

## Vertrauliche Kommunikation im Internet

Durch die stark zunehmende Nutzung von Internetdiensten und das Interesse, diese auch privat zu nutzen, stellt sich verstärkt die Frage nach der Sicherheit der übermittelten Daten. Seit einiger Zeit ist es sogar möglich, über das Internet einzukaufen und online zu bezahlen. Spätestens dann, wenn eine Kreditkartennummer oder sogar sensible personenbezogene Daten wie z.B. Krankenakten übertragen werden, sollte ernsthaft analysiert werden, wie Vertraulichkeit und Integrität der Nachrichten gesichert werden können. In diesem Artikel soll am Beispiel von E-Mail - Electronic Mail - dargestellt werden, welche potentiellen Sicherheitslücken auf die Anwender warten und welche Maßnahmen ergriffen werden können, um ihnen zu begegnen. Nach der Darstellung einiger technischer Grundlagen und der auftretenden Sicherheitsprobleme wird eine Auswahl relevanter mathematischer bzw. kryptographischer Methoden vorgestellt. Zum Abschluß soll das Programm PGP (Pretty Good Privacy) besprochen werden, mit dem die theoretischen Verfahren praktisch genutzt werden können.

Der Austausch von E-Mail im Internet basiert auf dem SMTP-Protokoll (Simple Mail Transfer Protocol). Der Weg einer Mail läßt sich, etwas vereinfacht, so darstellen: Der Nutzer generiert seine Mail mit einem Programm, dem sogenannten MUA (Mail User Agent). Das können z.B. die UNIX-Programme mail, elm oder pine sein. Eine Mail besteht aus dem Header, der wichtige Informationen wie Empfänger, die Organisation u.v.m. enthält. Der Body nimmt dann den eigentlichen Text auf. Das Mailprogramm sendet die Mail an einen anderen Rechner, genannt MTA (Mail Transfer Agent). Der MTA bestimmt die Empfängeradresse und nimmt entweder direkt mit dem MTA auf der Zielseite Kontakt auf oder leitet die Mail erst einmal an einen anderen MTA weiter. Die Kommunikation unter den MTA erfolgt über das SMTP-Protokoll, das selbst wieder auf den Internetprotokollen TCP/IP beruht, so daß die Mail eine Vielzahl von Rechnern passieren muß.

Die Mailbox eines Nutzers ist üblicherweise nichts weiter als eine einzige Datei in einem bestimmten Verzeichnis des Mailservers. Damit ist klar, daß zumindest die Systemadministratoren die Mail ohne Mühe lesen können, wenn sie denn wollen! In vielen lokalen Netzen wird das Verzeichnis des Mailservers, in dem die Mailboxen liegen, über den Network File Service (NFS) auf allen Rechnern verfügbar gemacht. Wenn auch die normalen UNIX-Sicherheitsmechanismen wie Zugriffsrechte auf Dateien gelten, so ist doch stets die Gefahr vorhanden, daß ein Hacker sich durch Sicherheitslücken des Systems oder der Systemprogramme Administratorrechte verschafft und somit die Mails sämtlicher Nutzer lesen kann. Diese Fälle sind nicht so selten, wie man vermuten könnte. Auch ein Löschen

der Mails nach dem Lesen bietet keine absolute Sicherheit, da Backups sämtlicher Mailboxen existieren können, die mitunter Monate gespeichert werden. Da die Mail auf dem Weg zum Empfänger zahlreiche andere Rechner (Gateways, Router) passiert, besitzt jeder dieser Rechner die Möglichkeit, den gesamten Mailverkehr abzuhören. Die Nachrichten werden im Klartext verschickt, so daß es möglich ist, auf bestimmte Absender oder Nachrichteninhalte hin zu filtern. Da die Mailübertragung im Endeffekt auf Internetprotokollen beruht, gibt es auch auf dieser Ebene zahlreiche Möglichkeiten, Mails mitzuschneiden. Auf diesem Wege ist es sogar möglich, Paßwörter für Rechner mitzuhören, da selbst diese im Klartext über das Netz übertragen werden. Durch die unterschiedlichen Rechner, die an der Zustellung beteiligt sind, ergibt sich zwingend, daß die Zeit, die eine Mail zum Empfänger braucht, nicht genau bestimmbar ist, obwohl die Zustellung innerhalb einer Organisation oder eines Landes meist nur Sekunden braucht. Der Zustellungsprozeß kann im schlimmsten Falle ein paar Tage dauern, was jedoch selten ist. Trotzdem ist hier natürlich die Möglichkeit gegeben, eingegangene Mails zwar weiterzuleiten, doch ihren Inhalt ein wenig zu verändern. Niemand kann die Verzögerung in der Zustellung bemerken. Diese Form der Manipulation ist sogar kritischer anzusehen als das simple Mitlesen, da den Kommunikationspartnern vorgegaukelt wird, daß ihre Mails vernünftig verschickt wurden, jedoch ist der Inhalt ein völlig anderer. Natürlich kann eine Mail auch völlig unterdrückt werden, so daß sie den Empfänger nie erreicht.

