Digital Security: Cybersecurity, Privacy and Trust
DS-02-2014 Access Control

European Commission | Horizon 2020 European Union funding for Research & Innovation

From Real-world Identities to Privacy-preserving and Attribute-based CREDentials for Device-centric Access Control

WP3– Beyond the Password: Device-centric Authentication

Deliverable D3.1 "Description of DCA protocols and technology support"

Editor(s):	George Gugulea (CSGN)
Author(s):	CSGN: George Gugulea, Ionut Florea, Nicolae Rosia, Mihai Togan CNIT: Alberto Caponi, Claudio Pisa, Luigi Stammati CUT: Michael Sirivianos UPCOM: Vangelis Bagiatis UPRC: Christos Xenakis, Nikos Vavoulas VERIZON: Simon Dennis, Bharadwaj Pulugundla WEDIA: Evangelos Kotsifakos, Michael Pramateftakis, Michael Koulinas
Dissemination Level:	PU - Public
Nature:	R
Version:	1

ReCRED Project Profile

Contract Number	653417
Acronym	ReCRED
Title	From Real-world Identities to Privacy-preserving and Attribute-based CREDentials for Device-centric Access Control
Start Date	May 1st, 2015
Duration	36 Months

Partners

University of Piraeus	University of Piraeus research center	Greece
Telefónica Investigación y Desarrollo	Telefonica Investigacion Y Desarrollo Sa	Spain
verizon	Verizon Nederland B.V.	The Netherlands
certSIGN BY UTI	Certsign SA	Romania
wedia	Wedia Limited	Greece
EXUS	EXUS Software Ltd	U.K.
upcom Bringing business and IT together	Upcom Bvba (sme)	Belgium
THE PRODUCTIZERS	De Productizers B.V.	The Netherlands
2004	Cyprus University of Technology	Cyprus
	Universidad Carlos III de Madrid	Spain
cnit	Consorzio Nazionale Interuniversitario per le Telecomunicazioni	Italy
BAKER & MCKENZIE	Studio Professionale Associato a Baker & Mckenzie	Italy

Document History

Version	Date	Author	Remarks
0.1		George Gugulea (CSGN)	Initial Table of Contents
0.2		George Gugulea (CSGN)	Revised Table of Contents
0.3		Ionut Florea (CSGN)	Integrated version to include CNIT, CUT, TID, UPCOM, UPRC, WEDIA contribution
0.4		Ionut Florea (CSGN)	Updated version to include CNIT, CUT, UPCOM, UPRC, VERI, WEDIA contribution
0.5		Ionut Florea (CSGN)	Comments and corrections
1.0		Ionut Florea (CSGN)	Proof reading - final version

Executive Summary

This document is part of the WP3– Beyond the Password: Device-centric Authentication of the ReCRED project. The purpose of this report is to define and describe the Device Centric Authentication (DCA) protocols, user/device and device/server interfaces, including description of needed extensions to FIDO standards in the context of federated authentication and attribute based authentication. The developed DCA protocol introduces two interfaces: one between the user and a local device and one between the local device and any service on the Internet. The user will have to log-in only to the local device which subsequently handle access credentials for each distinct service with minimal user involvement.

The document also describes how the trusted device execution environment can be exploited for human-to-device authentication.

This document is organized as follows. The Introduction presents an overview on the authentication schemes, the possible extensions of FIDO with new innovative features: Federated FIDO and ABAC FIDO and the way ReCRED’s FIDO will combine the two features under a Federated Device-centric Attribute-based Authentication scheme.

The user to device authentication and how biometrics are used are described within the chapter 2. In chapter 3 is described the Device to service authentication which includes the behavioural second factor authentication, FIDO and QR login. Chapter 4 describes the way trusted Execution Environment (TEE) can be used for DCA, chapter 5 presents the privacy and security considerations in the context of DCA and chapter 6 concludes this document.

Table of Contents

List of Figures

Table of Abbreviations

ABAC	Attribute-Based Access Control
API	Application Programming Interface
ASM	Authenticator Specific Module
BAA	Behavioral Authentication Authority
CA	Client Application
CCS	Cryptographic Credentials Storage
DM	Delegation Mediating
DCA	Device-Centric Authentication
DNA	Deoxyribonucleic Acid
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
FIDO	Fast IDentity Online
FIDO UAF	FIDO Universal Authentication Framework
HAL	Hardware Abstraction Layer
HMAC	Hashed Message Authentication Code
HTTP	Hypertext Transfer Protocol
IdP	Identity Provider
JSON	JavaScript Object Notation
JWT	JSON Web Token
MDM	Mobile Device Management
MITM	Man in the Middle
MTM	Mobile Trusted Module
PABAC	Privacy-preserving ABAC
PIN	Personal Identification Number
PoP	Proof of Possession
QR	Quick Response

REE	Rich Execution Environment
RSA	Rivest, Shamir, & Adleman (public key encryption technology)
SE	Secure Element
SAML	Security Assertion Markup Language
SoC	Systems on Chip
SHA	Secure Hash Algorithm
SP	Service Provider
SSO	Single-Sign-On
SoC	Systems on a Chip
TA	trusted Application
TEE	Trusted Execution Environment
TIMA	TrustZone-based Integrity Measurement Architecture
TLS	Transport Layer Security
TPM	Trusted Platform Module
UUID	Universally Unique Identifier

1 Introduction

ReCRED will modify and extend FIDO to offer two new innovative features: Federated FIDO and ABAC FIDO. In addition, ReCRED’s FIDO will combine the two features under a Federated Device-centric Attribute-based Authentication scheme.

Traditional Single Sign On systems, such as OpenID Connect/OAuth [1] are account-based. That is, the user registers with a single identity that encompasses his online presence, such as a username along with its password or cryptographic credentials. SSO offers ease of use for the end-users as well as the system administrators by allowing the identities to be stored and verified once at the Identity Providers (also referred to as Authentication Servers). Thus a user needs not authenticate multiple times to distinct services, and it falls only on the ID provider to perform the authentication and to handle the identity attributes. The services (also referred to as relying parties or clients) do not need to maintain databases with user credentials or run complex cryptographic protocols.

To use FIDO in conjunction with Single Sign On (FIDO/SSO), we first need to modify FIDO and OpenID so that the authentication between the user and OpenID’s ID provider takes place using the UAF protocol. The account-based use case will involve the user unlocking his device, generating a public/private key pair and registering the public key with the OpenID ID provider. Subsequently, the ID provider will be the party responsible for authenticating the user and his device across the relying online services. When a user tries to login to a relying party, the party redirects the user to the appropriate ID provider. The ID provider will then ask from the user to perform UAF authentication. If it succeeds it pushes the appropriate temporary tokens to the relying party and the user.

Using OAuth, we can extend the SSO functionality to Attribute-based Access Control (FIDO/SSO/ABAC). In this scenario, the ID provider will store the distinct ID attributes of each user. The relying parties can request authentication of specific attributes, the user will authenticate to the ID provider using his FIDO device, the user will then authorize the showing of the requested credential to the relying party, and the ID provider will prove this attribute to the relying party using OAuth.

Alternatively, FIDO can be extended to support cryptographic privacy-preserving ABAC [2], [4] (FIDO/PABAC). This extension will require the online verifying service to run the verifier software stack of the specialized ABAC protocol. In addition, it will require the authenticator to run the idemix or YouProve cryptographic stack. This entails the implementation of ABAC cryptographic operations in the trusted execution environment so that the cryptographic keys involved remain as protected as the private key of a normal RSA or ECDSA UAF operation. This modality does not entail the integration with a Federated Identity Solution, because in privacy-preserving ABAC all relying parties must be able to assert cryptographically that an attribute is valid.

In the FIDO-PABAC use case, the user attempts to prove an attribute to an online service. The service will indicate that it supports an ABAC cryptographic protocol and prompt the user to unlock the authenticator with the corresponding private key for the requested attribute. The user will unlock his device using a trusted computing biometric sensor and the ABAC protocol will be executed proving the pertinent attribute to the verifier.

We also consider the Federated FIDO privacy-preserving ABAC use case (FIDO/PABAC/OAuth). This modality is motivated by the fact that not all online services (verifiers) can run a complete privacy-preserving cryptographic ABAC stack. To this end, the task of authenticating an attribute is offloaded to specialized OpenID Connect servers that act as Authentication servers. These Authentication servers run the complete cryptographic ABAC stack (idemix or YouProve) and can act as federated verifiers. The online service needs not run the ABAC software as it acts as the federated relying party (client), which obtains the attribute verification from the Authentication server using OAuth. Note that in this use case the OpenID Connect server does not act as Identity Provider, this is why we refer to it as Authentication server instead.

The privacy of users (unlinkability and untraceability) is still preserved as the relying parties need not know with which users they negotiate each time an attribute needs to be verified. The relying party needs only redirect the user to the PABAC verifier (Authentication server) which determines cryprographically and in a privacy-preserving way if that specific user has an attribute. If he does, the Authentication server pushes both to the user and the relying party a token (which includes the attribute, a secret key and a session ID), which the user can use to verify his ID attribute.

Unlike the FIDO/SSO credentials, which are generated by the device, the FIDO/ABAC credentials must be issued on the device by a trusted authority. In addition, unlike FIDO/SSO, in FIDO/ABAC the verifiers do not store a cryptographic credential associated with the user. This is the reason that verifiers cannot operate as ID providers in SSO, thus requiring all verifiers to run the cryptographic protocol. The verifiers only store cryptographic tokens associated with the trusted authority. In our architecture, the role of the trusted authority is carried by the ID consolidation service. The ID consolidation service acquires verified attributes for the user and issues idemix or YouProve directly to the device using the ID Consolidator Credential Management Module (WP4)

2 User to device authentication

2.1 Biometric identity

The term “biometrics” is derived from the Greek words “bio”, meaning “life”, and “metrics”, meaning “to measure”, and it is a method through which we can establish the identity of a person based on physical or behavioral attributes of that person.

Biometrics have been used as an identification method for many decades, and many identification biometric systems have been implemented, mainly for use in forensics, where physical evidence recovered from crime scenes (fingerprints, palm prints, DNA etc.) are compared against many biometric templates stored in large databases. More recently, biometric systems (mainly face recognition systems) are also used in order to identify potential threats (wanted persons, fugitives, known criminals or terrorists etc.) in public places.

Another recent use of biometric systems is for verification (or authentication) purposes, that is to verify that a person is who he claims to be (also known as “positive recognition”). In that case, verification (or authentication) is established by comparing a biometric captured by the person to be verified (e.g. a fingerprint) against a previously captured biometric template of the same type and from the same person. The first verification biometric systems used hand geometry recognition and were mainly used for physical access control and for recording time and attendance. More recently, though, there has been an increased interest in the use of biometrics as an authentication factor, alternatively (1F) or supplementary (2F) to the more traditional authentication factors (cards, passwords, PINs, tokens, keys etc.). Many banks across the world use biometric authentication in order to verify their customers and grant them access to ATMs[5]. There are also stores that use fingerprint recognition for biometric payments. Car manufacturers incorporate biometric authentication systems into newest cars, in order to unlock their doors or start the ignition [6]. And mobile phones can now capture and store biometric templates, allowing their owners to authenticate to their devices by using their own biometrics.

On September 2013 Apple released the iPhone 5S, the first mobile device with an embedded fingerprint scanner. Since then, millions of users across the world have been using their thumbs in order to unlock their devices, purchase applications and authenticate to remote services. During the following years, many phone manufacturers followed: Samsung, HTC, LG, Huawei, Xiaomi, Oppo, ZTE are only some of the manufacturers that have included fingerprint scanners on their phones and tablets [7]. Moreover, many manufacturers have gone a step further and have incorporated (or plan to incorporate) sophisticated iris and/or retina scanners into their newest models (Microsoft 950 XL, Fujitsu NX F-04G, ZTE Grand S3, etc.) [8]. Biometric authentication seems to have gained mobile users’ acceptance, and, according to the Biometrics Research Group, biometric smartphone users will increase, from 200 million users in 2015, to two billion users by 2020 [9].

One of ReCRED’s main goals is to replace the use of PINs and passwords for the authentication of users to remote services, through their mobile devices. In order to achieve this, ReCRED will use biometrics both as 1st and 2nd authentication factors. Initially, the use of fingerprint scanning will be

supported, along with the behavioral biometrics described in the previous chapter (keystroke dynamics). However, ReCRED is a scalable solution and will be able to adjust to new developments by supporting additional biometrics traits, such as face recognition and retina/iris scan, provided that the required software and hardware have matured and new biometric capabilities will be embedded into mobile devices.

2.1.1 Why Use Biometrics for User-to-Device Authentication

For many years, providing something the user knows (e.g. a PIN or a password) has been the most popular method to authenticate the identity of a person. Something the user has (e.g. a hardware token) is also often used, usually as a supplementary, 2nd factor authentication class for critical applications (e.g. e-Banking). However, both the aforementioned methods have some serious drawbacks that have increased the need for the adoption of a third authentication class: what the user is. Using biometrics for user-to-device authentication has many advantages over the aforementioned methods. More specifically:

- Biometric authentication is based on traits that are unique to each individual and rarely change over time, thus providing a more reliable identification method than traditional authentication methods.
- Biometric traits are very hard to forge (although not impossible), and they can’t be guessed, as is the case with PINs and passwords. Moreover, users cannot pass their biometric characteristics to other users as they can do with their passwords or cards, thus providing true and complete accountability.
- Users aren’t required to remember anything (e.g. PINs or multiple complex passwords that need to be changed frequently) or carry things with them (e.g. cards, tokens). Our identity is always with us. We cannot lose it nor forget it.

Using our biometrics to identify ourselves can be a very fast and easy process, thus providing a user-friendly and convenient authentication method. All that is required by the user is to press a button with his thumb, scan his iris or simply use his own voice. Biometric authentication can even happen transparently, as is the case with behavioral authentication.

2.1.2 Assessment of Biometric Characteristics

Biometrics can be categorized as either physiological (iris, fingerprint, DNA etc.) or behavioral (voice, gait, signature etc.). Physiological biometrics are used in order to identify and/or verify a person by using one or more anatomical or biological characteristics, including (but not restricted to): fingerprints, palm prints, hand geometry, face, iris, retina, DNA. Behavioral biometrics use the behavior of a person, that is characteristics that are learned and acquired over time, in order to identify him. They can include: voice, signature, gait, keystroke.

According to Jain et al [10], a biometric system can be assessed by assessing the following properties of the physiological or behavioral characteristic on which it is based.

- **Universality**, which measures the degree to which the characteristic can be found in the majority of people.
- **Uniqueness**, which measures the degree to which the characteristic is unique among different people.
- **Permanence**, which measures the characteristic’s resistance to change due to advancing age, illness and/or accidents.
- **Collectability**, which measures how easy and convenient it is to capture and measure the characteristic.
- **Performance**, which measures factors such as the speed and accuracy of the capturing of the characteristic.
- **Acceptability**, which measures peoples’ willingness to accept a biometric system based on that characteristic.
- **Circumvention**, which measures how easy it is to use fraudulent techniques in order to fool a biometric system based on that characteristic.

It is important to emphasize here that there is no single answer as to how suitable a characteristic is for a biometric system. Each characteristic has different attributes and it should be assessed regarding the context and the application of the biometric system to be built. A characteristic that seems to be a poor candidate for one biometric system could be an excellent candidate for some other system. Following is a short description of the main biometric characteristics, as well as their assessment as to the degree to which each characteristic could be suitable for the ReCRED’s purposes, that is to be used in a biometrics Device Centric Authentication (DCA) system.

- **Fingerprint Recognition**: Fingerprints have been used as an identification method for many centuries, although they were first studied on a scientific basis in 1892 by Francis Galton [11]. Initially extracted by creating ink impressions on paper, fingerprint patterns are nowadays captured by fingerprint sensors, based on various technologies (optical, thermal, CMOS, ultrasonic, etc.). Fingerprints are unique to each person (even to identical twins) and they don’t change due to advancing age, although they can be temporarily or permanently damaged due to accidents (burned, cut, etc.). Fingerprint scanners are very accurate and inexpensive and have recently been embedded in many mobile devices, thus making fingerprint recognition the most popular and publicly accepted biometrics authentication method. Fingerprint scanning will probably become an even more integral part of our lives, since over 50% of the smartphones are expected to have a fingerprint sensor by 2019 [12]. Fingerprint recognition is a very easy (the user only has to use one thumb) and very fast (usually it takes less than a second) authentication method. Finally, although it is possible to fool a fingerprint verification system by using forged fingerprints, the required techniques are difficult and time consuming and newest scanners become less resilient to fraud. For all those reasons, we consider fingerprint recognition as the currently ideal method for the implementation of a biometrics DCA system, and it is the one physiological characteristic that we chose to implement during the ReCRED project.

- **Palm Print Recognition**: Palm print recognition is similar to the fingerprint recognition and it uses a pattern that typically comes from the butt of the palm and that contains lines, wrinkles and epidermal ridges. Palm prints are unique and universal and they don’t change over time. Although palm print recognition is widely used by police and forensics across the world for the identification of criminal subjects, its use as a system authentication method is extremely limited. This is mainly due to the fact that palm print scanners need to capture a larger area and are more expensive. As with fingerprints, palm prints are very difficult, although not impossible, to be forged. Although palm prints can be highly rated in most of the assessment properties, the big size of the scanners is a major hindrance to their use in a biometrics DCA system, especially when fingerprints provide similar functionality at a lower cost and with higher user acceptance.

- **Hand Geometry Recognition**: Hand geometry is the first and longest implemented biometrics authentication method, since hand geometry readers debuted in the market in the mid 1980s and have since been used, mainly for physical access control as well as for time and attendance records. Hand geometry readers measure the shape of a person’s hand and they have been installed at the entrances of nuclear power plants, restricted areas in airports, amusement parks (e.g. Disneyland), they were even used for the athletes’ entrance to the Olympic Village during the 1996 Olympic Games in Atlanta [13]. Hand geometry is a universal characteristic but it is not highly unique, comparing to other physiological characteristics. Although the hand geometry doesn’t usually change during an adult’s life, big changes occur during the growth period of children and it can be affected by accidents and certain illnesses, such as arthritis. Hand geometry systems are fast, easy to use and among the most acceptable verification systems. However, the required readers are big in size, and therefore cannot be embedded into mobile devices. Overall, although hand geometry recognition can have many practical applications, it is a very poor candidate when it comes to a biometrics DCA system.

- **Face Detection & Recognition**: Face detection and recognition systems use specialized algorithms in order to compare facial features from a subject’s freshly taken photograph to those from an archived template. Face recognition is among the least reliable and effective biometric verification methods, since it can perform poorly under certain conditions, such as poor lightning, not clear or low resolution images, not neutral facial expressions, bizarre angles, dark skin colors etc. In addition, it is a method that can be forged relatively easily, by using someone else’s printed photo or by playing a recorded video (face spoofing attacks) [14]. Moreover, the identification / verification process becomes much more difficult when it comes to identical twins [15]. On the other hand, face detection’s main advantage is that it relies on a piece of H/W available in most modern mobile devices: a camera with some decent resolution. Therefore, face detection could play a role in a biometrics DCA system, but only supplementary, as an additional biometrics authentication factor.

- **Iris Recognition**: Iris recognition systems apply pattern recognition and analysis to images of a person’s irides (the annular region of the eye bounded by the pupil and the sclera - white of the eye), in order to verify the identity of that person. Not only is iris a universal characteristic but also in 1985, ophthalmologists Leonard Flom and Aran Safi proposed that irides are unique for each person, and were later awarded a patent for the iris identification concept. In fact, irides are indeed unique, even those of identical twins. Furthermore, the iris doesn’t change over time and is a very well protected internal organ. Iris recognition requires only a small sensor which

could easily be embedded in any mobile device and the iris scanning can happen from a distance, making it a fairly acceptable verification method. As with fingerprint scanning, iris scanning can be a very fast and easy authentication method. On the other hand, iris scanners are still rather expensive and only recently did appear the first mobile devices with embedded sensors. As the technology evolves and iris scanners become more widely accessible, iris recognition is likely to become a very popular and effective biometric DCA system.

- **Retinal Scan**: Retinal scan is another eye recognition method, with many similarities to the iris recognition. It is universal, highly unique (even between both eyes of the same person) and is very hard to change or replicate. Retinal scanners can be easily embedded into any mobile device and are very accurate. However, retinal scanning requires the user to peep into the scanner’s eye-piece, which makes the process somewhat inconvenient and hinders its acceptability among users. Overall, both iris and retinal scans make very good candidates for a biometrics DCA system, with retinal scans being more accurate but having lower user acceptance.

