Arbeitsanleitung und Begleittext zum Vortrag über Kryptografie, Zertifikate und Smartcards

Vorwort

Alle in diesem Dokument und den Vortragsfolien vorgestellten Anweisungen und Befehle sind nach bestem Wissen zusammengestellt, für die Richtigkeit übernehme ich keine Haftung. Ich beschränke mich auf Debian GNU/Linux und auf die Verwendung von CAcert Zertifikaten. Das Erstellen von Zertifikaten ist nicht Gegenstand dieser Anleitung. Es wird ein vorhandenes Zertifikatspaket (.p12) vorausgesetzt.

Im März 2014, Rolf Wald

Inhaltsverzeichnis

1 Was ist ein Zertifikatspaket

Ein Zertifikatspaket mit der Endung .p12 ist eine Datei im Archivformat pkcs #12. Sie enthält normalerweise einen privaten Schlüssel und das dazugehörige X.509 Zertifikat, dazu wahlweise die X.509 Zertifikatskette bis zur RootCA. Ein X.509 Zertifikat enthält einen öffentlichen Schlüssel, dessen dazugehörige Eignerdaten von einer Zertifizierungsstelle (CA) bestätigt/signiert worden sind. Die Wahl der CA ist frei, auch Eigenzertifikate des Besitzers des öffentlichen Schlüssels sind möglich. Wieviel Vertrauen der jeweiligen CA geschenkt wird, muss jeder selbst entscheiden.
Eine .p12 Datei kann jederzeit zerlegt und in verschiedene Formate umgewandelt werden. Dazu geben die Manpages von openssl Auskunft. Die notwendigen Umwandlungen werde ich an passender Stelle erwähnen.

2 Was ist für die Nutzung von Smartcards nötig

Für die Nutzung einer Smartcard ist eine Kombination Smartcard/-leser nötig, die unter Linux funktioniert. Die folgenden Programmpakete müssen installiert werden

```
apt-get install pcscd opensc
```

Weitere Programme sind für spezielle Kartenleser verfügbar.
'pcscd' stellt die Schnittstelle des Systems zu den verschiedenen Kartenlesern zur Verfügung, eine vielgenutze Spezifikation hat die PC/SC Group entwickelt, so dass etliche Leser ohne weitere Spezialmodule funktionieren (Stichwort: ccid-kompatibel).
'opensc' ist für die Kommunikation mit der Smartcard zuständig. Da jeder Kartenhersteller ein eigenes Layout und spezifische Befehle zur Kommunikation mit dem Kartenchip hat, muss opensc jeden Kartentyp individuell behandeln. Das gelingt den Programmentwicklern nur, wenn sie genügend Informationen vorliegen haben. Freie und offene Software verträgt sich an dieser Stelle nicht mit der Haltung der meisten Hersteller, die ihre Produkte als Betriebsgeheimnis einstufen. Daher ist die Auswahl an unterstützten Smartcards nicht sehr groß, und es werden nicht immer alle Möglichkeiten unterstützt.

3 Der Einsatz von Zertifikaten (Client)

Zertifikate, der Sprachgebrauch schließt hier die privaten Schlüssel mit ein, können in Dateiform genutzt werden. Da es keinen absoluten Schutz vor Kopien gibt, ist die Absicherung des privaten Schlüssels durch ein gutes Passwort (lange Passphrase mit Ziffern, großen und kleinen Buchstaben und Sonderzeichen) geboten. Werden Zertifikate auf einer Smartcard gespeichert, so erhöht sich die Sicherheit vor unbefugtem Zugriff erheblich. Zum einen sind die dort gespeicherten privaten Schlüssel nicht kopierbar - alle Berechnungen werden auf dem Kartenchip ausgeführt, zum anderen ist es leichter, sich nur eine Karten-PIN zu merken. Leider sind nicht alle Programme, die den Einsatz von Zertifikaten erlauben, auch smartcardgeeignet. Im Gegenzug eröffnen sich mit der Smartcard zusätzliche Möglichkeiten in der Verschlüsselung von Datenträgern und bei der lokalen Authentifizierung des Benutzers.

3.1 E-Mail Kommunikation

Die Kommunikation über E-Mails hat in unserem Leben den herkömmlichen Schriftwechsel fast ersetzt. Wollte man die elektronische Kommunikation von heute mit dem früheren Schriftverkehr vergleichen, so würden heute viel mehr Postkarten als Briefe verschickt. Nur eine verschlüsselte

E-Mail ist mit einem zugeklebten Brief zu vergleichen, und die Signatur des Absenders entspricht einem persönlichen Siegel, mit dem der Empfänger die Identität des Schreibers und die Unversehrtheit des Inhalts feststellen kann. Meine Empfehlung lautet daher, alle augehenden E-Mails zu signieren. Wenn das öffentliche Zertifikat des Empfängers vorhanden ist, sollten die E-Mails auch verschlüsselt werden. Für die Verwendung von Zertifikaten ist der S/MIME Standard eingeführt worden. Das Fehlen eines zentralen Archivs aller vorhandenen öffentlichen Zertifikate ist m.E. keine Einschränkung, vor allem, wenn alle Nutzer von Zertifikaten ihre E-Mails immer signieren. Die Signatur einer E-Mail nach S/MIME Standard enthält immer den öffentlichen Schlüssel des Signierenden, nach Erhalt bereits der ersten signierten E-Mail, kann ab diesem Zeitpunkt der E-Mail Verkehr mit diesem Adressaten verschlüsselt erfolgen.

