionen aus, bedingt durch die Angaben zu den vorhandenen/akzeptierten Verfahren. Im besten Fall sucht der Server ein Public Key-Verfahren zur Übermittlung des Session Key aus und sendet diesen. Signiert der Server diese Nachricht, so werden die entsprechenden Zertifikate, welche die verwendete Signatur bestätigen, bereits mitgeschickt.

- Besteht von beiden Seiten der Wunsch, die Übertragung per Session Key weiterzuführen, so generiert der Client einen solchen (in der Regel ein zufällig

bestimmter Schlüssel) und verschlüsselt diesen mit dem Public Key des Servers. Anschliessend schickt er diesen an den Server, der wiederum entschlüsselt den Session Key und nutzt ihn für die weitere Datenübertragung.

- Alle weiteren Übertragungen werden dann unter den ausgehandelten kryptographischen Vorraussetzungen durchgeführt, also mittels des Session Key und einer passenden Verschlüsselung auf beiden Seiten.

Zu den verwendbaren Verfahren sei noch gesagt, dass immer der stärkste zur Verfügung stehende kryptographische Schutz ausgenutzt werden soll, so lange Server und Client in dieser Hinsicht eine passende Konfiguration haben. So besteht zum Beispiel die Möglichkeit der rekursiven Verschlüsselung (**recursive encryption + encapsulation**), mittels derer S-HTTP-Nachrichten nach Anbringen von Sicherheitsvorkehrungen noch einmal einer solchen Prozedur unterzogen werden können. Auf der Seite des Empfängers wird diese Nachricht dann quasi „zwiebelähnlich“ entschlüsselt, die erste durch S-HTTP geschützte Schale wird entfernt. Da anstatt einer lesbaren HTTP-Nachricht nun wieder S-HTTP-Vorkehrungen offenliegen, werden diese ebenfalls entschlüsselt und so weiter.

Nach Möglichkeit wird dazu noch jede Nachricht mittels eines **MAC** auf Authentizität überprüft (Header : **Message Digest Algorithms**), wobei unter S-HTTP in diesem Zusammenhang auch die Verwendung einer/eines „**Nonce**“ ermöglicht wird, eines Zufallswertes, der die Aktualität der Nachricht anzeigen soll und mit verschiedenen „Lebensdauern“ ausgestattet wird (Header : **Nonce**). So können Nonces nur für die versendete Nachricht gültig oder für die Antwort ebenfalls zulässig sein. Da ein solcher MAC beziehungsweise eine solche Nonce für jede Session und für jede Nachricht neu generiert wird, ist eine Attacke mittels eines bereits versendeten MAC oder Nonce nicht möglich. Sogenannte „Replay Attacks“, bei denen einfach nur der Ablauf einer alten Verbindung nachgespielt wird, werden so unterbunden.

4.1 PreEnhanced Data

Bei Daten, die sehr häufig abgerufen/angefordert werden, werden diese normalerweise im Cache ihres Servers platziert, um bei einer Anfrage schneller bereit zu stehen. Bei sogenannter **Precached Data** kann natürlich der Prozess des **Option Negotiation** nicht mehr durchgeführt werden. Normalerweise ist die Prozedur des Servers bei Anforderung dieser Daten darauf beschränkt, nur die reinen Daten aus dem Cache zu versenden. Handelt es sich bei diesen Daten aber um schutzbedürftige Daten, so stehen mehrere Möglichkeiten zur Verfügung:

- **PreSigned Response/Request**: Eine Nachricht wird signiert und im Cache des Servers platziert. Bei Abruf dieser Nachricht braucht also keine weitere Signierung vorgenommen werden, die Nachricht wird einfach verschickt. Diese Methode hat zwei Vorteile, zum einen ist sie schnell, einfach und kann zum anderen auf unsicheren Servern ebenso durchgeführt werden wie auf gesicherten, da der Server ja nichts mehr mit dem Prozess des signieren zu tun haben muss, also auch nicht im Besitz eines Schlüssels sein muss.