- **DNA Recognition**: Deoxyribonucleic Acid (DNA) is the most universal, unique and permanent biometric feature. However, and despite its wide recent use in forensics for the identification of criminals, DNA has serious drawbacks when it comes to user verification especially for a DCA system. First and foremost, there is currently no way to apply automatic and real-time verification using DNA, since DNA analysis requires various chemical methods and an expert’s skills. For that reason, DNA is the characteristic with the lowest collectability among all the biometrics. Moreover, most users would be reluctant to provide their DNA, mainly due to privacy issues, therefore DNA recognition has a very low acceptability. Finally, DNA recognition has a very high circumvention, since it is fairly easy to steal a piece of DNA from a person and use it to authenticate as that person.

2.1.3 Modules and Operation of Biometric Systems

Every biometric system is a pattern recognition system that contains the following main modules [16]:

- A **sensor module**, which is responsible for capturing the biometric data of an individual. Different sensors are required for different characteristics. For example, a fingerprint scanner is required for fingerprint recognition, a camera for face recognition, a microphone for voice recognition etc.

- A **feature extraction module**, which is responsible for assessing the quality of the acquired biometric data and processing it in order to extract a set of features.

- A **matcher module**, which is responsible for fetching one (in verification mode) or more (in identification mode) biometric templates stored in a database and comparing them with the features extracted by the feature extraction module. The user’s identity is then confirmed (verification) or established (identification) based on a matching score generated by the matcher module. The matcher module is not involved in the enrollment operation.

- A **system database module**, which is responsible for storing the biometric templates of the enrolled users. It can store multiple templates for each user, which may or may not be updated over time (depending on the permanence of the stored characteristic). The biometric templates can be stored in a central database, in a local database (e.g. mobile device) or on a smart card issued to the user.

As stated before, a biometric system can operate either in verification or identification mode. Regardless of the mode on which it operates, every biometric system also supports an enrollment operation, during which the biometric characteristic is initially captured (or updated) and its features are extracted (after quality assessment) in order to be stored in the system database as a biometric template. The following diagrams demonstrate the three operations of biometric systems.

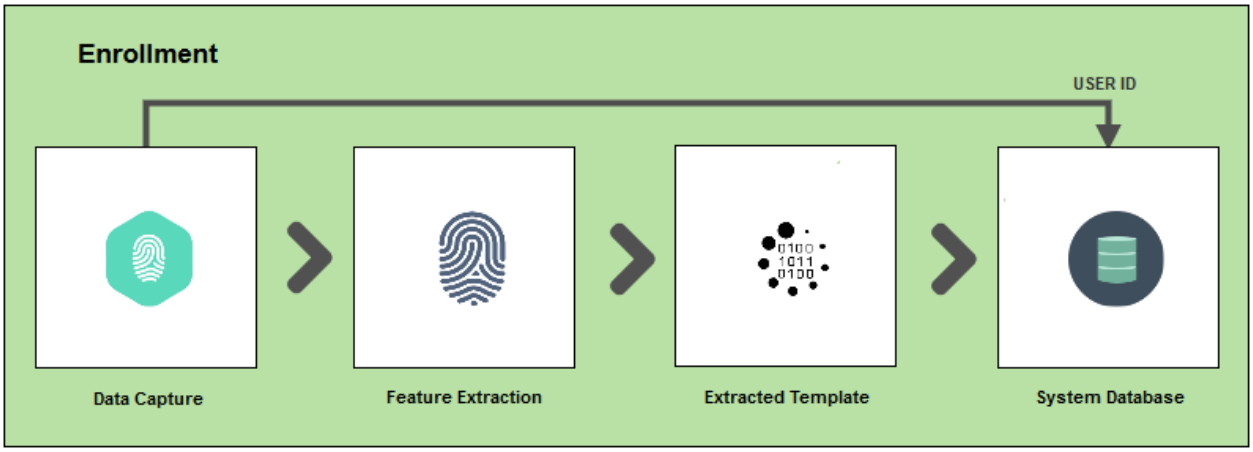

Figure 1*: Enrollment Operation*

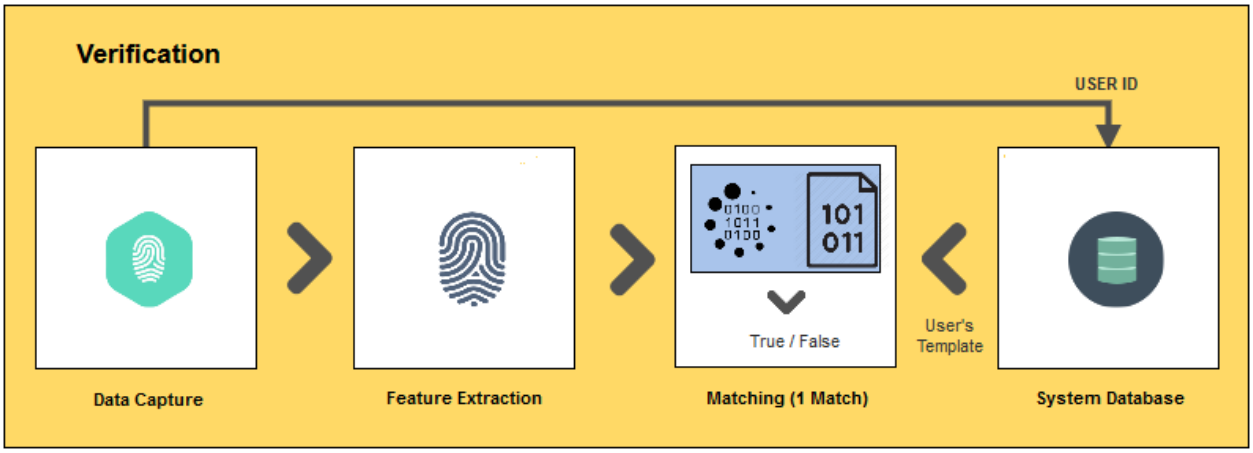

Figure 2*: Verification Operation*

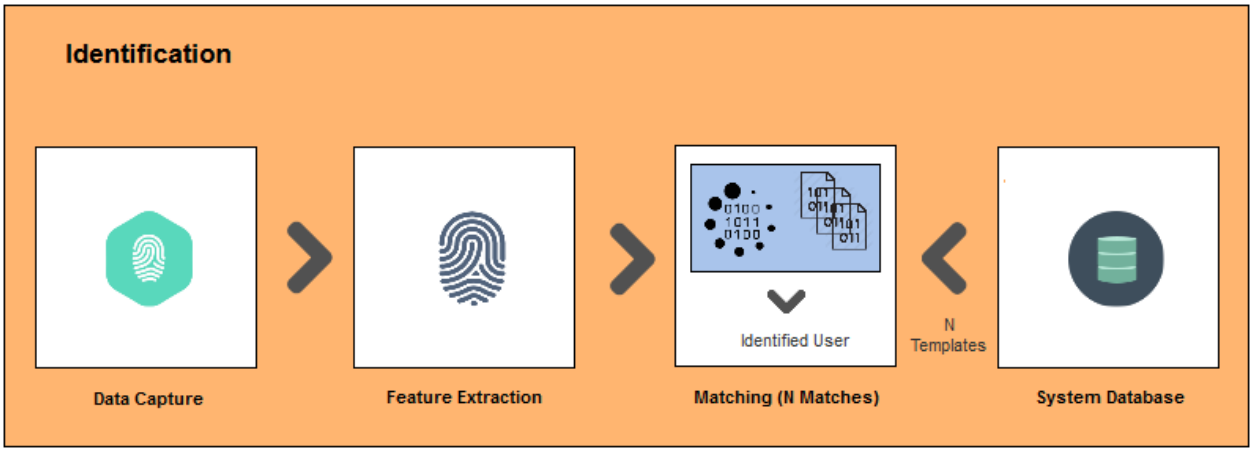

Figure 3: *Identification Operation*

ReCRED uses biometrics as a user-to-device authentication method, therefore it supports the enrollment and verification operations. During enrollment, the user is able to use his device’s sensors (initially the fingerprint scanner) in order for the device to capture the user’s data. After that, the Biometrics Authenticator, sitting on the user’s device, performs a quality check on the captured data and, if passed, it extracts the biometric feature and stores it as a template on the user’s device.

During identification (user-to-device authentication), the user again uses the suitable sensor and the Biometrics Authenticator extracts the biometric feature, after assessing the quality of the captured data. It then retrieves the user’s relevant biometric template from the device and compares it to the extracted feature, in order to verify the user’s identity.

2.1.4 Authentication Topologies

The biometric templates that are extracted during the user’s enrollment are stored in a system database, which, in the case of DCA, can be either a central database or the device itself. In addition, all the required processing (feature extraction and matching) can also happen locally on the device or remotely on an authentication server.

Karatzouni et al [17] examine two extreme alternatives: a device-centric topology and a network-centric topology. They also assess those two alternatives, taking into consideration the trade-offs that exist between various issues, such as the privacy of user data, the storage and processing requirements, the bandwidth requirements, the service availability and the mobility of each approach.

At the device-centric topology, the data capture and processing (feature extraction and matching), as well as the storing of the biometric templates, are all handled locally, on the user’s device, and no remote authentication server is used.

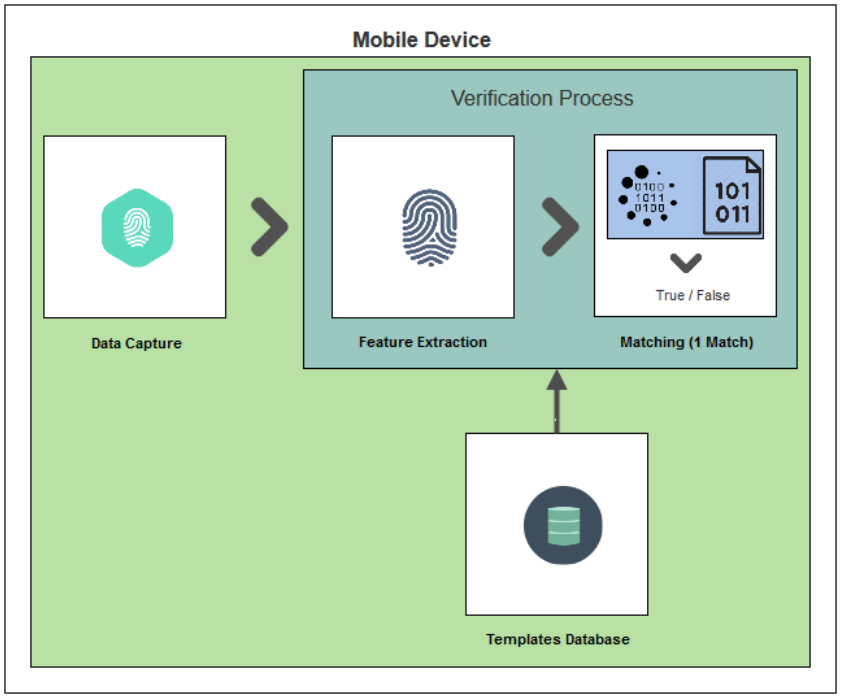

Figure 4: *Device-Centric Topology*

At the network-centric approach, only the biometric data capture happens locally, on the user’s device. All the data processing and storing is handled remotely by an authentication server.

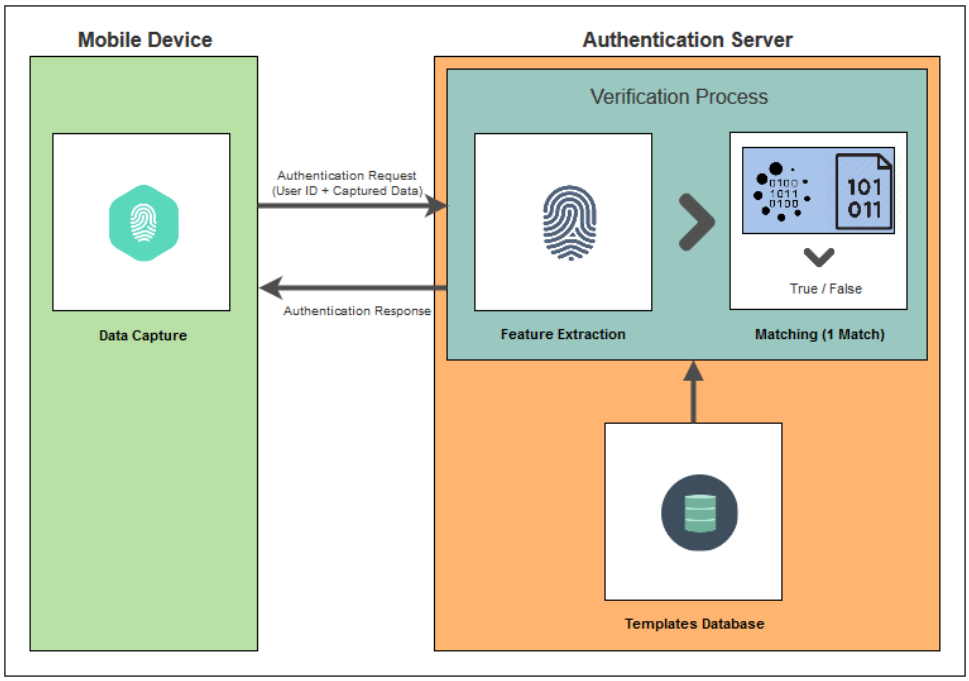

Figure 5: *Network-Centric Topology*

Overall, the device-centric topology has many advantages over the network-centric topology. No requests are being made over the internet, therefore there are no bandwidth requirements and the availability of the service is much higher. Although the storage and processing capacities of authentication servers are better than those of a mobile device, all the modern devices have high

enough specifications to easily support the processing of the enrollment and authentication operations and the storing of the biometric templates.

As for the user acceptance, regarding the security and privacy of their biometric data, no approach is 100% secure and there is always a chance that biometric data is compromised, whether it resides on the device or on a remote server. The main advantage of the device-centric approach is that if a device is compromised, only the biometric templates of that device’s owner will be breached. If a cloud server is hacked, there will be a potential breach of thousands or even millions of private biometric data. For that reason, Apple decided to store iPhone’s fingerprint data on the device (more specifically on a secure, non-accessible area called the Secure Enclave) and not on Apple servers or on iCloud [18]. Every other manufacturer also chose the mobile-centric approach, with both the processing and storing of data taking place on the device itself.

The main advantage of a network-centric topology is the mobility factor, since the user only has to enroll once for any given biometric characteristic and then authenticate himself from any device. When the biometric templates are stored on the device, then the user has to enroll again to any new device.

The ReCRED architecture follows the mobile-centric topology and all the biometric data processing (enrollment, acquisition, recognition) and storing occur in the user’s device, more specifically in the Trusted Execution Environment (TEE), by implementing Android’s Fingerprint Hardware Abstraction Layer (HAL) [19].

3 Device to service authentication

3.1 Behavioral second factor authentication

User authentication is the process of verifying claimed identity. The authentication is accomplished by matching some short-form indicator of identity, such as a shared secret that has been pre-arranged during enrollment or registration for authorized users. This is done for the purpose of performing trusted communications between parties for computing applications [22].

User authentication is categorized into three classes [30]:

- • Knowledge - based,
- • Object or Token - based,
- • Biometric - based.

The knowledge-based authentication is based on something one knows and is characterized by secrecy. The examples of knowledge-based authenticators are commonly known passwords and PIN codes. The object-based authentication relies on something one has and is characterized by possession. Usually the token-based approach is combined with the knowledge-based approach. An example of this combination is a bankcard with PIN code. In knowledge-based and object-based approaches, passwords and tokens can be forgotten, lost or stolen.

Biometric technologies are defined as automated methods of verifying or recognizing the identity of a living person based on uniquely a linked physiological or behavioral characteristics to this person [22].

Physiological biometrics refers to a person’s physical attribute, such as fingerprint, face, and iris. It is well known for its permanence and high uniqueness that promote high recognition accuracy. Unfortunately, it is not likely to be revoked if compromised (unable to change fingerprint pattern) [23], may possibly suffer low public acceptance due to invasiveness (iris scanning), and could be unlikely practical in large-scale deployment due to implementation cost (DNA analysis) [21].

Behavioral biometrics are the human intrinsic behaviors and attributes such as their voice or overall way of speaking, their signature or overall way of writing, the way the type or their keystroke dynamics, and finally their walking style or gait recognition [21].

Behavioral biometrics has a competitive advantage when compared with physiological biometrics on the ability to work in stealth mode verification. Because of that, user’s acceptability is enhanced due to the minimal interaction required during the authentication process and the reduced invasiveness. Furthermore if one’s behavioral attribute is compromised, it is likely to be replaced (changing to a new password, thus, new keystroke print or new written signature) [24]. However, in terms of variability, these merits are normally inferior to physiological biometrics in terms of variability (voice changes along with aging factor) and may consequently influence verification accuracy, no matter how encouraging they may be [21].

The stability of physiological features in time, is the reason why most security systems are based on them. At the same time behavioral features create some difficulty on their utilization as the samples of one user may vary strongly every time they are recorded. For example, one's typing rhythm may depend on several factors such as the time on which it is recorded, but also on the type of hardware used and the text being typed. Thus, to create a user’s profile in such circumstances becomes really challenging [25].

3.1.1 Behavioral second factor authentication in ReCRED.

ReCRED implements behavioral second factor authentication mechanisms based on typing style (Keystroke Dynamics) and on mobility patterns used in conjunction with local device-based password-less protocols;

ReCRED uses multi-factor authentication mechanisms based on behavioral and physiological user signatures not bound to the device, among other techniques, in order to address key security and privacy issues such as resilience to device loss, theft and impersonation.

Besides using the behavior to authenticate the user to the device, the behavior of the user will be employed to uniquely identify him to remote services or physical access points. In particular, though trusted computing mechanisms, the device can capture how the user handles the device and where he goes. These behavioral profiles will be stored and managed by the Behavioral Authentication Authorities (BAA).

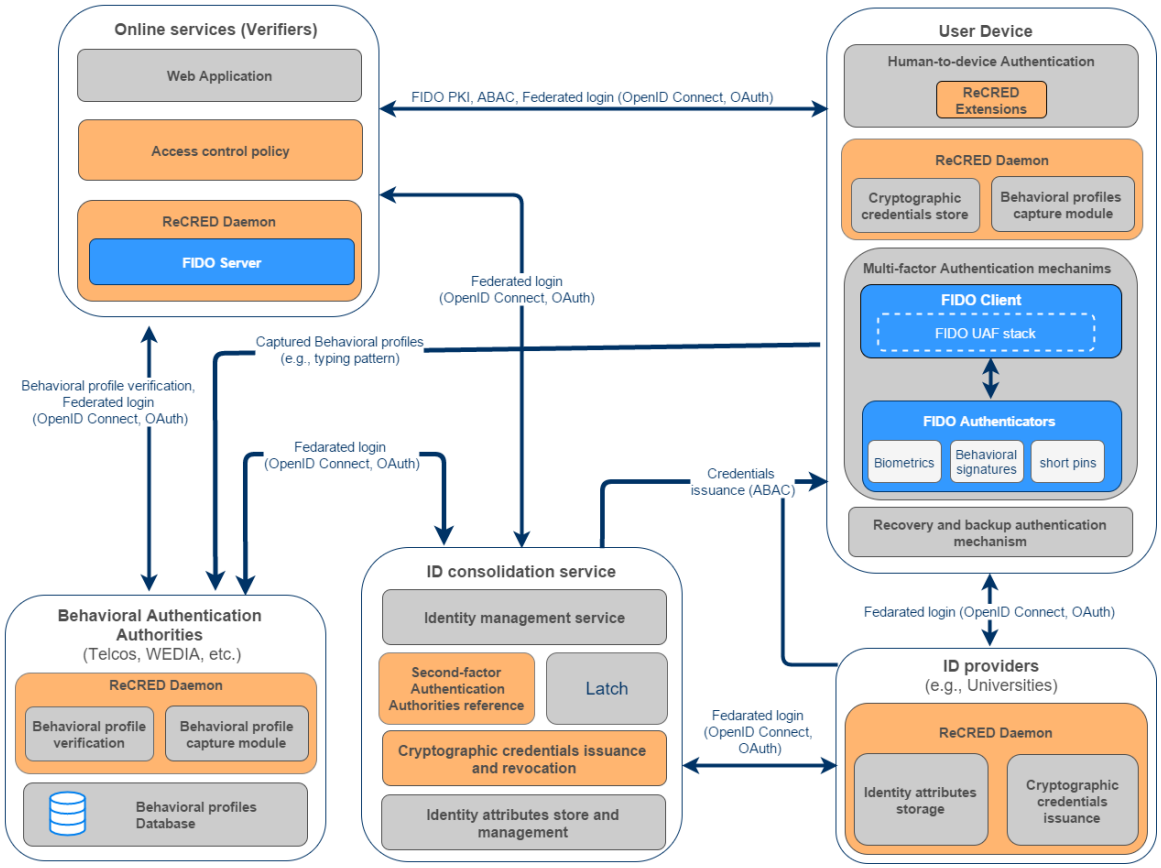

Figure 6 *ReCRED reference architecture*

According to ReCRED reference architecture (described in deliverable D2.3), Behavioral Authentication Authorities are responsible for capturing and maintaining the behavior of the users on their devices (local behavior) or on their network or online service (remote behavior). Additionally, they are responsible for performing behavioral second-factor authentication. These authorities (e.g., Telcos or online services) can capture and store, on their behavioral profile database, the behavior of the user (specifically location patterns and typing patterns, etc.). In addition, BAAs receive the locally captured from the user’s device behavioral profiles and store them for future verification.

