System Architecture Trustworthy Client Platform

Interface and Module Specification and Documentation

Document Identification	
Date	24/01/14
Status	Final
Version	1.2

Related SP / WP	SP 3/ WP 35	**Document Reference**	D35.2
Related Deliverable(s)	D 35.1	**Dissemination Level**	PU
Lead Participant	G&D	**Lead Author**	Dr. F.-M. Kamm
Contributors	TUD, TUG	**Reviewers**	KUL, CA

This document is issued within the frame and for the purpose of the *FutureID* project. This project has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement no. 318424.

1. Executive Summary

This deliverable concentrates on developing a system architecture for a trustworthy client platform. For this purpose, various methods and elements for secure execution environments and their access control are discussed. As another building block, protocols for trust measurement and trust establishment in heterogeneous systems are analysed. As an outcome of this discussion, system architectures for stationary (PC-like) and mobile devices for establishing a trustworthy platform are proposed.

For an identity federation system like FutureID it is essential to handle eID data and assurance claims in a trustworthy manner, preferably with measurable trustworthiness. A validated trustworthiness of the client platform is one prerequisite to achieve this. When eID data or metadata can be modified by malware on the client or when the eID token can be accessed by malware without the user consent (e.g. by intercepting access PINs), the FutureID backend cannot rely on the asserted credential claims anymore. Therefore a “chain of assurance” has to be established from the eID token to the FutureID backend. The proposed system architecture is designed to support this goal.

Developing a universal system architecture is quite challenging since the boundary conditions on PC-like systems and mobile devices differ significantly. In addition, there are different architecture options for mobile as well as stationary systems depending on the hardware or software stack being used. For this reason, several architecture options are presented for both device classes and the achievable level of trustworthiness (as defined in deliverable D35.1) is discussed (chapter 7).

The proposed architectures are based on (depending on the desired assurance level) Trusted Execution Environments (TEEs), Secure Elements that can be accessed by the TEEs, and a trusted User Interface (UI) which is also provided or secured by a TEE. The preferred choice for an actual implementation will depend on the availability of solutions in the area of software-based Trusted Execution Environments. As long as fully hardware-anchored TEEs are not yet available, a bridging solution based on virtualization can also be considered.

Concepts of trusted protocols make it reasonable to derive the actually achieved trust level on the server side (i.e. the FutureID backend) while the actual integrity determination occurs on the client side.

2. Document Information

2.1 Contributors

Name	Partner
Frank-Michael Kamm, Jan Eichholz, Alexander Summerer	G&D
Jon Rios, Christoph Busold	TUD
Christof Rath, Martin Pirker, Peter Lipp	TUG

2.2 History

Version	Date	Author	Changes
0.1	20.12.2013	F.-M. Kamm, J. Eichholz, A. Summerer	Initial part G&D
0.2	20.12.2013	J. Rios	TUD part added
0.3	07.01.2014	M.Pirker, C. Rath	TUG part 1 added
0.4	08.01.2014	F.-M. Kamm	Executive Summary
0.5	09.01.2014	F.-M. Kamm	Interfaces mobile devices
0.6	10.01.2014	M. Pirker, C. Rath	TUG part 2 added
0.7	10.01.2014	F.-M. Kamm	Conclusions added
1.0	10.01.2014	J. Rios, C. Busold	Interfaces PC platform
1.1	23.01.2014	F.-M. Kamm	Integration of 1st rev. comments
1.2	24.01.2014	F.-M. Kamm	Integration of 2nd rev. comments

2.3 Table of Acronyms

APDU	Application Programming Data Unit
API	Application Programming Interface
ARA	Access Rule Applet
CPU	Central Processing Unit
eID	Electronic identity
IFD	Interface Device Layer

IMC	Integrity Measurement Collector
IMV	Integrity Measurement Verfier
MTM	Mobile Trusted Module
NFC	Nearfield Communication
OS	Operating System
OTP	One-Time Password
PCR	Platform Configuration Register
PIN	Personal Identification Number
PIV	Personal Identification Verification
RTR	Root of Trust for Reporting
RTS	Root of Trust of Storage
SE	Secure Element
SEAC	Secure Element Access Control
SIM	Subscriber Identity Module
TA	Trusted Application
TCG	Trusted Computing Group
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TNC	Trusted Network Connect
TPM	Trusted Platform Module
UI	User Interface
UICC	Universal Integrated Circuit Card

3. Table of Contents

1. Executive Summary 1

2. Document Information 2
2.1 Contributors 2
2.2 History 2
2.3 Table of Acronyms 2

3. Table of Contents 4

4. Project Description 6

5. Software Architectures 7
5.1 Assurance and Security Validation 7
5.2 Secure Execution Environments 7
5.2.1 Trusted Platform Module (TPM) 7
5.2.2 Mobile Trusted Module (MTM) 9
5.2.3 TEE 9
5.2.4 Secure Element 10
5.2.5 Virtualization 11
5.3 Secure Element Access Control Enforcement Systems 12
5.4 Secure Peripherals Access 14

6. Protocols 17
6.1 Protocols for real-time trust measurement 17
6.2 Protocols for trust establishment in a heterogeneous environment 18
6.3 Protocol Interfaces 20

7. System Architectures 21
7.1 Introduction 21
7.2 System Architecture PC 21
7.2.1 General Architecture 22
7.2.2 Platform-specific Variants 22
7.2.3 Trusted Execution Environments on PC-based Platforms 25
7.2.4 Interfaces 26
7.3 System Architecture Mobile Device 27
7.3.1 General Architecture 27
7.3.2 Platform-specific Parts 29
7.3.3 Interfaces 29

8. Summary/Conclusions 31

9. References 32

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, allowing them to choose 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. Software Architectures

5.1 Assurance and Security Validation

The system architecture for mobile and PC-based devices developed in this deliverable should allow the FutureID client to validate the trustworthiness of the client platform. This functionality is known as attestation of the platform state, which can be divided into two categories, local and remote. *Local attestation* means that the platform validates itself or attests its state to a locally connected component. The FutureID client might, for example, want to ensure that a PIN for an eID card is entered via a trusted user interface and cannot be intercepted by the local operating system, which requires the device to be in a certain state. *Remote attestation* means that the platform attests its state to a remote verifier. A server, for example, might want to validate the platform security of a user device before granting access to certain functionality.

Furthermore, it should be possible to easily validate the security of the system architecture itself. Therefore the architecture should be developed with security design principles in mind. This includes that the design and the implementation are open source, so that they can be evaluated by a large community.

A validated trustworthiness of the client platform is also a prerequisite for the trustworthy handling of assurances of eID data and metadata. When this information can be modified by malware on the client or when the eID token can be accessed by malware without the user consent (e.g. by intercepting access PINs), the FutureID backend cannot rely on the asserted credential claims anymore. Therefore a “chain of assurance” has to be established from the eID token to the FutureID backend. This chain requires a corresponding system architecture (which will be discussed in chapter 7) which is built on the basis of secure environments (discussed in section 5.2 - 5.4) and corresponding protocols (discussed in chapter 6).

5.2 Secure Execution Environments

When considering the architecture of a trustworthy client platform, it is worth looking at corresponding secure elements and secure execution environments that can act as a trust anchor or that actually provide a trustworthy environment. The following subsections discuss the existing hardware and software elements that can be used to build the foundation of a trustworthy environment.

5.2.1 Trusted Platform Module (TPM)

Within the context of a trustworthy client platform the TPM can have two purposes. In a first use-case it will act as a hardware crypto module so that it can be used to create and securely store

key material, as a random number generator and to create RSA signatures with SHA-1 hashes. In deliverable D31.2 it was decided to exclude the TPM from the Interface Device Service, since the TPM API does not fit into the ISO/IEC 7816 based architecture of the IFD [D31.2]. Therefore, the TPM has to be accessed directly. For the Java programming language at least two APIs are available. On the base level there is jTSS[1], which is an implementation of the full TCG Software Stack for the Java programming language. Above that and for common use-cases certainly more convenient is the JSR 321[2], the Trusted Computing API for Java.

