Einführung in GnuPG

Mario Haustein

8. Januar 2021

Was ist GnuPG

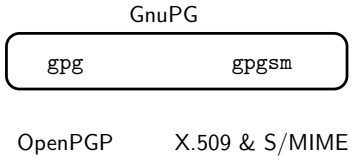
- GNU Privacy Guard
- Kryptografie zum Schutz von Dateien, Mails, Chats, Git-Commits, SSH-Anmeldung, ...
- Implementiert OpenPGP- und X.509-S/MIME-Standard
- In viele Softwareprodukte integriert

- Hauptentwickler: Werner Koch
- Aktuelle Version: 2.2

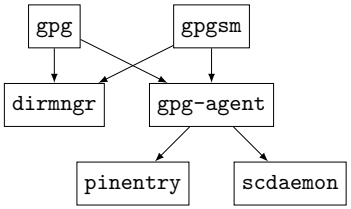
- Vom Bundesamt für Sicherheit in der Informationstechnik (BSI) für „Verschlusssachen – nur für den Dienstgebrauch“ (VS-NfD) zugelassen.
- Viele zweckmäßige Designentscheidungen
- Viele Funktionen

Begriffe

OpenPGP	Offener Standard für Dateiformate zur verschlüsselten Kommunikation (RFC 4880, RFC 5581, RFC 6637)
X.509	Internationale Standard für Public-Key-Infrastrukturen
S/MIME	Offener Standard für verschlüsselte MIME-Nachrichten (RFC 2633)
GnuPG	Implementierung von OpenPGP (und X.509 + S/MIME)
`gpg`	GnuPG-Werkzeug für OpenPGP
`gpgsm`	GnuPG-Werkzeug für X.509 und S/MIME



Module (Teilmenge)



`gpg` Kommandozeilenwerkzeug für OpenPGP

`gpgsm` Kommandozeilenwerkzeug für X.509 + S/MIME

`dirmngr` Programm zum Zugriff auf Schlüssel- und Zertifikatsverzeichnisse

`gpg-agent` Hintergrundprozess zur Verwaltung von geheimen Schlüsseln, Passwörtern und Durchführung kryptografischer Operationen

`pinentry` Programm zur sicheren Eingabe von Geheimnissen

`scdaemon` Hintergrundprozess zum Zugriff auf Krypto-Hardware (Smart-Cards)

Kommandozeile

`$ gpg [`⟨*Optionen*⟩ `...] `⟨*Kommando*⟩

- **Mehrere** Optionen
  - z.B.: Datenformate, Empfänger, ...
  - `-o`, `--output` → Ausgabedatei
  - `-a`, `--armor` → Nur druckbare Zeichen in Ausgabe verwenden
- **Ein** Kommando
  - Immer am Ende
  - z.B.: verschlüsseln, entschlüsseln, signieren, prüfen, Schlüssel bearbeiten, ...

- Es gibt auch grafische Werkzeuge
  - GPA → `http://gpa.wald.intevation.org/`
  - Kleopatra → `https://apps.kde.org/en/kleopatra`
  - Gpg4win → `https://www.gpg4win.de/`
  - Kgpg → `https://apps.kde.org/en/kgpg`

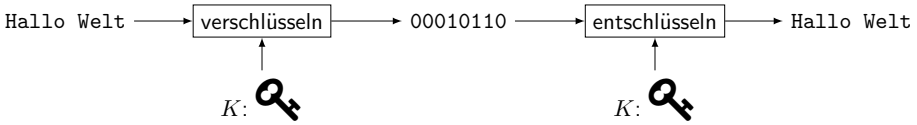
Kerckhoffs' Prinzip & symmetrische Chiffren

Kerckhoffs' Prinzip

Die Sicherheit eines Verschlüsselungsverfahrens hängt ausschließlich von der Geheimhaltung des Schlüssels ab.

- Symmetrische Verschlüsselung

⇒ Zum ver- und entschlüsseln wird der selbe Schlüssel K verwendet.



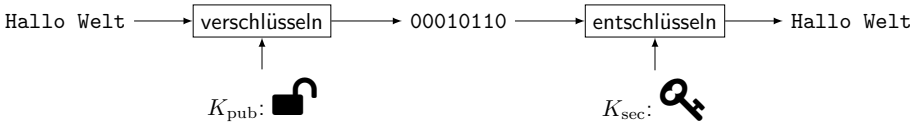
Symmetrische Verschlüsselung

```
$ echo "Hallo Welt" | gpg -a -o geheim.asc --symmetric
$ cat geheim.asc
-----BEGIN PGP MESSAGE-----

jA0EBwMC18E+uCQgQ6Pw0kAB1vJKjHLDFwvsDRCO6xemQCB+URYS2et20PXIdIR6
qyAaKzLDEM8TfCbeeYfrtZvaqNbjgM+j4EJfwKl2fIS6
=c/vG
-----END PGP MESSAGE-----

$ gpg --decrypt geheim.asc
gpg: AES verschlüsselte Daten
gpg: Verschlüsselt mit einer Passphrase
Hallo Welt
```

- Während des Vorgangs fragt `pinentry` ein Passwort ab, aus dem der Schlüssel abgeleitet wird.

Asymmetrische Chiffren

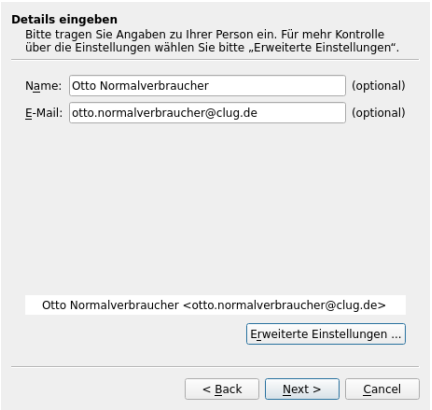
- Zum ver- und entschlüsseln werden verschiedene Schlüssel verwendet.
  - öffentlicher Schlüssel: verschlüsseln, Unterschriften prüfen
  - geheimer Schlüssel: entschlüsseln, unterzeichnen

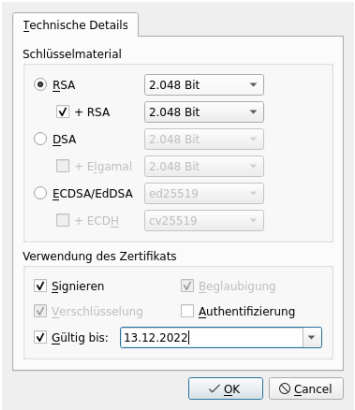


Schlüsselerzeugung

```
$ gpg --generate-key
Ihr Name ("Vorname Nachname"): Otto Normalverbraucher
Email-Adresse:
Sie haben diese User-ID gewählt:
    "Otto Normalverbraucher"

Ändern: (N)ame, (E)-Mail oder (F)ertig/(A)bbrechen? f
Öffentlichen und geheimen Schlüssel erzeugt und signiert.

pub   rsa2048 2020-12-13 [SC] [verfällt: 2022-12-13]
      33BF4ABDE277F319316061D37F9C6BD409D4F133
uid                      Otto Normalverbraucher
sub   rsa2048 2020-12-13 [E] [verfällt: 2022-12-13]
```


Schlüsselerzeugung mit Kleopatra





Schlüssel-IDs

- **User ID**: lesbarer Bezeichner eines Schlüssel
- Die UID besteht aus Name oder E-Mail-Adresse (oder beidem).
- Ein Schlüssel kann mehrere UIDs haben.
- Auf eine UIDs können mehrere Schlüssel ausgestellt sein.