3.1.1 Thunderbird

Das E-Mail Programm Thunderbird (bei Debian Icedove genannt) hat die Unterstützung von S/MIME bereits eingebaut. Die Konfiguration für den Einsatz von Zertifikaten ist daher besonders einfach, das gilt auch bei einer funktionierenden Smartcardinstallation. Das native Zertifikatspaket (.p12) kann direkt unter `Bearbeiten - Einstellungen - Erweitert - Zertifikate (2x)`, dann `Importieren...` eingefügt werden. Ähnlich unkompliziert ist die Nutzung einer Smartcard. Statt des Zertifikatspakets wird unter `Bearbeiten - Einstellungen - Erweitert - Zertifikate - Kryptographie-Module` und dann `Laden` der Ort des opensc-Moduls (z.B. **/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so**) eingetragen. Das genügt, und nach Eingabe der PIN stehen die Zertifikate wie beim Import des .p12 Pakets zur Verfügung.

3.1.2 Kmail,...

Sämtliche E-Mail Programme ohne native S/MIME Unterstützung bedienen sich der Hilfe des Programms gpgsm (KMail, Claws-Mail,...). Falls noch nicht installiert, müssen die Pakete **gpgsm**, **gnupg2**, **dirmngr** und **gnupg-pkcs11-scd** (ersetzt den scdaemon für unsere Smartcards) installiert werden.

```
apt-get install gpgsm gnupg2 dirmngr gnupg-pkcs11-scd
```

Die Konfiguration erfolgt über einfache Textdateien, die wie folgt angepasst werden müssen

/etc/dirmngr/dirmngr.conf

```
log-file /var/log/dirmngr/dirmngr.log
# Der systemweite dirmngr wird bereits vor der Useranmeldung
# gestartet und schreibt seine Logs in eine eigene Datei
allow-ocsp
# die Gültigkeitsprüfung erfolgt online über OCSP
add-servers
# Beschreibung man-pages dirmngr
```

<userhome>/.gnupg/gpg-agent.conf

```
scdaemon-program /usr/bin/gnupg-pkcs11-scd
# Beschreibung siehe man-pages gpg-agent
pinentry-program /usr/bin/pinentry-gtk-2
# pinentry je nach Oberfläche wählen
allow-mark-trusted
```

<userhome>/.gnupg/gpgsm.conf

```
enable-ocsp
# man-pages gpgsm
prefer-system-dirmngr
auto-issuer-key-retrieve
include-certs -1
disable-policy-checks
(default-key <fingerprint_of_your_default_key>)
# optional, wenn mehrere Schlüssel vorhanden sind
```

Im Verzeichnis **/etc/dirmngr/trusted-certs** müssen die root-Zertifikate der verwendeten CA(s) in binärer Form (.der) hinterlegt werden. Für CAcert sind dies das Root-CA und das Class3-Root-CA Zertifikat. Wenn sie noch nicht vorhanden sind, so können diese bei cacert.org heruntergeladen werden.
Jetzt können der/die Schlüssel (.p12-Pakete) mit

gpgsm --import <Zertifikatspaket.p12>

zum Keyring hinzugefügt werden. Ähnlich einfach geht es mit Schlüsseln, die auf einer Smartcard sind.

gpgsm --learn-keys

macht die Schlüssel fürs System nutzbar.

Ob alles richtig funktioniert, kann mit

gpgsm --list-secret-keys –with-validation

überprüft werden. Hier wird zunächst für das CAcert Root-Cert ein Fehler gemeldet, da die Überprüfung dieses Zertifikats fehlgeschlagen ist und damit alle davon abgeleiteten Zertifikate als ungültig angesehen werden. Es ist jetzt in der **<userhome>/.gnupg/trustlist.txt** (eine Auflistung aller vorhandenen Root-Zertifikate, die durch den gpg-agent dort gespeichert werden) noch der Parameter 'relax' hinter dem Fingerprint-Eintrag des CAcert Root-CA anzufügen (man pages gpg-agent). Erst nach einem Neustart des gpg-agent (abmelden des users und neu anmelden oder dem Prozess ein HUP-Signal senden) wird die gesamte Zertifikatskette als gültig betrachtet.

Wenn der systemweite dirmngr nicht verwendet/gestartet werden soll, muss das dirmngr-Verzeichnis mit allen Einträgen und Dateien in **<userhome>/.gnupg/** vorhanden sein. Dann wird für die Überprüfung der Zertifikate lokal ein temporärer dirmngr gestartet.

Als Beispiel soll hier jetzt KMail dienen, dieser greift auf das Programm kleopatra als Zertifikat-Manager zurück. Kleopatra ist ein Frontend für gnupg und gpgsm, es verwaltet alle Schlüssel

die in den Keyrings des angemeldeten Benutzers verfügbar sind. Um einen bestimmten Schlüssel in KMail für eine Identität zu verwenden ist der Aufruf

```
Einstellungen - KMail einrichten - Identities - (hinzufügen/ändern)
– [Tab]Kryptografie
```

nötig, dann werden die verfügbaren Schlüssel angezeigt und es kann gewählt werden. Dabei wird gleich die Gültigkeit überprüft - nach Mausklick auf den Eintrag erscheint nach kurzem Warten ein grüner Haken, wenn alles in Ordnung ist.

3.2 Client-Authentifikation im Web

Viele Webseiten lassen sich nicht nur über das http-Protokoll aufrufen, sondern auch über https. Das bedeutet die Einfügung einer neuen Protokollebene zwischen TCP/IP und HTTP mittels SSL/TLS. Damit erfolgt eine Verschlüsselung und Authentifizierung der Kommunikation zwischen Webserver und Browser. Der Browser als Client kann durch das ihm übermittelte Zertifikat feststellen, ob die Kommunikation wirklich mit dem gewünschten Server stattfindet. Eine weniger genutzte Möglichkeit ist der umgekehrte Weg, dass sich der Browser als Client gegenüber dem Webserver authentifiziert. Es können Webseiten mit Zugangsbeschränkungen elegant und ohne Passwörter erreicht werden, wenn der Webserver eine Prüfung des vom Client übermittelten Zertifikats durch seine Konfiguration erlaubt. Hierbei geht es nur um die Authentifizierung des Clients, für die verschlüsselte Kommunikation genügt das Zertifikat des Servers.

