



<ZÜRCHER HOCHSCHULE WINTERTHUR>

<KOMMUNIKATIONS-SYSTEME>

PROJEKTARBEIT 12.03.01 – 18.05.2001
Signierte eMails aus Web-Applikationen versenden

Dozent: Dr. Andreas Steffen

Partnerfirma: netcetera
Dr. sc. Techn. ETH Peter Kohler

Gruppe: Dejan Miletic & Lukas Valle



1 Inhaltsverzeichnis

1 Inhaltsverzeichnis.....	II
2 Zusammenfassung.....	1
3 Aufgabenstellung.....	2
3.1 Beschreibung.....	2
3.1.1 Aufgaben	2
3.1.2 Infrastruktur.....	3
3.1.3 Software.....	3
4 Einleitung.....	4
5 Grundlagen.....	5
5.1 email Security.....	5
5.1.1 Sicherheitsanforderungen.....	5
5.1.2 Asymmetrische Verschlüsselung.....	5
5.1.3 Hash–Algorithmen.....	6
5.2 Digitale Umschläge.....	7
5.2.1 Digitale Signaturen.....	8
5.3 Echtheit der öffentlichen Schlüssel.....	9
5.4 Zertifikate für öffentliche Schlüssel nach CCITT X.509.....	10
5.5 Pretty Good Privacy (PGP).....	10
5.5.1 PGP Verschlüsselungsverfahren.....	11
5.5.2 Web of Trust.....	12
5.5.3 Symmetrische Verschlüsselungsverfahren.....	13
5.5.4 Digitale Signaturen bei PGP.....	13
5.6 Secure / Multipurpose Internet Mail Extensions (S/MIME).....	13
5.6.1 Multipurpose Internet Mail Extension (MIME).....	14
5.6.2 PKCS und S/MIME.....	14
5.6.3 Die erweiterten MIME–Typen.....	15
5.6.4 Kryptographische Algorithmen.....	17
5.6.5 Zertifikate für öffentliche Schlüssel.....	17
5.7 Vergleich der vorgestellten Systeme.....	18
6 Übersicht.....	19
7 Grundkonfiguration GnuPG.....	20
7.1 Gnu Privacy Guard	20
7.2 Installation und Schlüsselgeneration.....	20

7.3 Aufbau der Webseite.....	22
7.4 Signier– und Versandvorgang.....	23
7.5 Tests.....	26
7.6 Vorteile / Nachteile.....	28
7.7 Schnellinstallation.....	28
8 Grundkonfiguration OpenSSL.....	29
8.1 OpenSSL / S/MIME.....	29
8.2 Installation.....	29
8.3 Aufbau der WebSeite.....	29
8.4 Signier– und Versandvorgang.....	29
8.5 Tests.....	32
8.6 Vorteile / Nachteile.....	33
8.7 Schnellinstallation.....	33
9 Grundkonfiguration servlet–smime.....	34
9.1 Java–Servlet.....	34
9.2 Installation der Java–Umgebung.....	35
9.3 Installation von Tomcat.....	37
9.4 Keys erstellen.....	37
9.5 Anpassen der Quelldateien.....	38
9.6 Versandvorgang.....	39
9.7 Klassenbeschreibung.....	39
9.8 Vorteile / Nachteile.....	41
10 Schlüssellgeneration.....	42
10.1 Root–Zertifikat erstellen.....	42
10.2 User Zertifikate erstellen.....	42
10.2.1 Sender.....	43
10.2.2 Reciever.....	43
10.3 Zertifikate mit Root Zertifikat unterschreiben.....	44
10.3.1 Sender.....	44
10.4 Receiver.....	44
10.5 PKCS12 Dateien erstellen.....	45
10.5.1 Sender.p12.....	45
10.5.2 Receiver.p12.....	45
11 Zertifikate.....	46
11.1 Netscape 4.76	46
11.2 Internet Explorer.....	48

12 Webshell / TCL.....	49
12.1 InstallationTCL.....	49
12.2 Installation webshell.....	49
12.3 Apache Modul generieren.....	49
12.4 Error-Logs.....	49
13 Apache – Server.....	50
14 CD Aufbau.....	51
15 Schlusswort.....	52
15.1 Fazit.....	52
15.2 Weiterentwicklungsmöglichkeiten / Ausblick.....	52
15.3 Dank.....	52
16 Zeitplan.....	53
16.1 Soll Zeitplan.....	53
16.2 Ist-Zeitplan.....	55
17 Literaturverzeichnis.....	57
17.1 RFC's.....	57
17.2 Webseiten.....	57
17.3 Magazine.....	58
17.4 Bücher.....	58
18 Glossar.....	59
19 Anhang.....	61
19.1 JCSI – Lizenzbedingungen.....	61

2 Zusammenfassung

In vielen Web–Applikationen werden sensitive Informationen in ein Formular eingegeben. Unsere Aufgabe bestand darin, einige Lösungen für eine Server–Applikation zu erforschen, welche die Daten dieses Formulars signiert, eventuell verschlüsselt und an Dritte weiterleitet. Wir studierten verschiedene mögliche Varianten und einigten uns drei Varianten weiterzuverfolgen, mit welchen wir dann schlussendlich ans Ziel gelangten. Dabei handelt es sich um folgende Technologien: GnuPG, OpenSSL und Java–Servlet.

Anfänglich beschäftigten wir uns mit der Materie indem wir uns in die Thematik vertieften. GnuPG und OpenSSL haben eine wichtige Gemeinsamkeit. Beides sind reine Konsolenprogramme die stabil unter Linux arbeiten. Daher ist es möglich bei beiden Varianten den gesamten Signiervorgang jeweils durch ein Skript zu automatisieren. Wir verwendeten für beide Lösungen ein beinahe identisches *webshell* Skript, welches den webseiten Aufbau erledigt. In diesen Skripten werden sowohl GnuPG als auch OpenSSL mit den jeweils benötigten Parametern aufgerufen. Dies ermöglichte eine parallele und effiziente Entwicklung dieser beiden Lösungen.

Beim intensiven Durchsuchen des Netzes stiessen wir auf das Produkt JCSI, welches uns ermöglichte, die 3. Variante unter Java zu entwickeln. JCSI stellt Klassen für das Verschlüsseln und Signieren von Mails zur Verfügung. Ausserdem fanden wir auf dieser Homepage kleine Beispiele, womit der Einstieg wesentlich erleichtert wurde. Für das Versenden von Mails verwendeten wir die Java–Mail Klasse 1.2. Für die Tests und die Präsentation der ersten beiden Varianten konfigurierten wir den Apache Webserver und für die Java–Servlet Lösung Jakarta Tomcat.

So konnten wir in relativ kurzer Zeit unsere ersten lauffähigen Versionen präsentieren, mit denen wir Mails signiert und verschlüsselt versenden konnten. Grosse Mühe bereitete uns aber die Dokumentation unserer Arbeit bzw. der einzelnen Arbeitsschritte. Wir brauchten sehr viel Zeit für die Fertigstellung des Projektes und die exakte Dokumentation. Es schien uns auch wichtig, die Generierung eigener Schlüssel so klar als möglich aufzuzeigen, da es nicht zwingend offizieller Schlüssel einer professionellen Zertifizierungsstelle bedarf.

Wir sind sehr zufrieden mit dem erreichten Resultat, sowie der Betreuung von Herrn Dr. Steffen und Herrn Dr. Kohler. Unklarheiten wurden binnen weniger Stunden kompetent beantwortet, was unsere Zusammenarbeit und Motivation wesentlich förderte.

Winterthur 03.05.01

Lukas Valle

Dejan Miletic

3 Aufgabenstellung

3.1 Beschreibung

In dieser Arbeit sollen verschiedene Varianten zum Verschicken von signierten E-Mails aus Web-server-Applikationen untersucht und deren Vor- und Nachteile analysiert werden. Zudem sollte eine einfache Implementation mit OpenSource-Tools durchgeführt werden. Weiter soll der Aufwand für eine allfällige Verschlüsselung abgeklärt werden.

Eine Auswahl möglicher Technologien:

Server-Applikation: Perl
CGI Skript
Java Servlet
C++

Secure Mail Standards: S/MIME
OpenPGP

Crypto-Libraries: OpenSSL
Cryptix
cryptlib
PGP / GPG
JCSI

3.1.1 Aufgaben

Folgende Aufgaben galt es zu lösen. Eine genauere Zusammenstellung inklusive des zeitlichen Aufwandes befindet sich im Kapitel 16

- Installieren des Betriebssystems SuSE Linux 7.1 mit Kernel 2.4
- Studium von X.509 Zertifikaten
- Studium von PGP / GnuPG
- Zusammentragen von allgemeinen Informationen aus dem Netz
- Suchen von Technologien welche Verschlüsseln und Signieren unterstützen
- Entscheiden über Wahl der Technologien
- Aufteilung der Arbeit
- Entwickeln
- Dokumentieren
- Schlüssellgeneration mit openssl
- Testumgebung aufsetzen/konfigurieren (Tomcat / Apache)
- Vorbereitung der Präsentation (Homepage etc.)
- Präsentation

3.1.2 Infrastruktur

Gebäude:	ZHW Winterthur, Gebäude E
Raum:	E523
Rechner:	2 PCs
CPU:	PIII-450MHz
RAM:	128MB
Ethernetkarte:	Digital DE 500 (PCI)
Betriebssystem:	SuSE Linux 7.1 Kernel 2.4

3.1.3 Software

TCL:	tclsh8.3
webshell:	3.0b2
OpenSSL:	0.9.6 4 Sep. 2000
GnuPG:	1.0.4
PGP Freeware:	7.0.3
Apache:	1.3.14
Java:	JDK1.3 Java Mail 1.2 JCSI 2.0
JBuilder Professional:	4.0
Tomcat-Jakarta:	3.2.1
KDE:	2.1
KMail:	1.2
Netscape Linux:	4.76
Netscape Windows:	4.75
Outlook Express:	5
sendmail:	8.11.2

4 Einleitung

Ein Problem des Internets, das sowohl die kommerzielle, als auch die private Nutzung behindert, ist die mangelnde Sicherheit dieses Mediums. So kann sich ohne besondere Vorkehrungen niemand sicher sein, dass seine übertragenen Daten beim vorgesehenen Empfänger ankommen, ohne abgefangen, mitgelesen oder verändert zu werden. Im gleichen Masse, wie die kommerzielle Nutzung des Internets wächst, nimmt auch der Bedarf an geschützter Kommunikation zu.

Ein Problem aller email-Systeme welche die beiden verbreiteten Protokolle SMTP oder X.400 verwenden, ist, dass sie nach dem Store-and-Forward-Prinzip arbeiten. Die Nachrichten werden dabei über mehrere Knotenrechner weitergeleitet und so mehrmals im Klartext zwischengespeichert. Wer Zugriff auf einen dieser Knoten hat, kann die ein- und ausgehenden Mails problemlos mitlesen, verändern oder fälschen. Durch Filterprogramme ist es möglich, automatisch nach bestimmten Adressen oder Stichworten zu suchen. Letzteres ist sicher ein Feature, das nicht nur für Geheimdienste und Wirtschaftsspione, sondern auch für die Werbeindustrie von Interesse sein dürfte.

Ein weiteres Problem besteht darin, dass jemand problemlos eine email versenden kann, und bei dieser die kompletten Daten der Absendererkennung beliebig abändern kann. Der Empfänger kann also nicht einmal sicher sein, ob die email überhaupt von der Person erstellt wurde, von welcher sie zu kommen scheint.

Um nun ein email-System sicherer zu machen, gilt es diese Probleme zu lösen. In den folgenden Kapiteln wird zuerst auf die Anforderungen eingegangen, welche man an sichere emails stellt und deren technische Umsetzung. Anschliessend werden konkrete Produkte vorgestellt und diese miteinander verglichen. Danach folgen die drei ausgewählten Technologien GnuPG, OpenSSL und Java-Servlets, deren Beschreibung, sowie deren Konfiguration. Das Handling der Zertifikate sowie das Aufsetzen der Webserver wird am Schluss beschrieben.

5 Grundlagen

5.1 email Security

Wie im vorigen Kapitel angedeutet, gibt es viele Gründe, die Emailübertragung sicherer zu gestalten. Um kompatibel mit den bestehenden Anwendungen und Protokollen zu bleiben wird nur der Inhalt der Nachricht durch Signatur und Verschlüsselung entsprechend angepasst, ohne aber die grundlegenden Übertragungsmechanismen und Protokolle zu ändern.

5.1.1 Sicherheitsanforderungen

Das OSI security reference model definiert vier Sicherheitsdienste für elektronische Nachrichten, die von einem email-System unterstützt werden sollen:

- Authentizität
- Integrität
- Nachweisbarkeit des Ursprungs
- Vertraulichkeit

Authentizität bedeutet, dass der Sender einer Nachricht verlässlich festgestellt werden kann. Über die *Integrität* wird gewährleistet, dass eine Nachricht nicht auf unauthorisierte Weise verändert wird, nachdem sie den Sender verlassen hat. Dagegen stellt die *Nachweisbarkeit des Ursprungs* sicher, dass der Sender nicht abstreiten kann, die Daten gesendet zu haben. Die *Vertraulichkeit* einer Nachricht soll verhindern, dass der Inhalt jemandem bekannt wird, der dazu nicht autorisiert ist.

Zur Durchsetzung der obigen Sicherheitsanforderungen werden verschiedene Sicherheitsmechanismen verwendet, die im folgenden genauer beschrieben werden.

5.1.2 Asymmetrische Verschlüsselung

Erst mit den Mitteln der asymmetrischen Kryptographie ist es möglich geworden, die oben genannten Sicherheitsanforderungen in grossem Rahmen zu verwirklichen. Im Gegensatz zur symmetrischen Verschlüsselung, bei der nur ein einziger Schlüssel für Ver- und Entschlüsselung existiert, gibt es bei den asymmetrischen oder public-key-Verfahren ein Schlüsselpaar bestehend aus einem privaten (geheimen) Schlüssel und einem Schlüssel der öffentlich bekannt gemacht wird. Die beiden Schlüssel verhalten sich dabei gegensätzlich, d.h. was mit dem einen verschlüsselt wird, kann mit dem anderen wieder entschlüsselt werden und umgekehrt. Es muss natürlich nahezu unmöglich sein, den privaten Schlüssel aus dem öffentlich bekanntgegebenen Schlüssel zu berechnen, denn davon hängt der grösste Teil der Sicherheit dieses Verfahrens ab. Der bekannteste Vertreter dieser Art ist der nach den Initialen seiner Entwickler (Rivest, Shamir und Adleman) benannte RSA-Algorithmus.

Mit den asymmetrischen Kryptoverfahren lassen sich nun zwei der oben angesprochenen Sicherheitsdienste verwirklichen:

- **Vertraulichkeit:**
Um jemandem eine vertrauliche Nachricht zukommen zu lassen, verschlüsselt man den Text mit seinem öffentlichen Schlüssel. Nur der Empfänger ist dann in der Lage, die Nachricht wieder zu dechiffrieren.

- **Authentizität:**

Verwendet man umgekehrt den eigenen privaten Schlüssel zur Verschlüsselung der Nachricht, so kann der Empfänger durch die Entschlüsselung mit dem eindeutig zugeordneten öffentlichen Schlüssel nachvollziehen, wer der Sender ist (digitale Unterschrift).

5.1.3 Hash-Algorithmen

Ein nützliches Mittel um feststellen zu können ob ein Text verändert wurde, sind sichere Hash-Algorithmen. Diese bilden Daten von beliebiger Länge auf einen deutlich kürzeren Textblock ab, der quasi ein Fingerabdruck der ursprünglichen Nachricht darstellt.

Drei wichtige Eigenschaften hierbei sind,

- der ursprüngliche Text darf aus dem Hashwert nicht wieder hergestellt werden können
- es darf nicht möglich sein, zu einem gegebenen Hashwert einen passenden Text zu konstruieren
- es muss sich ein völlig anderer Fingerabdruck ergeben, wenn sich auch nur ein einziges Bit der Nachricht ändert

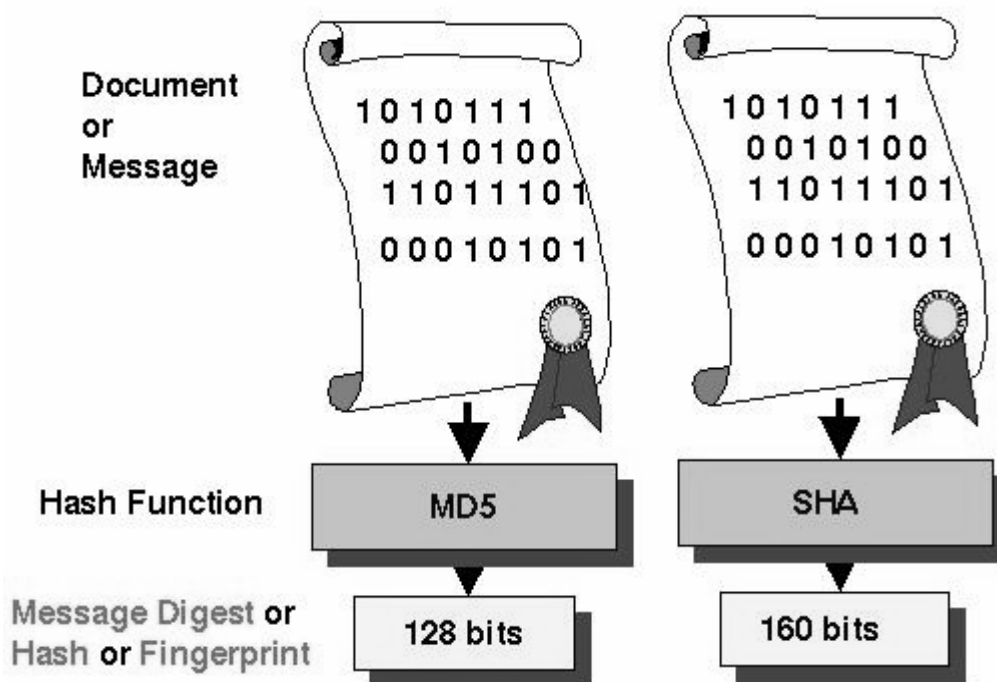


Abbildung 1 – Hash-Algorithmen

Unter diesen Voraussetzungen folgt dann, dass zwei Texte, die denselben Hashwert besitzen, identisch sein müssen. Bekannte Hash-Algorithmen sind SHA-1 oder MD5. Wobei der SHA-1 als der sicherere gilt, da er eine grössere Schlüssellänge besitzt. Zusammen mit der asymmetrischen Verschlüsselung können auf diese Weise digitale Signaturen erstellt werden (Kapitel 5.2.1).