- **PreSigned Documents**: Die zu versendenden Daten werden bereits vor der Platzierung in den Cache des Servers mit einer **Digitalen Signatur** versehen, der

Public Key des Servers ist ja bekannt oder kann zumindest in Erfahrung gebracht werden. Da die zur Überprüfung der Signatur nötigen Daten nun nicht mehr in eine HTTP – Nachricht passen(die passenden Header für die verwendeten Verfahren fehlen), muss diese erweiterte Nachricht wiederum in einer S-HTTP – Nachricht untergebracht werden. Dafür wird eine S-HTTP – Nachricht mit den signierten Daten generiert und als Inhalt einer S-HTTP – Nachricht verschickt.

- **PreEncrypted Messages**: Die Verschlüsselung der Daten findet bereits vor der Versendung durch den Server statt, lediglich die Schlüssel-Übergabe-Optionen (**Key Exchange**) werden neu generiert und ausgehandelt.

5 Schematischer Ablauf einer Anforderung

Eine Anforderung beginnt bereits im HTML-Dokument, im Link zu dem gesicherten Dokument werden die kryptographischen Optionen des Servers für dieses Dokument platziert, die der Client dann übernimmt und in seiner Anfrage verarbeitet.

5.1 Client :

Bereits in der **Request Line** wird der Unterschied zwischen einer HTTP–Message und einer S-HTTP–Message deutlich. In letzterer wird, wie auch in HTTP, die Versionsnummer und das Protokoll genannt, allerdings noch spezifiziert durch eine **spezielle Methode** „**secure**“ :

*Secure * Secure-HTTP/1.1*

Während die meisten signifikanten neuen Header von S-HTTP erst im Zuge der Option Negotiation zum Zuge kommen, so werden doch einige neue HTTP-Header bereits in der ersten **Header Section** eingebaut. Die für die Funktionsweise von S-HTTP interessantesten sind:

- ***Certificate-Info*** : Um die Überprüfung geeigneter **Zertifikate von Digitalen Signaturen** nicht über weitere Nachrichten klären zu müssen, gibt diese Zeile dem Client die Möglichkeit diese direkt in der Anforderung zu schicken.

- ***Key-Assign*** : Dieser Header ist für die Bestimmung und den Austausch der verwendeten Schlüssel zuständig. Den angesprochenen Schlüsseln wird zusätzlich noch eine Lebensdauer (**Lifetime**) zugeteilt:

  - *This* : Schlüssel ist nur für diese Nachricht gültig

  - *Reply* : Schlüssel ist für eine (ggf. mehrere) Antworten gültig

- ***Nonce*** : Um die Aktualität einer Nachricht anzuzeigen, generiert der Server eine dieser Zahlen, die der Client dann als Zeichen der Aktualität in seiner Antwort angeben muss. Unter S-HTTP werden hierfür Zufallswerte empfohlen.

- ***Prearranged-Key-Info*** : Um die eventuell schon verschlüsselte und signierte Nachricht für den Server lesbar zu machen, werden in diesem Header Informationen über den benutzten Schlüssel offengelegt und zwar jeweils in einem eigenen Feld :

  - *Type of Cipher*
  - *Covered Data Exchange Key (DEK)*
  - *CoverKey-ID*

  Hierbei wird im ersten Teil nur die Art des Schlüssels beschrieben beziehungsweise das zugehörige Verfahren, unter Data Exchange Key findet man wiederum den verschlüsselten DEK (Session Key). CoverKey-ID setzt sich aus der Methode (inband/outband) und dem Namen des Schlüssels zusammen. Ein solcher Header könnte beispielsweise so aussehen :

Prearranged-Key-Info: des-ecb,697fa820df8a6e53,inband:1

Bei diesem Header wird also ausgesagt, das DES zum Verschlüsseln des DEK benutzt wurde, und dass dieser DEK „inband“, das heisst also innerhalb der Übertragung, ausgetauscht wurde.

