Integration of the Security Relevant Modules into the FutureID Client

WP 35 Trustworthy Client Platform

D 35.5	
Date	15/10/2015
Status	Final
Version	1.0

Related SP / WP	SP 3/ WP 35	**Document Reference**	D. 35.5
Related Deliverable(s)	D 35.2, D 35.1, D 35.3	**Dissemination Level**	Public
Lead Participant	G&D	**Lead Author**	Dr. F.-M. Kamm
Contributors	TUG, TUD	**Reviewers**	RU, DTU

This document is issued within the frame and for the purpose of the FutureID project. This project has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318424

This document and its content are the property of the FutureID Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the FutureID Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the FutureID Partners.

Each FutureID Partner may use this document in conformity with the FutureID Consortium Grant Agreement provisions

1. Abstract

This deliverable documents the integration of security enhancing components on the client platform into the FutureID client architecture. The additional security allows for a better protection of credentials, especially those outside of secure elements, and an easier access over various transport channels like NFC, USB or the SIM card. Supported by this infrastructure, mobile ID solutions as well as smartcard based use cases are easier to roll out and to use.

The integration of the extended OpenMobile API with plugin-terminals allows establishing a single communication channel to the Secure Elements. The implementation of the plugin-terminals has initially been demonstrated with a PKI applet on a SIM card, a USB token, and a contactless card (NFC). This demonstration however took place with a standalone app that integrates the extended OpenMobile API as a JCE provider. To make this implementation usable for FutureID it was necessary to integrate the API into the FutureID Android client. This has been achieved by implementing the SCIO API accordingly. For a smooth integration of the PKI applet use case into the FutureID client it was necessary to make a small modification of the SEEK V3.2.1 implementation. In addition, a CardInfo file had to be generated that describes the applet functionality. Since this is not a standard PKCS#15 type of data structure, significant modifications had to be done on the automatically generated file. With this CardInfo file, the extended OpenMobile API and the integration into the FutureID client IFD architecture it is now possible to demonstrate the use of a PKI applet, as it can be found in many mobile ID applications. Details of this implementation are described in chapter 5.

The integration of Trusted Execution Environments (TEEs) into the FutureID client is described in chapter 6. Based on the ARM TrustZone the TEE can be used to store and manage confidential information, such as certificates, and perform computations, such as signature creation, without exposing any security relevant data. The ARM TrustZone provider architecture is based upon the Service Access Layer (SAL) provider architecture presented in deliverable D32.5. For creating signatures and generating key pairs it is necessary to interact with the TEE which is achieved by implementing a Java native interface (JNI) wrapper. In order to demonstrate the use of a TEE, a prototypical GUI was implemented. The FutureID client allows for creating signatures, generating key pairs and the corresponding Certificate Sign Request (CSR), and importing certificates into the TEE.

In another scenario, the use of the Android Security Modules (ASM) framework was proposed to enforce access control on Android. Since a full integration of the ASM framework is beyond the scope of FutureID, this document will elaborate scenarios of how the ASM framework can be utilized for FutureID. One of the proposed integration scenarios is that the ASM framework, using the hooks to the file system, restricts the access to the software credentials so that only the FutureID client is able to access to this file. Another proposed integration scenario is to protect the access to the secure element using the Android Security Modules framework. In this scenario the NFC plugin-terminal is used to provide the FutureID client with an access to a contactless smartcard. In order to secure this access the ASM framework is used to give the user the choice on whereas to allow the access or deny it. The integration

scenarios are discussed in chapter 7, while chapter 8 provides conclusions and an outlook beyond the FutureID project.

2. Document Information

2.1 Contributors

Name	Partner
F.-M. Kamm	G&D
David Derler	TUG
Christof Rath	TUG
Jon Rios	TUD

2.2 History

Version	Date	Author	Changes
0.1	10.09.2015	F.-M. Kamm	Initial version
0.2	14.09.2015	F.-M. Kamm	Chapter 5 added
0.3	15.09.2015	F.-M. Kamm	Chapter 5 updated
0.4	24.09.2015	Wolfgang Schoechl	Chapter 6 added
0.5	06.10.2015	F.-M. Kamm	Citations added
0.6	09.10.2015	J. Rios	Chapter 7 added
0.9	12.10.2015	F.-M. Kamm	Chapter 1,8 added, internal reviewer version finalized
1.0	15.10.2015	F.-M. Kamm	Reviewer comments integrated

3. Table of Contents

4. Project Description

The FutureID project builds a comprehensive, flexible, privacy-aware and ubiquitously usable identity management infrastructure for Europe, which integrates existing eID technology and trust infrastructures, emerging federated identity management services and modern credential technologies to provide a user-centric system for the trustworthy and accountable management of identity claims.

The FutureID infrastructure will provide great benefits to all stakeholders involved in the eID value chain. Users will benefit from the availability of a ubiquitously usable open source eID client that is capable of running on arbitrary desktop PCs, tablets and modern smart phones. FutureID will allow application and service providers to easily integrate their existing services with the FutureID infrastructure, providing them with the benefits from the strong security offered by eIDs without requiring them to make substantial investments.

This will enable service providers to offer this technology to users as an alternative to username/password based systems, providing them with a choice for a more trustworthy, usable and innovative technology. For existing and emerging trust service providers and card issuers FutureID will provide an integrative framework, which eases using their authentication and signature related products across Europe and beyond.