Mit den Hash-Verfahren lassen sich die beiden anderen, der oben angesprochenen Sicherheitsdienste verwirklichen:

- **Integrität:**
Um die Integrität einer Nachricht sicherzustellen, wird ein Hashwert gebildet, den man mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Nur der Empfänger ist dann in der Lage, den Hashwert wieder zu dechiffrieren und zu überprüfen.
- **Nachweisbarkeit des Ursprungs:**
Verwendet man umgekehrt den eigenen privaten Schlüssel zur Verschlüsselung des Hashwertes, so kann der Empfänger durch die Entschlüsselung mit dem eindeutig zugeordneten öffentlichen Schlüssel nachvollziehen, wer der Sender ist und ob die Nachricht verändert wurde.

5.2 Digitale Umschläge

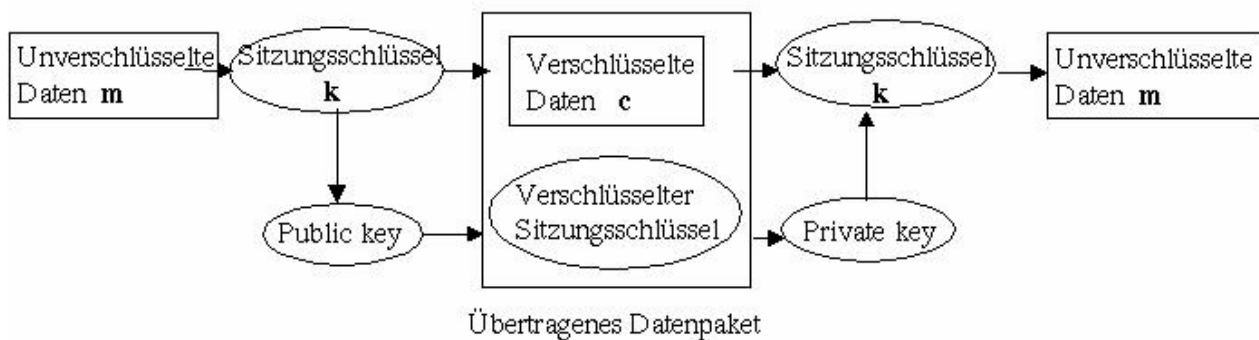


Abbildung 2 – Digitaler Umschlag

Ein Nachteil der public-key-Verschlüsselung ist, dass sie deutlich langsamer ist als symmetrische Verfahren mit vergleichbarer Sicherheit. Dies ist der Hauptgrund für den Einsatz Digitaler Umschläge. Ausserdem ist die rein asymmetrische Verschlüsselung anfällig gegen die sogenannte *Known-Plaintext-Attack*: Da das Chiffrierverfahren und der öffentliche Schlüssel dem Angreifer bekannt sind, kann er versuchen, einen Klartext zu raten, diesen zu verschlüsseln und durch Vergleich zu überprüfen, ob er richtig geraten hat. Daher kombiniert man in der Praxis beide Verschlüsselungsverfahren (Abbildung 2):

- Der Sender verschlüsselt die Nachricht m symmetrisch mit einem zufällig erzeugten Sitzungsschlüssel k und erhält den codierten Text c . Dann verschlüsselt er den Schlüssel k mit dem öffentlichen Schlüssel des Empfängers. Anschliessend versendet er sowohl den verschlüsselten Text c als auch den chiffrierten Schlüssel.
- Der Empfänger dechiffriert den Schlüssel mit seinem private key und erhält wieder k . Dann entschlüsselt er den gesendeten Text c mit k und bekommt die ursprüngliche Nachricht m .

Der Text c und der verschlüsselte Sitzungsschlüssel bilden zusammen den sogenannten „digitalen Umschlag“.

5.2.1 Digitale Signaturen

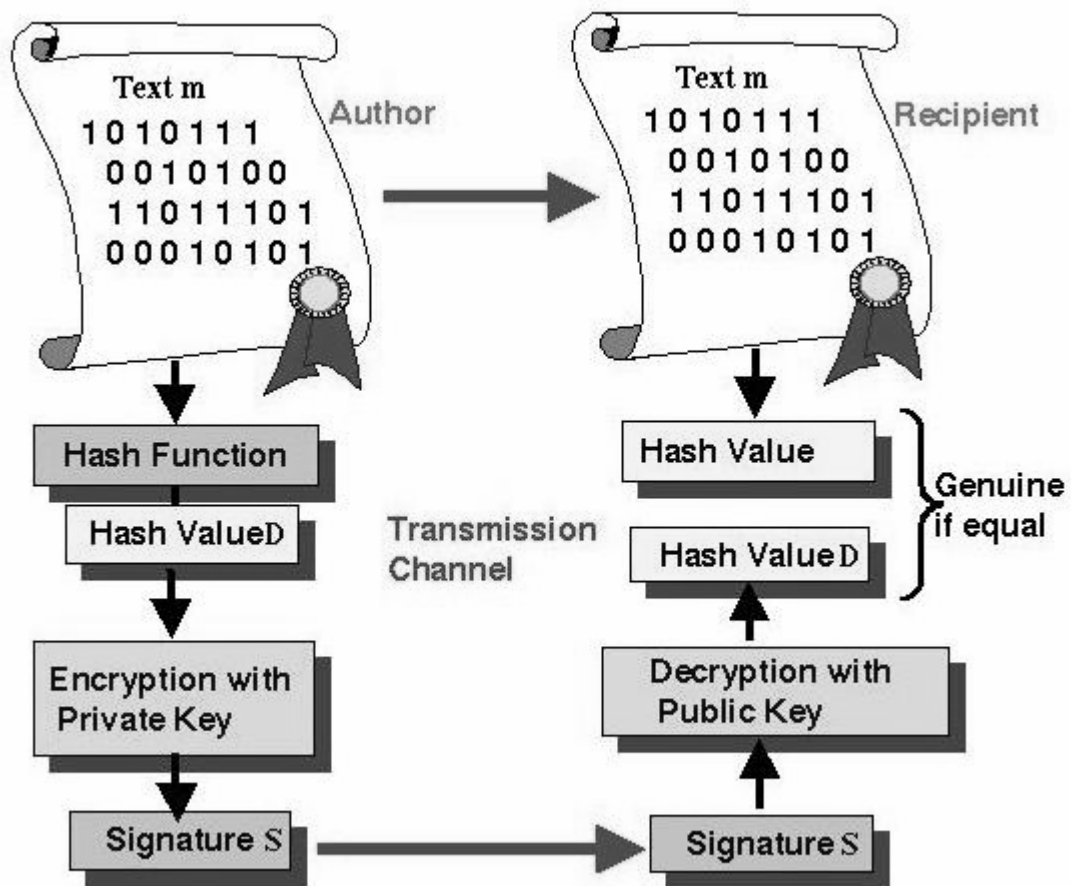


Abbildung 3 – Digitale Signatur

Bei Schriftdokumenten wird die Authentizität normalerweise durch eine Unterschrift bestätigt. Eine digitale Form der Unterschrift muss folgende Eigenschaften erfüllen:

- Nur der rechtmässige Absender eines Dokuments kann die Unterschrift erzeugen
- Der Empfänger des Dokuments kann die Unterschrift zweifelsfrei prüfen
- Die Unterschrift gilt nur im Zusammenhang mit dem gegebenen Dokument

Die typische Implementation einer digitalen Signatur z.B. nach dem RSA-Verfahren erreicht dies durch Verwendung einer Hash-Funktion und einer public-key-Verschlüsselung des Hashwertes (Abbildung 3):

- Der Sender reduziert die Nachricht m auf ihren Fingerabdruck d mit einem sicheren Hash-Algorithmus. Dann verschlüsselt er den Hashwert d mit seinem privaten Schlüssel und erhält die Signatur s . Zusammen bilden der Text m und der chiffrierte Hashwert s die digital signierte

Nachricht. Diese sendet er an den Empfänger.

- Der Empfänger entschlüsselt die Signatur s mit dem public key des Senders und erhält wieder den Hashwert d . Darauf reduziert er die Nachricht m mit demselben Hash-Algorithmus und vergleicht den sich ergebenden Wert mit d . Wenn beide übereinstimmen, kann der Empfänger sicher sein, dass der Text während der Übertragung nicht verändert wurde und tatsächlich vom Besitzer des öffentlichen Schlüssels stammt.

Jeder, der den öffentlichen Schlüssel des Senders kennt, kann auf diese Weise die digitale Signatur der Nachricht überprüfen.

5.3 Echtheit der öffentlichen Schlüssel

Ein zentrales Problem aller email-Systeme, die auf der public-key-Verschlüsselung beruhen, ist es, die Echtheit der öffentlichen Schlüssel sicherzustellen. Oft sind die Teilnehmer so weit voneinander entfernt, dass es keine Möglichkeit für einen persönlichen Schlüsselaustausch gibt. Wenn es aber jemandem gelingt, an die Stelle des Empfängerschlüssels seinen eigenen zu setzen und die Nachricht abzufangen, so kann er den Text problemlos entschlüsseln (*Man-In-The-Middle-Attack*). Diese Angriffsmöglichkeit ist gewissermassen die Achillesferse der asymmetrischen Verfahren. Es muss also sichergestellt werden, dass nur authentische öffentliche Schlüssel verwendet werden.

Grundsätzlich gibt es mehrere Möglichkeiten, die Echtheit eines public key sicherzustellen:

- direkte physikalische Übertragung des Schlüssels (z.B. auf Diskette)
- akustische Verifikation anhand eines sogenannten Fingerprints (z.B. am Telefon)
- Bestätigung des Keys durch einen beiderseits vertrauenswürdigen Partner (z.B. durch eine Signatur)
- Zertifizierung des Schlüssels durch eine etablierte Instanz (z.B. eine zentrale Zertifizierungsbehörde).

Die direkte physikalische Übertragung stellt dabei die sicherste, aber auch umständlichste Möglichkeit dar. Sie ist nicht anwendbar, wenn Sender und Empfänger nicht persönlich miteinander bekannt sind.

Die zweite Alternative ist sicher, wenn sich die Kommunikationspartner am Telefon gegenseitig erkennen. Der Schlüssel wird dabei durch eine für ihn eindeutige Zahlenfolge, den sogenannten Fingerprint, verifiziert. Diese Werte am Telefon zu diktieren, ist jedoch wieder aufwendig und setzt voraus, dass beide Personen telefonisch erreichbar sind.

Die dritte Möglichkeit bedingt, dass Sender und Empfänger eine vertrauenswürdige dritte Person kennen, die die Echtheit der Schlüssel (z.B. durch ihre Unterschrift) garantiert. Dieses Verfahren wird beispielsweise von PGP eingesetzt.

Der letzte und am weitesten verbreitete Weg ist die zentrale Zertifizierung der Schlüssel durch eine etablierte Institution z.B. Verisign. Diese unterhält ein Verzeichnis für die öffentlichen Schlüssel, in dem diese unterschrieben und zusammen mit dem Namen des Besitzers abgelegt sind. Ein Kommunikationspartner kann den öffentlichen Schlüssel eines anderen abrufen und anhand des bekannten Schlüssels der Zertifizierungsstelle die Echtheit überprüfen.

5.4 Zertifikate für öffentliche Schlüssel nach CCITT X.509

Die CCITT-Norm X.509 beschreibt einen Standard für die oben genannte Möglichkeit der zentralen Schlüsselzertifizierung. Ein Zertifikat für einen öffentlichen Schlüssel ist eine Datenstruktur, die einen public key sicher an bestimmte Attribute wie z.B. den Personennamen des Schlüsselbesitzers bindet. Ein Zertifikat nach X.509 (Tabelle 1) bindet einen öffentlichen Schlüssel an einen Verzeichnisnamen und an die Instanz, die sich dafür verbürgt. Diese Instanz wird Certification Authority (CA) genannt. Die vollständige Datenstruktur wird digital signiert.

Jede Instanz gibt ihren Zertifikaten eine eindeutige Seriennummer. Im Herausgeberfeld ist die Institution eingetragen, die sich für das Zertifikat verbürgt. Die Gültigkeitsdauer gibt den Zeitraum an, für den das Zertifikat gilt. Dies ist vergleichbar mit dem Ablaufdatum bei Kreditkarten.

Version	0
Seriennummer	12345
Signatur-Algorithmus	RSA mit MD2, 512
Herausgeber	C=US, S=MA, O=BBN, OU=Comm Div
Gültigkeitsdauer	1/1/98 – 1/1/99
Person	C=US, ..., CN= Max Mustermann
Öffentlicher Schlüssel	RSA, 512, *****
Signatur	

Tabelle 1 – X.509-Zertifikat

Ein Zertifikat wird verifiziert, indem die Signatur mit dem öffentlichen Schlüssel der herausgebenden Instanz überprüft wird. Dies verlagert das Problem, sicher an den Schlüssel des Empfängers zu gelangen, dahin, den public key der herausgebenden Instanz zu bekommen. Diese wird für ihren Schlüssel wieder ein Zertifikat einer anderen Instanz besitzen und so weiter. Daraus ergibt sich ein gerichteter Zertifizierungsgraph, der bei einer hierarchischen Anordnung der Instanzen einen Baum darstellt.

Ein Zertifikat wird ungültig, wenn es zeitlich abgelaufen ist oder die herausgebende Instanz es widerruft, indem sie es auf einer besonderen Liste, der Revocation List, einträgt. In dieser Liste ist die Seriennummer der widerrufenen Zertifikate, das dazugehörige Datum und eventuell der Grund für den Widerruf vermerkt. Bei der Verifizierung eines Zertifikates muss also zusätzlich überprüft werden, ob es nicht in der Revocation List steht und somit ungültig ist.

5.5 Pretty Good Privacy (PGP)

Seit 1991 ist das als Shareware implementierte Pretty Good Privacy, kurz PGP, erhältlich. Durch die Tatsache, dass PGP für nicht kommerzielle Verwendung im C-Quellcode frei verfügbar war, fand dieses Programm zur Verschlüsselung von Daten und email eine schnelle Verbreitung und wurde auf viele Rechnerplattformen konvertiert (darunter MSDOS, Unix und Mac). Von der Firma Network Associates, Inc. wird mittlerweile eine kommerzielle Version vertrieben, wovon der Quelltext nicht mehr frei erhältlich ist. Die nicht kommerzielle Version ist weiterhin frei erhältlich. In der Open Source Szene wird auch ein PGP Produkt unter der GPL entwickelt. Es handelt sich um das GnuPG Projekt. Zeitgleich mit S/MIME ist PGP als OpenPGP und PGP/MIME bei der IETF in der Standardisierungsphase und hat den Stand eines Internet-Drafts erreicht.

PGP brachte seinem Programmierer Phil Zimmermann einerseits den Pioneer Award der *Electronic Frontier Foundation* ein, auf der anderen Seite bescherte es ihm ein staatliches Ermittlungsverfahren wegen Verletzung der Exportbeschränkungen für kryptographische Software.

PGP bietet natürlich auch Funktionen zum Verschlüsseln und/oder Signieren von Nachrichten. Bei der Verschlüsselung verwendet PGP die in Kapitel 5.2 beschriebenen Digitalen Umschläge. Es wird ein zufälliger Sitzungsschlüssel erzeugt, mit dem der Text symmetrisch verschlüsselt wird. Anschliessend wird der Sitzungsschlüssel mit einem public-key-Verfahren chiffriert. Eine PGP SIGNED MESSAGE wie in Abbildung 4 zu sehen, enthält die Daten im Klartext mit angehängter Signatur.