Using the behavioral information, a BAA can determine whether a device currently behaves as it usually does. Depending on the result, it can certify to the verifier (the online service) whether it believes the device is held by its legitimate user.

The following diagram shows the process that is supported by the ReCRED BAAs.

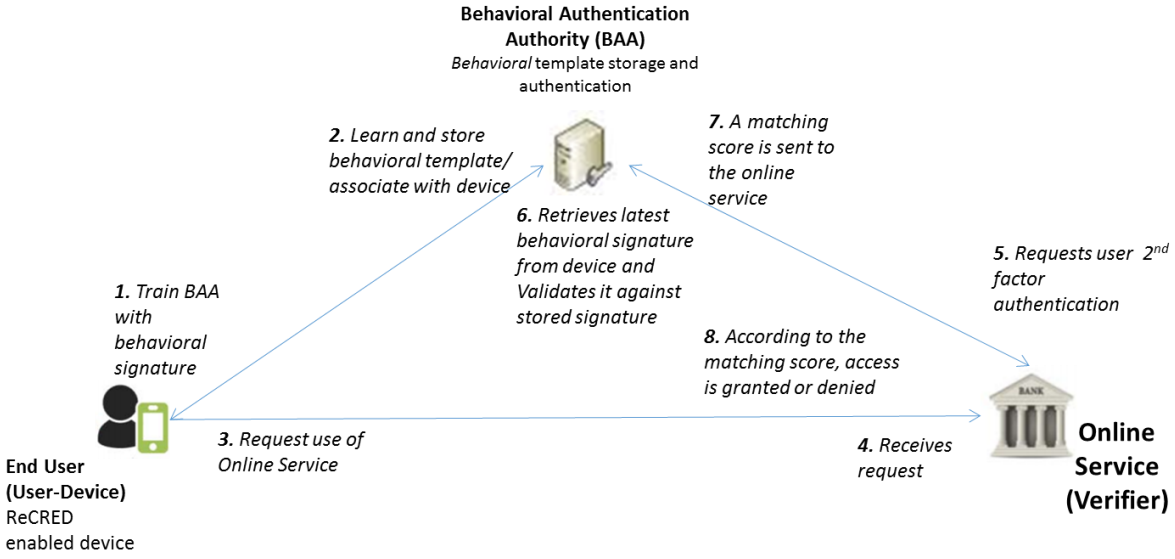

Figure 7 ***The process of creating user behavioral signature and validating it for an online service***

The steps depicted in the Figure above are the following.

1. User, through the ReCRED enabled device, sends his training behavioral information.
2. BAA application creates the typing template for the specific user/device, the behavioral profile of the user.
3. User requests access to an online service (ReCRED enabled)
4. Online service receives the request.
5. Online service requests BAA to authenticate the user.
6. BAA requests the latest behavioral template from the user device and tries to validate it against the one that exists in the database for the specific user/device. A matching score is generated.
7. BAA responds to the authentication request of the online service with the matching score.
8. The service provider checks the matching score against a pre-defined threshold and grants or denies the user access to the service.

3.1.2 The B-Verifier application

3.1.2.1 Keystroke dynamics

The behavioral biometric characteristic called keystroke dynamics, is the analysis of a computer user's habitual typing pattern when interacting with a computer keyboard [26]. Typing involves predominantly subconscious control of finger movement (for those who type regularly); it incorporates movement characteristics that are different between individuals and consistent over time [27].

The development of a typing signature that is distinguishable enough between people to be used for authentication purposes is facilitated by the habitual nature of typing [20].

The fact the keystroke dynamics does not require any special hardware as it is a very natural type of biometrics, is its main advantage. Additionally, it is possible to collect the keyboard rhythm information during the normal computer usage, so a user does not have to spend time on any additional typing, therefore it is considered much less intrusive than other methods [25].

B-Verifier application holds a database of the registered users and their personal typing “signature”-template, which is used for the authentication process. In order to build this database, B-Verifier uses sample text that users type in a web form from which typing features are captured. A clustering algorithm runs on this training set and produces a unique pattern for each user – their typing template. This template will be used each time the application is requested to verify the specific user.

When an online service wishes to authenticate the user, the latest typing template will be captured by the ReCRED enabled device and will be sent for verification. B-Verifier will then check if this template matches with the one that is stored in the database for the specific user. Finally, the B-Verifier will respond to the request of the online service with a matching score (percentage). Each online service can have its own thresholds for accepting a matching score as “verified” user and permit or deny access to the service.

3.1.2.2 Implementation

The B-Verifier application implements modules that are part of the ReCRED Daemon in Figure 6:

1. User Device – Behavioral profiles capture module
2. Behavioral Authentication Authorities (BAA) - Behavioral profiles capture module
3. Behavioral Authentication Authorities (BAA) - Behavioral profile verification
-

User Device – Behavioral profiles capture module

A B-Verifier enabled application runs on user device and captures the typing signature of the user. A set of features is extracted while the user types in the application and these features are sent to the BAA - Behavioral profiles capture module. At the same time the typing signature (behavioral signature) is stored in the device.

We are using the following features.

- Keystroke interval
- Words per minute
- Punctuation percent
- Backspace percent

- Mean word length

The set of features that are extracted is under consideration and will be evaluated against various test cases at the implementation phase.

Parameters to be considered.

The following parameters have to be configured regarding the way that B-Verifier will work.

1. Application to be used for the training and testing signature.

B-verifier module will capture typing data from one specific “favorite” application on the user’s device. This application should be user defined but needs to meet specific requirements. E.g. It should contain a specific type of text field, should be used at least twice a day etc.

2. Training period.

In order for the BAA-capture module to build a solid user typing signature, a specific number of training typing instances will be needed. This number will be defined after evaluating the output of the clustering algorithms on a real dataset that will be obtained for test purposes.

3. Validity period.

A typing signature / user behavioral profile may be valid for a specific period of time in order to be sure that we can face changes in the typing behavior of the user. This number will also be specified after evaluating real datasets that have been captured in different time periods.

4. User Device stores the signature of the last user session(s).

In order for the online service to authenticate a user, the latest typing signature of the user, should be available to be sent to the BAA. The BAA will compare this signature with the typing signature of the user stored in the database.

5. User Device does not store the reference signature.

The user behavioral profile signature should not be stored in user’s device for security reasons.

6. Signature of latest sessions on the device should be encrypted.

As the typing signature of the user can be intercepted and modified while transmitted to the BAA, this communication should be encrypted.

Behavioral Authentication Authorities - Behavioral profiles capture module

At this module, the captured typing signature (the feature vector) from the user device is processed, while in training mode. The typing features extracted are processed by a clustering algorithm and a unique pattern is created for the specific user and is stored in the behavioral profiles database.

We will evaluate clustering algorithms like the K-Means algorithm and the EM algorithm. We use the WEKA data mining tool [31] to test the efficiency of the features and the clustering algorithms.

Parameters to be considered.

1. Re-create typing signature

When the validity period of the typing signature is over, the BAA – capture module should request a new signature creation from the user device capture module.

Behavioral Authentication Authorities - Behavioral profile verification

At this module the verification of the behavioral profile is implemented. The test typing signature that is sent from the user device is compared with the stored behavioral profile of the user using a distance measure (e.g. Euclidean distance) and a score is calculated.

Parameters to be considered.

1. Reference signature validity period.

The signature that is stored in the user database should be considered invalid after a specific amount of time. This parameter should be defined in the user profile at the ID consolidation service.

2. Tolerance for the signature matching score.

A minimum threshold matching score should be set for the similarity/distance measure that will compare the current typing signature with the stored typing signature. This threshold could be defined automatically or by the user. This parameter should be defined in the user profile at the ID consolidation service.

3. Options when there is a low matching score

In the case that the matching score is very low, the application should log the attempt, notify the user (through email or through the id consolidation service). A new real-time challenge of the user is to be considered as an option while a new user to device authentication should also be considered – lock the device.

3.1.3 Mobility and Browsing signatures

ReCred will leverage the vantage point of a telecommunication provider to authenticate users based on their behavior and interaction with their mobile service provider.

In particular ReCred will provide the technology to authenticate users based on their mobility pattern and their browsing behavior. In both cases, the behavioral profiles can be computed and stored by the mobile service provider without any user interaction or burden. Therefore ReCred can cater for behavioral authentication mechanisms that are completely transparent to the user.

People exhibit fairly stable mobility patterns across their day to day activities. For example, most people move from their home to their workplace and back during working days. ReCred will therefore leverage user mobility to authenticate users of ReCred applications. Previous work [28]has shown that data produced by users interacting with a telecommunication providers (e.g., phone calls, sms, etc.) may be used to authenticate users. The system of [28] authenticate users by leveraging Call Description Records (CDR), a data structure used by telecommunication providers to store relevant data of a network event generated by a user (including caller id, callee id, timestamp, duration, etc.). However, the results shown by [28] on 14,000,000 CRDs generated by 10,000 users over 30 days, highlight usability issues since the false positive rate reach 30% for some users. In other words, employing the authentication system of [9], legitimate users may be flagged as impersonators one out of three times. The mobility authenticator of ReCred will simply use the Cell ID information contained in the CRD generated by a network event initiated by a user. The Behavioral Authentication Authority will constantly monitor and update the user mobility profile based on the CRD generated by the activity with her phone. Upon request from an online service, the BAA will check the mobility pattern of the purported user in the last time unit and match it against that user’s profile to decide whether the device is still in possession of the legitimate user. The matching function will use state-of-the-art machine learning techniques. The length of the time unit will be one of the configurable parameters that may affect accuracy. A matching score will be generated and sent back to the online service which can ultimately decide whether or not to authenticate the user. Alternatively, the BAA may only provide a binary answer to online service.

ReCred will also leverage the way people surf the web to authenticate users. Previous work [29] has shown that user could be uniquely identified by using the intersection between the set of websites in their browsing history and a set of the 6000 most popular websites. The authors of [29] asked more than 300,000 users to visit an ad-hoc website that would check the presence of 6,000 websites in their browsing histories. The intersection set was then used as a user profile and results show that profiles are largely unique, posing a significant threat to user privacy. ReCred will take advantage of the recurring patterns in browsing behavior and will use browsing histories to authenticate users. Acting from the vantage point of a telecommunication provider, the BAA will leverage the whole browsing history to build a user profile. Furthermore, profile building is completely transparent to the user as the profile is updated in the provider’s core network as the user interacts with her web browser. Similarly to the scenario where users are authenticated by their mobility pattern, in this scenario, the BAA will constantly monitor and update the user profile. Upon request from an online service, the latest browsing activity of the user is matched against her profile. The matching score or a binary answer can be provided to the online service.

To summarize, ReCred will show that both mobility pattern and browsing history can provide convenient and reliable user authentication. In both scenarios user profile building is completely transparent to the user. The user is not asked to perform any action and his experience with the mobile service provider is not affected in any way. The authentication process is also carried out without user interaction, minimizing user burden as in the overall goal of the project.

3.2 FIDO with Federated Authentication

3.2.1 FIDO Protocol overview

ReCRED uses the mobile device to realize the authentication of the user to a service. In the first step, the user authenticates to the mobile device using a biometric factor. ReCRED uses authentication realized in this step in the FIDO UAF authentication process that takes part between the local device and the online service. This authentication process uses FIDO UAF ASM API, that is FIDO UAF Authenticator-Specific Module API as it is described by FIDO alliance ([32], [33], [34], [35], [36]).

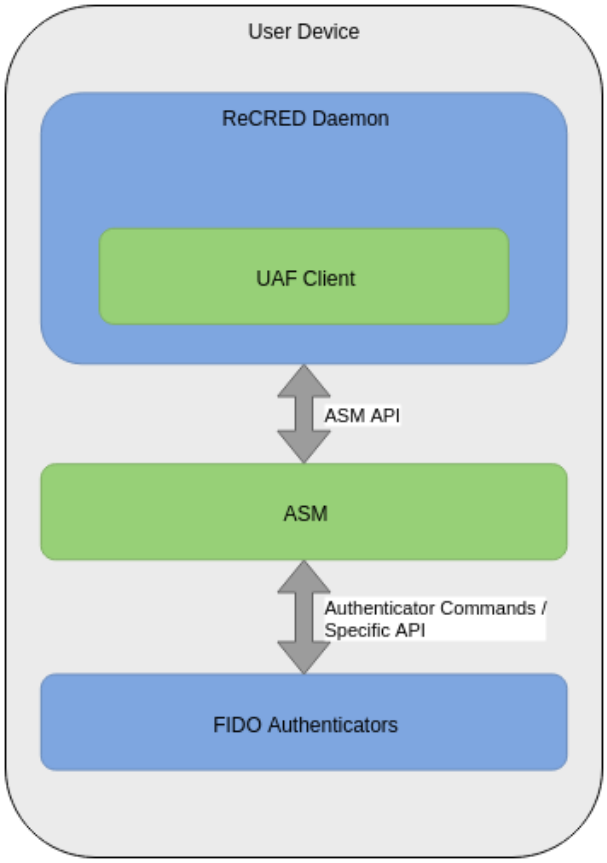

Figure 8 *FIDO UAF client block diagram*

There are three entities involved in the registration and authentication process as regarding the relation with FIDO protocols: a FIDO UAF client, a Authenticator-Specific Module and A FIFO authenticator. The UAF Client is part of the ReCRED Daemon running on the user device; it will be responsible for communicating with the ASM module through ASM API. The UAF authenticator can be in fact one or more authenticators, all of which will be queried and only one of them will be used in the authentication step.

Depending on the cryptographic scheme used in authentication, the FIDO Authenticator uses digital signatures algorithms to prove the existence of a first or second factor; this can be either a public/private key pair (RSA/ECC) or a another specific cryptographic credential defined by an assertion scheme.

The ASM component holds a list of key handlers associated with application ID and user ID along. The ReCred daemon has the responsibility to back-up this metadata associated with the key in the Identity Consolidator service.

When the user needs to access a restricted area in the Online Service, the Verifier, which is part of the Online Service initiate the authentication protocol and announces the Client that a prove of credential ownership is needed. The client which is running on the local device asks the user to authenticate to the local device and further, the local device performs an authentication to the service, realizing the authentication of the user to the service.

3.2.2 UAF Protocols Description

In order to realize authentication, there are four protocols defined:

- Registration Protocol
- Authentication Protocol
- Transaction Confirmation Protocol
- Deregistration Protocol

3.2.2.1 Registration Protocol

In Registration protocol, FIDO generates a private key and the corresponding handles, associates them and sends the proof of key possession to the server.

3.2.2.2 *Authentication Protocol Description*

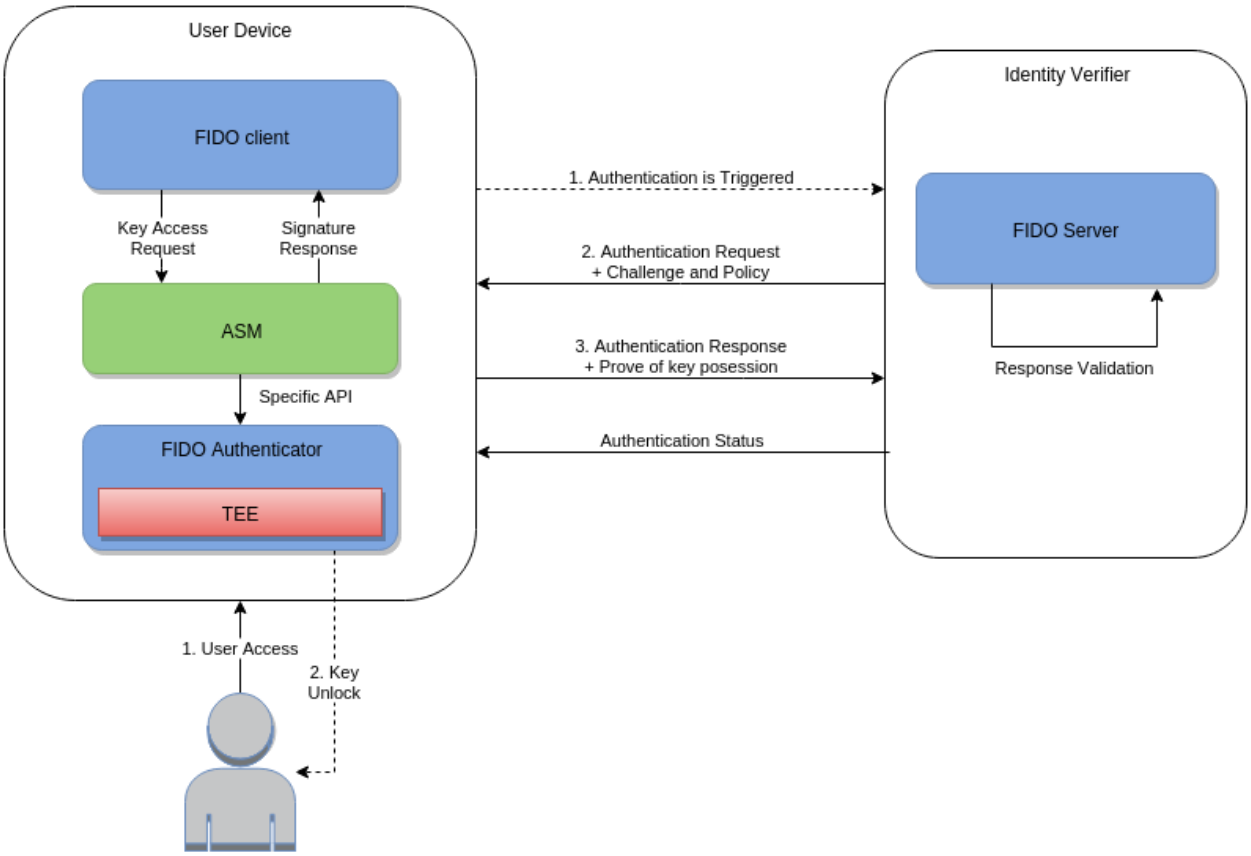

Figure 9 *FIDO UAF authentication protocol diagram*

The main operations that the ASM implements, are:

- `GetInfo`,
- `Register`,
- `Authenticate`,
- `Deregister`,
- `GetRegistrations`,
- `OpenSettings`

Besides Register, Authenticate and Deregister, the other protocols are local protocols that involves only a communication between the FIDO client, ASM and authenticator.

The communication between this entities is realized using a simple request/response protocol.

An ASM request is comprised of the following components:

ASMRequest {

required Request requestType;

Version asmVersion;

unsigned short authenticatorIndex;

object args;

```
Extension[]      exts;

}
```

3.2.3 FIDO and Federated Authentication

FIDO and Federated Authentication complementary

FIDO standardize the way human actors authenticate to online services. In this way, it improves the classic authentication mechanism (e.g. key based) by creating standard interfaces to communicate between a server, a client and a hardware device named authenticator. But this protocol resumes to authentication and does not specify what and how data travels between entities (e.g. user personal data like email, name, age)

In the same manner, federated authentication protocols does not specify how a user authenticates to a service. Independent of the authentication mechanism that the service chooses to expose to users (ex: public/private keys, symmetrical keys, passwords), federated authentication protocols creates a standard on how the user attributes and trust (obtained with a successful authentication) are transferred from one service to another

Although the two concepts can compete at some point, meaning that FIDO UAF can implement a method to transfer user attributes (e.g. through extensions), they in fact complement each other very well. As an example, FIDO separates server-side authentication logic from the client-side credential storage (more than that, even the client-side authentication logic is separated from the credential storage through abstract interfaces), it focuses on authentication and does not support attribute transport/sharing between entities and only two major entities are involved in authentication mechanism: a server and a client. On the other side, Federated protocols separates user identity from application logic, does not focus primary on authentication but gives a mechanism to enable authentication and attribute sharing and establishes a communication of user attributes from the identity provides to service providers.

Putting altogether, FIDO and federated protocols combined realizes an authentication protocol that combines the advantages of both mechanisms such that the user does not have to authenticate very often or in an unsecure manner. With the introduction of ReCRED improvements, the authentication mechanism is extended in a more secure manner by using password less authentication schemes.