To demonstrate the applicability of the developed technologies and the feasibility of the overall approach FutureID will develop two pilot applications and is open for additional application services who want to use the innovative FutureID technology

Future ID is a three-year duration project funded by the European Commission Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318424

5. Integration of OpenMobile API and Plugin-Terminals

As outlined in deliverable D 35.3 (Implementation of the Security-relevant Modules for Selected Platform) the Open Mobile API is an API that allows applications on mobile devices to access various kinds of secure elements in a standardized way [1]. These elements can be SIM cards, secure microSD cards, or other kinds of secure tokens embedded in or attached to mobile devices. The API definition is independent of a specific platform or programming language and can therefore be implemented on any type of device and operating system.

Deliverable D 31.2 (Interface and Module Specification and Documentation) describes how the OpenMobile API is integrated into the mobile FutureID client architecture for Android devices and how it interacts with the IFD and the IFD proxy layer [2]. This integration allows establishing a single communication channel to the Secure Element(s), which is also preferable from a security point of view. In the basic version, the OpenMobile API supports a channel to the SIM card as Secure Element. In order to offer other channels as well, the concept of plugin-terminals was introduced and plugin-terminals for NFC support and USB support have been implemented (see deliverable D 35.3) [1]. The implementation of the plugin-terminals was demonstrated with a PKI applet on a SIM card, a USB token, and a contactless card (NFC). The communication to all of these Secure Elements is possible via one channel through the OpenMobile API and the corresponding plugin-terminals.

This demonstration however took place with a standalone app that integrates the extended OpenMobile API as a JCE provider. To make this implementation usable for FutureID it was necessary to integrate the API into the FutureID Android client. The following sections describe how the integration was achieved and which further modifications were necessary to allow the interoperability with the IFD layer.

5.1 Integration into IFD Architecture

The Interface Device (IFD) service provides a common interface for communication with arbitrary credentials. It encapsulates card terminals, smart cards, and secure elements and provides an interface to easily access these devices. As described further in D 31.3, Figure 1 illustrates the architecture of the implemented IFD [3]. As highest layer, the IFD provides an API to access the service. The API is specified in D31.2 (Section 4) and provides a common interface for communication with arbitrary smart cards [2].

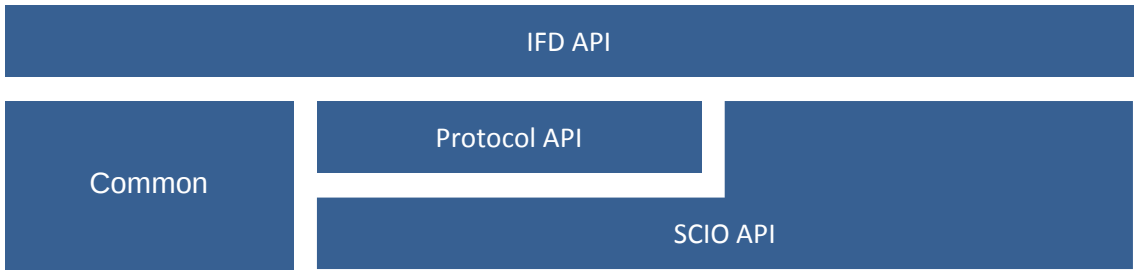

Figure 1: IFD Architecture comprising the IFD API and the underlying service layers (see D 31.3).

The Common module contains generic data structures and provides convenience functions of other modules. The Protocol API provides an interface for protocols, which establish a secure channel between the IFD service and connected smart cards. The SCIO (Smart Card Interface Input Output) API provides an interface for common smart card operations and abstracts from the particular interfaces technology like PC/SC, NFC, or the Open Mobile API.

To manage a situation in which a device may contain more than one IFD, a proxy layer concept has been introduced and implemented. This proxy transparently abstracts multiple IFD implementations towards the SAL. The basic layout is shown in Figure 2. Modules that use the IFD proxy shall be unaware that the actual execution is handled by one of the registered sub IFDs.

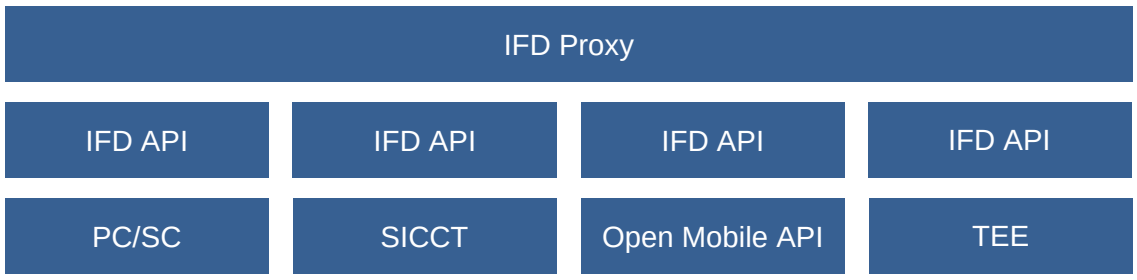

Figure 2: IFD Proxy layer concept for handling multiple IFDs on one device (see D 31.3).

As can be seen in Figure 2, one of the possible IFDs is the OpenMobile API which is also the preferred choice for mobile devices. Since PC/SC and SICCT are more frequently found on desktop devices and TEEs are not yet available on all mobile devices, the OpenMobile API may be the only required IFD channel. Especially with the new plugin-terminal implementation the API is also able to address several types of Secure Elements, thus eliminating the need for other parallel IFDs. In this case, the IFD proxy layer would not be needed anymore. However, for reasons of simplicity and to keep the required modifications as small as possible the extended OpenMobile API was integrated into the existing implementation containing the IFD proxy.