3.2.1 Firefox

Für die Verwendung von Zertifikaten und Smartcards verweise ich auf **3.1.1** , da dort bereits alles beschrieben ist. Firefox (Debian: Iceweasel) und Thunderbird verwenden die gleiche Technik.

3.2.2 Chromium

Bei Zertifikaten als .p12 Datei ist die Vorgehensweise bei chromium ähnlich der von firefox. Unter

```
Einstellungen - erweiterte Einstellungen - HTTPS/SSL Zertifikate
verwalten - ihre Zertifikate
```

ist über **Importieren...** die direkte Angabe der .p12 Datei möglich. Die Nutzung von Smartcards ist hingegen etwas aufwändiger, es gibt keine GUI (grafische Eingabe) dafür. Zunächst muss das Paket **libnss3-tools** installiert sein:

```
apt-get install libnss3-tools
```

Dann ist der Befehl

```
modutils -dbdir sql:<DB-Verz.> -add <Modul-Name> -libfile \
/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so friendly
```

auf einer Konsole einzugeben. Das DB-Verzeichnis kann sein: **/etc/pki/nssdb** (root-Rechte nötig) oder <**userhome**>/**.pki/nssdb** . Prüfen auf korrekte Funktion erfolgt mit:

```
modutils -dbdir sql:<DB-Verz.> -list
```

Dort sollte dann der Kartenleser und die Karte unter dem Namen, den man als <Modul-Name> vergeben hat, aufgeführt werden.

3.3 Authentifizierung im Netzwerk

Etliche Dienste im Netzwerk sind auch über Zertifikate nutzbar.

3.3.1 WPA Enterprise

Das Hauptmerkmal bei WPA Enterprise ist die Nutzung eines externen Radius-Servers, der die Authentifizierung übernimmt. Dabei gibt es verschiedene Möglichkeiten, aus denen ich den TLS-Mode ausgewählt habe. Hier genügt ein Zertifikat zur Authentifizierung am Radius-Server, der den angefragten Dienst freigibt, nachdem die Auswertung des vorgelegte Zertifikats erfolgreich war. Ich betrachte an dieser Stelle nicht die Konfiguration z.B. eines Wlan-APs zur Kommunikation mit einem Radius-Server, sondern einzig die Einstellungen auf der Rechnerseite.

3.3.1.1 (GUI) Network-Manager -wireless

Die grafische Oberfläche eines **network-manager** erlaubt über die *Verbindungseinstellungen* und Auswahl der gewünschen Verbindungsart und dem Namen der Verbindung, ganz einfach im Tab *Sicherheit des Funknetzwerks* die Eintragungen vorzunehmen:

```
Sicherheit - WPA&WPA2 Enterprise
Legitimierung – TLS
Identität - xxx (Feld darf nicht leer sein, wird aber von mir im
Radius-Server nicht ausgewertet)
Benutzerzertifikat - <cert.pem>
CA-Zerifikat - <ca.pem>
Geheimer Schlüssel - <key.pem>
Passwort des geheimen Schlüssels - <password>
```

Bei neueren Versionen der GUI kann auch direkt die .p12-Datei verwendet werden. Ansonsten werden die Schlüsselbestandteile mit folgenden Befehlen extrahiert:

```
openssl pkcs12 -in <cert>.p12 -out <cert>.pem -nokeys -clcerts
```

--> cert.pem

```
openssl pkcs15 -in <cert>.p12 -out <cert>ca.pem -cacerts -nokeys
```

--> certca.pem

```
openssl pkcs12 -in <cert>.p12 -out <cert>key.pem -nocerts
```

--> certkey.pem.

3.3.1.2 (NoGUI) wpa_supplicant -wireless

Für die Nutzung einer Smartcard gibt es keine grafische Oberfläche, die Einstellungen werden direkt in der wpa_supplicant.conf vorgenomen:
Zuvor muss noch das Paket libengine-pkcs11-openssl installiert werden.

```
apt-get install libengine-pkcs11-openssl
```

/etc/wpa_supplicant.conf (Beispielkonfiguration Balista Wlan-Netz)

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=users
pkcs11_engine_path=/usr/lib/engines/engine_pkcs11.so
pkcs11_module_path=/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
update_config=1
ap_scan=1

network={
        ssid="balista"
        key_mgmt=WPA-EAP
```

```
        eap=TLS
        identity="User"
        engine_id="pkcs11"
        key_id="1:a4082239dXXXXXXXXXXXXXXXXXXXXXXXc1c10f7fd3"
        cert_id="1:a4082239dXXXXXXXXXXXXXXXXXXXXXXXc10f7fd3"
        ca_cert_id="1:ef47e5fcaXXXXXXXXXXXXXXXXXXXXX0e54fc3af"
        engine=1
        disabled=1
}
```

Die IDs werden mit
pkcs15-tool -D
von der Karte ausgelesen, das vorangestellte 1: legt die Nummer des Readers bzw. Slots fest.

Selbstverständlich ist diese Art der Konfiguration auch für Zertifikatsdateien möglich. Hier ein Beispiel: Die .p12 Datei muss wie unter **3.3.1.1** beschrieben in .pem Dateien umgewandelt werden.

/etc/wpa_supplicant.conf (Beispielkonfiguration Balista Wlan-Netz)

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=users
update_config=1
ap_scan=1

network={
        ssid="balista"
        key_mgmt=WPA-EAP
        eap=TLS
        identity="User"
        private_key="/etc/cert/<name>key.pem"
        client_cert="/etc/cert/<name>cert.pem"
        ca_cert="/etc/cert/<name>ca.pem"
        disabled=1
}
```

Für die Nutzung der *ifup/ifdown* Funktionalität muss das entsprechende Interface in **/etc/network/interfaces** wie folgt eingetragen werden:

/etc/network/interfaces

```
    ...
    ...
iface wlan0 inet dhcp
pre-up wpa_supplicant -B -Dwext -c /etc/wpa_supplicant.conf -iwlan0
post-down killall -9 wpa_supplicant
    ...
    ...
```

Für die Eingabe von PIN/Passwort und die Verwaltung muss nach Start des Interfaces *wpa_gui* bzw *wpa_cli* verwendet werden.