Weiterhin ist es jedem Nutzer ohne weiteres möglich, in seinem Mail-Client eine falsche Absenderadresse einzutragen. So könnte man, wenn man denn unbedingt möchte, Mails unter dem Absender von Bill Gates versenden.

Aus den dargestellten Sicherheitsrisiken lassen sich also folgende Forderungen an eine sichere Kommunikation aufstellen:

- **Vertraulichkeit** - Nur der vom Absender bestimmte Empfänger soll in der Lage sein, die Nachricht zu lesen.
- **Authentizität** - Der Empfänger soll einwandfrei überprüfen können, ob die Nachricht auch wirklich von dem angegebenen Absender und nicht von einem Dritten stammt. Hier wird das Prinzip der digitalen Unterschrift benutzt.
- **Integrität** - Die Nachricht soll beim Empfänger genau so ankommen, wie sie abgeschickt worden ist, d.h. auf dem Transportweg sollen keine Veränderungen an der Nachricht vorgenommen werden können, ohne daß der Empfänger dies bemerkt.

All diese Forderungen lassen sich mit unterschiedlichen mathematischen bzw. kryptographischen Methoden erfüllen.

Das Anliegen von Kryptographie läßt sich etwas verkürzt etwa so darstellen: Ein Text soll so aufbereitet werden, daß er nur von den von mir vorgesehenen Personen gelesen werden kann. Dazu wird ein Schlüssel verwendet, der den ursprünglichen Text so verändert, daß er von niemand anderem mehr gelesen werden kann, es sei denn, er besitzt wiederum den passenden Schlüssel. Diese beiden Schlüssel müssen nicht notwendig gleich sein. Es ist offensichtlich, daß der Verschlüsselungserfolg in erster Linie von der Sicherheit des Algorithmus und der Länge des verwendeten Schlüssels abhängig ist. Je länger dieser ist, desto mehr Möglichkeiten gibt es, die ausprobiert werden müssen, um ihn zu knacken. Ein idealer Algorithmus läßt sich nur durch systematisches Durchprobieren aller möglichen Schlüssel (Brute Force Attack) knacken. Doch sind nicht alle Algorithmen gleich stark. Unter Ausnutzung meist komplizierter mathematischer Zusammenhänge ist es möglich, eine Menge von möglichen Schlüsseln zu eliminieren, so daß die Zahl derer, die durchzuprobieren sind, kleiner wird. In den folgenden Abschnitten sollen zwei Grundprinzipien von Verschlüsselungsalgorithmen betrachtet werden.

### Symmetrische Verschlüsselung

Bei der symmetrischen Verschlüsselung benutzen zwei Kommunikationspartner zum Austausch der Informationen einen gemeinsamen Schlüssel. Das einfachste Beispiel ist der bei Kindern sehr beliebte Code A=1, B=2 usw. Das Grundproblem dieser Methode ist die Übermittlung des gemeinsamen Schlüssels. Wenn eine Methode bekannt ist, den Schlüssel geheim und sicher auszutauschen, wieso sollte man nicht gleich auf diesem Wege die Nachrichten verschicken?

Wenn viele Personen jeweils miteinander kommunizieren möchten, benötigt man für jede denkbare Kombination einen Schlüssel, z.B. benötigt man für 60.000 Menschen schon 1.779.970.000 verschiedene Schlüssel. Diese Zahl bedarf keiner weiteren Erläuterung.

