Zürcher Hochschule Winterthur

Diplomarbeit

Linux Netzwerk-Login mit RSA Smartcard

Dozent: Dr. Andreas Steffen

Autoren: Martin Sägesser & Mario Strasser

September/Oktober 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

Zusammenfassung

Übliche Login-Verfahren mit einem Passwort, sogenannte *One Factor Authentication Systems*, genügen aktuellen Sicherheitsanforderungen nicht mehr. Grund dafür sind weniger die technischen Mängel, als die Bequemlichkeit der Benutzer. Diese sind oftmals nicht gewillt, sich sichere Passwörter auszudenken, zu merken und diese periodisch zu ändern. Gebräuchlich ist es auch, sich die zu komplizierten und daher sicheren Passwörter aufzuschreiben und neben dem Bildschirm zu befestigen. Ebenfalls problematisch ist die Übertragung des Passwortes im Klartext durch die heute gängigen Netzwerk-Protokolle.

Mehr Sicherheit bieten die *Two Factor Authentication Systems*, bei denen der Benutzer nicht nur das Passwort wissen, sondern auch noch eine Art Schlüssel besitzen muss. Als weitere Verbesserung kann zusätzlich ein persönliches Merkmal abgefragt werden. Denkbare Verfahren sind Iris-Scan, Fingerabdruck- oder Stimmenerkennung (*Three Factor Authentication Systems*).

In einer vorgängigen Projektarbeit erarbeiteten wir ein *Two Factor Authentication System* für Linux, welches die lokale Anmeldung an einem Rechner mit einer RSA-Smartcard als Schlüssel erlaubt. Diese Lösung wurde im Rahmen unserer Diplomarbeit netzwerkfähig, um heutigen Ansprüchen zu genügen.

Es wird für jeden Benutzer ein Zertifikat mit zugehörigem Schlüsselpaar erstellt und auf einer Smartcard gespeichert. Beim Anmeldevorgang wird als Erstes ein Passwort abgefragt, womit die Verschlüsselungs-Funktionen der Smartcard freigeschaltet werden. Danach wird das gespeicherte Zertifikat des Benutzers von der Karte ausgelesen. Dieses kann anhand des lokal vorhandenen Root-Zertifikats und einer netzwerkweit verfügbaren *Certificate Revocation List* (CRL) auf seine Gültigkeit geprüft werden. Anschliessend wird eine vom Computer erstellte Zufallszahl als Challenge zur Karte gesendet, von ihr mit Hilfe des privaten Schlüssels signiert und zurückgesandt. Das Resultat wird mit dem öffentlichen Schlüssel aus dem Zertifikat entschlüsselt und mit der ursprünglichen Zufallszahl verglichen. Stimmen diese überein, gilt der Benutzer als authentisiert.

Das auf der Smartcard gespeicherte Passwort und der private Schlüssel können nach ihrer Erstellung nie mehr ausgelesen werden. Sämtliche kryptografischen Funktionen werden von der Karte erst nach Eingabe des Passworts ausgeführt. Dadurch stellt ein Kartenverlust kein Sicherheitsrisiko dar.

Für einen sicheren Remote-Zugang auf andere Netzwerk-Rechner passten wird das bestehende OpenSSH-Projekt unseren Bedürfnissen an. Anstelle von öffentlichen Schlüsseln werden nun X.509 Zertifikate versendet. Die Smartcard führt die zur Authentisierung notwendige Signierung aus.

Im Rahmen dieser Diplomarbeit entstanden neben dem für den Login zuständigen PAM-Modul auch ein Patch für das OpenSSH-Paket, sowie diverse Administrations- und Installationstools. Mit diesen ist eine einfache Inbetriebsetzung und Wartung des Login-Systems möglich. Für die zentrale Benutzer- und Gruppenverwaltung, sowie die Speicherung der CRL, benutzen die Tools einen LDAP-Server. Das gesamte Paket kann vom Internet heruntergeladen werden und steht unter *GNU Public License*.

Abstract

The common login procedures using a password, called *one factor authentication systems*, do not meet the security requirements of nowadays. This is not only caused by technical faults, but also by the convenience of the user who are often not willing to devise and remember safe passwords and change them periodically. It is also common to write down complex and therefore safe passwords and store them near the monitor. Also problematical is the transmission of the password in plain text by the today common network protocols.

Two factor authentication systems offer more security as the user has to own a kind of key in addition to remembering the password. As a further improvement, a personal attribute can also be verified. Possible techniques are iris scan, fingerprints or voice recognition (*three factor authentication systems*).

In a previous project we developed a two factor authentication system for linux. It allows to login to a computer using a RSA smartcard. We advanced this solution during our diploma thesis to enhance the system so that it fulfills all requirements and can be used on networks.

A certificate and its appropriate key pair are created for each user and stored on a smartcard. During a login procedure the password is asked for unlocking the encryption functions of the smartcard. If it was correct, the saved certificate of the user is read from the card. This is verified with the local stored root certificate and a network wide available *certificate revocation list* (CRL). If all these tests are passed, the computer sends a random number as a challenge to the card, which signs it with the private key and returns the computed value. The result is decrypted with the public key and compared with the original random number. If they match, the user is authenticated.

The password and the private key can never be readout again from the smartcard after their creation. All cryptographic functions are not available without the password. Therefore a lost card does not cause a security risk.

For a secure remote access we modified the existing OpenSSH project to our requirements. We now send X.509 certificates instead of public keys. The necessary signing to authenticate the user is done by the smartcard.

Apart of the PAM Module, which is responsible for the login procedure, several administration and installation tools were implemented. They make it easy to setup and maintain our login system. For the central user and group management, as well as distributing the CRL, the tools use a LDAP server. The changes made at the OpenSSH project are provided as a patch, which is also part of our package. This can be downloaded from the internet and is under *GNU Public License*.

Kapitel 1

Einleitung

1.1 Aufgabenstellung

Kommunikationssysteme (KSy)

Praktische Diplomarbeiten 2001 - Sna01/1

Linux Netzwerk-Login mit RSA Smartcard

Studierende:

- Martin Sägesser, IT3b
- Mario Strasser, IT3b

Termine:

- Ausgabe: Donnerstag, 6.09.2001, 16:00 im E523
- Abgabe: Montag, 29.10.2001, 12:00

Beschreibung:

In einer vorgängigen Projektarbeit wurde ein Linux PAM-Modul erstellt, das ein Login auf einen Einzelrechner mittels einer Smartcard ermöglicht. Die Authentisierung basiert auf dem bekannten RSA Public Key Algorithmus, wobei der Private Key die Smartcard nie verlässt und somit geschützt ist.

In der vorgeschlagenen Diplomarbeit soll das Konzept auf ein netzwerkfähiges Login erweitert werden. Durch die Speicherung des Public Key in Form eines X.509 Zertifikates auf der Smartcard soll sich ein Benutzer lokal auf einen beliebigen Netzwerkrechner einloggen können. Die Berechtigung wird durch den jeweiligen Host auf der Basis des CA Zertifikats und einer aktuellen Certificate Revocation List (CRL), die im Netz abrufbar

ist, verifiziert. Bei der Realisierung kann auf vorhandenem C-Source Code aus dem Linux IPsec Projekt *X.509 Trust Chain Unterstützung* aufgebaut werden.

Als zusätzliche Erweiterung könnte die Smartcard dazu verwendet werden, einem Benutzer den Remote-Zugriff auf einen beliebigen Netzwerkrechner via die sicheren Protokolle ssh, sftp und scp zu ermöglichen.

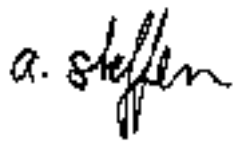
Ziele:

- Erstellung eines Linux Smartcard Konzepts für ein netzwerkfähiges, kryptografisch sicheres Login auf der Basis von RSA-fähigen Smartcards.
- Erstellung eines Linux Smartcard Konzepts für ein netzwerkfähiges, kryptografisch sicheres Login auf der Basis von RSA-fähigen Smartcards.
- Implementation dieses Konzepts durch Realisierung als Linux PAM-Modul.
- Dokumentation und Benutzeranleitung (HOWTO)
- Erweiterung des Linux Smartcard Konzepts auf kryptografisch sichere Remote-Zugriffe via ssh, sftp und scp.
- Falls vom geschätzten Aufwand und der Terminplanung her möglich, Implementation des sicheren Remote-Zugriffs auf der Basis von RSA-fähigen Smartcards.

Infrastruktur / Tools:

- Raum: E523
- Rechner: 2 PCs mit Dual-Boot: SuSE Linux / Windows NT 4.0
- Hardware: 2 Reflex 60 Smartcard-Reader, Cyberflex Access Smartcards

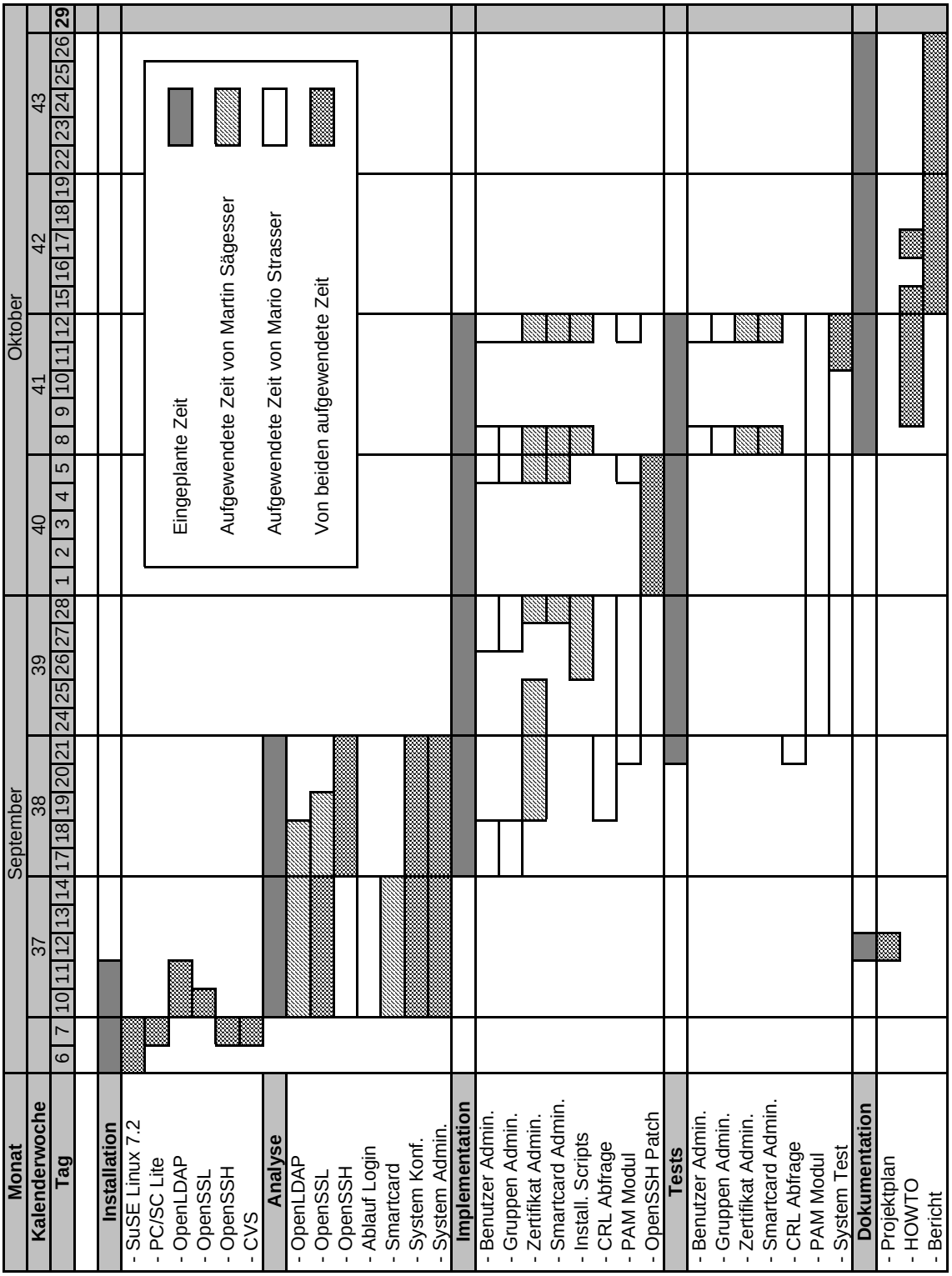
Winterthur, 6. September 2001



Dr. Andreas Steffen

1.2 Projektplan

Der unten abgebildete Projektplan zeigt den geplanten und den wirklichen Projektverlauf. Wir versuchten die Aufgaben gemäss unseren Fähigkeiten und Erfahrungen aus anderen Projekten aufzuteilen. Aus der Legende wird ersichtlich, wer für was verantwortlich war und welche Arbeiten gemeinsam ausgeführt wurden.



1.3 Typographische Konventionen

Zur besseren Lesbarkeit unserer Dokumentation haben wir uns an folgende Konventionen gehalten:

1. Funktionen und Argumente sind durch Fettdruck hervorgehoben: **funktion()**
2. Dateinamen und Verzeichnisse sind in monospace Schrift gesetzt: `/dir/file`
3. Codeauszüge im Text sind in kleiner, serifenloser Schrift und mit Syntaxhervorhebung abgedruckt:

```
/* example */
for (int i=0; i < 10; i++) {
  printf("This is number %d", i);
}
```

4. Quellenverweise werden durch einen eindeutigen Kürzel bezeichnet: [pcsc]
5. Verweise auf Webseiten sind nach dem Bezeichner in runden Klammern angegeben: Zürcher Hochschule Winterthur (http://www.zhwin.ch). In der PDF Version des Dokumentes sind die Bezeichner direkt anwählbar.

1.4 Übersicht

Das Dokument gliedert sich in sieben Kapitel mit folgendem Inhalt:

1. Einleitung Eine kurze Einführung in diese Arbeit.

2. Grundlagen Dieses Kapitel gibt eine kurze Einführung in die einzelnen Fachgebiete, die im Zusammenhang mit dieser Diplomarbeit stehen. Dem Leser soll es dadurch möglich sein, die nachfolgenden Ausführungen besser zu verstehen.

3. Analyse und Design Es werden die einzelnen Aspekte unserer Arbeit analysiert und mögliche Lösungen diskutiert.

4. Realisierung Hier werden die erstellten Applikationen und Module genauer beschrieben und auf Details ihrer Implementierung eingegangen.

5. Installation und Anwendung Der Inhalt dieses Kapitels entstammt dem HOWTO unserer Arbeit und ist daher in Englisch verfasst.

6. Tests Es wird aufgezeigt, wie beim Testen der erstellten Programme und des PAM-Moduls vorgegangen wurde und wo gegebenenfalls Probleme auftraten.

7. Schlussbemerkungen Zum Schluss wird ein Fazit gezogen und ein Dank an alle Beteiligten ausgesprochen.

Kapitel 2

Grundlagen

2.1 Name Service Switch

Früher wurden die System-Informationen durch die C Library direkt aus Dateien herausgelesen. Mit dem Aufkommen von neuen Diensten wie dem Network Information Service (NIS) oder dem Domain Name Service (DNS) musste eine flexiblere Lösung gefunden werden, die es erlaubt, neue Zugriffsmöglichkeiten hinzuzufügen.

Der von SUN entwickelte *Name Service Switch* () Standard ermöglicht dies, durch eine Auslagerung der eigentlichen Zugriffsfunktionen auf die Dienste in externe Module. In einer Konfigurationsdatei wird spezifiziert, welche Module für welchen Dienstzugriff zu verwenden sind. Durch die modulare Lösung können neue Möglichkeiten einfach hinzu programmiert werden.

Leider funktioniert das Ganze zum aktuellen Entwicklungszeitpunkt erst beim Auslesen von Informationen. Das Schreiben funktioniert deshalb nicht, weil sich das System noch nicht merkt, woher es die entsprechenden Daten empfangen hat und so natürlich auch nicht weiss, wo es etwas ändern muss.

2.1.1 Konfiguration

Für die Konfiguration des NSS-Dienstes dient die Datei `/etc/nsswitch.conf`. Hier kann festgelegt werden, wie und in welcher Reihenfolge auf die einzelnen Datenbanken zugegriffen werden soll. Eine solche Konfigurationsdatei könnte ungefähr so aussehen:

```
# /etc/nsswitch.conf
#
# Name Service Switch configuration file.
#

passwd:         ldap files nis
shadow:         files
group:          db files nis
```

```
hosts:          files nisplus nis dns
networks:       nisplus [NOTFOUND=return] db files
ethers:         nisplus [NOTFOUND=return] db files
protocols:      nisplus [NOTFOUND=return] db files
rpc:            nisplus [NOTFOUND=return] db files
services:       nisplus [NOTFOUND=return] db files
```

Die erste Spalte gibt die Datenbank an. Der Rest der Zeile besagt, in welcher Reihenfolge der Suchvorgang arbeiten soll. Wie man sieht, kann für jede Datenbank separat angegeben werden, wie verfahren werden soll. Die man page von `nsswitch.conf` gibt noch weitere Informationen, wie der NSS-Dienst konfiguriert wird.

2.2 Linux-PAM

Linux-PAM (Pluggable Authentication Modules for Linux) ist eine Sammlung von Authentifikations-Modulen, die eine gemeinsame (im [rfc86.0] festgelegte) Schnittstelle aufweisen. Sie erlauben es dem Systemadministrator gezielt festzulegen, wie die einzelnen Applikationen Benutzer authentisieren, Passwörter ändern oder ähnliches. Um zum Beispiel zu erreichen, dass ein Programm nicht mehr die lokale *passwd* Datei zur Überprüfung der Passworteingabe verwendet, sondern auf einen NIS-Server zugreift, wäre normalerweise eine Änderung und Neukompilation des Sourcecodes notwendig. Bei einem PAM-kompatiblen Programm genügt es, die zugehörige Konfigurationsdatei anzupassen. Dies wird durch eine Auslagerung der für die Authentifikation zuständigen Funktionen in separate Module erreicht. Eine zur Applikation gehörende Konfigurationsdatei legt fest, welche Module in welcher Reihenfolge abgearbeitet werden. Die Module ihrerseits können für mehr als eine Aufgabe konzipiert sein und zum Beispiel Funktionen zur Authentisierung des Benutzers, zur Passwortänderung oder auch für das Sessionmanagement bereitstellen. Ein PAM-Programm kümmert sich dann nicht mehr um das Wie, sondern ruft nur noch die zuständige PAM-Bibliotheksfunktion auf. Diese wiederum greift auf die dafür konfigurierten Module zurück.

Das PAM-System wurde ursprünglich von SUN entwickelt und ist neben Linux auch für Solaris, HP-UX und FreeBSD verfügbar.

2.2.1 Grundlagen der Konfiguration

Grundsätzlich gibt es zwei Möglichkeiten, um Linux-PAM zu konfigurieren: Eine ältere Variante mit nur einer Konfigurationsdatei (`/etc/pam.conf`) und die heute gängige, verzeichnisbasierte, mit einer Konfigurationsdatei pro Programm (unter `/etc/pam.d/`). Bei der ersten Variante bestimmt das sogenante *service-name* Feld in der Konfigurationsdatei für welches Programm die Einstellung gilt, bei der zweiten ist dies durch den Namen der Konfigurationsdatei festgelegt (z.B. `/etc/pam.d/login` für das Programm `login`) und das *service-name* Feld fehlt. Ansonsten sind die Konfigurationsdateien identisch und besitzen einen für die UNIX-Welt typischen Aufbau:

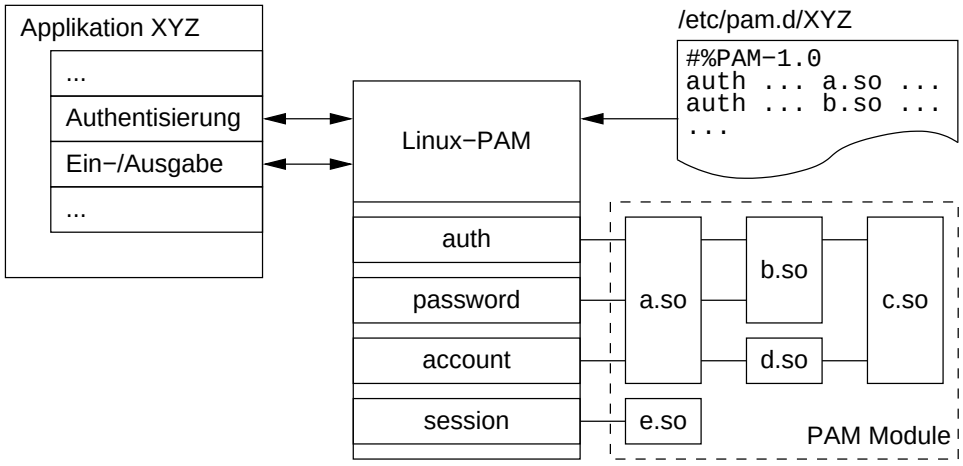


Abbildung 2.1: Organisation von Linux-PAM

- Jede Zeile entspricht einem Konfigurationseintrag für ein Modul folgender Form: `[service-name] module-type control-flag module-path arguments`
- Überlange Zeilen können mit '\' umgebrochen werden.
- Mit '#' beginnende Zeilen werden als Kommentar interpretiert und ignoriert.

Die einzelnen Felder haben folgende Bedeutung:

service-name Dieses Feld gibt an, zu welchem Programm der Eintrag gehört und entspricht normalerweise dem Programmnamen. Dies ist jedoch nicht zwingend und es können durchaus zwei unterschiedliche Programme auf ein und dieselbe Konfiguration zurückgreifen. Bei der verzeichnisbasierten Konfiguration entspricht dieses Feld dem Dateinamen und entfällt somit.

module-type definiert einen der folgenden vier Modul Typen:

- `auth`: Das Modul wird zur Authentifikation des Benutzers verwendet.
- `account`: Das Modul wird für das Account Managing verwendet und prüft zum Beispiel, ob sich ein Benutzer zu dieser Tageszeit anmelden darf.
- `password`: Das Modul wird zum Ändern des *authentication token* (meist ein Passwort) verwendet.
- `session`: Das Modul stellt Sessionmanagement-Funktionen zur Verfügung, welche unter anderem Umgebungsvariablen setzen oder Verzeichnisse mounten.

In der Konfiguration eines Programmes können für dieselbe Aufgabe mehrere Einträge (Zeilen) angegeben werden. Die entsprechenden Module werden dann zu einem Stapel zusammengefasst und in der Reihenfolge ihrer Definition abgearbeitet. Dem aufrufenden Programm wird am Schluss nicht das Resultat eines einzelnen Moduls, sondern ein für den ganzen Stapel repräsentatives Endergebnis zurückgegeben.

control-flag Das Flag gibt an, wie auf das Ergebnis eines Moduls reagiert werden soll und in wie weit davon das Endergebnis des betroffenen Modul-Stapels abhängt. Folgende Werte sind möglich:

- `required`: Der Erfolg dieses Moduls ist zwingend für ein positives Endergebnis. Bei einem Misserfolg werden aber dennoch alle folgenden `required` Module abgearbeitet.
- `requisite`: Auch hier ist der Erfolg dieses Moduls zwingend für ein positives Endergebnis. Bei einem Misserfolg wird die Verarbeitung abgebrochen und die folgenden Module nicht mehr ausgeführt.
- `sufficient`: Ist das Modul erfolgreich, ist auch das Endergebnis positiv und die Verarbeitung wird abgebrochen. Ansonsten wird mit dem nächsten Modul fortgefahren und das Endergebnis bleibt unbeeinflusst.
- `optional`: Das Modul wird ausgeführt, beeinflusst das Endergebnis aber nicht.

Ist das Endergebnis nicht eindeutig, weil z.B. nur `optional` Module definiert waren, wird der Aufruf als Misserfolg gewertet.

module-path Hiermit wird der Pfad zum eigentlichen PAM-Modul angegeben.

arguments Eine Liste von Argumenten, die an das Modul weitergegeben werden. Wird ein Argument nicht berücksichtigt, muss dies vom Modul mit `syslog` festgehalten werden.

2.2.2 PAM-Modul

Je nach dem, für welche Aufgaben ein PAM-Modul vorgesehen ist, muss es andere Funktionen bereitstellen. Es wird jedoch empfohlen, in allen Modulen alle Funktionen zu definieren und in den nicht unterstützten eine Fehlermeldung mittels `syslog` auszugeben. Dem Administrator wird so seine Arbeit erleichtert, da fehlerhafte Konfigurationen anhand der Meldungen identifiziert werden können. Die folgende Übersicht zeigt, für welche Aufgabe welche Funktionen zu implementieren sind und was sie genau tun müssen. Für eine detailliertere Beschreibung der möglichen Parameter und Rückgabewerte verweisen wir auf [pammod].

- Authentifikation (`auth`)

  `pam_sm_authenticate()` authorisiert den Benutzer, zum Beispiel durch Abfragen eines Passwortes.

  `pam_sm_setcred()` dient dem Setzen und Löschen der Anmeldeberechtigung. Ist zur Zeit aber noch nicht genauer spezifiziert und sollte entweder den gleichen Wert wie `pam_sm_authenticate()`, oder ein positives Ergebnis zurückgeben.

- Account Management (`account`)

  `pam_sm_acct_mgmt()` überprüft, ob sich ein Benutzer überhaupt anmelden darf oder ob z.B. schon eine maximale Anzahl Benutzer angemeldet sind.

- Session Management (`session`)

  `pam_sm_open_session()` wird zum Öffnen einer Benutzer-Session aufgerufen.

`pam_sm_close_session()` wird zum Schliessen einer Benutzer-Session aufgerufen.

- Passwort Management (`password`)

  `pam_sm_chauthtok()` dient dem Ändern des *authentication token* (Passwort o.ä.) und wird dazu zweimal hintereinander aufgerufen. Beim ersten Mal wird nur überprüft, ob überhaupt geändert werden kann (Netzwerkverbindung vorhanden etc.), und erst beim zweiten Mal wirklich geändert.

2.2.3 PAM Applikation

Damit eine Applikation die Vorteile der PAM-Module nutzen kann, muss sie mit der Bibliothek `libpam` verlinkt sein. Diese stellt eine Reihe von Funktionen bereit, um mit dem Linux-PAM System zu kommunizieren. Im Folgenden werden nur die wichtigsten kurz erläutert. Für eine detailliertere Beschreibung, die alle Funktionen umfasst, verweisen wir auf [pamapp].

`pam_start()` initialisiert das Linux-PAM System und sollte als erste Bibliotheksfunktion aufgerufen werden. Ihr wird der *service-name* übergeben, der angibt, welche Konfiguration zu dem Programm gehört. Wie bereits erwähnt, wird hier meist der Programmname verwendet. Des weiteren wird auch ein Konstrukt übergeben, das unter anderem die Adresse einer sogenannten Konvertierungsfunktion enthält, die an das Modul weitergegeben wird. Sinn dieser Funktion ist es, dem Modul eine Schnittstelle zur Daten Ein- und Ausgabe bereitzustellen. Dies hat den entscheidenden Vorteil, dass die Darstellung und Entgegennahme der Daten in den Händen der Applikation bleibt und das Modul nicht zu wissen braucht, ob es von einer Konsolen-, X11- oder Netzwerkapplikation benutzt wird.

`pam_end()` sollte als letzte Funktion der PAM-Bibliothek aufgerufen werden und beendet eine mit `pam_start()` geöffnete Session.

`pam_set_item()` dient dem Setzen einer den PAM-Modulen zugänglichen Variablen.

`pam_get_item()` dient dem Auslesen einer den PAM-Modulen zugänglichen Variablen.

`pam_authenticate()` authentisiert einen Benutzer mit Hilfe der dafür konfigurierten Module und liefert das Endergebnis des ganzen Modul Stapels zurück.

`pam_open_session(), pam_close_session()` öffnet bzw. schliesst eine Session.

`pam_acct_mgmt()` diese Funktion dient der Überprüfung, ob ein Benutzer sich anmelden darf, oder ob zum Beispiel sein Passwort abgelaufen ist.

`pam_chauthtok()` hiermit wird der *authentication token* eines Benutzers geändert. Wie bereits bei der Modulbeschreibung erwähnt, muss die Funktion zweimal hintereinander aufgerufen werden. Beim ersten Mal wird nur überprüft, ob der *authentication token* geändert werden kann. Ausgeführt wird die Änderung erst beim zweiten Mal.

2.3 Grundlagen zu Smartcards

2.3.1 Die verschiedenen Smartcard-Arten

Da es sehr viele verschiedene Anwendungsmöglichkeiten für Smartcards gibt, wurden im Laufe der Zeit immer mehr verschiedene Karten entwickelt.

Speicherkarten beinhalten nur elektronischen Speicher (meist EEPROM). Normalerweise werden hier nur das Löschen oder Schreiben in den Speicher bzw. in einen Speicherbereich von einer eher einfachen Sicherheitslogik überwacht. Es gibt aber auch Speicherkarten mit einer komplexeren Sicherheit. Diese können dann auch einfache Verschlüsselungen durchführen. Die Funktionalität dieser Karten ist meistens für eine spezielle Anwendung optimiert. Dadurch wird zwar die Flexibilität eingeschränkt, die Speicherkarten werden aber auch preisgünstiger. Typische Anwendungen sind vorbezahlte Telefonkarten oder die Krankenversicherungskarte in Deutschland.

Mikroprozessorkarten Diese Karten sind sehr flexibel einsetzbar. Im einfachsten Fall enthalten sie genau ein Programm, das auf eine spezielle Anwendung optimiert wurde und deshalb auch nur noch dafür einsetzbar ist. Moderne Chipkarten ermöglichen jedoch auch mehrere, verschiedene Anwendungen in eine einzige Karte zu integrieren. Es ist sogar möglich, die Anwendung erst nach der Auslieferung der Karte, also beim Kartenbenutzer, auf die Karte zu laden (JavaCards).

Kontaktlose Karten Wie es der Name schon sagt, besteht der grosse Vorteil dieser Karten darin, dass sie kontaktlos arbeiten. Dadurch müssen sie nicht mehr unbedingt in einen Kartenleser geschoben werden. Es existieren Systeme, die auf eine Entfernung von über einem Meter funktionieren. Dies ist vor allem bei Zugangskontrollen interessant, da man hier bei der Tür nicht mehr die Karte zücken muss. Die häufigste Anwendung ist das elektronische Ticket im öffentlichen Personenverkehr.

2.3.2 Smartcard Dateitypen

Obwohl moderne Chipkarten Mechanismen zur Identifizierung und Authentisierung haben, sind sie dennoch vor allem Datenspeicher. Dabei haben Chipkarten den Vorteil, dass der Zugriff auf einzelne Dateien an Bedingungen geknüpft sein kann.

Damit der Speicherverbrauch möglichst gering ist, wird normalerweise auf eine aufwändige Speicherverwaltung verzichtet. Gewisse ältere Karten können deshalb nach dem Löschen einer Datei den besetzten Speicherplatz nicht wieder freigeben. Die Art des benutzten Speichers spielt ebenfalls eine Rolle bei der Dateiverwaltung. Ein EEPROM kann nicht unbegrenzt beschrieben oder gelöscht werden. Deshalb wurden Dateiattribute entwickelt, um Informationen redundant und eventuell sogar korrigierbar abzuspeichern.

Im Unterschied zu den bekannten Dateien unter DOS oder Unix gibt es keine anwendungsspezifischen Dateien auf Smartcards. Es können nur die genormten Datenstrukturen verwendet werden.

Es gibt zwei grundlegende Kategorien von Dateien auf Smartcards. Verzeichnisdateien, *Dedicated Files* (DF) genannt und die *Elementary Files* (EF), welche die eigentlichen Nutzdaten enthalten. Die DFs stellen eine Art Ordner für weitere DFs oder EFs dar.

MF Das *Master File* stellt das Root-Verzeichnis dar. Nach einem Reset der Karte wird automatisch dieses File selektiert. Das MF ist ein Sonderfall des DF und stellt den gesamten vorhandenen Speicher auf der Smartcard dar. Es muss auf jeder Smartcard vorhanden sein.

DF Das *Dedicated File* ist eine Art Verzeichnis, in dem weitere Dateien zusammengefasst sein können. Ein DF kann also auch noch weitere DFs enthalten. Dabei ist die Schachtelungstiefe theoretisch nicht beschränkt. Allerdings führt der knappe Speicherplatz dazu, dass selten mehr als zwei Ebenen unter dem MF aufgebaut werden.

