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 D3.4 "Multifactor authentication for DCA: user-to-device and device-to-network support (revised)"

Editor(s):	Claudio Soriente, Carmelo Acosta (TID)
Author(s):	CNIT: Alberto Caponi, Claudio Pisa, Tooska Dargahi CSGN: George Gugulea. Bogdan Chifor, Alberto Carp CUT: Savvas Zannettou EXUS: Vasileios Sarris TID: Claudio Soriente, Carmelo Acosta UC3M: Ruben Cuevas, Antonio Fernandez, Rafael García and Jose Herrera UPCOM: Vangelis Bagiatis, George Savvas UPRC: Christoforos Ntantogian, Eleni Veroni, George Karopoulos, Christos Lyvas, Panagiotis Nikitopoulos VERIZON: Steven Gevers, Mark Fidell WEDIA: Evangelos Kotsifakos, Panagiotis Georgatsos, Michalis Koulinas
Dissemination Level:	PU - Public
Nature:	R
Version:	0.3

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		Claudio Soriente (TID)	Initial Table of Contents
0.2		Claudio Soriente (TID)	First consolidated version
0.3		Claudio Soriente (TID)	Final version

Executive Summary

This deliverable is part of “WP3 – Beyond the Password: Device-centric Authentication” of the ReCRED project. The deliverable extends Deliverable D3.2 by providing the implementation details of the authentication mechanisms used in the ReCRED framework (for both local and remote authentication) and details the improvements and changes made to the protocols since the delivery of date of Deliverable D3.2.

The rest of the document is organized as follows. Section 1 introduces the concepts and the technologies presented in the deliverable. Section 2 describes user-to-device authentication based on face-recognition. Section 3 presents the first-factor authentication using FIDO. Section 4 introduces the authentication mechanisms that involve Behavioural Authentication Authorities. Section 5 presents QR-Login while Section 6 explains how ReCRED implements the virtual lock on the user accounts.

Table of Contents

List of Figures

Table of Abbreviations

AAR	Android Archive Library
AMM	Account Management Module
API	Application Program Interface
BAA	Behavioural Authentication Authority
BVK	B-Verifier Keyboard
CDR	Call Description Record
FIDO	Fast Identity Online
FLANN	Fast Library for Approximate Nearest Neighbours
HTTP	Hyper-Text Transfer Protocol
HTTPS	Hyper-Text Transfer Protocol Secure
IDP	IDentity Provider
JSON	JavaScript Object Notation
LBPH	Local Binary Patterns Histograms
OIDC	Open ID Connect
REST	Representational State Transfer
SP	Service Provider
TF-IDF	Term Frequency-Inverse Document Frequency
UAF	Universal Authentication Framework

1 Introduction

This is the revised version of “D3.2 Multifactor authentication for DCA: user-to-device and device-to-network support” and also the last deliverable of WP3 that happens to be due at the end of the work package. As such, rather than focusing on a general overview, this deliverable focuses on more technical details and, in particular, on implementation details of the authentication mechanisms used in the ReCRED framework (for both local and remote authentication) and details the improvements and changes made to the protocols since the delivery of Deliverable 3.2.

The deliverable starts detailing the last advances in the development of the face-recognition application – a user-to-device authentication mechanism that we include in the ReCRED framework given the ubiquity of front-facing cameras available in smartphones.

Next, the deliverable moves to the device-to-service authentication techniques. It starts giving a retrospective on the implementation efforts on FIDO UAF detailed in previous deliverable. The deliverable also explains how the behavioral techniques detailed in Deliverable 3.1, Deliverable 3.2 and Deliverable 3.3 have been harmonized and wrapped in a common technology that we name Behavioral Authentication Authority (BAA). We instantiated four BAAs based on browsing behaviour, on mobility patterns as recognized by the user mobile operator, on gait, and on keystroke dynamics, respectively. We also detail how BAAs are integrated with OpenAM to provide standards services of an Identity Provider such as authorization and authentication. Finally, we provide the implementation details of the account locking mechanism --- deployed within the Account Management Module of the ReCRED’s Identity Consolidator --- which allows user accounts to be locked or unlocked based on user-define policies or malicious events detected by the components of the ReCRED framework.

Compared to deliverable “D3.2 Multifactor authentication for DCA: user-to-device and device-to-network support” the present deliverable does not provide information on “Mobile Connect” and on “Fail-over authentication mechanism” since we felt that both technologies had a more natural allocation within WP4 and were thus described in the deliverable “D4.2 Full Identity Consolidator and Attributes Acquisition”

2 User to device authentication

2.1.1 The Face Recognition Module

ReCRED face recognition module is an Android application that is used to verify the identity of a device owner. The solution is based on OpenCV (Open Source Computer Vision), which is a popular and well-established library, providing programming interfaces to C, C++, Python and Android. OpenCV provides three different algorithms for face recognition: Eigenfaces, Fisherfaces and Local Binary Patterns Histograms (LBPH). Our face recognition module makes use of the LBPH algorithm. OpenCV also incorporates FLANN (Fast Library for Approximate Nearest Neighbours), an open-source library that contains a collection of algorithms optimized for fast nearest neighbour search in large datasets and for high dimensional features. The enrolment and authentication functions can be called by a third-party application with the use of explicit intents.

The face recognition module supports two operations: the user enrolment and the user verification. More details regarding the implementation of these operations are included in Deliverable 3.2 (Section 2.1 Physiological Biometrics). This section also includes an evaluation of the face recognition engine and its expected results (true/false positives/negatives) under different conditions.

In Deliverable 3.3 (Section 2.1 Face Recognition), we describe how the face recognition engine was adapted to be used as an authenticator, for user-to-device authentication. We also describe the main components of the face recognition, being the enrolment service, the authentication service and the retraining mobile app.

During the reference period of this deliverable, our effort was focused on securely saving the extracted facial templates (n-dimensional feature vectors) in the user's device.

For the enrolment part, the intent should set its package field as the "eu.upcom.recred.facerec" label, and as an action "eu.upcom.INVOKE_ENROLL". During this phase, the face captured is converted to a bitmap array and then encrypted and securely stored.

For the authentication part, the intent should set its package to "eu.upcom.recred.facerec" and as its Action to "eu.upcom.INVOKE_EVALUATE", and should be broadcasted with the permission "eu.upcom.javacv.permission.INVOKE". For training the classifier for the newly captured face, we get the encrypted faces from the secure storage, decrypt them and then convert them into an image.

The functionality of the Face Recognition module is very simple. At first, the user needs to enrol one or more profile pictures (Figure 1), in order to train the face recognition engine. For this, all the user needs to do is to use his device's camera and take a few pictures of himself. The app automatically detects the user's face and indicates this by placing a green rectangle around it. If no face is detected the app keeps waiting until a face is finally detected.

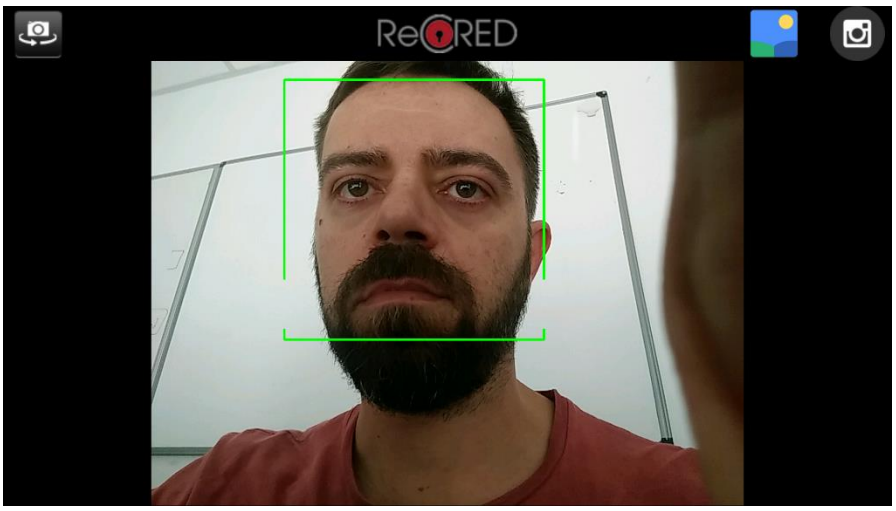

Figure 1: Face detection / capturing

The user can also call the retraining feature of the mobile app in order to add or remove facial templates. The functionality for adding new templates is similar to the initial enrolment.

Finally, the user can also see all the stored templates and delete any of them (**Figure** 2).

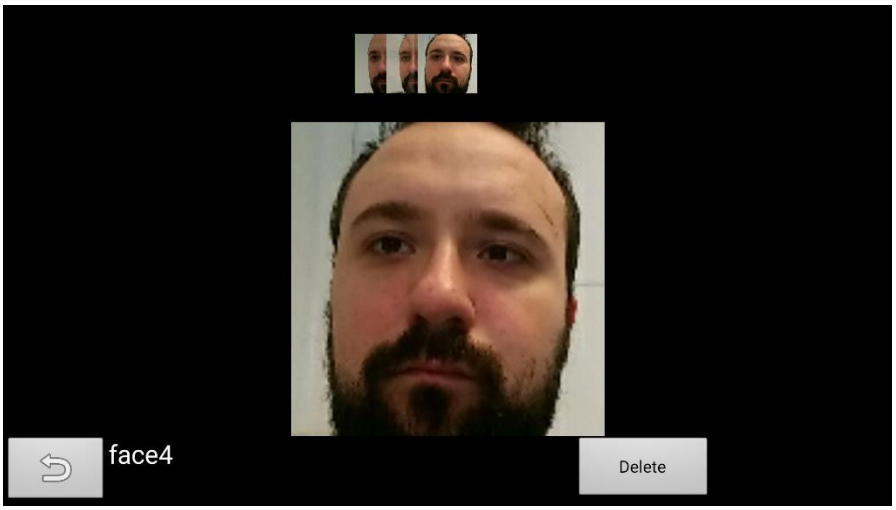

Figure 2: Management of facial templates

3 Device to service authentication

At the core of the authentication process, ReCRED relies on the FIDO UAF protocol to use the public/private key algorithms in order to implement a standardized challenge-response authentication protocol. Although FIDO UAF authentication can easily be used in any environment, in ReCRED we integrated it in an OpenID Connect infrastructure, such that FIDO UAF protocol processes are triggered every time the protocol needs to register, authenticate or deregister the user.

In the Deliverable 3.2 of this work package we described the FIDO UAF protocols: the registration, the authentication and the deregistration. In Section 3 of the same deliverable, we described how FIDO integrates with TEE on the user device and how FIDO UAF implementation is integrated with gateSAFE, on the server side.

In the ReCRED project, certSIGN implemented the entire FIDO UAF software stack from scratch. On the user-device there are three layers: the FIDO UAF Client, the FIDO UAF ASM and the FIDO UAF Authenticator. Below the authenticator, FIDO integrates with TEE; this component is already implemented in the Android software stack, starting with version 6.0. We described the integration and the mechanisms through which Android triggers user authentication to the FIDO UAF private key also in the Deliverable 3.2, Section 3.

certSIGN also implemented the server-side of the FIDO UAF protocol and obtained the FIDO Alliance certification of the implementation; we called this implementation *ReCRED FIDO UAF Server* and released it as open source software under Apache V2 license; the client-side components of FIDO UAF are also released under the same open source license. The FIDO Alliance certification was obtained in December 2016, as described in the Deliverable 3.3, Section 1. The certification proves the correctness and the completeness of the implementation, and what is probably most important, the interoperability with the other implementations.

Deliverable 3.3 also covers the implementation details, giving code examples of key function implementation for the user device and for the server side. Besides the protocol implementation that follows precisely the FIDO UAF specifications, we also extended the *ReCRED FIDO UAF Server* implementation with a few more REST API functions, that allows the integration of the server in the OpenID Connect protocol, by the mean of OpenAM server. The extensions are also described in Deliverable 3.3.

No changes were made in the implementation of the FIDO UAF protocol since the submission of Deliverables 3.2 and Deliverable 3.3., except bug fixing. The debugging process is expected to continue, as certSIGN is voluntarily participating in the interoperability test sessions as a reference implementation.

4 Behavioral second-factor authentication

The Behavioral Authentication Authority (BAA) is comprised of four different BAAs, according to the type of behavior considered to predict the authenticity of the user. The following subsections describe both the implementation and the exposed HTTP REST API of the BAAs.

4.1 Weblog implementation

The BAA weblog consists of two main modules – BP capture an BP verify. Figure 1 depicts all modules and external entities that communicate with BAA weblog, in order to serve as a browsing behavioral authority.

Red boxes in the diagram denote REST API entry points, the text shows URL address suffix. Black arrows show calls between different modules. Each arrow shows which module calls which REST API: empty end denotes a caller; sharp end denotes the end point.

