Projektdokumentation im Modul Smartcard

Patientendateninformationskarte im medizinischen Sektor

Robert Kupferschmied B.Sc., Roy Meissner B.Sc.,
Sebastian Krause B.Sc.

July 11, 2014

1 Anwendungsbeschreibung

Ziel der Anwendung ist es eine Smartcard mit Patientendaten zur Verfügung zu stellen, die im medizinischem Sektor Abläufe beschleunigt bzw. Automatisierung zulässt. Zu diesem Zweck stellt die Smartcard folgende Informationen bzgl. eines Patienten bereit:

- eine Patienten-ID
- die Patientenblutgruppe
- eine Liste nicht verträglicher Medikamente
- eine List regelmäßig einzunehmender Medikamente, incl. Dosierungsinformationen

Die Patientendaten auf der Smartcard können von verschiedenen Rollen (Personen) ausgelesen werden, die jeweils verschiedene Rechte besitzen. Beispielsweise darf ein Pfleger nur die Liste regelmäßig einzunehmender Medikamente auslesen, der Hausarzt darf hingegen alle Daten auslesen.

Um die Datenübertragung zwischen Lesegerät bzw. Offcard-Software und Smartcard abzusichern, wird die Kommunikation kryptographisch gesichert. Zu diesem Zweck wird

per RSA ein DES-Schlüssel ausgetauscht, der im folgenden den gesamten Datenverkehr verschlüsselt.

Als Anwendungsfälle könnten gelten:

Notarzt Im Falle eines Unfalls muss der Notarzt schnell auf Informationen wie Blutgruppe, nicht verträgliche Medikamente und aktuell verschriebene Medikamente zugreifen können, auch wenn der Patient bewusstlos ist.

Automatisierte Medikamentenausgabe Über die Liste regelmäßig einzunehmender Medikamente, inklusive Dosierungsinformationen, könnten dem Patienten an einem Automaten Medikamente aushändigt werden.

1.1 Aufgabenverteilung

Das gesamte Projekt wurde innerhalb der vergebenen Aufgabenbereiche zunächst separat entwickelt und am Ende zusammengeführt. Die Verantwortlichkeiten sind dabei der folgenden Tabelle zu entnehmen, haben sich bei der Zusammenführung jedoch nochmals überschnitten.

Name	Aufgabenbereich
Krause, Sebastian	Offcard-Anwendung
Kupferschmied, Robert	Verschlüsselung/Kryptografie
Meissner, Roy	Oncard-Anwendung

Die gesamte Dokumentation, der Code und eine Änderungshistorie kann auf Github gefunden werden. Das zugehörige Repository ist unter `https://github.com/neonlzf/smartcard` zu finden.

2 OffCard

Dieser Abschnitt beschreibt die OffCard-Anwendung und geht auf die zugrunde liegende Architektur ein.

2.1 Beschreibung der Offcard-Anwendung

Abbildung 1 zeigt das Hauptfenster der Offcard-Anwendung. Über die Schaltflächen "Load" und "Save" werden die Daten von der SmartCard gelesen bzw. auf die SmartCard geschrieben. Über das im oberen linken Teil des Fensters befindliche Drop-Down-Menü kann der jeweilige Akteur gewählt werden. So hat beispielsweise der Hausarzt vollen Zugriff, während der Pfleger nur Daten lesen kann.

