Erste Schritte in GPG

GPG aufrufen von SHELL:

```
moneysac:~> gpg --help
gpg (GnuPG) 1.3.6 ...
Home: ~/.gnupg
Supported algorithms:
Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA
```

Auflistung der in der Version verfügbaren asymmetrischen Algorithmen. Dabei bedeutet `Algname-E` Verschlüsselung und `Algname-S` Signatur. Bei Verfahren, von denen klar ist, wozu sie benutzt werden, steht kein extra Buchstabe dahinter, wie z.B. `DSA`.

```
Cipher: 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH
```

Auflistung der in der Version verfügbaren symmetrischen Verfahren, z.B. Triple-DES und AES. Anschließend folgt eine Auflistung der Hash- und Kompressionsfunktionen. Den Algorithmus `SHA-1` haben wir in abgewandelter Form benutzt in der praktischen Aufgabe auf Blatt 4.

```
Hash: MD5, SHA1, RIPEMD160, SHA256, SHA384, SHA512
Compression: Uncompressed, ZIP, ZLIB, BZIP2
...
```

GPG: Schlüsselerzeugung

Als nächstes wollen wir ein Schlüsselpaar erzeugen, mit dem wir signieren und andere uns verschlüsselte Mails schicken können. Dies geschieht durch die Eingabe folgenden Befehles in die SHELL:

```
moneysac:> gpg --gen-key
gpg (GnuPG) 1.3.6; Copyright (C) 2004 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.
gpg: please see http://www.gnupg.org/faq.html for more information
gpg: /homes/algebra9/wagner/.gnupg: directory created
gpg: new configuration file '/homes/algebra9/wagner/.gnupg/gpg.conf' created
gpg: WARNING: options in '/homes/algebra9/wagner/.gnupg/gpg.conf' are not yet active during this run
gpg: keyring '/homes/algebra9/wagner/.gnupg/secring.gpg' created
gpg: keyring '/homes/algebra9/wagner/.gnupg/pubring.gpg' created
```

Zunächst wird automatisch im Home-Verzeichnis ein Unterverzeichnis namens .gnupg angelegt. In diesem Verzeichnis werden dann u.a. Dateien namens secring.gpg und pubring.gpg geschrieben. Ersteres ist ein Schlüsselbund von geheimen Schlüsseln, Letzteres ein Schlüsselbund von öffentlichen Schlüsseln.
WICHTIG: Zugriffsrechte auf das Verzeichnis .gnupg ändern mit Eingabe von

```
moneysac: > chmod -R og-rwx /homes/algebra9/wagner/.gnupg
```

GPG: Schlüsselerzeugung

Als nächstes muß man festlegen, ob man verschlüsseln und signieren oder einfach nur signieren möchte:

```
Please select what kind of key you want:
   (1) DSA and Elgamal (default)
   (2) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
```

Mit der Eingabe 1 haben wir uns also für das erstere entschieden. Der DSA-Schlüssel ist auf 1024 Bit festgelegt. Beim ElGamal-Schlüssel kann man nun im Folgendem die Bit-Größe wählen:

```
DSA keypair will have 1024 bits.
About to generate a new ELG-E keypair.
              minimum keysize is  768 bits
              default keysize is 1024 bits
    highest suggested keysize is 2048 bits
What keysize do you want? (1024) 2048
Requested keysize is 2048 bits
```

Wir haben uns also für eine Länge von 2048 Bits entschieden. Augenblicklich sind bis 4096 Bit möglich, allerdings nimmt die Erzeugung ca. 15 Minuten in Anspruch. Bei einer solchen Erzeugung eines Schlüsselpaares sollte man für höchstmögliche Entropie sorgen. Der Rechner bezieht Zufall aus diversen Ereignissen, die auf dem Rechner stattfinden. Man könnte z.B. einen Kurzfilm schauen, da dann gleichermaßen Video- und Audioausgabe sowie Speicher- und Festplattenzugriffe stattfinden. Als nächstes wird die Geltungsdauer des Schlüsselpaares festgelegt:

GPG: Schlüsselerzeugung

```
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 3y
Key expires at Wed Jul 11 21:09:03 2007 CEST
Is this correct (y/n)? y
```

