Sicher verschlüsseln mit GnuPG

Werner Koch

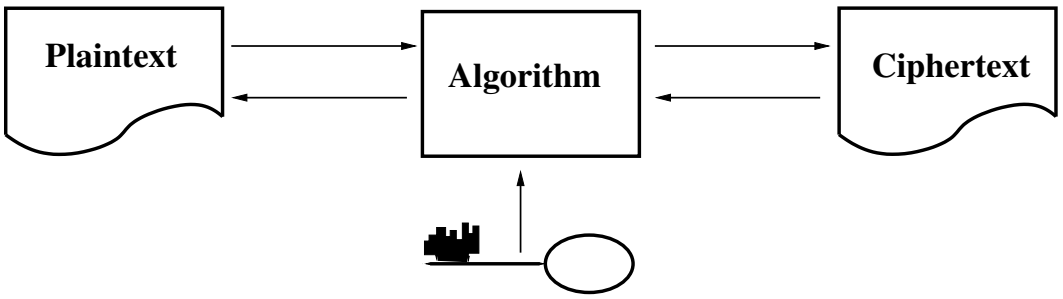
Sommerakademie 2015 — Kiel, 31. August 2015

Inhaltsverzeichnis

1 Schnellkurs Kryptographie

GnuPG implementiert verschiedene Methoden der Verschlüsselung. Für eine sichere Anwendung werden im Folgenden einige Grundlagen beschrieben.

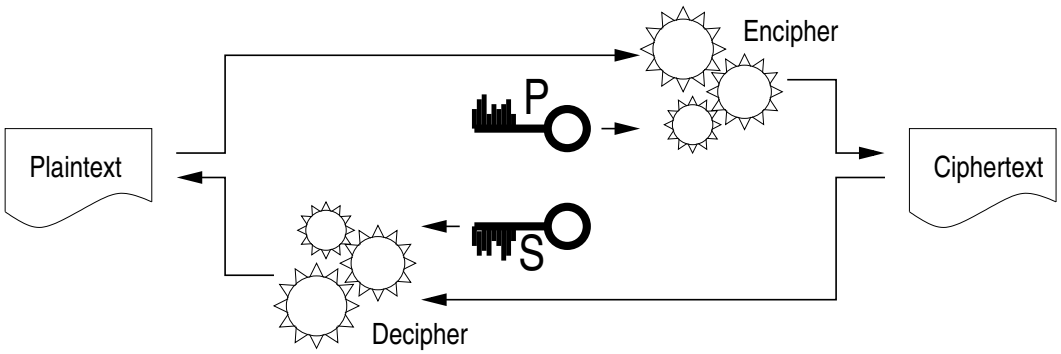
1.1 Symmetrische Verschlüsselung



Dies ist die klassische Verschlüsselung, wie sie seit dem Mittelalter verwendet wird. Stichworte wie „Triple-DES“ oder „AES“ deuten auf diese Verfahren hin.

- Derselbe Schlüssel wird zum Ver- und Entschlüsseln benutzt.
- Sender und Empfänger kennen beide diesen Schlüssel und halten ihn geheim („Shared Secret“).
- Es ist ähnlich einer Passphrase zum Anmelden bei einem Online Service.
- Passphrase basierte Systeme sind unsicher, sofern die Passphrase nicht über eine Zufallsgenerator erzeugt wurde.
- Schlüsselaustausch und -verwaltung sind schwierig und nur praktikabel bei wenigen Relationen.

1.2 Asymmetrische Verschlüsselung (Public-Key)



Diese Art der Verschlüsselung wurde erst in den 1970er Jahren entdeckt. Stichworte wie „RSA“ oder „ECC“ deuten auf diese Verfahren hin.

- Es wird ein Schlüsselpaar aus öffentlichem und privatem (geheimen) Schlüssel verwendet. (P und S in der obigen Darstellung.)
- Der verwendete Algorithmus stellt eine Art Falltür bestehend aus 2 komplementären Funktionen (Encipher und Decipher) dar.
- Der öffentliche Schlüssel kann zur zum Verschlüsseln benutzt werden. Mit diesem Schlüssel kann nicht wieder entschlüsselt werden.
- Nur der private (geheime) Schlüssel ist in der Lage zu entschlüsseln.
- Durch Einstellen des öffentlichen Schlüssels in ein öffentliches Verzeichnis können, ohne vorherigen Schlüsselaustausch, verschlüsselte Dateien ausgetauscht werden.