In a second use-case the TPM can be used to attest the platform integrity. For this, it is crucially important that a chain-of-trust form the hardware via the boot process and the operating system to all running processes can be established. If this chain is broken, or has missing links, an adversary is able to tamper with the TPM measurements and to emulate the platform attestations of an unmodified system. The status of a system is represented as a set of hash values, stored in the Platform Configuration Registers (PCR). A PCR cannot be modified directly, rather a new measurement value is the result of hashing the old register value together with a new measurement. In the acTvSM project [TOE10] the complexity to implement secure boot for an off-the-shelf Debian-Linux has been evaluated. It could be shown that for general-purpose computers with standard operating systems like Microsoft Windows or Linux and settings where the users can change the configuration or are allowed to install software, this approach is not feasible, since every change in the configuration or the executed software or even the order of the execution influences the content of the PCR.

In order to get the complexity to a manageable level the supported systems have to be strictly limited. For example, the systems functionality has to be reduced to the bare minimum required to fulfil the job at hand. Furthermore, only privileged users must be allowed to make changes to the system. Usually that means that the systems are tailored to one specific task with dedicated system administrators to maintain the system functionality. In [PIR12] a system is proposed that uses a striped down Android operating system to offer verifiable and trustable computing capacity to the cloud. As the Android OS was developed for mobile platforms it offers a particular interesting feature, namely, the separation of read-only and writable parts of the operating system. The complete read-only part can be seen as a single blob of data and must be measured only once.

The restrictions required for the computer systems to get to a manageable complexity is, from a usability perspective, nothing that could be imposed on the end users of *FutureID* and, hence, is not feasible for the client platform per se. One solution could be to separate the user agent and

[1] http://trustedjava.sourceforge.net

[2] https://jsr321.java.net

the *FutureID* client and to operate the *FutureID* client on a dedicated platform. For this, the operating system could be the aforementioned measureable Android system with the *FutureID* client as the only running process. The smartcard readers must be attached to the measured platform and also all security relevant operations would be executed on this device. The user agent could still run on the user's standard platform. This raises two issues. First, it must be investigated how to access the *FutureID* client. Currently it is assumed that the client runs on the same machine using a specific port. Furthermore, there is an issue with the user interface of the client. It could be displayed on the user's standard platform by means of remote desktop clients. However, this imposes again a security risk. If the user's standard platform is compromised an attacker might be able to change the frame buffer of the remote desktop. Additionally, the *FutureID* client platform could, of course, use its own display.

On the server-side the approach of [PIR12] could be used to show, in a verifiable way, to other components that a certain system state is in operation. However, since the server-side components of *FutureID* are closed source, outside users can only verify that a certain system is running but cannot guess what the system is doing.

5.2.2 Mobile Trusted Module (MTM)

A Mobile Trusted Module (MTM) is the equivalent to a TPM on mobile platforms, where it defines a Root of Trust of Storage (RTS) and a Root of Trust for Reporting (*RTR*). In contrast to the TPM specification, which explicitly requires a hardware implementation, an MTM can also be implemented in software. The MTM provides trusted resources using a subset of the TPM 1.2 structures and commands. As stated in D31.1 *Requirements Report for Interface Device Service* Section 2.5, MTMs are currently not deployed in any existing mobile device and it does not seem that this situation would change in the near future. This is why MTMs are not considered in the requirements for the IFD service and therefore it does not make sense to include them in the trustworthy client platform.

5.2.3 TEE

The Trusted Execution Environment (TEE) for mobile devices is a trustworthy execution environment that is completely separated from the normal ("rich") operating system context. Mechanisms are implemented to protect the integrity of the TEE software itself with a secure boot process and the integrity of applications running within the TEE context. The separation of the TEE and the rich OS is deeply supported by the mobile device processor architecture (e.g. ARM TrustZone or Intel TXT). The processor provides a dedicated hardware section to run trusted applications and to establish a trusted environment. Additionally, the access to shared resources (e.g. memory) is controlled and strictly separated. A more detailed discussion of the TEE and its properties can be found in Deliverable D35.1, chapter 7.3

In the context of reference architectures for a trustworthy client platform, the potential role of a TEE and the according trust level that can be achieved with it need to be analysed. The applications running in a TEE can be considered as trustworthy since the TEE itself and the application are part of a chain of trust that is verifying the integrity of the respective component. In addition, access to TEE resources is strictly controlled which makes it impossible for other rich OS applications to modify the execution logic of a Trusted Application (TA) in the TEE. Therefore it seems reasonable to consider shifting security critical modules of the client into the TEE.

In contrast to a Secure Element (see section 5.2.4), the TEE does not provide hardware security in terms of tamper resistance against hardware attacks and side channel leakage. In the current form, it also does not have access to dedicated hardware resources - like a dedicated memory – other than the separate processor execution environment. Nevertheless, the TEE can store information in the main memory (or other storage resources) in encrypted form and can control the encryption and decryption by a Trusted Application. Therefore, this information is never accessible in plain text for applications running in the rich OS.

Thus the TEE could support a secure storage space which is resistant against software attacks, as it is requested for trust level 4. On the other hand, a trust level 5 cannot be supported by a TEE only without an additional hardware secure element.

Another important aspect that can be supported by a TEE is a trustworthy User Interface (UI). It can be realized when the TEE can obtain exclusive control of the peripherals drivers. This concept is described further in section 5.4 and deliverable D34.4.

5.2.4 Secure Element

Hardware Secure Elements (SE) have been extensively analysed and described in Deliverable D35.1, chapter 7.1. Their main property in the context of a trustworthy client platform is the resistance against hardware attacks and against side channel leakage. Therefore, a local attacker, including a potentially malicious user, is not able to extract secret credentials from a Secure Element.

Thus, for a trustworthy client platform the presence of a Secure Element on a mobile device or PC could support a secure storage of user credentials and keys. This would also refer to credentials which have been derived from an external smart card, like an eID card. If they are stored locally in the Secure Element of a device, this storage space would offer the highest possible security level. When an external eID card is used, this smart card automatically provides a secure storage space for the eID credentials and also provides the necessary access control.

Nevertheless, it has to be taken into account that the secure environment of a Secure Element ends on the device itself, where potentially malware can be present. The most vulnerable part on

the device level is the entry of a PIN or password that is used to obtain access to SE functionality. Without a trusted environment this information could be intercepted by malware and could allow an attacker to obtain access to the SE. Even if the attacker would not be able to extract secret data from the SE, he could still use the SE for his own purpose (e.g. triggering an authentication procedure that has not been intended by the user).

Therefore, the highest trust level 5 can only be reached, when the PIN/password entry for SE access occurs in a trustworthy environment, like it is provided by an external class 3 reader or a TEE in conjunction with a secure PIN entry (see chapter 5.4).

It should be noted that when an internal Secure Element is used, such as a SIM card, an embedded Secure Element or a secure microSD card, an appropriate application (applet) on the SE is required to manage the locally stored credentials and to perform the access control. While standards and requirements for derived credentials on Secure Elements and their according applets exist, e.g. for derived PIV cards in the USA, nothing comparable currently exists in Europe. However, it can be anticipated that requirements for derived credentials will be integrated into the European CEN 14890 standard (Secure Signature Creation Devices). It will nevertheless take some time, before European eID cards allow to derive credentials on Secure Elements.

5.2.5 Virtualization

Virtualization can be used to provide an isolated environment for software components on a host. In a virtualized environment, a host system is partitioned into virtual machines using a hypervisor. The hypervisor controls the isolated execution of independent software stacks, such as operating systems.

In general, two approaches exist how to implement hypervisors: Type-1 hypervisors, also denoted by bare-metal hypervisors, are executed directly on top of the host hardware without an intermediate rich operating system (e.g., Linux or Microsoft Windows). A type-2 hypervisor, or hosted hypervisor, is executed on top of a rich operating system. The main problem with operating system virtualization is that operating systems are designed to be executed directly on the host hardware and are thus able to use privileged CPU instructions.