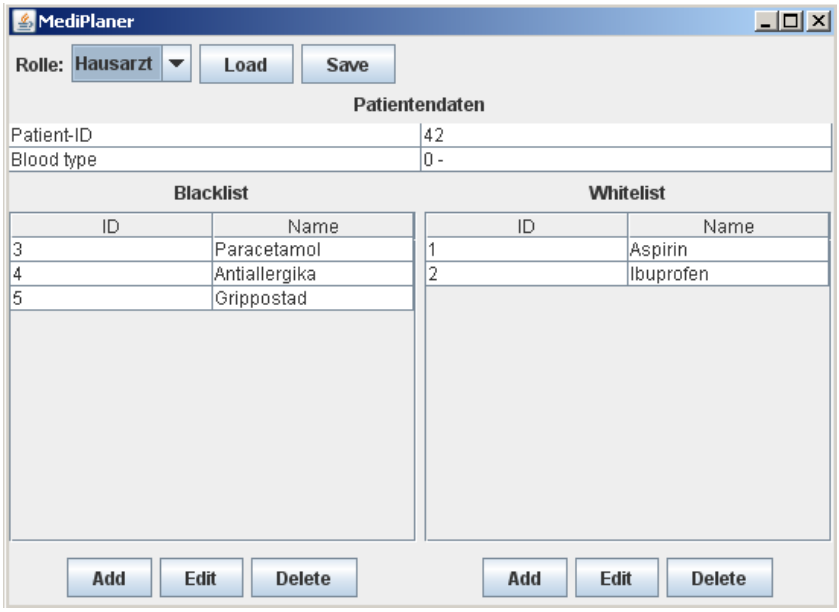

Figure 1: Hauptfenster der Offcard-Anwendung

Im Bereich Patientendaten sind die Stammdaten des Patienten zu sehen. Dabei handelt es sich um seine ID, zb. die Krankenversicherungsnummer, und seine Blutgruppe. Durch die Speicherung einer eindeutigen ID, ist es möglich patientenbezogene Daten von Online-Diensten anzufordern und in die Anwendung einzubinden. Die Blutgruppe wird darüber hinaus als wichtige Information bereitgehalten.

Darunter sind die Medikamenten-Whitelist und -Blacklist angeordnet. Diese können über Markierung und die Schaltflächen "Add", "Edit" und "Delete" bearbeitet werden.

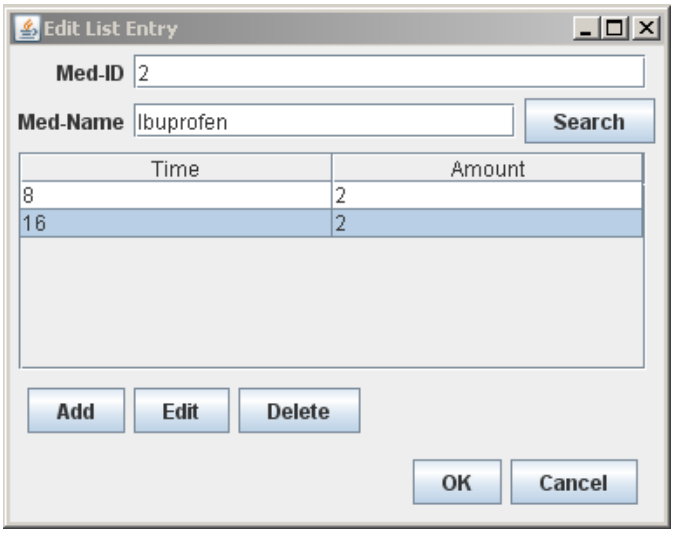

Figure 2: Fenster für Medikamenteneditierung der Offcard-Anwendung

Die Abbildung 2 zeigt das Fenster zur Editierung eines Medikamentenlisteneintrages. Durch Eingabe eines Teilstrings im Feld "Med-Name" und anschließender Betätigung der "Search"-Schaltfläche wird aus den Medikamentenstammdaten der vollständige Name und die entsprechende Medikamenten-ID geladen. Diese Stammdaten werden in einer XML-Datei vorgehalten, welche zu Programmstart geladen wird.

Des Weiteren enthält das Medikamenteneditierungsfenster eine Liste der vom Arzt vorgeschriebenen Dosierungen, bestehend aus Uhrzeit in vollen Stunden und Einheiten. Diese kommt jedoch nur bei der Whitelist, der Liste einzunehmender Medikamente, zum Einsatz.

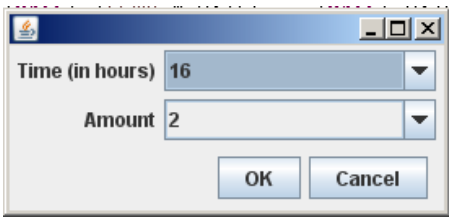

Figure 3: Fenster zur Festlegung der Dosierung

Abbildung 3 stellt das Menü zur Bearbeitung eines Dosierungseintrages dar. Hier können Uhrzeit und Anzahl dieses Eintrages festgelegt werden.

