Institut für theoretische Nachrichtentechnik
PROF. DR.-ING. ANTON KUMMERT
Bergische Universität GH Wuppertal

Institut für Nachrichtenübermittlung
PROF. DR. RER. NAT. CHRISTOPH RULAND
Universität GH Siegen

Juli 1998

Diplomarbeit

Entwurf, Realisierung und Verifikation eines Signatursystems für das World-Wide-Web

CHRISTIAN H. GEUER,
christian.geuer@crypto.gun.de

25. Juli 1998

Hiermit versichere ich, daß ich die Arbeit selbständig verfaßt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und Zitate kenntlich gemacht habe.

Inhaltsverzeichnis

1. Einführung

Die vorliegende Diplomarbeit wurde an der „Bergischen Universität GH Wuppertal“ erstellt. Herzlichen Dank an PROF. ANTON KUMMERT vom „Lehrstuhl für theoretische Nachrichtentechnik“ und an PROF. CHRISTOPH RULAND vom „Institut für Nachrichtenübermittlung“ an der „Universität GH Siegen“ für die Betreuung der Arbeit. Mein Dank gilt ebenfalls der Firma Kryptokom (Aachen) für die Bereitstellung der SmartGuard-MailTrusT-Funktionsbibliothek.

1.1. Thema der Diplomarbeit

> Webseiten aus dem Internet sind nicht vertrauenswürdig, solange nicht zusätzliche Sicherheitsmechanismen eingeführt und genutzt werden. Es soll ein Sicherheitssystem konzipiert und realisiert werden, das es dem Endbenutzer ermöglicht zu überprüfen, ob die an ihn übertragenen und ihm angezeigten Webseiten authentisch sind.
>
> Eine Seite besteht aus zusammengesetzten Teilen (Text und Graphiken) und Links auf andere Seiten, Java-Applets, ActiveX-Elemente etc.. Derartige geschachtelte Strukturen müssen berücksichtigt werden. Es müssen verschiedene Lösungsansätze untersucht werden, ob die digitalen Signaturen Bestandteil der Seiten, bzw. der Inhalte werden, oder ob sie als Protokollelemente (z.B. von HTTP[1]) ausgetauscht werden.
>
> Die Realisierung der kryptographischen Funktionen zum Signieren und Verifizieren von Objekten ist nicht Bestandteil der Arbeit, sondern wird als Bibliothek bereitgestellt. Mit Hilfe dieser kryptographischen Funktionen werden die Inhalte digital signiert. Die Signaturen werden auf dem Webserver mit den Inhaltsteilen abgelegt.
>
> Beim Aufrufen der Informationen werden diese mit den Signaturinformationen übertragen und der Endbenutzer kann die Vertrauenswürdigkeit der Webinhalte optimal überprüfen. Dabei erhält er Informationen über den Aufbau der Objekte und über den Sicherheitsstatus der einzelnen Inhaltsteile.

[1] Hypertext transfer protocol

1.2. Motivation

Das Internet wächst mit jedem Tag, neue Hosts und neue Nutzer kommen hinzu. Überall werden Informationen im Überfluß angeboten, deren Echtheit gewährleistet werden muß. Vertrauensverhältnisse zwischen Geschäftspartnern basieren auf der Echtheit und Unverfälschtheit der Daten, ebenso das Ansehen von Personen, Institutionen und Unternehmen.

In der Vergangenheit wurden verschiedene Systeme eingeführt, die die Sicherheit im Internet gewährleisten sollen. Dabei existieren verschiedene sicherheitstechnische Grundbedürfnisse:

- **Integrität** bzw. Unversehrtheit der Daten bedeutet, daß eine Information gegen beabsichtigte oder ungewollte Manipulationen geschützt ist. Der Empfänger der Informationen muß eine Möglichkeit erhalten, die Integrität zu prüfen.
- **Verfügbarkeit** von Diensten und Informationen gewährleistet, daß die berechtigten Nutzer zur rechten Zeit am rechten Ort auf Dienste und Daten zugreifen können. Dieses Merkmal wird oft durch organisatorische Konzepte erreicht. Darunter sind beispielsweise Backup-Systeme für Datenbanken oder Notfallverbindungsstrecken zu verstehen.
- **Vertraulichkeit** sorgt dafür, daß eine Information für alle bis auf die authorisierten Personen geheim bleibt. Private Korrespondenz, Paßwörter und Zahlungsinformationen fallen in diesen Bereich.
- **Authentifikation** ist die Identifikation des Kommunikationspartners. Bei digitalen Daten fehlen die natürlichen Identifikationsmerkmale für Information wie z.B. der Klang der Stimme oder die signifikante Unterschrift. Diese Anonymität der Daten kann durch kryptografische Methoden aufgehoben werden.
- **Nichtabstreitbarkeit/Verbindlichkeit** hat zur Folge, daß frühere Zugeständnisse, Aussagen oder Aktionen nicht mehr abgestritten werden können. (z.B. noratiell beglaubigte Verträge)
- **Authorisierung** oder Zugriffskontrolle klärt nach der Authentifikation eines Benutzers oder Systems ab, ob der Zugriff auf bestimmte Daten erlaubt ist. Nachdem die Authentifikation z.B. über Passwörter erfolgt ist, bestimmt die Datenbank anhand der Identität die eingeräumten Rechte.
- **Anonymität** ist der Authentifikation entgegengerichtet und schützt Kommunikationsteilnehmer vor der Offenlegung ihrer Identität. Bei verschiedenen Anwendungen fordert der Datenschutz dieses Merkmal, beispielsweise im Zahlungsverkehr: die Händler und Banken müssen zwar die Gewähr haben, daß es bei elektronischen Geschäften „mit rechten Dingen“ zugeht und jeder die ihm zustehenden Güter erhält; *wer* aber mit *wem* Geschäfte abwickelt, sollte im Normalfall nicht von Belang sein, solange die Sicherheitsbedürfnisse gewahrt bleiben.

Die für elektronische Signaturen und somit für diese Arbeit relevanten Punkte sind **Integrität** der Nachricht, **Authenzität** des Absenders und **Verbindlichkeit** der getroffenen Aussagen.

Die beiden „Killer"-Applikationen des Internet, d.h. die am häufigsten und offensichtlichsten genutzten Dienste sind e-mail und das WWW. Unter e-mail versteht man das Senden persönlicher Nachrichten von einer Person zur Anderen. WWW ist das sog. „Internet-surfen". Zur Absicherung der beiden Dienste wurden in der Vergangenheit unterschiedlich große Anstrengungen unternommen:

Im Bereich e-mail-Sicherheit gibt es eine große Auswahl an sicheren Produkten, die Vertraulichkeit, Integrität, Authenzität und Verbindlichkeit auf einmal gewährleisten. So können die e-mail-Verschlüsselungsprogramme heute Verschlüsseln (Vertraulichkeit) und Signieren (Integrität, Authenzität und Verbindlichkeit).

Zur Absicherung des WWW wurden verschiedene Ansätze gewählt, die immer nur Teile der obigen Sicherheitskriterien realisieren. Der am weitesten verbreitete Ansatz ist die verschlüsselte Übertragung von Inhalten über „Secure Socket Layers" (SSL). Mit Hilfe von SSL wird eine Verbindung zum Internet-Server aufgebaut. Dieser muß sich gegenüber dem Client eindeutig identifizieren (Authenzität). Dann handeln die beiden Kommunikationspartner geheime Schlüssel aus, die die nachfolgende Verbindung verschlüsseln (Vertraulichkeit). Speichert der Client die übertragenen Daten auf seiner Festplatte und beendet die SSL-Verbindung, so ist die Integrität und Verbindlichkeit der Daten nicht mehr gewährleistet, da diese Kriterien nur während der bestehenden Verbindung erfüllt waren. Da die Daten nur auf der Transportebene gesichert wurden, sind sie ungeschützt, sobald sie abgespeichert werden.

Für viele Anwendungen ist es aber notwendig, Authenzität, Integrität und Verbindlichkeit auch über die Dauer der Netzwerk-Verbindung hinweg zu garantieren. Aus diesem Grunde soll in dieser Arbeit ein Verfahren entwickelt werden, das diese Möglichkeiten bietet.

1.3. Begriffsdefinitionen

An dieser Stelle sollen kurz einige Begriffe erklärt werden, die später noch häufig vorkommen werden.

1.3.1. Internet und Intranet

Internet und Intranet bezeichnen ein- und dieselbe Technik in unterschiedlichen Maßstäben. Das Internet ist ein Verbund von Millionen Computern weltweit. Das zugrundeliegende Netzwerk-Protokoll ist TCP/IP. Innerhalb des Netzes werden viele verschiedene Dienste angeboten.

Das Intranet ist ein Netz, das die gleichen Möglichkeiten wie das Internet hat. Ein Intranet ist bedeutend kleiner und vernetzt beispielsweise innerhalb einer Firma die verschiedenen Abteilungen miteinander.

1.3.2. Kryptografie

Im Abschnitt Kryptografie wird nicht auf die zugrundeliegenden mathematischen Verfahren und Methoden eingegangen. Für Informationen zu diesem Thema sei hier auf [MvV97, Sch96, Rul93, Sta95, Smi98, Wob97, Bau94, FR94, Hor85, BSW95, Wel98] verwiesen.

Zertifikate und Certification Authorities

In vielen Kryptosystemen werden heutzutage asymmetrische Verfahren eingesetzt, d.h. die Schlüssel um Verschlüsselungen bzw. Überprüfungen von Signaturen vorzunehmen sind andere Schlüssel als die zum Entschlüsseln bzw. Signieren von Daten. Das Problem bei diesen Verfahren liegt im Schlüsselmanagment. Sind in einem Kommunikationsnetzwerk nur wenige Parteien eingebunden, erfordert der Schlüsselaustausch keinen großen Aufwand; darüber hinaus wird die Authenzität von Schlüsseln durch den persönlichen Kontakt zwischen den Kommunikationspartnern gewährleistet.

Bei großen Kommunikationsnetzen wie dem Internet bestehen diese persönlichen Bindungen und Verifikationsmöglichkeiten für die Echtheit von Schlüsseln nicht mehr. Aus diesem Grund wurden sog. **Zertifikate** eingeführt. Ein Zertifikat ist ein Datensatz, in dem die wichtigen Informationen zum öffentlichen Schlüssel einer Person gespeichert werden. Zu diesen Informationen gehören neben dem Schlüssel selbst die Identität der Person, der Gültigkeitszeitraum des Schlüssels und eine Seriennummer. Da diese Informationssammlung noch nicht authentisch ist, wurde eine dritte (vertrauenswürdige) Partei, die sog. „Trusted Third Party“ (TTP) eingeführt. Diese TTP garantiert für die Echtheit und Authenzität eines Zertifikates. Im Zertifikat wird dann auch noch die Identität der TTP festgehalten. Die Bezeichnung für diese Art der TTP lautet „**Certification Authority**“, oder kurz CA. Die CA beglaubigt die Echtheit des Zertifikats, indem die das Zertifikat mit ihrem öffentlichen Schlüssel signiert.

Das Grundprinzip, das hinter der Idee mit der CA steht, ist, daß **jeder** Kommunikationsteilnehmer den öffentlichen Schlüssel der CA kennt. Dieser Schlüssel muß auf einem sicheren Weg von der CA zum Teilnehmer gelangen, beispielsweise, wenn der Teilnehmer sein eigenes Zertifikat ausgestellt bekommt.

Zertifikate sind die elektronischen Ausweise in den Datennetzen. Damit ein solches Zertifikat eindeutig kennzeichnet, zu welchem Menschen es gehört, sind im Zertifikat Namen etc. festgehalten. Zertifikate sind gewöhnlich gemäß X.509 (siehe [Int93]) aufgebaut. X.509 beschreibt den Aufbau von digitalen Zertifikaten. Die Namen des Zertifikatsinhabers und der CA sind gemäß X.500 aufgebaut. X.500 ist ein ITU-Standard, der Directory-Services beschreibt. Ein solcher Name kann z.B. „`c=DE o=Universität Wuppertal sn=Geuer`“ lauten. Dabei sind den verschiedenen Attributen unterschiedliche Bedeutungen zugewiesen: „c“ steht für Country, „o“ für Organisation und „sn“ bedeutet Surname. Diese Attribute zusammen bilden zusammen den „Distinguished Name“, der in Zusammenhang mit Zertifikaten immer wieder auftaucht.

Public Key Infrastructure

Die nächste Schwierigkeit tritt beim Handling dieser Zertifikate auf. Gäbe es weltweit nur eine einzige CA, hätte das weitreichende Konsequenzen. Diese CA hätte durch ihre Monopol-Stellung eine unglaubliche Macht. Teilnehmer, denen diese CA das Zertifikat verwehrt, wären in den Datennetzen praktisch nicht existent, da sie keine gültigen Unterschriften leisten könnten.

Ein anderes Problem wäre die Machbarkeit. Wenn 1 Milliarde Menschen weltweit ein Zertifikat ausgestellt bekommen wollten oder beglaubigt bekommen lassen wollten, wäre das organisatorisch fast nicht zu lösen.

Aus diesem Grund sind verschiedene Infrastrukturen denkbar, in denen mehrere CA's Zertifikate ausstellen könnten. So könnte beispielsweise die CA der UNO für die Echtheit der Zertifikate der CA's der einzelnen Staaten garantieren. Die CA's der jeweiligen Staaten könnten Zertifikate für die CA's der einzelnen Städte ausstellen und diese wiederum würden die Zertifikate der Bürger beglaubigen.

Die verschiedenen möglichen Strukturen werden einfach unter dem Oberbegriff „**Public Key Infrastructure**“ (PKI) zusammengefasst.

1.3.3. Interessengruppen – Definitionen

In dieser Arbeit kommen verschiedene Interessengruppen vor. Die Bewertung bezüglich des Vertrauens erfolgt vom Endbenutzer aus gesehen:

Der Unterschreibende

Der **Signierer** oder **Inhalte-Anbieter** ist die Person, die die Inhalte und Signaturen anbietet. Der Signierer kann z.B. ein Unternehmen sein, welches Preiskataloge auf dem eigenen Web-Server anbietet oder auch ein Journalist, der seine Artikel auf dem Web-Server eines beliebigen Internet-Providers ablegt und signieren möchte.

Für den Signierer ist wichtig, daß er die Signaturen nicht auf dem Web-Server erzeugen muß, sondern z.B. Zuhause oder in der Firma auf einem abgetrennten System ohne Zugriff zum Internet generieren kann und dann auf einem beliebigen Weg auf den Web-Server übertragen kann. Zum Zeitpunkt der Signatur sollte keine Verbindung zwischen Internet und dem PSE (Personal Security Environment) bestehen, um im Vorfeld Gefahren ausschalten zu können. Das PSE ist der sichere Speicher für die privaten Signaturschlüssel.

Wenn der Signierer ein gültiges Zertifikat vorweisen kann (Authenzität) und Inhalte mit diesem Zertifikat signiert, muß davon ausgegangen werden, daß der Signierer zu diesen Aussagen steht (Verbindlichkeit).

Der Server-Betreiber

Der **Web-Server-Betreiber** als Dienst-Anbieter speichert die Inhalte und Signaturen für den Inhalte-Anbieter. Er hat mit Erzeugung oder Verifikation der Signatur nichts zu tun. Seine Aufgabe liegt nur in Speicherung und Übertragung der Inhalte.

Der Web-Server-Betreiber hat kein großes Interesse daran, auf dem Server liegende Daten zu modifizieren, da dieser Betrug dem Inhalte-Anbieter auffallen könnte und dieser dann sein Geschäftsverhältnis mit dem Web-Server-Betreiber beenden könnte. Nichts desto trotz darf es dem Betreiber des Server nicht möglich sein, unbemerkt Modifikationen durchzuführen oder ungültige Angebote und Inhalte einzuspielen.

Der Verbindungsanbieter

Die Proxy-Server im Internet werden von **Kommunikationsgesellschaften** betrieben. Proxy-Server[2] sind Server im Internet, die zwischen dem Web-Server und den Clients (Surfern) stehen. Proxy-Server können mehrere Aufgaben wahrnehmen: Als Proxy-Cache können sie die Inhalte zwischenspeichern und als Application-Proxy auf Firewalls nach außenhin als Client auftreten.

Caching bedeutet, daß häufig erfragte Inhalte nicht jedesmal neu vom Ursprungsserver übertragen werden müssen, sondern im Cache des Proxy zwischengespeichert werden. Fordert ein Client einen bestimmten URL an, so prüft der Proxy mit einer kurzen Frage beim Original-Server, ob die zwischengespeicherte Information noch aktuell ist. Ist der Cache noch aktuell, kann der zwischengespeicherte Datensatz zum Client geliefert werden. Ansonsten muß die Information neu geholt werden.

Application-Proxy auf einem Firewall-Rechner schützen die Clients im internen Netz vor den Gefahren des außen liegenden Internet. Dieser Application-Proxy entscheidet nicht auf Transport-Ebene, ob gewisse Datenpakete durchgelassen werden, sondern analysiert das HTTP-Protokoll, welches zum Anfordern von Web-Informationen verwendet wird. So kann er beispielsweise Zugriffe auf bestimmte Sites abblocken oder die Angabe von Cookies und e-mail-Adressen wegfiltern. Application-Proxies können natürlich auch noch ein zusätzliches Caching implementieren.

Proxy-Server sind als vollkommen unsicher einzustufen. Ihnen wird kein Vertrauen entgegengebracht. Da man den Weg der Datenpakete durch das Internet nicht ohne weiteres bestimmen kann, muß davon ausgegangen werden, daß die Daten viele nicht-vertrauenswürdige Rechner passieren. Überall dort können neue Daten in den Datenstrom eingefügt werden, Daten gelöscht und modifiziert (Integrität) und aufgezeichnet (Vertraulichkeit) werden.

[2] Proxy – Stellvertreter

Der Prüfer der Unterschrift

Der **Client** ist der eigentliche Internet-Surfer. Die auf dem Web-Server abgelegten Inhalte und Signaturen werden vom Client angefordert und ausgewertet. Der Client ist aus eigener Sicht gesehen absolut vertrauenswürdig. Ihm gegenüber muß von den anderen Parteien das Vertrauen bewiesen werden.

Zertifikats-Beglaubiger

Certification-Authorities müssen als a-priori vertrauenswürdig gelten. CA's sind zweifelsohne Unternehmen mit finanziellen Interessen, doch für die Funktion des Systems sind CA's als sicher einzustufen.

1.3.4. HTTP

HTTP steht für „Hypertext Transfer Protocol" und ist in [FFB96, FGM+97] spezifiziert. HTTP ist ein stromorientiertes, zustandsloses, auf TCP/IP[3] aufsetzendes Transport-Protokoll. Stromorientiert ist im Gegensatz zu Datagramm-orientierten Diensten wie UDP[4] so zu sehen, daß die Reihenfolge der Symbole, die in die Kommunikationsverbindung gesendet werden, ihre Reihenfolge nicht ändern und daß das Protokoll TCP für die Sicherung der Verbindung zuständig ist.

Zustandslos bedeutet, daß der HTTP-Server keine Verbindung zwischen zwei Requests herstellen kann. Der Server kann sich nicht merken, welche Requests zusammengehören. Jeder Request ist eine in sich abgeschlossene Einheit.

Eine Verbindung in HTTP sieht im Normalfall so aus, daß der Client sich via TCP/IP mit Port 80 auf dem HTTP-Server verbindet (z.B. www.uni-wuppertal.de und dort seinen Auftrag absetzt. Hat der Client seinen Auftrag (Request) übermittelt, sendet der Server seine Antwort (Response) zum Client zurück und beendet die Verbindung.

Struktur eines Client-Request Der Client-Request läßt sich in verschiedene Teile untergliedern:

Request-Zeile: Die erste vom Client gesendete Zeile umfaßt die Methode, die URI und die HTTP-Version. Ein Beispiel könnte so aussehen:

```
GET /index.html HTTP/1.0
```

Die verwendete Methode GET besagt, daß die nachfolgende URI /index.html angefordert wird. Die von HTTP/1.0 gebotenen Methoden sind GET, POST, HEAD, PUT, LINK, UNLINK und DELETE.

[3] Transport Control Protocol – Internet Protocol
[4] User Datagram Protocol

Die gebräuchlisten sind `GET` zum Anfordern von Dateien, `POST` zum Senden von Formularen und `HEAD` zum Abfragen der Parameter der geforderten URI.

Allgemeine Header: Dann können allgemeine Header wie z.B. das Datum folgen:

```
Date: Sun, 19 Jul 1998 14:47:34 GMT
```

Request-Header: Darauf folgen Request-spezifische Header wie der verwendete Browser, die letzte geladene URI (der Referer) oder die vom Client unterstützten Bildformate:

```
User-Agent: Mozilla/3.0 Gold (WinNT; I)
Referer: http://www.stud.uni-wuppertal.de/~geuer/links.html
Accept: image/gif, image/jpeg, */*
```

Body-Header: Wenn noch ein Body gesendet wird, können für den Inhalt des Bodys spezielle Parameter festgelegt werden, wie z.B. den MIME-Typ oder die Länge des Bodys:

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 23
```

Dann folgt eine Leerzeile. Wenn im Body-Header festgelegt wurde, daß noch Daten folgen, wartet der Server mit der Verarbeitung des Requests.

Body: Wenn noch ein Body folgt, sendet der Client den Inhalt.

Struktur eines Server-Response Der Server-Response läßt sich ebenfalls in verschiedene Teile untergliedern:

Die erste Response-Zeile: Die erste vom Server gesendete Zeile umfaßt die HTTP-Version, einen Statuscode und noch einen „menschenlesbaren“ Ergebnis-Code. Ein Beispiel könnte so aussehen:

```
HTTP/1.1 200 OK
```

Als erstes bekommt der Client die HTTP-Versionsnummer des Servers geliefert. Der Statuscode (200) gibt numerisch das Ergebnis des Requests wieder. Der Grund kann dem Client im Fehlerfall präsentiert werden.

Allgemeine Header: Die allgemeinen Header können auch hier wieder Angaben wie das Datum umfassen:

```
Date: Sun, 19 Jul 1998 14:47:34 GMT
```

Response Header: Die Response-Header geben Aufschluß über den verwendeten Server:

```
Server: Apache/1.2.5 mod_perl/1.08
```

In diesem Fall besteht die Server-Software aus dem Apache-Server mit Versionsnummer 1.2.5 und einem eincompilierten Modul `mod_perl` in der Version 1.08.

Body Header: Die Body-Header geben Aufschluß über den Inhalt des Body; hier ist beispielsweise das letzte Modifikationsdatum der Datei festgehalten, die Länge des Inhalt (`Content-Length`) oder ein Base64-codierter Hash über den Response-Body (`Content-MD5`). Im `Content-Type` ist der MIME-Typ des Response-Body festgehalten.

```
Last-Modified: Sat, 18 Jul 1998 16:03:58 GMT
Content-Length: 1343
Content-MD5: NI11oooQLhbxG7+crfhtOg==
Connection: close
Content-Type: text/html
```

Nach einer Leerzeile folgen die eigentlichen Daten, also in diesem Fall eine HTML-Datei mit 1343 Bytes Länge, die das letzte Mal am 18. Juli 1998 modifiziert wurde.

Eine HTTP-Beispiel-Session könnte also folgendermaßen aussehen:

```
GET /index.html HTTP/1.0
Date: Sun, 19 Jul 1998 14:47:32 GMT
User-Agent: Mozilla/3.0 Gold (WinNT; I)
Referer: http://www.stud.uni-wuppertal.de/~geuer/links.html
Accept: image/gif, image/jpeg, */*
Content-Type: text/plain
Content-Length: 5

Hallo

HTTP/1.1 200 OK
Date: Sun, 19 Jul 1998 14:47:34 GMT
Server: Apache/1.2.5 mod_perl/1.08
Last-Modified: Sat, 18 Jul 1998 16:03:52 GMT
```

```
Content-Length: 1343
Content-MD5: NI11oooQLhbxG7+crfhtOg==
Connection: close
Content-Type: text/html

<HTML>
<TITLE>
<HEAD>Titelseite</HEAD>
</TITLE>
<BODY>
...
</BODY>
</HTML>
```

1.3.5. HTML

Die „Hypertext Markup Language“ (HTML) ist die Seitenbeschreibungssprache des Internet. Ein Hypertext ist ein Text, in dem Querverweise auf andere Textstellen innerhalb desselben Textes oder auf andere Texte vermerkt sind. Durch einfaches Auswählen dieser Querverweise, auch Hyperlinks genannt, bekommt der Leser Zugriff auf den neuen Text.

HTML stellt dem Autor Stilelemente für eine Seitenstruktur zur Verfügung. In HTML können beispielsweise Überschriften, Tabellen, Listen oder Absätze geschrieben werden. Zusätzlich zum normalen Text können auch Hyperlinks in die Seite eingebunden werden. Diese Hyperlinks sind Querverweise auf andere Seiten oder Inhalte. Durch diese Querverweise, im folgenden nur noch Hyperlinks oder Links genannt, erhält das WWW seine Dynamik. Der Nutzer kann durch einfaches Anklicken mit der Maus einen neuen Inhalt in sein Internet-Programm (Browser) laden. So kann eine HTML-Seite beispielsweise ein Inhaltsverzeichnis enthalten und die Links zeigen auf die Seiten, in denen die jeweiligen Kapitel gespeichert sind. Wählt der Nutzer nun einen Hyperlink aus, verschwindet das Inhaltsverzeichnis und das Programm stellt das ausgewählte Kapitel dar.

HTML bietet aber noch mehr Möglichkeiten. So kann der Browser angewiesen werden, einen Link sofort zu expandieren und an der aktuellen Stelle einzufügen. Auf diese Art und Weise werden Bilder, die in einer anderen Datei gespeichert sind, im Fließtext mit angezeigt. Dieses Einbinden in das aktuelle Dokument bieten auch sog. Frames (Rahmen). In einem Frame kann eine andere HTML-Seite dargestellt werden. Bei einem zweigeteiltem Fenster könnte beispielsweise auf der linken Seite das Inhaltsverzeichnis und auf der Rechten das eigentliche Kapitel angezeigt werden.

HTML unterstützt Formulare. Je nach Formular kann Text eingegeben werden, verschiedene Auswahlen können getroffen werden und die Formulardaten können zurück zum Internet-Server gesendet werden. Dieser kann die Daten dann mit sog. CGI-Programmen weiterverarbeiten und dem Nutzer neue HTML-Seiten oder Daten zurücksenden. CGI steht für „Common Gateway

Interface“ und ist eine Methode, um beliebige Formulardaten vom Client zum Server übertragen zu können.

Innerhalb von HTML-Seiten können auch Programme eingekapselt werden. Bei diesen Programmen handelt es sich um Skripten, die der Browser ausführen kann. Bekanntestes Beispiel ist JavaScript. JavaScript hat (außer den Anfangsbuchstaben) nichts mit Java zu tun. JavaScript-Programme werden direkt in den HTML-Quelltext eingefügt und können beispielsweise kleine Berechnungen durchführen, abhängig von der Mausposition über dem Dokument verschiedene Texte anzeigen oder vor dem Absenden von Formulardaten eine einfache Konsistenz-Prüfung der Daten durchführen. JavaScript und andere Skriptsprachen sollen einfache Aktionen beim Client ausführen können, ohne daß jedesmal der Server involviert sein muß.

Ein Beispiel für eine einfache HTML-Seite könnte z.B. so aussehen:

```
<HTML>
<HEAD>
<TITLE>Homepage des Beispiel-Servers</TITLE>
</HEAD>
<BODY>

<H1>Haupt-&Uuml;berschrift</H1>

Umlaute werden mit einer speziellen Codierung (wie bei der Ueberschrift
gesehen) codiert. In spitzen Klammerpaaren werden die sog. HTML-Tags
eingebaut. Das obige Tag H1 besagt beispielsweise, daß hier eine
Ueberschrift (Heading) des Typen 1 steht.

