Zürcher Hochschule Winterthur

Projektarbeit

Linux-Login mit RSA-SmartCard

Dozent: Dr. Andreas Steffen

Autoren: Martin Sägesser & Mario Strasser

Mai 2001

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Motivation und Ziele . . . 1

1.2 Ausblick . . . 1

1.3 Danksagungen . . . 2

2 Theoretische Grundlagen **3**

2.1 Linux-PAM . . . 3

2.1.1 Grundlagen der Konfiguration . . . 3

2.1.2 PAM Modul . . . 5

2.1.3 PAM Applikation . . . 6

2.2 Grundlagen zu SmartCards . . . 7

2.2.1 Die verschiedenen SmartCard-Arten . . . 7

2.2.2 SmartCard Dateitypen . . . 7

2.2.3 Dateistrukturen der EFs . . . 8

2.2.4 Dateinamen . . . 10

2.2.5 Zugriffsbedingungen auf Dateien . . . 10

2.2.6 Struktur der SmartCard-Befehle . . . 12

2.3 Zwei SmartCards von Schlumberger . . . 13

2.3.1 Schlumberger Cyberflex Access Class 00 . . . 13

2.3.2 Schlumberger Cryptoflex 8K . . . 18

2.4 Das RSA Public Key Verfahren . . . 20

2.4.1 Schlüsselerzeugung . . . 20

2.4.2 Verschlüsselung . . . 20

2.4.3 Entschlüsselung . . . 21

3 Realisierung 22

3.1 Projektaufbau 22

3.1.1 Übersicht 22

3.1.2 Bibiliotheken 23

3.2 Verwendete Hard- und Software 24

3.2.1 Verwendete SmartCards 24

3.2.2 Verwendete Kartenleser 25

3.2.3 Verwendete Middleware 26

3.2.4 Verwendete Software 26

3.3 Aufbau unserer Login SmartCards 27

3.3.1 Datei-Struktur auf unseren SmartCards 27

3.3.2 Zugriffsrechte der einzelnen Dateien 29

3.4 Das PAM Modul 30

3.4.1 Authentisierung 30

3.4.2 Passwortänderung 34

3.5 Entstandene Tools 36

3.5.1 Programm udat 36

3.5.2 Programm makecard 37

3.5.3 Programm managecard 38

3.5.4 Programm cleancard 39

4 Installation 40

4.1 Installation of the Smartcard Reader 40

4.1.1 Installing PC/SC 40

4.1.2 Installing the Reader Driver 41

4.1.3 Configure the Driver 41

4.1.4 Make the PC/SC Daemon starting on startup 42

4.2 Installation of the PAM Module and the Smartcard Tools 42

4.3 Creating the User's Login Smartcard 43

4.3.1 Creating a User Account 43

4.3.2 Creating a minimal Smartcard 43

4.3.3 Adding a Profile 43

4.4 Configuring the Smartcard Login 43
4.4.1 Configuring the su Command 44
4.4.2 Configuring the login 44
4.4.3 Configuring the passwd Command 44
4.4.4 Additional Possibilities with the Password 45
4.5 If Something goes wrong 45
4.6 Example configurations 46
4.6.1 Example of a `reader.conf` file 46
4.6.2 Example of a `pcsc` script 46
4.6.3 Example of a `/etc/pam.d/login` file 49
4.6.4 Example of a `/etc/pam.d/su` file 49
4.6.5 Example of a `/etc/pam.d/passwd` file 49

Tabellenverzeichnis

2.1 Reservierte FIDs 11
2.2 Von Cyberflex-Karten unterstützte Befehle 15
2.3 Von Cryptoflex-Karten unterstützte Befehle 19

3.1 Verzeichnistruktur des Projektes 23

Abbildungsverzeichnis

2.1 Organisation von Linux-PAM . . . 4
2.2 Transparente Dateistruktur . . . 8
2.3 Linear fixed Dateistruktur . . . 9
2.4 Linear variable Dateistruktur . . . 9
2.5 Cyclic Dateistruktur . . . 10
2.6 Die Struktur einer Kommando-APDU . . . 12
2.7 Die Struktur einer Antwort-APDU . . . 13
2.8 Die Systematik der nach ISO/IEC 7816-4 spezifizierten Returncodes. . . . 13

3.1 Aufbau des SmartCard Login Projektes . . . 23
3.2 Schlumberger Reflex 60 . . . 25
3.3 Gemplus GCR410 . . . 26
3.4 Towitoko Chipdrive . . . 26
3.5 Dateistruktur unserer Login-SmartCards. . . . 28
3.6 Ablauf eines Benutzerlogins . . . 31
3.7 Ablauf einer Passwortänderung . . . 35

Zusammenfassung

Die schon seit langem verwendeten Login-Verfahren mit Passwörtern, sogenannten *One Factor Authentication Systems*, bergen gewisse Sicherheitsrisiken in sich. Obwohl sie technisch sehr sicher gemacht werden können, scheitern sie meist an der 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.

Ein weiteres Problem stellt das Speichern der Passwörter dar. Obwohl sie normalerweise verschlüsselt abgelegt werden, bieten sie eine gute Angriffsmöglichkeit. Zum Beispiel gelang es einigen Studenten dieser Schule einen Grossteil der hiesigen Passwörter mit einem simplen Wörterbuch-Angriff zu knacken.

Um nun zu einem etwas geschützeren *Two Factor Authentication System* zu gelangen, muss der Benutzer nicht nur das Passwort wissen, sondern auch noch eine Art Schlüssel besitzen. Dieser kann zum Beispiel in Form einer SmartCard vorliegen. Für eine weitere Verbesserung (*Three Factor Authentication*) wird noch zusätzlich ein persönliches Merkmal des Benutzers abgefragt. Hier sind Fingerabdrücke, Iris-Scans oder Stimmenerkennung denkbar.

Das Ziel dieser Arbeit ist es, eine Verbesserung des aktuellen Sicherheitsstandards unter Linux zu erreichen. Durch das dort verwendete *Pluggable Authentication System (PAM)* ist es ohne grosse Änderungen möglich, eine zusätzliche Sicherheitsstufe in den Anmeldevorgang zu integrieren. Wir beschlossen, ein *Two Factor Authentication System* mit RSA-SmartCards als Schlüssel zu erstellen.

Je nach verwendeter SmartCard wird das RSA-Schlüsselpaar direkt von der Karte selbst oder von unseren Tools erstellt. Die Private Keys können nach der Erstellung nicht mehr ausgelesen werden und sind deshalb auch im Falle eines Kartenverlustes sicher. Die Public Keys werden zentral in einer Datei verwaltet und den einzelnen Benutzern zugeordnet.

Beim Anmeldevorgang wird als erstes ein Passwort abgefragt, womit die SmartCard freigeschaltet wird. Danach wird eine vom Computer erstellte Zufallszahl als Challenge zur Karte gesendet, von ihr mit Hilfe des Private Keys signiert und zurückgesandt. Das Resultat kann nun mit Hilfe des Public Keys des Benutzers entschlüsselt und mit der ursprünglichen Zufallszahl verglichen werden. Stimmen sie überein, gilt der Benutzer als authentisiert. Bei diesem Verfahren wird nur der Public Key auf dem Computer gespeichert. Das Passwort liegt sicher auf der Karte und kann, wie der Private Key, nicht mehr gelesen werden.

Die im Rahmen dieser Arbeit programmierten Tools ermöglichen das Erstellen der benötigten SmartCards und erlauben somit die Installation eines lauffähigen Systems. Das gesamte Paket (Tools und PAM-Module) kann im Internet heruntergeladen werden und steht unter *GNU Public License*. Aus zeitlichen Gründen realisierten wir nur den lokalen Login. Das Anmelden über ein Netzwerk ist zum jetzigen Zeitpunkt nicht möglich. Vom Konzept her ist es aber vorgesehen und auch realisierbar.

Kapitel 1

Einleitung

1.1 Motivation und Ziele

In den letzten Jahren setzten sich sogenannte *One Factor Authentication Systems* als standard Login-Verfahren durch. Dabei wird nur das Passwort des Benutzers abgefragt. Dieses Verfahren birgt jedoch gewisse Sicherheitsrisiken, da viele Benutzer zu bequem sind sich sichere Passwörter zu merken. Obwohl diese auf dem System verschlüsselt abgelegt werden, ist es mit einem simplen Wörterbuch-Angriff möglich, unsichrere Passwörter zu knacken.

Desshalb beschlossen wir, uns dieser Problematik anzunehmen und das modular aufgebaute Sicherheitssystem von Linux (Linux-PAM) zu verbessern. Eine genauere Betrachtung zeigt, dass das Hauptproblem darin besteht, dass geheime Daten öffentlich zugänglich sind. Das Verwenden eines Public Key Verfahrens würde diese Problematik aufheben, falls der Private Key an einem sicheren Ort abgelegt werden könnte. Dies brachte uns auf die Idee, SmartCards zu verwenden. Neuere Prozessorkarten können eine RSA Verschlüsselung selbständig durchführen und die notwendigen Schlüsselpaare selbst erstellen. Dadurch muss der Private Key die SmartCard nie mehr verlassen.

Eine Suche im Internet ergab, dass die Firma Schlumberger für unsere Zwecke geeignete SmartCards im Angebot hat, für die auch eine Entwicklungsumgebung für Linux vorhanden ist. Unser Dozent, Herr Dr. Andreas Steffen, zeigte sich interessiert und ermöglichte uns, unsere Ideen im Rahmen einer Projektarbeit zu verwirklichen.

1.2 Ausblick

Das von uns gesetzte Ziel, das Realisieren eines Logins mit Hilfe einer RSA SmartCard, wurde erreicht. Aus zeitlichen Gründen realisierten wir nur den lokalen Login. Das Anmelden über ein Netzwerk ist zum jetzigen Zeitpunkt nicht möglich. Vom Konzept her ist es aber vorgesehen und auch realisierbar. Das Projekt steht unter *GNU Public License* und steht zum freien Download bereit. Bei positiver Resonanz kommt eine Weiterführung als Diplomarbeit in Frage. Diese könnte neben der Erstellung eines netzwerkfähigen Systems auch die bessere Nutzung der verwendeten Karten (mehrere Profile auch auf der *Cryptoflex*, Erstellung des RSA Schlüsselpaares auf der *Cyberflex*) beinhalten.