3.3.2 802.1X (Ethernet)

Auch ein Ethernet Anschluss kann bei geeigneten, managebaren Routern und Switches über 802.1x abgesichert werden. Auch hier wird wieder ein Radius-Server benötigt.

3.3.2.1 (GUI) Network-Manager -wired

Die Einstellungen im Network-Manager unterscheiden sich fürs Kabelnetzwerk nur durch den Namen, der entsprechende Tab heißt '802.1x-Sicherheit'. Ich verweise auf **3.3.1.1**.

3.3.2.2 (NoGUI) wpa_supplicant -wired

Die Einstellungen sind ähnlich der in **3.3.1.2** , es muss aber eine anderer Name für die Konfigurationsdatei verwendet werden, da diese Einträge nicht zusammen mit den Einstellungen für Wireless Geräte stehen können.

/etc/wpa_supplicant-eth.conf

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=users
pkcs11_engine_path=/usr/lib/engines/engine_pkcs11.so
pkcs11_module_path=/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
update_config=1
ap_scan=0

network={
        key_mgmt=IEEE8021X
        eap=TLS
        identity="User"
        engine_id="pkcs11"
        key_id="1:a4082239dcfxxxxxxxxx40467170cc1c10f7fd3"
```

```
        cert_id="1:a40822xxxxxxxxxxxxxxxxxx170cc1c10f7fd3"
        ca_cert_id="1:ef47e5xxxxxxxxxxxxxxx0192d725eb0e54fc3af"
        engine=1
    eapol_flags=0
        disabled=1
}
```

und auch hier die Version für eine .p12 Datei (Sie muss vorher wie unter **3.3.1.1** beschrieben in .pem Dateien aufgeteilt werden)

/etc/wpa_supplicant-eth.conf

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=users
update_config=1
ap_scan=0

network={
        key_mgmt=IEEE8021X
        eap=TLS
        identity="User"
        private_key="/etc/cert/<name>key.pem"
        client_cert="/etc/cert/<name>cert.pem"
        ca_cert="/etc/cert/<name>ca.pem"
    eapol_flags=0
        disabled=1
}
```

Der Eintrag in der **/etc/network/interfaces**

/etc/network/interfaces

```
    ...
    ...
iface ethX inet dhcp
pre-up wpa_supplicant -B -Dwired -c /etc/wpa_supplicant-eth.conf -iethX
post-down killall -9 wpa_supplicant
    ...
    ...
```

Und die Verwaltung erfolgt ebenfalls über *ifup/ifdown* und *wpa_gui/wpa_cli*

3.3.3 OpenVPN

3.3.3.1 (GUI) Network-Manager -openvpn

Falls noch nicht installiert, wird das Paket **network-manager-openvpn** benötigt. Es stehen dann in der network-manager Konfiguration auch die Tabs für openvpn zur Verfügung. Die Einträge dort erfolgen unter dem Punkt `Sicherheit` wie bereits in **3.3.1.1** beschrieben.

3.3.3.2 (NoGUI) wpa_supplicant -openvpn

Es besteht selbstverständlich auch die Möglichkeit, OpenVPN über die Textdateien zu konfigurieren und zu starten. Für den Einsatz von Smartcards muss auf diese Möglichkeit zurückgegriffen werden.

/etc/openvpn/<Name-der-Konfigurationsdatei>.conf

```
client
dev tun
proto udp
remote xxxxx.balista.xx 1196
pkcs11-providers  /usr/lib/opensc-pkcs11.so
pkcs11-protected-authentication 1 /usr/lib/opensc-pkcs11.so
ca /home/rwald/.pkcs11/CAcert_chain.pem
cipher BF-CBC
comp-lzo
persist-tun
persist-key
key-method 2
keepalive 10 120
reneg-sec 0
# -Zeilenanfang- alles in einer Zeile!
pkcs11-id
'EnterSafe/PKCS\x2315/2443151516111010/LUG\x2DBalista\x20Testcard\
x281\x29\x20\x28User\x20PI/EAC901Fxxxxxxxxxx2AAD6579B43EBC4E284E3C
6'
# -Zeilenende-
# -Zeilenanfang-
#pkcs11-id
'EnterSafe/PKCS\x2315/2958054713181210/LUG\x2DBalista\x20Testcard\
```

```
x282\x29\x20\x28User\x20PI/E476CE814Cxxxxxxxxxx726EED89B3A6B5D5C4A
4'
# -Zeilenende-
```

Die einzutragende **id** wird über den Befehl
openvpn --show-pkcs11-ids /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
ermittelt und wie oben gezeigt eingetragen. Es kann nur jeweils eine Karte (eine **id**) aktiv sein. Falls mehrere Karten zum Einsatz kommen, braucht jede eine angepasste Konfiguration. Zu beachten sind die Einträge, die mit dem OpenVPN-Server übereinstimmen müssen: UDP oder TCP, Portnummer und Cipher. Der Eintrag `reneg-sec 0` sorgt dafür, dass keine erneute Abfrage der PIN nach 1 Stunde (default-Wert) erfolgt, allerdings muss diese Einstellung auch auf dem Server vorhanden sein, es gilt die kleinste Zeiteinheit beider Beteiligten.

Für eine .p12 Datei vereinfacht sich die Konfiguration:

/etc/openvpn/<Name-der-Konfigurationsdatei>.conf

```
client
dev tun
proto udp
remote xxxxx.balista.xx 1196
pkcs12 <Pfad-zur-Datei>/<certname>.p12
ca /home/rwald/.pkcs11/CAcert_chain.pem
cipher BF-CBC
comp-lzo
persist-tun
persist-key
keepalive 10 120
```

Gestartet wird die Verbindung in einem Terminal, das bis zum Ende der Verbindung geöffnet bleibt, (mit root-Rechten, da das tun/tap device angelegt wird)
(sudo) openvpn --config <Pfad-und-Name-der_Konfig>.conf
In diesem Terminal wird dann auch nach der PIN bzw. der Passphrase gefragt.