Im Anschluss an die Statusmeldung des Servers beginnt der Vorgang der **Option Negotiation**, im Zuge derer die neuen S-HTTP – Header eine Rolle spielen. Hierbei werden sowohl vom Server wie auch vom Client zwei „Richtungen“ unterschieden :

- **„recv“** (Receive) und
- **„org“** (Originate)

Diese wiederum können drei Ausprägungen haben :

- **„optional”**
- **“required”** (erforderlich)
- **“refused”** (unzulässig)

Unter „recv“ benennt der Client/Server die Verfahren, die er bei Erhalt verschlüsselter Daten akzeptiert (optional), erfordert (required) oder ablehnt (refused). Hier können mehrere Verfahren genannt werden. Dasselbe Schema wird für „org“ verwendet, hier erklärt der Client/Server, welche Schlüssel er beim Senden benutzen kann, will oder verweigert. Ein beispielhafter Headerblock :

```
SHTTP-Privacy-Domains: recv-optional=MOSS, CMS;
      orig-required=CMS
SHTTP-Certificate-Types: recv-optional=X.509;
      orig-required=X.509
SHTTP-Key-Exchange-Algorithms: recv-required=DH;
      orig-optional=Inband,DH
SHTTP-Signature-Algorithms: orig-required=NIST-DSS;
      recv-required=NIST-DSS
SHTTP-Privacy-Enhancements: orig-required=sign;
      orig-optional=encrypt
```

Wie zu sehen ist, können auch Vorgaben wie „sign“ und „encrpyt“ in den entsprechenden Headern gemacht werden, in diesem Falle wird unter Privacy-Enhancements klargestellt, dass die Nachricht beim Versenden immer signiert werden und bei Bedarf auch verschlüsselt werden wird. Jeder dieser Header macht also Angaben über die Möglichkeiten des Clients/des Servers beim Senden (z.B. SHTTP-Signature-Algorithms: orig-required=NIST-DSS) und Empfangen (z.B. SHTTP-Signature-Algorithms: recv-required=NIST-DSS).

Auch an dieser Ausprägungsmöglichkeit erkennt man wieder das Anliegen, S-HTTP möglichst flexibel zu gestalten, um so viele Kombinationen wie möglich zu ermöglichen. Weiterhin kann noch eine Schlüssellänge (**Key Length**) angegeben werden, respektive ein Bereich, in dem sich diese Länge bewegen muss. Wird keine Länge angegeben, so wird jeder Schlüssel unabhängig von seiner Länge akzeptiert. Ein Header dieser Art könnte also beispielsweise so aussehen :

SHTTP-Key-Exchange-Algorithms: recv-required=RSA[1024]

Womit ausgesagt wird, dass erhaltene Schlüssel mit RSA und Schlüssellänge 1024 Bit chiffriert sein müssen.

Die **Option Negotiation Header** sind im einzelnen :

- ***SHTTP-Privacy-Domains*** : Legt fest, nach welchen Schemata Nachrichten erstellt werden, bzw. beim Erhalt akzeptiert werden, beispielsweise :

  SHTTP-Privacy-Domains: orig-required=cms;
  recv-optional=cms,MOSS

  sagt aus, dass beim Senden immer Nachrichten nach dem CMS (Cryptographic Message Syntax)-Standard erzeugt werden und bei Erhalt sowohl CMS als auch MOSS (Mime Object Security Services) akzeptiert werden.

- ***SHTTP-Certificate-Types*** : Zuständig für die Klärung der benötigten und/oder bereitgestellten Public-Key-Zertifikate.

- ***SHTTP-Key-Exchange-Algorithms***:

  - *Outband* : Schlüssel wurden/werden ausserhalb dieser Nachricht/Session ausgetauscht, das heißt, es ist kein weiterer Schlüsselaustausch notwendig. Alle benötigten Schlüssel werden als vorhanden angesehen.

  - *Inband* : Schlüssel werden innhalb der Nachrichten-Session ausgetauscht, in diesem Fall sollte für die Übergabe noch ein Algorithmus definiert werden.

  - *RSA* : Der RSA-Public-Key-Algorithmus wird zum Austausch benutzt.

- ***SHTTP-Signature-Algorithms*** : Erlaubte/Erforderte Algorithmen bzw. Verfahren für Digitale Signaturen.

- ***SHTTP-Message-Digest-Algorithms*** : Erlaubte/Erforderte Algorithmen für Nachrichtenauthentisierung

- ***SHTTP-Symmetric-Content-Algorithms***: Dieser Header beschreibt die symmetrischen Verschlüsselungsverfahren, die zum Verschlüsseln des Inhalts benutzt werden/wurden (z. B. DES)

- ***SHTTP-Symmetric-Header-Algorithms***: In diesem Header werden die zum Verschlüsseln der Header benutzten Verfahren angeführt, ähnlich dem vorhergehenden Header.

