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 D3.2 "Multifactor authentication for DCA: user-to-device and device-to-network support"

Editor(s):	Claudio Soriente (TID)
Author(s):	CNIT: Alberto Caponi, Tooska Dargahi, Claudio Pisa, Luigi Stammati CSGN: Bogdan Chifor, Ionut Florea, George Gugulea CUT: Charalambos Chrysostomou, Michael Sirivianos EXUS: Dimitris Katsaros TID: Jeremy Blackburn, Michal Ficek, Nicolas Kourtellis, Claudio Soriente UC3M: Rubén Cuevas UPCOM: Vangelis Bagiatis, Kostas Flokos, George Savvas UPRC: Faidon Lalagiannis, Christoforos Ntantogian, Nikolaos Vavoulas, Eleni Veroni, Christos Xenakis VERIZON: Arsalan Najwani, Bharadwaj Pulugundla WEDIA: Evangelos Kotsifakos, Michael Koulinas, Michael Pramateutakis, George Gonidelis
Dissemination Level:	PU - Public
Nature:	R
Version:	1

ReCRED Project Profile

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

Partners

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

Document History

Version	Date	Author	Remarks
0.1		Claudio Soriente (TID)	Initial Table of Contents
0.2		Claudio Soriente (TID)	Summary and Introduction
0.3		Claudio Soriente (TID)	Merging contribution from partners
0.4		Claudio Soriente (TID)	Revision after internal review
0.5		Claudio Soriente (TID)	Proofreading, minor changes
1.0		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 defines and describes authentication mechanisms based on behavioural and physiological biometrics that are used within the ReCRED framework. The deliverable also details the use of Mobile Connect and its integration within the ReCRED platform, so that service providers can offload user authentication to mobile operators. Finally, the deliverable also relates with the Identity Consolidator (WP4) and presents additional measures to prevent fraudulent access to online accounts, as well as techniques for ReCRED users to backup and restore their credentials.

The rest of the document is organized as follows. Chapter 1 introduces the concepts and the technologies presented in the deliverable. Chapter 2 describes user-to-device authentication based on face-recognition. Chapter 3 presents the first-factor authentication using FIDO. Chapter 4 introduces the authentication mechanisms that involve Behavioural Authentication Authorities. Chapter 5 and Chapter 6 present Mobile Connect and QR-Login, respectively. Chapter 7 explains how ReCRED implements the virtual lock on the user accounts while Chapter 8 presents the restore mechanism to be used in case the user loses her device and wants to enrol a new one.

Table of Contents

List of Figures

Table of Abbreviations

AMM	Account Management Module
API	Application Program Interface
APK	Android Package Kit
BAA	Behavioural Authentication Authority
BMI	Body Mass Index
CDR	Call Description Record
FIDO	Fast Identity Online
FLANN	Fast Library for Approximate Nearest Neighbour
GSMA	GSM (Groupe Spéciale Mobile) Association
HTTP	Hyper-Text Transfer Protocol
HTTPS	Hyper-Text Transfer Protocol Secure
IDP	IDentity Provider
JSON	JavaScript Object Notation
LBPH	Local Binary Patterns Histograms
MCX	Mobile Connect Accelerator
MNO	Mobile Network Operator
MSISDN	Mobile Station International Subscriber Directory Number
OIDC	Open ID Connect
REST	Representational State Transfer
SP	Service Provider
SSL	Secure Socket Layer
TF-IDF	Term Frequency-Inverse Document Frequency
UAF	Universal Authentication Framework
URL	Uniform Resource Locator

1 Introduction

ReCRED is committed to simplify online authentication for both users and service providers. On the one hand user burden should be reduced. On the other hand, service provider should enjoy strong user authentication while minimizing their infrastructure. This deliverable presents techniques that reach both goals.

In order to mitigate user burden when authenticating to the local device, ReCRED provides the technology for user authentication based on behavioural and physiological fingerprints. In particular, user-to-device authentication in ReCRED is based on face-recognition techniques (apart from the fingerprint-based technique provided by FIDO), so that users seamlessly log into their devices by simply looking at the device camera.

In order to minimize the authentication infrastructure at the service providers, ReCRED introduces several identity providers that offer authentication-as-a-service based on a range of features. Identity providers of ReCRED can provide FIDO-as-a-service and authenticate users based on FIDO authenticators. Furthermore, ReCRED also integrates the Mobile Connect authentication service that allows third parties to offload user authentication to Mobile Network Operators. ReCRED also introduces Behavioural Authentication Authorities (BAAs), i.e., identity providers of the ReCRED framework that authenticate users based on behavioural characteristics. BAAs of ReCRED will authenticate users based on gait, typing pattern, mobility pattern, or browsing behaviour. As such, BAAs transparently collect data and validate the purported user identity on behalf of service providers.