3.3.4 ssh

Normalerweise benutzt ein User für ssh-Verbindungen ein mit ssh-keygen erstelltes Schlüsselpaar. Es ist ohne weiteres möglich, auch das bereits vorhandene Zertifikatspaket (.p12) dafür zu nutzen.

```
openssl pkcs12 -in <certname>.p12 -nocerts -nodes | openssl rsa > \
id_rsa
```

erstellt den privaten Schlüssel. '

```
openssl pkcs12 -in <certname>.p12 -clcerts -nokeys | openssl x509 \
-pubkey -noout | ssh-keygen -f /dev/stdin -i -m PKCS8 | tee       \
id_rsa.pub > /dev/null
```

den öffentlichen Schlüssel (im ssh-Format). Dieser wird jeweils bei den Servern, an die man sich als <username> anmelden möchte, unter **<userhome>/.ssh/authorized_keys** hinterlegt.

Wenn die Schlüssel auf einer Smartcard benutzt werden sollen, vereinfacht sich die Erstellung des öffentlichen Schlüssels zu

```
pkcs15-tool –list-certificates
```

dort dann die gewünschte **id** merken, zu der auch der private Schlüssel vorhanden ist, dann mit

```
pkcs15-tool --read-ssh-key <id> > id_rsa.pub
```

den öffentlichen Schlüssel (ssh-konform) ausgeben. Für die zertifikatsbasierte Anmeldung an einem Server, auf dem der öffentliche Schlüssel hinterlegt ist, genügt es, wenn entweder der private Schlüssel in **~/.ssh/id_rsa** gespeichert ist (keine weiteren Parameter beim Aufruf nötig), oder der ssh-Aufruf mit dem Parameter **-I /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so** erfolgt. Dann wird die Authentifizierung direkt mit dem privaten Schlüssel auf der Karte vorgenommen.

3.4 Authentifizierung lokal

Jeder Benutzer authorisiert sich im Regelfall mit Benutzername und Passwort (login, su, sudo, screensaver usw.), die Anwendung einer .p12 Datei würde die Sicherheit gegen Angriffe auf das Passwort nicht verändern, sie gebraucht selber eines. Nur die Verwendung eines Schlüssels auf einer Smartcard schafft hier Abhilfe. Alle privaten Schlüssel auf einer Smartcard können nicht ausgelesen und/oder kopiert werden. Die PIN der Karte schützt vor unberechtigtem Zugriff, ein Brute-Force-Attacke ist wegen der geringen Anzahl der Versuche (im Regelfall 3x) praktisch nicht möglich. Die sogenannte 2-Faktor Authentifizierung erhöht die Sicherheit gegen den Missbrauch des Rechners und der Daten erheblich (Verschlüsselung der Daten mit Hilfe der Smartcard). Die beste Stelle, an der die Änderung der Authentifikationsmethode eingefügt wird, ist die Datei **common-auth** im Verzeichnis **etc/pam.d/**, sie wird bei allen anderen Programmen im letztgenannten Verzeichnis eingebunden und ist damit überall wirksam.
Wir benötigen das Paket libpam-pkcs11

```
apt-get install libpam-pkcs11
```

und die Konfiguration einiger Dateien.

Zunächst kopieren wir die mitgelieferte Beispieldatei und passen diese dann an unsere Wünsche an.

```
mkdir /etc/pam-pkcs11

cp /usr/share/doc/libpam-pkcs11/examples/pam_pkcs11.conf.example.gz \
/etc/pam_pkcs11/pam_pkcs11.conf.gz

uncompress /etc/pam_pkcs11/pam_pkcs11.conf.gz
```

/etc/pam_pkcs11/pam_pkcs11.conf

```
    ...
    ...
#use_mappers = digest, cn, pwent, uid, mail, subject, null;
# ZEILE AUSKOMMENTIEREN - MIT NÄCHSTER ZEILE ERGÄNZEN!
use_mappers = openssh, null;
    ...
    ...
```

Die Konfigurationsdatei sucht die opensc-pkcs11 library im falschen Verzeichnis. Am Besten ist es einen Link zu setzen, abhängig vom tatsächlichen Ort (32bit vs. 64Bit)

```
ln -s /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so   \
/usr/lib/opensc-pkcs11.so
```

Die Authorisierung über den ssh-Schlüssel erfordert den Eintrag des öffentlichen ssh-Schlüssels in eine **authorized_keys** Datei, sie befindet sich jeweils im **HOME/.ssh** -Verzeichnis des jeweiligen Users. Dort sind alle öffentlichen Schlüssel verzeichnet, deren Besitzer des passenden privaten Schlüssels sich unter dem Usernamen anmelden können. Die Befehle sind bereits unter **3.3.4 Absatz 2** aufgeführt. Nun müssen noch die Verzeichnisse **/etc/pam-pkcs11/cacerts/** und **/etc/pam_pkcs11/crls** erstellt werden

```
mkdir /etc/pam_pkcs11/cacerts && mkdir /etc/pam_pkcs11/crls
```

In das cacerts Verzeichnis müssen die CAcert Root Dateien, die bereits in **/etc/dirmngr/trusted-certs** vorhanden sind

```
cp /etc/dirmngr/trusted-certs/* /etc/pam_pkcs11/cacerts/
```

dann müsen dort noch Hash-Links erstellt werden

```
cd /etc/pam_pkcs11/cacerts && pkcs11_make_hash_link
```

Nach diesen Vorarbeiten wird jetzt die Datei **common-auth** bearbeitet und damit die Authentifikation per Smartcard fertiggestellt. Unmittelbar nach Bearbeitung und Speicherung dieser Datei kann ein Test mit z.B. *su* erfolgen. Wenn alles funktioniert, kann die Möglichkeit einer Passworteingabe völlig ausgeschaltet werden. Das sollte nur erfolgen, wenn alles stabil und problemlos läuft.

