D07-QM Technische Spezifikation der Open-Source-Basisbibliothek

Editor: AGETO

Prüfung: Sven Wohlgemuth (TU Darmstadt/CASED)

Typ: [TECHNISCHER BERICHT]

Projekt: „PersoApp“

Version: 1.0

Datum: 30. September 2013

Status: [FREIGABE]

Klasse: [ÖFFENTLICHKEIT]

Datei: D07-QM_Technische_Spezifikation_der_Open-Source-Basisbibliothek.doc

Zusammenfassung

Das vorliegende Dokument D07-QM beschreibt die technische Umsetzung der Kernkomponenten der Basisbibliothek der PersoApp. Dies erfolgt aufbauend auf den Architekturbetrachtungen in D06-QM.

Konsortialleitung:
Prof. Dr. Ahmad-Reza Sadeghi und Dr. Sven Wohlgemuth
System Security Lab, TU Darmstadt/CASED, Mornewegstr. 32, 64293 Darmstadt
Tel.: +49-6151-16-75561 Fax: +49-6151-16-7215
E-Mail: persoapp@trust.cased.de Web: http://www.persoapp.de

Nutzungslizenz

Mitglieder des Konsortiums

1. **AGETO Service GmbH**, Deutschland
2. **Center for Advanced Security Research Darmstadt (CASED)**, Deutschland
3. **Fraunhofer Institut für Sichere Informationstechnologie (SIT)**, Deutschland
4. **Technische Universität (TU) Darmstadt**, Deutschland

Versionen

Version	*Datum*	*Beschreibung (Editor)*
0.8	2013-09-27	Initiale Version
0.9	2013-09-30	Reviewkommentare
1.0	2013-09-30	Finale Version (Reviewkommentare eingearbeitet)

Autoren

Autoren	*Beiträge*
AGETO	Initiale Version des Dokumentes erstellt

[1] http://creativecommons.org/licenses/by-sa/3.0/

Inhaltsverzeichnis

1 Ziel des Dokumentes

Ziel dieses Dokumentes ist die Ergänzung des Architekturkonzeptes (D06) um detaillierte technische Beschreibungen der Haupt-Basiskomponenten (siehe untenstehende Auflistung).

2 Anwendungsbereich

Das Dokument „D07-QM – Technische Spezifikation der Open-Source-Basisbibliothek“ ist für die Software-Entwicklung und das Release Management bestimmt. Es soll aktuellen und künftigen Entwicklern Aufschluss über den Aufbau und die Struktur der Software geben. Dies ermöglicht es, einen schnellen Einstieg in die Entwicklung zu erhalten.

3 Abkürzungen und Begriffsdefinitionen

3.1 Abkürzungen

Abkürzung	*Erläuterung*
API	Application Programming Interface
BSI	Bundesamt für Sicherheit in der Informationstechnik
CASED	Center for Advanced Security Research Darmstadt
CC	Creative Commons
CRM	Customer Relationship Management
eID	Elektronische IDentität
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HCI	Human Computing Interface
HTTP	Hypertext Transfer Protocol
IFD	Interface Device
iOS	Apple iOS (iPhone OS)
JDK	Java Development Kit
JRE	Java Runtime Environment
nPA	neuer Personalausweis
OS	Operating System
P3P	Platform for Privacy Preferences Project
PACE	Password Authenticated Connection Establishment (Protokoll)
PAOS	Reversed HTTP Binding for SOAP

PC/SC	Personal Computer/Smart Card
PIN	Persönliche Identifikations-Nummer
PKI	Public-Key-Infrastruktur
QM	Qualitätsmanagement
SAL	Service Access Layer
SIT	Sichere Informationstechnologie (s. Fraunhofer SIT)
SOAP	Simple Object Access Protocol
TLS	Transport Layer Security
TR	Technische Richtlinie
TU	Technische Universität
WSDL	WebServices Description Language
XSD	XML Schema Definition
XML	eXtensible Markup Language

3.2 Begriffsdefinitionen

Begriff	Definition
Open-Source-Core	Open-Source-Software-Bibliothek mit den Modulen zur Implementierung der Online-Ausweisfunktion des neuen Personalausweises nach den Technischen Richtlinien des BSI

4 Zuständigkeiten und Verantwortlichkeiten

Das Dokument wurde von dem Projektpartner AGETO Service GmbH entwickelt.

