Anbindung einer Personalisierungsstraße an das FlexiTRUST-System

Betreuer: Daniel Walther, Markus Ruppert

Ausgearbeitet von Jürgen Henge-Ernst und Sebastian Linke
# Inhaltsverzeichnis

1 Einleitung  

2 Umfeldanalyse  
   2.1 PKI mit FlexiTRUST  
      2.1.1 Prozeßablauf  
      2.1.2 Formularaufbau  
   2.2 Personalisierungsstraße  
      2.2.1 Peripherie  
      2.2.2 Chipkarten  
      2.2.3 Steuerungs-Software  
   2.3 Fertigungsumfeld  
      2.3.1 Produktionsspezifische Aspekte der Fertigung  
      2.3.2 Sicherheitsspezifische Aspekte der Fertigung  

3 Konkretisierung der Aufgabenstellung  

4 Szenarien für die Anbindung an FlexiTRUST  
   4.1 Einstufige Personalisierung der Chipkarte  
      4.1.1 Einstufige Personalisierung durch FlexiTRUST  
      4.1.2 Einstufige Personalisierung durch den Listener  
   4.2 Mehrstufige Personalisierung der Karten  
      4.2.1 Szenario I: Personalisierung in drei Schritten  
      4.2.2 Szenario II: Getrennte optische und elektronische Personalisierung  
      4.2.3 Szenario III: Aktivierung durch den Endbenutzer  

5 Konzept für die Anbindung der Personalisierungsstraße an FlexiTRUST  
   5.1 Caller  
   5.2 Listener  
   5.3 Kommunikationsprotokoll Caller/Listener  
      5.3.1 Aufruf des Listeners durch den Caller  
      5.3.2 Rückmeldung des Listeners  
   5.4 Kommunikationsprotokoll Listener/FlexiTRUST  
      5.4.1 Anbindung an die RA  
      5.4.2 Abholen der Zertifikate  

6 Implementierung des Callers  
   6.1 Badgemail als Caller  
      6.1.1 Parametrisierung der bmchip32.dll  
      6.1.2 Kommunikationsprotokoll  
   6.2 PREPERSO als Caller  

7 Implementierung des Listeners  
   7.1 Aufbau des Listeners  
   7.2 Konfiguration des Listeners  
      7.2.1 Die [input] Sektion  

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Einleitung</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Umfeldanalyse</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2.1 PKI mit FlexiTRUST</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2.1.1 Prozeßablauf</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2.1.2 Formularaufbau</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.2 Personalisierungsstraße</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2.2.1 Peripherie</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2.2.2 Chipkarten</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2.2.3 Steuerungs-Software</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2.3 Fertigungsumfeld</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2.3.1 Produktionsspezifische Aspekte der Fertigung</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2.3.2 Sicherheitsspezifische Aspekte der Fertigung</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Konkretisierung der Aufgabenstellung</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Szenarien für die Anbindung an FlexiTRUST</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>4.1 Einstufige Personalisierung der Chipkarte</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>4.1.1 Einstufige Personalisierung durch FlexiTRUST</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>4.1.2 Einstufige Personalisierung durch den Listener</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>4.2 Mehrstufige Personalisierung der Karten</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>4.2.1 Szenario I: Personalisierung in drei Schritten</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>4.2.2 Szenario II: Getrennte optische und elektronische Personalisierung</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>4.2.3 Szenario III: Aktivierung durch den Endbenutzer</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>Konzept für die Anbindung der Personalisierungsstraße an FlexiTRUST</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>5.1 Caller</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>5.2 Listener</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>5.3 Kommunikationsprotokoll Caller/Listener</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>5.3.1 Aufruf des Listeners durch den Caller</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>5.3.2 Rückmeldung des Listeners</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>5.4 Kommunikationsprotokoll Listener/FlexiTRUST</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>5.4.1 Anbindung an die RA</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>5.4.2 Abholen der Zertifikate</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>Implementierung des Callers</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>6.1 Badgemail als Caller</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>6.1.1 Parametrisierung der bmchip32.dll</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>6.1.2 Kommunikationsprotokoll</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>6.2 PREPERSO als Caller</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>Implementierung des Listeners</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>7.1 Aufbau des Listeners</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>7.2 Konfiguration des Listeners</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>7.2.1 Die [input] Sektion</td>
<td>36</td>
</tr>
</tbody>
</table>
7.2.2 Die [output] Sektion ........................................ 36
7.2.3 Die [smartcard] Sektion ................................. 37
7.2.4 Die [ra] Sektion ........................................... 40
7.2.5 Die [database] Sektion .................................... 40

8 Ausblick .................................................................. 43
A Beispiele für die Konfigurationsmöglichkeiten des Listeners 44
   A.1 Erzeugen von zwei Schlüsselpaaren mit PREPERSO .... 44
   A.2 Erstellen eines RA-Antrages mit Hilfe von BadgeMaker ... 44
   A.3 Speichern zweier Zertifikate ................................ 45
   A.4 Komplette Personalisierung mit BadgeMaker .......... 45

B Quellcode .................................................................. 47
   B.1 bmchip32.h ....................................................... 47

C Literaturverzeichnis .................................................. 50
# Abbildungsverzeichnis

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Titel</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gewünschter kombinierter Produktionsprozeß</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Erfolgreiche Verarbeitung eines Zertifikats-Antrags</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Erfolgreicher Ablauf eines Personalisierungsprozesses</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Zusammenhang der einzelnen Karten-Schnittstellen</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>Interner Ablauf der elektronischen Personalisierung</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Konkretisierung der Aufgabenstellung</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>Einstufige Personalisierung durch FlexiTRUST</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>Einstufige Personalisierung durch den Listener</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>Szenario I - Vorpersonalisierung</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>Szenario I - Zertifizierung</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>Szenario I - Personalisierung</td>
<td>21</td>
</tr>
<tr>
<td>12</td>
<td>Szenario II - Optische Personalisierung</td>
<td>22</td>
</tr>
<tr>
<td>13</td>
<td>Szenario II - Elektronische Personalisierung</td>
<td>23</td>
</tr>
<tr>
<td>14</td>
<td>Szenario III - Personalisierung</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>Szenario III - Aktivierung der Karte</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>Interner Prozeßablauf des Listeners</td>
<td>27</td>
</tr>
<tr>
<td>17</td>
<td>Informationsfluß innerhalb des Listeners (I)</td>
<td>34</td>
</tr>
<tr>
<td>18</td>
<td>Informationsfluß innerhalb des Listeners (II)</td>
<td>35</td>
</tr>
</tbody>
</table>
Tabellenverzeichnis

1 Parameter des Programms PREPERSO ................................ 32
2 Aufbau der [input] Sektion ........................................... 36
3 Bedeutung der [input] Variablen .................................... 36
4 Aufbau der [output] Sektion ........................................... 37
5 Bedeutung der [output] Variablen .................................... 37
6 Interne Listener-Variablen ............................................ 38
7 Aufbau der [smartcard] Sektion ...................................... 38
8 Bedeutung der [smartcard] Variablen ................................ 39
9 Aufbau der [ra] Sektion ................................................ 40
10 Bedeutung der [ra] Variablen ........................................ 40
11 Aufbau der [database] Sektion ..................................... 40
12 Bedeutung und Werte der [database] Variablen ................. 41
1 Einleitung

Um die Kommunikation innerhalb offener Netze absichern zu können, wird heutzutage verstärkt Public-Key-Kryptographie eingesetzt. Aufbau und Betrieb von Public-Key-Infrastrukturen (PKI) ermöglichen es einem Anwender u.a., sich gegenüber anderen zu authentifizieren, Daten zu verschlüsseln oder Nachrichten zu signieren. Für solche möglichen Anwendungsfälle werden oft Chipkarten zur Unterstützung von Softwarelösungen verwendet, da dadurch für den Anwender eine höhere Sicherheit gewährleistet werden kann. Eine personalisierte Chipkarte dient dem Anwender nicht nur als Ausweis, etwa indem die Karte bedruckt und Daten auf dem Chip gespeichert wurden, sondern sie stellt ihm auch Funktionen zur Nutzung der Public-Key-Infrastruktur zur Verfügung.

Mit Hilfe der Software FlexiTRUST, die von der wissenschaftlichen Arbeitsgruppe um Prof. Dr. Johannes Buchmann entwickelt wurde, kann ein Trustcenter für eine PKI aufgebaut werden. Auf dem Markt sind verschiedene sogenannte Personalisierungsstraßen verfügbar, die eine Personalisierung von Chipkarten in elektronischer (Chip) und optischer (Druck) Hinsicht ermöglichen.

2 Umfeldanalyse


Abbildung 1: Gewünschter kombinierter Produktionsprozeß

2.1 PKI mit FlexiTRUST

2.1 PKI mit FlexiTRUST

- Die *Registration Authority* (RA) nimmt die Anträge der Teilnehmer auf Zertifikate oder andere Dienstleistungen des Trustcenters entgegen, entscheidet ob den Anträgen stattgegeben werden kann und leitet diese gegebenenfalls zur Verarbeitung weiter. Für die Verarbeitung der Anträge werden sogenannte Formulare verwendet, die in Kapitel 2.1.2 näher betrachtet werden.

- Die *Certification Authority* (CA) stellt Zertifikate für Personen oder Instanzen aus. Dort ist der sicherheitskritische Bereich eines Trustcenters angesiedelt, denn hier werden die auszustellenden Zertifikate mit dem geheimen Schlüssel der CA signiert oder es können Schlüsselpaare für Teilnehmer erzeugt werden.

- Die *Infrastructure Services* (IS) sind für die Verwaltung der Zertifikate verantwortlich. Über die IS können Zertifikate für bestimmte Personen gesucht werden oder man kann überprüfen, ob ein Zertifikat noch gültig ist.

### 2.1.1 Prozeßablauf

Abbildung 2 zeigt in vereinfachter Form die Dienste, die innerhalb dieser Ausarbeitung in Anspruch genommen werden. Im ersten Schritt wird eine Anfrage an die RA gestellt, die daraufhin ein Antragsformular zurückliefernt. Dieses Formular wird dann vom Antragsteller ausgefüllt und zurück an die RA geschickt. Diese prüft dann, ob das Formular korrekt ausgefüllt wurde, teilt dieses dem Antragsteller mit und leitet das ausgefüllte Formular gegebenenfalls an die CA weiter. Die CA erstellt dann das Zertifikat, welches von der IS daraufhin in einem Verzeichnis bereitgestellt wird.