/etc/pam.d/common-auth

```
# an den Anfang der Datei einfügen...
auth sufficient  pam_pkcs11.so   config_file=/etc/pam_pkcs11/pam_pkcs11.conf
# ermöglicht bei Problemen weiterhin die Passworteingabe
# Die obere Zeile auskommentieren und die nachfolgende aktivieren wenn nur
# Smartcard-Authentifizierung möglich sein soll.
# -Zeilenanfang- nur eine Zeile!
# auth     [success=done new_authtok_reqd=ok ignore=ignore default=die]
    pam_pkcs11.so   config_file=/etc/pam_pkcs11/pam_pkcs11.conf
# -Zeilenende-
    ...
    ...
```

3.5 Verschlüsselte Daten

Es gibt etliche kryptografische Möglichkeiten, Daten zu verschlüsseln. Ich möchte hier nur Möglichkeiten vorstellen, die mit einer Smartcard realizierbar sind.

3.5.1 cryptsetup-luks

Debian GNU/Linux, und wahrscheinlich viele andere Distributionen, bietet unter **/lib/cryptsetup/scripts** eine Reihe von Scripten an, die auf der Standardausgabe eine Passphrase bereitstellen, um ein verschlüsseltes Device zu öffnen. Für meine Kombinationen aus Smartcard/-lesern ist **decrypt_opensc** geeignet. Leider ist die Originalversion, die im Paket **cryptsetup** enthalten ist, mit einem Fehler behaftet, der die Nutzung des Scripts verhindert. Hier sind sich die Entwickler wohl nicht einig, wer für das Problem zuständig ist. Die Nutzung des privaten Schlüssels auf der Karte wird mit

```
/usr/bin/pkcs15-crypt --decipher --input $1 --pkcs1 --raw ...(...)
```

durchgeführt. Dieses Programm (**pkcs15-crypt**) stammt aus dem Paket **opensc** und berechnet aus einem Startwert (Parameter $1, in der Regel eine Datei) das mit dem privaten Schlüssel behandelte Ergebnis. Der eingetragene Parameter --pkcs1 soll den Eingabewert auf eine passende Länge zur vorhandenen Schlüssellänge bringen, sogenanntes paddding, ist aber funktionsunfähig und verhindert eine Ausgabe. Die einzige mir bekannte Möglichkeit ist es, diesen Parameter zu entfernen und nur eine Startdatei mit passender Länge zu benutzen (exakte Länge: 128/256 Byte bei 1024/2048 bit Schlüsseln). Dazu habe ich mir einmalig eine Startdatei (256 Byte lang) mit

```
dd if=/dev/urandom of=/boot/key/home bs=256 count=1
```

erzeugt. Den Speicherort habe ich in die boot-Partition gelegt, weil sie immer unverschlüsselt vorhanden sein muss, um das System zu starten.

Die korrigierte Version des Scripts decrypt_opensc ist nun einsatzfähig, verschlüsselte Partitionen, jedoch ohne die root-Partition, zu bearbeiten. Es muss z.B. für eine verschlüsselte Home-Partition folgender Eintrag in /etc/crypttab erfolgen

```
# <target name>   <source device>         <key file>  <options>
<target:home_crypt>     UUID=eXXXXXf7-2573-4fd0-b4d1-2XXXXX281d34 /boot
/key/home   luks,keyscript=/lib/cryptsetup/scripts/decrypt_opensc
```

Ich habe ein Script geschrieben, um die Handhabung von cryptsetup (mit luks-Erweiterung) zu vereinfachen, insbesondere die Erstellung und Verwaltung von Passphrasen und Schlüsseln auf Smartcards. Bezugspunkt ist immer die Startdatei /boot/key/home. Die Erstellung von Crypto-devices ist nicht auf Festplatten beschränkt, sondern funktioniert mit allen Datenträgern, wie z.B. USB-Sticks, USB-Festplatten, SD-Karten. Das Script ist downloadbar unter http://wiki.lug-balista.de/lib/exe/fetch.php/balista:projekte:add-luks-key.sh.gz

Für die 'fast' vollständige Verschlüsselung des Rechners (root-Partition) sind noch weitere Ergänzungen notwendig, da auch hier die vorhandenen Scripte fehlerhaft sind. Wichtig ist die vollständige Einbindung aller erforderlichen Programme und Bibliotheken in der initram, damit bereits beim Start Leser und Smartcard gefunden werden und benutzbar sind. Folgende Änderungen sind vorzunehmen: Die folgenden Dateien sind von **/usr/share/initramfs-tools/…** nach **/etc/initramfs-tools/…** zu kopieren.

…/hooks/cryptopensc

```
Änderungen laut diff
62,69c62
< cp -L /usr/lib/x86_64-linux-gnu/libpcsclite.so.1 ${DESTDIR}/usr/lib/x86_64-linux-gnu/libpcsclite.so.1
< cp -L /lib/x86_64-linux-gnu/libusb-1.0.so.0 ${DESTDIR}/lib/x86_64-linux-gnu/libusb-1.0.so.0
< cp -L /lib/x86_64-linux-gnu/libusb-0.1.so.4 ${DESTDIR}/lib/x86_64-linux-gnu/libusb-0.1.so.4
< cp /etc/libccid_Info.plist ${DESTDIR}/etc/libccid_Info.plist
< cp /usr/sbin/pcscd ${DESTDIR}/sbin/pcscd
< cp -r /etc/reader.conf.d ${DESTDIR}/etc/reader.conf.d
< cp -L /lib/x86_64-linux-gnu/libgcc_s.so.1 ${DESTDIR}/lib/x86_64-linux-gnu/libgcc_s.so.1
< cp /usr/bin/killall ${DESTDIR}/usr/bin/killall
---
> cp /usr/lib/x86_64-linux-gnu/libpcsclite.so.1 ${DESTDIR}/usr/lib/x86_64-linux-gnu/libpcsclite.so.1
```

…/scripts/local-bottom/cryptopensc

```
Änderungen laut diff
30,32c30
< #start-stop-daemon --stop --quiet --pidfile /var/run/pcscd.pid --name pcscd
< /usr/bin/killall -9 pcscd
< sleep 5
---
```

```
> start-stop-daemon --stop --quiet --pidfile /var/run/pcscd.pid --name pcscd
```

Erst nach diesen Änderungen kann auch die root-Partition verschlüsselt werden. Bei einem Kernel Update werden die Dateien in /etc nicht überschrieben und bleiben in Funktion.

