GnuPG-Card Anleitung

Andere Anleitungen

GnuPG-Card

- Anleitung der Free Software Foundation Europe
- Anleitung von spin.atimoicobject.com

Best Practice (Nutzung von Haupt- und Subkeys) für gnupg

- http://www.openpgp-schulungen.de/kurzinfo/schluesselqualitaet/
- http://www.openpgp-schulungen.de/inhalte/
- http://blog.andreas-haerter.com/2010/04/12/gnupg-schluesselhierarchie-passender-algorithmus-fuer-jede-aufgabe-schluesselaustausch-ohne-verlust-des-web-of-trust#fn__8
- https://alexcabal.com/creating-the-perfect-gpg-keypair/
- http://www.bootc.net/archives/2013/06/07/generating-a-new-gnupg-key/

Grundsätzliche Infos über Smart Cards

Sogenannte Smart Cards können zur Authentifizierung, Verschlüsselung und Signatur mit unterschiedlichen Diensten und Programmen dienen.

Es gibt zwei unterschiedliche Konzepte:

- Die „pki cards“ (PKCS#11), siehe hier: https://wiki.debian.org/Smartcards
- und die vor allem unter Linux sehr gut nutzbaren Gnupg-Cards https://wiki.debian.org/Smartcards/OpenPGP

Während erstere den Quasi-Standard für x509 Zertfikate und eine Public-Key-Infrastructure darstellen scheint die gnupg Variante unter Linux einfacher einzurichten und besser unterstützt zu sein.

Ähnliche Konzpete verfolgen auch

- der Yubikey - eine Mischung aus USB-smartcard und One-Time-Password Generator
- und der Cryptostick der Privacy-Foundation.

Ein Überblick über SmartCard, Cryptostick und Yubikey findet sich im Privacy-Handbuch

Unterschiede der verschiedenen Systeme:

System	Vorteile	Nachteile

Gnupg Card	Karten leicher erhältlich und preiswerter. Unterstützung unter Linux einfacher	Authentifizierung mit Clients wie Thunderbird etc. nicht so einfach
PKCS#11	De Facto Standard	komplizierter in der Einrichtung, Karten teurer?, Beschaffung komplizierter, Kompatibilität nicht immer gegeben

Vor- und Nachteile sind z.B. hier diskutiert worden:
http://lists.gnupg.org/pipermail/gnupg-users/2006-February/027936.html

Diese Anleitung beschäftigt sich nur mit der Einrichtung der Gnupg-Card.

Voraussetzungen

Wir benötigen

- eine Gnupg-Card
- einen kompatiblen Cardreader

Beides ist am einfachsten über den Shop von Kernel Concepts zu erhalten. Eine Gnupg-Card erhält auch, wer der Free Software Foundation Europe beitritt.

Software installieren

Als erstes installieren wir die Software. Wichtig: die gnupg-card funktioniert nur mit gnupg Version 2.

```
apt-get install gnupg2 scdaemon gpgsm pcscd
```

Wenn der PKCS#11-Support verwendet werden soll evtl. auch:

```
apt-get install gnupg-pkcs11-scd
```

Card-Reader einrichten

In der Anleitung der Free Software Foundation Europe wird beschrieben, wie der Card-Reader so eingerichtet wird, dass das USB-Device die richtigen Rechte und Gruppenzugehörigkeit erhält.

Unter Debian Wheezy war das nicht nötig. Der Card-Reader funktionierte nach der Installation der Software „Out of the Box“.

Das Device des Card-Readers muss die richtigen Dateirechte erhalten, damit die User darauf zugreifen können. Das erfolgt mittels udev und wird hier beschrieben

Als schneller Workaroundzum Testen oder zur Nutzung innerhalb von Live-Systemen (zur sicheren Generierung der Schlüssel) kann das folgendermaßen getan werden

```
lsusb
Bus 001 Device 008: ID ... Card-Reader ....
```

damit wissen wir die Bus und Device ID:

```
ls -l /dev/bus/usb/001/008
```

als temporärer Workaround:

```
chmod 666 /dev/bus/usb/001/008
```

Schlüssel erzeugen und auf Karte transferrieren

Es gibt grundsätzlich mehrere Möglichkeiten, die Schlüssel für die Karte zu erzeugen

1. Keypair (Hauptkey) kann direkt auf der Karte erzeugt werden
2. Ein Keypair wird außerhalb erzeugt werden. Anschließend werden Subkey generiert (für signieren, verschlüsseln und Authentifizieren) und auf die Karte importiert. Diese Variante ist die von der empfohlene und im Folgenden beschriebene.