In general there are two approaches to virtualize operating systems: Full virtualization emulates the complete host hardware with the primary goal of running unmodified operating systems designed for the host hardware architecture. This approach requires either to rewrite privileged instructions of the virtualized systems or it requires hardware virtualization extensions in the host CPU which allow the hypervisor to be executed with higher privileges than the virtualized operating systems and to control their execution.

In contrast, para-virtualization requires that the virtualized system is aware of the virtualization layer. In para-virtualized systems privileged instructions are replaced with calls to the hypervisor, denoted by hypercalls, before the virtualized system is deployed. From a security point of view, a Type-1 hypervisor can potentially expose a smaller Trusted Computing Base than a Type-2 hypervisor, since no rich host operating system is part of the Trusted Computing Base.

A well-known example of a type-1 hypervisor from the desktop world is VMware vSphere Hypervisor (ESXi), whereas VMWare Workstation is a type-2 hypervisor.

Virtualization could also be used, to place a trustworthy OS next to a standard OS and to provide trustworthy services, like a trusted UI (see section 5.4).

5.3 Secure Element Access Control Enforcement Systems

One central element of a trustworthy client platform can be a hardware Secure Element (SE) or a software-based Trusted Execution Environment (TEE), see section 5.2. While these components offer a secure environment for credential storage and execution of trustworthy applications (like cryptographic primitives or authentication protocols), they always have to interact with the rich environment which cannot always be regarded as trustworthy. In this context, various threats can arise when the access to the secure component is not strictly controlled. Therefore it is essential to develop access control mechanisms for Secure Elements and TEEs so that only allowed applications can access these environments.

One option to control SE access has been specified by GlobalPlatform and FutureID partners have contributed to the development of these specifications [GPSE]. The Secure Element Access Control (SEAC) specification defines a concept which allows SE application issuers (like card issuers and service providers) to control the access to their SE application from mobile device applications.

This specification relies on a policy-based concept. With this concept a Secure Element application issuer can deploy policies in a Secure Element. These policies are evaluated by the device OS before a device application may access the Secure Element. The policy enforcement on the device is done by an SEAC enforcer, which is part of the Secure Element API stack. As discussed and specified in work package 31 (IFD layer), this stack would preferably be the OpenMobile API for mobile devices. The Secure Element API (like the OpenMobile API) is the gateway for device applications to communicate with Secure Elements via APDUs.

Policies may be stored in an applet (so called Access Rule Applet (ARA)) which is standardized in GlobalPlatform. In order to deploy policies in different Security Domains, the GlobalPlaform specification also allows the usage of several ARA applets in a Secure Element, which can be installed separately in the different Security Domains. These ARA applets are linked in a tree-structure hierarchy in order to provide a common interface to the device. In case of the Universal

IC Card (UICC), policies may also be stored in the UICC's file system. GlobalPlatform specifies the coding and format of the access control files (ARF). An overview of the SE Access Control concept can be seen in Figure 1.

When no Access Rule Applet is available, the normal access rules of the SE file system can be used as fall-back option. For some types of Secure Elements, like the UICC, the use of the ARA is mandatory while for other types (like external eID cards) it is optional.

Therefore, for FutureID it depends on the type of Secure Element that is used for credential storage. In case of an internal SE according to GlobalPlatform specifications the ARA together with the AC Enforcer (e.g. as part of the OpenMobile API) can be used, while for external eID cards the classical file system access rules apply.

For additional security it could be considered to place the AC Enforcer component into a TEE. This would require however a direct (and preferably unique) access of the SE by the TEE. While this option is already taken into account in the standardization work, a practical implementation may require some more time.

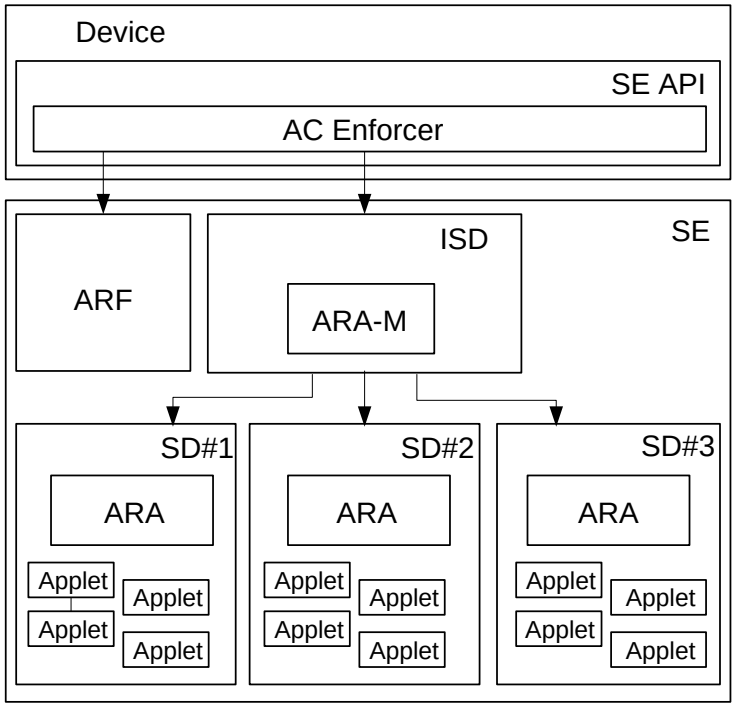

Figure 1: Principle of Secure Element Access Control scheme as defined by GlobalPlatform.

5.4 Secure Peripherals Access

One of the major issues for establishing a closed chain of trust on mobile devices and PCs is a trustworthy User Interface (UI) by which access credentials (PIN/password) are entered and security relevant information is made available to the user. If these UI components cannot be considered to be trustworthy, it is typically easy for any kind of malware to intercept credentials and/or to modify the content of the display. This may lead to leakage of secret credential information or the misuse of transactions for other purposes than originally intended by the user.

In external devices, like a class-3 smart card reader with its own keyboard and (simple) display, the trustworthiness of the UI can usually be guaranteed by providing accordingly tested firmware and by establishing mechanisms to block access to the firmware. In addition, the communication with the device can be protected by secure messaging. Therefore, any malware present on the device attached to the reader has no chance to modify UI components or to intercept communication.

In contrast to special external devices, for mobile devices and PCs special measures need to be taken to provide a trustworthy UI. The approaches which are currently under discussion and under development for mobile devices are based on Trusted Execution Environments (TEE), using the ARM TrustZone or Intel TXT secure environment within the application processor (see section 5.2.3). To establish a trusted UI, the component which controls the trusted user input and output has to run in the TEE as a trusted application and needs verified integrity. Once the trusted UI is activated it is essential, that the trusted application obtains exclusive control of the display and keyboard drivers in order to suppress any interference with other processes and potentially malware.

Therefore, any implementation of a trusted UI based on a TEE requires a deep integration into the device Operating System (OS) to allow privileged access to the peripherals drivers. Since this issue is rather complex and requires a significant modification of the system architecture, it is still unresolved whether the TEE should obtain complete control over the peripherals and in which use cases (like for an incoming call) the rich OS should maintain a certain control. For this reason, no implementation of a trusted UI based on a TEE is currently available. Global Platform has specified an API for a trusted UI based on the TEE, which also includes the definition of UI elements like PIN entry fields or control buttons like “ok”, “validate” or “cancel” [GPTU]. The general TEE access concept for a trusted UI is shown in Figure 2.

The specification also describes the use of a status indicator which clearly indicates to the user that the trusted UI is active. This type of trustworthy indicator is essential to indicate to the user that the input and output is in a trusted mode. Otherwise, malware components could simulate a trusted UI with the same “look & feel” and misguide the user to enter secret credential information. Further details on a trusted UI can also be found in deliverable D34.4.