- **Fingerabdruck**: eindeutiger Bezeichner des Schlüssels
  - kryptografischer Hash-Wert des Schlüssels
  - gleicher Hash ⇒ gleicher Schlüssel (Hash-Kollision vernachlässigt)

- **Key-ID**: letzten 8 bzw. 16 Stellen des Fingerabdrucks
  - selten verwendet

Geheime Schlüssel anzeigen

```
$ gpg --keyid-format long -K
sec   rsa2048/7F9C6BD409D4F133❸ 2020-12-13 [SC]❺ [verfällt: 2022-12-13]❹
      33BF4ABDE277F319316061D37F9C6BD409D4F133❷
uid                 [ ultimativ ] Otto Normalverbraucher❶
ssb   rsa2048/266FE5B7B1151472  2020-12-13 [E]❺ [verfällt: 2022-12-13]
```

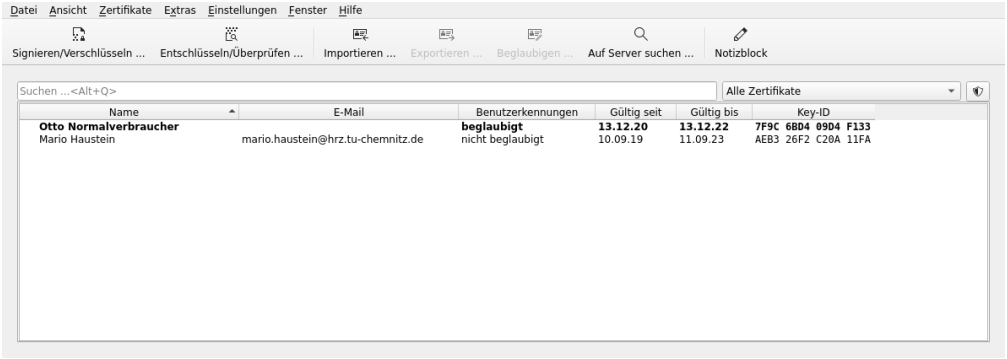
❶ Schlüssel-UID
❷ Fingerabdruck
❸ Key-ID (16-stellig)
❹ Gültigkeitszeitraum

❺ Schlüsselzweck
- S sign: unterzeichnen
- C certify: Vertrauen aussprechen
- E encrypt: verschlüsseln
- A authenticate: Identität beweisen

Öffentliche Schlüssel anzeigen (= alle Schlüssel)

```
$ gpg -k
pub   rsa2048 2020-12-13 [SC] [verfällt: 2022-12-13]
      33BF4ABDE277F319316061D37F9C6BD409D4F133
uid         [ ultimativ ] Otto Normalverbraucher
sub   rsa2048 2020-12-13 [E] [verfällt: 2022-12-13]

pub   rsa4096 2019-09-10 [SC] [verfällt: 2023-09-11]
      D1D5B143F82D129E1172AADDAEB326F2C20A11FA
uid         [ unbekannt ] Mario Haustein <mario.haustein@hrz.tu-chemnitz.de>
uid         [ unbekannt ] Mario Haustein <hamari@tu-chemnitz.eu>
uid         [ unbekannt ] Mario Haustein <mario.haustein@cs.tu-chemnitz.de>
uid         [ unbekannt ] Mario Haustein <mario.haustein@informatik.tu-chemnitz.de>
uid         [ unbekannt ] Mario Haustein <mario.haustein@s2005.tu-chemnitz.de>
uid         [ unbekannt ] Mario Haustein <hamari@informatik.tu-chemnitz.de>
uid         [ unbekannt ] Mario Haustein <hamari@cs.tu-chemnitz.de>
uid         [ unbekannt ] Mario Haustein <hamari@hrz.tu-chemnitz.de>
sub   rsa4096 2019-09-10 [E] [verfällt: 2023-09-11]
```


Schlüsselliste in Kleopatra



Schlüsselexport / -import

- Öffentlichen Schlüssel in Datei exportieren

  ```
  $ gpg -o otto.gpg --export "Otto Normalverbraucher"
  ```

- Geheimen Schlüssel in Datei exportieren (z. B. als Sicherheitskopie)

  ```
  $ gpg -o otto.gpg --export-secret-keys "Otto Normalverbraucher"
  ```

- Schlüssel importieren

  ```
  $ gpg --import otto.gpg
  ```

Ex- und Import über die Standardausgabe und -eingabe

- Export

```
$ gpg -a --export "Otto Normalverbraucher"
-----BEGIN PGP PUBLIC KEY BLOCK-----
mQENBF/WcTMBCADrIma/WeowYnW7ADiOlHJM4Pn1UZCzzItB6DfDEyLBowJvEmQK
ZEiGttueVT/t57yw7X5sgQYsAFYfmpXzbOt9gvacZqMwlbaq9w3XzXZU0YgDMBbo
...
xc6xyhTh8LvCPhLOcROUoc6gZs26GJE/oVif73nCOdxxdb7bROOKtNm+KcrSOtgR
FTqgHiVakgGi+YpDPa8tLlGewfhBwKPb6LkfjWHY
=H17G
-----END PGP PUBLIC KEY BLOCK-----
```

- Import

```
$ gpg --import << EOF
> -----BEGIN PGP PUBLIC KEY BLOCK-----
> ...
> -----END PGP PUBLIC KEY BLOCK-----
> EOF
```


Versuchen wir eine Botschaft zu verschlüsseln

- ❶ . . . Empfänger ❷ . . . Nachricht

```
$ gpg -a -r mario.haustein@hrz.tu-chemnitz.de❶ --encrypt << EOF
> Hallo Welt❷
> EOF
gpg: E3BE67AFA5B8FC22: Es gibt keine Garantie, daß dieser Schlüssel wirklich dem
     angegebenen Besitzer gehört.

sub   rsa4096/E3BE67AFA5B8FC22 2019-09-10 Mario Haustein <mario.haustein@...>
 Haupt-Fingerabdruck  = D1D5 B143 F82D 129E 1172  AADD AEB3 26F2 C20A 11FA
 Unter-Fingerabdruck  = 2866 4B36 4640 98AC 7CD7  B6C3 E3BE 67AF A5B8 FC22

...
Diesen Schlüssel trotzdem benutzen? (j/N) n
gpg: [stdin]: encryption failed: Unbrauchbarer öffentlicher Schlüssel
```

⇒ GnuPG beschützt uns vor unserer eigenen Fahrlässigkeit.