EF Das *Elementary File* enthält die Nutzdaten, die für eine Anwendung notwendig sind. Damit Speicherplatz auf der Karte gespart werden kann, besitzen EFs eine interne Dateistruktur. Dies ist der Hauptunterschied zu den Dateien auf PCs, bei denen die Filestruktur durch die benutzte Anwendung gegeben ist. EFs sind weiter noch in *Working EFs* und *Internal EFs* unterteilt.

Working EF sind alle Daten einer Anwendung, die von einem Reader gelesen oder geschrieben werden können, also für die äussere Welt bestimmt sind.

Internal EF sind interne Systemdaten, die vom Betriebssystem auf der Karte selbst verwaltet werden. Der Zugriff auf diese Dateien ist besonders geschützt, man kann also nicht darauf zugreifen.

2.3.3 Dateistrukturen der EFs

Jedes EF hat eine interne Struktur. Man kann die verwendete Struktur je nach Anwendungszweck individuell für jedes einzelne EF bestimmen. Durch die interne Struktur der Dateien kann man sehr schnell und gezielt auf die gewünschten Daten zugreifen.

transparent Diese Dateistruktur wird auch *binäre* oder *amorphe* Struktur genannt und ist eine blosse Aneinanderreihung von Bytes. Man greift auf die Daten byte- oder blockweise mittels eines Offset zu. Die Abbildung 2.2 zeigt den Aufbau dieser Dateistruktur.

Abbildung 2.2: Transparente Dateistruktur

linear fixed Diese Dateistruktur, die in Abbildung 2.3 auf der nächsten Seite gezeigt wird, ist eine Verkettung von gleich langen Datensätzen, Records genannt. Diese sind wiederum eine Aneinanderreihung von Bytes. Auf die einzelnen Datensätze kann beliebig zugegriffen werden. Allerdings kann man nicht einzelne Bytes neu

schreiben, sondern muss immer den ganzen Record aktualisieren. Der erste Datensatz hat immer die Nummer 1. Die grösste Recordnummer ist 'FE', also 254. Die maximale Länge der einzelnen Datensätze ist 254, da die Zugriffsbefehle für die Längenangabe nur ein Byte zur Verfügung stellen.

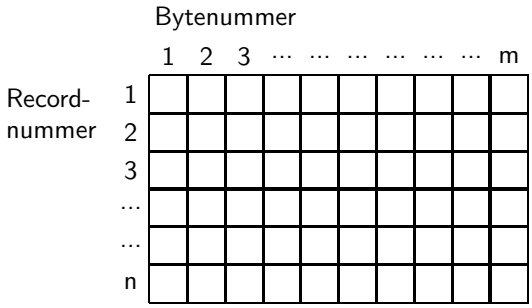


Abbildung 2.3: Linear fixed Dateistruktur

linear variable Im Gegensatz zu der *linear fixed Dateistruktur* können hier die einzelnen Records unterschiedliche Längen aufweisen (Abbildung 2.4). Damit kann bei Daten, die sehr unterschiedlich lange Datensätze haben, Speicher gespart werden. Allerdings braucht es für die Verwaltung der unterschiedlichen Längen ein zusätzliches Informationsfeld. Sonst ist der Aufbau und auch der Zugriff auf die einzelnen Records gleich wie bei der *linear fixed Dateistruktur*. Da die Verwaltung der unterschiedlichen Record-Längen zusätzlichen Programmcode benötigt, gibt es Smartcards, die diese Dateistruktur nicht anbieten.

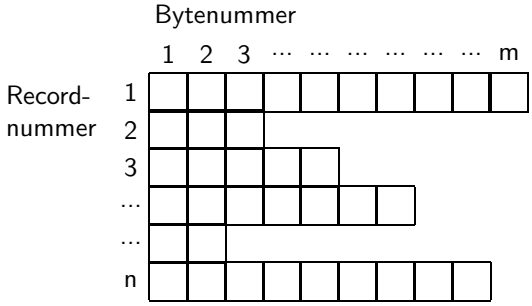


Abbildung 2.4: Linear variable Dateistruktur

cyclic Diese Struktur basiert auf der *linear fixed Dateistruktur*. Auch hier haben alle Records die gleiche Länge. Zusätzlich gibt es noch einen Zeiger, der auf den zuletzt geschriebenen Record zeigt. Erreicht der Zeiger den letzten Datensatz im File, wird er beim nächsten Schreibzugriff wieder auf den ersten Record in der Datei gesetzt. Dadurch ergibt diese Dateistruktur eine Art Ringspeicher. Die Abbildung 2.5 auf der nächsten Seite zeigt den Aufbau dieses Dateityps. Die Anzahl und Länge der Datensätze ist genau gleich wie bei *linear fixed*. Eine typische Anwendung dieses EF-Typs ist ein Speicher, in dem die letzten Anweisungen zwischengespeichert werden, um eine Art Undo-Funktion zu realisieren.

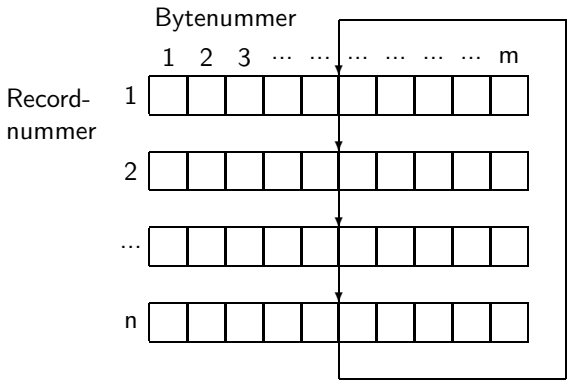


Abbildung 2.5: Cyclic Dateistruktur

2.3.4 Dateinamen

Bei modernen Smartcards werden die Dateien ausnahmslos logisch adressiert und nicht über direkte physikalische Adressen. Das heisst, dass die Dateien über Dateinamen angesprochen werden.

Alle Dateien haben einen zwei Byte langen *File Identifier* (FID), unter dem man die Datei ansprechen kann. Die Tabelle 2.1 auf der nächsten Seite zeigt, welche Namen schon fest vergeben sind. Von diesen abgesehen können die FIDs frei gewählt werden. Allerdings muss auch die Dokumentation der verwendeten Karte herbeigezogen werden, da sich nicht alle Kartenhersteller an die Standards halten und einige Dateinamen für sich reservieren.

Der Name muss so gewählt werden, dass die Datei eindeutig selektiert werden kann. Es ist also unmöglich, unter dem gleichen DF zwei gleichnamige Dateien zu speichern. Damit keine Probleme bei der Wahl eines Dateinamens entstehen, können folgende Regeln angewendet werden:

- EFs innerhalb eines Verzeichnisses dürfen nicht den gleichen FID haben.
- Verschachtelte Verzeichnisse (DFs) dürfen nicht den gleichen FID haben.
- EFs innerhalb eines Verzeichnisses (d.h. MF oder DF) dürfen nicht den gleichen FID wie das über- oder untergeordnete Verzeichnis haben.

2.3.5 Zugriffsbedingungen auf Dateien

Alle Dateien auf einer Smartcard haben Attribute für verschiedene Zugriffsrechte. Diese sind normalerweise im *Header* der Datei gespeichert. Die Zugriffsbedingungen werden beim Erstellen der Datei festgelegt und sind im Regelfall nicht mehr veränderbar. Je nach den von der Karte unterstützten Befehle sind mehr oder weniger Attribute vorhanden. So hat es zum Beispiel keinen Sinn, Zugriffsbedingungen für READ RECORD zu definieren, wenn dieses Kommando von der Smartcard nicht unterstützt wird.

FID	Name und Zweck	Norm
0000	Die Datei EFCHV1 wird zur Speicherung von PIN Nr. 1 inklusive korrespondierenden Informationen benutzt.	EN 726-3
0001	Die Datei EFKeyMan wird zur Speicherung von Schlüsseln für Anwendungszwecke benutzt.	EN 726-3
0002	Die Datei EFICC wird zur Speicherung von herstellungs- und betriebssystemrelevanten Informationen über die Smartcard benutzt (z.B.: Kartenseriennummer, Kartenhersteller, Moduleinbetter, Profil der Karte, Modi beim Anhalten des Taktes, ...).	EN 726-3
0003	Die Datei EFID wird zur Speicherung von Informationen über die Smartcard benutzt (z.B.: Aktivierungsdatum des MF, Verfallsdatum der Karte, ...).	EN 726-3
0004	Die Datei EFName enthält den Namen des Kartenbenutzers.	EN 726-3
0005	Die Datei EFIC wird zur Speicherung von Informationen über den Chip benutzt (z.B.: Chipseriennummer, Chiphersteller, ...).	EN 726-3
0011	Die Datei EFKeyMAN wird zur Speicherung von Schlüsseln für Verwaltungszwecke benutzt.	EN 726-3
0100	Die Datei EFCHV2 wird zur Speicherung von PIN Nr. 2 inklusive dazugehöriger PUK und diversen korrespondierenden Informationen benutzt.	EN 726-3
2F00	Die Datei EFDIR wird zur Speicherung von *Application Identifiers* (AID) mit dazugehöriger Pfadangabe zur korrespondierenden Anwendung benutzt.	EN 726-3 ISO/IEC 7816-4
2F01	Diese FID ist reserviert für die Datei EFATR mit den Erweiterungen zum ATR.	ISO/IEC 7816-4
2F05	Die Datei EFLANG wird zur Speicherung der bevorzugten Sprachen des Kartenbesitzers benutzt.	EN 726-4
3F00	Das MF ist das Wurzelverzeichnis für alle Dateien einer Smartcard.	ISO/IEC 7816-4 GSM 11.11 EN 726-3
3FFF	Diese FID ist reserviert für die Dateiselektion durch Pfadangabe.	ISO/IEC 7816-4
FFFF	Diese FID ist reserviert für zukünftige Benutzung durch ISO/IEC.	ISO/IEC 7816-4

Tabelle 2.1: Reservierte FIDs

Zu beachten ist, dass EFs hautpsächlich Informationen über den Datenzugriff (Schreib- und Leserechte) speichern, wohingegen DFs (inkl. MF) Bedingungen zum Erstellen und Löschen von Dateien innerhalb dieser Organisationsstruktur ablegen.

Je nach Betriebssystem der Smartcard werden mehr oder weniger Attribute unterstützt. Die üblichsten Befehle für EFs sind hier aufgelistet:

APPEND	Vergrössern einer Datei
DELETE FILE	Löschen einer Datei
INCREASE / DECREASE	Berechnungen innerhalb der Datei
INVALIDATE	Blockieren einer Datei
LOCK	Endgültiges Sperren einer Datei
READ / SEEK	Lesen / Suchen einer Datei
REHABILITATE	Entblocken einer Datei
WRITE / UPDATE	Schreiben in eine Datei

Die Zugriffsbedingungen für DFs unterscheiden sich grundlegend von denen der EFs. Es wird hier angegeben, welche Funktionen innerhalb des Verzeichnisses ausgeführt werden können:

CREATE	Erzeugen einer neuen Datei
DELETE FILE	Löschen einer Datei
REGISTER	Registrieren einer neuen Datei

2.3.6 Struktur der Smartcard-Befehle

Der gesamte Datenaustausch zwischen Smartcard und Terminal findet über sogenannte *Application Protocol Data Units* (APDUs) statt. Dabei wird grundsätzlich zwischen Kommando-APDUs und Antwort-APDUs unterschieden.

Struktur der Kommando-APDUs

Die Kommando-APDU setzt sich wie in Abbildung 2.6 aus einem Header- und einem Body-Teil zusammen.

Header				Body		
Class	INS	P1	P2	Lc-Feld	Datenfeld	Le-Feld

Abbildung 2.6: Die Struktur einer Kommando-APDU

Der Header besteht aus den Elementen Class, Instruction (INS) und den beiden Parametern P1 und P2. Das *Class-Byte* wird dazu verwendet, den spezifischen Befehlssatz auszuwählen. So verwenden beispielsweise GSM-Karten das *Class-Byte* 'A0' und ISO-Karten '0x'. Das INS-Byte bestimmt das gewünschte Kommando. Aus protokollspezifischen Gründen muss dieses immer geradzahlig sein. Die letzten beiden Felder im Header dienen dazu, den gewählten Befehl noch weiter zu spezifizieren. Sie werden deshalb vor allem dazu verwendet, die verschiedenen Optionen eines Kommandos auszuwählen.

Der Bodyteil der Kommando-APDU setzt sich aus dem Lc-Feld, einem Datenfeld und dem Le-Feld zusammen. Das Datenfeld beinhaltet die Daten, die zur Karte gesendet werden sollen. Das Lc-Feld legt die Länge des Datenfeldes fest. Dieses kann durchaus einen Wert von 0 haben, was bedeutet, dass der Datenteil vollständig wegfällt. Mit dem *Le-Feld* kann ein Erwartungswert für die Menge der Antwort-Bytes angegeben werden. Falls dieses Byte auf 0 gesetzt wird, erwartet der Reader das Maximum der für dieses Kommando zur Verfügung stehenden Daten von der Karte.

Struktur der Antwort-APDUs

In Abbildung 2.7 wird der Aufbau der Antwort-APDU gezeigt.

Body	Trailer	
Datenfeld	SW1	SW2

Abbildung 2.7: Die Struktur einer Antwort-APDU

Der Body-Teil besteht nur aus dem Datenfeld. Die Länge wurde in der Kommando-APDU schon festgelegt. Es kann allerdings sein, dass diese Länge auf null gesetzt wird, wenn während der Verarbeitung des Kommandos ein Fehler aufgetreten ist. In diesem Fall werden nur noch die beiden *Status-Wörter* (SW1 und SW2) zurückgesendet.

Der Trailer muss von der Karte auf jeden Fall als Antwort auf ein Kommando gesendet werden. Die beiden Bytes SW1 und SW2, die auch als *Returncode* bezeichnet werden, beinhalten den Status des ausgeführten Kommandos. '61xx' und '9000' bedeuten zum Beispiel, dass der angegebene Befehl fehlerfrei ausgeführt werden konnte. Die Abbildung 2.8 zeigt die Systematik bei den nach ISO/IEC 7816-4 definierten Returncodes.

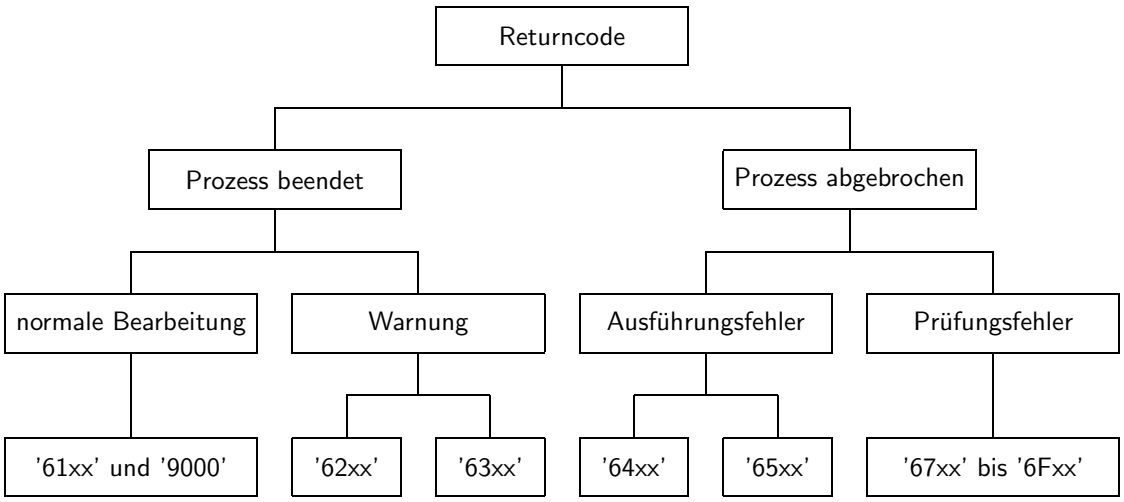


Abbildung 2.8: Die Systematik der nach ISO/IEC 7816-4 spezifizierten Returncodes.

2.3.7 Schlumberger Cyberflex Access Class 00

Diese Karte ist eine sogenannte JavaCard. Das heisst, es ist möglich, kleine Java Applikationen auf die Karte zu schreiben und auszuführen. Eine weitere positive Eigenschaft dieser Karte ist, dass sie Daten direkt auf der Karte digital unterzeichnen kann. Das hat den Vorteil, dass der *Private RSA-Key* die Karte nie verlässt.

Als weitere Sicherheit müssen *Cardlets*, die auf die Karte gespeichert werden sollen, digital signiert werden. Dadurch kann niemand unbefugt Programme auf die Karte schreiben und damit die Sicherheitseinstellungen umgehen.

Vorhandene Befehle

Wie jede Karte kann auch diese nicht den gesamten Befehlssatz, der in den verschiedenen Normen spezifiziert ist, unterstützen. Mit ihrem begrenzten Satz an Befehlen ist man jedoch sehr flexibel, wenn man eine sicherheitsrelevante Anwendung realisieren will. Dazu kommt, dass man sich mit der Java-Funktionalität leicht die fehlenden Befehle hinzuprogrammieren könnte. Die Tabelle 2.2 auf der nächsten Seite zeigt alle vorhandenen Befehle, die von der Cyberflex Karte unterstützt werden. Die vollständige Dokumentation dieser sehr empfehlenswerten Smartcard ist unter [cybe] zu finden.

Append Record fügt einen neuen Datensatz an ein EF an. Dies funktioniert aber nur, wenn hinter der Datei noch genügend freier Speicherplatz auf der Karte vorhanden ist. Der Parameter *length* muss bei *linear fixed* und *cyclic* EFs mit der bereits definierten Länge der Records übereinstimmen. Bei *linear variable* EFs wird damit die Länge des neuen Datensatzes übergeben.

Ask Random Mit diesem Befehl wird auf der Karte ein Zufallswert erzeugt. Mit dem *length* Parameter kann die Anzahl der zurückzugebenden zufälligen Bytes angegeben werden.

Change CHV Damit wird der CHV1 oder CHV2 Schlüssel überschrieben. Falls der Befehl erfolgreich ausgeführt wurde, hat man sich auch gleich als Kartenbesitzer authentisiert. Mit dem Parameter *chv_nb* kann der zu ändernde Schlüssel gewählt werden.

Change File ACL Mit diesem Befehl werden die Zugriffsbedingungen neu gesetzt. Dazu muss die betreffende Datei erst angewählt werden.

Change Java ATR Dieser Befehl ändert die Bytefolge, die vor dem Ausführen einer Java-Applikation gesendet wird. *Length* gibt die Anzahl zu sendender Bytes an.

Create File erstellt eine neue Datei. Zu beachten ist, dass zuerst das richtige Verzeichnis anzuwählen ist. Das Dateiformat ist dabei wie folgt definiert:

File Information	Dateilänge		FID		Typ	Status	Rec lgth	Nb rec
Bytes	1	2	3	4	5	6	7	8
Filerechte	Default	CHV1	CHV2	Aut0	Aut1	Aut2	Aut3	Aut4
Bytes	9	10	11	12	13	14	15	16

Hierbei ist der Dateityp einer der folgenden:

Befehl	**Class**	**INS**	**P1**	**P2**	**P3**
Change Java ATR	00/F0	FA	00	00	length
Get Data	00/F0	CA	00	01	16
Select	00/F0	A4	04	00	type
Get Response	00/F0	C0	00	00	length
Create File	00/F0	E0	00	00	10
Delete File	00/F0	E4	cmd1	00	cmd2
Directory	00/F0	A8	00	file_nb	length
Change File ACL	00/F0	FC	00	00	08
Get File ACL	00/F0	FE	00	00	08
Manage Instance	00/F0	08	cmd	00	00
Manage Program	00/F0	0A	cmd1	00	cmd2
Execute Method	00/F0	0C	cmd	00	length
Select	00/F0	A4	selector	00	length
Read Binary	00/F0	B0	hoffset	loffset	length
Update Binary	00/F0	D6	hoffset	loffset	length
Read Record	00/F0	B2	rec_nb	mode	length
Update Record	00/F0	DC	rec_nb	mode	length
Append Record	00/F0	E2	00	00	length
Verify CHV	00/F0	20	00	chv_nb	08
Change CHV	00/F0	24	00	chv_nb	10
Unblock CHV	00/F0	2C	00	chv_nb	10
Verify Key	00/F0	2A	00	key_nb	length
Logout All	00/F0	22	07	00	00
Invalidate	00/F0	04	00	00	00
Rehabilitate	00/F0	44	00	00	00
Ask Random	00/F0	84	00	00	length
Internal Auth	00/F0	88	algo_ID	key_nb	length
External Auth	00/F0	82	algo_ID	key_nb	length
Increase	00/F0	32	00	00	03
Seek	00/F0	A2	00	type	length
Enable CHV	00/F0	28	00	01	08
Disable CHV	00/F0	26	00	01	08
Sleep	00/F0	FA	00	00	00
Status	00/F0	F2	00	00	length

Tabelle 2.2: Von Cyberflex-Karten unterstützte Befehle

DF	0x20
Application	0x21
Program file	0x03
EF transparent	0x02
EF linear fixed	0x0C
EF linear variable	0x19
EF cyclic	0x1D

Für EFs (*linear fixed* und *cyclic*) gibt der Parameter *Rec lgth* die Länge der Records, und der Parameter *Nb rec* die Anzahl zu erstellender Datensätze an.

Delete File löscht entweder ein einzelnes File (*cmd1* = 0x00; *cmd2* = 0x02) oder gleich alle im aktuellen Verzeichnis (*cmd1* = 0x80; *cmd2* = 0x00). Falls nur eine einzelne Datei gelöscht werden soll, wird der FID als Daten gesendet.

Directory Dieser Befehl gibt Informationen über die im aktuell selektierten Verzeichnis enthaltenen Dateien.

Disable CHV deaktiviert den CHV1 Schlüssel. Das heisst, dass man sich mit nicht mehr mit als CHV1 authentisieren kann. Als Datenteil muss der aktuelle CHV1-Schlüssel übermittelt werden.

Enable CHV aktiviert den CHV1 Schlüssel wieder. Auch hier muss der CHV1-Schlüssel im Datenteil übermittelt werden.

Execute Method Dieser Befehl kann entweder ein Java Programm auf der Karte installieren, oder von einem *Java Cardlet* die *main()-Methode* aufrufen.

External Auth Mit diesem Befehl authentisiert sich ein Terminal gegenüber der Smartcard.

Get Data gibt Daten über die Karte zurück, beispielsweise die eindeutige Kartenidentifikationsnummer.

Get File ACL gibt die Zugriffsbedingungen der zur Zeit angewählten Datei zurück.

Increase erhöht den Wert des zuletzt geschriebenen Datensatzes und schreibt ihn in den nächsten vorhandenen Record. Dieser ist ein GSM spezifischer Befehl und kann nur bei *cyclic* EFs angewendet werden.

Internal Auth Mit diesem Befehl wird eine *Challenge* an die Karte gesendet, die das Ganze mit einem wählbaren Algorithmus verschlüsselt.

Invalidate Dieser Befehl setzt das angewählte EF ausser Kraft. Das heisst, es kann nicht mehr benutzt werden.

Log Out All Damit meldet man sich bei allen in dieser Sitzung angemeldeten Identitäten (z.B. Aut0 oder CHV1) ab.

Manage Instance Mit diesem Befehl wird das sogenannte *Instance-File* verwaltet. Dabei kann eine der folgenden Funktionen verwendet werden:

- Delete - Zum Löschen des *Instance-Container* und allen zugehörigen Dateien.
- Reset - Damit wird der *Instance-Container* neu initialisiert. Das heisst, es werden alle darin gespeicherten Objekte gelöscht.
- InitCurrent - Damit wird der aktuelle *Instance-Container* zum standard Container, der nach einem Reset der Karte angewählt wird.
- InitLoader - Damit wird der Default-Loader der Karte gesetzt.
- Block - Damit wird der aktuelle *Instance-Container* gesperrt oder wieder freigegeben.

Manage Program Damit wird der Status eines Programms von geladen zu erstellt geändert und umgekehrt.

Read Binary Mit diesem Befehl können Daten aus einem *transparent EF* gelesen werden. Dabei werden ab dem Offset (gebildet aus *hoffset* und *loffset*) *length*-Bytes ausgelesen.

Read Record Mit diesem Befehl wird ein Record aus einem *cyclic*, *linear fixed* oder *linear variable* EF ausgelesen. Zu Beachten gilt hier, dass *length* mit der wirklichen Länge des Datensatzes übereinstimmen muss.

Rehabilitate Das zuvor mit *Invalidate* gesperrte EF kann wieder brauchbar gemacht werden.

Seek sucht nach einem 'Text' am Anfang eines jeden Datensatzes im aktuell angewählten EF.

Select File wählt die angegebene Datei an. Je nach dem, wie der *selector* eingestellt ist, kann die Datei nach der FID oder nach dem *application name* angewählt werden.

Unblock CHV Der angegebene Schlüssel wird wieder entblockt. Dabei muss der *unblock CHV* und ein neuer CHV-Schlüssel angegeben werden.

Update Binary schreibt Daten in ein schon angewähltes *transparent* EF. Funktioniert gleich wie *Read Binary*.

Update Record überschreibt einen Datensatz.

Verify CHV Mit diesem Befehl wird der CHV Schlüssel überprüft. Damit kann man sich als Karteninhaber ausweisen.

Verify Key Dies ist ein genereller Befehl, der einen angegebenen Schlüssel mit dem auf der Karte vergleicht. Nach dem erfolgreichen Authentisieren ist man als einer der vier vorhandenen Aut-Benutzerlevels eingewählt.

2.4 Das RSA Public Key Verfahren

RSA ist wohl das bekannteste und am weitesten verbreitete asymetrische Verschlüsselungsverfahren. Sein Name stammt von seinen geistigen Vätern Rivest, Shamir und

Adleman, die den Algorithmus 1978 veröffentlichten. Sein Algorithmus gilt noch immer als sicher, da die Primfaktorzerlegung *grosser* Zahlen auch heute nur mit einem immensen Rechenaufwand lösbar ist. Beim jetzigen Stand der Technik sind damit Zahlen von mindestens 1024 Bit Länge (>300 Dezimalstellen) gemeint. Für das effiziente Rechnen in solchen Grössenordnungen, auf heute üblichen 32 Bit Rechnern, existieren diverse Bibliotheken. Es wird daher im folgenden nicht mehr genauer darauf eingegangen. Interessierte verweisen wir auf [sederwick] und andere Algorithmenbücher.

2.4.1 Schlüsselerzeugung

Um ein Schlüsselpaar der Länge N zu erzeugen wird wie folgt vorgegangen:

1. Wähle zwei grosse Primzahlen p und q von N-1 Bit Länge.
2. Bilde das Produkt $n = pq$, n sei N Bit lang.
3. Wähle ein $e > 1$, so dass es zu $(p-1)(q-1)$ teilerfremd ist.
4. Berechne ein d mit $de = 1 \, mod \, (p-1)(q-1)$.

n und e bilden den *Public Key*
n und d den *Private Key*

Das Erstellen der Primzahlen für den ersten Schritt der Schlüsselpaarerzeugung wird normalerweise mit Hilfe von Zufallszahlen bewerkstelligt. Dabei wählt man zufällig zwei Zahlen der erforderlichen Grösse aus und testet, ob es sich um Primzahlen handelt. Ein vollständiges Überprüfen der Zahlen würde aber zu lange dauern, weshalb man auf statistische Tests zurückgreift. Mit ihrer Hilfe kann die Wahrscheinlichkeit festgestellt werden, dass es sich um Primzahlen handelt. Einer der bekanntesten ist der *Rabin-Miller Test*. Besteht eine Zahl diese Überprüfung, so ist sie zu 99.9% eine Primzahl, ansonsten mit Sicherheit nicht. Schon nach sechs Testdurchläufen beträgt die Irrtumswahrscheinlichkeit weniger als 2^{-50} was mehr als ausreichend sein sollte. Eine detaillierte Beschreibung des *Rabin-Miller Tests* findet man in [wobst].

2.4.2 Verschlüsselung

Das Verschlüsseln erfolgt mit Hilfe des *Private Key*, bestehend aus Private Exponent d und Modul n, und läuft folgendermassen ab:

1. Unterteile den Klartext in Blöcke von N-1 Bits Länge (N sei die Schlüssellänge).
2. Verschlüssle jeden dieser Blöcke x durch Anwenden der Formel $y = x^d \, mod \, n$.
3. Setze die chiffrierten Blöcke y wieder zusammen.

Es gilt zu beachten, dass die verschlüsselten Blöcke N Bits lang sind, also um ein Bit länger als der Klartextblock.

2.4.3 Entschlüsselung

Um ein Chiffrat wieder zu entschlüsseln, wird der *Public Key*, bestehend aus Public Exponent e und Modul n, benötigt. Das Vorgehen ist mit dem Verschlüsseln nahezu identisch:

1. Unterteile das Chiffrat in Blöcke von N Bits Länge (N sei die Schlüssellänge).
2. Entschlüssle jeden dieser Blöcke y durch Anwenden der Formel $x = y^e \, mod \, n$
3. Setze die dechiffrierten Blöcke x wieder zusammen.

Beim Zusammensetzen der entschlüsselten Blöcke muss berücksichtigt werden, dass diese jeweils nur N-1 Bit lang sind.

2.5 X.509 Zertifikate

In einer klassischen Public-Key Umgebung werden die öffentlichen Schlüssel von einem zentralen Server geladen. Dabei ist es möglich, dass ein potentieller Angreifer den echten Schlüssel durch einen anderen ersetzt. Man kann also nicht sicher sein, dass man tatsächlich den richtigen Schlüssel der gewünschten Person bekommt.

Um dieses Problem zu lösen, werden die öffentlichen Schlüssel in einem Zertifikat gespeichert und signiert. Anhand dieses Zertifikates kann geprüft werden, ob dieser verändert wurde.

Grundsätzlich gibt es zwei Möglichkeiten, wer die öffentlichen Schlüssel unterschreibt:

- Web of Trust
- Hierarchische Vertrauensketten

2.5.1 Web of Trust

Beim Web of Trust ist jeder für sich selbst verantwortlich, welchen Zertifikaten er vertraut. Festgelegt wird dies durch das Unterschreiben des Zertifikats mit dem eigenen privaten Schlüssel. Alle Benutzer können in den Zertifikaten nachsehen, wer sie schon unterzeichnet hat. Dadurch hat jeder die Möglichkeit zu entscheiden, dass er allen Zertifikaten, die von einem bestimmten Benutzer unterschrieben wurden, vertraut.

In einer kleineren Gruppe, bei der jeder zu mehr oder weniger jedem eine persönliche Beziehung hat, funktioniert dieses System sehr gut. Jeder bittet jeden seiner Freunde um die Signierung seines öffentlichen Schlüssels.

Ein kleines Beispiel (siehe auch Abbildung 2.9 auf der nächsten Seite): Alice unterschreibt den öffentlichen Schlüssel von Bob. Also vertraut sie ihm. Ebenfalls traut Bob Dave und unterschreibt dessen Schlüssel.

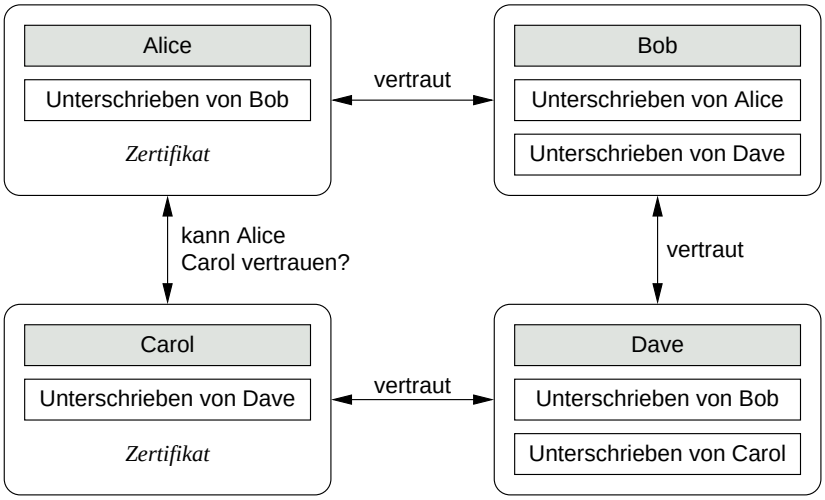


Abbildung 2.9: Beispiel eines Web of Trust

Wenn nun Alice eine von Dave unterschriebene Mail erhält, kann sie anhand des Zertifikates von Dave's öffentlichem Schlüssel erkennen, dass Bob den Schlüssel unterschrieben hat. Und da sie Bob vertraut (sie hat ja den Schlüssel von Bob signiert), kann sie davon ausgehen, dass das empfangene Mail tatsächlich von Dave ist.