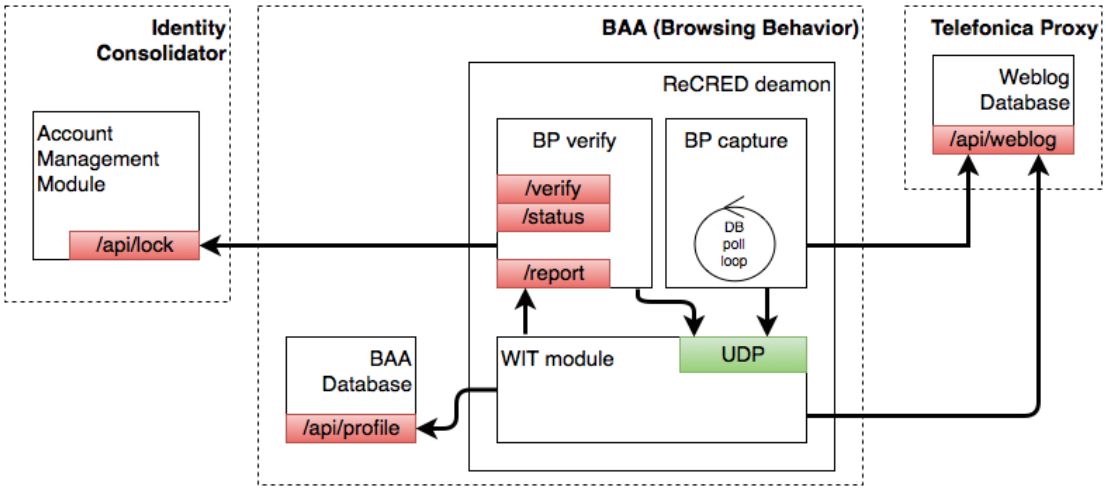

Figure 3: Weblog BAA

4.1.1 Description of modules

- BAA Database: stores behavioral profiles of different users. Offers REST API at /api/profile.
- Weblog Database: database with user's data in a security perimeter of Telefonica. Contains a time stamped history of web page addresses a user accessed. Offers REST API at /api/weblog.
- BP Verify: accepts verification requests. These are passed to WIT module which calculates similarity of behavioral profiles and the most recent user's behavioral data.
- BP capture: periodically polls Weblog Database and feeds WIT module with user's browsing data.
- WIT module: Web Identity Translator (WIT) module is used to fingerprint users based on the web addresses they visit. Accepts UDP messages, stores BP profiles into BAA Database over a REST API, queries Weblog Database for user's recent data, reports signatures similarity to BP verify module.
- Account Management Module (AMM): a component of the Identity Consolidator, main component of the ReCRED project. AMM implements locking functionality that allows locking

user-defined accounts when a suspicious behavioral activity is detected. BP verify module reports deviation between user's behavioral profile and recent data signature through REST API of the AMM.

4.1.2 Functionality

The basic idea behind BAA is that user's profile is created from a short history of his data. This is called learning phase and it is a variable parameter set by the WIT module. It could be time-dependent, for example, learning phase takes 10 days, or it can be message-driven, such as after first 100 web browsing addresses the profile is calculated.

Once a profile is calculated, it is stored in BAA Database to achieve persistence and robustness in case the BAA has to be restarted. BAA Database then can be used to inform user what type of information is it stored and how his behavioral profile looks like.

There are two main modes of operation of a BAA: authentication on demand and continuous authentication. Authentication on demand is invoked externally, by 3rd party using /verify or /status APIs. The purpose is to get confirmation that a user's recent behavior corresponds to his behavioral profile. Continuous authentication is a pro-active process in which the WIT module periodically checks for a discrepancy between user's recent data and his behavioral profile and when a mismatch is found, the result is reported to the Identity Consolidator's AMM which implements account locking functionality. The BAA requests locking of all user's accounts and it is up to AMM's internal logic whether it locks accounts eventually or not.

4.1.3 Database structure

The structure of weblog database is rather simple. The internal representation is a tuple (timestamp, user_id, web_address). This allows for filtering for records for particular users as well as to limit the DB query to select records according to time.

The structure of BAA database is a histogram of web addresses visited by a user. It is represented as a tuple (user_id, web_address, count) where count denotes number of visit at the particular web_address during the learning phase.

4.1.4 Message flows

Learning phase

The message flow for learning phase of BAA is depicted in Figure 4. Once a BAA instance is started, the polling of a Weblog database takes place. Data are requested via REST API on /api/weblog end-point regularly. Only data newer than the last request are queried, this is achieved by passing a datetime filter in GET query string.

Response from Weblog DB contains a list of tuples (user_id, web_addres). Each tuple represents one visit of a user on a particular web address. The BP capture module then feeds those tuples into a WIT module through UDP messages in a form of *HISTORY_UPDATE user_id web.*

Once enough messages are received by the WIT module, or after a specified learning time, the behavioral profile is calculated for the user.

User's behavioral profile is then posted to BAA database via REST API and POST method in which the data part contains tuples (user_id, web, count), simply a histogram of user's visits at web pages. Successful creation of database entry is confirmed by a CREATED response.

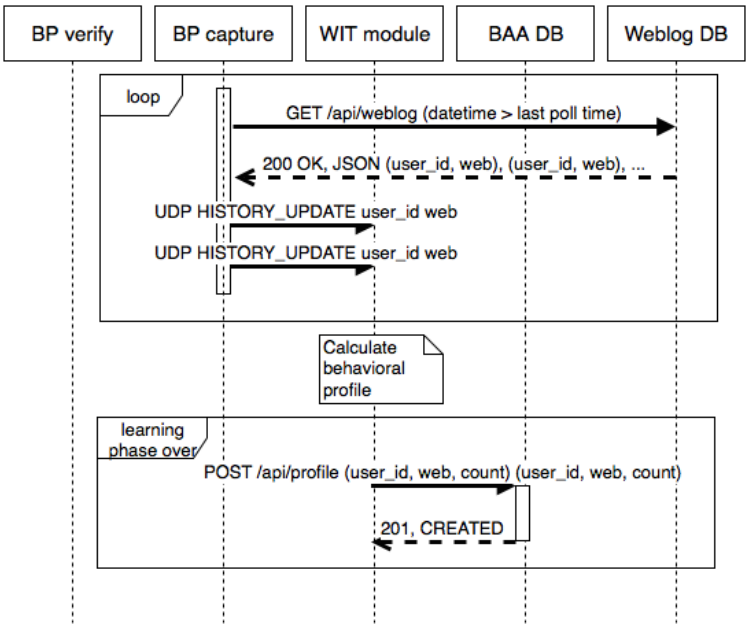

Figure 4: Weblog BAA – Learning phase

Verify profile

The message flow to demonstrate how verification of a profile works is depicted in Figure 5. Any party that wants (and is allowed) to verify user's behavior posts a GET message via a REST API to BP verify module endpoint /verify/user_id, where user_id is the system-wide user's identifier. The response is accompanied with JSON data that contains a UTC Unix timestamp denoting when the WIT module's received the request. This value can be later used to assess whether the verification has been finished or not.

Since the verification of user's behavioral profile could be a time-consuming process in which many parties and modules are involved, an immediate response is returned that confirms an accepted request and it contains a location header with an URL of result.

The verification of profile is requested by BP verify module by sending an UDP message VERIFY user_id to the WIT module. No response is awaited through the same channel. Instead, based on the result of profile verification, the WIT module later posts a response to BP verify module's endpoint /report/user_id. There are three possible answers:

- OK means that user's behavioral profile corresponds to his recent history
- UNAUTH means that user's behavioral profile doesn't correspond to his recent history
- UNKNOWN means that a learning phase is still in place for that particular user, or in case of any error during BP verification processing

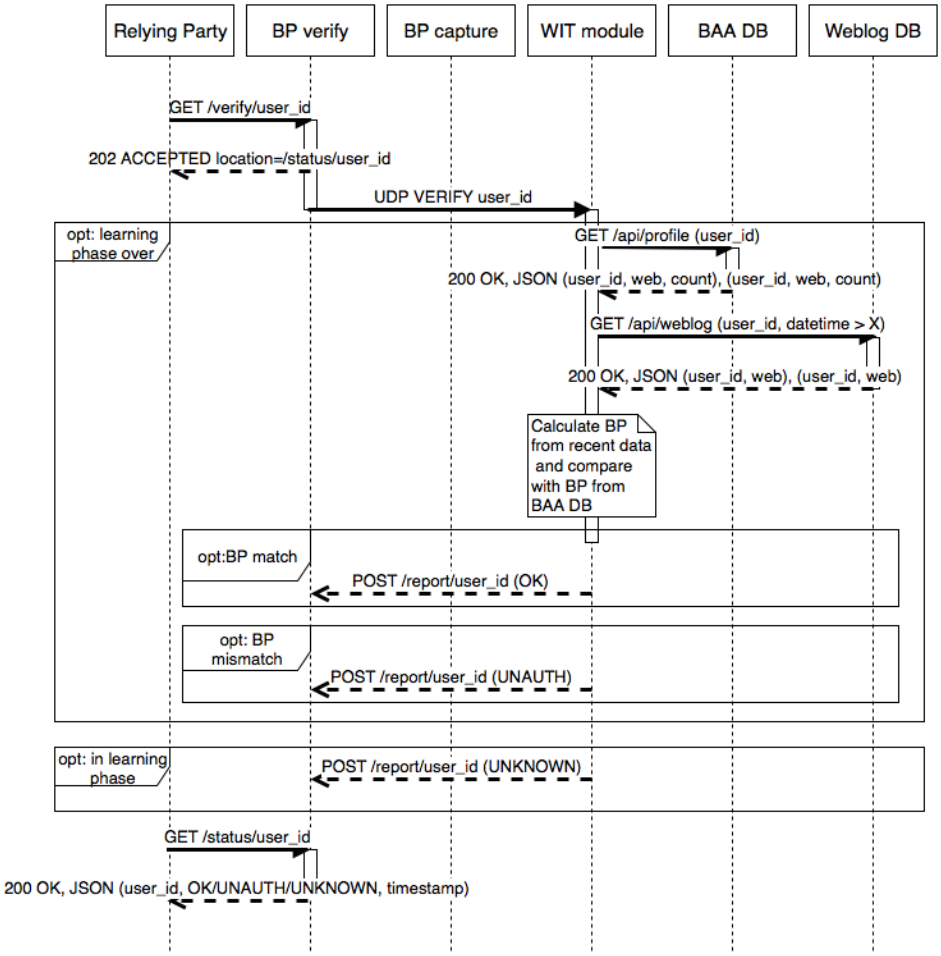

Figure 5: Weblog BAA – Verify phase

To achieve behavioral profile verification, the WIT module first queries the BAA DB on /api/profile REST API interface with a query string specifying user’s user_id. Next, it queries the Weblog database to get recent history of user’s browsing. The history extent is an internal parameter and can be changed. As such it is passed in a query string in GET method request on /api/weblog REST API. The comparison between user’s pre-calculated BP profile and similar representation of the recent history is carried out and the result is returned to BP verify module.

Finally, the Relying Party (i.e., Service Provider) asks for status of user’s behavior verification via /status/user_id REST API end-point using a GET method. The response contains user_id, status of verification and a timestamp, when the verification has been finished. The Relying Party should compare the timestamp received in verify response with status response. If the timestamp of status response is smaller than verify response timestamp, it means that the verification process has not finished yet.

Note that the Relying Party can skip the verify request and just query the status of user’s verification. The continuous BP profile verification in background (if turned on) updates automatically user’s verification status. Based on the timestamp, the Relying Party can decide if the most recent status is fresh enough to serve as a trusted reference or if it better invokes the verify request to obtain more recent status.

4.2 Mobility implementation

The BAA mobility, that authenticates a user based on his mobility patterns, consists of two main modules – BP capture an BP verify. Figure 6 depicts all modules and external entities that communicate with the BAA mobility.

Red boxes in the diagram denote REST API entry points, the text shows URL address suffix. Black arrows show calls between different modules. Each arrow shows which module calls which REST API: empty end denotes a caller; sharp end denotes the end point.