3.2.3.1 Protocols description

3.2.3.1.1 Registration protocol

The following diagram describes the registration of a user to a service provider using FIDO UAF protocol as primary registration protocol and federated authentication as secondary registration protocol

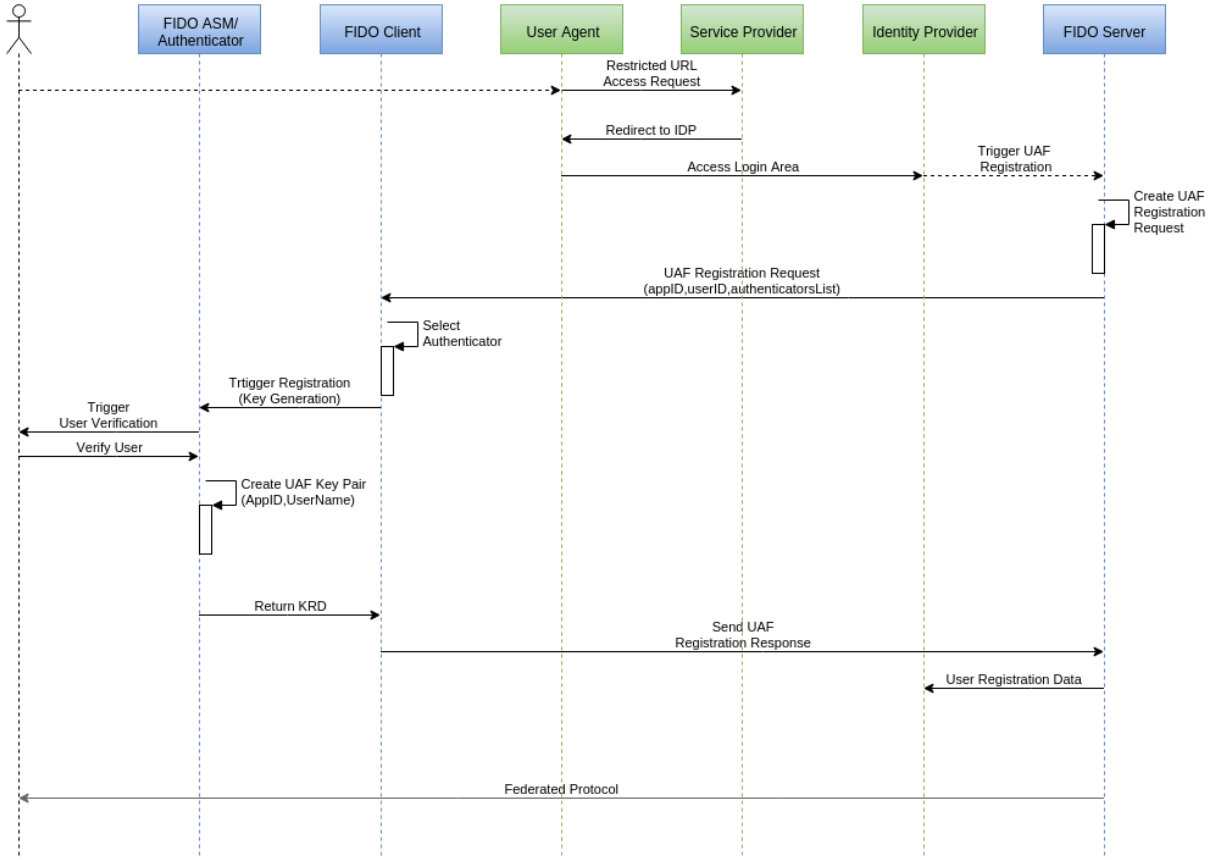

Figure 10 *User registration to a service*

The main actors involved in this protocol are:

- The User
- The FIDO Client
- The FIDO ASM and Authentication: in this diagram the FIDO ASM and Authenticator are depicted as a single actor, although they do have different specialized jobs. This jobs are specified in the FIDO protocol description.
- The User Agent
- The Service Provider
- The Identity Provider
- The FIDO Server

User Regitration Description:

1. The User accesses a private space on the Service Provider, action that is realized by the mean of User Agent (which is a browser or a Client Application). The Service Provider also contains a module that enables it to implement “service provider” part from federated authentication protocols (e.g.: OpenID, OpenID Connect, OAuth 1.0/2.0)

2. Service Provider responds with a redirect message that cause User Agent to access the Login Area on Identity Provider
3. Identity Provider triggers the FIDO UAF registration protocol implemented by the FIDO Server.

Note: FIDO Server resides on the Identity Provider premises such that the communication between them is local and trusted.

4. FIDO Server creates a UAF Registration Request message according to FIDO specifications and send this message to the FIDO Client through User Agent
5. FIDO Client triggers registration process on the Authenticator. The communication between FIDO Client and Authentication is always realized through the ASM interface.
6. The Authenticator triggers User Verification which causes a popup to be displayed on the User Device that asks for a biometric verification mechanism.
7. If the verification is successful, the Authentication and ASM creates a FIDO key pair (e.g.: ECC) based on the AppID and UserName received.
8. A Key Registration Data structure is passed by the ASM to FIDO Client
9. FIDO Client creates a Registration Response message that includes the KRD structure and sends this Response to the FIDO Server through The Identity Provider.
10. FIDO Server verifies the response and finalizes the FIDO Authentication
11. From this point on, the federated authentication protocol continues to finalize the registration

Protocol Messages

- **RegistrationRequest** message contains:
  - OperationHeader header;
  - ServerChallenge challenge;
  - DOMString username;
  - Policy policy;
- **Policy** is a collection of attributes that specifies the accepted and rejected authenticators.
- **OperationHeader** contains:
  - Version upv;
  - Operation op;
  - DOMString appID;

- DOMString serverData;
- Extension[] exts;

- **RegistrationResponse** message contains:
  - OperationHeader header;
  - DOMString fcParams;
  - AuthenticatorRegistrationAssertion[] assertions;
- **FinalChallengeParams** contains
  - DOMString appID;
  - ServerChallenge challenge;
  - DOMString facetID;
  - ChannelBinding channelBinding;

Registration Request message example


```
[{
        "header": {
                "upv": {
                        "major": 1,
                        "minor": 0
                },
                "op": "Reg",
                "appID": "https://uaf-testtest.com/SampleApp/uaf/facets",
                "serverData":  "SR_[…]g"
        },
        "challenge": "H9iW9yA[…]zCnCqKDpo",
        "username": "apa",
        "policy": {
                "accepted": [
```

```
                        [
                        {
                                "userVerification": 512,
                                "keyProtection": 1,
                                "tcDisplay": 1,
                                "authenticationAlgorithms": [1],
                                "assertionSchemes": ["UAFV1TLV"]
                        }
                        ],
                ]
        }
}]
```


3.2.3.1.2 Authentication protocol

The following diagram depicts the user authentication to a service provider using FIDO UAF protocol as primary authentication protocol and federated authentication as secondary authentication protocol

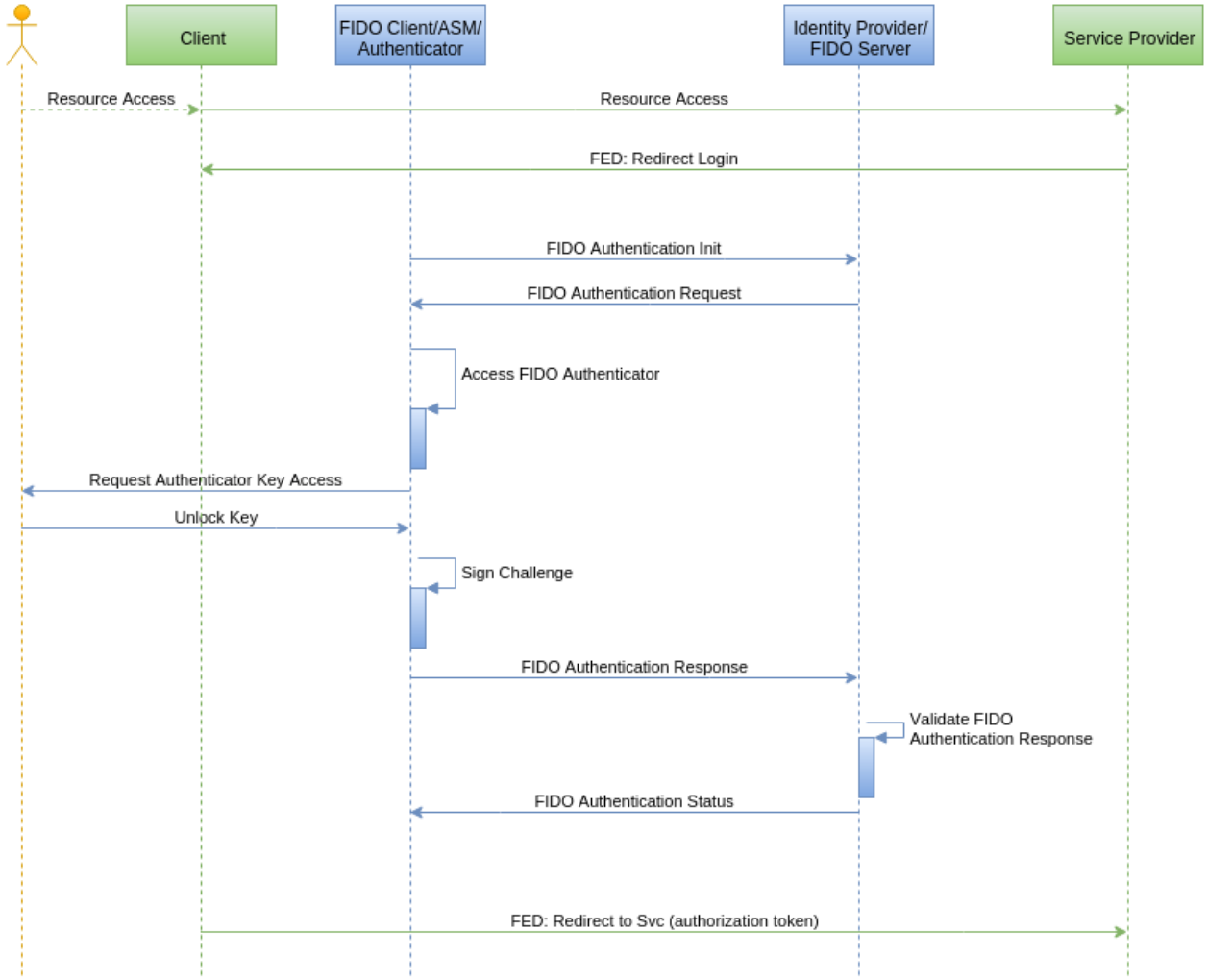

Figure 11 *User authentication to a service provider using FIDO UAF and federated authentication*

The main actors involved in this protocol are:

- The User
- The Client Application
- The FIDO Client: in this diagram the FIDO Client, ASM and Authenticator are depicted as a single actor, although they do have different specialized jobs. This jobs are specified in the FIDO protocol description.
- The FIDO Server (and Identity Provider)
- The Service Provider

Authentication description:

1. User access a private space on the Service Provider, action that is initiated by the user through the Client (which is a browser or a Client Application). The Service Provider also contains a module that enables it to implement “service provider” part from federated authentication protocols (e.g.: OpenID, OpenID Connect, OAuth 1.0/2.0)

2. If the client is not authenticated (e.g. does not send an authentication token – cookie), the Service Provider initiates the federated authentication by sending a redirect message to the Client Application. The redirect message contains the redirect address of the Identity Provider/FIDO Server and, optionally, the redirect back address (the address of the service provider)

3. The Client Application (e.g. Browser) is redirected to the Identity Provider login page, initiating the FIDO UAF Authentication protocol. The Identity Provider sends this messages to the FIDO Server and the FIDO Authentication part of the protocol starts.

Note: If the user is already authenticated (due to a previous login action), the authentication step is skipped

4. The FIDO Server responds with a FIDO UAF Authentication Request message that arrives at the FIDO Client

5. The FIDO Client analyses the Request message and sends a private key access to the FIDO Authenticator that is enabled on the local device

Note: The three entities that are part of this actor (FIDO UAF Client, ASM and Authenticator) communicates one to each other through the FIDO UAF protocol specified in the previous chapter

6. The User is prompted with a private key access request. The private key is contained inside the FIDO UAF Authenticator and the access is realized through the standardized ASM interface.

7. The User unlocks the private key material through a local authentication mechanism (e.g. biometric authentication)

Note: there are other actions that happens in this step regarding private key unlocking: authenticator discovery and chosen and possible specific messages display, etc.

8. The Authenticator signs the protocol challenge and passes this to the FIDO Client that constructs a FIDO UAF Response, according to the UAF protocol

9. The FIDO Server validates the authentication Response and sends back to the FIDO Client a status message with the authentication result.

10. In case of a successful authentication, the Identity Provider/FIDO Server creates an access token (specific to the federated authentication protocol in use) and sends it to the Client Application

Note: the use of access token is simplified in this scenario, there can be other tokens like: authorization code that is valid only one time, a code that the client uses to obtain the access token, refresh token, a token that enables the Client to obtain other access tokens in case of longer time periods are needed to keep the user authenticated

11. The Client Application sends access token to the Service Provider redirect address specified in the step 2 and so the federated authentication completes

ReCRED enables the use of Attribute Based Encryption schemes in the authentication process, as the use of a Trusted Execution Environment on the user device as well. This does not change the way that the two authentication protocols communicates and combines the messages one with each other to achieve the desire effect of secure authentication with delegated authentication, but only changes the way that the primary authentication mechanism is realized.

Identity Provider and FIDO Server enables the usage of U-Prove and/or Idemix protocols by introducing special entities in the protocol (e.g. a protocol adapter on the server side to combine the authentication mechanisms). The details about the introduction of the two Attribute Based Access Control protocols are not the subject of this chapter but are covered in different chapters.

FIDO UAF Authentication protocol as well as usage of Trusted Execution Environment as key containers are also described in details in the other chapters of this document.

3.2.3.1.3 Deregistration protocol

Using the deregistration protocol the Service Provider, Identity Provider and user device components deletes the generated key pairs associated with a service, an application and a specified user. On the User Device, even if the authenticator does not store key handles, this key handles are stored on the ASM component. This means that the ASM should delete the key handles as well. The FIDO Client can also contain data associated with the registered user that has to be deleted.

Protocol Messages

- **DeregistrationRequest**
  - OperationHeader header;
  - DeregisterAuthenticator[] authenticators;
- **DeregisterAuthenticator**
  - AAID aaid;
  - KeyID keyID;
- **DOMString AAID**;

Deregistration Request message example

[{

"header": {

"op": "Dereg",

"upv": {

"major": 1,

"minor": 0

},

"appID": "https://test.com/SampleApp/uaf/facets"

},

"authenticators": [{

"aaid": "ABCD#ABCD",

"keyID": "ZM[…]Ng"

}]

}]

3.2.3.2 OpenID Connect Dynamic Client / Relying Party Registration & Management with OpenID Provider

As per OpenID specification, OpenID Connect 1.0 is an identity layer that is built on top of the OAuth 2.0 protocol. It enables clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-user in an interoperable and REST-like manner.

The main purpose of this document is to define how we can on-board OpenID Connect Client dynamically with OpenID Provider. This document also defines use-cases in which the properties of a Client may need to be changed during the lifetime of the Client.

Registration Workflow

This flow diagram illustrates the interaction between OpenID Connect Client and the Authorization Server (registration and configuration endpoint).

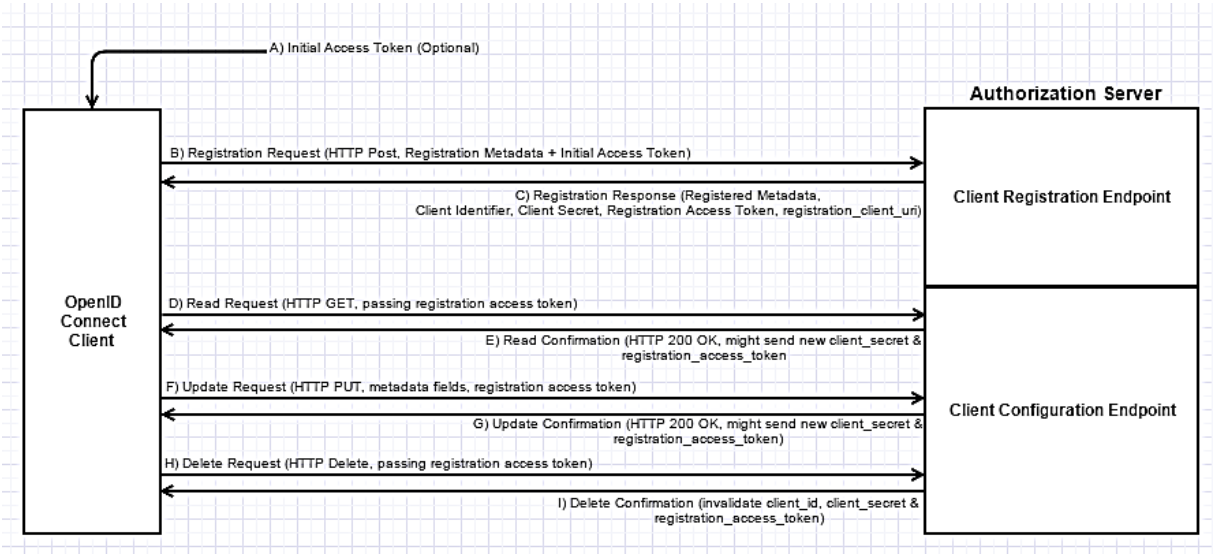

Figure 12 ***Interaction between OpenID Connect Client and the Authorization Server***

This flow diagram includes the following steps:

A) Optionally, the OpenID Connect Client is issued an initial access token that might be provisioned out-of-band to restrict registration requests to only authorized Clients.

B) To register a new Client at the Authorization Server, the Client sends an HTTP POST to the Client Registration Endpoint with a content type of "application/json". The HTTP Entity Payload is a JSON document consisting of a JSON object and all requested client metadata values as top-level members of that JSON object. The Client also sends the initial access token along-with the request.

C) The Authorization Server registers the client’s registered metadata, a set of client credentials such as a client secret if applicable for this client, a URI pointing to the client configuration endpoint, and a registration access token to be used when calling the client configuration endpoint.

D) Optionally, to read the current configuration of the client on the authorization server, the client makes an HTTP GET request to the client configuration endpoint, authenticating with its registration access token.

E) The authorization server responds with an HTTP 200 OK Message with content type of "application/json" and a payload. Some values in the response e.g. client_secret and registration_access_token might be different from those in the initial registration response. If the authorization server includes a new client_secret and registration_access_token in its response, then the client must discard its previous client_secret and registration_access_token. The value of the client_id must not change from the initial registration response.

F) Optionally, to update client’s registered data, the client sends an HTTP PUT to the client configuration endpoint with a content type of "application/json". This request should contain the previous registered metadata fields with updated values, excluding registration_access_token, registration_client_uri, client_secret_expires_at, client_id_issued_at fields. The client should also include its previous client_id and client_secret field in the request

G) The authorization server responds with an HTTP 200 OK Message with content type of "application/json" and a payload. Some values in the response e.g. client_secret and registration_access_token might be different from those in the initial registration response.

If the authorization server includes a new client_secret and registration_access_token in its response, then client must discard its previous client_secret and registration_access_token. The value of the client_id must not change from the initial registration response.

H) To deprovision itself on the authorization server, the client makes an HTTP Delete request to the client configuration endpoint. The client passes registration access token along-with this request.

I) If a client is successfully deprovisioned, the authorization server responds with HTTP 204 No Content Message. During deprovisioning, the authorization server invalidates the client_id, client_secret and registration_access_token for the client.

3.2.4 FIDO in context of Attribute-Based Authentication Protocols (IdeMix, U-Prove)

Privacy-preserving ABAC (PABAC) protocols use different cryptographic operations to offer user authentication while preserving anonymity. To achieve this, the authentication schemes rely upon cryptographic schemes that guarantee the unlinkability and untraceability of attribute authentication. The first refers to the inability of the verifier to discern whether the same user shows proofs of distinct attributes to him, and the second to the inability of a group of verifiers to determine that the same user shows proofs of the same attribute to all of them.

ReCRED takes advantage of the user device trusted computing elements to keep the PABAC cryptographic keys secure in trusted storage, thus further enhancing the PABAC protocol's privacy assurances by performing specific operations with this private credentials in such a way that they never need to leave the secure storage. Using FIDO UAF Authenticators the attributes can be securely stored and used in the FIDO UAF protocol that will be modified in such a way that the two protocol credentials IdeMix/UProve will be supported.

Keeping the ABAC credentials on a secure storage, or on an unsecure storage but always encrypted with a secure key (kept in a secure storage) will ensure that even in the case of a stolen device the attacker will not be able to decrypt the credentials and therefore to use them in an authentication process.