Ab einer gewissen Verknüpfungslänge kann man sich nicht mehr sicher sein, ob man dem Zertifikat vertrauen kann. So ist es beim Beispiel in Abbildung 2.9 fraglich, ob Alice Carol vertrauen soll. Der grosse Vorteil ist aber, dass die Schlüssel nicht von einer zentralen Stelle aus verwaltet, respektive unterschrieben werden.

Als Zertifikatstyp werden bei diesem System hauptsächlich OpenPGP Zertifikate verwendet [rfc2440].

2.5.2 Hierarchische Vertrauensketten

In einer hierarchischen Vertrauenskette gibt es als oberste Instanz eine Root Certificate Authority (Root CA). Dessen Zertifikat muss zwingend vertraut werden, da alle weiteren auf diesem aufbauen. Diese Root CA unterzeichnet entweder direkt Schlüssel von Benutzern oder erstellt Intermediate Certificate Authorities, die ihrerseits Benutzerschlüssel unterzeichnen. Dies ist dann sinnvoll, wenn zum Beispiel eine Firma ihre eigenen Zertifikate erstellen möchte.

Als mögliches Beispiel sei die Struktur aus Abbildung 2.10 auf der nächsten Seite gegeben. Bekommt nun Carol eine signierte Nachricht von Bob mit dessen Zertifikat, kann sie erkennen, dass es von der ZHW unterschrieben wurde. Anhand des Zertifikats der ZHW stellt Carol fest, dass dieses wiederum von Verisign signiert wurde. Da diese eine ihr bekannte Root-CA ist, kann Carol dem Zertifikat von Bob vertrauen und damit die Signatuer der Nachricht überprüfen.

Bei der hierarchischen Vertrauenskette werden hauptsächlich ITU-T X.509 Zertifikate verwendet [rfc2459], die im nächsten Kapitel noch genauer beschrieben werden.

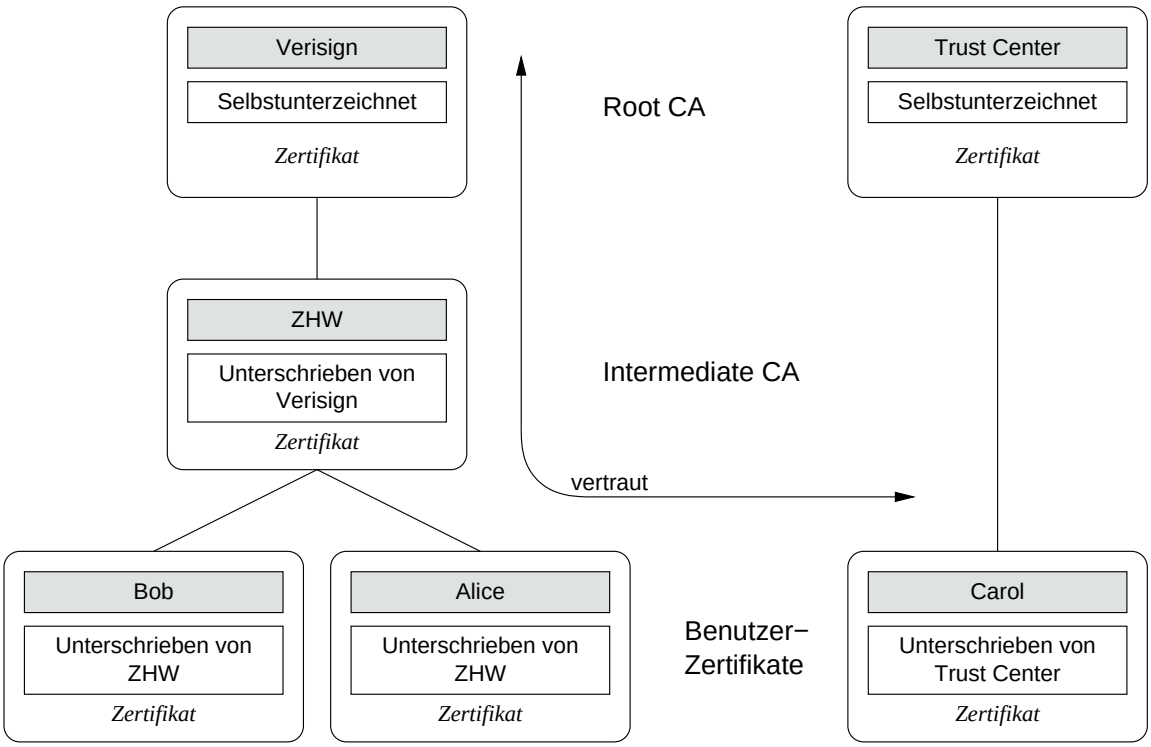


Abbildung 2.10: Beispiel einer hierarchischen Vertrauenskette

2.5.3 Aufbau eines X.509v3 Zertifikates

Wie in der Abbildung 2.11 auf der nächsten Seite gezeigt, besteht der Aufbau eines X.509v3 Zertifikats grundsätzlich aus drei Teilen.

Body Der Body-Teil des Zertifikats enthält die folgenden Elemente:

- Versionsnummer (aktuell ist v3)
- Eine einmalige Seriennummer, die von der verantwortlichen CA vergeben wird.
- Den Algorithmus, der für die Signatur verwendet wird.
- Die ID der CA, die das Zertifikat erstellt hat.
- Die Gültigkeitsdauer (Gültig von / bis)
- Die Benutzer ID
- Den öffentlichen Schlüssel des Benutzers
- optionale v2 oder v3 Erweiterungen (extensions)

Signatur Algorithmus Definiert den verwendeten Algorithmus, mit dem das Zertifikat unterschrieben wurde.

Signatur Beinhaltet die Signatur des Hash-Wertes aus dem Body-Teil des Zertifikats.

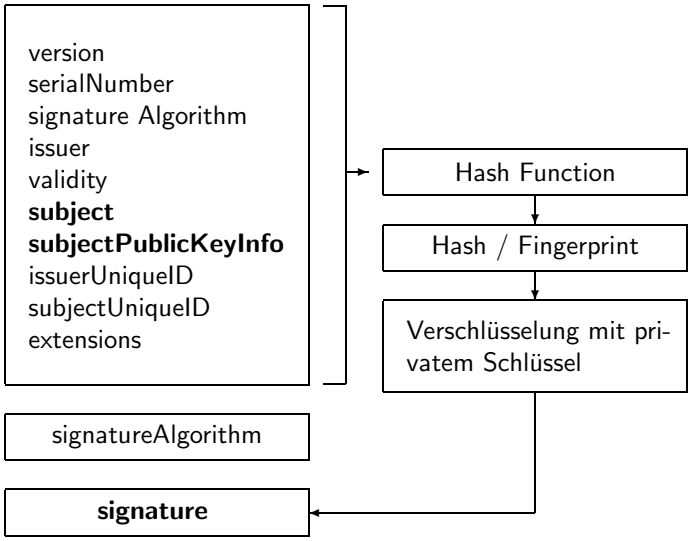


Abbildung 2.11: Aufbau eines X.509-Zertifikats

Beim Signieren eines Zertifikats wird zuerst mit der angegebenen *Hash Function* ein Hash über den Body-Teil gebildet. Dieser wird dann mit dem privaten Schlüssel der Zertifizierungsstelle verschlüsselt, was die eigentliche Signatur ergibt. Dann wird dem Body-Teil noch der *signature Algorithm* und die Signatur selbst angehängt. Abbildung 2.11 zeigt diesen Ablauf noch grafisch.

Um ein Zertifikat für die verschiedensten Anwendungen brauchen zu können, werden Extensions hinzugefügt. Dadurch können weitere, für die Applikationen wichtige Informationen gespeichert werden. So können zum Beispiel Dinge wie *Key Usage*, *Subject Alternative Name* und *CRL Distribution Points* angegeben werden, um zu spezifizieren, wozu der öffentliche Schlüssel im Zertifikat gebraucht werden darf, welche alternativen Namen der Besitzer des Zertifikats hat und an welcher Stelle man die CRL findet. Daneben können aber auch eigene Extensions definiert werden. So existieren zum Beispiel einige Netscape-Extensions, die extra für diesen Internet-Browser erstellt wurden.

2.5.4 Certificate Revocation List

Obwohl ein Zertifikat nur eine gewisse Gültigkeitsdauer hat, ist es sinnvoll, eine Möglichkeit zu bieten, mit der man bestimmte Zertifikate als ungültig erklären kann. Dazu dient die *Certificate Revocation List* (CRL).

Diese CRL enthält eine Liste aller gesperrten Zertifikate. Damit kein Unbefugter diese Liste ändern kann, wird sie ebenfalls von der entsprechenden Zertifizierungsstelle signiert. Um ein Zertifikat zu prüfen, muss daher zusätzlich noch anhand der CRL festgestellt werden, ob es nicht gesperrt wurde. Falls doch, gilt das Zertifikat als ungültig.

2.6 Verzeichnisdienste

2.6.1 Allgemeines

Bei Verzeichnisdiensten handelt es sich um eine im Netzwerk verteilte Datenbank. In ihr können nahezu beliebige Informationen gespeichert werden. Diese Informationen können dazu verwendet werden, verschiedene Anwendungen zu konfigurieren und zu administrieren.

Die Daten können dabei

- flexibel,
- dynamisch und
- sicher

aufbewahrt werden. Die Struktur der Daten in Verzeichnisdiensten können vom Anwender bestimmt und im laufenden Betrieb geändert werden. Es ist ferner möglich, Sicherheitsregeln zu definieren, welche die Zugriffsmöglichkeiten der Anwender auf die Informationen im Verzeichnisdienst beschreiben.

Da in den Verzeichnisdiensten häufig Konfigurationen gespeichert sind, wird meist nur lesend zugegriffen. Das führt zu einer höheren Geschwindigkeit beim Zugriff auf die Informationen als bei üblichen Datenbanken.

Durch die Möglichkeit der Replizierung wird zudem die Ausfallsicherheit erhöht, da die Informationen nicht nur auf einem Server gespeichert sind.

Es kann von jedem Client im Netzwerk auf den Verzeichnisdienst zugegriffen werden. Auch die Administration ist von irgend einem Client im Netzwerk aus möglich.

2.6.2 Aufbau

Bei einer Verzeichnisdienst-Datenbank handelt es sich um eine baumartige Struktur. In ihr werden die gewünschten Informationen gespeichert. Der Baum (englisch Tree) hat grundsätzlich eine Wurzel (Root). Unter ihr können weitere Strukturen ergänzt werden, bis eine Art Baum ensteht.

Das Beispiel in Abbildung 2.12 auf der nächsten Seite zeigt einen möglichen Aufbau einer Verzeichnisdienst-Datenbank für eine Schule.

Unter *Root* wird die Struktur mit dem Name *ZHW* angelegt, was dem Namen der Schule entspricht. Darunter existieren die zwei Bereiche *Dozenten* und *Studenten*, in denen die Namen der einzelen Personen gespeichert sind. Diese Einträge wiederum haben die verschiedensten Eigenschaften, um sie genauer zu beschreiben. So könnte zum Beispiel ein Eintrag unter *Dozenten* neben dem Namen und der Adresse der Person auch der Tätigkeitsbereich, interne Telefonnummer, Lohn usw. gespeichert werden. Bei den *Studenten* könnte neben der Adresse auch noch die Klasse, der Studiengang und die Noten

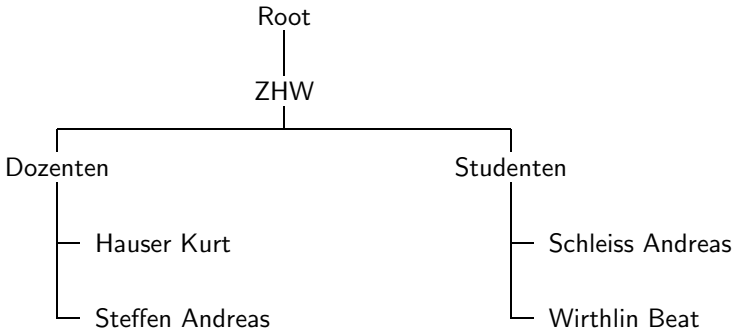


Abbildung 2.12: Beispiel eines Verzeichnisdienstes für eine Hochschule

gespeichert sein. Es ist aber nicht notwendig, dass in einem Bereich immer die gleichen Eigenschaften anzugeben sind. Für jeden einzelnen Eintrag kann gewählt werden, was wichtig ist.

Zusammenfassend kann der Aufbau eines Verzeichnisdienstes mit folgenden Punkten beschrieben werden:

- Die Verzeichnisdienst-Datenbank ist baumartig aufgebaut.
- Es existiert immer eine Wurzel (Root).
- In der Baumstruktur existieren Einträge.
- Die Einträge haben Eigenschaften.

Eine Verzeichnisdienst-Datenbank kann auf mehrere verschiedene Rechner aufgeteilt werden. So könnte für unser Beispiel mit der Schule ein Rechner zuständig sein für die *Dozenten*, ein zweiter für die *Studenten* und, um auch noch die Replizierung ins Spiel zu bringen, ein dritter, der beide Bereiche gespeichert hat. Dadurch ist die Datenbank auf verschiedene Rechner verteilt (Lastenverteilung), und ebenfalls repliziert, das heisst, mehrfach gespeichert. Für den Anwender des Verzeichnisdienstes ist dies jedoch völlig transparent. Er profitiert lediglich von der erhöhten Ausfallsicherheit.

2.6.3 Funktion

Im Zusammenhang mit Verzeichnisdiensten gibt es zwei grundlegende Funktionen:

- Speichern der Informationen
- Verwenden der Informationen

Speichern der Informationen. Die strukturierte Speicherung der Daten erfolgt in der bereits bekannten Baumstruktur. Auf die gespeicherten Informationen kann dann von jedem Client aus zugegriffen werden.

Verwenden der Informationen. Der wichtigere Aspekt ist die direkte Verwendung der gespeicherten Informationen zur Konfiguration anderer Dienste und Anwendungen. Es wäre beispielsweise wünschenswert, wenn sich die im Verzeichnisdienst gespeicherten Benutzer mit dem dort aufgeführten Namen am System anmelden könnten. Ebenfalls praktisch wäre eine direkte Passwort-Verifizierung am Verzeichnisdienst.

Was genau an Daten für die Konfiguration anderer Dienste und Anwendungen benutzt wird, hängt davon ab, welche der folgenden Zugriffsvarianten auf den Verzeichnisdienst möglich sind:

Grafische Anwendungen Um bequemer auf die gespeicherten Informationen zugreifen zu können, ist eine grafische Oberfläche sehr praktisch. Da der Zugriff auf Verzeichnisdienste standardisiert ist, ist es möglich, auch mit anderen Produkten und anderen Systemen auf die Daten zuzugreifen. Auch die meisten heute gängigen Internet-Browser können die Daten anzeigen.

Kommandos Mit grafischen Anwendungen können die erlangten Daten nur eingesehen, jedoch nicht weiter verwendet werden. Mit den zur Verfügung stehenden Kommandos können Scripts in der Textkonsole erstellt werden, welche die Informationen auswerten und weiterleiten können.

Funktionen und Routinen Um auch neue Programme entwickeln zu können, stehen für einige Programmiersprachen Schnittstellen für die Verwendung von Verzeichnisdiensten zur Verfügung. So gibt es beispielsweise Libraries für C und Perl, die einfach zu benutzen sind, aber dennoch die gesamte Funktionalität des Dienstes nutzen.

2.7 LDAP

In der Anfangsphase der Verzeichnisdienste erfolgte der Zugriff auf die X.500-konformen Daten mit dem *Directory Access Protocol* (DAP). Dieses Protokoll konnte sich aber aufgrund seiner Komplexität nicht durchsetzen und wurde durch eine vereinfachte Form ersetzt. Das Lightweight Directory Access Protocol (LDAP) ist heute der Standard, wenn es um den Zugriff auf X.500-Daten geht. Wie sein grosser Bruder DAP basiert LDAP auf dem TCP/IP-Protokoll und ist funktionell und schnell, jedoch leichter zu implementieren.

2.7.1 LDAP-Aktionen

LDAP-Aktionen können grob in drei Kategorien eingeteilt werden:

Beglaubigung Der Client sendet in der Regel einen Benutzernamen und ein Passwort an den LDAP-Server. Dieser überprüft die Angaben und sofern alles in Ordnung ist, wird dem Anwender die Benutzung der ihm erlaubten Funktionen gewährt.

Lesender Zugriff Die meisten Zugriffe auf eine X.500-Datenbank sind lesender Art. Neben dem grundsätzlichen Suchen, kann auch ein gegebener Wert mit dem der Verzeichnisdienste-Datenbank verglichen werden.

Schreibender Zugriff Um Daten im Verzeichnisdienst ändern zu können, muss der Benutzer über die nötigen Schreibrechte verfügen. Falls er diese besitzt, kann er Werte hinzufügen, löschen, ändern und umbenennen.

2.7.2 Das Schema

Die Struktur der Datenbank eines Verzeichnisdienstes wird mit dem Begriff Schema bezeichnet. Dieses umfasst alle möglichen Einträge im Verzeichnisbaum sowie die Eigenschaften in der strukturellen Darstellung.

Es gliedert sich in die Punkte

Objekte Die Einträge in der Baumstruktur bezeichnet man als Objekte.

Attribute Die Eigenschaften der Objekte heissen Attribute.

Regeln Welches Objekt an welcher Stelle im Verzeichnisbaum stehen darf und welche Attribute es aufweisen kann, bzw. muss, wird mit Regeln im Schema beschrieben.

Objekte

Die Objekte eines Verzeichnisdienstes werden in sogenannten Objektklassen beschrieben. Diese zentralen Einträge haben die folgenden Aufgaben und Eigenschaften:

Eindeutiger Name Jede Objektklasse wird durch einen eindeutigen Namen identifiziert. Zum Beispiel könnte eine Klasse mit dem Namen *person* existieren. Im Verzeichnisbaum könnten dann Objekte dieses Typs gespeichert werden. *Steffen Andreas* wäre ein Beispiel dafür. Die Namen der Objekte und Objektklassen sind nicht *case-sensitive*. Es wird also nicht zwischen Gross- und Kleinschreibung unterschieden.

Eindeutige Identifikation Identifiziert werden die Objektklassen anhand einer eindeutigen Kennung. Der Name dient nur zur einfacheren Verwendung.

Notwendige Attribute Es wird definiert, welche Attribute zwingend notwendig sind.

Mögliche Atrribute Bei den meisten Objektklassen kann man neben den notwendigen Attributen auch eine Reihe weiterer optionaler Attribute angeben.

Die Objekte der Baumstruktur werden in zwei Bereiche unterteilt:

Containerobjekte Als Container werden Objekte bezeichnet, die ihrerseits wieder Objekte enthalten können. In unserem Beispiel mit der Schule, wäre *Studenten* ein solches Containerobjekt.

Blattobjekte Blattobjekte sind die Blätter im Verzeichnisbaum. Dementsprechend ist das Objekt *Hauser Kurt* ein Blattobjekt.

Container lassen sich weiter noch in vier Klassen einteilen:

- Root
- Country (c)
- Organization (o)
- OrganizationalUnit (ou)

Country-Container können verwendet werden, um ein Land anzugeben. Durch die Klasse Organization wird eine Firma o. ä. repräsentiert, die durch die Klasse OrganizationalUnit noch weiter unterteilt werden kann. Blattobjekte werden generell als

- Common Name (cn)

bezeichnet.

Attribute

Wie die Objekte haben auch die Attribute eine definierte Struktur und vorgegebene Aufgaben:

Eindeutiger Name Wie bei den Objekten, werden auch Attribute durch einen eindeutigen Namen repräsentiert.

Eindeutige Identifikation Auch hier gilt der Name lediglich als Synonym für eine Identifikation.

Syntax Den Attributen können Werte zugewiesen werden. Dabei ist fest definiert, welcher Syntax diese Werte unterliegen. So könnte der Eigenschaft *Telefon* ein Wert mit maximal zwölf Bytes übergeben werden. Ferner kann definiert werden, ob das Attribut case-sensitive ist und welche Zeichen es enthalten darf. Dadurch können Buchstaben in Telefonnummern generell unterbunden werden.

Werte des Attributs Es ist möglich, dass ein Attribut mehrere Werte annimmt. So könnte man für das oben erwähnte Attribut `Telefon` mehrere Nummern zuweisen.

Regeln

Wenn ein Verzeichnisdienst erstellt wird, sind einige Regeln zu beachten. Durch sie wird definiert, an welcher Stelle im Verzeichnisbaum Container und Blattobjekte stehen dürfen.

1. Das oberste Element in einem Verzeichnisbaum ist das Objekt Root. Es ist immer vorhanden und kann nicht verschoben oder gelöscht werden.
2. Unterhalb von Root darf entweder ein Country-Objekt (c) oder eine Organization (o) gespeichert sein.

3. Es muss kein Country-Objekt existieren.
4. Es darf höchstens ein Country-Objekt vorhanden sein.
5. Die Organisation muss direkt unter dem Country-Objekt oder der Root stehen.
6. Auf der gleichen Ebene dürfen mehrere Organisationen sein.
7. Unterhalb einer Organisation (o) dürfen entweder direkt Blattobjekte (cn), oder OrganizationalUnit (ou) stehen.
8. Den organisatorischen Einheiten können nur Blattobjekte oder weitere ou-Objekte folgen.
9. Es dürfen mehrere Objekte der Klasse ou auf gleicher Ebene stehen.
10. Blattobjekte (cn) dürfen nur unter o und ou vorkommen.

2.7.3 Distinguished Name

Da Objekte im Verzeichnisdienst einen eindeutigen Namen haben müssen, wird jeweils der gesamte Pfad von der Wurzel bis zum Objekt angegeben. Dieser Pfad wird **Distinguished Name** (**dn**) genannt und besteht aus den einzelnen Objekten, die der Reihe nach durchlaufen werden müssen, bis man zum Objekt kommt. Abbildung 2.13 zeigt die einzelnen Distinguished Names vom Schule-Beispiel.

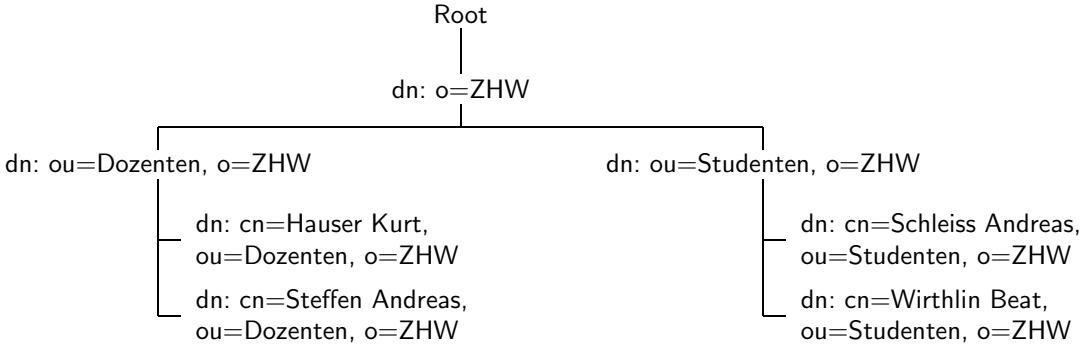


Abbildung 2.13: Distinguished Names

2.8 Secure Shell

Secure Shell (SSH) ist ein Programm, mit dem man sich bei einem anderen Rechner im Netzwerk einloggen kann, um Programme auszuführen und Dateien zu kopieren. Dabei wird die Kommunikation über eine unsichere Verbindung verschlüsselt, damit die Datenübertragung nicht abgehört werden kann. Zusätzlich bietet SSH eine zuverlässige gegenseitige Authentisierung der Partner.

Da die Verschlüsselung der Kommunikation vor der Identifizierung gestartet wird, werden niemals Passwörter oder andere Informationen im Klartext übertragen. Für den Benutzer geschieht das Ganze vollständig transparent.

Die folgende Liste zeigt einige interessante Eigenschaften von SSH:

- Durch die Verwendung einer Authentisierung des Servers werden einige Sicherheitslücken wie IP-Spoofing oder DNS-Spoofing geschlossen.
- Es können beliebige Daten zwischen den Maschinen ausgetauscht werden. Dabei kann optional eine ZIP-Kompression verwendet werden, um langsame Verbindungen besser nutzen zu können.
- Normalerweise erfolgt eine automatische und vollständig transparente Verschlüsselung der gesamten Verbindung mittels eines ausgewählten symmetrischen Verfahrens.
- Es können auch X11- und beliebige TCP-Verbindungen verschlüsselt werden.

Kapitel 3

Analyse und Design

3.1 Übersicht

Wie aus Abschnitt 1.1 auf Seite 1 ersichtlich, ist das Ziel dieses Projektes, den im Rahmen einer Projektarbeit erstellten *RSA Smartcard Login* netzwerkfähig zu machen. Die zur Authentisierung benötigten öffentlichen Schlüssel der Benutzer sollen diesmal mit Hilfe von X.509 Zertifikaten abgelegt werden. Des weiteren ist auch die Realisierung eines Remote-Logins anzustreben. Falls möglich, sollte nur auf freie Programme und Bibliotheken zurückgegriffen werden, um eine Veröffentlichung des Paketes unter der *GNU Public License* zu ermöglichen.

Von diesen Anforderungen ausgehend kommt man zu folgenden fünf wesentlichen Punkten, die zu erfüllen sind:

1. In einem Netzwerk können die Benutzer- und Gruppen-Daten nicht mehr auf jedem Host lokal abgelegt werden, sondern müssen in einem zentralen Verzeichnis gespeichert sein, um den Administrationsaufwand in Grenzen zu halten und Inkonsistenzen zu vermeiden.

2. Damit nicht alle Programme, die in irgend einer Weise Benutzer authentisieren, angepasst werden müssen, ist ein auf Linux-PAM aufbauendes System Voraussetzung.

3. Alle Hosts müssen einen lauffähigen Smartcard-Reader angeschlossen haben.

4. Die Verwendung von X.509 Zertifikaten erfordert es, eine Certificate Authority (CA) aufzubauen.

5. Auf jedem Host muss ein Dienst eingerichtet sein, der den Zugriff auf das lokale System von einem entfernten Host aus regelt.

3.1.1 Verteiltes Benutzer- und Gruppenverzeichnis

Verzeichnisdienste für die Verwaltung von Benutzer- und Gruppeninformationen können unter Linux auf die verschiedensten Arten realisiert werden. Neben Traditionellem wie NIS

und NIS+ können auch rationale Datenbanken, LDAP-Server oder gar Windows Domänen Server verwendet werden. Ermöglicht wird diese Vielfalt durch die Verwendung einer NSS-fähigen GNU C Library, was seit längerem als Standard gilt. Wie in Abschnitt 2.1 auf Seite 5 gezeigt, wird die Anbindung der Dienste an die Library mit Hilfe von NSS-Modulen realisiert.

Für unser Projekt fällt die Wahl auf den freien LDAP-Server OpenLDAPv2 (http://www.-openldap.org) und das zugehörige NSS-Modul nss_ldap (http://www.padl.com). Folgende Gründe führten zu dieser Wahl:

- Neben den grundlegenden Benutzer- und Gruppendaten müssen von uns noch weitere Informationen wie öffentliche Schlüssel, CRLs und ARLs abgelegt werden.
- In einem LDAP-Verzeichnis können nachträglich, ohne nennenswerten Aufwand weiterführende Daten zu einem Benutzer wie Telefonnummer, E-Mail etc. hinzugefügt werden.
- Seit LDAP Version 3 ist eine Verschlüsselung der Verbindung mittels TLS oder SSL möglich.
- Die Portierung der lokalen Daten aus den `passwd` und `group` Dateien sowie der Zugriff auf diese ist in [rfc2307] standardisiert.
- Das erwähnte LDAP NSS-Modul von PADL Software gilt als Referenz Implementation für das [rfc2307].

3.1.2 Linux-PAM

Das Sicherstellen einer funktionierenden PAM-Umgebung ist durch uns nur bedingt machbar. Obwohl alle uns bekannten Linux Distributionen schon seit geraumer Zeit auf Linux-PAM aufbauen, gibt es noch immer Applikationen, welche den PAM-Stack ignorieren und ihre eigene Authentisierungsfunktion implementieren. Da es für uns unmöglich ist, all diese Programme aufzuspüren und anzupassen, werden wir uns auf die Entwicklung eines ausgereiften und breit einsetzbaren PAM-Moduls konzentrieren. Eine Untersuchung der von uns verwendeten SuSE 7.2 Distribution zeigte, dass alle für uns relevanten Applikationen (login, xdm, kdm, passwd, xlock und KDE 2.1) den Linux-PAM Standard unterstützen.

3.1.3 Smartcard-Reader

Wie bei der Projektarbeit wird auch hier die als Standard für Prozessor Smartcards geltende PC/SC Lite Middleware (http://www.linuxnet.com) benutzt. Die wachsende Treiberunterstützung garantiert auch eine längerfristige Benutzbarkeit dieser Bibliothek. Wegen einiger Mängel in der aktuellen Version 1.0.0beta, wird die etwas ältere Version 0.9.3 benutzt. Die Kompatibilität mit der neuen Version ist trotzdem gewährleistet.

3.1.4 Certificate Authority

Die Verwendung von X.509 Zertifikaten zum Vertrieb der öffentlichen Schlüssel hat den Vorteil, dass die Zuordnung eines Benutzers zu dessen Schlüsselpaar mit Hilfe des Root Zertifikates relativ einfach und mit grosser Sicherheit überprüft werden kann. Als Nachteil muss, falls nicht bereits geschehen, eine CA aufgebaut und allen Netzwerkbenutzern ein Zertifikat ausgestellt werden. Um den Verwaltungsaufwand in Grenzen zu halten, soll jeder Benutzer nur ein Login-Zertifikat besitzen. Die einzelnen Identitäten (Profile) unter denen er sich im System anmelden darf (z.B. user1, root, webadmin usw.), werden in diesem einen Zertifikat abgelegt und während der Anmeldung überprüft.

Um den Aufwand dieser Diplomarbeit im Bereich des Machbaren zu halten, können nicht noch eigene Applikationen zur Erstellung und Verwaltung der Zertifikate entwickelt werden. Stattdessen wird auf das, als standard geltende OpenSSL (http://www.openssl.org) Paket zurückgegriffen.

3.1.5 Remote-Login

Das produktive Arbeiten in einem Netzwerk erfordert die Möglichkeit, sich auf entfernten Rechnern einwählen zu können. Als sicherer Ersatz für die bekannten Remote-Tools (rlogin, rcp usw.) bietet sich das freie OpenSSH (http://www.openssh.org) Paket an. Es sollen mindestens die beiden elementaren Operationen Remote-Login (ssh, sshd) und Dateitransfer (scp) implementiert werden.

3.2 Verwendete Hardware

Für die Realisierung unseres Projektes sind mindestens zwei Hardware-Teile erforderlich:

- Smartcard-Reader
- Smartcard

In unserer ersten Projektarbeit *Linux Login mit RSA-Smartcards* haben wir schon Erfahrung im Umgang mit den Readern

- Schlumberger Reflex 60
- Towitoko Chipdrive

gesammelt. Da diese Kartenleser grösstenteils unseren Anforderungen entsprochen haben, werden wir auch dieses Mal wieder mit diesen arbeiten.

Bei der Wahl der Smartcards müssen wir vor allem berücksichtigen, dass die Karten einige kryptographische Funktionen unterstützen. Zumindest sollten sie eine an die Karte gesendete Nachricht mit einem gespeicherten privaten Schlüssel verschlüsseln können.

Auch hier setzen wir auf die Erfahrung aus unserer ersten Projektarbeit und wählen die *Schlumberger Cyberflex Access* Smartcard. Diese bietet mit der *Internal Auth* Funktion die benötigte Möglichkeit, etwas zu verschlüsseln. Kapitel 2.3.7 auf Seite 17 geht noch genauer auf diese Smartcard und ihre Befehle ein.

Auf den ersten Blick wäre die Smartcard *Schlumberger Cryptoflex* auch gut für unser Projekt geeignet. Sie würde auch noch eine Generierung eines Schlüsselpaares auf der Karte selbst unterstützen, so dass dieses die Karte nie verlassen würde. Da wir aber wie erwähnt mit OpenSSL ein Zertifikat erstellen, brauchen wir das Schlüsselpaar auf der Harddisk, so dass dies kein Vorteil mehr ist.