5.2 IFD/OpenMobile API Integration

In order to integrate the extended OpenMobile API (SEEK V3.2.1) into the IFD, the SCIO API has to be implemented accordingly to address the OpenMobile API-specific methods and to use the plugin-terminals [4]. In D 35.3 it was already described how the terminal interface of the OpenMobile API interacts with the plugin-terminal modules [1]. The integration into the SCIO is illustrated by the class diagram as shown in Figure 3. It shows how the SCIO classes use the terminal interfaces as provided by the extended OpenMobile API.

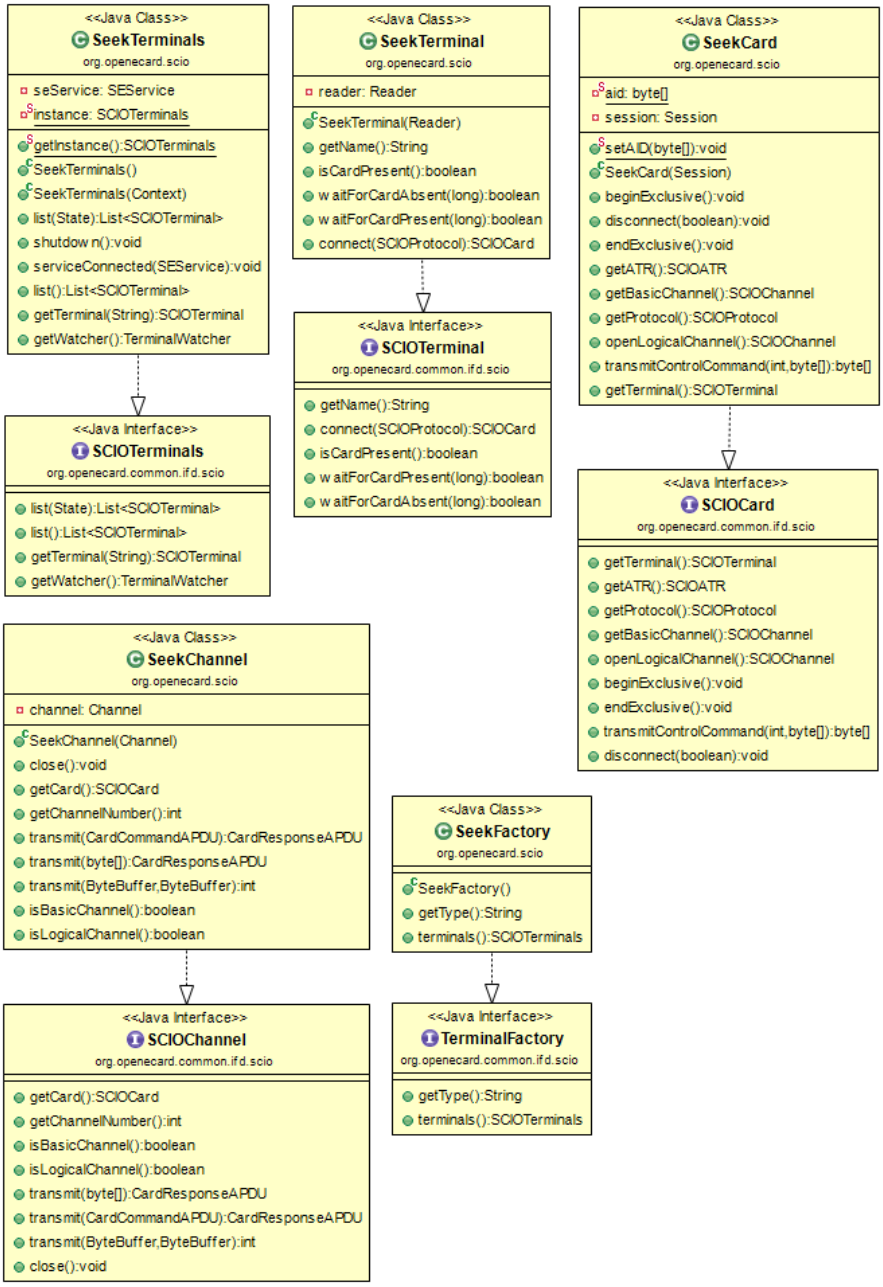

Figure 3: Class diagram of the SCIO layer implementation using the extended OpenMobile API with plugin-terminals.

For a smooth integration of the PKI applet use case into the FutureID client it was necessary to make a small modification of the SEEK V3.2.1 implementation. Due to the limitations of the OpenMobile API it is not foreseen that a `select` APDU from the IFD layer for selecting an applet with a specific Application Identifier (AID) is simply passed on to the applet. Therefore, it is necessary to open a new channel to the OpenMobile API with `channel.getSession().openBasicChannel(AID)`. With this extension it is possible to select the PKI applet with its AID.

5.3 CardInfo File for PKI Applet

One of the foundations of the FutureID client for accessing various types of eID cards are the CardInfo files, as defined in CEN 15480-4. These XML-files describe the available applications on a card and required information for accessing and using cryptographic keys. Thus, for each card that shall be supported by FutureID it is necessary to generate a CardInfo file. In WP 32.6 an automatic tool for generating CardInfo files has been developed [5].

Accordingly, for supporting the PKI applet use case a CardInfo file has to be generated that describes the applet functionality. Since this is not a standard PKCS#15 (ISO/IEC 7816-15) type of data structure, significant modifications had to be done on the automatically generated file.