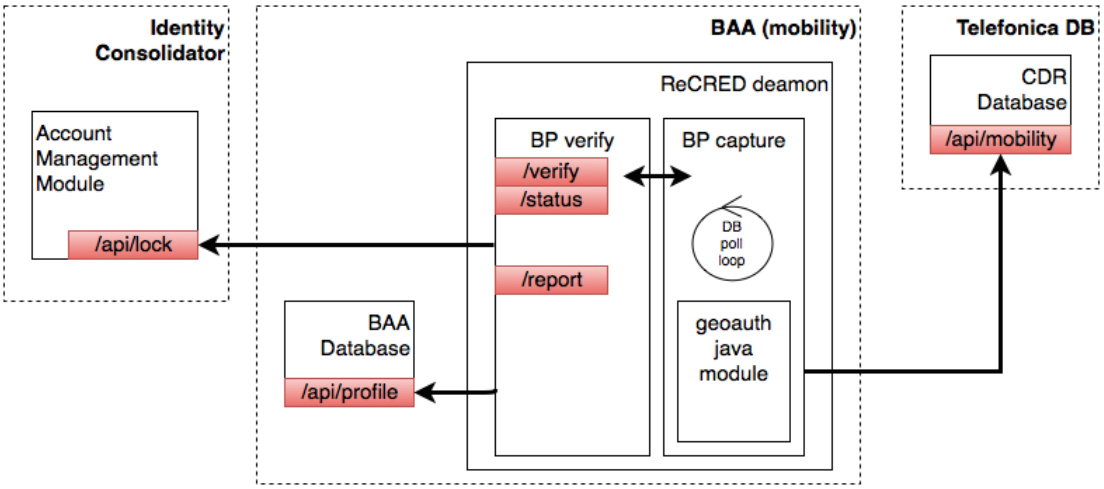

Figure 6: Mobility BAA

4.2.1 Description of modules

- BAA Database: stores behavioral profiles of different users. Offers REST API at /api/profile.
- CDR Database (Mobility Database): database with user's Call Data Records in a security perimeter of Telefonica. Contains a time stamped history of calls being made or received or text messages sent and received. Offers REST API at /api/mobility.
- BP Verify: accepts verification requests. These are passed to BP capture module which calculates similarity of behavioral profiles and the most recent user's behavioral data.
- BP capture: periodically polls Mobility Database and feeds geoauth java module with user's mobility data.
- geoauth java module: a set of java functions, wrapped in one geoauth package. It consists of four functions:
    - CDRsLocationVectorExtended parses mobile CDRs and construct location vectors per user (aka behavioral mobility profile). The mobility profile is stored in BAA Database.

- CDRsVectorRandomPairwiseFixedComparisonExtended computes the cosine similarity between vector of a user-set provided with X random users from the population. Similarity score is used later on.

- CDRsVectorSameUserComparisonExtended is used to compute cosine similarity between the historical profile and current fingerprint, obtained from recent data, of the same user.

- CDRsRankSimilarityExtended computes the rank of user when comparing his self-similarity score with random other user's pairwise scores. Final rank is then used to decide on authentication: if user's rank scores best among other users, then the user is considered as being different enough from all other users and thus is authenticated.

4.2.2 Functionality

The basic idea behind BAA is that a user's profile is created from a short history of his data. This is called learning phase and it is a variable parameter provided to the BP capture module. It is time-dependent, for example, learning phase takes last 10 days of user's CDR's and from the cells visited, a mobility profile is created.

Once a profile is calculated, it is stored in BAA Database to achieve persistence and robustness in case the BAA has to be restarted. BAA Database then can be used to inform user what type of information is it stored and how his behavioural profile looks like.

There are two main modes of operation of a BAA: authentication on demand and continuous authentication. Authentication on demand is invoked externally, by 3rd party using /verify or /status APIs. The purpose is to get confirmation that a user's recent behavior corresponds to his behavioural profile. Continuous authentication is a pro-active process in which the BP capture module periodically checks for a discrepancy between user's recent data and his behavioral profile and when a mismatch is found, the result is reported to the Identity Consolidator's AMM which implements account locking functionality. The BAA requests locking of all user's accounts and it is up to AMM's internal logic whether it locks accounts eventually or not.

4.2.3 Database structure

The structure of CDR (mobility) database is rather simple. The internal representation is a tuple (timestamp, user_id, cdr). This allows for filtering for records for particular users as well as to limit the DB query to select records according to time.

The structure of BAA database for storing behavioral profile is a vector of visited cells together with corresponding tf-idf value per user. It is represented as a tuple (user_id, caller_vector, calee_vector) where caller_vector denotes vector of cells and tf-idf values for calls in which the user is a caller; callee_vector has a similar meaning except the user is a callee.

4.2.4 Message flows

Learning phase

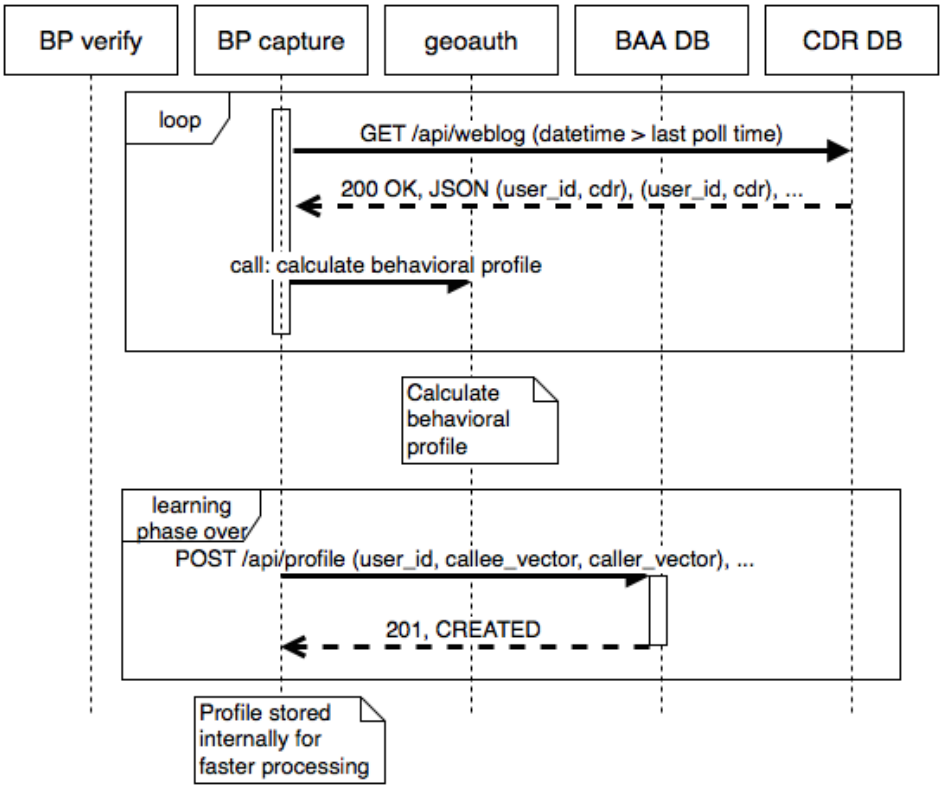

Figure 7: Mobility BAA – Learning phase

The message flow for learning phase of BAA is depicted in Figure 7. Once a BAA instance is started, a fingerprinting of all users in the database starts.

Response from CDR DB contains a list of tuples (user_id, CDR). Each tuple represents one call or text message made/received/sent. The BP capture module then feeds those tuples into geoauth module through a file. The geoauth module returns user's behavioral profile, which is a vector of visited cell-ids together with their corresponding tf-idf values.

User's behavioural profile is then posted to BAA database via REST API and POST method in which the data part contains tuples (user_id, caller_vector, callee_vector). Successful creation of database entry is confirmed by a CREATED response.

After the learning phase is done, a regular polling of a database takes place. Data are requested via REST API on /api/mobility end-point regularly. Only data newer than the last request are queried and this is achieved by passing a datetime filter in GET query string.

Verify profile

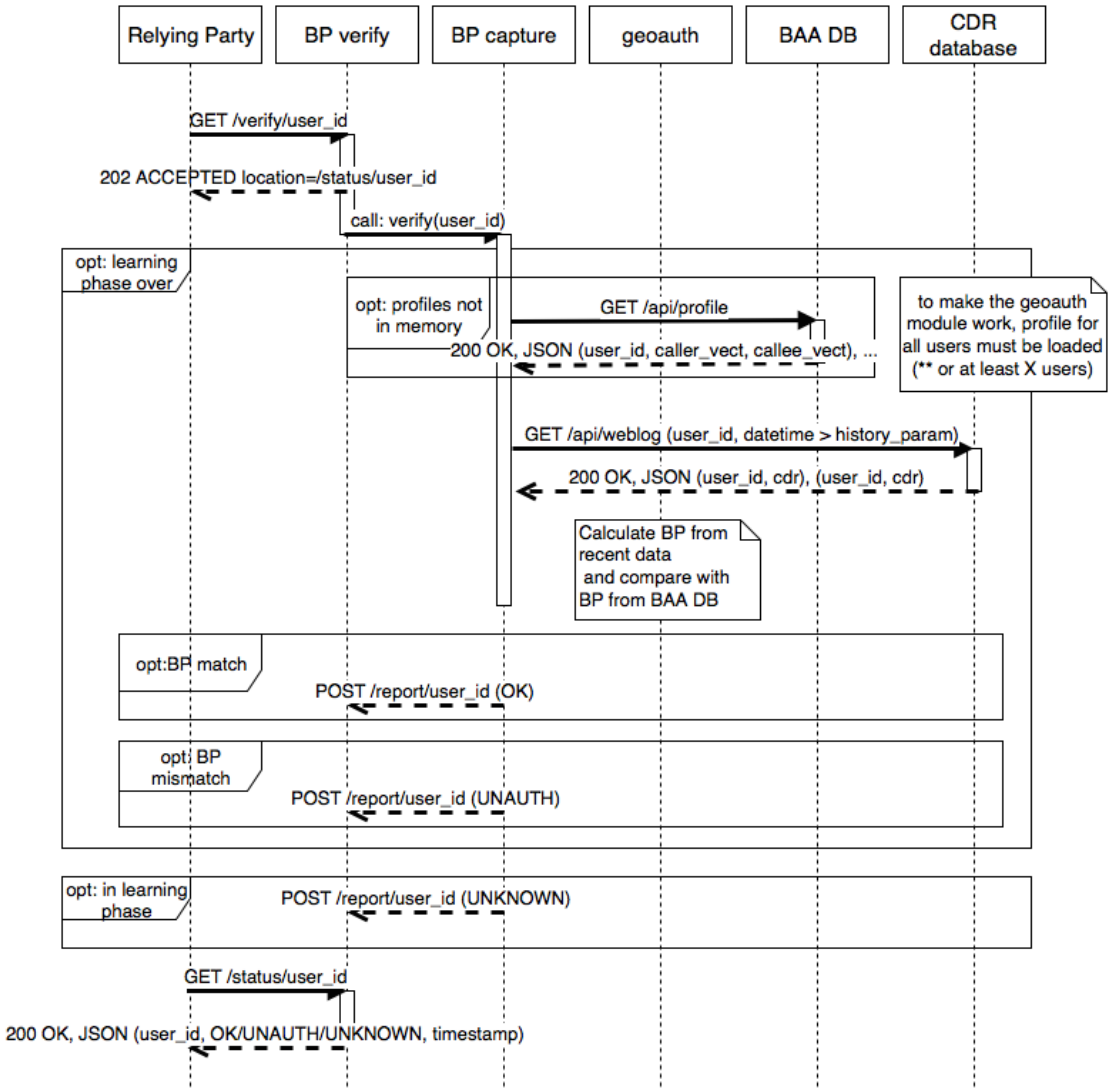

Figure 8: Mobility BAA – Verify phase

The message flow to demonstrate how verification of a profile works is depicted in Figure 8. Any party that wants (and is allowed) to verify user's behavior posts a GET message via a REST API to BP verify module endpoint /verify/user_id, where user_id is the system-wide user's identifier. The response is accompanied with JSON data that contain UTC Unix timestamp denoting when the BP verify module's received the request. This value can be later used to assess whether the verification has been finished or not.

Since the verification of user's behavioral profile could be a time-consuming process in which many parties and modules are involved, an immediate response is returned that confirms an accepted request and it contains a location header with an URL of result.

The verification of profile is requested by BP verify module by internally calling verify function of BP capture module. The call ends by spawning a verification thread because the verification process might

be time-consuming. The BP capture later posts a response to BP verify module’s endpoint /report/user_id. There are three possible answers:

- OK means that user’s behavioural profile corresponds to his recent history
- UNAUTH means that user’s behavioural profile doesn’t correspond to his recent history
- UNKNOWN means that a learning phase is still in place for that particular user, or in case of any error during BP verification processing

To achieve behavioral profile verification, the BP capture module queries BAA DB on /api/profile/mobility REST API interface. For geoauth module it is crucial to have a profile of all users. To prevent overhead of loading the behavior profiles for all users each time, the profiles are requested once and then stored in BP capture module’s memory. Next, BP capture queries CDR Mobility database to get last user’s mobility patterns in form of raw CDRs. The history extent is an internal parameter and can be changed. As such it is passed in a query string in GET method request on /api/mobility REST API. The comparison between user’s pre-calculated BP profile and similar representation of the recent history is carried out and the result is returned to BP capture module.

Finally, the Relying Party asks for status of user’s behavioral verification via /status/user_id REST API end-point using a GET method. The response contains user_id, status of verification and a timestamp, when the verification has been finished. The Relying Party should compare a timestamp received in verify response with status response. If the timestamp of status response is smaller than verify response timestamp, it means that the verification process still takes place.

Note that the Relying Party can skip the verify request and just query the status of user’s verification. The continuous BP profile verification in background (if turned on) updates automatically user’s verification status. Based on the timestamp, the Relying Party can decide if the most recent status is fresh enough to serve as a trusted reference or if it better invokes the verify request to obtain more recent status.

4.3 GAIT implementation

The ReCRED daemon that authenticates users based on their gait patterns consists of three main components: the BAA main process, the data acquisition process, and the storage. Each of these components comprehend some other modules. Figure 9 shows the components and modules that constitute the GAIT BAA.