1.3 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 es uns ermöglichte, die von uns vorgeschlagene Projektarbeit durchzuführen.
- Der **Zürcher Hochschule Winterthur** für die Bereitstellung der von uns benötigten Infrastruktur.
- Herrn **David Corcoran**, dem Leiter des M.U.S.C.L.E Projektes für das kompetente und prompte Beantworten unserer Fragen.

Kapitel 2

Theoretische Grundlagen

2.1 Linux-PAM

Linux-PAM (Pluggable Authentication Modules for Linux) ist eine Sammlung von Authentifikations Modulen, die eine gemeinsame (im RFC-86.0 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 ihrerseit können für mehr als eine Aufgabe konzipiert sein und zum Beispiel Funktionen zur Authentisierung des Benutzers, zur Passwortänderung oder auch für das Sessionmanagement bereit stellen. 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.

2.1.1 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 *servcie-name* Feld fehlt. Ansonsten sind die Konfigurationsdateien identisch und besitzen einen für die UNIX-Welt typischen Aufbau:

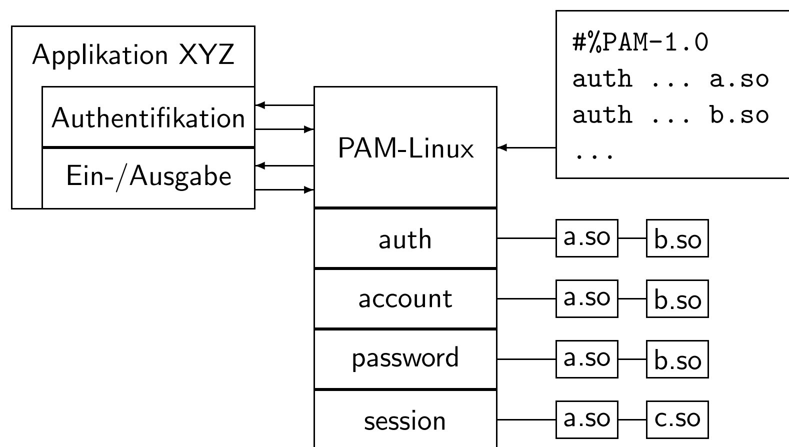

Abbildung 2.1: Organisation von Linux-PAM

- 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 eine Passworte) 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 Modules 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 Modules 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 vom Modul nicht berücksichtigt, muss dies von ihm mit `syslog` festgehalten werden.

2.1.2 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 deteilliertere 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.

2.1.3 PAM Applikation

Damit eine Applikation die Vorteile der Linux-PAM Module nutzen kann, muss sie gegen die Bibliothek `libpam` gelinkt 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, liefert das Endergebnis des ganzen Modul Stapels zurück.

`pam_open_session(), pam_close_session()` öffnet bzw. schliesst eine Benutzer-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.

2.2 Grundlagen zu SmartCards

2.2.1 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 im öffentlichen Personenverkehr als elektronisches Ticket.

2.2.2 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.

2.2.3 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.

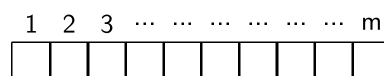

Abbildung 2.2: Transparente Dateistruktur

linear fixed Diese Dateistruktur, die in Abbildung 2.3 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.

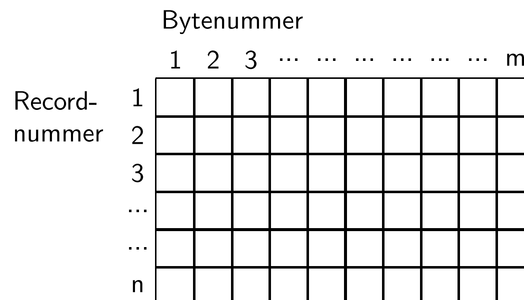

Abbildung 2.3: Linear fixed Dateistruktur

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.

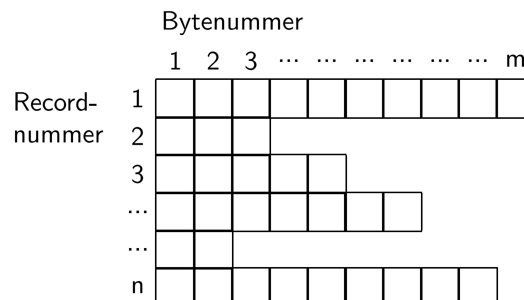

Abbildung 2.4: Linear variable Dateistruktur

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 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.

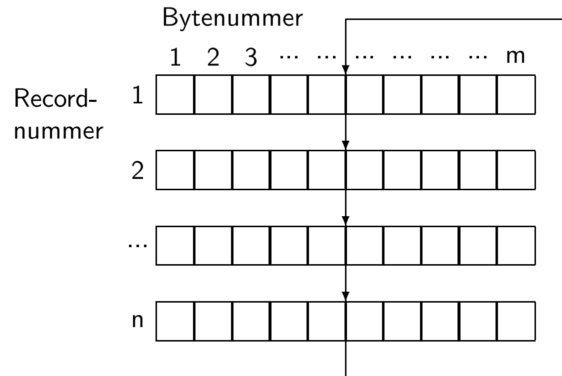

Abbildung 2.5: Cyclic Dateistruktur

2.2.4 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 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 gleich 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.

2.2.5 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.

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.

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 DateiDIR 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

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

2.2.6 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.

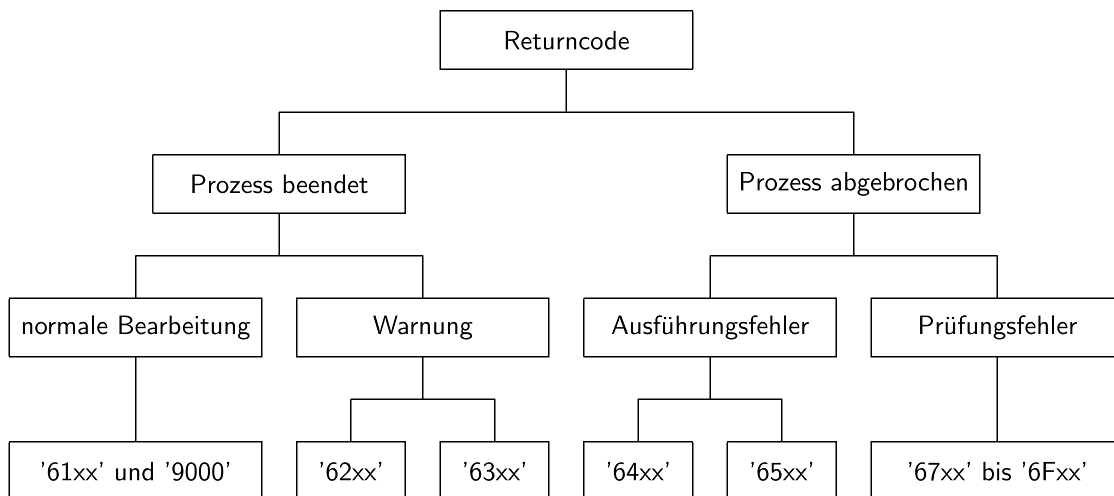

Abbildung 2.8: Die Systematik der nach ISO/IEC 7816-4 spezifizierten Returncodes.

2.3 Zwei SmartCards von Schlumberger

2.3.1 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 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

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

Hierbei ist der Dateityp einer der folgenden:

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 CHV-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 Datensatzen 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 angegebene 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.

2.3.2 Schlumberger Cryptoflex 8K

Die Cryptoflex Karte hat einen Mikroprozessor, der auf kryptographische Anwendung optimiert wurde. Schlüssel und Zertifikate können sicher auf der Karte aufbewahrt werden. Wie bei der Cyberflex können Daten direkt auf der Karte signiert werden, was bedeutet, dass der *Private Key* niemals die Karte verlassen muss. Die Signatur mit einem 1024-bit RSA-Schlüssel wird von der Karte in weniger als einer halben Sekunde ausgeführt.

Vorhandene Befehle

In der Tabelle 2.3 sind alle vorhandenen Befehle der Cryptoflex 8K aufgeführt. Des weiteren werden wir nur noch die Befehle, die noch nicht bei der Cyberflex Karte beschrieben wurden kurz erläutert. Eine vollständige Dokumentation dieser Karte ist unter [crypt] zu finden.

Create File erstellt eine neue Datei. Das Dateiformat ist aber geringfügig anders als bei der Cyberflex Karte.

Create Record entspricht dem Befehl *Append Record* der Cyberflex.

Dir Next listet die in einem Verzeichnis verfügbaren Dateien auf. Dabei kann man bei jedem Aufruf gezielt Informationen über die aktuelle Datei abrufen.

Generate RSA Keys Mit diesem Befehl wird auf der Karte ein neues RSA-Schlüsselpaar generiert. Dabei kann mit dem Parameter *key lgth* angegeben werden, wie gross das Schlüsselpaar sein soll (40, 60 oder 80 Byte). Der Parameter *pub exp lgth* gibt an, wie gross der *Public Exponent* sein soll. Dieser muss im Datenteil der APDU übergeben werden.

Get AC Keys wird dazu verwendet, die Zugriffsbedingungen einer gerade angewählten Datei anzuzeigen. Leider ist es bei dieser Karte nicht möglich, diese zu ändern. Dazu müsste man die Datei erst löschen und dann mit den gewünschten Zugriffsrechten neu erstellen.

Get Challenge entspricht dem Befehl *Ask Random* der Cyberflex Karte.

Logout AC entspricht dem Befehl *Logout All* der Cyberflex Karte.

RSA Signature entspricht dem Befehl *Internal Auth* der Cyberflex Karte.

SHA-1 Intermediate wird benutzt, um den Hash einer Meldung zu bilden. Ist die Meldung kleiner als 64 Bytes, muss der Befehl *SHA-1 Last* verwendet werden.