Hinweis: Wesentliche Teile des technischen Konzeptes bauen auf den Technischen Richtlinien des BSI, insbesondere auf BSI TR-03112 [21], auf. Es zeichnet sich ab, dass Teile dieser Richtlinie in der nächsten Zukunft auf die TRs 03116-4 und 03124-1 aufgeteilt werden. Durch diese Änderungen können daher Anpassungen an der hier beschriebenen Architektur notwendig werden.

5 Technische Spezifikation

5.1 Überblick

Im Dokument D06-QM [1] wurde das Architekturkonzept behandelt, das die Grobstruktur und Funktion der Komponenten, sowie den modularen Aufbau der PersoApp beschreibt. Das Architekturkonzept dient als Ausgangspunkt für eine weiterführende technische Spezifikation, die Inhalt dieses Dokuments (D07-QM) ist. Gleichzeitig ist damit die Abgrenzung zu D06-QM gegeben.

Analog zu D06-QM sind die technischen Richtlinien des BSI grundlegend, insbesondere TR 03112 [2] (bzw. TR 03116-4 und TR 03124-1, siehe [3, 4]). Zusammen mit

ISO 24727 [5] bilden sie die technische Basis für die Umsetzung eines eID-Clients basierend auf der PersoApp-Open-Source-Basisbibliothek.

Der Aufbau des weiteren Dokumentes orientiert sich im wesentlichen an der Gliederung aus D06-QM (vgl. Abb. 1), d.h.

1. **Network-Layer:** Webservice-Container, PAOS-Client, TLS 1.1+, lokaler HTTP-Server
2. **ISO-24727-Services:** SAL-Service, IFD-Service, eCard-API (vgl. auch TR 03112-1, v.1.1.2)
3. **Card-Services:** eCardHandler, PACE, Transport-Provider, PC/SC
4. **(MISC:** GUI, Crypto-Algs., Configuration) - siehe D06-QM

Die Applikation ist auf zwei Projekte aufgeteilt, dem Core-Projekt und dem Desktop-Projekt. Einstiegspunkt in die Applikation ist die Main-Methode der Klasse "`PersoApp.java`", die im Desktop-Projekt zu finden ist. Dort werden die Hauptkomponenten der verschiedenen Layer innerhalb eines eigenen Threads initialisiert und gestartet. Die folgende Liste gibt die wichtigsten Bestandteile dieses Ablaufes wieder:

- Inititalisierung und Start des HTTP-Servers und des ECApiHttpHandler zur Verarbeitung des lokalen Requests ("alternativer Aufruf")
- Initialisierung des ECardHandlers
- Initialisierung des Webservice-Containers bzw. Webservice-Kontextes `wsCtx`, sowie der einzelnen ISO-24727 Services (SAL-Service, IFD-Service, Protokoll-Service, etc.)
- Erzeugen des "MainView"

Nach der Initialisierung ist die Applikation bereit und es kann über den "alternativen Aufruf" eine eID-Session initiiert werden. Eine Beispiel-URL für diese Aufruf ist:

```
    http://127.0.0.1:24727/eID-
Client?tcTokenURL=https://eid.services.ageto.net/persoapp/eidtest.jsp
```

In den folgenden Abschnitten gehen wir näher auf die einzelnen Komponenten und ihrer Abhängigkeiten ein.