<A HREF="http://www.perl.com/"> Bitte hier clicken,um zur Perl-Homepage
zu kommen </A>

</BODY>
</HTML>
```

Mit dem `<A HREF=URL> ... </A>`-Tag wird ein Link auf eine andere Internet-Seite erzeugt.

1.3.6. URL

Der „Uniform Resource Locator“ ist das Beschreibungsformat für einen Link. Ein URL ist eine Internet-Adresse. Ein URL kann z.B. so aussehen:

```
http://www.uni-wuppertal.de:80/verz1/verz2/inhalt.html
```

Ein HTTP-URL besteht aus verschiedenen Komponenten:

1. Das Schema (`http:`) beschreibt die Methode, die zum Zugriff auf den URL angewendet werden muß. Im vorangegangenen Beispiel ist hier die HTTP-Methode anzuwenden. Die gebräuchlisten Methoden sind:

- `http:` Beim Zugriff über HTTP wird das Hypertext Transfer Protocol zur Übertragung der Daten verwendet.
- `ftp:` beschreibt den Zugriff über das „File Transfer Protocol“. FTP war früher der gebräuchliste Dienst zum übertragen von Dateien. FTP kommt auch heute noch zum Einsatz. `ftp://ftp.uni-wuppertal.de/pub/comp/os/unix/ls.tar.gz`
- `mailto:` enthält immer eine e-mail-Adresse. Durch Auswählen eines mailto-URL's wird im Normalfall eine Nachricht für die angegebene Adresse erzeugt. Beispiel: `mailto:christian.geuer@crypto.gun.de`
- `news:` ist ein Verweis auf eine Usenet-Newsgroup. Newsgroups sind die öffentlichen Diskussionsforen im Internet. Beispiel: `news:sci.crypt.research`

2. Die Internet-Adresse (voll qualifizierter Hostname) des anzusprechenden Rechners. Im obigen Beispiel wäre das `www.uni-wuppertal.de`
3. Wahlweise kann zusammen mit dem Hostnamen durch einen Doppelpunkt abgetrennt der zu verwendende Port angegeben werden. Wird kein spezieller Port mit angegeben, wird die Standard-Portnummer (80 für HTTP) verwendet. Die Portnummer ist deshalb wichtig, weil auf einem Rechner u.U. mehrere Server parallel arbeiten, die auf unterschiedliche Daten zugreifen.
4. Der Pfad zur angeforderten Datei. Im obigen Beispiel wäre das `/verz1/verz2/`
5. Der Dateiname selbst: `inhalt.html`

2. Bestehende Signatur-Systeme im Internet

Das Kapitel „Bestehende Systeme“ beschreibt und stellt kurz die schon produktiv einsetzbaren bzw. spezifizierten Signatur-Verfahren vor, die im Bereich Internet/Intranet existieren. Dieser Abschnitt behandelt keine Systeme, die die Transportschicht oder Anwendungsebene sichern. „Secure Socket Layers“ beispielsweise sichert die Transportschicht ab und gewährt Integrität und Vertraulichkeit für die Dauer der Verbindung. Die hier untersuchten Systeme erzeugen Signaturen, die auch nach dem Transport der Nachricht noch bestehen.

Im folgenden soll kurz erläutert werden, welche prinzipiellen Unterschiede bestehen, bevor genauer auf die einzelnen Systeme eingegangen wird.

Bei den heute existierenden Systemen können zwei Klassen von Verfahren unterschieden werden:

- **Hersteller- bzw. datengebundene Verfahren:** Es existieren verschiedene Ansätze, um spezielle Datenformate zu schützen, z.B. Signatur-Systeme zum sog. „Code-Signing“, d.h. dem Signieren von Programmen. Diese signierten Programme können einzelne Java-Programme (Applets) sein, Sammlungen von Java-Code in sog. JAR-Files (Java-ARchive) oder auch ActiveX-Programme. Darüber hinaus sind auch andere datengebundene Formate denkbar, z.B. signierte Bilder (Copyright) oder Tondateien.

  Die Festlegung auf einen speziellen zu signierenden Datentyp hat einen großen Vorteil: Die Signatur kann auf Anwendungsebene interpretiert und abgetrennt werden. Der Programm-Interpreter oder das Bild-Darstellungsprogramm können selbstständig entscheiden, wie spezielle Inhalte zu behandeln sind, welche Rechte einem auszuführenden Programm einzuräumen sind oder ob ein Bild dargestellt werden soll.

  Ein weiterer Vorteil liegt darin begründet, daß ein so gekapseltes Objekt die Signatur als integraler Bestandteil beinhaltet, d.h., die Signatur bleibt bei der Übertragung oder Speicherung fest an die Daten gebunden und kann erst von der Anwendungsebene entfernt werden.

- **Universell einsetzbare Verfahren:** Bei den universell einsetzbaren Verfahren handelt es sich um Systeme, die ursprünglich für die Verschlüsselung und Signatur von e-mail entwickelt wurden. Diese Systeme kapseln die signierten Daten in der Signatur ein (ein einzelnes Objekt) oder erzeugen die Signatur als separates Objekt, welches zu dem Datenobjekt korrespondiert (zwei separate Objekte). Durch den allgemeineren Ansatz können im Gegensatz zu den Formatgebundenen Systemen beliebige Eingangsdaten signiert werden.

Diesen Vorteil erkauft man sich durch eine schlechtere (d.h. weniger spezifische) Anbindung an das jeweilige Datenformat.

2.1. Datengebundene Signatur-Systeme

2.1.1. Code Signing

„**Code Signing**“ bezeichnet das Signieren von Programmen. Das Signieren von Programmen soll dem Anwender und dem Betriebssystem Aufschluß über die Herkunft von Programmen bieten. In verteilten Umgebungen wie dem Internet werden nicht nur Nutzdaten ausgetauscht, sondern auch Programm-Code. In den letzten Jahren wurden verschiedene Ansätze getätigt, Plattform-übergreifende Sprachen zu entwickeln, die auf der Client-Seite ausgeführt werden. Zu Beginn der WWW[1]-Nutzung bestand lagen auf den Servern nur statische Inhalte, d.h., man konnte Dokumentation in Text- und Bildform auf den Client laden.

Danach wurden die ersten Ansätze zur Dynamisierung und Automatisierung unternommen: Der Client kann über HTML-Formulare CGI-Programme (Common Gateway Interface) auf dem WWW-Server starten, welche dann z.B. eine Datenbankabfrage durchführen und dem Client entsprechende formatiert die Ergebnisse zurücksenden.

Der bislang letzte Schritt liegt im Herunterladen von Programmen, die dann auf der Client-Seite ausgeführt werden. Diese heruntergeladenen Programme beinhalten eine große potentielle Gefahr: Da der Programmcode auf der Clientmaschiene ausgeführt oder interpretiert wird, kann ein Angreifer auf dem Server Programme zum Download ablegen, die die Systemintegrität des Clients verletzen, wenn dieser sie ausführt. Diese Programme können beispielsweise sensitive Daten vom Rechner des Client lesen und über das Netzwerk zurück zum Angreifer senden (Vertraulichkeit verletzt) oder auch Manipulationen an Daten oder Programmen auf dem Rechner des Client durchführen (Daten-Integrität verletzt).

Um diesen Gefahren nicht ausgesetzt zu sein, wählen viele Nutzer des WWW den Weg des Sperrens dieser Möglichkeiten: In den Internet-Browsern wird das automatische Herunterladen und Ausführen von Code komplett gesperrt. Dadurch ist der Client nicht mehr der Gefahr von „bösen“ Programmen ausgesetzt. Die Problematik an diesem Ansatz ist, daß jetzt auch die vielen ungefährlichen Programme nicht mehr ausgeführt werden, die eine Internet-Seite dynamisieren, bunter und schöner machen oder auch Berechnungen für den Client durchführen.

Daher wurde das Code-Signing entwickelt. An die Programme wird eine digitale Signatur angefügt, anhand deren der Client von Fall zu Fall entscheiden kann, ob die Quelle des Programms vertrauenswürdig ist, bzw. ob des Programm verändert wurde und die Signatur deshalb nicht mehr gilt. Der Client kann dann beispielsweise allen Programmen eines bestimmten Entwicklers uneingeschränkten Zugriff auf Systemressourcen geben, ohne dabei die ganze Systemsi-

[1] World Wide Web – Das Übertragungsprotokoll HTTP (HyperText Transfer Protocol) und die Seitenbeschreibungssprache HTML (HyperText Markup Language) bilden zusammen den Dienst, der landläufig als WWW bezeichnet wird.

cherheit zu gefährden. Je nach verwendetem System können dann recht feine Unterschiede in der Abstufung der Rechte für verschiedene Programme machen.

Problematisch wird das System durch die Aussagekraft der digitalen Signatur. Diese digitalen Signaturen basieren auf Zertifikaten, die dem Programmentwickler von TTP's (Trusted Third Party's) bzw. CA's (Certificate Authority) ausgestellt werden. Diese garantieren mit ihrem Zertifikat für die Identität des Entwicklers. Welche Programme der Entwickler mit diesem Zertifikat signiert, bleibt ihm überlassen. Daß ein Programm mit einem gültigen Zertifikat signiert wurde, sagt also nichts über die Ungefährlichkeit des Programms aus. Die CA führt kein Code-Review durch, um zu sichern, daß das Zertifikat nicht für kriminelle Zwecke mißbraucht wird.

Java und „Signed Applets"

Java ist **die** Sprache, wenn es um plattformübergreifenden Code geht. Java wurde von der Firma SUN spezifiziert und stellt eine Möglichkeit dar, ein Programm auf vielen verschiedenen Plattformen starten zu können. SUN hat ein recht restriktives Sicherheitsmodell für Java definiert, welches jedes Java-Programm, im folgenden auch Applet genannt, als potentiell gefährlich einstuft, wenn es aus dem Netz geladen wurde. Diese Applets werden dann in einer Sicherheitsumgebung, der sog. **Sandbox**, ausgeführt. Zugriffe auf Resourcen außerhalb der Sandbox sollen abgeblockt werden. Applets, die des Anwender lokal auf dem Rechner speichert und von dort aus aufruft, unterliegen keinen Restriktionen. Sie können auch Operationen auf Dateiebene ausführen oder Netzwerkverbindungen zu beliebigen Systemen aufbauen.

Java-Quellcodes selbst werden mit einem Compiler in eine Art Maschienencode übersetzt, welcher dann auf den Clientmaschienen von der sog. *Java Virtual Machine* (JVM) ausgeführt wird. Die JVM prüft, ob verwendete Variablen initialisiert sind, ob bei Funktions- bzw. Methodenaufrufen die Anzahl und Typisierung der Parameter stimmt, die Instruktionen gültig sind und der Stack nicht überläuft. Der **Security-Manager** erzwingt die Einhaltung der Sandbox-Regeln. Jede Aktion, die über die Granzen der Sandbox hinaus Auswirkungen haben kann, muß vom Security-Manager erlaubt werden. Lehnt dieser die Zustimmung ab, wird eine Security-Exception ausgelöst und der Zugriff unterbunden.

Seit Java 1.1 können Applets signiert werden [Kop98]. Ab Java 1.1 werden die Applets in JAR-Dateien zusammengefaßt. Dieses Format ist eng an das ZIP-Archiv-Format angelehnt und faßt die einzelnen Java-Dateien zu einer einzelnen Datei zusammen. Dadurch reicht die Übertragung einer einzelnen Datei, um ein Java-Programm zu transferrieren. Es können Applets als auch JAR-Dateien signiert werden. Signierte Applets fragen durch den Browser beim Benutzer nach, ob der Zugriff gewährt werden soll. Bei Java 1.1 werden dem Applet bei Zustimmung alle Möglichkeiten des Zugriffs eingeräumt. Erst ab dem *Java Development Kit* JDK 1.2 können auch feinere Abstufungen in den Zugriffsrechten gesetzt werden. Beispielsweise kann mit dem `policytool` eine Sicherheits-Policy definiert werden, die genau die Rechte für Applets definiert.

Die zum Signieren von Applets verwendeteten Zertifikate sind X.509-Zertifikate, die i.A. self-signed sind, d.h., der Entwickler erzeugt und unterschreibt zunächst einmal sein eigenes Zertifikat. Darüber hinaus kann der Entwickler sein Zertifikat bei verschiedenen CA's beglaubigen

lassen.

Zur Zeit wird das Verfahren der signierten Applets noch recht wenig eingesetzt, da die verschiedenen Browser-Hersteller unterschiedliche Auffassungen über die Implementierung von Security-Managern etc. haben.

ActiveX und Authenticode

ActiveX ist ein Produkt der Firma Microsoft. Der Oberbegriff „ActiveX“ beinhaltet Technologien, Protokolle und API's (Application Programming Interface – Programmierschnittstelle), die dazu dienen, direkt ausführbaren Code aus dem Netzwerk zu laden. Ähnlich wie bei Java können verschiedene Programmteile in einer einzigen Datei gebündelt werden, die „ActiveX Control“ genannt wird und die Datei-Erweiterung `.OCX` trägt [GS97].

Bei ActiveX handelt es sich nicht um ein Konkurrenzprodukt zu Java, da der geladene Maschienencode plattformspezifisch ist. Ein ActiveX-Control kann aber in Java geschrieben sein. [GS97] vergleichen ActiveX eher mit einem automatisch geladenen und installierten Plug-In, welches einer speziellen Internet-Seite erweiterte Funktionalitäten bietet. ActiceX-Controls können Intel-x86-spezifischen Maschienencode enthalten, der aus C- oder C++-Compilern stammt, aber auch Visual Basic oder Java-Bytecode können im Control eingekapselt sein.

Das Sicherheitsmodell von ActiveX ist ausgesprochen unsicher: Der Maschienencode muß sich an keine Regeln halten, da er direkt auf der Hardware der Zielplattform ausgeführt wird, ohne daß ein Security-Manager oder Bytecode-Verifier wie bei Java die Instruktionen auf verbotene Aktionen untersuchen. Wenn das Control Java-Bytecode enthält, läd der Browser die Verantwortung für das Control bei der JVM ab, die den Java-Code prüft und ausführt.

Authenticode wurde von Microsoft entwickelt, um eine digitale Signatur von ActiveX-Controls zu ermöglichen. Mit Authenticode wird die digitale Signatur zusammen mit dem zugehörigen Zertifikat an den Programmcode gehängt und der Browser kann den Benutzer anhand der Identität des Entwicklers vor die Wahl stellen, ob das Control ausgeführt werden soll oder nicht. Gerade bei reinem Maschienencode kann nur eine Ja-Nein–Entscheidung über die weitere Vorgehendweise getroffen werden, da der Browser keine Code-Prüfungen durchführen kann.

Dem „Microsoft ActiveX Software Developers's Kit (SDK)“ liegen Tools wie `signcode` (der Code Signing Wizard) bei, mit denen schon bestehende Binärprogramme signiert werden können.

Abschließend ist zu Authenticode zu bemerken, daß hier extrem hersteller- bzw. plattformgebunden digitale Signaturen erzeugt werden.

2.1.2. W3C's „Digital Signature Initiative“ und PICS

Das World-Wide-Web-Consortium (W3C) ist ein Gremium, das sich weltweit um die Standardisierung von Protokollen und Diensten im World Wide Web bemüht. Das W3C hat die „Digital Signature Initiative“ (DSig) spezifiziert [Wor97]. Ziel der DSig war die Implementation eines

Signatur-Systems für das WWW mit folgenden Kriterien:

- Die Signatur sollte die zu signierenden Daten nicht einkapseln und in Bezug auf kryptografische Verfahren und verwendetete Zertifikate neutral sein, also keine Verfahren ausklammern.
- Das Format der Signatur und der Aussagen innerhalb der Signatur sollte standardisiert sein und eine beliebige Anzahl und Schachtelung von Statements zulassen.
- Es sollten soweit möglich bestehende Standards (PKCS-7, X.509v3, PICS, PGP, etc.) eingebaut werden.
- Identität des Dokumentenerzeugers und Integrität des Dokumentes sollten garantiert werden können.

In [CDLL97] wird genau spezifiziert, wie diese Ziele erreicht werden können. Nachfolgend nun ein kurzer Überblick über das System.

PICS - Platform for Internet Content Selection

DSig verwendet zur Repräsentation der Signaturen PICS-Label. PICS realisiert ein Rating-System, mit dem eine Klassifikation von Inhalten im WWW ermöglicht wird. In der Praxis ermöglicht PICS dem Client-Computer eine automatische Bewertung von WWW-Inhalten. An einem Beispiel läßt sich dieser Sachverhalt leicht verdeutlichen:

Der Betreiber eines Rechnersystems richtet in der Grundkonfiguration des Internet-Browsers eine Rating-Policy ein, d.h. er bestimmt, welche Inhalte der Web-Browser laden darf und welche nicht. Eine solche Policy könnte (umgangssprachlich formuliert) z.B. besagen:

> „Texte und Bilder, die der Klasse „Gewalt gegen Mensch und Tier“ angehören und deren Inhalte auf einer Skala zwischen 0 und 1 kleiner gleich 0.2 innerhalb dieser Klasse bewertet werden, dürfen dargestellt werden. Als Bewertungskriterium dienen die vom Dokumentenanbieter definierten Selbstbewertungen. Existieren keine solchen Eigenbewertungen, so führt der Rating-Service `http://christian-ratings.pics.vatikan.it` diese Bewerung in meinem Auftrag durch.“

Analog zur Klasse „Gewalt gegen Mensch und Tier“ läßt sich diese Klassifikation beliebig auf andere Themen anwenden. Ein Rating-Service kann nach eigenem Ermessen neue Klassen difinieren.

Wenn der Benutzer des Rechner-Systems nun „surft“, wertet der Browser die vom WWW-Server übermittelten Ratings aus. Existiert im angeforderten Dokument keine Ratings, übermittelt der Browser die URL der geforderten Internet-Seite an einen Rating-Service. Dieser Rating-Service

katalogisiert Inhalte im Web und führt Bewertungen durch. Solche Bewerungen kann der Rating-Service auch ohne Wissen oder Einverständnis der betroffenen Inhalte-Anbieter durchführen. So sind generelle Bewertungen ganzer Internet-Sites möglich. Kommt vom Rating-Service nun ein abschlägiger Bescheid oder existiert kein Rating für die geforderte URL, so stellt der Browser die geforderten Inhalte nicht dar. Erhält der Client einen positiven Bescheid, kann sie dargestellt werden.

PICS und DSig

DSig verwendet nun die Syntax und das Grundgerüst, welches PICS zur Verfügung stellt, um digitale Signaturen bereitzustellen. Daraus resultieren verschiedene Eigenschaften:

- DSig stellt ein durch die Verwendung von PICS gut erweiterbares Signatursystem für Web-Inhalte zur Verfügung.
- Aufgrund der Verwendung von PICS sind die zugrundeliegenden Mechanismen sehr aufwendig und erfordern komplexe Programme zur Analyse der Statements.
- PICS wird gesellschaftlich sehr kontrovers diskutiert, da es sehr geeignet ist, Zensur im Internet durchzuführen [Gru98]:

  > „Für PICS spricht die Durchsetzung des Jugendschutzes im Cyberspace, die im Gegensatz zu der gängigen und fehlerhaften Filtersoftware wie Cyber Patrol möglich gemacht wird. [...] Mit dem selbstbestimmten Maß an eigenzensur bliebe die Freiheit des Internet gewahrt, zumindest in den Augen der Erwachsenen, da sie ja auf PICS verzichten könnten. [...] Free-Speech-Initiativen, wie die EFF und Netfreedom, befürchten, daß dann auch Internetseiten politischer Extremisten oder Oppositioneller gesperrt werden könnten. Schließlich wäre es für ein mit Regierungsmitgliedern besetztes Kontrollgremium ein leichtes, willkürlich unliebsame Inhalte zu indizieren und durch Proxy-Server bereits im Vorfeld sperren zu lassen. [...] Heute kann der Internetsurfer noch selbst bestimmen, ob er ein Rating-System in Anspruch nehmen möchte oder nicht. Populäre Suchmaschinen wie Lycos oder Yahoo kündigten aber an, bald nur noch PICS-registrierte Angebote zu verzeichnen. Damit wären Webmaster, die ihre Seiten einer möglichst großen Zahl von Surfern zugänglich machen wollen, gezwungen, die Inhalte schon aus reinem Selbsterhaltungstrieb PICS-kompatibel und möglichst Jugendfrei zu gestalten. Die Gegner warnen immer dringlicher vor den Gefahren dieser neuen Technologie. Der Internet-Expert SIMSON GARFINKEL nennt PICS gar „die effektivste globale Zensurtechnik aller Zeiten“. Da die Zensur nicht mehr von der Staatsgewalt ausgeht, erscheine es als natürlicher Vorgang, wenn bestimmte Internetinhalte verschwinden.“

PICS ist also eine ausgesprochen umstrittene Plattform. Es bleibt abzuwarten, ob PICS von der Internet-Community angenommen wird und in vielen Produkten implementiert wird.

- Um Daten, eine Signatur und evtl. Zertifikate etc. zu transportieren, sind viele HTTP-Operationen notwendig. Von der Übertragungszeit her ist das System relativ unperformant.

Was die grundlegenden Ziele von DSig angeht, ist zu sagen, daß diese Diplom-Arbeit die gleichen Ziele verfolgt. Aufgrund der unklaren Zukunft von PICS wurde in dieser Arbeit ein anderer Weg gewählt.

2.2. Universelle Signatur-Systeme

2.2.1. PGP

Pretty Good Privacy (PGP) ist ein System zum Verschlüsseln und Signieren von Daten. PGP wird hauptsächlich zur Verschlüsselung von E-mail eingesetzt.

Es wurde im Juni 1991 (v1.0) von PHILIP ZIMMERMANN veröffentlicht. Seit Version 2.3a unterstützt PGP die Algorithmen RSA, IDEA und MD5 und bot den Nutzern erstmals öffentlich und frei verfügbar starke Kryptografie. Zur Zeit ist Version 5.5 aktuell, bei der neben RSA auch Diffie-Hellman-Key-Exchange (DH) und der „Digital Signature Standard“ (DSS) und verschiedene Blockchiffren verfügbar sind.

PGP ist das am weitesten verbreitete e-mail-Verschlüsselungsprogramm im Internet. Der Erfolg von PGP liegt in der Einfachheit der Verwendbarkeit und in der Vielfalt der PKI (Public Key Infrastructure) begründet. Anders als bei streng hierarchisch strukturierten PKI's kann jeder Nutzer die öffentlichen Schlüssel anderer Nutzer signieren. Eine Root-CA ist nicht zwingend notwendig.

Der Schlüssel, den ein Nutzer selbst erzeugt, wird vom Nutzer selbst unterschrieben; dadurch entsteht ein self-signed-Certificate. Wenn man die öffentlichen Schlüssel von Bekannten bekommt, können diese signiert und die Unterschrift an die jeweiligen Personen zurückgeschickt werden, die diese dann in den eigenen Schlüsselbund einlesen. Dadurch sammelt man Unterschriften anderer Nutzer für den eigenen Schlüssel. Das sukzessive Wachsen der Schlüsseldatenbank erzeugt ein „Web of Trust“. Erhält ein Benutzer den öffentlichen Schlüssel eines Unbekannten, ist unklar, ob der Schlüssel wirklich echt ist oder einer anderen Person als angegeben gehört. Findet man unter dem erhaltenen Schlüssel u.U. die Unterschrift eines Bekannten , dem man traut, wird die Echtheit des neuen Schlüssels recht wahrscheinlich. Das ist natürlich abhängig vom Vertrauen, das man der Person entgegenbringt, die für den neuen Schlüssel unterschrieben hat.

Durch diesen dezentralen Ansatz ist das System sehr geeignet, im Privatbereich sicher zu kommunizieren. Entweder wird das System nur für private Kommunikation unter Freunden verwen-

det; dann ist die Prüfung von Schlüsseln unkompliziert, da man die Schlüssel auch persönlich übergeben kann. Der andere Fall ist der, daß man sich den Schlüssel der anderen Person aus einem der zahlreichen Key-Server im Internet holt. Abhängig vom Inhalt der Nachricht ist es u.U. garnicht so wichtig, daß der Schlüssel 100%-tig zu der Person gehört; dann besteht natürlich die Gefahr, daß die Nachricht in falsche Hände gelangt. Um diese Problematik zu umgehen, kann man auf einem dritten Weg Prüfsummen (Fingerprints) der Schlüssel austauschen, um die Wahrscheinlichkeit für die Echtheit eines Schlüssels zu steigern.

In letzter Zeit bieten immer mehr CA's und Organisationen die Möglichkeit, PGP-Schlüssel zertifizieren zu lassen. So bieten z.B. der Heise-Verlag, das „Individual Network (IN) Deutschland“ und auch das Deutsche Forschungsnetz (DFN)“ eigene Zertifizierungsdienste an. Von verschiedenen CA's wird dieser Service für die private Nutzung kostenlos übernommen, wobei die Prüfung der Identität von CA zu CA unterschiedlich streng gehandhabt wird. Dieser Service wird in Deutschland z.B. von „TC Trust Center“ angeboten. Durch diese PGP-(Root)-CA's wird der dezentrale Ansatz ein klein wenig zentralisiert, um die Echtheit von Schlüsseln besser gewährleisten zu können.

PGP unterstützt RSA-Schlüssel von bis 2048 bit Modulus-Länge und Diffie-Hellman-Schlüssel von bis zu 4096 bit. Als Hash-Funktion dient MD5.

Eine PGP-signierte Nachricht könnte z.B. so aussehen:

```
-----BEGIN PGP SIGNED MESSAGE-----

Hallo,

Dies ist eine signierte PGP-Nachricht.

Gruß, Christian

-----BEGIN PGP SIGNATURE-----
Version: 2.6.3i
Charset: latin1

iQEVAwUBNbWIUU/ylwIE5/aVAQHUVgf+O+CDd38lBs5ugqobKP2KKA6cviqWjSRL
XMDiSYbqyjvO8T+zmfmqtNL/LtZ1kRH60GCQbeXUC5irM2RY5k4sS0Yc/3iJxmuu
3vfo7PFfuSq/9EGY5Rr1CmMqLUKvc/TxHs+nCcrXSAc7B1/d+AxANhGzrAF2uIYh
ZBZfupNciH4Xvpb/8NGF15/+4zs4Q1ysfof5Mj7XKvIPU664O0jBZyYxUx9ARE1k
OxOBLqnBnf0X3hysRZl0aslrwyvHejhWeXwcmGSXOkk8BEitjrhTvb7EkKDaP6LA
QHXS+Zjx0HPEYPajx111xMDJU5B7h7eoBsGkTTVUMs2fNxTjs1ey3Q==
=bmDJ
-----END PGP SIGNATURE-----
```

2.2.2. MailTrusT

MailTrusT (MTT) Version 1.1 wurde im Dezember 1996 spezifiziert. Herausgeber der Spezifikation ist „TeleTrusT: Verein zur Förderung der Vertrauenswürdigkeit von Informations- und Kommunikationstechnik“; ein Zusammenschluß verschiedener, namhafter Firmen und Orga-

nisationen, die mit dieser Spezifikation eine „nicht-proprietäre, herstellerübergreifende, interoperable und sichere Komponente für den elektronischen Dokumentenaustausch“ bereitstellen möchten.

[Bau96]:

> Die MailTrusT-Spezifikation baut überwiegend auf bereits etablierten Standards und Spezifikationen auf, wie PEM ([Lin93, Ken93, Bal93, Kal93]), X.509 ([Int93]) und PKCS#11([RSA95]) auf.

PEM (Privacy Enhanced Mail) ist ein Internet-Standard für Verschlüsselung, der in den RFC's 1421–1424 definiert ist. Der große Unterschied zwischen PEM und PGP liegt in der PKI. Bei PEM und MailTrust ist ein streng hierarchischer Ansatz vorgesehen, d.h. die PKI hat eine Baumstruktur:

1. Die TLCA (Top-Level-Certificate-Authority) bildet den Ursprung der Zertifizierungshierarchie und fungiert als Root-CA. Der Schlüssel der TLCA wird allgemein verfügbar gemacht (Mit jedem Programm ausgeliefert und überall via Internet verfügbar gemacht) und dient als Basis für die Schlüsselverifizierung. Durch die TLCA können Schlüssel zwischen verschiedenen Teilbäumen der PKI interoperabel ausgetauscht werden.

   Die TCLA zertifiziert die PCA's.

2. Die PCA's (Policy-CA) bilden die zweite Ebene der PKI. Eine Policy-CA stellt eine Security-Policy auf, die das Verhalten und des Aussage-Inhalt von Zertifikaten des jeweiligen Teilbaums festlegt. Die PCA organisiert und kontrolliert den Betrieb des ihr untergeordneten Teilbaums.

3. Die CA's sind das vorletzte Glied in der PKI. Die CA's werden von den PCA's oder evtl. noch zwischengeschalteten Ebenen zertifiziert. Die CA's stellen die Zertifikate für die eigentlichen Teilnehmer der PKI aus.

4. Die Teilnehmer (TN) stellen die unterste Ebene der PKI dar. Ein Teilnehmerzertifikat besteht aus dem Zertifikat, das die CA dem TN ausgestellt hat. Darüber hinaus sollten alle darüberliegenden Zertifikate mit eingebunden sein, um immer den kompletten Pfad bis zur TLCA prüfen zu können.

MTT stellt *keine* Anforderungen an die Tiefe einer PKI. Es muß aber nun von System zu System ein Tradeoff zwischen Prüfungsaufwand für ein spezielles Zertifikat und der Last der einzelnen CA's gemacht werden. Flache Hierarchien erlauben ein einfaches und schnelles Prüfen einzelner Zertifikate. Dadurch müssen aber die einzelnen CA's viele TN's zertifizieren und die TLCA muß viele CA's zertifizieren (wenn man die PCA's wegläßt).

MTT-Zertifikate sind nach dem X.509-Standard Version 1 (X.509v1) zusammengebaut. Der Nachrichtenaufbau (das Format) ist gemäß PEM, wobei auch neue Algorithmen spezifiziert

sind, die nicht mehr in PEM betrachtet wurden. Daher ist das *Format* zwar PEM-Kompatibel, die einzelnen Nachrichten können aber nicht von PEM decodier werden, da die Algorithmen teils unbekannt sind. MailTrusT nutzt PKCS#11 als Schnittstelle zum Speichern von privaten Schlüsseln; die sog. Cryptoki-Schnittstelle (PKCS#11) definiert eine einheitliche Möglichkeit, wie geheime Schlüssel beispielsweise auf SmardCards (Chipkarten) sicher gespeichert werden können. Im Zusammenhand mit MTT stellt das „Personal Security Environment“ (PSE) eine Möglichkeit zum sicheren Aufbewahren und Verwenden privater und geheimer Schlüssel dar. So kann ein PSE-Token eine SmartCard sein, aber auch die Speicherung auf Disketten ist (noch) möglich. Da im „Gesetz zur digitalen Signatur“ sichere Möglichkeiten gefordert werden, um private Schlüssel zu speichern, wird sich auf diesem Gebiet in den nächsten Jahren noch recht viel tun.

Eine MailTrusT-Nachricht könnte z.B. so aussehen:

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Proc-Type: MTT-1,MIC-BIN
Content-Domain: Binary,gzip,signatur.txt,
Originator-Certificate:
 MIIB9zCCAWACADANBgkqhkiG9w0BAQQFADBIMQswCQYDVQQGEwJkZTEcMBoGA1UE
 ChMTVW5pdmVyc2l0YWV0IFNpZWdlbjEbMBkGA1UEAxMSQ2hyaXN0aWFuIEguIEdl
 dWVyMBoXCzk4MDEwMTEyMDBaFwswMDAxMDExMjAwWjBIMQswCQYDVQQGEwJkZTEc
 MBoGA1UEChMTVW5pdmVyc2l0YWV0IFNpZWdlbjEbMBkGA1UEAxMSQ2hyaXN0aWFu
 IEguIEdldWVyMIGcMAoGBFUIAQECAgQAA4GNADCBiQKBgQDRxduJ51QJXbjyZCEp
 UWFp7KKSkoec/DRdDTCthwIbJ1WRibtOoc/VxIp6rzGUPIdHnflgdBmjaG/tVXAs
 iJFW49wdJhNN6DgC1+KeQxbCyiNrIdB/vzYB13khRUhb/cDQPgLr+JiOqpIQodA+
 ATZAP8lBn+ZtL465khon+KRf3QIDAQABMA0GCSqGSIb3DQEBBAUAA4GBABP2w4f0
 sUB8nPy7db31lhGZcuNIdescoBuOhjHJPQQ6qaZGGEgM2+LSGapnLmNZyuWBdARX
 p0qgvtVDtrttb4CG1LbSGZMS4dcRJ9CR1O4miBletJBaQ3eyrR7TTaJm0k7sE7Fl
 UWO6Xt4z4qkACg2Qj+bQVnzfFPpN+74hvr0Y
MIC-Info: SHA-1,RSA,
 uWV4d50ubcEkD4Il8t0lD1hYzFo9nyfvFptbl2fOzFPPYnmIriXrs85qymJnBxqp
 aFDbEL1L+H7AJ6iq4tc3CwLEp0gE5f210TGNwRxMFBHy1lh+RLscfB0ArvLBiorL
 MFko1IwYdk+SHv3ldXRPAHE6XRq1tQhhVACtGmZXa7s=

eJxdzsFOwzAQBNB7vmI+oKm8dtzYeyuEFhBBqGrhbDkbGilNpOBKfD5uj0hzfDOa
0+GNMUyd/K7P6VI8zlOSKZXP4efM0CZECrFCMHUdYoyonfUm9g620paUFTizcdFW
rrhVyu34PTMOLx9PbUMbVTQhCeNLuhW0xut1BHnvQBVbxUTYt8fiM4xDV77PqXyQ
fl6yP14le0IbluxrDaX4nn9+2ydZGLtlWMEQGom3eZ9/s/U5d/4HLIE9Qw==
-----END PRIVACY-ENHANCED MESSAGE-----
```

Gut zu erkennen sind hier das `Originator-Certificate`-Feld, in dem das Zertifikat des Signierers festgelegt wird sowie die Signatur (`MIC-Info: SHA-1,RSA,`) und die eigentlich signierten Daten (die vorletzten vier Zeilen).

3. Spezifikation und Design des Systems

Das folgende Kapitel beschreibt die grundlegenden Ideen dieser Arbeit und zeigt Wege auf, wie diese in bestehende Strukturen integriert werden können.

Nachdem die Anforderungen an das System erläutert werden, wird die zu etablierende PKI beschrieben. Daran anschließend wird erklärt, wie das Signaturmodell funktioniert und wie die Signaturen effizient im Netz übertragen werden können.

3.1. Anforderungen an das System

Ziel dieser Arbeit ist die Erarbeitung eines Systems, welches es ermöglicht, sichere Signaturen für beliebige Inhalte im Internet zu generieren. Es sollte egal sein, ob HTML-Dokumente, Bilddateien, Ton-Dateien oder Zufallsdaten signiert werden.