SHA-1 Last wird benutzt um den Hash des letzten Datenteils einer Meldung zu generieren.

Befehl	**Class**	**INS**	**P1**	**P2**	**P3**
Change CHV	F0	24	00	CHV num	10
Create File	F0	E0	init	num recs	lgth
Create Record	C0	E2	00	00	lgth
Decrease	F0	30	00	00	03
Delete File	F0	E4	00	00	02
Dir Next	F0	A8	00	00	lgth
External Authenticate Using DES	C0	82	00	00	07
Generate RSA Keys	F0	46	key num	key lgth	pub exp lgth
Get AC Keys	F0	C4	00	00	03
Get Challenge	C0	84	00	00	output lgth
Get Response	C0	C0	00	00	output lgth
Increase	F0	32	00	00	03
Internal Authenticate Using DES	C0	88	00	key num	08
Invalidate	F0	04	00	00	00
Logout AC	F0	22	acc cond	00	00
Read Binary	C0	B0	offset MSB	offset LSB	lgth
Read Record	C0	B2	rec num	mode	lgth
Rehabilitate	F0	44	00	00	00
RSA Signature (Internal Auth)	C0	88	00	key num	key lgth
Seek	F0	A2	offset	mode	lgth
Select	C0	A4	00	00	02
SHA-1 Intermediate	14	40	00	00	40
SHA-1 Last	04	40	00	00	08-40 oct
Unblock CHV	F0	2C	00	CHV num	10
Update Binary	C0	D6	offset MSB	offset LSB	lgth
Update Record	C0	DC	rec num	mode	lgth
Verify CHV	C0	20	00	CHV num	08
Verify Key	F0	2A	00	key num	08/0F lgth

Tabelle 2.3: Von Cryptoflex-Karten unterstützte Befehle

2.4 Das RSA Public Key Verfahren

RSA ist wohl das bekannteste und am weitesten verbreitete asymetrische Verschlüsselungs Verfahren. 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 verweise 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.

Kapitel 3

Realisierung

3.1 Projektaufbau

3.1.1 Übersicht

Wie bereits erwähnt, war das Ziel unserer Projektarbeit das Erstellen einer voll funktionsfähigen, wenn auch eingeschränkten Erweiterung des Linux Login-Systems. Das bedeutete für uns nicht nur die Implementation des PAM Modules, sondern auch diverser Administrations Tools zur Verwaltung der benötigten SmartCards. Da dabei viele Aufgaben, wie z.B. das Erstellen von Dateien auf der SmartCard, wiederholt auftraten, drängte sich eine Modularisierung geradezu auf. Des weiteren liess sich so eine gute Arbeitsteilung im Team realisieren. Ausgehend von den drei Hauptgebieten PAM Modul, RSA Verschlüsselung und SmartCards versuchten wir festzustellen, welche Funktionen wann und wo benötigt werden und kamen zu folgendem Ergebnis:

- Das Projekt wurde in die beiden Teilgebiete *PAM Modul* und *Administrations Tools* aufgeteilt. Ersteres wurde von Mario Stasser und letzteres von Martin Sägesser bearbeitet.
- Es sollten drei Administrations Tools für das Löschen (`cleancard`), das Erstellen (`makecard`) und das Administrieren (`managecard`) der SmartCards erstellt werden.
- Zusätzlich wird dem normalen Benutzer ein Tool zur Verwaltung seiner Daten auf der SmartCard zur Verfügung gestellt (`udat`).
- Gemeinsam benötigte Funktionen sollten in Bibliotheken ausgelagert werden. RSA betreffende Dinge wie die Ver- und Entschlüsselung, sowie das Erstellen von Schlüsselpaaren wurden in der Bibliothek `librsa` zusammengefasst. Die Bibliothek `libcardtools` abstrahiert die Kommunikation mit den Readern bzw. SmartCards, indem sie für alle von uns benötigten Aufgaben Funktionen bereitstellt.

Abbildung 3.1 zeigt die Zusammenhänge der einzelnen Module und Programme grafisch auf. Die Verzeichnisstruktur des Projektes und die Namen der einzelnen Quelldateien können Tabelle 3.1 entnommen werden.

Name	Datei	Author
RSA Bibliothek	libs/rsa.c	Mario Strasser
SmartCard Bibliothek	libs/cardtools.c	Martin Sägesser
PAM Modul	pam_smartcard/pam.c	Mario Strasser
udat	udat/udat.c	Martin Sägesser
cleancard	cat/cleancard.c	Martin Sägesser
makecard	cat/makecard.c	Martin Sägesser
managecard	cat/managecard.c	Martin Sägesser
HOWTO	howto/sgml/SmartCard-Login-HOWTO.sgml	Mario Strasser
pcscd Init-Script	addons/pcsc	Mario Strasser
udat Man-Page	man/udat.1	Martin Sägesser
cleancard Man-Page	man/cleancard.1	Martin Sägesser
makecard Man-Page	man/makecard.1	Martin Sägesser
managecard Man-Page	man/managecard.1	Martin Sägesser

Tabelle 3.1: Verzeichnistruktur des Projektes

3.1.2 Bibiliotheken

Die beiden von uns erstellten Bibliotheken sind nur für den internen Gebrauch unseres Projektes gedacht. Alle Funktionen sind in den jeweiligen Header-Dateien ausführlich dokumentiert und werden hier nicht nochmals extra aufgeführt. Eine ausdruckbare Version der API-Dokumentation kann mit Hilfe von KDOC oder einem anderen, zu JavaDoc kompatiblen Tool aus den Header-Dateien generiert werden.

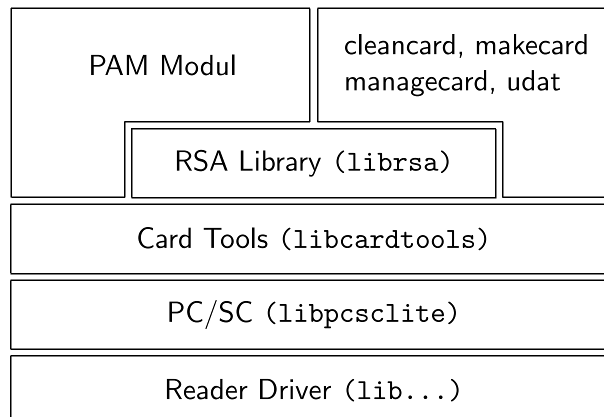

Abbildung 3.1: Aufbau des SmartCard Login Projektes

3.2 Verwendete Hard- und Software

3.2.1 Verwendete SmartCards

Wie schon im Kapitel 2.3 aufgezeigt, verwendeten wir in unserem Projekt zwei Kartentypen von Schlumberger. Leider mussten wir feststellen, dass die zwei Karten nicht unterschiedlicher sein könnten. Obwohl sie von der gleichen Firma geliefert werden, sind wichtige Dinge wie Zugriffsrechte und Kommandos gänzlich verschieden.

Cyberflex Access

Ein Ziel unserer Projektarbeit war, dass die RSA-Schlüssel direkt auf der Karte erstellt werden. Um dies zu realisieren, verfolgten wir anfangs eine Lösung mit der JavaCard *Cyberflex Access*. Wie von Sun in der *JavaCard 2.1.1 API* [javacardapi] dokumentiert, sollte es kein Problem sein, ein *Cardlet* zu schreiben, das ein neues Schlüsselpaar generiert.

Leider zeigte sich jedoch, dass Sun hier zur Zeit nur die Dokumentation bereitgestellt hat, aber noch keinen Code. Alles was die Methode zur Schlüsselgenerierung macht, ist ein Fehler zurückzugeben.

Bis zu diesem Zeitpunkt hatten wir unsere Tools schon so weit lauffähig, dass sie einsatzbereit waren und wir sie nicht mehr verwerfen wollten. Deshalb schlugen wir eine neue Strategie ein und bestellten noch *Cryptoflex* Karten, welche die RSA-Schlüsselgenerierung auf der Karte beherrschen sollten.

Cryptoflex

Wie erwartet funktionierten unsere Tools nicht mit diesen Karten, da diese einen vollständig anderen Befehlssatz verwenden und auch die Class-Bytes verschieden sind. Leider half das blosse Anpassen der *APDUs* nicht, da gewisse Befehle der *Cyberflex* Karte nicht vorhanden sind und die *Cryptoflex* noch andere Gemeinheiten auf Lager hat.

Nach genauerem Studium der Dokumentation von Schlumberger [crypt] stellte sich heraus, dass das Löschen von Dateien auf einer *Cryptoflex* Karte in der umgekehrten Reihenfolge ihrer Erstellung zu erfolgen hat. Deshalb mussten wir bei der *Cryptoflex* Karte auf mehrere Profile auf einer Karte verzichten.

Ebenfalls problematisch war die Anpassung der bei der *Cyberflex* gewählten Zugriffsbedingungen. Bei der *Cryptoflex* Karte wurde eine sehr undurchsichtige und unnötig komplizierte Variante der Zugriffsbedingungen realisiert. Das führte dazu, dass wir Kompromisse eingehen mussten, um eine ähnliche Struktur auf der *Cryptoflex* wie auf der *Cyberflex* zu erhalten.

Als grösstes Problem zeigte sich, dass der von uns anfänglich verwendete Kartenleser *Schlumberger Reflex 60* durch ein *Timeout-Problem* nicht fähig ist, die SmartCard-Funktion *Generate RSA Keys* fehlerfrei auszuführen. Nach längerem Suchen im Quellcode gaben wir schliesslich auf und versuchten unser Glück mit zwei weiteren Kartenlesern (siehe Kapitel 3.2.2).

3.2.2 Verwendete Kartenleser

Da sich im Verlauf unseres Projektes herausstellte, dass unser erster Kartenleser nicht einwandfrei mit der Cryptoflex Karte funktionierte, testeten wir noch zwei weitere Leser, von denen wir nur den Towitoko Chipdrive empfehlen können.

