Implementation of the Security-relevant Modules for Selected Platform

WP 35 Trustworthy Client Platform

D 35.3	
Date	18/08/2015
Status	Final
Version	1.0

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

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

1. Abstract

This deliverable describes the results of the implementation of security-relevant modules on mobile Android devices. The purpose of the implemented modules is to enhance device platform security and thus to increase the overall security of the FutureID solution. One of the major security challenges in a client application like FutureID is the secure and trustworthy handling of identity credentials. The FutureID client acts as the main communication module with hardware-based tokens on the client device. In principle, this modular architecture can also be used for pure software-based tokens, which are on the other hand even more subject to potential security threats.

After discussing the general challenges of platform security for identity applications in chapter 5, chapter 6 describes the implementation work of the Android Security Modules (ASM) framework which allows to enforce fine-grained access control policies, e.g. on file-based identity credentials or on the OpenMobile API as mobile Interface Device Layer (IFD). Chapter 7 documents the implementation of signature modules within an ARM-Trustzone based Trusted Execution Environment and shows how security can be increased by operating central cryptographic components from within a trustworthy environment. How the access to Secure Elements can be simplified by providing a unified access channel via the OpenMobile API is documented in chapter 8. As an alternative to hardware based security, chapter 9 documents the implementation effort on software-security protected environments.

Overall, the implementations show that by integrating some key components into the device platform the security of the FutureID client can be enhanced significantly. For future productive roll-outs it is recommended to integrate one or more of these modules, depending on the specific use case and the targeted device platforms.

2. Document Information

2.1 Contributors

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

2.2 History

Version	Date	Author	Changes
0.1	06.07.2015	F.-M. Kamm	Initial version
0.2	08.07.2015	F.-M. Kamm	Chapter 5/8 added
0.3	15.07.2015	F.-M. Kamm	Chapter 9 added
0.4	16.07.2015	Marc Obrador	Chapter 8 updated
0.5	17.07.2015	Christof Rath	Chapter 7 added
0.6	18.07.2015	David Derler	Rework 7.2, 7.3
0.7	04.08.2015	F.-M. Kamm	Chapter 10 added
0.8	06.08.2015	J. Rios	Chapter 6 added
0.9	07.08.2015	F.-M. Kamm	Internal review version
1.0	18.08.2015	Christof Rath	Review comments

2.3 Glossary of Terms

access control

The process of restricting access to resources to privileged entities. Access control policies determines who can view access what resources, under which conditions, what they can do with it, and the type of device which may be used.

address

An address is the identifier for a specific termination point and is used for routing to this termination point.

application (A)

An application is a software component run by a service provider that implements an actual service offering. Applications are typically accessible via HTTP through a web server. A service provider can run one to several applications; they typically run in some application server, container, or framework, as for examples servlets in a J2EE servlet container.

asset

Anything that has value to the organization, its business, its operations and its continuity.

assurance

A measure of certainty that a statement or a claim is true.

attribute

An attribute is a physical or abstract named property belonging to an entity. An attribute has a key and a value.

authentication

Authentication is the process of corroborating a claimed set of attributes or facts with a specified, or understood, level of confidence.

availability

Availability can be described as the property of being accessible and useable upon demand by an authorized entity.

In the context of service level agreements, availability generally refers to the degree to which a system may suffer degradation or interruption in its service to the customer as a consequence of failures of one or more of its parts.

binding

An explicit established association, bonding, or tie.

certificate

A certificate is an affidavit whereby a certification body attests to the truth of certain stated facts. In identity management, the term is often used to refer to a public-key digital certificate. X.509 Digital Certificates are a prominent example thereof.

claim

A statement made by one entity about itself or another entity that a relying party considers to be in doubt until it passes Claims Approval.

consent

The general permission granted by an individual to a requesting entity to use the individual's personal information in some agreed manner. Consent can be expressed, implied, or provided through an authorized representative.

context

A context can be described as a particular setting of an entity's environment. It can, for example, be a sphere of activity, a geographic region, a communication platform, a logical, or physical (security) domain.

In identity management the term 'context' is typically used to refer to a (1) communicational context or (2) a (security) domain:

(1) A communicational context is defined by roles and behavioral schemes assumed by the participants in the communication. In organizational systems communicational contexts in most cases map with sectors defined by the organization;

(2) A security domain is an environment that is defined by security models and a security architecture, incorporating a set of resources and set of system entities that are authorized to access some or all of the resources. The traits defining a given security domain typically evolve over time.

credential

A credential is a verifiable set of attributes that describe an entity. The major properties of a credential are issuer and format (such as X.509 token or SAML bearer assertion). User credentials are credentials that are typically long-lived and controlled by the user; Session credentials are typically short-lived and are created and consumed as part of the overall authentication process.

data

Data is a carrier of information. This information is encoded in a specific physical representation, usually a sequence of symbols that have meaning and can be processed or produced by a computer.

eID services

Services for entity authentication and/or signing data.

electronic identity

An electronic identity is a collection of identity attributes in an electronic form (eq. digital identity)

electronic signature

Data in electronic form which is attached to or logically associated to other electronic data and serves as a method of authentication.

From a legal perspective, an electronic signature is not necessarily considered equivalent to a handwritten signature. When it meets a number of conditions, it can be put on par with a handwritten one.

encryption

Encryption is a means of transforming data from a readable form (known as plain text or clear text) to one that is unintelligible (referred to as cipher text).

federated identity

A federated identity is a partial identity which is the result of federation, and which usually implies that the entity to which this identity corresponds is recognized by several service providers or applications that are part of the federation.

See also federation. federation; circle of trust.

FutureID client (FC)

The FutureID Client (FC) is a software component installed on the user platform that is a rich client supporting user-centric authentication. While users without FC can participate in the FutureID infrastructure, only the use of a FC achieves full functionality and privacy. The FC can take full control of the authentication flow, provides the user interface for user decisions on disclosure, trust and policy, supports digital signature, and interfaces to various tokens (such as smart cards).

hardware token

A hardware token is a physical token that contains credentials, i.e. information attesting to the integrity of identity attributes. Proving physical possession of the token may involve one of several techniques, including the supply of a PIN or biometric.

identifier

An identifier can be defined as a non-empty set of attributes of an entity which uniquely identifies the entity within one or more specific contexts or sectors.

identity

The identity of an entity is the dynamic collection of all of the entity's attributes.

An entity has only one identity. As such, the identity of an entity this is more a fluid and evolving ("philosophical") concept, rather than a practical one.

An entity has only one identity, but it is neither possible nor desirable for an information system to gather all the attributes of a specific entity. Information systems focus on a specific subset of relevant attributes, commonly referred to as 'partial identities'.

identity management (IdM)

The definition, designation and administration of identity attributes and the management of access to and the usage of applications, services and resources.

integrity

Quality which implies that the items of interest (facts, data, attributes etc.) have not been subject to manipulation by unauthorized entities.

metadata

Metadata is data about data. Metadata may describe how, when and by whom a particular set of data was collected, and how the data is formatted.

name

A name is the identifier of an entity (e.g., subscriber, network element) that may be resolved/translated into an address.

object

An object is a non-acting entity that contains or receives data or information, to which access is controlled. An object is for example a file, a program, a document, an area of main storage (such as a database).

party

A natural person or a legal entity.

performance

Performance is the speed measured and perceived by individual users of the system (latency), or the speed perceived by the total infrastructure, from the point of view of the server (throughput, number of accepted transactions per second).

permission

A permission describes a set of defined actions, which correspond to a subset or the entirety of all possible uses of the controlled or protected resources, which an entity is entitled to perform (typically 'read', 'modify', 'create' and/or 'delete' resources).

personal identification number (PIN)

A Personal Identification Number is a factor used for entity authentication, whereby an entity enters a usually four digit number, which is only known to this entity.

In certain instances, the term is also used to refer to (person) identifiers.

policy

A policy is one or more definite goals, courses or methods of action to guide and determine present and future decisions.

Policies are implemented or executed within a particular context (such as policies defined within an organization or business unit) or across contexts (e.g., in the case of an identity federation).

