WP31 - Interface Device Service

D31.3 - Implementation of the IFD Service for selected platforms

Document Identification	
Date	25.04.2014
Status	Final
Version	1.1

Related SP / WP	SP3 / WP31	**Document Reference**	D31.3
Related Deliverable(s)	D31.1, D31.2	**Dissemination Level**	PU
Lead Participant	G&D	**Lead Author**	Frank-Michael Kamm
Contributors	Frank-Michael Kamm (G&D), Daniel Albert (G&D), Moritz Horsch (TUD), Christof Rath (TUG), David Derler (TUG), Tobias Wich (ECS), Detlef Houdeau (IFAG)	**Reviewers**	Omar Almousa (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

Executive Summary

This deliverable describes the implementation of the Interface Device (IFD) Layer of the FutureID client. The implementation is done in Java and supports the Operating Systems Windows, Linux, Mac and Android as well as the terminal interfaces PS/SC [1], NFC [2] [3], and Open Mobile API [4].

The IFD provides a common interface for external applications to use arbitrary card terminals and smart cards. The IFD API is specified in D31.2 [5] and is based on the ISO/IEC 24727-4 [6] and TR-03112-6 [7]. The IFD comprises the SCIO API which provides a common interface for card terminals and smart cards and abstracts from platform-dependend interfaces like the Java Smart Card I/O [8] and the Transport API of the Open Mobile API [4].

Furthermore, the IFD comprises an IFD Proxy which allows running multiple IFDs simultaneously. For instance, a mobile device may run a first IFD using Open Mobile API, a second IFD using Android NFC, and a third IFD using PC/SC to access a local USB card terminal. For other components the IFD Proxy acts as a normal IFD and provides a uniform API. Internally it starts multiple Sub-IFDs using different terminal interfaces (NFC, Open Mobile API, PC/SC).

This deliverable document contains the main considerations and design choices when implementing the IFD layer. It also addresses influences of external specifications on possible future implementations of the IFD layer (chapter 5). The actual source code which results from the implementation is stored in the FutureID Redmine code repository.

Document Information

History

Version	Date	Author	Changes
0.1	07.02.2014	F.-M. Kamm	Initial version
0.11	14.03.2014	Moritz Horsch	Update template
0.12	20.03.2014	Moritz Horsch, Tobias Wich	Added section General Implementation
0.13	21.03.2014	Moritz Horsch	Added section Executive Summary
0.14	24.03.2014	Detlef Houdeau	Added section on TPM 2.0 and FIDO
0.2	27.03.2014	F.-M. Kamm	Added OpenMobile section, adjustment of TPM/FIDO section.
0.3	06.04.2014	C. Rath	Integration Proxy Section
1.0	08.04.2014	F.-M. Kamm	Finalization reviewer version
1.01	17.04.2014	Moritz Horsch	Updated General Implementation section
1.1	22.04.2014	F.-M. Kamm	Integration reviewer comments
1.1	25.04.2014	F.-M. Kamm	Finalized for submission

Table of Contents

1. Introduction

1.1 Scope

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

One important aspect of achieving that goal is to support existing credentials like smart cards, secure elements, and hardware/software tokens. The Interface Device (IFD) service provides a common interface for communication to such credentials. Therefore, users, developers, and applications must not consider how such credentials are integrated or addressed. The IFD abstracts from the specific interfaces and physical properties like contactless interfaces. This provides platform independency and interoperability for applications.

The scope of this document is to describe the implementation of the IFD as specified in interface and module specification D31.2.

1.2 Outline

This document is structured as follows: Section 2 provides a brief overview of the IFD architecture and the general implementation. Section 3 explains the proxy support of the IFD and describes its application flow and internal message handling. Section 4 describes the handling of the Open Mobile API limitations, while section 5 discusses the potential impact of external specifications (TPM 2.0 and FIDO) on the IFD layer.

1.3 Terminology

1.3.1 Key Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [9].

1.3.2 Abbreviations and Notations

APDU	Application Protocol Data Unit
API	Application Programming Interface
ATR	Answer To Reset
FIDO	Fast IDentity Online
IFD	Interface Device
JCE	Java Cryptography Extension
NFC	Near Field Communication

PC/SC	Personal Computer/Smart Card
SAL	Service Access Layer
SCIO	Smart Card Interface Input Output
TCG	Trusted Computing Group
TPM	Trusted Platform Module

2. General Implementation

The Interface Device (IFD) service provides a common interface for communication with arbitrary credentials. In detail, the IFD encapsulates card terminals, smart cards and secure elements and provides an interface to easily access such devices.