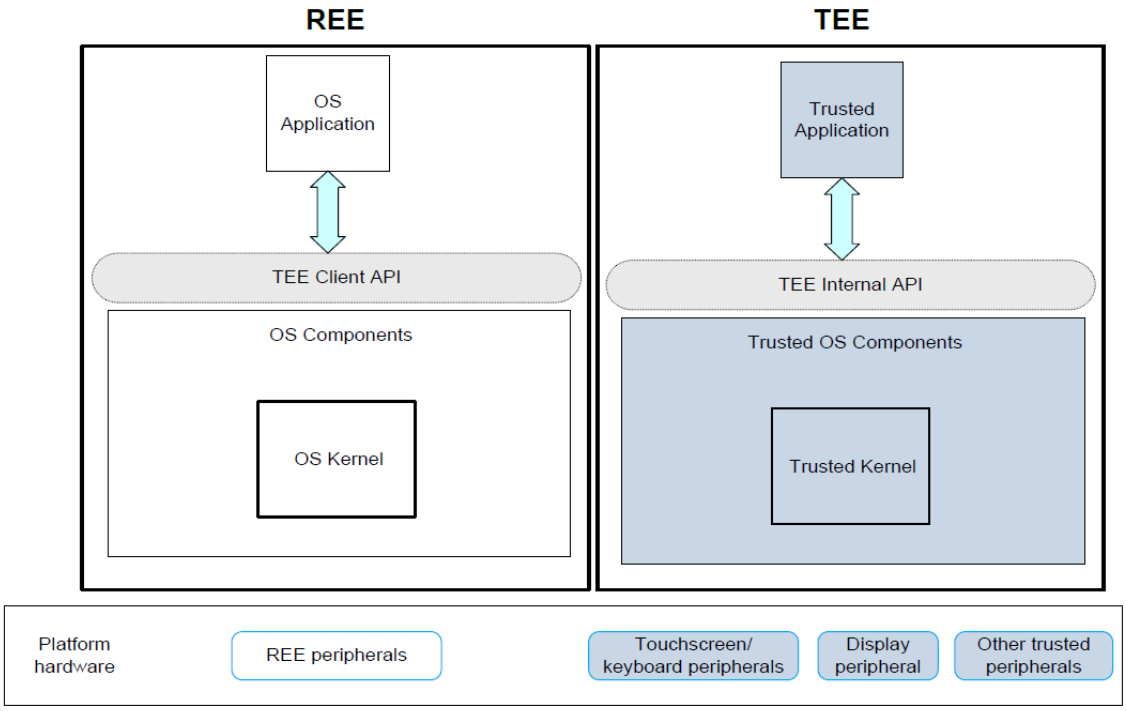

Figure 2: Concept of peripherals access via a TEE to obtain a trusted User Interface. From **[**GPTU**]**.

Since it may require still some time until a trusted UI based on a TEE will be available in the market, another alternative could be envisioned for FutureID, based on virtualization. Virtualization technologies can be used to execute multiple isolated operating systems on one device (see section 5.2.5). For the purpose of a trustworthy platform, virtualization could be used to place next to the application OS (e.g. Android or iOS) a second OS, which main purpose is to provide security relevant services like integrity-protected code execution and secure access to peripherals (e.g. Display/Keypad).

In this context, we recommend to follow the type-1 hypervisor approach to reach a clear separation between the application OS and the secure OS. This scheme is shown in Figure 3.

With a trusted UI and components controlling this UI being executed in a TEE or a secure virtualized environment, the trust levels 3, 4 and 5 (see: D35.1 for a definition) could be supported. Levels 4 and 5 would also require a secure storage space, which could either be provided by the TEE (level 4) or by tamper-proof hardware of a secure element (level 5).

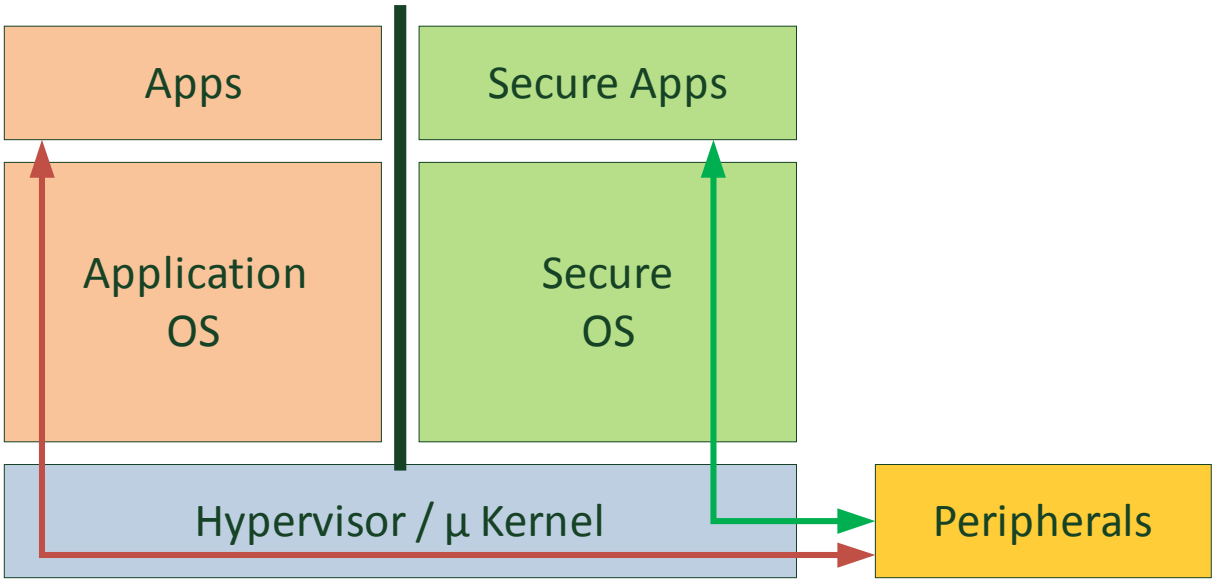

Figure 3: Virtualized access to peripherals using the type-1 hypervisor approach.

6. Protocols

Interactions between two networked entities require the establishment of a certain level of trust between the two partners. The fundamental requirement for this is that every communication partner platform has hardware and software capable to assist in determining and reporting of a platform's state. There are several approaches for this, as already described in Section 5.2.1 and 5.2.3. Once a platform state can be reliably measured, this needs to be communicated over the network. Consequently, this requires a suitable message and protocol framework, for exchange, comparison and negotiation of trust states.

6.1 Protocols for real-time trust measurement

For the primary problem of establishing a secure connection between two network endpoints there are already solutions available. The use of cryptographic keys embedded in certificates, signed by accepted certification authorities, is a common approach for robust identification of a communication partner. Building on that, the TLS protocol [RFC5246] is the widely deployed solution to establish an encrypted communication link protecting against eavesdropping and man-in-the-middle attacks.

For the question of what kind of protocols to run for trust negotiation inside the communication link there is no such a widespread agreement as in the case of TLS. Indeed, the domain of trusted computing technologies is a quite new field as mass-market end-user hardware implementing the trusted computing hardware technologies has only been available for a few years. Consequently, there are several efforts by academics and industry to find a practically usable protocol and message set for trust signalling and negotiation. The practical experiences with trusted hardware drive the development of software components, which at a later time want to communicate about trust and platform integrity to other entities on the network.

The simplest approaches are demonstrated by the projects already referenced in Section 5.2.1 [PIR12] and [TOE10], where (remote) system attestation is essentially the exchange of a TPM signed current platform state report. Thus, a simple data structure coming from the Trusted Platform Module of the system is transmitted via a simple custom protocol. Also, the inspection and trust assessment is custom and quite simple.

The industry effort to specify protocols in this area is spearheaded by the Trusted Computing Group (TCG) and their Trusted Network Connect (TNC) suite[3] of protocols. This suite of documents promises to provide open standards on communicating about network endpoint integrity metadata. The available set of documents is quite comprehensive as they are intended

[3] http://www.trustedcomputinggroup.org/developers/trusted_network_connect

for large enterprise-wide deployments. They specify all components and interactions in a trusted connection architecture, from the integrity measurement collector to the TNC client, sending the measurement data to the TNC server, who is asking a measurement verifier, whose result may then influence a network access authority to grant access to a client. The years long “in-development” nature of the industry's TNC effort reflects the complexity of the problem while simultaneously the research continues on how to accurately describe a platform's state.