Common examples of such policies are security policies, privacy policies, access control policies, registration policies etc.

privacy policy

A privacy policy is a policy which is based mainly data protection and privacy considerations.

It may specify, for example, what personal data is being collected, for what purpose the collected personal data is being used, how long the data is being retained, how a person may access and correct data relating to him/her, how and whether a person can opt-put; and what security measures are being taken by the entities that process and/or control the data.

privilege

Right to perform a defined action or to use a defined service/resource. Privileges normally relate either to the ability to access data (e.g., update a payroll record) or the ability to use some feature (e.g., surf the Internet).

protocol

A protocol may be described as a set of rules (i.e., formats and procedures) to implement and control some type of association (e.g., communication) between systems (e.g. an internet protocol).

In more general terms, a protocol can be qualified a series of ordered steps involving computing and communication that are performed by two or more system entities to achieve a joint objective.

public key infrastructure (PKI)

System of certification authorities (CAs) (and, optionally, registration authorities and other supporting servers and agents) that perform some set of certificate management, archive management, key management, and token management functions for a community of users in an application of asymmetric cryptography.

registration

Registration is the process of collecting and corroborating a specific set of attributes of an entity, which typically relate to the partial identity (e.g., the age), a characteristic or a mandate of that entity, with sufficient certainty, before putting at the disposal means by which the entity can be authenticated, or the characteristic or mandate can be verified.

resource

The term resource is a generic term which is generally used to refer to entities with some value, systems and services.

Computing systems and services are for example disk space on a file server, electronic mailboxes, the system software, applications, services, data repositories, data objects, ...

Non-computing systems and services may be anything from natural persons (e.g., employees) to physical assets (e.g., desk, telephone, mobile phone, laptop).

risk

Likelihood that an unwanted incident will occur and the impact that could result from the incident.

security policy

A security policy is a set of rules and practices that specify (or determine) how a system or organization should protect (or protects) data exchange, data storage, sensitive and/or critical system resources, and the use and provision of security services and facilities.

service

A digital entity comprising software, hardware and / or communications channels that interacts with subjects.

service provider (SP)

Entity that provides services to other entities.

token

A token is any hardware or software component that contains credentials related to attributes. They may take any form, ranging from a digital data set to smart cards or mobile phones. They can be used for both data/entity authentication and authorization purposes.

transport

The functional process of transferring information between different locations.

trust

A measure of reliance on the character, ability, strength, or truth of someone or something.

trust infrastructure

The technical infrastructure used by system entities in order to trust each other. Trust infrastructures may be built on Public Key Infrastructure, Kerberos, etc.

trust service (TS)

A FutureID Trust Service manages trust related knowledge in the FutureID infrastructure. In particular, it manages which issuers of user credentials (e.g. X.509 certificates) and session credentials (e.g. SAML Assertions, or WS-* claims) are trusted and at which trust level.

validation

Confirming that information given is correct, often by seeking independent corroboration or assurance.

verification

The process or an instance of establishing the truth or validity of something.

2.4 Acronyms

A
application

PIN
personal identification number

FC
FutureID client

SP
service provider

TS
: trust service

IdM
: identity management

PKI
: public key infrastructure

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 smartphones. 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. General Challenges

5.1 Security of Identity Credentials

One of the major security challenges in any identity-related client application like FutureID is the secure and trustworthy handling of identity credentials. The FutureID client acts as the main user interface during the authentication process and allows for user choices regarding the type of authentication credentials and possibly optional attributes that can be sent to the service provider. Since the client also contains the Interface Device Layer (IFD, WP 31) and the eID services (WP 32) as well as the signature add-ons (WP 33) it acts as the main communication module with hardware-based tokens on the client device. In principle, this modular architecture can also be used for pure software-based tokens, which are on the other hand even more subject to potential security threats.

As it was investigated in deliverable D 22.2 (Security Requirements) there are many assets that need to be protected on the client side in order to mitigate several main attack vectors [1]. These attack scenarios include the theft and modification of eID data or metadata as well as potential forging of identities. To protect these data there are several types of trustworthy execution environments and security anchors available, especially for mobile devices which are even more vulnerable to attacks. The most important environments have been discussed in deliverable D 35.1 (Requirements Report) [2]. Depending on the availability of these environments various Levels of Assurance (LoA) can be reached on the device side. A more concrete analysis of system architectures comprising various trustworthy environments has been conducted in deliverable D 35.2 (Interface and Module Specifications) [3]. The architectures outlined in D 35.2 show that typically several options exist in order to reach a required LoA.

While hardware tokens like eID smart cards ensure a high degree of protection and in the ideal case an end-to-end protection in the communication protocol (as with the EAC protocol), software tokens like certificates or attribute-based credentials require a much higher trust in device security. These credentials should be protected against copy, modification or extraction.

In practice the handling and access of most of the trustworthy environments, like Secure Elements (SE) and Trusted Execution Environments (TEE) imposes some additional challenges. Due to the strong fragmentation of technologies, standards and devices – especially for mobile devices – it is often difficult to ensure accessibility and availability of these environments. But even if the trusted environments are available and accessible it requires some care to integrate them into a solution in a meaningful way. Therefore, the main focus of this task has been to close some of the accessibility gaps of secure environments and to demonstrate an efficient use of enhanced security mechanisms based on existing device OS architectures.

Since mobile identities require to store credentials locally on a device and since it has been requested during the Year 1 project review to concentrate more on mobile identities, the main focus of these activities has been put on mobile devices.

5.2 Challenges of Trusted Environments

As outlined in deliverable D 35.2 the landscape of trustworthy environments is relatively diverse and therefore creates several challenges. Not all of these challenges can be addressed by FutureID but some selected issues have been covered with the performed implementation:

- **Availability**: hardware based SEs and TEEs are more and more integrated into many million devices but a full coverage can still not be guaranteed. While SIM cards are typically available in most mobile devices (typically in any smartphone but not in every tablet computer), other elements like embedded SEs or TEE are not available on a widespread basis. This issue cannot be covered by FutureID since it is based on business decisions, ownership models and use case scenarios. However, with the investigation and implementation of software-secured environments an alternative path can be offered, as will be shown in chapter 9. While this approach does not provide the same security level hardware security anchors it can be rolled out with less effort and can be used for devices where other environments are not available.

- **Accessibility**: Even if a trustworthy environment is available it may not be possible to access it either due to technical constraints or questions of ownership and the underlying business models. While the business related aspects cannot be covered by FutureID it will be shown in chapter 8 that from a technical point of view accessibility channels can be established that allow for a secure communication with a trustworthy environment.

- **Complexity**: Some of the complexity challenges result from the strong fragmentation of available solutions while others are caused by the complex management of the trustworthy environments. The implementation activities in this task cannot solve all of these issues but they can demonstrate simplified paths that allow for an easy establishment of access control mechanisms (chapter 6) and unified access channels to various types of devices (chapter 8.1).

- **Performance**: Depending on the required tasks, some of the hardware-based SEs may show too strong limitations on available computing or memory resources. While the software-protected environments can overcome this issue, they offer only limited security. On the other hand, TEEs with a hardware-based trust anchor can offer a good compromise between security and computing performance. Chapter 7 demonstrates how a TEE can be used by FutureID in a meaningful way.

6. Mandatory Access Control

6.1 Android Security Modules

The goal of the Android Security Modules Framework project is to provide a modular security architecture for the Android OS that can serve as a flexible base to instantiate different access control models. Modern operating systems for mobile devices, such as the popular Android operating system, provide rich services to installed applications through designated and well-defined interfaces. These service interfaces offer access to security and privacy critical information, such as location sensor information, access to contacts and calendar entries or telephony and short message services. This convergence of functionality into specified system services allows for system-centric access control, in case of Android based on a static permission model.