Figure 1 illustrates the architecture of the implemented IFD. On top, the IFD provides an Application Programming Interface (API) to access the service. The API is specified in D31.2 [5] (Section 4) and provides a common interface for communication with arbitrary smart cards. Among other things it allows retrieving information about connected card terminals and smart cards as well as functions to connect to cards and to exchange data.

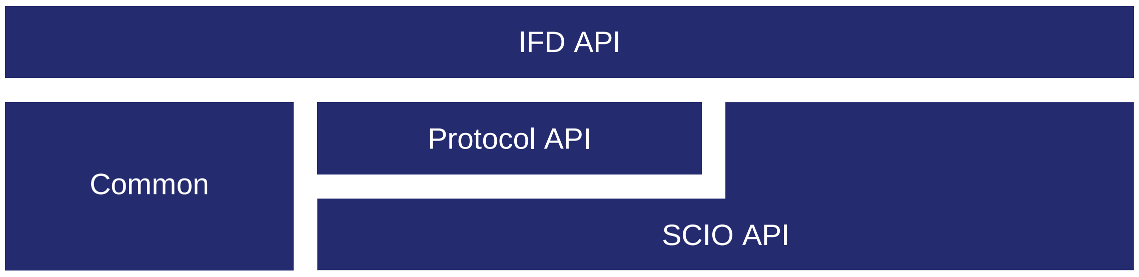

Figure 1: IFD Architecture

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 [1], NFC [2] [3] or Open Mobile API [4]. Further details about the different components are available in D31.2 [5].

2.1 IFD API

The IFD API provides the functionality as specified in D31.2 [5] (Section 4). It provides an interface according to ISO/IEC 24727-4 and BSI TR-03112-6 and is intended to be used by higher level components such as the Service Access Layer (SAL) which is defined in D32.3 (Section 5) or remote components.

The main purpose of the IFD interface is to provide a message oriented abstraction of the lower level SCIO API discussed in the next section. By doing so it is, besides direct access, possible to access the IFD API in a network transparent way over SOAP, PAOS and similar channels. The functionality is to retrieve information about connected card terminals and smart cards and to exchange information with either of them.

2.2 Protocol API

The Protocol API provides means to execute a secure messaging establishment protocol. By executing such a protocol each following messages from and to the smart card are automatically secured by the established channel.

2.3 SCIO API

The SCIO API (Smart Card Interface Input Output, SCIO) provides an interface for common card terminals and smart cards. It abstracts from platform-depended interfaces like the Java Smart Card I/O [8] and the Transport API of the Open Mobile API [4]. The interface comprises the functionality of both APIs and provides additional convenience functions.

The SCIO API contains a number of interfaces for terminals and smart cards, for instance, `SCIOCard`, `SCIOChannel`, `SCIOTerminal` and so forth. The functions define the common set of functions specified by the Java Smart Card I/O and the Transport API of the Open Mobile API. Furthermore, the API comprises convenience classes like `ATR` and `CardCapabilities`.