Zusätzlich haben wir in der ersten Projektarbeit festgestellt, dass die Cryptoflex-Karten einen eigenen Befehlssatz verwenden und nicht nach ISO 7816 arbeitet. Das Dateisystem auf der Karte ist ebenfalls etwas merkwürdig, da man erstellte Dateien nur in der umgekehrten Reihenfolge wieder löschen kann.

Wegen dieser Mängel der Cryptoflex-Karte entscheiden wir uns, nur noch die Cyberflex Smartcards zu verwenden. Damit werden auch andere ISO 7816 Smartcards unterstützt.

3.3 Aufbau unserer Login Smartcards

3.3.1 Datei-Struktur

In [pkcs15] wird definiert, wie die einzelnen Schlüssel und Zertifikate auf einer Smartcard gespeichert werden sollten. Wir werden jedoch unsere eigene Struktur erstellen. Dies deswegen, weil die Realisierung der PKCS #15 Struktur sehr komplex ist und unsere Smartcard dies noch nicht richtig unterstützen. Als weitere Abschreckung stolperten wir über eine Internetseite, auf der *nur* eine Implementation des PKCS #15 als sechs-monatige Diplomarbeit (http://www.iaik.at/teaching/11_diplomarbeiten/PKCS15.php) ausgeschrieben war.

Da wir die verschiedenen Profile der Benutzer in einem Zertifikat speichern wollen, wird die Verzeichnisstruktur auf den Smartcards sehr einfach. Die einzigen Dateien, die zusätzlich zu den PIN-Dateien gespeichert werden müssen, sind:

- Öffentlicher Schlüssel
- Privater Schlüssel
- Zertifikat

Um den Umgang mit den Karten weiter zu vereinfachen, werden diese drei Dateien direkt ins Hauptverzeichnis geschrieben. Die Abbildung 3.1 auf der nächsten Seite zeigt den Aufbau unserer Verzeichnisstruktur. Da Smartcards nur rudimentäre Dateisysteme implementieren, besitzen gespeicherte Dateien als Identifikator nur eine Nummer.

Die einzelnen Dateien haben dabei die folgenden Bedeutung:

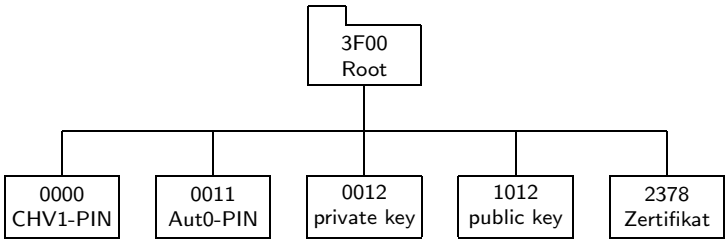


Abbildung 3.1: Dateistruktur unserer Login Smartcards

3F00 Wie in Kapitel 2.3.4 auf Seite 13 beschrieben, ist dies das Rootverzeichnis der Karte.

0000 In dieser Datei ist der Kartenbesitzer-PIN abgespeichert (CHV1). Dieser PIN wird dazu verwendet, den Besitzer gegenüber der Karte zu authentisieren. Erst nach Eingabe dieses PINs können weitere Befehle, wie das Verschlüsseln auf der Karte ausgeführt werden. Die Zugriffsrechte müssen so gesetzt sein, dass dieser PIN nicht ausgelesen werden kann.

0011 Hier ist der Administrator-PIN gespeichert (Aut0). Daneben sind auch noch weitere Schlüssel abgespeichert, die aber für unser Projekt keine weitere Bedeutung haben.

0012 Enhält den privaten Schlüssel des Kartenbesitzers. Nach ihrer Erstellung darf es nicht mehr möglich sein, diese Datei auszulesen.

1012 Enthält den öffentlichen Schlüssel des Kartenbesitzers. Es ist durch geeignete Befehle möglich, diesen Schlüssel wieder auszulesen, was aber unser Projekt nicht unterstützt.

2378 Hier wird das Zertifikat im DER-Form abgespeichert. Auch diese Datei sollte wieder ausgelesen werden können, was zum Beispiel bei der PAM-Authentisierung verwendet wird. Nebenbei: Die Zahl entstand durch das Schreiben von CERT auf einer Telefontastatur (2378 → CERT) und ist noch nicht anderweitig reserviert.

3.3.2 Zugriffsrechte

Damit wichtige Dinge wie der private Schlüssel nicht wieder ausgelesen werden können, werden wir restriktive Zugriffsbedingungen setzen. Wir sind der Meinung, dass es besser ist, wenn man zum Beispiel nach Ablauf eines Zertifikats eine Karte neu erstellen muss, dafür aber Gewissheit hat, dass die Daten nicht ausgelesen werden können.

Das Setzen der Zugriffsrechte bei der *Cyberflex* Smartcard ist sehr einfach gelöst. Man kann für jeden Benutzerlevel definieren, welche Rechte er hat. Da für unser Projekt nur die Zugriffsrechte für Aut0 (Administrator), CHV1 (Kartenbesitzer) und default (alle) wichtig sind, haben alle anderen keine Zugriffsrechte und werden auch in der folgenden Tabelle nicht aufgeführt:

Datei	Aut0	CHV1	default
3F00	Alle Rechte	Create File / Delete File	keine Rechte
0000	Alle Rechte ohne Read	Write	keine Rechte
0012	Alle Rechte ohne Read	keine Rechte	keine Rechte
1012	Alle Rechte	Read	Read
2378	Alle Rechte	Read	Read

Tabelle 3.1: Zugriffsrechte auf die Dateien auf der Smartcard

3.4 Erstellen eines Benutzers

Der grundlegende Ablauf, wie ein Benutzer in unserem System erstellt wird, sollte ungefähr so aussehen:

Account erstellen Zuerst muss ein Account auf den Namen des Benutzers erstellt werden.

Zertifikat erzeugen Dann wird mit geeigneten OpenSSL Befehlen ein neues Schlüsselpaar und ein Zertifikat für den Benutzer erstellt.

Smartcard erstellen Das vorher erzeugte Schlüsselpaar und das Zertifikat muss auf der Smartcard gespeichert werden.

Zertifikat auf dem LDAP-Server speichern Das Zertifikat wird für weitere Verwendung auf dem LDAP-Server gespeichert.

Löschen der Schlüssel Damit niemand den privaten Schlüssel des Benutzers auslesen kann, wird das Schlüsselpaar von der Harddisk gelöscht.

3.5 Ablauf einer Benutzeranmeldung

Benutzernamen und Passwort abfragen

Damit Anfragen an den Benutzer sowohl im Text- als auch im Grafikmodus oder übers Netzwerk korrekt dargestellt werden, dürfen vom Modul keine direkten Ausgaben gemacht werden, sondern es müssen die von der PAM-Bibliothek dazu zur Verfügung gestellten Funktionen aufgerufen werden.

Zertifikat von der Smartcard einlesen

Das Zertifikat des Benutzers ist in einer von uns eigens dafür vorgesehenen Datei (siehe Abschnitt 3.3.1 auf Seite 36) im DER Format auf der Smartcard abgespeichert und kann von dort eingelesen werden.

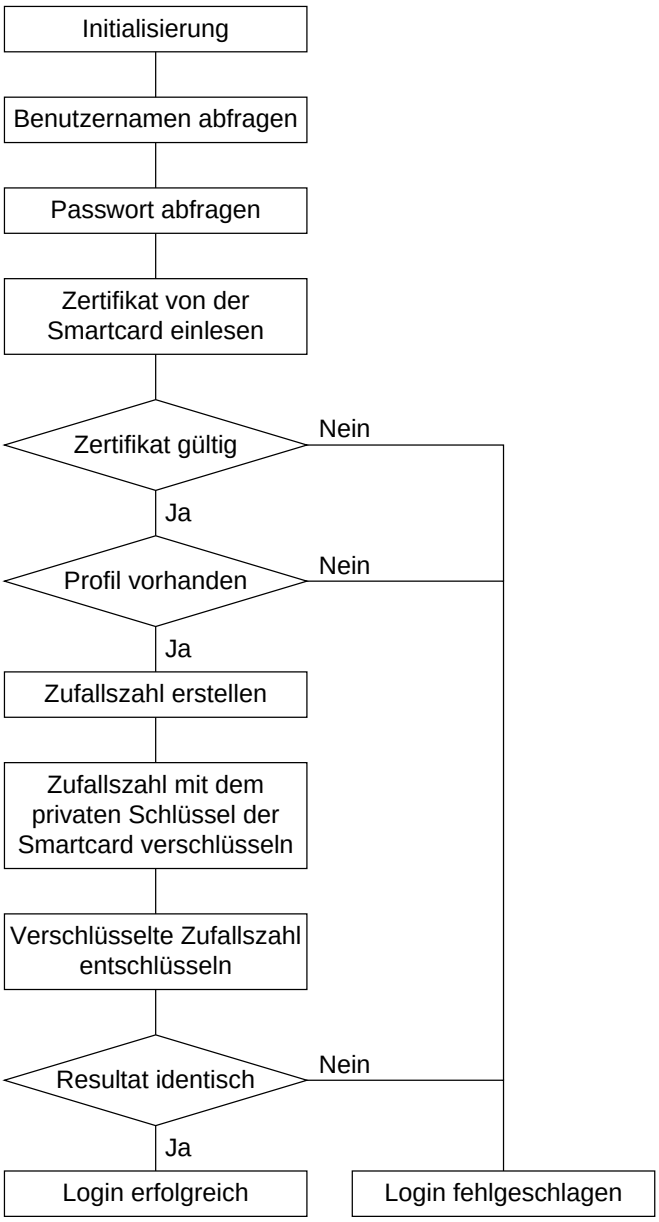


Abbildung 3.2: Ablauf einer Benutzeranmeldung

Zertifikat überprüfen

Um die Gültigkeit eines Zertifikates zu verifizieren werden folgende Punkte geprüft:

- Das aktuelle Datum muss innerhalb der Gültigkeitsdauer liegen.
- Die CRL muss verfügbar und gültig sein.
- Das Zertifikat darf in der CRL nicht vorhanden sein.
- Die Signatur muss korrekt sein.

Profil überprüfen

Ist der angegebene Benutzername nicht unter den im Zertifikat gespeicherten Profilen wird die Anmeldung abgelehnt. Mangels eines Eintrags für die Speicherung solcher Profile in einem X.509v3 Zertifikat, wird von uns, nach Absprache mit unserem Dozenten, dafür die *Subject Alternative Name* Extension verwendet. Diese darf in einem Zertifikat mehrmals vorkommen und kann unter anderem zur Speicherung von E-Mail Adressen nach [rfc822] verwendet werden. Wir nutzen dies aus, indem für jede mögliche Login-Identität ein solcher Eintrag der Form *loginname@localhost* eingefügt wird.

Zufallszahl erstellen

Linux bietet mit den beiden Devices `/dev/random` und `/dev/urandom` dem Programmierer eine einfache Möglichkeit, auf Zufallszahlen mit hoher Entropie zugreifen zu können. Diese verwenden dazu Unregelmässigkeiten in der Benutzereingabe und im Netzverkehr oder falls vorhanden, entsprechende Hardware. Da das Auslesen aus `/dev/random` blockiert falls die Entropie Anforderung nicht mehr erfüllt werden kann, verwenden wir für unsere Zwecke `/dev/urandom`. Dieses Device benutzt in einem solchen Fall zusätzlich einen internen Pseudo-Zufallsgenerator, um weiterhin Zufallszahlen liefern zu können. Die dadurch verminderte Entropie der Zufallszahl ist für unsere Zwecke jedoch mehr als ausreichend.

Zufallszahl mit dem privaten Schlüssel auf der Smartcard verschlüsseln

Die zuvor erstellte Zufallszahl wird mit einer speziellen Funktion der Smartcard verschlüsselt. Der private Schlüssel verlässt dabei die Smartcard nicht. Damit diese Funktion benutzt werden kann, muss man sich erst noch mit dem eingelesenen Passwort (PIN) auf der Smartcard anmelden. Dadurch wird verhindert, dass bei einem Verlust der Smartcard der gespeicherte, aber nicht lesbare private Schlüssel missbraucht werden kann.

Verschlüsselte Zufallszahl entschlüsseln, Resultat überprüfen

Mit Hilfe des öffentlichen Schlüssels aus dem Zertifikat wird die Antwort der Smartcard entschlüsselt. Entspricht das Resultat der ursprünglichen Zufallszahl, passen der private Schlüssel auf der Smartcard und der öffentliche Schlüssel aus dem Zertifikat zusammen und der Benutzer gilt als authentisiert. Für das Rechnen in den hier erforderlichen Grössenordnungen (mind. 1024Bit Zahlen) wird die *GNU Multiple Precision Arithmetic Library* [gnump] kurz GNU MP verwendet.

Verhalten im Fehlerfalle

Sollte beim Verifizieren des Zertifikates, bei der Kommunikation mit der Smartcard oder an einer anderen Stelle im Funktionsablauf ein Fehler auftreten, muss die Funktion korrekt beendet werden, d.h:

- der benutzte Speicher mit vertraulichem Inhalt wird überschrieben

- der allozierte Speicher wird freigegeben
- die Verbindung zur Smartcard über PC/SC wird ordnungsgemäss terminiert
- weitere, zu diesem Zeitpunkt allozierte Ressourcen werden freigegeben

3.6 LDAP-Verzeichnisstruktur

Damit wir auf den LDAP-Server mit dem NSS-Modul zugreifen können, müssen wir uns an das [rfc2307] halten. Unter anderem wird dort definiert, wie man auf die Benutzer- und die Gruppenverwaltung auf dem LDAP-Server zugreifen muss. Zusätzlich brauchen wir noch einen Speicherort für die CRL und das CA-Zertifikat.

Des weiteren sollte die Kommunikation zwischen den Clients und dem LDAP-Server verschlüsselt sein, damit diese nicht abgehört werden kann. Dazu müssen wir entweder SSL oder TLS für den LDAP-Datenverkehr einrichten.

SSL braucht für eine verschlüsselte Verbindung einen zusätzlichen Port. Demgegenüber steht TLS. Dieses arbeitet auf demselben Port wie die unverschlüsselte LDAP-Verbindung. Nach dem Verbindungsaufbau kann gewählt werden, ob die weitere Kommunikation verschlüsselt sein soll oder nicht. Dabei ist zu beachten, dass auch Benutzernamen und Passwörter verschlüsselt werden.

Da TLS als offizielle Nachfolger von SSL ein IETF Standard ist, werden wir die Verbindung mit TLS verschlüsseln. Dadurch wird auch die Konfiguration des LDAP-Servers vereinfacht, da man für SSL das LDAP-Startscript extra anpassen müsste.

3.7 Administrations-Tools

Um die Verwendbarkeit unseres Projektes zu verbessern, müssen einige Administrations-Tools erstellt werden. Dabei sollten sich die neu erstellten Programme möglichst an schon bestehende anpassen, damit sich die Benutzer nicht zu stark umstellen müssen, sondern sich gleich im neuen System zurechtfinden.

3.7.1 Benutzer- und Gruppenverwaltung

Für eine reibungslose Umstellung auf eine LDAP basierende Benutzer- und Gruppenverwaltung, sollten Programme zur Verfügung stehen, welche die zur Zeit verwendeten ersetzt. Dies ist notwendig, da Befehle wie useradd oder groupadd direkt in die entsprechenden Files schreiben und nicht über den NSS-Service gehen.

Es werden also die folgenden Programme benötigt:

Programm	ersetzt	Beschreibung
netuseradd	useradd	Fügt einen neuen Benutzer hinzu
netuserdel	userdel	Löscht einen bereits existierenden Benutzer
netusermod	usermod	Verändert einen bereits existierenden Benutzer
netgroupadd	groupadd	Fügt eine neue Benutzergruppe hinzu
netgroupdel	groupdel	Löscht eine bestehende Benutzergruppe
netgroupmod	groupmod	Verändert eine bestehende Benutzergruppe

Zusätzlich sollten noch Programme vorliegen, die die Benutzer- und Gruppendatenbanken im gleichen Format zurückgeben, wie sie in den Dateien `/etc/passwd` und `/etc/group` auch vorhanden sind. Dafür sind die folgenden zwei Programme vorgesehen:

Programm	ersetzt	Beschreibung
netuserls	`/etc/passwd`	Gibt die Benutzerdatenbank aus
netgroupls	`/etc/group`	Gibt die Gruppendatenbank aus

3.7.2 Account-Verwaltung

Da wir nur den Benutzern Zugriff auf das System gewähren, die ein gültiges Zertifikat besitzen, ergibt es sich, dass zwei verschiedene Arten von Benutzern im LDAP-Verzeichnis existieren: solche mit und solche ohne Zertifikat.

Der Grund für dieses Verhalten kommt daher, dass es Benutzer gibt, die nur als Aliase (root, postmaster, webadmin usw.) oder dem Betrieb eines Programms (daemon, at, www-run usw.) dienen. Dies sind jedoch keine *richtigen* Benutzer, respektive Personen und dürfen folglich auch kein Zertifikat besitzen.

Um dennoch eine Anmeldung als zum Beispiel root zu ermöglichen, können einem *richtigen* Benutzer diverse Profile zugewiesen werden. Den Benutzern Meier und Müller kann so erlaubt werden, unter der Kennung root zu arbeiten, ohne dass sie ihre wahre Identität verlieren.

Mit dem Programm **netuseradd** wird ein Benutzer im LDAP-Verzeichnis eingetragen, dieser besitzt aber weder Zertifikat noch Smartcard und ist somit auch nicht Login-fähig. Um aus einem solchen Benutzer einen *richtigen* Benutzer zu erstellen, und diesen wieder zu entfernen, benötigen wir zusätzliche Programme:

Programm	Beschreibung
netaccountadd	Wandelt einen existierenden Useraccount in einen real existierenden Benutzer um und fügt ein Zertifikat in die Datenbank ein. Gleichzeitig werden Private- und Public-Key, sowie das Zertifikat auf eine Smartcard gespeichert.
netaccountdel	Löscht den real existierenden Benutzer und wandelt diesen wieder in einen Useraccount um. Dabei wird das Zertifikat gesperrt und in der Benutzerdatenbank gelöscht.

3.7.3 CRL aktualisieren

Da die CRL in einem X.509 System abläuft, muss gewährleistet werden, dass diese aktualisiert wird. Damit dies nicht in eine mühselige Arbeit ausartet, sollte auch dafür ein Programm vorhanden sein, das diese Arbeit ausführt.

Programm	Beschreibung
update_crl	Erstellt eine neue CRL und speichert sie im entsprechenden LDAP-Eintrag.

Weil der private Schlüssel der CA mit einer Passphrase geschützt ist, kann die Aktualisierung der CRL leider nicht automatisiert werden.

3.7.4 Smartcard Administration

Damit der Umgang mit den Smartcards vereinfacht wird, sollte ein Administrationsprogramm für diese erstellt werden.

Dabei stehen die folgenden Aufgaben im Vordergrund:

Programm	Beschreibung
smartcard_admin unblock	Enstperrt eine gesperrte Karte.
smartcard_admin changeaut0	Ändert den Administrator-PIN.
smartcard_admin changeupin	Ändert den Entsperr-PIN.

3.7.5 Installations-Hilfen

Für die Installation unseres Projektes sollten Scripts vorhanden sein, die eine mehr oder weniger automatische Installation erlauben. Dabei ist es einerseits wichtig, eine korrekte CA aufzubauen, andererseits den LDAP-Server für unsere Zwecke vorzubereiten.

Programm	Beschreibung
create_net_login_ca.pl	Soll anhand des vorkonfigurierten Systems die CA auf dem CA-Server aufbauen.
create_net_login_ldap.pl	Soll eine für uns verwendbare Struktur im LDAP-Baum erstellen.

3.8 Funktionsweise von OpenSSH

Um OpenSSH für unser Projekt benutzbar zu machen, müssen wir zuerst seine innere Struktur kennen und verstehen lernen. Es stellt sich heraus, dass in der neusten Version schon eine einfache Smartcard Unterstützung vorhanden ist. Deren Benutzung kommt allerdings für uns aus folgenden Gründen nicht in Frage:

- Die Implementierung setzt nicht auf PC/SC auf.
- Es muss ein spezielles Cardlet auf der Smartcard installiert werden.
- Auf die Eingabe eines Passwortes (PIN) wird verzichtet.

Daher sind wir gezwungen, eine eigene Lösung zu verwirklichen.

3.8.1 Aufbau des OpenSSH Paketes

Abbildung 3.3 zeigt, welche Quelldateien den einzelnen Programmen zugeordnet sind. Die zugehörigen Header-Dateien wurden zur besseren Übersicht weggelassen. Markant ist die von allen Applikationen benutzte Bibliothek **libssh**. Sie beinhaltet neben den verwendeten Verschlüsselungsroutinen auch Funktionen zur Analyse der Konfigurationsdateien.

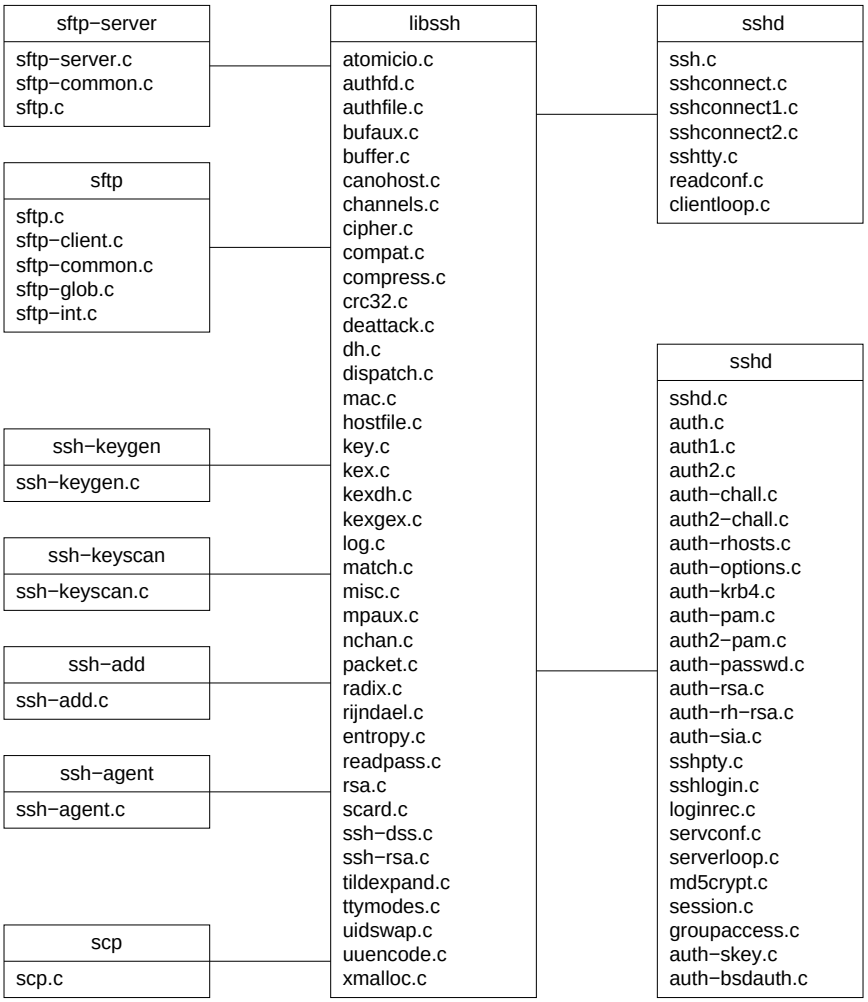


Abbildung 3.3: Dateistruktur des OpenSSH Paketes (Version 2.9.9p2)

3.8.2 Ablauf eines SSH-Logins

In Abbildung 3.4 ist zu sehen, wie ein auf dem Public-Key Verfahren basierender Benutzerlogin abläuft. Daneben bietet OpenSSH noch die Möglichkeit, eine rein Host- oder Passwort-basierende Anmeldung durchzuführen. Diese sind für uns jedoch nicht von Interesse und werden daher auch nicht näher betrachtet.

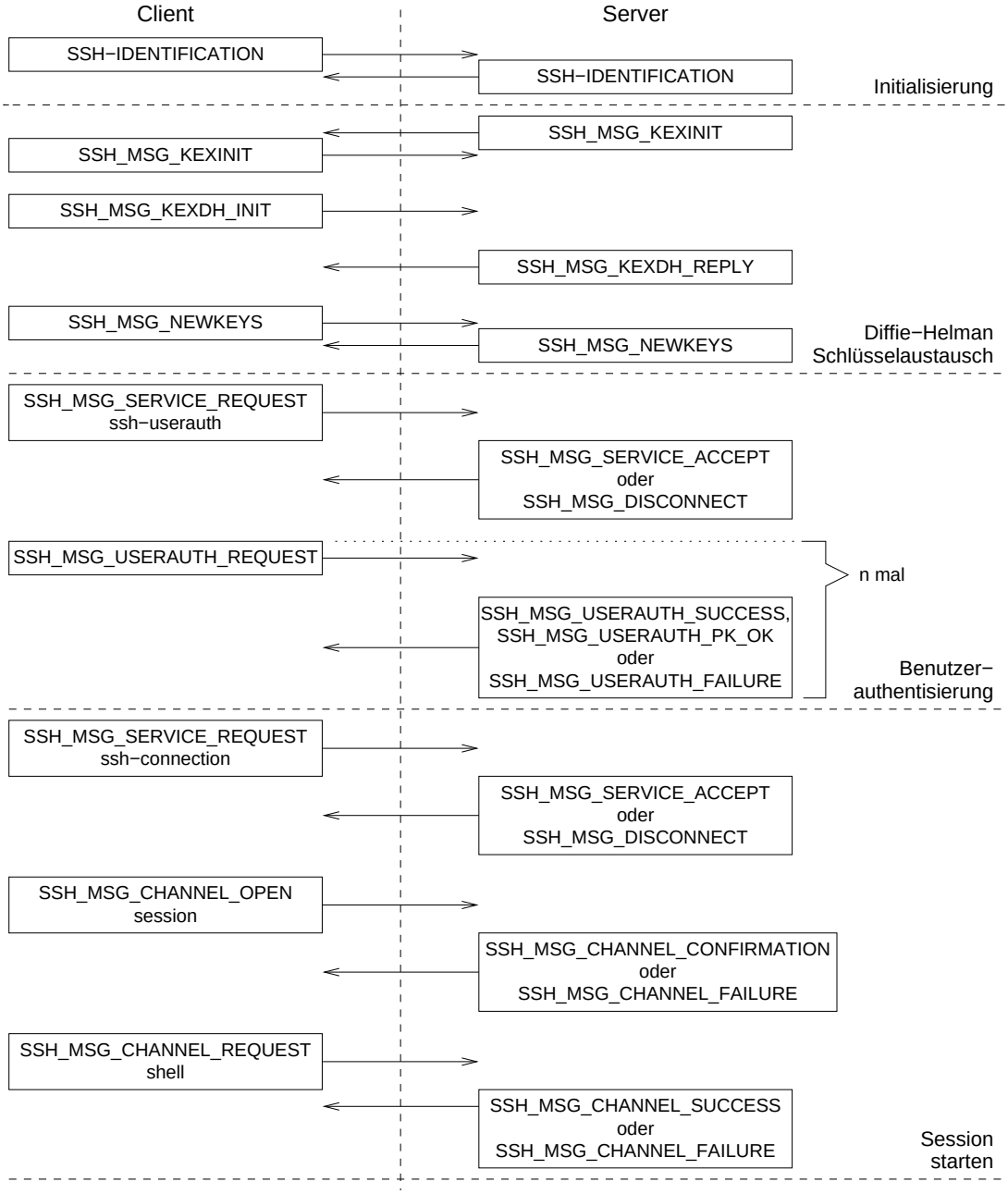


Abbildung 3.4: Ablauf eines SSH-Logins mit dem Public-Key Verfahren

Initialisierung Nach dem Aufbau der TCP/IP-Verbindung senden Server und Client eine SSH-IDENTIFICATION Meldung. Diese beinhaltet einen Identifikations-String, der die aktuelle Protokoll- und Software-Version enthält.

Diffie-Helman Schlüsselaustausch Nun senden sich beide Seiten ein Liste der bekannten Verschlüsselungsalgorithmen mittels einer SSH_MSG_KEXINIT Meldung zu. Wurde ein gemeinsamer Nenner gefunden, wird mittels eines DH-Schlüsselaustausches (SSH_MSG_KEXDH_INIT und SSH_MSG_KEXDH_REPLY) der entsprechende Session-Key sowie ein eindeutiger Session-Identifier gebildet. Als Teil der SSH_MSG_KEXDH_REPLY Meldung, sendet der Server einen, mit seinem privaten Host-Key signierten, Hash-Wert des Paketes sowie seinen öffentlichen Host-Key mit. Der Client kann nun die Echtheit des Servers überprüfen, indem er zuerst den öffentlichen Schlüssel überprüft (z.B. mit einer Tabelle oder einem Zertifikat) und anschliessend die Signatur verifiziert.

Benutzerauthentisierung Eine Benutzerauthentisierung wird vom Client durch eine SSH_MSG_SERVICE_REQUEST: userauth Meldung eingeleitet. Diese kann vom Server mittels SSH_MSG_SERVICE_ACCEPT akzeptiert oder durch SSH_MSG_DISCONNECT abgelehnt werden. Als nächstes sendet der Client eine SSH_MSG_USERAUTH_REQUEST:none Meldung. Wird vom Server eine reine Host-basierende Authentisierung verwendet, ist diese hiermit bereits abgeschlossen und der Server sendet ein SSH_MSG_USERAUTH_SUCCESS Paket zurück. Ansonsten wird dem Client mit der Antwort (SSH_MSG_USERAUTH_FAILURE) eine Liste der möglichen Authentisierungsmethoden übermittelt.

Bei der für uns interessanten *publickey* Methode folgt als nächstes eine weitere SSH_MSG_USERAUTH_REQUEST:publickey Nachricht, die unter anderem den öffentlichen Schlüssel des Benutzers enthält. Der Server überprüft nun, ob dieser Schlüssel zulässig ist (z.B. mit einer Tabelle). Falls er ihn akzeptiert, teilt er das dem Client mit einer SSH_MSG_USERAUTH_PK_OK Meldung mit. Ansonsten wird mit einem SSH_MSG_USERAUTH_FAILURE abgebrochen. War der öffentliche Schlüssel gültig, sendet der Client zum dritten Mal eine SSH_MSG_USERAUTH_REQUEST: publickey Nachricht. Dieses Mal enthält diese zusätzlich eine mit dem privaten Schlüssel unterschriebene Signatur. Der Server überprüft diese und sendet, falls alles in Ordnung war, ein SSH_MSG_USERAUTH_SUCCESS Paket zurück, ansonsten wird mit einer SSH_MSG_USERAUTH_FAILURE Meldung abgebrochen.

Session Start Mit den Nachrichten SSH_MSG_SERVICE_REQUEST: ssh-connection und SSH_MSG_CHANNEL_OPEN kann nun eine Datenverbindung zum Server geöffnet werden. Der so geöffnete Datenkanal kann für die verschiedensten Zwecke verwendet werden. Als Beispiel ist in Abbildung 3.4 auf der vorherigen Seite gezeigt, wie eine Remote-Shell geöffnet werden kann.

Kapitel 4

Realisierung des Projekts

4.1 PAM-Modul und X.509 Parser

Das PAM-Modul, das für den eigentlichen Anmeldevorgang verantwortlich ist, sollte möglichst unabhängig und flexibel sein. Dies erreichten wir, indem alle vom Modul benötigten Parameter entweder im Zertifikat gespeichert oder dem PAM-Modul als Argumente übergeben werden.

Für das Analysieren und Validieren der X.509 Zertifikate konnten wir auf bestehende Routinen des Linux IPSec Projektes zurückgreifen. Diese mussten von uns aus dem Paket extrahiert, angepasst und erweitert werden.

Im Folgenden wird zuerst das eigentliche PAM-Modul erläutert, bevor auf die relativ umfangreichen Anpassungen und Erweiterungen des X.509 Parsers eingegangen wird. Zum Schluss werden die für die Kommunikation mit der Smartcard zuständigen Funktionen beschrieben.