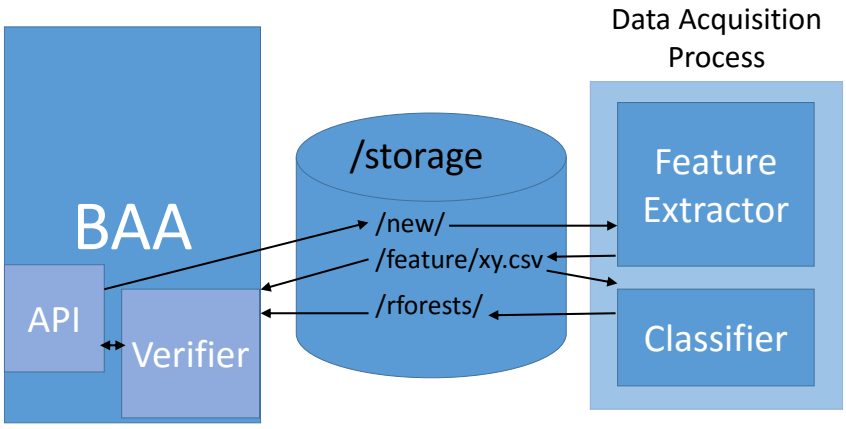

Figure 9: Gait BAA

4.3.1 Description of modules

Feature Extractor

Comprises all data processing steps required to start from a raw data input file, uploaded from the mobile app (see Deliverable 3.2) and stored in */storage/new*, until obtaining a new data sample, comprising 163 features, appended as a new line in the */feature/xy.csv* file. The main steps are:

- **Relative time calculation:** due to inconsistent sensor timestamps between different devices, all timestamps included in the input data are recomputed as relative, starting in time 0 for each input text file.

- **Time interpolation:** Since mobile sensors are low powered and has got low stability, the signal is interpolated, using linear interpolation, so that every pair of consecutive data points had the same time difference.

- **Zero normalization:** The device coordinate system is transformed into earth's coordinate system (with z-axis pointing towards sky). Due to smartphones accelerometer sensors instability, a zero normalization is required to minimize noise in the metrics.

- **Resultant acceleration:** In order to minimize any orientation related problem, the three dimensional acceleration is converted into its resultant form by calculating the L2 norm of the acceleration vector.

- **Walk annotation:** Since data is collected consecutively in time (each file comprises 1 minute) and some data may be part of walk or not, it is computed and annotated to each data sample

(as an additional feature) the predicted percentage of walk. This is done using a rolling 3-second-long window over the whole input data and a band-pass Butterworth filter with cut-off frequencies of [1, 3] Hz, transforming the signal window into frequency domain. Walk-like samples surpass the maximum frequency of the filter.

Classifier

Gait behavioral prediction uses a machine learning classification method called *Random Forests*. For each user with at least 30 data samples (i.e., 30 rows in the */feature/xy.csv* file, each one with 163 features) with a minimum walk annotation of 70%, a random forest is trained. The random forest training process involves both the 30 data samples of the user (as positive cases) and the data samples of all other users (as negative cases), in order to maximize the accuracy of the trained forest.

The Classifier reads all available data samples in */feature/xy.csv* file and creates a new random forest for each user with enough valid data that had not a trained random forest yet. Each created and trained forest is stored in a single file in the storage under */rforests/*

Verifier

Gait behavior is predicted using a priory trained random forest and recent gait data. On the one hand, the Verifier keeps in memory a trained random forest for each already verified users. That is, if a user has been either verified or his status has been checked, and a trained random forest is available for that user. In any of these cases, the Verifier checks whether there is available a trained random forest for the user in the storage, under */rforests/.* If that is the case, the Random Forest is loaded in memory.

On the other hand, the Verifier also keeps in memory all data samples acquired and processed so far, stored in storage's */feature/xy.csv* file. In order to compute the current status of a given user with an available Random Forest, the most recent 5 data samples are given as input to the trained Random Forest. Each of these data samples will end up with a True/False prediction (True means "the device is being used by the user"). If the percentage of True predictions is greater or equal to 60% the user is Authenticated; otherwise the user will be non-authenticated.

The BAA gait server runs a singleton instance of Verifier and checks it on each /status or /verify API call. If the Verifier does not have an available random forest for the requested user the returned status is "UNKNOWN".

API

Exposes the HTTP REST API that allows to retrieve whether a user is authenticated according to his recent gait behavior. It also allows the GAIT mobile app, running on the user device, to regularly upload data files with gait info collected by the device sensors.

Storage

Local and configurable storage that works as a communication channel between the two main processes that comprise the BAA GAIT. The main reason for relying process communication via physical storage is the high data volume. GAIT behavior predictions may involve thousands of files requiring up to tens of Gigabytes to store both raw and processed data.

Functionality

Gait data is collected by a mobile application running on the user device. Data collected from device's sensors is stored in text files and uploaded to the gait server regularly using the exposed HTTP REST API. Once the data files arrive to the server it is initiated a data processing, that includes some complex tasks, such as resampling to different frequency and extracting more than 160 features from raw data. In order to deal with such a computational complexity, and still attending incoming requests, the gait server triggers an additional process to detach data acquisition from attending incoming HTTP requests.

The data acquisition process works on a configurable storage and moves input files according to each of the intermediate steps of the data processing. Uploaded files are stored in */storage/new* by the BAA main process. Therefore, both processes communicate via the storage. There are two main processed data products yielded by the data acquisition process: data samples and Random Forests. Each of them will be used in the verification process in order to predict the current status of a user according to his gait behavior.

Data samples are comprised of a list of 163 features extracted from the incoming raw data. Each of these data samples are stored as a single row in the */feature/xy.csv* file. Once there are enough data samples for a single user it is possible to create a prediction model to determine whether subsequent data samples belong to the same user or not.

4.4 Keystroke Dynamics BAA implementation

As in previous cases, in the case of the BAA that uses keystroke dynamics to authenticate the user consists of two main modules, the capturing/training and the verification ones.

The server that receives the keystrokes from the B-Verifier Keyboard (BVK) application is the Behavioral Authentication Authority that is connected to the User Device, the ID consolidator and the Service Providers that want to authenticate the user with a second factor behavioral authentication.

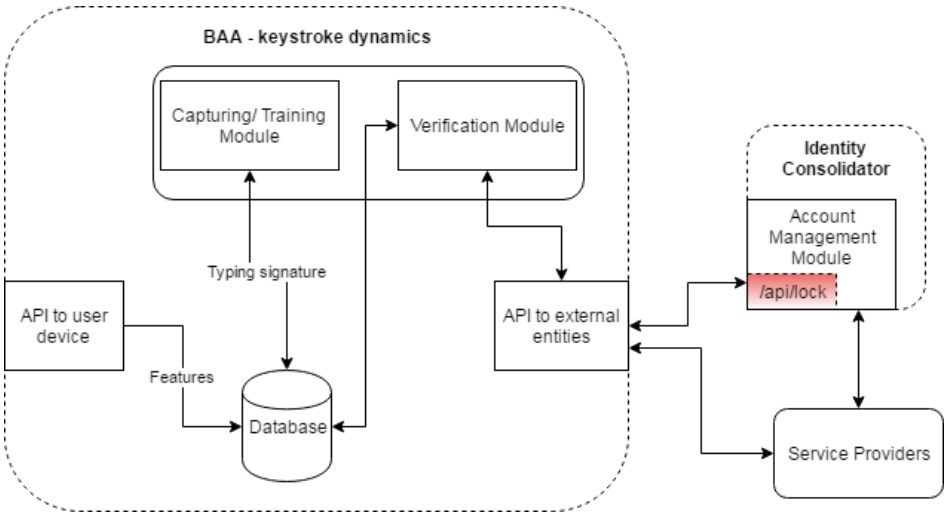

Figure 10 B-Verifier client and server modules

4.4.1 Description of modules

- BAA Database: stores behavioral profiles of different users, the raw data arriving from the device and also the verifications that take place.
- Verification module: accepts verification requests. These are passed to BP capture module which calculates similarity of behavioral profiles and the most recent user's behavioral data.
- Capturing/Training module: gets data from user device, stores them to the database.
- Account Management Module: a part of Identity Consolidator, main component of ReCRED project. AMM implements locking functionality that allows locking user-defined accounts when a suspicious behavioral activity is detected. BAA verification module reports deviation between user's behavioral profile and recent data signature through REST API of the AMM.

4.4.2 Functionality

The BAA runs a capturing and training phase and a verification phase for every device/user under the following procedure.

The BAA receives the user keystrokes and runs the following process.

1. Receives the session (set of keystroke sequences).
2. It checks whether the specific *deviceID* has already a typing signature associated with it (for the specific application – appID and for the device in general).
    a. If yes, then the incoming session is sent to the verification module (go to step 4).
    b. If no, then the incoming session will participate to the training phase. It is stored in the database in the appropriate table/file.
3. The **training module** checks if there are enough samples (keystrokes), based on the respective parameter of the configuration, in order to start the creation of the user typing signature.
    a. If the samples are less than those needed, it performs no action.
    b. If the samples are adequate, then it reads all the training keystrokes and produces the signature for the user. This is done by aggregating the training records.
    c. The user signature is stored in a separate table, along with the creation timestamp and information needed for the verification.

4. The **verification module** aggregates the session that is to be verified and produces the session signature, using the same process that is used to produce the signature of the user.
5. The aggregated session is then compared with the user signature and a matching score is produced.
6. The produced scores are stored in the database, along with the session signature, in a separate table that is to be used to answer the authentication request of the Service Provider.

Note that the process of building user typing signatures is done per device and per appID. A cross-application score is also generated every time a signature for a new appID is created. So, the BAA performs training for every appID that uses the BVK, creates a signature for each one of them and holds the latest N scores for every appID. A cross-application signature is also calculated and the N latest scores of the comparison of this signature and the latest session are also stored. These scores are to be used either when a Service Provider requests a second factor authentication, or to notify the Identity Consolidator (IDC) if an unusual behavior is noticed from the BAA. The IDC can then decide to lock some or all user accounts (according to the policies that the user has set), to notify the user or take no action, according to the user preferences.

4.4.3 Database

The database holds all raw data coming for user device, the user's signatures for every app and for the whole device and it also holds the verifications that are taking place.

4.4.4 Training and Verification

After evaluating models of training and verification based on data mining algorithms, clustering of the features and vector distance metrics, an extensive research has been also performed applying statistical methods as well. The combination of statistical methods allows the model to be adaptive to user's changing behavior, hence avoiding false negatives and false positives. Early evaluation results have shown that the model performs very well, recognizing the change of the users on the same mobile device. Therefore, this approach has been implemented in the final version of the BAA. Below an analysis of the theory and details of the implementation are provided.

Notions

Let U^{dev}, U^{app} be random variables denoting the particular user that uses a specific mobile device, in general or in the context of an application, respectively; they take the values $orig, other$, with $orig$ being the user that has supplied the first reference sessions based on which the usage signature of the device has been created.

Bh^{dev}, Bh^{app} be random variables denoting a specific usage ***behavior*** deduced from ***statistically comparing*** according to the Kolmogorov-Smirnov non-parametric goodness-of-fit test the captured times of certain ***incremental keystroke times*** (called features) with the corresponding reference ones which are assumed that have been produced from the legitimate owner of the devise (original user).

Five features are considered and therefore the domain of possible behaviors consists of 5-d vectors of the form $\underline{b} = (b_1, b_2, b_3, b_4, b_5)$, $b_i = g|y|r, i = 1..5$, being the aggregate outcomes of the Kolmogorov-Smirnov test from comparing the original and the captured samples per feature; a total of $243 = 3^5$ values.

From the Law of Total Probability, we have:

$$Prob\left(U^{dev} = orig\right) = \sum_{app} Prob(U^{app} = orig | A = app) Prob(A = app)$$

Bayesian Model

The Bayes' rule states:

$$Prob\left(U^{app} = orig \mid Bh^{app} = \underline{b}\right) = \frac{Prob\left(Bh^{app}=\underline{b} \mid U^{app}=orig\right) Prob(U^{app}=orig)}{Prob\left(Bh^{app}=\underline{b}\right)}$$

or

$$\frac{Prob\left(U^{app}=orig \mid Bh^{app}=\underline{b}\right)}{Prob(U^{app}=orig)} = \frac{Prob\left(Bh^{app}=\underline{b} \mid U^{app}=orig\right)}{Prob\left(Bh^{app}=\underline{b}\right)} \quad (1)$$

The above states that as we capture usage behaviors, the ***belief*** that the same user uses the mobile device increases or decreases depending on the belief that the captured behaviors match the usage profile of the original user –which is intuitively valid. Note that for the problem at hand, the distribution of the evidence in a Bayesian set-up, $Prob(Bh^{app})$, is not fixed/known but rather needs to be adaptively built as evidences (usage behaviors) are coming.