- Für den Server sollte der zusätzliche Aufwand möglichst minimal sein. Für den Client sollte der Übertragungsaufwand ebenfalls minimal sein.
- Diese Signaturen sollen möglichst einfach und übertragungssicher durch das Internet übertragen werden können, d.h. daß die Übertragung nicht durch Proxies jedweder Art behindert werden sollte.
- Der Client, d.h. der Internetsurfer, soll von der zusätzlichen Übertragung der Signaturen möglichst wenig bemerken, d.h. das System soll selbstständig erkennen, ob Signaturen vorhanden sind, diese übertragen und auswerten.
- Die Identität des Signierers sollte möglichst einfach und interoperabel überprüft werden können.
- Das System sollte unterschiedliche Signatur-Typen unterstützen. Auf dem Server können mehrere Signaturen für ein und den selben URI abgelegt sein. So können beispielsweise eine MailTrusT–Signatur und eine PGP–Signatur zusammen existieren.
- Der Client sollte frei wählen können, welchen Signatur-Typ er bevorzugt. Der Server versucht, den Wünschen des Clients nachzukommen und die gewünschten Signatur-Schemata zu liefern.

3.2. Public Key Infrastructure (PKI)

Der für diese Arbeit vorgeschlagene Entwurf einer Zertifizierungshierarchie (Public Key Infrastructure PKI) ist zweistufig:

- Es existiert eine Root-CA. Der Schlüssel (das Zertifikat) dieser Root-CA ist allen Teilnehmern des Netzes bekannt. Das wird z.B. ermöglicht, indem der Schlüssel in der Installationsversion des Client-Programmes mitgeliefert wird. Die Root-CA ist authentisch und vertrauenswürdig.

  Die Root-CA stellt Zertifikate für einzelne Internet-Hosts aus, d.h., jeder Web-Server bekommt von der Root-CA ein Zertifikat ausgestellt. Damit ist die Authenzität eines Server-Zertifikates gewährleistet. So garantiert die Root-CA beispielsweise für die Identität `http://www.uni-wuppertal.de/`.

- Das von der Root-CA für einen speziellen Web-Server ausgestellte Zertifikat erlaubt es diesem Web-Server, selbst wiederum Zertifikate für die Teilnehmer, d.h., die Signierer auszustellen. Damit die Last des Zertifikate-ausstellens für die Root-CA nicht zu groß wird, d.h. damit nicht jeder Teilnehmer zur Root-CA gehen muß, um sein Zertifikat zu erhalten, wird diese Aufgabe an die Web-Server-Betreiber delegiert. Der Web-Server-Betreiber kann die Identität der Teilnehmer, die auf seinem Server Inhalte anbieten, relativ einfach prüfen.

- Der Teilnehmer (Signierer) bekommt vom Web-Server-Betreiber ein Zertifikat ausgestellt, das es ihm erlaubt, eigene Inhalte zu signieren, die auf dem Web-Server des Betreibers ins Internet gespeist werden. Ein Zertifikat lautet also auf den Teilnehmer in Zusammenhang mit dem speziellen Server.

Der Vorteil einer zweistufigen PKI ist nicht in der Sicherheit zu suchen, sondern in der einfacheren Durchführbarkeit der Zertifikatsausstellung.

3.3. Die Signatur: Integraler Bestandteil der Daten oder abgetrennt?

Eine wichtige Frage, die geklärt werden mußte, war die Art der Signatur. Sollten Daten und Signatur als eine einzige Datei auf dem Server existieren oder als separate Komponenten:

- Das Problem, wenn die **Signatur in den Daten** enthalten ist liegt in der Notwendigkeit, für jedes Datenformat genau zu definieren, welche Teile des Objekts Nutzdaten sind und welche zur Signatur gehören. Eine solche Integration ist nur möglich, wenn man aus einem Objekt die Nutzdaten eindeutig gewinnen kann (um den Hash-Wert zu berechnen) und das

Datenformat darüber hinaus die Möglichkeit bietet, beliebig große Kommentare wie z.B. Signaturen einzukapseln.

Da diese Eigenschaften von fast keinem Datenformat erfüllt wird, wurde dieser Ansatz für diese Arbeit verworfen. DSig basiert auf der Möglichkeit, daß in HTML, der Seitenbeschreibungssprache des Internet, Kommentare und Meta-Tags eingekapselt werden können, wie z.B. PICS-Label. Andere Formate wie GIF- oder JPEG-Bilder udn Tondateien bieten diese Möglichkeit nicht.

- Wenn die **Nutzdaten Teil der Signatur** werden, wie das bei PGP oder MailTrusT der Fall sein kann, können die Nutzdaten und die Signatur in einem Rutsch übertragen werden und bilden auch ein autonomes, integeres Objekt. Das Problem liegt in der Spezialisierung auf ein Signatur-Format und in der Kompatibilität zu HTTP. Wenn ein Client eine URL wie z.B. `http://www.uni-wuppertal.de/index.html` anfordert, erwartet er ein reines HTML-Dokument, d.h. ein Objekt mit dem MIME-Typ `text/html`. Wenn die Daten aber in die Signatur eingekapselt sind, muß dem Client erst einmal klar gemacht werden, daß die Daten vor der Interpretation zu „entpacken“ sind.

  Hier ist also ein Problem mit der Rückwärtskompatibilität zu erwarten.

- Der hier gewählte Ansatz ist die **getrennte Speicherung** der Signatur in einer separaten Datei.

  Im Internet existieren eine Vielzahl von unterschiedlichen Servern und Clients. Auf Server-Seite reicht die Palette von kommerziellen Produkten wie dem Netscape–Enterprise-Server oder dem Microsoft–Information-Server bis zu frei verfügbaren Servern wie dem Apache–Server. Auf der Client-Seite existieren Internet-Browser wie der Netscape-Communicator, der Microsoft-Internet-Explorer oder freie Implementationen wie Arena, Mosaic, Lynx, Emacs u.v.m. Darüber hinaus agieren auch automatische Clients wie Internet-Suchmaschinen oder Mirror-Programme, die ganze Server-Strukturen kopieren.

  Um allen diesen Clients die Möglichkeit zu lassen, trotz digitalen Signaturen wie gewohnt mit dem Server zu kommunizieren, war nur eine separate Speicherung von Daten und Signatur möglich. Durch diesen Ansatz ist den normalen (nicht Signatur-fähigen) Clients egal, ob parallel zu den Daten noch eine Signatur verfügbar ist. Eine Signatur stellt sich für die Clients wie eine normale Datei auf dem Server dar.

  Wenn nun ein Signatur-fähiger Client eine URL fordert, erkennt ein Signatur-fähiger Server die Anfrage und versucht die gewünschten Daten und Signaturen zu liefern. Fordert ein Signatur-fähiger Client Daten und eine implizite Signatur von einem nicht-Signatur-fähigen Server, erkennt der Client das Unvermögen des Servers und kann dann explizit nach den gewünschten Signaturen fragen.

  Durch diesen Ansatz wird ermöglicht, daß alte und neue Clients sowie alte und neue Server in jeder Konstellation miteinander kommunizieren können.

3.4. Das „Simple Signature Protocol“ (SSP)

3.4.1. Einleitung

Viele Signaturprogramme (z.B. PGP) können abgetrennte Signaturen erzeugen. Eine solche Signatur hat eine relativ konstante Größe (ca. 500 Bytes ohne Zertifikate) und beinhaltet den mit dem privaten Schlüssel des Signierers signierten Hash-Wert des Dokumentes. In der Signatur ist darüber hinaus noch der Zeitpunkt der Signatur festgehalten. Außer der Information „*Wer hat signiert?*“ und „*Wann hat er signiert?*“ lassen sich in diesen einfachen Schemata keine weiteren Informationen unterbringen.

Damit diese Einschränkungen umgangen werden können, wird hier das „**Simple Signature Protocol**“ definiert. Das SSP realisiert eine zweistufige Signatur. Anstatt die eigentlichen Daten direkt zu signieren, wie das bei universell einsetztbaren Verschlüsselungsprogrammen der Fall ist, wird ein Zwischenobjekt erzeugt: die SSP-Datei. Im SSP sind die kennzeichnenden Parameter der Signatur festgeschrieben; Neben dem URL, an dem die zu signierenden Daten abgelegt werden, ist hier z.B. der Hash-Wert der eigentlichen Datei enthalten. Das SSP-Objekt kann dann in die Signatur des e-mail-Verschlüsselungsprogrammes eingekapselt werden.

Bevor die einzelnen Felder genauer beschrieben werden, soll an dieser Stelle ein kleines Beispiel zur Verdeutlichung dienen:

```
URL: http://crypto.gun.de/index.html
Content-Hash: bb925a11 4e28179e e57c3378 af27ebc2 f83574fa
Hash-Algo: RIPEMD160
Date: Wed, 22 Jul 1998 17:19:55 GMT
Valid-Not-Before: Thu, 01 Jan 1998 00:00:00 GMT
Valid-Not-After: Thu, 31 Dec 1998 23:59:59 GMT
Signer-Email: christian.geuer@crypto.gun.de
```

Neben dem URL, an der das Objekt im WWW angerufen werden kann, muß natürlich die kryptografisch sichere Hashsumme vorhanden sein. Der verwendete Hashalgirithmus ist RIPE MD-160. Die Signatur wurde am 22. Juli 1998 erzeugt und gilt das ganze Jahr über. Der Signierer ist unter der angegebenen E-Mail-Adresse erreichbar.

Dieser Text (das SSP-Objekt) wird nun mit einem universellen Verschlüsselungsprogramm signiert.

3.4.2. Interpretation der Felder

Die Felder sind wie in RFC822 aufgebaut, d.h., der Feldname und der Feldinhalt sind durch einen Doppelpunkt getrennt; der Feldname selbst enthält keinen Doppelpunkt.

URL

Der URL (Uniform Resource Locator) gemäß RFC1738 bzw. RFC1808 beschreibt die Stelle im Internet, an der die Daten zu finden sind. Ist dieses Feld gesetzt, so ist die Signatur nur gültig, wenn die Daten auch von dieser Stelle geladen wurden. Dieses Feld muß vom Browser interpretiert werden. Damit kann verhindert werden, daß Dritte Inhalte und Signaturen kopieren, an anderen Stellen im Netz ablegen und die Signaturen trotzdem ihre Gültigkeit behalten. Das Kopieren an sich kann zwar nicht verhindert werden; die Anerkennung durch den Client erfolgt aber nicht.

Ist das Feld leer, so können die Daten an beliebigen Stellen abgelegt werden, ohne ihre Gültigkeit zu verlieren.

Content-Hash

Damit die Daten selbst nicht verändert werden können, wird das Resultat einer kryptografisch sicheren Hash-Funktion mit in der Signatur abgelegt. Dieses Feld **muß** vorhanden sein.

Hash-Algo

Damit der Content-Hash vom Client verifiziert werden kann, muß der verwendete Hash-Algorithmus bestimmt werden. Dieses Feld **muß** vorhanden sein. Gültige Werte sind: MD5, SHA1 und RIPEMD160.

Valid-Not-Before und Valid-Not-After

Die Felder Valid-Not-Before und Valid-Not-After enthalten Gültigkeitszeiträume für die Signatur. Im Feld Valid-Not-Before kann der Startzeitpunkt festgelegt werden. Valid-Not-After kennzeichnet den Zeitpunkt, nach dem die Gültigkeit der Signatur erlischt. Mit diesen beiden Feldern können beispielsweise Preisabgebote zeitlich beschränkt werden.

Sowohl eines als auch beide Felder können weggelassen werden. Fehlt eines der Felder, erweitert sich das Gültigkeitsinterval in die jeweilige Richtung. Fehlen beide Felder, kann keine Aussage über den Gültigkeitszeitraum getroffen werden.

Das Datum wird gemäß RFC2068 dargestellt.

Date

Das Datum gemäß RFC2068, an dem die Signatur erzeugt wurde. Dieses Feld **muß** enthalten sein.

`Signer-Email`

In diesem Feld kann die e-mail-Adresse des Signierers gemäß RFC822 festgehalten werden.

`Reference-Allowed`

Im Web übertragene Inhalte verbrauchen nicht nur Übertragungsbandbreite, sondern auch Resourcen auf dem Server. Besonders groß wird der Leistungsverbrauch bei großen Dateien wie Bildern oder ähnlichem. Einige Anbieter versuchen ihre Serverbelastung dadurch zu minimieren, daß sie große Bilddateien nicht auf dem eigenen Server anbieten, sondern in den Hypertexten einfach Querverweise auf Bilder etc. auf anderen Web-Servern machen. Dadurch sparen sie Speicherplatz, Rechenleistung und Übertragungskapazität. Wieder andere „schwarze Schafe“ erzeugen Hyperlinks auf URL's, an denen sie keine Rechte besitzen. So werden Artikel und Texte mit Zeitungsbildern aufgepeppt, wobei die Bilder auf dem Server der Zeitung liegen und nicht auf dem des Text-Anbieters.

Um dieses Problem zu lösen, wird der Header `Reference-Allowed` eingeführt. Dieses Flag gibt an, ob andere Server Links auf diesen URL verweisen dürfen. Es dient dem Client als Entscheidungshilfe, ob z.B. die Signatur einer eingebundenen Grafik überhaupt gültig ist, wenn die Grafik von einem anderen Host eingebunden wurde. Dadurch kann eine signierte Grafik nicht in einem falschen Kontext mißbraucht werden.

Ist das Flag nicht gesetzt, wird implizit das Verbot von Referenzierungen angenommen. (Default: `no`)

`Comment`

Um auf einfache Weise Kommentare in einer Signatur unterzubringen, wird der `Comment`-Header spezifiziert. Es können beliebig viele Kommentarzeilen in der Signatur untergebracht werden.

3.4.3. Erweiterungen

Analog zu den vorangegangenen Headern sind noch viele andere Header-Definitionen denkbar. Durch den universellen Aufbau können im Nachhinein noch neuen Typen definiert werden.

3.5. HTTP und die Übertragung der Signatur

3.5.1. Kompatibilität und Kapselung

Die HTTP-Header bieten sich an, die Signatur mit übertragen zu können. Als Teil des HTTP-Protokolls können beliebige Informationen von Server zum Client „tunneln“. Um diese Kap-

selung zu ermöglichen, werden im folgenden vier neue HTTP-Header spezifiziert, die den Request bzw. den Response der Signatur aufnehmen können. Damit es zu keinen Konflikten mit den HTTP/1.x-Spezifikationen [FFB96, FGM+97] kommt, wurden die neuen Header mit einem speziellen Anfang versehen: Die ersten beiden Bytes der neuen Header lauten: „X-“. Dieser Anfang wurde schon in RFC822 für Email-Nachrichten als Auffangbecken spezieller Erweiterungen definiert. Ein Server oder Client kann X-Header einfach ignorieren, ohne daß die Funktion des Protokolls dadurch eingeschränkt sein darf.

Durch diese Integration ins HTTP-Protokoll ist gewährleistet, daß alle Proxy-Server auf dem Weg vom Client zum Server und zurück die Information durchlassen. Andererseits können auch nicht-Signatur-fähige Clients und Server derartige Header einfach ignorieren, ohne daß die HTTP-Grundfunktionalität dadurch gestört wird.

3.5.2. Spezifikation der neuen HTTP-Header

X-Signature-Request

Der Client formuliert seinen Wunsch für eine digitale Signatur durch den X-Signature-Request-Header. Dieser Header dient der Aufnahme von verschiedenen Clientseitig verarbeitbaren Signatur-Schemata. So sagt beispielsweise der Request

```
GET /index.html HTTP/1.0
Host: www.uni-wuppertal.de
X-Signature-Request: Mtt-V1 Pgp-Rsa
```

aus, daß der Client die Datei mit der URL http://www.uni-wuppertal.de/index.html anfordert und, wenn möglich, eine MailTrusT v1 konforme Signatur haben möchte. Falls das nicht möglich ist, ist auch eine PGP-Signatur mit RSA-Schlüsseln möglich.

Durch die Reihenfolge der angegebenen Signaturschemata kann der Client eine Priorisierung vornehmen, nach der der Server bei der Auswahl möglicher Schemata vorgehen kann.

X-Signature-Types-Avail

Der Server erkennt am Vorhandensein eines X-Signature-Request-Headers die Fähigkeit des Client, Signaturen verarbeiten zu können. Der Server versucht, für den geforderten URL Signatur-Informationen zu erhalten. Wird er fündig, liefert er im X-Signature-Types-Avail-Header die verfügbaren Signaturen zurück. Hirbei handelt es sich um eine unsortierte Liste der für den geforderten URL verfügbaren Signatur-Schemata. Ist der Server Signatur-fähig und existieren keine Signaturen für den geforderten URL, bleibt die Liste leer.

Das folgende Beispiel sagt beispielsweise, daß PGP-Signaturen mit Diffie-Hellman- und RSA-Schlüsseln sowie eine MailTrusT-Signatur verfügbar sind.

```
X-Signature-Types-Avail: Pgp-Dh Pgp-Rsa Mtt-V1
```

X-Signature-Error

Über den X-Signature-Error-Header kann der Server Fehlerzustände innerhalb des Signatur-Protokolls zurückliefern. Diese Klartext-Nachrichten können dem Benutzer durchgereicht werden.

X-Signature-Response-SCHEME-xx

Die eigentliche Übertragung der Signatur erfolgt im X-Signature-Response-Header. Der Header ist nicht genauer spezifiziert, da er je nach Signatur-Schema anders aussehen kann. Der Teil SCHEME wird durch das gelieferte Signatur-Schema ersetzt, der Teil xx ist eine Laufnummer, in der die Zeile durchnummeriert wird. Die erste Laufnummer ist die 0.

Der Wert des jeweiligen Feldes ist Base64-codiert ein Teil der Signatur. Um die ursprüngliche Signatur-Datei zurückzuerhalten, werden die Feld-Werte in aufsteigender Reihenfolge aneinander gehängt und die Base64-Codierung rückgängig gemacht.

Wäre auf dem Server eine Signatur des Schema Plain abgelegt und der Inhalt der Nachricht wäre: "Hallo. Das ist eine Nachricht, die gleich Base64-Codiert wird.", so antwortet der Server mit:

```
X-Signature-Response-Plain-0: SGFsbG8uIERhcyBpc3QgZWluZSBOYWNocmljaHQsIGRpZSBnbGVpY2ggQmFzZT...
X-Signature-Response-Plain-1: d2lyZC4=
```

3.5.3. Spezifikation der neuen MIME-Typen

application/signature-content-sig-SCHEME

Neben der Möglichkeit, eine Signatur implizit, also in den HTTP-Headern transportieren zu lassen, kann die Signatur-Datei direkt angefordert werden. Damit serverseitig auch ein korrekter MIME-Typ gesetzt werden kann, wird der MIME-Typ application/signature-content-sig-SCHEME eingeführt, wobei der Teil SCHEME wieder gegen das jeweilige Schema zu ersetzen ist.

Durch die Möglichkeit, Signaturen explizit zu erfragen, können Signatur-fähige Clients bei nicht-Signatur-fähigen Servern Signaturen abholen. Durch das Fehlen des X-Signature-Types-Avail-Headers erkennt der Client, daß er selbst versuchen muß, die möglichen Signaturen explizit zu laden.

3.6. Entwicklungsumgebung

3.6.1. Implementationsplattform: Unix

Implementationsplattform dieser Diplomarbeit ist Unix, genauer Linux. Linux ist eine frei verfügbare Implementation des Betriebssystems Unix. Unix bzw. Linux hat den Vorteil, daß

ein echtes präemptives Multitasking möglich ist, d.h. daß sich die gestarteten Prozesse i.A. nicht untereinander koordinieren müssen, wem wann eine Zeitscheibe zugeteilt wird.

Ferner findet man für Linux viele freie Werkzeuge, Tools und Bibliotheken, die die Entwicklungsarbeit sehr erleichtern.

3.6.2. Implementationssprache: Perl

Als Haupt-Implementationssprache für diese Diplomarbeit wurde Perl gewählt. Perl-Programme werden mit Hilfe des Programmes `perl` nach dem Start in einen internen Byte-Code compiliert und dann annähernd mit der Geschwindigkeit eines C-Programmes ausgeführt. Perl unterstützt Netzwerkprogrammierung, Interprozess-Kommunikation (IPC), Objektorientiertes Programmierung (OO), reguläre Ausdrücke (RegEx) und es existieren viele vorgefertigte Module, die einfach in eigene Programme eingebunden werden können.

Der Vorteil von Perl gegenüber Sprachen wie C ist das „Rapid Prototyping“, d.h. die Möglichkeit, nach kurzer Zeit einen funktionierenden Prototypen präsentieren zu können. Perl nimmt dem Programmierer viele lästige Aufgaben wie z.B. Speichermanagement ab; wenn einem Datentyp mehr Daten zugewiesen werden, als ursprünglich erwartet, wächst die Variable einfach mit. Daneben existieren Konzepte wie das „Tainting“, d.h. das Definieren verseuchter Variablen. Viele Probleme in bestehenden Applikationen resultieren aus der Problematik, daß Benutzereingaben als Input für Systemaufrufe verwendet werden. Bei Perl sorgt das Taint-Checking dafür, daß von außerhalb des Programms kommende Daten nicht einfach so verwendet werden können, um etwas Anderes außerhalb zu beeinflussen. Alle Kommandozeilenargumente, Dateieingaben oder Umgebungsvariablen werden als unsicher markiert; diese Daten dürfen dann weder direkt noch indirekt verwendet werden, um beispielsweise eine Sub-Shell zu starten oder Dateien, Verzeichnisse und Prozesse zu modifizieren.

Durch diese Features ist Perl beispielsweise für sichere Netzwerk-Server und Clients geeignet, da die häufigen Probleme wie ungeprüfte Eingaben oder Buffer-Überläufe keine Wirkung zeigen können.

Darüber hinaus können über spezielle Interface-Deklarationen (`XS`) auch andere Sprachen angebunden werden, wie z.B. `C`. Die `C`-Funktionen werden über einen speziellen Wrapper[1]-Code eingebunden und unter Unix in einer Shared-Library bzw. unter Windows in einer DLL[2] eingebunden. Der jeweilige Code wird dann erst zur Laufzeit geladen, wenn er wirklich aufgerufen wird.

Durch die vielfältigen Möglichkeiten der „Regulären Ausdrücke“[Fri98] können extrem einfach und elegant Daten aus Texten und Protokollen extrahiert werden.

Die verwendete Version von Perl ist „Perl 5.004 Patchlevel 04“.

Perl unterliegt der GNU-Public-License.

[1] Umschlag
[2] Dynamic Link Library

3.6.3. HTTP-Server: Apache und `mod_perl`

Zur Demonstration des Systems war ein Internet-Server für das `HTTP`-Protokoll notwendig. Neben verschiedenen kommerziellen Produkten existiert der Apache-Server. Der Apache-Server entstand, nachdem die Entwicklung des damals populären NCSA-Servers ins Stocken geriet und immer mehr Entwichler in eigener Regie Patches für den NCSA-Server entwickelten. Bald wurde aus dem gepatchten NCSA-Server (a patchy server) der Apache.

Apache stellt (in seinen verschiedensten Formen) z.Zt. mehr als 50% aller Internet-Server weltweit, gefolgt vom „Microsoft Information-Server“ und dem „Netscape Enterprise Server“.

Für den Apache-Server bestehen viele verschiedene Module. Jedes dieser Module kann dem Apache eine spezielle Funktionalität liefern. Für eine individuelle Konfiguration müssen nur die Module eincompiliert werden, die unbedingt notwendig sind. Dadurch bleibt der Server so klein und so schnell wie möglich.

Für diese Diplomarbeit wurde das Modul `mod_perl` (Version 1.14) eingebunden. Dieses Modul erlaubt es, neue Module nicht nur über die schon vorhandene C-API aufzusetzen, sondern das neue Modul komplett in Perl zu schreiben. Nach dem Start des Servers wird das Modul compiliert und resident im Speicher gehalten. Dadurch ist es sehr performant.

Der Apache unterliegt ebenfalls der GNU-Public-License. Die Versionsnummer der verwendeten Version ist v1.3.0.

3.6.4. Client: Perl-Proxy

Bei der Entwicklung des Systems stellte sich die Frage, welcher HTTP-Client (Browser) verwendet werden sollte. Anstatt einen komplett neuen Browser zu entwickeln, wurde ein Proxy-Server entwickelt, der als Mittler zwischen einem beliebigen Browser und dem Internet fungiert. Der Browser setzt seine Requests bei dem Signatur-Proxy ab wie bei jedem anderen Internet-Proxy auch. Dadurch kann der Surfer das gewohnte Programm weiter verwenden, d.h. daß das System unanhängig vom verwendeten Client-Programm ist.

Die Darstellung der Ergebnisse wird vom Proxy übernommen. Dazu erzeugt der Proxy ein Fenster auf dem X-Windows-Bildschirm. Konnte er eine Signatur übertragen, wird der zur Signatur gehörige URL im Fenster dargestellt. Durch anklicken des jeweiligen Eintrages kann der Benutzer genaue Informationen zu Art, Aussage der Sigenatur bekommen und das Ergebnis der Signaturptüfung erfahren.

In dieser Arbeit sollten geschachtelte Strukturen (z.B. Hypertexte) berücksichtigt werden. Aufgrund des Proxy-Konzeptes kann der Proxy nicht unterscheiden, welcher Inhalt über einen anderen Inhalt eingebunden wurde (z.B. Bilder auf einer Seite). Der Proxy behandelt alle Requests gleichberechtigt und unabhängig voneinander. Die Analyse, welche Inhalte durch einen Hypertext eingebunden wurden, kann nur das Programm erledigen, das die Repräsentation der Daten übernimmt. Somit müßte dieses Feature in den Internet-Browser integriert werden.

3.6.5. Kryptografie: SGMTT und `Crypt::RIPEMD160`

SGMTT

In dieser Arbeit wurde die „SmartGuard MTT“-Bibliothek (SGMTT) der Aachener Firma „Kryptokom“ verwendet. SGMTT bildet den MTT-Standard auf eine C-Bibliothek ab.

Zur Realisierung dieses Projektes mußte eine Schnittstelle zwischen der SGMTT-C-Bibliothek und der Sprache Perl geschrieben werden. Diese Schnittstelle ist im Meta-Format `XS` geschrieben, welches von Perl in echten C-Quellcode und die eigentliche Schnittstelle umgewandelt wird. Ziel dieser Lösung war, den Anwender von den Verwaltungsaufgaben wie Speicher-Allokation und Freigabe und dem Wissen um die C-API von SGMTT zu befreien. Innerhalb von Perl wurde ein Objekt-orientierter Ansatz gewählt: Der Anwender erzeugt zuerst ein `Crypt::SGMTT::Signature`-Objekt. Dieses Objekt geht von Standard-Pfaden für Zertifikatsdatenbank etc. aus. Der Anwender kann dem Objekt aber auch andere Konfigurationen mit übergeben. Ruft der Anwender die Methoden `sign()` bzw. `verify()` auf, werden die übergebenen Daten signiert bzw. verifiziert. Über die Methode `error()` kann der Status der letzten Operation abgefragt werden. Die Methode `as_string()` liefert das Ergebnis der letzten Operation zurück, d.h. nach einer Signatur die signierte Nachricht und nach einer Verifikation den Klartext. Darüber hinaus kann man noch explizit über die `sig_as_string`- und `clear_as_string`-Methoden auf die jeweiligen Teile zugreifen.

`Crypt::RIPEMD160`

Als kryptografisch sichere Hash-Funktion wurde der RIPE-ME 160 ausgewählt. Die Realisierung als Perl-Modul stammt vom Autor der Diplomarbeit und ist im Internet verfügbar. RIPEMD 160 ist eine Hash-Funktion mit einer Ausgangs-bit-Breite von 160 bit. RIPEMD160 wurde in Europa entwickelt.

3.7. HTTP-Server

Der Apache-Server wurde durch ein Apache/Perl-Modul erweitert. Module für den Apache werden über die Apache-interne API eingebunden. Das Modulkonzept basiert auf dem Grundsatz, daß jedes Modul seine eigenen Konfigurationsdirektiven verstehen muß, die in den Konfigurationsdateien festgehalten sind; jedes Modul muß selbstständig erkennen, ob es für einen speziellen Request zuständig ist. Ein Modul kann einen Request komplett abarbeiten und für beendet erklären (Status: `OK`) oder das Modul kann sich „für nicht-zuständig erklären“ und den Request ans nächste Modul weiterleiten (Status: `DECLINED`).

Das Signatur-Modul erledigt eine Zwischenlösung. Es prüft, ob eine Signatur (sowohl implizit als auch explizit) verlangt wird. Ist das der Fall, wird die Existenz von Signaturen auf der Festplatte geprüft. Kann dem Signaturwunsch nachgekommen werden, setzt das Modul die entsprechenden Header und beendet sich mit dem Status `DECLINED`, d.h., die Auslieferung der

eigentlichen Daten wird den Nachfolgemodulen überlassen.

Um die Signatur-Datei finden zu können, existiert eine Übersetzungstabelle. Mit dieser Tabelle erweitert der Server den Dateinamen der eigentlichen URL um die Signaturschema-spezifische Erweiterung. Beispielsweise trägt die Signatur der Datei `index.html` die Erweiterung `.mtt`, wenn nach dem Schema `Mtt-V1` gefragt wurde. Die Signatur liegt also unter `index.html.mtt` im gleichen Verzeichnis, so sie denn existiert. Die Tabelle erledigt einfach die Auflösung von `Mtt-V1` nach `.mtt`.

Eine *explizit* geforderte Signatur kann der Server am Signatur-Suffix (.mtt) erkennen.

Um zu verhindern, daß bei Nutzung eines Verzeichnisses durch mehrere Nutzer eine Signatur zweckentfremdet werden kann, liefert der Server nur Signaturen aus, die den gleichen Besitzer wie die Datendatei haben.

3.8. HTTP-Client

Der HTTP-Client ist der Signatur-Proxy. Der Proxy bindet sich auf der lokalen Maschine an einen frei wählbaren Port und wartet auf einkommende Verbindungen. Im Browser wird dieser Port (im Normalfall 8080) als Proxy-Port eingetragen, d.h. `http://localhost:8080/` ist der URL, unter dem der Signatur-Proxy erreichbar ist. Die Konfiguration des Proxy läßt zu, daß dieser ebenfalls nur über einen Proxy mit dem WWW-Server kommuniziert.

Eine Proxy-Verbindung unter HTTP unterscheidet im wesentlichen nur durch die Request-Zeile am Anfang des Requests. Anstatt den relativen URL anzugeben, wie dies bei direkten Verbindungen zum WWW-Server der Fall ist, muß hier ein absoluter URL angegeben werden[3].

Der Proxy liest den Client-Request, bestimmt den zu ladenden URL und fügt die neuen Signatur-Request-Header ein. Danach verbindet er sich mit dem WWW-Server bzw. mit dem dazwischengeschalteten (echten) Proxy und setzt seinen Request ab. Danach arbeitet er als Byte-Weiterleiter vom Server zum Client, bis eine der beiden Seiten die Verbindung schließt. Nachdem der Request abgeschlossen ist, versucht er eine Signatur zu decodieren. Im Erfolgsfall legt er die Daten und die Signatur auf der Festplatte ab und schickt dem Programm zum Anzeigen der Signatur eine Nachricht.