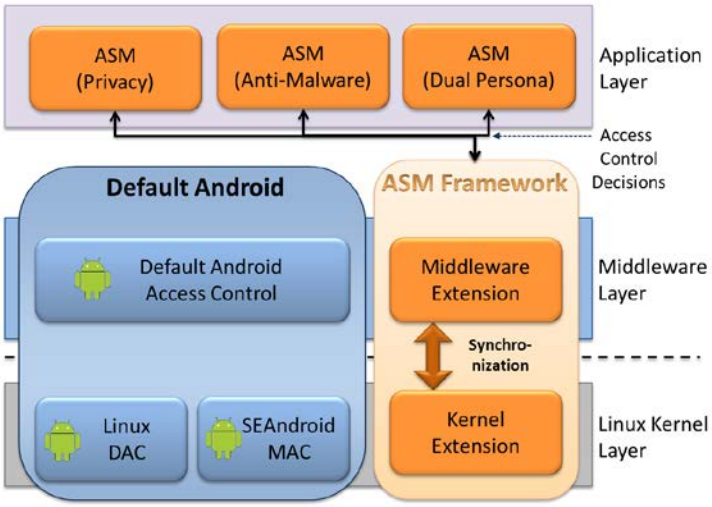

Figure 1: ASM Framework Architecture.

The Android operating system provides limited means for developers to integrate their own access control logic into this model. Moreover, it does not allow administrators, developers and end-users to define fine-grained and flexible access control rules based on use-case specific security models, such as user-defined access control rules for privacy-sensitive information to address over-privileged but non-malicious apps or sophisticated type enforcement on the middleware and kernel level [4]. In the Android Security Modules Framework project we tackle this limitation by providing a modular and extensible system-centric security framework. Inspired by the Linux Security Modules (LSM) architecture, we enable different stakeholders to integrate their own security modules, denoted as Android Security Modules (ASMs), at runtime, which augment Android‘s default permission-based access control logic with additional use-case specific access control rules.

The ASM framework (Figure 1) operates on both the middleware and kernel layer and takes into consideration the different semantics of both layers. Android‘s operating system design mandates that

access to low-level resources, such as files and network sockets, is mediated by the kernel. Since Version 4.4 Android uses a Kernel-Level mandatory access control framework, denoted Security Enhanced Android (SEAndroid) [5], which is based on Security Enhanced Linux (SELinux), to mediate access to security-critical resources and to harden the operating system against privilege escalation attacks. However, stock SELinux/SEAndroid adheres to a single policy, which is pre-defined by the device vendor, and only provides very limited support for dynamic access control decisions, as has been shown in our FlaskDroid [4] architecture.

In ASM, the Android Linux kernel is extended to allow for multiple kernel-level access control frameworks which operate in concert. The standard SELinux/SEAndroid security module enforces security invariants as defined in the stock Android Operating System. An additional security module specific to the ASM architecture communicates with the access control framework located in the middleware layer to further restrict access if required by active Android Security Modules.

The ASM framework supports multiple stakeholders (e.g., end users, device administrators and application developers) by executing multiple Android Security Modules in parallel. Each ASM operates only within a designated namespace, which describes the operations it mediates in the operating system and the applications affected by it. In general, global ASMs are considered, which control access on all resources of the operating system by all applications and local ASMs, which are restricted access control decisions in their designated namespace (e.g., all enterprise applications on a BYOD device).

The ASM Framework allows for the implementation of a variety of use cases. For example, in the BYOD context the ASM framework enables administrators to restrict access to sensitive enterprise resources on a mobile device to enterprise applications and to control access to the enterprise network infrastructure [6]. End-users may adopt privacy-preserving ASMs, which restrict access to privacy-critical information, such as contact or sensor information, to apps trusted by the user, optionally also considering the user's current security context [7].

6.2 Adapting ASM to Android 4.4.4

Initially, the implementation of the Android Security Modules Framework was based on Android version 4.4, whereas OpenMobile API was intended to run in Android 4.4.4. This version mismatch caused different errors when integrating both in Android. In order to avoid this problem, ASM was updated and made compatible with Android 4.4.4. Thus, with the current implementation it is now possible to control access rights to the OpenMobile API using the ASM mechanism. As an example, access could be restricted to the FutureID client exclusively.

7. Electronic Signature Creation based on ARM TrustZone

In this chapter we describe the creation of electronic signatures in the Trusted Execution Environment ARM TrustZone. Thereby, the goal is to protect a software-based credential from malicious actions by isolating the security relevant computations using the ARM TrustZone technology and to implement an interface that enables the seamless integration of such a credential into the FutureID client.

Before we present further details on our implementation in Section 7.3, we give an overview of the ARM TrustZone technology in Section 7.1 and briefly introduce the used environment in Section 7.2.

7.1 ARM TrustZone

Parts of this section are adapted from [8] and included here for convenience of the reader.

In a nutshell, the ARM TrustZone technology is a security extension for ARM CPUs that emulates two virtual CPUs and hardware access control to those virtualizations. This allows the implementation of two separate domains, commonly referred to as *normal world* and *secure world*.

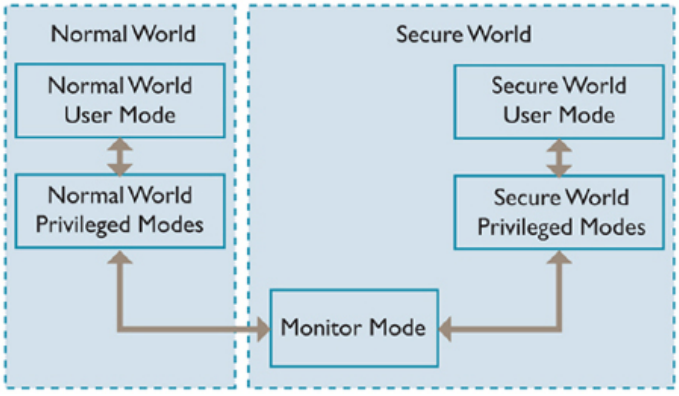

Figure 2: TrustZone Hardware Architecture [8]

While the two worlds share common resources like the physical arithmetic logical unit (ALU) and the memory, they are completely isolated. In particular, each world has exclusive access to the hardware and access to resources allocated by the other world is prevented by the hardware access control. For instance, the memory management unit (MMU), which handles the access to the physical memory, prevents the access to physical memory regions of the other domain.

By executing a certain processor instruction, or by means of external interrupts, it is possible to enter the *monitor mode*. This processor state is responsible for the actual context switch from one world to the other. In this state it is checked whether a context switch is allowed. Also the data exchange from one world to the other happens in this phase. During a switch from the one world to the other the complete processor state gets stored away and is replaced by the previous state of the world to be activated.

Each world has its own operating system. The operating system of the Normal World is also referred to as *rich OS*; in Figure 3 it is called *embedded OS*. Usually this is an Android or Linux OS. For the *secure world* there are only a few operating systems and these are mostly company driven, which impose stringent terms for secure application developers.

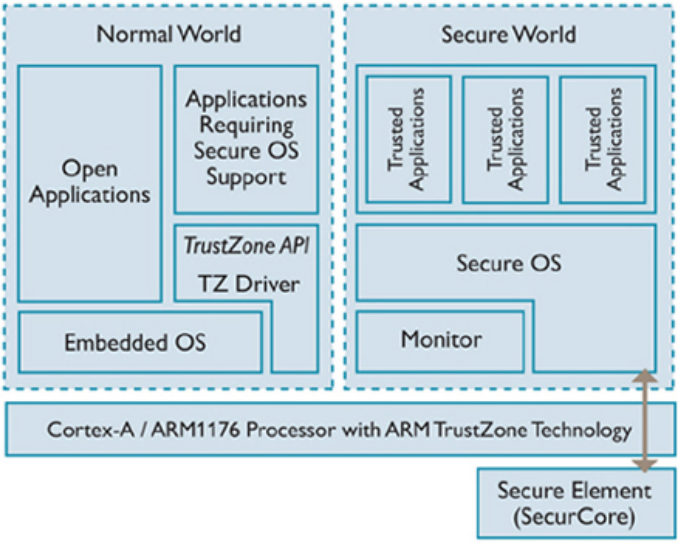

Figure 3: TrustZone Software Architecture [8]

Using a secure element and a secure boot process it is possible to guarantee the integrity of the Secure OS and the Trusted Applications that run within the secure world. Furthermore, it is not possible that even a corrupted operating system in the normal world can access resources of the secure world directly and, thus, threaten the integrity of the processes running in the secure world. Thereby, we can effectively protect a software credential from malicious access once it resides in the secure world.