The following code contains the result of the CardInfo file generation and adaption:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<iso:CardInfo xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:ns10="http://uri.etsi.org/01903/v1.3.2#"
xmlns:ns11="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:ns12="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:ns13="http://www.w3.org/2001/04/xmlenc#"
xmlns:ns14="http://ws.openecard.org/schema"
xmlns:ns15="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:ns16="http://www.w3.org/2007/05/xmldsig-more#"
xmlns:ns2="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:ns3="http://www.w3.org/2000/09/xmldsig#"
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" xmlns:ns9="urn:oasis:names:tc:dss-
x:1.0:profiles:verificationreport:schema#">
  <iso:CardType>
       <iso:ObjectIdentifier>CardInfo_PKI_Giesecke_Devrient.xml</iso:ObjectIdentifier>
              <iso:CardTypeName xml:lang="en">PKI Giesecke Devrient</iso:CardTypeName>
              <iso:CardTypeName xml:lang="de">PKI Giesecke Devrient</iso:CardTypeName>
              <!-- version corresponds to cif version -->
              <iso:Version>
                    <iso:Major>0</iso:Major>
                    <iso:Minor>0</iso:Minor>
                    <iso:SubMinor>1</iso:SubMinor>
              </iso:Version>
```

```
            <iso:Status>Final</iso:Status>
            <iso:Date>2014-24-10</iso:Date>
     </iso:CardType>
     <iso:CardIdentification>
  <iso:ATR>
    <iso:TS>
      <iso:Value>3B</iso:Value>
      <iso:Mask>FF</iso:Mask>
    </iso:TS>
    <iso:T0>
      <iso:Value>9F</iso:Value>
      <iso:Mask>FF</iso:Mask>
    </iso:T0>
    <iso:InterfaceBytes>
      <iso:Tx1>
        <iso:TAi>
          <iso:Value>C7</iso:Value>
          <iso:Mask>FF</iso:Mask>
        </iso:TAi>
        <iso:TBi>
          <iso:Value>A0</iso:Value>
          <iso:Mask>FF</iso:Mask>
        </iso:TBi>
        <iso:TDi>
          <iso:Value>3F</iso:Value>
          <iso:Mask>FF</iso:Mask>
        </iso:TDi>
      </iso:Tx1>
      <iso:Tx2>
        <iso:TAi>
          <iso:Value>C7</iso:Value>
          <iso:Mask>FF</iso:Mask>
        </iso:TAi>
        <iso:TBi>
          <iso:Value>A0</iso:Value>
          <iso:Mask>FF</iso:Mask>
        </iso:TBi>
        <iso:TDi>
          <iso:Value>3F</iso:Value>
          <iso:Mask>FF</iso:Mask>
        </iso:TDi>
      </iso:Tx2>
      <iso:Tx3>
        <iso:TAi>
          <iso:Value>C7</iso:Value>
          <iso:Mask>FF</iso:Mask>
        </iso:TAi>
        <iso:TBi>
          <iso:Value>A0</iso:Value>
          <iso:Mask>FF</iso:Mask>
        </iso:TBi>
        <iso:TDi>
          <iso:Value>3F</iso:Value>
          <iso:Mask>FF</iso:Mask>
```

```
        </iso:TDi>
      </iso:Tx3>
      <iso:Tx4/>
    </iso:InterfaceBytes>
    <iso:HistoricalBytes>
      <iso:Ti>
        <iso:Value>80</iso:Value>
        <iso:Mask>00</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>31</iso:Value>
        <iso:Mask>00</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>E0</iso:Value>
        <iso:Mask>FF</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>73</iso:Value>
        <iso:Mask>FF</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>F6</iso:Value>
        <iso:Mask>FF</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>21</iso:Value>
        <iso:Mask>FF</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>13</iso:Value>
        <iso:Mask>FF</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>57</iso:Value>
        <iso:Mask>FF</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>4A</iso:Value>
        <iso:Mask>FF</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>4D</iso:Value>
        <iso:Mask>FF</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>0E</iso:Value>
        <iso:Mask>FF</iso:Mask>
      </iso:Ti>
      <iso:Ti>
        <iso:Value>1D</iso:Value>
        <iso:Mask>00</iso:Mask>
      </iso:Ti>
      <iso:Ti>
```

```
          <iso:Value>34</iso:Value>
          <iso:Mask>00</iso:Mask>
        </iso:Ti>
        <iso:Ti>
          <iso:Value>41</iso:Value>
          <iso:Mask>00</iso:Mask>
        </iso:Ti>
        <iso:Ti>
          <iso:Value>00</iso:Value>
          <iso:Mask>00</iso:Mask>
        </iso:Ti>
      </iso:HistoricalBytes>
    </iso:ATR>
  <iso:CharacteristicFeature>
                    <!--
                           The recognition of the card type will be realized by
                           (1.) selecting the file EF.DIR and
                           (2.) reading the content with the READ BINARY command and
comparing the result with the value specified below.
                    -->
                    <iso:CardCall>
                           <!-- 1. CL=00, INS=A4=SELECT, P1= 02, P2=0C, Lc=02,
Data=2F00 (FI of EF.DIR), Le=absent -->
                           <iso:CommandAPDU>00A4020C022F00</iso:CommandAPDU>
                           <iso:ResponseAPDU>
                                  <iso:Trailer>9000</iso:Trailer>
                           </iso:ResponseAPDU>
                    </iso:CardCall>
                    <iso:CardCall>
                           <!-- 2. CL=00, INS=B0=Read Binary, P1=00, P2=00 (no offset),
Lc=00, Le=5A -->
                           <iso:CommandAPDU>00B000005A</iso:CommandAPDU>
                           <iso:ResponseAPDU>
                                  <!-- specific content of EF.DIR -->
                                  <iso:Body>
                                         <iso:MatchingData>
                                                <iso:Offset>00</iso:Offset>
                                                <iso:Length>5A</iso:Length>