3.3 QR Login

QR Login realizes an authentication bridging between desktop and a mobile phone that contains user credential by taking advantage of the device’s camera. For this mLogin application shall be further extended.

The four involved entities: mobile phone, user desktop, Service Provider and the QR Authenticator realize a temporary identity transfer from the smart phone to the desktop by using QR codes.

The service provider sends a temporary identifier to the user desktop in the form of a QR code, the smartphone scans it and sends it to the QR authentication server. With this process the mobile and the desktop (or any other device) are coupled and associated as belonging to the same user.

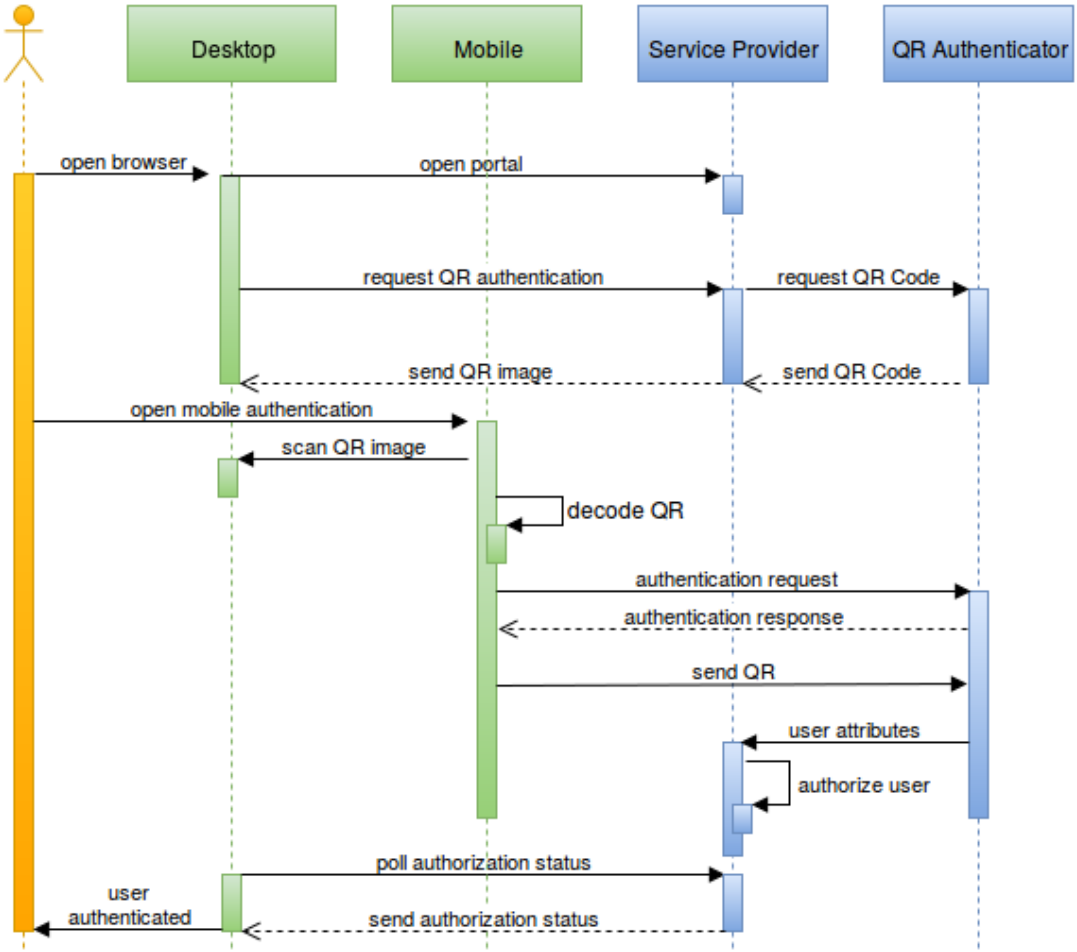

Figure 13 *QR Authentication*

The desktop authentication flow using mLogin and QR codes is depicted in Figure 13.

The user who wants to access a particular resource must be authorized by the Service Provider. In order for a user to be authorized, the Service Provider must receive the proper attributes from the QR Authenticator: credentials or security assertions.

A trust relation is established between the Service Provider and the QR Authenticator by using a shared secret. All the messages exchaged between the two server entities must contain a HMAC signature (HMAC-SHA512).

3.3.1 Authentication description

1. The user opens the portal using the web browser from its computer and start a QR Authentication.

2. The Service Provider generates a random token associated with the current user session and sends a QR generation request to the QR Authenticator module by using the session token. The Service Provider must not disclose any user session related information in order to mitigate vector attacks like session hijacking. The Service Provider request must contain a policy which describes the requested attributes in order for the user to be authorized.

3. The QR Authenticator verifies that the QR generation request is from a trusted entity, generates a QR code (nonce) associated with the received session token and along with an expiration time saves it into a database. The QR Authenticator sends the generated code to the Service Provider.

4. The Service Provider generates a QR code image using the information received from the QR Authenticator and sends it to the user browser.

5. The user starts the mLogin application and scans the QR code image displayed in the browser.

6. The mLogin application decodes the QR code image and starts an authentication process with the QR Authenticator. The authentication steps presented in Figure 13 are generic. The mLogin application could authenticate to the QR server by means of FIDO (described in another chapter), mutual TLS or other mechanisms.

7. The mLogin application sends the QR code to the QR authenticator server through an authenticated and secure channel.

8. The QR Authenticator verifies the correctness of the received QR code and expiration time. The requested attributes are send to the Service Provider.

9. The Service Provider verifies the received attributes and authorizes the user.

10. The Service Provider client-side component polls periodically the server in order to find the authorization status.

Protocol description

The Service Provider and the QR Authenticator exchange messages using REST (representational state transfer). The data media type will be JSON, the payload being encoded as JWT (JSON Web Token – RFC 7519). A JWT structure has the following components: header, payload and signature.

Each message will have a header which will contain the protocol version, the algorithm used to compute the signature and the token type (JWT). A header example is presented in the following structure:

```
{
  “version”: “1.0”,
  “alg”: “HS512”,
  “typ”: “JWT”
}
```

1. QR Code request.

This message is sent from the Service Provider to the QR Authenticator. The REST call will have the following form:

POST https://<qr_auth_address>/QRAuthenticator/QRCodeSession

The JWT will have the following payload:

{

“session_token”: “”,

“policy”: [],

“callback_url:”: “”

}

The response will contain the generated QR code. The session token is replied back in order to ensure the message freshness.

{

"session_token": "",

"qr_code": ""

}

2. Mobile application request. The QR Authenticator verifies if the requested resource already exists and if the expiration time is still valid. The message is not signed because it is assumed that the mobile is authenticated to the QR Authenticator server. In order to increase flexibility, the mobile application request could also be encoded using JWT, and the signature method could be RSA (FIDO).

The REST call will have the following form:

POST https://<qr_auth_address>/QRAuthenticator/QRCode

{

"qr_code" : "",

}

3. The message sent from the QR Authenticator to the Service Provider will contain a contain the requested user attributes and the session token.

The REST call will have the following form:

PUT https://<service_provider_callback_url>/Authorization/<session_token>

{

“session_token”: “”,

“attributes”: []

}

3.3.2 QR Authentication with Federated Identity

In the existing architecture the Service Provider must implement a QR software module. In order for the Service Provider to be QR implementation agnostic, the QR authentication protocol could be delegated to the QR Authenticator module. In addition a federated identity protocol, like OpenID, will exist between the QR module and the Service Provider.

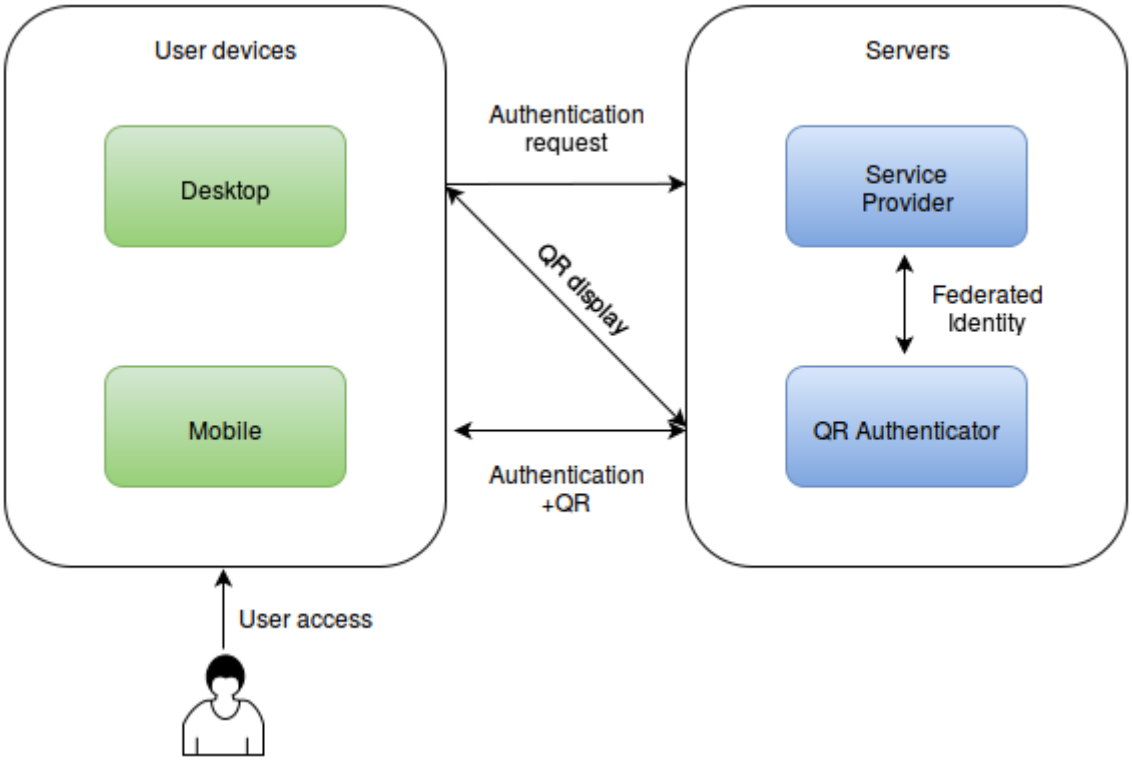

Figure 14 *QR Authentication with Federated Identity*

This approach will provide a modular architecture design, but some drawbacks will exist in some business case scenarios (WiFi access).

The proposed architecture is presented in Figure 14.

4 Trusted Execution Environment

Most modern devices come packed with a Rich Execution Environment (REE) which provide an extensive operating environment that applications can use. While these environments provide flexibility for developers and vendors, they also leave device vulnerable to various security threats. Moreover, many types of devices are constrained in space, cost or power dimensions that make the use of a discrete Trusted Platform Module chip (TPM) difficult. However advances in on-processor technologies, combined with the flexibility of the TPM protocols mean that it is possible to implement a TPM as an integrated solution or in firmware. The GlobalPlatform Trusted Execution Environment (TEE) [37] defines a standardized isolation environment for Systems on Chip (SoC) in which sensitive code, data and resources are processed away from the main operating environment, software and memory on the device. This isolation is enforced by hardware architecture and the boot sequence uses a hardware root of trust in the SoC package making it highly robust against software and probing attacks. In addition, code running in the TEE and using protected resources (known as Trusted Applications or Trustlets) is cryptographically verified prior to execution, leading to high integrity assurance. Because it provides an isolated runtime environment entirely inside the SoC (processor chip), the TEE enables advanced device or peripheral security use cases such as securing the user interface, or controlling access to an NFC chip. As such the TEE can be used as a distinct security coprocessor or to provide a trusted “bridge” between the user and other security technologies such as secured UI or OS user permissions on one side, and Secure Element access control on the other. The main operating system and rich applications then run as normal on the device, accessing the functionality of the Trusted Applications via a standardized “Client API”. Trusted Applications are written to an “Internal API” which ensures portable trustworthy access to secure resources, cryptographic operations and secure storage regardless of the underlying SoC hardware (Figure 15).

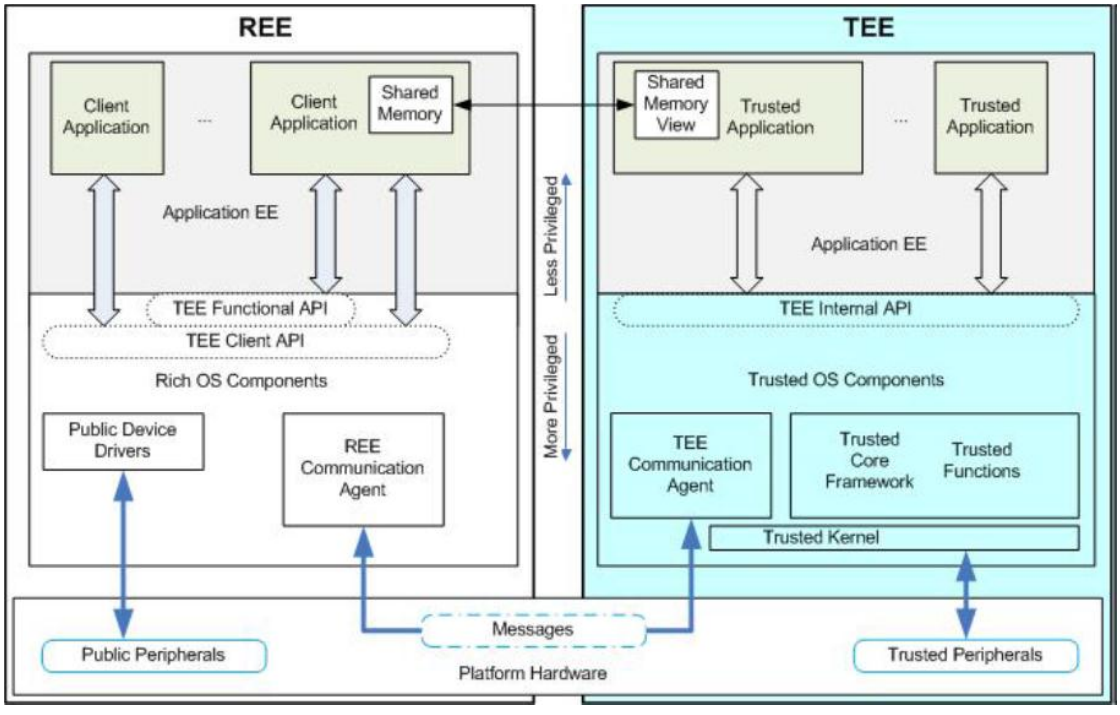

Figure 15 *TEE System Architecture*

The TEE is embedded in the device and runs alongside a standard OS or Rich Execution Environment. Figure 15 provides a high level view of the software components of a TEE-enabled device, independently of any hardware architecture. The TEE software architecture identifies two distinct classes of components:

- The Trusted Applications that run on the TEE and use the TEE Internal API
- The Trusted OS Components whose role is to provide communication facilities with the REE software and the system level functionality required by the Trusted Applications, accessible from the TEE Internal API

The REE software architecture identifies also two distinct classes of components:

- The Client Applications (CAs) which make use of the TEE Client API to access the secure services offered by Trusted Applications (TAs) running on the TEE
- The Rich OS, which provides the TEE Client API and sends requests to the TEE

The TEE software external interface comprises the TEE Internal API (used by the Trusted Applications) and the TEE Communication Agent protocol (used by the REE). The communication protocol between the REE and the TEE, used below the TEE Client API level, is implementation-dependent, and therefore this Protection Profile does not mandate any particular such protocol. The security targets conformant to this PP shall describe all software interfaces used for communication with the TEE from the REE.

At the highest level, in every TEE the following principles should apply:

- Code executed inside the TEE is trusted in terms of authenticity and integrity.
- The other assets are also protected in terms of confidentiality.
- Both assets and code are protected from unauthorized tracing and control through debug and test features.

4.1 TEE Overview

4.1.1 Hardware Architecture

The TEE is embedded in a device platform including:

- Hardware processing unit(s)
- Hardware resources such as
    - Physical volatile memory

  - Physical non-volatile memory
  - Peripherals, like keyboard and display
  - Cryptographic accelerators
  - Secure clock
  - Secure element
- A set of connections between the processing unit(s) and the hardware resources

Schematically, a TEE-enabled device is structured in four layers:

- The die layer, System-on-Chip (SoC), which contains processor(s) and resources such as memories, crypto-accelerators, peripherals (e.g. JTAG, USB, serial, HDMI), etc.
- The package layer, which embeds the SoC and contains further resources, e.g. non-volatile and volatile memories, pins or buses. Resources inside the same package layer are connected using buses that are not externally accessible. External buses in the specification are outside the package layer. “3D” die stacking techniques may be used to place more facilities inside the package that may not be in the die layer.
- The PCB layer, which contains SoC, package, non-volatile and volatile memories, wireless and contactless interface chips, security modules and other resources.
- The user layer, which contains user interfaces to the package, such as the touch screen or keyboard, and may contain other resources.

The TEE is typically implemented in the die and package layers of one package but it may be instantiated in a number of separate packages using cryptographic linking (secure channels) between TEE components. The TEE hardware external interface stands for the package input and output interfaces, which provide access to the package resources and indirectly to the SoC internals, both from the user layer and from the SoC itself. This PP considers the package internals as a black-box. Nevertheless, the physical boundary of the TEE is implementation-dependent. Furthermore, the set of “trusted” resources used to realize the security functionality, which is controlled by the TEE, can change dynamically. For instance, some communication resources such as the keyboard may sometimes be within the TEE boundaries if the TEE enforces exclusive access to these resources. From a logical point of view, the “trusted” resources used by the TEE are separated from the “un-trusted” resources used by the REE. That is, the TEE and the REE coexist in the device but isolated from each other (Figure 16).

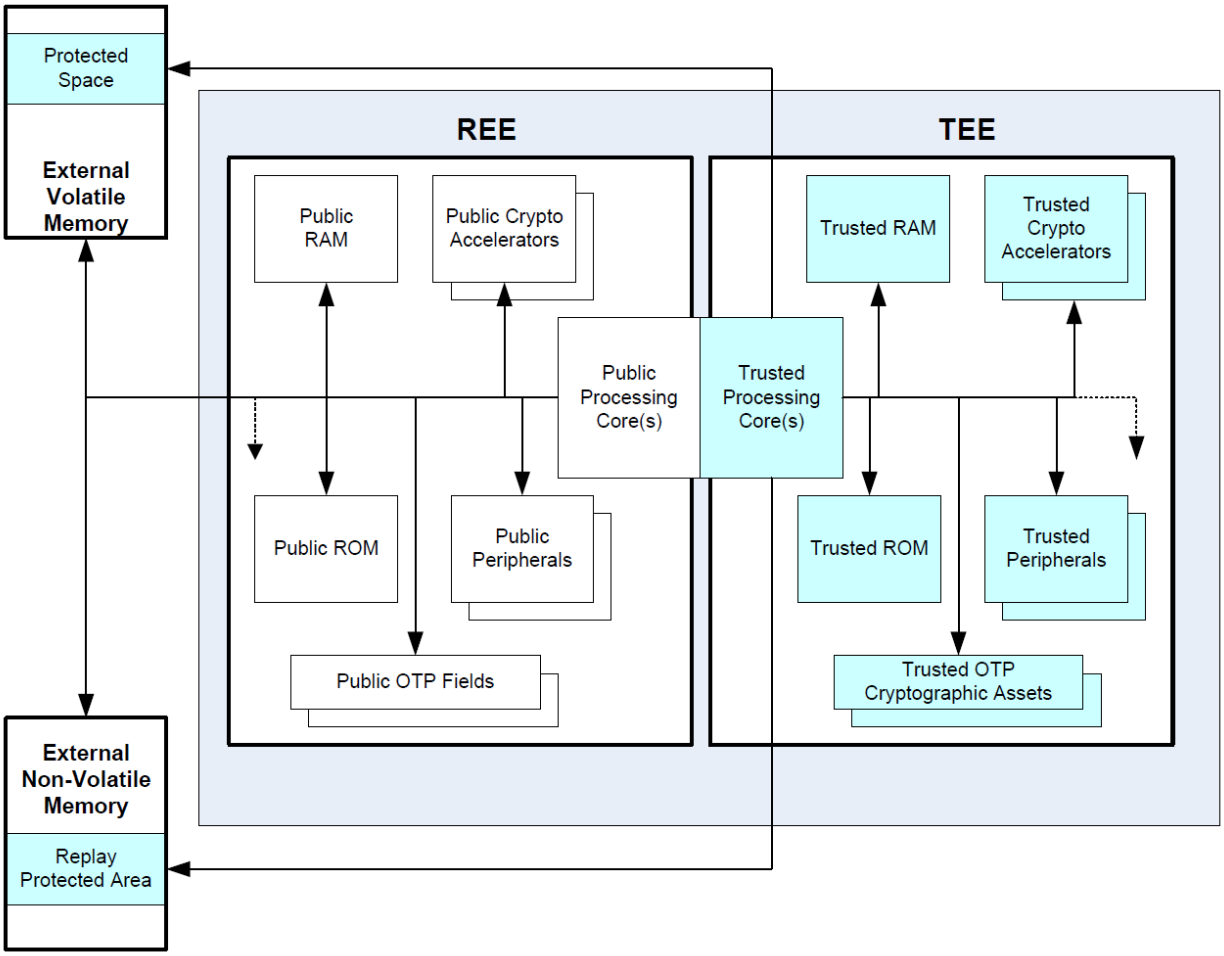

Figure 16 *Hardware Architectural View of REE and TEE*

4.1.2 TEE Security Functionality

The TEE security functionality consists of:

- TEE instantiation through a secure initialization process using assets bound to the SoC, that ensures the authenticity and contributes to the integrity of the TEE code running in the device
- Isolation of the TEE services, the TEE resources involved and all the Trusted Applications from the REE
- Isolation between Trusted Applications and isolation of the TEE from Trusted Applications
- Protected communication interface between CAs and TAs within the TEE, including communication endpoints in the TEE
- Trusted storage of TA and TEE data and keys, ensuring consistency (cf. section 1.4), confidentiality, atomicity and binding to the TEE

- Random Number Generator
- Cryptographic API including: o Generation and derivation of keys and key pairs
    - Support for cryptographic algorithms such as SHA-256, AES 128/256, T-DES, RSA 2048, etc. (this list is for example only, see the Application Note below)
- TA instantiation that ensures the authenticity and contributes to the integrity of the TA code
- Monotonic TA instance time
- Correct execution of TA services
- TEE firmware integrity verification
- Prevention of downgrade of TEE firmware.