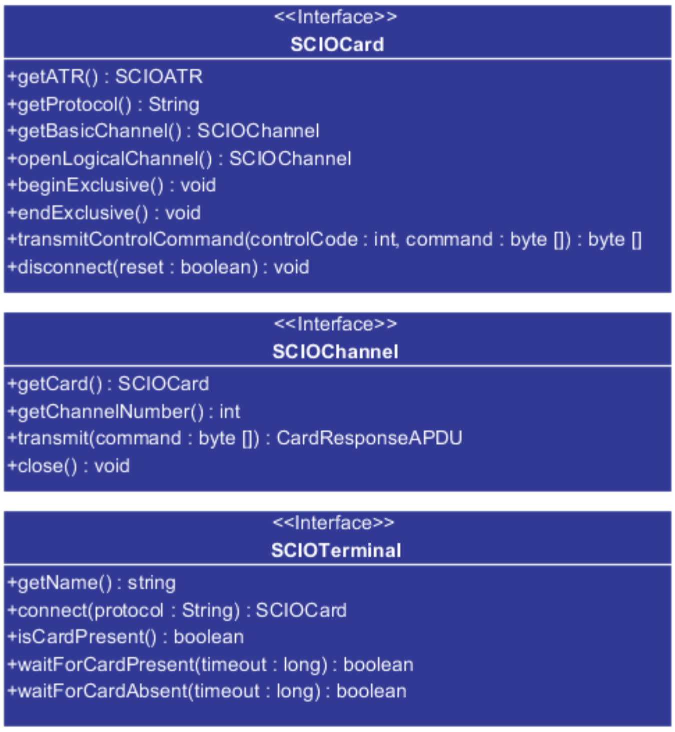
<<Interface>>
SCIOCard
+getATR() : SCIOATR
+getProtocol() : String
+getBasicChannel() : SCIOChannel
+openLogicalChannel() : SCIOChannel
+beginExclusive() : void
+endExclusive() : void
+transmitControlCommand(controlCode : int, command : byte []) : byte []
+disconnect(reset : boolean) : void
<<Interface>>
SCIOChannel
+getCard() : SCIOCard
+getChannelNumber() : int
+transmit(command : byte []) : CardResponseAPDU
+close() : void
<<Interface>>
SCIOTerminal
+getName() : string
+connect(protocol : String) : SCIOCard
+isCardPresent() : boolean
+waitForCardPresent(timeout : long) : boolean
+waitForCardAbsent(timeout : long) : boolean

Figure 2: SCIO API

Document name:	Implementation of the IFD Service for selected platforms					**Page:**	7 of 19
Reference:	D31.3	**Dissemination:**	PU	**Version:**	1.1	**Status:**	Final

3. Implementation of Proxy Layer

When using different physical interfaces, like contact readers and NFC readers, different IFD implementations will be used. To transparently abstract multiple IFD implementations towards the SAL an IFD proxy layer has been introduced, as defined in D31.2 [5].

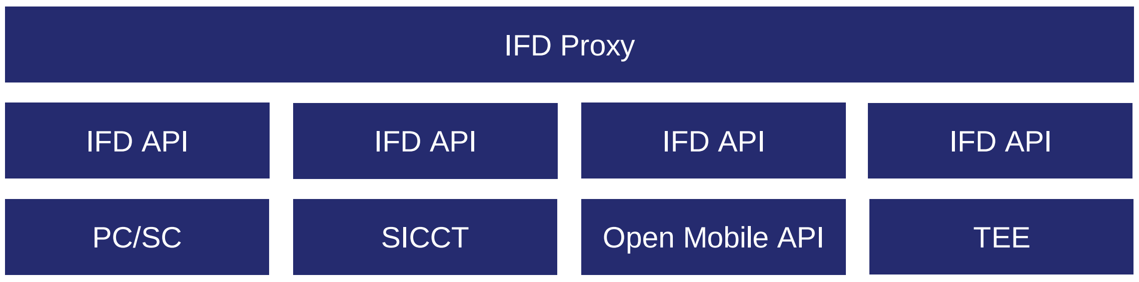

Figure 3: IFD Proxy

The basic layout is shown in Figure 3. Modules that use the IFD proxy shall be unaware that the actual execution is handled by one of the registered sub IFDs. To achieve this, two types of calls have to be distinguished. On the one hand, stateless requests like `ListIFDs` which lists all available interface devices. On the other hand, we have stateful requests like `Transmit` which are bound to a session identified, for example by the `SlotHandle`.

For the stateless requests, all sub IFDs have to be contacted using the same request and the respective results must be combined in a single result. It is important to ensure that the results are unambiguous when combining them. Several requests require a unique name of the actual interface device, that is, the card reader, for example. It is ensured that this name is unique within a single sub IFD, however, chances are, that two different sub IFDs use the same naming scheme. Therefore, it is important to prefix/suffix the name of an interface device, as given by the sub IFD, with an identifier of the sub IFD itself. Thus, the uniqueness of an interface device is ensured after merging all the results.

For the stateful requests it is important to keep mappings between the sub IFD and the respective session identifiers, like slot handles. For successive calls, these mappings are used to forward the request to the correct IFD. The current implementation uses two such mappings. The first keeps track of the context handles of the sub IFDs and the second of the slot handles for pending transactions.