Schlumberger Reflex 60

Weil im Vorfeld der Projektarbeit die Wahl der SmartCard auf eine Schlumberger Cyberflex fiel, dachten wir uns, dass es sinnvoll wäre, auch einen Reader der gleichen Firma zu verwenden.

Da es nur für den Reflex 60 Linux-Treiber gibt, fiel die Wahl nicht schwer. Dieser kleine Kartenleser hat als Zusatz noch eine PIN-Tastatur, die wir jedoch nicht verwendet haben, da wir die Benutzer nicht einschränken wollten und auch Buchstaben in den Passwörtern erlaubten.

Abbildung 3.2: Schlumberger Reflex 60

Durch seine zu leichte Bauform muss man eine Karte immer mit beiden Händen einführen und herausziehen. Die Lösung der Firma Schlumberger ist das Anbringen eines Klett-Streifens an der Unterseite des Lesers. Damit könnte man den Leser an eine Unterlage festkleben und trotzdem noch versetzen.

Als wir in einem zweiten Anlauf noch die Cryptoflex Karte testeten, fiel uns auf, dass man bei diesem Reader keine Möglichkeit hat, den Timeout für die Befehlsausführung auf der Karte zu erhöhen. Das führte dazu, dass bei einem *Generate RSA Keys* Befehl die Kommunikation vom PC zur Karte unterbrochen und ein Reset der Karte ausgeführt wurde.

Gemplus GCR410

Der Aufbau macht auch einen eher billigen Eindruck. Dadurch, dass der Kartenschlitz auf der oberen Seite angebracht ist, braucht man für das Einsetzen einer Karte nur eine Hand.

Auch nach längerem Ausprobieren mit den eigentlich vorhandenen Linux-Treibern brachten wir diesen Leser nicht zum Laufen.

Abbildung 3.3: Gemplus GCR410

Towitoko Chipdrive

Dieser Reader besticht durch seine solide Bauform. Im Sockel ist ein Gewicht eingearbeitet, damit man den Kartenleser nicht verrückt, wenn man eine SmartCard einsetzt oder herauszieht.

Abbildung 3.4: Towitoko Chipdrive

Dieser Kartenleser lief auch sofort nach dem Installieren des Treibers. Allerdings mussten wir für die Datenübertragung noch den Timeout auf einen höheren Wert stellen, damit wir überhaupt eine 80 Byte lange *Challenge* auf die Karte übertragen konnten. Immerhin war es sehr schnell möglich, die Stelle im Quellcode zu finden: In der Datei `ifd_towitoko.c` musste der Wert von `IFD_TOWITOKO_TIMEOUT` auf `400` erhöht werden.

3.2.3 Verwendete Middleware

Als Middleware für die Kommunikation mit der SmartCard hat sich unter Windows der PC/SC Standard durchgesetzt. Dieser besitzt einen zentralen Ressourcen-Manager der die Zugriffe auf den Reader bzw. die SmartCard verwaltet. Seit einiger Zeit bestehen Bemühungen, diesen Standard im Rahmen des M.U.S.C.L.E Projektes nach Linux zu portieren. Obwohl noch weitere Linux-Projekte mit ähnlichen Zielen bestehen, fiel unsere Wahl auf die PC/SC Portierung von M.U.S.C.L.E. Neben einer gut dokumentierten API [pcsc] zeichnet sie sich auch durch eine wachsende Anzahl von Readertreibern aus.

3.2.4 Verwendete Software

Vor allem in der Startphase des Projektes war es nötig, sich mit den SmartCards vertraut zu machen. Dabei kamen uns die folgenden Tools zu Hilfe.

Schlumberger `xcard`

Als Bestandteil des *Cyberflex for Linux Starter's Kit 2.1* von Schlumberger ist dieses Programm eine grosse Hilfe beim Einstieg mit SmartCards. Allerdings wurde es extra für die *Cyberflex* Karte programmiert, was zu kleineren Problemen führt, wenn es mit einer *Cryptoflex* Karte verwendet wird.

Die eigentliche Hauptfunktion dieses Programmes ist das Anzeigen der einzelnen Verzeichnisse und das Absenden von APDUs. Obwohl die Menüpunkte noch einiges mehr versprechen, sind weitere Funktionen kaum brauchbar.

Sehr praktisch ist der APDU-Manager, mit dem es möglich ist, einzelne APDUs an die Karte zu senden. Dadurch ist es möglich, einzelne Funktionen der Karte zu testen, ohne jedesmal ein eigenes Programm schreiben zu müssen.

Als sehr negativ fiel hier auf, dass es nicht möglich ist, ganze Abfolgen von APDUs ablaufen zu lassen. Ebenfalls nicht sehr praktisch ist die Tatsache, dass keine APDUs gespeichert werden können. Dies war aber für das Erstellen der Verzeichnisstruktur wichtig, damit man sofort kleine Änderungen am ganzen System anbringen konnte, ohne dass man die vielen Befehle neu eingeben musste. Für diesen Zweck kam bei uns das Tool SKAM zum Einsatz.

SKAM von Fernando Llobregat Baena

Mit diesem Programm ist es möglich, vorbereitete Abfolgen von APDU-Befehlen abzuarbeiten. Dadurch konnten wir notwendige Änderungen an den einzelnen Skripts sehr schnell und bequem anbringen. So hatten wir nach jeder Änderung die Möglichkeit, die positiven und negativen Eigenschaften einer Verzeichnisstruktur auszutesten.

Leider ist auch dieses Programm nicht ausgereift. Man kann nicht direkt im Programm die Skripts erstellen und es ist auch nicht möglich, die einzelnen Befehle Schritt für Schritt ablaufen zu lassen. Dennoch ist dieses Tool sehr nützlich, wenn man eine Sequenz von Befehlen austesten möchte, aber nicht jedesmal alles selbst neu programmieren will.

3.3 Aufbau unserer Login SmartCards

Da wir eine möglichst flexible Lösung für den Gebrauch der SmartCards anstrebten, versuchten wir mehrere Benutzerprofile auf einer Karte abzuspeichern.

Ein Benutzer kann sich dadurch bei mehreren verschiedenen Accounts am System anmelden, ohne dass er dafür verschiedene Karten braucht. Dies ist zum Beispiel für den Administrator sehr praktisch, da dieser ja sehr oft zwischen normalem Account und 'root'-Account wechselt.

3.3.1 Datei-Struktur auf unseren SmartCards

Die Datei-Struktur auf der SmartCard wählten wir nach dem in Abbildung 3.5 gezeigten Format.

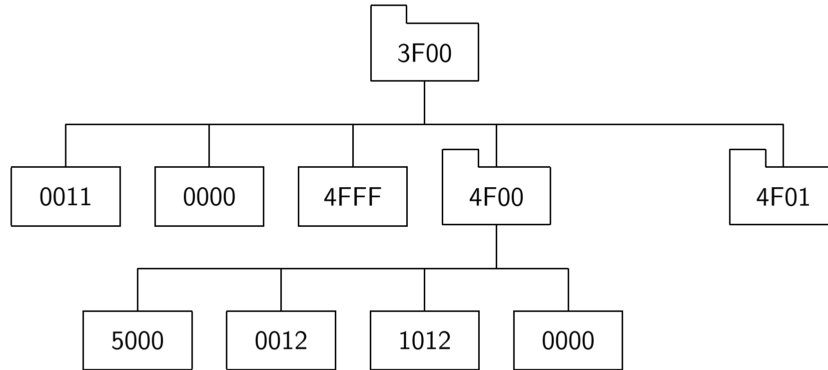

Abbildung 3.5: Dateistruktur unserer Login-SmartCards.

Die einzelnen Dateien haben dabei folgende Bedeutung:

3F00 Wie in Kapitel 2.2.4 beschrieben ist dies das Rootverzeichnis der Karte.

0011 In dieser Datei ist der Schlüssel für *Aut0* gespeichert. Je nach verwendeter Karte sind hier auch noch weitere Schlüssel gespeichert, die aber für unser Projekt keine Bedeutung haben.

0000 beinhaltet einen PIN-Code. Diese Datei ist bei den *Cyberflex* Karten schon vorhanden, wird aber von uns nicht verwendet. Bei den *Cryptoflex* Karten müssen wir, wegen der restriktiveren Zugriffsrechte, diese Datei temporär anlegen. Sobald die Karte aber erstellt wurde, hat sie keine Bedeutung mehr und wird nicht mehr gebraucht.

4FFF Dies ist die erste Datei, die wirklich von unserem Projekt stammt. Hier werden die Namen der auf der Karte vorhandenen Profile gespeichert. Die Datei selbst ist vom Typ *linear variable*.

Anz	Name 1	Nr	Name 2	Nr	Name x	Nr

Der erste Record (Anz) ist ein Byte lang und beinhaltet die Anzahl vorhandener Profile. Der zweite Record enthält den Namen, der nächste die Nummer des Profils. Je nachdem wie viele Profile auf der Karte vorhanden sind, wird dieser Aufbau wiederholt (gestrichelt dargestellt). Die Nummer des Profils gibt an, in welchem Unterverzeichnis die betreffenden Schlüssel zu finden sind.

4F00 Dies ist das Hauptverzeichnis des Kartenbesitzers. In diesem werden neben den normalen Profil-Daten (Schlüssel) auch noch persönliche Daten wie Adresse, Telefonnummer und e-mail-Adresse gespeichert.

4F01 In diesem und den folgenden (bis 4FFE) Verzeichnissen werden nur noch die Schlüssel der betreffenden Profile abgespeichert. Diese Verzeichnisse existieren jedoch nur, wenn mehrere Profile auf der Karte gespeichert sind.

Da man bei der *Cryptoflex* Karte Dateien nur in der umgekehrten Reihenfolge des Erstellens wieder löschen kann, mussten wir bei diesen Karten darauf verzichten, mehrere Profile zu speichern.