Der Kopf der verschlüsselten Nachricht oder der Signatur enthält lediglich die verwendete Version und gegebenenfalls die verwendete Hashfunktion, alle anderen Angaben werden in einem eigenen 8-Bit-Format gespeichert.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
Welcome to PGP! With PGP for Personal Privacy, you can easily and
securely protect the privacy of your email messages and file
attachments by encrypting them so that only the intended recipients
can read them. You can also digitally sign messages and files, which
ensures their authenticity. A signed message verifies that the
information in it has not been tampered with in any way.
-----BEGIN PGP SIGNATURE-----
Version: PGPFreeware 5.5.3i for non-commercial use <http://www.pgpi.com>
iQA/AwUBNZJuKhOd8wsYiMEOEQILFgCeLGgj4h0RKOSQehTJT/WWezN7mUsAnj9v
bZeTzcmXSB079TTIV+SYbEJO
=K11q
-----END PGP SIGNATURE-----
```

Abbildung 4– email im PGP-Format

5.5.1 PGP Verschlüsselungsverfahren

PGP basiert auf asymmetrischen Verschlüsselungsverfahren, d.h. jeder Benutzer muss ein Schlüsselpaar bestehend aus einem öffentlichen und geheimen Schlüssel generieren. Bei der Kommunikation mit PGP müssen Sender und Empfänger eine Kopie des öffentlichen Schlüssels des jeweils anderen besitzen. Der geheime Schlüssel wird zum Signieren von Nachrichten verwendet und um empfangene verschlüsselte Daten wieder zu dechiffrieren. Umgekehrt benötigt man die öffentlichen Schlüssel anderer, um deren Signaturen zu überprüfen und ihnen verschlüsselte Nachrichten zu senden.

PGP kennt zwei Typen von Schlüsseln:

- Diffie-Hellman/DSS mit einer Schlüssellänge von 512 bis 4096 Bits (1024 für DSS)
- und ElGamal mit beliebiger Schlüssellänge

- teilweise wird auch noch RSA mit einer Schlüssellänge von 512 bis 2048 Bit verwendet. Bei Diffie–Hellman/DSS–Schlüsseln wird ein Teil für die asymmetrische Verschlüsselung nach Diffie–Hellman verwendet und ein anderer Teil für das Signieren von Nachrichten nach dem NIST *Digital Signature Standard* (DSS). RSA–Schlüssel werden lediglich unterstützt, um kompatibel zu älteren Versionen von PGP zu bleiben.

Jedes Schlüsselpaar ist einem Namen und einer email–Adresse zugeordnet und besitzt neben einem Zeitstempel ein bei der Erzeugung definiertes Verfallsdatum. Der geheime Schlüssel wird durch ein Kennwort, die sogenannte Mantra, geschützt. Es besteht die Möglichkeit Schlüssel mit anderen Schlüsseln zu signieren. So kann man z.B. seinen neu erzeugten Schlüssel mit seinem alten unterschreiben. Zu jedem Schlüssel gibt es einen *fingerprint*, der bei der Generierung gebildet wird und es erlaubt zu überprüfen, ob die Kopie des Schlüssels echt ist.

Die Schlüssel werden in Schlüsselbünden, sogenannte Keyrings, verwaltet. Der Keyring enthält für jeden Schlüssel ein Vertrauensflag (Trust), ein Gültigkeitsfeld (Validity) und die für diesen Schlüssel vorhandenen Signaturen (Signatures).

5.5.2 Web of Trust

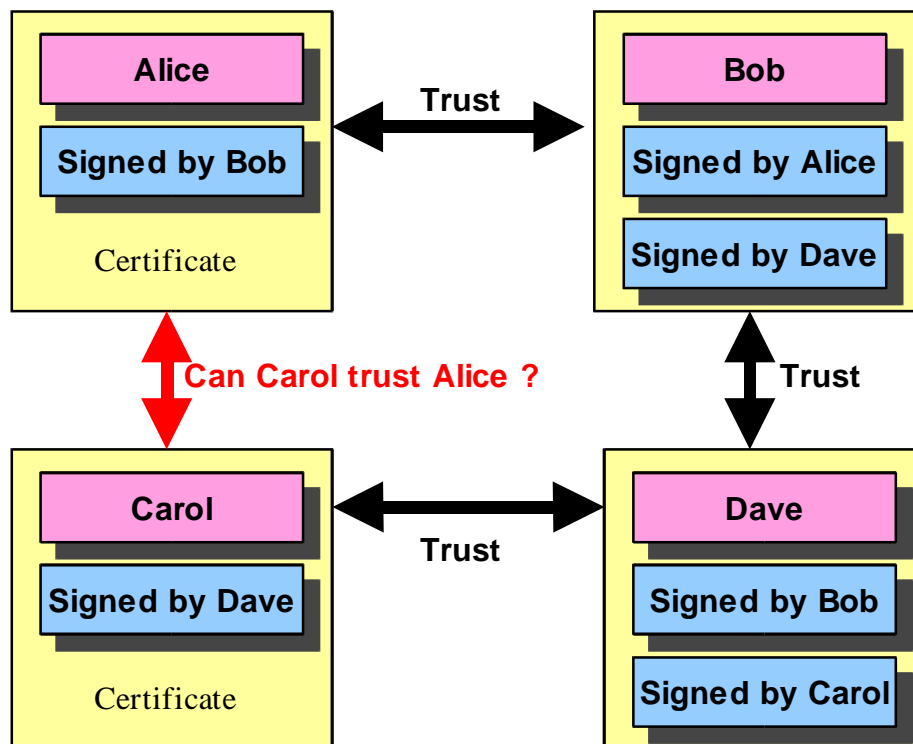


Abbildung 5– PGP Web of Trust

PGP verwendet zur Sicherstellung der Echtheit der Schlüssel ein System, das auf Vertrauen beruht. Im Trust–Feld wird vermerkt, wie sehr dem zugeordneten Benutzer vertraut wird, andere Schlüssel zu signieren. Dieser Wert muss PGP manuell mitgeteilt werden. Aus den Vertrauenswerten der Signaturen eines Schlüssels berechnet PGP die Gültigkeit, d.h. ein Mass für die Wahrscheinlichkeit, dass der Schlüssel authentisch ist. Die Schlüssel, die mit dem eigenen public key signiert wurden, besitzen die höchste Gültigkeit. Besitzt ein Schlüssel keine Signatur, so wird er als ungültig ange–

nommen und der Benutzer erhält bei seiner Verwendung einen entsprechenden Hinweis. Der Vorteil dieses Systems auf Vertrauensbasis ist, dass es sowohl zentralisiert mit einer Zertifizierungsinstanz arbeitet, als auch dezentralisiert, wie es im privaten Umfeld vorkommt. Ein Nachteil ist, dass es keine hierarchische Anordnung der Instanzen erlaubt und nicht die Möglichkeit bietet, weitere Attribute an den öffentlichen Schlüssel zu binden.

5.5.3 Symmetrische Verschlüsselungsverfahren

Die aktuelle PGP-Versionen bieten die Verwendung von mehreren symmetrischen Algorithmen an:

- CAST
- Triple-DES (Data Encryption Standard)
- Blowfish
- Rijndael

Weitere Infos zu den Verschlüsselungsalgorithmen befinden sich im Glossar.

5.5.4 Digitale Signaturen bei PGP

Durch eine digitale Signatur stellt PGP die Authentizität und Integrität einer Nachricht sicher. Beim Signieren wird mit dem privaten Schlüssel des Senders ein *message digest* verschlüsselt. Der Empfänger (oder jeder andere) kann mit den öffentlichen Schlüssel des Senders die Signatur überprüfen. Dadurch ist nachvollziehbar, wer den Text gesendet hat und ob er intakt angekommen ist. Der Sender dagegen kann nicht abstreiten, dass die erzeugte Signatur von ihm stammt.

Der *message digest* wird durch eine sichere Hash-Funktion berechnet und ist 160 oder 128 Bit lang. Der hierfür in PGP zur Zeit verwendete Algorithmus ist der *Secure Hash Algorithm* (SHA), der von der NSA für das National Institute of Standards and Technology (NIST) entwickelt wurde. PGP verwendet den SHA nur mit DSS-Schlüsseln. Aus Gründen der Kompatibilität wird bei RSA-Schlüsseln der *Message Digest Algorithm* (MD5) benutzt. MD5 erzeugt einen 128 Bit langen Hashwert.

5.6 Secure / Multipurpose Internet Mail Extensions (S/MIME)

Die *Secure / Multipurpose Internet Mail Extensions* erweitern den MIME-Standard für multimediale email um die im Kapitel 5.1.1 beschriebenen Sicherheitsdienste. Dies wird wie bei PGP durch digitale Signaturen und public-key-Verschlüsselung erreicht. S/MIME ist nicht auf email beschränkt, sondern kann überall dort eingesetzt werden, wo MIME-Daten transportiert werden können, z.B. mit HTTP.

Zur Zeit wird S/MIME als Internet-Standard vorbereitet. Problematisch ist hierbei, dass die RSA Data Security als Erfinder von S/MIME einerseits nicht die Kontrolle verlieren möchte, und auf der anderen Seite die Internet Engineering Task Force (IETF) sich gegen diesen Versuch der Einflussnahme wehrt. Aktuell ist zur Zeit der Entwurf S/MIMEv3, der neue Krypto-Algorithmen und Signaturverfahren als Alternative zu den patentgeschützten RSA-Techniken einführt.

5.6.1 Multipurpose Internet Mail Extension (MIME)

Durch den Internet-Standard RFC 822 wird der Aufbau von Textnachrichten im Internet festgelegt. Er ist nur für Textformate ausgelegt und erlaubt lediglich die Verwendung von 7-Bit-Zeichen nach dem US-ASCII-Standard. Dies schliesst nichtamerikanische Sonderzeichen wie ä, ö, ü und ß, aber auch andere 8-Bit-Daten wie Audio oder Video aus.

Um diesen Beschränkungen zu entgehen wurden die *Multipurpose Internet Mail Extensions* (MIME) entwickelt. MIME erweitert die RFC 822 um eine Codierung für multimediale Nachrichten. Der Kopf einer email wird um einige zusätzliche Felder erweitert:

- MIME-Version,
- Content-Type: legt die Art des Nachrichteninhalts fest, z.B. Text, Grafik oder Audio,
- Content-Transfer-Encoding: beschreibt das verwendete Kodierungsverfahren

Als Content-Type wurden sieben Nachrichtenklassen festgelegt, die wiederum Unterklassen beinhalten können:

Content-Type	mögliche Untertypen (unvollständig)
Text	plain, richtext, html
Message	rfc822, partial
Image	gif, jpeg
Audio	basic
Video	mpeg, quicktime
Application	octet-stream, postscript, pdf, msword, zip, ...
Multipart	mixed, alternative, digest, encrypted

Tabelle 2– MIME Content Types

Der Typ Multipart erlaubt es, einen Nachrichtenrumpf aus mehreren einzelnen Rumpfteilen (Body Parts) zusammenzusetzen. Die festgelegten Nachrichtentypen können um eigene erweitert werden, die dann mit X- beginnen müssen.

5.6.2 PKCS und S/MIME

S/MIME basiert auf den Public Key Cryptography Standards (PKCS) (Tabelle 3). PKCS wurde 1991 von einem Konsortium der Computer-Industrie geschaffen, darunter Apple, Digital, Lotus, MIT, RSA und Sun. Besonders hervorzuheben ist der PKCS #7, Cryptographic Message Syntax, der das Format von digitalen Signaturen und digitalen Umschlägen definiert. Ausser in S/MIME wird PKCS #7 in zahlreichen anderen Standards verwendet. Mastercard und Visa beispielsweise verwenden es als Grundlage für die SET-Spezifikation, einem Standard für sichere Online-Transaktionen mit Kreditkarten.

Übersicht über den Public Key Cryptographic Standard (PKCS)	
PKCS #1	RSA Encryption Standard
PKCS #3	Diffie–Hellmann Key–Agreement Standard
PKCS #5	Password–Based Encryption Standard
PKCS #6	Extended–Certificate Syntax Standard
PKCS #7	Cryptographic Message Syntax Standard
PKCS #8	Private–Key Information Syntax Standard
PKCS #9	Selected Attribute Types
PKCS #10	Certification Request Syntax Standard
PKCS #11	Cryptographic Token Exchange Syntax Standard
PKCS #12	Personal Information Exchange Syntax Standard
PKCS #13	Elliptic Curve Cryptographic Standard

Tabelle 3– Public Key Cryptographic Standard

In PKCS #7 werden sechs verschiedene Inhaltstypen (contentType) definiert:

- data (unverschlüsselte 8–Bit–Daten)
- signedData (signierte Daten)
- envelopedData (verschlüsselte Daten mit chiffriertem Sitzungsschlüssel)
- signedAndEnvelopedData (signierte und verschlüsselte Daten)
- digestData (Signatur)
- encrypted Data (verschlüsselte Daten ohne Schlüssel).

In S/MIME werden davon hauptsächlich signedData, envelopedData und signedAndEnvelopedData (die Kombination aus den beiden erstgenannten) verwendet.

5.6.3 Die erweiterten MIME–Typen

S/MIME erweitert MIME um die folgenden drei Inhaltstypen:

- application/pkcs7–mime: für SignedData und EnvelopedData
- multipart/signed – application/pkcs7–signatur: für Klartext plus Signatur
- application/pkcs10: für Zertifizierungsanfragen

Der application/pkcs7–mime–Typ wird verwendet um PKCS#7–Objekte darunter enveloped–Data und signedData zu übertragen. Da PKCS#7–Objekte Binärdaten sind, wird in den meisten Fällen base–64 als Transportcodierung verwendet. Um dem Empfänger bei der Unterscheidung der verschiedenen PKCS#7–Typen zu helfen, sollte im Nachrichtenkopf der optionale Parameter smime–type angegeben sein.

Im folgenden sind zwei Beispiele für envelopedData () und signedData () angegeben.

Content–Type: application/pkcs7–mime; smime–type=**enveloped–data**;
name=smime.p7m

Content–Transfer–Encoding: base64

Content–Disposition: attachment; filename=smime.p7m

```
rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```

Abbildung 6 – email im S/MIME Format (envelopedData)

```
Content-Type: application/pkcs7-mime; smime-type=signed-data;  
name=smime.p7m
```

```
Content-Transfer-Encoding: base64
```

```
Content-Disposition: attachment; filename=smime.p7m
```

```
567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj7  
77n8HHGT9HG4VQpfyF467GhIGfHfYT6rfvbnj756tbBghyHhHUujhJhjH  
HUujhJh4VQpfyF467GhIGfHfYGTTrfvbnjT6jH7756tbB9H7n8HHGghyHh  
6YT64V0GhIGfHfQbnj75
```

Abbildung 7 – email im S/MIME Format (signedData)

Der Nachteil, einen signierten Text als PKCS#7-Objekt zu versenden, ist, dass die Nachricht nicht mehr ohne eine entsprechende S/MIME-Anwendung lesbar ist. Soll die signierte email an eine Empfängergruppe gesendet werden, in der Benutzer sind, die kein S/MIME verwenden, ist dieses Verfahren natürlich ungeeignet. In diesem Fall wird das multipart/signed-Format verwendet. Dieser MIME-Typ besteht aus zwei Teilen, von denen der erste die zu signierende MIME-Nachricht im Klartext enthält und der zweite Teil die Signatur. Im protocol-Feld muss dann application/pkcs7-signature angegeben sein.

```
Content-Type: multipart/signed;  
protocol="application/pkcs7-signature";  
micalg=sha1; boundary=boundary42
```

```
--boundary42  
Content-Type: text/plain
```

```
This is a clear-signed message.
```

```
--boundary42  
Content-Type: application/pkcs7-signature; name=smime.p7s  
Content-Transfer-Encoding: base64  
Content-Disposition: attachment; filename=smime.p7s
```

```
ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6  
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj  
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4  
7GhIGfHfYT64VQbnj756
```

```
--boundary42--
```

Abbildung 8 – email im S/MIME Format (multipart/signed-Format)

Um einen Text zu signieren und gleichzeitig zu verschlüsseln, ist es erlaubt, die Formate für signierte und verschlüsselte Nachrichten beliebig zu verschachteln. Dies ist möglich, da jedes der Formate wieder eine MIME-Einheit darstellt.

Das application/pkcs10-Format wird genutzt, um Zertifizierungsanfragen nach PKCS#10 an eine CA zu übertragen, die als Antwort ein Zertifikat für die übertragenen Daten erstellt.

5.6.4 Kryptographische Algorithmen

Ein besonderes Designziel neben der Kompatibilität zwischen verschiedenen Herstellern und der einfachen Benutzbarkeit von S/MIME war die Exportierbarkeit. Aus diesem Grunde enthält S/MIME die symmetrische Block-Chiffre RC2, deren Schlüssellänge variabel ist. RC2 wird mit einer Schlüssellänge von 40 Bit verwendet, die heutzutage kryptographisch nicht mehr als sicher einzuschätzen ist.

S/MIME verwendet die drei symmetrische Verschlüsselungsalgorithmen DES, Triple-DES sowie RC2 mit 40 Bit Schlüssellänge.

Ursprünglich wurde ausschliesslich das RSA-Verfahren nach PKCS #1 für die public-key-Verschlüsselung genutzt. Neu hinzugekommen ist hier die Diffie-Hellman-Verschlüsselung.

Algorithmen für die Berechnung des Message Digest sind MD2, MD5 und SHA-1. Nach Version3 soll der Digital Signature Standard (DSS) in Zukunft das RSA-Verfahren zur Erzeugung der Signatur ersetzen.

5.6.5 Zertifikate für öffentliche Schlüssel

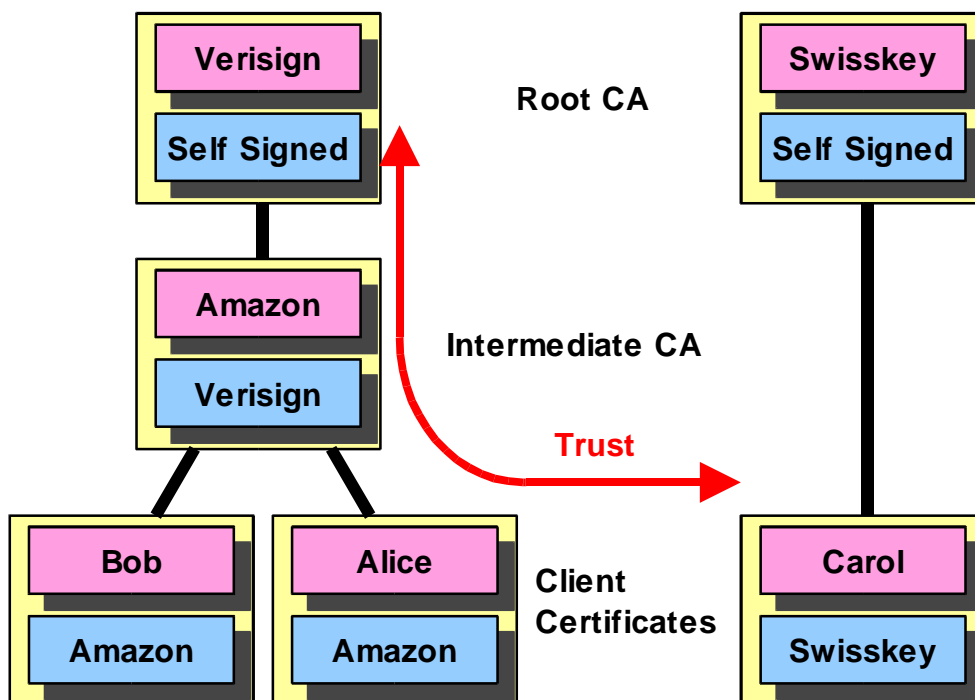


Abbildung 9 – Trust Hierarchie with Certification Authorities

S/MIME verwendet für die öffentlichen Schlüssel Zertifikate nach X.509 (Kapitel 5.4). Durch die Kennzeichnung des Herausgebers und die Seriennummer eines Zertifikates erfolgt eine eindeutige Zuordnung zwischen Schlüssel und Zertifikat. Damit ist eine hierarchische Anordnung von Zertifizierungsinstanzen möglich. Um selber Schlüssel und Zertifikate zu generieren siehe Kapitel 10.