Vertrauen

Fehlerhafte Annahme

Öffentliche Schlüssel sind nicht geheim $\Longrightarrow$ Es gibt nichts zu beachten.

Realität

Geheime Schlüssel erfordern viel Sorgfalt. Öffentliche Schlüssel auch!

- Vertrauen $\Longleftrightarrow$ öffentlicher Schlüssel gehört der Person, auf die er ausgestellt ist.
- Erfordert persönliche Überzeugung.

Einem Schlüssel das Vertrauen aussprechen

1. Öffentlichen Schlüssel anfordern bzw. herunterladen und importieren.
2. Vom Schlüsselinhaber auf sicherem Weg den Fingerabdruck erfragen.
3. Übereinstimmung mit importieren Schlüssel prüfen.
4. Fremden Schlüssel kryptografisch mit dem eigenen geheimen Schlüssel unterschreiben.

⇒ Niemand kann unser Vertrauensverhältnis manipulieren.

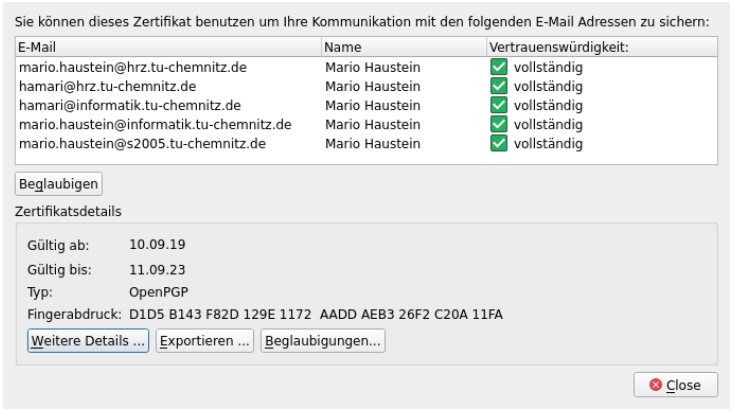
- `--lsign-key` → nichtöffentliche Unterschrift
- `--sign-key` → öffentliche Unterschrift

⇒ Wir bezeugen anderen Teilnehmern die Echtheit des fremden Schlüssels.

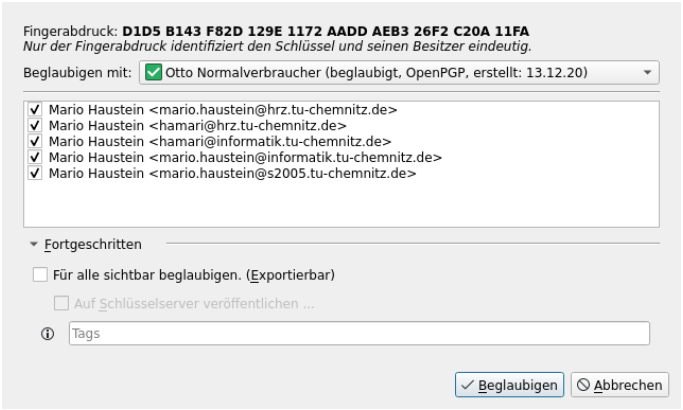
Fremde Schlüssel unterschreiben

```
$ gpg --lsign-key mario.haustein@hrz.tu-chemnitz.de

...
Wirklich alle User-IDs beglaubigen? (j/N) j

pub   rsa4096/AEB326F2C20A11FA
      erzeugt: 2019-09-10  verfällt: 2023-09-11  Nutzung: SC
      Vertrauen: unbekannt      Gültigkeit: unbekannt
 Haupt-Fingerabdruck  = D1D5 B143 F82D 129E 1172  AADD AEB3 26F2 C20A 11FA

      Mario Haustein <mario.haustein@hrz.tu-chemnitz.de>
      ...

Dieser Schlüssel wird 2023-09-11 verfallen.
Sind Sie wirklich sicher, daß Sie vorstehenden Schlüssel mit Ihrem
Schlüssel "Otto Normalverbraucher" (7F9C6BD409D4F133) beglaubigen wollen

Die Signatur wird als nicht-exportfähig markiert werden.

Wirklich signieren? (j/N) j
```

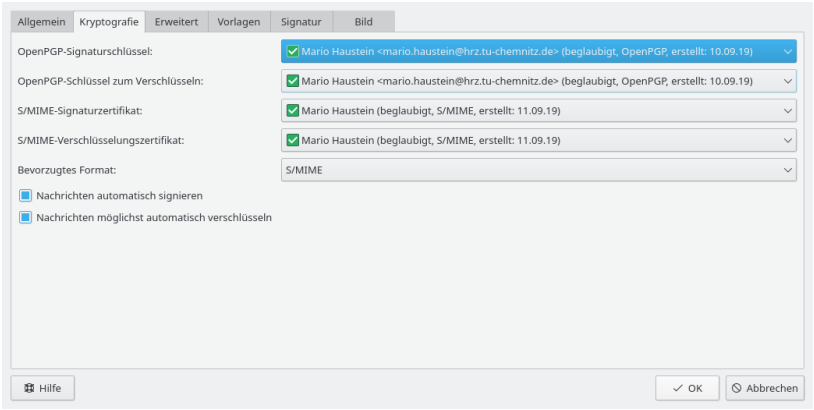

Schlüssel mit Kleopatra unterschreiben



Schlüssel mit Kleopatra unterschreiben



Die Einrichtung in KMail ist einfach.



Die Benutzung in KMail ist einfach.

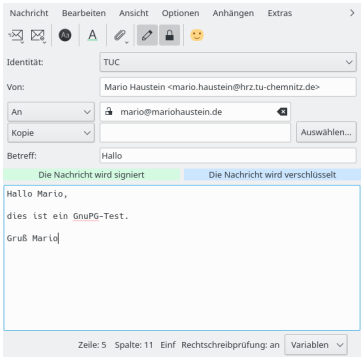


Abbildung: Sender

Abbildung: Empfänger

Die Benutzung in KMail ist einfach.