3.5.2 truecrypt

Die Linux-Variante von truecrypt ist bereits auf die Nutzung von Smartcards vorbereitet. Der Schlüssel wird nativ als Datenblock auf der Karte gespeichert. Die Bedienung ist einfach, solange nur ein einfaches Keyfile benutzt wird. Ein oder mehrere Keyfiles lassen sich auch auf bestimmten Smartcards speichern. Die Einschränkung besteht darin, dass truecrypt kein asymmetrisches Schlüsselpaar verwendet, sondern einen einfachen Datenblock. Dieser lässt sich beim Speichern zwar mit einer PIN schützen, jedoch ist dieser nach Eingabe der PIN wieder exportierbar, anders als die auf der Karte gespeicherten privaten Schlüssel. Truecrypt lässt alternativ/zusätzlich die Verwendung von Passwörtern/Passphrases zu. Bei dem Spezialfall Hidden Partition ist es nicht möglich, nur mit einen einzigen Datenblock sowohl die normale wie auch die hidden Partition zu öffnen. Die Bedienung mit mehreren Datenblöcken auf der Karte ist umständlich und fehlerträchtig. Ob der Einsatz einer Smartcard sinnvoll ist, muss jeder selber entscheiden.

4 Servereinstellungen

Hier werden exemplarisch Anwendungen und Einstellungen erwähnt, die für die Administratoren interessant sind.

4.1 apache Webserver

Für die Client-Authentifikation muss der Webserver eingerichtet werden. Vorausgesetzt ist erst einmal eine korrekte Auslieferung von SSL-verschlüsselten Webseiten (HTTPS). Ergänzt wird z.B. die **/etc/apache2/sites-available/default-ssl** folgendermaßen

/etc/apache2/sites-available/default-ssl

```
    …
    …
# Certificate chain for test of  client certificate
SSLCACertificateFile /etc/apache2/Cacert_chain.pem
# individuelle Restriktionen werden in der .htaccess im jeweiligen Verzeichnis
eingestellt. Required verhindert SSL Kommunikation ohne Client-Zertifikat.
SSLVerifyClient optional
SSLVerifyDepth 10
# alle Zertifikatdaten werden für Scripte und Programme auf dem Webserver
verfügbar gemacht.
SSLOptions +StdEnvVars +ExportCertData
```

Eine Beispiel .htaccess

.htaccess

```
SSLRequireSSL
SSLVerifyClient optional
SSLVerifyDepth 2
SSLOptions +StdEnvVars +ExportCertData
SSLCACertificateFile /home/www/balista/certs/cacertrootchain.pem
SSLRequire ( %{SSL_CLIENT_S_DN_O} in {"LUG-Balista Hamburg e.V."} )
<Files env.cgi>
RewriteEngine on
RewriteCond %{SSL_CLIENT_S_DN_O} !^LUG-Balista\ Hamburg\ e\.V\.$ [OR]
RewriteCond %{SSL_CLIENT_V_REMAIN} <1
#RewriteCond %{SSL_CLIENT_M_SERIAL} !^(0103FC|0101F1|0114AF|0100AF)$
RewriteRule .* /env.failed.html [L]
</Files>
```

4.1.1 Wordpress

Es existiert ein Plugin **HTTP Authentication**, dass die Authorisierung vom Apache Webserver übernimmt. Wenn Wordpress darüber hinaus leicht angepasst wird, kann die normale Anmeldeseite abgeschaltet werden. Diese Änderungen sind speziell an den Anwendungsfall anzupassen. Wenn das automatische Anlegen neuer Benutzerkonten eingeschaltet ist, wird mit dem erstmaligen Anmelden mit Zertifikat der Name aus diesem entnommen und mit den voreingestellten Rechten der Benutzer angelegt. Im Hintergrund wird zwar ein Passwort angelegt, es ist aber nicht in Funktion und wird nirgends angezeigt.

4.1.2 Dokuwiki

Auch hier gibt es eine Ergänzung, dokumentiert und downloadbar unter
https://www.dokuwiki.org/auth:smartcard
Die User, die sich mit Zertifikat/Smartcard anmelden sollen, müssen vorher administrativ angelegt worden sein, dabei ist zwar ein Passwort anzugeben, aber es wird nicht weiter verwendet. Der User bekommt die notwendigen Gruppenrechte eingetragen und dazu den Gruppeneintrag seiner Zertifikats-Seriennummer. Die Authentifizierung erfolgt also über die Seriennummer, diese darf nur einmal im Benutzersystem vorhanden sein. Als Username wird dann der dazu passende angelegte Name verwendet. Zusätzlich kann eine HTTP Authentifikation über den Apache vorgenommen werden, um nicht passende Zertifikate vorher auszufiltern.