1.3 Digitale Signaturen

Einige asymmetrische Verschlüsselungsverfahren können auch für digitale Signaturen benutzt werden. Dies ist ein mindest so wichtiges Anwendungsgebiet wie die Verschlüsselung, da man hiermit feststellen kann, ob Daten authentisch sind. Die Verwendung des Schlüsselpaares wird hierbei umgedreht:

- Mit den privaten (geheimen) Schlüssel wird die Signatur erstellt.
- Mit dem öffentlichen Schlüssel kann die Signatur von beliebigen Stellen überprüft werden.

Obgleich es bei einigen Verfahren (z.B. RSA) technisch möglich ist, dasselbe Schlüsselpaar zum Signieren und Verschlüsseln zu benutzen, sollte dies in der Praxis vermieden werden. Entweder werden zwei getrennte Schlüsselpaare erzeugt, oder, wie bei OpenPGP, der öffentliche Schlüssel zum Verschlüsseln kryptographisch gesichert an den Schlüssel zum Signieren gebunden und als eine Einheit verwaltet.

1.4 Algorithmen

Gängige Public-Key Verfahren sind:

- RSA (verschlüsseln, signieren)
- DSA (signieren)
- Elgamal (verschlüsseln)
- ECC, Elliptische Kurven (verschlüsseln, signieren)
  - Kürzere Schlüssel (z.B. 256 bit)
  - Gleiche Sicherheit (z.B. wie RSA mit 4096 bit)

1.5 Hybridverfahren

Public-Key Verfahren sind um mehrere Größenordnungen langsamer als symmetrische Verfahren. Sie lassen sich deswegen nicht einsetzen um große Datenmengen zu verschlüsseln. Stattdessen wird praktisch immer ein Hybridverfahren eingesetzt:

- Ein zufälliger Sitzungsschlüssel von 256 Bit wird erzeugt,
- dieser wird mit einem Public-Key Verfahren an den Empfänger verschlüsselt,
- die Daten werden mit dem Sitzungsschlüssel symmetrisch verschlüsselt.

Soll eine Datei an mehrere Empfänger verschlüsselt werden, so wird lediglich derselbe Sitzungsschlüssel für jeden Empfänger neu verschlüsselt. Hierdurch wird die verschlüsselte Datei nur minimal größer.

1.6 Zertifikate und PKI

Obgleich das Public-Key Verfahren die Schlüsselverwaltung wesentlich vereinfacht, stellt sich immer noch die Frage: „Ist der öffentliche Schlüssel authentisch?“. Es gibt verschiedene Wege dieses Frage zu beantworten.

- Von Hand verwaltete Liste gültiger Schlüssel (z.B. im Adreßbuch).
- Ein Verzeichnis von gültigen Schlüsseln.
- Eine zentrale PKI (Public-Key Infrastructure) die auf einem hierarchisch aufgebauten System von Zertifizierungsstellen beruht.
- Eine dezentrale PKI wie das Web-of-Trust.
- Ein lokales Trust-On-First-Use Verfahren erkennt geänderte Schlüssel nach deren ersten Verwendung.

2 Basisfunktionen

Im Weiteren werden die Basisfunktionen von GnuPG anhand von Beispielen dargestellt. Es werden die englischen Ausgaben verwendet; durch entsprechende Installation können landessprachliche Ausgaben eingestellt werden.
Folgende Konventionen werden in den Beispielen verwendet: Eine Eingabezeile auf der Shell (Kommandointerpreter) wird durch ein `$` am Anfang der Zeile gekennzeichnet. Ein \ am Ende einer Zeile zeigt an, daß die Eingabezeile auf der nächsten Zeile fortgesetzt wird. Teilweise werden nicht relevante Ausgaben weggelassen; dies wird durch ein `[...]` angezeigt.