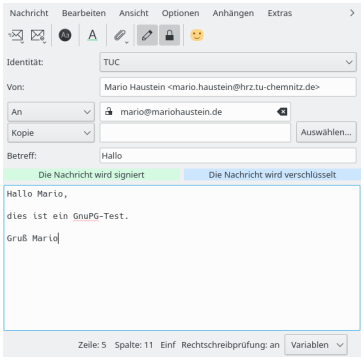


Abbildung: Sender

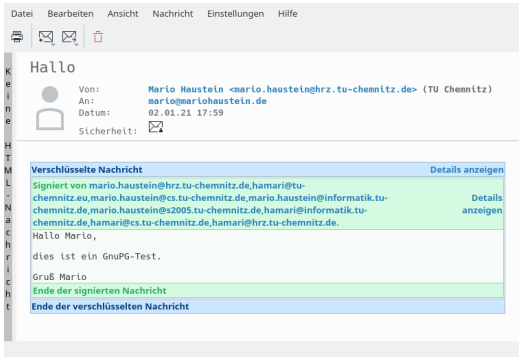


Abbildung: Empfänger

Verschlüsseln

- Über Standardein- und ausgabe.

```
$ gpg -a -r "Otto Normalverbraucher" --encrypt << EOF
> Hallo Welt
> EOF
-----BEGIN PGP MESSAGE-----
hQEMAyZv5bexFRRyAQf+Oc+BARrKyAEx1W5WO+Md/v6tQFxymbNWfYr+XpeToiB1
...
sc/g6wE9gQ==
=MwgC
-----END PGP MESSAGE-----
```

- Datei verschlüsseln.

```
$ echo "Hallo Welt" > geheim.txt
$ gpg -r "Otto Normalverbraucher" --encrypt geheim.txt
$ ls -l geheim.txt*
-rw-r--r-- 1 otto otto  11  1. Jan 23:42 geheim.txt
-rw-r--r-- 1 otto otto 353  1. Jan 23:42 geheim.txt.gpg
```


Weitere Verschlüsselungsoptionen

- `-R` → Versteckte Empfänger.
  - Jeder Empfänger kann die IDs aller anderen Empfängerschlüssel auslesen.
  - Für verdeckte Empfänger wird keine Schlüssel-ID hinterlegt.
  - Verhindert Metadatenanalyse.
  - Vermutungen lassen sich aber leicht durch andere berechtigte Empfänger überprüfen.
- `-f` bzw. `-F` → Empfänger zeilenweise aus Datei lesen.
- `--multifile` → mehrere Dateien in einem Rutsch verschlüsseln.

Entschlüsseln

- Über Standardein- und ausgabe.

```
$ gpg --decrypt << EOF
> -----BEGIN PGP MESSAGE-----
> hQEMAyZv5bexFRRyAQf+Oc+BARrKyAEx1W5W0+Md/v6tQFxymbNWfYr+XpeToiB1
> ...
> sc/g6wE9gQ==
> =MwgC
> -----END PGP MESSAGE-----
> EOF
gpg: verschlüsselt mit 2048-Bit RSA Schlüssel, ID 266FE5B7B1151472, erzeugt 2020-12-13
      "Otto Normalverbraucher"
Hallo Welt
```

- Datei entschlüsseln.

```
$ gpg -o geheim.txt --decrypt geheim.txt.gpg
gpg: verschlüsselt mit 2048-Bit RSA Schlüssel, ID 266FE5B7B1151472, erzeugt 2020-12-13
      "Otto Normalverbraucher"
```


Unterschreiben I

- Vollständige Signatur enthält Klartext **und** Signatur (Dateiendung `.gpg`).
  - Nur die Ausgabedatei muss übermittelt werden.

```
$ gpg --sign vertrag.txt
$ ls -l vertrag.txt*
-rw-r--r-- 1 otto otto  11  1. Jan 23:42 vertrag.txt
-rw-r--r-- 1 otto otto 362  1. Jan 23:42 vertrag.txt.gpg
```

- Abgesetzte Signatur enthält **nur** die Signatur (Dateiendung `.sig`).
  - Ein- und Ausgabedatei müssen übermittelt werden.

```
$ gpg --detach-sign vertrag.txt
$ ls -l vertrag.txt*
-rw-r--r-- 1 otto otto  11  1. Jan 23:42 vertrag.txt
-rw-r--r-- 1 otto otto 310  1. Jan 23:42 vertrag.txt.sig
```


Unterschreiben II

- Klartextsignatur enthält Klartext **und** Signatur (Dateiendung `.asc`).
  - Datei ist menschenlesbar.

```
$ gpg --clear-sign vertrag.txt
$ cat vertrag.txt.asc
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

Hallo Welt
-----BEGIN PGP SIGNATURE-----
iQEzBAEBCAAdFiEEM79KveJ38xkxYGHTf5xr1AnU8TMFAl/vU2cACgkQf5xr1AnU
...
63NaSMJM52B3qymlEC0ZPPLJolR7Fw==
=Kul+
-----END PGP SIGNATURE-----
```


Signaturoptionen

- `--sign` kann gemeinsam mit `--encrypt` bzw. `--symmetric` verwendet werden.
  - `--decrypt` kümmert sich dann auch um die Signaturprüfung.
- `-u` → Unterzeichner
  - Vom Standardschlüssel abweichenden geheimen Schlüssel verwenden.
  - Es können auch mehrere geheime Schlüssel unterschreiben.

Unterschrift prüfen I

- Vollständige Signatur prüfen.

```
$ gpg --verify vertrag.txt.gpg
gpg: Signatur vom Fr 01 Jan 2021 23:42:00 CET
gpg:                mittels RSA-Schlüssel 33BF4ABDE277F319316061D37F9C6BD409D4F133
gpg: Korrekte Signatur von "Otto Normalverbraucher" [ultimativ]
```

- Vollständige Signatur prüfen und Ergebnis extrahieren.

```
$ gpg -o vertrag.txt --verify vertrag.txt.gpg
...
```


Unterschrift prüfen II

- Abgesetzte Signatur prüfen.

```
$ gpg --verify vertrag.txt.sig vertrag.txt
gpg: Signatur vom Fr 01 Jan 2021 23:42:00 CET
gpg:                mittels RSA-Schlüssel 33BF4ABDE277F319316061D37F9C6BD409D4F133
gpg: Korrekte Signatur von "Otto Normalverbraucher" [ultimativ]
```

- Abgesetzte Signatur über Standardeingabe prüfen.

```
$ gpg --verify vertrag.txt.sig - < vertrag.txt
...
```

- Abgesetzte Signatur prüfen (ohne Angabe des Klartextes).

```
$ gpg --verify vertrag.txt.sig
gpg: die unterzeichneten Daten sind wohl in 'vertrag.txt'
...
```


Unterschrift prüfen III

- Klartextsignatur prüfen.

```
$ gpg --verify vertrag.txt.asc
gpg: Signatur vom Fr 01 Jan 2021 23:42:00 CET
gpg:                mittels RSA-Schlüssel 33BF4ABDE277F319316061D37F9C6BD409D4F133
gpg: Korrekte Signatur von "Otto Normalverbraucher" [ultimativ]
```


Stolperfallen bei der Unterschriftsprüfung

- Nicht ausschließlich auf Rückgabe-Code von gpg verlassen!
- In Skripten auch die Identität des **Unterzeichners** verarbeiten.
- Vollständige Signaturen bevorzugen.
- In Skripten die Ausgabedatei schreiben und weiterverwenden.
- Bei abgesetzten Signaturen immer den Dateinamen oder – angeben.
- Klartextsignaturen vermeiden.