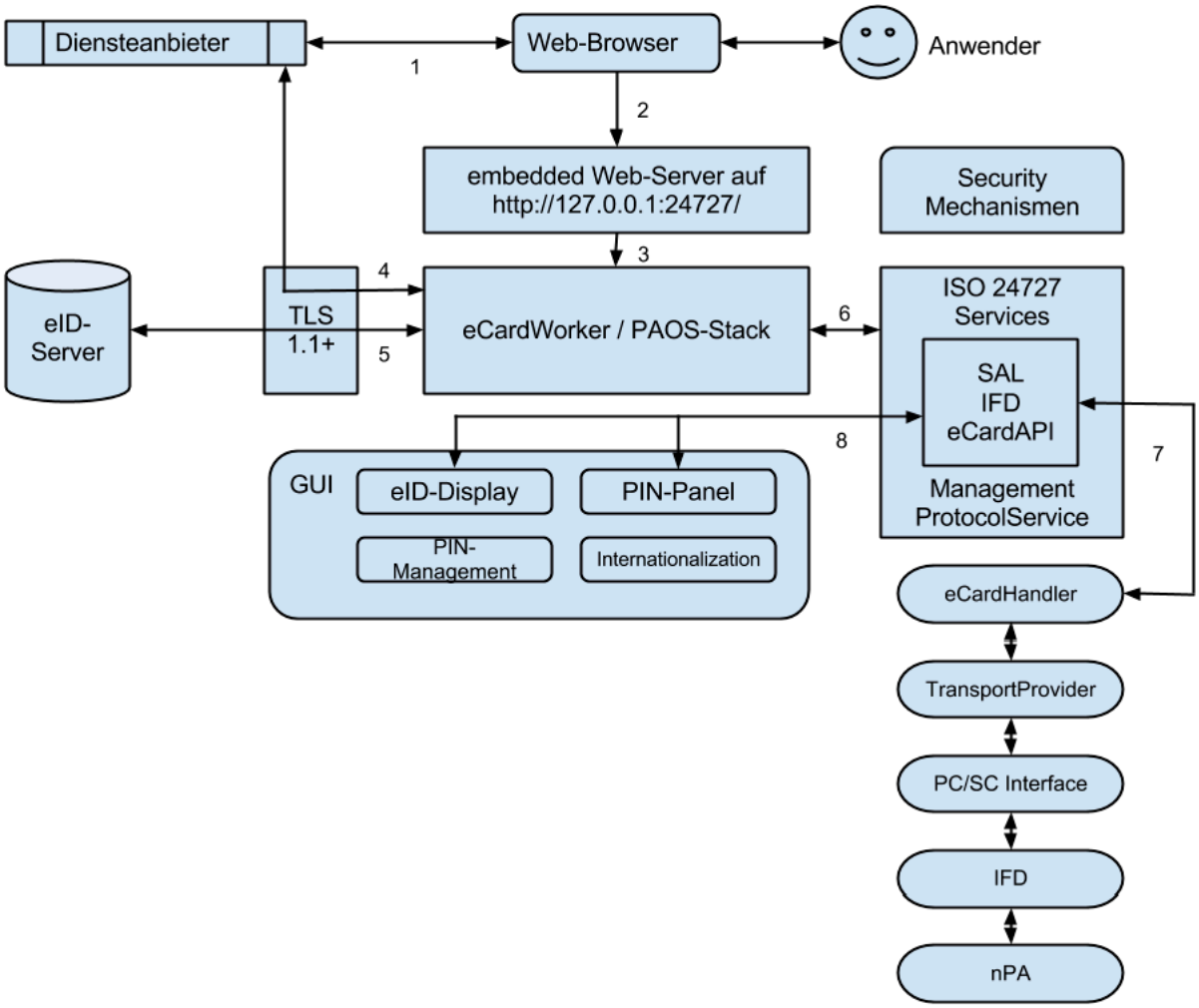

Abbildung 1: Module im Kern der PersoApp und Außenkomponenten

5.2 Network-Layer

5.2.1 Der lokale HTTP-Server und ECApiHttpHandler

Der lokale HTTP-Server ist eine HTTP 1.1 konformer Server. Er wird im Fall der PersoApp durch Standardkomponenten aus Java 1.6 realisiert, genauer durch

```
final HttpServer server = hsp.createHttpServer(
            new InetSocketAddress("127.0.0.1", "24727"), 10
      );
```

Dabei ist `hsp` eine Instanz der Klasse

```
sun.net.httpserver.DefaultHttpServerProvider
```

Die Aufgabe des HTTP-Servers betrifft den "alternativen Aufruf" der eID-Funktion. Zur eigentlichen Umsetzung dieses Aufrufmechanismus muss ein entsprechender HTTP-Kontext erstellt werden, der dann mit dem eCardApiHandler assoziiert wird.