2.1 Erzeugen eines Schlüsselpaares

Um Gpg mit dem Public-Key Verfahren zu verwenden, ist ein Schlüsselpaar zu erzeugen:

```
$ gpg --gen-key
gpg (GnuPG) 2.1.7; Copyright (C) 2015 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: keybox '/home/wk/b/gnupg/kiel2015/pubring.kbx' created
Note: Use "gpg2 --full-gen-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: John Steed
Email address: steed@example.org
You selected this USER-ID:
    "John Steed <steed@example.org>"

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

Gpg erzeugt nun den Schlüssel und nach einiger Zeit (normalerweise wenige Sekunden, je nach Betriebssystem aber u.U. auch einige Minuten) wird dieser angezeigt:

```
gpg: key 3F567FB6 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: public key of ultimately trusted key 912FCB93 not found
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
pub   rsa2048/3F567FB6 2015-08-12
      Key fingerprint = AF19 1E21 6B28 0B02 65E6  50C1 6415 179B 3F56 7FB6
uid         [ultimate] John Steed <steed@example.org>
sub   rsa2048/63B40B8C 2015-08-12
```

2.2 Backup

Der private Schlüssel ist wichtig! Ohne diesen Schlüssel können verschlüsselte Daten nicht mehr entschlüsselt werden. Es sollte darauf geachtet werden, ein Backup zu erzeugen. Einige Hinweise hierzu:

- Für ein Disaster Recovery sollte ein Ausdruck erstellt werden und verschlossen verwahrt werden. Bei GnuPG Versionen vor 2.1 kann hierzu das Programm „paperkey" benutzt werden. Für neuere GnuPG Versionen existiert noch kein Tool; der private Schlüssel kann aber auch mit `gpg -a --export-secret-key` exportiert und gedruckt werden.
- Passphrase getrennt notieren
- Lokalen Drucker verwenden.
- Ein Backup unter Unix kann so erfolgen:

```
$ tar czf backup-keys-DATUM.tar.gz  --exclude random_seed ~/.gnupg
```

- Für ein Backup unter Windows kann ein dediziertes Backup Tool verwendet werden oder aber mit diesen Kommandos:

```
$ gpgconf --list-dirs
$ cd DIR
$ del random_seed
$ gpgtar --skip-crypto -eo backup-keys-DATUM.tar .
```

2.3 Import von Schlüsseln

Schlüssel sind gelegentlich auf der entsprechenden Webseite vorhanden. Diese sollten dann in einer Datei abgespeichert und mit

```
gpg --import DATEI
```

importiert werden . Unter Unix kann man es auch einfacher mit folgenden Kommandos machen:

```
$ wget -O a.key https://www.datenschutzzentrum.de/uploads/uld/uld.asc
$ gpg --import a.key
$ rm a.key
```

Alternativ können Schlüssel über einen Keyserver geholt werden:

```
$ gpg --keyserver keys.gnupg.net --recv-key 0D75199E11357324
gpg: key 0D75199E11357324: public key "ULD-SH <mail[...]>" imported
gpg: public key of ultimately trusted key 0F1EB16A912FCB93 not found
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
gpg: Total number processed: 1
gpg:               imported: 1
```

Die angegebene Adresse ist ein Pool von Keyservern, sie sich untereinander replizieren. Die Verbindung wird zu einem zufälligen Server aufgebaut. Es ist aber auch möglich, einen festen Keyserver anzugeben. Um die Option für den Keyserver nicht immer eingeben zu müssen, sollte sie in die `gpg.conf` Datei eingetragen werden.

2.4 Verschlüsseln

Das erste Beispiel verschlüsselt ein PDF Dokument:

```
$ gpg -v -e -r mail@datenschutzzentrum.de datei.pdf
gpg: using PGP trust model
gpg: using subkey A749BED409A66C9A instead of primary key 0D75199E11357324
```

- Das `-v` ist optional um anzuzeigen was passiert.
- Das `-e` wählt Verschlüsselung aus.
- Das `-r` gibt den Empfänger an (hier über eine Mailadresse).
- `datei.pdf` ist die zu verschlüsselnde Datei.
- Gpg wählt automatisch einen passenden Unterschlüssel aus. Ein OpenPGP Schlüssel besteht aus einem Hauptschlüssel (Primary Key) und beliebig vielen Unterschlüsseln (Subkey, oder Secondary Key)

Nun wird Gpg allerdings feststellen, daß keine Informationen zu der Gültigkeit des Schlüssels vorliegen und den Benutzer um Bestätigung bitten:

```
gpg: A749BED409A66C9A: There is no assurance this key belongs to the named user
sub  elg4096/A749BED409A66C9A 2008-04-11 ULD-SH <mail@datenschutzzentrum.de>
Primary key fingerprint: D092 F1B5 AB9F D68E 4DA0  3633 0D75 199E 1135 7324
     Subkey fingerprint: 4E31 0B46 A394 DE69 D56A  6F82 A749 BED4 09A6 6C9A

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? (y/N) y
gpg: reading from 'datei.pdf'
gpg: writing to 'datei.pdf.gpg'
gpg: ELG/AES256 encrypted for: "A749BED409A66C9A ULD-SH <mail@[...]entrum.de>"
```

Um derartige manuelle Prüfungen zu vermeiden, sollte man nach der Überprüfung des Fingerprints (die Prüfung des „Primary key fingerprint“ ist vollkommen ausreichend) den Schlüssel vertrauenswürdig setzen.

2.5 Schlüssel vertrauenswürdig setzen

```
$ gpg --lsign-key 0D75199E11357324
[...]
pub  dsa3072/0D75199E11357324
     created: 2008-04-11  expires: never       usage: SC
     trust: unknown       validity: unknown
Primary key fingerprint: D092 F1B5 AB9F D68E 4DA0  3633 0D75 199E 1135 7324

     ULD-SH <mail@datenschutzzentrum.de>

Are you sure that you want to sign this key with your
```

```
key "John Steed <steed@example.org>" (6415179B3F567FB6)

The signature will be marked as non-exportable.

Really sign? (y/N) y
```

Es wird hierbei das Kommando `--lsign-key` zum lokalen signieren verwendet (dies ist eine vereinfachte Version des Kommandos `--edit-key`). Soll die Bestätigung veröffentlicht werden (Stichwort: Web-of-Trust), so ist das Kommando `--sign-key` (ohne das „l“ für „local“) zu verwenden. Dies kann auch nachträglich gemacht werden.
Wichtig ist, daß der angezeigte Fingerprint und die Mailadresse zusammenpassen; dies kann durch den Vergleich mit einer Publikation des Fingerprints und der Mailaddresse geschehen oder aber auch durch ein Telefonat. Es muß immer ein anderer Kommunikationskanal verwendet werden; eine Nachfrage per Email ist unsicher.
Sofern deutsche Ausgaben erscheinen, kann neben „y“ oder „yes“ auch „j“ oder „ja“ eingeben werden.

2.6 Schlüsselstatus anzeigen

Um den Fingerprint eines Schlüssels anzusehen kann folgendes Kommando verwendet werden:

```
$ gpg --fingerprint 0D75199E11357324
gpg: checking the trustdb
[...]
pub   dsa3072/0D75199E11357324 2008-04-11
      Key fingerprint = D092 F1B5 AB9F D68E 4DA0  3633 0D75 199E 1135 7324
uid                  [ full  ] ULD-SH <mail@datenschutzzentrum.de>
sub   elg4096/A749BED409A66C9A 2008-04-11
```

Im Beispiel wurde die Key-ID angegeben, es kann aber auch der Name oder die Mailadresse angegeben werden. Hier ist der Schlüssel als vertrauenswürdig markiert (das „full“ in eckigen Klammern). Die möglichen Vertrauensstufen sind in der folgenden Tabelle angegeben:

Stufe	(deutsch)	Bedeutung
unknown	unbekannt	Vertrauensstufe wurde noch nicht ermittelt
expired	verfallen	Der Schlüssel ist verfallen
undefined	undefiniert	Es liegen nur ungenügende Informationen vor
never	niemals	Diesem Schlüssel darf nicht vertraut werden
marginal	marginal	Teilweises Vertrauen ist vorhanden (via Web-of-Trust)
full	vollständig	Diesem Schlüssel wird vollständig vertraut
ultimate	ultimativ	Diesem Schlüssel wird ultimativ vertraut; dies ist gleichgestellt mit dem Besitz des privaten Schlüssels

2.7 Signieren

Eine normale Signatur kann so erstellt werden:

```
$ gpg -v -s datei.pdf
gpg: using PGP trust model
gpg: writing to 'datei.pdf.gpg'
gpg: RSA/SHA256 signature from: "6415179B3F567FB6 John Steed <steed@example.org>"
```

- Das `-s` (oder `--sign`) wählt Signieren aus.
- `datei.pdf` ist die zu signierende Datei.
- `datei.pdf.sig` ist die erstellte Datei mit Signatur.

Oft ist es wünschenswert, die Signatur zusätzlich zu der Originaldatei zu erstellen. So kann die Datei benutzt werden, ohne daß sie erst mit gpg ausgepackt werden muß. Dies wird durch eine abgetrennte (detached) Signatur erreicht:

```
$ gpg -v -b datei.pdf
gpg: using PGP trust model
gpg: writing to 'datei.pdf.sig'
gpg: RSA/SHA256 signature from: "6415179B3F567FB6 John Steed <steed@example.org>"
```

- Das `-b` (oder `--detach-sign`) wählt abgetrenntes Signieren aus.
- `datei.pdf` ist die zu signierende Datei.
- `datei.pdf.sig` ist die erstellte abgetrennte Signatur.

Gelegentlich soll ein anderer privater Schlüssel zum signieren benutzt werden. Hierzu wird die Option `-u` verwendet:

```
$ gpg -v -b -u peel datei.pdf
gpg: writing to 'datei.pdf.sig'
gpg: EDDSA/SHA256 signature from: "EA9644E68E27FD07 Emma Peel <peel@example.org>"
```

Der Schlüssel kann, wie üblich, über den Namen, die Mailadresse oder die Key-ID ausgewählt werden. Es ist auch möglich, diese Option mehrfach zu verwenden, um mit mehreren privaten Schlüsseln zu signieren. Dieses Beispiel verwendet übrigens einen anderen Algorithmus als RSA (EdDSA, welches auf elliptischen Kurven basiert).

2.8 Signatur prüfen

Eine der häufigsten Operationen ist die Prüfung einer Signatur. Hiermit wird festgestellt, ob eine Datei authentisch und nicht modifiziert ist.