Wir haben uns also für eine Geltungsdauer von 3 Jahren entschieden. Der Schlüssel ist nach Ablauf von 3 Jahren noch benutzbar, aber man legt durch diese Einschränkung lediglich seine persönliche Geltungsdauer fest und teilt auf diesem Wege anderen mit, dass der Schlüssel dann nicht mehr benutzt werden soll. Bis jetzt ist das Schlüsselpaar noch nicht an die Person gebunden. Dies geschieht erst mit der Eingabe von Real name und Email address , eventuell kann man noch ein Comment folgen lassen:

```
You need a User-ID to identify your key; the software constructs the user id
from Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Marcus Wagner
Email address: math.tu-berlin.de
Comment:
You selected this USER-ID:
    "Marcus Wagner <math.tu-berlin.de>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
```

GPG: Schlüsselerzeugung

Damit niemand anderes einfach unsere Identität benutzen kann, schützen wir sie mit einer sogenannten passphrase :

```
You need a Passphrase to protect your secret key.

Enter passphrase:  1234567890
Repeat passphrase: 1234567890
```

Die Passphrase sollte natürlich länger sein als im obigen Beispiel. Als Faustregel hat sich eingebürgert Bitlänge geteilt durch Hundert viele Zeichen zu benutzen. Zum Beispiel sucht man sich eine Textpassage aus einem Buch und benutzt nur die Anfangsbuchstaben aller Worte und streut Sonderzeichen dazwischen. Hier würden jetzt diverse Ausgaben der Schlüsselerzeugung folgen, die der Übersicht halber weggelassen wurden. Anschließend gibt GPG eine Zusammenfassung der Schlüsselgeneration aus:

```
gpg: /homes/algebra9/wagner/.gnupg/trustdb.gpg: trustdb created
gpg: key 9C25F3F5 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2007-07-11
pub 1024D/9C25F3F5 2004-07-11 Marcus Wagner <math.tu-berlin.de>
Key fingerprint = 6658 15B8 F51C F7D2 9041 9247 03F8 9490 9C25 F3F5
sub 2048g/87AD1F6F 2004-07-11 [expires: 2007-07-11]
```

GPG: Web of trust

Da public-key Verschlüsselung gestattet, dem Gesprächspartner verschlüsselte Nachrichten zukommen zu lassen, ohne ihn je vorher getroffen zu haben, muß gewährleistet sein, daß hinter einem öffentlicher Schlüssel auch wirklich die Person steckt, die die man erreichen möchte. Genauso wichtig ist auch, seinem Gegenüber glaubhaft zu machen, dass man die Person ist, für die man sich ausgibt. Da GPG als Open-Source Projekt keine zentrale Instanz (Trust-Center) zur Verifizierung der Identität besitzt, behilft man sich mit dem Konzept des Web of trust. Das Web of trust basiert auf der Transitivitätsannahme, dass wenn Person A Person B und Person B der Person C traut, dann auch A der Person C traut. Es gibt 6 Stufen des Vertrauens:

- `e` – das Vertrauen läßt sich gerade nicht berechnen (Schlüssel ist wahrscheinlich über dem Verfallsdatum)
- `q` – ich habe wenig Informationen/ich bin mir nicht sicher
- `n` – ich vertraue dem Schlüssel nie
- `m` – ich vertraue dem Schlüssel halbwegs
- `f` – ich vertraue dem Schlüssel voll
- `u` – ich vertraue dem Schlüssel uneingeschränkt (ultimately)

In der Zusammenfassung von der vorigen Folie sieht man zum Beispiel:

```
gpg: /homes/algebra9/wagner/.gnupg/trustdb.gpg: trustdb created
gpg: key 9C25F3F5 marked as ultimately trusted
public and secret key created and signed.
```

Wie man sieht, traut man seinem eigenen Schlüssel ultimativ (dies sollten auch die einzigen Schlüssel sein, denen man ultimativ traut, da man ihnen selbst nach Ablauf der Geltungsdauer noch trauen würde).