- ***SHTTP-Privacy-Enhancements*** : Definiert die eingesetzten Verfahren
    - "auth" : Authentisierung wird angewendet, ein MAC wird bereitgestellt
    - "sign" : Eine Digitale Signatur wird bereitgestellt
    - "encrypt" : Die Nachricht wird mit Verschlüsselungsalgorithmen behandelt.

- ***Your-Key-Pattern*** : Angaben, welche Schlüssel die Gegenseite innerhalb der vereinbarten Optionen nutzen soll, können gemacht werden :

    Your-Key-Pattern: DN-1485
    /OU=Persona Certificate, O="RSA Data Security Inc\." /

    Mit diesem Header wird ausgesagt, dass ein von RSA-Data-Security zertifiztierter Schlüssel benutzt werden soll.

Werden in den zuständigen Headerzeilen (zum Beispiel S-HTTP-Signature Algorithms) keine Angaben gemacht, so geht S-HTTP von den vorgegebenen Standards aus :

- **RSA** zum Austausch von Schlüsseln und Signieren von Nachrichten
- **DES** (Data Encryption Standard) mit 56 Bit Schlüssellänge zum Verschlüsseln von Nachrichten

5.2 Server:

Bei Erhalt der Request Line antwortet des Server mit einer **Status Line**, die genau wie bei HTTP angibt, welches Protokoll benutzt wird und ob die Anfrage diesbezüglich in Ordnung war.

Secure-HTTP/1.1 200 OK

Anschliessend generiert der Server bei Erhalt der Option Negotiation Header, also der kryptographischen Wünsche des Clients, passende Antworten mit seinen Vorschlägen oder den verwendeten Verfahren. Bestenfalls läuft ab diesem Zeitpunkt der gesamte Datenverkehr nur noch mittels Session Key-Verschlüsselung (symmetrische Verschlüssellung, wobei der Schlüssel (DEK) vorher ausgetauscht wurde) und die Daten sind geschützt.

Sollte die Anfrage des Clients nicht S-HTTP – kompatibel sein oder in einer anderen Art und Weise nicht die entsprechenden Sicherheitsvorkehrungen mit sich bringen beziehungsweise vorschlagen, so wird die Antwort des Servers

Unauthorized 401

lauten und es kommt kein Kontakt zustande (s. nächsten Abschnitt).

5.3 Fehler :

Während bei einer Verbindung natürlich die üblichen Fehlermeldungen auftreten können, wie z.B. :

- *Unauthorized 401*
- *PaymentRequired 402*

müssen die S-HTTP – spezifischen Fehlermeldungen nicht den Abbruch der Verbindung oder einen wirklichen Fehler bedeuten, falls diese im Verlaufe der Option Negotiation auftreten. Ein Fehler, der in diesem Falle zurückgeliefert wird, ist :

- *Security Retry 420*

Tritt dieser Fehler bei einer HTTP – Anfrage auf, so sollte die Anfrage wiederholt werden, dieses Mal jedoch unter Einsatz von S-HTTP (soweit vorhanden) und unter Angabe von kryptographischen Optionen.

Während einer S-HTTP – Anfrage bedeutet diese Fehlermeldung in den meisten Fällen nur, dass die angegebenen kryptographischen Optionen vom Server zurückgewiesen wurden. In diesem Fall liefert der Server allerdings in der Header Section seiner Antwort die geforderten kryptographischen Optionen mit, die dann von Client übernommen werden sollten. Unter anderem kann diese Fehlermeldung auch ausgegeben werden, wenn Session Keys veraltet oder eine Entschlüsselung fehlgeschlagen ist, so das ein neuer Schlüssel ausgehandelt werden muss.

Weitere Fehlermeldungen werden in der Regel unter

- ***BogusHeader 421***

ausgegeben, unter anderem aufgrund von Komplikationen mit Proxy-Servern oder Umleitungen.