Um die Funktion ein wenig zu verfeinern, sei an dieser Stelle auf die genauen Detailles eingegangen:

1. Nach dem Start des Programmes wird eine Unix-Message-Queue initialisiert. Durch diese Message-Queue werden zu einem späteren Zeitpunkt die Nachrichten über erfolgreich decodierte Signaturen zum Display-Programm geschickt.

[3] Zur Verdeutlichung zwischen relativ und absolut: Bei einer direkten Verbindung zum Host `www.uni-wuppertal.de` wird der URL `http://www.uni-wuppertal.de/dir1/dir2/index.html` relativ als `/dir1/dir2/index.html` angegeben. Der verwendete Host ist ja durch die Verbindung zu eben diesem implizit bekannt und muß deshalb nicht mit angegeben werden.

2. Dann versucht das Programm, über den Unix-Systemaufruf `fork()` eine Kopie des aktuellen Prozesses zu erzeugen. Der Parent-Prozess baut danach das Fenster zum Anzeigen der Signaturen auf und wartet auf eingehende Nachrichten. Dieser Prozess wird im nachfolgenden einfach nur „Tk-Prozess“ genannt.

3. Die Kopie des Prozesses (der Child-Prozess) beginnt jetzt mit der eigentlichen Netzwerk-Arbeit. Im nachfolgenden wird der Child-Prozess einfach „Empfangsdame“ genannt. Die Aufgabe der Empfangsdame ist es, an den lokalen Proxy-Port (Default: 8080) zu binden und auf eingehende Verbindungen zu warten. Baut der Internet-Browser nun eine Verbindung zur Empfangsdame auf, forkt die Empfangsdame und erzeugt einen neuen Child-Prozess, im folgenden „Worker“ genannt. Der Worker übernimmt die neu aufgebaute Verbindung, während die Empfangsdame wieder auf dem Proxy-Port auf weitere Verbindungen wartet.

   Durch dieses mehrfache `fork()`en wird gewährleistet, daß es zu keinen Blockierungen kommt. Der ursprüngliche Tk-Prozess muß zu einem späteren Zeitpunkt die grafische Benutzeroberfläche bedienen und sollte daher nicht mit Netzwerkaufgaben betraut sein, damit während Aktionen wie einem Screen-Redraw die Netzwerk-Ebene trotzdem weiterarbeiten kann. Ebenso sollte die Empfangsdame nur für das Annehmen neuer Verbindungen zuständig sein. Durch die Möglichkeit, neue Prozesse zu erzeugen, vereinfacht sich der Netzwerk-Code erheblich, da nicht auf viele verschiedene Verbindungen geachtet werden muß.

4. Der Worker kann nun mit der eigentlichen Netzwerk-Arbeit beginnen. Zuerst prüft der Worker, ob die eingehende Verbindung vom lokalen Rechner stammt. Diese Prüfung erfolgt aus Basis der IP-Adresse. Diese muß 127.0.0.1 lauten. Damit soll verhindert werden, daß andere Rechner im Netzwerk den Signatur-Proxy als regulären Proxy für das Internet verwenden.

   Der Worker liest über den neu zugewiesenen Port den Request vom Browser, fügt die Signatur-Request-Header hinzu und verbindet sich mit dem Server. Ist ein Proxy eingetragen, verbindet er sich mit dem Proxy, anstatt den Server direkt zu kontaktieren. Bei einer direkten Verbindung zum Server muß der Worker den URL von einem absoluten URL in den relativen URL umwandeln, bevor er ihn in der Request-Zeile an den Server weiterreicht. Zusätzlich zum Client-Request-Header muß noch ein evtl. vorhandener Request-Body gelesen und gespeichert werden. Nach der Analyse und Modifikation der Client-Daten baut der Signatur-Proxy die Verbindung auf.

   Nachdem der Worker den neuen Request inclusive evtl. vorhandenem Body zum Server gesendet hat, versucht er, einen vollständigen Response-Header vom Server zu erhalten. Nachdem der vom Server kommende Header komplett gelesen wurde, wird er auf korrekten Aufbau geprüft. Fürs nächste fungiert der Worker nur als Byte-Weiterleiter, d.h. er versucht alle vom Server gelesenen Bytes sofort an den Client weiterzuleiten. Dadurch soll die Benutzung des Proxy vollkommen transparent für den Nutzer sein, also auch in

Bezug auf die momentane Übertragungsrate. Nachdem einer der beiden Kommunikationspartner (Browser oder Server) den Socket schließt, beendet der Proxy die andere noch offene Verbindung ebenfalls.

5. Nachdem alle Daten zwischen Browser und Server ausgetauscht sind, muß der Worker herausfinden, ob im Response eine Signatur vorhanden war. Er untersucht die Header des Response auf `X-Signature-Response`-Header. Sind keine Signatur-Header vorhanden, weiß der Proxy, daß er mit einem nicht-Signatur-fähigen Server spricht und daß er versuchen muß, die gewünschten Signaturen von Hand zu beschaffen. Dazu baut der Proxy eigenständig eine Verbindung auf und versucht, die Signaturdatei direkt (explizit) zu laden. Sind mehrere Schemata möglich, muß er alles durchprobieren, bis er Erfolg hat.

   War es dem Worker möglich, eine Signatur zu bekommen, speichert er die Daten und die Signatur in zwei separaten Dateien auf der Festplatte ab und sendet über die IPC-SysV-Message-Queue den erfolgreichen URL an den Tk-Prozess. Um eine einfache Methode zu haben, einen URL-Inhalt und die Signatur zu speichern und wiederzufinden, wird der URL gehasht und das Ergebnis als Dateiname verwendet. Nach dem Speichern der beiden Dateien beendet sich der Worker.

6. Der Tk-Prozess fragt regelmäßig die Message-Queue ab. Sind Nachrichten mit URLs eingetroffen, werden die URLs auf dem Bildschirm angezeigt. Mit einem Doppelclick auf den entsprechenden Eintrag kann der Benutzer Einzelheiten zu der Signatur erhalten.

4. Die Implementation

Die Struktur und Funktionsweise der einzelnen Komponenten wurde in Kapitel 3 beschrieben. Im aktuellen Kapitel soll nur auf Besonderheiten in der Implementierung eingegangen werden.

4.1. SmartGuard MTT

4.1.1. Einleitung

Die MailTrusT-Funktionsbibliothek der Firma Kryptokom/Aachen ist eine C-Bibliothek. Bei der Implementation ergaben sich leichte Einschränkungen:

PKI

SGMTT realisiert nur ein **einstufiges Zertifizierungsmodell**. Geplant war die Verwendung einer zweistufigen Zertifizierungshierarchie. Durch diese Einschränkung konnte die Implementierung also nur eine Root-CA berücksichtigen. Die Ebene der Web-Server-CA's wurde in dieser Beispielimplementierung weggelassen. Die Sicherheit des Systems ist durch diese Einschränkung nicht betroffen.

Ein weiteres Problem entstand in der **Eindeutigkeit der Zertifikatsidentifier**. Ein Zertifikat wird bei SGMTT nur über den „Distinguished Name" des Zertifikatsinhabers, den „Distinguished Name" der Zertifizierungsstelle und die Seriennummer des Zertifikats gekennzeichnet. Diese 3 Parameter sind nicht eindeutig genug, da die Zertifizierungsstelle mehrere Zertifikate mit diesen Parametern generieren könnte. Wünschenswert wäre an dieser Stelle ein zusätzlicher eindeutiger Identifier wie z.B. der PGP-Fingerprint, also eine kryptografisch sichere Hashsumme über den öffentlichen Schlüssel. Dadurch wäre ein Zertifikat nicht nur durch die Namen gekennzeichnet, die ja „Schall und Rauch" sind, sondern durch den eindeutigen mathematischen Zusammenhang zwischen kryptografischen Schlüsseln und Fingerprint.

SGMTT sammelt alle Zertifikate in einer Zertifikatsdatenbank. Verifiziert man eine MTT-Nachricht, die mit einem bis dato unbekannten Zertifikat signiert wurde, so muß zur korrekten Verifikation das Zertifikat in der Datenbank stehen. Das SGMTT-Modul wird mit den Optionen `MTT_FLAG_CHECKSIGPATH` und `MTT_FLAG_UPDATETRUSTED` betrieben.

`MTT_FLAG_CHECKSIGPATH` bewirkt, daß bei Zertifikaten der komplette Pfad bis zur Root-CA überprüft werden muß („Testen des kompletten Pfades von Signaturzertifikaten" ([Sch98b])). Leider gibt es z.Zt. keine Möglichkeit, ein einzelnes Zertifikat der Zertifikatsdatenbank als *das*

Root-Zertifikat zu definieren. Dadurch tritt der Effekt auf, daß auch self-signed Zertifikate bei der Signaturverifikation den Rückgabewert `MTT_ERR_MIC_VALID_WITH_PATH` („Signature verified with root PK“) liefern. Da die Zertifizierungskette bis zu irgendeinem „Root“-Zertifikat hochgeprüft werden konnte, nimmt SGMTT das Zertifikat als vollständig geprüft an; dabei ist SGMTT egal, um welches Root-Zertifikat es sich handelt.

Da man über die Rückgabewerte der Funktion nicht prüfen kann, ob das Zertifikat von der gewählten Root-CA zertifiziert wurde oder ob es self-signed ist, muß verhindert werden, daß Zertifikate ohne korrekten Root-CA-Stempel in die Datenbank aufgenommen werden. Dieses Verhalten steuert der Schalter `MTT_FLAG_UPDATETRUSTED`: Nur Zertifikate, die von der (schon bekannten) Root-CA zertifiziert sind, können in die Datenbank übernommen werden und so zur Prüfung der Signaturen verwendet werden. Dadurch wird eine „Verschmutzung“ der Datenbank mit self-signed-Zertifikaten verhindert.

Aufgrund dieser Einschränkungen ist es nicht möglich, Signaturen zu prüfen, die kein gültiges Zertifikat vorweisen können. Eine Notlösung wäre gewesen, parallel zur normalen Datenbank eine Mülldatenbank zu betreiben, in der Zertifikate eingelesen werden können, die die reguläre Datenbank nicht annimmt. Von diesem Ansatz wurde abgesehen.

4.1.2. Schnittstellen und Funktionen

Die Funktionsaufrufe sind in der Wrapper-Sprache XS geschrieben. Mit Hilfe von XS können C-Funktionen an Perl-Programme durchgereicht werden, ohne daß sich Perl um die Interna der Funktionen kümmern muß. Im XS-File werden zwei Funktionen verwendet: `crypt__Signatur` und `crypt__Scan`. Diese beiden XS-Funktionen bekommen die zu signierenden bzw. zu verifizierenden Daten übergeben. Die Konfiguration der beiden Funktionen geschieht über einen zusätzlich übergebenen Perl-Hash[1]. Für weitere Informationen zu den internen Funktionsaufrufen bzw. der SGMTT-API sei an dieser Stelle auf [Sch98b] verwiesen.

Alle Aufrufe der Signatur- bzw. Verifikationsoperationen sind zustanslos, d.h. daß bei aufeinanderfolgenden Aufrufen die MTT-Engine jedesmal neu konfiguriert werden muß.

Der Anwender greift mit seinen Programmen nicht direkt auf die beiden XS-Funktionen zu. Die Funktionalität ist innerhalb eines `Crypt::SGMTT::Signature`-Objektes eingebunden. Dieses Objekt kann über mehrere Unterobjekte konfiguriert werden. Unterbleibt diese Konfiguration, werden sinnvolle Default-Werte angenommen.

Für die Programmierung von C-Extensions für Perl sei hier auf [Roe96, Oka96, OR97] hingewiesen.

`Crypt::SGMTT` legt die Datenbanken etc. unter UNIX defaultmäßig im Verzeichnis `$HOME/.sgmtt/` ab. Die Bibliothek ist so compiliert, daß keine Ausgaben auf der Console erfolgen sollen. Damit unterbleiben störende Meldungen, wenn eine grafische Benutzeroberfläche verwendet wird.

[1] Der Hash bei Perl hat nichts mit kryptografischen Hashfunktionen zu tun: Es handelt sich um einen der 3 Grunddatentypen von Perl; hier werden Werte über beliebige Schlüsselworte indexiert.

4.1.3. Ein Beispiel

Zur Demonstration der Funktion des Moduls sei hier ein kurzes Beispiel angeführt. (Die `while()`-Schleife hängt alle auf des Standard-Eingabe kommenden Zeilen an `$data` an. Dieses Beispiel arbeitet als Unix-Filter. In diesem Beispiel wurde die PIN zur Nutzung des privaten Schlüssels hart im Programmcode eincodiert, was in der Praxis natürlich nicht sein darf.

Das Erzeugen einer einfachen Signatur kann somit folgendermaßen aussehen:

```
while ($line = <STDIN>) {
   $data .= $line;
}

$sig = new Crypt::SGMTT::Signature('CFG' => new Crypt::SGMTT::CFG('CFG_PIN' => '1234'));

$sig->sign($data);

if ($sig->error eq 'MTT_ERR_OK') {
   print $sig->as_string();
} else {
   print $sig->error, ": ", $sig->error_as_string, "\n";
   exit(0);
}
```

Eine Verifikation

Nach dem gleichen Schema läuft die Signatur-Verifikation ab:

```
while ($line = <STDIN>) {
   $data .= $line;
}

$sig = new Crypt::SGMTT::Signature();

$sig->verify($data);

if ($sig->sig_ok) {
   print "Diese Signatur wurde von ", $sig->cert_d_name, " erzeugt:\n";
   print $sig->as_string;
} else {
   print $sig->error, ": ", $sig->error_as_string, "\n";
   exit(0);
}
```

4.2. Simple Signature Protocol (SSP)

Das SSP wird durch das Perl-Objekt `Crypt::SSP::Data` implementiert. Der Konstuktor `new()` bekommt die verschiedenen Signatur-Parameter übergeben. Der Parameter `Data` enthält die zu

signierenden Daten. Damit wird die Hash-Berechnung dem Objekt überlassen.

Die `new_from_dat`-Methode konstuiert ein `Crypt::SSP::Data`-Objekt aus einem Text.

`time_valid(time)` prüft, ob das spezifizierte Zeitintervall der Signatur den angegebenen Zeitpunkt noch mit einschließt. Ohne Parameter wird die aktuelle Zeit verwendet.

`verify('Data' => $data, 'Url' => $url)` prüft, ob die Daten mit der Hash-Summe eines SSP-Objektes übereinstimmen und ob der Url korrekt eingetragen ist. Der Url ist optional; wird er angegeben, muß er mit dem gespeicherten identisch sein.

Die Methode `as_string()` liefert das SSP-Objekt als Text zurück.

Ein Beispiel

Das nachfolgende Beispiel erzeugt ein neues `Crypt::SSP::Data`-Objekt aus den übergebenen Daten und initialisiert über die Textrepräsentation ein neues SSP-Objekt. Am zweiten Objekt werden dann verschiedene Parameter geprüft.

```
use Crypt::SSP::Data;
use HTTP::Date;

$sig_data = new Crypt::SSP::Data('URL' => 'http://www.uni-wuppertal.de/~cgeuer/index.html',
                                 'Data' => 'Der signierte Text',
                                 'Valid-Not-Before' => HTTP::Date::time2str(time+10*(60*60*24)),
                                 'Valid-Not-After' => HTTP::Date::time2str(time+100*(60*60*24)),
                                 'Reference-Allowed' => 'no',
                                 'Comment' => ['Dies', 'sind', 'verschiedene', 'Zeilen'],
                                 );

$sig_data_2 = Crypt::SSP::Data->new_from_dat($sig_data->as_string());

print "URL: ",  $sig_data_2->url, "\n";

print "Der Hash-Wert der Daten ist ", $sig_data_2->verify('Data' => '') ?
        "OK\n" : "nicht OK\n";

print "Nicht gültig vor ", $sig_data_2->Valid_Not_Before(), "\n";

print "Die Signatur ist jetzt ", $sig_data_2->timevalid ? "gültig\n" : "nicht gültig\n";

__END__;
```

4.3. HTTP-Server

Das `mod_perl`-Modul für den Apache-Server realisiert die Funktionalität zum Übertragen der Signaturen. `mod_perl` bildet die Apache-interne API auf Perl-Module ab. Zu Interna bezgl. Apache sei auf die Online-Dokumentation unter `http://www.apache.com/`, `http://perl.apache.org` sowie [Eil97, LL97, Mac97, Mac98, Won98] hingewiesen.

Die Funktion, die den Einsprungpunkt für den Apache bildet, lautet `handler()`. Diese Funktion bekommt als erstes Argument ein Apache-Request-Objekt übergeben. Über die Methode `filename()` kann der Dateiname des Requests bestimmt werden. Mit Hilfe der Methode `header_in(Name)` können einzelne Request-Header abgefragt werden. `header_out(Name,Data)` setzt den entsprechenden Header für den Response.

Die verwendeten Return-Werte lauten DECLINED, OK und NOT_FOUND. DECLINED sagt dem Server, daß der Request an nachfolgende Module weitergereicht werden soll. OK bedeutet für den Server, daß der Request komplett beendet ist und daß die Verbindung zum Client geschlossen werden kann. NOT_FOUND läßt den Server eine Standard-Fehlermeldung ausgeben.

Die Grobstruktur des Moduls ist relativ einfach. Zuerst wird anhand des Dateinamens geprüft, ob es sich um einen impliziten oder expliziten Request handelt. Diese Unterscheidung wird auf Basis der Extension getroffen.

Handelt es sich um eine bekannte Signatur-Extension, also um einen expliziten Request, so wird versucht, die zugehörige Datendatei zu lokalisieren und zurückzuliefern. Bei impliziten Requests muß die Signatur in die Header eingekapselt werden. Bei allen Prüfungen über Lieferbarkeit von Signaturen müssen immer die Dateibesitzer berücksichtigt werden.

4.4. HTTP-Client

Vom Programmiervolumen her war der Signatur-Proxy neben der SGMTT-Anbindung das aufwendigste Stück Software. Der Proxy besteht aus den Codesegmenten für die grafische Tk-Oberfläche, dem Code für die Empfangsdame sowie dem Worker-Code.

Nachdem der Tk-Client die Empfangsdame erzeugt hat, macht er sich an den Aufbau des Hauptfensters und fragt regelmäßig die Message-Queue ab. Die Empfangsdame wartet einfach auf eingehende Verbindungen und erzeugt für jede Verbindung einen eigenen Worker. Der Worker selbst muß den Client-Request lesen, die Signaturforderung hinzufügen, den Request zum Server weiterleiten, den Serverresponse lesen und zum Client weiterleiten und danach die evtl. decodierte Signatur speichern und eine Benachrichtigung zum Tk-Client schicken.

Da der Worker die Prüfung auf Signaturen erledigt, nachdem der Client seine Daten erhalten hat, ist er zu diesem Zeitpunkt nicht mehr in der zeitkritischen Phase. Der Signatur-Proxy wurde so entworfen, daß er versucht, die Signaturen manuell (explizit) zu beschaffen, wenn sie nicht automatisch mitgeliefert wurden. Der Worker baut also u.U. nochmal eine oder mehrere HTTP-Verbindungen auf, um auch von nicht-Signaturfähigen Servern Signaturen zu holen.

Zu bemerken wäre vielleicht noch, daß das Speichern von Signatur und Inhalten vom Modul `Stored.pm` übernommen wird. Durch das Perl-Feature des Bindens von Variablen an Funktionsaufrufe wird der Eindruck erweckt, daß die Daten und Signaturen nur einem Hash zugewiesen werden. In Wirklichkeit speichert das Stored-Modul die Daten und Signaturen auf der Festplatte. Zur Grundidee der Speicherung ist nur zu sagen, daß der URL mit einer kryptografischen Hash-Funktion in eine eindeutige Zeichenkette fester Länge umgewandelt wird, die als Dateiname für

Daten und Signatur dient. Dadurch stören Sonderzeichen in URLs oder besonders lange URLs nicht.

5. Ergebnisse

Im nachfolgenden Kapitel soll die korrekte Funktion des Systems überprüft werden. Dazu werden die Anforderungen noch einmal genau definiert, die Testfälle und Testdaten konstruiert sowie die Ergebnisse dokumentiert und zusammengefasst.

5.1. Anforderungen der Spezifikation sowie mögliche Angriffsszenarien und Protokollfehler

In diesem Teil wird die Spezifikation auf korrekte Funktion hin untersucht. Zuerst werden Möglichkeiten aufgezeigt, welche Fälle in der Prüfung berücksichtigt werden müssen.

Die ersten Testfälle (Nr. 1-6) verifizieren die Funktion des Servers und des korrekten Signaturtransportes. Die Testfälle 7-14 prüfen, ob die Client-Implementierung korrekt arbeitet und die Signaturen korrekt verifiziert.

5.1.1. Server-Funktionalität

Der Abschnitt „Server-Funktionalität“ beschreibt die Forderungen, die an den HTTP-Transportmechanismus gestellt werden. Hauptziel der Forderungen ist ein einfaches und logisches Verhalten des HTTP-Servers.

Testfall 1

Der einzige Fall, der dem Server erlaubt, eine Signatur zurückzuliefern, ist der Betriebsfall mit einem korrekten Request durch den Client und einem korrekten Response durch den Server. Dieser Fall stellt mehrere Bedingungen:

1. Der Request durch den Client muß korrekt decodiert werden können.
2. Die zur Signatur gehörigen Klartext-Daten müssen auf dem Server vorhanden sein.
3. Von den durch den Client geforderten Signaturschemata muß mindestens eines existieren.
4. Die geforderten Daten und die zugehörige Signatur müssen den gleichen Besitzer haben.

Dies ist der *einzige* Fall, wo der Server eine Signaturdatei ausliefern darf.

Testfall 2

Fall 2 erwartet, daß der Eigner der Datendatei und der Eigner der Signaturdatei nicht identisch sind. Deshalb darf keine Lieferung erfolgen:

1. Der Request durch den Client muß korrekt decodiert werden können.
2. Die zur Signatur gehörigen Klartext-Daten müssen auf dem Server vorhanden sein.
3. Von den durch den Client geforderten Signaturschemata muß mindestens eines existieren.
4. Die geforderten Daten und die zugehörige Signatur haben nicht den gleichen Besitzer.

Der Server darf die Datendatei liefern; nicht aber die Signaturdatei.

Dieser Fall kann beispielsweise eintreten, wenn der Inhalteanbieter A seine Daten auf dem Web-Server abgelegt hat, ohne sie zu signieren. Speichert nun Inhalteanbieter B eine einzelne Signatur auf dem Server, die vorgibt, zu A's Inhalten zu gehören, verhindert der Server die Auslieferung, weil die Dateibesitzer nicht übereinstimmen.

Testfall 3

Fall 3 beschreibt den Request von nicht vorhandenen Schemata:

1. Der Request durch den Client muß korrekt decodiert werden können.
2. Die zur Signatur gehörigen Klartext-Daten müssen auf dem Server vorhanden sein.
3. Es existiert keine Signaturdatei, die auf die vom Client gewünschten Schemata paßt.

Es darf keine Signaturdatei geliefert werden. Damit der Client erfährt, daß der Server trotzdem Signatur-fähig ist, liefert der Server den Signatur-Fehler zurück, daß für den geforderten URL keine Signaturen verfügbar sind.

Testfall 4

In Fall 4 findet der Server eine oder mehrere „verwaiste" Signaturen ohne zugehörige Datendatei. In diesem Fall darf der Server keine Informationen über die Signaturen liefern, sondern muß mit einem `NOT_FOUND` terminieren. Dieses Verhalten muß sowohl bei einem impliziten Request auf die Daten als auch bei einem expliziten Request direkt auf die Signatur-Datei erhalten bleiben.

Wenn der Inhalteanbieter seine Daten vom Server löscht und vergißt, die Signaturen mit zu löschen, verhindert dieses Verhalten eine Lieferung Signaturen, deren korrespondierende Inhalte nicht mehr existieren.

Testfall 5

Fall 5 geht auf die Möglichkeit ein, daß der Request zwar einen `X-Signature-Request`-Header enthält, dieser aber defekt ist. In diesem Fall darf der Server keine Signaturen zurückliefern.

Fehlerhafte Protokoll-Elemente dürfen nicht verarbeitet werden.

Testfall 6

Auf dem Server existiert nur eine einzelne Signatur-Datei ohne zugehörige Datendatei. Der Client versucht, die Signaturdatei explizit zu laden. Dieser Versuch muß vom Server mit `NOT_FOUND` abgelehnt werden.

5.1.2. Client-Funktionalität

Im Abschnitt „Client-Funktionalität“ sollen Anforderungen für die Auswertung der digitalen Signatur an sich definiert werden. Hier ist der Transportmechanismus nicht mehr von Bedeutung, sondern die digitale Signatur steht im Mittelpunkt.

Testfall 7

Der absolut korrekte Fall einer korrekt verifizierten Signatur beinhaltet mehrere notwendige Voraussetzungen:

1. Die digitale Signatur muß decodierbar sein, d.h. daß sie das spezifizierte Format aufweisen muß (in diesem Fall MailTrusT). Der Decodieralgorithmus für das MailTrusT-Format muß einen Rückgabewert für eine korrekt verifizierte Signatur liefern.
2. Das Zertifikat, das zusammen mit der Signatur geliefert wird, muß von der Root-CA zertifiziert und noch gültig sein.
3. Das SSP-Objekt, das in der MailTrusT-konformen Nachricht eingekapselt ist, muß formal korrekt aufgebaut sein.
4. Die Hash-Wert, der im SSP-Objekt festgehalten ist, muß mit dem Hash-Wert über die signierten Daten übereinstimmen.
5. Der URL, dem im SSP-Objekt festgeschrieben ist, muß mit dem URL übereinstimmen, von dem die Daten geladen wurden.
6. Der im SSP-Objekt definierte Gültigkeitszeitraum für die Signatur muß den Zeitpunkt der Signatur-Verifikation einschließen.

Wenn diese sechs Voraussetzungen zutreffen, wird die Signatur als gültig anerkannt.

Testfall 8

Der Unterschied zu Testfall 7 liegt nur darin, daß im SSP-Objekt kein Zeitintervall spezifiziert wurde. Dieser Fall bedeutet, daß die Signatur unabhängig vom Zeitpunkt ist. Treffen die in Testfall 6 getroffenen Voraussetzungen alle zu (mit Ausnahme des fehlenden Zeitintervalls), so ist die Signatur als ebenfalls gültig zu klassifizieren.

Testfall 9

Testfall 9 beschreibt ein Szenario, in dem das Zeitintervall den aktuellen Zeitpunkt nicht einschließt. In diesem Fall ist die Signatur als nicht gültig einzustufen.

Testfall 10

Testfall 10 beschreibt den Fall, daß sowohl SGMTT, das Zertifikat, das SSP-Objekt und der Content-Hash korrekt sind, aber nicht der URL. In diesem Fall muß die Signatur für ungültig erklärt werden.

Dieser Fall tritt z.B. ein, wenn die Daten und die Signatur kopiert wurden und auf einem anderen Server gespeichert wurden. So wäre hier denkbar, daß mit den Daten und der Unterschrift einer bekannten Institution Mißbrauch getrieben werden soll, indem der Server des Angreifers durch die kopierten Signaturen aufgewertet werden soll und der Eindruck hervorgerufen werden soll, der Signierer würde Inhalte auf dem Server des Angreifers anbieten.

Testfall 11

Testfall 11 beschreibt den Fall, daß die gelieferten Daten und die Signatur nicht zusammengehören, d.h. daß der Hash-Wert über die Daten nicht mit dem im SSP-gespeicherten Wert übereinstimmt. Trotzdem stimmt der URL im SSP-Objekt mit dem geladenen URL überein.

Dieser Fall tritt beispielsweise auf, wenn ein Angreifer einen zwischengeschalteten Internet-Proxy modifizieren kann oder es einem Angreifer gelungen ist, auf dem Web-Server die korrekten Inhalte gegen eigene auszutauschen. Der Fall kann aber auch eintreten, wenn während der Übertragung ein Bitfehler unterlaufen sein sollte (was bei TCP/IP zugegebenermaßen relativ unwahrscheinlich ist).

Testfall 12

Testfall 12 geht davon aus, daß die MailTrusT-Nachricht zwar decodiert werden konnte und ein korrektes Zertifikat vorliegt. Das eingebundene SSP-Objekt ist aber fehlerhaft.

Dies kann geschehen, wenn während der Signatur ein Fehler passiert ist oder wenn eine gewöhnliche MailTrusT-signierte Nachricht als Signatur eingespielt wurde.

Testfall 13

Im vorletzten Fall kann eine SGMTT-Nachricht nicht verifiziert werden, weil das Zertifikat nicht in die Datenbank aufgenommen werden durfte. Das ist der Fall, wenn das Zertifikat nicht von der Root-CA zertifiziert wurde.

Bei einem solchen Vorfall könnte es sich um ein self-signed Zertifikat handeln.

Eine solche Signatur ist ungültig.

Testfall 14

Der letzte untersuchte Fall ist der, daß die MailTrusT-Nachricht selbst nicht erkannt wird. Das ist immer dann der Fall, wenn Übertragungsfehler die Nachricht zerstören oder andere Daten als `.mtt`-Datei auf dem Server abgespeichert wurden.

In diesem Fall ist überhaupt keine Interpretation möglich und die Signatur ist als defekt zu markieren.

5.2. Testdaten, Testdurchführung und Auswertung

5.2.1. Server-Funktionalität

Um die Funktionalität des Transportmechanismus zu zeigen, ist es nicht notwendig, daß die Dateien korrekte Formate aufweisen. Für die Tests 1-6 ist der Inhalt einer Datei einfach gleich ihrem Dateinamen.

Testfall 1

Auf dem Server liegt eine Daten-Datei mit mehreren Signaturen: `/test1.txt`, `/test1.txt.mtt`, `/test1.txt.pgp2` und `/test1.txt.pgp5`. Die ersten drei Dateien gehören Alice, `/test1.txt.pgp5` gehört Bob. Dem Server werden also mehrere Signaturschemata als akzeptabel präsentiert. Der Server muß anhand der angegebenen Reihenfolge (Priorisierung) entscheiden, welche Signatur zu liefern ist.

Die Kommunikation sieht folgendermaßen aus:

```
GET /test1.txt HTTP/1.0
X-Signature-Request: Pgp-Dh Ssleay Mtt-V1 Pgp-Rsa

HTTP/1.1 200 OK
Date: Fri, 24 Jul 1998 20:09:43 GMT
Server: Apache/1.3.0 (Unix) mod_perl/1.14
X-Signature-Types-Avail: Mtt-V1 Pgp-Rsa
X-Signature-Response-Mtt-V1-0: dGVzdDEuaHRtbC5tdHQK
Last-Modified: Fri, 24 Jul 1998 20:06:57 GMT
```

```
ETag: "979b-b-35b8e961"
Accept-Ranges: bytes
Content-Length: 11
Content-MD5: r5yVQCYxeUb+lsSq70n8NQ==
Connection: close
Content-Type: text/plain

test1.txt
```

Dem Wunsch `Pgp-Dh` kann nicht entsprochen werden, da die Dateibesitzer von `/test1.txt` und `/test1.txt.pgp5` unterschiedlich sind. Dem Wunsch `Ssleay` kann nicht entsprochen werden, da die Datei`/test1.txt.ssl` nicht existiert. Der Server liefert in den Signatur-Header die Datei `/test1.txt.mtt`, da diese die nächstmögliche ist.