7.2 Environment

We build our implementation on top of the ARM TrustZone aware operating system ANDIX OS [9] as *secure-world* OS and a Linaro Ubuntu ARM Linux as *normal world* OS. ANDIX OS provides means to load so-called trustlets (i.e., trusted applications in Figure 3) in the *secure world*. Once those trustlets are loaded, one can interact with them in a system-call like manner by using the provided *normal-world* kernel module as well as the provided *normal-world* library. As development environment we use a TrustZone-enabled QEMU [10]. For demonstration purposes it is intended to execute the code on a *freescale i.MX53*[1] development board.

[1] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=IMX53QSB

7.3 Architecture and Implementation

Before we start, we emphasize that the description of the implementation is kept very brief and we refer the reader to the source code provided in the FutureID repository and to [9], [10] for further details.

Basically, our implementation is composed of three components building up on each other.

- Firstly, there is an ANDIX OS **trustlet**, capable of creating signatures and handling keys. One can interact with this trustlet in a system-call like manner. More precisely, the *normal-world* library provides a method, taking a command identifier together with four parameters as input. The library (together with the ANDIX kernel module) then triggers an interrupt, which causes a change into the secure monitor mode. In further consequence this yields a switch to the *secure world* and an invocation of the command handler method of the trustlet with the provided parameters. Further details of the commands supported by the trustlet will implicitly be given in the description of the next layer. The signature creation inside the trustlet is performed using the Relic library [11].
- On top of the trustlet, we provide a simple **C library** for the *normal world*. This library is responsible for wrapping the functionality of the trustlet. In particular, it provides the methods shown in Figure 4 by forwarding the respective calls to the trustlet.

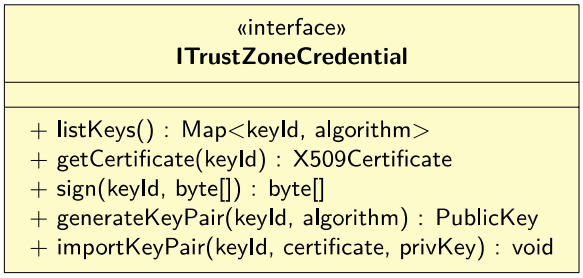

Figure 4: The JNI Interface

- Finally, on top of our C library, a **Java native interface (JNI) wrapper** is implemented. This wrapper handles the data type conversions between Java and C and acts as a Java Cryptographic Provider (JCP). One can then interact with the credential stored inside the TrustZone via simple Java cryptographic extensions (JCE) objects such as `java.security.Signature` or `java.security.PublicKey`.

This architecture follows the design principle of the IAIK PKCS#11 provider, and, thus, enables an easy integration into the eSign Services as depicted in Figure 5.

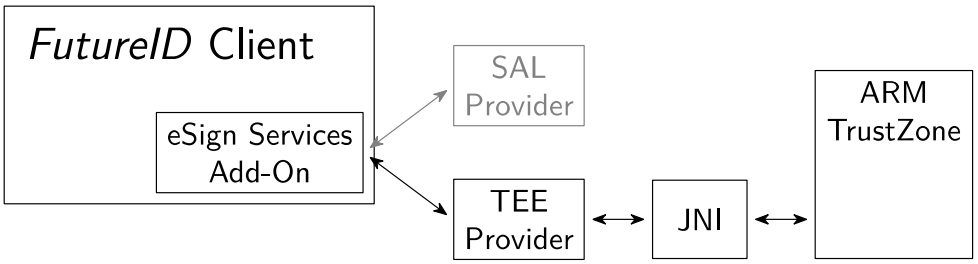

Figure 5: TrustZone Integration of the FutureID eSign services.

8. Open Mobile API Extensions

As it was described in deliverable D 31.1 (IFD – Requirements Report) [12]the Open Mobile API as specified by the SIMalliance is an API that allows applications on mobile devices to access various kinds of secure elements in a standardized way [13]. These elements can be SIM cards, secure microSD cards or other kinds of secure tokens embedded in or attached to the mobile device. 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.

The API consists of a Transport Layer and a Service Layer. The Transport Layer provides general access to the secure elements via Application Protocol Data Units (APDU) and acts as the foundation of the Service Layers. Communication takes place based on the concept of the ISO/IEC 7816-4 standard with logical channels separating communication between different applications. The Service Layer provides a higher abstraction of secure element functions, which offers additional convenience to application developers.

The following sections describe which extensions to the OpenMobile API have been implemented within this task and how they can be used to enable mobile identity use cases.

8.1 Plug-in terminal NFC/USB

Since the OpenMobile API was chosen as the IFD layer of the FutureID client on mobile devices it can be regarded as the ideal access channel to secure environments on the device. This is especially true for the SIM card which is already supported by the existing versions of the OpenMobile API. For other trusted environments however, no access channel via the OpenMobile API existed so far. Thus, for the access of USB tokens or NFC-based tokens a separate channel needed to be established.

From the security point of view managing different parallel access channels is not the desired approach. These parallel channels make a secure access control difficult and can create the risk of communication conflicts and access control conflicts. As a consequence this may result in a security risk.

In order to simplify the support of other tokens than the SIM card when using the OpenMobile API an extension has been implemented in this task which uses the plugin-terminal concept of the OpenMobile API [13]. An overview of the API architecture including the plugin-terminal is shown in Figure 6. The terminal is part of the transport layer and the concept allows to add new terminals for customer-specific types of new secure elements or secure environments. As an example, these secure elements can be contactless tokens which are accessible via the NFC interface of the mobile device or secure tokens with a USB interface. Alternatively, the access to a TEE could also be managed with this concept.

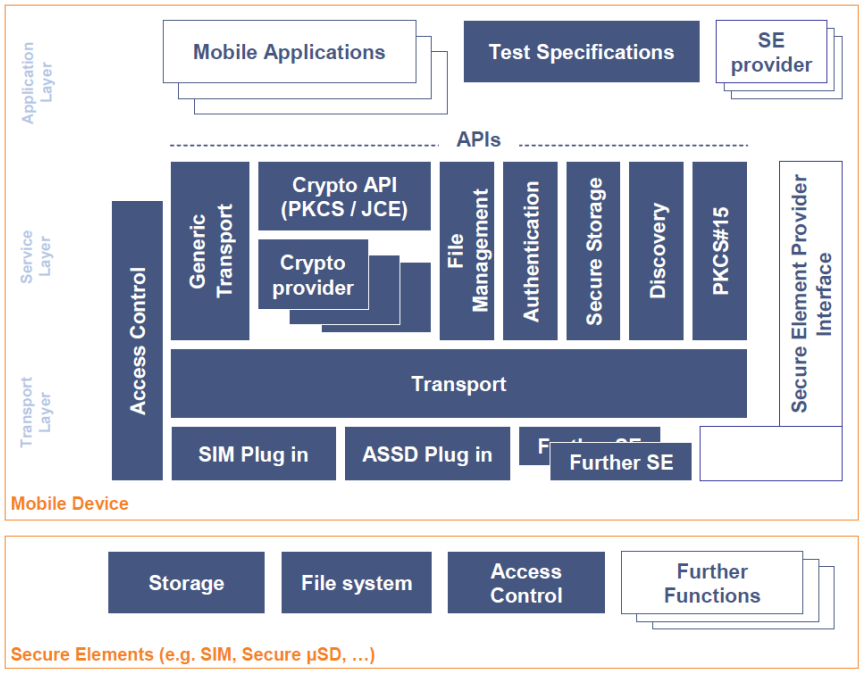

Figure 6: Overview of the OpenMobile API architecture, containing the plug-in terminal option [13].

One of the main features of the FutureID client is the communication with smart card based eID tokens. If the tokens are compatible with the NFC standards they can be accessed via the NFC interface of a mobile device without the need of an additional contactless card reader. Therefore, it was decided to implement a plugin-terminal for NFC communication. On the other hand, many tokens that are used in the PKI landscape are USB based tokens. Mobile ID or signature use cases can therefore be supported when an access channel to USB tokens is established. Therefore, a USB plugin terminal has been implemented as well.