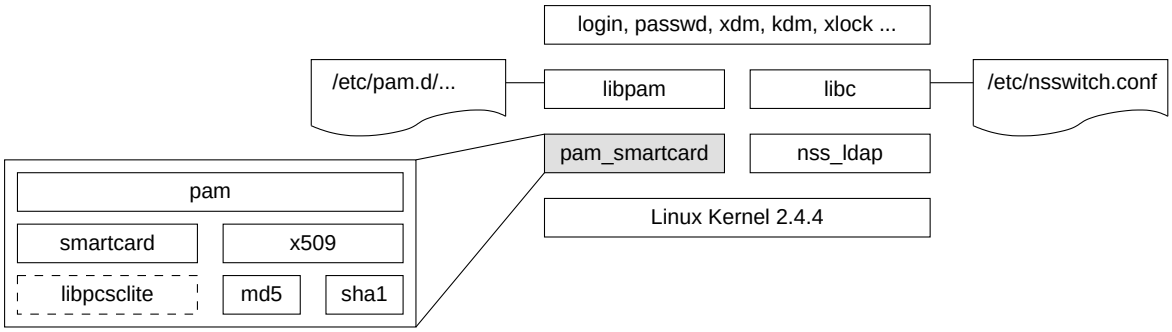


Abbildung 4.1: Aufbau des PAM-Moduls und seine Einordnung im System

4.1.1 Funktionsweise des PAM-Moduls

Das von uns entwickelte PAM-Modul (`pam_smartcard`) hat im Wesentlichen zwei Aufgaben, welche im Folgenden noch genauer erläutert werden.

1. Authentisierung des Benutzers (Funktion `pam_sm_authenticate()`)
2. Ändern des Passwortes auf der Smartcard (Funktion `pam_sm_chauthtok()`)

Wie in [pammod] beschrieben wurden auch alle weiteren, von uns nicht unterstützten Funktionen implementiert. Wie vom Standard verlangt, schreiben diese eine Meldung in die Sys-Log Datei und kehren mit einem Fehler zurück:

```
PAM_EXTERN int pam_sm_X_Y(pam_handle_t *pamh, int flags, int argc,
                          const char **argv)
{
  openlog(LOGNAME, LOG_CONS | LOG_PID, LOG_AUTHPRIV);
  syslog(LOG_WARNING, "Function pm_sm_X_Y() not implemented in this module");
  closelog();
  return PAM_SERVICE_ERR;
}
```

Authentisierung des Benutzers

Tritt im folgenden Ablauf (siehe auch Abbildung 3.2 auf Seite 39) an irgend einer Stelle ein Fehler auf, werden alle Ressourcen freigegeben und die Modul Funktion mit einem entsprechenden Fehler terminiert:

```
if (some_kind_of_error) {
  syslog(LOG_ERR, "error message, describing some_kind_of_error");
  pcsc_release();
  pam_release_data(sd);
  ...
  return PAM_SOME_KIND_OF_ERROR;
}
```

- Als Erstes werden die dem Modul übergebenen Argumente analysiert und ausgewertet:

  reader=*n* Benutze den Smartcard-Reader Nummer *n*. Als Standard wird Reader Nummer 0 verwendet.

  cadir=*path* Gibt den Pfad zu den Root CA Zertifikaten an. Fehlt die Angabe wird im Verzeichnis `/etc/cacerts/` gesucht.

  try_first_pass Verwende ein bereits eingelesenes Passwort.

  use_first_pass Verwende ein bereits eingelesenes Passwort. Falls keines vorhanden ist, beende mit einer Fehlermeldung.

  Wurden noch weitere Argumente angegeben werden diese mit einer Warnmeldung in Sys-Log protokolliert.

- Der benötigte Speicher wird alloziert.
- Als nächstes werden der Benutzername und das Passwort, unter Berücksichtigung der übergebenen Argumente, eingelesen. Das heisst: Falls **try_first_pass** oder **use_first_pass** angegeben wurden, wird versucht, das Passwort mittels der PAM-Bibliotheksfunktion `pam_get_item()` aus dem PAM-Speicher auszulesen. Wurde keines

gefunden und **use_first_pass** ist gesetzt, terminiert die Funktion mit einem Fehler. Ansonsten wird eine Eingabeaufforderung ausgegeben und beim Benutzer nachgefragt. Wie bereits erwähnt, darf diese Ausgabe nicht direkt vom Modul ausgeführt werden, sondern nur indirekt durch Aufruf einer dafür vorgesehenen Funktion.

- Nun wird der Smartcard-Reader initialisiert und eine Verbindung zur Karte hergestellt (`pcsc_init()`). Anschliessend wird das Root-File selektiert (`selectFile()`) und der Kartenbesitzer CHV1 mit Hilfe des Passwortes auf der Karte angemeldet (`verifyCHV()`).

- Das Zertifikat wird von der Smartcard gelesen (`readBinary()`) und analysiert (`parse_x509cert()`).

- Um das Zertifikat überprüfen zu können, müssen noch die Root CA Zertifikate und die CRL eingelesen werden. Die CA Zertifikate werden von dem, durch das Argument **cadir** angegebenen Verzeichnis geladen (`load_cacerts()`), die URI der CRL ist durch die *CRL Distribution Point* Extension des Benutzer-Zertifikates gegeben (`load_crl()`).

- Anschliessend kann das Benutzer-Zertifikat verifiziert werden (`verify_x509cert()`)

- Falls das Zertifikat gültig war, werden die einzelnen Benutzerprofile ausgelesen und überprüft, ob der am Login-Prompt angegebene Name enthalten ist. Konnte der Benutzer nicht gefunden werden, ist eine Anmeldung unter dieser Kennung nicht erlaubt und es wird mit einem Fehler terminiert.

- Als nächstes wird eine Zufallszahl erstellt. Dazu werden 128 Byte (1024 Bit) aus dem Random-Device `/dev/urandom` ausgelesen. Anschliessend wird diese durch die Smartcard verschlüsselt (`internalAuth()`) und in eine GNU MP Integer Zahl umgewandelt.

- Der public Key wird aus dem Zertifikat ausgelesen und die verschlüsselte Zahl damit entschlüsselt. Entspricht der so erhaltene Wert der ursprünglichen Zufallszahl gilt der Benutzer als authentisiert.

```
/* get public key from certificate */
n_to_mpz(n, sd->cert.modulus.ptr, sd->cert.modulus.len);
n_to_mpz(e, sd->cert.publicExponent.ptr, sd->cert.publicExponent.len);

/* decode cipher */
n_to_mpz(y, sd->cipher, KEY_SIZE);
mpz_init(x);
/* x = y ^ e mod n */
mpz_powm(x, y, e, n);

/* compare original value and result from smartcard */
n_to_mpz(z, sd->rand, KEY_SIZE);
if (!mpz_cmp(z, x)) result = PAM_SUCCESS;
else result = PAM_AUTH_ERR;
```

- Zuletzt wird die Verbindung zum Smartcard-Reader getrennt und alle allozierten Ressourcen wieder freigegeben.

Ändern des Passwortes

Auch bei dieser Funktion werden im Fehlerfalle, analog zur Authentisierung des Benutzers, alle Ressourcen freigegeben und mit einem Fehler terminiert.

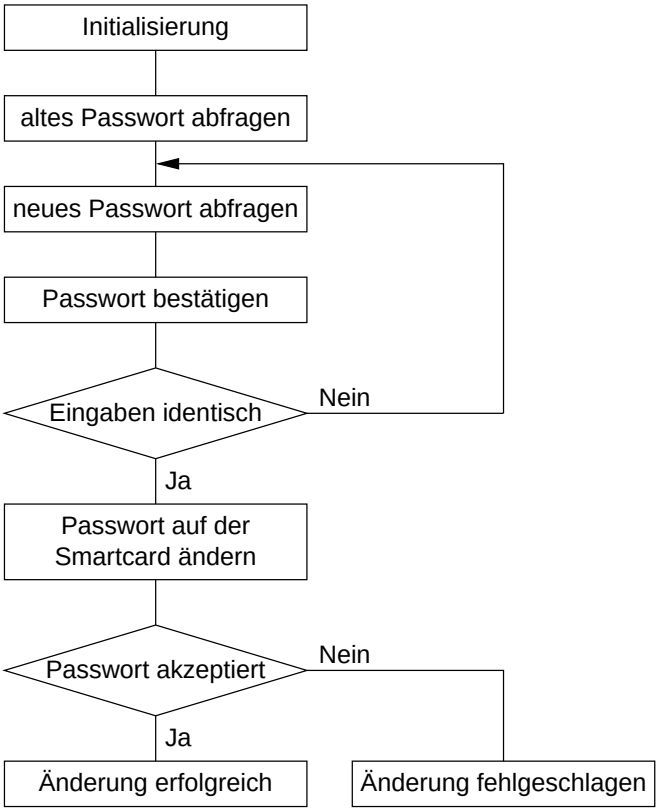


Abbildung 4.2: Ablauf einer Passwort-Änderung auf der Smartcard

- Als Erstes werden die dem Modul übergebenen Argumente analysiert und ausgewertet:

  reader=*n* Benutze den Smartcard-Reader Nummer *n*. Als Standard wird Reader Nummer 0 verwendet.

  use_authtok Verwende ein bereits eingelesenes neues Passwort.

  use_first_pass Verwende ein bereits eingelesenes Passwort. Falls keines vorhanden ist, beende mit einer Fehlermeldung.

  Wurden noch weitere Argumente angegeben, werden diese mit einer Warnmeldung in die Sys-Log-Datei protokolliert.

- Der benötigte Speicher wird alloziert.

- Nun wird das ursprüngliche Passwort des Benutzers abgefragt. Falls **use_first_pass** gesetzt wurde, wird versucht ein bereits vorhandenes zu verwenden. Ist keine solches vorhanden, terminiert die Funktion mit einem Fehler.

- Wurde **use_authtok** als Argument angegeben, wird zuerst versucht ein bereits vorhandenes neues Passwort zu verwenden. Existiert keines, wird das neue Passwort zwei Mal abgefragt. Dies wird solange wiederholt, bis beide Eingaben identisch sind.
- Nun wird der Smartcard-Reader initialisiert und eine Verbindung zur Karte hergestellt (`pcsc_init()`). Anschliessend wird das Root-File selektiert (`selectFile()`) und das Passwort von CHV1 mit Hilfe der Funktion `changeCHV()` geändert. Der Funktion müssen sowohl das alte als auch das neue Passwort übergeben werden.
- Zuletzt wird die Verbindung zum Smartcard-Reader getrennt und alle allozierten Ressourcen wieder freigegeben.

4.1.2 Anpassung des X.509 Parsers

X.509v3 Extensions

Der ursprüngliche Parser unterstützte nur **einen** *Subject Alternative Name*. In den von uns verwendeten Zertifikaten können jedoch beliebig viele solcher Erweiterungen vorkommen. Zusätzlich verwenden wir die Extension *CRL Distribution Point*.

Das Hinzufügen der zusätzlichen X.509v3 Extension erforderte einiges an Aufwand. Neben der Anpassung einer internen Struktur, welche den Aufbau des zu parsenden ASN.1 Objektes definiert, mussten mehrere Konstanten hinzugefügt, respektive modifiziert werden. Des Weiteren musste die Kernroutine des Parsers so geändert werden, dass alle bekannten Extensions ausgewertet und unbekannte überlesen werden. Zuletzt mussten noch einige Variablen in verkettete Listen umgewandelt werden, um das Ablegen von mehr als einem Wert zu ermöglichen.

Laden der CRL mittels einer URI

Bei einem auf X.509 Zertifikaten basierenden Login ist die einzige Möglichkeit, einen Benutzer zu sperren, das Widerrufen von dessen Zertifikat. Die CRL ist daher für uns von zentraler Bedeutung. Beim ursprünglichen Parser befand sich diese im gleichen Verzeichnis wie die Root CA Zertifikate auf der lokalen Festplatte. Eine solche Lösung wäre für uns undenkbar, da viel zu träge und unsicher. Wie bereits erwähnt verwenden wir die X.509v3 Extension *CRL Distribution Point*. In dieser wird ein URI gespeichert, der angibt, wo die CRL zu finden ist.

Als Folge wurde die Funktion `load_crl()` komplett neu geschrieben. Unsere Implementation erwartet als Argument die URI der CRL und gibt **TRUE** zurück, falls diese geladen werden konnte sowie gültig ist, ansonsten **FALSE**. In allen Fällen wird die CRL im DER Format erwartet. Zur Zeit werden die Protokolle **file**, **http** und **ldap** unterstützt, wobei **file** nur lokale Dateisysteme unterstützt, d.h. eine (optionale) Angabe des Hosts wird ignoriert. Erwähnenswert ist, dass vom Parser mehrere CRLs gleichzeitig unterstützt werden, da diese in einer Liste gespeichert werden. Der wiederholte Aufruf von `load_crl()` überschreibt folglich eine schon gespeicherte CRL nicht, sondern fügt einen neuen Eintrag in die Liste ein. Im Folgenden wird der komplette Funktionsablauf kurz erläutert.

- Als Erstes wird die übergebene URI analysiert und in die drei Bestandteile: Protokoll, Host und Pfad unterteilt.
- Anschliessend wird abhängig vom angegebenen Protokoll die CRL geladen:

  file Der Host Eintrag wird ignoriert, die durch den Pfad angegebene Datei geöffnet und aus ihr die CRL geladen.

  http Es wird eine TCP Verbindung zum http-Port (80) des angegebenen Hosts geöffnet. Anschliessend wird mit dem GET Kommando die durch den Pfad definierte CRL geladen. Vom Web-Server zurückkommende REDIRECT Anweisungen werden ausgewertet und die CRL vom neuen Zielort geladen.

  ldap Eine anonyme LDAP-Verbindung wird zum Rechner Host aufgebaut und das durch path definierte Objekt geladen.
- Das zur CRL passende CA Zertifikat wird gesucht und gegebenenfalls die Signatur der CRL damit verifiziert. Konnte keine zugehöriges Zertifikat gefunden werden, wird mit einem Fehler abgebrochen.
- Zusätzlich wird überprüft, ob die CRL schon gültig aber noch nicht abgelaufen ist.
- Waren alle Tests erfolgreich, wird die geladene CRL in die Liste eingefügt und die Funktion erfolgreich beendet.

Der angepasste Parser wurde im Modul **x509** (.c und .h Dateien) gespeichert. Eine ausführliche Beschreibung aller Funktionen, Parameter und Rückgabewerte kann der Header-Datei `x509.h` entnommen werden. Zusätzlich mussten die beiden Module **md5** und **sha1** aus dem Linux IPSec Projekt eingebunden werden, da diese vom Parser für die Überprüfung der Signaturen benötigt werden.

4.1.3 Smartcard Routinen

Das Modul **smartcard** ist für die Kommunikation mit der Smartcard zuständig und konnte grösstenteils aus unserer früheren Projektarbeit übernommen werden. Es folgt eine kurze Erläuterung der wichtigsten Funktionen. Eine ausführliche Beschreibung aller Funktionen, Parameter und Rückgabewerte kann der Header-Datei `smartcard.h` entnommen werden.

pcsc_init() Erstellt eine Verbindung zum Smartcard-Reader. Es muss die Nummer des zu verwendenden Readers angegeben werden.

pcsc_release Trennt eine mit **pcsc_init** erstellte Verbindung.

verifyCHV() Verifiziert den Besitzer (cardholder) gegenüber der Karte.

changeCHV() Ändert das Passwort (PIN) des Kartenbesitzers. Es müssen das alte und das neue Passwort angegeben werden.

internalAuth() Verschlüsselt die übergebenen Daten mit dem gespeicherten privaten Schlüssel. Damit die Funktion benützbar ist, muss sich der Kartenbesitzer zuerst gegenüber der Karte verifizieren.

selectFile() Selektiert eine Datei.

readBinary() Liest aus einer zuvor selektierten Datei. Es müssen der Offset zum Dateianfang sowie die Anzahl zu lesender Bytes übergeben werden.

4.2 Installations-Scripts

Wie bereits in der Analyse unter Kapitel 3.7 auf Seite 41 erwähnt, wäre es hilfreich, die Installation des Projektes durch einige Scripts zu unterstützen. Deshalb sind während des Projektes auch die folgenden zwei Installations-Scripts entstanden:

`create_net_login_ca.pl` Erstellt auf dem CA-Server anhand der Konfigurationsdateien eine Zertifizierungsstelle (CA).

`create_net_login_ldap.pl` Erstellt anhand der Konfigurationsdateien die Struktur auf dem LDAP-Server.

4.2.1 create_net_login_ca.pl

Dieses Script ist verantwortlich für die Erstellung der CA auf dem CA-Server. Bevor dieses Script gestartet wird, müssen die Konfigurationsdateien `/etc/smartcard_netlogin` und `ldap.conf` angepasst werden, wie in Kapitel 5.4 auf Seite 77 beschrieben.

Anhand der Angaben in diesen Dateien werden die folgenden Schritte ausgeführt:

Verzeichnis erstellen Hier wird anhand des Parameters **ca_dir** in der Datei `/etc/smartcard_netlogin` das Verzeichnis für die CA erstellt. Dabei wird rekursiv geprüft, ob das Verzeichnis schon existiert. Falls ja, wird das Script abgebrochen und die Meldung ausgegeben, dass man das Verzeichnis zuerst löschen soll, wenn man die neue CA dort hinein schreiben will. Es wird ebenfalls abgebrochen, wenn es eine Datei mit dem Namen des CA-Verzeichnisses gibt.

Nach dem Erstellen des Verzeichnisses werden die drei Unterverzeichnisse `certs`, `crl` und `private` erstellt, womit die Struktur für die CA festgelegt wird.

Erstellen der Konfigurationsdateien für die CA In diesem Abschnitt werden anhand der Parameter **ca_dir** und **ca_rdn** der Datei `/etc/smartcard_netlogin` und **base** der Datei `ldap.conf` die Konfigurationsdateien für die CA erstellt. Dabei werden die folgenden Dateien in das Verzeichnis der CA geschrieben:

- PCA.cnf
- SCA.cnf
- ldapcert.cnf

`PCA.cnf` wird für das Generieren des selbstunterschriebenen CA-Zertifikats und für das Erstellen der CRL verwendet. `SCA.cnf` braucht es für das Erstellen des Server-Zertifikats und `ldapcert.cnf` gilt als Basis für die Generierung der einzelnen Benutzerzertifikate.

CA erstellen Hier wird die eigentliche CA erstellt. Zuerst wird ein selbstunterschriebenes CA-Zertifikat erstellt, für dessen private Key man eine Passphrase angeben muss. Danach wird eine leere CRL generiert. Da es bedingt durch den LDAP-Aufbau auch eine ARL braucht, wird diese kopiert.

Danach wird das Zertifikat und die CRL/ARL in das DER-Format umgewandelt. Zum Schluss wird noch das Server-Zertifikat erstellt.

Obwohl hier nur wenige der Einstellungen in den Konfigurationsdateien verwendet werden, überprüfen wir alle auf Vorhandensein, damit zu einem späteren Zeitpunkt keine bösen Überraschungen auftreten, weil nicht alles richtig konfiguriert wurde.

4.2.2 create_net_login_ldap.pl

Dieses Script erstellt die Struktur auf dem LDAP-Server. Abbildung 4.3 zeigt einen möglichen Aufbau des Verzeichnisbaumes.

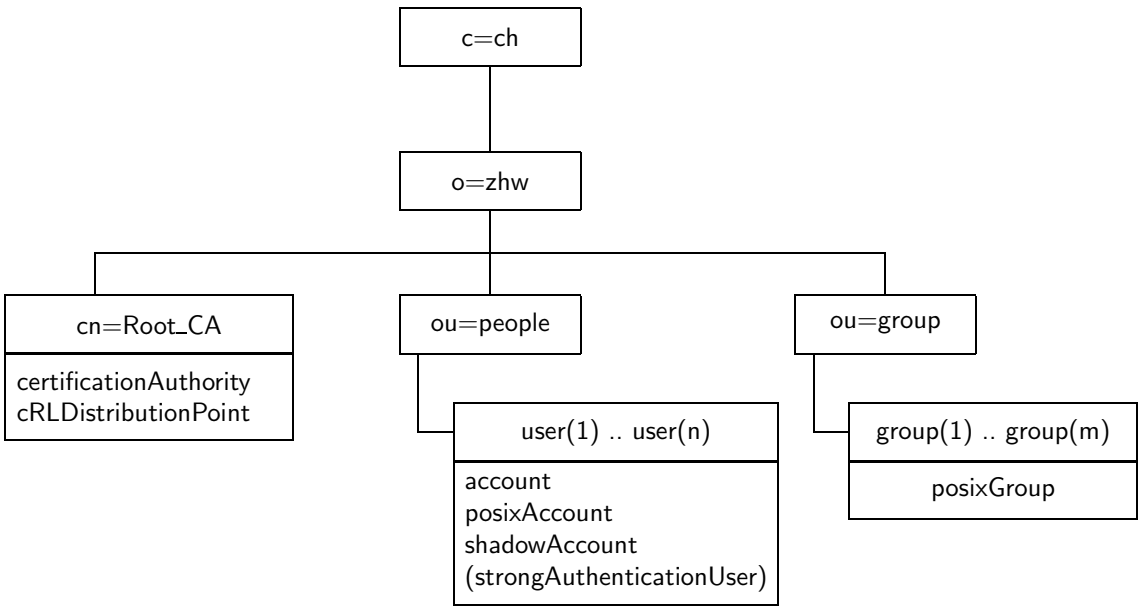


Abbildung 4.3: Möglicher Verzeichnisbaum auf dem LDAP-Server

Zuerst wird das Objekt Root_CA in den Verzeichnisbaum eingefügt. Obwohl dieses für unser Projekt immer direkt unter der *Base* des LDAP-Servers ist, wird dieses Objekt rekursiv erstellt. Dadurch werden auch die in diesem Beispiel angegebenen Container *c=ch* und *o=zhw* gleich mit erstellt, falls sie nicht vorhanden sind. Mit den beiden Containern *ou=group* und *ou=people* wird gleichermassen vorgegangen. Zu beachten ist, dass wenn diese Objekte schon bestehen, keine Änderungen angebracht werden.

Als nächstes wird geprüft, ob das Root_CA Objekt im LDAP-Verzeichnisbaum schon existiert. Falls ja, wird nachgefragt, ob sie ersetzt werden soll.

Zum Schluss werden noch die CRL, die ARL und das CA-Zertifikat auf den LDAP-Server in das Root_CA Objekt gespeichert. Da die ARL wegen der LDAP-Struktur vorhanden sein muss, wir diese aber für unser Projekt nicht brauchen, speichern wir eine Kopie der leeren CRL. Falls man die ARL auch benutzen will, müsste man dafür sorgen, dass diese ebenfalls periodisch aktualisiert wird, da auch diese nach einer gewissen Zeit abläuft.

Nach diesem Ablauf sind noch keine Benutzer und Gruppen auf dem LDAP gespeichert. Diese müssen entweder mit **netuseradd** und **netgroupadd** oder mit einem der unter Kapitel 5.5 auf Seite 82 beschriebenen Migrations-Scripts erstellt werden.

4.3 LDAP Verzeichnisstruktur

Wir mussten darauf achten, dass das LDAP-NSS-Modul auf die Daten des LDAP-Servers zugreifen kann. Deshalb waren wir mehr oder weniger an gewisse Regeln gebunden.

Die Abbildung 4.3 auf der vorherigen Seite zeigt einen möglichen Aufbau der LDAP Verzeichnisstruktur. Die einzelnen Objekte werden im Folgenden noch genauer beschrieben:

Objekt	Schemen	Beschreibung
Root_CA	certificationAuthority cRLDistributionPoint	Beinhaltet die CRL und das Zertifikat der Zertifizierungsstelle.
group	posixGroup	Hier wird die Gruppenverwaltung gespeichert.
people	account posixAccount shadowAccount strongAuthenticationUser	Ist für die Benutzerverwaltung verantwortlich.

Root_CA Dies ist der zentrale Speicherort für die CRL. Zusätzlich wird hier das CA-Zertifikat abgelegt. Obwohl es für unser Projekt nicht benötigt wird, muss, gegeben durch das Schema *certificationAuthority*, auch noch die ARL gespeichert werden. Dies ist bei uns einfach eine Kopie der anfänglich erstellten leeren CRL.

group Damit das LDAP-NSS-Modul auf die Gruppeneinträge zugreifen kann, müssen die einzelnen Objekte das Schema *posixGroup* enthalten. Es sind nur sehr wenige Attribute notwendig, damit das Ganze funktioniert: Im Attribut **cn** (common name) wird der Name der Gruppe abgelegt. Die Nummer der Gruppe wird in **gidNumber** gespeichert.

Damit festgehalten werden kann, welche Benutzer zu einer Gruppe gehören, können mehrere Attribute vom Typ **memberUID** in einem Gruppenobjekt vorhanden sein. Jedes dieser Attribute enthält dann den Namen eines Benutzers.

people Diese Objekte sind die komplexesten, die wir in unserem Projekt brauchen. Die Tabelle 4.1 auf der nächsten Seite zeigt die für NSS notwendigen Attribute und deren Bedeutung.

Attribut	Beschreibung
uid	Hier wird der Login-Name des Benutzers abgelegt.
cn	Speichert den vollen Namen (common name) des Benutzers.
uidNumber	Repräsentiert die Benutzernummer.
gidNumber	Gibt die Hauptgruppen-Nummer des Benutzers an.
homeDirectory	Zeigt auf das Home-Verzeichnis des Benutzers.
loginShell	Gibt an, welche Shell der Benutzer beim Einloggen hat.
userPassword	Speichert das Passwort des Benutzers. Wird aber von unserem Projekt nicht verwendet und mit einem x gesperrt.
shadowInactive	Anzahl Tage, die der Benutzer schon gesperrt ist.
shadowExpire	Anzahl Tage, ab wann der Benutzer gesperrt werden wird (Gezählt ab 1.1.1970).
shadowMin	Anzahl Tage, in denen der Benutzer das Passwort nicht ändern kann.
shadowMax	Anzahl Tage, in denen der Benutzer das Passwort geändert haben muss.
shadowWarning	Anzahl Tage vor der Sperrung des Passworts, in denen der Benutzer gewarnt wird.
shadowFlag	Wird für spätere Benutzung reserviert.

Tabelle 4.1: Attribute für people-Objekte

4.3.1 Konfiguration

Für das NSS-Modul gibt es in der Konfigurations-Datei `ldap.conf` die Parameter **nss_base_passwd** und **nss_base_group**, die festlegen, wo im LDAP-Verzeichnisbaum die Informationen für die Benutzer- und Gruppenverwaltung stehen.

Damit wir diese Parameter nicht noch einmal speichern müssen, greifen wir direkt auf diese Eintragungen zu und benutzen diese in unseren Scripts und Tools.

Um sicherzustellen, dass unsere Programme auch das benötigte Administrator-Passwort kennen, haben wir uns an das System des PAM-LDAP Moduls gehalten. Dort wird das Passwort in einer Datei `ldap.secret` gespeichert und ausgelesen. Dadurch muss das Passwort nicht an mehreren Stellen konfiguriert werden, sondern nur einmal an einem definierten Ort, der zudem vor unbefugtem Auslesen geschützt werden kann.

4.4 Benutzer-Administration

4.4.1 netuseradd

Wie der Name sagt, können mit diesem Tool Netzwerkbenutzer ins LDAP-Verzeichnis eingetragen werden. Durch Angabe des Parameters **-D** wird kein neuer Benutzer angelegt, sondern die angegebenen Werte werden als Standardwerte für neue Benutzer verwendet. Beim Original wurden diese lokal in der Datei `/etc/default/useradd` gespeichert. Wir

legen diese Werte im LDAP-Verzeichnis unter dem Benutzernamen *default$* mit numerischer ID 99 ab. IDs kleiner als 100 sind für das System reserviert und sollten nicht an normale Benutzer vergeben werden.

Ablauf

- Als Erstes werden die Konfigurationsdateien `smartcard_netlogin`, `ldap.conf` und `ldap.secret` analysiert und die benötigten Parameter ausgelesen. Im Fehlerfalle oder bei Nichtvorhandensein eines obligatorischen Eintrages wird mit einer Fehlermeldung abgebrochen. Folgende Parameter werden ausgelesen:
  - Der Rechnername und (falls vorhanden) der Port des LDAP-Servers
  - Der DN für die Benutzer- und Gruppeneinträge
  - Der DN des LDAP-Administrators und dessen Passwort
  - Ein Flag, das angibt, ob die TLS Unterstützung aktiviert werden soll
- Anschliessend wird eine Verbindung zum LDAP-Server aufgebaut. Falls die Verbindung unverschlüsselt ist (ohne TLS Unterstützung), wird eine entsprechende Warnmeldung ausgegeben.
- Die Standardwerte für neue Benutzer werden aus dem LDAP-Verzeichnis ausgelesen. Sind diese nicht vorhanden (Benutzer *default$* existiert nicht), werden folgende Werte verwendet:
  - gidNumber = 1
  - homeDirectory = /home/*username*
  - loginShell = /bin/bash
  - shadowInactive = -1
  - shadowExpire = -1
  - shadowMin = 0
  - shadowMax = 99999
  - shadowWarning = 7
  - shadowFlag = 0

  Eine Beschreibung der Parameter ist in Tabelle 4.1 auf der vorherigen Seite gegeben.
- Die dem Programm übergebenen Argumente werden analysiert und bei Unstimmigkeiten eine Fehlermeldung ausgegeben, welche den Programmaufruf erläutert.
- Wurde dem Programm das Argument **-D** übergeben, werden die zuvor ausgelesenen Standardwerte anhand der weiteren Argumente modifiziert und zurück ins LDAP-Verzeichnis geschrieben. Falls der Benutzer *default$* noch nicht existiert, wird er erstellt. Sind keine weiteren Parameter vorhanden, werden die Standardwerte ausgegeben und der LDAP-Eintrag bleibt unverändert. Anschliessend wird das Programm terminiert.

- Andernfalls (Aufruf ohne **-D**), werden die Eigenschaften des neuen Benutzers anhand der übergebenen Argumente gesetzt und, wo nötig, durch Standardwerte ergänzt. Wurde die nummerische Benutzer ID nicht angegeben, wird ein um Eins grösserer Wert als die grösste vergebene Nummer, mindestens aber 100 verwendet.
- Die folgenden Punkte werden überprüft und im Fehlerfalle mit einer Fehlermeldung abgebrochen:
  - Der angegebene Benutzer darf noch nicht existieren
  - Die nummerische ID darf noch nicht vergeben sein, es sei denn, dies wird durch Angabe des Argumentes **-o** explizit erzwungen
  - Die Hauptgruppe des Benutzers muss vorhanden sein
- Waren alle Tests erfolgreich, wird der Benutzer bei seiner Hauptgruppe und weiteren allfällig angegebenen Gruppen hinzugefügt.
- Zuletzt wird der Benutzer im LDAP-Verzeichnis eingetragen und die Verbindung zum LDAP-Server abgebaut.

Argumente

Neuen Benutzer erstellen:

```
netuseradd [-c comment] [-d home_dir] [-e expire_date] [-f inactive_time]
           [-g initial_group] [-G group[,...]] [-s shell]
           [-u uid [ -o]] login
```

Argument	Beschreibung
-c comment	Beschreibung des Benutzers
-d home_dir	Home-Verzeichnis des Benutzers
-e expire_date	Datum, an dem das Benutzerkonto gesperrt wird
-f inactive_time	Anzahl Tage nach Ablauf des Passwortes bis das Benutzerkonto gesperrt wird
-g initial_group	Hauptgruppe des Benutzers
-G group[,...]	weitere Gruppen, denen der Benutzer zugefügt werden soll
-s shell	zu verwendende Login-Shell
-u uid	nummerische ID des Benutzers
-o	falls gesetzt, wird die angegebene nummerische ID auch verwendet, wenn diese schon existiert
login	Name des Benutzers, der erstellt werden soll

Tabelle 4.2: Argumente von netuseradd (1)

Neuen Standard setzen:

```
netuseradd -D [-g default_group] [-b default_home] [-f default_inactive]
              [-e default_expire] [-s default_shell]
```

Argument	Beschreibung
-g default_group	standard Hauptgruppe für neue Benutzer
-b default_home	standard Home-Verzeichnis für neue Benutzer
-f inactive_time	Anzahl Tage nach Ablauf des Passwortes bis das Benutzerkonto gesperrt wird
-e default_expire	Datum, an dem das Benutzerkonto gesperrt wird
-s default_shell	standard Login-Shell für neue Benutzer

Tabelle 4.3: Argumente von netuseradd (2)

4.4.2 netusermod

Dieses Programm ermöglicht das nachträgliche Modifizieren eines Netzwerkbenutzers. Benutzerinformationen von vorhandenen Dateien und Verzeichnisse bleiben davon unberührt und müssen unter Umständen mit dem **chown** Kommando angepasst werden.

