SSH-Anmeldung mit GnuPG und SmartCards

Mario Haustein

Chemnitzer Linux-Tage 2025

Ziele

Wir werden heute ...

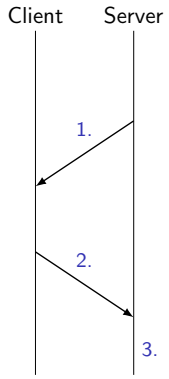
- behandeln, wie Schlüssel-Authentifizierung in SSH funktioniert
- feststellen, dass es unklug ist, Schlüssel ungeschützt als Datei herum liegen zu lassen
- stattdessen GnuPG als Speicher für unsere SSH-Schlüssel verwenden
- unsere SSH-Schlüssel auf einer Smartcard speichern

Es war einmal ein Passwort ...

1. Client prüft Identität des Servers per Host-Key.
2. Client sendet Passwort
3. Server prüft Passwort.

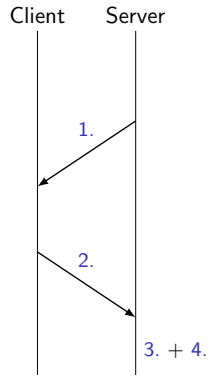
Don't do that!

- Schwache Passworte können durch Ausprobieren ermittelt werden.
- Kompromittierte Server können unser Passwort ausleiten.



SSH-Public-Key-Authentifizierung

1. Client prüft Identität des Servers per Host-Key.
2. Client signiert Session-ID und weitere Authentifizierungdaten mit geheimen Schlüssel `~/.ssh/id_rsa`.
3. Server prüft, ob öffentlicher Schlüssel in `~/.ssh/authorized_keys` eingetragen ist.
4. Server prüft Signatur.



Schon besser

- Wir geben ggü. dem Server kein Geheimnis preis.
- ABER: `~/.ssh/id_rsa` ohne Passwortschutz ist gefährlich.
- Besser: Geheimen Schlüssel durch Passwort verschlüsseln.

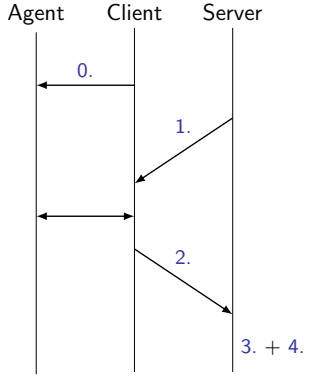
⇒ `$ ssh-keygen -p -f ~/.ssh/id_rsa`

Funktionsweise des SSH-Agents

- Wiederholte Passworteingabe umständlich
- Schlüssel durch SSH-Agent im RAM vorhalten

0. Agent starten und geheimen Schlüssel laden
   - Ggf. Abfrage und Eingabe des Passworts der Schlüsseldatei

1.–4. wie gehabt, aber Agent übernimmt die Signierung



Funktionsweise des (Open)SSH-Agents

- OpenSSH-Agent starten

```
$ eval $(ssh-agent)
Agent pid 28953
$ export | grep SSH_
declare -x SSH_AGENT_PID="28953"
declare -x SSH_AUTH_SOCK="/tmp/ssh-XXXXUKyWh5/agent.28952"
```

- Geheimen SSH-Schlüssel laden

```
$ ssh-add ~/.ssh/id_rsa
Enter passphrase for /home/demo/.ssh/id_rsa:
Identity added: /home/demo/.ssh/id_rsa (/home/demo/.ssh/id_rsa)
```

- Geladene SSH-Schlüssel anzeigen

```
$ ssh-add -l
3072 SHA256:yT/Y2OUTOCY+Xc2sB/aKWtwWeXkWjEzYlo3YeVpPooQ /home/demo/.ssh/id_rsa (RSA)
```


Sicherheitsbeurteilung

- Kein Agent
  - Schlüssel u. U. nicht passwortgeschützt ⇒ unsicher
  - Passworteingabe bei jeder Benutzung ⇒ Akzeptanzproblem
  - Bei verschachtelten SSH-Verbindungen muss geheimer Schlüssel bei jedem Zwischenschritt vorliegen ⇒ u. U. unerwünscht
- OpenSSH-Agent
  - Schlüssel werden standardmäßig unbegrenzt lang vorgehalten ⇒ Organisationsproblem
  - Lebenszeit nur per Kommandozeilenparameter reduzierbar, nicht per Konfigurationsdatei
  - Agent läuft unbeabsichtigt weiter ⇒ Organisationsproblem