      <iso:MatchingValue>61324F0FE828BD080FA000000167455349474E500F434941207A752044462
E655369676E5100730C4F0AA000000167455349474E61094F07A0000002471001610B4F09E80704007F0007
0302610C4F0AA000000167455349474E</iso:MatchingValue>
                                         </iso:MatchingData>
                                  </iso:Body>
                                  <iso:Trailer>9000</iso:Trailer>
                           </iso:ResponseAPDU>
                    </iso:CardCall>
             </iso:CharacteristicFeature>
  </iso:CardIdentification>
  <iso:ApplicationCapabilities/>
</iso:CardInfo>
```

With this CardInfo file, the extended OpenMobile API and the integration into the FutureID client IFD architecture it is now possible to demonstrate the use of a PKI applet, as it can be found in many mobile ID applications, with the FutureID client. The applet can be accessed by the client when it is available on a SIM card, as well as on a dual interface card via NFC or a USB token. Thus, with this extension the FutureID client also supports mobile ID use cases.

In addition, with establishing a single communication channel to various types of Secure Elements via the OpenMobile API, the system architecture as outlined in deliverable D 35.2 (section 7.3) has been realized [6].

6. Integration of ARM TrustZone-based eSign Modules

As it was described in deliverable D35.3 (Implementation of the Security-relevant Modules for Selected Platform) the ARM TrustZone is a Trusted Execution Environment where security relevant computations can be isolated from other, non-secure operations [1]. Therefore, the ARM TrustZone can be used to store and manage confidential information, such as certificates, and perform computations, such as signature creation, without exposing any security relevant data.

6.1 ARM TrustZone Provider

The ARM TrustZone (TZ) Provider's architecture is based upon the SAL Provider architecture presented in deliverable D32.5 (Implementation of protocol-specific modules for selected protocols) [7]. Thus, the `TZProvider` was implemented as a Java Cryptographic Service Provider (JCP) and, thereby, allows for transparently calculating signatures as well as generating key pairs utilizing the TZ.

In order to manage multiple key pairs and make handling of these more convenient, every key pair has an alias associated with it. To configure the alias as well as the key size of the generated key pair, the KeyPairGenerator must be initialized with a TZKeyParamSepc object containing the alias and the key length. Figure 4 illustrates the `TZKeyPairGenerator` and the `TZKeyParamSpec` classes.

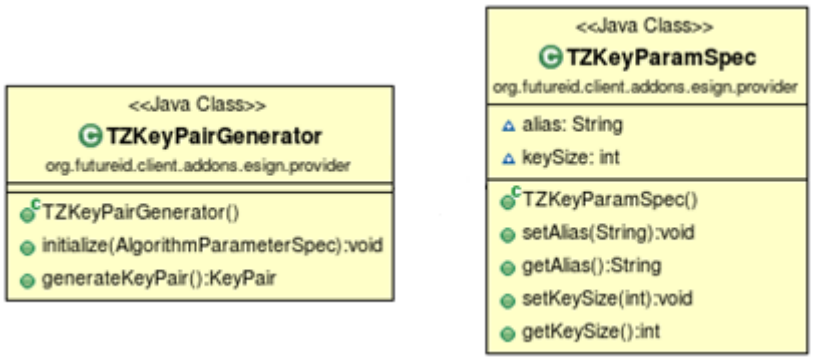

Figure 4: The key pair generator and parameter specification classes.

A key pair consists of a `TZRSAPublicKey` and `TZPrivateKey`. `TZRSAPublicKey` is an `RSAPublicKey` in the means of the JCE provider of the IAIK Institute of TU Graz, whereas the `TZPrivateKey` stores the alias and the key length and can be seen as a reference to the actual private key stored in the TZ. Figure 5 depicts the `TZRSAPublicKey` and `RSAPublicKey` classes.

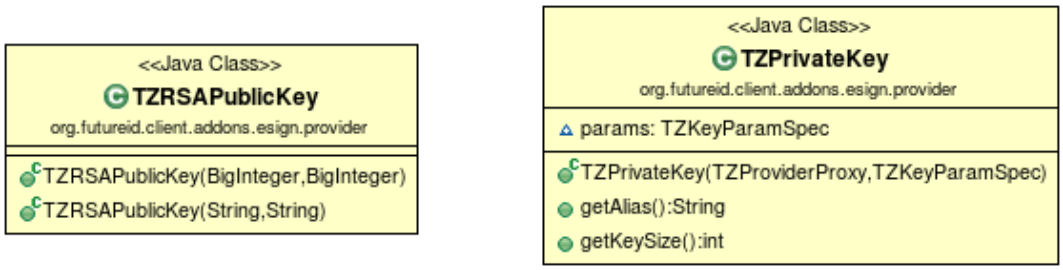

Figure 5: The TZRSAPublicKey and TZPrivateKey classes.

6.2 Java Native Interface (JNI) Wrapper