In addition to the aforementioned request types, special treatment is required for the `Wait` request. For this request the handling as foreseen in D31.2 turned out not to be sufficient. In D31.2, the expected behaviour, if no specific IFD was requested, was to call `Wait` sequentially on all sub IFDs. As soon as the first IFD returned successfully, `Cancel` should have been called on all remaining IFDs. Since the goal of the IFD proxy is to be transparent, this approach was not sufficient, as the semantic of the `Wait` request would have changed.

The current implementation calls `Wait` asynchronously on all sub IFDs. When all sub IFDs returned within the given timeout, the respective results are merged into a combined result.

For this implementation a new Maven module `ifd-proxy` was created under `FUTUREID_ROOT/ifd/ifd-proxy`. The main class of this implementation is `org.openecard.ifd.proxy.IFDProxy`. The currently only implementation of a sub IFD is based on the PC/SC library. It was renamed from `IFD` to `CoreIFD` and a new interface `org.openecard.common.ifd.IFD` was added to the `ifd-common` project as new abstraction. Finally, the web service endpoint was moved from `CoreIFD` to the `IFDProxy`.

4. OpenMobile API Related Topics

4.1 Handling of limitations on mobile devices with OpenMobile API

As has been pointed out in deliverable D31.2 (Interface and module definition), there are several limitations on mobile devices using the OpenMobile API as IFD layer. While some of these limitations result from the OpenMobile API itself and can only be circumvented when the API is modified or extended, others are rather a limitation from the mobile device.

In the following sections it will be briefly discussed how these limitations are handled in an actual client implementation.

4.1.1 GetIFDCapabilities

The method `GetIFDCapabilities` is not supported by the OpenMobile API. This is a limitation coming from the mobile device itself, rather than from the OpenMobile API. Currently, standard mobile devices like Android-based smartphones or tablets do not offer a secure input (e.g. for PIN entry) or a secure output (e.g. secure display for documents to be signed). While such a solution may exist in future, based on a Trusted Execution Environment and the corresponding GlobalPlatform specifications for a trusted user interface (see for example: deliverable D34.4), current devices do not provide this capability.

Therefore, the corresponding IFD API layer (or – if used – the IFX proxy layer) need to detect this limitation and provide a “`not supported`” response to the SAL.

If in future versions of the OpenMobile API external card readers, like a class 3 reader with PIN pad and display, should be made addressable via the OpenMobile API as well, the API would have to be extended to provide the full `GetIFDCapabilities` functionality.

4.1.2 Transactions

Currently, this limitation is not regarded as critical since no known application uses this concept anymore. The corresponding functionality on the secure element is typically executed in an atomic way, making the use of transactions obsolete.

4.1.3 Transmit and Channel Management

The OpenMobile API does not support channel management APDUs. Therefore, the IFD API layer (or, if available the IFD proxy layer) must take action to compensate for this limitation. When the application sends all APDUs on the basic channel, the IFD API or proxy layer must actively open a logical channel and forward the APDUs without further modification. On the other hand, when the application uses logical channels, the IFD API or proxy layer must filter the corresponding APDUs and translate them into corresponding APDUs on the defined channel.

4.2 JCE provider management with OpenMobile API

A significant difference between Android as a mobile OS and stationary solutions like Java on a Windows OS is the handling of Java Cryptography Extension (JCE) providers. While on desktop Java platforms the JCE providers can be registered and made available for all applications (e.g. by providing a DLL), on Android the JCE provider comes with the apk-file of the application and is only available for the corresponding application. As a consequence, several providers can exist in parallel, each only visible for the respective application.

Since the JCE interface can be regarded as top-level interface for calls of cryptographic methods by an application, it is reasonable to have the JCE interface above an IFD layer. In this case there is no further encapsulation required below the IFD layer.

The application will typically request a cryptographic method with certain properties, like a specific certificate. When several secure elements are available on one device (e.g. a SIM card and a µSD card), it is beneficial to establish one JCE provider for each endpoint. In this case the corresponding IFD layer only has to forward the communication to the secure device. Figure 4 shows the resulting architecture modification for mobile devices using the OpenMobile API.