⇒ Daten außerhalb des PGP-Blocks sind nicht Bestandteil der Signatur, könnten aber vom Anwender so interpretiert werden.

- Das Tool gpgv ist speziell für Unterschriftsprüfungen gedacht.

Schlüsselaustausch

- Der Schlüsselaustausch mittels `--export` und `--import` ist umständlich-
- Es gibt Verbund öffentlicher Schlüsselserver.

- Jeder kann Schlüssel hochladen.
- Jeder kann Schlüssel durchsuchen.
- Es wird nicht gelöscht (das ist ein Sicherheitsfeature).
  - Es gibt Schlüsselserver, die das Löschen unterstützen.

- Seit 2019 gibt es Änderungen ...
  - Sog. SKS-Server gehen außer Betrieb.
  - Einige Server erfordern Bestätigung der E-Mail-Adressen.
  - Es werden nur noch Schlüssel, aber keine Schlüsselsignaturen verteilt.

  ⇒ Verhinderung von Signatur-Spam.

Schlüsselsuche

- Schlüsselserver in `/etc/gnupg/gpg.conf` bzw. `~/.gnupg/gpg.conf` konfigurieren.

```
keyserver hkps://keys.openpgp.org
```

- Schlüssel suchen.

```
$ gpg --search-key mario.haustein@hrz.tu-chemnitz.de
gpg: data source: https://keys.openpgp.org:443
(1)     Mario Haustein <hamari@hrz.tu-chemnitz.de>
        Mario Haustein <hamari@informatik.tu-chemnitz.de>
        Mario Haustein <mario.haustein@hrz.tu-chemnitz.de>
        Mario Haustein <mario.haustein@informatik.tu-chemnitz.de>
        Mario Haustein <mario.haustein@s2005.tu-chemnitz.de>
          4096 bit RSA key AEB326F2C20A11FA, erzeugt: 2019-09-10
```


Schlüssel veröffentlichen und abrufen.

- Veröffentlichen bzw. aktualisierte Version hochladen.
- Bei der Erstveröffentlichung fordert der Schlüsselserver eine Bestätigung per E-Mail an.

```
$ gpg --send-key D1D5B143F82D129E1172AADDAEB326F2C20A11FA
gpg: sende Schlüssel AEB326F2C20A11FA auf hkps://keys.openpgp.org
```

- Abrufen bzw. aktuelle Version herunterladen.

```
$ gpg --recv-key D1D5B143F82D129E1172AADDAEB326F2C20A11FA
gpg: Schlüssel AEB326F2C20A11FA: Öffentlicher Schlüssel
     "Mario Haustein <mario.haustein@hrz.tu-chemnitz.de>" importiert
gpg: Anzahl insgesamt bearbeiteter Schlüssel: 1
gpg:                importiert: 1
```

- Alle Schlüssel aktualisieren.

```
$ gpg --refresh-keys
...
```


WKD / WKS: Web Key Directery / Service[1]

- Aus einem BSI-Projekt zur Förderung von Kryptografie hervorgegangen.
- Öffentliche Schlüssel werden per HTTPS vom Web-Server des E-Mail-Anbieters abgerufen.
- Ableitung der URL aus der E-Mail-Adresse, z. B.
  - `mario@mariohaustein.de`

  ⇒ `https://openpgpgkey.mariohaustein.de/.well-known/openpgpkey/mariohaustein.de/hu/izpwqke6hfw9gcryduuukef3phzk6e8k?l=mario`
- U. a. umgesetzt durch ...
  - debian.org
  - f-droid.org
  - gentoo.org
  - gnupg.org
  - kernel.org
  - mailbox.org
  - Posteo
  - Protonmail
- Schlüssel wird automatisch heruntergeladen.
- Manuelle Abfrage mit:

  ```
  $ gpg --locate-external-key mario@mariohaustein.de
  ```

[1] Entwurf: `https://tools.ietf.org/html/draft-koch-openpgp-webkey-service-11` (Stand November 2020)

Wenn etwas schief geht . . .

- Ist der Schlüssel nicht mehr sicher oder unbrauchbar ⇒ **öffentlicher Widerruf**
- Bei Schlüsselerzeugung automatisch wird automatisch unter `~/.gnupg/openpgp-revocs.d` ein Widerrufszertifikat erzeugt.

```
$ gpg -o revoke.asc --gen-revoke 33BF4ABDE277F319316061D37F9C6BD409D4F133

sec  rsa2048/7F9C6BD409D4F133 2020-12-13 Otto Normalverbraucher

Ein Widerrufszertifikat für diesen Schlüssel erzeugen? (j/N) j
Grund für den Widerruf:
  0 = Kein Grund angegeben
  1 = Hinweis: Dieser Schlüssel ist nicht mehr sicher
  2 = Schlüssel ist überholt
  3 = Schlüssel wird nicht mehr benutzt
  Q = Abbruch
(Wahrscheinlich möchten Sie hier 1 auswählen)
Ihre Auswahl? 1

$ gpg --import revoke.asc
$ gpg --send-key 33BF4ABDE277F319316061D37F9C6BD409D4F133
```


Wann ist ein Schlüssel gültig?

Die einfache (aber nicht ganz korrekte) Antwort

1. Fingerabdruck prüfen.
2. Schlüssel unterschreiben.

Begriffe: Vertrauen vs. Gültigkeit?

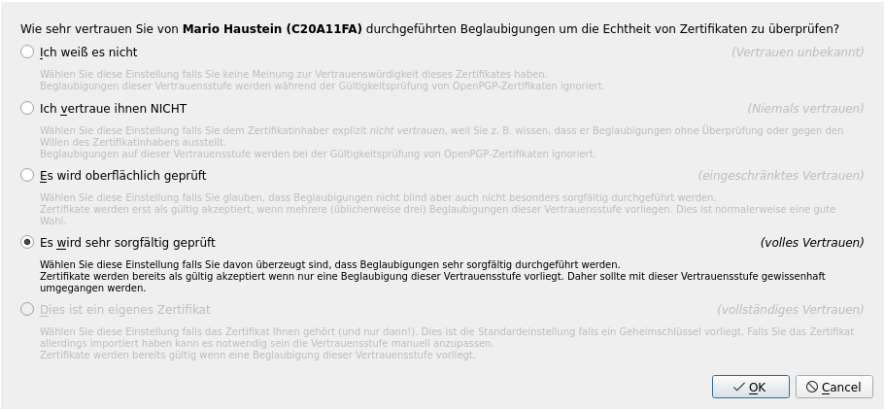
Vertrauen Unsere **persönliche** Überzeugung, wie gründlich ein anderer Teilnehmer bei der Schlüsselüberprüfung ist.