Import und Export von keys

Um anderen nun verschlüsselte Nachrichten zukommen zu lassen, brauchen wir zunächst deren öffentliche Schlüssel. Dazu importieren wir die Schlüssel von einer zentralen Ablage (dem keyserver) in unseren öffentlichen Schlüsselbund (pubring.gpg).

```
moneysac: > gpg --recv-keys 0x135EA668
gpg: key 135EA668: duplicated user ID detected - merged
gpg: key 135EA668: public key Richard Stallman (Chief GNUisance) <rms@gnu.org>ïmported
gpg: 3 marginal(s) needed, 1 complete(s) needed, classic trust model
gpg: depth: 0 valid: 4 signed: 6 trust: 0-, 0q, 0n, 0m, 0f, 4u
gpg: depth: 1 valid: 6 signed: 3 trust: 6-, 0q, 0n, 0m, 0f, 0u
gpg: next trustdb check due at 2005-09-22
gpg: Total number processed: 1
gpg: imported: 1
```

Hier haben wir den Schlüssel mit der Schlüssel-ID `0x135EA668` importiert was man mit `moneysac: > gpg --list-keys` verifizieren kann. Ferner werden neben den Schlüsseldaten auch Informationen zum trust-Modell angezeigt.

Unseren eigenen Schlüssel können wir auf ähnliche Weise auf den keyserver exportieren:

```
moneysac: > gpg --send-keys 0x71F00DAC
```

Verschlüsseln von Dokumenten

Wir wollen eine Datei namens `letter` folgenden Inhalts (z.B. den Text einer mail) verschlüsseln und an den Besitzer des Schlüssels mit der Schlüssel-ID `0xc2e7d7cf` senden:

```
Hallo Test,

Gruss
Testende
```

Dazu benutzen wir das `--encrypt`-Kommando (kurz `-e`). Den Empfänger spezifizieren wir anhand seiner Schlüssel-ID mit der `--recipient`-Option (kurz `-r`):

```
moneysac: > gpg -e -r 0xc2e7d7cf -a -o letter.gpg letter
```

Die Option `-a` (kurz für `--ascii-enarmor`) liefert eine lesbare Datei (base64-Enkodierung). Die Option `-o` (kurz für `--output`) legt die Ausgabedatei fest (ansonsten wird das Ergebnis nach `stdout` ausgegeben). Die Ausgabedatei `letter.gpg` hat nun folgenden Inhalt:

```
1. ----BEGIN PGP MESSAGE-----
Version: GnuPG v1.3.6 (GNU/Linux)

hQQOA7dhaJ/naFKjEA/+LJglKK8JuWrfir+Dwjny6Usn0HBtHDSBt2td1FH+ppi/
bv+8Qd2ICabkjfK2HLljuXGwuEzGP2zdOuHg4ww63Kkwv0domNzwMlGCNqmIzOyn
GrR6DT1r6nUPnNpwsPvRN1/6ncg4cezM1s9QazSLeS/ic3G2SVv1cX7kYv3yxvG2
...
1. ----END PGP MESSAGE-----
```

Der gesamte Block kann nun einfach per mail verschickt werden.

Verschlüsseln von Dokumenten

Wollen wir nun dieselbe Datei an mehrere Benutzer schicken, so müssen wir für jeden öffentlichen Schlüssel ein Chiffre anlegen, weil ja nur derjenige, der auch den passenden secret-Key besitzt, den Block wieder entschlüsseln kann.

```
moneysac: > gpg -ea -r 0xc2e7d7cf -r 0x82C9390E -o letter.an2.gpg letter
gpg: 32FBE1D3: There is no assurance this key belongs to the named user
4096g/32FBE1D3 2002-03-26 "Sebastian Freundt <sfreundt@hlidskjalf.de>"
Primary key fingerprint: 6CB0 D61E 23A4 275C C2CF A161 94C9 A1AC 82C9 390E
Subkey fingerprint: 4A34 D857 9CED DA0D 84F4 0CF6 3328 D55F 32FB E1D3

It is NOT certain that the key belongs to the person named
in the user ID. If you *really* know what you are doing,
you may answer the next question with yes

Use this key anyway? yes
```

Hier sehen wir noch eine Besonderheit: GPG macht uns darauf aufmerksam, daß wir dem Schlüssel mit der ID `0x82C9390E` nicht genügend trauen. Er fragt uns deshalb, ob wir uns sicher sind, diesem Schlüssel eine Nachricht zukommen zu lassen.