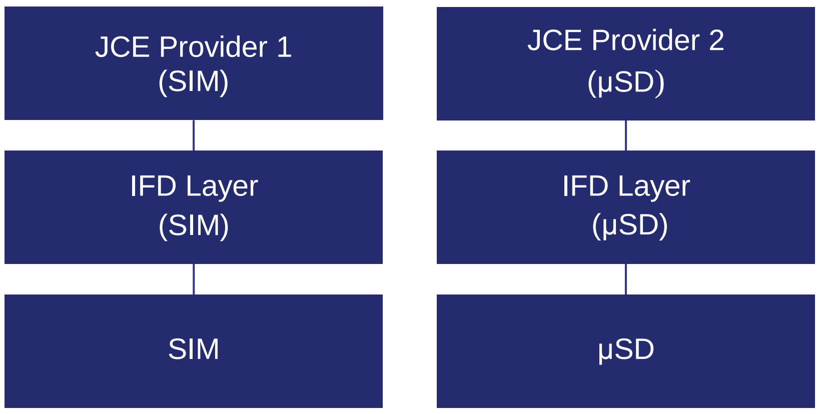

Figure 4: JCE provider concept on mobile devices having several secure elements available

5. Impact of external Specifications on IFD Layer

During the course of the project, several external specifications have been published or updated that could have an impact on the FutureID client and thus on the IFD layer and other client components. Especially if FutureID decides to be compliant with these specifications or to integrate use cases that require compliance with these specifications, it is worth analyzing the impact on the client architecture and/or requirements.

Therefore, the following sections briefly describe the main features of the most relevant new specifications in this area, i.e. the Trusted Platform Module (TPM) 2.0 and the FIDO alliance specification. In addition, their impact on client specifications is briefly discussed.

5.1 Trusted Platform Module (TPM) 2.0 Specifications

The Trusted Platform Module (TPM) is a secure crypto-processor designed to secure hardware by integrating cryptographic keys into devices. TPM is based on an international standard created from a computer industry consortium called Trusted Computing Group (TCG), with 135 member organizations. Some members of FutureID are also member of TCG. The TCG specification has been standardized and published under ISO/IEC 11889 in CY 2009. The first specification covers the version TPM 1.2. The last revision, number 116 was published in March 2011.

The main use cases for TPMs are today

- Platform integrity
- Disc encryption
- Password protection
- Digital rights management
- Protection and enforcement of software licenses

The main applications that use TCG technology are today

- Machine identity
- VPN/wireless access
- Data at reset
- SCADA
- Clientless endpoint meta data management
- Hardware based cloud subscriber management
- Trusted execution

The TPM 1.2 is used by nearly all PC and notebook manufactures. The operation systems from MICROSOFT, WINDOWS VISTA, WINDOWS 7 and WINDOWS 8 used TPM1.2 in conjunction with the included disk encryption software named BitLocker. TPM 2.0 is planned for PCs which have WINDOWS 8.1 hardware certification. MICROSOFT has dated this generation for the 1st of January 2015. This means production development, qualification and certification must be done by end of CY 2014.

5.1.1 TPM 2.0 Improvements

Based on the circumstance, that the TPM 1.2 specification is round 10 years old a new specification is needed for

- Increasing the security
- Make the TCG technology more user friendly
- Address other devices than PCs/notebooks

Increasing the security (stronger crypto):

TPM 1.2 is defined with

- SHA_1
- RSA_2048

Today SHA_1 is not anymore classified as secure. For the bit length of asymmetric RSA-keys the US NIST has published an SP800-57 (table 2), defining the following classes:

- ≥ 2048bit key length: secure
- ≥ 3072 bit key length: secret
- ≥ 7680 bit key length: top secret

Some SOGIS bodies like the BSI in Germany see the end of RSA_2048 by CY 2017. Then the next higher level should be implemented, which means RSA_3072 or ECC_256 or higher.

Increase performance (faster crypto):

RSA key lengths above 2048 byte need too much computing time. The change from RSA to ECC could avoid this aspect. Based on this performance topic TPM 2.0 is using ECC instead of RSA.

The relevant standards on asymmetric encryption are published under a broad range of documents such as

- ISO/IEC 14888-2 and others
- PKCS#1
- IEEE 1363
- IETF RFC 6090 and others
- In the US: ANSI X 9.62, FIPS 186-3
- In Germany: BSI TR 03111
- In South Korea: KISA EC KCDSA

More security agility (flexible crypto):

TPM 2.0 defines two pillars of cryptography:

- Mandatory elements
- Optional elements

5.1.2 Hierarchies

In TPM 1.2 the “child” keys were encrypted using the asymmetric parent key. Every key load was an asymmetric key operation. In TPM 2.0, the parent has an additional seed value that is used to generate the symmetric keys for protecting the child keys. This means that for encryption and HMAC keys, each set of keys is unique to the child and loading a child is a pair of symmetric operations (HMAC and decryption). In TPM 2.0, the asymmetric part of a storage key used for secret sharing a key import, the activating an identity and the starting an authorization session.