From an academic research project's perspective the industry developed components are not an ideal solution. Rather, open source solutions, which can be customized to the project's need, are preferred. There are two notable open-source efforts in this area: OpenAttestation[4] and Trust@HsH[5].

While both of them are work-in-progress and do not provide full deployment-ready solutions from bare-metal TPM measurements to high-level database management of integrity measurement values, they are a good start for further investigation. They may provide readily implemented protocol components and data structures which can be customized to FutureID's needs.

6.2 Protocols for trust establishment in a heterogeneous environment

Following the discussion in 5.2.1, fully measured platforms in the sense of trusted computing cannot be expected. Nevertheless, means are required to communicate the current status of the client platform to assess the security risk and to determine the capabilities of a client platform. The result of such an assessment can only be trusted under the assumption that an adversary has not compromised the assessing components or the transport layer.

Examples of the information to be used for an assessment are listed in Table 1:

Hardware	• Is a TPM available and enabled? • Is a smartcard reader available and of which class? • Are other secure elements available?
Operating System	• Type and version of the operating system • Type and version of a virus scanner

[4] https://github.com/OpenAttestation/OpenAttestation

[5] http://trust.f4.hs-hannover.de/

	• Type and version of a firewall • Content of PCR if secure boot is available
***FutureID* Software**	• Version of the *FutureID* client • Integrity of the components • Installed plug-ins and trustworthiness of their origin

Table 1: Possible Integrity Measurements

In order to communicate the integrity measurements, which are the basis for the platform assessment, a subset of the TNC protocol suite should be used. The relevant part of the TNC architecture is shown in Figure 4:

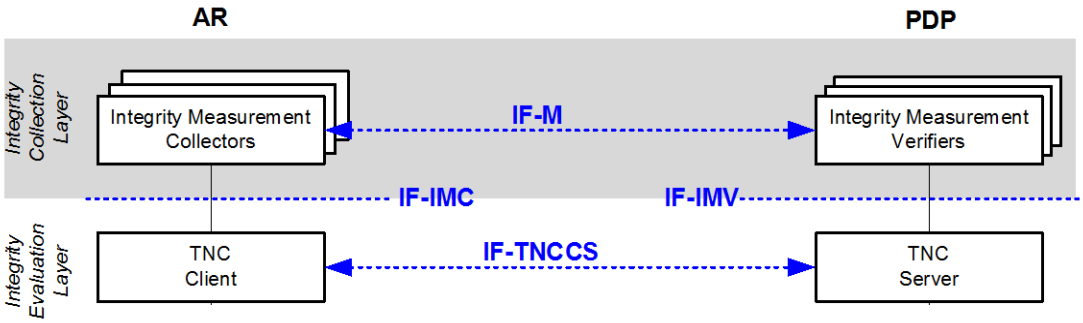

Figure 4: Relevant Part of the TNC Architecture
(AR: Access Requestor, PDP: Policy Decision Point)

The Integrity Measurement Collectors (IMC) are modules that provide integrity measurement messages for the TNC Client, for example, the type and version of the operating system. As shown in Figure 4, a single TNC client can operate multiple IMC and IMC can be added or removed dynamically. Upon request by the TNC Server, the TNC Client collects the integrity measurement messages from all IMC and compiles them into a batch, which is transferred to the TNC Server. The TNC Server propagates the messages to the Integrity Measurement Verifiers (IMV). If an IMV has already enough information, it sends a recommendation to the TNC Server. Additionally, an IMV can send messages back to the client, for example, if an IMC requires additional data from an IMV. This will result in another message batch from the client to the server. Eventually, all IMV provide their recommendations to the TNC Server, which in turn will calculate the final decision based on an access policy.

For FutureID this means that the IMC should run on the client platform while the IMV would be located at the server backend. Thus, the server backend would actually determine the achieved trust level, depending on the client IMC results.

6.3 Protocol Interfaces

For the interfaces [IF-IMC] and [IF-IMV] Java interfaces are available, for the interface [IF-TNCCS] an XML schema is available. The interface [IF-M] defines a set of standard attributes to be used by IMC and IMV to guarantee interoperability.

7. System Architectures

7.1 Introduction

This section describes potential system architectures that can support the requirements for a trustworthy client platform. In deliverable D35.1, five different trust levels have been defined, depending on the type of used credentials and the trust-related properties of the environment in which the authentication takes place. When considering system architectures for stationary devices and mobile devices these trust levels should be taken into account.

Thus, it needs to be analysed which specific version of the system architecture can reach which trust level. In the optimum case the architecture could be built in a modular way, thus reaching a higher level when an additional module is included. The architectures in the following sections have been developed using a modular approach. Nevertheless, since there is a large variety of possible hardware and software components that may be present or not, several variations for typical configurations will be described.

To simplify the visual representation of the architectures for the different trust levels, a colour coding is used. This coding identifies the trust level that can be achieved when a certain component is present. In order to reach a certain level, all components of the same colour coding have to be available, unless they are described as independent alternatives.

The following coding is used:

Other components printed in dark blue are standard middleware components that are present independent of the actual trust level.

7.2 System Architecture PC

In this section we describe the system architecture for desktop PC-based platforms.

7.2.1 General Architecture

Figure 5 shows an overview of the system architecture of the trustworthy client platform. On top is the Interface Device Service (IFD), which serves as abstraction layer to the remaining parts of the FutureID client.

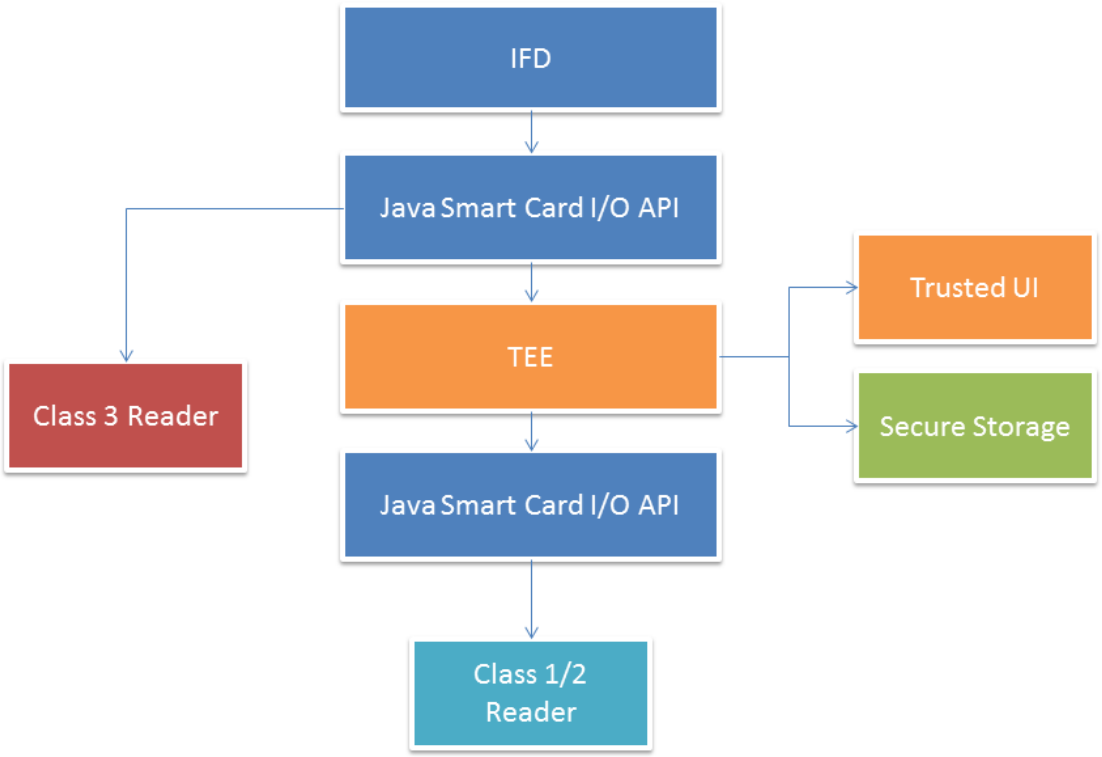

Figure 5: General System Architecture PC

As specified in D31.1 *Requirements Report for Interface Device Service*, the FutureID client will be developed in Java and therefore use the Java Smart Card IO API to access ISO 7816 and ISO 14443 based smart cards using Application Protocol Data Units (APDUs). The Java Smart Card IO API in turn uses system specific interfaces like PC/SC in order to transmit commands to smart card readers.