Both terminals are based on and compatible with the open source implementation of Open Mobile API, SEEK for Android. Specifically, these terminals have been developed and tested using SEEK version 3.2.1 in a Google Nexus 5 running Android 4.4 KitKat.

Figure 7 shows how terminals are implemented in SEEK v3.2.1. There is a base class, Terminal, that implements the ITerminal interface, which represents the communication with any kind of Terminal (built-in or add-on). The AddOnTerminal is a subclass of Terminal that communicates using reflection with the Terminals located in third-party apps (as long as they follow the structure defined in [14]).

Internally, both terminals have a similar implementation: there is the main class which has the appropriate structure as required by SEEK, and then the logic for controlling the state of the system (whether the SE is present or not, etc.) is offloaded to a Service.

Regarding the NFC Terminal, it uses Android's standard NFC APIs, such as the IsoDep and Tag classes, in package android.nfc.*. IsoDep class have methods for transmitting APDUs to the Tag, so the implementation was straightforward. The main issue found during the development of the NFC Terminal is that Android APIs can only offload the handling of an NFC tag to UI components (i.e., an Activity). So it was required to add an "NfcTagHandler" Activity to the NFC Terminal, which is in charge of detecting that a new Tag is present on the field and of notifying the NFC Terminal service. The testing was done using a G&D's SmartCafe 6.0 Dual Interface card.

In regards to the USB Terminal, it is implemented on top of Android's raw USB APIs (classes UsbConnection and UsbEndpoint in android.hardware.usb.* package). In this case, Android has no implementation of basic communication to a SmartCard through USB, so a CCID protocol driver was implemented in order to deal with topics that go beyond transmitting APDUs: powering on and off the SE, waiting time extensions, correct sequencing of messages... The testing was done using an Omnikey 3x21 connected to the Android device using an OTG cable.

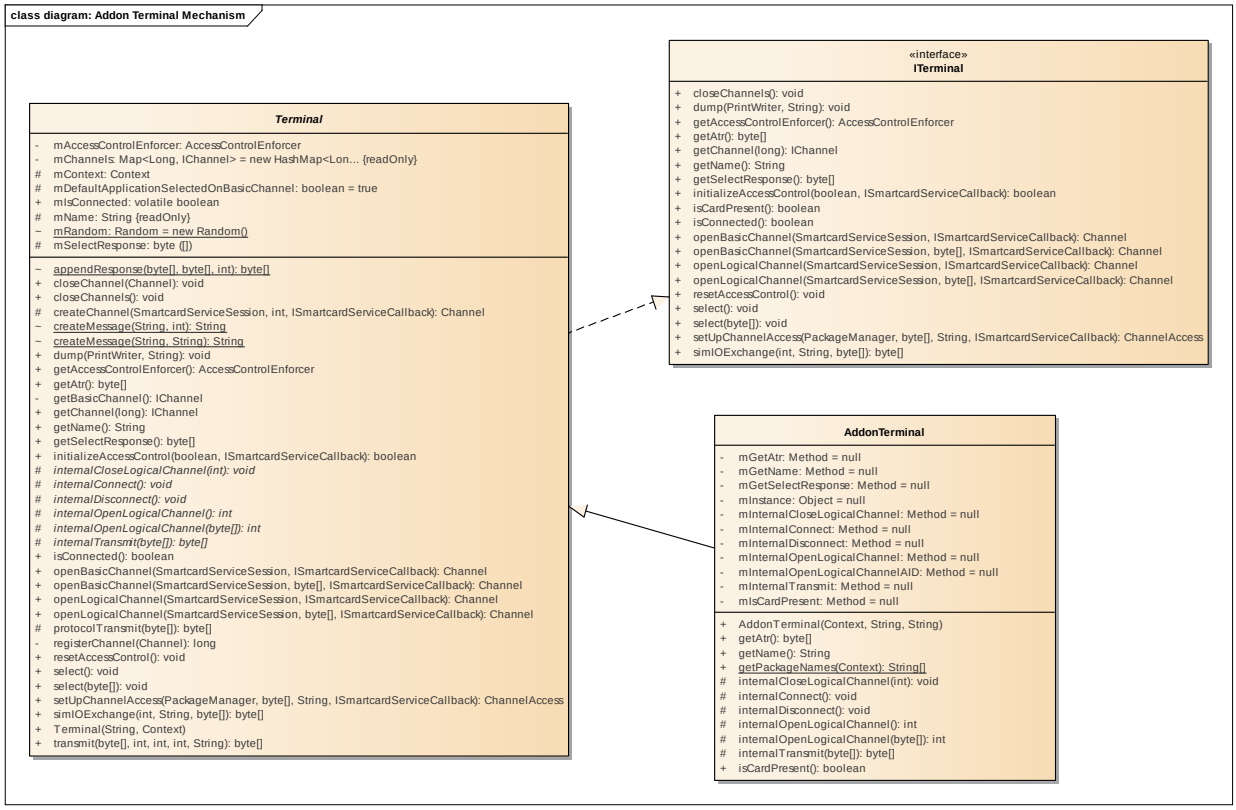

Figure 7: Overview of the terminal architecture of SEEK 3.2.1.

8.2 PKI Applet Demo

In order to demonstrate the plugin terminal implementation, a simple mobile ID use case was chosen based on a PKI applet. The Java Card applet can act as a key store and either imports external cryptographic keys and certificates or generates the keys within the secure element. The keys can then be used for mobile authentication or transaction signing as an example. Since the applet is based on the Java Card API specifications it can be used on any secure element compliant with the Java Card specifications. This can be a SIM card (UICC) or a contactless smart card or a USB token, as an example.

For demonstration purpose the applet has been deployed on three different tokens:

- A contactless smart card with a Java Card operating system (SmartCafé Expert 6.0 from Giesecke & Devrient),
- A UICC demo SIM card with Java Card operating system (SkySIM CX from Giesecke & Devrient),
- A Java Card USB token (StarSign Crypto USB Token from Giesecke & Devrient).

The applet can be selected via its specific AID and provides a key store of up to 8 public/private key pairs. Besides the actual key information each entry can optionally include a certificate. In addition, an alias name is stored that can be used for accessing the key via the JCE interface (see below). The structure of the applet key store is shown in Figure 8.

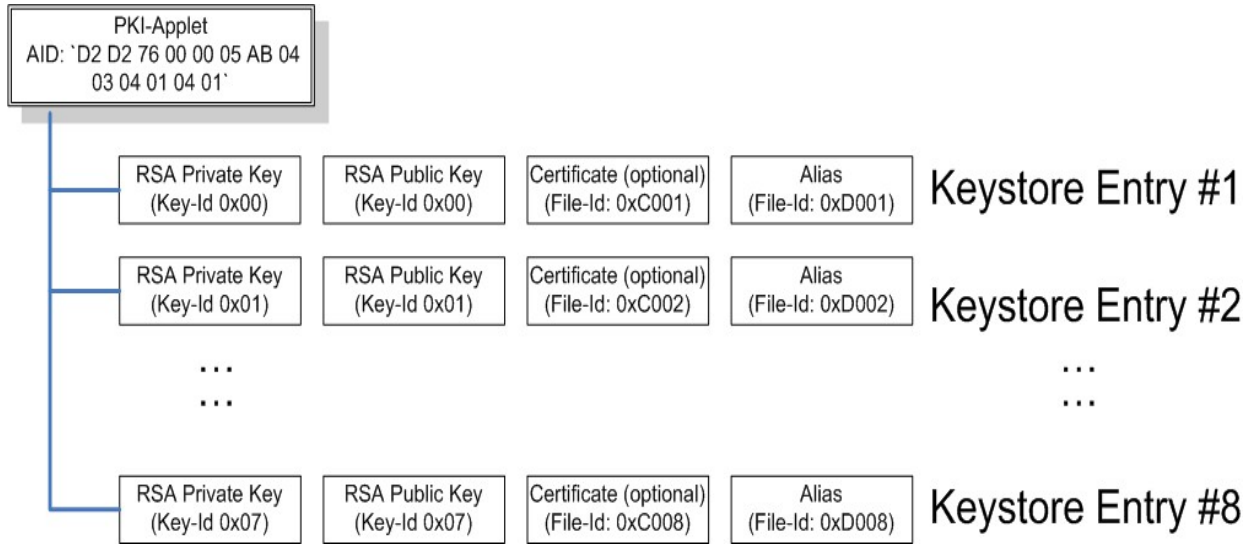

Figure 8: Key store structure of the PKI applet used for the OpenMobile API plugin terminal demonstrator.