![Prozeßablauf Diagramm](image)

**Abbildung 2: Erfolgreiche Verarbeitung eines Zertifikats-Antrags**
2.1.2 Formularaufbau


Ein Antragsformular enthält Informationen über das Trustcenter, sowie personenbezogene Daten und den Public Key des Antragstellers. Um eine eindeutige Identifizierung des Antragstellers gewährleisten zu können, wird aus den personenbezogenen Daten der sogenannte Distinguished Name (dName oder DN) erzeugt. Der dName enthält mehrere Attribute, die so gewählt werden müssen, daß eine Eindeutigkeit garantiert werden kann. Diese geforderte Eindeutigkeit erfüllt auf der einen Seite den Zweck, daß eine fehlerfreie Kommunikation zwischen dem Inhaber des Zertifikats und Dritten stattfinden kann. Auf der anderen Seite wird dies auch für die Verwaltung des Trustcenters benötigt, bspw. wenn bei einer Zertifikatsverlängerung geprüft werden soll, ob der Zertifikatinhaber eine Verlängerungsgebühr entrichtet hat. Oft wird neben dem Namen des Antragstellers (CN-Attribut) auch seine Organisation (O), die Organisationsseinheit (OU) und das entsprechende Land (C) verwendet. Der dName könnte somit den folgenden Aufbau haben:

CN=Vorname Name, OU=Organisationseinheit, O=Organisation, C=Land

Im Falle einer Universität könnte OU dem Fachbereich, O dem Universitätsnamen und C dem Land, in dem sich die Universität befindet, entsprechen. Falls zwei "Max Mustermann" am selben Fachbereich eingeschrieben wären, würden beide aber den selben dName erhalten, bspw.:

CN=Max Mustermann, OU=Informatik, O=TUD, C=DE


Da in vielen Anwendungsfällen eindeutige Nummern wie Personal- oder Matrikelnummern vorhanden sind, erscheint der Einsatz des SER-Attributs universeller zu sein, da so auch eine Überprüfung auf mögliche Kollisionen obsolet ist. Im Kontext von Universitäten ergeben sich aber Probleme bei der Generierung der Daten wie die folgenden Beispiele zeigen:

- Da die Zertifikate öffentlich und somit für jeden einsehbar sind, ein Paar bestehend aus Matrikelnummer und zugehörigem Student aber nicht veröffentlicht werden darf (siehe auch [8]), kann die Matrikelnummer nicht direkt für das SER-Attribut verwendet werden.

- Auch die Verwendung von Hashwerten der Matrikelnummern ist problematisch, da - falls die Hashfunktion bekannt ist - eine vollständige Enumeration des sehr
kleinen Zeichenraumes durchführbar ist. Somit wäre es möglich, aus dem Hashwert auf die Matrikelnummer zu schließen, indem man alle denkbaren Matrikelnummern hasht und die Ergebnisse solange vergleicht, bis die richtige Nummer gefunden wurde. Des weiteren ist der Hashwert, der von Verfahren wie z.B. MD5 erzeugt wird, relativ lang im Gegensatz zur Matrikelnummer und somit für einen Menschen nicht einfach zu lesen, einzugeben oder zu vergleichen.


- Die Nutzung einer nur schwach kollisionsresistenten Hashfunktion, z.B. durch das Bilden der Quersumme der Matrikelnummer, stellt ebenfalls eine mögliche Lösung dar. Durch die Kollisionen wäre gewährleistet, daß ein Angreifer nicht exakt die dazugehörige Matrikelnummer bestimmen kann. Auf der anderen Seite muß die Hashfunktion so kollisionsresistent sein, daß immer noch eindeutige dNames erzeugt werden können. Das finden einer solchen Hashfunktion kann sich als schwierig erweisen.

Wie die vorherigen Beispiele zeigen, braucht man somit eine Funktion, die aus den vorhandenen Daten einen Wert generiert, der eindeutig ist und keinen Rückschluß auf die verwendeten Daten zuläßt, aber dennoch den Datensatz genau identifiziert. Eine einfache Lösung für dieses Problem scheint die Erweiterung der Hauptdatenbank um ein zusätzliches Feld zu sein, dessen Inhalt zufällig ist und in keinem direkten Zusammenhang zu den Matrikelnummern steht. Dafür bietet sich die Verwendung von Zufallszahlen oder Zählern an. Bei der Verwendung eines Zählers muß berücksichtigt werden, daß dieser nicht mit den Matrikelnummern korreliert, was bei der Erzeugung innerhalb einer nach Matrikelnummern sortierten Tabelle der Fall sein könnte.

Da sowohl die Verwendung des dnQualifier, als auch die des SER-Attributs problematisch sein kann, sollte in Abhängigkeit der Problemdaten, bzw. des spezifischen Umfeldes, situativ entschieden werden welcher Ansatz zu wählen ist. Im Rahmen dieses Projekts wird davon ausgegangen, daß die Eingangsdaten bereits so aufbereitet wurden, daß die fehlerfreie Erzeugung eindeutiger dNames möglich ist, da unter Umständen entsprechende Eingriffe in die Datenverwaltung und Organisation der betroffenen Institution nötig sind, die hier nicht berücksichtigt werden können. Ebenso wird nicht berücksichtigt wie z.B. Änderungen von Namen, bspw. durch Berichtigung einer Fehleingabe, Heirat eines Zertifikatsinhabers oder eine verspätete Ersatzzug von Umlauten, zu handhaben sind, da dies ein organisatorisches Problem darstellt, welches nicht zur primären Aufgabenstellung dieser Ausarbeitung gehört.

### 2.2 Personalisierungsstraße

Mit Hilfe von Personalisierungsstraßen werden personenspezifischen Daten auf die Karte gedruckt, bzw. im elektronischen Speicher der Chipkarte gespeichert. Da der Aufdruck der Karte für eine spätere visuelle Authentifizierung, bspw. durch Vorzeigen
Umfeldanalyse
der Karte, benutzt wird, nennt man das Bedrucken auch optische Personalisierung. Bei
dieser wird die Karten mit personenspezifischen Daten wie bspw. Name, Foto oder
Unterschrift bedruckt. Unter Personalisierung in elektronischer Hinsicht versteht man
das Speichern von Daten auf dem Magnetstreifen, dem Speicherchip oder dem Mikro-
Controller der Karte.
Eine Personalisierungsstraße besteht aus einem Drucker, der den Chip einer Karte be-
schreiben und die Karte bedrucken kann und einer Steuerungs-Software, die den Per-
sonalisierungsprozeß lenkt und den Drucker ansteuert.
Abbildung 3 zeigt einen erfolgreichen Personalisierungsprozeß. Im ersten Schritt wird
der Drucker durch die Steuerungs-Software veranlaßt, eine Chipkarte einzuziehen und
diese unter dem Kartenleser zu positionieren. Anschließend können Informationen
über die aktuelle Karte abgerufen werden. Danach wird von der Software eine Anfrage
an eine Datenbank gestellt, die den ausgewählten Datensatz zurückliefernt. Nach dem
erfolgreichen Beschreiben des Chips kann die Steuerungs-Software den verwendeten
Datensatz in der Datenbank aktualisieren, bspw. um Daten der Chipkarte zu speichern
oder um das erfolgreiche elektronische Personalisieren zu vermerken. Abschließend
wird die Karte bedruckt und ausgeworfen.
Auch wenn schon während der elektronischen Personalisierung eine Aktualisierung
der Datenbank durchgeführt werden kann, wird im Rahmen dieser Ausarbeitung da-
von ausgegangen, daß die Aktualisierung stets der letzte Schritt vor dem Bedrucken
der Karte ist.
In den folgenden Abschnitten werden die Merkmale und Eigenschaften der einzel-
nen Komponenten einer Personalisierungsstraße und ihres Umfelds erläutert und die
jeweiligen auftretenden Prozesse näher untersucht.

2.2.1 Peripherie
Für diese Studienarbeit wurde der Drucker Eltron P310, der von der Firma InterCard
zur Verfügung gestellt wurde, verwendet. Er ermöglicht es, den Chip einer Karte zu
beschreiben und die Karte einseitig zu bedrucken. Das Bedrucken der Karte findet
dabei stets nach dem Beschreiben des Chips statt. Dieser Drucker bietet dabei nicht
die Möglichkeit, fehlerhafte Karten automatisch auszusortieren.

2.2.1.1 Druckereinheit
Die Druckereinheit wird über die LPT-Schnittstelle angesprochen. Bei der Verwen-
dung des Druckers unter Windows muß der Drucker-Spooler auf "Druckaufträge di-
rekt an den Drucker leiten" gesetzt werden, da ansonsten die Karte nicht rechtzeitig
unter dem Kartenleser positioniert werden kann.

2.2.1.2 Chipkartenleser
Der zur Verfügung gestellte Drucker beinhaltete einen GemPlus Kartenleser vom
Typ GCI400 ECM, Firmware Version 1.0. Dieser Kartenleser kann in Verbindung mit
den verwendeten Treibern allerdings die von InterCard zur Verfügung gestellten Kar-
ten nicht ansprechen, so daß statt dessen ein Kartenleser vom Typ GCI410 ECM ver-
wendet werden mußte.
Im Rahmen dieses Projekts wird der Kartenleser, respektive die Chipkarte über eine
PKCS#11-Software-Schnittstelle angesteuert. Bei PKCS#11 handelt es sich um eine


Um eine Karte auch aus einer Java-Umgebung heraus ansprechen zu können, wurde zusätzlich der von Vlad Coroama entwickelte CDC-PKCS#11 Provider verwendet.
(siehe auch [3]).
Abbildung 4 zeigt wie die einzelnen APIs bei einem Programmaufruf aus C, bzw. aus Java zusammenhängen.

2.2.2 Chipkarten
Für dieses Projekt wurden von der Firma InterCard zur Verfügung gestellte Chipkarten vom Typ GemPlus GPK8000 verwendet.
Die Karte bietet zusätzlich zwei Benutzerebenen, denen ein unterschiedlicher Funktionsumfang zugewiesen ist. Man unterscheidet zwischen der Endanwender- und der Administrator-Ebene. Beide Bereiche sind durch eine PIN (Personal Identification Number) geschützt, da aber auf Administrator-Ebene die Endanwender-PIN überschreiben und freigeschaltet werden kann, wird die Administrator-PIN auch PUK (PIN Unblocking Key) genannt. Obwohl der Chip bis zu achtstellige PINs unterstützt, konnten nur vierstellige Zahlen verwendet werden (siehe auch Kapitel 7.2.3).
2.2 Personalisierungsstraße

neue Karten zunächst mit einem Profil versehen werden müssen oder ob sich schon ein Standard-Profil auf der Karte befindet.

2.2.2.1 Lebenszyklus einer Chipkarte


1. Herstellung von Chip und Chipkarte
   In dieser Phase wird das Design des Chips festgelegt, das Betriebssystem erstellt, Chip und Kartenkörper produziert und der Chip in den Kartenkörper eingebettet.

2. Kartenvorbereitung
   Anschließend wird das Chipkarten-Betriebssystem komplettiert. Somit ist es möglich, noch kleinere Änderungen am Betriebssystem vorzunehmen und es so an die spezifischen Anwendungen anzupassen.

3. Anwendungsvorbereitung
   Während dieser Phase wird die Datenstruktur auf der Karte angelegt und die Karte optisch und elektronisch personalisiert. Anschließend wird die Karte an den Benutzer ausgegeben.

4. Kartenbenutzung
   Diese Phase betrifft ausschließlich die Kartennutzung durch Anwender. Es kann sich dabei auch um die Aktivierung, bzw. Deaktivierung von auf der Karte befindlichen Anwendungen handeln.

5. Ende der Kartennutzung
   In der letzten Phase werden die auf der Karte befindlichen Anwendungen vollständig deaktiviert.

2.2.3 Steuerungs-Software


2.2.3.1 BadgeMaker Komponenten

- DataBase Layout Creator
  Dieses Programm ermöglicht die Anbindung vorhandener Datenbanken an BadgeMaker mittels der ODBC-Schnittstelle\(^1\). Hier werden die Datenfelder ausgewählt, die dann später im BadgeCreator und im eigentlichen Personalisierungsprogramm zur Verfügung stehen.

\(^1\)Genormte Windows-Schnittstelle, um verschiedene Datenbank Systeme mittels eines einheitlichen Befehlsatzes ansprechen zu können
**Umfeldanalyse**

- **BadgeCreator**
  Mit Hilfe dieses Programms können graphische Layouts für das Bedrucken der Chipkarte erstellt werden. Des Weiteren werden hier auch die elektronischen Layouts definiert, d.h. es wird festgelegt, welche Daten für die elektronische Personalisierung verwendet werden und welche Bibliothek für die Kodierung der Chipkarte zuständig ist.