2.2 Architektur

Im Bild 4 ist das Klassendiagramm der Offcard-Anwendung zu sehen. Das Projekt ist dabei in die Pakete "card", "data" und "gui" unterteilt.

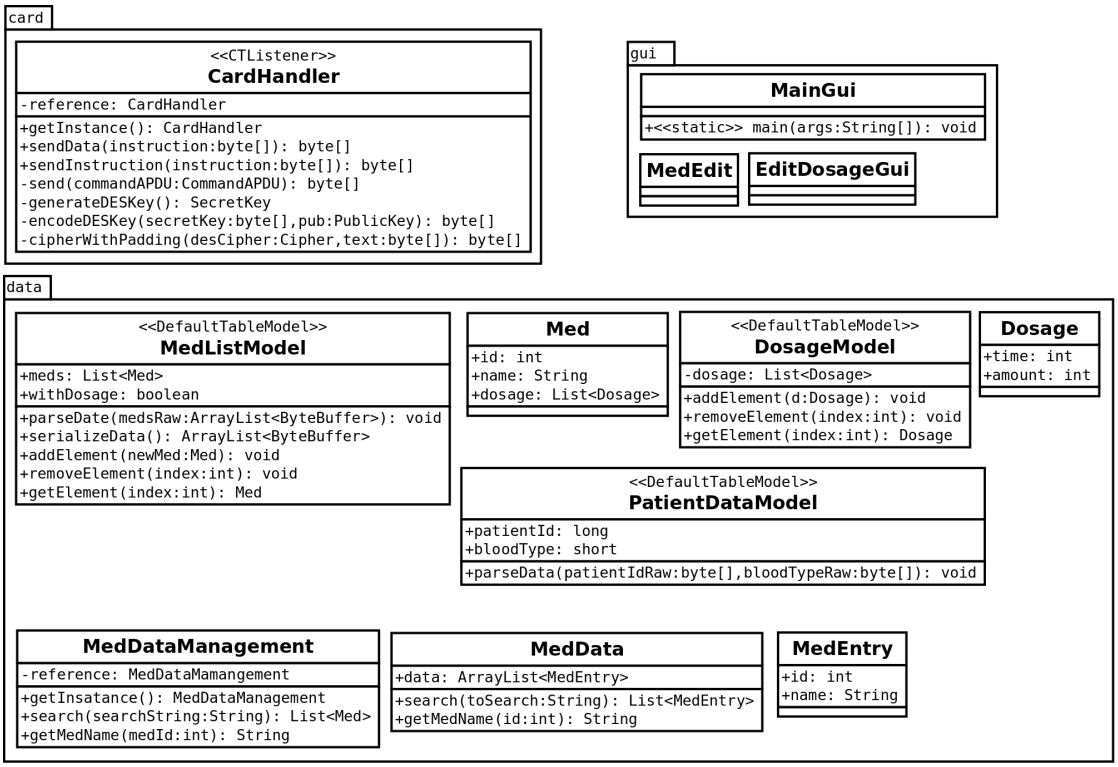

Figure 4: Klassendiagramm der Offcard-Anwendung

Im "card"-Paket befindet sich die Klasse "CardHandler". Diese kapselt den vollständigen Zugriff zur SmartCard und stellt darüber hinaus Methoden zur Verschlüsselung bereit. Realisiert ist die Klasse dabei als Singleton, um sicherzustellen, dass nur eine Instanz für die gesamte Anwendung instantiiert wird.

Das Paket "gui" enthält die drei Klassen für die grafische Benutzeroberfläche. Darüber hinaus stellt "MainGui" die main-Methode zum Programmeinstieg bereit.

Im Paket "data" liegen alle Klassen zur anwendungsseitigen Datenhaltung. Die Klasse "PatientDataModel" enthält die von der SmartCard gelesenen Stammdaten des Patienten. Die Klassen "MedListModel", "Med", "DosageModel" und "Dosage" kapseln die

Daten der Medikamentenlisten. Die Model-Klassen erweitern "DefaultTableModel" um die Darstellung in einer Tabelle zu gewährleisten. Darüber hinaus enthalten "PatientDataModel" und "MedListModel" die Methoden "parseData()" und "serializeData()". Diese Klassen konvertieren die von der Karte gelesenen Byte-Arrays in von der Anwendung verarbeitbare Daten und zurück.