Die gebräuchlichsten Verfahren symmetrischer Verschlüsselung sind *DES* (Data Encryption Standard) und *IDEA* (International Data Encryption Algorithm).

### Asymmetrische Verschlüsselung

Während bei der symmetrischen Verschlüsselung beide Kommunikationspartner denselben Schlüssel benutzen und damit das Problem des Schlüsselaustauschs besteht, werden bei der asymmetrischen Verschlüsselung (oder auch *public key*-Verschlüsselung) für jeden Teilnehmer zwei Schlüssel generiert, der *public key* und der *secret key*. Daten, die mit dem *public key* verschlüsselt wurden, können nur mit dem *secret key* gelesen werden. Der *public key* kann, wie der Name deut-

lich zum Ausdruck bringt, veröffentlicht werden. Der Absender benutzt den *public key* des Empfängers, um ihm Nachrichten zu schicken. Dieser kann mit Hilfe seines *secret keys* die Nachrichten entschlüsseln.

Der große Vorteil dieser Methode liegt darin, daß kein sicherer Kanal genutzt werden muß, um einen Schlüssel zu übertragen. Alle Informationen für eine vertrauliche Kommunikation können öffentlich gemacht werden. Im Gegensatz zur symmetrischen Verschlüsselung, bei der für jede individuelle Kommunikation ein eigener Schlüssel verwendet werden muß, braucht hier jeder Kommunikationspartner nur ein Paar Schlüssel, um mit allen anderen kommunizieren zu können.

Weiterhin ergibt sich noch die Möglichkeit der Nutzung digitaler Unterschriften. Hierbei wird das oben beschriebene Verfahren einfach umgekehrt. Wenn A an B eine Nachricht versenden will, kodiert er die Nachricht mit seinem *secret key* und verschickt sie. Wenn B daraufhin den *public key* von A anwendet und die Nachricht lesbar ist, weiß B, daß die Nachricht von A stammen muß. In der Praxis werden oft beide Methoden gleichzeitig benutzt, so daß verschlüsselte und authentifizierte Nachrichten verschickt werden können. Der bekannteste Algorithmus ist das nach seinen Erfindern Rivest, Shamir und Adleman benannte *RSA-Verfahren*.

In engem Zusammenhang mit der digitalen Unterschrift stehen die Message-Digest-Verfahren. Mit ihrer Hilfe wird die Integrität einer Nachricht gesichert, d.h. die Daten können nicht geändert werden, ohne daß dies vom Empfänger bemerkt wird. Diese Verfahren sind in der Lage, aus einem beliebig langen Text eine kleine Zahlenfolge konstanter Länge zu erzeugen. So bildet z.B. die *MD5*-Funktion, die auch von *PGP* benutzt wird, jeden Text in eine Zeichenfolge von 128 bit ab. Das Besondere ist nun, daß schon eine minimale Modifikation des Ausgangstextes eine Veränderung des *MD5*-Codes nach sich zieht. So kann sofort erkannt werden, daß der empfangene Text nicht mit dem Ausgangstext übereinstimmt. Es ist klar, daß aufgrund der geringen Länge des *MD5*-Codes die theoretische Möglichkeit besteht, daß zwei verschiedene Texte denselben Code besitzen, dies ist jedoch wesentlich unwahrscheinlicher als das Auftreten identischer Fingerabdrücke.

Nachdem nun in aller Kürze die theoretischen Grundlagen dargestellt worden sind, soll im folgenden Abschnitt das Programm *PGP* erläutert werden, das es jedem Anwender erlaubt, seine eigene Kommunikation sicherer zu gestalten.