Dies erfolgt durch

```
server.createContext("eID-Client").setHandler(
            new ECApiHttpHandler()
      );
```

Die Klasse ECApiHttpHandler implementiert demnach das Interface

```
com.sun.net.httpserver.HttpHandler
```

d.h. insbesondere, dass die spezifische Request-Response-Abwicklungslogik in der Methode

```
@Override
public final void handle(final HttpExchange he) throws IOException;
```

gekapselt wird.

Das "HttpExchange"-Object dient dabei als das eigentliche Kommunikations-Objekt für die zugehörigen HTTP-Requests bzw. HTTP-Responses.

```
final HttpServer server = hsp.createHttpServer(
            new InetSocketAddress("127.0.0.1", "24727"), 10
      );
```

Wir geben nun in vereinfachter Form die in der "handle"-Methode enthaltene Kommunikations-Logik an, ohne aber dabei näher auf den Umgang mit "Redirects" einzugehen.

1. Ermittlung des URL-Parameters "tcTokenURL"
2. GET-Request auf die in Punkt 1 ermittelte URL, falls das Protokoll "HTTPS" ist, sonst Fehlermeldung. Die Server-Zertifikate werden überprüft und als Liste ("sourceCerts") gespeichert (vgl. TR-03124-1).
3. Verarbeitung zu Punkt 2 gehörenden Responses und Ermittlung der Parameter für Schritt 4.
4. Initialisierung und Start des ECardWorkers mit den in Punkt 3 ermittelten Parametern
5. Auswertung des ECardWorker-Ergebnisses (Callback)

Die in **Schritt 3** ermittelten Parameter sind

- Server-Adresse
- Session-Identifier
- Refresh-Adresse

- PSK (**P**re-**S**hared-**K**ey)

Sie werden in Form von "text/xml"-Kontent als "TCTokenType" übergeben. Ein Beispiel ist:

```
<TCTokenType>
    <ServerAddress>https://eid.vx4.net:443/eid/ws/paos/</ServerAddre
ss>
    <SessionIdentifier>6D43337443A95CD1A6A49FC4656D68A1</SessionIden
tifier>
    <RefreshAddress>https://eid.services.ageto.net/gw/_auth/exec?id=
DAF067DDABE411048CC5BF8658C94D33DEE26196F9EEA5A0A9FE274EB6B0F1E6</Re
freshAddress>
    <Binding>urn:liberty:paos:2006-08</Binding>
    <PathSecurity-Protocol>urn:ietf:rfc:4279</PathSecurity-Protocol>
    <PathSecurity-Parameters>
        <PSK>2C542F99BEB91B9D3B342962F9F854566A281289266F7503FBBD23E
F68B6E273</PSK>
    </PathSecurity-Parameters>
</TCTokenType>
```

Diese Daten werden für den weiteren Prozess in einer Map: String -> String "params" abgelegt. Obiges Beispiel übersetzt sich dann zu:

```
{
      SessionIdentifier=6D43337443A95CD1A6A49FC4656D68A1,
      ServerAddress=https://eid.vx4.net:443/eid/ws/paos/,
      RefreshAddress=https://eid.services.ageto.net/gw/_auth/exec?id=DAF067
DDABE411048CC5BF8658C94D33DEE26196F9EEA5A0A9FE274EB6B0F1E6,
      PathSecurity-Protocol=urn:ietf:rfc:4279,
      Binding=urn:liberty:paos:2006-08,
      PathSecurity-
Parameters=2C542F99BEB91B9D3B342962F9F854566A281289266F7503FBBD23EF68B6E273
}
```

In **Schritt 4** wird anschließend der sogenannte "ECardWorker" gestartet. Dies geschieht durch den Aufruf

```
startECardWorker(params, sourceCerts, tcTokenURL, he);
```

bzw. durch

```
Object Callback = ECardWorker.start(ch, psk, tcTokenURL,
sourceCerts);
```

Der Wert `ch` ist vom Typ

```
iso.std.iso_iec._24727.tech.schema.ChannelHandleType
```

und enthält einerseits die Serveradresse des eID-Server

```
iso.std.iso_iec._24727.tech.schema.ChannelHandleType.setProtocolTermi
nationPoint
```

und andererseits den Session-Identifier

```
iso.std.iso_iec._24727.tech.schema.ChannelHandleType.setSessionIdenti
fier
```

Daneben werden die Werte `psk` Pre-Shared-Key und die in Schritt 2 ermittelten Server-Zertifikate `sourceCerts` übergeben. Der weitere Ablauf im ECardworker, das betrifft insbesondere den Aufruf des PAOS-Stacks, wird im nächsten Abschnitt "E-CardWorker" angegeben.