5000 Dieses *linear variable* EF ist für die Speicherung der persönlichen Daten des Kartenbesitzers zuständig:

Nr	Bedeutung
1	First Name
2	Last Name
3	Address
4	ZIP
5	City
6	Phone Business
7	Mobile Business
8	Phone Private
9	Mobile Private
10	e-mail Business
11	e-mail Private

0012 Enthällt den *Private Key*. Nach der Erstellung ist es nicht mehr möglich, diese Datei auszulesen.

1012 Der *Public Key* könnte durch geeignete Befehle wieder ausgelesen werden. Dies wird aber von unseren Tools nicht unterstützt. Lediglich bei der Erstellung eines neuen Profils wird dieser Schlüssel ausgelesen und auf der Harddisk in der Datei `/etc/pubkey` abgespeichert.

0000 In dieser Datei ist der Kartenbesitzer-PIN abgespeichert (CHV). Dieser PIN wird dazu verwendet, den Besitzer gegenüber der Karte zu authentisieren. Erst nach Eingabe dieses PINs können weitere Befehle auf der Karte ausgeführt werden. Dabei überprüft die Karte selbst den übergebenen Wert. Der gespeicherte PIN verlässt, wie der *Private Key*, nie die Karte und kann auch nicht ausgelesen werden.

3.3.2 Zugriffsrechte der einzelnen Dateien

Damit wichtige Dinge wie der *Private Key* nicht ausgelesen werden können, haben wir sehr restriktive Zugriffsbedingungen gesetzt. Wir waren der Meinung, dass es besser ist, wenn man ab und zu eine Karte neu erstellen muss, dafür aber Gewissheit hat, dass die Daten nicht ausgelesen werden können.

Zugriffrechte bei der Cyberflex Karte

Das Setzen der Zugriffsrechte ist bei dieser Karte sehr einfach gelöst. So kann für jeden einzelnen Benutzerlevel definiert werden, was möglich ist und was nicht. Da für unser Projekt nur die Zugriffsrechte für Aut0, CHV1 und der default wichtig sind, haben alle anderen keine Rechte und werden auch in der folgenden Tabelle nicht aufgeführt:

Datei	Aut0	CHV1	default
3FFF	Alle Rechte	Create File / Delete File	keine Rechte
4FFF	Alle Rechte	Alle Rechte	Read
4F00	Create File / Delete File	Alle Rechte	keine Rechte
0000	Alle Rechte ohne Read	Write	keine Rechte
0012	Alle Rechte ohne Read	keine Rechte	keine Rechte
1012	Alle Rechte	Read	keine Rechte
5000	Alle Rechte	Alle Rechte	Read

Zugriffsrechte bei der Cryptoflex Karte

Leider ist bei dieser Karte das Setzen der Zugriffsrechte sehr kompliziert. So kann man hier nicht, wie sonst üblich, Rechte für einen bestimmten Zugriffslevel setzen. Bei der *Cryptoflex* ist es so gelöst, dass man für die verschiedenen möglichen Aktionen, die auf der Karte ausgeführt werden können, zuweisen muss, welcher *Benutzerlevel* mit welchem PIN zugreifen darf.

Da es wenig Sinn machen würde, die verwirrenden Zugriffsbedingungen aufzuschreiben, werden nur die für unser Projekt wesentlichen Restriktionen aufgezeigt. Die zwei Dateien 0000 (im 4Fxx Verzeichnis) und 0012, die es besonders zu schützen gilt, können hier nie ausgelesen werden. Es ist dabei gleichgültig, ob man sich als Aut0 und/oder als CHV ausgewiesen hat. Da deshalb die übrigen Dateien nicht mehr sonderlich geschützt werden müssen, sind diese auch nicht mehr so stark eingeschränkt.

Sehr wichtig war es, dass der CHV (das heist der Kartenbesitzer) seine persönlichen Daten ändern kann. Dadurch mussten wir gewährleisten, dass die Datei 5000 als letztes auf die Karte geschrieben wird. Da im PAM-Modul die Suche nach einem geeigneten Profil vor der eigentlichen Authentifizierung stattfindet, mussten wir bei der Datei 4FFF dafür sorgen, dass man diese auch ohne Anmeldung auslesen kann.

3.4 Das PAM Modul

Die Hauptaufgabe unseres PAM Modules ist es, den Benutzer mit Hilfe der SmartCard zu authentisieren. Die dafür zu implementierenden Funktionen heissen `pam_sm_authenticate()` und `pam_sm_setcred()`. Erstere nimmt die eigentliche Authentisierung vor, und letztere liefert zur Zeit, mangels einer sinnvollen Aufgabe, immer ein positives Resultat. Des weiteren unterstützt das Modul auch das Ändern des Passwortes auf der SmartCard, wofür die Funktion `pam_sm_chauthtok()` zuständig ist.

3.4.1 Authentisierung

Im Folgenden wird nun aufgezeigt, wie die zuständige Funktion `pam_sm_authenticate()` einen Benutzer im Detaill authentisiert. Es wird darauf verzichtet, den eigentlichen Sourcecode abzudrucken und zu erläutern, da dies Sache des Sourcecode-Kommentars ist. Stattdessen wird Schritt für Schritt beschrieben, wie eine Benutzeranmeldung abläuft und was wann und wie überprüft wird.

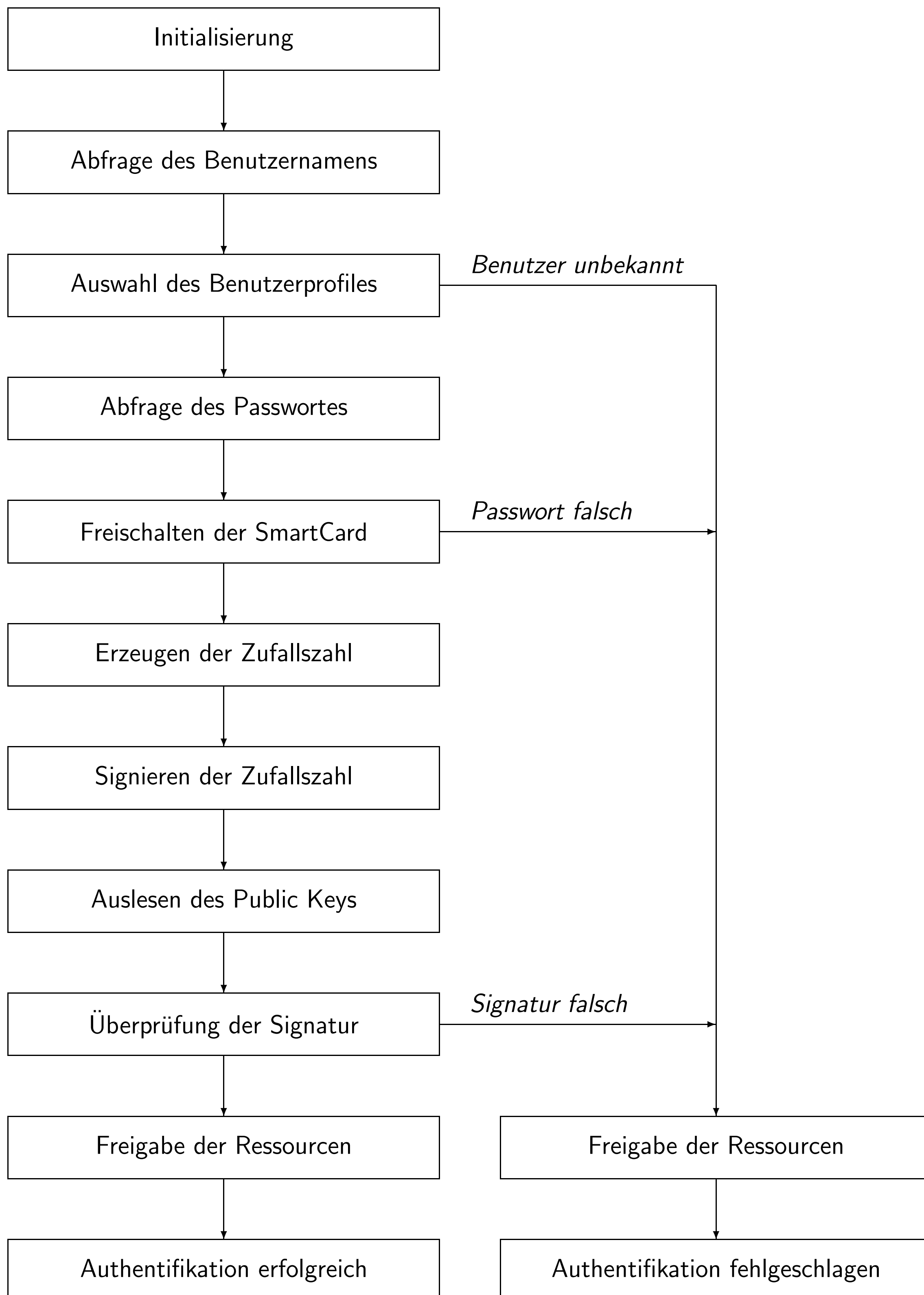

Abbildung 3.6: Ablauf eines Benutzerlogins

Initialisierung

Als erstes werden die dem Modul übergebenen Parameter analysiert. Der PAM Standard sieht vor, dass für jeden nicht unterstützten Wert eine Meldung mittels *syslog* ausgegeben wird. Des weiteren werden die benötigten Datenstrukturen alloziert und die Verbindung mit dem SmartCard Reader bzw. der SmartCard hergestellt. Falls dabei (oder auch im Folgenden) ein System- oder Bibliotheksfehler auftritt, werden die Daten, wie unter dem Punkt *Freigabe der Ressourcen* beschrieben, freigegeben.

Abfrage des Benutzernamens

Die Ermittlung des Benutzernamens erfolgt indirekt mit einer durch das PAM System bereitgestellten Funktion. Dadurch ist das Modul von der jeweiligen Benutzeroberfläche unabhängig und funktioniert sowohl mit Konsolen, als auch mit X11 Programmen. Der Funktion wird nur ein auszugebender String, der die Eingabeaufforderung enthält, übergeben. Darstellung der Meldung und Entgegennahme der Antwort sind Aufgabe der Applikation. Falls der Benutzername dem PAM-System schon bekannt ist, wird dieser verwendet und auf die Ausgabe einer Eingabeaufforderung verzichtet.