5.7 Vergleich der vorgestellten Systeme

Kriterium	PGP	S/MIME
Erscheinungsjahr	1991	1997
Internet-Standard	RFC 1991, 2015	RFC 2311, 2312
Symmetrische Verschlüsselung	CAST, Triple-DES	RC2/40, DES, Triple-DES
Asymmetrische Verschlüsselung	DH/DSS, RSA	RSA, DH/DSS
Hash-Algorithmen	MD5, SHA-1	MD5, SHA-1
Inhaltsdaten	Binär	Binär
Attachments	Nach MIME	Nach MIME
Verbreitung	Privat	Kommerziell bis privat
Zertifikate	Web of Trust	X.509
Besonderheiten	Im Quelltext verfügbar	MIME-Inhalt

Tabelle 4 – Vergleich der Systeme

Man sieht, dass es gute Sicherungsmöglichkeiten für email gibt. Wie zu Anfang gefordert, lässt sich gewährleisten, dass übertragenen Daten beim vorgesehenen Empfänger ankommen, ohne mitgelesen oder verändert zu werden.

Wie in obiger Aufstellung ersichtlich funktionieren alle diese Systeme mit ähnlichen Verschlüsselungsalgorithmen und Schlüsselzertifizierung. Man sollte jedoch auf eine ausreichende Schlüssellänge achten. Ebenfalls sollte man immer die Echtheit der öffentlichen Schlüssel überprüfen. Nur weil es sich um ein signiertes und verschlüsseltes email handelt, darf man nicht darauf vertrauen, dass es sich dabei automatisch um eine authentische und unveränderte Nachricht handelt.

PGP ist aufgrund der flachen Zertifizierungsstruktur eher für den privaten Bereich geeignet S/MIME dagegen mehr für den kommerziellen Bereich, da mit der hierarchischen Struktur besser Unternehmens- bzw. Behördenstrukturen nachgebildet werden können.

6 Übersicht

Übersicht über die von uns gewählten Lösungen:

	GnuPG	Openssl-smime	Servlet-smime
Serverplattform	Linux	Linux	Linux / Windows
Signieren	SHA1	SHA1	SHA1
Verschlüsseln	–	TripleDES	TripleDES
Lizenz	GPL	Apache-style	Eigene
Clientplattform	Linux / Windows	Linux / Windows	Linux / Windows
Technologie	GnuPG	S/MIME	S/MIME
Authentizität	Ja	Ja	Ja
Integrität	Ja	Ja	Ja
Ursprungsnachweis	Ja	Ja	Ja
Vertraulichkeit	Nein	Ja	Ja
Schlüssel-Passwort	Keines	Mehrere Möglichkeiten	In File
Entwicklungsaufwand	Klein	Klein	Mittel
Erweiterbarkeit	Begrenzt	Begrenzt	Beliebig
Mail Subject	Definierbar	Definierbar	Definierbar
From Adresse	Definierbar	Definierbar	Definierbar
Reply-To Adresse	Aus Formular	Aus Formular	Aus Formular
Mailer	Sendmail	Sendmail	Java Mail 1.2

Tabelle 5 – Übersicht der gewählten Varianten

Folgende Lösungen haben wir nicht berücksichtigt:

Java Cryptix V3: Die Java Cryptix V3 Klassen ermöglichen ebenfalls eine Java Lösung, jedoch nicht mit S/MIME sondern mit PGP. Wir entschieden uns gegen diese Möglichkeit, da die Klassen nicht mit dem RFC 2440 (OpenPGP) kompatibel sind.

Cryptlib: Gegen die C-Librarys von Peter Gutmann entschieden wir uns, weil C-Applikationen bei Webdiensten nicht sehr verbreitet sind, und GnuPG alle unsere Anforderungen bereits abdeckt.

Perl: Da wir noch keine Erfahrung in der Entwicklung von Perl Programmen hatten, entschieden wir uns für TCL und *webshell*.

7 Grundkonfiguration GnuPG

7.1 Gnu Privacy Guard

Dieser Lösungsvorschlag basiert auf dem Programm GnuPG welches digitale Signaturen erstellen kann. Mit der TCL Erweiterung *webshell* wird dem Benutzer ein Formular zur Eingabe der Daten angezeigt. Beim Übermitteln des Formulars wird eine Skriptfunktion aufgerufen, welche GnuPG mit den benötigten Parametern aufruft und den signierten Text an den Empfänger sendet.

GnuPG entspricht der im RFC2440 festgelegten OpenPGP-Spezifikation und ist kompatibel zu PGP 5.x der Firma NAI. Im Gegensatz zu anderen Verschlüsselungsprogrammen wie beispielsweise PGP ist GnuPG freie Software, was bedeutet, dass der Programm-Quellcode frei verfügbar, frei von Patenten und frei von einschränkenden Lizenzbedingungen ist.

7.2 Installation und Schlüsselgeneration

GnuPG ist bei SuSE Linux 7.1 bereits als RPM-Paket enthalten und wird bei der Standard-Installation auf das System kopiert (Paket `gpg`, Kategorie `sec`). Die neueste Version kann auch als Source-Code jeweils von <http://www.gnupg.org/download.html> kopiert werden. Sowohl das RPM als auch die Sourcen zu GnuPG sind auf der beiliegenden CD im Verzeichnis `/tools/gnupg` enthalten. Informationen über die aktuell installierte Version können mit folgendem Befehl abgefragt werden.

```
valleluk@linux:~ > gpg --version
gpg (GnuPG) 1.0.4
Copyright (C) 2000 Free Software Foundation, Inc.
Home: ~/.gnupg
Unterstützte Verfahren:
Cipher: 3DES, CAST5, BLOWFISH, RIJNDAEL, RIJNDAEL192, RIJNDAEL256,
TWOFISH
Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA, ELG
Hash: MD5, SHA1, RIPEMD160
```

1. Als erstes muss nun ein neues Schlüsselpaar generiert werden. (sollte bereits ein Key existieren, siehe Punkt 2)

```
valleluk@linux:~ > gpg --gen-key
gpg: Warnung: Sensible Daten könnten auf Platte ausgelagert
werden.
Bitte wählen Sie, welche Art von Schlüssel Sie möchten:
  (1) DSA und ElGamal (voreingestellt)
  (2) DSA (nur signieren/beglaubigen)
  (4) ElGamal (signieren/beglaubigen und verschlüsseln)
Ihre Auswahl?
```

Mit GnuPG können verschiedene Typen von Schlüsselpaaren erzeugt werden, doch muss der primäre Schlüssel Unterschriften liefern können. Es gibt daher nur drei mögliche Optionen:

- Option 1: erzeugt zwei Schlüsselpaare, ein DSA-Schlüsselpaar das nur zum Unterschreiben geeignet ist, und ein untergeordnetes ElGamal-

- Option 2: Schlüsselpaar für die Verschlüsselung.
erzeugt nur das DSA–Schlüsselpaar.
- Option 4: erzeugt ein einzelnes ElGamal–Schlüsselpaar, das sowohl zum Unterzeichnen als auch zum Verschlüsseln verwendbar ist.

Als nächstes muss die Schlüsselgrösse angegeben werden. Bei einem DSA–Schlüssel muss diese zwischen 512 und 1024 Bits liegen, ein ElGamal–Schlüssel dagegen, kann zumindest theoretisch, eine beliebige Grösse haben. GnuPG fordert allerdings, dass die Schlüssel nicht kleiner als 768 Bits sind. Wenn Option 1 mit einer Schlüssellänge von mehr als 1024 Bit gewählt wird, hat der ElGamal–Schlüssel die verlangte Grösse, doch der DSA–Schlüssel wird maximal 1024 Bits lang sein. Weiter Infos zu PGP/GnuPG Schlüsseln im Kapitel 5.5. Falls erwünscht, kann für die Schlüssel noch ein Verfallsdatum festgelegt werden. Dies lässt sich später aber immer noch ändern. Im nächsten Schritt wird die Benutzer–ID eingegeben. Als letztes kann noch eine Mantra (Passwort) eingegeben werden. Hier sollte aber **keine Mantra eingeben** werden, sondern einfach mit Return zweimal bestätigt werden. So wird ein Schlüssel ohne Passwort erzeugt. Auf der Homepage des GnuPG–Projekts wird ebenfalls auf die Verwendung von GnuPG in einer automatischen Umgebung eingegangen (<http://www.gnupg.org/de/faq.html#q4.13>). In dieser wird ebenfalls empfohlen kein Passwort für den Secretkey zu verwenden, da das Passwort normalerweise nicht sicherer abgelegt werden kann, als der Schlüssel selbst. Jedoch steht, dass es nicht möglich sei, beim Erstellen der Schlüssel bereits ein Schlüssel ohne Passwort einzugeben, sondern das Passwort erst nachträglich entfernt werden soll. Dies war jedoch bei der von uns verwendeten Version möglich. Sollten aber andere Versionen verwendet werden, als die hier gebrauchte und dieses Problem auftreten, so sollte oben genannte FAQ weiterhelfen.

2. Sollte bereits ein Schlüssel existieren kann natürlich auch dieser verwendet werden.

```
valleluk@linux:~ > gpg --allow-secret-key-import --import keyfile
```

Wird ein Secretkeyring mit mehr als einem Schlüssel verwendet, so muss im Optionsfile der default Key definiert werden.

Nun kann die korrekte Arbeitsweise von GnuPG kontrolliert werden. Am einfachsten erstellt man ein kleines Text–File welches nun signiert und überprüft werden kann.

```
valleluk@linux:~ > gpg --sign -u username test2.txt
```

```
valleluk@linux:~ > gpg --verify test2.txt.gpg
```

```
gpg: Unterschrift vom Mit 28 Mär 2001 10:44:09 CEST, DSA Schl?ssel  
ID F97297BC
```

```
gpg: Korrekte Unterschrift von "root luki valle (root key) <valleluk@zhwin.ch>"
```

Nun läuft der Gnu Privacy Guard korrekt auf dem System.

Um den Schlüssel auf den Client kopieren zu können, muss zuerst der Publickey wieder aus dem Pubring exportiert werden. Dazu muss GnuPG mit den Parametern keyring und export aufgerufen werden. Wird der Parameter keyring nicht korrekt angegeben, so versucht GnuPG einen Schlüssel aus dem Keyring im Homeverzeichnis zu exportieren. Im *Key.txt* befindet sich nun ein Key, welcher z.B. problemlos in Windows importiert werden kann.

```
valleluk@linux:/gnupg > gpg --keyring pubring.gpg --export valle-  
luk@zhwin.ch > key.txt
```

7.3 Aufbau der Webseite

Unsere Test-Website haben wir mit der TCL Erweiterung *webshell* erstellt. Die Testseite besteht aus einem Eingabeformular mit 2 Eingabefeldern, einem „Abschicken“ und einem „Zurücksetzen“ Button. Durch Drücken des „Abschicken“ Buttons wird der Text in den Eingabefeldern überprüft und bei korrekter Eingabe signiert und versendet.



Abbildung 10 – Webseite mit Eingabeformular

Da es bei Shell Skripten unter Linux für einen Angreifer möglich ist, mit besonderen Eingabest-rings Code auf dem System auszuführen, werden als Eingabezeichen nur Zahlen und Buchstaben verwendet. Zusätzlich werden beim Feld für die Bestellung noch Leerzeichen und Zeilenumschaltungen akzeptiert. Die email Adresse wird neben der Korrektheit der Zeichen auch noch auf den logischen Aufbau des Strings überprüft. Dieser muss folgendermassen aufgebaut sein: [Zeichen]+[@]+[Zeichen]+[.]+[Zeichen]. Auszug aus dem *webshell* Skript:

```
proc testeEingaben {} {
    #überprüfen der E-Mail auf Sonderzeichen wie ; oder "
    set eMail [web::formvar eMail]
    # make sure we have alpha-numeric stuff separated by "@"
    if {![regexp -nocase {^([a-z0-9]+)@([a-z0-9]+)\.+([a-z]+)$} \
        $eMail ]} {
        return 1
    }
    #überprüfen der Bestellung auf Sonderzeichen wie ; oder "
    set Bestellung [web::formvar Bestellung]
    if {![regexp -nocase {\A([a-z0-9\n\r[:blank:]]+)\Z}
        $Bestellung]} {
        return 2
    }
    #Keine Fehler, email versenden
    return 0
}
```

Erläuterung der Zeichen:

- `^` Das Vergleichen der Zeichen soll am Anfang der Zeile beginnen.
- `$` Das Vergleichen der Zeichen soll bis ans Ende der Zeile durchgeführt werden.
- `+` Mindestens eine Übereinstimmung.
- `\A` Das Vergleichen der Zeichen soll vom Anfang des Strings erfolgen.
- `\Z` Das Vergleichen der Zeichen soll bis zum Ende des Strings durchgeführt werden.
Text kann sich auch über mehrere Zeilen erstrecken.
- `\n` Neue Zeilen sind auch erlaubt.
- `\r` Carriage–Returns sind auch erlaubt.
- `[:blank:]` Leerzeichen sind auch erlaubt.

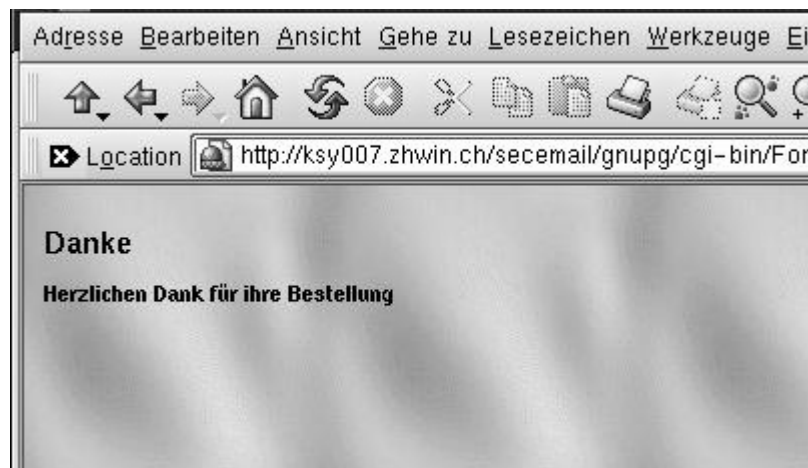


Abbildung 11 – Bestätigung und Dank

Nach erfolgreichem Versenden des emails bekommt der Benutzer eine positive Rückmeldung im Webbrowser. Die Quellen des Skripts liegen der CD im Verzeichnis `/secemail/gnupg/` bei.

7.4 Signier– und Versandvorgang

Nachdem der Text in den Eingabefeldern, sowie der Kommentar für den email–Body richtig formatiert in einer Variabel abgelegt worden ist, kann mit dem Signieren begonnen werden (beides Prozedur `sigEmail`). Wichtig beim Generieren des email Bodys ist, dass am Anfang eine Leerzeile steht. Diese wird benötigt, weil GnuPG anstelle des Passwortes einen Return erwartet. Eventuell könnte in dieser Zeile auch ein allfälliges Passwort angegeben werden.

Auszüge aus dem *webshell* Skript `Formular.ws3`:

```
proc sigEmail {} {
```

```
#GnuPG Binary-File
global gpgbinary

#echo Binary-File
global echobinary

#GnuPG Options-File
global gpgoptionsfile

#Empfänger des eMails aus dem config-file
global toadr

#Die erste Leerzeile ist wichtig, da GnuPG obwohl ohne
Passwort auf den Return wartet.
set emailtxt {

Folgende Bestellung wurde aufgegeben: }
append emailtxt "\nAbsender:\n[web::formvar eMail]\n"
append emailtxt "Bestellung:\n[web::formvar Bestellung]\n"
append emailtxt {
}
set emailaddr { }
append emailaddr [web::formvar eMail]

#Nun wird signiert!
set sigtext [exec $echobinary "$emailtxt" | $gpgbinary
--options $gpgoptionsfile]
sendEmail $sigtext $toadr "[web::formvar eMail]"
}
```

Die als global definierten Variablen und ihre Werte sind im File `./etc/config` gespeichert. Zuerst wird nun der Text für die email an das Programm `/bin/echo` als Parameter übergeben. Der Befehl `/bin/echo` schreibt diesen Text in den Standard Output, von welchem GnuPG ihn über eine Pipe wieder einliest. Um einen sauberen Ablauf des Skriptes zu gewährleisten, müssen noch einige Parameter für GnuPG gesetzt werden. Diese Parameter können direkt im Skript angegeben werden, oder in ein separates Optionsfile geschrieben werden. Wir verwenden das Optionsfile, da uns dies als übersichtlicher erscheint. Der Aufruf `gpg --options [filename]` dient dazu, dass GnuPG nicht versucht das options File im Homeverzeichnis des Users zu lesen, sondern unseres übernimmt. Zusätzliche Einträge im options-File:

<code>clearsign</code>	Durch Verwenden des Attributes <code>clearsign</code> ist der Text für den Empfänger auch noch lesbar, wenn dieser GnuPG nicht verwendet. Alternativ könnte auch das Attribut <code>-s</code> (<code>--sign</code>) angegeben werden. Dadurch ist jedoch der übermittelte Text ohne GnuPG nicht mehr lesbar, da der Text gleichzeitig komprimiert wird.
<code>batch</code>	Für die Verwendung von GnuPG in automatischen Abläufen ist das Attribut <code>batch</code> vorgesehen. Es unterdrückt jeden Output von GnuPG.
<code>Passphrase-fd nr</code>	Hiermit wird das Passwort nicht von der Tastatur gelesen, sondern von einem Filedescriptor. Der Filedescriptor 0 entspricht dem Stan-

	dard Input.
<code>Secret-keyring file</code>	Normalerweise verwendet GnuPG den Schlüsselring mit den geheimen Schlüsseln der im Homeverzeichnis des ausführenden Users liegt. Da aber der User, mit dem der Webserver läuft, kein eigenes Homeverzeichnis hat, teilen wir GnuPG hier mit, wo der secretkey-ring zu finden ist..
<code>Keyring file</code>	Normalerweise verwendet GnuPG den Schlüsselring mit den öffentlichen Schlüsseln der im Homeverzeichnis des ausführenden Users liegt. Da aber der User, mit dem der Webserver läuft, kein eigenes Homeverzeichnis hat, teilen wir GnuPG hier mit, wo der publickey-ring zu finden ist.
<code>No-random-seed-file</code>	Normalerweise wird im Homeverzeichnis des Users ein File mit dem Namen <code>random_seed</code> angelegt. Dies kann zu einer Beschleunigung beim Berechnen von Zufallszahlen führen. Da aber dieser Geschwindigkeitsunterschied in diesem Fall keine Rolle spielt und wir keine Dateien im Homeverzeichnis ablegen können, deaktivieren wir hier dies.
<code>No-secmem-warning</code>	GnuPG meldet bei jedem signieren, dass nun eventuell Daten in einen unsicheren Bereich im Memory oder auf der Platte ausgelagert werden. Diese Meldungen machen jedoch Probleme mit dem <code>exec</code> Befehl. Die Option <code>no-secmem-warning</code> unterdrückt diese Meldung.