For creating signatures and generating key pairs it is necessary to interact with the TZ. As described in deliverable D35.3 section 7.3 (Architecture and Implementation) [1], this is achieved by implementing a Java native interface (JNI) wrapper. The `JniWrapper.class` contains the methods required to interact with the TZ. This class is responsible for calling the native TZ methods, and, additionally, for pre-processing data passed to/from the TZ. The methods provided by the JNI wrapper are shown in Figure 6.

Figure 6: The JniWrapper class.

6.2.1 JNI Wrapper Method Description

- **getAvailableKeyAliases**

  As mentioned in Section 6.1, the TZ uses aliases to distinguish between multiple keys. This method retrieves the aliases from all available key pairs stored in the TZ.

- **getCertificate**

  With this method, the certificate for a key, identified by its alias, can be retrieved from the TZ.

- **sign**

  The sign method calculates the signature for the given digest using the private key the given alias references.

- **generateKeyPair**

  This method allows for generating a RSA key pair. The key size and the alias are specified in a TZKeyParamSpec object.

- **setCertificate**

  In order to use previously generated keys for signature creation, a valid certificate must be imported. This method is provided therefor.

- **checkPinValid**

  This method compares a given PIN with the PIN stored in the TZ.

6.3 Integration into the FutureID client

In order to demonstrate the aforementioned features, a prototypical GUI was implemented. The FutureID client allows for creating signatures, generating key pairs and the corresponding Certificate Sign Request (CSR), and importing certificates into the TZ.

As described in Section 6.1, the TZ identifies key pairs by an alias. In order to be able to select the key, which should be used for signing or to which an imported certificate belongs to, the GUI, shown in Figure 7, is provided.

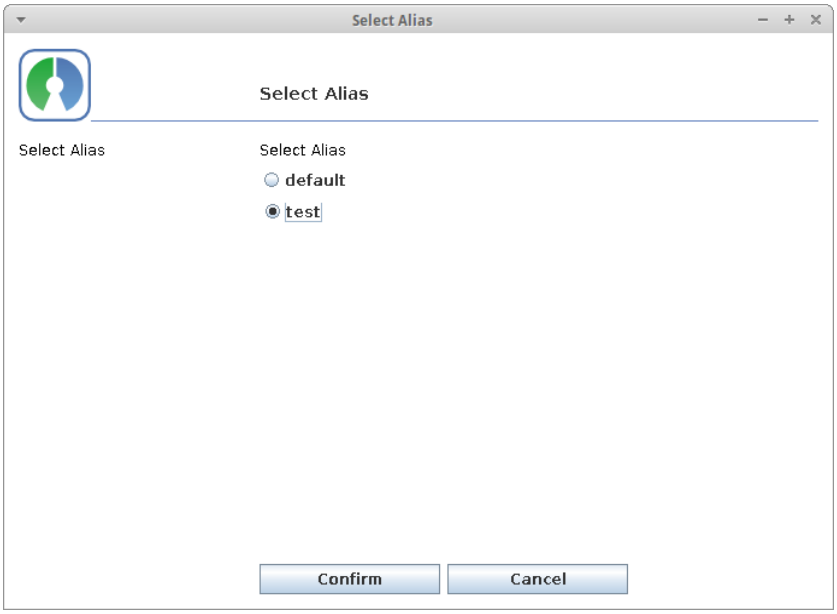

Figure 7: The dialog for selecting a key alias.

6.3.1 Sign

After a sign request was received and confirmed, the user is instructed to select a key which should be used for signature creation. By pressing the “Confirm” button, the data, for which the signature should be calculated, is passed into the TZ. The TZ then calculates the signature using the previously selected key.

6.3.2 Generate Key Pair

In order to be able to generate a key pair and the corresponding CSR, the following steps have to be performed. A separate dialog for each step is provided.

- Firstly, the user must specify an alias. If an alias is not specified, a MessageBox, informing the user that an alias must be specified, is shown (not depicted here).
- After defining the alias, the user must choose the desired key length. The dialog presented in Figure 8 is provided therefore. A key length of 2048 bits is selected by default.

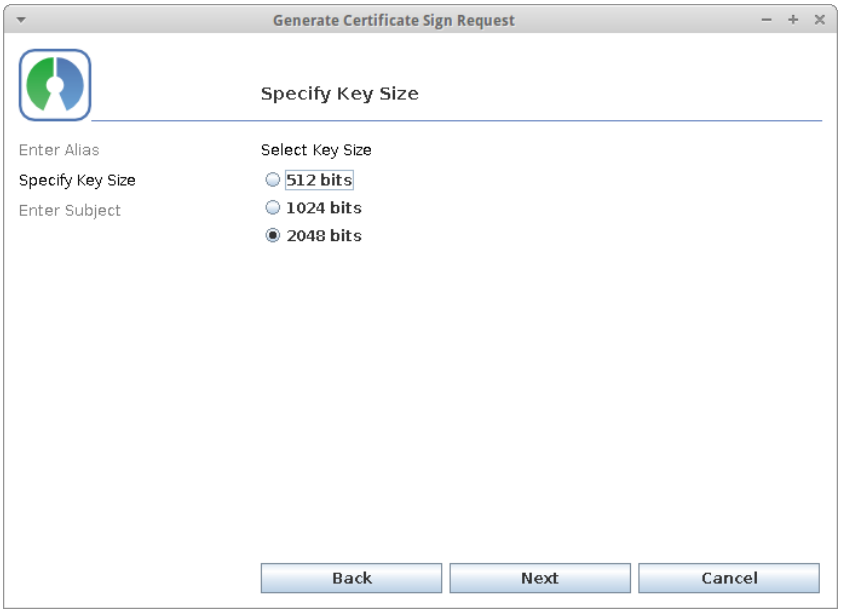

Figure 8: The dialog for selecting the key length.

- Finally, the user must define the subject. Figure 9 shows the dialog, where the subject (Country, Locality, Organization, Organizational Unit, Common Name), can be defined.

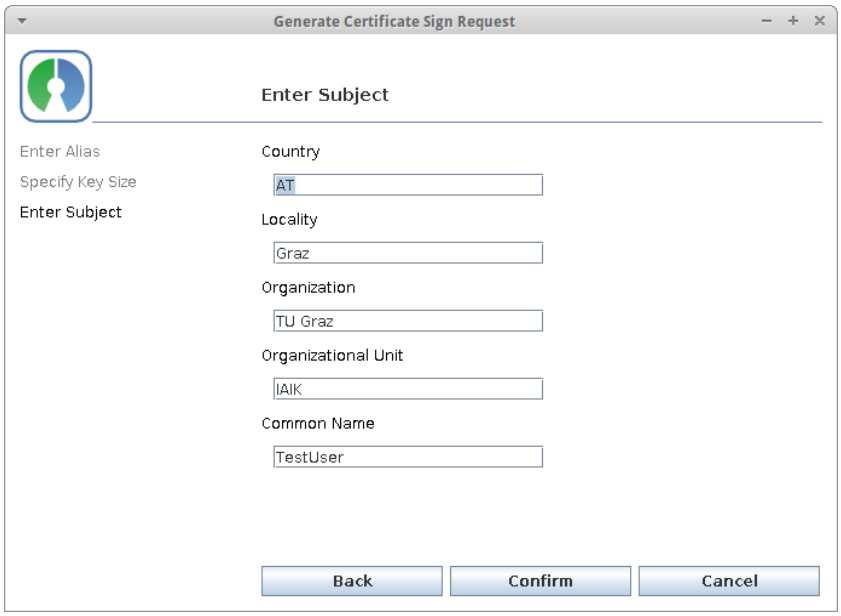

Figure 9: The dialog for defining the subject.

By pressing the “Confirm” button the TZ is instructed to generate a key pair. If the key pair generation was successful, a CSR, consisting of the generated public key and the previously defined subject, is created, signed, and finally presented to the user. Figure 10 shows the CSR dialog with a generated CSR.