Ablauf

- Als Erstes werden die Konfigurationsdateien `smartcard_netlogin`, `ldap.conf` und `ldap.secret` analysiert und die benötigten Parameter ausgelesen. Im Fehlerfalle oder bei Nichtvorhandensein eines obligatorischen Eintrages wird mit einer Fehlermeldung abgebrochen. Folgende Parameter werden ausgelesen:
  - Der Rechnername und (falls vorhanden) der Port des LDAP-Servers
  - Der DN für die Benutzer- und Gruppeneinträge
  - Der DN des LDAP-Administrators und dessen Passwort
  - Ein Flag, das angibt, ob die TLS Unterstützung aktiviert werden soll
- Anschliessend wird eine Verbindung zum LDAP-Server aufgebaut. Falls die Verbindung unverschlüsselt ist (ohne TLS Unterstützung), wird eine entsprechende Warnmeldung ausgegeben.
- Die dem Programm übergebenen Argumente werden analysiert und bei Unstimmigkeiten eine Fehlermeldung ausgegeben, welche den Programmaufruf erläutert.
- Nun werden die Daten des zu verändernden Benutzers aus dem LDAP-Verzeichnis ausgelesen und gemäss den übergebenen Argumenten modifiziert. Falls der Benutzer nicht existiert, wird mit einem Fehler abgebrochen.
- Bevor die Werte zurückgeschrieben werden können, müssen folgende Punkte noch überprüft werden:
  - Falls der Benutzer umbenannt wurde, darf dieser noch nicht existieren
  - Die nummerische ID darf noch nicht vergeben sein, es sei denn, dies wird durch Angabe des Argumentes **-o** explizit erzwungen
  - Die Hauptgruppe des Benutzers muss vorhanden sein

- Wurde die Gruppenzugehörigkeit des Benutzers modifiziert, muss er aus den alten Einträgen entfernt und zu den neuen im LDAP-Verzeichnis hinzugefügt werden.
- Falls der Benutzer umbenannt wurde, müssen alle Gruppeneinträge entsprechend angepasst werden.
- Die Modifikationen werden zurückgeschrieben. Wurde der Benutzer umbenannt, wird der alte Eintrag gelöscht und ein neuer erstellt.
- Zuletzt wird die Verbindung zum LDAP-Server abgebaut.

Argumente

```
netusermod [-c comment] [-d home_dir] [-e expire_date] [-f inactive_time]
           [-g initial_group] [-G group[,...]] [-l login_name] [-s shell]
           [-u uid [ -o]] login
```

Argument	Beschreibung
-c comment	Beschreibung des Benutzers
-d home_dir	Home-Verzeichnis des Benutzers
-e expire_date	Datum, an dem das Benutzerkonto gesperrt wird
-f inactive_time	Anzahl Tage nach Ablauf des Passwortes bis das Benutzerkonto gesperrt wird
-g initial_group	Hauptgruppe des Benutzers
-G group[,...]	weitere Gruppen, denen der Benutzer zugefügt werden soll
-l login_name	neuer Name des Benutzers
-s shell	zu verwendende Login-Shell
-u uid	nummerische ID des Benutzers
-o	falls gesetzt, wird die angegebene nummerische ID wird auch verwendet, wenn diese schon existiert
login	Name des Benutzers, der modifiziert werden soll

Tabelle 4.4: Argumente von netusermod

4.4.3 netuserdel

Dieses Tool löscht einen angegebenen Benutzer. Das Home-Verzeichnis und andere vom Benutzer erstellte Dateien und Verzeichnisse bleiben erhalten und müssen explizit gelöscht werden.

Ablauf

- Als Erstes werden die Konfigurationsdateien `smartcard_netlogin`, `ldap.conf` und `ldap.secret` analysiert und die benötigten Parameter ausgelesen. Im Fehlerfalle oder bei Nichtvorhandensein eines obligatorischen Eintrages wird mit einer Fehlermeldung abgebrochen. Folgende Parameter werden ausgelesen:

  - Der Rechnername und (falls vorhanden) der Port des LDAP-Servers
  - Der DN für die Benutzer- und Gruppeneinträge
  - Der DN des LDAP-Administrators und dessen Passwort
  - Ein Flag, das angibt, ob die TLS Unterstützung aktiviert werden soll

- Anschliessend wird eine Verbindung zum LDAP-Server aufgebaut. Falls die Verbindung unverschlüsselt ist (ohne TLS Unterstützung), wird eine entsprechende Warnmeldung ausgegeben.
- Die dem Programm übergebenen Argumente werden analysiert und bei Unstimmigkeiten eine Fehlermeldung ausgegeben, welche den Programmaufruf erläutert.
- Der angegebene Benutzer wird im LDAP-Verzeichnis gesucht und falls vorhanden gelöscht. Ansonsten wird mit einer Fehlermeldung abgebrochen.
- Zuletzt wird der Benutzer noch aus allen Gruppen gelöscht und die Verbindung zum LDAP-Server abgebaut.

Argumente

`netuserdel user`

Argument	Beschreibung
user	Name des Benutzers, der gelöscht werden soll

Tabelle 4.5: Argumente von netuserdel

4.4.4 netuserls

Das Programm gibt Informationen zu einem oder mehreren Benutzern aus. Die Ausgabe entspricht dabei dem Aufbau einer `passwd` Datei. Dadurch wird es Tools und Scripts ohne LDAP Unterstützung ermöglicht, auf die Benutzerdaten zuzugreifen.

Ablauf

- Als Erstes werden die Konfigurationsdateien `smartcard_netlogin`, `ldap.conf` und `ldap.secret` analysiert und die benötigten Parameter ausgelesen. Im Fehlerfalle oder bei Nichtvorhandensein eines obligatorischen Eintrages wird mit einer Fehlermeldung abgebrochen. Folgende Parameter werden ausgelesen:
  - Der Rechnername und (falls vorhanden) der Port des LDAP-Servers
  - Der DN für die Benutzer- und Gruppeneinträge
  - Der DN des LDAP-Administrators und dessen Passwort
  - Ein Flag, das angibt, ob die TLS Unterstützung aktiviert werden soll

- Anschliessend wird eine Verbindung zum LDAP-Server aufgebaut. Falls die Verbindung unverschlüsselt ist (ohne TLS Unterstützung), wird eine entsprechende Warnmeldung ausgegeben.
- Die dem Programm übergebenen Argumente werden analysiert und bei Unstimmigkeiten eine Fehlermeldung ausgegeben, welche den Programmaufruf erläutert.
- Falls das erste Argument eine Zahl ist, werden alle Benutzereinträge ausgegeben, deren nummerische ID gleich dieser Zahl ist. Ansonsten wird das Argument als Benutzernamen gewertet und der entsprechende Eintrag ausgegeben. Wurde kein Argument übergeben, werden alle Einträge ausgegeben.
- Zuletzt wird die Verbindung zum LDAP-Server abgebaut.

Argumente

```
netuserls [user|uid]
```

Argument	Beschreibung
user	Name des Benutzers, der angezeigt werden soll
uid	nummerische ID des Benutzers, der angezeigt werden soll

Tabelle 4.6: Argumente von netuserls

4.4.5 netaccountadd

Dieses Programm wird für das Erstellen eines real existierenden Benutzers verwendet. Dazu wird ein neues Zertifikat anhand der Programm-Parameter und der Konfigurationsdateien erstellt. Dieses Zertifikat und die zugehörigen Schlüssel werden auf der Smartcard gespeichert. Zum Abschluss wird das Zertifikat auf dem LDAP-Server dem entsprechenden Benutzer hinzugefügt.

Programmablauf

Der genaue Programm-Ablauf ist wie folgt:

Einlesen der Argumente Zuerst werden die beim Programmaufruf übergebenen Argumente analysiert und ausgewertet. Die Tabelle 4.7 auf Seite 64 zeigt die möglichen Argumente und deren Bedeutung.

Einlesen der Konfigurationsdateien Hier werden die Konfigurationsdateien `/etc/-smartcard_netlogin`, `ldap.conf` und `ldap.secret` ausgelesen und die nötigen Konfigurationen ausgelesen. Insbesondere müssen

- ca_dir,

- ca_rdn,
- ldap_conf und
- ldap_password

in der Datei `/etc/smartcard_netlogin`,

- host,
- rootbinddn und
- nss_base_passwd

in der Datei `ldap.conf` und das Passwort in der Datei `ldap.secret` vorhanden sein. Ist dies nicht der Fall, wird mit einer entsprechenden Fehlermeldung abgebrochen. Falls eine verschlüsselte Verbindung erwünscht ist, müssen zusätzlich noch der Parameter **sslpath** vorhanden sein und der Parameter **ssl** auf **start_tls** gesetzt sein. Auch hier wird das Programm beendet, falls **start_tls** gesetzt ist, aber kein SSL-Pfad angegeben wurde.

Testen der Smartcard In diesem Abschnitt wird geprüft, ob auf die Smartcard zugegriffen werden kann. Es wird ebenfalls getestet, ob der angegebene Aut0-PIN (Smartcard-Administrator-PIN) gültig ist. Dies wird getan, damit nicht ein neues Zertifikat unnötig erstellt wird, welches nie zum Einsatz kommt, da nicht auf die Smartcard zugegriffen werden kann.

Prüfen, ob Benutzer im LDAP schon existiert In dieser Phase wird eine Verbindung zum LDAP-Server aufgebaut. Wie immer beim Verbindungsaufbau wird, sobald **start_tls** gesetzt ist, eine verschlüsselte Verbindung aufgebaut. Andernfalls wird unverschlüsselt verbunden und eine Warnung ausgegeben. Nach dem Verbindungsaufbau wird geprüft, ob es einen Eintrag gibt, der dem übergebenen Benutzernamen entspricht. Falls nein, wird das Programm beendet und die Meldung ausgegeben, dass man zuerst mit **netuseradd** den Benutzer erstellen muss.

Sollte der Benutzer jedoch existieren, wird geprüft, ob dieser bereits ein Zertifikat auf dem LDAP-Server gespeichert hat. Auch hier wird das Programm beendet, falls dies der Fall ist. Um ein solches Zertifikat zu löschen, ist **netaccountdel** zuständig, das in Kapitel 4.4.6 auf der nächsten Seite genauer beschrieben wird.

Generierung des *Subject Alternative Name* Jetzt wird anhand der Benutzer ID und der eventuell angegebenen zusätzlichen Profile der *Subject Alternative Name* gebildet. In dieser Extension des Zertifikats speichern wir die verschiedenen Benutzerprofile, die benutzt werden können. Diese müssen in der Form *email:benutzer@localhost, email:root@localhost* angegeben werden.

Erstellen der Benutzerkonfigurationsdatei Hier wird die temporäre Konfigurationsdatei `user.cnf` erstellt. Diese Datei besagt, wie OpenSSL das Zertifikat erstellen muss. Hier werden vor allem Dinge wie der Ort, an dem die CRL veröffentlicht ist, der DN des Benutzers und der *Subject Alternative Name* angegeben.

Zertifikat erstellen Jetzt wird mit einer Reihe von OpenSSL Befehlen das Benutzerzertifikat erstellt. Dabei muss die Passphrase des Schlüssels der CA eingegeben werden. Gleich nach dem Erstellen werden der Request und die Konfigurationsdatei wieder gelöscht.

Parsen des Schlüsselpaares Hier werden mit einem ASN1-Parser der öffentliche und der private Schlüssel, die zuvor erstellt wurden, ausgelesen. Dazu wird ein Perl-ASN1-Parsermodul verwendet, bei dem man nur noch die Struktur der Schlüsseldatei angeben muss. Sofort nach dem Einlesen wird die Schlüsseldatei wieder von der Harddisk gelöscht.

Speichern der Schlüssel und des Zertifikats auf der Smartcard Jetzt können die Schlüssel und das erstellte Zertifikat auf der Smartcard abgelegt werden. Gleichzeitig werden die Zugriffsbedingungen auf die einzelnen Dateien der Smartcard so gesetzt, dass nur noch Befugte diese auslesen und schreiben können. Dabei wird auch das CHV1-PIN-File (Benutzer-PIN) so geschützt, dass es niemand mehr auslesen kann. Dann werden der Reihe nach der öffentliche Schlüssel, der private Schlüssel und das Zertifikat auf die Karte geschrieben. Dabei werden die einzelnen Dateien zuerst gelöscht, falls sie schon vorhanden sind. Zum Abschluss wird das Zertifikat wieder von der Harddisk gelöscht, bleibt aber im Speicher des Computers noch erhalten.

Schreiben des Zertifikats auf den LDAP-Server Zum Schluss des Programms wird noch das Zertifikat auf den LDAP-Server geschrieben. Da wir am Anfang schon geprüft haben, ob mit dem Benutzer alles in Ordnung ist, können wir jetzt direkt auf das Objekt zugreifen und die zusätzlichen Daten speichern.

Argumente

```
Usage: netaccountadd -uid BenutzerID [-Option]
```

Argument	Beschreibung
-uid BenutzerID	Gibt die Benutzer-Identifikation an. Dieses Argument muss gesetzt sein.
-p Profile	Hier kann man Profile angeben, die der Benutzer zusätzlich anwenden darf. Dabei sind die einzelnen Profile durch Kommas zu trennen: *root,dialout*. Dieses Argument ist optional.
-aut0 aut0-PIN	Mit diesem Argument wird der Administrator-PIN der Smartcard angegeben. Falls dieses Argument nicht angegeben wird, wird automatisch der Standard-Shipping-Code der Schlumberger Cyberflex Access Smartcard verwendet (0xad9f61fefa20ce63).
-r Readernummer	Hier kann angegeben werden, welcher Smartcard Kartenleser verwendet werden soll. Dabei bezeichnet 0 den ersten Leser. Falls dieses Argument nicht gesetzt wird, wird automatisch 0 angenommen.

Tabelle 4.7: Argumente von netaccountadd

4.4.6 netaccountdel

Um einen Benutzer zu sperren, haben wir das Programm `netaccountdel` erstellt. Es sperrt das Zertifikat in der CA und erzeugt eine neue CRL, die es auch gleich veröffentlicht.

Programmablauf

Die Abbildung 4.4 zeigt den Programmablauf für `netaccounddel`. Im Folgenden werden nur noch die Punkte erläutert, die nicht schon im Abschnitt 4.4.5 auf Seite 62 aufgezeigt sind.

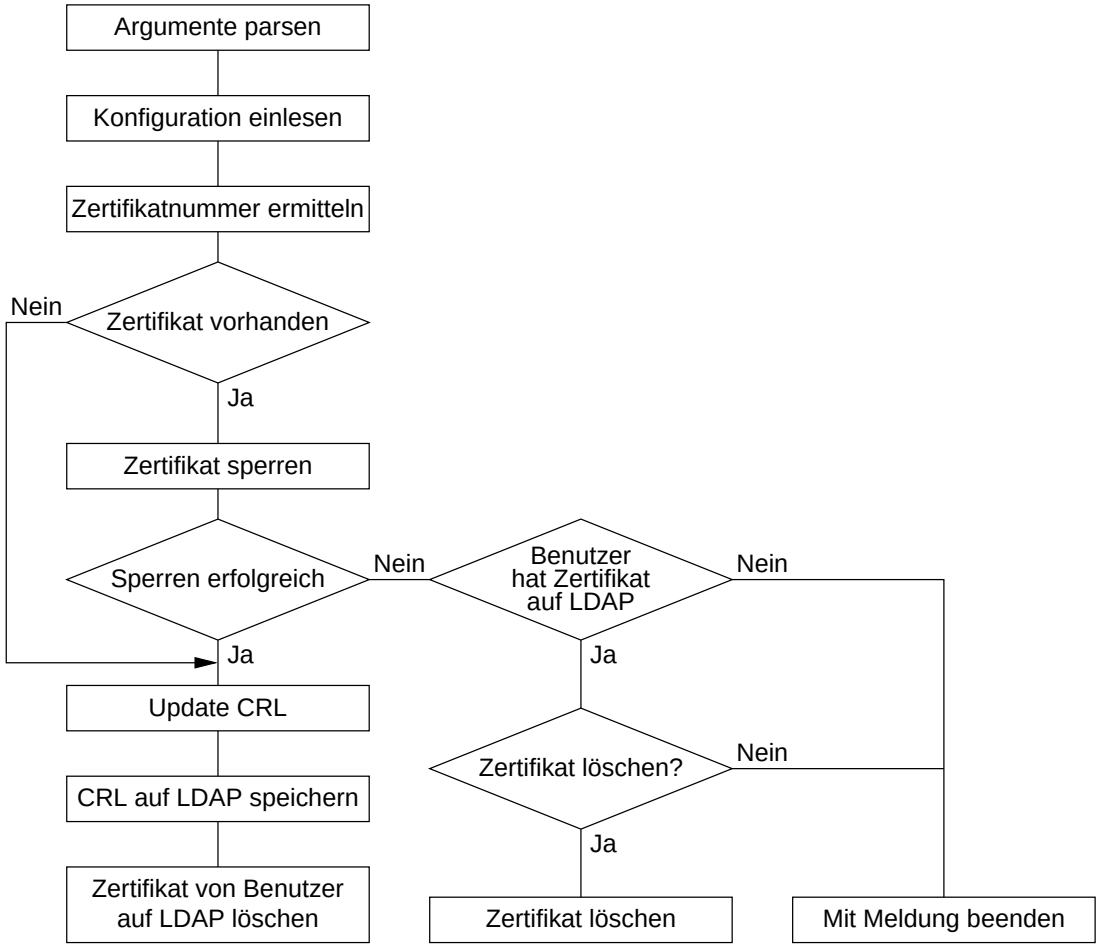


Abbildung 4.4: Programmablauf netaccountdel

Zertifikatnummer ermitteln Anhand der Datei `index.txt` der CA wird die zum angegebenen Benutzer gehörige Zertifikatsnummer ermittelt. Wir verwenden nur den letzten Eintrag der Liste, da eventuell vorher vorhandene Einträge schon gesperrt sein müssen, da jedes Zertifikat einen eindeutigen Bezeichner hat.

Zertifikat in der CA sperren Hier wird das Zertifikat des Benutzers gesperrt. Dazu muss die Passphrase des privaten Schlüssels der CA eingegeben werden.

Update CRL Es wird eine neue CRL mit OpenSSL erstellt. Auch hierzu muss die Passphrase des privaten Schlüssels der CA eingegeben werden.

CRL auf LDAP speichern Beim Speichern der CRL im Verzeichnisbaum des LDAP-Servers wird zuerst geprüft, ob der Eintrag für die Root-CA überhaupt existiert. Falls nicht, wird mit einer Fehlermeldung abgebrochen. Falls die Root-CA auf dem LDAP-Server vorhanden ist, wird die vorhandene CRL durch die neue ersetzt.

Zertifikat von Benutzer auf LDAP löschen Als letzten Schritt wird das Zertifikat des Benutzers auf dem LDAP-Server gelöscht. Dazu wird geprüft, ob er überhaupt existiert. Falls nicht, wird wiederum abgebrochen und eine Fehlermeldung ausgegeben. Falls er existiert, wird das Zertifikat gelöscht.

Sperren war erfolglos Falls das Sperren eines Zertifikats fehlgeschlagen ist, wird zuerst geprüft, ob der Benutzer noch ein Zertifikat auf dem LDAP-Server gespeichert hat. Falls ja, wird nachgefragt, ob dieses gelöscht werden soll. Vorsicht: Das Sperren könnte auch durch die Eingabe einer falschen Passphrase misslungen sein. Falls der Benutzer kein Zertifikat auf dem LDAP-Server gespeichert hat, wird die Meldung: „no certificate stored in CA and ldap for ...“ ausgegeben.

Argumente

```
Usage: netaccountdel -uid BenutzerID
```

Argument	Beschreibung
-uid BenutzerID	Gibt den Benutzer an, der gelöscht werden soll. Dieses Argument muss gesetzt sein.

Tabelle 4.8: Argumente von netaccountdel

4.5 Gruppen-Administration

4.5.1 netgroupadd

Das Programm dient dem Hinzufügen einer Netzwerkgruppe.

Ablauf

- Als Erstes werden die Konfigurationsdateien `smartcard_netlogin`, `ldap.conf` und `ldap.secret` analysiert und die benötigten Parameter ausgelesen. Im Fehlerfalle oder bei Nichtvorhandensein eines obligatorischen Eintrages wird mit einer Fehlermeldung abgebrochen. Folgende Parameter werden ausgelesen:
  - Der Rechnername und (falls vorhanden) der Port des LDAP-Servers
  - Der DN für die Benutzer- und Gruppeneinträge
  - Der DN des LDAP-Administrators und dessen Passwort
  - Ein Flag, das angibt, ob die TLS Unterstützung aktiviert werden soll
- Anschliessend wird eine Verbindung zum LDAP-Server aufgebaut. Falls die Verbindung unverschlüsselt ist (ohne TLS Unterstützung), wird eine entsprechende Warnmeldung ausgegeben.

- Die dem Programm übergebenen Argumente werden analysiert und bei Unstimmigkeiten eine Fehlermeldung ausgegeben, welche den Programmaufruf erläutert.
- Die Parameter der Gruppe werden gemäss den übergebenen Argumenten erstellt. Wurde die nummerische Gruppen ID nicht angegeben, wird ein um Eins grösserer Wert als die grösste vergebene Nummer, mindestens aber 100 verwendet.
- Die folgenden Punkte werden überprüft und im Fehlerfalle mit einer Fehlermeldung abgebrochen:
  - Die angegebene Gruppe darf noch nicht existieren
  - Die nummerische ID darf noch nicht vergeben sein, es sei denn, dies wird durch Angabe des Argumentes **-o** explizit erzwungen
- Waren alle Tests erfolgreich, wird die Gruppe ins LDAP-Verzeichnis eingefügt.
- Zuletzt wird die Verbindung zum LDAP-Server abgebaut.

Argumente

```
netgroupadd [-g gid [-o]] group
```

Argument	Beschreibung
-g gid	nummerische ID der Gruppe
-o	falls gesetzt, wird die angegebene nummerische ID auch verwendet, wenn diese schon existiert
group	Namen der Gruppe, die erstellt werden soll

Tabelle 4.9: Argumente von netgroupadd

4.5.2 netgroupmod

Dieses Programm ermöglicht das nachträgliche Modifizieren einer Netzwerkgruppe. Gruppeninformationen von vorhandenen Dateien und Verzeichnisse bleiben davon unberührt und müssen unter Umständen mit dem **chgrp** Kommando angepasst werden.

- Als Erstes werden die Konfigurationsdateien `smartcard_netlogin`, `ldap.conf` und `ldap.secret` analysiert und die benötigten Parameter ausgelesen. Im Fehlerfalle oder bei Nichtvorhandensein eines obligatorischen Eintrages wird mit einer Fehlermeldung abgebrochen. Folgende Parameter werden ausgelesen:
  - Der Rechnername und (falls vorhanden) der Port des LDAP-Servers
  - Der DN für die Benutzer- und Gruppeneinträge
  - Der DN des LDAP-Administrators und dessen Passwort
  - Ein Flag, das angibt, ob die TLS Unterstützung aktiviert werden soll

- Anschliessend wird eine Verbindung zum LDAP-Server aufgebaut. Falls die Verbindung unverschlüsselt ist (ohne TLS Unterstützung), wird eine entsprechende Warnmeldung ausgegeben.
- Die dem Programm übergebenen Argumente werden analysiert und bei Unstimmigkeiten eine Fehlermeldung ausgegeben, welche den Programmaufruf erläutert.
- Nun werden die Daten der zu verändernden Gruppe aus dem LDAP-Verzeichnis ausgelesen und gemäss den übergebenen Argumenten modifiziert. Falls die Gruppe nicht existiert, wird mit einem Fehler abgebrochen.
- Bevor die Werte zurückgeschrieben werden können, müssen folgende Punkte noch überprüft werden:
  - Falls die Gruppe umbenannt wurde, darf diese noch nicht existieren
  - Die nummerische ID darf noch nicht vergeben sein, es sei denn, dies wird durch Angabe des Argumentes **-o** explizit erzwungen.
- Die Modifikationen werden zurückgeschrieben. Wurde die Gruppe umbenannt, wird der alte Eintrag gelöscht und ein neuer erstellt.
- Zuletzt wird die Verbindung zum LDAP-Server abgebaut.

Argumente

```
netgroupmod [-g gid [-o]] [-n group_name] group
```

Argument	Beschreibung
-g gid	nummerische ID der Gruppe
-o	falls gesetzt, wird die angegebene nummerische ID auch verwendet, wenn diese schon existiert
-n group_name	der neue Name der Gruppe
group	Namen der Gruppe, die modifiziert werden soll

Tabelle 4.10: Argumente von netgroupmod

4.5.3 netgroupdel

Dieses Tool löscht eine angegebene Gruppe. Gruppeninformationen von vorhandenen Dateien und Verzeichnisse bleiben davon unberührt und müssen unter Umständen mit dem **chgrp** Kommando angepasst oder mit **rm** gelöscht werden.

- Als Erstes werden die Konfigurationsdateien `smartcard_netlogin`, `ldap.conf` und `ldap.secret` analysiert und die benötigten Parameter ausgelesen. Im Fehlerfalle oder bei Nichtvorhandensein eines obligatorischen Eintrages wird mit einer Fehlermeldung abgebrochen. Folgende Parameter werden ausgelesen:

  - Der Rechnername und (falls vorhanden) der Port des LDAP-Servers
  - Der DN für die Benutzer- und Gruppeneinträge
  - Der DN des LDAP-Administrators und dessen Passwort
  - Ein Flag, das angibt, ob die TLS Unterstützung aktiviert werden soll

- Anschliessend wird eine Verbindung zum LDAP-Server aufgebaut. Falls die Verbindung unverschlüsselt ist (ohne TLS Unterstützung), wird eine entsprechende Warnmeldung ausgegeben.
- Die dem Programm übergebenen Argumente werden analysiert und bei Unstimmigkeiten eine Fehlermeldung ausgegeben, welche den Programmaufruf erläutert.
- Es wird überprüft, ob die Gruppe bei einem Benutzer als Hauptgruppe eingetragen ist. Falls dem so ist, darf die Gruppe nicht gelöscht werden und das Programm bricht mit einer Fehlermeldung ab.
- Der angegebene Benutzer wird im LDAP-Verzeichnis gesucht und falls vorhanden gelöscht. Ansonsten wird mit einer Fehlermeldung abgebrochen.
- Zuletzt wird der Benutzer noch aus allen Gruppen gelöscht und die Verbindung zum LDAP-Server abgebaut.

Argumente

```
netgroupdel group
```

Argument	Beschreibung
group	Namen der Gruppe, die gelöscht werden soll

Tabelle 4.11: Argumente von netgroupdel

4.5.4 netgroupls

Das Programm gibt Informationen zu einer oder mehreren Gruppen aus. Die Ausgabe entspricht dabei dem Aufbau einer `group` Datei. Dadurch wird es Tools und Scripts ohne LDAP Unterstützung ermöglicht, auf die Gruppendaten zuzugreifen.

- Als Erstes werden die Konfigurationsdateien `smartcard_netlogin`, `ldap.conf` und `ldap.secret` analysiert und die benötigten Parameter ausgelesen. Im Fehlerfalle oder bei Nichtvorhandensein eines obligatorischen Eintrages wird mit einer Fehlermeldung abgebrochen. Folgende Parameter werden ausgelesen:
  - Der Rechnername und (falls vorhanden) der Port des LDAP-Servers
  - Der DN für die Benutzer- und Gruppeneinträge
  - Der DN des LDAP-Administrators und dessen Passwort

  - Ein Flag, das angibt, ob die TLS Unterstützung aktiviert werden soll

- Anschliessend wird eine Verbindung zum LDAP-Server aufgebaut. Falls die Verbindung unverschlüsselt ist (ohne TLS Unterstützung), wird eine entsprechende Warnmeldung ausgegeben.

- Die dem Programm übergebenen Argumente werden analysiert und bei Unstimmigkeiten eine Fehlermeldung ausgegeben, welche den Programmaufruf erläutert.

- Falls das erste Argument eine Zahl ist, werden alle Gruppeneinträge ausgegeben, deren nummerische ID gleich dieser Zahl ist. Ansonsten wird das Argument als Gruppennamen gewertet und der entsprechende Eintrag ausgegeben. Wurde kein Argument übergeben, werden alle Einträge ausgegeben.

- Zuletzt wird die Verbindung zum LDAP-Server abgebaut.

Argumente

```
netgroupls [group|gid]
```

Argument	Beschreibung
group	Namen der Gruppe, die angezeigt werden soll
gid	nummerische ID der Gruppe, die angezeigt werden soll

Tabelle 4.12: Argumente von netgroupls

4.6 OpenSSH Patch

Unser Ziel war es, das OpenSSH Paket so zu modifizieren, dass eine Benutzerauthentisierung zusätzlich mit den von uns verwendeten Smartcards möglich ist.

Als ersten Schritt mussten wir dafür sorgen, dass X.509 Zertifikate anstelle von öffentlichen Schlüsseln verwendet werden. [sshtrans] sieht dafür den Schlüssel-Typ *x509v3-sign-rsa* vor. Dieser wird in der **SSH_MSG_USERAUTH_REQUEST** Nachricht (siehe Abbildung 3.4 auf Seite 45) an den Daemon gesendet. Der Daemon muss nun nicht, wie bei den anderen Schlüssel-Typen, anhand von Tabellen feststellen, ob der öffentliche Schlüssel vertrauenswürdig ist, sondern kann das erhaltene Zertifikat mit den schon vorhandenen Root CA-Zertifikaten und der CRL überprüfen. Dadurch wird eine nahtlose Integration in unser Login-Projekt erreicht.

Mehrere Dateien mussten angepasst oder erstellt werden. Wir versuchten, möglichst alle Änderungen an der zentralen Bibliothek anzubringen. Die einzelnen Applikationen sollten so wenig wie möglich verändert werden.

4.6.1 Erweiterung der Bibliothek sshlib

Für den Umgang mit den Zertifikaten und den Smartcards werden einige grundlegende Funktionen aus unserem bestehenden Projekt benötigt. Folgende Module (.c und .h Dateien) wurden deshalb zur Bibliothek hinzugefügt:

smartcard Enthält die wichtigsten Routinen für die Benutzung von Smartcards.

x509 Diese Dateien werden für das Parsen und Verifizieren der Zertifikate und CRLs verwendet.

md5 und sha1 Wurden von uns aus dem Linux IPSec Projekt übernommen und werden von dem x509 Modul zur Überprüfung der Signaturen verwendet.

ssh_x509 Beinhaltet alle von uns zusätzlich benötigten Funktionen, die in Tabelle 4.13 erläutert werden.

Funktion	Beschreibung
SMART_sign()	Signiert die übergebenen Daten mit Hilfe einer RSA Smartcard.
extract_cert_key()	Extrahiert den öffentlichen Schlüssel aus einem X.509 Zertifikat.
verify_certificate()	Überprüft ein X.509 Zertifikat (Ablaufdatum, Signatur, CRL).
load_cert()	Lädt ein X.509 Zertifikat von einer Smartcard.
parse_x509_conf()	Analysiert die Konfigurationsdateien `smartcard_netlogin` und `ldap.conf`.

Tabelle 4.13: Funktionen des Moduls ssh_x509

Zusätzlich mussten folgende Dateien angepasst werden:

key.c und key.h Für die von uns verwendeten X.509 Zertifikate wurde ein zusätzlicher Schlüssel-Typ erstellt. Ausserdem wurden alle Funktionen für den Umgang mit diesem erweitert. Als Name für diesen neuen Schlüssel-Typ verwendeten wir, wie in [sshtrans] vorgeschlagen *x509v3-sign-rsa*.

readconf.c und readconf.h Diese Dateien sind für das Auslesen der SSH-Konfiguration zuständig und wurden um den Eintrag **useX509cert** erweitert. Dieser ermöglicht das Ein- und Ausschalten unserer Smartcard-Unterstützung.

ssh-rsa.c Die Funktion **ssh_rsa_sign()** wurde so angepasst, dass für das Signieren des Hash-Wertes die Smartcard verwendet wird.

4.6.2 Anpassung des SSH-Clients

Beim Client musste nur die Datei `ssh.c` geändert werden. Wir haben **-z Nummer** als zusätzliches Argument hinzugefügt. Damit kann unsere Smartcard-Unterstützung aktiviert werden. **Nummer** bezeichnet dabei den zu verwendenden Smartcard-Reader. Zusätzlich wird das auf der Smartcard gespeicherte Zertifikat ausgelesen und zur Schlüsselliste hinzugefügt.