Es genügt, wenn das Skript dem gleichen User gehört wie der Webserverdienst und nur mit dem `executable` und `read` Recht für diesen User ausgestattet ist.

```
(chmod 500 echoskript)
```

```
-r-x----- 1 wwwrun users 4780 Mai 9 11:02 Formular.ws3
```

7.5 Tests

Unter Linux lässt sich das korrekte Arbeiten mit KMail 1.2 für das KDE 2.1 problemlos überprüfen. Nachdem der public Key des Servers im lokalen public Keyring des Users hinzugefügt worden ist, erkennt KMail diesen und überprüft die Signatur automatisch.

```
Nachricht enthält Signatur von sender key (key des Servers) <valleluk@zhwin.ch>
Folgende Bestellung wurde aufgegeben:
Absender:
absender@mail.web
Bestellung:
5 Artikel a
10 Artikel b
20 Artikel c

Und bitte keine Werbung
Zahlung per Nachnahme
```

End of pgp message

Abbildung 12 – KMail Ausgabe bei korrekter Signatur

Abgeänderte Message mit 500 Artikel a:

```
Warnung: Fehlerhafte Signatur von sender key (key des Servers) <valleluk@zhwin.ch>
Folgende Bestellung wurde aufgegeben:
Absender:
absender@mail.web
Bestellung:
500 Artikel a
10 Artikel b
20 Artikel c

Und bitte keine Werbung
Zahlung per Nachnahme
```

End of pgp message

Abbildung 13 – KMail Ausgabe bei gefälschtem Mailinhalt

Unter Windows installiert man am einfachsten eine Windows PGP Version. Auch hier sollte das importieren eines Schlüssels keine Probleme darstellen. Die neusten Windows PGP Versionen arbeiten direkt mit dem Mailclient Outlook Express zusammen. Hier geschieht die Überprüfung durch einen Doppelklick auf das email.

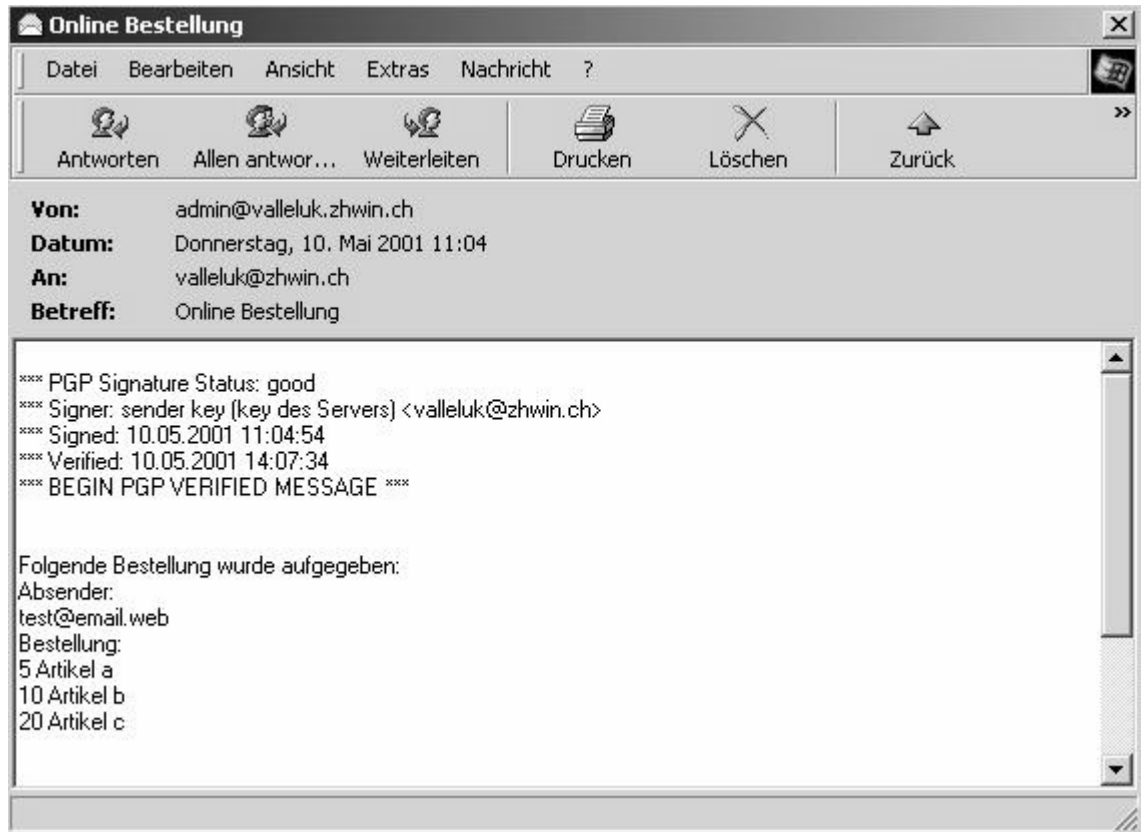


Abbildung 14 – FreePGP Ausgabe bei korrekter Signatur

Abgeänderte Message mit 500 Artikel a:

```
*** PGP Signature Status: bad
*** Signer: sender key (key des Servers) <valleluk@zhwin.ch>
*** Signed: 10.05.2001 11:04:54
*** Verified: 10.05.2001 14:48:08
*** BEGIN PGP VERIFIED MESSAGE ***

Folgende Bestellung wurde aufgegeben:
Absender:
test@email.web
Bestellung:
500 Artikel a
10 Artikel b
20 Artikel c

Und bitte keine Werbung
Zahlung per Nachnahme
```

Abbildung 15 – FreePGP Ausgabe bei gefälschtem Mailinhalt

7.6 Vorteile / Nachteile

Vorteile:

Open Source
Einfaches Key-Handling
Kleines handliches Tool
Client für fast alle Betriebssysteme

Nachteile:

Noch keine Verschlüsselung
Relativ viele verschiedene PGP-Versionen
Öffentliche Zertifizierungsstellen arbeiten nicht mit PGP Schlüsseln
Bei PGP Freeware unter Windows sieht man den Unterschied bad/good fast nicht

7.7 Schnellinstallation

1. GnuPG installieren (Kapitel 7.2)
2. Schlüssel für GnuPG erzeugen (Kapitel 7.2)
3. Webshell installieren (Kapitel 12)
4. gnupg.tar.gz in ein über den Webserver ansprechbares Verzeichnis entpacken
5. In Verzeichnis wechseln: **cd gnupg**
6. Auf alle Files die Rechte setzen
 - **chmod 700 ***
 - **chmod 400 ./etc/***
 - **chmod 500 ./cgi-bin/Formular.ws3**
7. Bei allen Files den Eigentümer und die Gruppe ändern: **chown -R wwwrun:users ***
8. Apache Webserver konfigurieren (Kapitel 13)
9. In der Datei ./gnupg/cgi-bin/Formular.ws3 den Pfad zum configfile ändern
10. In der Datei ./gnupg/etc/options die absoluten Pfade zu den beiden Keyrings setzen
 - Pfad zum Secretkeyring
 - Pfad zum Publickeyring
11. In der Datei ./gnupg/etc/config mindestens folgende Werte überprüfen und anpassen
 - Die Absenderadresse der email
 - Die Empfängeradresse der email
 - Der Pfad zum Optionsfile von GnuPG

8 Grundkonfiguration OpenSSL

8.1 OpenSSL / S/MIME

Diese Variante basiert auf den OpenSSL–Applikationen welche digitale Signaturen erstellen und Inhalte verschlüsseln können. Mit der TCL Erweiterung *webshell* wird dem Benutzer ein Formular zur Eingabe der Daten angezeigt. Beim Übermitteln des Formulars wird eine Skriptfunktion aufgerufen, welche OpenSSL mit den benötigten Parametern aufruft und den signierten und verschlüsselten Text an den Empfänger sendet.

OpenSSL ist eine frei verfügbare Implementierung des SSL/TLS–Protokolls und bietet zusätzliche Funktionen zur Zertifikat–Verwaltung, sowie verschiedene kryptographische Funktionen an. Es basiert auf dem SSLeay–Paket, das von Eric A. Young und Tim Hudson entwickelt wurde.

OpenSSL als Nachfolger von SSLeay wird derzeit von einer unabhängigen Gruppe weiterentwickelt. Das Paket umfasst mehrere Applikationen, z.B. das Erzeugen von Zertifikaten und Zertifizierungsanträgen sowie Verschlüsselungsroutinen. Diese Applikationen können alle über das Kommandozeilen Programm `openssl` angesprochen werden.

8.2 Installation

Openssl ist bei SuSE Linux 7.1 bereits als RPM–Paket enthalten und wird bei der Standard–Installation auf das System kopiert (Paket `openssl`, Kategorie `sec`). Die neuste Version kann auch als Source–Code jeweils von <http://www.openssl.org/source/> kopiert werden. Sowohl das RPM als auch die Sourcen zu OpenSSL sind auf der beiliegenden CD im Verzeichnis `/tools/openssl` enthalten. Informationen über die aktuell installierte Version können mit folgendem Befehl abgefragt werden.

```
valleluk@linux:~ > openssl version  
OpenSSL 0.9.6 24 Sep 2000
```

Falls nicht bereits Zertifikate bestehen, dann können auch selbst Zertifikate erstellt werden (Kapitel 10).

8.3 Aufbau der Webseite

Der Aufbau der Webseite erfolgt genau gleich wie bei GnuPG (Kapitel 7.3)

8.4 Signier– und Versandvorgang

Nachdem der Text der Eingabefelder sowie der Kommentar für das email richtig formatiert in einer Variabel abgelegt worden ist, kann mit dem Signieren begonnen werden (beides Prozedur `sigE–mail`). Diese schreibt den Text für das email in eine temporäre Datei. Dies ist nötig, da wir `openssl` nicht dazu bringen konnten, den Text über eine Pipe korrekt vom Standard Input zu lesen. Der *webshell*–Befehl `web::tempfile` generiert automatisch einen eindeutigen Namen für das temporäre File. Dadurch wird die Datei nicht plötzlich überschrieben, wenn gleichzeitig zwei User etwas bestellen. Danach wird dieses File `openssl` übergeben, signiert, verschlüsselt und als S/MIME email ausgegeben. Nun kann dieser Output direkt über eine Pipe an `sendmail` übergeben werden, welches

das email dann an den Empfänger versendet. Die temporäre Datei wird wieder entfernt. Komischerweise kann aber nach dem Signieren der Output von openssl zum endgültigen Verschlüsseln dann doch über eine Pipe an openssl übergeben werden, und muss nicht wieder den Umweg über ein temporäres File gehen. Ein ähnliches Beispiel aus der Homepage des openssl Projektes geht ebenfalls auf diese Weise vor.

Auszüge aus dem *webshell* Skript Formular.ws3:

```
proc sigEmail {} {  
    #openssl Binary-file  
    global opensslbinary  
  
    #echo Binary-File  
    global echobinary  
  
    #openssl Passwort-Modus und Passwort  
    global opensslpass  
  
    #Sender Zertifikat  
    global sendercer  
  
    #Sender Schlüssel  
    global senderkey  
  
    #Empfänger Schlüssel  
    global receivercert  
  
    # die Absender Adresse aus dem config file  
    global fromadr  
  
    # das Mail-Subject  
    global mailsubject  
  
    #Empfänger des eMails aus dem config-file  
    global toadr  
  
    set emailtxt {  
        Folgende Bestellung wurde aufgegeben: }  
    append emailtxt "\nAbsender:\n[web::formvar eMail]\n"  
    append emailtxt "Bestellung:\n[web::formvar Bestellung]\n"  
    append emailtxt {  
    }  
    set emailaddr { }  
    append emailaddr [web::formvar eMail]  
    set tempfile [web::tempfile]  
    exec $echobinary "$emailtxt" > $tempfile  
    set sigtext [exec $opensslbinary smime -in $tempfile  
        -sign \  
        -passin $opensslpass -text \  
        -signer $sendercer \  
        -inkey $senderkey |\  
        $opensslbinary smime -encrypt \  
        -to $toadr\  
    ]  
}
```

```
        -from $fromadr \  
        -subject $mailsubject\  
        -des3 $receivercert]  
    sendEmail $sigtext $toadr "[web::formvar eMail]"  
    exec rm $tempfile  
}
```

Die als global definierten Variablen und ihre Werte sind im File `/etc/config` gespeichert. Um einen sauberen Ablauf zu gewährleisten können einige Parameter direkt an openssl übergeben werden. Die openssl Parameter:

smime	Als erstes muss openssl der Modus mitgeteilt werden. Hier verwenden wir den Parameter smime. In diesem Modus wird der Inputtext im S/MIME email-Format wieder ausgegeben.
in	Nach dem in Parameter muss der Pfad zum Inputtext angegeben werden. Bei uns ist es die vorhin im <code>/tmp/-</code> Verzeichnis erstellte Textdatei.
sign	Der eingelesene Text soll signiert werden.
passin	Über den Parameter passin kann openssl mitgeteilt werden, woher openssl das Passwort für den Secretkey bekommt. Openssl bietet fünf Möglichkeiten an, um zum Passwort zu gelangen. pass:password Das Passwort wird direkt in den openssl Aufruf integriert env:var Das Passwort ist in der Umgebungsvariablen „var“ abgelegt file:pathname Das Passwort ist in der Datei „pathname“ abgelegt fd:number Das Passwort wird vom Filedescriptor mit der Nummer „number“ gelesen Stdin Das Passwort wird vom Standard Input gelesen
text	Automatisches setzen des text/plain Mailheaders beim Output
signer	Das Zertifikat des Benutzers, mit welchem signiert werden soll.
inkey	Der Private Key des Benutzers, welcher den Text signiert
to	Hier wird ein To:-Wert für den email-Header übergeben
from	Hier wird ein From:-Wert für den email-Header übergeben
subject	Hier kann die Betreffzeile der Mails gesetzt werden
des3	Der absolute Pfad zum Zertifikat des Empfängers, mit welchem die email schliesslich noch verschlüsselt wird. Wichtig ist dabei, dass dieser Vorgang erst nach den Parametern für die email durchgeführt wird.

Es genügt, wenn das Skript dem gleichen User gehört, wie der Webserver läuft und nur mit dem

executable und read Recht für diese User ausgestattet ist.

```
(chmod 500 echoskript)
```

```
-r-x----- 1 wwwrun users 4543 Mai 10 00:34 Formular.ws3
```

8.5 Tests

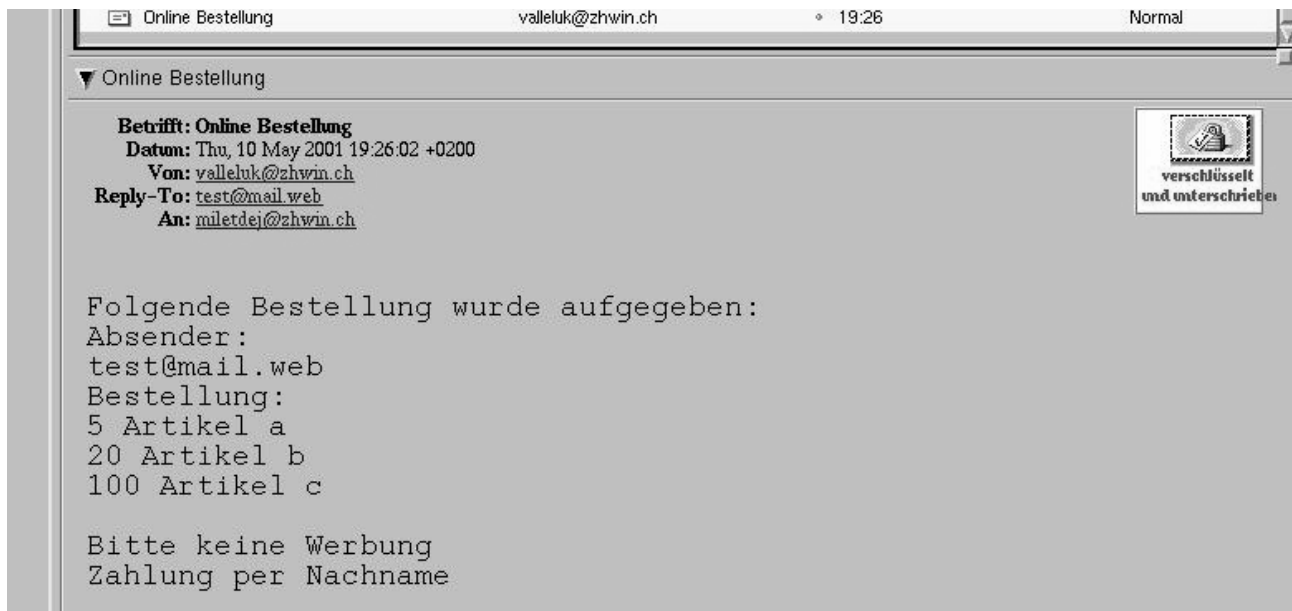


Abbildung 16 – Netscape Ausgabe bei korrekter Signatur



Abbildung 17 – Netscape Ausgabe bei gefälschter Signatur

Weiter wurde diese Lösung mit Netscape für Windows und Microsoft Outlook Express ebenfalls erfolgreich getestet.