Auswahl des Benutzerprofiles

Alle auf der SmartCard vorhandenen Profile werden ausgelesen und mit dem Benutzernamen verglichen. Falls die Suche erfolgreich war, wird das entsprechende Profil selektiert, ansonsten wird dieser Umstand mittels *syslog* protokolliert und der Anmeldevorgang mit einem Fehler abgebrochen.

Abfrage des Passwortes

Bei der Abfrage des Passwortes wird gleich verfahren, wie bei der Ermittlung des Benutzernamens. Auch hier ist die eigentliche Abfrage vom Modul unabhängig und muss vom darüberliegenden Programm bereitgestellt werden. Anschliessend wird das Passwort noch in einer globalen Variablen des PAM Systems abgelegt, die auch weiteren Modulen zugänglich ist. Der Sinn dahinter ist, dass ein folgendes Modul im PAM Stapel das bereits ermittelte Passwort nicht erneut vom Benutzer abfragen muss. Programmen oder fremden Modulen eines anderen PAM Stapels bleibt der Zugriff verwehrt.

Freischalten der SmartCard

Damit die für das Signieren von Daten zuständigen Funktionen der SmartCard verwendet werden können, muss sich der zum selektierten Profil gehörende Benutzer bei der Karte anmelden. Dies erfolgt durch das Übermitteln des abgefragten Passwortes. Schlägt die Anmeldung fehl, wird dies mittels *syslog* protokolliert und der Anmeldevorgang mit einem Fehler abgebrochen.

Erzeugung der Zufallszahl

Die für die Authentifikation mit der SmartCard verwendete Zufallszahl sollte etwa gleich lang sein, wie die eingesetzten RSA Schlüssel, muss aber kleiner als der Modul sein. Aufgrund der von uns verwendeten 1024 Bit Schlüssel wählten wir 1023 Bit lange Zahlen, die daduch ausreichend gross, aber mit Sicherheit kleiner als der Modul sind. Eine so grosse Zufallszahl mit genügend *echter* Entropie zu erzeugen, ist nicht ganz trivial. Das Modul erstellt diese darum nicht selbst, sondern liest sie aus dem durch Linux bereitgestellten Random Device aus. Dieses greift für das Erstellen der Zufallszahlen auf vorhandene Systemdaten der Hardware zurück und nutzt Abweichungen in der Tippgeschwindigkeit sowie Mausbewegung des Benutzers aus. Die so erzeugte Folge von Zufallszahlen weist eine für unsere Zwecke genügend hohe Entropie auf.

Signieren der Zufallszahl

Die Signatur-Funktion (*Internal Auth*) der SmartCard macht nichts anderes, als die ihr zugesandten Daten mit dem gespeicherten *RSA Private Key* zu verschlüsseln. Anschliessend kann das Resultat mit einem *Get Response* Aufruf zurückgelesen werden.

Auslesen des Public Keys

Eine wichtige Voraussetzung für das korrekte Arbeiten des Anmeldevorganges ist das richtige Zuordnen der *Public Keys* zu den einzelnen Benutzern. Idealerweise würde dies durch Abfragen eines LDAP-Servers oder einer Zertifizierungsstelle erfolgen. Dies hätte jedoch den Rahmen dieser Arbeit gesprengt und ist für zukünftige Erweiterungen vorgesehen. In der jetzigen Situation wird beim Erstellen einer SmartCard der *Public Key* eines Benutzers zusammen mit seinem Loginnamen in einer zentralen Konfigurationsdatei (`/etc/pubkey`) abgelegt und beim Anmeldevorgang von dort wieder ausgelesen. Die Datei ist textbasiert und besitzt pro Zeile einen Login-Eintrag mit folgendem Aufbau:
`Kartenbesitzer:Profilname:Public Exponent:Modul`
Der Public Exponent und das Modul sind in textform als Hex-String (`A37F01`...) abgelegt.

Überprüfung der Signatur

Die von der SmartCard verschlüsselte Zufallszahl wird mit Hilfe des *Public Key* des Benutzers entschlüsselt und das Resultat mit der ursprünglichen Zahl verglichen. Stimmen die beiden Zahlen überein, war die Authentifikation des Benutzers und somit der ganze Anmeldevorgang erfolgreich. Ansonsten wird der Misserfolg mittels *syslog* protokolliert und der Anmeldevorgang mit einem Fehler abgebrochen

Freigabe der Ressourcen

Während des Anmeldevorganges werden diverse geheime Daten wie z.B. das Passwort des Benutzers im Speicher abgelegt. Bevor das Modul den Anmeldevorgang abbricht, gilt es, diese Daten zu löschen d.h. mit Nullen zu überschreiben. Dies sowohl am Ende des Anmeldevorganges als auch bei vorzeitigem Abbruch aufgrund eines Fehlers.

3.4.2 Passwortänderung

Die zweite Aufgabe des Modules ist es, das Passwort des Benutzers auf der SmartCard zu ändern. Wie bereits mehrmals erwähnt, wird dabei die dafür zuständige Funktion zweimal hintereinander aufgerufen. Beim ersten Mal muss nur geprüft werden, ob ein Ändern überhaupt möglich ist. Bei diesem Modul bedeutet das konkret, festzustellen, ob Verbindung zum Reader besteht und eine SmarCard eingelegt ist. Die Änderung darf erst beim erneuten Aufruf geschehen. Die Unterscheidung der beiden Aufrufe erfolgt durch ein, der Funktion übergebenes Flag, bei dem beim ersten mal der Wert `PAM_PRELIM_CHECK` und beim zweiten mal `PAM_UPDATE_AUTHTOK` gesetzt ist. Der genaue Ablauf ist Abbildung 3.7 zu entnehmen. Die einzelnen Schritte werden im Folgenden erläutert.

Initialisierung

Als erstes werden die dem Modul übergebenen Parameter analysiert. Der PAM Standard sieht vor, dass für jeden nicht unterstützten Wert eine Meldung mittels *syslog* ausgegeben wird. Des weiteren werden die benötigten Datenstrukturen alloziert und die Verbindung mit dem SmartCard Reader bzw. der SmartCard hergestellt. Falls dabei (oder auch im Folgenden) ein System- oder Bibliotheksfehler auftritt, werden die Daten, wie unter dem Punkt *Freigabe der Ressourcen* beschrieben, freigegeben.

Abfrage des Benutzernamens

Die Ermittlung des Benutzernamens erfolgt indirekt mit einer durch das PAM System bereitgestellten Funktion. Dadurch ist das Modul von der jeweiligen Benutzeroberfläche unabhängig und funktioniert sowohl mit Konsolen als auch mit X11 Programmen. Der Funktion wird nur ein auszugebender String, der die Eingabeaufforderung enthält, übergeben. Darstellung der Meldung und Entgegennahme der Antwort sind Aufgabe der Applikation. Falls der Benutzername dem PAM-System schon bekannt ist, wird dieser verwendet und auf die Ausgabe einer Eingabeaufforderung verzichtet.

Auswahl des Benutzerprofiles

Alle auf der SmartCard vorhandenen Profile werden ausgelesen und mit dem Benutzernamen verglichen. Falls die Suche erfolgreich war, wird das entsprechende Profil selektiert, ansonsten wird dieser Umstand mittels *syslog* protokolliert und der Anmeldevorgang mit einem Fehler abgebrochen.

Abfrage des alten Passwortes

Grundsätzlich wird hier gleich verfahren wie bei der Ermittlung des Benutzernamens. Bevor aber beim Benutzer nachgefragt wird, überprüft das Modul, ob ihm der Parameter `use_first_pass` übergeben wurde. Trifft dies zu, wird das alte Passwort aus der globalen PAM Variable `PAM_OLDAUTHTOK` ausgelesen. Ist diese nicht gesetzt, bricht das Modul mit einem Fehler ab, ohne beim Benutzer nachzufragen. Wurde der Parameter nicht übergeben, wird immer beim Benutzer nachgefragt.