Zur Verwaltung der Medikamentenstammdaten dienen die Klassen "MedDataManagement", "MedData" und "MedEntry". Die eigentlichen Daten bilden dabei "MedData" und "MedEntry", wobei "MedDataManagement" zur Deserialisierung dieser aus einer XML-Datei dient. Dies wird über das JAXB-Framework realisiert.

3 OnCard

Die OnCard-Anwendung bietet eine vollständig definierte Schnittstelle in Form einer APDU-Beschreibung[1] an. Über diese Schnittstelle kann mit den Daten, die auf der Smartcard gespeichert sind, interagiert werden.

Die Patienten-ID wird zur Installation des Applets auf der Karte fest kodiert. Die ID ist eine Zahl vom Typ long, die auf der Smartcard in Form von 8 Bytes abgespeichert wird. Long wurde gewählt um jedem Menschen eine solche Karte aushändigen zu können, ohne das die IDs ausgehen. Die ID kann über die Schnittstelle nur abgerufen, jedoch nicht verändert werden.

Ein beispielhafter Abruf der Patienten-ID sieht wie folgt aus:

```
1   \send 00 08 00 00
2   Response:
3   00 00 00 00 00 00 00 01 90 00 //Patienten-ID + No-Error-Code
```

Die Blutgruppe des Patienten wird, wie die Patienten-ID, zur Installation des Applets festgelegt. Sie ist als ein Byte codiert und kann anhand einer Kodierungstabelle[2] übersetzt werden. Wie die Patienten-ID ist auch die Blutgruppe nicht manipulierbar.

[1] in der Datei APDUs.txt zu finden
[2] diese ist in der Datei APDUs.txt zu finden

Ein beispielhafter Abruf der Patienten Blutgruppe sieht wie folgt aus:

```
\send 00 07 00 00
Response:
04 90 00 //Bloodtype + No-Error-Code
```

Auf der Smartcard wird eine Liste nicht verträglicher Medikamente gespeichert. Diese ist als eine List von jeweils 4 Byte IDs angelegt. Diese können in der OffCard-Anwendung in ein Int umgewandelt und über eine Kodierungstabelle dem entsprechenden Medikament zugeordnet werden. Die Smartcard bietet an, die Liste Elementweise auszulesen, neue Medikamente zur Liste hinzuzufügen, als auch die Liste zu verwerfen.

Ein beispielhaftes Hinzufügen eines Medikaments zur Liste sieht wie folgt aus:

```
\send 00 02 00 00 04 00 00 42 83 //Add Element with ID 00 00 42 83
Response:
Success: 90 00
Failure: 6A 83
```

Ein beispielhafter Abruf eines Medikaments der Liste sieht wie folgt aus:

```
\send 00 01 00 09 //Fetch Element 9
Response:
Success: 00 00 42 83 90 00 //Drug-ID + No-Error-Code
Failure: 6A 83
```

Ein beispielhaftes Löschen der Liste sieht wie folgt aus:

```
\send 00 03 00 00
Response:
90 00
```

Die Liste der regelmäßig einzunehmenden Medikamente ist gleich der Liste nicht verträglicher Medikamente aufgebaut, einzig die Eintragslänge entspricht im Gegensatz 28 Byte. Die ersten 4 Byte entsprechen der Medikamenten-ID, die folgenden 24 Byte den Einnahmezeiträumen sowie der Einnahmemenge. Diese Liste kann über die gleichen Methoden wie die Liste der nicht verträglichen Medikamente ausgelesen bzw. manipuliert wer-

den.

Intern werden die Listen über eine, eigens für diesen Zweck entwickelte, Datenstruktur organisiert. Diese bietet ein Ressourcenschonendes List-Interface für das kontrollierte Hinzufügen und Auslesen der Listenelemente, als auch ein verwerfen der gesamten Liste.

Alle zwischen der OnCard- und der OffCard-Anwendung übertragenen Daten werden verschlüsselt. Zu diesem Zweck bietet der OnCard zwei Funktionen, um den Public RSA-Key Exponenten und Modulus exportieren zu können. Aus diesen kann sich die OffCard-Anwendung den Public-RSA-Key der SmartCard errechnen und mit diesem den DES-Key verschlüsseln. Dieser wird zur SmartCard übertragen und für die Lebenszeit der Session gespeichert. Mit diesem DES-Key wird wiederum jedwede Datenübertragung gesichert, egal ob ankommende Daten oder ausgehende Daten.