- ultimativ Wir verfügen über den geheimen Schlüssel
- vollständig Der Inhaber prüft gründlich
- marginal Der Inhaber prüft oberflächlich
- niemals Dem Inhaber nicht vertrauen
- . . .

Gültigkeit Ist das Vertrauen ausreichend um den Schlüssel zu nutzen?

- ultimativ Wir verfügen über den geheimen Schlüssel
- vollständig Wir trauen dem Schlüssel
- marginal Vertrauen nicht ausreichend
- . . .

Vertrauen in einen Schlüssel mit Kleopatra festlegen

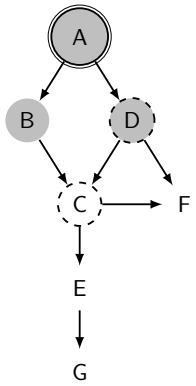


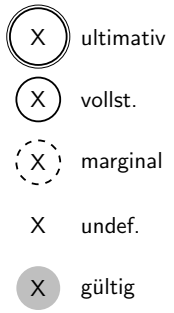
Web of Trust: Vom Vertrauen zur Gültigkeit

- Beispiel:[2] ein Schlüssel ist gültig, wenn

1. er durch **uns** unterschrieben wurde oder
2. er durch **einen** gültigen Schlüssel unterschrieben wurde, dem wir **vollständig** vertrauen oder
3. er durch mind. **zwei** gültige Schlüssel unterschrieben wurde, denen wir **marginal** vertrauen

- Bei 2. und 3. darf die Unterschriftskette von unserem Schlüssel aus höchstens **4 Schritte** lang sein.





[2] Quelle: `https://www.gnupg.org/gph/en/manual/x334.html`

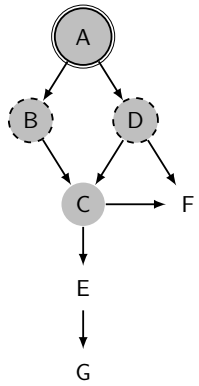
OpenPGP Web of Trust

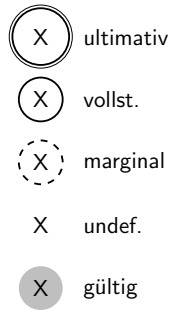
Web of Trust: Vom Vertrauen zur Gültigkeit

- Beispiel:[2] ein Schlüssel ist gültig, wenn

1. er durch **uns** unterschrieben wurde oder
2. er durch **einen** gültigen Schlüssel unterschrieben wurde, dem wir **vollständig** vertrauen oder
3. er durch mind. **zwei** gültige Schlüssel unterschrieben wurde, denen wir **marginal** vertrauen

- Bei 2. und 3. darf die Unterschriftskette von unserem Schlüssel aus höchstens **4 Schritte** lang sein.





[2] Quelle: `https://www.gnupg.org/gph/en/manual/x334.html`

Mario Haustein Einführung in GnuPG 8. Januar 2021 44 / 64

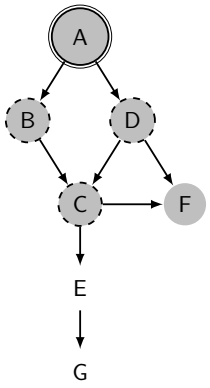
OpenPGP Web of Trust

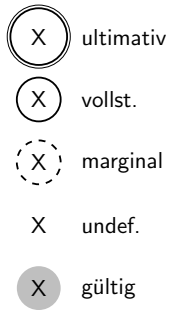
Web of Trust: Vom Vertrauen zur Gültigkeit

- Beispiel:[2] ein Schlüssel ist gültig, wenn

1. er durch **uns** unterschrieben wurde oder
2. er durch **einen** gültigen Schlüssel unterschrieben wurde, dem wir **vollständig** vertrauen oder
3. er durch mind. **zwei** gültige Schlüssel unterschrieben wurde, denen wir **marginal** vertrauen

- Bei 2. und 3. darf die Unterschriftskette von unserem Schlüssel aus höchstens **4 Schritte** lang sein.





[2] Quelle: `https://www.gnupg.org/gph/en/manual/x334.html`

Mario Haustein Einführung in GnuPG 8. Januar 2021 44 / 64

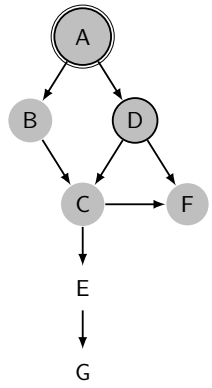
OpenPGP Web of Trust

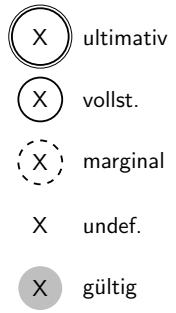
Web of Trust: Vom Vertrauen zur Gültigkeit

- Beispiel:[2] ein Schlüssel ist gültig, wenn

1. er durch **uns** unterschrieben wurde oder
2. er durch **einen** gültigen Schlüssel unterschrieben wurde, dem wir **vollständig** vertrauen oder
3. er durch mind. **zwei** gültige Schlüssel unterschrieben wurde, denen wir **marginal** vertrauen

- Bei 2. und 3. darf die Unterschriftskette von unserem Schlüssel aus höchstens **4 Schritte** lang sein.





[2] Quelle: `https://www.gnupg.org/gph/en/manual/x334.html`

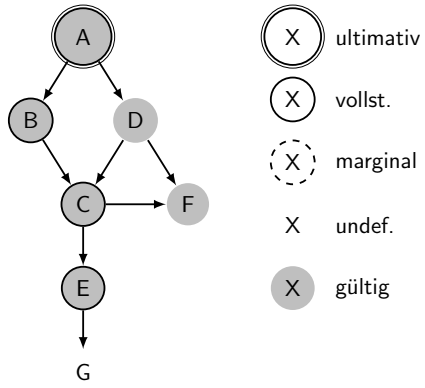
Mario Haustein Einführung in GnuPG 8. Januar 2021 44 / 64

Web of Trust: Vom Vertrauen zur Gültigkeit

- Beispiel:[2] ein Schlüssel ist gültig, wenn

1. er durch **uns** unterschrieben wurde oder
2. er durch **einen** gültigen Schlüssel unterschrieben wurde, dem wir **vollständig** vertrauen oder
3. er durch mind. **zwei** gültige Schlüssel unterschrieben wurde, denen wir **marginal** vertrauen

- Bei 2. und 3. darf die Unterschriftskette von unserem Schlüssel aus höchstens **4 Schritte** lang sein.