8.6 Vorteile / Nachteile

Vorteile:

Open Source

Passwortübergabe auf diversen Wegen möglich

Öffentliche Zertifizierungsstellen arbeiten mit X.509 Schlüsseln

Nachteile:

Keyhandling etwas mühsam, insbesondere wenn Keys selber erstellt werden

Daten werden kurzfristig in ein temporäres File ausgelagert

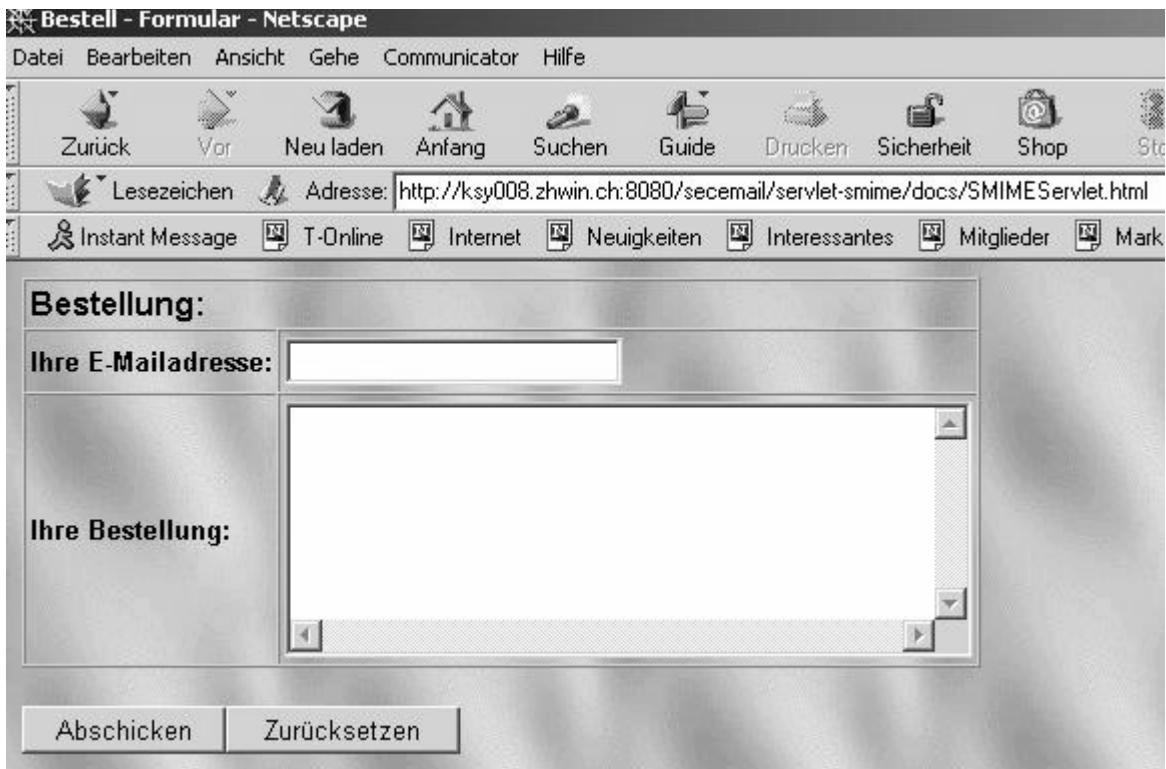
8.7 Schnellinstallation

- 1.Openssl installieren (Kapitel 8.2)
- 2.Schlüssel für Openssl erzeugen (Kapitel 10)
- 3.Webshell installieren (Kapitel 12)
- 4.openssl-smime.tar.gz in ein über den Webserver ansprechbares Verzeichnis entpacken.
- 5.In Verzeichnis wechseln: **cd openssl-smime**
- 6.Auf alle Files die Rechte setzen.
 - **chmod 700 ***
 - **chmod 400 ./etc/***
 - **chmod 500 ./cgi-bin/Formular.ws3**
- 7.Bei allen Files den Eigentümer und die Gruppe ändern: **chown -R wwwrun:users ***
- 8.Apache konfigurieren (Kapitel 13)
- 9.In der Datei ./openssl-smime/cgi-bin/Formular.ws3 den Pfad zum configfile ändern
- 10.In der Datei ./openssl-smime/etc/config mindestens folgende Werte überprüfen und anpassen.
 - Das Passwort für den Secret Key des Servers
 - Pfad zum Zertifikat des Servers
 - Pfad zum Secret Key des Servers
 - Pfad zum Zertifikat des Empfängers
 - Die Absenderadresse
 - Die Empfängeradresse

9 Grundkonfiguration servlet-smime

9.1 Java-Servlet

Mit diesem Lösungsvorschlag werden Mails signiert und verschlüsselt an den Empfänger verschickt. Eine HTML – Seite wird dem Benutzer zur Verfügung gestellt, wo er seine Daten eingeben kann.



The screenshot shows a Netscape browser window with the title 'Bestell - Formular - Netscape'. The address bar contains the URL 'http://ksy008.zhwin.ch:8080/secemail/servlet-smime/docs/SMIMEServlet.html'. The main content area displays a form with the heading 'Bestellung:'. Below this heading, there is a label 'Ihre E-Mailadresse:' followed by a text input field. Underneath the email field is a large text area labeled 'Ihre Bestellung:'. At the bottom of the form, there are two buttons: 'Abschicken' and 'Zurücksetzen'.

Abbildung 18 – Bestell Formular Java Servlet

Nach dem Betätigen des Buttons „Abschicken“, erhält das auf dem Server laufende Servlet diese Eingaben per POST. Die Prozedur „doPost“ wertet diese Eingaben aus und es werden weitere Klassen aufgerufen die ein email kreieren und dieses signiert und verschlüsselt verschicken. Der Empfänger kann nach dem Erhalten des emails bei Fragen oder Unklarheiten mit „Reply-To“ direkt an den Client eine email schicken (Reply-To = email vom Client, From = email vom Server).

Um mit Servlets zu arbeiten wählten wir das JDK 1.3. Für das Verschicken der email ist die JavaMail 1.2 Klasse zuständig, die bei Sun frei erhältlich ist. Für das Signieren und Verschlüsseln der email gibt's die Klassen von JCSI – Java Crypto and Security Implementation, die aber bei kommerzieller Nutzung gewissen Lizenzbedingungen unterliegen, siehe Anhang. Der Apache-Server genügt für die Arbeit von Servlets nicht. Daher installierten wir für diese Variante den Jakarta Tomcat, der sowohl statische Contents (HTML-Seiten) als auch Servlets ausliefern kann.

Die Schlüssel die der Server braucht:

- sender.p12 (CA.cer, receiver.cer & sender key sind enthalten)

mit dem receiver.cer kann er Mails verschlüsseln die der Empfänger lesen kann. Mit dem sender.cer kann er Mails signieren.

Die Schlüssel die der Empfänger braucht:

- receiver.p12 (CA.cer, sender.cer, & receiver key sind enthalten)
somit kann er beim Erhalten eines Mails erkennen ob es vom Sender signiert ist (sender.cer) und er kann es entschlüsseln (receiver key). Er bekommt das CA.cer und weiss somit wer die Instanzierungsstelle ist, und ob er ihr vertrauen kann.

9.2 Installation der Java-Umgebung

Voraussetzungen:

- Mailserver der SMTP- und POP3 Protokoll unterstützt.
- JDK1.3
- JavaMail 1.2

1. Installation von JDK1.3

Mit dem Yast / Yast2 muss das Paket java2 (Java Development Kit 1.3) aus der Kategorie „pay“ installiert werden und das Paket JDK1.18 aus der Kategorie „de“ deinstalliert werden.

2. Einen neuen Link auf das installierte jdk 1.3 setzen

Als root einloggen

Wechseln des Verzeichnisses: **cd /usr/lib**

Löschen des alten Links: **rm java**

Setzen des neuen Links: **ln -s jdk1.3 java**

Wieder als User einloggen

Kontrolle des neuen Links: **java -version**

Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.0)

Classic VM (build 1.3.0, J2RE 1.3.0 IBM build cx130-20001025

(JIT enabled:jitc))

Ort von Programmen wie java, javac: **export PATH=\$PATH:/usr/lib/jdk1.3/bin**

3. Kopieren aller notwendigen „jar’s“

- Von der CD aus dem Verzeichnis /tools/servlet-smime/jar alle jar’s ins Verzeichnis /usr/lib/jdk1.3/jre/lib/ext/kopieren
cp /tools/servlet-smime/* /usr/lib/jdk1.3/jre/lib/ext/

Quellen der Jar’s:

- JavaBeans Activation Framork (activation.jar) erhältlich auf <http://java.sun.com/beans/glasgow/jaw.html> oder auf der CD /tools/servlet-smime/jaf1_0_1.zip
- JavaMail 1.2 (mail.jar, mailapi.jar, smtp.jar, imap.jar, pop3.jar) erhältlich auf <http://java.sun.com/products/javamail/> oder auf der CD /tools/servlet-smime/javamail-1_2.zip

- JCSI – Java Crypto and Security Implementation (jcsi_base.jar, jcsi_jce.jar, jcsi_krb.jar, jcsi_pki.jar, jcsi_provider.jar, jcsi_smime.jar, jcsi_ssl.jar) erhältlich auf <http://security.dstc.edu.au/projects/java/jcsi.html> oder auf der CD /tools/servlet-smime/JCSI-2.0.zip

4. Ändern der Providereinträge in /usr/lib/jdk1.3/jre/lib/security/java.security:

```
#
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun

#DSTC Providers
security.provider.2=com.dstc.security.provider.DSTC
security.provider.3=com.dstc.security.keymanage.keystore.DSTC
security.provider.4=com.dstc.security.x509.DSTC

#
# Default keystore type.
#
#keystore.type=jks
    keystore.type=pkcs12
```

5. Zusätzliche Änderungen bei Arbeit mit JBuilder4.0 professional (Servlet Unterstützung)

Beim Arbeiten mit dem JBuilder 4.0 prof. kann man Servlets direkt testen, indem man den Webserver im JBuilder laufen lässt. (WebRun)

1. Installation ins Verzeichnis /opt/jbuilder4.0
2. Von der CD aus dem Verzeichnis /tools/servlet-smime/jar alle jar's ins Verzeichnis /opt/jbuilder4/jdk1.3/jre/lib/extkopieren.
cp /tools/servlet-smime/* /opt/jbuilder4/jdk1.3/jre/lib/ext
3. Ändern der Providereinträge in /opt/jbuilder4/jdk1.3/jre/lib/security:

```
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun

#DSTC Providers
security.provider.2=com.dstc.security.provider.DSTC
security.provider.3=com.dstc.security.keymanage.keystore.DSTC
security.provider.4=com.dstc.security.x509.DSTC

# Default keystore type.
#
#keystore.type=jks
    keystore.type=pkcs12
```

4. JBuilder starten und unter dem Menu-Punkt *Tools* -> *Configure JDKs* den JDK home Path ändern auf /opt/jbuilder4/jdk1.3

9.3 Installation von Tomcat

Ein Webserver wird so konfiguriert, dass er alle Seiten mit einer speziellen Erweiterung (normalerweise .jsp) an einen JSP–Prozessor weiterleitet. Dieser Prozessor erzeugt beim ersten Aufruf einer JSP–Seite aus dieser ein Servlet, kompiliert es und führt es aus. Dieses Servlet liefert dann eine HTML–Seite zurück. Tomcat ist ein JSP–Prozessor aus dem Apache–Jakarta Projekt. Der Tomcat Server in der Version 3.1 implementiert die Java Servlet Spezifikation in der Version 2.2 und die JSP Spezifikation in der Version 1.1. Nach dem Installieren und Starten des Tomcat, läuft dieser auf der Adresse <http://localhost:8080/>. Nun kann mit Servlets gearbeitet werden.

Tomcat ist erhältlich auf <http://jakarta.apache.org/builds/jakarta-tomcat/release/v3.2.1/> oder auf der CD /tools/jakarta-tomcat-3.2.1.tar.gz.

1. Tomcat 3.2.1 ins Verzeichnis /usr/local/jakarta-tomcat-3.2.1/ installieren
tar -xvfz jakarta-tomcat-3.2.1.tar.gz
2. Im Verzeichnis /usr/local/jakarta-tomcat-3.2.1/webapps ein neues Verzeichnis erstellen mit dem Namen secemail.
cd /usr/local/jakarta-tomcat-3.2.1/webapps
mkdir secemail
3. Auf der CD ins Verzeichnis /secemail/ wechseln und das Verzeichnis servlet–smime ins Verzeichnis /usr/local/jakarta-tomcat-3.2.1/webapps/secemail kopieren
cp -r /cdrom/secemail/servlet-smime/ /usr/local/jakarta-tomcat-3.2.1/webapps/secemail/
4. In der Datei /usr/local/jakarta-tomcat-3.2.1/conf/server.xml folgendes am Ende einfügen:

```
<Context path="/secemail/servlet-smime"
        docBase="webapps/secemail/servlet-smime"
        crossContext="false"
        debug="0"
        reloadable="true"
        trusted="false" >
</Context>
```
5. **export TOMCAT_HOME=/usr/local/jakarta-tomcat-3.2.1** eingeben
6. **export JAVA_HOME=/usr/lib/java** eingeben
7. Tomcat starten mit: **\$TOMCAT_HOME/bin/startup.sh**
(stoppen mit \$TOMCAT_HOME/bin/shutdown.sh)

Das Formular sollte jetzt auf <http://localhost:8080/secemail/servlet-smime/docs/SMIMEServlet.html> aufrufbar sein.

9.4 Keys erstellen

Die von uns erstellten keys liegen in /usr/local/jakarta-tomcat-3.2.1/webapps/secemail/servlet-smime/etc. Es kann mit ihnen gearbeitet werden. Es sind folgende:

CA.cer, sender.cer, sender.p12, receiver.cer und receiver.p12

Der Server braucht lediglich seinen Privatkey sender.p12. Die anderen können gelöscht werden.

Der Empfänger muss in seinem Browser seinen Privatkey receiver.p12 einfügen (Kapitel 11)

Um neue Keys zu erstellen bitte Kapitel 10 ausführen. Falls Keys gelöscht worden sind z.B. bei VeriSign, sollten einfachheitshalber die Namen *sender* für den Server und *receiver* für den Empfänger beibehalten werden.

9.5 Anpassen der Quelldateien

1. Pfad ändern

Im File SMIMESend.java ist der Pfad zum send.properties – File standardmässig gesetzt:
/usr/local/jakarta-tomcat-3.2.1/webapps/secemail/servlet-smime/etc/send.properties.

Beim Installieren muss also darauf geachtet werden, dass dieser Pfad richtig gesetzt wird und dieses File neu compiliert wird.

2. Ändern der Datei send.properties

Wahl des symmetrischen Schlüssels (Empfohlen TripleDES).

```
# Symmetric Cipher; Triple DES = "DESede", RC2 = "RC2"  
cipherAlg=DESede
```

Wahl des Hash-Algorithmus (Empfohlen SHA-1)

```
# Signatur digest Algorithm; "SHA-1" or "MD5"  
digestAlg=SHA-1
```

```
mail.keystore=/usr/local/jakarta-tomcat-3.2.1/webapps/secemail/servlet-  
smime/etc/sender.p12
```

```
mail.store.protocol=pop3
```

```
mail.transport.protocol=smtp
```

```
#Recipient (Empfänger)
```

```
mail.ruser=valleluk@zhwin.ch
```

```
#Senders (Sender)
```

```
mail.suser=miletdej@zhwin.ch
```

```
mail.host=mailserver.zhwin.ch
```

```
# To make a .P12 PKCS#12 file that Outlook Express can load
```

```
# these passwords must be the same.
```

```
keystore.store_password=sender_pw
```

```
keystore.user_password=sender_pw
```

3. Owner der Ordner ändern

Der Ordner /etc/ enthält den private key und das send.properties File, und darf daher nur von Tomcat oder root gelesen werden. Momentan läuft Tomcat noch unter dem Benutzer miletdej. als root ins Verzeichnis/usr/local/jakarta-tomcat-3.2.1/webapps/secemail/servlet-smime wechseln

```
chown -R miletdej:users etc/
```

```
chmod -R 700 etc/
```

4. Rechte der Dateien ändern

```
cd etc/
```

```
chmod 400 *
```

9.6 Versandvorgang

Nach dem Drücken des Buttons „Abschicken“, erhält der Benutzer folgenden Output vom Servlet:



Abbildung 19 – Servlet Output nach versenden der email

Output auf der Console:

```
----- Sending Clear Signed/Encrypted Message -----
o Prepare SMIMEKeyStore ...
o Create email message
From: miletdej@zhwin.ch
To: valleluk@zhwin.ch
Subject: Clear Signed/Encrypted Message
o Sign with keystore owner's signature ...
Evaluation licence will expire 21 June 2001
o Encrypt message for all recipients ...
o Send message to mail server ...
**** Content Type: application/x-pkcs7-mime; smime-type=enveloped-data;
name=smime.p7m
Successfully sent: Online Bestellung
```

9.7 Klassenbeschreibung

Durch das Drücken des Buttons „Abschicken“, erhält das auf dem Tomcat laufende SMIMEServlet die Parameter „mailFrom“ und „message“ per POST. Es wird eine Instanz von der Klasse SMIMESend erzeugt und ihr die beiden Parameter übergeben. SMIMESend ist dafür verantwortlich, dass das Mail zuerst signiert und dann verschlüsselt wird. Zuerst wird aber ein Keystore geladen, wo überprüft wird, ob das angegebene Passwort im File send.properties zum Privaten Key sender.p12 gehört. Danach wird ein Mail mit den Angaben im send.properties erzeugt und mit einer Standard Nachricht verschickt.