7.2.2 Platform-specific Variants

The following paragraphs discuss the possible variants of the system architecture, depending on the capabilities of the underlying platform.

7.2.2.1 Class 3 Reader

Probably the most common platform configuration is a PC with connected class 3 smart card reader. This can be connected to the IFD directly via the Java Smart Card IO API, as shown in Figure 6.

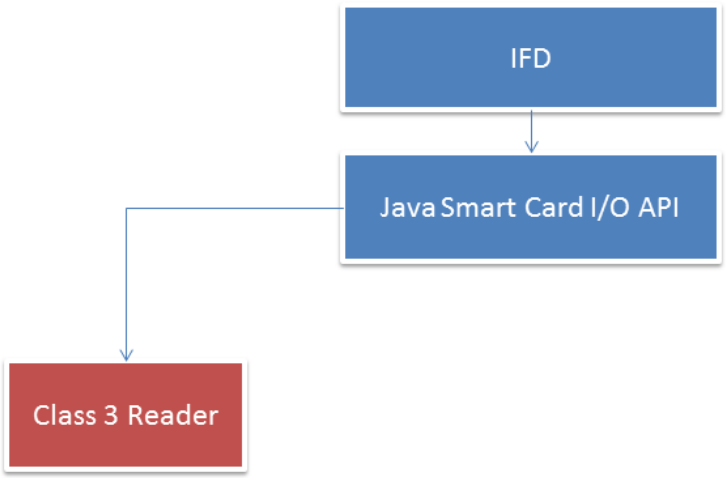

Figure 6: System Architecture PC with Class 3 Reader

In this variant the credentials can be protected by cryptographic keys, which are stored and processed inside a hardware-based secure element. Furthermore, class 3 readers are required to have a dedicated display and PIN pad, which can be used as trusted user interface. Therefore this variant fulfils the requirements for Trustworthiness Level 5.

7.2.2.2 Trusted Execution Environment

Trusted execution environments (TEEs) are security extensions, which allow the processor to execute certain code in a special mode and thus isolated from the remaining software, including the operating system. Their advantage is that security critical data can be processed inside the same system, without requiring extra hardware like smart cards or OTP generators.

Naturally this is very useful for mobile devices, where external hardware is not only expensive but also unhandy. However, recent and future instruction set extensions introduce security features also to commodity desktop CPUs. Details on how to implement TEEs on desktop system will be described later in section 7.2.3.

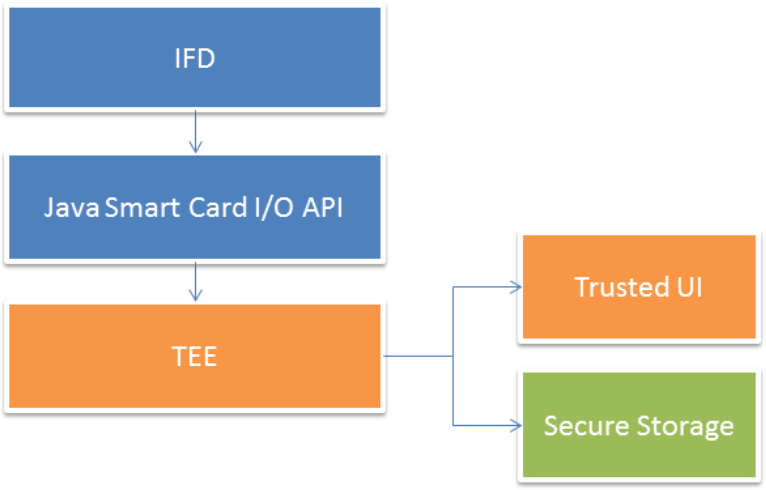

Figure 7: System Architecture PC with TEE

Figure 7 shows a variant of the system architecture, which uses a TEE to process credential information. The shown configuration can achieve a Trustworthiness Level of 3 or 4, depending on the existence of a trusted user interface, which is not available in all TEE implementations. Trusted Execution Environments usually have only limited internal storage; therefore application data is often stored encrypted and integrity-protected in the normal device memory.

7.2.2.3 Trusted Execution Environment and Secure Element

Trusted Execution Environments alone cannot achieve the highest level of trustworthiness, since they are designed to withstand only software attacks, not hardware attacks. This requires storing and processing credentials inside secure elements, which are dedicated tamper-resistant chips like the ones used in smart cards.

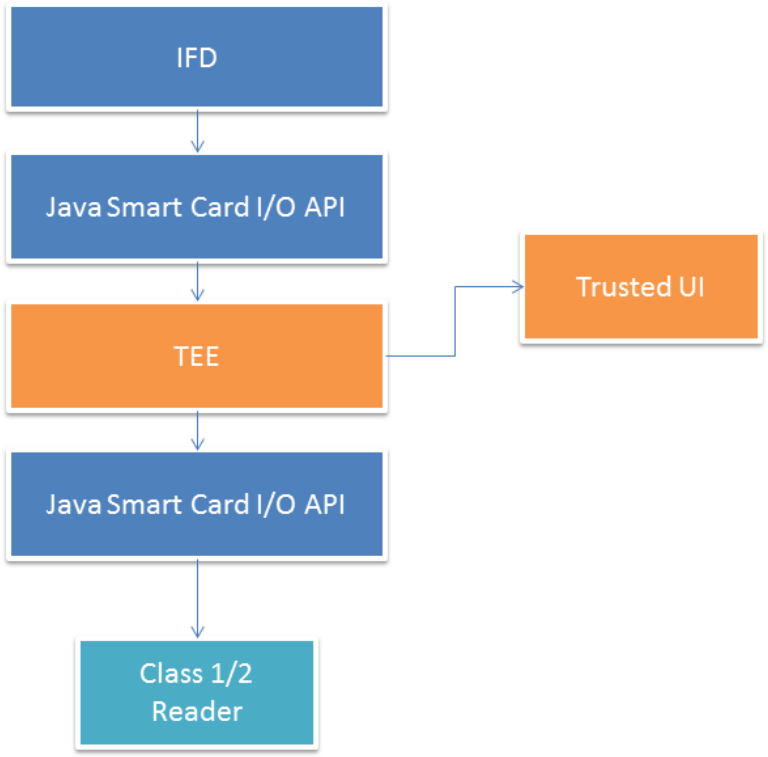

Figure 8: System Architecture PC with TEE and SE

On desktop PC platforms secure elements can be connected externally using a card reader. Since class 1 and 2 readers do not have a trusted display, they alone can only achieve Trustworthiness Level 4. For Level 5 they should be combined with a TEE, which provides a trusted user interface.

7.2.3 Trusted Execution Environments on PC-based Platforms

Trusted Execution Environments (TEEs) can be implemented in software with virtualization, as described in section 5.2.5.

Newer generations of desktop system, however, bring advanced hardware support for virtualization, including security extensions for the late launch of hypervisors. In current Intel processors this is implemented as Trusted Execution Technology (TXT), which allows together with a TPM v1.2 to launch a trusted startup code, independent from the operating system. Similar functionality exists in AMD processors with the SKINIT instruction.

Scientific publications [CUN07],[CUN08],[BRA12] have shown, that this functionality can be used to implement a hardware-based Trusted Execution Environment on a platform with only commodity PC components. As the main operating system is suspended during the operation, it cannot intercept inputs and outputs. Therefore this also supports a trusted user interface. Secure storage is provided by the TPM sealing function, which is used to unseal data only to the Trusted Execution Environment.