4 Kryptologie

In diesem Kapitel werden die kryptografischen Methoden des Projektes erklärt. Zum Verschlüsseln der Daten wird ein symmetrisches Verfahren verwendet. Symmetrische Verschlüsselungsverfahren sind sehr effizient und sicher. Leider werden bei symmetrischen Verfahren nur ein Schlüssel zum Ver- und Entschlüsseln verwendet. Deswegen ist der Schlüsselaustausch sehr schwierig. Asymmetrische Verfahren können dieses Problem lösen.

4.1 Asymmetrische Verschlüsselung - RSA

Zur asymmetrischen Verschlüsselung stehen auf den Smartcards prinzipiell zwei Algorithmen zur Verfügung. Zum einen der “elliptic Curve”-Algorithmus und der RSA[3]-Algorithmus. Die Sicherheit des RSA-Algorithmus beruht auf dem Problem der Primzahlzerlegung. Das Problem der Primzahlzerlegung ist ein schwieriges Problem, da kein effizienter Algorithmus bekannt ist der das Problem lösen könnte. Weiterhin ist RSA der bekannteste und am weitesten verbreitete Algorithmus zur asymmetrischen Ver-

[3]Rivest, Shamir und Adleman

schlüsselung. Um zu möglichst vielen Systemen kompatibel zu sein, wird der RSA-Algorithmus in diesem Projekt verwendet.

Bei asymmetrischen Verschlüsselungsverfahren werden Schlüsselpaare für die Verschlüsselung benutzt. Diese Schlüsselpaare bestehen aus einem Public-Key und einem Private-Key. Der Public-Key kann publiziert werden. Den Private-Key muss man geheim halten.

Die Schlüssel setzen sich aus einem Exponenten und einem Modulus zusammen. Der Modulus ist eine sehr große Zahl, bei RSA-1024 eine Zahl im Bereich von 2^{1024} und setzt sich aus zwei Primzahlen zusammen. Der Modulus ist im Public- und Private-Key gleich. Die Exponenten unterscheiden sich allerdings.

Zur Ver- und Entschlüsselung werden folgende Funktionen benutzt:

$$G = K^e \mod n$$

$$K = G^d \mod n$$

G ... Geheimtext
K ... Klartext
e ... öffentlicher Exponent
d ... geheimer Exponent
n ... Modulus

Die Schlüssel werden in der Datei "irgendwas.java" erzeugt. Hier sind diese Zeilen wichtig:

```
1       Cipher rsaCipher = Cipher.getInstance(Cipher.ALG_RSA_PKCS1,false);
2       KeyPair keyPair = new KeyPair(KeyPair.ALG_RSA_CRT, (short)1024);
3       keyPair.genKeyPair();
4       RSAPrivateCrtKey rsa_privateKey = (RSAPrivateCrtKey) keyPair.getPrivate();
5       RSAPublicKey rsa_publicKey = (RSAPublicKey) keyPair.getPublic();
```

Die JCOP Shell hat beim Erstellen der Schlüssel leider ein paar Fehler. Hier kann beim Erzeugen des öffentlichen Schlüssels kein echter öffentlicher Exponent erzeugt werden, stattdessen wird nur eine konstante binäre 17 zurückgegeben.

Auf einer echten Karte dürfte dieser Fehler nicht auftreten.

Die Verschlüsselung mit asymmetrischen Verfahren ist nicht so effizient, wie die Verschlüsselung mit symmetrischen Verfahren. Deswegen ist es sinnvoll die symmetrische und asymmetrische Verfahren zu kombinieren um die Vorteile aufzuwiegen. Asymmetrische Verfahren werden dazu verwendet um den Schlüssel der symmetrischen Verfahren auszutauschen.