Erläuterungen zu Schritt 5

Das Ergebnis des ECardWorkers wird über eine Callback-Funktion übergeben, die technisch durch ein "wait-notifiyAll" realisiert wird. Mögliche Callback-Werte (Objects) sind:

- PROCESSING
- FINALLY
- TA_OK
- CANCEL
- TA_ERROR

Beispielsweise wird im positiven Callback-Fall, der durch

```
ECardWorker.CALLBACK_RESULT.TA_OK
```

definiert wird, zusätzlich noch die "Refresh-URL" geprüft. War diese Prüfung in Ordnung, so erfolgt die Generierung des Redirect-Response durch

```
sendResponse(he, 303, addParam(refreshURI, "ResultMajor=ok"),
null)
```

Dies markiert den Austritt aus dem lokalen Request, also dem "alternativen Aufruf" aus dem Browser.

5.2.2 ECardWorker

Wie erwähnt, ist die Aufgabe des „eCardWorkers“ die Verarbeitung des initialen Requests aus dem "alternativen Aufruf" und der anschließenden Etablierung einer, mit den aus dem "TCTokenType" stammenden Werten parametrisierten, PAOS-Kommunikation zwischen eID-Server und den integrierten ISO 24727 WebServices (SAL-Service, IFD-Service, etc.). Dies wurde in D06-QM angedeutet und im vorherigen Abschnitt näher beschrieben.

Wir werden nun Schritt 4 aus dem vorherigen Abschnitt aufgreifen und die gerade erwähnte Etablierung einer PAOS-Kommunikation, auf die wir bisher noch nicht eingegangen sind, in groben Zügen erläutern. Bevor der PAOS-Client gestartet werden kann, müssen jedoch noch einige Werte zu Verfügung gestellt werden.

Diese Werte-Initialisierung des PAOS-Clients gliedert sich in drei Schritte. Erstens der Erzeugung eine PAOS-Initiator-Instanz, durch

```
paosInitiator = PAOSInitiator.getInstance(wsCtx, serverAdress,
sessionIdentifier, psk);
```

zweitens der Initialisierung der beiden Werte

```
slotHandle,      contextHandle
```

welche gemäß den Vorgaben in TR-03112-6 (IFD) generiert werden, etwa durch

```
private final byte[] contextHandle = new byte[32];
private final byte[] slotHandle = new byte[32];

final Random sr = new Random();
sr.nextBytes(contextHandle);
sr.nextBytes(slotHandle);
```

und drittens der Erstellung einer ECardSession-Instanz

```
session = new ECardSession(. . ., eCardHandler);

session.setAttribute(ECardWorker.class.getName(), this);
session.setAttribute("SPServerCert", serverCerts);
session.setAttribute("tcTokenURL", tcTokenURL);
session.setAttribute(PAOSInitiator.class.getName(),
paosInitiator);
```

in der die gesammelten Initialisierungswerte zusammengefasst werden.

Die ECardSession wird zusätztlich noch dem WebserviceContext `wsCtx` durch folgenden Aufruf

```
wsCtx.getMessageContext().put(ECardSession.class.getName(),
session);
```

zugeordnet. (Der WebserviceContainer/Context wurde bereits ganz am Anfang beim Start der Applikation (in der "Main"-Methode) erstellt.)

Nachdem diese Vorarbeit, die Initialisierung, erfolgt ist, wird der PAOS-Client durch den Aufruf

```
ResponseType startPAOSResponse =
paosInitiator.start(contextHandle, slotHandle);
```

gestartet, die PAOS-Kommunikation beginnt.

(Die beiden übergebenen Werte hatten wir oben bereits erwähnt. Sie dienen während der Dauer einer PAOS-Sitzung der Zuordnung zu einer konkreten Session bzw. der Adressierung eine vom IFD gelieferten Karte. Eine genaue Beschreibung ist in TR-03112-6 enthalten.)