⇒ Mittelweg benötigt

Der GnuPG SSH-Agent

- SSH-Agent-Unterstützung in `~/.gnupg/gpg-agent.conf` bzw. `/etc/gnupg/gpg-agent.conf` aktivieren.

```
enable-ssh-support
```

- Agent ggf. starten

```
$ gpg-connect-agent /bye
gpg-connect-agent: Kein aktiver gpg-agent - '/usr/bin/gpg-agent' wird gestartet
gpg-connect-agent: Warte bis der agent bereit ist ... (5s)
gpg-connect-agent: Verbindung zum agent aufgebaut
```

- Agent-Socket bekannt machen

```
$ export SSH_AUTH_SOCK=$(gpgconf --list-dirs agent-ssh-socket)
$ export | grep SSH_
declare -x SSH_AUTH_SOCK="/home/demo/.gnupg/S.gpg-agent.ssh"
```


Den GnuPG-Agent starten

1. Manuell wie bereits gesehen
2. Automatisch per `~/.bashrc` o. ä.

```
# Agent bei Bedarf starten
gpg-connect-agent -q /bye

# Agent-Socket bestimmen
unset SSH_AUTH_SOCK
if [ "${gnupg_SSH_AUTH_SOCK_by:-0}" -ne $$ ]; then
  export SSH_AUTH_SOCK="$(gpgconf --list-dirs agent-ssh-socket)"
fi
```

3. Automatisch per systemd-Socket-Activation (z. B. Debian)
   - systemd legt Socket an.
   - Beim ersten Socket-Zugriff startet systemd den Prozess.

- Unbedingt das Distro-Handbuch beachten!

Schlüsselerzeugung

- Der Schlüssel liegt nicht in `~/.ssh` sondern im GnuPG-Schlüsselbund.
- Der Verwendungszweck des Schlüssel muss „authentifizieren“ erlauben.
- interaktiv

```
$ gpg --expert --full-generate-key
```

- kurz und bündig

```
$ gpg --batch --generate-key << EOF
Key-Type: RSA
Key-Length: 3072
Key-Usage: auth
Name-Real: Mario Haustein
Name-Comment: SSH
Expire-Date: 0
EOF
```


Schlüssel zur Verwendung im Agent eintragen

- Schlüssel anzeigen

```
$ gpg --with-keygrip -K
/home/demo/.gnupg/pubring.kbx
-----------------------------
sec   rsa3072 2025-03-14 [CA]❶
      AFB803D27B0F5110D1004660208B30E174DECB24
      Keygrip = 211B2DEF0FC050A80B3E097DBF9F7F51E73CED5D❷
uid         [ ultimativ ] Mario Haustein (SSH)
```

- Verwendungszweck ❶: `A` ⇒ authentifizieren
- Keygrip ❷: in `~/.gnupg/sshcontrol` eintragen.

```
211B2DEF0FC050A80B3E097DBF9F7F51E73CED5D
```

- Schlüssel ist verfügbar.

```
$ ssh-add -l
3072 SHA256:wCu2d549qj3G1D10ZM6iMnErc5b9IgXmZl8zflPphC0 (none) (RSA)
```

Öffentlichen Schlüssel für den Server exportieren

- Öffentlichen Schlüssel exportieren

```
$ gpg --export-ssh-key AFB803D27B0F5110D1004660208B30E174DECB24
ssh-rsa AAAAB3NzaC [...] aXAcGvj+M= openpgp:0x74DECB24
```

- Auf dem Server in `~/.ssh/authorized_keys` eintragen.

Benutzung

Es ist ganz einfach

$ ssh ⟨*User*⟩@⟨*Server*⟩

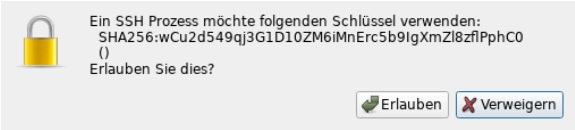
- Bei der ersten Benutzung eines Schlüssels wird das Passwort abgefragt.
- Erneute Passwortabfrage, erst ...
  - 30 Minuten nach letzter Schlüsselnutzung
  - 2 Stunden nach letzter Passworteingabe

Bestätigungsfreigabe

- Trägt man in `~/.gnupg/sshcontrol` das `confirm`-Flag ein ...

```
211B2DEF0FC050A80B3E097DBF9F7F51E73CED5D 0 confirm
```

- ... muss jede Verwendung des Schlüssels bestätigt werden.