5.1.3 Authorization elements

TPM 1.2 has only three authorization elements, a) authorization value, b) PCR state and c) “locality” – hardware privilege level. TPM 2.0 has 12 authorization elements, with a) authorization value, b) locality, c) asymmetric signature, d) symmetric shared secret, e) time limited, f) specific command, g) PCR state, h) “physical presence”, i) specific objects, j) duplication, k) NV written and l) contents of NV.

5.1.4 Field updates

In April 2014 ANSSI in France and BSI in Germany will publish a new Protection Profile (PP) which addresses field updates. This functionality is needed in case that the lifetime of the embedded security device is long and the asymmetric encryption is not more secure enough. The new PP called PP0084 addresses primarily the hardware, i.e. the secure anchor. It is expected, that the next logical step will be to create and publish another PP, which address the software, like the OS, the COS or the eCard API.

5.1.5 Impact of the TPM 2.0 on the FutureID client

First products on TPM 2.0 in the market are expected in CY 2015. Due to its flexibility, the FutureID platform should be able and flexible enough to address the most important changes on the security domain as well as in hierarchies, and in authorization elements. In case that a TPM shall be addressed by FutureID (which is currently not planned), the respective protocol and plugin modules would have to be modified to address the 2.0 version of the TPM specifications. Beyond these modifications, there is no further impact on the IFD design expected.

5.2 FIDO Alliance and specifications

The following chapter describes the scope of the FIDO Alliance, the relevant specifications and the possible impact on the FutureID client.

5.2.1 Motivation and targets

Back in June 2012, the Fast IDentity Online (FIDO) Alliance was formed by 6 companies with PayPal, Lenovo, Nok Nok Labs, Validity Sensor, Infineon and Agnitio as a Non-Profit-Organization (NPO). After launching FIDO in 2013 many companies have joint this NGO, including big players like Google.

The mission of FIDO is to change the nature of existing online authentication by

- Developing technical specifications that define an open, scalable, interoperable set of mechanism that reduce the reliance on passwords to authenticate users,
- Operating industry programs to help to ensure successful worldwide adoption of the specifications,
- Submitting mature technical specifications to recognized standards development organizations for formal standardization.

FIDO has collected and published two specifications:

- Passwordless experience (UAF standard)
- Second factor experience (U2F standard)

These new specifications for security devices and browser plug-ins will allow any website or cloud application to interface with a broad variety of existing and future FIDO-enabled devices that the user has for online authentication. The most interesting part for FutureID is the U2F specification which is complementary to the U2F.

5.2.2 Universal 2nd Factor

The second factor FIDO experience is based on the Universal Second Factor (U2F) protocol. This experience allows online services to augment the security of their existing password infrastructure by adding a strong second factor for user login. The user initially logs in with a username and password as before. The service can also prompt the user to present a second factor device at any time it chooses. The strong second factor allows the service to simplify its passwords without compromising security. During the registration and authentication, the user presents the second factor by simply pressing a button on a USB device or tapping over NFC. The user can use their FIDO U2F device across all online services that support the protocol leveraging built-in support in web browsers.

5.2.3 FIDO U2F Device

From Google's points of view, the defined FIDO U2F device for the two factor authentication can be a small USB-dongle with embedded secure element and with an NFC- and a Bluetooth interface. This dongle can be used as universal two factor key for plug in and can be used in many services. It should be a driverless USB on Win, Mac, Linux, and Chrome OS.

With this dongle a direct access from the browser would be supported. No client middleware must be installed. A simple JavaScript API, based on "Create Key Pair" and "Sign" is needed. The same API can be used for Android.