### Wie funktioniert PGP?

*PGP* heißt Pretty Good Privacy und ist ein Programm, das nach den Vorstellungen seines Entwicklers Phil Zimmermann sichere Kryptographie für jedermann be-

reitstellen soll. *PGP* wurde nach mehrjähriger Entwicklung im Jahre 1991 erstmals vorgestellt, und zwar als Freeware, d.h. das Programm kann frei weitergegeben werden. Allerdings hatte Zimmermann nicht beachtet, daß seine Software Algorithmen verwendete, die durch Patente geschützt waren. Somit war die Verwendung von *PGP* illegal. Dies schränkte vielleicht die offene Benutzung von *PGP*, nicht jedoch die „illegale“ Verbreitung ein. Auch außerhalb der USA war das Programm schnell zu finden. Hier kam ein weiteres Problem dazu. Der Export von starken Verschlüsselungsalgorithmen unterliegt in den USA dem Waffenexportkontrollgesetz. Eine Reihe dieser Probleme ist in folgenden Versionen gelöst worden. Durch die Verwendung des RSAREF-Kits, das für nichtkommerzielle Anwendungen freigegeben ist, konnten die Patentbestimmungen erfüllt werden. In Zusammenarbeit mit der Firma Viacrypt wird eine kommerzielle Version vertrieben. Internationale Versionen arbeiten ohne das RSAREF-Kit, diese Versionen haben ein kleines *i* am Ende der Versionsnummer und sind zur Nutzung außerhalb der USA vorgesehen. Das Problem der Exportbeschränkung hat sich auf eindrucksvolle Weise vor ein paar Monaten erledigt. Das Verfahren gegen Phil Zimmermann wegen Verletzung dieser Beschränkungen wurde ohne große Begründung eingestellt.

Bei der folgenden Beschreibung wird davon ausgegangen, daß *PGP* auf dem verwendeten Rechner korrekt installiert ist. Das Programm ist für MS-Windows, MAC und alle UNIX-Plattformen verfügbar.

Kernstück von *PGP* ist die Implementation des RSA-Verfahrens, d.h. jeder Kommunikationsteilnehmer benötigt ein Schlüsselpaar (*public key/secret key*), und der Absender benötigt zum Verschlüsseln der Nachricht die *public keys* der Empfänger. Damit diese Schlüssel nicht jeweils einzeln gespeichert werden müssen, erstellt *PGP* Schlüsselbunde (*key rings*), so daß sie effektiv verwaltet und benutzt werden können.

Für die eigentliche Verschlüsselung der Nachricht benutzt *PGP* jedoch nicht den RSA-Algorithmus, da er zu langsam arbeitet. Hier wird ein symmetrisches Verfahren (IDEA) verwendet.

Im folgenden wird beschrieben, wie *PGP* bei der Verschlüsselung einer Nachricht vorgeht. Dabei soll zur Sicherung der Integrität der Nachricht ein Message-Digest-Verfahren verwendet und eine digitale Unterschrift erzeugt werden.

1. *PGP* erzeugt einen zufälligen *session key*, damit verschlüsselt der IDEA-Algorithmus den Text,
2. der RSA-Algorithmus wird benutzt, um den *session key* mit dem *public key* des Empfängers zu verschlüsseln,
3. für den Ausgangstext wird der MD5-Code berechnet und nach dem RSA-Verfahren mit dem *secret key* des Absenders verschlüsselt.

Alle Verschlüsselungsergebnisse werden in einer Datei zusammengefügt, die dann als E-Mail verschickt werden kann, doch ist die Anwendung von *PGP* keineswegs auf die Internetkommunikation beschränkt, es lassen sich ebenso gespeicherte Dateien verschlüsseln und somit vor illegalem Zugriff sichern.

Welche Vorbereitungen sind nun zu treffen, um mit *PGP* arbeiten zu können?