Zu Fehlermeldungen sei noch gesagt, dass die Autoren von S-HTTP empfehlen, im Falle eines Fehlers eine Eingabe/Bestätigung/Entscheidung des Users abzuwarten, um die integrierte Sicherheit der Session nicht zu gefährden. Beispiel für die Notwendigkeit einer User-Eingabe ist sicherlich der Fall, dass eine Digitale Signatur vom Server nicht akzeptiert wurde, beziehungsweise eine solche Meldung beim User erscheint. Da diese Meldung von einem Dritten eingespielt worden sein könnte, sollte die signierte Nachricht nicht ohne Rückfrage noch einmal signiert und verschickt werden.

5.4 Beispiel einer S-HTTP-Anfrage

Eine HTML-Seite, die einen Link zu einer mit S-HTTP gesicherten Seite enthält, sieht beispielsweise so aus :

<A DN="CN=TEST SERVER, O=RTFM, Inc., C=US"
CRYPTOPTS="
SHTTP-Privacy-Enhancements: recv-required=encrypt;
SHTTP-Key-Exchange-Algorithms: recv-required=RSA;
SHTTP-Symmetric-Content: recv-required=DES-EDE3-CBC"
HREF="shttp://www.rtfm.com./test.html">

Bei Aufruf dieser mit S-HTTP gesicherten Seite (die S-HTTP-Erweiterung gegenüber normalen HTTP-Seiten ist am Präfix zu erkennen), sieht eine Anfrage eines S-HTTP-Clients beispielsweise so aus :

Secure Secure-HTTP/1.4*
Content-Type: message/http
Content-Privacy-Domain: CMS

```
GET /secret HTTP/1.0
Security-Scheme: S-HTTP/1.4
User-Agent: Web-O-Vision 1.2beta
Accept: *.*
Key-Assign: Inband,1,reply,des-ecb,des-ede3-ebc;
     7878787878787878787878787878787878787878
SHTTP-Privacy-Enhancements: recv-required=encrypt,auth
SHTTP-Symmetric-Content-Algorithms: recv-required=DES-EDE3-CBC
SHTTP-Key-Exchange-Algorithms: recv-required=inband
SHTTP-Symmetric-Header-Algorithms: recv-required=DES-EDE3-ECB
```

An der *Secure*-Methode ist für den Server erkennbar, dass es sich um eine S-HTTP-Anfrage handelt, die eine normale HTTP-Nachricht umgibt, erkennbar am Header *Content-Type*. CMS bei *Content-Privacy-Domain* sagt aus, dass die Nachricht dem Cryptographic Message Syntax Standard entspricht. Der restliche Teil der Nachricht ist, wie im Text zu erkennen, unter den vom Server im HTML-Code angegebenen Optionen verschlüsselt. Im Header Key-Assign wird ein vom Client generierter Session Key geliefert, mit dem die restliche Verschlüsselung im Private Key-Verfahren abgewickelt wird. Das bedeutet, dass beide Parteien den gleichen geheimen Schlüssel haben, mit dem sowohl ver- als auch entschlüsselt wird. Dieser geheime Schlüssel ist für die Antwort des Servers als gültig markiert, erkennbar am Reply im *Key Assign*-Header, wird *Inband* ausgetauscht, also innerhalb einer Nachricht (dieser Nachricht) und wurde mit 3DES (Triple-Data-Encryption Standard, Veschlüsselungsalgorithmus von IBM mit drei Schlüsseln der Länge 56 Bit) verschlüsselt. 3DES wird vom Client ebenfalls als Algorithmus zur Verschlüsselung der Header und des Inhalts der folgenden Nachrichten bestimmt (*S-HTTP-Symmetric-Header/Content-Algorithms*). Mit dem Zusatz *recv-required* gibt der Client an, dass an ihn gerichtete, verschlüsselte Nachrichten/Inhalte mit diesem Algorithmus verschlüsselt sein müssen um gelesen zu werden.