- **BadgeMaker 5500**
  Während die Programme *DataBase Layout Creator* und *BadgeCreator* für die Produktionsvorbereitung verwendet werden, wird das Programm *BadgeMaker 5500*, welches im folgenden auch nur als BadgeMaker bezeichnet wird, für die eigentliche Produktion eingesetzt. BadgeMaker ermöglicht die Auswahl von Datensätzen, deren Inhalte anschließend auf dem Chip gespeichert, bzw. auf die Karte gedruckt werden.

### 2.2.3.2 Interener Ablauf der elektronischen Personalisierung

Neben der Kern-Software von BadgeMaker wird bei der elektronischen Personalisierung die Bibliothek *bmchip32.dll* an zwei Stellen aufgerufen. Diese Bibliothek kann vom Entwickler an die speziellen Anforderungen des jeweiligen Personalisierungsprozesses individuell angepaßt werden.

In den folgenden Abschnitten werden mögliche Funktionalitäten der Bibliothek sowie die Konventionen für die Übergabe des Datensatzes und der Parameter näher beschrieben.

**DLL-Parameterstruktur _BMCHIPINFO**


**ChipInit**

Diese Funktion kann u.a. dazu benutzt werden, um einen anderen Datensatz für die Personalisierung auszuwählen. Dies kann sinnvoll sein, falls eine Tabelle existiert, die Seriennummern von Karten bestimmte Daten zuordnet. In diesem Fall könnte man mit der Funktion ChipInit die Seriennummer der Karte auslesen, um mit dieser Information das Flag BM_LOCATE_RECORD aus der Struktur _BMCHIPINFO zu setzen. BadgeMaker würde in diesem Fall nicht den vom Benutzer ausgewählten Datensatz für die Personalisierung verwenden. Des weiteren könnten auch vom Datensatz unabhängige Prozesse ausgeführt werden. Unter diese Kategorie fällt u.a. das Anlegen einer elektronische Geldbörse oder die Überspielung der in Kapitel 2.2.2 angesprochenen Profile.

**ChipEncodeEx**

Nach der Ausführung der Funktion ChipInit wird der ausgewählte Datensatz in einer Datei gespeichert und die Funktion ChipEncodeEx aufgerufen. Der Dateiname wird dabei mit Hilfe der Variablen szChipInfoFile aus der Struktur _BMCHIPINFO übermittelt. Der Aufbau der Datei wird im nachfolgenden Abschnitt erläutert. Die primäre Aufgabe dieser Funktion ist die elektronische Personalisierung der Chipkarte. Mit Hilfe der Variablen szUpdateData, nFieldsToUpdate und nFields kann aber auch das Verändern von Daten in der Datenbank veranlaßt werden. Die Art der Aktualisierung kann durch die Flags BM_UPDATEDB_AFTER_ENCODE und BM_UPDATEDB_IF_NO_ERROR festgelegt werden.

**Format der BadgeMaker Datendatei**

BadgeMaker legt nach der Ausführung von ChipInit in einer Datei u.a. den Inhalt des Datensatzes, der durch den Benutzer oder die ChipInit-Funktion festgelegt wurde, ab. Es werden dabei nur die Felder verwendet, die zuvor durch den Benutzer im BadgeCreator dem elektronischen Layout hinzugefügt wurden. Im folgenden wird ein exemplarisches Beispiel für eine solche Textdatei vorgestellt:

```plaintext
[BExtra]
BMFile=C:\SC\BM550032\DATA\Perso_Priv_MF.BM
CHPFile=C:\SC\BM550032\Encodes\MF.CHP
```
D11Param1=
D11Param2=

[ChipFields]
TotalFields=3
Field1=NAME
Field2=MTKNR
Field3=GEBDATUM

[NAME]
Type=Character
Data=Max Mustermann

[MTKNR]
Type=Numeric
Data=123456

[GEBDATUM]
Type=Date
Data=01/01/1970


2.3 Fertigungsumfeld


2.3.1 Produktionsspezifische Aspekte der Fertigung

Bei einer Klassifizierung eines einzelnen Fertigungsprozesses sind zwei Betrachtungsweisen möglich. Durch eine Unterscheidung zwischen Massen- und Einzelfertigung wird der Personalisierungsprozeß in den gesamten Fertigungsprozeß eingeordnet, wohingegen eine Unterscheidung zwischen einstufigem und mehrstufigem Personalisierungsprozeß den einzelnen Fertigungsprozeß an sich näher beschreibt.

2.3.1.1 Betrachtungsweise: Massen- vs. Einzelfertigung

Falls aufgrund technischer Veränderungen für jeden Benutzer eine neue Karte erstellt werden muß oder für den Fall der Neueinführung eines Chipkartensystems, ist
2.3 Fertigungsumfeld

der Einsatz einer Massenfertigung sinnvoll, da sehr viele Karten auf einmal produziert werden müssen. Im Kontext einer Universitätsanwendung wäre dieses bspw. zum Semesteranfang der Fall, da dann eine Vielzahl an Erstsemestern mit Chipkarten ausgerüstet werden müßten.


2.3.1.2 Betrachtungsweise: Einstufige vs. Mehrstufige Personalisierungsprozesse


Die Entscheidung für einen mehrstufigen Personalisierungsprozeß ermöglicht gleichzeitig eine flexiblere Gestaltung des Produktionsprozesses im Hinblick auf die Einzel- und Massenfertigung.

2.3.2 Sicherheitsspezifische Aspekte der Fertigung

Für einen Personalisierungsprozeß werden sowohl Anforderungen an den elektronischen und räumlichen Aufbau, als auch die organisatorischen Abläufe gestellt. Die Integrität einer späteren Authentifizierung mittels Chipkarte ist vor allem von der Korrektheit der verwendeten Datensätze abhängig. Wurden an dieser zentralen Stelle
falsche Informationen eingegeben, wird der Benutzer auch falsch ausgewiesen. Bei der Einzelfertigung können diese Daten durch den Sachbearbeiter und den Endbenutzer kontrolliert werden, was bei der Massenfertigung hingegen nicht möglich ist. Da die verwendete Datenbank eine auch durch das Datenschutzrecht begründete sicherheits-, bzw. unternehmenskritische Ressource darstellt, sollte die Integrität der Daten durch die bestehende Infrastruktur gewährleistet sein, so daß die Gefahr einer Manipulation der Daten von außen als gering einzuschätzen ist. Im Rahmen dieser Studienarbeit wird deshalb davon ausgegangen, daß nur berechtigte Personen physischen Zugang zu der Datenbank und den Komponenten von FlexiTRUST, bzw. der Personalisierungsstraße haben. Es wird in einfachsten Fall davon ausgegangen, daß sich die für die Personalisierung benötigten Daten, die Personalisierungsstraße und das Trustcenter inklusive CA und ihr privater Schlüssel in einer nicht-öffentlichen Umgebung befinden und auch keine Netzwerkverbindungen nach außen existieren.

Es kann allerdings die Sicherheitsanforderung bestehen, daß die CA komplett vom Rest des Personalisierungssystems getrennt ist. Eine Erweiterung der Sicherheitsanforderungen könnte darin bestehen, daß nur von bestimmten Rechnern ein Personalisierungsprozeß gestartet werden darf. Ein solcher räumlicher und elektronischer Aufbau führt zu einem mehrstufigen Personalisierungsprozeß, da sonst die Personalisierung nicht in vertretbarer Zeit bewerkstelligt werden kann.

Eine weiteres sicherheitstechnisches Problem entsteht durch die Ausgabe der Karte und die Zuteilung der PIN. Die Sicherheit ist dabei weitestgehend abhängig vom gewählten organisatorischen Ablauf. Im folgenden werden mehrere Arten des Managements vorgestellt, für die im Rahmen dieser Studienarbeit eine Infrastruktur geschaffen wird.


- Noch unsicherer wird die Ausgabe, falls alle Karten mit einer gleichen PIN (Null-PIN) versehen und ausgeben werden. Dann kann jeder, der die Null-PIN kennt und eine noch nicht aktivierte Karte findet, die Karte verwenden.

Um diese denkbaren Formen des PIN/PUK-Managements umsetzen zu können, muß es möglich sein, während der Personalisierung die PIN/PUK zu ändern, bzw. bei einer Änderung die neuen Werte sicher hinterlegen zu können.
3 Konkretisierung der Aufgabenstellung

Aufgrund der Umfeldanalyse in Kapitel 2 kann das dort vorgestellte grobe Konzept eines kombinierten Personalisierungsprozesses (vergleiche Abbildung 1) weiter verfeinert werden.

Abbildung 6 zeigt ein abstraktes Konzept, anhand dessen im nachfolgenden Kapitel mehrere denkbare Szenarien vorgestellt und untersucht werden.

Ein sogenanntes Caller-Programm, bei dem es sich bspw. um BadgeMaker handeln könnte, startet zu Beginn einen Personalisierungsprozeß indem es eine Verbindung zu dem zu entwickelnden Programm, das auch Listener genannt wird, aufbaut. Der Listener hat die Aufgabe, für eine Verknüpfung der Arbeitsprozesse (vergleiche Abbildung 2 und 3) zu sorgen, um somit einen kombinierten Personalisierungsprozeß ermöglichen zu können.

Wie genau die Kommunikation zwischen den einzelnen Software-Komponenten aussehen, ist offen. Auch wurde noch nicht festgelegt, welche Komponenten mit der Hardware kommunizieren. Dagegen wird aber angenommen, daß die Caller-Software den Personalisierungsprozeß startet und von ihr der zu verwendende Datensatz ausgewählt wird.


Abbildung 6: Konkretisierung der Aufgabenstellung
4 Szenarien für die Anbindung an FlexiTRUST

In diesem Kapitel werden verschiedene denkbare Szenarien vorgestellt, die den Ablauf des Personalisierungsprozesses auf unterschiedliche Art und Weise gestalten.

4.1 Einstufige Personalisierung der Chipkarte

Innerhalb dieses Kapitels wird zunächst untersucht, wie sich ein einstufiger Personalisierungsprozeß realisieren lassen könnte. Es werden zwei vereinfachte Szenarien vorgestellt, die den Personalisierungsprozeß von unterschiedlichen Stellen aus durchführen und eine Grundlage für die nachfolgenden mehrstufigen Szenarien bieten. Alle Komponenten befinden sich in einem Netzwerk.

4.1.1 Einstufige Personalisierung durch FlexiTRUST

In dieser Variante wird der Listener nur für das Durchreichen von Informationen verwendet, die Personalisierung der Chipkarte wird von FlexiTRUST ausgeführt.

Abbildung 7: Einstufige Personalisierung durch FlexiTRUST

1. Die Karte wird durch BadgeMaker eingezogen und in die Leseposition des Kartenlesers befördert. Der Caller `bmchip32.dll` ruft mit dem aktuellen Datensatz den Listener auf.

2. Der Listener stellt mit den übermittelten Daten einen Antrag an die RA.
3. Als erstes überprüft FlexiTRUST, ob bereits ein Schlüsselpaar auf der Karte vorhanden ist. Falls die Karte noch nicht über ein solches Paar verfügt, wird ein Paar erzeugt und der öffentliche Schlüssel ausgelesen.