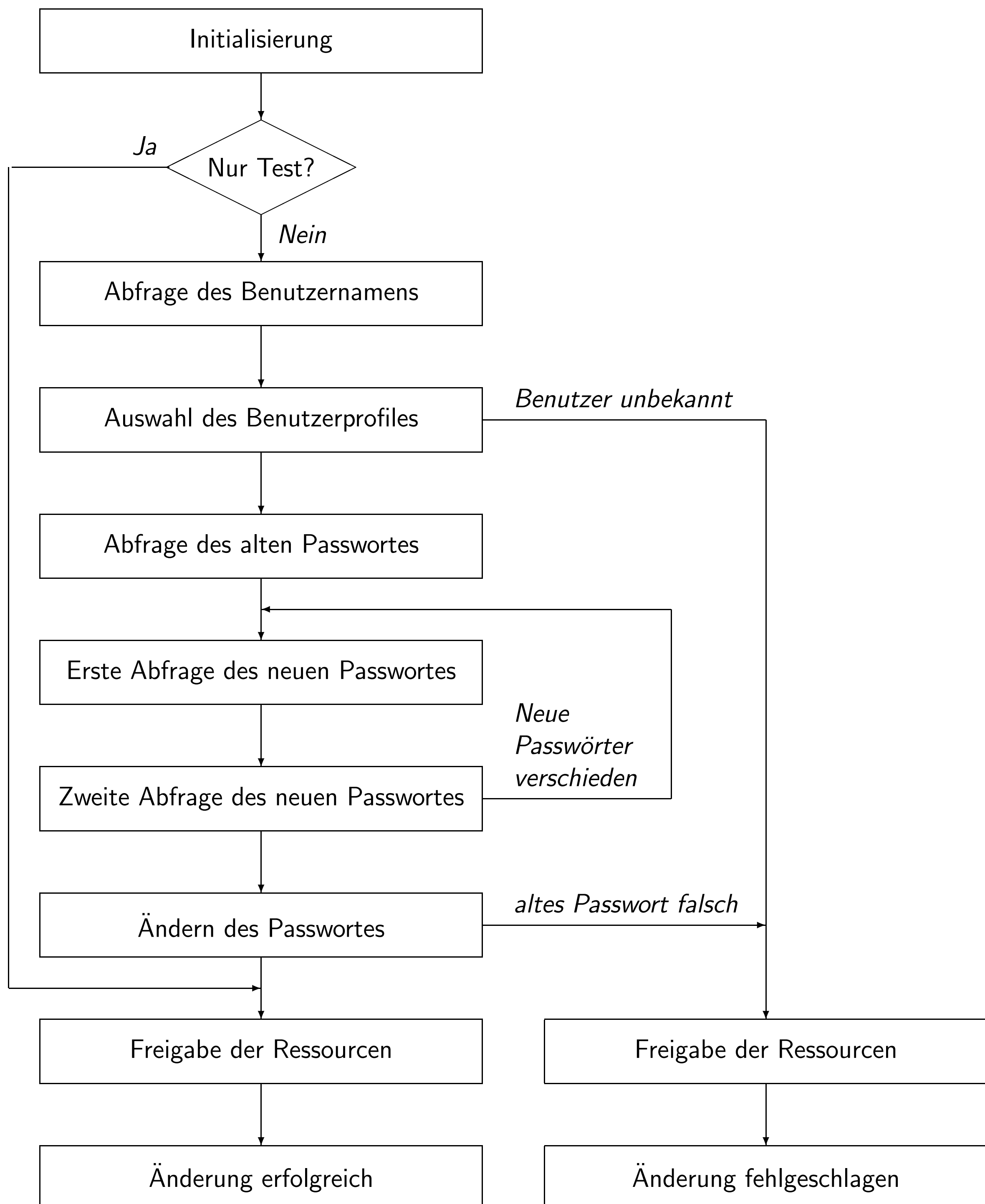

Abbildung 3.7: Ablauf einer Passwortänderung

Erste Abfrage des neuen Passwortes

Der einzige Unterschied zur Abfrage des alten Passwortes ist, dass der zu überprüfende Parameter `use_authtok` heisst und gegebenenfalls die globale PAM Variable `PAM_AUTHTOK` abgefragt wird.

Zweite Abfrage des neuen Passwortes

Nachdem das Passwort, wie im letzten Schritt beschrieben, erneut abgefragt wurde, wird überprüft, ob beide identisch sind. Ist dies nicht der Fall werden sie erneut abgefragt.

Ändern des Passwortes

Das eigentliche Ändern des Passwortes erfolgt mittels des *Change CHV* SmartCard Befehles. Falls dieser fehlschlägt, wird mit `syslog` eine entsprechende Meldung ausgegeben und die Verarbeitung mit einem Fehler abgebrochen.

Freigabe der Resourcen

Während der Passwortänderung werden diverse geheime Daten wie z.B. das Passwort des Benutzers im Speicher abgelegt. Bevor das Modul den Anmeldevorgang abbricht, gilt es diese Daten zu löschen, d.h. mit Nullen zu überschreiben. Dies sowohl am Ende des Anmeldevorganges als auch bei vorzeitigem Abbruch aufgrund eines Fehlers.

3.5 Entstandene Tools

Damit ein funktionstüchtiges System aufgebaut werden kann, braucht es neben dem PAM-Modul auch noch einige Programme, um die notwendigen SmartCards zu erstellen. Während unserer Projektarbeit erstellten wir die vier Tools, die im Folgenden kurz erklärt werden. Nach der Installation unseres Paketes steht die Dokumentation dieser Tools auch als Man-Pages zur Verfügung.

3.5.1 Programm udat

Dieses Programm ist für den normalen Anwender im System gedacht. Er kann damit seine persönlichen Daten anzeigen und ändern. Ebenfalls ist es möglich, den persönlichen PIN auf der Karte neu zu setzen. Der Aufruf erfolgt durch

```
udat Funktion [--Option=...]
```

Dabei kann die Funktion eine der folgenden sein:

show zeigt die aktuell auf der Karte gespeicherten Benutzerdaten an.

change ändert die Benutzerdaten auf der Karte.

pwd Mit diesem Befehl kann der PIN auf der Karte geändert werden. Dabei ist zu beachten, dass der Benutzername angegeben werden muss.

Folgende Optionen sind verfügbar:

--reader=Readernummer Damit kann optional ein anderer Reader angegeben werden.

--name=Benutzername Wenn die Funktion `pwd` verwendet wird, muss mit dieser Option der gewünschte Benutzername angegeben werden.

3.5.2 Programm makecard

Mit diesem Programm kann eine neue SmartCard für unser System erstellt werden. Dabei ist zu beachten, dass die Karte leer sein muss. Das heisst, dass die von uns verwendeten Dateien noch nicht vorhanden sein dürfen. Der Aufruf ist ähnlich wie bei `udat`:

```
makecard --name=Benutzername [--Option=...]
```

Da dieses Programm nur für die Erstellung des Hauptbenutzerprofils zuständig ist, entfällt die Wahl der Funktion. Es muss jedoch der Name des Hauptbenutzers angegeben werden.

Folgende Optionen sind hier anwendbar:

--name=Benutzername Der Name des Hauptbenutzerprofils muss in jedem Fall angegeben werden.

--reader=Readernummer Damit kann optional ein anderer Reader angegeben werden.

--aut0=PIN des Aut0 Hier wird der aktuell auf der Karte gespeicherte PIN des Aut0-Profils angegeben. Dieser PIN entspricht dem Kartenmanager-Passwort. Dieses muss hier in hexadezimaler Form angeben werden. Falls diese Option weggelassen wird, wird der Standard-PIN (*shipment code*) der jeweiligen Karte verwendet. Mit diesem Programm kann das Kartenmanager-Passwort nicht verändert werden. Dazu steht das Tool `managecard` zu Verfügung.

--apin=Administrator PIN Hier wird der Administrator-PIN angegeben. Damit kann zu einem späteren Zeitpunkt eine geblockte Karte wieder entsperrt werden. Falls diese Option nicht angegeben wird, wird der Standard-Wert '00000000' verwendet.

--pin=Benutzer PIN Hiermit wird der Benutzer PIN bestimmt. Falls diese Option nicht angegeben wird, wird der Standard-Wert '00000000' verwendet.

3.5.3 Programm managecard

Zum Erstellen weiterer Benutzerprofile ist dieses Programm zuständig. Zusätzlich kann man eine Liste der vorhandenen Profile ausgeben, ein bestimmtes Profil löschen, eine gesperrte Karte entblocken oder das Kartenmanager-Passwort ändern. Hier ist der Aufruf der folgende:

```
managecard Funktion [--Option=...]
```

Hier die möglichen Funktionen:

add wird zum Hinzufügen eines neuen Benutzerprofiles verwendet. Dabei muss der Benutzername zwingend angegeben werden. Wie bereits erwähnt, ist es bei *Cryptoflex* Karten nicht möglich, mehrere Profile auf einer Karte zu erstellen.

rem wird zum Löschen eines Benutzerprofiles verwendet. Auch hier muss der Name des zu löschenden Profils angegeben werden. Da das Hauptprofil nicht gelöscht werden kann, ist diese Funktion bei *Cryptoflex* Karten nicht verwendbar.

show gibt eine Liste der vorhandenen Benutzerprofile aus.

unblock Damit kann eine gesperrte Karte wieder entsperrt werden. Dabei muss der Profilname, der Administrator-PIN und der neue Benutzer-PIN angegeben werden.

changeaut0 Mit dieser Funktion kann des Kartenmanager-Passwort geändert werden. Diese Funktion steht bei *Cryptoflex* Karten nicht zu Verfügung. Es muss das alte und das neue Kartenmanager-Passwort angegeben werden.

Die folgenden Optionen sind verfügbar:

--name=Benutzername Hiermit wird der Benutzerprofilname angegeben.

--aut0=PIN des Aut0 Hier wird der aktuell auf der Karte gespeicherte PIN des Aut0-Profils angegeben. Dieser PIN entspricht dem Kartenmanager-Passwort. Dieses muss hier in hexadezimaler Form eingeben werden. Falls diese Option weggelassen wird, wird der Standard-PIN (*shipment code*) der jeweiligen Karte verwendet.

--reader=Readernummer Damit kann optional ein anderer Reader angegeben werden.

--apin=Administrator PIN Hier wird der Administrator-PIN angegeben. Falls diese Option nicht angegeben wird, wird der Standard-Wert '00000000' verwendet.

--pin=Benutzer PIN Hiermit wird der Benutzer PIN bestimmt. Falls diese Option nicht angegeben wird, wird der Standard-Wert '00000000' verwendet.

--naut0=Neuer PIN des Aut0 Damit wird, wenn die Funktion `changeaut0` aufgerufen wurde, der neue Kartenmanager-PIN angegeben.

3.5.4 Programm cleancard

Mit diesem Programm kann eine durch uns erstellte Karte wieder gelöscht werden.

```
cleancard [--Option=...]
```

Die folgenden Optionen sind verfügbar:

--aut0=PIN des Aut0 Hier wird der aktuell auf der Karte gespeicherte PIN des Aut0-Profils angegeben. Dieser PIN entspricht dem Kartenmanager-Passwort. Dieses muss hier in hexadezimaler Form eingeben werden. Falls diese Option weggelassen wird, wird der Standard-PIN (*shipment code*) der jeweiligen Karte verwendet.

--reader=Readernummer Damit kann optional ein anderer Reader angegeben werden.

Kapitel 4

Installation

4.1 Installation of the Smartcard Reader

If not already done, first of all you have to download the pc/sc middleware and a dedicated driver for your smartcard reader. You can find the latest versions under the following links.

- PC/SC Middleware: http://www.linuxnet.com/middle.html
- Reader drivers: http://www.linuxnet.com/drivers.html

For the further steps you are supposed to be logged in as root!

4.1.1 Installing PC/SC

Unpack the archive and install it into `/usr/local/pcsc/`. Replace pcsc-lite-X.Y.Z. tar.gz and pcsc-lite-X.Y.Z with the real name of the current package for example pcsc-lite-0.9.1.tar.gz and pcsc-lite-0.9.1 ;-).