To allow for a simpler integration into applications, a JCE provider has been implemented as well. This allows application developers to integrate the solution into an existing JCE infrastructure. JCE offers high-level methods, e.g. for encryption, decryption, verification and signature. For the FutureID client the IFD interface will be used instead.

With these implementations, a layer structure as shown in Figure 9 can be realized. This structure allows to establish a single and well controlled access channel to the different secure environments via the OpenMobile API. The highest layer comprises the JCE interface or alternatively the IFD of the FutureID client. Below, the OpenMobile API provides the Service Layer and the Transport Layer. Since the IFD of the FutureID client partly overlaps with the OpenMobile API Service Layer the integration into the FutureID client actually starts with the Tansport Layer.

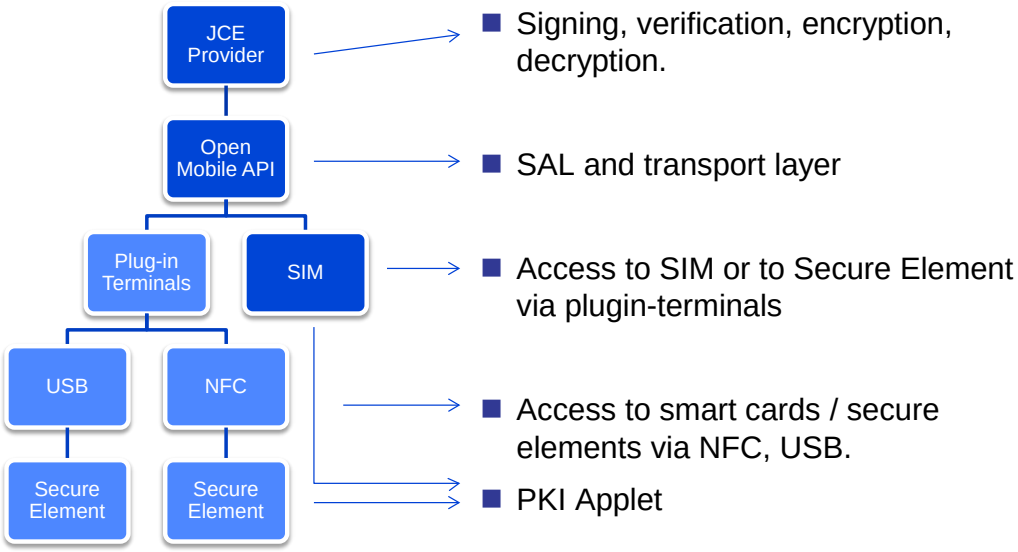

Figure 9: Layer concept of a singular access channel to SIM, USB and NFC with the OpenMobile API plugin terminal implementation.

If the applet runs on a SIM card, the standard SIM access via the OpenMobile API will be used. For the other tokens, the newly implemented plugin terminal provides the access path. Via the USB or NFC interface it can access a USB token and contactless tokens respectively.

To demonstrate the performance of this solution and its integration into an Android application, a simple demonstrator application for the Android operating system has been developed. This application uses the access path as outlined in Figure 9 and allows to switch between the different tokens.

On a first screen the applet can be initialized and the applet PIN can be defined. After this, cryptographic keys can either be imported from a certificate file or from an external key generator. Alternatively the keys can be generated within the secure element directly. An alias name for each key can be defined. With the keys it is possible to perform a simple encryption and decryption or a signature. The results of the encryption or signature creation as well as the results of the decryption or signature verification are shown. On a separate log screen the content of the log file can be show. A screenshot of some of the demo application screens is shown in Figure 10.

This demonstrator application shows that the access path as shown in Figure 9 can be realized with the plugin terminal implementation done in this task. It is usable even beyond the FutureID context and simplifies the handling of different types of secure environments significantly. The actual integration of the extended OpenMobile API into the FutureID client is currently ongoing and is performed outside this task.

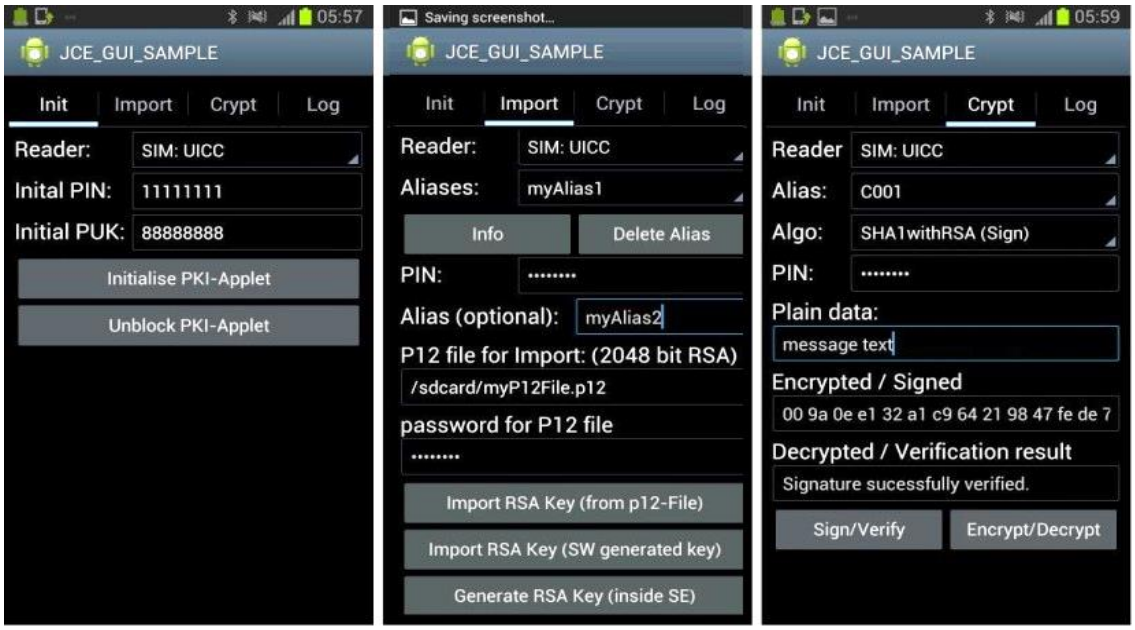

Figure 10: Screenshots of the demonstrator application showing access to different tokens with a PKI applet by using the OpenMobile API plugin terminal implementation.

To enable a wider use of this solution even beyond the FutureID context it is planned to contribute this work to the SEEK for Android OpenSource project[2], which provides an Android implementation of the OpenMobile API.

[2] See: https://code.google.com/p/seek-for-android/

9. Software Security

9.1 Introduction

While hardware-based or hardware-backed execution environments like Secure Elements and TEEs offer a high level of trust and security, they are not readily available on all devices. TEEs are about to penetrate the market further but are currently available only on a limited number of device types. Secure Elements on the other hand either have limited availability (like embedded SEs) or may not be accessible for normal applications. The SIM card, as an example, is available in every mobile phone but in most cases access is not possible without further support of the mobile network operator (if at all).

In addition, FutureID shall also include software-based credentials, like certificates or Attribute Based Credentials (ABCs). These credentials are especially vulnerable to attacks if they are not stored and handled in a trustworthy and protected environment. Thus, when no SEs or TEEs are available, there is a significant risk that these credentials may be accessible for an attacker. But even if the credentials themselves are stored in a secure environment there may be a security risk if security-critical operations with these credentials occur outside a trusted environment. In this case, an attacker might be able to extract or manipulate critical data by pure software attacks using side channels.