An erster Stelle steht die Erzeugung eines persönlichen Schlüsselpaares. In Abb. 1 ist ein Screenshot des entsprechenden Programmaufrufs zu sehen.

```
[root@nuffzi root]# pgp -kg
No configuration file found.
Pretty Good Privacy(tm) 2.6.2i - Public-key encryption for the masses.
(c) 1990-1995 Philip Zimmermann, Phil's Pretty Good Software. 7 May 95
International version - not for use in the USA. Does not use RSAREF.
Current time: 1996/06/03 00:34 GMT
Pick your RSA key size:
  1) 512 bits- Low commercial grade, fast but less secure
  2) 768 bits- High commercial grade, medium speed, good security
  3) 1024 bits- "Military" grade, slow, highest security
Choose 1, 2, or 3, or enter desired number of bits: 3
Generating an RSA key with a 1024-bit modulus.

You need a user ID for your public key. The desired form for this
user ID is your name, followed by your E-mail address enclosed in
<angle brackets>, if you have an E-mail address.
For example: John Q. Smith <12345.6789@compuserve.com>
Enter a user ID for your public key:
Daniel Ohst <ohst@informatik.hu-berlin.de>

You need a pass phrase to protect your RSA secret key.
Your pass phrase can be any sentence or phrase and may have many
words, spaces, punctuation, or any other printable characters.

Enter pass phrase:
Enter same pass phrase again:
Note that key generation is a lengthy process.

We need to generate 648 random bits. This is done by measuring the
time intervals between your keystrokes. Please enter some random text
on your keyboard until you hear the beep:
  0 * -Enough, thank you.
.....**** .....****
Key generation completed.
```

Abb. 1: Erzeugung eines Schlüsselpaares

*PGP* wird mit der Option *-kg* aufgerufen. Es stehen drei verschiedene Schlüssellängen zur Auswahl, wobei die größte (1024 bit) gewählt werden sollte. Der geringfügig höhere Zeitaufwand für die Verschlüsselung ist im Vergleich zur stark erhöhten Sicherheit zu vernachlässigen.

Als User-Id wird üblicherweise der vollständige Name und die in spitzen Klammern eingeschlossene E-Mailadresse verwendet.

Die *pass phrase* dient dazu, den *secret key* vor mißbräuchlicher Verwendung zu schützen, da dieser auch in einer Datei auf der Festplatte gespeichert wird. Sollte es jemandem gelingen, diese Datei illegal zu kopieren, wäre er in der Lage, die verschlüsselten Daten zu lesen. Um dies zu verhindern, wird vor jeder Aktion, die den *secret key* erfordert, die *pass phrase* abgefragt. Zur Erhöhung der Sicherheit kann hier nicht nur ein Wort sondern sogar ein ganzer Satz eingegeben werden, der durch Return (Zeilenende) abgeschlossen wird.

Anschließend sind noch einige Tastenanschläge erforderlich, um ein paar zufällige Daten in die Schlüsselgenerierung eingehen zu lassen.

Danach wird das Schlüsselpaar erzeugt, was je nach verwendeter Rechenleistung zwischen einigen Sekunden und mehreren Minuten dauern kann.

Mit dem Kommando *PGP -kv* lassen sich alle gespeicherten Schlüssel ausgeben. Dort sollte jetzt das persönliche Schlüsselpaar erscheinen.

Um eine Nachricht versenden zu können, benötigt man nun noch den *public key* des gewünschten Empfängers. Um diesen zu erhalten, gibt es u.a folgende Möglichkeiten:

- der Empfänger stellt den *public key* in der Ausgabe des Kommandos *finger* zur Verfügung (z.B. `finger ohst@informatik.hu-berlin.de`).
- der *public key* wird vom Empfänger auf seiner WWW-Seite veröffentlicht.
- man sucht nach dem *public key* auf einem Keyserver (z.B. `http://math-www.uni-paderborn.de/pgp/`). Dort kann sich jeder mit seinem Schlüssel eintragen lassen.

Der Schlüssel liegt im ASCII-Format vor und beginnt mit dem String `-----BEGIN PGP PUBLIC KEY BLOCK-----`. Er sollte in einer Datei gespeichert werden, danach wird *PGP* mit der Option `-ka` aufgerufen. Der in der Datei vorhandene Schlüssel wird erkannt und der Name des Inhabers angezeigt. Auf die Frage *Do you want to certify any of these keys yourself (y/N)?* sollte erst einmal mit N geantwortet werden. Der Schlüssel wird dann eingetragen und kann mit *PGP -kv* angesehen werden. Auf diese Weise werden die *public keys* aller potentiellen Empfänger eingetragen.

Jetzt soll eine Nachricht verschickt und gleichzeitig mit einer digitalen Unterschrift versehen werden. Abbildung 2 zeigt den entsprechenden Screendump.