4.2 Symetrische Verschlüsselung - DES, AES

Für die symmetrische Verschlüsselung ist nur ein Schlüssel für die Ver- und Entschlüsselung notwendig. Dieser wird von dem Off-Card Programm erzeugt und via RSA-Verschlüsselung an die Karte gesendet. Als Verschlüsselungsalgorithmus wird DES benutzt. Dieser besitzt eine Schlüssellänge von 56-Bit. DES gilt nicht mehr als sicher da die Schlüssellänge zu kurz ist. Normalerweise nutzt man bei der modernen Verschlüsselung AES. Dieser gilt als mathematisch sicher und man kann ihn nur angreifen indem man alle möglichen Schlüssel durchprobiert. AES besitzt einen Schlüsselraum von 128 Bit und kann auf 256 Bit erweitert werden.

AES wird nicht von der Simulationsumgebung JCOP unterstützt, deswegen muss man DES einsetzen. Im Betrieb mit einer echten Karte sollte man allerdings AES verwenden.

Sowohl AES als auch DES verschlüsseln die Daten Blockweise. Deswegen gibt es verschiedene Modi beim Verschlüsseln. Hier kann man folgende Modi einstellen:

```
1       Cipher.getInstance(Cipher.ALG_DES_ECB_NOPAD,false);
2       Cipher.getInstance(Cipher.ALG_DES_ECB_PKCS5,false);
```

Das Cipher-Objekt spricht den Kryptologie-Prozessor der Smartcard an. Der Algorithmus ist DES. ECB steht für Electronic Code Book, dies ist besonderer Sicherheitsmodus damit ähnliche Daten nicht erkannt werden. NOPAD steht für No Padding, hier können nur Daten verschlüsselt werden deren Länge ein vielfaches der Blocklänge entspricht. Im Modus PKCS5 wird der Klartext um die fehlenden Bytes ergänzt.

5 OpenCard Framework - OCF

Das OpenCard Framework ist eine Smartcard Middleware. Hiermit kann man mit Smartcards kommunizieren. Das OpenCard Framework ist komplett in Java programmiert und damit Betriebssystemunabhängig.

Um mit einer Smartcard kommunizieren zu können, muss man zunächst die Verbindung aufbauen. Dafür ist folgende Methoden ausführen:

```
1    CardRequest cardRequest = new CardRequest(CardRequest.ANYCARD, null,
         PassThruCardService.class);
2    cardRequest.setTimeout(1);
3    card = SmartCard.waitForCard(cardRequest);
```

Nun kann man über die "card" Variable APDU's an die Smartcard senden. Hierfür muss man folgende Methoden senden:

```
1    CommandAPDU commandAPDU = new CommandAPDU(instruction.length);
2    commandAPDU.setLength(instruction.length);
3    System.arraycopy(instruction, 0, commandAPDU.getBuffer(), 0, instruction.length);
4    PassThruCardService passThru = (PassThruCardService)
         card.getCardService(PassThruCardService.class, true);
5    ResponseAPDU responseAPDU1 = passThru.sendCommandAPDU(commandAPDU);
```

Zunächst wird eine APDU angelegt, danach kann man sie mit Bytes beschreiben. Über den Befehl "sendCommandAPDU" wird die APDU an die Smartcard gesendet und als Rückgabewert wird die empfangene APDU zurückgeben.

6 Schlüsselaustausch mit OCF

Die Abbildung 5 visualisiert den Ablauf.