These side channels can be an access to the memory used by the application and extracting key material by a memory dump, or the interception of internal or external interface calls as an example. Another option is to reverse-engineer the application code using standard de-compilation tools. Especially on the Android platform using Java applications this is relatively easy to achieve. Thus any hard-coded secrets can be subject to simple de-compilation attacks.

An example of a de-compilation is shown in Figure 11. With standard tools from the Google Playstore it is possible to de-compile any unprotected application and thus to extract hard-coded secrets. As a consequence, it is advisable to further protect security-critical applications by further software-based

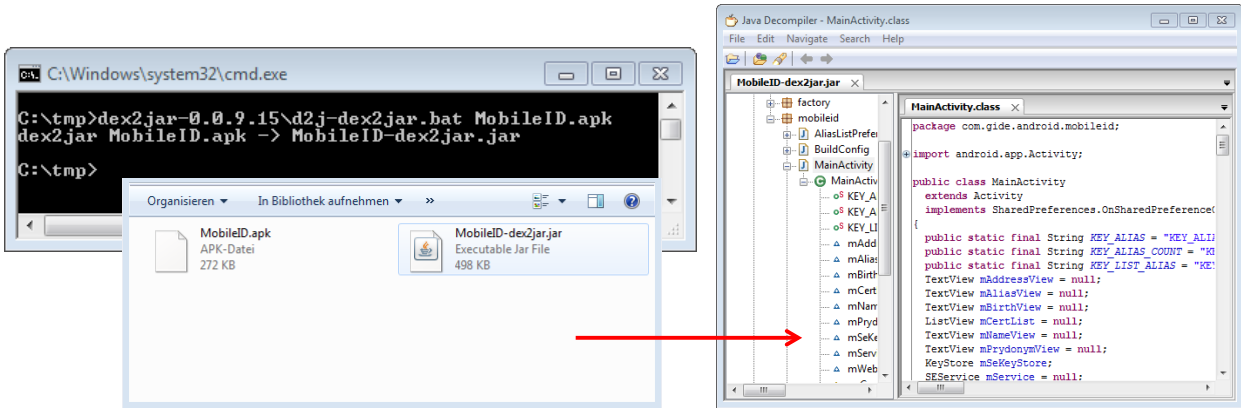

Figure 11: Example of the de-compilation of an Android application using a standard de-compiler tool. Any hard-coded secrets can be easily extracted without further protection.

measures. These measures can include:

- Use of native libraries wherever possible, in the best case a complete deployment of the application in native code,
- Avoiding library calls that can be subject to interception attacks (use end-to-end protection),
- Use of code obfuscation technologies,
- Use of Whitebox Cryptography (WBC) for cryptographic services.

In general, no system-wide secrets should be stored on the client device but only session-related credentials whose exposure will only lead to limited damage.

9.2 Protected Software Environment

Based on the considerations above, the concept of a protected software environment has been developed. This environment is a toolbox that is split into a server part and a client library providing well-defined APIs for being integrated into an application. The high-level architecture is shown in Figure 12.

The client API is a library that is linked into the client application and cannot serve as a standalone application or module. For security reasons it is a static library. The library and its interfaces are never exposed directly to an attacker as there is no public API in the final client application. As the library is a toolbox of different components to enhance the overall security level of the client application, it is split into several device components with corresponding parts in the service backend.

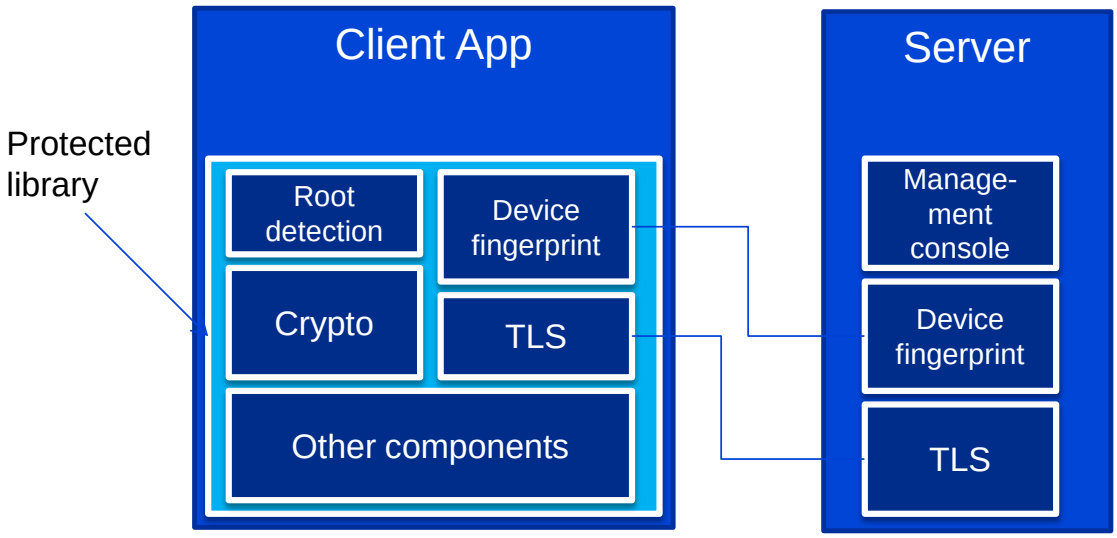

Figure 12: High-level architecture of protected software environment, including a client library and corresponding server-based services.

9.2.1 Device Fingerprinting

The device-fingerprinting component is used to generate a device-specific set of information which can be used to identify a certain device after enrolment. With the corresponding server component this information can be handled and used for two main purposes:

- Device personalization (e.g. cryptographic key material),
- Device binding.

Upon retrieval of the app from a typical app store all installations are delivered with the same keys and values on every device. If an attacker could compromise one installation, the same keys and values apply for all installations on different devices which makes automated attacks simpler. The device-fingerprinting component (in conjunction with a secure TLS connection to the server) mitigates such attacks that upon installation and first-time start of the application, device specific vectors are retrieved and sent to the server as an initial registration process. After the initial registration, the fingerprinting service backend will create device specific keys and sends them back to the device for further TLS connections or other security services.

Initially, the application is delivered with an app-specific secret (key or certificate) used for client authentication. When the application is started for the first time the library collects all available fingerprint data of the device and sends it to the server with a registration request. The registration request is signed or encrypted with the app secret, hence the sever can be sure that only official clients

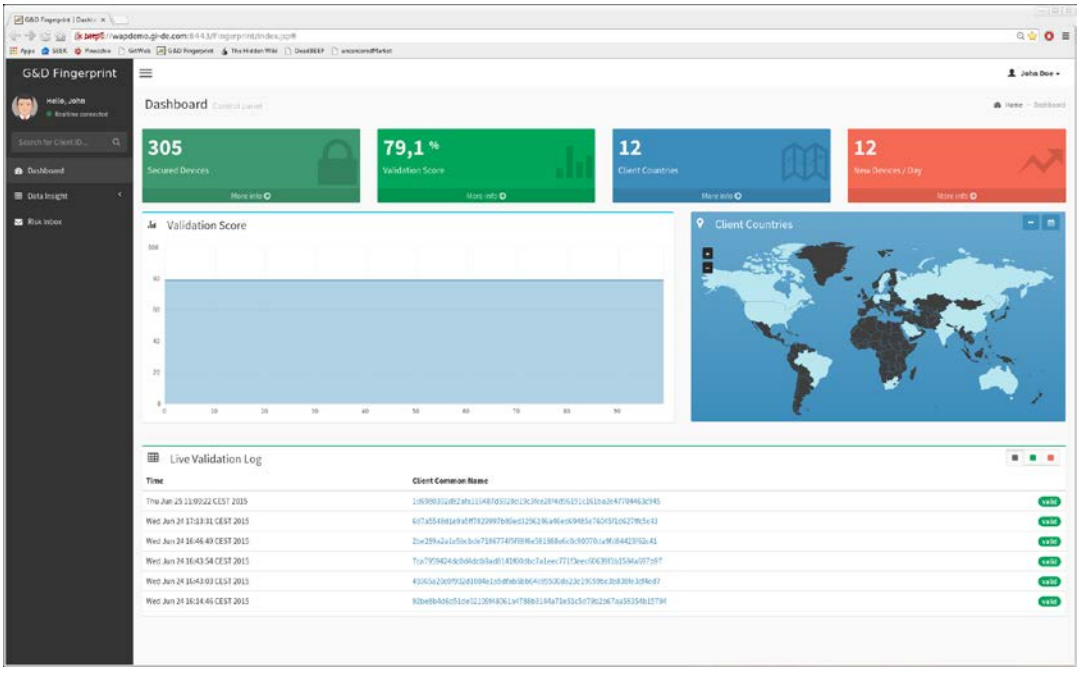

Figure 13: Screenshot of the server interface for the device fingerprinting component. It can provide further statistical data for administrative purposes.

send a registration request. When the server receives a registration request it creates a new device-specific client authentication key for further communication. The server stores the transmitted device IDs together with the authentication key. Upon further requests from the client the server executes the same calculation with the values stored during the registration and validates the correctness of the device key.