Recently, Intel introduced the Software Guards Extensions (SGX) [INT13],[KEE13], a new instruction set extension that allows to create TEEs (so-called enclaves) without requiring a TPM. There can be different independent enclaves, which are isolated from all other code including the operating system. Therefore enclaves have to trust only their own code and the basic protection mechanisms of the processor and chipset. The access control for enclave data is integrated directly into the processor, so that data of an enclave can only be accessed by its own code. Enclaves organize data and code in pages, which can also be evicted to a page file. Enclave data is protected with encryption, integrity checks and state counters whenever it is stored outside the processor itself. Furthermore, enclave code can only be called at valid entry points and upon an interrupt the processor ensures that its state is clean before it transfers execution to the operating system's interrupt handler and back again. SGX is supposed to be integrated into future Intel processors.

7.2.4 Interfaces

The proposed architecture described in Figure 8 can be partly implemented using existing interface definitions. For convenience, they are listed in this subsection.

Java Smart Card IO API:

The Java SCIO API defines a communication interface with ISO7816 and ISO14443 smart cards. It allows Java applications to interact with smart cards via exchange of Application Protocol Data Units (APDUs) and provides basic management functionality for smart card readers.

Trusted Execution Environment:

There is no standardized API for Trusted Execution Environments on desktop systems yet, since the hardware extensions are rather new and not widely used in production systems so far. Furthermore, the standardization process is expected to be more complicated, since desktop PC platforms involve many different hardware and software vendors in contrast to mobile systems, where hardware and operating system are usually provided together by a single manufacturer.

However, it is possible that the GlobalPlatform TEE specification (see section 7.3.3) will be adapted to desktop systems in the future.

7.3 System Architecture Mobile Device

7.3.1 General Architecture

The general architecture proposal for mobile device platforms is shown in Figure 8. It shows the components that are required to reach a specific trust level as defined in deliverable D35.1. The Interface Device Layer (IFD) and the OpenMobile API are standard middleware layers as specified in work package 31. They are required to establish a communication between the FutureID client and secure environments like a Secure Element, a TEE or an external card reader. In the envisioned architecture, the service layers of the OpenMobile API would allow a transparent communication independent of the type of secure environment.

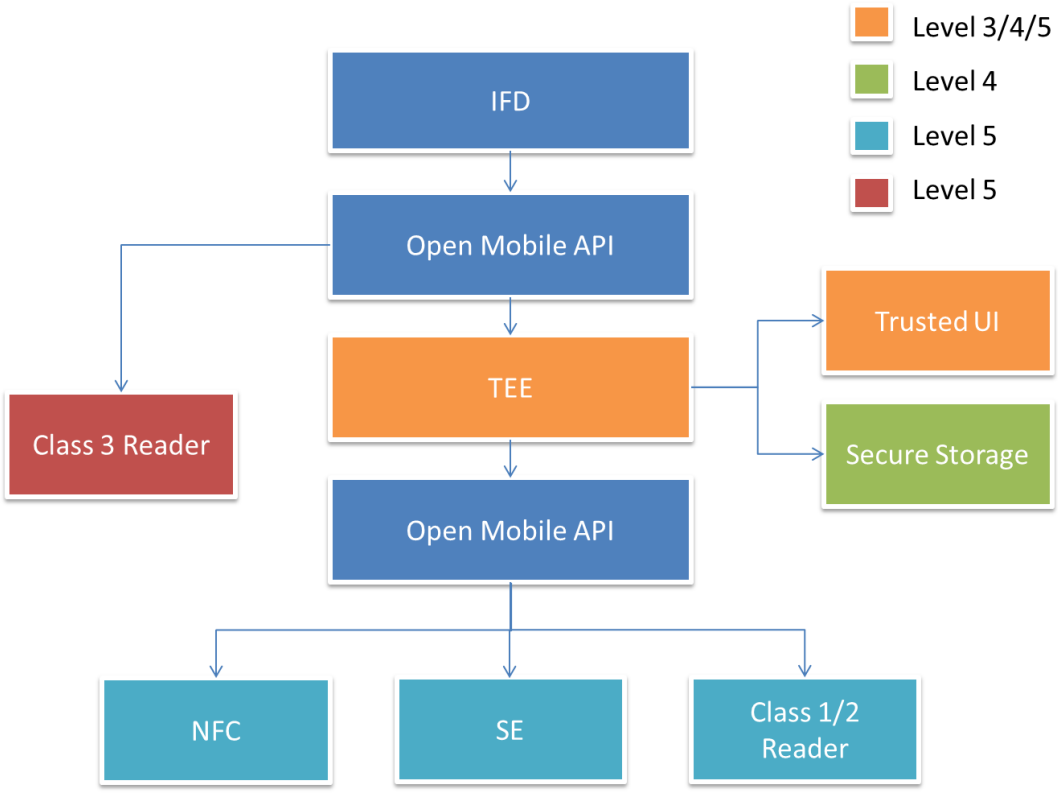

Figure 9: Overview of general platform architecture for mobile devices for the various trustworthiness levels.

As defined in D35.1, the trust levels 1 and 2 do not require any secure environment. For these levels it is assumed that all credentials are handled in a potentially insecure environment. For the higher levels, the following components can be used to fulfil the requirements:

Level 3:

This level already requires a trusted UI for secure input of user credentials (password/PIN) and a trusted output for documents to be signed. This could be achieved by adding the orange components in Figure 8. The TEE would control the user input and output in this configuration and would therefore need to get exclusive access to the respective peripherals (keyboard, display). Since a trusted UI is also desired for the levels 4 and 5, the orange components would also support these levels. Further details regarding the trusted UI and the TEE access to it can be found in deliverable D34.4.

Level 4:

When the green module (secure storage) is available in addition to the orange modules, a level 4 can be reached. In this configuration the TEE would provide the secure storage capability, either by providing exclusive access to a separate memory (preferably separate memory hardware) or by encrypting the information before it is stored in the standard memory.

Level 5:

This level requires a tamper-proof hardware element for secure storage of credentials. This can be realised by a secure element like a UICC card (e.g. SIM) or an external eID card either contactless via NFC or as contact card via an external class 1 or 2 reader, illustrated by the turquoise boxes. The communication to these elements would occur via the transport layers of the OpenMobile API (for this reason the OpenMobile API box appears twice). In order to reach level 5, the orange components (TEE + trusted UI) would additionally be required.

Another alternative is to use an external class 3 reader that provides its own keyboard and display. This reader could be connected to the mobile device via USB or Bluetooth. In this case, neither a TEE nor a secure element is required on the mobile device to reach level 5.

The two variations for reaching level 5 are shown in Figure 9.

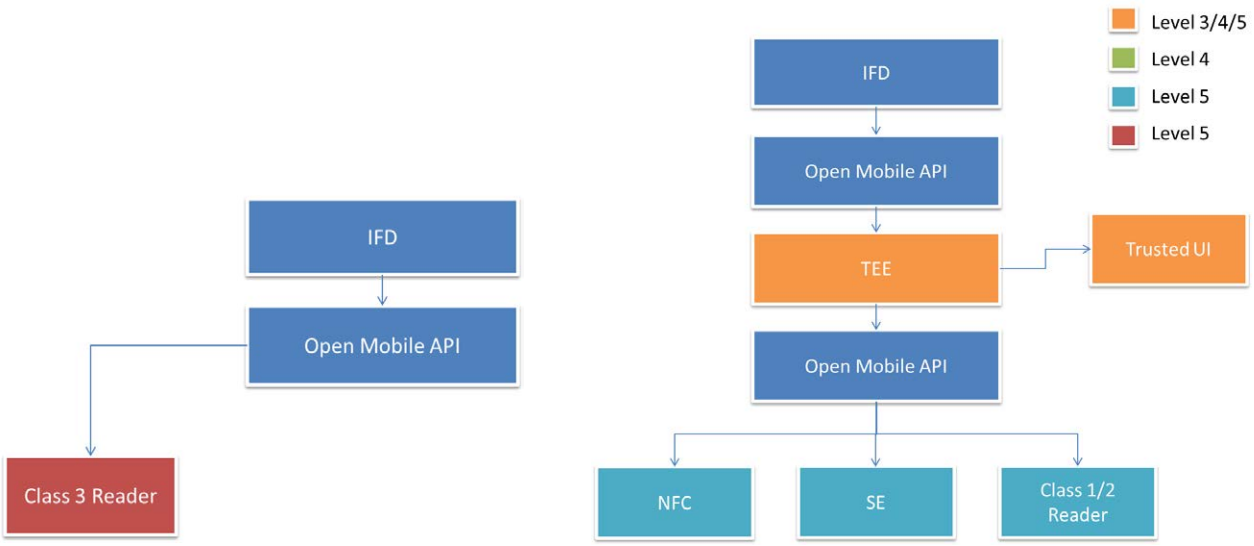

Figure 10: Architecture options to reach a trust level 5, either by using an external class 3 reader (left) or by using a Secure Element with a trusted UI provided by the mobile device (right).