4. Danach wird ein Zertifikat erstellt und auf der Karte gespeichert.

5. Anschließend wird die Kontrolle zurück an den Listener, respektive BadgeMaker übergeben.

6. Im letzten Schritt bedruckt BadgeMaker die Karte.

Für eine räumliche Trennung der Komponenten müßte neben der Netzwerkverbindung zwischen FlexiTRUST und Listener auch eine serielle Verbindung zwischen Drucker und FlexiTRUST verwendet werden. Aus diesem Grund ist die Personalisierung durch den Listener eher geeignet, da sich in diesem Fall die Trustcenter-Komponente besser vom Rest der Personalisierungsstraße trennen läßt. Im folgenden Abschnitt wird ein solches Szenario vorgestellt.

4.1.2 Einstufige Personalisierung durch den Listener

Die Zugriffe auf die Chipkarte werden durch den Listener durchgeführt, der die Zertifikatsfunktionen von FlexiTRUST in Anspruch nimmt. Der vereinfachte Ablauf gestaltet sich wie folgt:

1. Die Karte wird durch BadgeMaker eingezogen und in die Leseposition des Kartenlesers befördert. Der Caller `bmchip32.dll` ruft mit dem aktuellen Datensatz den Listener auf.
2. Als erstes überprüft der Listener, ob bereits ein Schlüsselpaar auf der Karte vorhanden ist. Falls die Karte noch nicht über ein solches Paar verfügt, erzeugt das Programm ein Paar und merkt sich den öffentlichen Schlüssel.

3. Der Listener stellt nun einen Antrag an FlexiTRUST und übermittelt die benötigten Daten.


5. Anschließend wird die Kontrolle zurück an BadgeMaker übergeben.

6. Im letzten Schritt bedruckt BadgeMaker die Karte.

### 4.2 Mehrstufige Personalisierung der Karten

Bei diesem Verfahren ist die Personalisierung in mehrere Schritte unterteilt, die teilweise zeitlich und räumlich voneinander getrennt werden können. Die Szenarien orientieren sich dabei zum Teil an der einstufigen Personalisierung durch den Listener des vorherigen Abschnitts.

#### 4.2.1 Szenario I: Personalisierung in drei Schritten

Innerhalb dieses Szenarios wird die Karte in drei Schritten personalisiert. Zuerst werden Schlüsselpaare für die Chipkarten erzeugt, die an die Seriennummer der Karte gekoppelt sind. Im zweiten Schritt werden Benutzerdaten der Seriennummer und somit auch dem Schlüsselpaar zugeordnet und das Zertifikat erstellt. Im letzten Schritt wird die Karte bedruckt und das Zertifikat auf die Chipkarte geschrieben. Voraussetzung ist dabei, daß die einzelnen Personen über Seriennummern, die denen der Karten entsprechen, eindeutig identifizierbar sind.

![Diagramm](image_url)  
**Abbildung 9: Szenario I - Vorpersonalisierung**
Vorpersonalisierung

1. Die Chipkarte wird an die Leseposition des Kartenlesers gebracht.

2. Es wird für die Karte ein Schlüsselpaar auf der Karte selbst oder von dem angeschlossenen Rechner erzeugt und auf dem Chip gespeichert.

3. Die ausgelesene Seriennummer wird zusammen mit dem Public Key in eine Datenbank geschrieben.

4. Die Karte wird in den Ausgabeschacht befördert.

Mit diesem Verfahren kann eine große Menge an Karten mit fertig erzeugten Schlüsselpaaren produziert werden. Diese Methode läßt sich auch zum Beschleunigen der im vorherigen Kapitel 4.1 vorgestellten Szenarien verwenden. In einem solchen Fall wäre die Speicherung der Seriennummer zusammen mit dem Public Key in einer Datenbank obsolet.

Zertifizierung

1. Mit einem gesonderten Programm wählt man aus, für welche Datensätze der Datenbank personalisierte Karten erstellt werden sollen. Es ist auch denkbar, daß das Programm automatisch nach neuen Einträgen in der Datenbank sucht.

2. Für diese Datensätze wird dann jeweils ein Antrag bestehend aus Seriennummer, Public-Key und Benutzerdaten an FlexiTRUST gestellt.

3. Die CA stellt die Zertifikate aus und legt diese zusammen mit den Seriennummern in der Datenbank ab.

Abbildung 10: Szenario I - Zertifizierung
4.2 Mehrstufige Personalisierung der Karten

Abbildung 11: Szenario I - Personalisierung

Personalisierung

1. BadgeMaker zieht die Karte ein und liest die Seriennummer aus.

2. Danach wird das entsprechende Zertifikat zu dieser Karte in der Datenbank gesucht.


4. Im letzten Schritt wird die Karte bedruckt.


4.2.2 Szenario II: Getrennte optische und elektronische Personalisierung

Diese Verfahren ist in zwei Arbeitsschritte unterteilt. Im ersten Schritt wird die Karte bedruckt und Daten für eine spätere Identifizierung auf die Karte geschrieben. Später wird die Karte dann mit einem Zertifikat versehen.

Optische Personalisierung

1. Die Karte wird durch BadgeMaker eingezogen und in die Leseposition des Kartenlesers befördert. Der Caller \textit{bmchip32.dll} ruft mit dem aktuellen Datensatz den Listener auf.
2. Der Listener speichert die Daten auf der Chipkarte und übergibt die Kontrolle zurück an BadgeMaker.

3. Im letzten Schritt wird die Karte bedruckt.

**Abbildung 12: Szenario II - Optische Personalisierung**

**Elektronische Personalisierung**

1. Die Karte wird durch den Caller eingezogen und in die Leseposition des Kartenaufklebers befördert, anschließend wird der Listener aufgerufen.

2. Der Listener erzeugt Schlüssel auf der Karte und liest die vorhandenen Daten ein.

3. Der Listener stellt einen Antrag an FlexiTRUST und erhält ein Zertifikat zurück.

4. Im letzten Schritt wird das Zertifikat gespeichert und die Kontrolle zurück an den Caller gegeben.

Dieses Verfahren würde eine räumliche und zeitliche Trennung von optischer und elektronischer Personalisierung zulassen. Im Rahmen einer Massenproduktion bietet es sich an, die gesamten bedruckten Karten an das Trustcenter zu liefern, wo diese dann zertifiziert werden. Auch wenn das Trustcenter komplett von den Komponenten der optischen Personalisierung getrennt ist, ergibt sich eine Sicherheitslücke, falls eine bedruckte Karte abhanden kommt. Ein Dritter könnte diese dann mit anderen Daten versehen, um so ein falsches Zertifikat zu beantragen. Durch mögliche Programm- oder Prozeßfehler besteht auch die Gefahr, daß die aufgedruckten Daten nicht zu denen des Chips passen.
4.2 Mehrstufige Personalisierung der Karten

4.2.3 Szenario III: Aktivierung durch den Endbenutzer


**Personalisierung**

1. Die Karte wird durch BadgeMaker eingezogen und in die Leseposition des Kartensamplers befördert. Der Caller `bmchip32.dll` ruft mit dem aktuellen Datensatz den Listener auf.

2. Als erstes überprüft der Listener, ob bereits ein Schlüsselpaar auf der Karte vorhanden ist. Falls die Karte noch nicht über ein solches Paar verfügt, erzeugt das Programm ein Paar und merkt sich den öffentlichen Schlüssel.

3. Der Listener stellt nun einen Antrag an FlexiTRUST und übermittelt die benötigten Daten.

4. Anschließend wird die Kontrolle zurück an BadgeMaker übergeben.

5. Im letzten Schritt bedruckt BadgeMaker die Karte.
Nachdem die Anträge gestellt wurden, werden diese an die CA zur Zertifizierung übergeben. Die CA stellt die Zertifikate aus und veröffentlicht diese.

**Zertifizierung**

Nachdem die Anträge gestellt wurden, werden diese an die CA zur Zertifizierung übergeben. Die CA stellt die Zertifikate aus und veröffentlicht diese.

Abbildung 14: Szenario III - Personalisierung

Abbildung 15: Szenario III - Aktivierung der Karte
Aktivierung der Karte

1. Der Benutzer legt die Karte in den Kartenleser und authentifiziert sich mittels PIN.

2. Das Terminal-Programm liest den Schlüssel der Karte aus und sucht ein passendes Zertifikat.

3. Abschließend wird das Zertifikat auf der Karte gespeichert.
5 Konzept für die Anbindung der Personalisierungsstraße an FlexiTRUST


Das verwendete Konzept orientiert sich dabei eng an dem in Abschnitt 4.1.2 vorgestellten einstufigen Prozeß, wobei hier die einzelnen Arbeitsschritte entkoppelt werden können. Die vorgestellten mehrstufigen Szenarien sind nur bedingt geeignet, da sie teilweise eine einstufige Personalisierung erschweren (Szenario I und III), bzw. sicherheitstechnische Schwachpunkte aufweisen (Szenario II).

5.1 Caller

Der Caller, der auf der Seite der Personalisierungsstraße angesiedelt ist, startet anfangs den Personalisierungsprozeß. Er ist für die Ansteuerung der Druckereinheit zuständig und stellt dem Listener die für die Personalisierung benötigten Daten zur Verfügung. Abschließend verarbeitet er die Rückmeldung des Listeners. Die Verarbeitung kann die folgenden Punkte beinhalten:

- Die Karte wird bedruckt, bzw. nicht bedruckt.
- Die Karte wird ausgeworfen und mit der Produktion wird fortgefahren. Dabei könnte im Fehlerfall auch ein zweiter Auswurfschacht für defekte Karten verwendet werden.
- Die Produktion wird bei Auftritt eines Fehlers gestoppt.
- Der aktuelle Datensatz wird verändert, um bspw. Daten der Karte zu speichern oder den Status des Personalisierungsprozesses zu vermerken.

Aufgrund der zur Verfügung gestellten Personalisierungsstraße ist die Caller-Komponente in C programmiert.

5.2 Listener

Der in Java programmierte Listener hat die Aufgabe, die Anfragen des Callers zu verarbeiten und ihm nach Bearbeitung des Auftrags eine Rückmeldung zuzuschicken. Die Verarbeitung eines Auftrags beinhaltet neben der elektronischen Personalisierung die Kommunikation mit FlexiTRUST. Eine Rückmeldung kann neben dem Status auch Daten der Chipkarte wie Seriennummer, PIN, PUK oder öffentliche Schlüssel der Karte beinhalten. Innerhalb des Listeners können die folgenden drei Arbeitsschritte miteinander kombiniert und ausgeführt werden (vgl. Abbildung 16):

1. **Bereitstellen einer vorpersoanalisierten Karte**
   Zu Beginn kann - entsprechend der Konfiguration des Listeners - der Inhalt der Karte gelöscht werden. Falls die vorhandenen PINs nicht weiterverwendet
werden sollen, überschreibt der Listener diese im zweiten Schritt. Anschließend werden, falls die Karte zuvor nicht gelöscht wurde, evtl. noch fehlende Schlüsselpaare auf der Karte erzeugt. Wenn die Karte gelöscht wurde, werden entsprechend der Konfiguration Schlüssel auf der Karte generiert. Abschließend werden die öffentlichen Schlüssel der Karte ausgelesen. Falls beim Zugriff auf die Karte ein Fehler auftritt, wird die Personalisierung abgebrochen.

2. RA-Anträge stellen
Im zweiten Schritt werden Zertifikatsanträge an die RA gestellt. Für die Anträge können die vom Caller übermittelten Daten sowie die Daten der vorpersonalisierten Chipkarte verwendet werden. Falls die RA einen Fehler meldet, wird der Personalisierungsprozeß abgebrochen.