The client API consists of two function blocks: the registration of a device upon first time usage and validation of a device. The registration will gather all available device vectors and sends them to the fingerprint backend service over a secured TLS connection. As the service backend processes the device data, it creates a unique TLS key for this installation and returns the result to the client. A screenshot of the service backend is shown in Figure 13.

9.2.2 TLS Connection

The TLS component inside the API provides a TLS connection established with its own key store. It does not rely upon the implementation of cryptographic services or a key store of the underlying OS. The TLS API is based on low-level socket communication providing functionality to open a TLS connection, send and receive data over this link and close the connection.

The TLS connection is always based on a mutual authentication between client and server to mitigate Man-in-the-Middle attacks. Therefore, this component cannot be used standalone but always in conjunction with the corresponding application server backend.

9.2.3 Other components

Other components for which the APIs are defined and a concept has been developed but which have not been implemented so far are the secure storage container, the secured User Interface (UI) and the root detection. The storage container can be used to store sensitive data like ID credentials in case of FutureID. Encryption and decryption of this data is handled by the protected software environment. On the other hand, the secured UI provides a trustworthy UI for PIN entry or output of Data-to-be-signed. As it is part of the protected environment it provides a higher trust than standard OS-provided UIs.

The root detection component analyses if a device has been rooted. In this case, additional security risks apply since most of the OS security policies may not be enforced anymore. Depending on the application policy, the server can take appropriate actions for rooted devices or if the application instance was cloned and executed inside an emulator. In the same way as the backend service can trigger actions depending on the runtime environment, the client application can also initiate certain actions depending on the execution policy such as application crash, continue execution or wipe local data.

In a default mobile environment, the applications are secured through sandboxing and OS specific security considerations. However, the process of rooting (Linux-based platforms like Android) or

jailbreaking (iOS) is gaining full access to the system under a privileged account. When a device provides root access to applications it cannot necessarily be seen as attacked or critical. It will always depend on the client application and its policies if such an execution environment is allowed or not.

9.3 Future Usage in FutureID

Currently, the protected software environment has been implemented and tested as a standalone application. The actual integration into the FutureID client still has to take place but will be beyond the scope of this project. This is due to the high effort that would be required to fully integrate both components. For maximum security it is advisable to implement the FutureID client in a native language as well. Otherwise, function calls at the Java Native Interface (JNI) could be intercepted by malicious applications at the border between the secure software environment and the FutureID client. Since the FutureID client implementation has been done in Java this would require major efforts to port it to a native implementation.

Nevertheless, it is already possible to envision the potential that this solution has in the context of FutureID and beyond. Especially for software-based credentials, like certificates or ABCs, the secure storage functionality allows to protect the credentials against theft or modification. The credentials can be protected by a wrapper key that is part of the protected software environment. If it is implemented using Whitebox Cryptography as an example, the key would not be accessible to external (potentially malicious) applications since the interface is not exposed and the key is not directly visible at any time (not even during runtime).

In addition, the secured UI component can be used to enter PINs or passwords of external tokens as well as software credentials and for displaying data-to-be-signed in the signature use case. This would work without the presence of a TEE and a Trusted UI provided by the TEE, although with a somewhat lower security level. On the other hand, this software environment would work on all Android devices (and other OSs as well, when ported) and not only on TEE-equipped devices.

10. Conclusions

This deliverable describes the implementation work of security relevant modules on Android-based platforms. These modules address the topic of providing platform security for securely storing and managing secret credentials used for authentication in the FutureID context. Especially for software-based credentials (like certificates or Attribute-based Credentials, ABCs) it is important to provide secure storage containers and access control mechanisms to avoid misuse of the credentials. While in case of smartcard based credentials and secure end-to-end protocols the security is given by the smartcard, in case of software credentials secure components have to be available on the platform.

The implementation of the Mandatory Access Control modules for Android has shown that selective access control is possible, e.g. for a file-based access to the actual credential file. In addition it is possible to allow unique access to the OpenMobile API and thus to the Secure Element for the FutureID client application.

By implementing the signature components as a trusted application within a Trusted Execution Environment it is shown that critical cryptographic components can be deployed in a trusted environment and thus increase the security significantly. Although the current implementation was performed on a demo board equipped with ARM Trustzone it can be easily ported to other Trustzone-based TEE implementation, as they are already available on millions of consumer smartphone devices.

Since access to Secure Elements is typically complex to implement, especially when several transport channels shall be used in parallel, the implementation of the plugin-terminal for the OpenMobile API has shown that a unified channel can be established for SIM access, NFC tokens and USB tokens. Thus it is easy to roll out a secure applet on these tokens without the need to implement specific middleware solutions for each token. A single transport channel through the OpenMobile API also increases security since managing different channels may generate additional security threats. The concept was demonstrated with a PKI applet on a SIM card, a dual-interface card and a USB token. Such PKI applets are quite common for mobile ID solutions, thus also demonstrating the potential of the Future ID client architecture for the use in mobile ID.

As an alternative to hardware-based tokens, the implementation of a secure container protected by software-security technologies allows an easier rollout of secure authentication solutions also on mobile devices that do not support NFC for contactless tokens. The software-security environment allows a medium security level, which may be acceptable for many use cases.

Overall, the implementations show that by integrating some key components into the device platform the security of the FutureID client can be enhanced significantly. For future productive rollouts it is recommended to integrate one or more of these modules, depending on the specific use case and the targeted device platforms.

11. Bibliography

[1] FutureID, "WP 22 - Requirements Analysis, D 22.2, Security Requirements," 2013.

[2] FutureID, "WP 35 - Trustworthy Client Platform, D 35.1, Requirements Report," 2013.

[3] FutureID, "WP 35 - Trustworthy Client Platform, D 35.2, Interface and Module Specification and Documentation," 2014.

[4] S. Bugiel, S. Heuser and A.-R. Sadeghi, Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies, 2013.

[5] S. Smalley and R. Craig, Security Enhanced (SE) Android: Bringing Flexible MAC to Android, 2013.

[6] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi and B. Shastry, Practical and lightweight domain isolation on Android, 2011.

[7] M. Miettinen, S. Heuser, W. Kronz, A.-R. Sadeghi and N. Asokan, ConXsense - Context Sensing for Adaptive Usable Access Control, 2013.

[8] "ARM TrustZone Technology," [Online]. Available: http://www.arm.com/products/processors/technologies/trustzone/index.php.

[9] A. Fitzek, *Development of an ARM TrustZone aware Operating System ANDIX OS,* Master's Thesis, IAIK, Graz University of Technology, 2014.

[10] P. Wiegele, J. Winter, M. Pirker and R. Tögl, "Intrust," in *A Flexible Software Development and Emulation Framework for ARM TrustZone*, 2011.

[11] D. F. Aranha and C. P. L. Gouvea, "RELIC is an Efficient LIbrary for Cryptography".

[12] FutureID, "WP 31 - Interface Device Layer, D31.1, Requirements Report," 2013.

[13] SIMalliance, "Open Mobile API specification, V 3.1," 2015.

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

[15] S. Heuser, A. Nadkami, W. Enck and A.-R. Sadeghi, ASM: A Programmable Interface for Extending Android Security, 2014.