Klasse SMIMEServlet

//Parameter mailFrom & message wird vom SMIMEServlet mit der Prozedur doPost empfangen

```
public void doPost(HttpServletRequest request, HttpServletResponse response)throws ServletException, IOException
```

```
...
```

```
    mailFrom = request.getParameter("mailFrom");  
    message = request.getParameter("message");
```

```
....
```

// Instanz von der Klasse SMIMESend wird erzeugt, Prozedur _main(Parameter) ausgeführt

```
    SMIMESend s = new SMIMESend();  
    s._main(message, mailFrom, s);
```

Klasse SMIMESend

// muss selber geändert werden

```
filepath = "/usr/local/jakarta-tomcat-3.2.1/webapps/secemail/servlet-smime/etc/send.properties";
```

```
public void _main(String message, String mailFrom, SMIMESend o)
```

// File send.properties laden

```
    Properties props = o.loadProperties(filepath);
```

```
...
```

// Prozedur sendMessage aufrufen, verschlüsseln, signieren ...

```
    o.sendMessage(props, true, true, false, "Online Bestellung" , message, mailFrom);
```

//Opaque signing hat den Vorteil, dass die email beim Transport nicht verändert wird, aber den

//Nachteil, dass Clients die S/MIME MIME nicht unterstützen die email nicht lesen können.

```
private void sendMessage(Properties props, boolean encrypted, boolean signed, boolean opaque,  
String subject, String message, String mailFrom)
```

```
....
```

// Lädt SMIME Keystore und verbindet mit MailServer

```
    SMIMEKeyStore kssender = new SMIMEKeyStore(props);
```

```
...
```

// verschiedene Infos aus dem File send.properties holen

```
    msg.setFrom(new InternetAddress(mailfrom));
```

```
...
```

// signieren

```
    msg = kssender.sign(msg, opaque);
```

// verschlüsseln

```
    msg = kssender.encrypt(msg);
```

// Versenden

```
    Transport.send(msg);
```

Klasse SMIMEKeyStore

public SMIMEKeyStore(Properties props) throws SMIMEKeyStoreException

```
...
//verschiedene Info's holen
    this.props = props;
    this.owner = getProperty("mail.from"); // owner's email address
```

private void openKeyStore() throws SMIMEKeyStoreException

```
...
// sender.p12 laden und mit Passwort vergleichen
    this.keystore = KeyStore.getInstance("pkcs12" );
    this.keystore.load( fis, storePassword.toCharArray() );
```

private void initEncrypt(Address[] recipients) throws SMIMEException, SMIMEKeyStoreException, KeyStoreException

```
...
// Initialisiert einen SMIMECipher zum Verschlüsseln, mit Triple DES (standard) symmetrischen
Schlüsselalgorithmus
    this.cipher.initEncrypt(rand, cipherAlg, certs);
```

public MimeMessage sign(MimeMessage msg, boolean opaque)throws SMIMEKeyStoreException

```
...
JCSI-Klasse. Initialisiert Message
    this.signature.initSign(digestAlg, keyPriv, certChain, opaque);
...
// signierte Message zurückgeben
    return this.signature.sign();
```

Klasse SMIMEKeyStoreException

gibt Fehlermeldungen aus

9.8 Vorteile / Nachteile

Vorteile:

Java-Programme einfach abzuändern

Pfadänderungen alle in einem File, ausser Pfad zu diesem File

Java läuft auf auf vielen Plattformen

Nachteile:

Key unverschlüsselt auf Harddisk, sowie File mit Private Key
gewisse Lizenzen für kommerziellen Gebrauch

10 Schlüssellgeneration

Von der CD aus dem Verzeichnis `/tools/ssl` die Dateien `openssl_init` und `openssl.cnf` in ein Directory kopieren.

`./openssl_init` ausführen.

Im Home Verzeichnis ist ein Directory `ssl` generiert worden.

`cd Home/ssl`

10.1 Root-Zertifikat erstellen

```
root@linux:~/ssl/PCA > openssl req -config ../openssl.cnf -new
-x509 -days 1460 -newkey rsa:2048 -keyout private/PCAkey.pem -out
PCAcert.pem
```

Using configuration from `../openssl.cnf`

Generating a 2048 bit RSA private key

```
.....
.....+++
....+++
```

writing new private key to 'private/PCAkey.pem'

Enter PEM pass phrase:

Verifying password - Enter PEM pass phrase:

```
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
```

```
Country Name (2 letter code) [CH]:
Locality Name (eg, city) [Zuerich]:
Organization Name (eg, company) [Netcetera]:
Organizational Unit Name (eg, section) [IT]:
Common Name (eg, YOUR name) []:root
Email Address []:peter.kohler@netcetera.ch
```

Kopieren des Root CA Zertifikates

```
root@linux:~/ssl/PCA > cp PCAcert.pem certs/00.pem
```

10.2 User Zertifikate erstellen

```
root@linux:~/ssl/PCA > cd ~/ssl/UCA/
```

10.2.1 Sender

```
root@linux:~/ssl/UCA > openssl req -config ../openssl.cnf -newkey
rsa:1024 -keyout private/senderkey.pem -out senderreq.pem
Using configuration from ../openssl.cnf
Generating a 1024 bit RSA private key
.....+++++
.....+++++
writing new private key to 'private/senderkey.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [CH]:
Locality Name (eg, city) [Zuerich]:
Organization Name (eg, company) [Netcetera]:
Organizational Unit Name (eg, section) [IT]:
Common Name (eg, YOUR name) []:sender
Email Address []:miletdej@zhwin.ch

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

10.2.2 Reciever

```
root@linux:~/ssl/UCA > openssl req -config ../openssl.cnf -newkey
rsa:1024 -keyout private/receiverkey.pem -out receiverreq.pem
Using configuration from ../openssl.cnf
Generating a 1024 bit RSA private key
.....+++++
.....+++++
writing new private key to 'private/receiverkey.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
```

```
Country Name (2 letter code) [CH]:
Locality Name (eg, city) [Zuerich]:
Organization Name (eg, company) [Netcetera]:
Organizational Unit Name (eg, section) [IT]:
Common Name (eg, YOUR name) []:receiver
Email Address []:valleluk@zhwin.ch
```

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

10.3 Zertifikate mit Root Zertifikat unterschreiben

10.3.1 Sender

```
root@linux:~/ssl/UCA > openssl ca -config ../openssl.cnf -name
Root_CA -in senderreq.pem -out sender.pem
Using configuration from ../openssl.cnf
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName             :PRINTABLE:'CH'
localityName             :PRINTABLE:'Zuerich'
organizationName         :PRINTABLE:'Netcetera'
organizationalUnitName   :PRINTABLE:'IT'
commonName               :PRINTABLE:'sender'
emailAddress             :IA5STRING:'miletdej@zhwin.ch'
Certificate is to be certified until May 10 09:35:44 2003 GMT (730
days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

10.4 Receiver

```
root@linux:~/ssl/UCA > openssl ca -config ../openssl.cnf -name
Root_CA -in receiverreq.pem -out receiver.pem
Using configuration from ../openssl.cnf
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName             :PRINTABLE:'CH'
localityName             :PRINTABLE:'Zuerich'
organizationName         :PRINTABLE:'Netcetera'
organizationalUnitName   :PRINTABLE:'IT'
commonName               :PRINTABLE:'receiver'
emailAddress             :IA5STRING:'valleluk@zhwin.ch'
```



```
Certificate is to be certified until May 10 09:36:57 2003 GMT (730
days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

10.5 PKCS12 Dateien erstellen

10.5.1 Sender.p12

```
root@linux:~/ssl/UCA > cat ../PCA/PCAcert.pem receiver.pem >
certssend.pem
root@linux:~/ssl/UCA > openssl pkcs12 -export -inkey private/sen-
derkey.pem -name "miletdej@zhwin.ch" -in sender.pem -certfile
certssend.pem -caname "CA" -caname "valleluk@zhwin.ch" -out
sender.p12
Enter PEM pass phrase:
Enter Export Password:
Verifying password - Enter Export Password:
```

10.5.2 Receiver.p12

```
root@linux:~/ssl/UCA > cat ../PCA/PCAcert.pem sender.pem >
certsrec.pem
root@linux:~/ssl/UCA > openssl pkcs12 -export -inkey
private/receiverkey.pem -name "valleluk@zhwin.ch" -in receiver.pem
-certfile certsrec.pem -caname "CA" -caname "miletdej@zhwin.ch"
-out receiver.p12
Enter PEM pass phrase:
Enter Export Password:
Verifying password - Enter Export Password:
```

Für die Java Lösung werden .p12 Dateien benötigt, während für die openssl Skript Lösung .pem Files benötigt werden. Oft werden aber die Zertifikate von den Zertifizierungsstellen nur als .p12 Dateien abgegeben. Folgendermassen können aus den .p12 Files .pem Files erzeugt werden. Dabei ist zu beachten, dass in diesen .pem Files dann sowohl das Zertifikat als auch der Key gespeichert ist. Es muss also im config-File zweimal das gleiche File angegebenen werden.

```
root@linux:/key/> openssl pkcs12 -in receiver.p12 -clcerts -out
receiver.pem
Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

Die Option clcerts dient dazu, dass nur die Client Zertifikate ausgegeben werden, aber keine CA Zertifikate.

11 Zertifikate

11.1 Netscape 4.76

Damit der Netscape Messenger die emails entschlüsseln und die Signatur überprüfen kann, müssen zuerst die jeweiligen Zertifikate importiert werden. Dazu benötigt man das vorhin erstellte Schlüsselreceiver.p12. Dazu öffnet man im Netscape über das Menu *Communicator* → *Werkzeuge* → *Sicherheits Infos* → *Zertifikate* → *Ihre* folgende Maske.

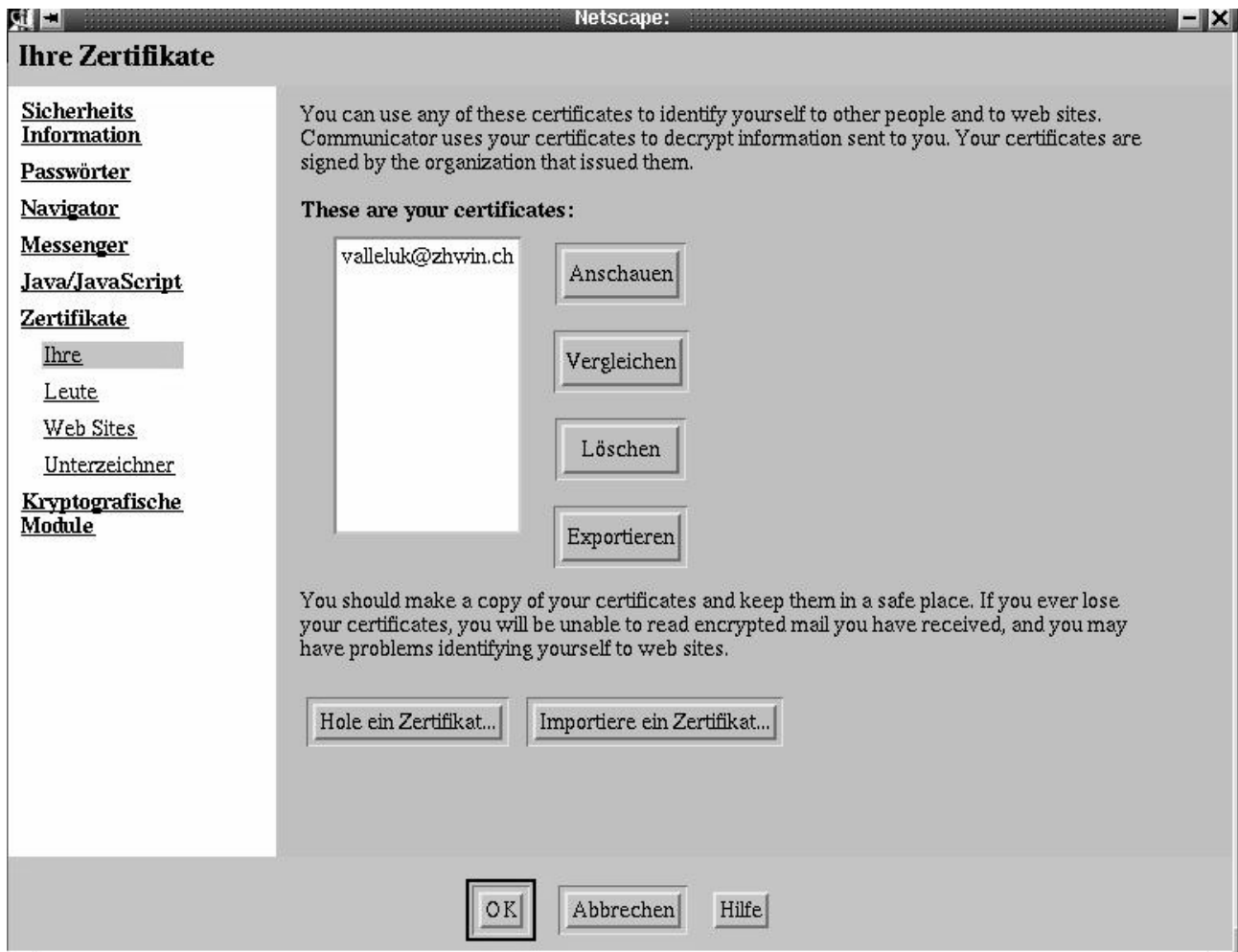


Abbildung 20 – Netscape Zertifikate

Um nun ein Zertifikat zu importieren müssen folgende Schritte durchgeführt werden:

1. Importiere ein Zertifikat
2. Netscape Passwort: *****
3. importieren von receiver.p12
4. password = password aus send.properties (receiver_pw)

Wird das receiver.p12 file verwendet, welches wir im Kapitel 10 erstellt haben, dann wird neben dem eigenen Zertifikat auch gleich das öffentliche Zertifikat des Servers importiert. Nun muss Netscape nur noch mitgeteilt werden, dass wir diesem Zertifikat auch wirklich vertrauen. Dazu muss das Zertifikat des Servers unter dem Punkt *Leute* angeklickt werden. Danach kann über den Button „Anschauen/Editieren“ die Glaubwürdigkeit des Zertifikates eingestellt werden.

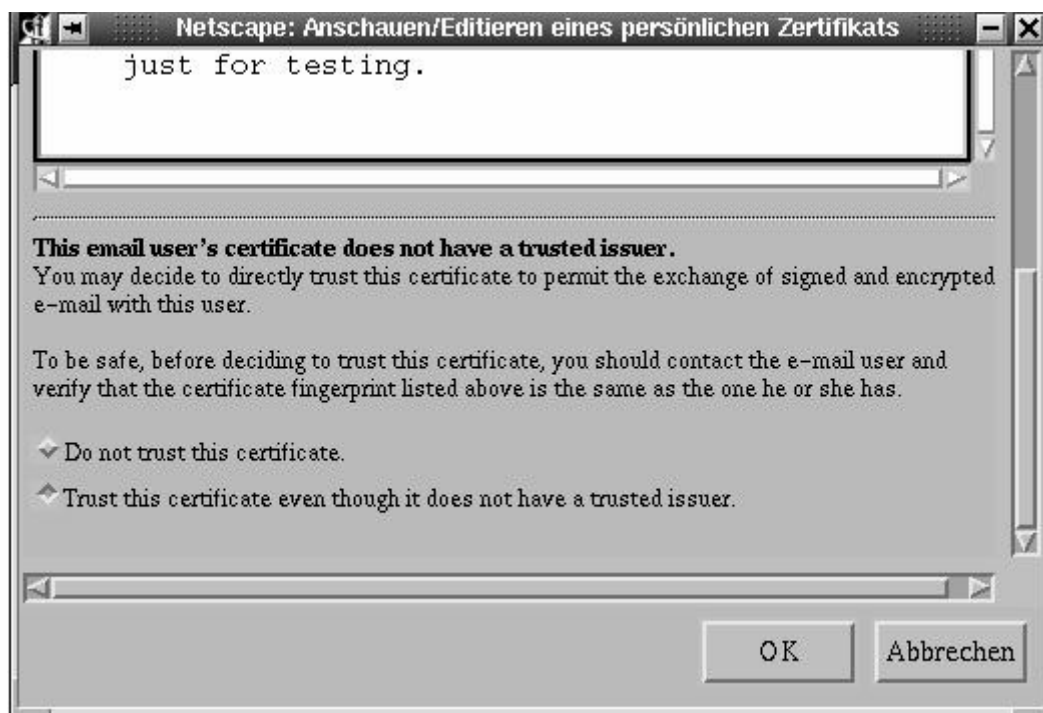


Abbildung 21 – Netscape Glaubwürdigkeit des Zertifikates einstellen

11.2 Internet Explorer

Man muss darauf achten, dass das High Encryption Pack für den Internet Explorer installiert ist. Dieser unterstützt die 128-Bit-Verschlüsselung. Das Pack findet man auf der Seite <http://www.microsoft.com/germany/ie>. Unter dem Punkt *Extras* → *Internetoptionen* kann man unter dem Punkt *Inhalt* → *Zertifikate* den private Key als auch das CA Zertifikat importieren.



Abbildung 22 – Internet Explorer Optionen

12 Webshell / TCL

12.1 Installation TCL

Entpacken des Sourcepaketes und Installation auf dem System:

```
tar -xvzf tcl8.3.2.tar.gz
./configure
make
make install
```

12.2 Installation webshell

Entpacken des Sourcepaketes und Installation auf dem System:

```
tar -xvzf websh-3.0b2.tar.gz
cd websh-3.0b2
cd unix
./configure
make
make install
```

12.3 Apache Modul generieren

```
./configure --with-httpdinclude=/usr/include/apache
make mod_websh.so
cp mod_websh3.0b2.so /usr/lib/apache/mod_websh.so
```

In der Datei /etc/httpd/httpd.conf folgende Zeilen einfügen

```
LoadModule websh_module /usr/lib/apache/mod_websh.so
AddHandler websh-script .ws3
```

Apache neu starten

```
/etc/init.d/apache restart
```

12.4 Error-Logs

Bei Problemen mit den *webshell* Skripten ist oft ein Blick in das Error-Log File /var/log/httpd/error_log hilfreich. Hier werden alle Fehlermeldungen hingeschrieben, welche von den Skripten ausgegeben werden, welche der Apache aufruft.

13 Apache – Server

Da der Apache Webserver unter SuSE Linux 7.1 bereits installiert wird und korrekt arbeitet, gehen wir hier nur noch auf die Änderungen ein, welche vorgenommen werden müssen, um unsere Lösungen sauber zum arbeiten zu bringen.

Um *CGI* oder *webshell* Skripte in anderen Verzeichnissen als dem Standardverzeichnis `/usr/local/httpd/cgi-bin` ausführen zu können, muss in der Apache Konfigurationsdatei (`/etc/httpd/httpd.conf`) noch eine Änderung vorgenommen werden. Folgender Eintrag muss aktiviert werden:

```
AddHandler cgi-script .cgi
```

Dann muss für das Verzeichnis in welchem die *CGI* Dateien liegen noch mit dem Recht zum ausführen von Skripten versehen werden. Dies geschieht im Apache konfigurationfile mit folgendem Eintrag:

```
<Directory /usr/local/httpd/secemail/gnupg/cgi-bin>
    Options +ExecCGI
</Directory>
```

Für OpenSSL muss hier der Pfad zum `cgi-bin` Verzeichnis der OpenSSL Lösung angegeben werden.