Agent-Forwarding

- Den Schlüssel kann man nur auf dem Client verwenden

```
user@client ~ $ ssh-add -l
3072 SHA256:wCu2d549qj3G1D10ZM6iMnErc5b9IgXmZl8zflPphC0 (none) (RSA)
user@client ~ $ ssh server
user@server ~ $ ssh-add -l
Could not open a connection to your authentication agent.
```

- Per Agent-Forwarding, kann man den Schlüssel des Clients auch auf dem Server verwenden.

```
user@client ~ $ ssh-add -l
3072 SHA256:wCu2d549qj3G1D10ZM6iMnErc5b9IgXmZl8zflPphC0 (none) (RSA)
user@client ~ $ ssh -A server
user@server ~ $ ssh-add -l
3072 SHA256:wCu2d549qj3G1D10ZM6iMnErc5b9IgXmZl8zflPphC0 (none) (RSA)
user@server ~ $ ssh server2
```

- Agent-Forwarding sollte nur zu vertrauenswürdigen Servern erfolgen.

Smartcards

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

Klartext →  → Signatur

Es wird unübersichtlich

- Smartcard ≠ Smartcard

⇒ Verschiedene Technologien und Protokolle

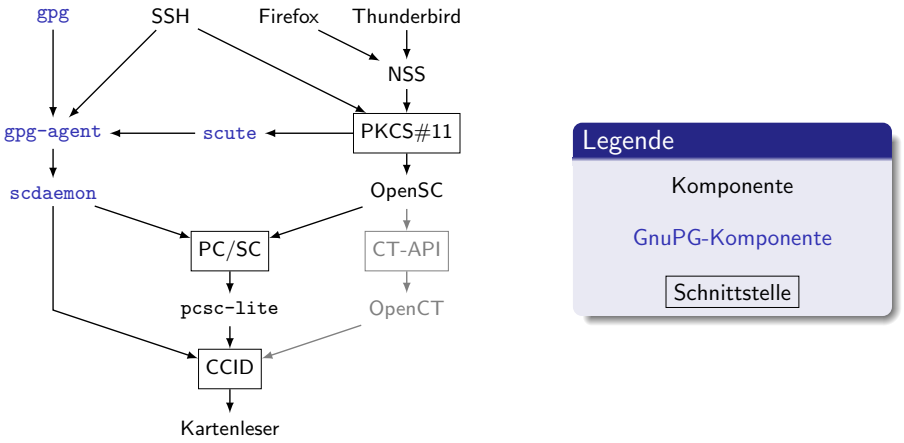
- Wir beschäftigen uns mit der OpenPGP-Card

- Es gibt viele Mitspieler

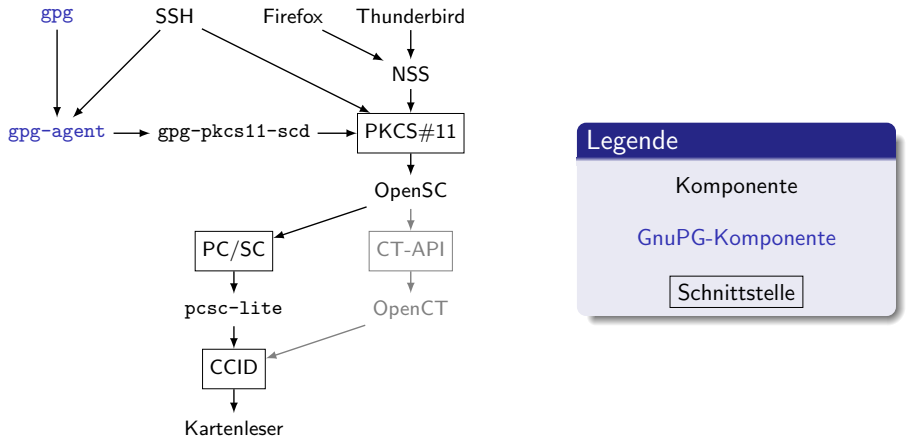
CCID	USB-Standard für Chipkartenleser
CT-API	deutsche API zum Zugriff auf Chipkarten
PC/SC	internationale API zum Zugriff auf Chipkarten
PKCS#11	API-Standard für Krypto-Anwendungen

- Es gibt viele Wege auf die Chipkarte zuzugreifen.

Wege zur Smartcard



Wege zur Smartcard



Smartcard-Troubleshooting

- Es sieht kompliziert aus, funktioniert meist aber „out of the box“.