Ideally, one would expect to capture only absolutely confirmative behaviors i.e. (g, g, g, g, g). However, this is not the case in practice because a) the usage of the same user cannot be taken to be deterministic, and b) inevitably there are errors in the statistical model used for inferring a behavior. Based on the above, we need to build the conditional and overall (unconditional) distribution of behaviors. We adopt a frequentist approach as follows:

Bayesian Model Realization

We distinguish between ***affirmative*** and ***suspicious*** behaviors. An affirmative behavior is the one that has a minimum number of g's. That is, in an affirmative behavior there are no reasons to doubt that the user was the original in at least a number of features. We consider that all behaviors captured until an affirmative one come from the original user, in other words we consider that affirmative behaviors indicate that the original user continues using the mobile device and as such it is safe to believe that all previous behaviors have been produced by the original user. Based on this reasoning, one might also argue that the behaviors encountered short after an affirmative one may also come from the original user, however this requires the incorporation of time information and this aspect is not considered further presently.

Based on the above:

- the overall behavior distribution, $Prob(Bh^{app})$, builds based on all captured behaviors (as proportions of occurrence over received sessions) , whereas
- the conditional behavior distribution, $Prob(Bh^{app}|U^{app} = orig)$, builds only with the behaviors encountered before and including an affirmative behavior.

That is, assuming that the last affirmative behavior has received at the Nth session and currently we are at the $(N + N')th$ session:

$$Prob\left(Bh^{app} = \underline{b} | U^{app} = orig\right) = \frac{\#occurences\ of\ \underline{b}\ in\ N}{N}$$

$$Prob\left(Bh^{app} = \underline{b}\right) = \frac{\#occurences\ of\ \underline{b}\ in\ N+N'}{N+N'}$$

Noting that

$$\frac{k+k'}{N+N'} > \frac{k}{N} \Leftrightarrow \frac{k'}{N'} > \frac{k}{N}$$

the probability of a suspicious behavior may increase over its conditional one (of the original user) during the period starting from the last affirmative behavior.

The question then, is the degree at which the overall probabilities (frequencies) of the suspicious behaviors exceed the conditional ones (of the original user) after an affirmative behavior is encountered. While waiting for the next affirmative behavior, if the overall probability (frequency) of a suspicious behavior:

- increases over the corresponding conditional one, the denominator of the right side of (1) becomes greater than the numerator, the fraction less than 1 and therefore the posterior belief that the user is the original, given the latest evidence, is shaken;
- remains within the corresponding conditional one, the belief that the user is the original is strengthened.

The magnitude of ratio could be seen in the following:

$$\frac{\frac{k}{N}}{\frac{k+k'}{N+N'}} = \frac{k}{k+k'} + \frac{1}{N}\frac{kk'}{k+k'} \cong \frac{k}{k+k'} \text{ (for large } N\text{'s)} \qquad (2)$$

As it can be seen, the ratio increases at a dropping rate as k increases (first derivative is positive while the second one is negative).

The above analysis leads us considering a criterion related to the number of consecutive suspicious behaviors and the amount of distortion they incur in (the conditional probabilities reflecting) the statistical behavior of the original user.

Misbelief Level

We define the misbelief level as:

$$misbelief\ level \triangleq m \triangleq$$

$$\triangleq 1 - \frac{Prob\left(Bh^{app}=\underline{b} | U^{app}=orig\right)}{Prob\left(Bh^{app}=\underline{b}\right)} \qquad (3)$$

where the numerator calculates the probabilities as frequencies until and the last affirmative behavior while the denominator calculates the probabilities as frequencies until and the last encountered behavior.

Based on (1) we have:

$$\frac{Prob\left(U^{app}=orig \mid Bh^{app}=\underline{b}\right)}{Prob\left(U^{app}=orig\right)} = \frac{Prob\left(Bh^{app}=\underline{b} \mid U^{app}=orig\right)}{Prob\left(Bh^{app}=\underline{b}\right)} = 1 - m \qquad (4)$$

Considering the change in noticing a behavior from what has been already accounted as expected from the original user, that is

$$\Delta(\underline{b}) = Prob(Bh^{app} = \underline{b}) - Prob(Bh^{app} = \underline{b} \mid U^{app} = orig) \quad (5)$$

the misbelief level can be alternatively defined as the relative error between the actual frequency with which a certain behavior appears and the one that the original user has previously exhibited:

$$\frac{\Delta(\underline{b})}{Prob(Bh^{app}=\underline{b})} = m \quad (6)$$

As it can be seen from (6), if a rare behavior is encountered the misbelief will be close to 1 (at its maximum); because the change in the numerator will be almost as the denominator, the same can be noticed from (2).

That is, as behaviors are encountered, the ***misbelief level expresses the degree of deviating from the belief that the current user is the original (=1)***. Negative or zero values strengthen the belief whereas positive values weaken the belief.

Bayesian Misbelief Criterion

As sessions with keystroke times are received and behaviors are encountered by seeing how close- in a statistical sense- are the received times and the reference ones, the misbelief level can be computed in a session-by-session manner.

A simple ***threshold-based criterion*** is proposed:

- If the misbelief level remains below a specific value, then the belief that the original user uses the device holds.
- If the misbelief level exceeds a specific value, then the belief that the original user uses the device is removed.

Typical values of the critical threshold on misbelief level may be 0.05, 0.1 or 0.2. Different alert levels could be defined for different threshold values. However, cautions should be taken in applying the above criterion since the values of the misbelief level may be ***misleading***. This is because, the misbelief level relates to probabilities measured as frequencies -proportions over a number of trials- and as such they are volatile -they may change significantly from trial to trial, see (2). Indicatively we refer to the following examples:

First, suppose that the first suspicious behavior is encountered at the first or fifth trial. The misbelief level will be 1 or 0.2, respectively, either of which is misleading in the sense that the original user may well produce suspicious behaviors early enough.

Second, suppose that suspicious behaviors are encountered at the 9th and 10th trials (received sessions). At the 10th trial the misbelief level will be 0.45 (=1-1/9/2/10=1-10/18=1-0.55), which is again misleading since the original user may well produce two consecutive suspicious behaviors.

We propose to use the misbelief criterion after ***200 trials*** (received sessions). Our analysis (see next chapter) shows that after this number of trials, the frequencies, from trial to trial, cannot differ, in absolute terms, more than 0.01 in total. We reckon that this stability makes the application of the misbelief criterion and the proposed threshold values safe enough.

Note that a smaller range of trials could be considered before applying the misbelief criterion e.g. after 20 or 40 trials after which the change of frequencies remains within neighbors of 0.1 or 0.05 units. However, we reckon that the threshold value should be suitably adapted according to this anticipated magnitude of frequency vector change -in other words, the inherent volatility in probability calculation should be suitably absorbed in setting the criterion –see (2). This aspect is left for further elaboration.

By applying the misbelief criterion after a (fairly) large number of trials, the repeated authentication process could be made more agile by splitting the space of suspicious behaviors into partitions until the thinnest partition level consisting of all individual suspicious behaviors. This way, if an individual suspicious behavior like the extremely suspicious one (r, r, r, r, r) is encountered for first time, the misbelief will be 1. The misbelief level will also be high (see (2)) for a rare suspicious behavior, say at 2/100, which is however encountered soon after e.g. at the 201th trial, but it will not be high if the rare suspicious behavior encountered at the 300th trial. The consideration of thinner sets of suspicious behaviors is also left for further elaboration.

Anticipated Success or Geometric 99-percentile Criterion

We view the encountering of an affirmative behavior as a 'success' and each session as an independent Bernoulli trial. The number of trials (received sessions) until and including (required to get) a success follows the well-known Geometric distribution:

$$Geometric\ Distribution; success\ probability\ p$$

$$PDF: P(G = k) = (i - p)^{k-1}p, k = 1,2,..$$

$$CDF: P(G \leq k) = 1 - (1 - p)^k, k = 1,2,..$$

$$Mean: {}^{1}/_{p}$$

$$Variance: (1 - p)/p^2$$

$$a - percentile: \lceil \ln(1 - \mathrm{a})/\ln(1 - p) \rceil$$

p	0.01	0.02	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.75
90-percentile	230	114	45	22	11	7	5	4	3	2	2
95-percentile	299	149	59	29	14	9	6	5	4	3	3
99-percentile	459	228	90	44	21	13	10	7	6	4	4

Under the Geometric random model, once an affirmative behavior (success) is encountered the criterion is to ***wait for 99-percentile trials*** for the next affirmative behaviors to appear and if it does not then to deduce that the user is not the original. Different alarm levels could be defined at different numbers of consecutive suspicious behaviors encountered.

We propose to consider a starting success probability of 0.5 and perhaps after the 20th or 40th trial to use the sample estimate of the success probability –proportion of the number of affirmative behaviors. This is for further thought –to be validated from experimentation.

This approach is prone to ***false negatives*** since it uses the 99-percentile value (of a certain distribution) in a strict manner, as a deterministic upper bound. There is a 0.01 (1%) probability for a success to appear after the 99-percentive value and as such, in 1 or 2 times, on average, over 100 or 200 trials this criterion may not be accurate.

Nevertheless, we propose to apply the proposed 99-percentile criterion for the first 200 trials until starting applying the misbelief criterion specified in the previous section.

The following points are worth-making:

The misbelief criterion may remove the false negatives pertinent to the Geometric random model since the 99-percentile criterion may be fulfilled whereas the misbelief one may not. That is, the trials to wait for the next affirmative behavior are allowed to exceed the 99-percentile value as long as the misbelief level remains below a set threshold.

At the same time, the misbelief criterion may be more agile than the 99-percentile criterion since a (very) rare behavior may be encountered before 99-percentile trials are passed after the encounter of an affirmative behavior –see previous discussion on making the suspicious space thinner.

Last, the Geometric random model provides useful insights for forming thinner suspicious behavior spaces in applying the misbelief criterion so that to be more agile in spotting significant misbelieves through rare behaviors. It specifies the least number of trials before seeing an event with certain anticipated probability e.g. at least 228/459 trials for events with likelihood 2%/1%. This can be used in determining when to make 'splits' in the suspicious behavior space. For example, if a suspicious behavior space is at 2%/1% likelihood at the Nth trial, it cannot be made autonomous (be split) unless 228/459 trials have passed.

Alternative Models

The repeated authentication process can also be seen as a ***chain*** (discrete time, discrete state stochastic process) alternating among a given number of states corresponding to aggregate behaviors. Under such a model, we should distinguish between 'expected' and 'unexpected' transitions or paths and accordingly set suitable. This approach is left for further study.

Change of Vectors of Frequencies over Trials

Consider a number of trials over which a number of well-defined alternative events $\{A_1, .., A_E\}$ may appear. We are interested in seeing how the vectors of their frequencies change from trial to trial.

Supposing that by the Nth trial, the vector of event frequencies is:

$$\underline{f} = \left(\frac{k_1}{N}, .., \frac{k_E}{N}\right)$$

where $k_i, i = 1..E$ is the number of occurrences of event A_i by the Nth trial; $k_i \leq N$, $\sum k_i = N$, then at the $(N+1)th$ trial the vector of event frequencies will become:

$$\underline{f'} = \left(\frac{k_1}{N+1}, \ldots, \frac{k_j+1}{N+1}, \ldots, \frac{k_E}{N+1}\right),$$

assuming that the event $A_j, j = 1..E$ appeared at the $(N+1)th$ trial.

Then, we have:

$$\left\|\underline{f}-\underline{f'}\right\|_1 \triangleq \sum_{i=1}^{E}|f_i - f_i'| =$$

$$= \sum_{i\neq j}\left|\frac{k_i}{N}-\frac{k_i}{N+1}\right| + \left|\frac{k_j}{N}-\frac{k_j+1}{N+1}\right| =$$

$$= \sum_{i\neq j}\frac{k_i}{N} - \sum_{i\neq j}\frac{k_i}{N+1} + \frac{k_j+1}{N+1} - \frac{k_j}{N} =$$

$$= \frac{N-k_j}{N} - \frac{N-k_j}{N+1} + \frac{k_j}{N+1} + \frac{1}{N+1} - \frac{k_j}{N} =$$

$$= \left(1 - \frac{N}{N+1} + \frac{1}{N+1}\right) - 2\left(\frac{k_j}{N} - \frac{k_j}{N+1}\right) =$$

$$= \frac{2}{N+1} - \frac{2\,k_j}{N(N+1)} =$$

$$= 2\frac{N-k_j}{N(N+1)} = 2\frac{1-f_j}{N+1} \leq$$

$$\leq \frac{2}{N+1} \leq \delta, \delta \in (0,1)$$

And so, if

$$N \geq \frac{2-\delta}{\delta}$$

then

$$\left\|\underline{f}-\underline{f'}\right\|_1 \leq \delta, \delta \in (0,1)$$

Indicative values of δ's and N's are depicted in the following:

If N ($\geq$)	9	13	19	39	199
Then δ ($\leq$)	0.2	0.15	0.1	0.05	0.01

Note that the above express a sufficient condition for the absolute total changes (first-norm distances) to be within certain accuracy, not a necessary condition meaning that the desired accuracy could be achieved in smaller number of trials.