The concept for the web site ("open strong security") is as follows:

- Open: not proprietary, multiple vendors, no central service required;
- Self-provisioned: no pre-seeding required; "bring your own token" would be possible;
- Strong security: non phishable;
- Strong privacy: one site cannot use credentials issued by another site;

5.2.4 FIDO U2F protocol

The core idea of FIDO is to use standard public key cryptography. This means

- User's device generates new key pair, gives public key to server;
- Server ask user's device to sign data to verify the user;
- One device, many services;

This means lots of refinement for this to be consumer facing. Examples

- Privacy: Site specific keys; no unique ID per device,
- Security: no phishing, man-in-the middle,
- Trust: verify who made the device,
- Pragmatics: affordable today, ride hardware cost curve down,
- Speed for user: fats crypto in device (elliptic curve),
- Feature growth: server – device encrypted communication; future trusted display.

Google has started in October 2012 with about 2.000 dongles in daily use. These dongles support Google intranet single-sign-on (SSO) based on U2F. The Google intranet is directly on the web. Thus, this SSO is just a web login. The dongle is using a Chrome Extension, not the final Java Script API. This approach is an integration on "lower" level.

5.2.5 Bridge between FIDO and FutureID

FIDO is designed as a universal protocol with various possibilities for hardware interfaces, like USB, NFC and Bluetooth with focus on modern consumer web services. On the other hand FutureID focuses on two-factor authentication with electronic ID-tokens issued from the public sector. FIDO does not necessarily require client middleware, while FutureID is based on client middleware with the focus on interoperability and open platform.

Thus, there are differences visible in the general architecture and setup. On the other hand, the FutureID architecture is flexible enough and future-proof to integrate new protocol approaches like the FIDO protocol. FIDO is still a young approach, starting on specification in CY 2013. It is not clear at this point which type of hardware dongle will be the dominating one in future. The FIDO approach allows enough flexibility to use existing hardware, like secure element, as well. Therefore it can be envisioned, that a FIDO-compatible applet on a secure element or smartcard could be integrated into a FIDO authentication scheme.

FutureID would easily be able to address this use case by adding the corresponding FIDO protocol support to the Universal Authentication service and the client eID services.

5.3 Conclusions

In conclusion, based on the current modular IFD architecture it seems feasible for future IFD and client implementations to address the needs of the TPM 2.0 and FIDO specifications, whenever desired. Currently, no TPM2.0 and FIDO support is planned for the two FutureID demonstrators. Yet, no fundamental issue for providing this support with the existing IFD architecture is currently visible.

6. Bibliography

[1] PC/SC Workgroup, *PC/SC Workgroup Specifications,* Version 2.01.11, Part 1 - 10, 2012.

[2] ISO/IEC, *Information technology - Telecommunications and information exchange between systems - Near Field Communication - Interface and Protocol (NFCIP-1),* International Standard, ISO/IEC 18092, 2003.

[3] ISO/IEC, *Information technology - Telecommunications and information exchange between systems - Near Field Communication Interface and Protocol -2 (NFCIP-2),* International Standard, ISO/IEC 21481, 2005.

[4] SIMalliance, *Open Mobile API specification,* Version 2.03, 2012.

[5] FutureID, *WP31 - Interface Device Service, D31.2 - Interface and module specification and documentation,* 2013.

[6] ISO/IEC, *Identification cards - Integrated circuit card programming interfaces - Part 4: Application programming interface (API) administration,* International Standard, ISO/IEC 24727-4, 2008.

[7] Bundesamt für Sicherheit in der Informationstechnik (BSI), *eCard-API-Framework - IFD-Interface,* Version 1.1.2, Part 6, 2012.

[8] Oracle Corporation, *Java Smart Card I/O API,* JSR 268, http://jcp.org/en/jsr/detail?id=268, 2006.

[9] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels,* RFC 2119, 1997.

[10] TeleTrusT Deutschland e.V.: SICCT - Secure Interoperable ChipCard Terminal. Version 1.6 (2009)

[11] Trusted Computing Group: TPM Main. Version 1.2, Part 1 - 3 (2011)

[12] Don, B., David, E., Gopal, K., Andrew, L., Noah, M.: Simple Object Access Protocol (SOAP) 1.1. (2000)

[13] Robert, A., John, K.: Liberty Reverse HTTP Binding for SOAP Specification. Liberty Alliance Specification, Version 2.0 (2006)

[14] Liberty Alliance Project: Liberty Reverse HTTP Bindin for SOAP Specification. Version 2.0, http://www.projectliberty.org/liberty/content/download/909/6303/file/liberty-paos-v2.0.pdf