Projektdokumentation im Modul Smartcard

Patientendateninformationskarte im medizinischen Sektor

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

8. Juli 2014

1 OffCard

2 OnCard

3 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.

3.1 Asymmetrische Verschlüsselung - RSA

Zur asymmetrischen Verschlüsselung stehen auf den Smartcards prinzipiell zwei Algorithmen zur Verfügung. Zum einen der „elliptic Curve“-Algorithmus und der RSA[1]-

[1] Rivest, Shamir und Adleman

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 Verschlüsselung. 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. 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:

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

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

Legende:
G .. Geheimtext
K .. Klartext
e .. öffentlicher Exponent
d .. geheimer Exponent
n .. Modulus
Die Schlüssel werden im der Datei „irgendwas.java“ erzeugt. Hier sind diese Zeilen wichtig:

```
Cipher rsaCipher = Cipher.getInstance(Cipher.ALG_RSA_PKCS1,false);
KeyPair keyPair = new KeyPair(KeyPair.ALG_RSA_CRT, (short)1024);
keyPair.genKeyPair();
RSAPrivateCrtKey rsa_privateKey = (RSAPrivateCrtKey) keyPair.getPrivate();
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.

3.2 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:

```
Cipher.getInstance(Cipher.ALG_DES_CBC_NOPAD,false);
Cipher.getInstance(Cipher.ALG_DES_CBC_PKCS5,false);
```

Das Cipher-Objekt spricht den Kryptologie-Prozessor der Smartcard an. Der Algorithmus ist DES. CBC steht für Cipher Block Chaining, 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

4 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 einfach folgende Methoden ausführen:

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

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

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

Zunächst wird eine APDU angelegt, danach kann man sie mit Bytes beschreiben. Über den Befehl „sendCommandAPDU“ wird die D