Also note that the above refer to absolute values thus they provide no clue what so ever whether the total changes will be in the positive or the negative coordinate directions.

4.5 HTTP REST API

All BAAs expose a similar API that allows the user to initiate the process of authentication verification and retrieve the current authentication status. Each API endpoint is started with the BAA identification

(i.e., <baa>), that specifies which of the four available BAAs is accessed (i.e., baaweblog, baamobility, baagait, baakeystroke).

HTTP GET <baa>/status/<user_id>

Returns a JSON with the authentication status info of the specified user_id.

Example:

Request:

curl http://127.0.0.1:9000/baagait/status/1

Response:

{ 'user_id': 1, 'status': AUTH, 'timestamp': 1488379464 }

Note: if the user_id is not authenticated, the ‘status’ is NONAUTH. If there are not enough data to determine authentication status, the ‘status’ is UNKNOWN.

HTTP GET <baa>/verify/<user_id>

Triggers verification process for user_id and updates the authentication status of the specified user_id.

Returns a JSON with the current timestamp.

Example:

Request:

curl http://127.0.0.1:9000/baagait/verify/1

Response:

{ 'timestamp' : 1488379464 }

Note: this specific request does not apply to the Keystroke dynamics BAA.

HTTP POST <baa>/upload

Only available in BAAs that need to upload data from a mobile app (i.e., baagait and baakeystroke). In the request body it must be specified the file to be uploaded, as multipart type.

Example:

Request:

curl -i -X POST -H "Content-Type: multipart/form-data" -F 'file=@gait-data-example.txt' http://127.0.0.1:9000/baagait/upload

Response:

{ 'timestamp' : 1488379464 }

*Note: the Keystroke BAA uses the following call instead of the "**<baa>/upload**".*

/api/session

Consumes a JSON object that contains the keystrokes of a session. Notifies the ID consolidator (IDC) if an unusual behavior is noticed. The JSON object should look like:

```
{"session_id": "e74dc7a5d911567b43d14c52f197cd", "device_id": "xxx", "scale_factor": "normal", "density_class": "XXHDPI", "app_id": "com.ogden.memo", "input_type": "search_form(example)", "orientation": "Portrait", "KeystrokesList": [[{"keyCode": -5, "keyDownToKeyDown": 0, "keyDownToKeyUP": 0, "keyUptoKeyDown": 0, "keyUptoKeyUp": 0, "keydownTime": 1478773093970, "keyupTime": 1478773093976}, ...], ...]}
```

REST API, supported methods: `POST`

Request: `POST /api/session`

Response: `OK 200, {alert: true/false}`

Notes: In the case of the keystroke BAA, the above call is used instead of the "upload" one. After uploading a session the system performs a verification and produces a matching score that is translated to an "alert" or not. This information is in the response of the call in order to be potentially used by the device in the future. At the same time, in case of unusual behavior (alert= true) an alert is sent to the IDC as well in order for the locking procedure to be triggered.

/api/device/<id>

Accepts request to delete data for the specified device id (i.e. retrain).

REST API, supported methods: `DELETE`

Request: `DELETE /api/device/<id>`

Response: `OK 200, {}`

Note: deviceID is used since we should be able to delete data for a specific device of a user and lot all data.

4.6 Integration with OpenAM

In several situations the different components discussed in this document require authentication and authorization in order to work together. For example, in case a BAA mobile app wants to send gait information, the solution must make sure that this information is stored in the right user account. For this purpose, the user will first authenticate the mobile app using an OpenID Connect authorization

request. The mobile app can then retrieve an access token. Each request (e.g. to store gait information) to the back-end servers then needs to be accompanied by this access token; as described by OpenID Connect and OAuth2. The back-end servers will have an authorization layer that will verify each request and link valid access tokens to the right user.

For *authentication*, the solution makes use of the standard OpenID Connect protocol. This approach is generic for both mobile as well as web applications. In case of behavioral verifications, Service Providers will be able to retrieve the BAA status using standard protocols.

4.6.1 Authorization of application to back-end server

In many occasions, the solution will provide back-end services that expose API calls to mobile applications or other entities. For example:

- BAA mobile applications might want to send behavioral information
- Service Providers might need the capability to update information in an Identity Provider. These API calls are protected via OAuth2.

A generic flow is depicted in Figure 11.

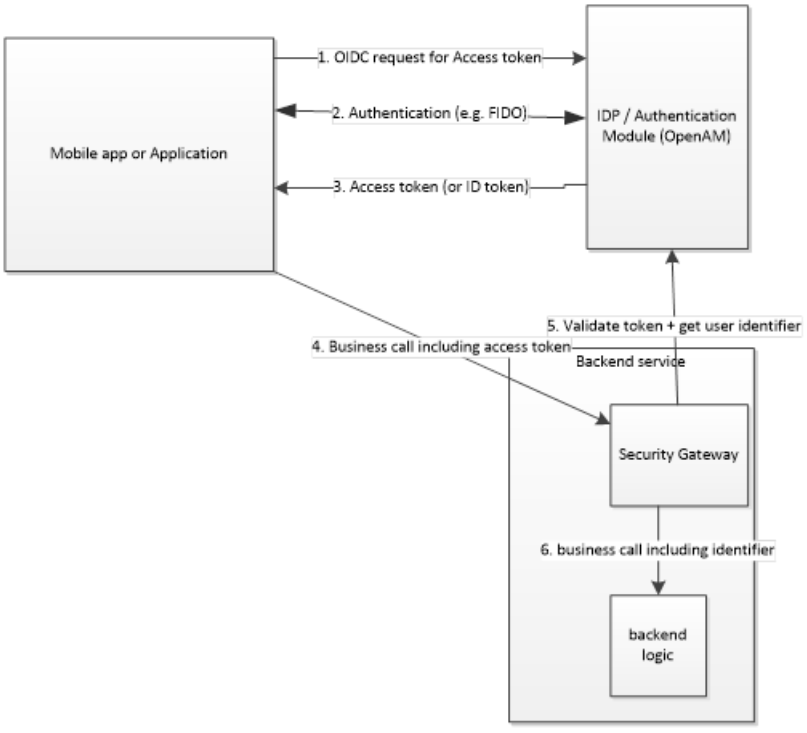

Figure 11: OAuth2 Authorization flow

1. When a mobile application or application wants to launch a business call to some back-end service of another entity, it first needs to have an access token. Access tokens can remain valid for a longer period of time. If the mobile app or application

already has one; the business call of Step 6 can be executed immediately. Otherwise, to obtain an access token, a standard OpenID Connect request is used; for example: http://consolidator.recred.eu:5000/openam/oauth2/authorize?response_type=code token&client_id=openidtestclient&realm=%2F&scope=address%20email%20phone%20openid%20profile%20change_phone%20recredid&redirect_uri=http%3A%2F%2Fconsolidator.recred.eu%3A5000%2Fdemo%2Fcb-basic.html&state=af0ifjsldkj

For each use case, the scopes of the request can vary. Scopes allow indicating what types of access a Service Provider needs. For example, in the context of a mobile app that wants to store gait data, the authorization layer just needs to find out the common name (cn) of the user. This corresponds to scope “cn”. If a Service Provider wants the user consent to update attributes on another entity, it can request specific scopes such as “change_language”, “change_phone” or “change_email”.

2. The Identity Provider that receives the authentication request now needs to know the identity of the end user. In order to do this, the solution will initiate FIDO authentication. This protocol is described in other deliverables.
3. When authentication is successful, the Identity Provider sends back some information about the authentication. The most important part for this response is the access token; other information includes session information (e.g. lifetime). The access token is now stored in a secure way. As per OpenID Connect specifications, an access token can have an expiration time. Until this time, the access token remains valid. Many calls can be made using the same access tokens. The solution implemented can be configured to set the right expiration time for each situation. The solution also supports the concept of refresh tokens. These can be used to obtain new access tokens without a separate authentication.
4. The mobile app or application can now execute backend calls. As per OAuth2 specifications, it does this by adding an Authorization header to the business call; for example:
   url: http://baa.com/gait/<<userid>>
   method: POST
   header: "Authorization: Bearer <<access_token>>"
   Postfield: {<<behavioral data>>}
5. The business call is passed by the authorization layer. The authorization layer first gets information from the Identity Provider about the access token. This way, the authorization layer gets to know whether the token is valid (e.g. not expired). It also learns the requested scope/authorizations (e.g. change_language) and some user information (e.g. the common name).
   Based on the use case, different types of authorization can be scripted on the authorization layer. For example, for the BAA use cases, it is sufficient that the business call can be linked to a person. As such, the access token must link to a useable identifier for the back-end. In other cases, e.g. if the Service Provider wants to update an attribute; the authorization layer must validate that the appropriate scopes exist. If anything goes wrong, the authorization layer will respond with an HTTP error (error code 401).
6. If everything went well, the authorization layer forwards the business call to the back-end. The user includes any user information (e.g. identifier) to the back-end who can then execute the transaction on the right user object.

Verizon has implemented a security gateway (based on Forgerock’s OpenIG) that is reusable in several different settings in the project (e.g. on the IDC as well as BAAs). It abstracts the authorization and authentication logic from the back-end logic.

4.6.2 Authentication to the BAA

When a certain requesting entity (for example, a bank) wants to have extra verification about a user’s behavior, the standard OpenID connect protocol will be used. The user will be redirected to the BAA’s Identity Provider. If there is no current session, the user will have to authenticate using FIDO. The Identity Provider will first show a QR code (or a link in case a mobile device is used) to initialize the FIDO authentication. Next the ReCRED FIDO authenticator will execute authentication using fingerprint. Now the user is known to the BAA, the right information (status and timestamp) about the user can be looked up. This information is then sent to the requesting entity using standard OpenID Connect: the requesting entity will receive an access token, which can be used to query the OpenID Connect userinfo endpoint which will return the BAA status. Based on this information, the requesting party can take the appropriate business decisions.

Figure 12 shows the different steps for this use case. More detailed interactions on FIDO authentication are explained in other deliverables (e.g. Deliverable 7.3).

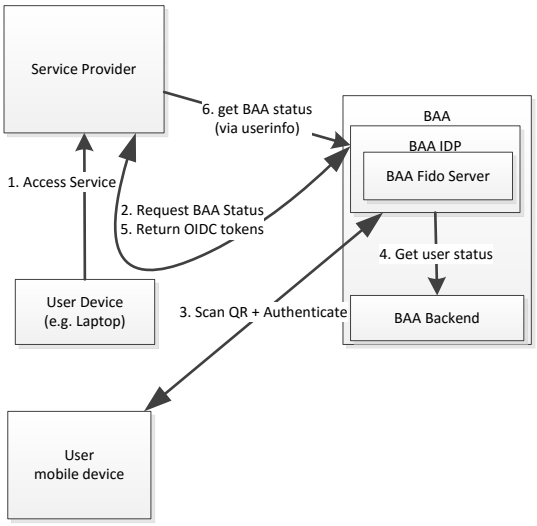

Figure 12: OAuth2 Authentication flow

1. The user accesses a Service Provider via his device.
2. The Service Provider needs more assurance of the user’s identity and redirects the user to the BAA using standard OpenID Connect.
3. The BAA IDP now needs to identify the end user and will initiate FIDO authentication. This is done by showing a QR code (or a link in case a mobile device is used). The user scans the QR code with the ReCRED Authentication app (or clicks the link). Using fingerprint, the user authenticates.
4. The BAA IDP can now get the user’s status from the BAA back-end.
5. The BAA IDP sends a standard OpenID Connect response to the Service Provider. The most important part of this response is an access token.
6. The Service Provider can now request the BAA status using the access token via the standard OpenID Connect userinfo endpoint.

5 QR Login

The QR Login module, described in the Deliverables 2.7 and 3.3, was used and integrated inside the Wi-Fi Pilot architecture. However, the use of QR mode within the Wi-Fi Pilot was made by introducing a number of code-level changes within the Relying-Party application. More specifically, it was used as a proxy for the Service Provider available within QR mode to keep the existing features. This approach was needed because in order to be able to authenticate the web application from laptop, the QR module had to update the LDAP directory (to obtain the necessary data from LDAP).

5.1 QR Login - Final Architecture

The final architecture where the QR module was used included the following components:

- mLogin Android Library (AAR) - which scans the QR code and sends it to the QR Authorization Server;
- A Service Provider that has been integrated into the Authorization Server. This component includes a generic view that defines the authentication process to a specific resource as well as a back-end that will be an extension of the REST interfaces within the Relying Party (RP);
- A QR Authorization Server which acts like a Service Provider and validates the QR received from the RP. The validation of the QR code is realized based on a previous request received from the Service Provider back-end (the service the user is trying to log in).

Unlike the implementation that is proposed in Deliverable 3.3, the method that involves authenticating the phone to the QR server has been introduced into the proxy of the Relying-Party. RP check to see whether there is an authenticated session opened with the Android terminal. If so, the RP forward the request to the QR Authenticator Server in order to validate the received data. After the success message is received from the QR Authorization Server, the RP performs the additional step in order to authenticate the desktop (i.e., give access to some resources). A more detailed version of this schema can be found in Deliverable 7.3, which provides adequate details regarding the use of the QR Authenticator module in the Wi-Fi pilot.