Hauptkey mit Subkeys

- Eine ausführlichere Anleitung (incl. Backup der keys findet sich hier: http://www.bootc.net/archives/2013/06/07/generating-a-new-gnupg-key/)

Das Generieren der Schlüsselpaare findet vorzugsweise auf einem Live-System statt, damit sicher gestellt werden kann, dass die Schlüssel nicht bereits bei der Erstellung von Dritten gelesen werden können. Ist der geheime Schlüssel einmal auf der Karte, kann er, so zumindest die Theorie, diese nicht mehr verlassen. Alle Operationen finden auf der Karte statt. Daher lohnt es sich, bei der Erstellung der Schlüssel wirklich sehr sorgfältig vorzugehen:

- vertrauensvolle Hardware
- Live-System möglichst ohne Netzwerkverbindungen

Ein angepasstes Live-System mit aller nötigen Software sowohl zum Erzeugen der Keys als auch für die Verwendung der Gnupg-Karte kann beim Datenkollektiv bestellt werden.

Ihr erhaltet einen USB-Stick mit

- angepasstem xfce4 Desktop und allen nötigen Crypto-Tools
- einer zusätzlichen LUKS-verschlüsselten Partition, die z.B. für die Backups der privaten Schlüssel dienen kann.

Hauptschlüssel-Paar generieren:

Ist alles so weit vorbereitet, geht es jetzt an das eigentliche Erstellen der Schlüssel. In einem Terminal werden jeweils die markierten Kommandos ausgeführt.

Als erstes wird der Hauptschlüssel erstellt, der nur zum Signieren der Unterschlüssel dient:

```
gpg2 --gen-key
```

Bei der Auswahl:

```
RSA (sign only)
```

- ggf. unbegrenzte Haltbarkeit

Erstellen der Unterschlüssel (Subkeys)

Mit

```
gpg --list-keys
```

finden wir unsere neue Key-ID heraus. Diese nutzen wir nun zum Editieren des Schlüssels:

```
gpg --expert --edit-key KEYIDXYZ
```

Im Anschluss generieren wir drei Subkeys für folgende Zwecke:

1. Signieren
2. Verschlüsseln
3. Authentifizieren

```
gpg>  addkey
```

legt einen neuen Subschlüssel an. Wir müssen jetzt je einen zum Signieren, Verschlüsseln und Authentifizieren anlegen, d.h. die folgende Prozedur wird **dreimal** wiederholt.

Bei der folgenden Auswahl:

```
Please select what kind of key you want:
(3) DSA (sign only)
(4) RSA (sign only)
(5) Elgamal (encrypt only)
(6) RSA (encrypt only)
(7) DSA (set your own capabilities)
(8) RSA (set your own capabilities)
Your selection?
```

nutzen wir „8“ „Set your own capabilities“. Signieren und Verschlüsseln funktioniert zwar auch mit den vorgegebenen, Authentifizieren funktioniert aber nur mit benutzerdefinierten Fähigkeiten.

Jetzt kann mit den angegebenen Buchstaben jeweils eine Fähigkeit zu- oder abgeschaltet werden.

Zuletzt sichert ein

```
gpg> save
```

unsere Änderungen.

Schlüssel auf die Gnupg-Karte kopieren

Im nächsten Schritt kopieren wir unsere Subkeys auf die gnupg-Karte. Hierfür benötigen wir das Passwort für den geheimen Hauptschlüssel und die Admin-Pin für die Karte.

```
gpg --expert --edit-key KEYIDXYZ
```

Ein

```
gpg> toggle
```

schaltet in die Ansicht der privaten Schlüssel um.

Jetzt markieren wir der Reihe nach unsere Subkeys:

```
gpg> key 1
```

und kopieren sie auf die Karte:

```
gpg> keytocard
```

Das wiederholen wir jetzt mit Schlüssel 2 u. 3. (Die Schlüssel müssen jeweils wieder entsperrt werden.)

Backup

- Der gesamte Keyring (Unterverzeichnis .gnupg) sollte nach der Prozedur auf ein verschlüsseltes Medium bzw. in einem Container gesichert werden.
- eine weitere Variante zum Backup von Keys ist paperkey - ein Programm um ausdruckbare Daten zu erhalten und den key auf Papier auszudrucken.