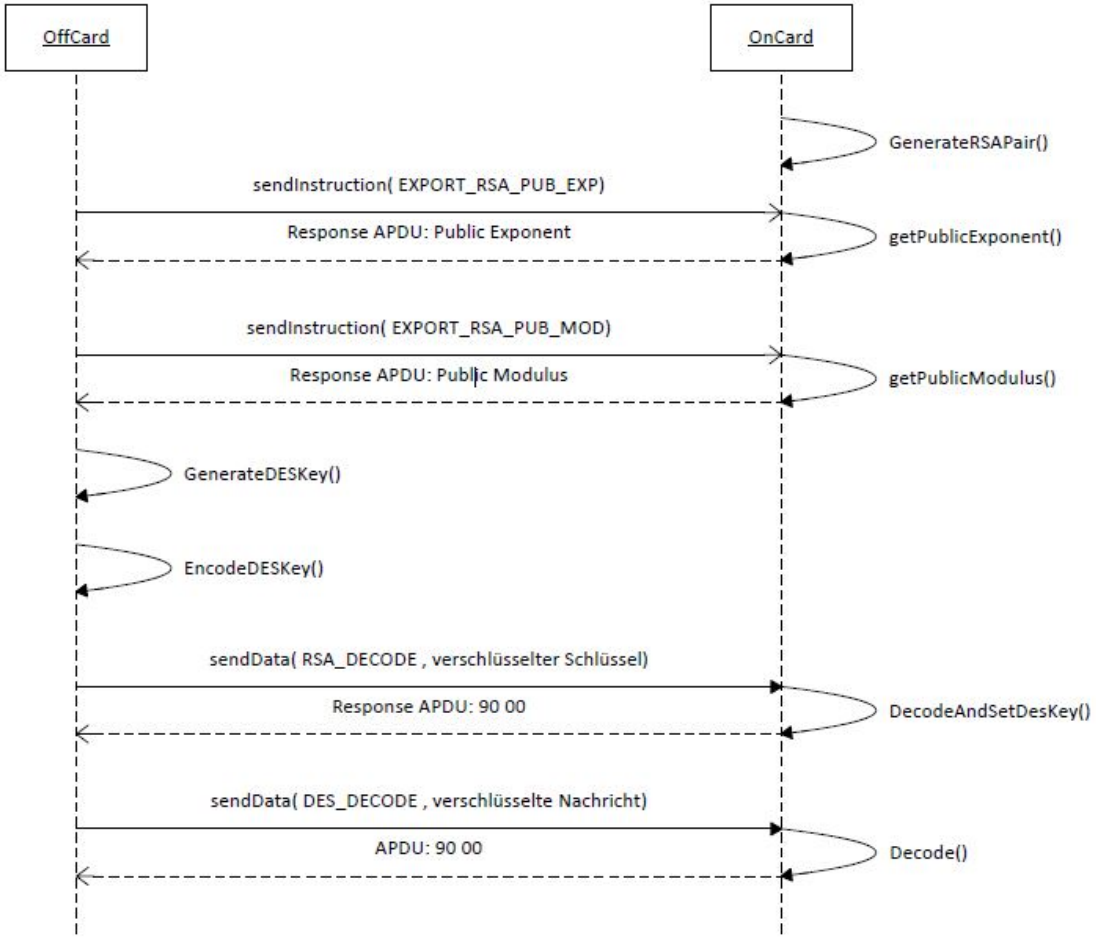

Figure 5: Schlüsselaustausch

Funktion	befindet sich in Datei
sendInstruction( )	CardHandler.java
sendData( )	CardHandler.java
GenerateRSAPair()	Patient.java
getPublicExponent()	Patient.java
getPublicModulus()	Patient.java
GenerateDESKey()	CardHandler.java
EncodeDESKey()	CardHandler.java
DecodeAndSetDesKey()	Patient.java
Decode()	Patient.java

Beim installieren der Smartcard-Applikation wird ein RSA Schlüsselpaar erzeugt. Dies geschieht über die in Kapitel 4.1. Über die definierten Kommando APDU's "EXPORT_RSA_PUB_EXP" und "EXPORT_RSA_PUB_MOD" kann man sich den Public-Key der Karte ausgeben lassen. Der Exponent und Modulus könne in Java durch die BigInteger-Klasse zu einem RSA-Key zusammengesetzt werden. Dann wird der DES-Key von der OffCard-Anwendung erzeugt. Diesen kann man nun mit RSA verschlüsseln und an die Karte senden. In der Karte wird der Schlüssl mit dem privaten RSA-Schlüssel entschlüsselt. Nun ist der DES-Schlüssel ausgetauscht und man kann effizient symmetrisch verschlüsseln. Um mit DES ver- und entschlüsseln zu können ist es wichtig die richtigen Modi einzustellen. Die virtuelle Karte von JCOP unterstützt nicht alle Modi. Wenn man den CBC[4]-Modus einstellt entschlüsselt die virtuelle Karte nicht korrekt. Dies kann jedoch auch auf eine Inkompatibilität zwischen dem Ver- und Entschlüsselungsalgorithmus von Java und JavaCard hinweisen. Wenn man den Modus ECB[5] einstellt wird korrekt entschlüsselt.

[4]Cipher Block Chaining
[5]Eletronic Code Book