Den internen Ablauf der PAOS-Kommunikation werden wir, zur besseren Übersicht, in einem eigenen Abschnitt (PAOS-Client) ausführlicher betrachteten.

5.2.3 PAOS-Client

Der PAOS-Client fungiert, wie in D06 beschrieben, sowohl als SOAP- und HTTP-Client, als auch als PAOS-Server gegenüber dem eID-Server. D.h., er vermittelt die Nachrichten zwischen eID-Server und Client, bei der der eID-Server als Client auftritt (das ist das PAOS-Spezifikum). Bei PAOS wird, im Gegensatz zum üblichen HTTP-SOAP, ein PAOS-SOAP-Request an einen HTTP-Response gebunden, auch "Reverse HTTP binding for SOAP" genannt.

Die eID-Server-Anfragen werden dabei kontinuierlich in einer Schleife, bis hin zu einem gewissen Abbruchkriterium, durch einen Dispatcher, genauer einen JAXB-Dispatcher, entgegengenommen und die erzeugten **Antworten** (siehe unten) zurückgegeben. Die Kommunikation zum eID-Server erfolgt, nach Spezifikation XML-

basiert, über einen eigenen HTTP-Client (MiniHttpClient), der insbesondere ein "TLS-PSK-Client" ist (vgl. hierzu auch Abschnitt TLS 1.1+).

Der vom JAXB-Dispatcher benötigte JAXB-Kontext zur (De-) Serialisierung enthält folgende Objekte (Object-Factories):

- SOAP-Envelope: "org.xmlsoap.schemas.soap.envelope.ObjectFactory.class"
- SOAP-Adressing Header: "org.w3._2005._03.addressing.ObjectFactory.class"
- PAOS-Header: "liberty.paos._2006_08.ObjectFactory.class"
- ISO-24727 messages:
  "iso.std.iso_iec._24727.tech.schema.ObjectFactory.class"

Generierung der Antworten

Die Ergebnisse werden erzeugt durch den Aufruf des internen, von aussen nicht erreichbaren Web-Service Containers (`wsCtx`). An diesen Web-Service-Container bzw. Web-Service-Context sind bereits früher alle benötigten ISO 24727 Services im JAX-WS 2.0 Stil (JAX-WS: "Java API for XML Web Services") gebunden und initialisiert worden.

Der prinzipielle Ablauf einer PAOS-Sitzung besteht aus drei Teilen

1. StartPAOS
2. RESPONSE -> REQUEST (= eID-Server-Request -> Client-Response)
3. StartPAOSResponse

Jeder HTTP-Request des Clients beginnt mit:

```
POST /eid/ws/paos/ HTTP/1.1
Host: eid.vx4.net
Connection: keep-alive
Accept-Encoding: gzip
Content-Length: ...
PAOS: ver="urn:liberty:2003-08","urn:liberty:2006-08";
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
Content-Type: application/vnd.paos+xml; charset=UTF-8
Accept: application/vnd.paos+xml
```

Zum Vergleich für den konkreten Ablauf dient [2] Teil 7, Kapitel 2.3, S. 1-14 und Bild 2, „PAOS Binding“ im Besonderen der Bildabschnitt „Application Logic“.

Der wesentliche Ablauf ist wie folgt:

StartPAOS

Ist der initialer Request vom Client zur Eröffnung der PAOS-Sitzung. Es werden die Parameter SessionIdentifier, ContextHandle und SlotHandle übertragen:

```
<ns5:StartPAOS>

<ns5:SessionIdentifier>5A2957B56BC66626E12E126CEFC0D7E3</ns5:SessionIdentif
ier>
            <ns5:ConnectionHandle>

<ns5:ContextHandle>49BC2442D2B87A22DEB521193723E1351DD4202745227024775FA3C6
4E38FB5B</ns5:ContextHandle>

<ns5:SlotHandle>7A56A25D0C3AB90F2E23D77A645861C8EE917F1EAEC0217ED75C9E4A892
B96B2</ns5:SlotHandle>
            </ns5:ConnectionHandle>
            <ns5:UserAgent>
                <ns5:Name>PersoApp</ns5:Name>
                <ns5:VersionMajor>0</ns5:VersionMajor>
                <ns5:VersionMinor>1</ns5:VersionMinor>
            </ns5:UserAgent>
            <ns5:SupportedAPIVersions>
                <ns5:Major>1</ns5:Major>
                <ns5:Minor>1</ns5:Minor>
                <ns5:Subminor>3</ns5:Subminor>
            </ns5:SupportedAPIVersions>
</ns5:StartPAOS>
```