4.1.3 TEE Internal API

The TEE Internal API defines a set of C APIs for the development of Trusted Applications (TAs) running inside a Trusted Execution Environment (TEE). A TEE is expected to meet the requirements defined in the GlobalPlatform TEE System Architecture specification, i.e., it is accessible from a Rich Execution Environment (REE) through the GlobalPlatform TEE Client API but is specifically protected against malicious attacks and runs only code trusted in integrity and authenticity. A TEE provides the Trusted Applications an execution environment with defined security boundaries, a set of security enabling capabilities, and means to communicate with Client Applications running in the Rich Execution Environment. A Trusted Application (TA) is a program that runs in a Trusted Execution Environment (TEE) and exposes security services to its Clients. A Trusted Application is command-oriented. Clients access a Trusted Application by opening a session with the Trusted Application and invoking commands within the session. When a Trusted Application receives a command, it parses the messages associated with the command, performs any required processing, and then sends a response back to the client. A Client typically runs in the Rich Execution Environment and communicates with a Trusted Application using the TEE Client API (see section 4.1.4). It is then called a “Client Application”. It is also possible for a Trusted Application to act as a client of another Trusted Application. The term “Client” covers both cases.

Each Trusted Application exposes an interface (the TA interface) composed of a set of entry point functions that the Trusted Core Framework implementation calls to inform the TA about life-cycle changes and to relay communication between Clients and the TA. Once the Trusted Core Framework has called one of the TA entry points, the TA can make use of the TEE Internal API to access the facilities of the Trusted OS, as illustrated in Figure 17. Each Trusted Application is identified by a Universally Unique Identifier (UUID) as specified in RFC 4122 [38]. Each Trusted Application also comes with a set of Trusted Application Configuration Properties. These properties are used to configure the Trusted OS facilities exposed to the Trusted Application. Properties can also be used by the Trusted Application itself as a means of configuration.

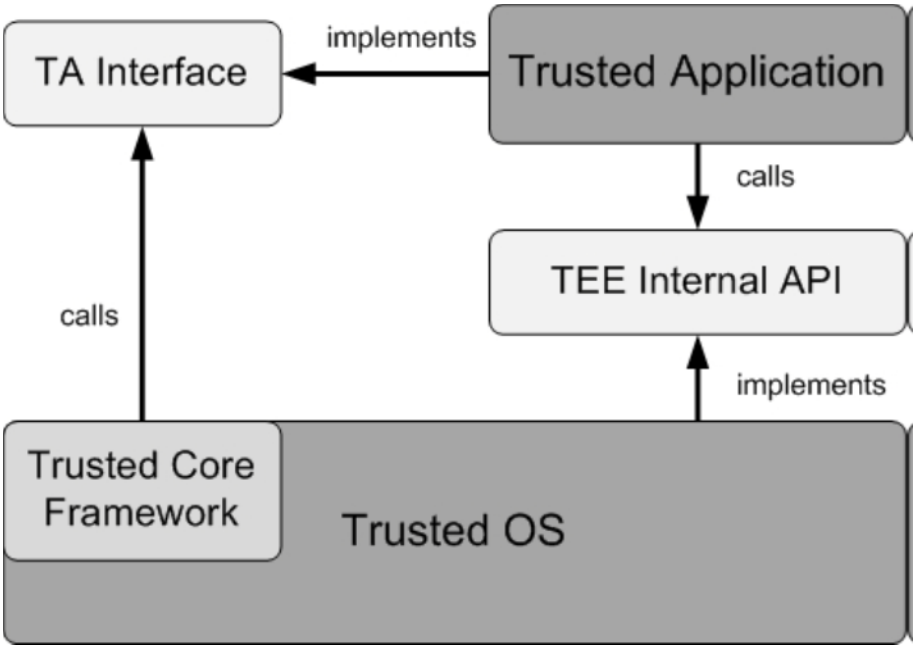

Figure 17 *TEE Internal API*

4.1.4 TEE Client API

The TEE Client API defines a communications API for connecting Client Applications running in a rich operating environment with security related Trusted Applications running inside a Trusted Execution Environment (TEE). For the purposes of this document a TEE is expected to be a trusted environment within the main device system-on-a-chip, which complements traditional security environments such as a UICC SIM card, although this is not a requirement of the API. A TEE provides an execution environment with security capabilities, which are either available to Trusted Applications running inside the TEE or exposed externally to Client Applications. A TEE may, for example, host a GPD/STIP runtime, but may also be based on other technologies such as a small operating system executing native code applications. Instead of trying to standardize a single monolithic API which covers a significant proportion of the interactions between a Client Application and the TEE-hosted functionality, the approach of the Global Platform standardization effort is modular. The TEE Client API covered by this specification concentrates on the interface to enable efficient communications between a Client Application and a Trusted Application running inside the TEE. The relationship between the major system components described in this specification are outlined in the block architecture below (Figure 18).

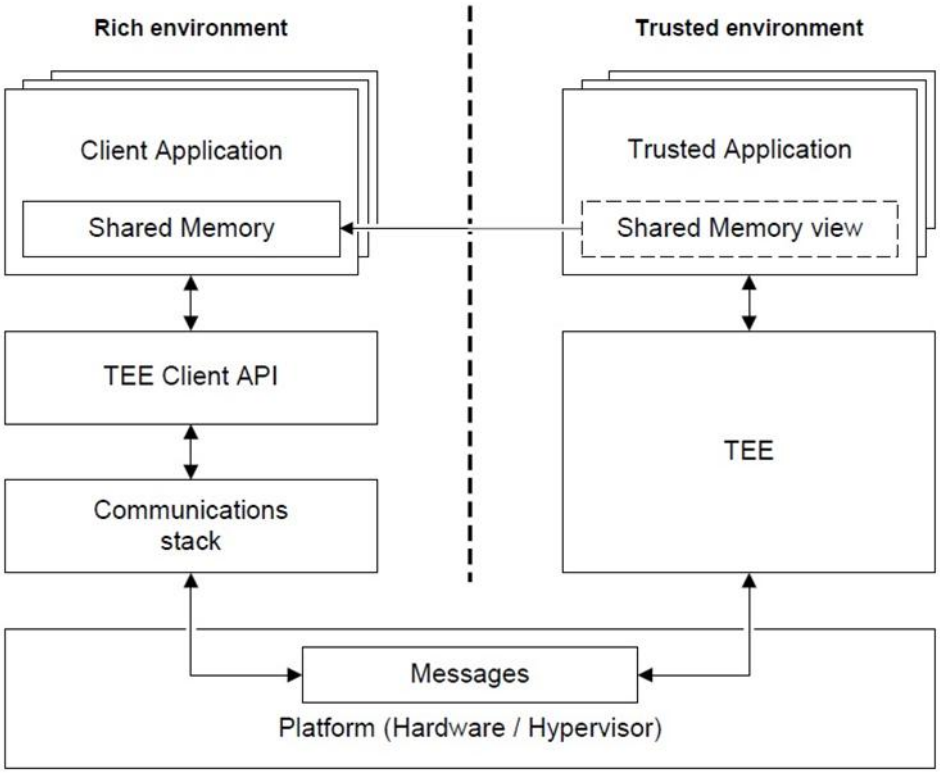

Figure 18 *TEE Client API*

When a Client creates a session with a Trusted Application, it connects to an Instance of that Trusted Application. A Trusted Application instance has physical memory space which is separated from the physical memory space of all other Trusted Application instances. The Trusted Application instance memory space holds the Trusted Application instance heap and writable global and static data. All code executed in a Trusted Application is said to be executed by Tasks. A Task keeps a record of its execution history (typically realized with a stack) and current execution state. This record is collectively called a Task context. A Task MUST be created each time the Trusted OS calls an entry point of the Trusted Application. Once the entry point has returned, an Implementation may recycle a Task to call another entry point but this MUST appear like a completely new Task was created to call the new entry point. A Session is used to logically connect multiple commands invoked in a Trusted Application. Each session has its own state, which typically contains the session context and the context(s) of the Task(s) executing the session. A Command is issued within the context of a session and contains a Command Identifier, which is a 32-bit integer, and four Operation Parameters, which can contain integer values or references to client-owned shared memory blocks. It is up to the Trusted Application to define the combinations of commands and their parameters that are valid to execute.

4.2 The ARM TrustZone

The TrustZone [39], [40], [41] platform is part of ARM's processor cores and system on chip (SoC) reference architecture. The associated hardware is part of the SoC silicon, and thus, it does not require any additional hardware. The primary objective of TrustZone is to establish a hardware-

enforced security environment providing code isolation, that is, a clear separation between trusted software, which is granted access to sensitive data like secret keys, and other parts of the embedded software. To achieve this, the TrustZone platform provides two virtual processing cores with different privileges and a strictly controlled communication interface, enabling the creation of two distinct execution environments, encapsulated by hardware. TrustZone technology is commonly used to run trusted boot and a trusted OS to create a Trusted Execution Environment (TEE). A platform with these characteristics can be used to build a wide ranging set of security solutions which are not cost-effective with traditional methods.

More specifically, the security of the system in ARM TrustZone is achieved by partitioning all of the SoC’s hardware and software resources so that they exist in one of two worlds - the Secure world for the security subsystem, and the Normal world for everything else. Hardware logic present in the TrustZone-enabled AMBA3 AXI bus fabric ensures that no Secure world resources can be accessed by the Normal world components, enabling a strong security perimeter to be built between the two. A design that places the sensitive resources in the Secure world, and implements robust software running on the secure processor cores, can protect almost any asset against many of the possible attacks, including those which are normally difficult to secure, such as passwords entered using a keyboard or touch-screen. The second aspect of the TrustZone hardware architecture is the extensions that have been implemented in some of the ARM processor cores. These additions enable a single physical processor core to safely and efficiently execute code from both the Normal world and the Secure world in a time-sliced fashion. This removes the need for a dedicated security processor core, which saves silicon area and power, and allows high performance security software to run alongside the Normal world operating environment. Figure 19 outlines the components of a typical ARM SoC.

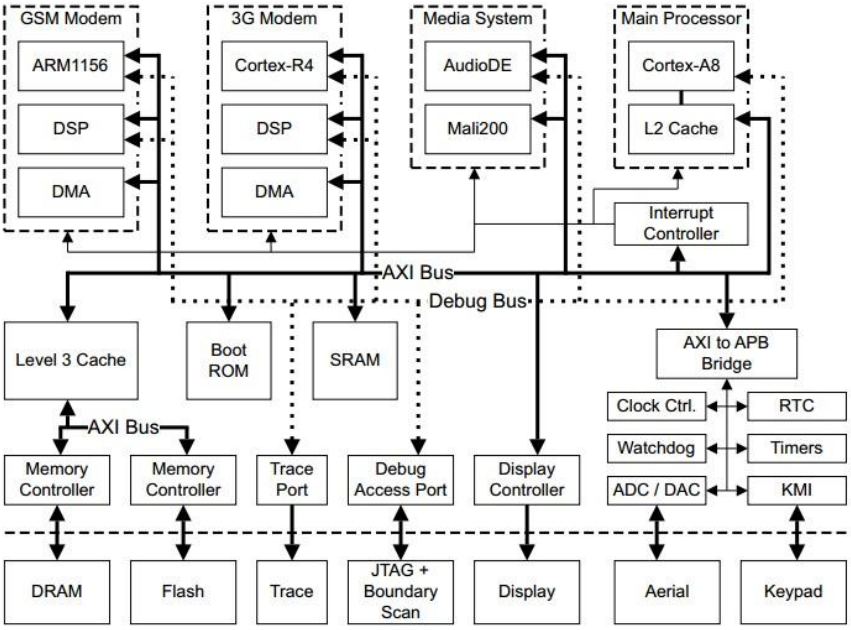

Figure 19 ***A simplified schematic of a typical ARM SoC design***

The most significant feature of the extended bus design in a TrustZone enabled SoC is the addition of an extra control signal for each of the read and write channels on the main system bus. These bits are known as the Non-Secure, or NS bits, and are defined in the public AMBA3 Advanced eXtensble Interface (AXI) bus protocol specification:

- AWPROT: Write transaction – low is Secure and high is Non-secure.
- ARPROT: Read transaction – low is Secure and high is Non-secure.

All bus masters set these signals when they make a new transaction, and the bus or slave decode logic must interpret them to ensure that the required security separation is not violated. All Non-secure masters must have their NS bits set high in the hardware, which makes it impossible for them to access Secure slaves. The address decode for the access will not match any Secure slave and the transaction will fail. If a Non-secure master attempts to access a Secure slave it is implementation defined whether the operation fails silently or generates an error. An error may be raised by the slave or the bus, depending on the hardware peripheral design and bus configuration, consequently a SLVERR (slave error) or a DECERR (decode error) may occur.

One of the most useful features of the TrustZone architecture is the ability to secure peripherals, such as interrupt controllers, timers, and user I/O devices. This enables the security environment to be extended so that it can solve some of the wider security issues which need more than just a secure data processing environment. A secure interrupt controller and timer allows a non-interruptible secure task to monitor the system, a secure clock source enables robust DRM, and a securable keyboard peripheral enables secure entry of a user password. The AMBA3 specification includes a low gate-count low-bandwidth peripheral bus known as the Advanced Peripheral Bus (APB), which is attached to the system bus using an AXI-to-APB bridge. The APB bus does not carry an equivalent of the NS bits. This ensures that existing AMBA2 APB peripherals are compatible with systems implementing TrustZone technology. The AXI-to-APB bridge hardware is responsible for managing the security of the APB peripherals; the bridge must reject transactions of inappropriate security setting and must not forward these requests to the peripherals.

In the TrustZone architecture, each of the physical processor cores provides two virtual cores, one considered Non-secure and the other Secure, and a mechanism to robustly context switch between them, known as monitor mode. The value of the NS bit sent on the main system bus is indirectly derived from the identity of the virtual core that performed the instruction or data access. This enables trivial integration of the virtual processors into the system security mechanism; the Non-secure virtual processor can only access Non-secure system resources, but the Secure virtual processor can see all resources. The two virtual processors execute in a time-sliced fashion, context switching through a new core mode called monitor mode when changing the currently running virtual processor. The mechanisms by which the physical processor can enter monitor mode from the Normal world are tightly controlled, and are all viewed as exceptions to the monitor mode software. The entry to monitor can be triggered by software executing a dedicated instruction, the Secure Monitor Call (SMC) instruction, or by a subset of the hardware exception mechanisms. The IRQ, FIQ, external Data Abort, and external Prefetch Abort exceptions can all be configured to cause the processor to switch into monitor mode. The software that executes within monitor mode is implementation defined, but it generally saves the state of the current world and restores the state

of the world being switched to. It then performs a return-from-exception to restart processing in the restored world. The world in which the processor is executing is indicated by the NS-bit in the Secure Configuration Register (SCR) in CP15, the system control coprocessor, unless the processor is in monitor mode. When in monitor mode, the processor is always executing in the Secure world regardless of the value of the SCR NS-bit, but operations on banked CP15 registers will access Normal world copies if the SCR NS-bit is set to 1.

The memory infrastructure outside of the core separates the system into two worlds, and a similar partitioning needs to be applied within the core to separate the data used and stored within the components of the level one (L1) memory system. The major component of the L1 memory system in an ARM applications profile processor is the Memory Management Unit (MMU), which is capable of mapping the virtual address space that is seen by the software running on the processor on to the physical address space that exists outside of the processor. The address translation is managed using a software-controlled translation table, which details which virtual address corresponds to each physical address, and some other attributes about the memory access, such as cacheability and access permissions. In an ARM core which has an MMU but not the Security Extensions, such as the ARM926EJ-S processor, there is a single mapping of virtual address to physical address at any instant in time. Privileged mode code typically rewrites, or points the hardware at a new set of, tables on process context switch to provide multiple independent virtual memory spaces. Within a TrustZone processor the hardware provides two virtual MMUs, one for each virtual processor. This enables each world to have a local set of translation tables, giving them independent control over their virtual address to physical address mappings. The ARMv6 and ARMv7 L1 translation table descriptor format includes an NS field which is used by the Secure virtual processor to determine the value of the NS-bit to use when accessing the physical memory locations associated with that table descriptor. The Non-secure virtual processor hardware ignores this field, and the memory access is always made with NS=1. This design enables the Secure virtual processor to access either Secure or Non-secure memory. To enable efficient context switching between worlds the ARM processor implementations may tag entries in the Translation Lookaside Buffers (TLBs), which cache the results of translation table walks, with the identity of the world that performed the walk. This allows Non-secure and Secure entries to co-exist in the TLBs, enabling faster world switching as there is then no need to flush TLB entries.

The ability to trap IRQ and FIQ directly to the monitor, without intervention of code in either world, allows for the creation of a flexible interrupt model for secure interrupt sources. Once the execution reaches the monitor, the trusted software can route the interrupt request accordingly. When combined with a security aware interrupt controller this allows a design to provide secure interrupt sources which cannot be manipulated by the Normal world software. The model recommended by ARM is the use of IRQ as a Normal world interrupt source, and FIQ as the Secure world source. IRQ is the most common interrupt source in use in most operating environments, so the use of FIQ as the secure interrupt should mean the fewest modifications to existing software. If the processor is running the correct virtual core when an interrupt occurs there is no switch to the monitor and the interrupt is handled locally in the current world. If the core is in the other world when an interrupt occurs the hardware traps to the monitor, the monitor software causes a context switch and jumps to the restored world, at which point the interrupt is taken. To prevent malicious Normal world software masking sensitive Secure world interrupts the processor hardware includes a configuration

register in CP15 which can be used to prevent any Normal world software modifying the F (FIQ mask) and A (external abort mask) bits in the CPSR. This control register can only be accessed by Secure world software. Note that there is no option to prevent the Normal world masking IRQ interrupts.

Typical use cases in mobile platforms include the protection of authentication mechanisms, cryptography, key material and DRM. The segregation of execution environments protects against runtime attacks and provides high flexibility to the platform, since additional security services may be added or updated in the future. As part of the ARM architecture, the silicon is designed focusing on mobile and embedded systems (i.e., limited computational and battery consumption). Finally, it incorporates secure storage and a cryptographic accelerator, which provides hardware acceleration for a wider set of crypto algorithms than TPMs, including SHA224/SHA256, AES, and elliptic curve cryptography, a viable alternative to RSA, well suited for resource restricted systems.

There are a wide variety of possible software architectures for the secure world, and the implementation of these is almost totally dependent on the application the user is targeting. A simple implementation may be focused on protecting a single asset which was implemented in a protected factory environment, such as authentication of a single secret. This requires a simple solution comprising of a lightweight secure kernel and integration of the monitor for switching between normal and secure domains. Alternatively the system architect may be looking for a robust and long term solution where trusted applications are appended over the lifetime of the device, for example in a mobile handset or tablet. In this case the designer may opt for a Trusted Execution Environment which mimics many of the dynamic functions of a traditional OS, including sandboxing of applications, while remaining small enough to be certified. Due to the inherent complexity of implementing a full Secure OS, and the potential need to certify its capabilities and performance, ARM recommends investigating commercial TEE solutions from Trusted OS suppliers that are members of GlobalPlatform [42].