Eine passende Antwort des Servers auf diese Anfrage lautet beispielsweise:

Secure-HTTP/1.4 200 OK
Content-Type: message/http
MAC-Info: 31ff8122, rsa-sha-hmac,
jk875gtshrtr784867rebvfeuiorzg8574885r8r8,inband:1
Content-Privacy-Domain: CMS

```
http/1.0 200 OK
Security-Scheme: S-HTTP/1.4
Content-Type: text/html

Congratulation, you've won.
```

Die *Response Line* ist, unabhängig vom Erfolg der eigentlichen HTTP-Anfrage, immer die oben stehende, der eigentliche Fehler ist dann im entschlüsselten HTTP-Header zu finden. Wie in der Anfrage erbeten, hat der Server den Inhalt der Nachricht mit dem bereitgestellten *Session Key* verschlüsselt, nach Entschlüsselung desselben erhält der Client dann den oben stehenden Klartext. Der mitgelieferte MAC wurde ebenfalls mit dem ausgetauschten Schlüssel und HMAC generiert. Nach Entfernen der S-HTTP-Erweiterungen erhält der Client dann eine HTTP-Nachricht, die Auskunft über den Status der eigentlichen Anfrage gibt und, wie in diesem Fall, die angeforderten Informationen enthält.

6 Zusammenfassung und Bewertung

S-HTTP vermittelt in seiner gesamten Konstruktion den Eindruck offen für so viele Sicherheits-Massnahmen wie möglich zu sein und flexibel allen Neuerungen und Erweiterungen der Kommunikationsstandards nachkommen zu wollen. Im Falle einer sich etablierenden neuen Technologie würde diese einfach in den S-HTTP-Standard implementiert und somit würde diese neue Technik unterstützt. Dieser Ansatz ist bei der rasanten Erweiterung des Internets und des kryptographischen Forschungsbereiches auch sicherlich notwendig. Auch die Einbindung mehrerer kryptographischer Optionen und deren Kombination ist sicherlich angebracht, da jede Client-Server-Kommunikation gegebenenfalls über eine eigene Konfiguration von Sicherheitsmassnahmen verfügt und somit verschieden starke Optionen angebracht werden können. Während zum Beispiel DES, von welchem ja bekannt ist, dass dieses mit 56-Bit Schlüssellänge bereits mehrfach und mit genügend Aufwand auch recht schnell geknackt wurde, unterstützt wird, so besteht für brisantere Inhalte die Möglichkeit, eine stärkere Sicherheitsvorkehrung wie zum Beispiel IDEA zu nutzen und genauso unterstützt zu werden.

Zur Bewertung von S-HTTP liegt sicher ein Vergleich mit SSL nahe, auch wegen der unterschiedlichen Auslegung (Anwendungsebene vs. Transportebene) dieser zwei Entwicklungen. SSL wurde als anwendungsunabhängige Absicherung entwickelt, die eher unflexibel den Schutz der Daten realisiert. Während die Gefährdungen wie auch die kryptographischen Techniken (Authentisierung, Verschlüsselung) in ihrer Natur gleich sind, also die Entschlüsselung der verwendeten Kryptographie oder das Abhören einer Verbindung, so bestehen doch Unterschiede in der Implementierung. Ein Vorteil von S-HTTP ist die Möglichkeit, Session Keys neu auszuhandeln (Key Re-Negotiation), während SSL zumindest in den älteren Versionen für die gesamte Zeit der Verbindung einen einzigen Schlüssel benutzt und somit zeitlich mehr Angriffsfläche bietet. Falls ein MAC eingesetzt wird und dieser häufig genug gewechselt wird, beziehungsweise die Lebensdauer des einzelnen Schlüssels kurz gehalten wird, ist die Verbindung auch vor Replay Attacks sicher. Ein Gebiet in welchem dies eine Rolle spielt, ist beispielsweise die Abwicklung von Börsengeschäften über das Internet. Hier könnte eine Anfrage von Dritten abgefangen und mehrfach gesendet werden, so dass die angestrebte Transaktion manipuliert wird . In jedem Fall ist es notwendig, die Authentizität der Nachricht bestimmen zu können, genauso wie die Unversehrtheit der Nachricht gegeben sein muss.