Damit konnte die Lieferung der ersten möglichen Datei demonstriert werden.

Es liegt Funktion vor.

Testfall 2

Auf dem Server liegt eine Daten-Datei `/test2.txt` und die zugehörige MailTrusT-Signatur `/test2.txt.mtt`. Die Datei `/test2.txt` gehört alice, `/test2.txt.mtt` gehört Bob.

Die Kommunikation sieht folgendermaßen aus:

```
GET /test2.txt HTTP/1.0
X-Signature-Request: Mtt-V1

HTTP/1.1 200 OK
Date: Fri, 24 Jul 1998 19:59:42 GMT
Server: Apache/1.3.0 (Unix) mod_perl/1.14
X-Signature-Error: Requested scheme not available
Last-Modified: Fri, 24 Jul 1998 19:58:48 GMT
ETag: "9799-b-35b8e778"
Accept-Ranges: bytes
Content-Length: 11
Content-MD5: zLmmKfF3xz25uIqVys48ew==
Connection: close
Content-Type: text/html

test2.txt
```

Es ist also nicht möglich, eine Signaturdatei mit dem falschen Besitzer anzurufen.

Es liegt Funktion vor.

Testfall 3

Auf dem Server liegt eine Daten-Datei `/test3.txt` und die zugehörige MailTrusT-Signatur `/test3.txt.mtt`. Beide Dateien gehören Alice.

Die Kommunikation sieht folgendermaßen aus:

```
GET /test3.txt HTTP/1.0
X-Signature-Request: Pgp-Rsa

HTTP/1.1 200 OK
Date: Fri, 24 Jul 1998 20:17:27 GMT
Server: Apache/1.3.0 (Unix) mod_perl/1.14
X-Signature-Types-Avail: Mtt-V1
X-Signature-Error: Requested scheme not available
Last-Modified: Fri, 24 Jul 1998 20:16:48 GMT
ETag: "97a2-b-35b8ebb0"
Accept-Ranges: bytes
Content-Length: 10
Content-MD5: mTjn724v9CSvX5yfFWRtKQ==
Connection: close
Content-Type: text/html

test3.txt
```

Es wird also die Fehlermeldung über fehlerhafte Schemata und die Liste der verfügbaren Signaturen geliefert.

Es liegt Funktion vor.

Testfall 4

Auf dem Server liegten drei Signaturdateien: `/test4.txt.mtt`, `/test4.txt.pgp2` und `/test4.txt.pgp5`. Alle Dateien gehören Alice.

Die Kommunikation sieht folgendermaßen aus:

```
GET /test4.txt HTTP/1.0
X-Signature-Request: Mtt-V1 Pgp-Rsa Pgp-Dh

HTTP/1.1 404 File not found
Date: Fri, 24 Jul 1998 20:23:16 GMT
Server: Apache/1.3.0 (Unix) mod_perl/1.14
Connection: close
Content-Type: text/html

<HTML><HEAD>
<TITLE>404 File Not Found</TITLE>
</HEAD><BODY>
<H1>File Not Found</H1>
The requested URL /test4.txt was not found on this server.<P>
</BODY></HTML>
```

Es wird also in keinster Weise aufgedeckt, daß noch drei verwaiste Signaturen auf der Platte liegen.

Es liegt Funktion vor.

Testfall 5

Auf dem Server liegt eine Daten-Datei `/test5.txt` und die zugehörige MailTrusT-Signatur `/test5.txt.mtt`. Beide Dateien gehören Alice.

Die Kommunikation sieht folgendermaßen aus:

```
GET /test5.txt HTTP/1.0
X-Signature-Doesn Wronf:d:342h k

HTTP/1.1 200 OK
Date: Fri, 24 Jul 1998 20:27:03 GMT
Server: Apache/1.3.0 (Unix) mod_perl/1.14
X-Signature-Types-Avail: Mtt-V1
Last-Modified: Fri, 24 Jul 1998 20:26:05 GMT
ETag: "97a7-b-35b8eddd"
Accept-Ranges: bytes
Content-Length: 10
Content-MD5: iBw0Jk0GtOhUTLxF8aVXAQ==
Connection: close
Content-Type: text/html

test5.txt
```

Obwohl die Signatur-Anforderung nicht lesbar ist, informiert der Server den Client korrekt über zur Verfügung stehende Schemata.

Es liegt Funktion vor.

Testfall 6

Auf dem Server liegt eine einzelne Signatur `test6.txt.mtt`. Der Client versucht, diese Datei direkt zu laden. Dieser Versuch muß verhindert werden.

Die Kommunikation sieht folgendermaßen aus:

```
GET /test6.txt.mtt HTTP/1.0

HTTP/1.1 404 Not Found
Date: Fri, 24 Jul 1998 20:35:08 GMT
Server: Apache/1.3.0 (Unix) mod_perl/1.14
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
```

```
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /test6.txt.mtt was not found on this server.<P>
</BODY></HTML>
```

Der Server kann an der Extension `.mtt` erkennen, daß es sich hierbei um eine MailTrusT-Signatur handelt. Das direkte Herunterladen wird nicht zugelassen, weil die Datendatei nicht mehr existiert.

Es liegt Funktion vor.

5.2.2. Client-Funktionalität

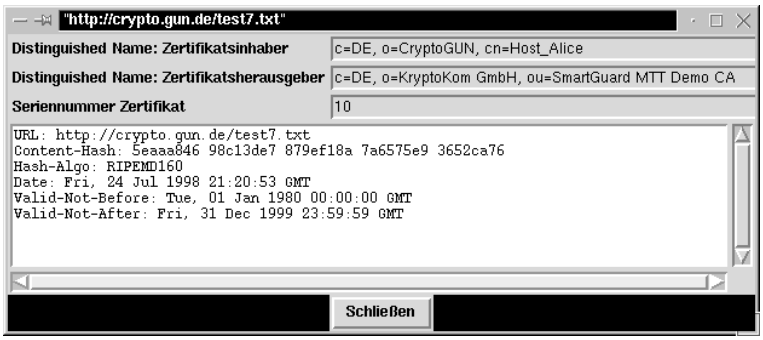
Zur Prüfung der Client-Funktionalität wird davon ausgegangen, daß mehrere Benutzer existieren: Alice, Bob und Mallet. Alice und Bob haben ein von der Root-CA zertifiziertes Zertifikat. Das Zertifikat von Mallet ist self-signed. Für diesen Abschnitt wird davon ausgegangen, daß die Übertragung der Signaturen durch HTTP korrekt funktioniert.

Testfall 7

Nachdem die Signatur korrekt übertragen wurde, muß sie geprüft werden. Der erste Fall der Signaturprüfungen betrachtet den Fall, daß die Signatur vollkommen korrekt erzeugt wurde (SGMTT, Zertifikat und SSP sind korrekt), daß die übertragenen Daten auch wirklich zu der Signatur gehören und nicht verändert wurden (Hash korrekt), daß die Signatur an der richtigen Stelle im Netz lag (URL korrekt) und daß die Gültigkeitszeiträume den aktuellen Zeitpunkt beinhalten.

Zur Prüfung dieser Kriterien wurde eine Nachricht erzeugt, die von Anfang 1980 bis Ende 1999 gültig ist. Diese Nachricht wurde mit einem von der Root-CA generierten Schlüssel erzeugt und am gleichen URL abgelegt, wie im SSP-Objekt festgehalten. Die Daten waren ebenfalls korrekt.

Nach der Übertragung erhielt der Client folgendes Fenster:



Es ist zu erkennen, daß die Signatur von „`c=DE o=CryptoGUN cn=Host_Alice`“ erzeugt wurde

und das Zertifikat von „c=DE o=KryptoKom GmbH, ou=SmartGuard MTT Demo CA“[1] bestätigt wurde. Die Seriennummer des Zertifikats ist 10. Zum Plausibilitätstest ist im Window auch noch das SSP-Objekt abgedruckt.

Es liegt Funktion vor.

Testfall 8

Zur Überprüfung von Testfall 8 wurde eine korrekte Signatur erzeugt, wobei kein Gültigkekts-intervall definiert wurde. Ansonsten waren die Parameter wie für Testfall 7.

Nach der Übertragung erhielt der Client folgendes Fenster:

Genau wie bei Testfall 7 wird das decodierte SSP-Objekt nach der Analyse im Fenster angezeigt. Der einzige Unterschied liegt im vollkommenen Fehlen eines Gültigkeitsintervalls.

Es liegt Funktion vor.

Testfall 9

Die Testdatem für Testfall 9 wurden wie für Testfall 7 erzeugt. Der Unterschied ist, daß diesmal der Ablaufzeitpunkt des Gültigkeitsintervalls in der Vergangenheit liegt.

Nach der Übertragung erhielt der Client folgendes Fenster:

Es werden keine SSP-Parameter mehr angezeigt, um nicht den Eindruck zu erwecken, daß die Signatur korrekt sein können. Statt dessen wird nur eine Message-Box angezeigt, die das abgelaufene Gültigkeitsintervall anzeigt.

[1] „ou=...“ steht in diesem Fall für Organizational Unit, also Abteilung

Es liegt Funktion vor.

Testfall 10

In den Testdaten für Testfall 10 wurde ein URL mit einem nicht-existenten Hostnamen angegeben. Da jetzt der SSP-URL und der vom Worker gelieferte URL nicht übereinstimmen, muß eine Ablehnung erfolgen.

Nach der Übertragung erhielt der Client folgendes Fenster:

Es werden keine SSP-Parameter mehr angezeigt, um nicht den Eindruck zu erwecken, daß die Signatur korrekt sein können. Statt dessen wird nur eine Message-Box angezeigt, die den im SSP eingetragenen URL anzeigt.

Es liegt Funktion vor.

Testfall 11

Für Testfall 11 wurden der Signatur aus Testfall 8 zugehörige, modifizierte Daten untergeschoben (1 Byte war geändert). Da der im SSP-Objekt festgeschriebene Hash-Wert nicht mehr stimmt, muß eine Ablehnung erfolgen.

Nach der Übertragung erhielt der Client folgendes Fenster:

Es wird eine Message-Box angezeigt, die den Benutzer über den Sachverhalt informiert, daß SSP-Objekt und die Übertragenen Daten nicht zusammengehören.

Es liegt Funktion vor.

Testfall 12

Um die Testdaten zu generieren, wurde mit dem SGMTT-Modul ein einfacher Text-String („Dies ist KEINE SSP-Nachricht“) korrekt signiert. Dadurch kann bei der Verifizierung der Signatur das SSP-Objekt nicht korrekt restauriert werden.

Nach der Übertragung erhielt der Client folgendes Fenster:

Es wird eine Message-Box angezeigt, die den Benutzer über den Sachverhalt informiert, daß das SSP-Objekt nicht gültig ist.

Es liegt Funktion vor.

Testfall 13

Um die Daten für Testfall 13 zu erzeugen, wurde eine MailTrusT-Nachricht mit einem self-signed Zertifikat erzeugt.

Nach der Übertragung erhielt der Client folgendes Fenster:

Es wird eine Message-Box angezeigt, die den Benutzer über den Sachverhalt informiert, daß das das Zertifikat nicht von der Root-CA erzeugt wurde.

Es liegt Funktion vor.

Testfall 14

Die Signaturdatei für Testfall 14 besteht aus einem kurzen Text, der keine MailTrusT-Datei enthält.

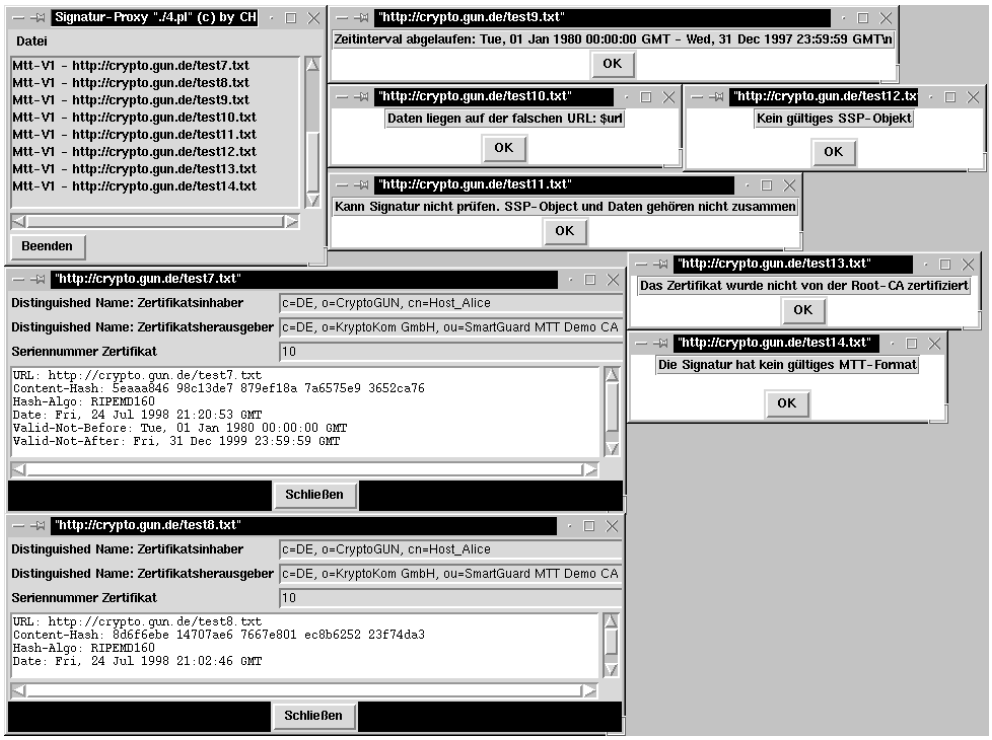
Nach der Übertragung erhielt der Client folgendes Fenster:

Der Benutzer wird darüber informiert, daß die Signatur keine MTT-Datei ist.

Es liegt Funktion vor.

5.3. Die grafische Oberfläche

In der nachfolgenden Grafik sind die Ergebnisse der Tests 7-14 noch einmal als Screen-Shot zu sehen. Im Rahmen jedes Fensters ist der zugehörige URL zu sehen.



Neben den jeweiligen Meldungen zu den einzelnen Verifikationen kann man links oben das Mainwindow der Applikation sehen, in dem die URLs mit Signaturen angezeigt werden.

5.4. Kurzübersicht der Ergebnisse

5.4.1. Server-Funktionalität

In der nachfolgenden Tabelle ist kurz die Überprüfung der Server-Funktion zusammengefasst. Dabei haben die verschiedenen Felder folgende Bedeutung:

1. **Request korrekt:** Konnte der HTTP-Server den Signatur-Request (den `X-Signature-Request`-Header korrekt decodieren?
2. **Daten vorhanden:** Liegen die zur Signatur gehörigen Daten auf dem Server bereit?
3. **Schemata gefunden:** Liegen die zu den Daten gehörigen Signaturdateien auf dem Server bereit?
4. **Dateieigner korrekt:** Haben die Daten und die Signatur-Dateien unter UNIX den gleichen Dateieigner?
5. **Signaturtransfer:** Darf die Signatur-Datei übertragen werden?
6. **Funktion:** Wurde die Signatur übertragen bzw. nicht übertragen wie gefordert?

Wenn die Angabe leer ist, ist das Ergebnis dieses Punktes für den jeweiligen Testfall nicht von Interesse:

Nr.	Request korrekt	Daten vorhanden	Schemata gefunden	Dateieigner identisch ?	Signaturtransfer erlaubt?	Funktion ?
1	ja	ja	ja	ja	ja	ja
2	ja	ja	ja	nein	nein	ja
3	ja	ja	nein		nein	ja
4	ja	nein	ja		nein	ja
5	nein				nein	ja
6	explizit	nein	ja		nein	ja

5.4.2. Client-Funktionalität

In der Client-Funktiona

1. **SGMTT:** Ist das SGMTT-Objekt an sich korrekt aufgebaut?
2. **Cert:** Ist das im SGMTT-Objekt mitgelieferte Zertifikat über die Root-CA prüfbar?
3. **SSP:** Ist das im SGMTT-Objekt eingekapselte SSP-Objekt korrekt aufgebaut?
4. **Hash:** Stimmt der Content-Hash im SSP-Objekt mit den übertragenen Daten überein?
5. **URL:** Ist der URL im SSP-Objekt der gleiche wie der Ursprung der Daten?
6. **Zeit:** Liegt der aktuelle Verifikationszeitpunkt innerhalb eines evtl. im SSP-Objekt spezifizierten Gültigkeitszeitraumes?
7. **Signatur OK? :** Welcher Status ist vom Programm zurückzuliefern?
8. **Funktion ?:** Erfüllt das Programm die Forderung?

Nr.	SGMTT	Cert	SSP	Hash	URL	Zeit	Signatur OK ?	Funktion ?
7	ja	ja	ja	ja	ja	ja	ja	ja
8	ja	ja	ja	ja	ja		ja	ja
9	ja	ja	ja	ja	ja	nein	nein	ja
10	ja	ja	ja	ja	nein		nein	ja
11	ja	ja	ja	nein	ja		nein	ja
12	ja	ja	nein				nein	ja
13	ja	nein					nein	ja
14	nein						nein	ja

5.5. Gesamtbewertung

Das entwickelte Signatur-System ermöglicht die einfache Formulierung von Signaturen und einen sicheren Transport durch das Internet. Durch den offenen Ansatz können beliebige Verschlüsselungsverfahren eingesetzt werden. Zu ein- und demselben Objekt im Internet können mehrere Signatur-Repräsentationen existieren, ähnlich der Möglichkeit, verschiedensprachige Versionen eines Dokumentes unter einem URL anzusprechen.

6. Ausblick

Das gerade spezifizierte und entwickelte System ist in der Lage, statisch abgelegte Inhalte sicher zu signieren. Ein Problem tritt auf, wenn dynamisch erzeugte Inhalte signiert werden sollen. Prinzipiell ist das System dazu in der Lage. Um diese Anforderung zu realisieren muß die Signatur erzeugt werden, nachdem das CGI-Programm bzw. der HTTP-Server die Inhalte berechnet und erzeugt hat. Dazu müßte das Signatursystem in den Datenstrom zwischen dem CGI-Programm und dem HTTP-Server „eingeschleift“ werden. Der verwendete Apache-Server bietet in der derzeitigen Version (v1.3.x) nicht diese Möglichkeit. Im Gespräch mit einem der Apache-Entwickler wurde mir mitgeteilt, daß diese Möglichkeit für Apache v2.x.x geplant ist. Bei der derzeitigen Entwicklungsgeschwindigkeit des Apache-Projektes bleibt die Hoffnung, daß dieser Zeitpunkt bald kommt.

Ein anderer Punkt für die Zukunft sind die realisierten Sicherheitsanforderungen. Die Sicherheitsanforderungen Integrität, Authentifikation und Verbindlichkeit sind mit dem hier erarbeiteten Ansatz abgedeckt. Was Vertraulichkeit, Zugriffskontrolle und Anonymität angeht, sind noch viele Probleme unerledigt: Zur Sicherung der Vertraulichkeit wäre es z.B. ein interessanter Gedanke, Dokumente verschlüsselt auf dem Server ablegen zu können und jedem Mitglied einer Arbeitsgruppe authorisiert Zugriff auf die Schlüssel zu geben, damit er auf die für ihn bzw. seine Gruppe bestimmten Dokumente zugreifen kann. Ein solcher Zugriff sollte für den Benutzer transparent geschehen, z.B. indem er seine Sicherheits-SmartCard in das Lesegerät seines Rechners steckt und danach vertraulich, authentisch und für den Benutzer transparent auf Inhalte eines Webservers zugreift.

In den nächsten Jahren und Jahrzehnten wird IPv4 (das aktuelle TCP/IP) durch IPv6 abgelöst. IPv6 bietet neben dem größeren Adressraum auch viele Sicherheitsfunktionen. Die Gesamtheit dieser neuen Sicherheiten betrifft aber nur die Transportebene. Damit kann z.B. die Vertraulichkeit einer Übertragung gesichert werden. Trotzdem werden die Anwendungsprogrammierer dadurch nicht davon erlöst sein, Sicherungsmechanismen in ihre Programme einzubauen: Um Integrität, Verbindlichkeit oder auch Anonymität zu sichern, sind Lösungen auf Anwendungsebene notwendig, die Daten auch nach dem Transport sichern, beispielsweise für die Speicherung.

Abschließend bleibt mir nur noch zu sagen, daß die Entwicklung des Systems großen Spaß gemacht hat und mit einfachen Mitteln ein funktionsfähiges System realisiert werden konnte. Ich hoffe, daß das Lesen dieser Arbeit ein bißchen Spaß gemacht hat... ;-)))

Christian Geuer

Düsseldorf, den 25. Juli 1998

A. Listings

In diesem Anhang wurden nur die wichtigsten Listings der jeweiligen Pakete aufgeführt. Neben diesen Quellcodes existieren noch weitere, die für das Verständnis des Systems aber nicht zwangsläufig notwendig sind.

A.1. `Crypt::SGMTT`

Das Perl-Modul `Crypt::SGMTT` stellt die Anbindung an die MailTrusT-Bibliothek dar.