- CCID bevorzugen

⇒ Bei buggy CCID-Lesern in `~/.gnupg/scdaemon.conf` deaktivieren.

```
# Internen CCID-Treiber deaktivieren und andere Treiber verwenden.
disable-ccid
```

- GnuPG beansprucht exklusiven Zugriff auf PC/SC-Leser

⇒ Weitere Zugriffe über `~/.gnupg/scdaemon.conf` erlauben.

⇒ Potentielle Schwachstelle!

```
# Weitere Zugriff auf PC/SC-Leser erlauben.
pcsc-shared
```


Karte löschen

```
$ gpg-card
gpg/card> factory-reset
OpenPGP Karte Nr. 0005 00007DAF erkannt

Hinweis: Dieses Kommando zerstört alle auf der Karte gespeicherten Schlüssel!

Fortsetzen? (j/N) j
Möchten Sie die Karte wirklich komplett löschen? ("yes" eingeben) yes

gpg/card> quit
```

- Erfordert Eingabe der Admin-PIN

[0] gpg-card gibt es erst ab GnuPG 2.4. In Version 2.2 muss gpg --card-edit verwendet werden.

Schlüssel erzeugen

- 3 Schlüssel-Slots
  1. Signaturschlüssel
  2. Entschlüsselungsschlüssel
  3. Authentifizierungsschlüssel

- Schlüssel auf Karte erzeugen

⇒ Geheimer Schlüssel kann nicht offengelegt werden

⇒ Nicht für Entschlüsselung empfohlen ⇒ Kartenverlust = Datenverlust

- Bestehenden Schlüssel auf Karte verschieben

⇒ Vorher ggf. Backup anlegen

Schlüssel auf Karte erzeugen I

1. Schlüssel erzeugen (Ausgabe gekürzt)

```
$ gpg-card
gpg/card> generate --algo=rsa3072 OPENPGP.3
OpenPGP Karte Nr. 0005 00007DAF erkannt
gpg/card> list
[...]
Signature key ....: [none]
      keyref .....: OPENPGP.1
      algorithm ..: rsa2048
Encryption key....: [none]
      keyref .....: OPENPGP.2
      algorithm ..: rsa2048
Authentication key: 623F1E010B25558CF3C48B1A9A16B272444C8D30
      keyref .....: OPENPGP.3  (sign,auth)
      algorithm ..: rsa3072
      stored fpr .: 206138E678FE07D2F6C8C7C8D56FCAE0C071973F
      created ....: 2025-03-16 22:47:14
gpg/card> quit
```

Schlüssel auf Karte erzeugen II

2. Eine UID (Name) für den Schlüssel anlegen

```
$ gpg --expert --full-generate-key

Bitte wählen Sie, welche Art von Schlüssel Sie möchten:
  (14) Vorhandener Schlüssel auf der Karte
Ihre Auswahl? 14
Karten-Seriennummer: D2760001240103030005000007DAF0000
Vorhandene Schlüssel:
   (1) 623F1E010B25558CF3C48B1A9A16B272444C8D30 OPENPGP.3 rsa3072 (sign,auth*)
Ihre Auswahl? 1
```

- Dem Dialog folgen
- Als Schlüsselzweck nur „Authentisierung“ festlegen
- Name und Gültigkeit nach Bedarf

Schlüssel auf Karte erzeugen III

3. Schlüssel nutzen, wie jeden anderen Schlüssel auch

```
$ gpg --with-keygrip -K 206138E678FE07D2F6C8C7C8D56FCAE0C071973F
sec>  rsa3072 2025-03-16 [CA]
      206138E678FE07D2F6C8C7C8D56FCAE0C071973F
      Keygrip = 623F1E010B25558CF3C48B1A9A16B272444C8D30
      Kartenseriennr. = 0005 00007DAF
uid         [ ultimativ ] Mario Haustein (SSH)
```

- SmartCard-Schlüssel werden auch ohne Eintrag in `~/.gnupg/sshcontrol` berücksichtigt.
- Nur mit Eintrag in `~/.gnupg/sshcontrol` kann der Nutzer zum Einstecken der SmartCard aufgefordert werden.

Was haben wir erreicht?

- Dank GnuPG werden SSH-Schlüssel nicht im Klartext abgelegt.
- Eine ständige Passworteingabe ist dank Agent nicht notwendig.
- Dennoch ist die Nutzbarkeit der SSH-Schlüssel ohne Passworteingabe zeitlich begrenzt.