5.2 QR REST API

As mentioned above the entire QR authentication module is composed by two REST servers written in Java and a mLogin Android library. The API provided by these components are listed and described below. The third component (i.e., the Service Provider) is optional and can be either modified (by extending the default implementation provided) or implemented by scratch. The REST API for all three components is being showed below.

5.2.1 QR Authorization Server

POST /login

Description: Performs the authentication of the desktop given the QR code received from a mobile device. The QR server acts like a Service Provider in this situation and doesn't make any authentication for the request received from the Android device.

REQUEST

```
POST /login
Accept: application/json

{
    version : 1.0 ,
    qrCode  : "238437463746786786428764742 4264",
    cryptoCode : null,
    extensions : {
       extension:"some value",
       value: "some value"
   }
}
```

RESPONSE EXAMPLE

```
200
Content-Type: application/json

{
  "some session_id that is authenticated"
}
```

Request Body: In the request body we can ignore the crypto code and the extensions field. The other fields are mandatory and will be validated on the server-side. If there is an error for the data conversion, then a *Fail to convert* error message will be thrown.

Response Body: The response in case of success represents the *sessionId* value that was authenticated by the QR server

GET /getQrCode

Description: Returns a JPEG image based on some information received from the Service Provider. The validity of the QR code is 30 seconds by default

REQUEST

```
GET /getQrCode?randomAccess=some_random_string
Accept: application/json
```

RESPONSE EXAMPLE

```
200
Content-Type: image
```

Request Parameters: The GET request contains a unique random identifier that has been used to differentiate the QR codes. An additional parameter also is added by the Javascript code when refreshing the QR code, in order to force the server to retrieve another image.

Response Body: The response in case of success represents the JPEG encoding of the QR code.

POST /sendSessionData

Description: This method is called by the Service Provider to send to the QR server unique identifiable information about the user that wants to be authenticated. In the most common scenario this is the id of the session that wants to connect to the Service Provider. If the authentication succeeded then this unique information will be sent back to the Service Provider on the callback function.

REQUEST

```
POST /sendSessionData
Accept: application/json

{
    version : 1.0 ,
    policy  : null,
    userSessionId : "session id",
    extensions : {
       extension:"some value",
       value: "some value"
   }
}
```

RESPONSE EXAMPLE

200
Content-Type: text/plain
 	
{
“SUCCESS”
}

Request Body: The Service Provider must send at least the version and userSessionId in order to be able to fulfill the protocol. However, it may enforce the QR to adapt some default properties (eg: the validity of the QR code) by sending a custom policy to the server. The QR Server also must implement (accept) these policies in order to modify those properties.

Response Body: The response in case of success represents the text message *SUCCESS*.

5.2.2 mLogin Android Library

In order to use the mLogin library for scanning the QR code the following Intent must be created.

```
Intent i = new Intent(this, QRActivity.class);
startActivityForResult(i, REQUEST_QR_SCAN);
```

After the scanning process succeeded then the QR code is retrieved to the user in order to send the code to the QR Authorization Server (or to a proxy server that check the authenticated connection) the following payload should be created).

```
private void processQRCode(String code) {
    QRDecoder qrDecoder = new QRDecoder(code);

    AuthenticatorPayload payload = new AuthenticatorPayload();
    payload.version = 1.0f;
    payload.cryptoCode = "null";
    payload.qrCode = qrDecoder.getQRCode();
    payload.extensions = new Extension[0];

    Log.d("SENDING QR Code", "processQRCode:  " + payload.qrCode);

    String res = Curl.postInSeparateThread(Constants.endpoints.get("qrURL"),
            "Content-Type:application/json", new Gson().toJson(payload),
this);

    Toast.makeText(this, res, Toast.LENGTH_LONG).show();
}
```

5.2.3 QR Service Provider Interface

In order to be able to use the QR Authorization Server, the Service Provider must implement three functions:

- onConnect – What the Service Provider does when the user try to connect to the Service Provider;
- onSendSessionData–What information about the user is being sent by the Service Provider to the QR Authorization Server. This usually is the session id, but the SP may choose to send another information that identifies the connection with the user;
- onLogin – What the Service Provider does when the user tries to login (in this architecture this process is an repetitive process).

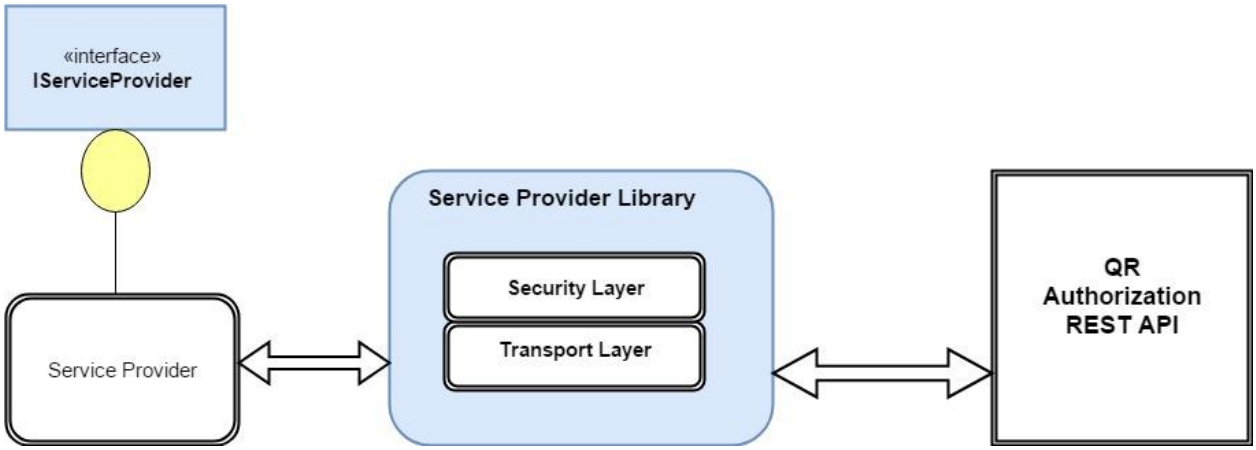

Figure 13: QR-Login – Service Provider interface

In Figure 13, the *Service Provider* implements the *IServiceProvider* interface which contains those three methods listed above. Also, the Service Provider can use a generic library which basically contains a Security Layer (which encodes the data in JWT encoding) and a Transport Layer (which sends the data to the QR Authorization Server. If the Service Provider does not use this library then it should implement by scratch the Security Layer (using another way of signing the data) and Transport Layer.

6 Account Locking

The ReCRED framework provides an additional security measure to protect user accounts from illegitimate login attempts. An Account locking mechanism allows to temporarily lock/unlock multiple third-party accounts belonging to a user.

In previous deliverables, this Account locking has been referred to as "Latch", since a similar product of ElevenPaths exist (https://latch.elevenpaths.com/) and we planned to integrate with ReCRED. However, the Latch framework lacked some important functionality we wanted to implement and therefore we decided to develop a similar, but lightweight and more flexible functionality.

Account locking mechanism does not replace identity management at Online Service. Each third-party service implements its own mechanisms to authenticate users. Atop of this mechanism, Account locking provides an additional security layer: after a user enters his credentials in the Online Service

login page, it queries ReCRED's Identity Consolidator to check the status of the user account. Each user account has a global status label – "locked" or "unlocked" – that defines whether the Online Service should accept authentication attempts for that particular account or not. If an account has the global status label set to "locked", the user will not be authenticated even if he entered valid credentials. The obvious advantage of such approach is that even if the login credentials have been leaked, an adversary will not be able to enter user's account.

Another advantage of Account locking mechanism in ReCRED framework is that the change of global status label for user accounts may be set to "locked" when the Identity Consolidator detects suspicious activity on the user accounts.

6.1 Account locking principle

Compared to previous deliverables, the Account locking mechanism has undergone a restyling that led to a leaner design. In particular, the current account locking mechanism is an integral part of Account Management Module (AMM) inside an Identity Consolidator and, as such, delegates to the AMM all the functionalities to add/remove accounts to the set of accounts of a user, to authenticate requests on behalf of service providers, to authenticate status updates originated at BAAs, and to expose a structured API through the third-party API defined in Deliverable 4.1. The following paragraphs describe the functionality of the account locking mechanism.

Check status

To check the status of a particular user's account when the user tries to log in the Online Service, the steps depicted in Figure 14 are executed. Note that in this figure and in subsequent ones, Online Service may refer to Service Providers, Identity Providers, or any other online entity that handles user authentication. Also note that the figure depicts the login to the online service by means of loginID and password, but the account locking mechanism is agnostic to the actual authentication means (e.g., FIDO).

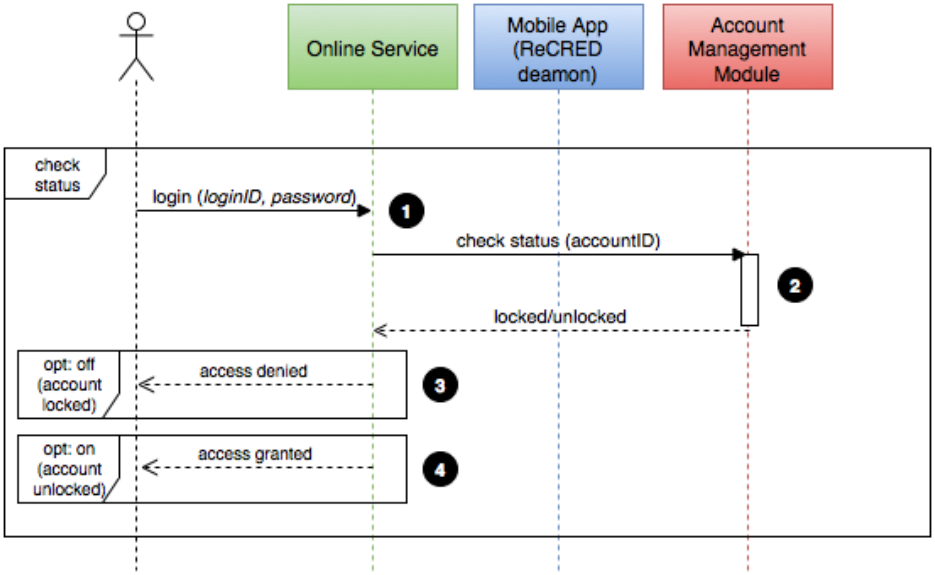

Figure 14 Check status message flow

First, the user enters his username and password in the Online Service web page (1). The Online Service knows (from previous registration) the accountID of the user. It sends a check status request to the AMM (2) that looks up the Identity Repository within ID consolidator for the status of the global status label for that particular account. The state is returned in the response. Depending on the state, two options are possible: if the account is locked (3), the Online Service stops the login process and denies user the access to the web page. If the account in unlocked (4), the Online Service grants access to the user and his login attempt is successful.

Change status

The status of user's account can be changed by the AMM based on the information provided by Behavioral Authentication Authorities (BAAs). Figure 15 depicts the process.

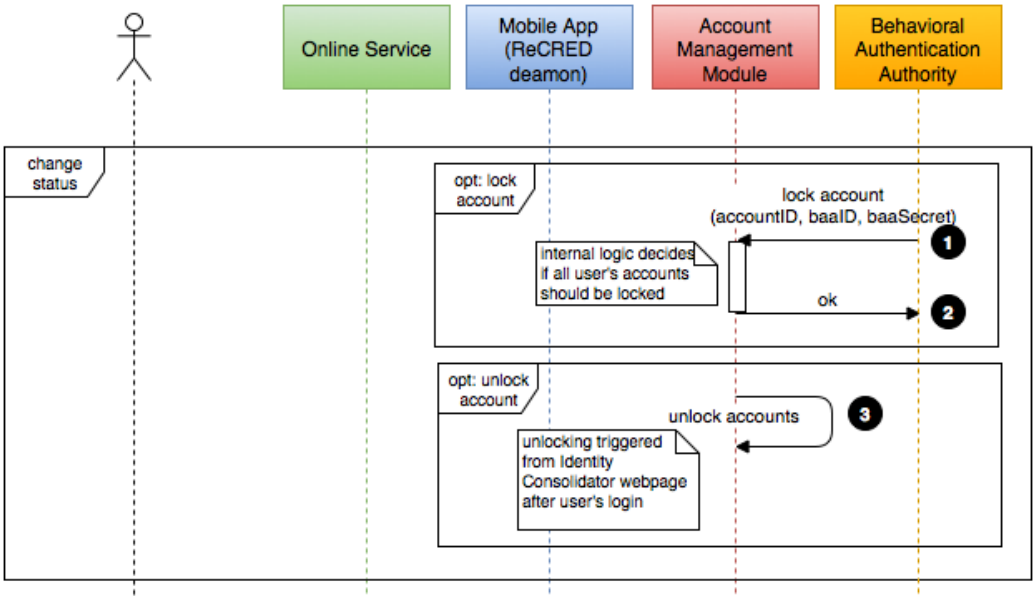

Figure 15 Change status message flow

To change the status of user's account a call to AMM's has to been established (1). Behavioral Authentication Authority calls the Locking mechanism API, authenticated with baaID and baaSecret, obtained during registration of BAA with user's account. The AMM returns an empty answer (2) and internally it stores the BAA's request for locking the account.