A.1.1. `SGMTT.xs`

`SGMTT.xs` ist der Meta-Code für die Anbindung an die C-Funktionen.

```
/* -*- C++ -*- */
/*
 * SGMTT.xs enthält den Wrapper-Code für die
 * XSUBs für Perl 5.004
 *
 */

#ifdef __cplusplus
extern "C" {
#endif

#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"

#include "mtt_ext.h"
#include "sgmtt.h"

#ifdef __cplusplus
}
#endif

#define CLEAR_DOCFLAG_MIC_INVALID

#define HV_SET_TRUE(HV,KEY,SSV) SSV = hv_store(HV, KEY , strlen( KEY ), &sv_yes, 0); \
                                if (SSV == NULL) { croak("Bad: key '%s' not stored", KEY ); }
```

```
#define MACRO_PROCESS_DOCFLAGS(FLAG,KEY) if((docflags & FLAG) == FLAG){\
        HV_SET_TRUE(docinfo_hv, KEY, ssv);}

#define CERT_PRINT_VAL(FUNC,STRING) tmp = (char *) FUNC(certinfo); \
    sv = newSVpv(tmp, strlen(tmp)); \
    ssv = hv_store(certinfo_hv, STRING, strlen(STRING), sv, 0); \
    if (ssv == NULL) { croak("Bad: key '%s' not stored", STRING); }

MODULE = Crypt::SGMTT           PACKAGE = Crypt::SGMTT     PREFIX = crypt__

void
crypt__Signatur(data_in, hv)
  UBYTE * data_in = NO_INIT
  STRLEN data_in_len = NO_INIT
  HV * hv
PREINIT:
  UBYTE * data_out;
  ULONG data_out_len;
  ULONG header_size;
  ULONG output_size;
  ULONG output_avail;
  ULONG footer_size;
  ULONG runs;
  ULONG i;
  int mtt_debug;
  char * doctype;
  char * packtype;
  char * filename;
  T_DOCINFO docinfo;
  T_ALGOINFO algoinfo;
  T_MTT_ERROR mtt_err;
  T_MTTFLAGS mtt_flags;
  T_CFG mttcfg;
  SV ** ssv;
  SV * sv;
  HV * docinfo_hv;
  char *key;
  I32 len;
PPCODE:
{
  data_in = (UBYTE *) SvPV(ST(0), data_in_len);

  mtt_debug = 0;
  hv_iterinit(hv);
  while ((sv = hv_iternextsv(hv, &key, &len)) != NULL) {
    if ((strcmp(key, "DEBUG") == 0) &&
        (strcmp(SvPV(sv, na), "1") == 0)) {
      mtt_debug = 1;
    }
  }
```

```
// if ((data_out = (UBYTE *) malloc((size_t)(GROW(data_in_len)))) == NULL) {

data_out = New(0, data_out, (size_t)(GROW(data_in_len)), UBYTE);

if (data_out == NULL) {
  if (mtt_debug != 0) {
    PerlIO_printf(PerlIO_stderr(), "Fehler bei data_out = New(GROW(x))\n");
  }
  XPUSHs (&sv_undef);
  XPUSHs (sv_2mortal(newSVpv("MTT_ERR_MEM", 0)));
  XSRETURN(2);
}

mttcfg = new_cfg();
hv_iterinit(hv);
while ((sv = hv_iternextsv(hv, &key, &len)) != NULL) {
  if (strncmp(key, "CFG_", 4) == 0) {
    set_cfg(mttcfg, STR_2_T_MTT_CFG(key), SvPV(sv, na));
  }
}
mtt_initialize(mttcfg);

ssv = hv_fetch(hv, "DOCTYPE", strlen("DOCTYPE"), 0);
if (ssv != NULL) {
  doctype = SvPV( *ssv, na );
} else {
  del_cfg(&mttcfg);
  Safefree(data_out);
  croak("DOCTYPE fehlt");
}

ssv = hv_fetch(hv, "PACKTYPE", strlen("PACKTYPE"), 0);
if (ssv != NULL) {
  packtype = SvPV( *ssv, na );
} else {
  del_cfg(&mttcfg);
  Safefree(data_out);
  croak("PACKTYPE fehlt");
}

ssv = hv_fetch(hv, "FILENAME", strlen("FILENAME"), 0);
if (ssv != NULL) {
  filename = SvPV( *ssv, na );
} else {
  del_cfg(&mttcfg);
  Safefree(data_out);
  croak("FILENAME fehlt");
}

docinfo = new_docinfo_direct(STR_2_T_DOCTYPE(doctype),
                             STR_2_T_PACKTYPE(packtype),
```

```
                              filename);

  set_flag(&mtt_flags, MTT_FLAG_NULL);

  hv_iterinit(hv);
  while ((sv = hv_iternextsv(hv, &key, &len)) != NULL) {
    if (strncmp(key, "MTT_FLAG_", 9) == 0) {
      add_flag(&mtt_flags, STR_2_TFLAG(key));
    }
  }

  hv_iterinit(hv);
  while ((sv = hv_iternextsv(hv, &key, &len)) != NULL) {
    if (strcmp(key, "HASHALGO") == 0) {
      algoinfo.hash_algo = STR_2_T_HASHALGO(SvPV(sv, na));
    } else if (strcmp(key, "SIGALGO") == 0) {
      algoinfo.sig_algo = STR_2_T_SIGALGO(SvPV(sv, na));
    } else if (strcmp(key, "CRYPTALGO") == 0) {
      algoinfo.crypt_algo = STR_2_T_CRYPTALGO(SvPV(sv, na));
    } else if (strcmp(key, "BLOCKALGO") == 0) {
      algoinfo.block_algo = STR_2_T_BLOCKALGO(SvPV(sv, na));
    }
  }

  mtt_err = mtt_sign_init(docinfo, &algoinfo, mtt_flags, &header_size);
  if (mtt_err != MTT_ERR_OK) {
    if (mtt_debug != 0) {
      PerlIO_printf(PerlIO_stderr(), "Return-Code mtt_sign_init(): %s\n",
              T_MTT_ERROR_2_STR(mtt_err));
    }
    del_cfg(&mttcfg);
    Safefree(data_out);
    XPUSHs (&sv_undef);
    XPUSHs (sv_2mortal(newSVpv(T_MTT_ERROR_2_STR(mtt_err), 0)));
    XSRETURN(2);
  }

  runs = data_in_len / MTT_BLOCK_SIZE;
  data_out_len = header_size;
  output_avail = GROW(data_in_len);
  output_size = output_avail;

  for (i=0; i<runs; i++) {
    mtt_err = mtt_sign_update(data_in + ((UBYTE)i)*MTT_BLOCK_SIZE,
                              MTT_BLOCK_SIZE,
                              data_out + data_out_len,
                              &output_size,
                              FALSE);
    if (mtt_err != MTT_ERR_OK) {
      if (mtt_debug != 0) {
        PerlIO_printf(PerlIO_stderr(), "Return-Code mtt_sign_update(): %s\n",
```

```
              T_MTT_ERROR_2_STR(mtt_err));
    }
    del_cfg(&mttcfg);
    Safefree(data_out);
    XPUSHs (&sv_undef);
    XPUSHs (sv_2mortal(newSVpv(T_MTT_ERROR_2_STR(mtt_err), 0)));
    XSRETURN(2);
  }
  data_out_len += output_size;
  output_avail -= output_size;
  output_size = output_avail;
}
mtt_err = mtt_sign_update(data_in + runs*MTT_BLOCK_SIZE,
                          data_in_len % MTT_BLOCK_SIZE,
                          data_out + data_out_len,
                          &output_size,
                          TRUE);
if (mtt_err != MTT_ERR_OK) {
  if (mtt_debug != 0) {
    PerlIO_printf(PerlIO_stderr(), "Return-Code mtt_sign_update(): %s\n",
            T_MTT_ERROR_2_STR(mtt_err));
  }
  del_cfg(&mttcfg);
  Safefree(data_out);
  XPUSHs (&sv_undef);
  XPUSHs (sv_2mortal(newSVpv(T_MTT_ERROR_2_STR(mtt_err), 0)));
  XSRETURN(2);
}
data_out_len += output_size;
output_avail -= output_size;
footer_size = output_avail;

mtt_err = mtt_sign_final(data_out,
                         header_size,
                         data_out + data_out_len,
                         &footer_size);
if (mtt_err != MTT_ERR_OK) {
  if (mtt_debug != 0) {
    PerlIO_printf(PerlIO_stderr(), "Return-Code mtt_sign_final(): %s\n",
            T_MTT_ERROR_2_STR(mtt_err));
  }
  del_cfg(&mttcfg);
  Safefree(data_out);

  XPUSHs (&sv_undef);
  XPUSHs (sv_2mortal(newSVpv(T_MTT_ERROR_2_STR(mtt_err), 0)));
  XSRETURN(2);
}
data_out_len += footer_size;

if (mtt_err == MTT_ERR_OK) {
```

```
    XPUSHs (sv_2mortal(newSVpv((char *) data_out, (int) data_out_len)));
  } else {
    XPUSHs (&sv_undef);
  }

  del_cfg(&mttcfg);
  Safefree(data_out);

  XPUSHs (sv_2mortal(newSVpv(T_MTT_ERROR_2_STR(mtt_err), 0)));
  XSRETURN(2);
}

void
crypt__Scan(data_in, hv)
  UBYTE * data_in = NO_INIT
  STRLEN data_in_len = NO_INIT
  HV * hv
PREINIT:
  UBYTE * data_out;
  ULONG data_out_len;
  ULONG header_size;
  ULONG output_size;
  ULONG output_avail;
  ULONG runs;
  ULONG i;
  int mtt_debug;
  char * doctype;
  char * packtype;
  char * filename;
  T_DOCINFO docinfo;
  T_MTT_ERROR mtt_err;
  T_MTTFLAGS mtt_flags;
  T_CERTINFO certinfo;
  T_DOCFLAGS docflags;
  T_CFG mttcfg;
  SV ** ssv;
  SV * sv;
  HV * docinfo_hv;
  char * tmp;
  char * key;
  I32 len;
  char buf[300];
PPCODE:
{
  data_in = (UBYTE *) SvPV(ST(0), data_in_len);

  mtt_debug = 0;
  hv_iterinit(hv);
  while ((sv = hv_iternextsv(hv, &key, &len)) != NULL) {
    if ((strcmp(key, "DEBUG") == 0) && (strcmp(SvPV(sv, na), "1") == 0)) {
      mtt_debug = 1;
```

```
  }
}

#
# Schon mal den Hash bereitstellen, in dem die verschiedenen Zusatzinformationen
# zurückgeliefert werden können...
#
docinfo_hv = newHV();

data_out = New(0, data_out, (size_t)(GROW(data_in_len)), UBYTE);

if (data_out == NULL) {
  if (mtt_debug != 0) {
    PerlIO_printf(PerlIO_stderr(), "Fehler bei data_out = New(GROW(x))\n");
  }

  sprintf(buf, "New");
  sv = newSVpv(buf, strlen(buf));
  ssv = hv_store(docinfo_hv, HV_ERRORPOINT,
                 strlen(HV_ERRORPOINT), sv, 0);
  if (ssv == NULL) {
    croak("Bad: key '%s' not stored", HV_ERRORPOINT);
  }

  XPUSHs (&sv_undef);
  XPUSHs (sv_2mortal(newSVpv("MTT_ERR_MEM", 0)));
  XPUSHs (sv_2mortal(newRV_noinc((SV *)docinfo_hv)));
  XSRETURN(3);
}

mttcfg = new_cfg();
hv_iterinit(hv);
while ((sv = hv_iternextsv(hv, &key, &len)) != NULL) {
  if (strncmp(key, "CFG_", 4) == 0) {
    set_cfg(mttcfg, STR_2_T_MTT_CFG(key), SvPV(sv, na));
  }
}
mtt_initialize(mttcfg);

docinfo = new_docinfo();
if (docinfo == NULL) {
  if (mtt_debug != 0) {
    PerlIO_printf(PerlIO_stderr(), "Fehler: new_docinfo()\n");
  }

  sprintf(buf, "new_docinfo");
  sv = newSVpv(buf, strlen(buf));
  ssv = hv_store(docinfo_hv, HV_ERRORPOINT,
                 strlen(HV_ERRORPOINT), sv, 0);
  if (ssv == NULL) {
    croak("Bad: key '%s' not stored", HV_ERRORPOINT);
```

```
    }

    del_cfg(&mttcfg);
    Safefree(data_out);
    XPUSHs (&sv_undef);
    XPUSHs (sv_2mortal(newSVpv("MTT_ERR_DOCINFO", 0)));
    XPUSHs (sv_2mortal(newRV_noinc((SV *)docinfo_hv)));
    XSRETURN(3);
  }

  set_flag(&mtt_flags, MTT_FLAG_NULL);
  hv_iterinit(hv);
  while ((sv = hv_iternextsv(hv, &key, &len)) != NULL) {
    if (strncmp(key, "MTT_FLAG_", 9) == 0) {
      add_flag(&mtt_flags, STR_2_TFLAG(key));
    }
  }

  header_size = GROW(data_in_len);

  mtt_err = mtt_scan_header(data_in,
                            data_in_len,
                            &header_size,
                            mtt_flags,
                            docinfo);
  if (mtt_err) {
    if (mtt_debug != 0) {
      PerlIO_printf(PerlIO_stderr(), "Return-Code mtt_scan_header(): %s\n",
                    T_MTT_ERROR_2_STR(mtt_err));
    }

    sprintf(buf, "mtt_scan_header");
    sv = newSVpv(buf, strlen(buf));
    ssv = hv_store(docinfo_hv, HV_ERRORPOINT,
                   strlen(HV_ERRORPOINT), sv, 0);
    if (ssv == NULL) {
      croak("Bad: key '%s' not stored", HV_ERRORPOINT);
    }

    del_cfg(&mttcfg);
    Safefree(data_out);
    XPUSHs (&sv_undef);
    XPUSHs (sv_2mortal(newSVpv(T_MTT_ERROR_2_STR(mtt_err), 0)));
    XPUSHs (sv_2mortal(newRV_noinc((SV *)docinfo_hv)));
    XSRETURN(3);
  }

  mtt_err = mtt_scan_body_init();
  if (mtt_err) {
    if (mtt_debug != 0) {
      PerlIO_printf(PerlIO_stderr(), "Return-Code mtt_scan_body_init(): %s\n",
```

```
                T_MTT_ERROR_2_STR(mtt_err));
  }

  sprintf(buf, "mtt_scan_body_init");
  sv = newSVpv(buf, strlen(buf));
  ssv = hv_store(docinfo_hv, HV_ERRORPOINT,
                 strlen(HV_ERRORPOINT), sv, 0);
  if (ssv == NULL) {
    croak("Bad: key '%s' not stored", HV_ERRORPOINT);
  }

  del_cfg(&mttcfg);
  Safefree(data_out);
  XPUSHs (&sv_undef);
  XPUSHs (sv_2mortal(newSVpv(T_MTT_ERROR_2_STR(mtt_err), 0)));
  XPUSHs (sv_2mortal(newRV_noinc((SV *)docinfo_hv)));
  XSRETURN(3);
}

runs = data_in_len / MTT_BLOCK_SIZE;
data_out_len = 0;
output_avail = GROW(data_in_len);
output_size = output_avail;

for (i=0; i<runs; i++) {
  mtt_err = mtt_scan_body_update(data_in + header_size + i*MTT_BLOCK_SIZE,
                                 MTT_BLOCK_SIZE,
                                 data_out + data_out_len,
                                 &output_size,
                                 FALSE);
  if (mtt_err != MTT_ERR_OK) {
    if (mtt_debug != 0) {
      PerlIO_printf(PerlIO_stderr(), "Return-Code mtt_scan_body_update(): %s\n",
                    T_MTT_ERROR_2_STR(mtt_err));
    }

    sprintf(buf, "mtt_scan_body_update(,,,,FALSE)");
    sv = newSVpv(buf, strlen(buf));
    ssv = hv_store(docinfo_hv, HV_ERRORPOINT,
                   strlen(HV_ERRORPOINT), sv, 0);
    if (ssv == NULL) {
      croak("Bad: key '%s' not stored", HV_ERRORPOINT);
    }

    del_cfg(&mttcfg);
    Safefree(data_out);
    XPUSHs (&sv_undef);
    XPUSHs (sv_2mortal(newSVpv(T_MTT_ERROR_2_STR(mtt_err), 0)));
    XPUSHs (sv_2mortal(newRV_noinc((SV *)docinfo_hv)));
    XSRETURN(3);
  }
```

```
    data_out_len += output_size;
    output_avail -= output_size;
    output_size = output_avail;
  }
  mtt_err = mtt_scan_body_update(data_in + header_size + runs*MTT_BLOCK_SIZE,
                                 data_in_len % MTT_BLOCK_SIZE,
                                 data_out + data_out_len,
                                 &output_size,
                                 TRUE);
  if (mtt_err != MTT_ERR_OK) {
    if (mtt_debug != 0) {
      PerlIO_printf(PerlIO_stderr(), "Return-Code mtt_scan_body_update(): %s\n",
                    T_MTT_ERROR_2_STR(mtt_err));
    }

    sprintf(buf, "mtt_scan_body_update(,,,,TRUE)");
    sv = newSVpv(buf, strlen(buf));
    ssv = hv_store(docinfo_hv, HV_ERRORPOINT,
                   strlen(HV_ERRORPOINT), sv, 0);
    if (ssv == NULL) {
      croak("Bad: key '%s' not stored", HV_ERRORPOINT);
    }

    del_cfg(&mttcfg);
    Safefree(data_out);
    XPUSHs (&sv_undef);
    XPUSHs (sv_2mortal(newSVpv(T_MTT_ERROR_2_STR(mtt_err), 0)));
    XPUSHs (sv_2mortal(newRV_noinc((SV *)docinfo_hv)));
    XSRETURN(3);
  }
  data_out_len += output_size;
  output_avail -= output_size;

  /*
   * Buggy
   *
   */
#ifdef CLEAR_DOCFLAG_MIC_INVALID
  docinfo_clear_docflags(docinfo, DOC_FLAG_MIC_INVALID);
#endif

  #
  # Nachdem der Body-Scan abgeschlossen wird, wird das erste Argument,
  # d.h. die decodierten Plaintext-Daten auf dem Rückgabestack abgelegt.
  #
  mtt_err = mtt_scan_body_final(&docinfo);
  if ((MTT_ERR_OK == mtt_err) ||
      (MTT_ERR_MICVALID == mtt_err) ||
      (MTT_ERR_MICINVALID == mtt_err) ||
      (MTT_ERR_NOTRECIPIENT == mtt_err) ||
      (MTT_ERR_MICVALID_WITH_CERT == mtt_err) ||
```

```
    (MTT_ERR_MICVALID_WITH_PATH == mtt_err)) {
  XPUSHs (sv_2mortal(newSVpv((char *) data_out, (int) data_out_len)));
} else {
  XPUSHs (&sv_undef);
}
del_cfg(&mttcfg);
Safefree(data_out);

/*
 * Daten aus der T_DOCINFO-Struktur extrahieren und
 * zurückliefern...
 *
 */

tmp = (char *) docinfo_get_name(docinfo);
sv = newSVpv(tmp, strlen(tmp));
ssv = hv_store(docinfo_hv, HV_NAME,
               strlen(HV_NAME), sv, 0);
if (ssv == NULL) {
  croak("Bad: key '%s' not stored", HV_NAME);
}

tmp = (char *) T_DOCTYPE_2_STR(docinfo_get_doctype(docinfo));
sv = newSVpv(tmp, strlen(tmp));
ssv = hv_store(docinfo_hv, HV_DOCTYPE,
               strlen(HV_DOCTYPE), sv, 0);
if (ssv == NULL) {
  croak("Bad: key '%s' not stored", HV_DOCTYPE);
}

tmp = (char *) T_PACKTYPE_2_STR(docinfo_get_packtype(docinfo));
sv = newSVpv(tmp, strlen(tmp));
ssv = hv_store(docinfo_hv, HV_PACKTYPE,
               strlen(HV_PACKTYPE), sv, 0);
if (ssv == NULL) {
  croak("Bad: key '%s' not stored", HV_PACKTYPE);
}

tmp = (char *) T_MTT_TYPE_2_STR(docinfo_get_mtttype(docinfo));
sv = newSVpv(tmp, strlen(tmp));
ssv = hv_store(docinfo_hv, HV_MTTTYPE,
               strlen(HV_MTTTYPE), sv, 0);
if (ssv == NULL) {
  croak("Bad: key '%s' not stored", HV_MTTTYPE);
}

certinfo = docinfo_get_certinfo(docinfo);
if (certinfo != NULL) {
  tmp = (char *) certinfo_get_dname(certinfo);
  sv = newSVpv(tmp, strlen(tmp));
  ssv = hv_store(docinfo_hv, HV_CERTDNAME,
```

```
                strlen(HV_CERTDNAME), sv, 0);
    if (ssv == NULL) {
      croak("Bad: key '%s' not stored", HV_CERTDNAME);
    }

    tmp = (char *) certinfo_get_idname(certinfo);
    sv = newSVpv(tmp, strlen(tmp));
    ssv = hv_store(docinfo_hv, HV_CERTIDNAME,
                strlen(HV_CERTIDNAME), sv, 0);
    if (ssv == NULL) {
      croak("Bad: key '%s' not stored", HV_CERTIDNAME);
    }

    tmp = (char *) certinfo_get_serno(certinfo);
    sv = newSVpv(tmp, strlen(tmp));
    ssv = hv_store(docinfo_hv, HV_CERTSERNO,
                strlen(HV_CERTSERNO), sv, 0);
    if (ssv == NULL) {
      croak("Bad: key '%s' not stored", HV_CERTSERNO);
    }
  }

  docflags = docinfo_get_docflags(docinfo);
  MACRO_PROCESS_DOCFLAGS(DOC_FLAG_NO_CERT, "DOC_FLAG_NO_CERT");
  MACRO_PROCESS_DOCFLAGS(DOC_FLAG_CERT_INVALID, "DOC_FLAG_CERT_INVALID");
  MACRO_PROCESS_DOCFLAGS(DOC_FLAG_NOT_RECIPIENT, "DOC_FLAG_NOT_RECIPIENT");
  MACRO_PROCESS_DOCFLAGS(DOC_FLAG_MIC_INVALID, "DOC_FLAG_MIC_INVALID");
  MACRO_PROCESS_DOCFLAGS(DOC_FLAG_INVALID_DOC, "DOC_FLAG_INVALID_DOC");
  MACRO_PROCESS_DOCFLAGS(DOC_FLAG_NO_VERIFY, "DOC_FLAG_NO_VERIFY");
  MACRO_PROCESS_DOCFLAGS(DOC_FLAG_PROCESS_ERR, "DOC_FLAG_PROCESS_ERR");
  MACRO_PROCESS_DOCFLAGS(DOC_FLAG_OK, "DOC_FLAG_OK");
  MACRO_PROCESS_DOCFLAGS(DOC_FLAG_VALIDCERT, "DOC_FLAG_VALIDCERT");
  MACRO_PROCESS_DOCFLAGS(DOC_FLAG_VALIDPATH, "DOC_FLAG_VALIDPATH");

  XPUSHs (sv_2mortal(newSVpv(T_MTT_ERROR_2_STR(mtt_err), 0)));
  XPUSHs (sv_2mortal(newRV_noinc((SV *)docinfo_hv)));
  XSRETURN(3);
}
```

A.1.2. SGMTT/Signature.pm

SGMTT/Signature.pm ist das Perl-Objekt zum einfachen Verwenden der SGMTT-Methoden.

```
package Crypt::SGMTT::Signature;

use Compress::Zlib;
use Crypt::SGMTT;
use Carp;

%Crypt::SGMTT::Signature::PossibleArgs =
(
```

```
 'CFG' => 'Konfigurations-Objekt new Crypt::SGMTT::CFG()',
 'DOCINFO' => 'DOCINFO-Objekt new Crypt::SGMTT::DOCINFO()',
 'ALGOINFO' => 'ALGOINFO-Objekt new Crypt::SGMTT::ALGOINFO()',
 'TFLAG' => 'TFLAG-Objekt new Crypt::SGMTT::TFLAG()',
 'DEBUG' => 'Debugging-Flag (\'1\' oder \'0\')'
);

sub new
{
    my ($pkg, %config) = @_;
    my ($self) = {};

    %{$self->{'config'}} = %config;

    my ($key);
    foreach $key (keys %{$self->{'config'}}) {
        croak "$key ist kein gültiger Parameter"
            unless defined $Crypt::SGMTT::Signature::PossibleArgs{$key};
    }

    unless (defined $self->{'config'}->{'CFG'}) {
        $self->{'config'}->{'CFG'} = new Crypt::SGMTT::CFG();
    }
    unless (defined $self->{'config'}->{'DOCINFO'}) {
        $self->{'config'}->{'DOCINFO'} = new Crypt::SGMTT::DOCINFO();
    }
    unless (defined $self->{'config'}->{'ALGOINFO'}) {
        $self->{'config'}->{'ALGOINFO'} = new Crypt::SGMTT::ALGOINFO();
    }
    unless (defined $self->{'config'}->{'TFLAG'}) {
        $self->{'config'}->{'TFLAG'} = new Crypt::SGMTT::TFLAG();
    }

    %{$self->{'cfg_data'}} = ($self->{'config'}->{'CFG'}->data(),
                              $self->{'config'}->{'DOCINFO'}->data(),
                              $self->{'config'}->{'ALGOINFO'}->data(),
                              $self->{'config'}->{'TFLAG'}->data());

    $self->{'cfg_data'}->{'DEBUG'} = '1' if $self->{'config'}->{'DEBUG'};

#    $self->{'cfg_data'}->{'MTT_FLAG_SHOW_RESULT'} = '1';

    bless $self, $pkg;
    $self;
}

sub DESTROY
{
    my ($self) = shift;
}
```

```
sub sign
{
    my ($self) = shift;
    my (@data) = @_;

    my $tmp;
    while (defined ($tmp = shift)) {
        $self->{'DATA_CLEAR'} .= $tmp;
    }

    unless ($self->{'DATA_CLEAR'}) {
        croak "Es müssen Daten zur Signatur vorhanden sein";
    }

    $self->{'OPERATION'} = 'sign';

    if ($self->{'config'}->{'DOCINFO'}->{'config'}->{'PACKTYPE'} eq 'PA_GZIP') {
        ($self->{'DATA_SIGNED'},
         $self->{'MTT_ERROR'}) = Crypt::SGMTT::Signatur(
                          Compress::Zlib::compress($self->{'DATA_CLEAR'}),
                          $self->{'cfg_data'});
    } else {
        ($self->{'DATA_SIGNED'},
         $self->{'MTT_ERROR'}) = Crypt::SGMTT::Signatur($self->{'DATA_CLEAR'},
                                                        $self->{'cfg_data'});
    }

    $self;
}

sub verify
{
    my ($self) = shift;

    my $tmp;
    while (defined ($tmp = shift)) {
        $self->{'DATA_SIGNED'} .= $tmp;
    }

    unless ($self->{'DATA_SIGNED'}) {
        croak "Es müssen Daten zur Signatur-Verifikation vorhanden sein";
    }

    $self->{'OPERATION'} = 'scan';

    ($self->{'DATA_VERIFIED'},
     $self->{'MTT_ERROR'},
     $self->{'DOCINFO'}) = Crypt::SGMTT::Scan($self->{'DATA_SIGNED'},
                                      $self->{'cfg_data'});

    if ((defined $self->{'DOCINFO'}->{'T_PACKTYPE'}) &&
```

```
        ($self->{'DOCINFO'}->{'T_PACKTYPE'} eq 'PA_GZIP')) {
        $self->{'DATA_VERIFIED'} = Compress::Zlib::uncompress($self->{'DATA_VERIFIED'});
    }

    $self;
}

sub scan
{
    my ($self) = shift;

    $self->verify(@_);
}

sub as_string
{
    my ($self) = shift;

    my ($retval);
    if ($self->{'OPERATION'} eq 'sign') {
        $retval = $self->{'DATA_SIGNED'};
    } elsif ($self->{'OPERATION'} eq 'scan') {
        $retval = $self->{'DATA_VERIFIED'};
    } else {
        $retval = undef;
    }

    $retval;
}

sub sig_as_string
{
    my ($self) = shift;

    $self->{'DATA_SIGNED'};
}

sub clear_as_string
{
    my ($self) = shift;

    my ($retval);
    if ($self->{'OPERATION'} eq 'sign') {
        $retval = $self->{'DATA_CLEAR'};
    } elsif ($self->{'OPERATION'} eq 'scan') {
        $retval = $self->{'DATA_VERIFIED'};
    } else {
        $retval = undef;
    }

    $retval;
```

```
}

sub error_as_string
{
    my ($self) = shift;

    $Crypt::SGMTT::T_MTT_ERROR{$self->{'MTT_ERROR'}};
}

sub error
{
    my ($self) = shift;

    $self->{'MTT_ERROR'};
}

sub error_func
{
    my ($self) = shift;

    if ($self->{'OPERATION'} eq 'sign') {
        return undef;
    } elsif ($self->{'OPERATION'} eq 'scan') {
        return $self->{'DOCINFO'}->{'Fehler-Funktion'};
    } else {
        return undef;
    }
}

sub sig_ok
{
    my ($self) = shift;

    (($self->{'MTT_ERROR'} eq 'MTT_ERR_OK') ||
     ($self->{'MTT_ERROR'} eq 'MTT_ERR_MICVALID') ||
     ($self->{'MTT_ERROR'} eq 'MTT_ERR_MICINVALID') ||
     ($self->{'MTT_ERROR'} eq 'MTT_ERR_NOTRECIPIENT') ||
     ($self->{'MTT_ERROR'} eq 'MTT_ERR_MICVALID_WITH_CERT') ||
     ($self->{'MTT_ERROR'} eq 'MTT_ERR_MICVALID_WITH_PATH'));
}

sub ok
{
    my ($self) = shift;

    ($self->{'MTT_ERROR'} eq 'MTT_ERR_OK');
}

sub AUTOLOAD
{
    my ($self) = shift;
```

```
    my ($pkg) = ref($self) || croak "$self ist kein Objekt";
    my ($name) = $AUTOLOAD;

    $name =~ s/.*:://;

    if ($self->{'OPERATION'} eq 'sign') {
        return "NOT_IMPLEMENTED_IN_AUTOLOAD";
    } elsif ($self->{'OPERATION'} eq 'scan') {
        unless (exists $self->{'DOCINFO'}->{$name}) {
            return undef;
        }

        if (@_) {
            return $self->{'DOCINFO'}->{$name} = shift;
        } else {
            return $self->{'DOCINFO'}->{$name};
        }
    } else {
        return undef;
    }
}

1;

__END__
```

A.2. Crypt::SSP::Data

Crypt::SSP::Data ist das Perl-Objekt zum Erzeugen und Verifizieren der SSP-Objekte.

```
package Crypt::SSP::Data;

use Carp;
use MIME::Base64 ();
use HTTP::Date;
use Crypt::RIPEMD160;

use strict;
use vars qw($VERSION @ISA @EXPORT @EXPORT_OK);

require Exporter;
require AutoLoader;

@ISA = qw(Exporter AutoLoader);
# Items to export into callers namespace by default. Note: do not export
# names by default without a very good reason. Use EXPORT_OK instead.
# Do not simply export all your public functions/methods/constants.
@EXPORT = qw();

$VERSION = '0.01';
```

```
# Preloaded methods go here.

# Autoload methods go after =cut, and are processed by the autosplit program.

use HTTP::Date;

use vars qw($TRANSLATE_UNDERSCORE);

my @signature_subject = ('Existence');

my @header_order = qw(URL
                      Content-Type
                      Content-Length
                      Signer-Distinguished-Name
                      Content-Hash
                      Hash-Algo
                      Date
                      Valid-Not-Before
                      Valid-Not-After
                      Signature-Subject
                      Policy-URL
                      Reference-From-Other-Site-Allowed
                      Comment
                      );

sub url                 {(shift->_header('URL',                        @_))[0] }
sub s_name              {(shift->_header('Signer-Distinguished-Name',  @_))[0] }
sub hash                {(shift->_header('Content-Hash',               @_))[0] }
sub hash_algo           {(shift->_header('Hash-Algo',                  @_))[0] }
sub date                { shift->_date_header('Date',                  @_); }
sub valid_from          { shift->_date_header('Valid-Not-Before',      @_); }
sub valid_till          { shift->_date_header('Valid-Not-After',       @_); }
sub subject             {(shift->_header('Signature-Subject',          @_))[0] }
sub reference_allowed   {(shift->_header('Reference-Allowed',          @_))[0] }

my %header_order;
my %standard_case;

{
    my $i = 0;
    for (@header_order) {
        my $lc = lc $_;
        $header_order{$lc} = $i++;
        $standard_case{$lc} = $_;
    }

    $TRANSLATE_UNDERSCORE = 1 unless defined $TRANSLATE_UNDERSCORE;
}

sub new
```

```
{
    my($class) = shift;

    my $self = bless {
        '_header'   => { },
    }, $class;
    $self->header(@_); # set up initial headers

    return undef unless defined $self->_header('Data');

    $self->push_header('Hash-Algo' => 'RIPEMD160');
    $self->push_header('Content-Hash' => Crypt::RIPEMD160->hexhash($self->_header('Data')));
    $self->remove_header('Data');

    $self->push_header('Date' => HTTP::Date::time2str(time));

    $self;
}

sub keys
{
    my $self = shift;

    keys %{$self->{'_header'}};
}

sub header
{
    my $self = shift;

    my($field, $val, @old);

    while (($field, $val) = splice(@_, 0, 2)) {
        @old = $self->_header($field, $val);
    }

    wantarray ? @old : $old[0];
}

sub _header
{
    my($self, $field, $val, $push) = @_;

    $field =~ tr/_/-/ if $TRANSLATE_UNDERSCORE;

    # $push is only used interally sub push_header

    Carp::croak('Need a field name') unless defined $field;
    Carp::croak('Too many parameters') if @_ > 4;

    my $lc_field = lc $field;
```

```
    unless(defined $standard_case{$lc_field}) {
        # generate a %stadard_case entry for this field
        $field =~ s/\b(\w)/\u$1/g;
        $standard_case{$lc_field} = $field;
    }

    my $this_header = \@{$self->{'_header'}{$lc_field}};

    my @old = ();
    if (!$push && defined $this_header) {
        @old = @$this_header;  # save it so we can return it
    }
    if (defined $val) {
        @$this_header = () unless $push;
        if (!ref($val)) {
            # scalar: create list with single value
            push(@$this_header, $val);
        } elsif (ref($val) eq 'ARRAY') {
            # list: copy list
            push(@$this_header, @$val);
        } else {
            Carp::croak("Unexpected field value $val");
        }
    }
    @old;
}

sub _header_cmp
{
    # Unknown headers are assign a large value so that they are
    # sorted last.  This also helps avoiding a warning from -w
    # about comparing undefined values.
    $header_order{$a} = 999 unless defined $header_order{$a};
    $header_order{$b} = 999 unless defined $header_order{$b};

    $header_order{$a} <=> $header_order{$b} || $a cmp $b;
}

sub scan
{
    my($self, $sub) = @_;
    my $field;
    foreach $field (sort _header_cmp keys %{$self->{'_header'}} ) {
        my $list = $self->{'_header'}{$field};
        if (defined $list) {
            my $val;
            for $val (@$list) {
                &$sub($standard_case{$field} || $field, $val);
            }
        }
    }
```

```
}

sub as_string
{
    my($self, $endl) = @_;
    $endl = "\n" unless defined $endl;

    my @result = ();
    $self->scan(sub {
        my($field, $val) = @_;
        if ($val =~ /\n/) {
            # must handle header values with embedded newlines with care
            $val =~ s/\s+$//;          # trailing newlines and space must go
            $val =~ s/\n\n+/\n/g;      # no empty lines
            $val =~ s/\n([^\040\t])/\n $1/g;  # intial space for continuation
            $val =~ s/\n/$endl/g;      # substitute with requested line ending
        }
        #my $val = MIME::Base64->encode_base64($val);
        push(@result, "$field: $val");
    });

    join($endl, @result, '');
}

sub clone
{
    my $self = shift;
    my $clone = new ref($self);
    $self->scan(sub { $clone->push_header(@_);} );
    $clone;
}

sub push_header
{
    Carp::croak('Usage: $h->push_header($field, $val)') if @_ != 3;
    shift->_header(@_, 'PUSH');
}

sub remove_header
{
    my($self, @fields) = @_;
    my $field;
    foreach $field (@fields) {
        $field =~ tr/_/-/ if $TRANSLATE_UNDERSCORE;
        delete $self->{'_header'}{lc $field};
    }
}

sub _date_header
{
    require HTTP::Date;
```

```
    my($self, $header, $time) = @_;
    my($old) = $self->_header($header);
    if (defined $time) {
        $self->_header($header, HTTP::Date::time2str($time));
    }
    HTTP::Date::str2time($old);
}

sub time_valid
{
    my $self = shift;
    my $now = shift || time;

    my $start = $self->header('Valid-Not-Before');
    my $end = $self->header('Valid-Not-After');

    $start = HTTP::Date::str2time($start) if defined $start;
    $end = HTTP::Date::str2time($end) if defined $end;

    if ((defined $start) && (defined $end)) {
        return (($start <= $now) && ($now <= $end));
    } elsif (defined $start) {
        return ($start <= $now);
    } elsif (defined $end) {
        return ($now <= $end);
    } else {
        return (1);
    }
}

sub verify
{
    my ($self) = shift;
    my %data;

    if ((scalar @_) == 1) {
        $data{'Data'} = shift;

        if ((defined $self->header('Hash-Algo')) &&
            ($self->header('Hash-Algo') eq 'RIPEMD160') &&
            ($self->header('Url') eq $data{'Url'})) {

            return ($self->header('Content-Hash') eq
                  Crypt::RIPEMD160->hexhash($data{'Data'}));
        }
    } else {
        %data = @_;
        croak "Key 'Data' fehlt" unless defined $data{'Data'};

        if ((defined $self->header('Hash-Algo')) &&
            ($self->header('Hash-Algo') eq 'RIPEMD160')) {
```

```
            if (defined $data{'Url'}) {
                return (($self->header('Url') eq $data{'Url'}) &&
                        ($self->header('Content-Hash') eq
                       Crypt::RIPEMD160->hexhash($data{'Data'})));
            } else {
                return ($self->header('Content-Hash') eq
                       Crypt::RIPEMD160->hexhash($data{'Data'}));
            }
        }
    }
    undef;
}

sub is_ssp
{
    my ($self) = shift;

    ((defined $self->header('Date')) &&
     (defined $self->header('Content-Hash')) &&
     (defined $self->header('Hash-Algo')));
}

sub verify_data
{
    my ($self) = shift;
    my ($data) = shift;

    if ((defined $self->header('Hash-Algo')) &&
        ($self->header('Hash-Algo') eq 'RIPEMD160')) {
        return ($self->header('Content-Hash') eq Crypt::RIPEMD160->hexhash($data));
    }
    undef;
}

sub verify_url
{
    my ($self) = shift;
    my ($url) = shift;

    if (defined $self->header('Url')) {
        return ($self->header('Url') eq $url);
    }
    undef;
}

sub AUTOLOAD
{
    no strict;
    my ($self) = shift;
    my ($pkg) = ref($self) || die "$self ist kein Objekt beim Aufruf von AUTOLOAD $AUTOLOAD";
    my ($name) = $AUTOLOAD;
```

```
    return if $name =~ m/::DESTROY$/;

    $name =~ s/.*::// ; # Package-Namen entfernen
    $name = lc($name);
    $name =~ s/_/-/g;
    $name =~ s/\b(\w)/\u$1/g;

    $self->header($name);
}

sub new_from_dat
{
    my($class) = shift;
    my($dat) = shift;

    my $self = bless {
        '_header'   => { },
        '_dat' => $dat
    }, $class;

    my %a;
    while ($dat =~ s/\s*([^:\s]+)\s*:\s*([^\012\015]*)\015?\012//) {
        my ($key, $val) = ($1, $2);

        if (not exists $a{$key}) {
            $a{$key} = $val;
        } elsif (!ref($a{$key})) {
            my @q = ($a{$key}, $val);

            $a{$key} = \@q;
        } elsif (ref($a{$key}) eq 'ARRAY') {
            push(@{$a{$key}}, $val);
        } else {
            die "komische Referenz: ".ref($a{$key});
        }
    }

    $self->header(%a); # set up initial headers
    $self;
}

1;

__END__
```