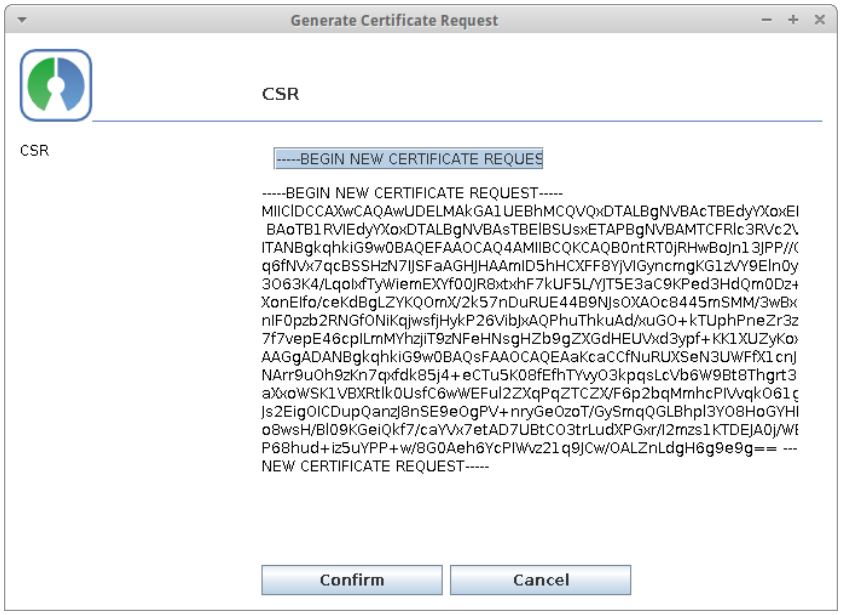

Figure 10: The CSR dialog.

6.3.3 Import Certificate

The first step, when importing a certificate, is to select the key alias to which the certificate belongs to. After selecting an alias, a file selection dialog, where the user can select the certificate which should be imported, is opened. By pressing the “Open" button, the certificate is parsed and stored in the TZ. The user is then able to use the generated key to fulfil signing requests.

6.4 Remarks

Currently, the TZProvider and the TZ operate with RSA keys only. However, the TZProvider as well as the TZ could be extended in order to handle ECDSA keys.

7. Integration Scenarios for Mandatory Access Control

The goal of Mandatory Access Control is to be able to regulate the access to critical resources and other software elements in order to ensure authorized use of such systems and avoid unwanted utilization of them [1]. In this scenario, the use of the Android Security Modules (ASM) framework was proposed to enforce access control on Android. Since a full integration of the ASM framework is beyond the scope of FutureID, this chapter will elaborate scenarios of how the ASM framework can be utilized for FutureID.

ASM provides a modular system to enhance controlled access to different parts of the Android framework. In this section we will explore how the ASM framework can be integrated with FutureID elements.

7.1 Protecting access to software credentials

In the FutureID scheme, users can use different types of credentials to, for example, create a signature or authenticate to a Service Provider. These credentials can be contained inside a SmartCard but also saved into a file on the device.

One of the proposed integration scenarios is that the ASM framework, using the hooks to the file system, restricts the access to the software credentials so that only the FutureID client is able to access to this file, and additionally creates a dialog prompting the user to confirm such access (see Figure 11).

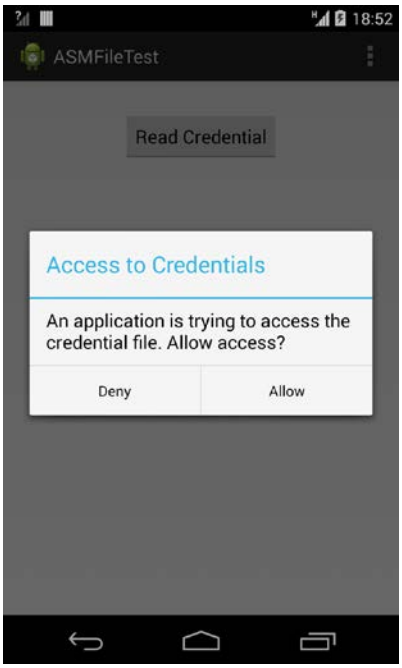

Figure 11: ASM callback warning the user of the access.

This way, the user will be aware that an application is accessing the credentials saved on the device and will be able to control this access and even deny it if this was unexpected.

7.2 Integration with OpenMobile API

As explained in section 5, the OpenMobile API has been integrated with the FutureID architecture to enable access to secure elements in Android platforms. Another proposed integration scenario is to protect the access to the secure element using the Android Security Modules framework. To make this integration possible, ASM was updated to the latest version of Android supported at the moment by the SEEK for Android project, which was the 4.4.4 version, as explained in section 6.1.1 of deliverable D35.3 [1].

In this scenario the NFC plugin-terminal is used to provide the FutureID client with an access to a contactless smartcard. In order to secure this access and in a similar way as with the software credentials, the ASM framework is used to give the user the choice on whereas to allow the access or deny it (Figure 12).

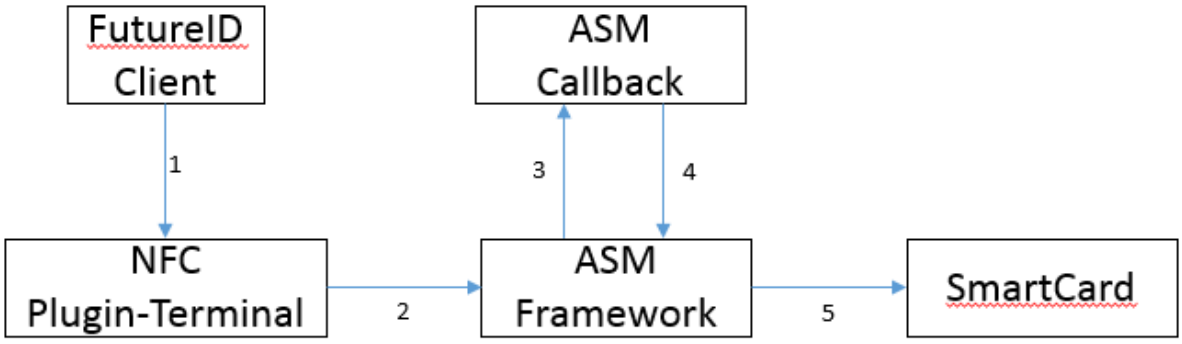

Figure 12: Integration scenario between OpenMobileAPI and ASM