[2] Quelle: `https://www.gnupg.org/gph/en/manual/x334.html`

Vertrauensmodelle

- Nutzer entscheidet, welches Vertrauensmodell zur Anwendung kommt.
- Klassisches Vertrauensmodell
  - mind. 1× vollständiges Vertrauen
  - mind. 3× marginales Vertrauen
  - max. 5 Schritte
- PGP 5 Modell
  - Schlüsselunterschriften können um die Vertrauenseinschätzung ergänzt werden.
  - Das fremde Vertrauen wird bei der Gültigkeitsprüfung berücksichtigt.
- Trust On First Use (TOFU)
  - Schlüssel einer unbekannten UID wird als vertrauenswürdig betrachtet.
  - Sobald ein neuer Schlüssel für diese UID auftaucht ⇒ beide Schlüssel zweifelhaft
  - Gegenseitige Signatur beider Schlüssel ⇒ keine Zweifel

Status des Web of Trust

Sommer 2019

- Schlüsselserver werden mit Unmengen sinnloser Signaturen überladen.

⇒ Signaturprüfung zu rechenaufwändig für Klienten.

⇒ prinzipielles Problem, keine Lösung

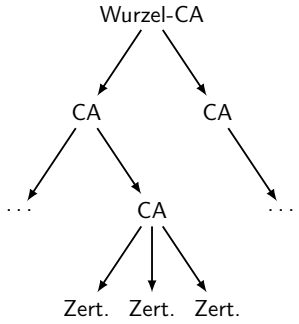
- Schlüsselserver verteilen keine Schlüsselsignaturen mehr.
- GnuPG importiert keine Schlüsselsignaturen von Schlüsselservern mehr.

⇒ Web of Trust ist de facto gescheitert.

⇒ Alternative: TOFU bzw. individuelle Prüfung der Schlüssel.

- Letztendlich war das bereits im Vorfeld die Praxis.

X.509-Zertifikate



- Hierarchisches Vertrauensmodell
- Ein Zertifikat belegt, dass ein Schlüsselpaar einem bestimmten Inhaber gehört.
- Zertifikate werden von Zertifizierungsstellen (CAs) ausgestellt.
- Diese prüft, dass der Inhaber tatsächlich Eigentümer des Schlüssels ist.
- CAs können durch übergeordnete CAs beglaubigt werden.
- Vertrauensanker ist die Wurzel-CA.

Vergleich OpenPGP vs. X.509

Eigenschaft	OpenPGP	X.509
Organisation	dezentral	hierarchisch
Kosten	kostenlos	i. d. R. laufende Kosten, z. T. kostenlos
Einrichtungsaufwand	niedriger	höher
Vertrauen	individuell ⇒ flexibler ⇒ aufwändiger	gesamte CA ⇒ alles oder nichts ⇒ einfacher
typ. Einsatzgebiet	Privatperson, Community	Unternehmen
Reichweite	beliebig	dort wo CA etabliert
Schlüsselverteilung	Verbundnetzwerk verteilter Schlüsselserver, einheitliche API	individuelle LDAP-Server pro CA bzw. Einrichtung

Ausstellung von X.509-Zertifikaten

1. Inhaber erzeugt ein Schlüsselpaar
2. Zu diesem Schlüsselpaar wird Zertifikatsantrag (CSR) erstellt.
   - enthält Name, E-Mail-Adresse, ...
3. Der CSR wird bei einer CA eingereicht.
4. Die CA prüft die Angaben.
   - z. B. durch Kontrolle von Ausweisen oder Registereinträgen
5. Die CA stellt ein Zertifikat aus und schickt es an den Schlüsselinhaber.
   Zudem stellt die CA ihre Zertifikatskette zur Wurzel-CA bereit.
6. Der Schlüsselinhaber verteilt das Zertifikat (ggf. zzgl. Kette) an die Kommunikationspartner.

Schlüsselerzeugung

```
$ gpgsm -o otto.csr --generate-key
```

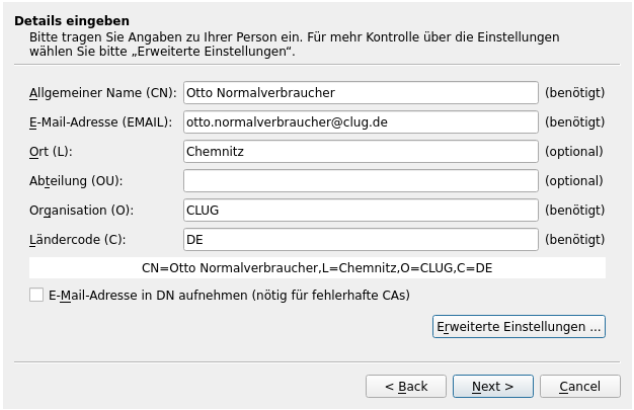
- Es folgt ein interaktiver Dialog. Folgende Eingaben sind relevant.
  - X.509-Subjekt: `CN=Otto Normalverbraucher,O=CLUG,L=Chemnitz,ST=Sachsen,C=DE`
  - E-Mail: `otto.normalverbraucher@clug.de`
- Es entsteht eine Datei `otto.csr`, die wir bei der CA einreichen.
- Nach der Prüfung durch die CA erhalten wir von dort das Zertifikat `otto.pem`

```
$ gpgsm --import otto.pem
```

- Sofern nicht bereits vorhanden, importierten wir noch die Zertifikatskette der CA (`chain.pem`).

```
$ gpgsm --import chain.pem
```