A.3. Die Signaturerzeugung: `sign.pl`

Hier wird ein kleines Beispiel-Signatur-Programm beschrieben, das aus der UNIX-Kommandozeile aufgerufen wird.

```
#!/usr/bin/perl -w

use strict;
use Getopt::Long;

use Time::Local;
use HTTP::Date;

use Crypt::SSP::Data;
use Crypt::SGMTT;

my (%sig_scheme_ext) = ('Mtt-V1' => 'mtt',
                        'Pgp-Rsa' => 'pgp2',
                        'Pgp-Dh' => 'pgp5',
                        'Pgp' => 'pgp',
                        'Sig-Simul' => 'simul',
                        'Sig' => 'sig',
                        'Ssleay' => 'ssl');

my ($PIN, # Die PIN muß angegeben werden
    $URL, # Der URL muß angegeben werden
    $EMAIL, # Die e-mail-Adresse kann angegeben werden
    $VALID_FROM, # Das Gültigkeitsintervall
    $VALID_NOT_AFTER,
    $DEBUG);

my $data;

######################
#
# Kommandozeile zerrupfen
#
######################

Getopt::Long::GetOptions("x|PIN=s" =>   \$PIN,
                         "u|URL=s" =>   \$URL,
                         "m|EMAIL=s" => \$EMAIL,
                         "f|FROM=s" =>  \$VALID_FROM,
                         "t|TO=s" =>    \$VALID_NOT_AFTER,
                         "d|DEBUG" =>   \$DEBUG,
                         );

sub usage
{
    print STDERR "Usage: $0 -PIN \"pin\" -URL \"url\"\n";
    print STDERR "\n";
    print STDERR "Optional:\n";
    print STDERR "       -EMAIL e-mail-adress\n";
    print STDERR "       -FROM  dd.mm.yyyy (GMT)\n";
    print STDERR "       -TO    dd.mm.yyyy (GMT)\n";
    print STDERR "\n";
```

```
    print STDERR "       -DEBUG\n";
    print STDERR "\n";

    exit(1);
}

usage unless defined $PIN;
usage unless defined $URL;

######################
#
# Dateinamen bestimmen
#
######################

my $filename = $URL; $filename =~ s/.*\///;

if (-f $filename) {
    local $/ = undef;

    open(FILE, "<".$filename) || die("Kann $filename nicht öffnen");
    $data = <FILE>;
    close(FILE) || die("Kann $filename nicht schließen");

    if ($DEBUG) {
        print STDERR "Signiere $filename\n";
    }
} else {
    die ("Liegt die Datei $filename auch im aktuellen Verzeichnis?");
}

######################
#
# SSP-Objekt erzeugen
#
######################

my $ssp_data = new Crypt::SSP::Data('URL' => $URL,
                                    'Data' => $data,
                                    'Signer-Email' => $EMAIL,
                                    );

if (defined $VALID_FROM) {
    my ($day, $mon, $year) = split(/\./, $VALID_FROM);
    $VALID_FROM = HTTP::Date::time2str(Time::Local::timegm(0,0,0,$day,$mon-1,$year));
    $ssp_data->push_header('Valid-Not-Before', $VALID_FROM);
}

if (defined $VALID_NOT_AFTER) {
    my ($day, $mon, $year) = split(/\./, $VALID_NOT_AFTER);
    $VALID_NOT_AFTER = HTTP::Date::time2str(Time::Local::timegm(59,59,23,$day,$mon-1,$year));
```

```
    $ssp_data->push_header('Valid-Not-After', $VALID_NOT_AFTER);
}

######################
#
# Signatur erzeugen und speichern
#
######################

my $sgmtt = new Crypt::SGMTT::Signature('CFG' => new Crypt::SGMTT::CFG('CFG_PIN' => $PIN));

$sgmtt->sign($ssp_data->as_string());

if ($sgmtt->error eq 'MTT_ERR_OK') {
    $filename .= "." . $sig_scheme_ext{'Mtt-V1'};
    open(FILE, ">".$filename) || die("Kann $filename nicht schreiben");
    print FILE $sgmtt->as_string();
    close(FILE);

    if ($DEBUG) {
        print STDERR "Signatur nach $filename geschrieben\n";
    }
} elsif ($sgmtt->error eq 'MTT_ERR_PSEINIT') {
    print STDERR "\nFehler: \n";
    print STDERR "Die PIN scheint nicht zu stimmen. Bitte mit der Option -PIN ";
    print STDERR "die korrekte PIN angeben\n\n";
    exit(1);
} else {
    print STDERR "\nFehler: \n";
    print STDERR $sgmtt->error(), ": ", $sgmtt->error_as_string(), "\n";
    exit(1);
}

if ($DEBUG) {
    print STDERR "Signaturinhalt: \n";
    print STDERR "#" x 80, "\n";
    print STDERR $ssp_data->as_string;
    print STDERR "#" x 80, "\n\n\n";
}
```

A.4. Der Server

A.4.1. `Apache::AppendSig`

Das `mod_perl`-Modul für den Apache-Server, das die Signaturen liefert.

```
package Apache::AppendSig;

use strict;
use Apache::Constants ':common';
```

```
use MIME::Base64;

#
# Client-Request-Header
#
my ($request_header_name)      = 'X-Signature-Request';
#my ($response_header_template) = 'X-Signature-Response-SCHEME-%.2d';
my ($response_header_template) = 'X-Signature-Response-SCHEME-%d';
my ($response_types_avail)     = 'X-Signature-Types-Avail';
my ($response_error)           = 'X-Signature-Error';

#
# MIME-Typen für "pure" Requests
#
my ($content_sig_template)     = 'application/signature-content-sig-SCHEME';
my ($content_cert_template)    = 'application/signature-certificate-SCHEME';

#
# Fehlermeldungen
#
my $request_error_1 = 'No known signature-types available';
my $request_error_2 = 'No known signature-types found';
my $request_error_3 = 'No known scheme requested / Scheme unknown';
my $request_error_4 = 'Requested scheme not available';
my $request_error_5 = 'Requested schemes not available';


#
# Translation von Signatur-Schema auf Datei-Extension
#
my (%sig_scheme_ext) = ('Mtt-V1' => 'mtt',
                        'Pgp-Rsa' => 'pgp2',
                        'Pgp-Dh' => 'pgp5',
                        'Pgp' => 'pgp',
                        'Sig-Simul' => 'simul',
                        'Sig' => 'sig',
                        'Ssleay' => 'ssl');

sub handler {
    my $r = shift;

    my $scheme; # Das Request-Schema
    my $schema; # Laufvariable
    my $type;   # 'implicit' oder 'explicit';

    my $request_schemata;
    my @request_schemata;

    my $sig_filename;
    my $data_filename;
```

```
my $uid_sig; # Besitzer der Signatur-Datei
my $uid_dat; # Besitzer der Daten-Datei

my $avail; # Zurückzuliefernde Auflistung der
           # verfügbaren Signaturen

#
# bestimme absoluten Dateinamen des Requests
#
my $filename = $r->filename;

# return DECLINED if ($filename =~ /cgi-bin/);

#
# Wurde eine Signatur expizit oder implizit verlangt?
#
my ($q);
foreach $scheme (keys %sig_scheme_ext) {
    if ($filename =~ /\.$sig_scheme_ext{$scheme}$/) {
        #
        # $filename zeigt auf eine Signatur des Typs $scheme
        #
        $data_filename = $filename;
        $data_filename =~ s/\.$sig_scheme_ext{$scheme}$//;

        $type = 'explicit';

        $q = $scheme;

        last;
    }
}
($type eq 'explicit') or $type = 'implicit';
$scheme = $q;

if ($type eq 'implicit') {
    if (-f $filename) {
        my $request_schemata = $r->header_in($request_header_name);

        $uid_dat = (stat($filename))[4];

        #
        # Was für Schemata können wir überhaupt liefern?
        #
        $avail = "";
        foreach $schema (keys %sig_scheme_ext) {
            $sig_filename = "$filename.$sig_scheme_ext{$schema}";
            $uid_sig = (stat($sig_filename))[4];

            if (-f $sig_filename && ($uid_sig == $uid_dat)) {
                $avail .= $schema . " ";
```

```
    }
}
if ($avail) {
    $r->header_out($response_types_avail, $avail);
} else {
    $r->header_out($response_error, $request_error_1);
}

return DECLINED unless defined $request_schemata;

$request_schemata =~ s/^\s*//;
@request_schemata = split(/\s+/, $request_schemata);

foreach $schema (@request_schemata) {
    next unless defined $sig_scheme_ext{$schema};

    #
    # Wurde eine normale Datei verlangt, für die eine Signatur
    # vorliegt? Wenn die Signatur existiert und den gleichen
    # Owner wie die Daten hat, wird sie mitgeschickt.
    #
    $sig_filename = "$filename.$sig_scheme_ext{$schema}";
    $uid_sig = (stat($sig_filename))[4];

    if (-f $sig_filename && ($uid_sig == $uid_dat)) {
        #
        # falls das öffnen der Signatur fehlschlägt,
        # wird die Lieferung einfach übergangen...
        #
        open(SIG,$sig_filename) || return DECLINED;
        flock(SIG,2);

        #
        # Die ganze Signatur auf einmal 'reinschlürfen
        #
        undef $/;
        my($sig) = <SIG>;
        flock(SIG,8);
        close(SIG);

        #
        # Die Daten nach Base-64 umwandeln
        #
        my @sig = split(/\n/, MIME::Base64::encode_base64($sig));
        my $i = 0;

        my $response_header_name = $response_header_template;
        $response_header_name =~ s/SCHEME/$schema/;
        foreach $sig (@sig) {
            my($head) = sprintf($response_header_name, $i++);
```

```
                $r->header_out($head, $sig);
            }
            #
            # Jetzt ist die Signatur im Header
            #
            return DECLINED;
        }
    }
    if (scalar @request_schemata == 0) {
        $r->header_out($response_error, $request_error_3);
    } elsif (scalar @request_schemata == 1) {
        $r->header_out($response_error, $request_error_4);
    } else {
        $r->header_out($response_error, $request_error_5);
    }

    return DECLINED;
} else {
    #
    # Die angeforderte Datei existiert nicht
    #
    # Aber vielleicht wurde nur eine Signaturdatei mit  unbekanntem
    # Signatur-Schema verlangt, d.h., die Datendatei _und_ mehrere
    # andere (gültige) Schemata sind verfügbar.
    #
    # Falls also die Datei ohne Extension existiert _und_ gültige
    # Schemata existieren, wird NOT_FOUND sowie der X-Signature-Types-
    # Avail-Header geliefert.
    #

    #
    # Aufgrund des 'greedy' (gierig) -Matching frißt $ext
    # alles nach dem letzten Punkt.
    #
    my ($data_filename, $ext) = ($filename =~ /^(.*)\.(.*)/);

    if (-f $data_filename) {
        $uid_dat = (stat($data_filename))[4];

        #
        # Was für Schemata können wir überhaupt liefern?
        #
        $avail = "";
        foreach $schema (keys %sig_scheme_ext) {
            $sig_filename = "$data_filename.$sig_scheme_ext{$schema}";
            $uid_sig = (stat($sig_filename))[4];

            if (-f $sig_filename && ($uid_sig == $uid_dat)) {
                $avail .= $schema . " ";
            }
        }
```

```
            if ($avail) {
                $r->header_out($response_types_avail, $avail);
            } else {
                $r->header_out($response_error, $request_error_2);
            }
        }

        #
        #
        # Ab hier wird das normale Verhalten des Apache wiederholt, das bei
        # 'return NOT_FOUND;' gefahren wird.
        # Aber jetzt sind evtl. die 'X-Signature-xxx'-Header dabei.
        #
        $r->status_line('404 File not found');

        $r->content_type('text/html');

        $r->send_http_header;

        my $root = $r->document_root;
        $filename =~ s/^$root//;

        $r->print("<HTML><HEAD>\n",
                  "<TITLE>404 File Not Found</TITLE>\n",
                  "</HEAD><BODY>\n",
                  "<H1>File Not Found</H1>\n",
                  "The requested URL $filename was not found on this server.<P>\n",
                  "</BODY></HTML>\n");

        return OK;
    }
} else { # explicit
    #
    # Damit die Signatur separat geliefert werden kann,
    # müssen die Signatur und Daten den gleichen
    # Datei-Owner haben und die Daten müssen existieren.
    #
    # in $scheme steht noch das geforderte Schema

    $sig_filename = $filename;

    $data_filename = $filename;
    $data_filename =~ s/\.$sig_scheme_ext{$scheme}$//;

    if (! -f $data_filename) {
        return NOT_FOUND;
    } else {
        my($uid_dat) = (stat($data_filename))[4];

        if (-f $sig_filename) { # Signaturdatei
            my($uid_sig) = (stat($sig_filename))[4];
```

```
                if ($uid_sig == $uid_dat) {
                    $content_sig_template =~ s/SCHEME/$scheme/;

                    $r->content_type($content_sig_template);

                    #
                    # Datei-Auslieferung dem Server überlassen
                    #
                    return DECLINED;
                }
            }

            #
            # Wenn das gewünschte Schema nicht vorhanden ist,
            # können wir vielleicht mit was anderem dienen ?
            #
            $avail = "";
            foreach $schema (keys %sig_scheme_ext) {
                my $sig_filename_alt = "$data_filename.$sig_scheme_ext{$schema}";
                my $uid_sig = (stat($sig_filename_alt))[4];

                if (-f $sig_filename_alt && ($uid_sig == $uid_dat)) {
                    $avail .= $scheme . " ";
                }
            }
            if ($avail) {
                $r->header_out($response_types_avail, $avail);
            } else {
                $r->header_out($response_error, $request_error_2);
            }
        }
        return DECLINED;
    }
    # Kann _nie_ erreicht werden, da ein Request entweder implizit oder
    # explizit gekennzeichnet wird.

    return DECLINED;
}

1;

__END__
```

A.4.2. `httpd.conf`

Der Eintrag für die Apache-Konfiguration, um das Modul im Apache einzubinden.

```
#
# httpd.conf
```

```
#
# Kann _nicht_ auf <Location *> angewendet werden, weil der
# Output von CGI oder SSI nicht verarbeitet werden kann.
# Erst ab Apache_2.x möglich
#
<Files *>
SetHandler  perl-script
PerlHandler Apache::AppendSig
</Files>
```

A.5. Der Proxy

A.5.1. `proxy.pl`

Der Signatur-Proxy und die grafische Benutzeroberfläche werden hier spezifiziert.

```
#!/usr/bin/perl -w

######################
#
# Bibliotheken
#
######################

use strict;          # Strikte Prüfung von Variablen-Identifiern, etc.

no strict 'subs';

require 5.00404;     # Mindestens Perl v5.004 Patchlevel 04 verlangen

use LWP::UserAgent;  # Stellt den Useragent zum WWW-Zugriff dar;
use HTTP::Request;   # Stellt das HTTP-Request-Objekt zur Verfügung
use HTTP::Response;  # Stellt das HTTP-Response-Objekt zur Verfügung


use URI::URL;        # Stellt die Umformung und Extraktion von URL's zur Verfügung
use MIME::Base64;    # Base64-Encoding und Decoding

use IO::Socket;      # Wrapper für Unix-Internet-Sockets
use IO::Select;      # Wrapper für die select()-Funktion

require "Stored.pm"; # tie-Bindung zum einfachen Speichern der Signaturen und Daten

#require "ipc.ph";
#require "msg.ph";

######################
#
# Globale Variablen
#
```

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

my $debug = 0;

my $IPC_KEY   = 4712;       # share among other processes
my $IPC_CREAT = 0001000;

my $use_proxy = 0;
my $proxy = 'http://proxy.rp-plus.de:8080/';
my $local_proxy_port = 8080;

my $timeout = 60;
my $INEThost = "localhost";
my $INETport = 80;
my $CRLF = "\015\012";

my %WORKER_PIDS; # Enthält die PID's der noch aktiven Worker

my @preferred_sig_schemes = qw/Mtt-V1 Pgp-Rsa/;
my $sig_directory = "/.signature-cache";
my (%sig_scheme_ext) = ('Mtt-V1' => 'mtt',
                        'Pgp-Rsa' => 'pgp2',
                        'Pgp-Dh' => 'pgp5',
                        'Pgp' => 'pgp',
                        'Sig-Simul' => 'simul',
                        'Sig' => 'sig',
                        'Ssleay' => 'ssl');

my ($request_header_name)      = 'X-Signature-Request';
my ($response_header_template) = 'X-Signature-Response-SCHEME-%.2d';
my ($response_types_avail)     = 'X-Signature-Types-Avail';
my ($response_error)           = 'X-Signature-Error';

my ($content_sig_template)     = 'application/signature-content-sig-SCHEME';
my ($content_cert_template)    = 'application/signature-certificate-SCHEME';

my ($BAD_LOGIN_HOST) = ("HTTP/1.0 500 Connects only from localhost\015\012".
                        "Content-Type: text/html\015\012".
                        "\015\012".
                        "<HTML><HEAD><TITLE>Bad login host</TITLE></HEAD><BODY>\n".
                        "<H1>Fehler:</H1>Connects sind nur von <B>localhost</B> erlaubt\n\n</BODY>");

use vars qw/ $main $menu $fmenu $f2 $f1 $lb $lb_hscroll $lb_vscroll/;

my $TO_GUI = 1;              # msgtyp (Priorität)
my $IPC_NOWAIT = "00004000"; # bei msgrcv() nicht blockieren
my $msg_rcv_timeout = 1; # Einmal pro Sekunde die Message-Queue abfragen

#####################
#
# Programm-Code
```

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

main();
exit();

######################
#
# Haupt-Prozess
#
######################

sub main
{
    #
    # Wird ein Internet-Proxy verwendet?
    #
    my $proxy_url;
    if ($use_proxy) {
        $proxy_url = new URI::URL($proxy);
    } else {
        undef $proxy_url;
    }

    #
    # Unix-Message-Queue einrichten
    #
    my $key;
    $key = msgget($IPC_KEY, $IPC_CREAT | 0600)
        or die "msgget(\$IPC_KEY, \$IPC_CREAT | 0600): $!\n";

    #
    # Empfangsdame erzeugen und Tk-Client initialisieren
    #
    my ($pid_of_empfangsdame);
    if (!defined($pid_of_empfangsdame = fork())) {
        die "fork() fehlgeschlagen: $!\n";
    } elsif ($pid_of_empfangsdame == 0) {
        # Child
        empfangsdame($proxy_url);
    } else {
        # Parent (Tk-GUI-Oberfläche)
        start_tk($pid_of_empfangsdame);
    }
}

######################
#
# Empfangsdamen-Code
#
######################
```

```
sub empfangsdame
{
    my ($proxy_url) = shift;

    print "empfangsdame()\n" if $debug;
    $0 .= '-Empfangsdame';

    $SIG{'KILL'} = \&SIG_empfangsdame_soll_sterben;
    $SIG{'CHLD'} = \&SIG_worker_gestorben;

    my($socket_80) = new IO::Socket::INET('LocalHost' => 'localhost',
                                          'LocalPort' => $local_proxy_port,
                                          'Proto' => 'tcp',
                                          'Listen' => 5,
                                          'Reuse' => 1);

    die "IO::Socket::INET() failed: $!" unless $socket_80;

    my ($new_socket);
    my ($pid_of_worker);
    while ($new_socket = $socket_80->accept()) {
        if (!defined($pid_of_worker = fork())) {
            die "fork() failed: $!\n";
        } elsif ($pid_of_worker > 0) {
            # Parent
            $WORKER_PIDS{$pid_of_worker} = "1";
            close($new_socket);
            print "Erzeuge Worker $pid_of_worker\n" if $debug;
        } else {
            #Child
            worker($new_socket, $proxy_url);
        }
    }
    die "accept() failed: $!";
}

sub SIG_worker_gestorben
{
    my $killed = wait();
    delete $WORKER_PIDS{$killed};
    print "SIG_worker_gestorben: $killed\n" if $debug;
}

sub SIG_empfangsdame_soll_sterben
{
    print "SIG_empfangsdame_soll_sterben()\n" if $debug;
    local $SIG{'KILL'} = 'IGNORE';
    local $SIG{'CHLD'} = 'IGNORE';

    my ($worker_pid);
```

```
    foreach $worker_pid (keys %WORKER_PIDS) {
        print "Töte Worker $worker_pid\n" if $debug;
        kill('USR1', $worker_pid);
        my ($killed) = wait();
        if ($killed == $worker_pid) {
            print("OK $killed\n");
        } else {
            print "Not OK\n";
        }
    }
    exit(0);
}

######################
#
# Worker-Code
#
######################

sub worker
{
    my ($client_socket) = shift;
    my ($proxy_url) = shift;

    #
    # verbindungen nur von localhost zulassen;
    #
    if ($client_socket->peerhost() !~ /127.0.0.1/) {
        $client_socket->syswrite($BAD_LOGIN_HOST, length($BAD_LOGIN_HOST));
        $client_socket->close();
        exit(0);
    }

    $SIG{'USR1'} = \&SIG_worker_soll_sterben;
    $0 .= '-Worker';

    my $server_socket;
    my $socket;

    my $size ||= 4096;
    my $url_obj;

    my $request = new HTTP::Request;
    my $response;

    my $error_handles = IO::Select->new($client_socket);
    my $client_handles = IO::Select->new($client_socket);
    my $server_handles = IO::Select->new();

    my $neu_lesbare;
    my $neu_schreibbare;
```

```
  my $fehlerhafte;

  my $cr_buf = ""; # Vom Client empfangene Daten  (client receive)
  my $ss_buf = ""; # Zum Server zu sendende Daten (server send)
  my $sr_buf = ""; # Vom Server empfangene Daten  (server reveive)
  my $cs_buf = ""; # Zum Client zu sendende Daten (client send)

  my ($host, $port, $fullpath);
  my ($method, $url, $proto);
  my ($ver, $code, $msg);
  my $read_server;

  my $request_line;
  my $response_line;

  my $content_length;
  my $te;
  my $ct;
  my $len;

READ_CLIENT_HEADER:
  while (1) {
      if (defined $client_socket) {
          die "read_timeout" if $timeout && !$client_handles->can_read($timeout);
          my $n = $client_socket->sysread($cr_buf, $size, length($cr_buf));
          die $! unless defined ($n);
          die "unexpected EOF before Request-Line seen" unless $n;
      }
      if ($cr_buf =~ /^\w+[^\012]+HTTP\/\d+\.\d+\015?\012/) {
          last READ_CLIENT_HEADER if $cr_buf =~ /(\015?\012){2}/;
      } elsif ($cr_buf =~ /^[^\012]+\015?\012/) {
          print "HTTP/0.9-Client" if $debug;
          last READ_CLIENT_HEADER;  # HTTP/0.9 client
      } else {
          die "Unbekannter Client";
      }
  }
  $cr_buf =~ s/^(\w+)\s+(\S+)(?:\s+(HTTP\/\d+\.\d+))?[^\012]*\012//;
  ($method, $url, $proto) = ($1, $2, $3);
  $proto ||= "HTTP/0.9";
  $request->protocol($proto);

  #
  # Pfade und Host für Proxy-Benutzung bestimmen
  #
  $url_obj = new URI::URL($url);
  if (defined $proxy_url) {
      $host = $proxy_url->host();
      $port = $proxy_url->port();
      $fullpath = $url_obj->as_string();
  } else {
```

```
        $host = $url_obj->host();
        $port = $url_obj->port();
        $fullpath = $url_obj->full_path();
    }
    $url = $fullpath;

    my($key, $val) = ("", "");
  PARSE_CLIENT_HEADER:
    while ($cr_buf =~ s/^([^\012]*)\012//) {
        $_ = $1;
        s/\015$//;
        if (/^([\w\-]+)\s*:\s*(.*)/) {
            $request->push_header($key, $val) if $key;
            ($key, $val) = ($1, $2);
        } elsif (/^\s+(.*)/) {
            $val .= " $1";
        } else {
            last PARSE_CLIENT_HEADER;
        }
    }
    $request->push_header($key, $val) if ($key);

    #
    # Existiert ein Client-Body?
    #
    $te = $request->header('Transfer-Encoding');
    $ct = $request->header('Content-Type');
    $len = $request->header('Content-Length');
    if ($te && lc($te) eq 'chunked') {
        # Handle chunked transfer encoding
        my $body = "";
      READ_CLIENT_BODY:
        while (1) {
            if ($cr_buf =~ s/^([^\012]*)\012//) {
                my $chunk_head = $1;
                $chunk_head =~ /^([0-9A-Fa-f]+)/;
                return undef unless length($1);
                my $buf_size = hex($1);
                last READ_CLIENT_BODY if $buf_size == 0;

                my $missing = $buf_size - length($cr_buf);
                $missing += 2; # also read CRLF at chunk end
                $body .= $cr_buf;
                $cr_buf = "";
                # must read rest of chunk and append it to the $body
                while ($missing > 0) {
                    die "READ_CLIENT_BODY read_timeout"
                        if $timeout && !$client_handles->can_read($timeout);
                    my $n = $client_socket->sysread($body, $missing, length($body));
                    die $! unless defined ($n);
                    die "Cannot read Body-Data from Client" unless $n;
```

```
                $missing -= $n;
            }
            substr($body, -2, 2) = ''; # remove CRLF at end
        } else {
            # need more data in order to have a complete chunk header
            die "read_timeout" if $timeout && !$client_handles->can_read($timeout);
            my $n = $client_socket->sysread($cr_buf, $size, length($cr_buf));
            die $! unless defined ($n);
            die "Cannot read Body-Data from Client" unless $n;
        }
    }
    $request->content($body);
}

#
# Client-Header verarbeiten
#
set_request_headers($request);

$request_line = $method . " " . $url . " " . $proto . $CRLF;

$ss_buf = $request_line . $request->headers_as_string($CRLF) . $CRLF . $request->content;

#
# Verbindung zum Host oder zum Proxy herstellen
#
$server_socket = IO::Socket::INET->new('PeerAddr' => $host,
                                       'PeerPort' => $port,
                                       'Proto'    => 'tcp',
                                       'Timeout'  => $timeout);
unless ($server_socket) {
    send_error_to_client($client_socket, $host, $port, $timeout, $@);
    die "IO::Socket::INET->new('$host':$port): $@";
}
$server_handles->add($server_socket);
$error_handles->add($server_socket);

WRITE_SERVER_REQUEST:
while (1) {
    die "write_timeout" if $timeout && !$server_handles->can_write($timeout);
    my $n = $server_socket->syswrite($ss_buf, length($ss_buf));
    die $! unless defined ($n);
    if ($n) {
        substr($ss_buf, 0, $n) = "";
        last WRITE_SERVER_REQUEST unless length($ss_buf);
    } else {
        die "WRITE_SERVER_REQUEST: Short syswrite()";
    }
}

READ_SERVER_HEADER:
```

```
while (1) {
    die "read_timeout" if $timeout && !$server_handles->can_read($timeout);
    my $n = $server_socket->sysread($sr_buf, $size, length($sr_buf));
    die $! unless defined ($n);
    die "unexpected EOF before status line seen" unless $n;

    if ($sr_buf =~ s/^(HTTP\/\d+\.\d+)[ \t]+(\d+)[ \t]*([^\012]*)\012//) {
        ($ver,$code,$msg) = ($1, $2, $3);
        $msg =~ s/\015$//;
        $response = HTTP::Response->new($code, $msg);
        $response->protocol($ver);

        until ($sr_buf =~ /\015?\012\015?\012/) {
            die "read timeout" if $timeout && !$server_handles->can_read($timeout);
            my $n = $server_socket->sysread($sr_buf, $size, length($sr_buf));
            die $! unless defined($n);
            die "unexpected EOF before all headers seen" unless $n;
        }

        my($key, $val);
      PARSE_SERVER_HEADERS:
        while ($sr_buf =~ s/([^\012]*)\012//) {
            my $line = $1;

            # if we need to restore as content when illegal headers
            # are found.
            my $save = "$line\012";

            $line =~ s/\015$//;
            last PARSE_SERVER_HEADERS unless length($line);

            if ($line =~ /^([a-zA-Z0-9_\-]+)\s*:\s*(.*)/) {
                $response->push_header($key, $val) if $key;
                ($key, $val) = ($1, $2);
            } elsif ($line =~ /^\s+(.*)/) {
                unless ($key) {
                    $response->header("Client-Warning" => "Illegal continuation header");
                    $sr_buf = "$save$sr_buf";

                    last PARSE_SERVER_HEADERS;
                }
                $val .= " $1";
            } else {
                $response->header("Client-Warning" => "Illegal header '$line'");
                $sr_buf = "$save$sr_buf";

                last PARSE_SERVER_HEADERS;
            }
        }
        $response->push_header($key, $val) if $key;
```

```
            last READ_SERVER_HEADER;
        } elsif ((length($sr_buf) >= 5 and $sr_buf !~ /^HTTP\//) or
                 ($sr_buf =~ /\012/)) {
            # HTTP/0.9 or worse
            $response = HTTP::Response->new(&HTTP::Status::RC_OK, "OK");
            $response->protocol('HTTP/0.9');

            last READ_SERVER_HEADER;
        } else {
            warn "need more data";
        }
    }

    $response->content($sr_buf);

    $response_line = ($response->protocol . " " .
                      $response->code . " " .
                      $response->message . $CRLF);

    {
        my $resp_copy = $response->clone();
        $response->headers->scan( sub { my($name, $value) = @_;
                                        $resp_copy->remove_header($name) if ($name
                                            =~ /^X-Signature-/);
                                    }
                                );

        $cs_buf = ($response_line . $resp_copy->headers_as_string($CRLF) .
                   $CRLF . $response->content);
    }
    $sr_buf = "";

    $read_server = 1;

MANAGE_RESPONSE_BODY:
    while (1) {
        ($neu_lesbare,
         $neu_schreibbare,
         $fehlerhafte) = IO::Select->select($server_handles,
                                            $client_handles,
                                            $error_handles,
                                            $timeout);

        foreach $socket (@$fehlerhafte) {
            if ($socket == $client_socket) {
                $client_handles->remove($socket);
                $error_handles->remove($socket);
                $socket->close();
                die "Fehlerhafter Client-Socket geschlossen";
            }
            if ($socket == $server_socket) {
```

```
                $server_handles->remove($socket);
                $error_handles->remove($socket);
                $socket->close();
                die "Fehlerhafter Server-Socket geschlossen";
            }
        }

        if ($read_server) {
            foreach $socket (@$neu_lesbare) {
                if ($socket == $server_socket) {
                    die "Server: read timeout" if $timeout &&
                        !$server_handles->can_read($timeout);
                    my $n = $server_socket->sysread($sr_buf, $size, length($sr_buf));
                    die $! unless defined ($n);
                    if ($n) {
                        $cs_buf .= substr($sr_buf, 0, $n);
                        $response->add_content(substr($sr_buf, 0, $n));
                        substr($sr_buf, 0, $n) = "";
                    } else {
                        $error_handles->remove($server_socket);
                        $server_handles->remove($server_socket);
                        $server_socket->close();
                        $read_server = 0;
                        next MANAGE_RESPONSE_BODY;
                    }
                }
            }
        }

        foreach $socket (@$neu_schreibbare) {
            if ($socket == $client_socket) {
                if (length($cs_buf)) {
                    die "write_timeout" if $timeout &&
                        !$client_handles->can_write($timeout);
                    my $n = $client_socket->syswrite($cs_buf, length($cs_buf));
                    die $! unless defined ($n);
                    if ($n) {
                        substr($cs_buf, 0, $n) = "";
                        next MANAGE_RESPONSE_BODY;
                    } else {
                        die "MANAGE_RESPONSE_BODY: Short syswrite()";
                    }
                } elsif (!$read_server) {
                    $error_handles->remove($client_socket);
                    $client_handles->remove($client_socket);
                    $client_socket->close();
                    last MANAGE_RESPONSE_BODY;
                }
            }
        }
    }
```

```
    parse_response_headers_or_fetch_sig($url_obj, $response);

    exit(0);
}

sub parse_response_headers_or_fetch_sig
{
    my $url_obj = shift;
    my $response = shift;

    my @signature;
    my $scheme;
    my $sig_error;

    #
    # Die Untersuchung der einzelnen Header wird auf einer Kopie
    # des Response gemacht, damit im Original die betreffenden Felder
    # gelöscht werden können.
    #
    my $response_copy = $response->clone();
    my $signature_included_in_http = 0;

    $response_copy->headers->scan( sub { my($name, $value) = @_;
                                         if ($name =~ /^X-Signature-Response-(.*)-(\d+)/) {
                                             $scheme = $1;
                                             $signature[$2] = $value;
                                             $signature_included_in_http = 1;
                                         } elsif ($name =~ /^$response_error/) {
                                             $sig_error = $value;
                                         }
                                       }
                                   );

    if ($signature_included_in_http) {
        my $sig = MIME::Base64::decode_base64(join('', @signature)) if @signature;
        if ($sig) {
            my %sigs;

            #
            # Signatur abspeichern
            #
            tie(%sigs, 'Signature::Stored', $ENV{'HOME'}.$sig_directory,
                $sig_scheme_ext{$scheme});
            $sigs{$url_obj->as_string()} = $sig;
            untie %sigs;

            #
            # Content abspeichern
            #
            tie %sigs, 'Signature::Stored', $ENV{'HOME'}.$sig_directory, 'dat';
            $sigs{$url_obj->as_string()} = $response->content;
```

```
            untie %sigs;

            #
            # URL in die Message-Queue schicken
            #
            my $key = msgget($IPC_KEY, 0);
            my $msg = $scheme." - ".$url_obj->as_string();

            msgsnd($key, pack("L", $TO_GUI).$msg, 0) or warn "msgsnd(): $!";
        } else {
            print "Signatur-Fehler: ", $url_obj->as_string, " - $sig_error\n" if $debug;
        }
    } else {
        my $ua = new LWP::UserAgent;
        #$ua->proxy(['http', 'ftp'], $proxy) if ($use_proxy);
        #$ua->no_proxy('localhost');

        my $scheme;
      RETRIEVE_MANUALLY:
        foreach $scheme (@preferred_sig_schemes) {
            my $request = new HTTP::Request('GET', $url_obj->as_string . "." .
                                            $sig_scheme_ext{$scheme});
            my $sig_response = $ua->request($request);

            if ($sig_response->is_success) {
                my %sigs;

                #
                # Signatur abspeichern
                #
                tie(%sigs, 'Signature::Stored', $ENV{'HOME'}.$sig_directory,
                    $sig_scheme_ext{$scheme});
                $sigs{$url_obj->as_string()} = $sig_response->content;
                untie %sigs;

                #
                # Content abspeichern
                #
                tie %sigs, 'Signature::Stored', $ENV{'HOME'}.$sig_directory, 'dat';
                $sigs{$url_obj->as_string()} = $response->content;
                untie %sigs;

                #
                # URL in die Message-Queue schicken
                #
                my $key = msgget($IPC_KEY, 0);
                my $msg = $scheme." - ".$url_obj->as_string();

                msgsnd($key, pack("L", $TO_GUI).$msg, 0) or warn "msgsnd(): $!";
                last RETRIEVE_MANUALLY;
            }
```

```
        }
    }
}

sub set_request_headers
{
    my $request = shift;

    my $schemes;
    foreach (@preferred_sig_schemes) {
        if (defined $sig_scheme_ext{$_}) {
            $schemes .= $_ . " ";
        }
    }
    if ($schemes) {
        $request->push_header($request_header_name, $schemes);
    }
    print "$request_header_name: $schemes\n" if $debug;
}

sub send_error_to_client
{
    my ($client_socket, $host, $port, $timeout, $error) = @_;

    my $cs_buf = "HTTP/1.0 503 Service Unavailable\015\012";
    $cs_buf .= "Content-Type: text/html\015\012";
    $cs_buf .= "\015\012";
    $cs_buf .= "<TITLE>$error</TITLE>\n";
    $cs_buf .= "<P>Der Host $host:$port ist nicht zuerreichen:\n$error\n";

    while (1) {
        my $client_handles = new IO::Select($client_socket);
        die "write_timeout" if $timeout && !$client_handles->can_write($timeout);
        my $n = $client_socket->syswrite($cs_buf, length($cs_buf));
        die $! unless defined ($n);
        if ($n) {
            substr($cs_buf, 0, $n) = "";
            next;
        } else {
            last;
        }
    }
}

sub SIG_worker_soll_sterben
{
    print "Worker $$ hat SIGUSR1 empfangen\n" if $debug;
    exit(1)
}

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