When the FutureID client wants to access the Smartcard, it will contact the NFC plugin-terminal. The ASM framework will detect this access and will call the correspondent callback method that will ask for interaction from the user, for example a switch being enabled beforehand or a dialog as shown in the example before.

8. Outlook/Conclusions

This deliverable documents that it is possible to add security enhancing features to the client device platform and to integrate them into the FutureID client architecture. This additional security allows for a better protection of credentials, especially those outside of secure elements, and an easier access over various transport channels like NFC, USB or the SIM card. Supported by this infrastructure, mobile ID solutions as well as smartcard based use cases are easier to roll out and to use.

The integration of the OpenMobile API with plugin terminal extension supports the establishment of a single and thus better to control channel to secure elements on various form factors. Using a contactless or dual interface card via NFC, a secure USB token or an applet on a SIM card can be managed over one dedicated channel. This solution comes close to the system architecture initially envisioned in deliverable D 35.2 [6].

By using Mandatory Access Control (MAC) mechanisms, additional support of software-based credentials can be achieved. Without this protection these credentials would be especially vulnerable to cloning attacks and thus a potential identity theft. However, Access Control alone will not prevent every type of cloning attack and therefore needs to be embedded into additional security measures for protecting software-only credentials.

Using a Trusted Execution Environment (TEE), wherever available, is a reasonable approach for significantly enhancing security compared to software-only solutions. This work has shown that TEEs can reasonably be integrated into the FutureID client architecture by providing security critical services from inside a TEE. This has been demonstrated for the signature use case but it can be extended to other use cases as well, including the use of software-based credentials and Attribute Based Credentials (ABCs).

All discussed solutions will have an impact beyond the FutureID project since they can be integrated into other security critical use cases as well. This can include mobile payment related applications as well as strong authentication, access control (physical and logical) and authorization of transactions.

9. Bibliography

[1] FutureID, *WP 35 - Trustworthy Client Platform, D35.3, Implementation of the Security Relevant Modules for the Selected Platform,* 2015.

[2] FutureID, *WP 31 - Interface Device Layer, D31.2, Interface and module specification and documentation,* 2013.

[3] FutureID, *WP 31 - Interface Device Layer, D31.3, Implementation of the IFD Service for selected Platforms,* 2014.

[4] "SEEK for Android," Google, 2014. [Online]. Available: https://code.google.com/p/seek-for-android/wiki/AddonTerminal.

[5] FutureID, *WP 32 - eID Serivces, D32.6, Tool for automated creation of CardInfo files,* 2014.

[6] FutureID, *WP 35 - Trustworthy Client Platform, D35.2, Interface and Module Specification and Documentation,* 2014.

[7] FutureID, *WP 32 - eID Services, D32.5, Implementation of protocol-specific modules for selected protocols,* 2015.

[8] FutureID, *WP 32 - eID Services, D 32.2, Interface and Module Specification and Documentation,* 2014.