StartPAOSResponse

Ist der finale Response vom eID-Server, der die PAOS-Sitzung beendet.

```
<ns3:StartPAOSResponse
xmlns="urn:oasis:names:tc:dss:1.0:core:schema"
     xmlns:ns2="http://www.w3.org/2000/09/xmldsig#"
     xmlns:ns3="urn:iso:std:iso-iec:24727:tech:schema"
     xmlns:ns4="http://www.bsi.bund.de/ecard/api/1.1"
     xmlns:ns5="http://uri.etsi.org/02231/v2.1.1#"
     xmlns:ns6="http://uri.etsi.org/02231/v2.x#"
     xmlns:ns7="http://uri.etsi.org/02231/v3.1.2#"
     xmlns:ns8="http://www.setcce.org/schemas/ers">
   <Result>

     <ResultMajor>http://www.bsi.bund.de/ecard/api/1.1/resultmajor#o
k</ResultMajor>
  </Result>
</ns3:StartPAOSResponse>
```

Mittelteil der PAOS-Kommunikation

Der mittlere Teil der PAOS-Kommunikation beginnt mit einem HTTP-Reponse vom eID-Server, d.h. mit einer PAOS-Anfrage vom eID-Server. Der anschliessende HTTP-Request vom Client ist die zugehörige Antwort. Nach diesem Schema werden dann Terminal-Authentification, Chip-Authentification, etc. abgewickelt.

5.2.4 TLS 1.1+, Bouncy-Castle

Diese Komponente bezeichnet einen TLS-Protokollstack, welcher für die verschlüsselte Kommunikation zwischen eID-Client und Diensteanbieter-System sowie zwischen eID-Client und eID-Server verantwortlich ist. Bei letzterem ist zu beachten, dass TLS-RSA-PSK mit mindestens AES-128 unterstützt werden muss (vgl. hierzu [2] Teil, Kap. 2.4.1.1-2). Damit scheiden die Standard-Bibliotheken im Regelfall aus.

Im Open-Source-Projekt "PersoApp" kommt Bouncy-Castle (http://www.bouncycastle.org/) zum Einsatz, welches alle erforderlichen, oben genannten, Voraussetzungen erfüllt. Die Anbindung von Bouncy-Castle erfolgt durch eine Corekomponente (Wrapper), die im Java-Paket

```
de.persoapp.core.tls
```

zu finden ist.

Dieses Paket beinhaltet folgende Klassen/Interfaces:

- `BCTlsAuthentication (implements org.bouncycastle.crypto.tls.TlsAuthentication)`
- `TLSClient (org.bouncycastle.crypto.tls.DefaultTlsClient)`
- `TLSPSKClient (org.bouncycastle.crypto.tls.PSKTlsClient)`

sowie

- `BCTlsSession (javax.net.ssl.SSLSession)`
- `BCTlsSocketImpl (javax.net.ssl.SSLSocket)`
- `BCTlsSocketFactoryImpl (javax.net.ssl.SSLSocketFactory)`

Die Schnittstelle nach "Außen" stellt die Klasse "BCTlsSocketFactoryImpl" dar. Sie besitzt entsprechende Konstruktoren für PSK-basierte Clients, als auch für nicht PSK-basiete Client, genauer:

1. BCTlsSocketFactoryImpl()
2. BCTlsSocketFactoryImpl(sessionID, psk)

Im PersoApp-Kontext wird der erste Konstruktor im Fall der Kommunikation mit dem Dienstanbietersystem ohne PSK (d.h. im "ECApiHttpHandler") genutzt und der zweite wird zur Kommunikation mit dem eID-Server mit PSK benötigt (siehe: Initialisierung des MinitHttpClients im PAOS-Stack).