4.2 Radius-Server

Radius, eine Kurzform für Remote Authentication Dial In User Service, ist ein Übertragungsprotokoll, welches ein gleichnamiger Server zur Kommunikation mit anderen Rechnern benutzt, um dort User zu authentifizieren und ihnen entsprechende Rechte zuzuteilen. Eine der möglichen Authentifizierungsarten ist EAP-TLS, für diese ist die Nutzung von Client Zertifikaten zwingende Voraussetzung. Als Demonstrationsobjekt habe ich einen ASUS-Router mit aufgespieltem OpenWRT-System. Gleichzeitig läuft darauf ein Radius-Server und ein OpenVPN-Server. Praktisch alle modernen Wlan-Router sind auf WPA Enterprise einstellbar und ermöglichen die

Angabe des Namens/der IP-Adresse eines Radius-Servers und eines Passworts. Die installierte Version eines freeradius-Servers ist mit einem Bündel vorkonfigurierter Einstellungen in gut dokumentierten Dateien versehen. Anpassungen in dessen Konfiguration beschränken sich auf die **clients.conf**, wo erlaubte Clients mit ihren IPs und den dazugehörenden Passwörtern gespeichert werden. Beispiel:

/etc/freeradius/clients.conf

```
        …
        …
client ip.ip.ip.ip {
        secret = xxx
       shortname = <Netzwerk- od. Gerätename>
}
```

und dann muss noch die eap.conf angepasst werden

/etc/freefadius/eap.conf

```
        …
tls {      <hier werden Servercert und -key und CAs von Server und
Client eingetragen>
        …
}
        …
verify {
         ...
client = "/etc/freeradius2/bin/check_client_cert.sh %
{TLS-Client-Cert-Filename}"
        …
}
```

Die Prüfung des Zertifikats auf bestimmte Eigenschaften kann durch die Wahl der Verify-Methode erreicht werden, hier die Prüfung auf den Namen der E-Mail Domain

/etc/freeradius2/bin/check_client_cert.sh

```
#!/bin/sh
email=`openssl x509 -email -noout -in $1`
echo $email > /tmp/email
grep xxxxxx-wald.de /tmp/email
```

Im Ergebnis werden alle CAcert-Zertifikate mit dem Eintrag einer bestimmten E-Mail Domainadresse akzeptiert.

4.3 OpenVPN-Server

Eine Serverinstallation kommt im Desktopbereich mit einer Reihe von Beispielkonfigurationen, bei OpenWRT ist das schon aus Platzgründen nicht möglich. Für individuelle Lösungen reicht die grafische LUA-Oberfläche des ASUS-Routers nicht aus. Die Alternative ist die native Bearbeitung der Konfiguration. Hier setze ich jetzt die grundsätzlichen Einstellungen (Adressen, Routen, Root-Zertifikate, ServerCert und Key usw.) voraus. Interessant ist der Teil der Verifikation der Clients mit CAcert Zertifikaten, der erfolgt ähnlich wie beim Radius-Server durch ein verify-Script (Speicherort und Aufruf stehen in der Konfigurationsdatei server.conf.

```
…
tls-verify /var/ipcop/ovpn/verify
…
```

/var/ipcop/ovpn/verify

```
#!/bin/sh
var1=$tls_id_0
var2=$tls_serial_0
var5=${var1##*CN=}
var6=${var5%%/*}
if [ $1 -eq 0 ]; then
echo "$var6,$var2" >> /tmp/vars
clientallowed=`/bin/grep -ic "$var6,$var2"
/etc/openvpn/ovpnallowed`
echo "allowed:$clientallowed" >> /tmp/vars
  if [ "$clientallowed" = "0" ]; then
      exit 1
  fi
fi
exit 0
```

Hier muss eine Datei ovpnallowed mit zugelassenen Zertifikaten nach dem Schema <Name>, Seriennummer(dezimal)> erstellt und gepflegt werden. Elegantere Methoden waren entgegen der Dokumentation nicht erfolgreich. Variable, die Teile des Zertifikats enthalten sollten, waren immer leer.

5 Hinweise und Bedienung

Diese Anleitung hat keinen Anspruch auf Vollständigkeit. Sie soll einen Überblick über den gegenwärtigen Stand – Frühjahr 2014 – geben. Die Entwicklung geht weiter, und neue Produkte kommen, die ältere vom Markt drängen. Ich hoffe, dass eine größer werdende Menge an Anwendern dieser Techniken für eine entsprechende Nachfrage bei den Herstellern führt und der Markt der freien und offenen Software zukünftig mehr beteiligt sein wird. Schon die Anwendung von Zertifikaten in Dateiform ist ein Einstieg, nur einen Schlüssel/eine Passphrase zu benötigen. Anders als beim OpenID-Verfahren bleiben die Nutzer aber Herr aller ihrer Daten.

Die Zertifikate auf einer Smartcard haben neben den aufgezeigten Vorteilen auch Nachteile, die nicht verschwiegen werden sollen.

- Höhere Einstiegskosten (Smartcard und -leser)
- Kompatibilität mit freier und offener Software teilweise mangelhaft
- Nicht jede Kombination aus funktionierenden Smartcards und -lesern funktioniert miteinander
- Firefox und Thunderbird müssen bei jeder Unterbrechung der Leitung zum Leser neu gestartet werden. Das wird uns als Sicherheitsfeature verkauft.
- Einige Programme und Scripte sind fehlerhaft und müssen korrigiert werden.
- Kenntnisse auf der Kommandozeile sind dafür erforderlich.

6 Getestete Kombinationen Smartcards

	SCR 3500	Gemalto ID Bridge	SPR 532	Gemalto PinPad	ACR 83 Pineasy
Feitian PKI	Ok	–	Ok	Ok	(1)
Aventra MyEID	Ok	Ok	Ok	Ok	Ok
Athena ASEPCOS (2)	Ok	Ok	(1)	(1)	(1)
SC_HSM (3)	Ok	Ok	Ok	(1)	(1)

Ok – Karte wird von Leser erkannt und wird vollständig unterstützt

(1) – Karte wird vom Leser erkannt und unterstützt, die Eingabe der PIN über Pinpad wird nicht unterstützt und muss in der Konfiguration (**/etc/opensc/opensc.conf**) ausgeschaltet werden.

– Karte von Leser nicht erkannt und ansprechbar.

(2) – Schlüsselfunktionen ok, aber keine normalen Datenblöcke speicherbar.

(3) – kann nicht mit pkcs15-init arbeiten. Schlüsselimport fehlerhaft.