7.3.2 Platform-specific Parts

Since mobile devices have an OS architecture that significantly differs from PC-like platforms, basically all modules described above are platform specific. This refers especially to the middleware and the TEE. Within the mobile device platforms, the architecture also differs significantly between Android platforms and others like iOS. Since FutureID will focus only on Andriod, the architecture described above assumes an Android platform.

7.3.3 Interfaces

The proposed architecture as outlined in Figure 8 and Figure 9 can be built on various existing standard interface definitions. For the sake of convenience they are listed again in this subsection.

Open Mobile API:

The Open Mobile API comprises interface definitions for service layers as well as transport layers. In the proposed architecture it will be used to address secure environments like the TEE, Secure Elements or an external class 3 reader. The API definition can be found in [OMA13].

Trusted Execution Environment:

The interfaces for the Trusted Execution Environment on mobile devices are defined by various Global Platform specifications. For the reference architecture, the following specifications are of interest:

- TEE System Architecture [SPE09].
- Communication of rich OS application with TEE: Client API [SPE07].
- Specification for Trusted Applications running inside the TEE: Internal API Specification [SPE10].
- Communication of TEE with Secure Elements: Secure Element API Specification [SPE24].
- Trusted User Interface: Trusted User Interface API Specification [SPE20].

8. Summary/Conclusions

The investigations within this task have shown that a rich set of elements is available to build up a client platform architecture that can achieve a considerable level of trustworthiness. Elements and components as well as protocols can be combined to establish a continuous “chain of trust” which will result in a “chain of assurances”. Interface definitions are available in most cases so that the interaction of the components can be established in an actual implementation. This is an important prerequisite for obtaining, maintaining and attesting a certain trust level and to base FutureID authentication decisions on this level. By defining reference architectures for each trust level in the stationary and mobile case it was shown that all five trust levels can be realized with an according combination of components and elements.

Nevertheless, the work has also shown that there are still challenges and limitations that remain. The differences in platform architecture as well as interface definition between stationary and mobile platforms require platform specific adaptions in the actual implementation. Even when all discussed secure environments are present, it cannot be expected that a platform with fully measured integrity will be available. Therefore, assumptions still have to be made that the relevant components and secure environment work correctly and are not compromised by some kind of malware. The complexity for providing a fully measured platform would simply be too high. This is also the reason why it can be expected that the TPM approach will in practice only be limited to certain well-defined use cases with moderate complexity (e.g. secure user login).

Another challenge comes from the availability of actual implementations. Since it is beyond the scope of this project to fully implement all secure environments for a trusted platform, the project also has to rely on third-party implementations, e.g. for Trusted Execution Environments. While in the mobile case all interfaces for interactions of the TEE for example with a Secure Element or a trusted User Interface are defined, there are still no implementations available. In case of the TEE for stationary devices, even the interface definition is not fully resolved yet. Therefore it seems reasonable to consider virtualized domains as a kind of bridging technology until all components are available. In this case the trusted environment would be realized by pure software means (virtualization) but would have no hardware anchor like in the case of a full TEE.

A careful analysis of the possibilities to measure platform integrity and to communicate this result with a trustworthy protocol has shown that for the FutureID use case it seems reasonable to perform the actual measurement on the client platform but to derive the resulting trust level on the server side within the FutureID backend. Depending on the result, the backend can then decide upon the consequences for a specific authentication request.

9. References

[GPSE] GlobalPlatform Device Technology, "*Secure Element Access Control"*, Version 1.0, May 2012, Document Reference: GPD_SPE_013

[GPTU] GlobalPlatform Device Technology , "*Trusted User Interface API Specification V1.0"*, June 2013, Document Reference: GPD_SPE020

[CUN07] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter and A. Seshadri, "Minimal TCB Code Execution (Extended Abstract)," in Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California, 2007.

[CUN08] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, Isozaki and Hiroshi, "Flicker: An Execution Infrastructure for TCB Minimization," in Proceedings of the ACM European Conference on Computer Systems (EuroSys'08), Glasgow, Scotland, 2008.

[BRA12] F. F. Brasser, S. Bugiel, A. Filyanov, A.-R. Sadeghi and S. Schulz, "Softer Smartcards: Usable Cryptographic Tokens with Secure Execution," in Financial Cryptography and Data Security (FC), Bonaire, Netherlands, 2012.

[INT13] Intel, "Software Guard Extensions Programming Reference," September 2013. [Online]. Available: http://software.intel.com/sites/default/files/329298-001.pdf.

[KEE13] F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue and U. Savagaonkar, "Innovative Instructions and Software Model for Isolated Execution," in Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, 2013.

[D31.2] *"Interface and Module Specification and Documentation"* (WP31 - Interface Device Service), FutureID, 2013

[PIR12] Martin Pirker, Johannes Winter, Ronald Tögl, "*Lightweight Distributed Heterogeneous Attested Android Clouds"*, Proceedings of the 5th International Conference on Trust & Trustworthy Computing (TRUST)

[TOE10] Ronald Tögl, Martin Pirker, and Michael Gissing, "*acTvSM: A Dynamic Virtualization Platform for Enforcement of Application Integrity",* Proceedings of INTRUST 2010, Springer-Verlag Berlin Heidelberg 2011

[OMA13] Simalliance, "Open Mobile API Specification V2.04", released 29.07.2013

[SPE09] GlobalPlatform Device Technology, "*TEE System Architecture*", Version 1.0, December 2011, Document Reference: GPD_SPE_009

[SPE07] GlobalPlatform Device Technology, "*TEE Client API*", Version 1.0, July 2010, Document Reference: GPD_SPE_007

[SPE10] GlobalPlatform Device Technology, "*TEE Internal API Specification*", Version 1.0, December 2011, Document Reference: GPD_SPE_010

[SPE24] GlobalPlatform Device Technology, "*TEE Secure Element API Specification*", Version 1.0, August 2013, Document Reference: GPD_SPE_024

[SPE20] GlobalPlatform Device Technology, "*Trusted User Interface API Specification*", Version 1.0, June 2013, Document Reference: GPD_SPE_020

[RFC5246] The Transport Layer Security (TLS) Protocol, Version 1.2, August 2008, http://tools.ietf.org/html/rfc5246

[IF-IMC] TCG Trusted Network Connect TNC IF-IMC, Version 1.3 Revision 18, February 2013, http://www.trustedcomputinggroup.org/files/static_page_files/1D8C8F15-1A4B-B294-D0CD725393CC0A93/TNC_IFIMC_v1_3_r18.pdf

[IF-IMV] TCG Trusted Network Connect TNC IF-IMV, Version 1.3 Revision 13, February 2013, http://www.trustedcomputinggroup.org/files/static_page_files/1D8CE1B6-1A4B-B294-D087F581D114F2DA/TNC_IFIMV_v1_3_r13.pdf

[IF-TNCCS] TCG Trusted Network Connect TNC IF-TNCCS, Version 1.2 Revision 6.00, May 2009, http://www.trustedcomputinggroup.org/files/resource_files/51ED9FF7-1D09-3519-AD3E3B2EBEECEB3F/TNC_IF-TNCCS_v1_2_r6.pdf

[IF-M] TCG Trusted Network Connect TNC IF-M: TLV Binding, Version 1.0 Revision 37, March 2010, http://www.trustedcomputinggroup.org/files/resource_files/495862FF-1D09-3519-AD8977DC98C1167C/TNC_IFM_TLVBinding_v1_0_r37a.pdf