Da der Key und das Passwort im Klartext in ihren Verzeichnissen liegen, muss dafür gesorgt werden, dass auf diese Verzeichnisse nicht zugegriffen werden kann. Mit folgendem Eintrag in der Datei `/etc/httpd/httpd.conf` können mit den `.htaccess` Dateien für jedes Verzeichnis spezielle Zugriffsrechte gesetzt werden.

```
<Directory /usr/local/httpd/secemail/>
    AllowOverride all
</Directory>
```

Nun muss im `etc Sub-Directory` noch die Datei `.htaccess` angelegt werden. Mit folgenden zwei Zeilen, kann der Zugriff auf die Dateien im Verzeichnis gesperrt werden.

```
order deny,allow
deny from all
```

Damit das Verzeichnis `secemail` direkt unter dem Verzeichnis `/usr/local/httpd` ansprechbar ist, muss im Apache Konfigurationsfile noch folgender Eintrag hinzugefügt werden.

```
DocumentRoot „/usr/local/httpd“
```

14 CD Aufbau

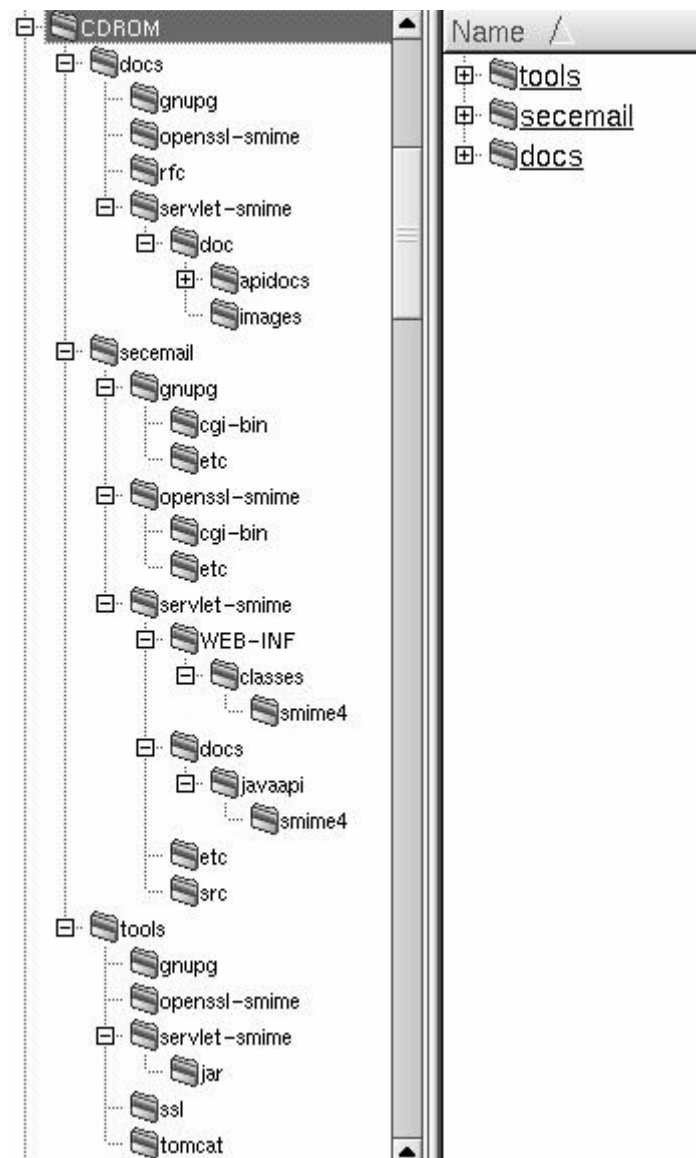


Abbildung 23 – CD Aufbau

15 Schlusswort

15.1 Fazit

Obwohl alle drei Varianten die Anforderungen erfüllen, gibt es erhebliche Unterschiede. Die GnuPG Implementation unterstützt zur Zeit noch keine Verschlüsselung. Jedoch ist dies sicherlich realisierbar. Weiter spricht gegen die GnuPG Variante, dass die Verbreitung von X.509 Zertifikaten im professionellen Umfeld wahrscheinlich eher Fuss fassen wird. Die beiden anderen Varianten unterscheiden sich hauptsächlich im Umfang. Da mit Java-Servlet noch gewisse Lizenzbedingungen eingehalten werden müssen, sowie Tomcat zusätzlich installiert werden muss, dürfte dies in erster Linie bei Projekten welche sowieso schon auf Java basieren, eine Alternative darstellen. Unserer Meinung nach ist aber die openssl-Variante den anderen Lösungen vorzuziehen, nicht zuletzt dank den diversen Möglichkeiten für die Passwortübergabe.

15.2 Weiterentwicklungsmöglichkeiten / Ausblick

Zwar werden die emails nun signiert und teilweise verschlüsselt versendet, jedoch ist noch keine sichere Verbindung vom Client zum Webserver vorhanden. Dieses Problem lässt sich aber mit dem SSL-Modul für den Apache sicherlich ohne grösseren Aufwand realisieren. Als weiterführende Arbeit wäre z.B. eine automatische email-Auswertung auf der Empfängerseite, welche die Bestellungen direkt in eine Datenbank einspielt, wünschbar. Zur Zeit verwenden wir für jeden Signier- und Verschlüsselungsvorgang Files mit den Schlüsseln und Zertifikaten, welche nur das jeweils Benötigte enthalten. Es ist also nicht möglich, je nach Bestellung ein anderes Zertifikat zu verwenden, und dieses dann an eine andere Empfängeradresse zu senden.

15.3 Dank

Wir bedanken uns bei unserem Industriepartner netcetera, speziell bei Herrn Dr. Kohler und unserem Dozenten Herrn Dr. Steffen für die gute Zusammenarbeit.

16 Zeitplan

16.1 Soll Zeitplan

Zeitplan der Projektgruppe Valle / Miletic Projektleiter Dr. Steffen

	W. 11	W.12	W.13	W.14	W. 15	W.16	W.17	W.18	W.19	W.20
Empfang der Arbeit Besprechung mit Hr. Steffen	Mo									
Arbeitsplatzeinrich- tung inkl. Betriebssystem auf- setzen	Di									
Zeitplan erstellen	Di									
Info's sammeln Überblick ver- schaffen	Mi- Fr									
Besprechung mit Hr. Kohler	Do									
Konkrete Zielsetzung		Mo*								
Besprechung mit Hr. Steffen		Di								
Realisierung		Di-Fr								
Realisierung			Mo- Fr							
Besprechung mit Hr. Steffen			Mo 16.00							
Realisierung				Mo- Fr						
Besprechung mit Hr. Steffen				Di						
Realisierung					Mo- Do					

Besprechung mit Hr. Steffen & Hr. Kohler					Mi 09:00					
Release 1.0					Do*					
Testen								Mi– Fr		
Besprechung mit Hr. Steffen								Mi		
Release 2.0								Fr*		
Doku									Mo– Fr	
Besprechung mit Hr. Steffen									Di	
Endversion									Fr*	
Doku										Mo– Fr
Besprechung mit Hr. Steffen										Di
Abgabe 18.05.01										Fr*

* = Meilenstein

16.2 Ist-Zeitplan

Zeitplan der Projektgruppe Valle / Miletic Projektleiter Dr. Steffen

	W. 11	W.12	W.13	W.14	W. 15	W.16	W.17	W.18	W.19	W.20
Empfang der Arbeit Besprechung mit Hr. Steffen	Mo									
Arbeitsplatzeinrich- tung inkl. Betriebssystem auf- setzen	Di									
Zeitplan erstellen	Di									
Info's sammeln Überblick ver- schaffen	Mi- Fr									
Besprechung mit Hr. Kohler	Do									
Konkrete Zielsetzung		Mo*								
Besprechung mit Hr. Steffen		Di								
Realisierung		Di-Fr								
Realisierung & Tests			Mo- Fr							
Besprechung mit Hr. Steffen			Mo 16.00							
1. Lauffähige Ver- sion			Di							
Abgabe der Version 1.0 und erster Dokumentation			Fr							
Testen				Mo- Do						
Besprechung mit Hr. Steffen				Di						
Release 2.0				Do*						

Dokumentation					Mo– Do					
Besprechung mit Hr. Steffen & Hr. Kohler					Mi 09:00					
Änderungen und Verbesserungen								Mi– Fr		
Besprechung mit Hr. Steffen								Mi		
Release 3.0								Fr*		
Handling verbessern									Mo– Mi	
Testen									Do– Fr	
Besprechung mit Hr. Steffen									Di	
Dokumentation										Mo– Fr
Präsentation vorbe- reiten										Di– Do
Endversion										Di*
Abgabe 18.05.01										Fr*
Präsentation bei Netcetera										Fr 14:00

* = Meilensteine

17 Literaturverzeichnis

17.1 RFC's

- [RFC1521]
IETF RFC 1521 MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies
- [RFC1847]
IETF RFC 1847 Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted
- [RFC2311]
IETF RFC 2311 S/MIME Version 2 Message Specification
- [RFC2312]
IETF RFC 2312 S/MIME Version 2 Certificate Handling
- [RFC2440]
IETF RFC 2440 OpenPGP Message Format
- [RFC2459]
IETF RFC 2459 Internet X.509 Public Key Infrastructure Certificate and CRL Profile

17.2 Webseiten

- [OpenSSL]
OpenSSL Projekt <http://www.openssl.org>
- [SSLHdb]
OpenSSL Handbuch <http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/>
- [IPGP]
International PGP Homepage <http://www.pgpi.org>
- [GnuPG]
GPG Gnu Privacy Guard <http://www.gnupg.org>
- [JBuild]
JBuilder installieren for Linux www.inprise.com/jbuilder/foundation/linux.html
- [JMail]
JavaMail API FAQ <http://java.sun.com/products/javamail/FAQ.html>

[ZDNET]

email-Verschlüsselung

http://www.zdnet.de/internet/artikel/sw/199812/krypto_00-wc.html

[Ulm]

Internet Security

[http://www-vs.informatik.uni-](http://www-vs.informatik.uni-ulm.de/Lehre/Seminar_Sicherheitsarch_SS99/email_ausarbeitung.pdf)

[ulm.de/Lehre/Seminar_Sicherheitsarch_SS99/email_ausarbeitung.pdf](http://www-vs.informatik.uni-ulm.de/Lehre/Seminar_Sicherheitsarch_SS99/email_ausarbeitung.pdf)

[JCSI]

Java Cryptography and Security at DSTC

<http://security.dstc.edu.au/projects/java/features.html>

[Websh]

Webshell Homepage

<http://www.websh.com>

[SNA]

Praktikum über Zertifikate

http://fbi.zhwin.ch/KSy/Block06/Praktikum/KSy_PA_8.pdf

17.3 Magazine

Linux-Magazin:

06/2000 Einführung in Tcl/Tk, Teil 2 S.148

07/2000 Servlet Programmierung S.92

08/2000 Einführung in Tcl/Tk, Teil 3 S.116

09/2000 Java Server Pages mit Tomcat S.130

10/2000 Einführung in Tcl/Tk, Teil 4 S.118

17.4 Bücher

[Krypt]

Albrecht Beutelspacher / Jörg Schwenk / Klaus-Dieter Wolfenstetter

Moderne Verfahren der Kryptographie

Vieweg, 1999, 3. Auflage

[Linux]

M. Wielsch / J. Prahm / H.-G. Esser

Linux intern

Data Becker, 1999, 1. Auflage

[Core]

Cay S. Horstmann / Gary Cornell

Core Java 2 Band 1 – Grundlagen

Markt+Technik, 1999, 1. Auflage

18 Glossar

- DSA** Digital Signatur Algorithmus. Ein von der NSA entwickelter, sehr sicherer Algorithmus zum Signieren von Daten. DSA verwendet den Hash-Algorithmus SHA1.
- DES** DES (Data Encryption Standard) ist ein Verschlüsselungsverfahren, das in den 70er Jahren von IBM entwickelt und 1977 von der US-Regierung als offizieller Standard anerkannt wurde. Die Schlüssellänge beträgt 56 Bit. Wegen des vergleichsweise kurzen Schlüssels gilt DES heute nicht mehr als sicher, da er durch einen Brute-Force-Angriff zu schnell geknackt werden kann.
- Triple-DES** Beim Triple-DES wird zuerst mit der ersten Hälfte des 112 Bit langen Schlüssels die Nachricht verschlüsselt. Danach wird das Ergebnis mit der zweiten Hälfte des 112 Bit langen Schlüssels entschlüsselt um dann zum Schluss nochmal mit dem ersten Schlüssel verschlüsselt zu werden. Der Algorithmus wird seit vielen Jahren studiert und ist als sehr stark einzuschätzen mit einer effektiven Schlüssellänge von 112 Bit. Triple-DES unterliegt keinerlei Patenten.
- RSA** RSA ist ein 1977 von Rivest, Shamir und Adleman vorgestellter Algorithmus, der unter anderem in PGP zur asymmetrischen Verschlüsselung verwendet wird. Die Schlüssel entstehen dabei durch Multiplikation zweier Primzahlen mit mehreren hundert Stellen. Die hohe Sicherheit von RSA beruht darauf, dass es erheblich aufwendiger ist, aus dem Ergebnis wieder die ursprünglichen Primzahlen zu ermitteln. Kritisch für die Sicherheit einer RSA-Verschlüsselung ist die Länge des verwendeten Schlüssels. Eine Mindestlänge von 128 Bit ist unbedingt empfehlenswert.
- Problematisch bei RSA ist, dass die Ver- beziehungsweise Entschlüsselung ungleich länger dauert als etwa bei den Algorithmen DES und IDEA. Aus diesem Grund setzen die meisten Public-Key-Produkte auf zwei Algorithmen: Zum Kodieren wird zum Beispiel IDEA verwendet, der dabei benutzte Schlüssel wird anschliessend mit RSA kodiert und (in kodierter Form) mit der Nachricht übertragen. Gegen RSA spricht, dass bis ins Jahr 2000 Lizenzgebühren abzuführen sind (allerdings auf die USA begrenzt).
- Diffie-Hellman-Verfahren, Elgamal-Algorithmus** Diffie und Hellman stellten 1976 als erste ein Public-Key-Verfahren vor, das ähnlich wie RSA zu einem Public-Private-Schlüsselpaar führt. Das Diffie-Hellman-Verfahren (DH) nutzt den Elgamal-Algorithmus, der bei gleicher Schlüssellänge genauso sicher ist wie der RSA-Algorithmus. Elgamal beruht auf der Schwierigkeit, diskrete Logarithmen zu berechnen.

DH und Elgamal haben den Vorteil, dass das Verfahren seit Mitte 1997 ohne Lizenzgebühren genutzt werden kann. Deshalb bietet zum Beispiel PGP das DH–Verfahren als Alternative zu RSA an, auch in Mail Guardian wird DH verwendet.

SHA1 Secure Hash Algorithm One. Von der NSA entwickelter Hashalgorithmus mit einer Schlüssellänge von 160 Bit

Keyring PGP verwaltet die Schlüssel in zwei Dateien, die als Schlüsselbund (key–ring) bezeichnet werden. Der geheime Schlüssel wird in der Datei secring.pgp und die öffentlichen Schlüssel in der Datei pubring.pgp gespeichert.

IETF Die Internet Engineering Task Force (IETF) ist eine international zusammengesetzte Gruppe von Netzwerktechnikern, Managern und Forschern, die sich um die technischen Grundlagen des Internet kümmert. Sie sorgt für die Standardisierung und Entwicklung wichtiger Internet–Protokolle.

19 Anhang

19.1 JCSI – Lizenzbedingungen

Evaluation License

Whereas Customer is accepting software developed by the Cooperative Research Centre for Distributed Systems Technology ("Centre") which is operated by DSTC on behalf of various organisations. The software is accepted on the following conditions:

1. DSTC grants the licensee a temporary, non-transferable source code licence to use the software herein including any related documentation solely for the purposes of testing and evaluation for a period of sixty (60) days which may be extended in writing at the sole discretion of DSTC.
2. The software is licensed, not sold. Title to, ownership of all applicable rights in patents, copyrights, trade secrets and other intellectual property in software and any copy shall remain with DSTC.
3. The Customer shall not transfer, sub-licence, copy, reverse engineer, decompile, disassemble translate or otherwise alter the software without the written consent of DSTC or distribute any copies to any other entity nor will the Customer cause or allow any of DSTC's products or part thereof to be placed in the public domain.
4. The Customer acknowledges that DSTC has stated that the software is proprietary and confidential to DSTC and agrees to maintain strict confidentiality concerning all aspects of the software with respect to all third parties, including benchmark test results, application specific results and observations regarding the software's quality, performance and features by treating it as it treats its own information of like kind (but not less than reasonable care).
5. The Customer acknowledges and understands that the software provided hereunder is required to be de-activated at the end of the nominated evaluation period, unless granted an extension of time in writing by DSTC, and return every item supplied under this agreement.
6. The software is provided "As is" and DSTC disclaims all warranties, express or implied, including but not limited to warranties of merchantability and fitness for a particular purpose. DSTC does not warrant that the software is error free, will operate without interruption or is compatible with all equipment and software configurations.
7. Under no circumstance shall DSTC be liable for indirect, incidental, or consequential damages including without limitation, loss of income, data, use or information, even if DSTC has been advised of the possibility of such damages. In no event shall the entire liability of DSTC exceed the sum of one thousand dollars. (,000.00)
8. This agreement may not be assigned or transferred without the express written permission of DSTC.
9. The Customer acknowledges that Customer has read and understood this agreement and agrees to be bound by its terms and conditions. Customer further agrees that it is the complete and exclusive statement of the agreement between Customer and DSTC and supercedes any earlier proposal or arrangement, whether oral or written, and any other communication between the parties relative to the subject matter of this agreement.
10. The Customer may be held legally responsible for any copyright infringement that is caused or encouraged by the Customer's failure to abide by the terms of this licence.
11. This agreement shall be governed by the laws of Queensland, Australia