4.3 Android Security and ARM TrustZone

Latest Android version (Android M) provide security enhancements if ARM TrustZone (or Secure Element) hardware is present. Many of these enhancements are utilized automatically by Android while others are available to Android Developers through the Android API. The ARM TrustZone features that the Android provides are:

1. Android Key Store
2. User to Device Authentication (GateKeeper and Fingerprint Hardware Abstraction Layer (HAL))
3. Verified Boot

4.3.1 The Android Keystore

The Android Keystore [43] system lets you store cryptographic keys in a container to make it more difficult to extract from the device. Since Android 6.0 (i.e. Android M), due to the high availability of devices that implement TEEs in a system on a chip (SoC) Android Keystore has been significantly enhanced in order to support access-controlled, hardware-backed keys. All the key-related operations are handled by the Keymaster Hardware Abstraction Layer (HAL), an OEM-provided, dynamically-loadable library used by the Keystore service to provide hardware-backed cryptographic services. HAL implementations must not perform any sensitive operations in user space, or even in kernel space. Sensitive operations are delegated to a secure processor reached through some kernel interface (Figure 20).

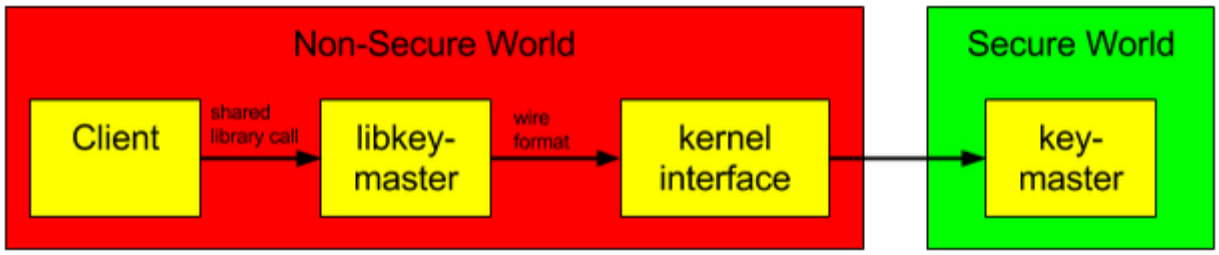

Figure 20 *Access to Keymaster HAL*

Once keys are in the Keystore, they can be used for cryptographic operations with the key material remaining non-exportable. Moreover, it offers facilities to restrict when and how keys can be used, such as requiring user authentication for key use or restricting keys to be used only in certain cryptographic modes. The Android Keystore system provides two security features regarding cryptographic keys generation, storage and use. These features are 1) Extraction Prevention and 2) Key Use Authorizations.

Extraction Prevention is achieved by ensuring that Key material never enters the application process. When an application performs cryptographic operations using an Android Keystore key, behind the scenes plaintext, ciphertext, and messages to be signed or verified are fed to a system process which carries out the cryptographic operations. In fact, in cases where ARM TrustZone is present, key material is never exposed outside of secure hardware if particular combination of key algorithm, block modes, padding schemes, and digests with which the key is authorized to be used are supported by the devices secure hardware.

Key Use Authorizations lets apps specify authorized uses of their keys when generating or importing them to mitigate unauthorized use of keys. These authorizations are impossible to change once a key is generated or imported and are enforced by Android whenever the key is used. Authorizations are only relevant if the application’s requirements include that application process compromise cannot lead to unauthorized use of the keys. These authorizations include:

1. ***Cryptography***: authorized key algorithm, operations or purposes (encrypt, decrypt, sign, verify), padding schemes, block modes, digests with which the key can be used;

2. ***Temporal validity interval***: interval of time during which the key is authorized for use;
3. ***User authentication***: the key can only be used if the user has been authenticated recently enough.

Keys whose key material lays into secure hardware, such as TEE or SE, key authorizations such as cryptographic and user authentication authorizations may be enforced by secure hardware, depending on the device. On the other hand temporal validity authorizations are highly unlikely to be enforced by secure hardware as they would require a secure and independent real-time clock, which in most cases isn’t present.

According to Android Open Source Project all hardware-backed keystore implementations must provide the following:

- RSA
    - 2048, 3072 and 4096-bit key support are required
    - Support for public exponent F4 (2^16+1)
    - Required padding modes for RSA signing are:
        - No padding (deprecated, will be removed in the future)
        - RSASSA-PSS (KM_PAD_RSA_PSS)
        - RSASSA-PKCS1-v1_5 (KM_PAD_RSA_PKCS1_1_5_SIGN)
    - Required digest modes for RSA signing are:
        - No digest (deprecated, will be removed in the future)
        - SHA-256
    - Required padding modes for RSA encryption/decryption are:
        - Unpadded
        - RSAES-OAEP (KM_PAD_RSA_OAEP)
        - RSAES-PKCS1-v1_5 (KM_PAD_RSA_PKCS1_1_5_ENCRYPT)
- ECDSA
    - 224, 256, 384 and 521-bit key support are required, using the NIST P-224, P-256, P-384 and P-521 curves, respectively
    - Required digest modes for ECDSA are:
        - No digest (deprecated, will be removed in the future)
        - SHA-256

- AES
    - 128 and 256-bit keys are required
    - CBC, CTR, ECB and and GCM. The GCM implementation must not allow the use of tags smaller than 96 bits or nonce lengths other than 96 bits.
    - Padding modes KM_PAD_NONE and KM_PAD_PKCS7 must be supported for CBC and ECB modes. With no padding, CBC or ECB mode encryption must fail if the input isn't a multiple of the block size.
- HMAC SHA-256, with any key size up to at least 32 bytes.

SHA1 and the other members of the SHA2 family (SHA-224, SHA384 and SHA512) are strongly recommended, but not required. Keystore do provide them in software if the hardware Keymaster implementation doesn't provide them.

Some cryptographic primitives that are strongly recommended but nor required are:

- Smaller key sizes for RSA
- Arbitrary public exponents for RSA

4.3.2 User to Device Authentication

In Android user can authenticate to the device (i.e. unlock the device) with one of the following ways: 1) Fingerprint, 2) Pattern, 3) Pin or 4) Password. The first is handled by Android’s Fingerprint Hardware abstraction Layer (HAL) while the other three are handled by the Gatekeeper HAL.

In an Android device with a fingerprint sensor a user is able to enroll one or more fingerprints and use them to unlock his device. Android uses the Fingerprint Hardware Abstraction Layer (HAL) [44] to connect to a vendor-specific library and fingerprint hardware. The Fingerprint HAL interacts with the following components (Figure 21):

- **Fingerprint API**: Interacts directly with the app process as each app has an instance of Fingerprint Manager which is a wrapper that communicates with FingerprintService.
- **FingerprintService**: A singleton service that operates in the system process and handles communication with **fingerprintd**.
- **fingerprintd** (Fingerprint daemon): A daemon that operates in its own process and wraps the Fingerprint HAL vendor-specific library.
- **Fingerprint HAL vendor-specific library**: A hardware vendor's implementation of the Fingerprint HAL. The vendor-specific library communicates with the device-specific hardware.
- **Keystore API and Keymaster**: These components provide hardware-backed cryptography for secure key storage in a Trusted Execution Environment (TEE).

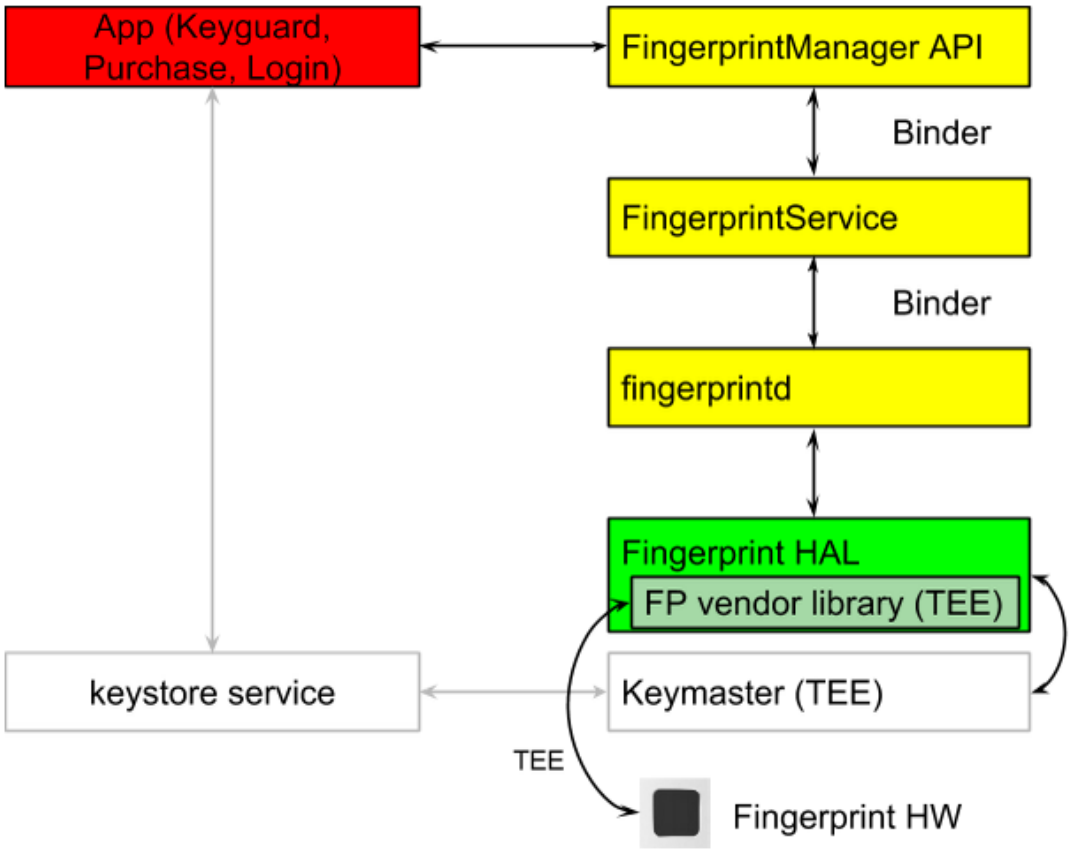

Figure 21 *High-level data flow for fingerprint authentication*

According to Android guidelines regarding Fingerprint HAL vendor-specific implementations, the following should be ensured:

1. Raw fingerprint data or derivatives (e.g. templates) must never be accessible from outside the sensor driver or Trusted Execution Environment (TEE). Hardware access must be limited to the TEE, if the hardware supports it, and must be protected by a SELinux policy. That is, the Serial Peripheral Interface (SPI) channel must be accessible only to the TEE, and there must be an explicit SELinux policy on all device files.

2. Fingerprint acquisition, enrollment and recognition must occur inside the TEE.

3. Only the encrypted form of the fingerprint data can be stored on the file system, even if the file system itself is encrypted.

4. Fingerprint templates must be signed with a private, device-specific key, for example with AES, with at least the absolute file-system path, group and finger ID such that template files are inoperable on another device or for anyone other than the user that enrolled them on the same device. For example, copying the fingerprint data from a different user on the same device, or from another device, must not work.

5. Implementations must either use the file system path provided by the set_active_group() function or provide a way to erase all user template data when the user is removed. It is

strongly recommended that fingerprint template files be stored as encrypted in the path provided. If this is infeasible due to TEE storage requirements, then the implementer must add hooks to ensure removal of the data when the user is removed.

As we previously mentioned other than Fingerprint authentication mechanisms in Android include password, pin or pattern and are handled by the GateKeeper HAL [45]. The Gatekeeper subsystem performs device pattern/password authentication in a Trusted Execution Environment (TEE) if present. Gatekeeper enrolls and verifies passwords via an HMAC with a hardware-backed secret key. Additionally, Gatekeeper throttles consecutive failed verification attempts and must refuse to service requests based on a given timeout and a given number of consecutive failed attempts. When users verify their passwords, Gatekeeper uses the TEE-derived shared secret to sign an authentication attestation to send to the hardware-backed Keystore. That is, a Gatekeeper attestation notifies Keystore that authentication-bound keys (for example, keys that apps have created) can be released for use by apps.

Gatekeeper involves three main components (Figure 22):

- **gatekeeperd (Gatekeeper daemon)**. A daemon containing platform-independent logic and corresponding to the GateKeeperService Java interface.
- **Gatekeeper Hardware Abstraction Layer (HAL)**. The HAL interface and the implementing module.
- **Gatekeeper (TEE)**. The TEE counterpart of gatekeeperd. A TEE-based implementation of Gatekeeper.

As shown in Figure 22, the LockSettingsService makes a request that reaches the gatekeeperd daemon in the Android OS. The gatekeeperd daemon makes a request that reaches its counterpart (Gatekeeper) in the TEE.

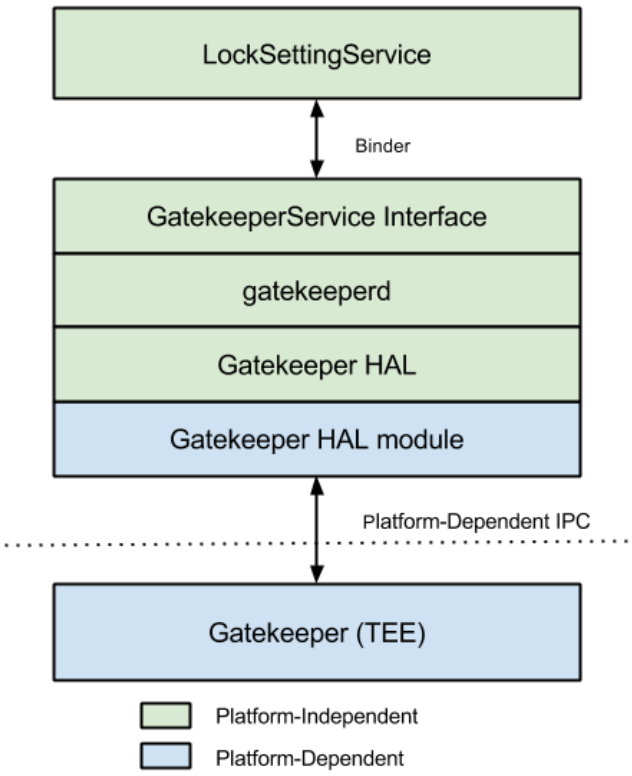

Figure 22 *High-level data flow for authentication by GateKeeper*

4.3.3 Verified Boot

Verified boot [46] guarantees the integrity of the device software starting from a hardware root of trust up to the system partition. During boot, each stage verifies the integrity and authenticity of the next stage before executing it. This capability can be used to warn users of unexpected changes to the software when they acquire a used device, for example. It will also provide an additional signal of device integrity for remote attestation, and together with encryption and Trusted Execution Environment (TEE) root of trust binding, adds another layer of protection for user data against malicious system software. If verification fails at any stage, the user is visibly notified and is given an option to continue using the device at their own discretion. This experimental feature helps Android users be sure when booting a device it is in the same state as when it was last used.

Malware with root privileges could hide from detection programs due to the fact that is more privileged than detection programs. Android’s dm-verity feature, which realizes the boot verification, looks at a block device, the underlying storage layer of the file system, and determine if it matches its expected configuration. It does this using a cryptographic hash tree. For every block (typically 4k), there is a SHA256 hash. Hash values are stored in a tree of pages (Figure 23). The “root” hash value must be trusted to verify the rest of the tree. Modifying any of the blocks would break cryptographic hash and thus the verification would fail.

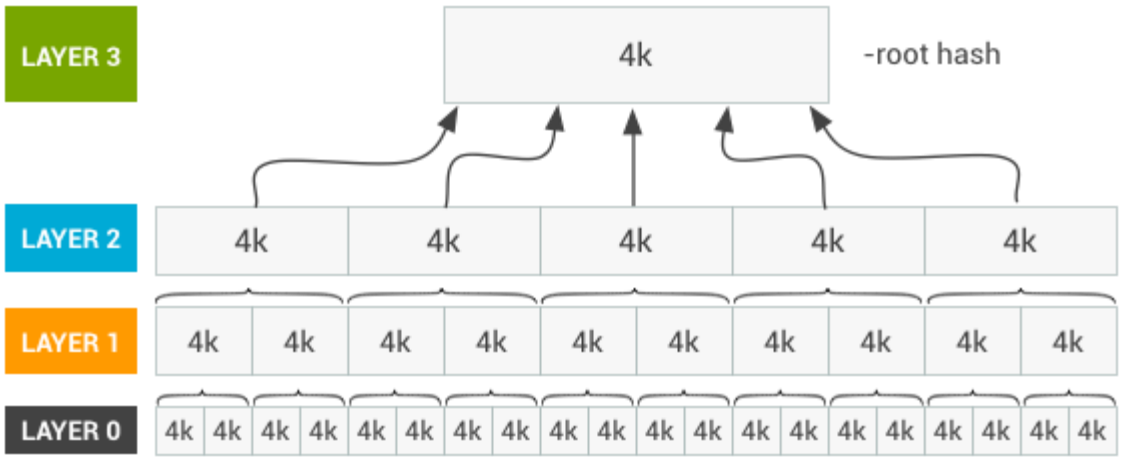

Figure 23 *dm-verity hash table*

4.4 ReCRED App and TEE module usage

The ReCRED application will utilize all TEE features provided by latest Android (Android 6.0 Marshmallow) to enhance the security of the project. For User-to-Device authentication ReCRED app will use Android’s phone unlock mechanism in order to ensure that both enrollment of credentials and authentication are executed in TEE and that fingerprint templates, patterns, pins, and passwords cannot be exported from the device (as described in 4.3.2). This design decision will allow us to use only the following (supported by Android) authentication methods:

- Pin
- Pattern
- Password
- Fingerprint OR Backup Password

For Device-to-Service authentication the ReCRED application will generate and store the appropriate for each service and authentication method keys using Android Keystore. As described in 5.3.1 this will ensure that keys are generated inside the TEE and stored in a secure fashion and thus are non-exportable from the device. Furthermore, while generating these keys appropriate key use authorizations will be applied (as described in 4.3.1). Other than cryptography and temporal validity authorizations, keys would be bound to user authentication thus user authentication will be triggered anytime a key is about to be used.

5 Privacy and Security Considerations

5.1 Biometric Identity

Despite the many advantages of biometrics over other traditional authentication methods (e.g. passwords and tokens), there are certain privacy and security considerations that also need to be addressed.

First of all, although more difficult, it is not impossible for biometric data to be compromised. If a password or token is stolen, we can change or replace it. But we can’t change our fingerprints or irides, therefore if they are compromised they are compromised forever. The problem is much bigger when the actual biometric templates are stored in the cloud or in a central authentication server, since if such a system is hacked thousands or even millions of templates will be stolen. For those reasons, ReCRED will be storing only an encrypted, binary representation of biometric templates (fingerprints). In addition, ReCRED will be using a device-centric topology, which means that all the processing and storing of biometric data happens inside the user device. Even if the actual features are compromised (e.g. during extraction), at least this will affect only the specific user and won’t constitute a major data breach. Fingerprints have an added advantage, since they are not totally irrevocable, in the sense that even if a fingerprint is compromised some other finger can be used.

Another major issue is that certain biometrics can and will be forged. However, different biometric characteristics require different levels of effort and expertise in order to be spoofed. While it could be quite easy to fool a face detection system by using just a photograph of the “victim”, fooling a fingerprint or a retinal scanner would require much more effort and expertise (e.g. use of advanced methods in order to replicate the finger or the retina). For that reason, ReCRED will be using fingerprints as a verification characteristic, instead of other features with high circumvention. Moreover, as the biometric scanners evolve, they are becoming more and more capable of detecting spoofing attempts.

In addition, there are biometric characteristics which can disclose private data and/or medical disorders, therefore lowering their acceptability among the public. For example, an iris can reveal genetic sex, fingerprints are known to be associated with certain disorders, such as Down’s and Turner’s syndrome, and DNA can disclose all kinds of private data. ReCRED will be storing only the binary representation of fingerprint patterns, so even if a template is compromised, the hacker won’t have access to the actual features and no private info will be derived.

Finally, the acceptability of biometrics is very much related to the degree to which certain biometric features can be easily acquired without the knowledge of their owners. For example, a photo of our face could be easily captured from a distance, our device could scan our irides without us knowing and pieces of our DNA can be found everywhere. Again, ReCRED will make use of fingerprint recognition, which requires of the user to actively press his thumb over a scanner, making it one of the most acceptable biometric methods.

5.2 Behavioral Authentication as a Second Factor

Regarding the B-Verifier application, the following issues should be considered.