*PGP* wird mit der Option `-esa` (`encrypt, sign, ascii`), der zu verschlüsselnden Datei und einem Teil des Namens des Empfängers aufgerufen. Die Option `-a` ist

```
[root@nuffzi root]# pgp -esa rdi.html Ohst
No configuration file found.
Pretty Good Privacy(tm) 2.6.2i - Public-key encryption for the masses.
(c) 1990-1995 Philip Zimmermann, Phil's Pretty Good Software. 7 May 95
International version - not for use in the USA. Does not use RSAREF.
Current time: 1996/06/03 00:38 GMT

A secret key is required to make a signature.
You specified no user ID to select your secret key,
so the default user ID and key will be the most recently
added key on your secret keyring.

You need a pass phrase to unlock your RSA secret key.
Key for user ID "Daniel Ohst <ohst@informatik.hu-berlin.de>"

Enter pass phrase: Pass phrase is good.
Key for user ID: Daniel Ohst <ohst@informatik.hu-berlin.de>
1024-bit key, Key ID ED7B0F69, created 1996/06/03
Just a moment....

Recipients' public key(s) will be used to encrypt.
Key for user ID: Daniel Ohst <ohst@informatik.hu-berlin.de>
1024-bit key, Key ID ED7B0F69, created 1996/06/03
.
Transport armor file: rdi.html.asc
```

Abb. 2: Verschlüsseln und Unterschreiben

meist erforderlich, wenn die Datei als E-Mail verschickt werden soll, um einen 7-bit-ASCII-Code zu erzeugen. Zum Unterschreiben der Nachricht wird der *secret key* verwendet, so daß die Eingabe der *pass phrase* erforderlich ist. Die erzeugte Datei kann dann verschickt werden.

Die Entschlüsselung ist denkbar einfach, *PGP* wird mit der Datei als Option aufgerufen, in der die erhaltene Nachricht gespeichert wurde. Nach Eingabe der *pass phrase* wird eine Datei mit dem Namen erzeugt, der beim Verschlüsseln auf Absenderseite verwendet wurde.

Ein auftretendes Problem ist die Verifizierung eines *public keys*. Wie soll man einwandfrei überprüfen, daß der vorliegende Schlüssel auch zu der Person gehört, die sie vorgibt zu sein? Eine Möglichkeit ist es, übergeordnete Gremien zu finden, die die Schlüssel verwalten und für die Echtheit garantieren. Allerdings wirft diese zentrale Verwaltung Probleme auf und ist zumindest bei privaten Nutzern nicht sehr beliebt. Ein anderes Verschlüsselungsverfahren (PEM), das auf diesem Prinzip der hierarchischen Schlüsselverwaltung basiert, hat bei weitem nicht die Verbreitung von *PGP* erreicht. Es scheint bei den Nutzern offenbar eine begründete Skepsis vor jeder übergeordneten Verwaltungsinstanz zu geben.

Eine andere Möglichkeit läßt sich wie folgt formulieren: „Unterschreiben sie einfach mit ihrem guten Namen“. Beim Ergänzen von *public keys* anderer Nutzer wird gefragt, ob man diese zertifizieren möchte. Hier sollte dann mit ja geantwortet werden, wenn man sicher ist, daß der Schlüssel auch wirklich der bezeichneten Person gehört. Dann wird die eigene digitale Unterschrift zum *public key* hinzugefügt. Damit ergibt sich ein neuer *public key*, den die Person veröffentlichen kann. Wenn nun ein neuer Nutzer diesen *public key* erhält, sieht er auch diese digitale(n) Unterschrift(en) und kann nun aufgrund des Vertrauensverhältnisses zu einem Unterzeichnenden entscheiden, auch diesem neuen *public key* zu vertrauen. So soll, Phil Zimmermann nennt es selbst so, ein „Web of trust“ entstehen. Allerdings sei davor gewarnt, mit dem Unterschreiben fremder Schlüssel allzu verschwenderisch umzugehen.

Diese Einführung in die Arbeitsweise von *PGP* konnte nur einen Bruchteil des gesamten Themenkreises erfassen. Wer tiefer in die Materie einsteigen möchte, dem sei das von Zimmermann selbst geschriebene Manual empfohlen, das jeder *PGP*-Distribution beigelegt ist.

Daniel Ohst