Schlüsselerzeugung mit Kleopatra

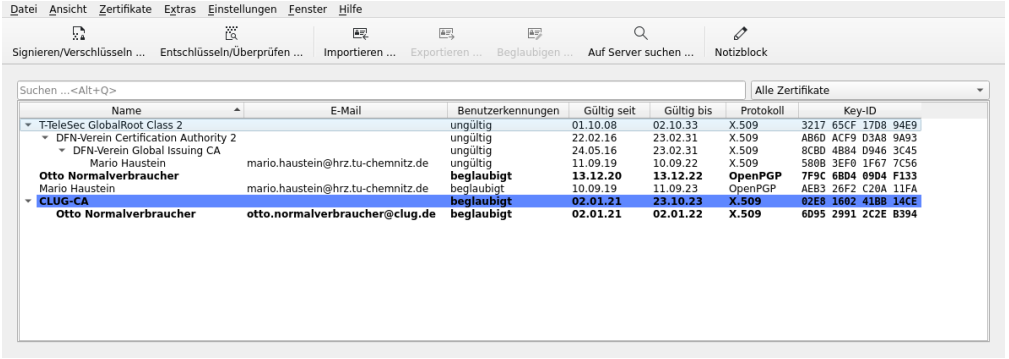


Schlüsselverwaltung

- Analog zu gpg: `-k` → öffentliche Schlüssel, `-K` → geheime Schlüssel

```
$ gpgsm -k
------------------------------
           ID: 0x2C2EB394
          S/N: 65D274BBB89ECA114C3BEB1016239B9150F940A6
       Issuer: /CN=CLUG-CA/O=CLUG/L=Chemnitz/ST=Sachsen/C=DE
      Subject: /CN=Otto Normalverbraucher/O=CLUG/ST=Sachsen/C=DE
          aka: otto.normalverbraucher@clug.de
     validity: 2021-01-01 23:42:00 through 2022-01-01 23:42:00
     key type: 3072 bit RSA
    key usage: digitalSignature nonRepudiation keyEncipherment dataEncipherment
  fingerprint: 86:77:1F:63:2B:22:B4:E2:EF:07:E9:5B:6D:95:29:91:2C:2E:B3:94

           ID: 0x41BB14CE
          S/N: 6E076018E180BED215A0B64EC58497A2FE4B7182
       Issuer: /CN=CLUG-CA/O=CLUG/L=Chemnitz/ST=Sachsen/C=DE
      Subject: /CN=CLUG-CA/O=CLUG/L=Chemnitz/ST=Sachsen/C=DE
     validity: 2021-01-01 23:42:00 through 2023-10-23 24:42:00
     key type: 4096 bit RSA
 chain length: unlimited
  fingerprint: AE:76:B6:67:E4:C8:63:15:09:09:58:7D:02:E8:16:02:41:BB:14:CE
```


Schlüsselliste in Kleopatra



Versuchen wir eine Botschaft zu verschlüsseln

```
$ gpgsm -r otto.normalverbraucher@clug.de -o geheimtext.p7m --encrypt klartext.txt
gpgsm: Das Wurzelzertifikat ist nicht als vertrauenswürdig markiert
gpgsm: Fingerprint=AE:76:B6:67:E4:C8:63:15:09:09:58:7D:02:E8:16:02:41:BB:14:CE
...
gpgsm: after checking the fingerprint, you may want to add it manually to the list of
     trusted certificates.
gpgsm: Interaktives vertrauenswürdig-Markieren ist in dieser Sitzung ausgeschaltet
gpgsm: can't encrypt to 'otto.normalverbraucher@clug.de': Nicht vertrauenswürdig
```

⇒ GnuPG beschützt uns vor unserer eigenen Fahrlässigkeit.

- In grafischen Umgebungen erscheint ein Bestätigungsdialog.
- Sonst Wurzelzertifikat in `~/.gnupg/trustlist.txt` eintragen.

`gpgsm` lässt sich wie gpg benutzen.

- Verschlüsseln

  ```
  $ gpgsm -r otto.normalverbraucher@clug.de -o geheimtext.p7m --encrypt text.txt
  ```

- Entschlüsseln

  ```
  $ gpgsm -o klartext.txt --decrypt geheimtext.p7m
  ```

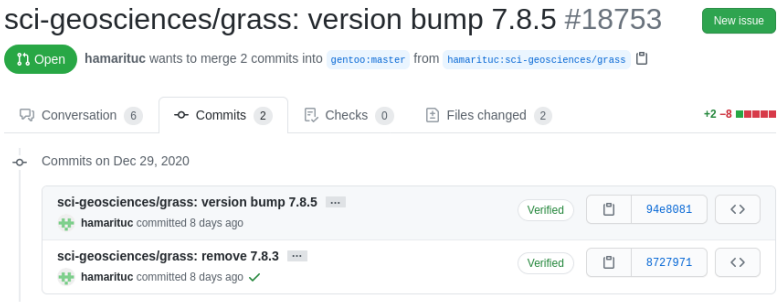
- Unterschreiben

  ```
  $ gpgsm -u otto.normalverbraucher@clug.de -o signatur.p7s --sign klartext.txt
  ```

- Prüfen

  ```
  $ gpgsm -o klartext.txt --verify signatur.p7s
  ```

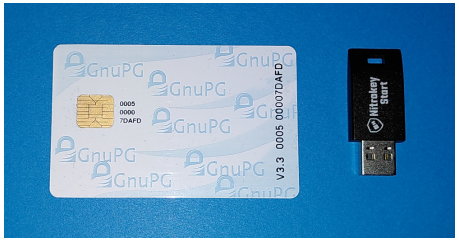
Git-Commits signieren



- Alle Commits sind vom selben Autor.
- Verbindliche Anerkennung der Open-Source-Lizenz.

Smartcards

- Kryptografisches Hardwaremodul übernimmt Entschlüsselung bzw. Signatur.
- Geheimer Schlüssel lässt sich konstruktionsbedingt nicht auslesen.

00010110 →  → `Hallo Welt`

GnuPG und OpenSSH

- GnuPG-Schlüssel lassen sich als SSH-Schlüssel verwenden.
- SSH-Schlüssel lassen sich in GnuPG verwalten.
- `gpg-agent` kann `ssh-agent` ersetzen.

⇒ `gpg-agent` fungiert als „Passwortmanager“ für SSH-Schlüssel.

- SSH-Anmeldung ist mit Smartcards möglich.

PKCS#11

- GnuPG ist ein PKCS#11-Provider

⇒ Nutzung von OpenPGP-Smartcards für
- Firefox (Webseiten-Anmeldung)
- Thunderbird (Verschlüsselung und Signatur von Mails)
- LibreOffice (Dokumenten-Signatur)
- OpenVPN (VPN-Anmeldung)
- Schlüsselspeicher für Server-Zertifikate

Literatur, Links I

- GnuPG Handbuch
  https://gnupg.org/documentation/manuals/gnupg.pdf,
  https://gnupg.org/documentation/manuals/gnupg/
  Unbedingt konsultieren, bevor man lange herumprobiert
- GPGME – GnuPG Made Easy
  https://gnupg.org/documentation/manuals/gpgme.pdf,
  https://gnupg.org/documentation/manuals/gpgme/
  Bibliothek zur Benutzung von GnuPG aus anderen Programmen heraus
- GPGP4Win
  https://www.gpg4win.de/,
  https://www.bsi.bund.de/DE/Themen/Kryptografie_Kryptotechnologie/Kryptotechnologie/Gpg4win/gpg4win_node.html
  GnuPG-Suite für Windows, inkl. Einbindung in Explorer und Outlook

Literatur, Links II

- Zulassung von GnuPG als Kryptoprodukt
  https://www.bsi.bund.de/DE/Themen/Sicherheitsberatung/ZugelasseneProdukte/Liste_Produkte/Liste_Produkte_node.html,
  https://gnupg.com/20200107-freigabe-vs-nfd.html
- Hintergrundinfos zu WKD / WKS
  https://www.bsi.bund.de/DE/Themen/Kryptografie_Kryptotechnologie/Kryptografie/EasyGPG/EasyGPG_node.html,
  https://www.bsi-fuer-buerger.de/BSIFB/DE/Empfehlungen/Verschluesselung/EMail_Verschluesselung/EasyGPG/EMail_EasyGPG_node.html