The text that user types when capturing the typing signature should not be recorder/stored in the device or in the Authentication authority. Only typing features will be extracted from that text and these features will be processed/ stored.

The reference signature, that is, the created profile for the user, will be not stored in the device but only in the Behavioral Authentication Authority.

On the other hand, the latest typing signature of the user has to be stored in the user device, in order to be available for the second factor authentication and for that reason it should be encrypted using the TPM.

The transfer of the typing signature from the device to the Behavioral Authentication authority should be also encrypted.

Another security issue that has to be considered is that the Behavioral Authentication Authority stores all the biometric (behavioral) information in its database. This implies the risk for the database to be hacked or damaged. The database should be encrypted and backed-up securely to avoid such issues.

Finally, the user has to accept an agreement for using the ReCRED platform as a secure service and thus will be notified about the use of their data.

5.3 FIDO and Federated Authentication

In order to achieve a sound security level, the following recommendations regarding algorithms and key lengths should be considered when implementing FIDO:

- ServerChallenge is defined as a byte array that contains random data. This data is generated by the FIDO UAF Server, sent to the Client and further included in a response message to protect from Reply Attacks.

The length of such an array is defined to be at least 8 bytes and no more than 64 bytes. 8 bytes should ensure a minimum security level and 64 bytes should ensure a SHA-512 compatibility

- Authenticators are recommended to use more sources of random seeds to increase the quality of random numbers generated (RFC4086)

- FIDO UAF Protocol relies on TLS to ensure messages privacy. Protocol messages requires a channel binding structure that binds security elements from the communication channel into protocol messages. The ChannelBinding structure is added by the client and verified by the server to protect from MITM attacks. The following bindings are supported (RFC5929):

- TLS channel ID
- Server End Point
- TLS server certificate
- TLS Unique

- The supported key type used in ChannelID is EC (elliptic curve) and more specifically, the P-256 EC key type as specified by NIST
- The server endpoint of TLS connection must be at the Relying Party and the client endpoint must be the FIDO UAF Client or the User Agent
- The used TLS protocol version must be v1.2 and only in case it is not available, v1.1 should be used. Insecure algorithms in TLS protocol should also be avoided (e.g. MD5, RC4, SHA1) – NIST recommendation (SP800-131A)
- A usage of SHA-1 in digital signatures generation context becomes deprecated in NIST standards (SP800-131A) and ENISA recommendations (Algorithms, Key Sizes and Parameters Report, 2013) the minimum recommended hash function is SHA-256.

5.4 QR Login

In order to ensure the QR Authentication security several prerequisites must be fulfilled. To mitigate MITM attacks and eavesdropping all the HTTP messages (exchanged between the Service Provider and the QR Authenticator) must be encrypted using a TLS secure channel (the used TLS protocol version must be v1.2 and only in case it is not available, v1.1 should be used). Regarding the server messages authenticity, the JWT (JSON Web Token) exchanged between the QR Authenticator and the Service Provider must be signed using a HMAC algorithm (the minimum recommended hash function is SHA-256). It is also assumed that the QR Authenticator and the Service Provider are sharing a secret key for HMAC computation and validation.

In order to mitigate web session hijacking attacks, the Service Provider session token will be a nonce, unrelared with the web session ID: both the session token and the QR code will be random byte strings with a minimum length of 32 bytes.

The message sent from the mobile application to the QR Authenticator must be transmitted over a secure and authenticated channel. It is assumed that the mobile application is already authenticated to the QR Authenticator module (using FIDO, two-way TLS or other mechanism) before scanning the QR code. Taking this into consideration, the QR code send from the mobile application to the QR Authenticator will be signed only if the authentication protocol requires it (for instance if using two-way TLS it is not necessary to sign the message).

5.5 TEE Privacy & Security Considerations

5.5.1 Android-related considerations

5.5.1.1 Verified Boot

Verified boot is not a mandatory operation in Android OS and its use is left entire to the device vendors. Moreover there are cases where device vendors do not verify the Android’s bootloader (aboot). This is called an unlocked device state and it is often preferred by vendors to allow users load custom Android OSs. Even if the Verified Boot is enabled by the device vendor but not in enforcing mode in case of verification failure the user is getting a warning on screen but the boot process is not stopped. It is on user’s discretion to whether or not he will use the mobile phone and his applications despite the warnings. As currently there is no API for an application to determine at runtime if the verification process has failed or not a ReCRED perquisite should be that the mobile phone has a verification boot process running in enforcing mode and with locked device state so that the user cannot use his device in case of validation failure.

5.5.1.2 Android Keystore

If the Android OS is compromised or an attacker can read the device's internal storage, the attacker may be able to use any app's Android Keystore keys on the Android device, but not extract them from the device. As access control to those keys is provided by the Google Keystore daemon which holds references to the keys stored in the TEE, it would be possible on a rooted phone to modify the permissions database so that an application with a random UID can have data signed with any key stored in the TEE. In order to minimize the risk of such attacks the keys should always be bound to specific key use authorizations, especially user authentication authorizations. If the user disables user authentication, or turns device unlock into “swipe” unlock which equals to no authentication, the keys bound to user authentication will become unusable until user authentication is re-enabled. Moreover the benefits of a rooted device detection mechanism that would trigger certain warning and/or block services should be examined.

It also be noted that key material is protected by the TEE only if it supports the particular combination of key algorithm, block modes, padding schemes, and digests with which the key is authorized to be used. The supported cryptographic primitives can be found on 5.4.1. However, to check whether the feature is enabled for a key, android provides the isInsideSecurityHardware function that should in all cases to ensure the security of the key.

5.5.2 The case of Samsung KNOX Platform

Samsung KNOX [47] is Samsung’s secure Android platform that addresses some of the fundamental security issues regarding Android by leveraging hardware security capabilities to offer multiple levels of protection to both operating systems and applications. Some key features include the KNOX container, Trusted Boot and TrustZone-based Integrity Measurement Architecture. In the following paragraphs some KNOX features will be further analyzed.

Samsung KNOX provides solutions to address the security needs of individual applications with application Containers. A Samsung KNOX Container (Figure 24) provides a separate Android environment within the mobile device, completed with its own home screen, launcher, applications, and widgets. Applications and data inside the container are isolated from applications outside the container, that is, applications outside the container cannot use Android inter-process communication (IPC) or data-sharing methods with applications inside the container. Likewise, applications inside the container generally do not have the ability to interact with applications or access data outside the container. However, some applications inside the container can be granted read-only access to data outside the container via a policy configuration. For example, photos taken from the camera inside the container won’t be viewable from the Gallery outside the container in a user’s personal area. Conversely, contacts and calendar events created outside the container are viewable inside the container.

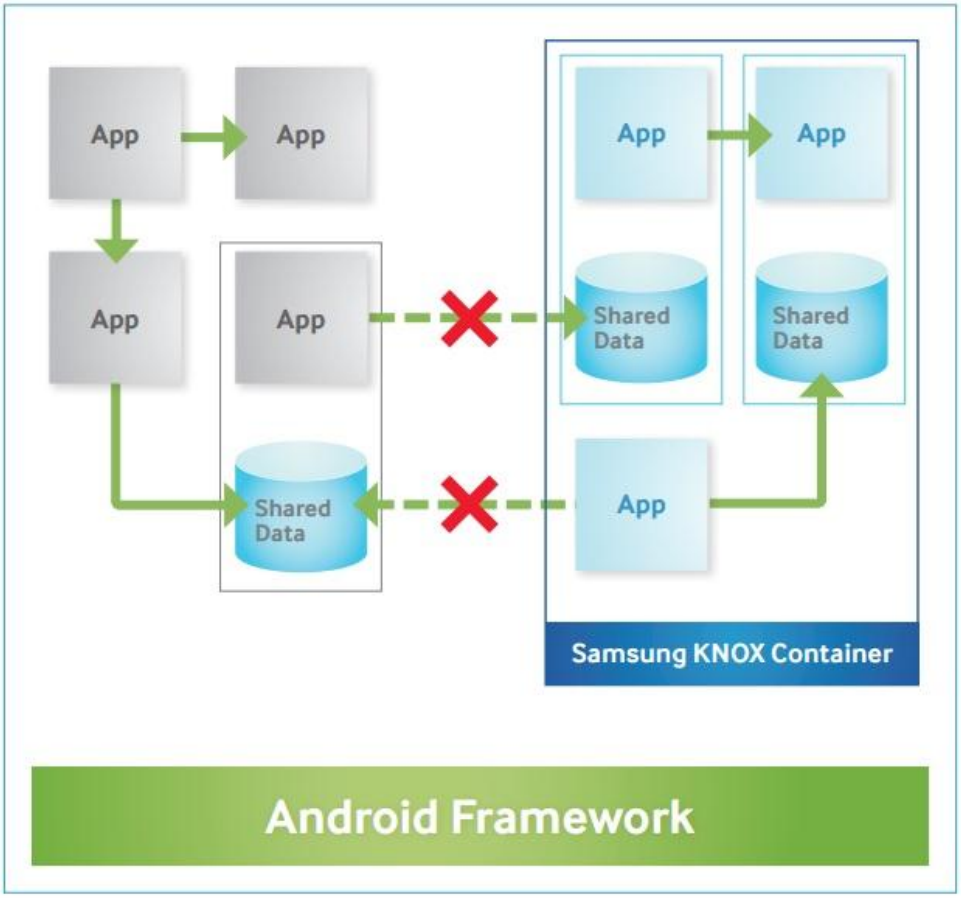

Figure 24 *Application isolation in KNOX*

Typically Android’s boot verification process is only performed until Android’s bootloader (known as aboot) is loaded, which itself does not verify the Android operation system. This practically means that users are able to install customized OS kernels and thus customized Android operating systems. As a result there is no guarantee that the custom Android is enforcing OS-level security protection. Samsung Knox leverages the verified boot functionality into a mechanism named Trusted Boot.

Trusted boot records measurements of each bootloader in secure memory during the boot up process. At runtime Trusted Apps use these measurements to make security critical decisions such as access to security keys etc. Additionally is Android kernel verification fails a one-time programmable memory area is permanently written to indicate the suspected tampering. This is used to prevent creation of KNOX containers and protect already created container and its data.

Several Android security features rely on the assumption of OS kernel integrity. If the kernel itself is compromised, security mechanisms could potentially be disabled. Samsung’s TrustZone-based Integrity Measurement Architecture (TIMA) was developed to ensure the integrity of the kernel and thus the functionality of security mechanisms through 3 different mechanisms. 1) **TIMA Periodic Kernel Monitoring** ensures that legitimate kernel code and data have not been modified by malicious software. It also monitors Android’s security related data structures in OS kernel memory to prevent malicious attacks from corrupting them and thus disabling Android’s security mechanisms. 2) **TIMA Real-time Kernel Protection** performs periodic real-time monitoring of the system from within the TrustZone to prevent tampering of kernel at runtime (i.e. malicious modification and injections of kernel code). 3) **Remote Attestation** offers verification of mobile device’s core system software (i.e. the bootloaders and the kernel) at runtime, based on the measurement data collected during trusted boot. Attestation can be requested at any time by an enterprise’s Mobile Device Management (MDM) system. All security critical operations of attestation are performed in TrustZone.

Adoption of the KNOX Platform by ReCRED has been rejected as it has several serious drawbacks. The implementation details of the various security mechanisms that the platform uses are not public. These implementation may contain security flaws that are not known yet. Furthermore, as this platform is targeting only Samsung apps it will reduce ReCRED’s applicability. Last but not least, relying on vendor-specific implementation is out of the scope of ReCRED project.

6 Conclusions

“Description of DCA protocols and technology support” is the first deliverable form WP 3. It extends the protocols related to DCA that have been described in the Reference architecture deliverable.

It proposed the usage of biometric identification for the user to device access and several different authentication protocols for the device to service authentication:

- Behavioral authentication that uses human intrinsic behaviors and attributes such as their voice or overall way of speaking, their signature or overall way of writing, the way the type or their keystroke dynamics, and finally their walking style or gait recognition. The document presents the B-Verifier application that analyses the keystroke dynamics, a computer user's habitual typing pattern when interacting with a computer keyboard.
- An innovative extension of FIDO to create a Federated Device-centric Attribute-based Authentication scheme that combines Federated FIDO and ABAC FIDO.
- QR Login which is an authentication bridging between desktop and a mobile phone that contains user credential by taking advantage of the device’s camera.

TEE is used within the DCA as it provides implementation of cryptographic operations in the trusted execution environment of the mobile device so that the cryptographic keys and attributes involved in the authentication remain as protected.

The privacy and security section is considering the threat model described in the reference architecture together with the specific threats and limitations of the DCA components as well as mitigation techniques.

The first small pilot setup, the “Campus-wide Wi-Fi and web services access control”, will benefit from the outcome of this deliverable and will provide the test bed of the results. Based on this and of the corresponding deliverables from WP4 – Identity de-fragmentation and real-world binding and WP5 – Attribute-Based Access Control the DCA protocols shall be further fine-tuned and adapted in order to support real life implementations.

7 References

[1] OpenID. “The Internet Identity Layer.” http://openid.net/
[2] IBM Identity Mixer, Idemix, http://www.zurich.ibm.com/security/idemix/
[3] OAuth. “Secure authorization in a simple and standard method”. http://oauth.net
[4] Gilbert, Peter, et al. "Youprove: authenticity and fidelity in mobile sensing." Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems. ACM, 2011.
[5] G. A. v. Graevenitz, "Biometric Authentication in Relation to Payment Systems and ATMs," Datenschutz und Datensicherheit, vol. 31, no. 9, pp. 681-683, 2007.
[6] "Biometric identification Boosting Automotive Security," 18 September 2015. [Online]. Available: http://www.iritech.com/blog/biometric-automotive-0915/. [Accessed 08 January 2016].
[7] "List of All Fingerprint Scanner Enabled Smartphones," 25 December 2015. [Online]. Available: http://webcusp.com/list-of-all-fingerprint-scanner-enabled-smartphones/. [Accessed 08 January 2016].
[8] "List of all Eye Scanner (Iris, Retina Recognition) Smartphones," 08 December 2015. [Online]. Available: http://webcusp.com/list-of-all-eye-scanner-iris-retina-recognition-smartphones/. [Accessed 08 January 2016].
[9] O’Neil, "Mobile Biometrics Market Analysis," Biometrics Research Group, Inc, 2015.
[10] A. Jain, R. Bolle and S. Pankanti, Biometrics: Personal Identification in Networked Society, New York: Springer, 2006.
[11] F. Galton, Finger Prints, London: McMillan, 1892.
[12] ResearchCapsule, "Fingerprint Sensors Market in Smart Mobile Devices 2012-2019," Research Capsule, Inc., 2015.
[13] GAO, "Technology Assessment Using Biometrics for Border Security," DIANE Publishing, 2002.
[14] N. Erdogmus and S. Marcel, "Spoofing 2D Face Recognition Systems with 3D Masks," in Idiap Research Institute, Darmstadt, 2013.
[15] E. Strickland, "Can Biometrics ID an Identical Twin?," 11 March 2011. [Online]. Available: http://spectrum.ieee.org/computing/software/can-biometrics-id-an-identical-twin. [Accessed 07 January 2016].
[16] A. K. Jain, A. Ross and S. Prabhakar, "An Introduction to Biometric Recognition," IEEE Transactions on Circuits and Systems for Video Technologies, vol. 14, no. 1, pp. 4-20, 2004.
[17] S. Karatzouni, N. Clarke and S. Furnell, "Device Versus Network-Centric Authentication Paradigms for Mobile Devices: Operational and Perceptual Trade-Offs," in Proceedings of 5th Australian Information Security Management Conference, Perth, 2007.
[18] Apple Inc., "About Touch ID security on iPhone and iPad," 02 November 2015. [Online]. Available: https://support.apple.com/en-us/HT204587. [Accessed 12 01 2016].
[19] Android, "Fingerprint HAL," [Online]. Available: https://source.android.com/security/authentication/fingerprint-hal.html. [Accessed 12 01 2015].
[20] Applying Feature Selection to Reduce Variability in Keystroke Dynamics Data for Authentication Systems. Mark Abernethy, Shri Rai. Proceedings of the 13th Australian

Information Warfare and Security Conference, Novotel Langley Hotel, Perth, Western Australia, 3rd-5th December, 2012

[21]A Survey of Keystroke Dynamics Biometrics. Pin Shen Teh, Andrew Beng Jin Teoh, Shigang Yue. The Scientific World Journal Volume 2013, Article ID 408280

[22]A Survey of Biometric keystroke Dynamics: Approaches, Security and Challenges. D. Shanmugapriya, G. Padmavathi. (IJCSIS) International Journal of Computer Science and Information Security, Vol. 5, No. 1, 2009

[23]B. Ngugi, B. K. Kahn, and M. Tremaine, “Typing biometrics: impact of human learning on performance quality,” Journal of Data and Information Quality, vol. 2, no. 2, article 11, 2011.

[24]B. Ngugi, M. Tremaine, and P. Tarasewich, “Biometric keypads: improving accuracy through optimal PIN selection,” Decision Support Systems, vol. 50, no. 4, pp. 769–776, 2011.

[25]Review of User Authentication by Using Keystroke Dynamics Technique. D. N. Rewadkar1, Dnyaneshwari S. Dhundad. International Journal of Science and Research (IJSR)

[26]Monrose, F. and Rubin, A. D. (2000). Keystroke Dynamics as a Biometric for Authentication. Future Generation Computer Systems, 16(4): 351-359

[27]Gaines, R. S., Lisowski, W., Press, S. J., and Shapiro, N. (1980). Authentication by Keystroke Timing: Some Preliminary Results. Technical report, Rand Corporation. Report number: R-2526-NSF.

[28]Paul Giura, Ilona Murynets, Roger Piqueras Jover, Yevgeniy Vahlis, “Is it really you?: user identification via adaptive behavior fingerprinting”, CODASPY 2014: 333-344

[29]Lukasz Olejnik, Claude Castelluccia, Artur Janc, “On the uniqueness of Web browsing history patterns”, Annales des Télécommunications 69(1-2): 63-74 (2014)

[30]Lawrence O’Gorman, “Comparing Passwords, Tokens, and Biometrics for User Authentication”, Proceedings of the IEEE, Vol. 91, No. 12, Dec, pp. 2019-2040, 2003.

[31]Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/~ml/weka/

[32] FIDO UAF Authenticator-Specific Module API, https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-asm-api-v1.0-ps-20141208.html

[33]FIDO UAF Authenticator Commands v1.0, https://fidoalliance.org/specs/uaf-v1.0-id-20141122/fido-uaf-authnr-cmds-v1.0-id-20141122.html

[34]FIDO UAF Authenticator Metadata Statements v1.0, https://fidoalliance.org/specs/uaf-v1.0-id-20141122/fido-uaf-authnr-metadata-v1.0-id-20141122.html

[35]UAF Architectural Overview, https://fidoalliance.org/specs/uaf-v1.0-id-20141122/fido-uaf-overview-v1.0-id-20141122.html

[36]FIDO UAF Protocol Specification v1.0, https://fidoalliance.org/specs/uaf-v1.0-id-20141122/fido-uaf-protocol-v1.0-id-20141122.html

[37]GlobalPlatform Device Technology, "TEE System Architecture," GlobalPlatform Inc., 2011.

[38]P. Leach, M. Mealling and R. Salz, A Universally Unique IDentifier (UUID) URN Namespace, Network Working Group, 2005.

[39]ARM Ltd., ARM Security Technolodgy - Building a Secure System using TrustZone Technology, 2005-2009.

[40]R. Coombs, Securing the Future of Authentication with ARM TrustZone-based Trusted Execution Environment and Fast Identity Online (FIDO), ARM Ltd., 2015.

[41]ARM Ltd., ARM1176JZF-S Technical Reference Manual, Revision: r0p7, 2004 - 2009.

[42]TRUSTED COMPUTING GROUP, "TPM MOBILE with Trusted Execution Environment for Comprehensive Mobile Device Security," 2012.

[43]Android Open Source Project, "Android Security - Hardware-backed Keystore," [Online]. Available: https://source.android.com/security/keystore/index.html. [Accessed 13 Jan 2016].

[44]Android Open Source Project, "Android Security - Fingerprint HAL," [Online]. Available: https://source.android.com/security/authentication/fingerprint-hal.html. [Accessed 13 Jan 2016].

[45]Android Open Source Project, "Android Security - Gatekeeper," [Online]. Available: https://source.android.com/security/authentication/gatekeeper.html. [Accessed 13 Jan 2016].

[46]Android Open Source Project, "Verified Boot," [Online]. Available: https://source.android.com/security/verifiedboot/index.html.

[47]Samsung Electronics Co., Ltd., White Paper: An Overview of Samsung KNOX Platform, September, 2015.