```
$ gpg -v --verify datei.pdf.sig datei.pdf
gpg: Signature made Sun 16 Aug 2015 09:24:06 AM CEST
gpg:                using EDDSA key EA9644E68E27FD07
[...]
gpg: Good signature from "Emma Peel <peel@example.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: CA4A EF4F 0065 91A8 DF82  761F EA96 44E6 8E27 FD07
gpg: binary signature, digest algorithm SHA256, key algorithm ed25519
```

- `--verify` wählt die Prüfung einer Signatur aus.
- `datei.pdf.sig` ist die abgetrennte Signatur.
- `datei.pdf` ist die zu prüfende Datei.

Das angezeigte Datum, ist das Datum an dem die Signatur erzeugt wurde. Dies ist aber nur relevant wenn die Signatur auch gültig ist; „Good Signature“ bescheinigt das dies der Fall ist. Es ist nun noch nicht klar, ob dies auch wirklich der Schlüssel von Emma ist. Um dies zu prüfen muß der angezeigte Fingerprint mit einer anderen Quelle verglichen werden. Dies ist identisch zu dem bereits beschrieben Verfahren bei der Verschlüsselung. Auch hier kann durch `--lsign-key` die positive Überprüfung des Fingerprints festgehalten werden, so daß die obige Warnung nicht mehr erscheint.

2.9 Wie man OpenPGP Dateien erkennt

Hier einige Hinweise wie OpenPGP Schlüssel oder mit diesen verschlüsselte oder signierte Daten erkannt werden können:

- Mittels eines Tools anhand des Inhalts: Entweder durch einen Versuch mit `gpg`, oder auf Unix, mit dem Tool `file`.
- Mittels der Bibliotheksfunktion `gpgme_data_identify` von Libgpgme.
- Bei „armored“ Dateien auch visuell anhand des Inhalts (z.B. `-----BEGIN PGP MESSAGE-----`).
- Anhand der Dateiendung (per Konvention):

`.sig`	Binäre abgetrennte Signatur
`.pub`	Datei mit Public-Key(s)
`.sec`	Datei mit Secret-Key(s)
`.asc`	Armored OpenPGP Datei
`.gpg`	Andere binäre OpenPGP Datei
`.pgp`	Dito, aber von PGP verwendet

2.10 Komprimieren

Hinweise zur Kompression von Daten:

- Verschlüsselte Daten können nicht mehr komprimiert werden,
- `gpg` komprimiert deswegen die Daten bevor sie verschlüsselt werden,
- Bereits verschlüsselte Daten werden i.d.R. erkannt und die Komprimierungsstufe wird ausgeschaltet. Explizit kann dies durch die Option `-c 0` erzwungen werden.
- Beim Entschlüsseln werden die Daten automatisch dekomprimiert. Wie bei allen Komprimierungsverfahren kann dies als ZIP Bombe benutzt werden (Denial-of-Service); die Option `--max-output` kann dem entgegenwirken.

2.11 S/MIME

Neben OpenPGP unterstützt GnuPG auch S/MIME. Genaugenommen wird hier X.509/CMS implementiert, da die eigentliche S/MIME Struktur vom Mailprogramm erstellt wird.

- S/MIME ist nicht kompatibel zu OpenPGP
- Es werden andere Schlüssel verwendet und diese müssen durch eine CA zertifiziert werden.
- Das Tool `gpgsm` wird anstatt von `gpg` verwendet. Es ist in der Bedienung sehr ähnlich, hat Aufgrund des anderen Protokolls einige Unterschiede.
- Es können auch X.509 Zertifikate und Zertifizierungsanfragen (CSR) für Webserver erstellt werden sowie mittels `gpgsm` verwaltet werden.

3 Automatisieren mit GnuPG

GnuPG wird meistens nicht direkt verwendet sondern als kryptographisches Backend von anderen Programmen benutzt. Die Integration in Mailprogramme sprengt den Rahmen dieses Vortrags, deswegen werden hier nur einige Hinweise zu Verwendung von GnuPG in Skripts gegeben.

3.1 Pipelines

Als Unix Programm arbeitet `gpg` sehr gut als Filter in einer Pipeline. Dies funktioniert prinzipiell auch unter Windows. Ein Beispiel mag dies verdeutlichen:

```
$ tar cf - /var/log \
  | gpg --batch -e --always-trust -r 0x12345678abcdef0 \
  | ssh backup@archive 'cat >"backup-$(date +%Y-%m-%d).tar.gpg"'
```

Hier werden alle Log Dateien eines Unix Systems verschlüsselt und auf dem Server „archive“ in einer Datei gespeichert. Im Einzelnen:

- Das Programm `tar` kopiert rekursiv alle Dateien im `/var/log/` Verzeichnis nach `stdout`,
- `gpg` liest diese Dateien, verschlüsselt sie an den Schlüssel mit der Key-ID 0x12345678abcdef0 und gibt die verschlüsselten Daten wiederum auf `stdout` aus.
- Die Secure Shell `ssh` stellt eine Verbindung zum Rechner „archive“ her und führt dort das Kommando `cat` aus, welches seine Ausgabe in eine Datei, mit Tagesdatum im Namen, im Heimatverzeichnis des Benutzers „backup” schreibt. Da `ssh` den `stdout` von `gpg` liest wird so die Ausgabe von `gpg` in die Datei auf dem anderen Rechner geschrieben.

Zur Automatisierung werden folgende Optionen für `gpg` verwendet:

- `--batch` verhindert Nachfrage von `gpg` und sollte immer verwendet werden, wenn keine Benutzernachfragen beantwortet werden können.
- `-e` fordert Verschlüsselung an.
- `--always-trust` ist ein Alias für `--trust-model=always` welches bestätigt, daß die angegebenen Empfängerschlüssel gültig sind. Dies ist in der Regel einfacher zu handhaben als dafür zu sorgen, daß der Schlüssel signiert ist (z.B. mittels `--lsign-key`).
- `0x12345678abcdef0` ist eine Key-ID und spezifiziert den zu verwendenden Public-Key. Für den Produktiveinsatz ist hier allerdings die Verwendung des Fingerprints zu empfehlen, da dieser den Schlüssel garantiert eindeutig spezifiziert.

Als weiteres Beispiel wird nun eine verschlüsselte Datei von einem entfernten Rechner geholt und entpackt:

```
$ cd restored-logs
$ ssh backup@archive 'cat DATEI.tar.gpg' \
  | gpg --batch -d  --max-output 0x80000000 \
  | tar xf -
```

Dies ist im Wesentlichen die Umkehrung des vorherigen Beispiels. Der Dateiname wird hier hier direkt angegeben und `tar` wird zum Entpacken mit Eingabedaten von `stdin` aufgerufen. Änderungen beim Aufruf von `gpg` sind:

- `-d` (oder `--decrypt`) wählt Entschlüsselung aus. Da dies die Voreinstellung ist, ist die Angabe dieses Kommandos nicht unbedingt notwendig.
- `--max-output` gibt die maximal erwartete Länge der Ausgabe in Bytes an. Dies dient der Verhinderung von ZIP Bomben bei der Verwendung von Dateien aus unbekannter Quelle. Es sollte ein Wert gewählt werden, der mit Sicherheit größer als die Dateigröße des entschlüsselten und dekomprimierten Archiv ist. Im Beispiel werden 2 GiB in Hex-Notation angegeben („0x“). Es ist zu beachten, daß auf 32-Bit Unix Systemen, sowie allen Windows Systemen, als maximaler Wert `0xffffffff` (4 GiB - 1) angegeben werden kann.

3.2 Optionen bei unbeaufsichtigtem Betrieb

Sofern GnuPG nicht interaktiv verwendet wird, sind diese Optionen interessant:

- `--batch` schaltet alle Abfragen aus.
- `--yes` benutzt implizit “Ja” für die meisten Abfragen und kann z.B. mit `--batch` benutzt werden um Dateien zu überschreiben. Auf einige Abfragen hat diese Option allerdings keine Auswirkung.
- Die normalen Ausgabe sind nur für den menschlichen Genuß gedacht. Sämtliche Automatisierung soll das Status Interface benutzen. Hierzu kann die Option `--status-fd 2` verwendet werden.
- `--max-output N` kann zur Verhinderung von ZIP Bomben benutzt werden, da hier die Ausgabe spätestens nach N Bytes abgebrochen wird.
- Anstatt einen Schlüssel mit `--lsign-key` zu signieren, kann bei wenigen Schlüsseln auch die Option `--trust-model=always` benutzt werden, um `gpg` anzuzeigen, daß die mit `-r` angegeben Schlüssel authentisch sind. Sinnvollerweise werden die Schlüssel in diesem Fall über den Fingerprint und nicht nur über den Namen oder die Key-ID spezifiziert.

3.3 Best Practices

Abschließend noch einige Hinweise, wie `gpg` am besten betrieben wird:

- Nach Möglichkeit, Schlüssel immer per Fingerprint spezifizieren.
- Signaturschlüssel explizit auswählen (Option `-u`).
- Option `--encrypt-to` benutzen um die verschlüsselten Dateien selbst noch entschlüsseln zu können. Wenn Anonymität des Absenders gefragt wird, darf dies allerdings nicht benutzt werden; die Option `--no-encrypt-to` sollte dann benutzt werden.
- Definierte Konfigurationsdateien benutzten. Falls das Standard GnuPG Verzeichnis nicht benutzt wird, so ist die Umgebungsvariable `GNUPGHOME` zu setzen.
- Auf Servern keine Passphrase setzen bzw. Smartcard benutzen.
- Immer aktuelle Versionen von GnuPG verwenden.

4 Zusammenfassung

- GnuPG ist vielseitig zu verwenden,
- es kann sehr leicht in Skripte eingebunden werden,
- es verwendet sichere und etablierte Algorithmen und Protokolle,
- es ist kostengünstig
- und zukunftssicher.

Weitere Information:

- `https://gnupg.org`
- `https://wiki.gnupg.org`
- `https://gnupg.org/ftp/blurbs/kiel-2015_sicher-verschl-mit-gnupg.pdf`
- `https://gnupg.org/ftp/blurbs/kiel-2015_sicher-verschl-mit-gnupg_print.pdf`
- `https://gnupg.org/ftp/blurbs/kiel-2015_sicher-verschl-mit-gnupg_handout.pdf`