Genau diese Flexibilität ist aber auch sicherlich die Problematik beim Einsatz von S-HTTP. Während die Sicherheits-Politik von SSL nach dem Schema „Alles verschlüsseln und schützen“ verläuft, lässt S-HTTP dem Nutzer mehr Freiheit in der Wahl seiner Sicherheitsmassnahmen, so dass selbige evtl. nur ungenügend eingesetzt werden. Alle HTML-Seiten, die mit SSL geschützt werden sollen, werden mit dem Attribut *secure* ausgestattet. Unter S-HTTP muss für jede Seite die kryptographische Behandlung definiert werden, was bei der Einrichtung eines Servers natürlich erheblich mehr Arbeit und Vorsicht seitens des Administrators erfordert. Dieser zusätzliche Aufwand erschwert natürlich auch die Entwicklung von Standard-Sicherheitslösungen, da auf jede einzelne Ressource gegebenenfalls eine andere Konfiguration angewandt werden soll. Weiterhin stellte sich heraus, dass die angebotene Flexibilität seitens der Nutzer kaum genutzt wurde, da die meisten Nutzer die Bequemlichkeit bei der Implementierung und Administration ihres Servers der Sicherheit einer speziellen Anpassung vorziehen.

Insgesamt stellt S-HTTP also eine flexible Methode, Datenübertragungen gegen Angriffe zu schützen dar, wenn die möglichen Sicherheitsmaßnahmen zunächst einmal vorhanden sind und dann noch seitens der Programmierer richtig eingesetzt werden. Wie gesagt ist diese Freiheit in der Optionswahl die Schwachstelle dieses Protokolls. Diese Freiheit ist sicherlich sowohl eine Möglichkeit viel richtig zu machen, in gleichem Masse bietet sie aber auch die Möglichkeit durch Fehler in der Implementierung und Administration große Löcher zu hinterlassen.

Abschließend bleibt noch zu sagen, dass sich SSL in punkto Verbreitung, Bekanntheitsgrad und Popularität deutlich gegen S-HTTP durchgesetzt hat, jedenfalls ist dies die vorherrschende Meinung, welche von der Tatsache, dass S-HTTP kaum Thema im WWW ist, unterstützt wird. Mit SSL gesicherte Internetseiten kann man am Präfix „https://“ erkennen, dieses steht für „HTTP over SSL“ und ist sicherlich den meisten Internetnutzern bereits einmal begegnet. S-HTTP hingegen würde das Präfix in „shttp://“ ändern. Sicherlich liegt die höhere Popularität an der ursprünglichen Verbreitung von SSL über den Netscape Navigator und der anschließenden Implementierung in den Internet Explorer, eine Rolle spielt aber sicher auch die einfacher einzusetzenden Sicherheitsvorkehrungen auf Seiten von SSL, in deren Steife natürlich auch ein gewisser Vorteil liegt. Der Einsatz von SSL scheint also den meisten Programmierern einfacher zu fallen als die gegebenenfalls umständliche Einrichtung von S-HTTP, die immer voraussetzt, dass beide Seiten dieselben Sicherheitsmechanismen unterstützen oder wenigstens aus einer Anzahl von Sicherheitsmechanismen eine genügend starke Auswahl treffen können.

Die Entwicklung von S-HTTP im Vergleich mit der von SSL beziehungsweise HTTPS (SSL over HTTP) wurde von E. Rescorla [9], einem der Entwickler von S-HTTP mit den Worten „HTTPS is the undisputed winner. [...] every major browser and server implements HTTPS whereas none implements S-HTTP.” charakterisiert. Dieses Eingeständnis zeigt deutlich den momentanen Stand von S-HTTP und auch die Aussichten dieses Protokolls für die Zukunft.

7 Quellenangaben

1 Rescorla/Schiffman, The Secure Hypertext Transfer Protocol
Internetdraft 2660

2 Adam Shostack - An Overview of SHTTP
http://hompeport.org/~adam/shttp.html

3 RSA Verfahren.
R.L.Rivest, A. Shamir, L.A. Adleman:
"A method for obtaining digital signatures and public-key cryptosystems";
Communications of the ACM, Vol.21, Nr.2, 1978, S.120-126

4 Nachschlagewerk
http://searchtechtarget.techtarget.com/

5 IDEA Algorithmus
http://www-lehre.informatik.uni-osnabrueck.de/vsin/vsin03/pgp/node5.html

6 MD5
http://www.faqs.org/rfcs/rfc1321.html

7 Kaufman, Perlman und Speciner, „Network Security“
Prentice Hall PTR, 2002

8 The SSL Protocol, Netscape
http://wp.netscape.com/eng/ssl3/

9 E. Rescorla: "SSL and TLS - Designing and Building Secure Systems", S. 400, Addison-Wesley, 2001