Es entsteht ebenfalls ein Block in der Ausgabedatei, der beide Chiffren enthält.

Entschlüsseln von Dokumenten

Natürlich wollen wir verschlüsselte Dokumente, die mit unserem public-key verschlüsselt worden sind und an uns geschickt werden, wieder lesen können. Wenn wir das mit unserem public-key verschlüsselte Dokument `letenc` entschlüsseln wollen, geben wir ein:

```
moneysac:~> gpg -d -o letdec letencr

You need a passphrase to unlock the secret key for
user: "Marcus Wagner <wagner@math.tu-berlin.de>"
2048-bit ELG-E key, ID 36998739, created 2004-07-12 (main key ID 48314A96)

Enter passphrase: xxxxxxxxxxxx

gpg: encrypted with 2048-bit ELG-E key, ID 36998739, created 2004-07-12
"Marcus Wagner <wagner@math.tu-berlin.de>"
```

und erhalten dadurch eine Datei `letdec`, welche das entschlüsselte Dokument enthält.

Signieren und Verifizieren von Dokumenten

Wenn wir zum Beispiel ein von uns geschriebenes Paper an ein Journal schicken, sollte der Editor nachvollziehen können, dass das Paper wirklich von uns kommt. Wir müssen unser Dokument signieren. Dies geschieht durch Eingabe von

```
moneysac: > gpg -a -o papsig -s paper
You need a passphrase to unlock the secret key for
user: "Marcus Wagner <wagner@math.tu-berlin.de>"
1024-bit DSA key, ID 48314A96, created 2004-07-12

Enter passphrase: xxxxxxxxxxxxxxxx
```

Es wurde nun eine Datei `papsig` angelegt. Diese schickt man an den Empfänger. Der Empfänger würde durch Eingabe von

```
moneysac: > gpg -o papver -d papsig
gpg: Signature made Mon 12 Jul 2004 11:08:33 PM CEST using DSA key ID 48314A96
gpg: Good signature from "Marcus Wagner <wagner@math.tu-berlin.de>"
```

eine Datei `papver` erhalten. Durch die Angabe von `Good signature from "Marcus Wagner <wagner@math.tu-berlin.de>"` weiß der Empfämger, daß die Signatur gültig war. Das Dokument kam also wirklich von M. Wagner.

Signieren und Verifizieren von Dokumenten

Der Nachteil an der ebigen Methode Dateien zu signieren ist, daß es sein kann, daß der Empfänger zwar das Dokument haben und lesen möchte, jedoch keinen Bedarf an der Verifikation der Signatur hat (weil z.B. gerade kein GPG installiert ist). GPG bietet für solche Anlässe die detached signature, also die Signatur in Reinform. Diese wird zusammen mit dem Dokument verschickt, und jeder, der Interesse hat, die Signatur zu überprüfen kann dies tun:

```
moneysac: > gpg -a -o papdetached -b paper
You need a passphrase to unlock the secret key for
user: "Marcus Wagner <wagner@math.tu-berlin.de>"
1024-bit DSA key, ID 48314A96, created 2004-07-12
Enter passphrase: xxxxxxxxxxxxxxxx
```

Damit wird eine Datei `papdetached` angelegt, deren Inhalt nur die reine Signatur darstellt:

```
1. ----BEGIN PGP SIGNATURE-----

Version: GnuPG v1.3.6 (GNU/Linux)
iD8DBQBA8wZJZWhnjUgxSpYRAsseAJ99Acc1Ikk6dv0UxOLySzM17tFHTACeMye3
pMjMyREEY8dTRrN0R4grE9c=
=PA49

1. ----END PGP SIGNATURE-----
```

Diese kann nun mittels

```
moneysac: > gpg --verify papdetached paper
gpg: Signature made Mon 12 Jul 2004 11:44:41 PM CEST using DSA key ID 48314A96
gpg: Good signature from "Marcus Wagner <wagner@math.tu-berlin.de>"
```

überprüft werden.