The logic which drives user's accounts locking could be very simple, such as "lock the account when a request comes" or more elaborated "lock the accounts when two BAA's request locking of user's accounts within one hour".

To unlock his account, the user has to visit an Identity Consolidator web page, login and there he can trigger the unlock action. This translates to calling the Account locking API and sending a request for unlocking accounts.

6.2 Account locking in ReCRED architecture

The Account locking mechanism is based purely on exchange of messages over a REST API. The Account Management Module (AMM) provides the Account locking functionality. The Account Management Module, an integral part of Identity Consolidator is the component that communicates

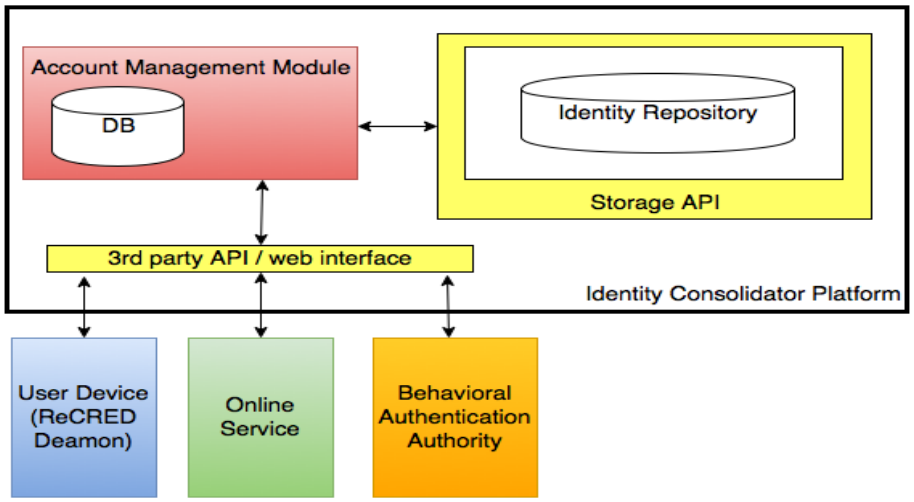

Figure 16 Account locking architecture

with Online Services, Behavioral Authentication Authorities and Identity Consolidator storage. Figure 16 shows the role of AMM together with all entry points and messages from different parties. Please note that the Identity Consolidator Platform consists of numerous modules, which are not depicted because are not related to the Account locking mechanism.

The AMM communicates with the outer worlds through a 3rd party API, which is a REST API defined in Deliverable 4.1.

Internally, user-related information such as 3rd Party Accounts are stored in the Identity Repository. As defined in Deliverable 4.1, the Identity Repository stores information about account address, account username and the status of the account (locked, unlocked).

For the Account locking operation, the AMM stores in its own database information about all calls to the API, which are related to the Account locking functionality. The purpose of this is twofold: First, it serves to decide whether to lock a user's account when a BAA suggests to do so. This depends on the desired logic and is part of AMM implementation. Second, information about locking and unlocking attempts can be presented to the user in a form of history listing. This is useful for better control over the Account locking functionality.

The communication between the AMM and the Identity Repository happens over a Storage API defined in Deliverable 4.1.

6.3 API description

The following paragraphs detail the Account locking API.

PUT /lock/<id>

Definition: Locks the specified Service/Identity Provider. If the provider_type is ´ACCOUNT´ and the requester_type is ´BAA´, then all Service/Identity Providers associated with that user account (i.e., with that <id> ==`Identifier` in Service_Provider_Users/Identity_Provider_Users tables) will be locked, whenever proceeds.

Request body:

{ ´provider_type`: *<provider_type>*, ´requester`: *<requester_type>* }

with:

<provider_type> in [´IDENTITY´, ´SERVICE´, ´ACCOUNT´]

<requester_type> in [´USER´, ´BAA´, ´POL´]

Response:

- HTTP 200: if the specified Service/Identity Provider exists and the requester is valid
- HTTP 404: if the specified Service/Identity Provider does not exists
- HTTP 400: if the request body is not valid.

PUT /unlock/<provider_id>

Definition: Locks the specified Service/Identity Provider

Request body:

{ ´provider_type`: *<provider_type>*, ´requester`: *<requester_type>* }

with:

<provider_type> in [´IDENTITY´, ´SERVICE´]

<requester_type> in [´USER´, ´POL´]

Response:

- HTTP 200: if the specified Service/Identity Provider exists and the requester is valid
- HTTP 404: if the specified Service/Identity Provider does not exists

- HTTP 400: if the request body is not valid.

PUT /nolock/<provider_id>

Definition: Sets the User Locking state of the specified Service/Identity Provider to a “none” status. After a valid call, the current status will depend on either BAA or Custom Locking Policies.

Request body:

{ ´provider_type`: *<provider_type>* }

with:

<provider_type> in [´IDENTITY´, ´SERVICE´]

Response:

- HTTP 200: if the specified Service/Identity Provider exists and the requester is valid
- HTTP 404: if the specified Service/Identity Provider does not exists
- HTTP 400: if the request body is not valid.

GET /status/<provider_type>/<provider_id>

Definition: Retrieves the current status (locked/unlocked) of a given Service Provider

with:

<provider_type> in [´identity´, ´service´]

Response:

- HTTP 200: if the specified Service/Identity Provider exists and the requester is valid. The body contains a JSON

{ 'data': {'status': *<status>*} }

With *<status>* in [‘locked’, ‘unlocked’]

- HTTP 404: if the specified Service/Identity Provider does not exist

POST /logs

Definition: Retrieves a list of lock/unlock actions performed so far, filtered by optional fields in the request body.

Request body:

{ ´id`: *<int>*, ´provider_type`: *<provider_type>*, ´older_than`: *<time>*, ´newer_than`: *<time>* }

with:

<provider_type> in [´IDENTITY´, ´SERVICE´]

Response:

- HTTP 200: if the specified Service/Identity Provider exists and the requester is valid. The body contains a JSON with a list of objects including StorageAPI’s Latch_Interaction_Logs table fields.

6.4 Locking status functionality

The current status of a given Service/Identity Provider is always explicitly specified in a single field called `locked` (Locked if True and Unlocked otherwise) in the corresponding Service_Provider_Users/Identity_Provider_Users table of the StorageAPI:

`locked` tinyint(1) DEFAULT '0',

Besides the above field, there are some additional fields that support the locking status flowchart in each Service_Providers_Users /Identity_Providers_Users table of the StorageAPI:

`userLocked` enum('NONE','LOCK','UNLOCK') DEFAULT 'NONE',

`baaPolicy` tinyint(1) DEFAULT '0',

`baaLocked` tinyint(1) DEFAULT '0',

`polLocked` tinyint(1) DEFAULT '0',

There is a strict priority level in the computation of the current status of any Service/Identity Provider:

Highest Priority | *User > BAA > User-Defined Locking Policy* | ***Lowest Priority***

The `userLocked` field specifies whether the user has explicitly chosen to lock/unlock a given provider (values 'LOCK','UNLOCK'), or whether it relies in a lower agent in the above priority chain. By default, that is if no agent says the opposite, any Service/Identity Provider is unlocked.

The `baaPolicy` field specifies whether the user has allowed the BAA to lock a Service/Identity Provider. Whenever a BAA sends an alert indicating to lock a Service/Identity Provider this alert is ignored if `baaPolicy` field is set to False.

The `baaLocked` field specifies whether the current ‘LOCK’ value of the ‘userLocked’ field is due to a prior BAA alert. Based on the above, in the following subsections it is described the Lock/Unlock functionality.

6.4.1 LOCK

- A user wants to explicitly lock a Service/Identity Provider: he goes to the interface and presses the "lock" button. Note that the interface will only allow the user to do this if the Service/Identity Provider is not already locked. That is, the `locked` field must be set to false and the `userLocked` field with either 'NONE' or 'UNLOCK' values. This action ends up calling the Latch’s /lock API call, specifing {´requester`: ‘USER’} in the body. Then the Latch sets the `userLocked` field to ‘LOCK’ and ‘locked’ field to true.

- A BAA sends an alert: BAAs only communicate with the AMM whenever a suspicious behavior for a user account is detected. That is, BAA only communicates to lock, never to unlock. In such a case the AMM, depending on a logic, may decide to lock all Service/Identity Providers associated to that user account by calling the Latch’s /lock API call, specifing {´provider_type`: ‘ACCOUNT’, ´requester`: ‘BAA’ } in the body. Then, the AMM reads the baaPolicy field for every Service/Identity Provider associated to that user account and, if it is true (and the `userLocked` field is set to 'NONE') then locks the account by setting both the "locked" and the “baaLocked” field to true.

- A user-defined policy is triggered: (e.g., lock at night/ weekend) the daemon calls the AMM’s /lock API call, specifing {´requester`: ‘POL’ } in the body. Then the AMM will set the “locked” field to true only if the `userLocked` is set to ‘NONE’. In any case, the “polLocked” field is set to true.

In any case, whenever the “locked” field (i.e., current status) changes from false to true it is registered a log indicating the agent (i.e., USER, BAA, POL) that locked the Service/Identity Provider.

6.4.2 UNLOCK

- A user wants to explicitly unlock a Service/Identity Provider: he goes to the interface and presses the "unlock" button. Notice that the interface will only allow the user to do this if the Service/Identity Provider is not already unlocked. That is, the `locked` field must be set to true and the `userLocked` field with either 'NONE' or 'LOCK' values. This action ends up calling the Latch’s /unlock API call, specifying {´requester`: ‘USER’ } in the body. Then, the AMM sets the “locked” field to false and the “userLocked” field to ‘UNLOCK’. Since the current locked status could be raised by a BAA (and BAAs never explicitly unlocks) it is also set the “baaLocked” field to false.

- A user-defined policy is triggered: (e.g., unlock at night/ weekend) the daemon calls the AMM’s /lock API call, specifing {´requester`: ‘POL’ } in the body. Then the Latch will set the “locked”

field to false only if the `userLocked` is set to 'NONE' and `baaLocked` is false (recall that BAAs has higher priority than policies). In any case, the "polLocked" field is set to false.

In any case, whenever the "locked" field (i.e., current status) changes from true to false it is registered a log indicating the agent (i.e., USER, BAA, POL) that unlocked the Service/Identity Provider.

6.4.3 NOLOCK

The user may explicitly decide to lock or unlock a given Service/Identity Provider. However, there is a third case: removing any explicit user locking status and relying in lower priority agents (i.e., either BAA or User-defined Policies). There are a few cases in which this decision must be considered.

- A user wants to remove a prior explicit Locked status: Occurs when the user decides that he does not want to continue forcing a locked status and wants to rely on either BAAs or User-Defined Policies. In this case, the user presses the "None" button (referring to user explicit lock/unlock) and it is called the AMM's /nolock API call. Then, the AMM sets the "userLocked" field to "NONE" and checks whether it affects the current status (i.e., the "locked" field). Any prior BAA alarm is ignored (the "baaLocked" is set to false) and the AMM checks whether a User-defined policy was triggered. Thus, if "polLocked" field is true, the "locked" status is not modified (the "locked" field is already true). However, if "polLocked" field is false, that means that the only reason to remain locked the account was the user explicit decision. In that latter case the "locked" field is set to false.

- A user wants to remove a prior explicit Unlocked status: Occurs when the user decides that he does not want to continue forcing an unlocked status and wants to rely on either BAAs or User-Defined Policies. In this case, the user presses the "None" button (refering to user explicit lock/unlock) and it is called the AMM's /nolock API call. Then, the Latch sets the "userLocked" field to "NONE" and checks whether it affects the current status (i.e., the "locked" field). Any prior BAA alarm is ignored (the "baaLocked" is set to false) and the AMM checks whether a User-defined policy was triggered. Thus, if "polLocked" field is false, the "locked" status is not modified (the "locked" field is already false). However, if "polLocked" field is true, that means that the only reason to remain unlocked the account was the user explicit decision. In that latter case the "locked" field is set to true.

6.5 Implementation

The Account locking solution is being implemented in Python 2.7 programming language. The Account Locking Module (a.k.a. Latch) is based on Flask (http://flask-sqlalchemy.pocoo.org/2.1/), a microframework for Python based on Werkzeug (http://werkzeug.pocoo.org/). Flask provides a way to keep the web server core simple but extensible. Moreover, it allows for using an arbitrary database engine. Thanks to its extensibility, many design decisions could be made even in the later phases of development.

To communicate with database, the Flask SQLAlchemy (http://flask.pocoo.org/) is used as an abstraction layer between Sqlite database and the Flask core. It provides even possibility to implement advanced patterns in SQLAlchemy or another database tool and take advantage of framework-agnostic tools built for WSGI, the Python web interfaces.