Nutzung zum Signieren / Verschlüsseln

Nutzung für ssh

- http://www.ozonesolutions.com/programming/2014/04/pgp-smart-card-ssh-login-gpg-agent-ubu

ntu/
- http://www.bootc.net/archives/2013/06/09/my-perfect-gnupg-ssh-agent-setup/
- https://grepular.com/Smart_Cards_and_SSH_Authentication

Wir müssen ssh-Support im gpg-agent konfigurieren:

```
echo "enable-ssh-support" >> ~/.gnupg/gpg-agent.conf
```

ssh über mehrere Server (ForwardAgent)

- http://livecipher.blogspot.co.uk/2013/02/ssh-agent-forwarding.html

Damit das funktioniert, muss der ssh-client so konfiguriert werden, dass er „agent-forwarding“ untestützt:

- /etc/ssh/ssh_config (sytemweit)
- /home/user/.ssh/config (per user)

```
Host *
      ForwardAgent yes
```

Eine ausführliche Beschreibung findet sich hier:

- http://www.unixwiz.net/techtips/ssh-agent-forwarding.html

Innerhalb von screen-sessions

Hier steht was dazu:

- http://www.bootc.net/archives/2013/06/09/my-perfect-gnupg-ssh-agent-setup/

Aber mit debian-wheezy sollte das out of the box funktinieren.

Nutzung für X2Go

- http://wiki.x2go.org/doku.php/wiki:advanced:authentication:passwordless-gpg-card

Auch der x2goclient kann auf die ssh-Authentifizierung über gpg-agent mittels gpg-card zurück greifen. x2goclient muss dazu mit der Option:

```
x2goclient --pgp-card
```

gestartet werden.

Allerdings beherrscht der x2goclient kein ssh-agent Forwarding. D.h. innerhalb der x2go-Sitzung kann der lokale gpg-agent für die ssh-Authentifizierung nicht genutzt werden.

Eleganter ist daher die Nutzung des alternativen X2go-Clients: pyhoca-gui.

```
apt-get install pyhoca-gui python-cups
```

Dieser beherrscht nicht nur die Authentifizierung via smart-card out of the box - sondern auch ssh-forward-agent.

Nutzung für Pam

- http://www.schiessle.org/howto/poldi.php

Die dort verwendeten Optionen für poldi-ctrl scheinen in der wheezy-Version noch nicht zu existieren. Evtl. hilft daher folgendes weiter:

- http://walter.silvergeeks.com/rechner/howto/how-to-anmeldung-mit-smart-card-oder-usb-token-am-lokalen-linux-system/

Nutzung für Luks

- http://digitalbrains.com/2014/gpgcryptroot
- http://lists.gnupg.org/pipermail/gnupg-users/2009-November/037599.html → hier wird gezeigt, wie mensch die vier private „DO“ data auf der Karte verwenden kann
- Hier eine komplette Anleitung für Ubuntu Luks mit 2-Faktor Authentifizierung: https://blog.kumina.nl/2010/07/two-factor-luks-using-ubuntu/

Nutzung mit Passwortmanagern

keepassX

- https://www.keepassx.org/forum/viewtopic.php?f=2&t=2188

Die dort beschriebene Lösung funktioniert offensichtlich nicht mehr. Schuld daran scheint zu sein, dass der über socat gebildete Socket keine Informationen über die Kodierung des zur Verfügung gestellten Files ausgibt, keepassx darauf aber zurück greifen will.

pass

Ein Kommandozeilen Passwort-Manager, der Passwörter mittels gpg verschlüsselt und daher auch „smart-card kompatibel“ ist: pass (in wheezy-backports)

```
apt-get install -t wheezy-backports pass
```

Dieses legt für jeden Passwort-Eintrag eine Datei unterhalb von `.password-store` an. Die Baumstruktur wird mittels Unterverzeichnissen erzeugt.

Features (Auswahl):

- git-Integration
- Mulitline-Passwörter (mit Kommentaren etc.)
- Baum-Struktur
- Suchen in Passwörtern und Namen des Eintrags
- Verschlüsselung der Passwörter für mehrere User-IDs

Mittels git und der gpg-Encryption für mehrere User-Ids ist der Passwort-Manager Multi-User fähig. Für einzelne Unterordner können auch verschiedene gpg-IDs zum Verschlüsseln genutzt werden: Auf diese Art können Zugriffsbeschränkungen gesetzt werden.

Eine Konvertierung aus einer keepassx xml-Export-Datei gelingt mit folgendem Python-Script:

- http://git.zx2c4.com/password-store/tree/contrib/importers/keepassx2pass.py

→ howto zu pass