```
#
# Haupt-Prozess
#
######################

sub get_msg_for_gui
{
    my $key = msgget($IPC_KEY, 0);
    my $msgbuf = "";
    my $type;
    my $text;

    #
    # Message-Queue leeren
    #
    while (1) {
        msgrcv($key, $msgbuf, 1000, $TO_GUI, $IPC_NOWAIT);

        if ($! =~ /No message/) {
            last;
        } else {
            ($type, $text) = unpack("L a1000", $msgbuf);

            $lb->insert('end', $text);
        }
    }

    $main->after($msg_rcv_timeout, \&get_msg_for_gui);
}

sub start_tk
{
    my $pid = shift;

    use Tk;           # Nur der Tk-Prozess sollte den Tk-Code einbinden, damit die
    use Crypt::SGMTT; # Nur der Tk-Prozess benötigt den Signatur-Code
    use Crypt::SSP::Data;

    $main = MainWindow->new('-title' => "Signatur-Proxy \"$0\" (c) by CHGEUER",
                            );

    $main->positionfrom('user');

    $menu = $main->Frame()->pack('-side' => 'top',
                                 '-fill' => 'x',
                                 );

    $fmenu = $menu->Menubutton('-text' => 'Datei',
                               '-tearoff' => '0',
                               )->pack('-side' => 'left',
                                       '-padx' => '2',
```

```
                                    );

    $fmenu->command('-label' => 'Beenden',
                    '-command' => sub { kill 9, $pid;
                                        exit()},
                    );

    $f2 = $main->Frame()->pack('-side' => 'bottom',
                               '-fill' => 'x',
                               );

    $f2->Button('-text' => 'Beenden',
                '-command' => sub { kill 9, $pid;
                                    exit(); })->pack('-side' => 'left');

    $f1 = $main->Frame()->pack('-side' => 'top',
                               '-fill' => 'both',
                               '-expand' => 'y',
                               );

    $lb = $f1->Scrolled('Listbox',
                        '-width' => 40,
                        '-height' => 6,
                        '-scrollbars' => 'se',
                        )->pack('-side' => 'left',
                                '-fill' => 'both',
                                '-expand' => 'y',
                                );

    $lb->bind('<Double-1>', \&show_sig);

    $main->after($msg_rcv_timeout, \&get_msg_for_gui);

    MainLoop;
}

sub show_sig
{
    my $message = $lb->get('active');
    $lb->selectionClear('active');

    return if (!$message);

    my ($scheme, $url) = split(/ - /, $message);

    my ($data, $sig);
    my ($cert_d_name, $cert_id_name, $cert_serno);

    {
        my %sigs;
```

```
        #
        # Signatur laden
        #
        tie(%sigs, 'Signature::Stored', $ENV{'HOME'}.$sig_directory,
            $sig_scheme_ext{$scheme});
        $sig = $sigs{$url};
        untie %sigs;

        #
        # Content laden
        #
        tie %sigs, 'Signature::Stored', $ENV{'HOME'}.$sig_directory, 'dat';
        $data = $sigs{$url};
        untie %sigs;
    }


    if ($scheme eq 'Mtt-V1') {
        my $sgmtt = new Crypt::SGMTT::Signature(
  'TFLAG' => new Crypt::SGMTT::TFLAG('MTT_FLAG_CHECKSIGPATH' => '1',
                                     'MTT_FLAG_UPDATETRUSTED' => '1',
                                     ),
                                               'DEBUG' => '0',
                                               );

        $sgmtt->verify($sig);

        if ($sgmtt->sig_ok()) {
            $cert_d_name = $sgmtt->cert_d_name();
            $cert_id_name = $sgmtt->cert_id_name();
            $cert_serno = $sgmtt->cert_serno();

            my $ssp = Crypt::SSP::Data->new_from_dat($sgmtt->as_string());

            if (!$ssp->is_ssp) {
                #
                # Objekt ist kein gültiges SSP-Objekt
                #
                dialog('-title' => "\"$url\"",
                       '-text' => 'Kein gültiges SSP-Objekt',
                       '-bg' => 'red');
            } elsif ($ssp->verify_data($data) &&
                $ssp->verify_url($url) &&
                $ssp->time_valid) {
                show_ssp('-title' => "\"$url\"",
                         '-cert_d_name' => $cert_d_name,
                         '-cert_id_name' => $cert_id_name,
                         '-cert_serno' => $cert_serno,
                         '-text' => $ssp->as_string,
                         '-bg' => 'white',
                         );
```

```
        } elsif (!$ssp->verify_data($data)) {
            #
            # Hash der Daten
            #
            dialog('-title' => "\"$url\"",
                   '-text' => 'Kann Signatur nicht prüfen. SSP-Object '.
                              'und Daten gehören nicht zusammen',
                   '-bg' => 'red');
        } elsif (!$ssp->verify_url($url)) {
            #
            # Daten liegen auf der falschen URL
            #
            dialog('-title' => "\"$url\"",
                   '-text' => sprintf("Daten liegen auf der falschen URL: %s", $ssp->url),
                   '-bg' => 'red');
        } elsif (!$ssp->time_valid) {
            #
            # Zeitinterval nicht OK
            #
            dialog('-title' => "\"$url\"",
                   '-text' => sprintf('Zeitinterval abgelaufen: %s - %s',
                                      $ssp->valid_not_before,
                                      $ssp->valid_not_after
                                      ),
                   '-bg' => 'red');
        } else {
            dialog('-title' => "\"$url\"",
                   '-text' => 'Kann Signatur nicht prüfen. SSP-Objekt ungültig',
                   '-bg' => 'red');
        }
    } else {
        #
        # Das SGMTT-Objekt zeigt keinen korrekten Signatur-Status
        #
        if ($sgmtt->error eq 'MTT_ERR_NOMTTFILE') {
            #
            # Die Signatur scheint nicht im MTT-Format zu sein
            #
            dialog('-title' => "\"$url\"",
                   '-text' => "Die Signatur hat kein gültiges MTT-Format",
                   '-bg' => 'yellow');
        } elsif (($sgmtt->error eq 'MTT_ERR_NOCERT') &&
                 ($sgmtt->error_func =~ /mtt_scan_body_init/)) {
            #
            # Zum Prüfen von Zertifikaten müssen diese in der Datenbank stehen.
            # Im eine "Verschmutzung" durch Zertifikate ohne Root-CA-Stempel zu
            # verhindern, steht im SGMTT-Objekt UPDATETRUSTED. Daher können
            # self-signed-Zertifikate nicht anerkannt werden
            #
            dialog('-title' => "\"$url\"",
                   '-text' => "Das Zertifikat wurde nicht von der Root-CA zertifiziert",
```

```
                    '-bg' => 'green');
            } else {
                #
                # Irgend ein anderer Fehler
                #
                dialog('-title' => "\"$url\"",
                       '-text' => sprintf("MailTrusT-Fehler %s: \"%s\" in %s", $sgmtt->error,
                                          $sgmtt->error_as_string, $sgmtt->error_func),
                       '-bg' => 'blue');
            }
        }
    } else {
        #
        # Es wurde ein andere Signatur-Schema als 'Mtt-V1' gewählt
        #
        dialog('-title' => "\"$url\"",
               '-text' => "Signatur-Schema $scheme unbekannt\n",
               '-bg' => 'orange');
    }
}

sub dialog
{
    my %data = @_;

    my $sig_main = MainWindow->new();
    $sig_main->configure('-title' => $data{'-title'}) if defined $data{'-title'};

    $sig_main->Label('-text' => $data{'-text'},
                     '-anchor' => 'n',
                     '-relief' => 'groove',
                     )->pack();
    $sig_main->Button('-text' => 'OK',
                      '-command' => sub { $sig_main->destroy(); },
                      )->pack('-side' => 'bottom');

    $sig_main->configure('-bg' => $data{'-bg'}) if defined $data{'-bg'};
}

sub show_ssp
{
    my %data = @_;

    my $sig_main = MainWindow->new('-title' => $data{'-title'});
    my $cert_frame = $sig_main->Frame('-width' => 100,
                                      '-height' => 20,
                                      )->pack('-fill' => 'x',
                                              '-side' => 'top',
                                              '-expand' => 'y',
                                              );
```

```
#
# DNAME
#
my $l1 = $cert_frame->Label('-text' => 'Distinguished Name: Zertifikatsinhaber');
$l1->grid('-row' => 0,
          '-column' => 0,
          '-sticky' => 'w');
$cert_frame->Entry('-textvariable' => \$data{'-cert_d_name'},
                   '-state' => 'disabled',
                   '-width' => 100)->grid('-row' => 0,
                                          '-column' => 1,
                                          '-sticky' => 'ew');

#
# IDNAME
#
my $l2 = $cert_frame->Label('-text' => 'Distinguished Name: Zertifikatsherausgeber');
$l2->grid('-row' => 1,
          '-column' => 0,
          '-sticky' => 'w');
$cert_frame->Entry('-textvariable' => \$data{'-cert_id_name'},
                   '-state' => 'disabled',
                   '-width' => 100)->grid('-row' => 1,
                                          '-column' => 1,
                                          '-sticky' => 'ew');

#
# SERNO
#
my $l3 = $cert_frame->Label('-text' => 'Seriennummer Zertifikat');
$l3->grid('-row' => 2,
          '-column' => 0,
          '-sticky' => 'w');
$cert_frame->Entry('-textvariable' => \$data{'-cert_serno'},
                   '-state' => 'disabled',
                   '-width' => 100)->grid('-row' => 2,
                                          '-column' => 1,
                                          '-sticky' => 'ew');

#
# Signatur-Frame
#
my $sig_frame = $sig_main->Frame('-width' => 100,
                                 '-height' => 20,
                                 '-bg' => 'black',
                                 )->pack('-fill' => 'both',
                                         '-side' => 'top',
                                         '-expand' => 'y',
                                         );

my $bq = $sig_frame->Button('-text' => 'Schließen',
```

```
                                ’-command’ => sub { $sig_main->destroy(); },
                                )->pack(’-side’ => ’bottom’);

    my $txt = $sig_frame->Scrolled(’Text’,
                                   ’-height’ => ’40’,
                                   ’-width’ => ’80’,
                                   ’-scrollbars’ => ’se’)->pack(’-fill’ => ’both’,
                                                                ’-expand’ => ’yes’);

    $txt->insert(’end’, $data{’-text’});

    $txt->configure(’-bg’ => $data{’-bg’});
    $txt->configure(’-state’ => ’disabled’);
}
```

A.5.2. Stored.pm

Das Stored-Modul bietet die Möglichkeit, die Daten und Signaturen einfach und transparent auf der Festplatte zu speichern.

```
package Signature::Stored;

use File::Path;
use Crypt::RIPEMD160;

#
# use Signature::Stored;
#
# tie %sigs, ’Signature::Stored’, $ENV{’HOME’}.’/.signature-cache’, ’mtt’;
#
# my $sig = "Dies ist die Signatur selbst\n";
# my $url = "http://crypto.gun.de/index.html";
#
# $sigs{$url} = $sig;
#
# $sigs{"http://www.uni-siegen.de:80/index.html"} = "uni siegen\n";
#
# print "Existiert\n" if exists $sigs{"http://www.uni-siegen.de:80/index.html"};
#
# untie %sigs;
#

sub TIEHASH {
    my $pkg = shift;
    my $sig_dir = shift;
    my $sig_scheme_ext = shift || ’dat’;

    my $obj = {’sig_dir’ => $sig_dir,
               ’sig_scheme_ext’ => $sig_scheme_ext};

    bless $obj, $pkg;
```

```
}

sub FETCH {
   my ($obj, $url) = @_;

   my $url_hash = Crypt::RIPEMD160->hexhash($url);
   $url_hash =~ s/\s//g;
   $url_hash .= "." . $obj->{'sig_scheme_ext'};

   my $dir1 = substr($url_hash, 0, 2);
   my $dir2 = substr($url_hash, 2, 2);

   my $dir = $obj->{'sig_dir'} . "/$dir1/$dir2";

   return undef unless -s "$dir/$url_hash";

   my $sig;
   {
       local($/);
       undef $/;

       open(F, "<$dir/$url_hash") || die "Kann $dir/$url_hash nicht lesen";
       flock(F, 2); # Exklusiv blockieren
       $sig = <F>;
       flock(F, 8); # Sperre aufheben
       close(F) || die "Kann $dir/$url_hash nicht schließen";
   }

   $sig;
}

sub STORE {
   my ($obj, $url, $sig) = @_;

   my $url_hash = Crypt::RIPEMD160->hexhash($url);
   $url_hash =~ s/\s//g;
   $url_hash .= "." . $obj->{'sig_scheme_ext'};

   my $dir1 = substr($url_hash, 0, 2);
   my $dir2 = substr($url_hash, 2, 2);

   my $dir = $obj->{'sig_dir'} . "/$dir1/$dir2";

   &File::Path::mkpath([$dir], 0, 0700) unless (-d $dir);

   open(F, ">$dir/$url_hash") || die "Kann $dir/$url_hash nicht schreiben";
   flock(F, 2); # Exklusiv blockieren
   print F $sig;
   flock(F, 8); # Sperre aufheben
   close(F) || die "Kann $dir/$url_hash nicht schließen";
```

```
   $sig;
}

sub DELETE {
   my ($obj, $url) = @_;

   my $url_hash = Crypt::RIPEMD160->hexhash($url);
   $url_hash =~ s/\s//g;
   $url_hash .= "." . $obj->{'sig_scheme_ext'};

   my $dir1 = substr($url_hash, 0, 2);
   my $dir2 = substr($url_hash, 2, 2);

   my $dir = $obj->{'sig_dir'} . "/$dir1/$dir2";

   return undef unless -s "$dir/$url_hash";

   unlink "$dir/$url_hash";
}

sub EXISTS {
   my ($obj, $url) = @_;

   my $url_hash = Crypt::RIPEMD160->hexhash($url);
   $url_hash =~ s/\s//g;
   $url_hash .= "." . $obj->{'sig_scheme_ext'};

   my $dir1 = substr($url_hash, 0, 2);
   my $dir2 = substr($url_hash, 2, 2);

   my $dir = $obj->{'sig_dir'} . "/$dir1/$dir2";

   (-s "$dir/$url_hash" > 0);
}

sub DESTROY {
}

1;
```

B. Online-Resourcen

Thema	URL
Perl	http://www.perl.com/
mod_perl	http://perl.apache.org/
Apache	http://www.apache.org/
MailTrusT & TeleTrusT	http://www.teletrust.de/
SmartGuard MailTrusT	http://www.kryptokom.de/
Pretty Good Privacy	http://www.pgpi.com/
Crypt::RIPEMD160	http://www.perl.com/CPAN/authors/id/C/CH/CHGEUER/
SUN	http://www.sun.com/
Netscape	http://www.netscape.com/
ActiveX	http://www.microsoft.com/activex/

Literaturverzeichnis

[And93] Ross Anderson. Why cryptosystems fail. *Proceedings of the First ACM Conference on Computer and Communications Security*, 1993.

[ASZ96] Atkins, D., Stallings, W. und Zimmermann, P. PGP Message Exchange Formats. *Request for Comments: 1991*, August 1996.

[Bal93] D. Balenson. Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers. *Request for Comments: 1423*, Februar 1993.

[Bar97] George Barwood. FAQ: Kryptografie mit elliptische Kurven, v1.8. *Internet:* `http://www.fitug.de/ulf/faq/ec_faq.html`, Mai 1997.

[Bau94] Friedrich L. Bauer. *Kryptologie: Methoden und Maximen*. Springer-Verlag, 1994.

[Bau96] Fritz Bauspieß. *MailTrusT Spezifikation, v1.1*. TeleTrusT: Verein zur Förderung der Vertrauenswürdigkeit von Informations- und Kommunikationstechnik, 1996.

[Beu94] Albrecht Beutelspacher. *Kryptologie*. Vieweg Verlag, 1994.

[BGG+97] Biddle, C. Bradford, Garfinkel, Simson, Gilmore, John, Khare, Robit und Liu, Cricket. *Web Security: A Matter of Trust*. O'Reilly & Associates, Inc., 1997.

[BS97] Buth, Karl Heinz und Sommer, Carsten. Sicherer Geschäftsverkehr über das Internet. *Mit Sicherheit in die Informationsgesellschaft*, Seiten 413–425, 1997.

[BSW95] Beutelspacher, Albrecht, Schwenk, Jörg und Wolfenstetter, Klaus-Dieter. *Moderne Verfahren der Kryptographie*. Vieweg Verlag, 1995.

[Bun97] Bundesamt für Sicherheit in der Informationstechnik. *Mit Sicherheit in die Informationsgesellschaft*. SecuMedia Verlag Ingelheim, 1997.

[CDLL97] Chu, Yang-hua, DesAutels , Philip, LaMacchia , Brian und Lipp, Peter. DSig 1.0 Signature Labels — Using PICS 1.1 Labels for Making Signed Assertions. *Internet:* `http://www.w3.org/TR/PR-DSig-label.html`, November 1997.

[DD98] Denning, Dorothy E. und Denning, Peter J. *Internet besieged: countering cyberspace scofflaws*. Addison Wesley, 1998.

[Dre97] Stephan Dresen. Fata Morgana: Datenwirrwarr durch Proxy-Manipulation und Spoofing. *iX*, August 1997.

[ECS94] Eastlake, Donald, Crocker, Stephen und Schiller, Jeffrey. Randomness Recommendations for Security. *Request for Comments: 1750*, Dezember 1994.

[Eil97] Lars Eilebrecht. *Apache Web-Server*. Internat. Thomson Publ., 1997.

[Elk96] M. Elkins. MIME Security with Pretty Good Privacy (PGP). *Request for Comments: 2015*, Oktober 1996.

[FFB96] Fielding, R., Frystyk, H. und Berners-Lee, T. Hypertext Transfer Protocol – `HTTP/1.0`. *Request for Comments: 1945*, Mai 1996.

[FGM+97] Fielding, R., Gettys, J., Mogul, J., Frystyk, H. und Berners-Lee, T. Hypertext Transfer Protocol – `HTTP/1.1`. *Request for Comments: 2068*, Januar 1997.

[Fla97] David Flanagan. *JavaScript: Das umfassende Referenzwerk*. O'Reilly & Associates, Inc., 1997.

[FM96] Fischer, Stefan und Müller, Walter. *Netzwerkprogrammierung unter LINUX und UNIX*. Hanser Verlag, 1996.

[FR94] Fumy, Walter und Rieß, Hans Peter. *Kryptographie: Entwurf, Einsatz und Analyse symmetrischer Kryptoverfahren*. R. Oldenbourg Verlag, 1994.

[Fri98] Jeffrey E.F. Friedl. *Reguläre Ausdrücke*. O'Reilly & Associates, Inc., 1998.

[Gar96] Simson Garfinkel. *PGP: Pretty Good Provacy*. O'Reilly & Associates, Inc., 1996.

[GP98] Grimm, Rüdiger und Pordesch, Ulrich. Wie sicher ist die digitale Signatur. *Spektrum der Wissenschaft: Dossier – Die Welt im Internet*, Seiten 60–62, 1998.

[GR98] Geppert, Martin und Roßnagel, Alexander. *Telemediarecht: Telekommunikations– und Multimediarecht*. Deutscher Taschenbuch Verlag, 1998.

[Gru98] Alexander Gruhler. PICS – eine moderne Version der Zensur? *`http://www.heise.de/tp/deutsch/special/krypto/6247/1.html`*, Juni 1998.

[GS97] Garfinkel, Simson und Spafford, Gene. *Web Security & Commerce*. O'Reilly & Associates, Inc., 1997.

[Gun96] Shishir Gundavaram. *CGI Programmierung im World Wide Web*. O'Reilly & Associates, Inc., 1996.

[Haj98] Fahrid Hajji. *Perl: Einführung, Anwendungen, Referenz*. Addison-Wesley, 1998.

[Her96] Helmut Herold. *UNIX–Systemprogrammierung*. Addison Wesley, 1996.

[Hof95] Lance J. Hoffmann. *Building in big brother: the cryptographic policy debate.* Springer-Verlag, 1995.

[Hor85] Patrick Horster. *Kryptologie*. Bibliographisches Institut, 1985.

[HS98] Hall, Joseph N. und Schwartz, Randal L. *Effective Perl Programming*. Addison Wesley, 1998.

[Hun95] Craig Hunt. *TCP/IP Netzwerk Administration*. O'Reilly & Associates, Inc., 1995.

[Int93] International Telecommunications Union. The Directory — Authentication Framework. *ITU-T Recommendation X.509*, November 1993.

[Kal93] B. Kaliski. Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services. *Request for Comments: 1424*, Februar 1993.

[Ken93] S. Kent. Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management. *Request for Comments: 1422*, Februar 1993.

[Kop98] Markus Kopp. Sich sichern: Javas Sandbox-Modell und signierte Applets. *iX*, Seiten 130–133, Juni 1998.

[Kun97] Michael Kunze. Kartentausch: Web-Konsortium will den Schutz der Privatsphäre im Netz vereinheitlichen. *c't Report 3: Geld online*, 1997.

[Kya96] Othmar Kyas. *Sicherheit im Internet: Risikoanalyse – Strategien – Firewalls*. DATACOM-Verlag, 1996.

[Lin93] J. Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. *Request for Comments: 1421*, Februar 1993.

[LL97] Laurie, Ben und Laurie, Peter. *Apache: The Definitive Guide*. O'Reilly & Associates, Inc., 1997.

[LPJ+95] Liu, Cricket, Peek, Jerry, Jones, Russ, Buus, Bryan und Nye, Adrian. *Internet–Server: Einrichten und Verwalten*. O'Reilly & Associates, Inc., 1995.

[LR97] Le, Dinh Khoi und Rauer, Matthias. Seminarausarbeitung: Elliptische Kurven. *Universität Paderborn: Seminar SS 1997*, Dezember 1997.

[Luc97a] Norbert Luckhardt. Die Mutter der Porzellankiste: Wie Kryptografie vor den Gefahren im Internet schützt. *c't Report 2: Geld online*, 1997.

[Luc97b] Norbert Luckhardt. SRT: Browser genügt. *c't Report 2: Geld online*, 1997.

[Mac97] Doug MacEachern. Apache-Perl interface to the Apache server API. `mod_perl` *Documentation*, Mai 1997.

[Mac98] Doug MacEachern. `mod_perl` - Embed a Perl interpreter in the Apache HTTP server. *`mod_perl` Documentation*, Marz 1998.

[MK97] Musciano, Chuck und Kennedy, Bill. *HTML: Das umfassende Referenzwerk*. O'Reilly & Associates, Inc., 1997.

[MP97] Müller, Günter und Pfitzmann, Andreas. *Mehrseitige Sicherheit in der Kommunikationstechnik: Verfahren, Komponenten, Integration*. Addison Wesley, 1997.

[MvV97] Menezes, Alfred J., van Oorschot, Paul C. und Vanstone, Scott A. *Handbook of Applied Cryptography*. CRC Press, 1997.

[MW97] Mraz, Victor und Weidner, Klaus. Falsch verbunden: Gefahr durch DNS-Spoofing. *c't*, Oktober 1997.

[NS78] Needham, R.M. und Schroeder, M.D. Using Encryption for Authentication in Large Networks od Computers. *Communications of the ACM, Band 21, Heft 12*, Seiten 993–999, 1978.

[Oka96] Jeff Okamoto. `perlXStut` – Tutorial für XSUBs. *Perl Documentation*, Oktober 1996.

[OR97] Okamoto, Jeff und Roehrich, Dean. `perlguts` – Perl's Internal Functions. *Perl Documentation*, Mai 1997.

[Rae98] Martin Raepple. *Sicherheitskonzepte für das Internet*. dpunkt.verlag, 1998.

[Roe96] Dean Roehrich. `perlxs` – XS language reference manual. *Perl Documentation*, Juli 1996.

[RSA78] Rivest, R.L., Shamir, A. und Adleman, L. M. A method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, Februar 1978.

[RSA95] RSA Laboratories. PKCS #11: Cryptographic Token Interface Standard. *An RSA Laboratries Technical Note*, April 1995.

[Rul93] Christoph Ruland. *Informationssicherheit in Datennetzen*. DATACOM-Verlag, 1993.

[Sch95] Wolfgang Schneider. Schutzregeln für den globalen Marktplatz gesucht. *NET-Spezial*, Januar 1995.

[Sch96] Bruce Schneier. *Angewandte Kryptographie*. Addison Wesley, 1996.

[Sch97] Michael Schilli. *Effektives Programmieren mit Perl 5*. Addison-Wesley, 1997.

[Sch98a] Jeffrey Schiller. Sicherheit im Daten-Nahverkehr. *Spektrum der Wissenschaft: Dossier – Die Welt im Internet*, Seiten 52–57, 1998.

[Sch98b] Rudolf Schöngarth. *SmartGuard MTT: MailTrusT-konforme Funktionsbibliothek – Schnittstellenbeschreibung*. KryptoKom GmbH, Aachen, 1998.

[SFP+97] Siever, Ellen, Futato, David, Patwardhan, Nate, Irving, Clay und Jepson, Brian. *Perl Resource Kit - Unix Edition*. O'Reilly & Associates, Inc., 1997.

[Sie97] Richard Sietmann. Zweifelsfrei authentisch: Ablösung des PIN-Codes als Zugangsschutz überfällig. *c't Report 3: Geld online*, 1997.

[Smi98] Richard E. Smith. *Internet-Kryptographie*. Addison Wesley, 1998.

[SNS88] Steiner, J.G., Neumann, B.C. und Schiller, J.I. Kerberos: An Authentication Service for Open Network Systems. *Usenix Conference Proceedings, Dallas (Texas)*, Seiten 191–202, 1988.

[Sri97] Sriram Srinivasan. *Advanced Perl Programming*. O'Reilly & Associates, Inc., 1997.

[Sri98] Sriram Srinivasan. *Fortgeschrittene Perl Programmierung*. O'Reilly & Associates, Inc., 1998.

[Sta95] William Stallings. *Sicherheit im Datennetz*. Prentice Hall, 1995.

[Sta97] William Stallings. *Data and Computer Communications*. Prentice Hall, 1997.

[Ste95] W. Richard Stevens. *Programmieren in der Unix–Umgebung: die Referenz für Fortgeschrittene*. Addison Wesley, 1995.

[SWE98] Scott, Charlie, Wolfe, Paul und Erwin, Mike. *Virtual Private Networks*. O'Reilly & Associates, Inc., 1998.

[Tan97] Andrew S. Tanenbaum. *Computernetzwerke*. Prentice Hall, 1997.

[WCS97] Wall, Lary, Christiansen, Tom und Schwartz, Randal L. *Programmieren mit Perl*. O'Reilly & Associates, Inc., 1997.

[Wei97] Rüdiger Weis. Sichere Datenübertragung mit SSL. *DOS*, April 1997.

[Wel98] Michael Welschenbach. *Kryptografie in* `C` *und* `C++`. Springer Verlag, 1998.

[Wob97] Reinhard Wobst. *Abenteuer Kryptologie*. Addison Wesley, 1997.

[Won98] Clinton Wong. *Web-Client Programmierung mit Perl*. O'Reilly & Associates, Inc., 1998.

[Wor97] World Wide Web Consortium. W3C Digital Signature Initiative Overview. *Internet:* `http://www.w3.org/Security/DSig`, November 1997.

[Zim95] Philip Zimmermann. *PGP source code and internals*. The MIT press, 1995.