4.6.3 Anpassung des SSH-Daemons

Die Anpassung des Daemons erforderte eine Modifikation der folgenden Dateien:

sshd.c Beim Starten des Dienstes muss unsere Smartcard-Login Konfiguration ausgelesen werden, damit der Speicherort der Root CA-Zertifikate bekannt ist.

auth2.c In dieser Datei musste die Funktion **userauth_pubkey()** für unseren neuen Schlüssel-Typ angepasst werden. Wir mussten sicherstellen, dass das empfangene Zertifikat nicht wie sonst üblich anhand einer Tabelle überprüft wird. Es wird mit Hilfe der Root CA-Zertifikate verifiziert und mit der CRL auf seine Gültigkeit getestet.

Kapitel 5

Installation und Anwendung

Dieses Kapitel beschreibt den Installationsvorgang für unser Projekt. Der Text entstammt dem HOWTO, welches mit dem Paket mitgeliefert wird. Um möglichst viele potenzielle Benutzer anzusprechen, haben wir es wie üblich in Englisch verfasst.

5.1 Overview

The following sections are a step by step description of how to make your Linux host using a RSA-smartcard for user authentication. There are three functional main parts:

- An LDAP-Server which - together with the NSS-modules on each client - works as a network wide user and group directory.
- A Certification Authority (CA) for creating the necessary user certificates and storing them on the user's smartcard.
- The Smartcard PAM-Module which is responsible for the effective login procedure.

The LDAP-Server and the CA can either be installed on one common server or on two independent servers. After a successful installation it is also possible to use a smartcard based user authentication on the servers themself. Be aware of the fact that for a successful login the Certificate Revocation List (CRL), which is mostly stored on the LDAP-Server, must be accessable by the PAM-Module during the login procedure. Therefore in such a case the LDAP-Server has to be running before you can login!

For all further steps you are supposed to be logged in as root.

5.2 Required Software

It follows a list of software packages and libraries which are needed by the Smartcard-Netlogin package. Most of them have been shipped with your distribution and are most probably already installed. The package was tested with the given versions, but should work with newer ones, too.

- OpenSSL 0.9.6a (http://www.openssl.org)
- OpenLDAP 2.0.11 (http://www.openldap.org)
- NSS LDAP Module 2.167 (http://www.padl.com)
- GNU MP Library 3.1.1 (http://www.swox.com/gmp/)
- Linux PAM-System 0.75 (http://www.kernel.org/pub/linux/libs/pam/)
- Flex 2.5.4 (http://www.gnu.org/software/flex/flex.html)
- Perl Net::IO-Socket-SSL Module 0.80 (http://cpan.org)
- Perl Net::LDAP Module 0.24 (http://cpan.org)
- Perl Net::LDAP-SSL Module 0.22 (http://cpan.org)
- Perl Net::ASN1 0.14 (http://cpan.org)

In case you have to download one or more of the packages, the following command sequence should install them properly:

```
bash# tar -xvzf package-x.y.z.tar.gz
bash# cd package.x.y.z
bash# ./configure
bash# make
bash# make install
```

The mentioned perl modules can be installed by doing something like:

```
bash# tar -xvzf package-x.y.z.tar.gz
bash# cd package.x.y.z
bash# perl Makefile.PL
bash# make
bash# make test
bash# make install
```

It could happen that a module asks for another one which isn't installed yet. To make things working you have to install that one first. In case of problems with a certain package, please read the dedicated documentation, especially the `README` and `INSTALL` files.

The installation of the two pc/sc related packages and the OpenSSH-patch is described in detail below.

- PC/SC Lite 0.9.3 (http://www.linuxnet.com/middle.html)
- PC/SC Reader Drivers (http://www.linuxnet.com/drivers.html)
- Perl PC/SC Module 1.0.1 (http://www.linuxnet.com/middle.html)
- OpenSSH 2.9.9p2 (http://www.openssh.org)

5.3 Installation of the Smartcard-Reader

Since a running smartcard reader is a basic requirement for a smartcard based login, you have to install it on each workstation and server as described below. The perl pcsc-module is only needed by some scripts on the CA-Server and is optional for all others.

5.3.1 Installing PC/SC

Unpack the archive, configure, compile and install it.

```
bash# tar -xvzf pcsc-lite-X.Y.Z.tar.gz
bash# cd pcsc-lite-X.Y.Z
bash# ./configure --enable-daemon --enable-usb
bash# make
bash# make install
```

If you don't have an usb reader, the parameter –**enable-usb** is optional.

5.3.2 Installing the Reader drivers

Unpack the archive and compile it. If the driver was created with autoconf call the **configure** script first.

```
bash# tar -xvzf the-driver-X.Y.Z.tar.gz
bash# cd the-driver-X.Y.Z
bash# ./configure
bash# make
bash# make install
```

if **make install** isn't supported copy the driver module (`libXXX.so`) to `/usr/local/lib` by yourself.

```
bash# cp libXXX.so /usr/local/lib
```

It is recommended to read the `README` file, as some drivers need special environment variables or symbolic links.

For each installed driver there has to be one configuration item as shown below in the `/etc/reader.conf` file.

```
FRIENDLYNAME   'reader name'
DEVICENAME     'device name'
LIBPATH        'path to the driver'
CHANNELID      'device id'
```

The exact parameter values depend on the used reader/driver and ought to be mentioned in the dedicated `README` file. The **LIBPATH** parameter defines the full path to the actually driver module and should be set to something like `/usr/local/lib/libXXX.so`. See 5.9.1 on page 89 for a possible configuration. After all modifications have been made, try to start the daemon.

```
bash# /usr/local/sbin/pcscd
```

If the daemon fails while loading a reader driver, check if its configuration item in `/etc/reader.conf` is set correctly and the reader is properly connected to the defined port. There is also a mailing list (http://www.linuxnet.com/list.html) which probably can help you. Please search first in the archive (http://www.mail-archive.com/sclinux-linuxnet.com/) to avoid asking questions which were already answered!

5.3.3 Make the PC/SC Daemon starting on startup

If you want to use a smartcard for login, it's important that the pc/sc and reader driver work immediately after startup. Therefore you have to install an init-script which starts the daemon in the necessary runlevels. First create a symbolic link in `/usr/sbin/` to the daemon. Then write the init-script for the daemon in `/etc/init.d/` (an example is shown in 5.9.2 on page 90) and install it with the **insserv** command.

```
bash# ln -s /usr/local/sbin/pcscd /usr/sbin/pcscd
bash# vi /etc/init.d/pcsc
bash# chmod a+x /etc/init.d/pcsc
bash# insserv /etc/init.d/pcsc
```

Now you can try to start the daemon by using the init-script and check if it is running.

```
bash# killall pcscd
bash# /etc/init.d/pcsc start
bash# /etc/init.d/pcsc status
```

Finally reboot your system, login as root and test again if the daemon is running.

```
bash# reboot
...
bash# /etc/init.d/pcsc status
```

5.3.4 Installation of the PC/SC Perl Module

Unpack the archive, configure, compile and install it. If **make test** reports an error make sure that there is a smartcard inserted into the reader.

```
bash# tar -xvzf pcsc-perl_X.Y.Z.tar.gz
bash# cd pcsc-perl-X.Y.Z
bash# perl Makefile.PL
bash# make
bash# make test
bash# make install
```

5.4 Installation of the Smartcard Netlogin Package

5.4.1 Installation

Unpack the archive, configure, compile and install it.

```
bash# tar -xvzf smartcard_netlogin-X.Y.tar.gz
bash# cd smartcard_netlogin-X.Y
bash# ./configure
bash# make
bash# make install
```

If you didn't install the pcsc middleware in the default directory (`/usr/local/`) you have to specify the path to it with the –**with-pcsclite-dir** parameter.

5.4.2 Configuration

For a properly running system it is necessary to modify or create several configuration files. The main configuration file `/etc/smartcard_netlogin` is responsible for some global settings of our project. The files `ldap.conf`, `slapd.conf` and `ldap.secret` are used to configure the ldap-client and server as well as the user and group administration tools. The files `/etc/smartcard_netlogin`, `ldap.conf` and `ldap.secret` have to be configured on each host. The `slapd.conf` configuration file is only needed by the LDAP-Server.

smartcard_netlogin

As mentioned above, this configuration file contains the central configuration of the Smartcard Netlogin project. There exist four different options, which are as follows:

ca_dir sets the directory, where the Certificate Authority (CA) is stored in.

ca_rdn sets the relative distinguished name (rdn) of the CA on the LDAP-Server. It is relative to base of the `ldap.conf` configuration file.

ldap_conf sets the exact path to the `ldap.conf` configuration file. It is more secure to use the main `ldap.conf` instead of the `.ldaprc` of each user. With this, the user can't setup another LDAP-Server and fake a running system to login by himself.

ldap_password sets the exact path to the `ldap.secret` configuration file.

The **ca_dir** and **ca_rdn** items are only needed by the CA-Server the others are always mandatory.

Here is an example `/etc/smartcard_netlogin` configuration file:

```
# CA-Directory
ca_dir        = /Root_CA/

# Relative DN for CA (Relative to BASE)
ca_rdn        = Root_CA

# ldap.conf
ldap_conf     = /etc/openldap/ldap.conf

# ldap password-file
ldap_password = /etc/openldap/ldap.secret
```

ldap.conf

This file is responsible for the configuration of the LDAP-Clients. There are several important options which *must* be set!

host sets the host of the ldap-server.

port sets the port, to which the ldap-clients try to connect. This is the only optional setting. All the others are mandatory.

base sets the base Distinguished Name (DN).

rootbinddn sets the root DN. To this DN the client binds if it wants to change something in the ldap-structure.

nss_base_passwd sets the DN where the useraccounts are stored in.

nss_base_group sets the DN where the groups are stored in.

sslpath sets the exact path to the Root-CA certificates of the LDAP-Server certificate.

ssl sets, whether to use a tls encrypted connection or not.

Here is an example of a `ldap.conf` configuration file:

```
BASE                    o=zhw, c=ch
HOST                    ksy006.zhwin.ch
PORT                    389

ROOTBINDDN              cn=admin, o=zhw, c=ch
nss_base_passwd         ou=people, o=zhw, c=ch?one
nss_base_shadow         ou=people, o=zhw, c=ch?one
nss_base_group          ou=group, o=zhw, c=ch?one

sslpath                 /etc/cacerts
ssl                     start_tls

SIZELIMIT               500
TIMELIMIT               15
DEREF                   never
```

ldap.secret

This file stores the administrator password for the ldap-server. Make sure that nobody else but root has read and write access to it.

```
bash# chmod 600 ldap.secret
```

slapd.conf

For a properly running LDAP-Server you have to check that the following schemas are included in the `slapd.conf` file.

```
include         /etc/openldap/schema/core.schema
include         /etc/openldap/schema/cosine.schema
include         /etc/openldap/schema/nis.schema
```

Although it isn't specially needed by our project, it is strongly recommended to configure the LDAP-Server to encrypt the communication. If you don't do this, all messages will be sent in clear text including the passwords. The following options must be set for a secure connection:

TLSCertificateFile sets the exact path to the Server-Certificate in PEM-format.

TLSCertificateKeyFile sets the exact path to the private key of the server.

TLSCypherSuite sets the encryption-algorithms.

TLSVerifyClient sets whether verify the client certificate or not.

An example for a `slapd.conf` configuration file can be found in 5.9.5 on page 92. For additional information about running an LDAP-Server, please have a look at the OpenLDAP homepage (http://www.openldap.org).

5.4.3 Creating the CA

After configuring the whole system, you can create your CA on the CA-Server with the

```
create_net_account_ca.pl
```

script which is stored in the `scripts` directory of the Smartcard Netlogin package. It parses the configuration files `smartcard_netlogin`, `ldap.conf` and `ldap.secret` and installs the CA with the given options.

The steps are the following:

1. **Create directories** The script tries to create the directory, in which the CA should be stored in. The path is given by the parameter **ca_dir** in the `smartcard_netlogin` configuration file.

2. **Create configuration files for the CA** The script writes some configuration files for the CA in the given directory. The CRL-distribution-point will be stored in this configuration files, as well as the DN of the certificates.

3. **Create CA** First, the script creates a self signed certificate for the Root-CA itself. Here you have to enter a new passphrase for the Root-CA and to confirm it. Then the script generates an empty CRL and ARL and converts them into DER-format. At last a server certificate is created. You have to enter the previously chosen passphrase another two times in this section of the script.

If something fails, you have to check what went wrong. Perhaps, some values in the configuration files are incorrect or missing. If so, correct the errors and try to start the script again. It may happen, that you have to remove the Root-CA-Directory manually, if it already exists (**rm 'Root-CA-Dir' -r**).

5.4.4 Distributing the Root-CA Certificate

An important step is the distribution of the Root-CA certificate. As the whole login security is based on the authenticity of it, you have to make sure that nobody can replace, modify or delete it. The directory where you have to copy the certificate into is the same as defined by the **sslpath** item in the `ldap.conf` file. Because some tools need it in

the DER and others in the PEM format, you have to distribute both formats. A third possibility is to access a certificate by a symbolic link of its hash value. You can create it as shown below using the **openssl** command. The Root-CA certificate can be found in the root directory of the CA as defined in the `smartcard_netlogin` configuration file.

```
bash# mount /floppy
bash# cd /path_to_ca
bash# cp PCAcert.pem PCAcert.der /floppy
bash# umount /floppy
```

We recommend to distribute the certificates by safe medium e.g. by floppy disc and copy it manually on each host. The path to the Root-CA certificates has to be the same as defined in the `ldap.conf` by the **sslpath** item.

```
bash# mkdir /path_to_ca_certs
bash# mount /floppy
bash# cd /path_to_ca_certs
bash# cp /floppy/PCAcert.* .
bash# ln -s PCAcert.pem ‘openssl x509 -hash -noout < PCAcert.pem‘.0
bash# chmod 644 *
bash# umount /floppy
```

5.4.5 Distributing the Server Certificate

For an encrypted connection to the LDAP-Server, you have to copy the LDAP-Server certificate onto the LDAP-Server. In the configuration file `slapd.conf` is given, where you have to store the certificate and the private key by the parameters **TLSCertificateFile** and **TLSCertificateKeyFile**. The copy procedure is similar to the distribution of the Root-CA certificate. First, you have to copy the LDAP-Server certificate and the private key onto a floppy disc. This must be done on the CA-Server.

```
bash# mount /floppy
bash# cd /path_to_ca
bash# cp ServerCert.pem private/ServerKey.pem /floppy
bash# umount /floppy
```

Then, on the LDAP-Server, you have to copy the certificate and the private key to their given locations and delete them from the floppy disc. The private key should only be readable by the user root (LDAP-Server).

```
bash# mount /floppy
bash# cp /floppy/ServerCert.pem /path_to_ServerCert
bash# cp /floppy/ServerKey.pem /path_to_ServerKey
```

```
bash# chmod 644 /path_to_ServerCert/ServerCert.pem
bash# chmod 600 /path_to_ServerKey/ServerKey.pem
bash# rm /floppy/Server*
bash# umount /floppy
```

Restart the server now, to make the changes take effect. Make sure that the LDAP-Server starts during the boot procedure.

5.4.6 Creating the LDAP-CA and storing the CRL

Now you can prepare the LDAP-Server for using with the Smartcard-Netlogin package. Start the script

```
create_net_login_ldap.pl
```

on the CA-Server. It adds the needed leafs to the LDAP tree and distributes the Root-CA Certificate and ARL/CRL to its given points.

5.5 User Migration

5.5.1 Migrating Users from existing passwd and group files

On the homepage of PADL Software (http://www.padl.com) you can find some perl scripts which allow you to migrate all your existing users and groups to the ldap directory. In addition to these scripts you need the **ldapadd** command which is part of the OpenLDAP package. After unpacking the scripts run **migrate_passwd.pl** and **migrate_group.pl** to create two ldif files, which then can be added to the ldap directory with **ldapadd**. You have to set the base DN, the DN of the ldap administrator and its password which are defined in the `lapd.conf` and `lapd.secret` files.

```
bash# tar -xvzf MigrationTools.tar.gz
bash# cd MigrationTools-XY
bash# export LDAP_BASEDN="base dn"
bash# ./migrate_passwd.pl /etc/passwd > passwd.ldif
bash# ./migrate_group.pl /etc/group > group.ldif
bash# ldapadd -D "admin dn" -x -W -ZZ -f passwd.ldif
bash# ldapadd -D "admin dn" -x -W -ZZ -f group.ldif
bash# rm passwd.ldif group.ldif
```

5.5.2 Adding and modifying users and groups

Apart of migrating existing users and groups into the ldap directory, the **netuseradd**, **netuserdel**,**netusermod**, **netgroupadd**, **netgroupdel** and **netgroupmod** commands allow it - according to the well known **useradd**, **userdel**, **usermod**, **groupadd**,**groupdel** and **groupmod** commands - to add, delete and modify user or group accounts network wide. For more information see the dedicated man pages.

The following example creates the user **test**:

```
bash# netuseradd -c "A Test User" test
```

To remove the user just call:

```
bash# netuserdel test
```

The default settings, which a new user is created with, can be set with the **netuseradd -D** command.

```
bash# netuseradd -D -g default_group -b default_home \
# -f default_inactive_time -e default_expire_date -s default_shell
```

The current defaults of the local login system are stored in the `/etc/default/useradd` file.

5.5.3 Create *real* Users and their Smartcards

For a properly working login, you have to make a *real* user with the command **netaccountadd** on the CA-Server. It will generate a new keypair and its correspondent certificate. The keys will be securely stored on the smartcard. The certificate will be stored on the ldap-server and the smartcard. It is necessary that the given user is already created and stored on the ldap-server using **netuseradd** or the migration tools mentioned before.

To create a *real* user, use the following command:

```
netaccountadd -uid username -p profiles -aut0 admin_pin -r reader_nr
```

All settings are optional, except for the -uid, where the user identification must be given. With the option **-p** you can set additional profiles, as which the user can login as. More than one additional profile are separated by comma. The option **-aut0** is used to give the administrator pin of the smartcard and is required if you don't want to use the standard shipping code from Schlumberger. With **-r** you can choose, which readernumber you

want to use. If this option is not given, readernumber 0 will be used. For additional information, see the **netaccountadd** man page.

Take care that at least one user has the rights to login as root. If you change the whole system and you have no smartcard with root access-rights, you have no chance to login as root anymore. A netaccount with root access you simply generate by executing something like:

```
netaccountadd -uid your_uid -p root
```

If you want to delete a *real* user, you can use the following tool:

```
netaccountdel -uid user
```

This command revokes the certificate of the given user and stores an updated CRL on the LDAP-Server. Thereby the given user can't login anymore. Worth mentioning is, that the user still exists on the LDAP-Server. If you want to remove the user completely, use the **netuserdel** command explained before.

5.6 Changing the Login Behavior

The following steps have to be done on each workstation and server which is going to use the new smartcard based login.

5.6.1 Configuring NSS

Since the GNU C Library supports the Name Service Switch (NSS) system, changing the behavior of the systemcalls which return information about users and groups is very easy.

To make the GNU C Library functions using the ldap directory instead of the local `passwd` and `group` files, we have to modify the NSS configuration file `/etc/nsswitch.conf`. All you have to do is changing the lines

```
passwd: files compat
shadow: files compat
group:  files compat
```

into

```
passwd: ldap
shadow: ldap
group:  ldap
```

5.6.2 Configuring Linux-PAM

To enable the smartcard login you have to change at least the `/etc/pam.d/login` configuration file. We recommend to change the `/etc/pam.d/passwd` and `/etc/pam.d/su` files for the **passwd** and **su** commands, too. 5.6.3 on the following page (if something goes wrong) describes how you can repair your system if the installation fails and it is not possible to login anymore.

Configuring the su command

All you have to do is changing the line

```
auth required /lib/security/pam_unix.so nullok
```

into

```
auth required /lib/security/pam_smartcard.so reader=0 cadir=/etc/cacerts
```

in the `/etc/pam.d.su` file. The **reader** parameter defines which reader to use and the **cadir** parameter sets the directory, where the module looks for Root-CA certificates. Now, you can test the new configuration by calling **su** as a normal user.

Configuring the login

This is almost the same as configuring the **su** command. Only that this time you must replace the line

```
auth required /lib/security/pam_unix.so nullok
```

with

```
auth required /lib/security/pam_smartcard.so reader=0 cadir=/etc/cacerts
```

in the `/etc/pam.d/login` file. Testing the new login is either possible by opening a new console with **ALT-F2** (**CTRL-ALT-F2** if you are working under X11 respectively) or - if you feel safe - by rebooting.

If you use a graphical login (XDM / KDM) you have to create or modify the `/etc/-pam.d/xdm` or `/etc/pam.d/kde` file respectively. You can find example configuration files in 5.9.8 on page 93.

Configuring the passwd command

The password used by the smartcard is often also called a PIN what stands for *P*ersonal *I*dentification *N*umber. However most smartcards allow you to use any printable character not only digits and you should make use of it. By editing the `/etc/pam.d/passwd` file you allow the users to change their new smartcard password the same way as they did it before with the shadow password. Replace the line

```
password required /lib/security/pam_unix.so use_first_pass
```

with

```
password required /lib/security/pam_smartcard.so cadir=/etc/cacerts
```

If there are other lines in the password stack like

```
password required /lib/security/pam_pwcheck.so nullok
```

change them into comment lines (insert '#' at the beginning of the line) or remove them.

5.6.3 If something goes wrong

If the **su** command and the login hang, the pcsc daemon probably crashed. If you are still logged in as root, try to restart the daemon or reboot your pc.

```
bash# /etc/init.d/pcsc restart
```

If it doesn't help, your configuration might be corrupt. To repair it start Linux in single user mode: Reboot again, hit a key while LILO is loading and type

```
LILO boot: linux single
```

on the LILO prompt. *(Replace 'linux' with 'name-of-your-normal-linux-image' ;-).* Depending on your distribution and whether there is a root password in `/etc/passwd` or not you will be asked for it. However, finally you will find yourself logged in as root in single user mode. Some distributions mount the root file system read only in this mode. To allow modifications you have to remount it:

```
bash# mount -o remount /
```

Now you have full access to the system and are able to do the necessary modifications. The error and warning messages in the `/var/log/messages` file may help you on that.

5.7 Patching OpenSSH 2.9.9p2

The X.509 Patch for OpenSSH adds the possibility to use the same smartcards and identities as for the smartcard netlogin system for secure local and remote login. Because of incompatibilities with the existing OpenSSH smartcard support, we had to write our own smartcard access, which uses the pc/sc middleware. Our patch uses the **x509v3-sign-rsa** keyformat to exchange X.509 certificates as defined in the IETF draft: `draft-ietf-secsh-transport-09.txt`. The patch can be found in the `openssh-patch` directory of the Smartcard Netlogin package.

5.7.1 Installation

If OpenSSH is already installed on your host you have to uninstall it first. Unpack the archive, patch, configure, compile and install it.

```
bash# tar -xvzf openssh-2.9.9p2.tar.gz
bash# cp path_to_patch/openssh-2.9.9p2-x509-smartcard-patch.gz .
bash# zcat openssh-2.9.9p2-x509-smartcard-patch.gz | patch -p 0
bash# cd openssh-2.9.9p2
bash# ./configure
bash# make
bash# make install
```

5.7.2 Configuration

Now you have two possibilities to enable the X.509 support: First, you can add the line

```
useX509cert 0
```

to the `ssh_config` file. Second, you can type **-z 0** when using the SSH-client (**ssh**). Worth mentioning is the **0**, that defines the readernumber which should be used.

To test the new SSH installation (re)start the SSH-daemon (**sshd**) and try to make a connection:

```
bash# ssh -z 0 your_SSH_daemon_host
```

If you want to see some debug information, start the daemon with **-d -d -d** and the client with **-v -v -v** (multiple **-d** / **-v** means more debugging).

5.8 Auxiliary Configurations

5.8.1 CRL Update

The Smartcard-Netlogin project sets the expiration of the CRL to 30 days. Therefore it is necessary to update it on a regular basis to avoid login problems. If an already expired CRL is stored on the LDAP-Server, nobody can login into the system and write a new one.

With the tool **crl_update** you can update the actual existing CRL. It parses the configuration files, generates a new crl and stores it on the LDAP-Server. You have to type in the passphrase of the Root-CA private key to sign the CRL. Also, everytime you revoke a netaccount using **netaccountdel** a new CRL will be created and stored on the LDAP-Server.

If you want to change the expiration-time of the crl, you can modify the `PCA.cnf` in the Root-CA directory. The option **default_crl_days** sets the default amount of days until a new generated CRL expires.

5.8.2 Changing the User's Password

If you have modified the `/etc/pam.d/passwd` as described in 5.6.2 on page 86 you can change your smartcard password by using the **passwd** as you are used to. Although the smartcard password is often called a PIN which stands for *P*ersonal *I*dentification *N*umber, most smartcards allow you to use any printable character not only digits and you should make use of it.

5.8.3 Unblocking the User's Password

After several unsuccessful tries of entering the user PIN, the card will be blocked. If you want to use your smartcard any further, you have to unblock the blocked PIN with the tool **smartcard_admin**.

For unblocking a blocked smartcard, you have to know the appropriate unblock pin, which is stored on the smartcard itself. If you have forgot the unblock pin, you can use the function described below to change the existing unblock pin by entering a new one.

The command to unblock a smartcard is:

```
smartcard_admin unblock -upin unblock_pin -chv1 new_chv1_pin
```

Where the **-upin** option is to give the current unblock pin and the **-chv1** option the new chv1-pin to be stored on the smartcard. For further information see the man page of **smartcard_admin**.

5.8.4 Changing the Smartcard unblock pin

In the case you have forgotten the unblock pin or simply want to change it, you can use **smartcard_admin changeupin**. For successful changing the unblock pin, you have to enter the smartcard administrator pin.

```
smartcard_admin changeupin -aut0 admin_pin -upin unblock_pin
```

With the **-aut0** option, you give the administrator pin, with the **-upin** option, the new unblock pin of the smartcard.

Take care that you don't use the wrong administrator pin too often. After several unsuccessful entries of a wrong aut0-pin, the card will be blocked and can't be unblocked anymore. For further information see the man page of **smartcard_admin**.

5.8.5 Changing the Smartcard Administrator's Password

All Cyberflex Access smartcards have an already stored aut0-pin (shipping code). If you want to change it, and we recommend that, you can use the following command:

```
smartcard_admin changeaut0 -aut0 admin_pin -naut0 new_admin_pin
```

With **-aut0** you give the actual applied administrator pin, with **-naut0** the new one, which should be stored on the smartcard. Take care that you don't use the wrong administrator pin too often. After several unsuccessful entries of a wrong aut0-pin, the card will be blocked and can't be unblocked anymore. For further information see the man page of **smartcard_admin**.

5.9 Example configurations

The following examples were tested under SuSE Linux 7.2 i386 systems.

5.9.1 Example of a `reader.conf` file

```
# Configuration file for pcsc-lite

FRIENDLYNAME     "Schlumberger Reflex 62"
DEVICENAME       GEN_SMART_RDR
LIBPATH          /usr/local/lib/libslb_rf60.so
CHANNELID        0x0102F8

FRIENDLYNAME     "Towitoko Chipdrive Micro"
DEVICENAME       TOWITOKO_CHIPDRIVE_MICRO
LIBPATH          /usr/local/lib/libtowitoko.so
CHANNELID        0x000001

# End of file
```

5.9.2 Example of a pcsc init-script

```
# Author: Mario Strasser (mast@gmx.net)
#         Martin Saegesser (m.sagi@bluemail.ch)
#
# init.d/pcsc
#
#   and symbolic its link
#
# /sbin/pcsc
#
# System startup script for the PC/SC daemon
#
### BEGIN INIT INFO
# Provides: pcsc
# Required-Start:
# Required-Stop:
# Default-Start:  S 1 2 3 5
# Default-Stop:   0 6
# Description:    Start the PC/SC daemon
### END INIT INFO

# Source SuSE config
. /etc/rc.config

# Determine the base and follow a runlevel link name.
base=${0##*/}
link=${base#*[SK][0-9][0-9]}

# Force execution if not called by a runlevel directory.
# test $link = $base && START_PCSC=yes
# test "$START_PCSC" = yes || exit 0

PCSC_BIN=/usr/sbin/pcscd
test -x $PCSC_BIN || exit 5

. /etc/rc.status

# First reset status of this service
rc_reset

case "$1" in
    start)
        echo -n "Starting PC/SC daemon (pcscd)"
        ## Start daemon with startproc(8). If this fails
        ## the echo return value is set appropriate.

        # startproc should return 0, even if service is
        # already running to match LSB spec.
        startproc $PCSC_BIN 2>&1

        # Remember status and be verbose
        rc_status -v
        ;;
    stop)
        echo -n "Shutting down PC/SC daemon (pcscd)"
        ## Stop daemon with killproc(8) and if this fails
```

```
        ## set echo the echo return value.

        killproc -TERM $PCSC_BIN

        # Remember status and be verbose
        rc_status -v
        ;;
    restart|reload)
        ## Stop the service and if this succeeds (i.e. the
        ## service was running before), start it again.
        $0 stop  &&  $0 start

        # Remember status and be quiet
        rc_status
        ;;
    status)
        echo -n "Checking for PC/SC daemon (pcscd): "
        ## Check status with checkproc(8), if process is running
        ## checkproc will return with exit status 0.

        # If checkproc would return LSB compliant ret values,
        # things could be a little bit easier here. This will
        # probably soon be the case ...
        checkproc $PCSC_BIN; rc=$?
        if test $rc = 0; then echo "OK"
        else echo "No process"
                if test -e /var/run/FOO.pid;
                then exit 1
                else exit 3
                fi
        fi
        #rc_status
        ;;
    *)
        echo "Usage: $0 {start|stop|status|restart|reload}"
        exit 1
        ;;
esac
rc_exit
```

5.9.3 Example of a `smartcard_netlogin` file

```
# CA-Directory
ca_dir        = /Root_CA/

# Relative DN for CA (Relative to BASE)
ca_rdn        = Root_CA

# ldap.conf
ldap_conf     = /etc/openldap/ldap.conf

# ldap password-file
ldap_password = /etc/openldap/ldap.secret
```

5.9.4 Example of a `ldap.conf` file

```
BASE                o=zhw, c=ch
HOST                ksy006.zhwin.ch
PORT                389

ROOTBINDDN          cn=admin, o=zhw, c=ch
nss_base_passwd     ou=people, o=zhw, c=ch?one
nss_base_shadow     ou=people, o=zhw, c=ch?one
nss_base_group      ou=group, o=zhw, c=ch?one

sslpath             /etc/cacerts
ssl                 start_tls

SIZELIMIT           500
TIMELIMIT           15
DEREF               never
```

5.9.5 Example of a `sldap.conf` file

```
include         /etc/openldap/schema/core.schema
include         /etc/openldap/schema/cosine.schema
include         /etc/openldap/schema/nis.schema

pidfile         /var/run/slapd.pid
argsfile        /var/run/slapd.args

loglevel        256
schemacheck     on

sizelimit       5000
timelimit       3600

#
# database
#

database        ldbm
lastmod         off

cachesize       1000
dbcachesize     100000

directory       /var/lib/ldap
suffix          "o=zhw, c=ch"

rootdn          "cn=Admin, o=zhw, c=ch"
rootpw          linux

#
# access rights
#

defaultaccess read
```

```
access to dn="ch=Admin, o=zhw, c=ch"
        by * none

access to *
        by * read

# SSL server

TLSCertificateFile    /etc/openldap/ServerCert.pem
TLSCertificateKeyFile /etc/openldap/ServerKey.pem
TLSCipherSuite        HIGH:MEDIUM
TLSVerifyClient       false
```

5.9.6 Example of a /etc/pam.d/su file

```
#%PAM-1.0
auth     sufficient     /lib/security/pam_rootok.so
auth     required       /lib/security/pam_smartcard.so   reader=0 cadir=/etc/cacerts
account  required       /lib/security/pam_unix.so
password required       /lib/security/pam_unix.so
#session required       /lib/security/pam_homecheck.so
session  required       /lib/security/pam_unix.so       debug # none or trace
```

5.9.7 Example of a /etc/pam.d/login file

```
#%PAM-1.0
auth     required       /lib/security/pam_smartcard.so  reader=0 cadir=/etc/cacerts/
auth     required       /lib/security/pam_securetty.so
auth     required       /lib/security/pam_nologin.so
#auth    required       /lib/security/pam_homecheck.so
auth     required       /lib/security/pam_env.so
auth     required       /lib/security/pam_mail.so
account  required       /lib/security/pam_unix.so
password required       /lib/security/pam_pwcheck.so
password required       /lib/security/pam_unix.so     use_first_pass use_authtok
session  required       /lib/security/pam_unix.so       none # debug or trace
session  required       /lib/security/pam_limits.so
```

5.9.8 Example of a /etc/pam.d/xdm file

```
#%PAM-1.0
auth     required       /lib/security/pam_smartcard.so  reader=0 cadir=/etc/cacerts
account  required       /lib/security/pam_unix.so
password required       /lib/security/pam_unix.so
session  required       /lib/security/pam_unix.so
session  required       /lib/security/pam_devperm.so
```

5.9.9 Example of a /etc/pam.d/kde file

```
#%PAM-1.0
auth     required       /lib/security/pam_smartcard.so  reader=0 cadir=/etc/cacerts
account  required       /lib/security/pam_unix.so
password required       /lib/security/pam_unix.so
session  required       /lib/security/pam_unix.so
session  required       /lib/security/pam_devperm.so
```

5.9.10 Example of a /etc/pam.d/passwd file

```
#%PAM-1.0
auth     required       /lib/security/pam_unix.so       nullok
account  required       /lib/security/pam_unix.so
password required       /lib/security/pam_smartcard.so  reader=0 cadir=/etc/cacerts
session  required       /lib/security/pam_unix.so
```

Kapitel 6

Tests

6.1 PAM-Modul

Das PAM-Modul ist das Kernstück des ganzen Projektes und bestimmt grösstenteils die Sicherheit des Login-Ablaufes. Der Überprüfung dieses Moduls wurde daher besondere Aufmerksamkeit geschenkt. Um sein Verhalten zu testen, wurde eine Liste von sinnvollen Fehlermöglichkeiten aufgestellt, Punkt für Punkt abgearbeitet und geprüft, ob das Verhalten den Erwartungen entspricht. Bei der Auswahl der Testpunkte wurden sowohl interne Abläufe als auch mögliche fehlerhafte Konfigurationen und Benutzereingaben berücksichtigt.