3. Speichern von Zertifikaten

Abbildung 16: Interner Prozeßablauf des Listeners
Der erste Arbeitsschritt wird immer ausgeführt. Aufgrund der Konfigurationsmöglichkeiten wird evtl. aber nur ein Teil der zuvor beschriebenen Unterschritte verwendet. Zum Ende dieses Arbeitsschritts stehen aber immer die sich auf der Karte befindlichen Daten wie öffentliche Schlüssel, PIN, PUK oder Seriennummer zur Verfügung. Der erste Arbeitsschritt kann, falls gewünscht, mit den zwei anderen Arbeitsschritten, die nur optional sind, verknüpft werden. Durch die Kombinationsmöglichkeiten ergeben sich die folgenden Anwendungsfälle für die elektronische Personalisierung:

1. **Einstufige elektronische Personalisierung**
   Durch einen einzigen Prozeß werden Karten vollständig personalisiert. Alle drei Arbeitsschritte werden nacheinander ausgeführt.

2. **Zweistufige elektronische Personalisierung (I)**
   In einem ersten Arbeitsprozeß werden Schlüssel erzeugt und anschließend Anträge an die RA gestellt. Danach werden in einem zweiten Prozeß die Zertifikate auf der Karte gespeichert.

3. **Zweistufige elektronische Personalisierung (II)**
   In einem ersten Arbeitsprozeß werden Schlüssel erzeugt. In einem zweiten Prozeß werden die RA-Anträge gestellt und die Zertifikate auf der Karte gespeichert.

4. **Dreistufige elektronische Personalisierung**
   Im ersten Prozeß werden Schlüssel auf der Karte erzeugt. Im zweiten Arbeitsprozeß werden evtl. fehlende Schlüsselpaare ergänzt und Anträge an die RA gestellt. In einem dritten Arbeitsprozeß werden die Zertifikate auf der Karte gespeichert.

Die optische Personalisierung kann mit diesen Anwendungsfällen kombiniert werden, da sie nur vom verwendeten Caller abhängig ist.

### 5.3 Kommunikationsprotokoll Caller/Listener
Als Schnittstelle zwischen Java- und C-Programmen bietet sich neben dem Java Native Interface (JNI) die Verwendung von Netzwerk-Sockets an. In dieser Studienarbeit wird die Kommunikation mittels Sockets bewerkstelligt, da so später nicht nur Programme, die sich an das festgelegte Protokoll halten, eingebunden werden können, sondern auch der Aufbau eines verteilten Caller/Listener-Systems möglich ist. Es handelt sich hierbei nicht um ein interaktives Protokoll, da beim Aufruf nur der Caller und nach Abschluß der Personalisierung nur der Listener Daten sendet. Sollte die Netzwerkverbindung unterbrochen werden, vermerken beide, daß die Personalisierung fehlgeschlug.

#### 5.3.1 Aufruf des Listeners durch den Caller
Der Caller kann dem Listener beim Aufruf Datensätze übermitteln. Pro Information, bestehend aus Feldname und Feldinhalt wird dem Listener eine Zeile zugesendet. Das folgende Protokoll wird vom Caller beim Aufruf des Listeners verwendet:

```
<N-1> ::= <Zeilenindex> . N-1 <Feldname> <Feldinhalt> \n
N: END \n```
Mit jeder übermittelten Zeile steigt der Zeilenindex um eins. Jede Zeile endet mit einem `Linefeed (\n)`, wobei beachtet werden muß, daß kein zusätzliches `Carriage Return (\r)` gesendet wird. Um dem Listener mitteilen zu können, daß alle Daten übermittelt wurden wird zum Schluß ein "END" übermittelt. Im folgenden ein Beispiel für einen möglichen Aufruf:

1: `NAME: Max Mustermann`
2: `MATR: 123456`
3: `END`

### 5.3.2 Rückmeldung des Listeners

Nach Abschluß des Personalisierung sendet der Listener eine Zeile an den Caller. Diese kann neben einer Statusmeldung zusätzliche Informationen beinhalten. Der folgende Ausdruck beschreibt die Rückmeldung:

```plaintext
<Status>[:<Variablen-Name>=<Variablen-Wert>]*
```


### 5.4 Kommunikationsprotokoll Listener/FlexiTRUST

#### 5.4.1 Anbindung an die RA


#### 5.4.2 Abholen der Zertifikate


Da sich der `dName` des Antrags von dem des Formulars unterscheiden kann, bspw. durch das Ersetzen von Umlauten, wird genau dann ein Zertifikat verwendet, wenn die öffentlichen Schlüssel von Formular und Karte übereinstimmen. Ein Zertifikat muß dabei atomar in das Verzeichnis kopiert werden.

Es wird dieser Weg gewählt, da aus sicherheitstechnischen Gründen nicht unbedingt eine direkte Kommunikation mit der CA möglich ist. Da nur anhand der Schlüssel ein Zertifikat gesucht werden kann ist die Suche in einem Datei-Verzeichnis, das nur "neue" Zertifikate beinhaltet, schneller als die Suche in einem mitunter großen LDAP-Baum, welches alle bisher ausgestellten Zertifikate beinhaltet.
Implementierung des Callers

Im folgenden werden zwei Caller vorgestellt, die für die komplette, bzw. elektronische Personalisierung eingesetzt werden können. Da BadgeMaker eng an die optische Personalisierung gekoppelt ist, wird ein zweiter Caller vorgestellt, mit dem sowohl eine elektronische Vorphersonalisierung, als auch eine vollständige elektronische Personalisierung möglich ist.

6.1 BadgeMaker als Caller

BadgeMaker selbst beinhaltet schon eine Schnittstelle, mit der eine individuelle elektronische Personalisierung der Chipkarten vor der optischen Personalisierung möglich ist (vgl. Kapitel 2.2.3). Dazu muß eine Bibliothek (bmchip32.dll) erstellt werden, die die vorgegebenen Funktionen beinhaltet die während der Personalisierung von BadgeMaker aufgerufen werden. Da BadgeMaker auch über eine Schnittstelle zur Datenbank verfügt, können nach dem elektronischen Personalisieren der Karte noch Felder in der Datenbank aktualisiert werden. So könnte man z.B. die Seriennummer der Karte im entsprechenden Datensatz vermerken. Im Fehlerfall zeigt BadgeMaker eine Meldungsbox mit dem aufgetretenen Fehler an.

Nach der Erläuterung der Parametrisierung der Bibliothek wird im folgenden das verwendete Kommunikationsprotokoll beschrieben.

6.1.1 Parametrisierung der bmchip32.dll

Um den Caller aus BadgeMaker heraus zu konfigurieren, können die Felder DllParam1 und DllParam2 im elektronischen Layout des BadgeCreators gesetzt werden. Diese Felder werden dann in die Textdatei geschrieben, die auch den für die Personalisierung verwendeten Datensatz beinhaltet. Mittels des ersten Parameters DllParam1 wird eine Adresse angegeben, an die die Daten für den Listener gesendet werden. Es wird dabei das folgende Format verwendet:

<IPv4-Adresse>:<Port>

An diese Adresse sendet die Bibliothek die verwendeten Daten gemäß des in Kapitel 5.3.1 vorgestellten Kommunikationsprotokolls. Falls kein Wert angegeben wird, verwendet die Bibliothek den Standardwert "127.0.0.1:2401".

Durch den zweiten Parameter DllParam2 kann der Debug-Modus der Bibliothek aktiviert werden. Dieser wird nur verwendet, falls DllParam2 auf "debug" gesetzt wurde. Mit Hilfe des Debug-Modus wird nach jeder erfolgreichen elektronischen Personalisierung eine Auswahl-Box angezeigt, mit deren Hilfe der Anwender entscheiden kann ob die Personalisierung erfolgreich war oder nicht. Falls zuvor ein Fehler auftrat wird die Box nicht angezeigt. Durch diesen Mechanismus kann der Benutzer die optische Personalisierung verhindern.
6.1.2 Kommunikationsprotokoll

Während der Aufruf des Listeners exakt in Kapitel 5.3.1 definiert wurde, kann die Rückmeldung des Listeners individuell an den Caller angepaßt werden (vgl. Kapitel 5.3.2). Im folgenden werden nun BadgeMaker-spezifische Rückmeldungs-Variable und ihre Werte näher erläutert.

Für eine Aktualisierung der Datenbank werden die Variablen FC, FN und FD verwendet:

- **FC** gibt an, wie viele Felder in der Datenbank aktualisiert werden sollen. Nur wenn dieser Parameter gesetzt ist, wird auch eine Aktualisierung in der Datenbank vorgenommen.


- **FD** enthält die Daten, die in die Datenbank zurückgeschrieben werden sollen. Dabei müssen die einzelnen Datenfelder durch "/-Symbole getrennt werden. Es ist auch ein anderes Trennzeichen möglich, dieses müßte aber im Quellcode des Callers neu definiert werden. Des weiteren muß beachtet werden, daß das komplette Datenfeld (inklusive Trennzeichen) nicht mehr als 256 Zeichen enthält, da die Datenstruktur von BadgeMaker nicht mehr Zeichen beinhalten kann.

Mit Hilfe der Variablen ER kann die Personalisierung unterbrochen werden. Wenn ER verwendet wird, öffnet BadgeMaker eine Fehler-Box mit dem Wert der Variablen nach Beendigung der elektronischen Personalisierung. Diese Variable sollte nur bei auftretenden Fehlern verwendet werden.

Beispiele für Rückgaben:

OK: fc=2: fn=5,7: fd=123456|ok

Die Personalisierung war erfolgreich. Im Datenbankfeld Nr. 5 wird der Wert "123456" und im Feld Nr. 7 der Wert "ok" gespeichert.

FAILED: fc=1: fn=8: fd=Karte defekt: er=Fehler!

Die Personalisierung war nicht erfolgreich. Im Datenbankfeld Nr. 8 wird der Wert "Karte defekt" gespeichert. Von BadgeMaker wird eine Box mit dem Inhalt "Fehler!" geöffnet.

6.2 PREPERSO als Caller

Mit Hilfe des PREPERSO-Programms ist es möglich, Chipkarten in elektronischer Hinsicht vorzupersonalisieren, bspw. durch das Erzeugen bestimmter Strukturen auf der Chipkarte. Bei dem Programm handelt es sich um eine veränderte Version der bm-chip32.dll, welches nicht die Möglichkeit bietet, Daten zurück in eine Datenbank zu schreiben. Es werden die gleichen Rückgaben des Listeners wie in Kapitel 5.3 erwartet, allerdings ist nur die Status-Meldung relevant.
Das Programm wird von der Kommandozeile mit den entsprechenden Optionen, die in Tabelle 1 aufgelistet sind, gestartet. Dort wird auch jeweils angegeben, ob eine Option weitere Parameter benötigt oder ohne Parameter verwendet werden kann.

Das Programm wurde an die Druckercodes des Eltron P310 angepaßt und müßte bei Verwendung eines anderen Druckers entsprechend abgeändert werden. Da der Drucker nicht über einen Aussortierungsschacht verfügt, kann die Option “-b” nicht mit diesem Drucker verwendet werden.

<table>
<thead>
<tr>
<th>Option</th>
<th>Bedeutung</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c [Filename]</td>
<td>Datei, die dem in Kapitel 2.2.3 vorgestellten Format entsprechen muß und beim Aufruf des Listeners übermittelt wird.</td>
</tr>
<tr>
<td>-P [int]</td>
<td>Port, an den der Drucker angeschlossen ist (Standardwert: ”1” für LPT1).</td>
</tr>
<tr>
<td>-h [IPv4-Adresse]</td>
<td>Host, auf dem der Listener auf Daten wartet (Standardwert: 127.0.0.1).</td>
</tr>
<tr>
<td>-p [int]</td>
<td>Port, der vom Listener verwendet wird (Standardwert: 2401).</td>
</tr>
<tr>
<td>-s [int]</td>
<td>Zeitspanne in Sekunden, die das Programm zwischen Karteneinzug und Kontaktierung des Listeners wartet. Der Wert sollte an die tatsächliche Zeitspanne, die der Drucker für den Karteneinzug benötigt, angepaßt werden.</td>
</tr>
<tr>
<td>-b</td>
<td>Falls der Drucker über einen Aussortierungsschacht verfügt, können defekte Karten dort abgelegt werden.</td>
</tr>
<tr>
<td>-w</td>
<td>Falls der Listener einen Fehler meldet, wird nach Ausgabe der Fehlermeldung nicht auf einen Tastendruck gewartet bevor der Personalisierungsprozeß fortgesetzt wird.</td>
</tr>
</tbody>
</table>

Tabelle 1: Parameter des Programms PREPERSO

Beispiel für einen PREPERSO Aufruf:

```bash
tabello -n 100 -s 10 -w
```

In diesem Fall würde das Programm 100 Karten vorpersonalisieren. Falls der Listener Fehler meldet, wird die entsprechende Karte ausgeworfen und der Personalisierungsprozeß wird nicht gestoppt. Nachdem der Befehl für den Karteneinzug an den Drucker gesendet wurde, wartet das Programm 10 Sekunden bevor es den Listener kontaktiert. Nachfolgend ein Beispiel für die Verwendung einer zusätzlichen Datendatei, welche dann von PREPERSO an den Listener übermittelt wird. Dieses entspricht im wesentlichen der BadgeMaker Datendatei (vgl. 2.2.3), allerdings werden nur die tatsächlich Daten-Sektionen benötigt.

Datei "daten.txt":
6.2 PREPERSO als Caller

[Name]
Data=Max Mustermann

[Fachbereich]
Data=FB20

Aufruf von PREPERSO:

preperso -c daten.txt
7 Implementierung des Listeners


Im folgenden wird kurz auf den Aufbau des Listeners eingegangen, abschließend werden die einzelnen Konfigurationsmöglichkeiten vorgestellt.

7.1 Aufbau des Listeners

Der Listen besteht aus vier Java Klassen. Listener.java ist die Hauptklasse, die die Verwendung der anderen Klassen steuert. LogFrame.java erzeugt ein Fenster, in dem während der Laufzeit entstehende Meldungen ausgegeben werden. Mit GenerateRARequest.java werden neue Anträge an die RA gestellt. Die Klasse Smartcard.java ist für das Beschreiben des Chips zuständig. Eine Beschreibung der einzelnen Funktionen findet sich in den dazugehörigen JavaDocs (vgl. elektronischer Anhang). Im folgenden werden die Informationsflüsse zwischen den einzelnen Klassen dargestellt, wobei aufgrund des Umfangs nur die Funktionen erwähnt werden, die an der direkten Kommunikation zwischen den einzelnen Komponenten beteiligt sind.

Abbildung 17: Informationsfluß innerhalb des Listeners (I)
7.2 Konfiguration des Listeners


Erfolgreiche Verarbeitung eines Auftrags

Abbildung 18 zeigt den Informationsfluß während eines erfolgreichen Personalisierungsauftrags, wobei die Chipkarte vollständig elektronisch personalisiert wird. Angemerkt sei, daß die Schritte 8 und 9 solange ausgeführt werden, bis ein entsprechendes Zertifikat gefunden wurde, respektive die Schritte 8-11 solange, bis alle Zertifikate gespeichert wurden.

7.2 Konfiguration des Listeners

Dem Listener kann beim Start eine Konfigurationsdatei übergeben werden, die die nötigen Parameter lädt, bspw. durch `java Listener listener.ini`. Falls keine Datei angegeben wird, versucht das Programm im selben Verzeichnis die Datei `config.cfg` zu öffnen. Eine Konfigurationsdatei hat dabei die folgenden Merkmale:

- Durch das Rautensymbol "#" werden Kommentare gekennzeichnet. Die gesamte Zeile wird ab diesem Zeichen ignoriert.
- Durch eckige Klammern werden einzelne Sektionen gekennzeichnet, bspw. *[Sektion1]*. Es stehen die folgenden Sektionen zur Auswahl:
  - *[input]* legt fest, von wo Daten kommen, bzw. von wem Daten akzeptiert werden
  - *[output]* legt fest, an wen welche Daten verschickt werden
  - *[smartcard]* legt fest, auf welche Weise die Smartcard codiert werden soll
Implementierung des Listeners

– \[ra\] legt fest, wie die RA angesprochen werden soll
– \[database\] legt fest, wie die Eingangsdaten in einen oder mehrere RA-Anträge umgewandelt werden

- Innerhalb der Sektionen kann einer Variable ein Wert zugewiesen werden. Syntax: \(<Variable> = <Wert>\).

Die Konfigurationsmöglichkeiten innerhalb der einzelnen Sektionen werden in den folgenden Unterkapiteln näher beschrieben.

7.2.1 Die [input] Sektion

<table>
<thead>
<tr>
<th>Variable</th>
<th>Typ</th>
<th>Case-Sensitive</th>
<th>Werte</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputport</td>
<td>int</td>
<td>-</td>
<td>Port</td>
</tr>
<tr>
<td>inputip</td>
<td>String</td>
<td>-</td>
<td>IPv4-Adresse</td>
</tr>
<tr>
<td>certificates</td>
<td>String</td>
<td>-</td>
<td>”disabled” oder &lt;Anzahl&gt;</td>
</tr>
</tbody>
</table>

Tabelle 2: Aufbau der [input] Sektion

<table>
<thead>
<tr>
<th>Variable</th>
<th>Bedeutung</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputport</td>
<td>Legt den Port fest, auf dem der Listener Daten erwartet</td>
</tr>
<tr>
<td>inputip</td>
<td>Von dieser Adresse akzeptiert der Listener Daten</td>
</tr>
<tr>
<td>certificates</td>
<td>Steuert die Verwendung von Zertifikaten</td>
</tr>
</tbody>
</table>

Tabelle 3: Bedeutung der [input] Variablen

Über die Variablen \(inputport\) und \(inputip\) wird die Verbindung zum Caller hergestellt. Mit jedem neuen Verbindungsaufbau durch den Caller wird ein neuer Personalisierungsprozeß gestartet. Die Variable \(inputip\) ist optional, ihr Standardwert ist "127.0.0.1".

Falls die Variable \(certificates\) auf "disabled" gesetzt wird, werden keine Zertifikate auf der Smartcard gespeichert. Ansonsten muß ihr Wert einen gültigen Pfad beinhalten, in dem das Programm nach Zertifikatsdateien sucht. Bei der Verwendung von Zertifikaten muß das Trustcenter diese atomar schreiben, da dem Listener nur vollständige Zertifikate zur Verfügung gestellt werden dürfen. Falls Zertifikate geschrieben werden sollen, wartet der Listener solange, bis - festgelegt durch \(<\text{Anzahl}>\) - genügend distinkte Zertifikate gefunden wurden. Die Variable \(certificates\) ist optional, ihr Standardwert ist "disabled".

7.2.2 Die [output] Sektion

Falls eine RA verwendet wird \((ra = "enabled")\), werden entsprechend der definierten Anzahl (siehe auch Abschnitt 7.2.5) Anträge generiert und an die RA übermittelt. Das Scheitern eines Antrags führt zum kompletten Abbruch des jeweiligen Personalisierungsprozesses. Die Variable \(ra\) ist optional, ihr Standardwert ist "disabled".
7.2 Konfiguration des Listeners

<table>
<thead>
<tr>
<th>Variable</th>
<th>Typ</th>
<th>Case-Sensitive</th>
<th>Werte</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>String</td>
<td>-</td>
<td>&quot;enabled&quot; oder &quot;disabled&quot;</td>
</tr>
<tr>
<td>return_1</td>
<td>String</td>
<td>ja</td>
<td>Text und/oder Listener-Variablen (%Variable%), vgl. Tabelle 6</td>
</tr>
<tr>
<td>return_0</td>
<td>String</td>
<td>ja</td>
<td>Text und/oder Listener-Variablen (%Variable%), vgl. Tabelle 6</td>
</tr>
</tbody>
</table>

Tabelle 4: Aufbau der [output] Sektion


```
return_1 = OK:fc=2:fn=5,7:fd=%SERIAL_NR%/ok
return_0 = FAILED:fc=1:fn=8:fd=%ERROR%/er=Fehler!
```

In diesem speziellen Fall wäre es auch möglich, die interne Listener-Variablen %STATUS% zu verwenden. In diesem Fall hätten die Return-Strings die folgenden Werte:

```
return_1 = %STATUS%:fc=2:fn=5,7:fd=%SERIAL_NR%/ok
return_0 = %STATUS%:fc=1:fn=8:fd=%ERROR%/er=Fehler!
```

Bei der Verwendung von zufällig generierten PINs (siehe auch Kapitel 7.2.3) sollten diese unbedingt an den Caller zurückgegeben werden. Zur Sicherheit werden die Kartendaten %SERIAL_NR%, %ADMIN_PIN% und %USER_PIN% für jede Karte einzeln in der Log-Datei des Listeners gespeichert. Falls BadgeMaker verwendet wird, kann keiner der öffentlichen Schlüssel %PUBLIC_KEY1% und/oder %PUBLIC_KEY2% zurückgegeben werden, da der Return-String für BadgeMaker zu lang wäre. Bei den beiden Schlüsseln handelt es sich im Falle der Return-Strings jeweils um String-Repräsentationen der jeweiligen Java-Objekte.

7.2.3 Die [smartcard] Sektion

Mittels der Variablen `chipdll` wird die DLL festgelegt, die als Schnittstelle zum Kartenleser und zur Smartcard fungiert. Bei der Verwendung von GemPlus Karten stehen
Implementierung des Listeners

<table>
<thead>
<tr>
<th>Listener-Variable</th>
<th>Bedeutung</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ADMIN_PIN%</td>
<td>Die neue Admin-PIN der Karte</td>
</tr>
<tr>
<td>%USER_PIN%</td>
<td>Die neue User-PIN der Karte</td>
</tr>
<tr>
<td>%SERIAL_NR%</td>
<td>Die Seriennummer der Karte</td>
</tr>
<tr>
<td>%PUBLIC_KEY1%</td>
<td>Der erste öffentliche Schlüssel</td>
</tr>
<tr>
<td>%PUBLIC_KEY2%</td>
<td>Der zweite öffentliche Schlüssel</td>
</tr>
<tr>
<td>%STATUS%</td>
<td>In Abhängigkeit des Personalisierungsausgangs &quot;FAILED&quot; oder &quot;OK&quot;</td>
</tr>
<tr>
<td>%ERROR%</td>
<td>Eine Fehlermeldung falls die Personalisierung fehlschlägt, ansonsten ein Leer-String</td>
</tr>
</tbody>
</table>

Tabelle 6: Interne Listener-Variablen

<table>
<thead>
<tr>
<th>Variable</th>
<th>Typ</th>
<th>Case-Sensitive</th>
<th>Werte</th>
</tr>
</thead>
<tbody>
<tr>
<td>chipdll</td>
<td>String</td>
<td>-</td>
<td>Dateiname</td>
</tr>
<tr>
<td>initialadminpin</td>
<td>int</td>
<td>-</td>
<td>Vierstellige Zahl</td>
</tr>
<tr>
<td>adminpin</td>
<td>String</td>
<td>-</td>
<td>Vierstellige Zahl oder &quot;random&quot;</td>
</tr>
<tr>
<td>userpin</td>
<td>String</td>
<td>-</td>
<td>Vierstellige Zahl oder &quot;random&quot;</td>
</tr>
<tr>
<td>secondkeypair</td>
<td>boolean</td>
<td>-</td>
<td>&quot;true&quot; oder &quot;false&quot;</td>
</tr>
<tr>
<td>keylength1</td>
<td>int</td>
<td>-</td>
<td>&quot;1024&quot; oder &quot;512&quot;</td>
</tr>
<tr>
<td>keylength2</td>
<td>int</td>
<td>-</td>
<td>&quot;1024&quot; oder &quot;512&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Wird nur benötigt falls secondkeypair auf &quot;true&quot; gesetzt wurde</td>
</tr>
<tr>
<td>erasecard</td>
<td>boolean</td>
<td>-</td>
<td>&quot;true&quot; oder &quot;false&quot;</td>
</tr>
<tr>
<td>cardlabel</td>
<td>String</td>
<td>ja</td>
<td>Text. Wird nur benötigt falls erasecard auf &quot;true&quot; gesetzt wurde</td>
</tr>
<tr>
<td>debug</td>
<td>boolean</td>
<td>-</td>
<td>&quot;true&quot; oder &quot;false&quot;</td>
</tr>
</tbody>
</table>

Tabelle 7: Aufbau der [smartcard] Sektion

die DLLs "gclib.dll" und "gctrack.dll" zur Auswahl. Die Bibliothek "gctrack.dll" ist für die Software Entwicklung gedacht, da sie selbst die Datei "gclib.dll" verwendet und zusätzlich nur die Aufrufe von Funktionen protokolliert.

Die Variable initialadminpin wird benötigt, um die Karte korrekt beschreiben zu können. Die initialen PINs sind dabei dem jeweiligen Handbuch zu entnehmen, für GemPlus Karten steht die Admin-PIN standardmäßig auf "1234" (siehe auch [7]). Falls userpin oder adminpin auf "random" gesetzt wurden, sollten die neuen PINs an den Caller zurückgesendet werden, da ansonsten die Karte nicht mehr benutzt werden kann. Wenn der Listener falsch konfiguriert wurde oder durch einen Fehler während der Personalisierung eine PIN nicht in die Datenbank geschrieben werden konnte kann diese immer noch anhand der Seriennummer der Karte in der Log-Datei gesucht werden. Der Listener vermerkt für jede Karte die Seriennummer und die letzten bekannten PINs in der Log-Datei.

Aufgrund eines Programmfehlers ist es derzeit nicht möglich, PINs mit mehr als 4 Zeichen zu verwenden obwohl GemPlus-Karten PINs mit bis zu 8 Zeichen unterstützen.
### 7.2 Konfiguration des Listeners

<table>
<thead>
<tr>
<th>Variable</th>
<th>Bedeutung</th>
</tr>
</thead>
<tbody>
<tr>
<td>chipdll</td>
<td>Verwendete PKCS#11-DLL</td>
</tr>
<tr>
<td>initialadminpin</td>
<td>Admin-PIN, die vor der Personalisierung auf der Karte gespeichert ist.</td>
</tr>
<tr>
<td>adminpin</td>
<td>Admin-PIN, die nach der Personalisierung auf der Karte gespeichert ist. Falls adminpin auf &quot;random&quot; gesetzt wurde, handelt es sich dabei um eine zufällige vierstellige PIN.</td>
</tr>
<tr>
<td>userpin</td>
<td>User-PIN, die nach der Personalisierung auf der Karte gespeichert ist. Falls userpin auf &quot;random&quot; gesetzt wurde, handelt es sich dabei um eine zufällige vierstellige PIN.</td>
</tr>
<tr>
<td>secondkeypair</td>
<td>Diese Variable legt fest, ob zusätzlich ein zweites Schlüsselpaar erzeugt wird.</td>
</tr>
<tr>
<td>keylength1</td>
<td>Schlüssellänge des ersten RSA-Moduls</td>
</tr>
<tr>
<td>keylength2</td>
<td>Schlüssellänge des zweiten RSA-Moduls</td>
</tr>
<tr>
<td>erasecard</td>
<td>Falls &quot;true&quot; werden sämtliche vorhandenen Kartenobjekte gelöscht und auf der Karte ein neuer Name (cardlabel) gesetzt.</td>
</tr>
<tr>
<td>cardlabel</td>
<td>Der Wert dieser Variablen wird im Falle von erasecard == &quot;true&quot; auf die Karte geschrieben.</td>
</tr>
<tr>
<td>debug</td>
<td>Mit dieser Variablen läßt sich die Kodierung der Smartcard verhindern.</td>
</tr>
</tbody>
</table>

Tabelle 8: Bedeutung der [smartcard] Variablen

Da während der Entwicklung nur wenige Karten zur Verfügung standen konnte dieser Fehler nicht näher eingegrenzt oder behoben werden, zumal eine Karte aufgrund dieses Fehlers unbrauchbar wurde. Es wird allerdings vermutet, daß der Fehler mit der veralteten Beta-Version des PKCS#11-SDKs zusammenhängt. So ist nach einer Änderung der Admin-PIN von "1234" auf "12345" immer noch die PIN "1234" gültig. Bei einer Änderung von "1234" auf "54321" ist die neue physisch gesetzte PIN bisher unbekannt.

Die Variable secondkeypair nimmt nur Einfluß auf die Codierung der Smartcard, nicht aber auf die Anzahl der eventuell zu stellenden Anträge an die RA. Falls sie auf "true" gesetzt wird, befinden sich nach einem erfolgreichen Ablauf zwei Schlüsselpaare auf der Karte, ansonsten nur ein Paar. Die beiden Schlüssellängen der RSA-Moduln keylength1 und keylength2 können theoretisch die Werte "512", "756" oder "1024" annehmen. Da allerdings der Listener die Schlüssel auf der Karte generiert, kann nur "512" oder "1024" verwendet werden. Der Exponent e des öffentlichen Schlüssels ist fest definiert und deshalb nicht parametrisierbar. Diese Restriktionen werden näher im Handbuch zur verwendeten CRYPTOKI Bibliothek [6] erläutert. keylength2 muß nur gesetzt sein, wenn secondkeypair auf "true" gesetzt wurde, ansonsten ist dieser Parameter optional.

Für neue Karten bietet es sich an, erasecard auf "true" zu setzen. Dadurch wird der bisherige Kartenname durch den Wert der Variablen cardlabel überschrieben und es wird sichergestellt, daß die Karte vor dem eigentlichen Personalisierungsprozeß wirklich leer ist. Der Wert "false" sollte benutzt werden, falls in einem vorherigen Prozeß...
Schlüssel auf den Karten gespeichert wurden. Der Listener kann, falls aufgrund seiner Konfiguration weder RA-Anträge gestellt, noch Zertifikate erwartet werden, für einen solchen Prozeß, der nur Schlüssel auf der Karte generiert, verwendet werden (siehe auch Beispiele in Anhang A).

Der Debug-Modus (\( \text{debug} == "true" \)) sollte nur für die Entwicklung benutzt werden, da ansonsten RA-Anträge nicht erfolgreich gestellt werden können und jedes gefundene Zertifikat akzeptiert werden würde. Eingebaut wurde dieser Parameter, da durch die Instabilität des PKCS#11-SDKs die verwendete \( \text{chipdll} \) unter Windows 98SE nur viermal geladen werden kann, bevor der Rechner neu gestartet werden muß. Die Verwendung dieser Variablen in der Konfigurationsdatei ist optional, ihr Standardwert ist "false".

7.2.4 Die [ra] Sektion

<table>
<thead>
<tr>
<th>Variable</th>
<th>Typ</th>
<th>Case-Sensitive</th>
<th>Werte</th>
</tr>
</thead>
<tbody>
<tr>
<td>formname1</td>
<td>String</td>
<td>ja</td>
<td>Formular-Name</td>
</tr>
<tr>
<td>formname2</td>
<td>String</td>
<td>ja</td>
<td>Formular-Name</td>
</tr>
<tr>
<td>racfg</td>
<td>String</td>
<td>-</td>
<td>Dateiname</td>
</tr>
</tbody>
</table>

Tabelle 9: Aufbau der [ra] Sektion

<table>
<thead>
<tr>
<th>Variable</th>
<th>Bedeutung</th>
</tr>
</thead>
<tbody>
<tr>
<td>formname1</td>
<td>Das verwendete Formular des ersten Antrags</td>
</tr>
<tr>
<td>formname2</td>
<td>Das verwendete Formular des zweiten Antrags</td>
</tr>
<tr>
<td>racfg</td>
<td>Pfad, bzw. Dateiname der RA Konfigurationsdatei</td>
</tr>
</tbody>
</table>

Tabelle 10: Bedeutung der [ra] Variablen

Für jeden der zwei möglichen Anträge an die RA (siehe Kapitel 7.2.5) kann ein eigenes Formular mittels \( \text{formname1} \) und \( \text{formname2} \) angegeben werden. Ob eine, beide oder keine Variablen gesetzt werden müssen, hängt dabei von der [database] Sektion ab und wird im folgenden Kapitel näher erläutert.

Mit Hilfe der Variablen \( \text{racfg} \) wird auf eine Konfigurationsdatei wie die \( \text{ra.ini} \) verwiesen, die dann zur Laufzeit geladen wird. Diese zusätzliche Datei beinhaltet Daten zur Ansteuerung der RA durch den Listener.

7.2.5 Die [database] Sektion

<table>
<thead>
<tr>
<th>Variable</th>
<th>Typ</th>
<th>Case-Sensitive</th>
<th>Werte</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Antragsnr.&gt;</td>
<td>&lt;Formular-Variable&gt;</td>
<td>String</td>
<td>ja</td>
</tr>
</tbody>
</table>

Tabelle 11: Aufbau der [database] Sektion
7.2 Konfiguration des Listeners

<table>
<thead>
<tr>
<th>Variable</th>
<th>Wert</th>
<th>Bedeutung</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Antragsnr.&gt;</td>
<td>&quot;1&quot; oder &quot;2&quot;</td>
<td>Zuordnung zu einem RA-Antrag</td>
</tr>
<tr>
<td>&lt;Formular-Variable&gt;</td>
<td>Text</td>
<td>Name der Formular-Variablen</td>
</tr>
</tbody>
</table>

Tabelle 12: Bedeutung und Werte der [database] Variablen

Der Listener ermöglicht es, bis zu zwei Anträge während eines einzelnen Personalisierungsprozesses zu stellen. Die Zugehörigkeit zu einem Antrag wird dabei durch die Variable <Antragsnr> festgelegt. Die Antragsnummer dient nur zur Unterscheidung innerhalb des Programms, bzw. zur Identifizierung des verwendeten Formulars (siehe Kapitel 7.2.4). Es ist irrelevant, ob bei Verwendung eines einigen Antrages dieser der Nummer "1" oder der Nummer "2" zugeordnet ist. Ebenso muß das erste Schlüsselpaar nicht zwingend Bestandteil des ersten Antrags sein.

Für einen erfolgreichen Antrag muß das entsprechende Formular vollständig ausgefüllt werden. Die einzelnen Felder eines Formulars werden "Formular-Variable" genannt. Der Name einer Formular-Variablen wird durch das Formular definiert und ihr Wert wird zur Laufzeit durch die jeweils vorhandenen Daten aus der Konfigurationsdatei oder durch die übermittelten Daten bestimmt.


Das folgende Beispiel zeigt, wie eine Formular-Variablen "EMAIL" des ersten Antrags aus statischem Text und den übermittelten Feldern %Name% und %Vorname% erstellt werden könnte:

1|EMAIL = %Vorname% . %Name%@tu-darmstadt.de

Neben solchen externen Daten können auch die internen Listener Variablen %ADMIN_PIN%, %USER_PIN%, %SERIAL_NR% und die beiden Schlüsselobjekte %PUBLIC_KEY1% und %PUBLIC_KEY2% verwendet werden (vgl. Tabelle 6). Die beiden öffentlichen Schlüssel werden dabei nicht wie Strings sondern wie Java-Objekte behandelt. Aus diesem Grund kann man einer Formular-Variablen nicht gleichzeitig einen Text und ein Schlüsselobjekt zuweisen. In einem solchen Falle wird der Text ignoriert und der Variablen nur das Schlüsselobjekt zugewiesen.

Eine Unterteilung in zwei Anträge ermöglicht es, Anträgen verschiedene Formen und Inhalte zuzuweisen. Für die Unterscheidung der öffentlichen Schlüssel bieten sich drei Möglichkeiten an.

Im folgenden Beispiel wird anhand der Namen der Formular-Variablen festgelegt, wofür die Schlüssel verwendet werden sollen:

1|SIGN_KEY = %PUBLIC_KEY1%
2|ENCODE_KEY = %PUBLIC_KEY2%

Mittels eines zusätzlichen Feldes "TYPE" könnte eine Differenzierung ebenfalls vorgenommen werden.
Als dritte Möglichkeit könnten auch die verwendeten Formulare festlegen, wofür ein Schlüssel verwendet werden soll. In diesem Fall kann auf eine Typen-Unterscheidung durch den Namen der Formular-Variable oder die Verwendung einer zusätzlichen Variable verzichtet werden:

1 | PUBLIC_KEY = %PUBLIC_KEY1%
2 | PUBLIC_KEY = %PUBLIC_KEY2%

Weitere Beispiele für Konfigurationsmöglichkeiten finden sich in Anhang A.
8 Ausblick

Im Rahmen dieser Studienarbeit haben wir - gemäß der FlexiTRUST Philosophie - eine flexible und sichere Anbindung einer Personalisierungsstraße an die Trustcenter-Software FlexiTRUST entwickelt.

Durch den offenen Rahmen, der durch das verwendete Kommunikationsprotokoll geschaffen wurde, respektive den geringen Funktionsumfang des Callers im Vergleich zum Listener, lassen sich auch andere Personalisierungsstraßen leicht einbinden.

Der Funktionsumfang der hier vorgestellten Caller bmchip.dll und PREPERSO ist allerdings mächtig genug, um einen umfassenden Rahmen für die Personalisierung bereitzustellen zu können. Mit BadgeMaker steht dem Sachbearbeiter eine komfortable graphische Software-Lösung zur Verfügung, die im Falle der kompletten optischen und elektronischen Personalisierung sowohl für die Einzelfertigung, als auch eine automatisierte Massenfertigung eingesetzt werden kann. Der Caller PREPERSO kann auf der einen Seite für die Massenfertigung von vorpersonalisierten Karten eingesetzt werden, auf der anderen Seite bietet dieses Programm einem Administrator die Möglichkeit, einzelne Karten individuell elektronisch zu personalisieren.


A Beispiele für die Konfigurationsmöglichkeiten des Listeners

In diesem Kapitel werden einige Beispielkonfigurationen vorgestellt. Im Falle der Verwendung von PREPERSO wird der jeweilige Programmaufruf mit angegeben.

A.1 Erzeugen von zwei Schlüsselpaaren mit PREPERSO

Aufruf durch PREPERSO:

```bash
preperso -h 130.83.23.1
```

Konfiguration des Listeners:

```bash
[input]
inputport = 2401
inputip = 130.83.23.1

[smartcard]
initialadminpin = 1234
adminpin = 1234
userpin = 1234
chipdll = gclib.dll
secondkeypair = true
keylength1 = 1024
keylength2 = 1024
cardlabel = FlexiTRUST Smartcard
erasecard = true
```

```bash
[output]
return_1 = OK:
return_0 = FAILED:er=%ERROR%
```

A.2 Erstellen eines RA-Antrages mit Hilfe von BadgeMaker

```bash
[input]
inputport = 2401

[smartcard]
initialadminpin = 1234
adminpin = 1234
userpin = 1234
chipdll = gclib.dll
secondkeypair = false
keylength1 = 1024
erasecard = false
```

```bash
[output]
```
A.3 Speichern zweier Zertifikate

ra = enabled
return_1 = OK:fc=1:fn=1:fd=%STATUS%
return_0 = FAILED:fc=1:fn=1:fd=%STATUS%;er=%ERROR%

[ra]
formname1 = BeispielFormular
racfg = ra.ini

[database]
1|CN = %VORNAME% %NAME%
1|OU = %FACHBEREICH%
1|0 = TUD
1|C = DE
1|PUBKEY = %PUBLIC_KEY1%

A.3 Speichern zweier Zertifikate

[input]
inputport = 2401
certificate = 2;d:\certs

[smartcard]
initialadminpin = 1234
adminpin = 4321
userpin = random
chipdll = gclib.dll
secondkeypair = true
keylength1 = 1024
keylength2 = 1024
erasecard = false

[output]
return_1 = OK:fc=2:fn=7,8:fd=%USER_PIN%\%STATUS%
return_0 = FAILED:fc=2:fn=7,8:fd=%USER_PIN%\%STATUS%;er=%ERROR%

A.4 Komplette Personalisierung mit BadgeMaker

[input]
inputport = 2401
inputip = 127.0.0.1
certificate = 2;d:\certs

[smartcard]
initialadminpin = 1234
adminpin = random
userpin = 1234
chipdll = gclib.dll
Beispiele für die Konfigurationsmöglichkeiten des Listeners

```plaintext
secondkeypair = true
keylength1 = 1024
keylength2 = 1024
cardlabel = FlexiTRUST Smartcard
erasecard = true

[output]
ra = enabled
return_1 = OK:fc=1:fn=11:fd=%ADMIN_PIN%
return_0 = FAILED:fc=1:fn=11:fd=%ADMIN_PIN%:er=%ERROR%

[RA]
formname1 = BeispielFormular
formname2 = Formular2
racfg = ra.ini

[Database]
1|CN = %VORNAME%, %NAME%
1|OU = %FACHBEREICH%
1|O  = TUD
1|C  = DE
1|PUBKEY = %PUBLIC_KEY2%

2|SubjectDN = CN=%FIRSTNAME%, %SURNAME%, OU=CDC, O=TUD, C=DE
2|pubKey = %PUBLIC_KEY1%
```
B Quellcode

B.1 bmchip32.h

    //bmchip32.h
    #ifdef _cplusplus
    extern "C" {
    #endif

    typedef struct _BMCHIPINFO
    {
        HAND hWnd;
        CHAR szChipInfoFile[MAX_PATH];
        DWORD dwFlags;
        CHAR szUpdateData[266];
        SHINT nSeparator;
        SHINT nFieldsToUpdate[32];
        SHINT nFields;
        CHAR szLocateData[32];
        CHAR szErrorMessage[64];
        BULL bSecurity;
        UCHAR ucKeySN[8];
        UCHAR ucKeyVal[8];
        CHAR cReserved1[32];
    } BMCHIPINFO; /* INTERFACE BMCHIPINFO;
    /*
    hWnd
    Handle to BadgeMaker window

    szChipInfoFile
    File that contains the data to be encoded.

dwFlags
    Instructs BM to do special tasks. Use the bitwise OR operator to combine tasks.

    BM_UPDATEDB_BEFORE_ENCODE
    Update the database with the data specified in the szUpdateData member
    before encoding. Can not be combined with the BM_UPDATEDB_AFTER_ENCODE flag.

    BM_UPDATEDB_AFTER_ENCODE
    Update the database with the data specified in the szUpdateData member
    after encoding. Can not be combined with the BM_UPDATEDB_BEFORE_ENCODE flag.

    BM_UPDATEDB_IF_NO_ERROR
    Update DB only if no error has occurred

    BM_INSERT_RECORD_BEFORE_ENCODE
    Use the data specified in the szUpdateData member to insert a record in
    the database before encoding.
    Can not be combined with the BM_UPDATEDB_XXXX flag.

    BM_INSERT_RECORD_AFTER_ENCODE
    Use the data specified in the szUpdateData member to insert a record in
    the database after encoding.
    Can not be combined with the BM_UPDATEDB_XXXX flag.

    BM_LOCATE_RECORD
    Use the data specified in the szLocateData member to locate a record in
    the database.

    BM_USERACTION_PRINT
    Indicates that the ChipInit() function was called from within the print
    loop.
BM_USERACTION_ENCODEMANUAL
Indicates that the ChipInit() function was called when the user clicked the 'Chip encode' button.

nSeparator:
Field separator. If you want to update more than one field, use this as field separator.

nFieldsToUpdate
Use this array to specify the indexes of the fields to update.

nFields
Specify the number of fields you want to update.

szLocateData
BM will use the data specified in this member to locate a record.
Note: the data specified here must identify a record in the database uniquely.

szErrorMessage
If anything goes wrong, you can specify here an error message. This message will be displayed by BM.

ucKeySN
Serial number of key. Every byte represents one hex digit.
This member will contain a legal value when called from the ChipInit() function and if the bSecurity member is set to TRUE.
+-----------------------------+
| O | O | E | F | A | 1 | B | F |
+-----------------------------+
| | | | | | | | |
+-----------------------------+
MSD | LSD

ucKeyVal
Special value stored in the key. Every byte represents one hex digit.
This member will contain a legal value when called from the ChipInit() function and if the bSecurity member is set to TRUE.
+-----------------------------+
| O | 1 | C | 2 | A | 3 | B | 2 |
+-----------------------------+
| | | | | | | | |
+-----------------------------+
MSD | LSD

cReserved1
Reserved for future use.
*/

#define DLLEXPORT __declspec(dllexport)

 tackleddllexport

//You can combine this flag with BM_UPDATEDB_BEFORE_ENCODE
//or with BM_UPDATEDB_AFTER_ENCODE
//If this flag is set, BM will check the return code of
//ChipInit () and ChipEncode() function,
//only if it is 0 (zero) then it will update the database,
// otherwise no check will be performed.
#define BM_UPDATED_DB_IF_NO_ERROR Ox00000000

//--------------------------------------------------------------------------------------------------
// Use these flags only in the encode routine
// e.g ChipEncodeEx()
#define BM_UPDATED_DB_AFTER_ENCODE Ox00000001
#define BM_INSERT_RECORD_AFTER_ENCODE Ox00000002

//--------------------------------------------------------------------------------------------------

// BM will set one of these flags before calling the ChipInit() function
// from the Chip DLL
#define BM_USERACTION_PRINT Ox00000000
#define BM_USERACTION_ENC0DEMANUAL Ox00000002

DLLEXPORT int ChipEncodeEx(LPCCHIPINFO);
DLLEXPORT int ChipInit(LPCCHIPINFO);

#ifdef __cplusplus
} }
#endif
C Literaturverzeichnis

Literatur