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

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

4.1.2 Installing the Reader Driver

Create the `drivers` directory under `/usr/local/pcsc` if it does not already exist. Then copy the driver archive to there and unpack it. If the driver was created with autoconf call the `configure` script before compiling it. Afterwards you can delete the package.

```
bash# mkdir /usr/local/pcscd/drivers
bash# cp the-driver-X.Y.tar.gz /usr/local/pcscd/drivers
bash# cd /usr/local/pcscd/drivers
bash# tar -xvzf the-driver-X.Y.tar.gz
bash# cd the-driver-X.Y
bash# ./configure
bash# make
bash# rm ../the-driver-X.Y.tar.gz
```

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

4.1.3 Configure the Driver

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 library>
CHANNELID        <device id>
```

The exact parameter values depend on the used reader and driver and ought to be mentioned in the dedicated README file. The `LIBPATH` parameter defines the full path to the actually driver module. If it isn't obvious, try to locate the module with the `find` command.

```
bash# cd /usr/local/pcscd/drivers/the-driver-X.Y
bash# find . -name="*.so"
```

See section 4.6.1 (Example of a reader.conf file) for a possible configuration file. After all modifications have been made, try to start the daemon.

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

If the daemon fails while loading a reader driver, check if its configuration item in `reader.conf` is set correctly and the reader is properly connected to the defined port. There's 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!

4.1.4 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 drivers work immediately after startup. First we create a symbolic link in `/usr/sbin/` to the daemon. Then we create a init-script in `/etc/init.d/` for the daemon and install it with `insserv`. There's an example of the script in the 4.6.2 (Example of a pcsc script) section.

```
bash# ln -s /usr/local/pcsc/sbin/pcscd /usr/sbin/pcscd
bash# cp pcsc /etc/init.d/
bash# insserv /etc/init.d/pcsc
```

Try to start the daemon by using the init-script. 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
```

4.2 Installation of the PAM Module and the Smart-card Tools

This step shows you how to install the *smartcard login* package which contains the PAM module and some additional smartcard tools. You can download it from http://www.strongsec.com/smartcards/. If you didn't install the pcsc middleware under `/usr/local/pcsc/` you have to specify the path to it with the `--with-pcsclite-dir` parameter.

```
bash# tar -xvzf smartcard-login.X.Y.tag.gz
bash# cd smartcard-login.X.Y
bash# ./configure --with-pcsclite-dir=/usr/local/pcsc
bash# make
bash# make install
```

Now there should be a file called `pam_smartcard.so` in your `/lib/security` directory.

4.3 Creating the User's Login Smartcard

The *smartcard login* package contains three tools for creating and managing the users' smartcards. The following text is a step by step description on how to create such a smartcard. For further informations about the mentioned programs please read the dedicated man-pages. The parameter `--reader` allows you to specify which reader to use. Default is the first reader mentioned in `/etc/reader.conf` (No. 0). To use the second reader from `/etc/reader.conf` set the `--reader` parameter to one, for the third to two and so on.

4.3.1 Creating a User Account

If the user doesn't already exist on the Linux system you have to create it by using `useradd` or an other tool like `yast` or `linuxconf`.

```
bash# useradd <username> -d
```

4.3.2 Creating a minimal Smartcard

After cleaning the card we create the main profile for the user. A rsa key pair is created automatically either by the smartcard itself or by software and downloaded to the card. The public key of the rsa key pair is also stored in the smartcard login configuration file `/etc/pubkey`.

```
bash# cleancard --reader=0
bash# makecard --reader=0 --name=<username> --pin=<password>
```

4.3.3 Adding a Profile

ISO-7816 compatible smartcards allow you to use them for more than one login account. For example to allow a user to login as root too, you can add an additional root profile.

```
bash# managecard add --name=root --pin=<password>
```

4.4 Configuring the Smartcard Login

To enable the smartcard login you have to change at least the `/etc/pam.d/login` configuration file. We recommend you to change the `/etc/pam.d/passwd` and `/etc/pam.d/su` files for the `passwd` and `su` command, too. 4.5 (If something goes wrong) describes how you can repair your system if the installation fails and it is not possible to login anymore.

4.4.1 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
```

in the `/etc/pam.d/su` file. The `reader` parameter defines which reader to use (See section 4.3 (Creating the User's Smartcards for Login) for a description of the enumerating). Now, you can test the new configuration by calling `su` as a normal user.

4.4.2 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
```

into

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

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.

4.4.3 Configuring the passwd Command

The password used by the smartcard is often also called a PIN what stands for *P*in *I*dentification *N*umber. However most smartcards allow you to use any printable character not only digits and you should make use of it. Actually, the `udat` command allows you to change the password of the smartcard. However, most user are used to the `passwd` command. 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 nullok use_first_pass
```

with

```
password required /lib/security/pam_smartcard.so
```

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. Last thing to do is disabling the shadow password.

```
bash# passwd -l <username>
```

4.4.4 Additional Possibilities with the Password

Some programs aren't able to use the pam smartcard module for example because they don't support the pam system. For this reason, disabling the shadow password isn't always the right solution. Since the pam system is very flexible, it's possible to use both, the pam module and the standard unix shadow module.

If you wish to use different passwords for your smartcard and the shadow system just leave the `/etc/pam.d/passwd` file untouched and use `udat` to change the password of your smartcard and `passwd` for the shadow system.

Another possibility is to use the same password for the smartcard and the shadow system. Thereby, you have to make sure that you always change both passwords together. You can realize this by inserting the line

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

after the line

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

in `/etc/pam.d/passwd`.

The `passwd` command now changes both passwords. *Make no more use of the `udat` command as it only changes the the password of the smartcard!* Before editing the `/etc/pam.d/passwd` file you have to make sure that both passwords are equal. If the passwords aren't equal every call of `passwd` will fail as the old password is only asked once and different for the smartcard and the shadow system. The best way to clean such a situation is to set the password of the smartcard to the same value as the shadow password using the `udat` command.

4.5 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 seems to 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 ask 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 <device> -o remount / -t <fs-type>
```

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.

4.6 Example configurations

The following examples were tested under a SuSE Linux 7.1 i386 system.

4.6.1 Example of a `reader.conf` file

A Schlumberger Reflex 62 reader on the second and a Towikoto reader on the first serial port.

```
# Configuration file for pcsc-lite

FRIENDLYNAME     "Schlumberger Reflex 62"
DEVICENAME       GEN_SMART_RDR
LIBPATH          /usr/local/pcsc/drivers/slb_rf60/libslb_rf60.so
CHANNELID        0x0102F8

FRIENDLYNAME    "Towitoko Chipdrive Micro"
DEVICENAME      TOWITOKO_CHIPDRIVE_MICRO
LIBPATH         /usr/local/pcsc/drivers/towitoko-2.0.3/libtowitoko.so
CHANNELID       0x000001

# End of file
```

4.6.2 Example of a `pcsc` script

```
#! /bin/sh
#
# 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
```

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

```
#%PAM-1.0
auth     requisite     /lib/security/pam_smartcard.so  nullok reader=0
auth     required      /lib/security/pam_securetty.so
auth     required      /lib/security/pam_nologin.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_unix.so       nullok
session  required      /lib/security/pam_unix.so       none
session  required      /lib/security/pam_limits.so
```

4.6.4 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  nullok reader=0
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       none
```

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

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

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

[cybe]

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

[crypt]

Schlumberger Cryptoflex Programmer's Guide
http://www.cyberflex.com/Support/CryptoflexPG.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

Index

A

amorphe Dateistruktur . . . 8
Antwort-APDU . . . 13
APDU . . . 12, 24, 27
- Antwort-APDU . . . 13
- Kommando-APDU . . . 12

Application Protocol Data Unit . . . 12

B

Befehle . . . 14, 18
Benutzerprofil . . . 27
Bibliotheken . . . 22
- libcardtools . . . 22
- librsa . . . 22

binäre Dateistruktur . . . 8

C

Cardlet . . . 14, 16, 24
CHV . . . 14, 16, 17, 29
Class . . . 12
Class-Byte . . . 12
Cryptoflex . . . 24
Cyberflex . . . 24
cyclic Dateistruktur . . . 9

D

Dateinamen . . . 10
Dateistruktur . . . 8
- amorphe Dateistruktur . . . 8
- binäre Dateistruktur . . . 8
- cyclic Dateistruktur . . . 9
- linear fixed Dateistruktur . . . 8
- linear variable Dateistruktur . . . 9
- transparente Dateistruktur . . . 8

Dedicated File . . . 8
DF . . . 8

E

EF . . . 8
- Internal EF . . . 8
- Working EF . . . 8

Elementary File . . . 8

F

FID . . . 10
File Identifier . . . 10

G

Gemplus GCR 410 . . . 25

I

Instance-Container . . . 17
Instance-File . . . 17
Internal EF . . . 8

J

JavaCard . . . 7, 13, 24

K

Kartenleser . . . 24, 25
Kommando-APDU . . . 12
Kontaklose Karte . . . 7

L

Lc-Feld . . . 12
Le-Feld . . . 12
linear fixed Dateistruktur . . . 8
linear variable Dateistruktur . . . 9
Linux-PAM . . . *siehe* PAM

M

Master File . . . 8
MF . . . 8
Mikroprozessorkarte . . . 7

P

PAM . . . 3
- Applikation . . . 6
- arguments . . . 5
- control-flag . . . 5
- Konfigurationsdatei . . . 3
- Modul . . . 5, 30
- module-path . . . 5
- module-type . . . 4
- service-name . . . 4

PC/SC Middleware 26
Private Key 20, 29, 33
Profil 27
Projektaufbau 22
Public Key 20, 29, 33

R

Reader 25
Records 8
Reflex 60 25
Returncode 13
Ringspeicher 9
Root-Verzeichnis 8
RSA 20
  Entschlüsselung 21
  Schlüsselerzeugung 20
  Verschlüsselung 20

S

service-name 6
Speicherkarte 7
Status-Wörter 13

T

Towitoko Chipdrive 26
Trailer 13
transparente Dateistruktur 8

W

Working EF 8

Z

Zugriffsbedingungen . . . 10, 16, 18, 24, 29