6.1.1 Login

Nr.	Beschreibung	Sys-Log Meldung	
1.1	keine Verbindung zum Reader	pc/sc error:	√
1.2	keine Smartcard eingesetzt	pc/sc error: no smartcard inserted	√
2.1	Passwort falsch	pc/sc error: internal error	√
3.1	kein Zertifikat	pc/sc error: internal error	√
3.2	falsches Zertifikat	can't parse certificate	√
3.3	Zertifikat abgelaufen	certificate is invalid	√
3.4	Zertifikat noch nicht gültig	certificate is invalid	√
3.5	Zertifikat gesperrt	certificate is invalid	√
3.6	Zertifikat verfälscht	certificate is invalid	√
4.1	keine ca Zertifikat	can't load ca certificates	√
4.2	kein passendes ca Zertifikat	can't load crl	√
5.1	keine CRL angegeben	can't load crl	√
5.2	falsche crl angegeben	can't load crl	√
5.3	crl kann nicht geladen werden	can't load crl	√
5.4	crl abgelaufen	can't load crl	√
5.5	crl verfälscht	can't load crl	√
6.1	Private und Public Key passen nicht	FAILED LOGIN X FROM...	√
6.2	kein passendes Profil vorhanden	user X is not allowed to login as Y	√
7.1	Passwort und Zertifikat I.O.	session started for user X, service Y	√

Da **su**, **xdm** und **kdm** die gleichen Modul-Funktionen wie **login** verwenden, wurden diese nicht mehr gleich ausführlich getestet, sondern nur ein prinzipieller Funktionstest durchgeführt.

6.1.2 Passwortänderung

Nr.	Beschreibung	Sys-Log Meldung	
1.1	keine Verbindung zum Reader	pc/sc error:	√
1.2	keine Smartcard eingesetzt	pc/sc error: no smartcard inserted	√
2.1	Altes Passwort falsch	pc/sc error: internal error	√
2.2	Neue Passwörter sind nicht identisch	*Aufforderung zur erneuten Eingabe*	√

6.2 Administrationstools

Allen Administrationstools sind die Konfigurationsdateien **ldap.secret**, **ldap.conf** und **smartcard_netlogin** gemein. Des weiteren wird von allen eine Verbindung zum LDAP-Server aufgebaut. Das Kontrollieren dieser beiden Punkte wird daher in den Abschnitten 6.2.1 und 6.2.2 auf der nächsten Seite für alle Tools zusammengefasst.

6.2.1 Parsen der Konfigurations Dateien

Die unten aufgeführte Liste testet das Verhalten bei Nichtvorhandensein einer Konfigurations-Datei oder eines Konfigurations-Eintrages. Durch Ausgabe der eingelesenen Werte als Debug-Information konnten diese zusätzlich auf ihre Korrektheit überprüft werden. Es gilt zu beachten, dass die Einträge **ca_dir** und **ca_rdn** nur von den Tools **netaccountadd**, **netaccountdel** und **update_crl** verwendet werden. Folglich wurden die Punkte 1.4 und 1.5 nur bei diesen getestet.

Nr.	Beschreibung	Fehlermeldung
1.1	Datei smartcard_netlogin fehlt	Error: Can't open configuration file smartcard_netlogin
1.2	Eintrag ldap_conf fehlt	Error: missing ldap_conf entry in smartcard_netlogin
1.3	Eintrag ldap_password fehlt	Error: missing ldap_password entry in smartcard_netlogin
1.4	Eintrag ca_dir fehlt	Error: missing ca_dir entry in smartcard_netlogin
1.5	Eintrag ca_rdn fehlt	Error: missing ca_dir entry in smartcard_netlogin
2.1	Datei ldap.conf fehlt	Error: Can't open ldap configuration file ldap.conf
2.2	Eintrag host fehlt	Error: missing host entry in ldap.conf
2.2	Eintrag rootbinddn fehlt	Error: missing rootbinddn entry in ldap.conf
2.2	Eintrag nss_base_group fehlt	Error: missing nss_base_group entry in ldap.conf
2.2	Eintrag ndd_base_passwd fehlt	Error: missing nss_base_passwd entry in ldap.conf
2.2	Eintrag ldap_people fehlt	Error: missing ldap_people entry in ldap.conf
2.2	Eintrag ldap_cadir fehlt	Error: missing ldap_cadir entry in ldap.conf
3.1	Datei ldap.secret fehlt	Error: Can't open ldap password file ldap.secret

Die Tests verliefen für alle Tools erfolgreich.

6.2.2 LDAP Verbindung

Ein Verbindungsaufbau ist mit (Punkte 1.1 bis 1.3) oder ohne TLS (Punkte 2.1 bis 2.5) Unterstützung möglich. Ist die Übertragung unverschlüsselt, wird mit einer Warnmeldung darauf hingewiesen.

Nr.	Beschreibung	Fehlermeldung
1.1	keine Verbindung zum LDAP-Server	timeout error
1.2	LDAP Passwort falsch	Error: ldap_insufficient_access
1.3	Administrator DN falsch	Error: ldap_insufficient_access
2.1	keine Verbindung zum LDAP-Server	timeout error
2.2	LDAP Passwort falsch	Error: ldap_insufficient_access
2.3	Administrator DN falsch	Error: ldap_insufficient_access
2.4	kein Root-CA Zertifikat	Error: ldap_operations_error
2.5	Root-CA Zertifikat ungültig	Error: ldap_operations_error

6.2.3 netuseradd, netuserdel, netusermod, netuserls

Das Testen der Netzwerkbenutzer Administrationstools wurde mit Hilfe der LDAP-Clients [gq] und **ldapsearch** durchgeführt. Als Erstes wurde mit den Tools das Benutzerverzeichnis auf dem LDAP-Server gezielt verändert. Anschliessend wurden die Änderungen mit Hilfe der LDAP-Clients angezeigt und auf ihre Richtigkeit überprüft.

netuseradd

Nr.	Parameter	Ergebnis/Fehlermeldung	
1.1	`-D`	Standardwerte werden angezeigt	√
1.2	`-D -g 100 -b /home/ -f -1 -e -1 -s /bin/bash`	Standardwerte werden nach Angabe gesetzt	√
2.1	`test1`	Benutzer `test1` wird erstellt	√
2.2	`test1`	Error: The user test1 does already exist!	√
2.3	`-c TEST2 -d /home/test2 -e 1 -f 1 -g game -s /bin/sh2 test2`	Benutzer `test2` wird nach den Angaben erstellt	√
2.4	`-g 800 test3`	Error: The group 800 does not exist!	√
2.5	`-G users,game test4`	Benutzer `test4` wird erstellt und den angegebenen Gruppen hinzugefügt	√
3.1	`-u 600 test5`	Benutzer `test5` mit Benutzernummer 600 wird erstellt	√
3.2	`-u 600 test6`	Error: A user with id 600 does already exist!	√
3.3	`-u 600 -o test6`	Benutzer `test6` mit Benutzernummer 600 wird erstellt	√
4.1		Usage: netuseradd ...	√
4.2	`-x test2`	Usage: netuseradd ...	√

netusermod

Nr.	Parameter	Ergebnis/Fehlermeldung	
1.1	`-l test7 test1`	Benutzer `test1` wird in `test7` umbenannt	√
1.2	`-l test7 test2`	Error: The user test7 does already exist!	√
1.3	`-l test7 test1`	Error: The user test1 does not exist!	√
2.1	`-c TEST` `-d /home/test -e 0` `-f 0 -g audio` `-s /bin/sh test2`	Einstellungen von Benutzer `test2` werden modifiziert	√
2.2	`-g 800 test2`	Error: The group 800 does not exist!	√
2.3	`-G audio,root test4`	Benutzer `test4` wird aus den alten Gruppen entfernt und zu den neuen hinzugefügt	√
3.1	`-u 601 test5`	Benutzernummer von Benutzer `test5` wird auf 601 geändert	√
3.2	`-u 601 test6`	Error: A user with id 601 does already exist!	√
3.3	`-u 601 -o test6`	Benutzernummer von Benutzer `test6` wird auf 601 geändert	√
4.1		Usage: netusermod ...	√
4.2	`-x test2`	Usage: netusermod ...	√

netuserdel

Nr.	Parameter	Ergebnis/Fehlermeldung	
1.1	`test2`	Benutzer `test4` wird gelöscht	√
1.2	`test1`	Error: The user test1 does not exist!	√
2.1		Usage: netuserdel user	√

netuserls

Nr.	Parameter	Ergebnis/Fehlermeldung	
1.1		Liste aller Benutzer wird ausgegeben	√
1.2	`test4`	Benutzer `test4` wird ausgegeben	√
1.3	`601`	Benutzer `test5` und `test6` werden ausgegeben	√
1.4	`test1`	keine Ausgabe	√

6.2.4 netgroupadd, netgroupdel, netgroupmod, netgroupls

Analog zur Überprüfung der Netzwerkbenutzer Administrationstools wurden auch beim Testen der Netzwerkgruppen Administrationstools vorgegangen. Folgende Punkte wurden im Einzelnen getestet:

netgroupadd

Nr.	Parameter	Ergebnis/Fehlermeldung	
1.1	`test1`	Gruppe `test1` wird erstellt	√
1.2	`-g 800 test2`	Gruppe `test2` mit der Gruppennummer 800 wird erstellt	√
1.3	`-g 800 test3`	Error: A group with id 801 does already exist!	√
1.4	`-g 800 -o test3`	Gruppe `test3` mit der Gruppennummer 800 wird erstellt	√
2.1		Usage: netgroupadd ...	√
2.2	`-x test2`	Usage: netgroupadd ...	√

netgroupmod

Nr.	Parameter	Ergebnis/Fehlermeldung	
1.1	`-n test4 test1`	Gruppe `test1` wird in `test4` umbenannt	√
1.2	`-n test4 test2`	Error: The group test4 does already exist!	√
1.3	`-n test4 test1`	Error: The group test1 does not exist!	√
1.2	`-g 801 test3`	Gruppennummer der Gruppe `test2` wird auf 801 geändert	√
1.3	`-g 801 test4`	Error: A group with id 800 does already exist!	√
1.4	`-g 801 -o test4`	Gruppennummer der Gruppe `test4` wird auf 801 geändert	√
2.1		Usage: netgroupmod ...	√
2.2	`-x test2`	Usage: netgroupmod ...	√

netgroupdel

Nr.	Parameter	Ergebnis/Fehlermeldung	
1.1	`test2`	Gruppe `test2` wird gelöscht	√
1.2	`test1`	Error: The group test1 does not exist!	√
2.1		Usage: netgroupdel group	√

netgroupls

Nr.	Parameter	Ergebnis/Fehlermeldung	
1.1		Liste aller Gruppen wird ausgegeben	√
1.2	`test4`	Gruppe `test4` wird ausgegeben	√
1.3	`801`	Gruppen `test3` und `test4` werden ausgegeben	√
1.4	`test1`	keine Ausgabe	√

6.2.5 netaccountadd, netaccountdel, crl_update

Auch hier wurde zum Überprüfen der Änderungen im LDAP-Verzeichnis die LDAP-Clients [gq] und **ldapsearch** verwendet. Die Test-Smartcards waren vom Typ Schlumberger Cyberflex Access.

netaccountadd

Nr.	Beschreibung	Ergebnis/Fehlermeldung	
1.1	keine Verbindung zum Reader	Can't connect to reader	√
1.2	keine Smartcard eingesetzt	Can't connect to reader	√
1.3	falsche Smartcard	Can't select MF ...	√
1.4	Smartcard Administrator PIN falsch	Can't authenticate ...	√
2.1	Root CA Verzeichnis fehlt	Can't change directory	√
2.2	Root CA Verzeichnis falsch	Can't open file ldapcert.cnf	√
2.3	kein Schreibrecht im Root CA Verzeichnis	Can't change directory	√
3.1	falsche Passphrase	unable to load ca private key	√
4.1	Benutzer ist nicht im LDAP Verzeichnis	User does not exist on ldap-server	√
4.2	Benutzer hat schon ein Zertifikat	User already has a certificate ...	√
4.3	Benutzer ist nur in der CA registriert	Can't sign certificate ...	√
5.1	Aufruf: `netaccountadd`	Usage: netaccountadd ...	√
5.2	Aufruf: `netaccountadd -x`	Usage: netaccountadd ...	√

Eine Überprüfung, ob die Daten korrekt auf die Smartcard übertragen werden, wurde einerseits anhand der Debug Ausgaben und andererseit implizit durch die Verwendung mit dem PAM-Modul durchgeführt.

netaccountdel

Nr.	Beschreibung	Ergebnis/Fehlermeldung	
1.1	Root CA Verzeichnis fehlt	Can't change directory	√
1.2	Root CA Verzeichnis falsch	Can't open openssl index file	√
1.3	kein Schreibrecht im Root CA Verzeichnis	Can't change directory	√
2.1	falsche Passphrase	unable to load ca private key	√
3.1	Benutzer ist nicht in der CA registriert	no such user in CA	√
3.2	Benutzer ist bereits widerrufen, besitzt aber noch ein Zertifikat im LDAP Verzeichnis	Something failed while revoking the certificate. Do you want to remove the certificate from the LDAP-Server? (y/n)	√
3.3	Benutzer ist nur in der CA registriert	done.	√
3.4	Benutzer ist in der CA registriert und besitzt ein LDAP Eintrag	done.	√
4.1	Aufruf: `netaccountdel`	Usage: netaccountdel ...	√
4.2	Aufruf: `netaccountdel -x`	Usage: netaccountdel ...	√

crl_update

Nr.	Beschreibung	Ergebnis/Fehlermeldung	
1.1	Root CA Verzeichnis fehlt	Can't change directory	√
1.2	Root CA Verzeichnis falsch	crl generation failed	√
1.3	kein Schreibrecht im Root CA Verzeichnis	Can't change directory	√
2.1	falsche Passphrase	unable to load ca private key	√

6.2.6 smartcard_admin

Für die folgenden Tests wurde eine original Schlumberger Cyberflex Access Smartcard verwendet (aut0 = AD9F61FEFA20CE63, upin1 = 11111111 und chv1 = 00000000).

Nr.	Beschreibung/Parameter	Ergebnis/Fehlermeldung	
1.1	keine Verbindung zum Reader	Can't connect to reader	√
1.2	keine Smartcard eingesetzt	Can't connect to reader	√
1.3	falsche Smartcard	Can't select MF	√
2.1	10 mal Aufruf von `passwd` mit Passwort 22222222	Authentication token manipulation error	√
2.2	Aufruf von `passwd` mit Passwort 00000000	Authentication token manipulation error	√
2.3	`unblock -upin 11111111 -chv1 33333333`	done	√
2.4	Aufruf von `passwd` mit Passwort 00000000	Authentication token manipulation error	√
2.5	Aufruf von `passwd` mit Passwort 33333333	-	√
3.1	15 mal Aufruf von `unblock -upin 22222222 -chv1 00000000`	Error: can't unblock chv1 pin ...	√
3.2	`unblock -upin 11111111 -chv1 00000000`	Error: can't unblock chv1 pin ...	√
3.3	`changeupin -aut0 0xad9f61fefa20ce63 -upin 44444444`	done	√
3.4	`unblock -upin 11111111 -chv1 00000000`	Error: can't unblock chv1 pin ...	√
3.5	`unblock -upin 33333333 -chv1 00000000`	done	√
4.1	`changeaut0 -aut0 0xad9f61fefa20ce63 -naut0 55555555`	done	√
4.2	`changeaut0 -aut0 0xad9f61fefa20ce63 -naut0 66666666`	Error: Can't authenticate ...	√
4.3	`changeupin -aut0 0xad9f61fefa20ce63 -upin 11111111`	Error: Can't authenticate ...	√
4.4	`changeupin -aut0 55555555 -upin 11111111`	done	√
4.5	`changeaut0 -aut0 55555555 -naut0 0xad9f61fefa20ce63`	done	√

6.3 Installations-Scripts und HOWTO

Die beiden Perl-Scripts **create_netlogin_ca.pl** und **create_netlogin_ldap.pl** wurden im Zusammenhang mit der HOWTO Erstellung geschrieben und vereinfachen die Installation des **Smartcard-Netlogin** Paketes. Sie wurden gleich wie das HOWTO im Rahmen des Systemtests (Abschnitt 6.5 auf Seite 103) überprüft.

6.4 SSH-Patch

Weil einerseits die Kernfunktionen betreffend Smartcards und X.509 Zertifikaten vom PAM-Modul übernommen wurden und daher als getestet galten, und andererseits das Testen von einzelnen Modifikationen bei einem Patch nur sehr schwer möglich ist, konzentrierten sich unsere Tests auf den komplett angepassten und übersetzten SSH-Client bzw. SSH-Daemon. Das auf der Test-Smartcard gespeicherte Zertifikat enthielt zusätzlich ein Profil für den Benutzer Root.

Der von uns geschriebene OpenSSH Patch sollte die ursprüngliche Funktionalität von SSH-Client und SSH-Daemon in keiner Weise beeinträchtigen. Diesem Umstand wurde bei den Tests Rechnung getragen, indem zusätzlich die ursprünglichen Authentisierungsverfahren überprüft wurden. Ausgehend von einer funktionierenden original SSH Konfiguration wurden unsere angepassten Programme installiert und bei eingeschaltetem Debug Modus folgende Tests durchgeführt:

Nr.	Aufruf	Ergebnis	
1.1	`sshd -d -d -d` (auf Host ksy006)	Daemon started, liest die Konfiguration ein und setzt insbesondere den Pfad zu den Root Zertifikaten korrekt	√
2.1	`ssh -v -v -v ksy006`	Anschliessend zum Diffie-Helman Schlüsselaustausch wird eine Verbindung im Modus: **password** aufgebaut. Nach Eingabe des Passwortes wird eine Remote Shell geöffnet.	√
2.2	`ssh-keygen ...`	Ein neuer RSA Schlüssel erzeugt anschliessend auf den Host ksy006 kopiert	√
2.3	`ssh -v -v -v ksy006`	Anschliessend zum Diffie-Helman Schlüsselaustausch wird eine Verbindung im Modus: **publickey: ssh-rsa** aufgebaut. Nach Eingabe der Passphrase des Private Key wird eine Remote Shell geöffnet.	√
3.1	`ssh -v -v -v -z 0 ksy006`	Anschliessend zum Diffie-Helman Schlüsselaustausch wird eine Verbindung im Modus: **publickey: x509v3-sign-rsa** aufgebaut. Nach Eingabe des Smartcard PINs wird eine Remote Shell geöffnet.	√
3.2	`ssh -v -v -v -z 0 -l root ksy006`	Anschliessend zum Diffie-Helman Schlüsselaustausch wird eine Verbindung im Modus: **publickey: x509v3-sign-rsa** aufgebaut. Nach Eingabe des Smartcard PINs wird eine Remote Shell für den Benutzer root geöffnet.	√
3.3	`ssh -v -v -v -z 0 -l XXX ksy006`	Anschliessend zum Diffie-Helman Schlüsselaustausch wird eine Verbindung im Modus: **publickey: x509v3-sign-rsa** aufgebaut. Nach Eingabe des Smartcard PINs wird die Verbindung abgelehnt.	√
3.4	`ssh -v -v -v -z ksy006`	Usage: ssh ...	√

Zuletzt wurde im SSH Konfigurationsfile (`ssh_config`) der Eintrag **useX509cert 0** eingefügt und die Tests 3.1 bis 3.3 ohne die Angabe **-z 0** wiederholt und sichergestellt, dass die Resultate identisch waren.

6.5 System-Test

Das Austesten des ganzen Login-Systems wurde auf drei Arten realisiert:

1. Sobald erste lauffähige Versionen des PAM-Moduls und der Tools verfügbar waren, wurden diese auf den benutzten Arbeitsplätzen installiert und so durch den täglichen Gebrauch über einen Zeitraum von ca. 4 Wochen getestet.

2. Auf zwei Testsystemen wurde SuSE Linux 7.2 installiert und anschliessend anhand des von uns geschriebenen HOWTOs unser **Smartcard-Netlogin** Paket installiert. Dabei entdeckte Unstimmigkeiten wurden aufgenommen und korrigiert. Anschliessend wurden diverse Benutzer und Gruppen angelegt und mit ihrer Hilfe folgendes ausgetestet:

   - Konsolen Login (login)
   - Benutzerwechsel (su)
   - Passwortänderung (passwd)
   - Grafischer Login (kdm, xdm)
   - Remote Login (ssh)
   - Secure Copy (scp)
   - Bildschirmschoner (xlock, kde-lock)

   Dabei wurde festgestellt, dass einzig der KDE Bildschirmschoner bzw. die KDE Arbeitsplatzsperrung, die beide auf dem gleichen Prozess aufbauen, nicht korrekt arbeiteten. Der Grund dafür lag in einem zu kurzen Timeout der ansonsten korrekt arbeitenden PAM-Login-Funktion. Diese bricht die etwas länger dauernde Smartcard Login Prozedur frühzeitig ab. Die einzige Möglichkeit, dieses Fehlverhalten zu korrigieren, liegt in einer Anpassung und Neukompilation der entsprechenden KDE Quellen. Des Weiteren wurden einzelne Benutzer auch widerrufen und das Verhalten der oben erwähnten Programme (respektive des PAM-Moduls) überprüft. Aufgrund der Änderungen am Installations-HOWTO wurden die beiden Testsysteme anschliessend gelöscht und der ganze Vorgang nochmals wiederholt. Bis auf die bekannten Probleme mit dem KDE Bildschirmschoner traten keine weiteren Probleme mehr auf.

3. Nach Abschluss der Implementations- und Test-Phase wurde das ganze Paket ins Internet gestellt und auf den entsprechenden Mailinglisten angekündigt. Die von den Benutzern zurückgelieferten Bugs wurden behoben und die Verbesserungsvorschläge soweit möglich noch umgesetzt.

Kapitel 7

Schlussbemerkungen

7.1 Fazit und Ausblick

Zusammenfassend lässt sich sagen, dass wir mit dem Erreichten grundsätzlich zufrieden sind. Dank der reibungslosen Implementierung des PAM-Moduls und der Administrationstools konnten wir den eng gesetzten Zeitplan einhalten und schafften es auch noch, einen funktionstüchtigen OpenSSH Patch zu erstellen. Dabei kamen uns unsere, in früheren Projekten gesammelten, Erfahrungen im Umgang mit Smartcards und PAM-Modulen zugute. Ohne diese Vorkenntnisse wäre die Diplomarbeit im gesetzten Rahmen wohl kaum realisierbar gewesen, da die anderen beiden Schwerpunkte, LDAP und X.509 Zertifikate, zu umfangreich waren.

Der aktuelle Stand unserer Arbeit weist noch einige kleine Mängel auf. So ist zum Beispiel das Speichern der Login-Profile mit Hilfe der **Subject Alternative Name** Extensions eine in unseren Augen unbefriedigende Lösung. Sinnvoller wäre es, dies mit Hilfe von *Attributszertifikaten* [itu9594] zu realisieren. Mangels unterstützung durch Bibliotheken und Programme (z.B. OpenSSL) war dies zur Zeit jedoch nicht im Rahmen unserer Diplomarbeit realisierbar. Einer prinzipiellen Umsetzung steht jedoch nichts im Wege.

Des weiteren könnte die OpenSSH Unterstützung weiter ausgebaut werden. Neben einer zertifikatsbasierten Host Authentisierung, wäre eine Anpassung des SSH-FTP-Server (**sftp-server**) und des SSH-FTP-Clients (**sftp**) wünschenswert. Mittelfristig sollten auch Smartcards verwendet werden, die PKCS #15 Standard [pkcs15] kompatibel sind.

7.2 Danksagungen

Wir möchten uns bei dieser Gelegenheit bei allen bedanken, die uns bei unserer Projektarbeit unterstützt haben. Insbesondere bei:

- Unserem Dozenten Herrn **Dr. Andreas Steffen**, der uns mit Rat und Tat zur Seite stand.
- Der **Zürcher Hochschule Winterthur** für die Bereitstellung der von uns benötigten Infrastruktur.
- Den Experten Herrn **M. Uhlig** und **W. Krammel**, dass sie sich die Zeit und Mühe nehmen sich unsere Arbeit anzusehen und zu bewerten.

Winterthur, 29.10.2001

Martin Sägesser

Mario Strasser

Anhang A

Inhalt der CD-ROM

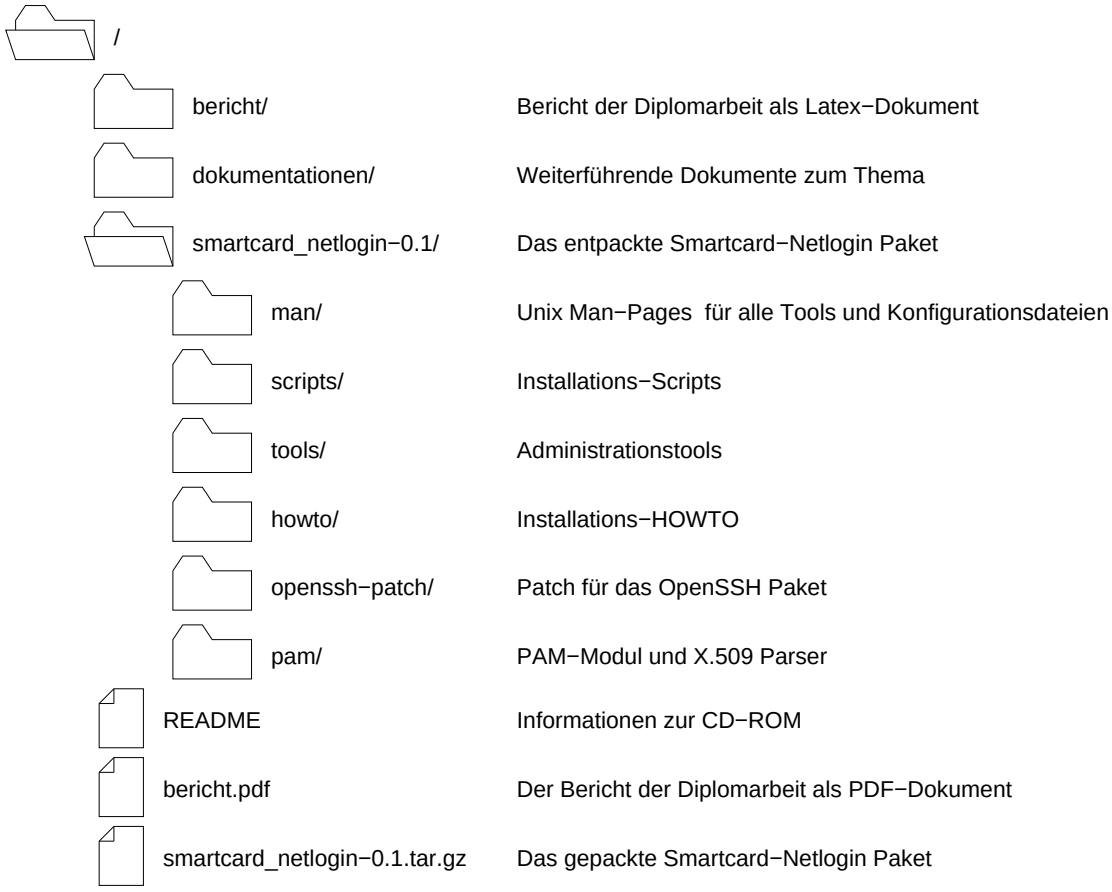


Abbildung A.1: Inhalt der CD-ROM

Literaturverzeichnis

[wolf]

Wolfgang Rankl / Wolfgang Effing
Handbuch der Chipkarten
Hanser, 1999, 3. Auflage

[wobst]

Reinhard Wobst
Abenteuer Kryptologie - Methoden, Risiken und Nutzen der Datenverschlüsselung
Addison-Wesley, 1998, 2. Auflage

[sederwick]

Sederwick Robert
Algorithmen in C++
Addison-Wesley, 1999, 5. Auflage

[ldap]

Jens Banning
LDAP unter Linux
Addison-Wesley, 2001

[cybe]

Schlumberger Cyberflex Programmer's Guide
http://www.cyberflex.com/Support/CyberflexPG.pdf

[pamadm]

The Linux-PAM System Administrator's Guide
http://www.kernel.org/pub/linux/libs/pam/

[pamapp]

The Linux-PAM Application Developer's Guide
http://www.kernel.org/pub/linux/libs/pam/

[pammod]

The Linux-PAM Module Writer's Guide
http://www.kernel.org/pub/linux/libs/pam/

[pcsc]

M.U.S.C.L.E PC/SC API
http://www.linuxnet.com/docs.html

[javacardapi]

Sun JavaCard 2.1.1 API
http://java.sun.com/products/javacard/javacard21.html

[klamssh]

SSH - Secure Shell
http://www.klammeraffe.org/~brandy/artikel/ssh.html

[gq]

GQ LDAP client
http://biot.com/gq

[openssh]

OpenSSL Handbuch
http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/

[openLDAP]

OpenLDAP Homepage
http://www.openldap.org

[gnump]

The GNU MP Homepage
http://www.swox.com/gmp/

[tc]

Trust Center - Key to Internet Security
http://www.trustcenter.de/

[sshtrans]

Internet Draft - SSH Transport Layer Protocol
http://www.ietf.org/internet-drafts/draft-ietf-secsh-transport-09.txt

[pkcs15]

PKCS #15: Cryptographic Token Information Format Standard
http://www.rsa.com/rsalabs/pkcs/pkcs-15/

[itu9594]

ITU-T Recommendation X.509 - ISO/IEC 9594-8 - Public-Key and Attribute Certificate Frameworks
http://www.itu.int

[rfc2440]

RFC 2440 - OpenPGP Message Format
http://www.ietf.org/rfc/rfc2440.txt

[rfc2459]

RFC 2459 - X.509 Public Key Infrastructure Certificate and CRL Profile
http://www.ietf.org/rfc/rfc2459.txt

[rfc2307]

RFC 2307 - Using LDAP as a Network Information Service
http://www.ietf.org/rfc/rfc2307.txt

[rfc822]

RFC 822 - Standard for the Format of ARPA Internet Text Messages
http://www.ietf.org/rfc/rfc822.txt

[rfc86.0]

RFC 86.0 - Unified Login with Pluggable Authentication Modules (PAM)
http://www.kernel.org/pub/linux/libs/pam/pre/doc/rfc86.0.txt.gz

Index