- SmartCards ermöglichen eine 2-Faktor-Authentifizierung.
  - Besitz Verfügungsgewalt über die SmartCard
  - Wissen Eingabe der PIN
- Mit SmartCards lassen sich Schlüssel sicher zwischen verschiedenen Endgeräten transportieren.

Antworten auf Fragen, die im Vortrag evt. unklar oder konfus waren I

- Sollte man nicht lieber `ssh-copy-id` benutzen?
  - Ja
  - Ohne `-i` Parameter werden alle Schlüssel des Agents eingerichtet.
  - Sonst öffentlichen Schlüssel exportieren und mit `-i` angeben.
  - Voraussetzung für `ssh-copy-id` bleibt, dass man sich initial per alternativen Verfahren anmelden können muss (z. B. Passwort).

- Sollte man nicht liebe elliptische Kurven verwenden?
  - Das ist empfehlenswert, wenn die SmartCard es kann.
  - Operationen auf elliptischen Kurven sind deutlich schneller. Man merkt das auch während des Verbindungsaufbaus durch `ssh`.
  - Eine aufwändige Primzahlsuche wie bei RSA entfällt. Jede zufällige Zahl ist ein gültiger geheimer EC-Schlüssel.

Antworten auf Fragen, die im Vortrag evt. unklar oder konfus waren II

- Ist der Signaturverfahren `ssh-rsa` nicht veraltet?
  - Ja. Das Signaturverfahren `ssh-rsa` basiert auf SHA1 und sollte nicht mehr verwendet werden.
  - Der Ausdruck „`ssh-rsa`“ in `authorized_keys` definiert aber den Schlüsseltyp und hat nichts mit dem Signaturverfahren zu tun. Es heißt nur gleich.
- Genauere Erklärung was auf der Karte passiert und was nicht:
  - Für eine Signatur wird ein Hash über die zu signierenden Daten erstellt (Session-ID).
  - Bei RSA muss dieser Hash auf Schlüssellänge aufgefüllt werden (Padding), bei elliptischen Kurven u. U. gekürzt werden. Für das Padding gibt es zudem verschiedene Verfahren (PKCS vs. PSS).
  - Das Hashing passiert in der Praxis außerhalb der SmartCard (auch wenn die Karte es theoretisch in Hardware kann), das Padding in der Karte.
  - Damit ist das Hashing-Verfahren vollständig unter Kontrolle der Software. Es wird per `PubkeyAcceptedAlgorithms` festgelegt, nicht in `authorized_keys`.

Antworten auf Fragen, die im Vortrag evt. unklar oder konfus waren III

- Führt das Ausprobieren von SSH-Keys nicht zu fehlerhaften Anmeldungen?
  - Nein
  - Innerhalb einer Session bietet der Client dem Server seine öffentlichen Schlüssel an. Der Server kann die Schlüssel ablehnen oder unter Angabe eines öffentlichen Schlüssels zur Erstellung einer Signatur auffordern (Abschnitt 7 RFC 4252).
  - Der gesamte Vorgang wird als ein Anmeldeversuch gewertet.
- Wie kann man bei der Verwendung von SSH-Agents einen bestimmten Schlüssel erzwingen?
  - Durch `IdentityFile` in der Client-Konfiguration kann man zu verwendende Schlüssel (ggf. Server-spezifisch) festlegen.
  - Bei der Schlüsselspeicherung in Dateien wird i. d. R. der die geheime Schlüsseldatei angegeben. Man kann aber auch die öffentliche Schlüsseldatei angeben.
  - Bei der Verwendung eines SSH-Agents muss der öffentliche Teil des gewünschten Schlüssels als Datei exportiert und mit `IdentityFile` konfiguriert werden.

Wo ist `GPG_TTY` notwendig?

- Programme in einer Pipe haben kein Terminal.
- Für die Passwortabfrage ist jedoch ein Terminal notwendig.
- Wir können per Umgebungsvariable `GPG_TTY` signalisieren, auf welchem Terminal wir arbeiten.
- Per `~/.bashrc` festlegen

```
# Terminal für gpg-Aufruf in Pipe festhalten
export GPG_TTY=$(tty)
```


Was bedeutet `UPDATESTARTUPTTY`?