5.3 ISO-24727 Services

In BSI-TR 03112-3, -4 und -6 werden generisch die von der „PersoApp“-Open-Source-Software-Bibliothek unterstützten Dienste eines eID-Clients definiert, dazu gehören

- „Management Interface“: BSI TR-03112-3
- „Service Access Layer“ (SAL): BSI TR-03112-3, 4
- „Interface Device“ (IFD): BSI TR-03112-6
- „Protocol Service“: BSI TR-03112-3, 4

Die Interfaces dieser Services sind im Paket

```
iso.std.iso_iec._24727.tech.schema
```

zu finden.

5.4 eCardHandler

Der eCardHandler stellt eine abstrakte Schnittstelle zu den Funktionen der eID-Karte dar. U.a. wird beim Einsatz eines Basislesers das PACE-Protokoll (Password Authenticated Connection Establishment) aber auch weite Teile der Terminal- und Chip-Authentication hier direkt abgebildet.

Zu diesem Zweck wird durch die Klasse

```
de.persoapp.core.card.ECardHandler
```

das Interface

```
de.persoapp.core.card.ICardHandler
```

bereitgestellt, welches die folgenden Methoden beinhaltet:

```
      TransportProvider getECard();

      int hasPACE(Object transport);

      int doPINUnblock(TransportProvider tp, byte verifySecret,
SecureHolder verifySecretInput, byte unblockSecret);

      int doPINChange(TransportProvider tp, byte verifySecret, SecureHolder
verifySecretInput, byte modifySecret,
                  SecureHolder modifySecretInput);

      public int doESignInit(TransportProvider tp0);

      public int doESignChange(TransportProvider tp0);

      public int doESignUnblock(TransportProvider tp0);

      public int doESignTerminate(TransportProvider tp0);

      boolean startAuthentication(byte[] CHAT, SecureHolder secret, byte[]
termDesc);

      void reset();

      byte[] getIDPICC();

      List<byte[]> getCAReferences();

      byte[] getEFCardAccess();

      boolean verifyCertificate(byte[] cvc);

      void initTA(byte[] ephemeralKey, byte[] auxData);

      byte[] getTAChallenge();

      boolean verifyTASignature(byte[] taSignature);

      byte[] getEFCardSecurity();

      byte[] execCA();

      byte[] transmit(byte[] cmd);
```

TransportProvider

Da nicht nur die PC/SC-Schnittstelle eine Möglichkeit darstellt auf Lesegeräte (IFDs) zuzugreifen wurde der „TransportProvider“ eingeführt. Darüber werden die Fähigkeiten der Lesegeräte abstrakt abgebildet. Beispielsweise existieren Transportprovider Implementierungen auf PC/SC, USB CCID, die im Desktop-Projekt relevant sind.

Aus Sicht eines hypothetischen „Generic Card Access Layer“ bildet der „Transport-Provider“ den Übergang in Hardware-nahe Schichten.

6 Mitgeltende Dokumente

[1] D06-QM-Architekturkonzept-der-Open-Source-Core v 1.0 (https://www.persoapp.de/wp-content/uploads/2013/07/D06-QM-Architekturkonzept-der-Open-Source-Core.pdf)

[2] BSI - Technische Richtlinie TR 03112-(1-7), Version 1.1.2, Bundesamt für Sicherheit in der Informationstechnik, 2012.

[3] BSI - Technische Richtlinie TR 03116-4 eCard-Projekte der Bundesregierung, Teil 4 – Vorgaben für Kommunikationsverfahren im eGovernment, 2013.

[4] BSI - Technische Richtlinie TR 03124-1 eID-Client – Part 1: Specifications. Bundesamt für Sicherheit in der Informationstechnik, 2013

[5] ISO/IEC 24727-(1-3): Identification cards, 2007/2008.

[6] Liberty Reverse HTTP Binding for SOAP Specification (PAOS). Liberty Alliance Project: Version: v1.1 (http://www.projectliberty.org/liberty/content/download/1219/7957/file/liberty-paos-v1.1.pdf)

[7] The Legion of the Bouncy Castle (http://www.bouncycastle.org/)