- GnuPG muss auf einem Terminal oder Display nach der Passphrase fragen dürfen.
- Das SSH-Agent-Protokoll hat keine Möglichkeit das aktuelle Terminal bzw. Display zu signalisieren.
- Folgendes Kommando teilt dem GnuPG-Agent das aktuelle Terminal / Display mit.

```
$ gpg-connect-agent UPDATESTARTUPTTY /bye
```

⇒ Muss vor jedem SSH-Aufruf ausgeführt werden.

- `/etc/ssh/ssh_config` bzw. `~/.ssh/config`

```
Match host * exec "gpg-connect-agent UPDATESTARTUPTTY /bye"
```

Tipps und Tricks für gpg-agent.conf

- Bei vielen parallelen SSH-Verbindungen:

```
auto-expand-secmem
```

- Manche Fenstermanager haben eigene Passwort-Caches.

⇒ Ausschalten:

```
no-allow-external-cache
```


Schlüsselimport I

- `ssh-add` lädt den Schlüssel in den GnuPG-Schlüsselbund. Es wird die bestehende und die zukünftige Passphrase abgefragt.

```
$ ssh-add ~/.ssh/id_rsa
Enter passphrase for /home/demo/.ssh/id_rsa:
Identity added: /home/demo/.ssh/id_rsa (/home/demo/.ssh/id_rsa)
```

- Schlüssel ist verfügbar.

```
$ ssh-add -l
3072 SHA256:yT/Y2OUTOCY+Xc2sB/aKWtwWeXkWjEzYlo3YeVpPooQ [...]
```


Schlüsselimport II

- Der Schlüssel wird automatisch in `~/.gnupg/sshcontrol` eingetragen.

```
# RSA key added on: 2025-03-15 08:37:17
# Fingerprints:  MD5:5c:6f:a7:b8:6b:e1:57:76:f4:8a:ba:bc:60:2f:eb:b5
#                SHA256:yT/Y2OUTOCY+Xc2sB/aKWtwWeXkWjEzYlo3YeVpPooQ
EE06F225415CC9B823D774A80DC1E49CD397E9B3❶ 0
```

- Dem Schlüssel müssen wir noch einen Namen geben.
  Den Keygrip ❶ entnehmen wir `~/.gnupg/sshcontrol`.[1]

```
$ gpg --batch --generate-key << EOF
Key-Type: RSA❷
Key-Grip: EE06F225415CC9B823D774A80DC1E49CD397E9B3❶
Key-Usage: auth
Name-Real: Mario Haustein
Name-Comment: SSH
Expire-Date: 0
EOF
```

[1] Für andere Schlüsseltypen bei ❷ „`ECDSA`“ (EC-Signatur) oder „`EDDSA`“ (Edwards-Signatur) einsetzen.

Schlüssel auf Karte verschieben I

1. Bestehender Schlüssel

```
$ gpg --with-keygrip -K AFB803D27B0F5110D1004660208B30E174DECB24
sec   rsa3072 2025-03-14 [CA]
      AFB803D27B0F5110D1004660208B30E174DECB24
      Keygrip = 211B2DEF0FC050A80B3E097DBF9F7F51E73CED5D
uid         [ ultimativ ] Mario Haustein (SSH)
```

Schlüssel auf Karte verschieben II

2. Schlüssel verschieben (Ausgabe gekürzt)

```
$ gpg --edit-key AFB803D27B0F5110D1004660208B30E174DECB24

Geheimer Schlüssel ist vorhanden.

sec  rsa3072/208B30E174DECB24
     erzeugt: 2025-03-14  verfällt: niemals     Nutzung: CA
     Vertrauen: ultimativ     Gültigkeit: ultimativ
[ ultimativ ] (1). Mario Haustein (SSH)

gpg> keytocard
Den Hauptschlüssel wirklich verschieben? (j/N) j
Wählen Sie den Speicherort für den Schlüssel:
   (1) Signatur-Schlüssel
   (3) Authentisierungs-Schlüssel
Ihre Auswahl? 3
Hinweis: Die lokale Kopie des geheimen Schlüssels wird nur nach "save" gelöscht.
gpg> save
```


Schlüssel auf Karte verschieben III

3. Fertig

```
$ gpg --with-keygrip -K AFB803D27B0F5110D1004660208B30E174DECB24
sec>  rsa3072 2025-03-14 [CA]
      AFB803D27B0F5110D1004660208B30E174DECB24
      Keygrip = 211B2DEF0FC050A80B3E097DBF9F7F51E73CED5D
      Kartenseriennr. = 0005 00007DAF
uid         [ ultimativ ] Mario Haustein (SSH)
```