Service providers that participate to the ReCRED framework can, therefore, enjoy authentication-as-a-service, minimizing their own authentication infrastructure. Alternatively, service providers in the ReCRED framework can complement their own authentication mechanisms with the ones offered by ReCRED identity providers, thereby leveraging a richer set of authentication techniques and improving the security of the user accounts.

This deliverable also details two mechanisms integrated at the Identity Consolidator to increase the security of online user accounts and to improve the usability of the whole ReCRED framework.

Security of user accounts is improved by offering an account locking service. The identity consolidator keeps track of which user accounts are locked and provides this information to the service provider where the account is hosted. When an account is “unlocked”, login is granted based on the authentication mechanism enforced by the service provider. When an account is “locked”, the service provider does not allow login, even if the user presents the right login credentials. This feature allows service providers to improve the security of their user accounts even if the account credentials have been leaked.

Usability of the ReCRED framework is improved by implementing a profile backup/restore service at the Identity Consolidator. In case the user device is lost/stolen/destroyed ReCRED users must be able to re-establish connectivity with the ReCRED infrastructure using a new device. The backup/restore mechanism entails the re-registration of the user with a new device, and the restoration of the user’s profile (e.g., credentials, user settings etc.).

2 User to device authentication

2.1 Physiological biometrics

Humans have the inherent ability to easily distinguish faces and we use this ability in order to effectively recognize other persons. Therefore, it is only normal that face recognition has been used in biometric systems for the identification and/or verification of individuals. For this, most face recognition systems extract facial features and compare them against a database of existing facial templates. More specifically, a typical face recognition system contains the following modules:

- **Camera Module**: It is responsible for capturing a facial picture of an individual, which can be either used for identifying / verifying that individual or stored as a facial template.
- **Feature Extraction Module**: It is responsible for extracting the facial features out of a captured photo.
- **Matcher Module**: It is responsible for matching the extracted feature against one or more facial templates, based on some matching algorithm. A matching score is calculated and this score is used in order to confirm (verify) or establish (identify) the user's identity.
- **Database Module**: It is responsible for storing facial templates. It can store one or more templates of the same individual (for verification) or many templates of different individuals (for identification). According to the nature of the face recognition system, the facial templates can be stored on a central database (e.g. server) or on a local database (e.g. mobile device).

A face recognition system typically supports the following operations:

- An **enrolment operation**, during which facial features of individuals are extracted and stored on the system database as facial templates.
- A **verification operation**, during which the extracted facial features of an individual are compared against one or more facial templates of the same individual in order to confirm his identity.
- An **identification operation**, during which the extracted facial features of an individual are compared against a set of facial templates from many individuals, in order to establish his identity.

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, as part of user-to-device authentication. The solution is based on OpenCV[1] (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. An architectural overview of the fare recognition module is shown in Figure 1.

[1] http://docs.opencv.org/2.4/index.html

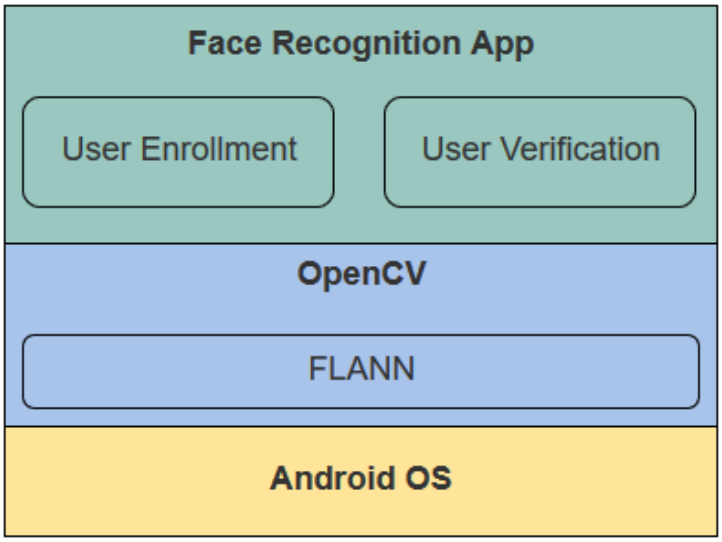

Figure 1 Face Recognition Module

The face recognition module supports two operations: the user enrolment (Figure 2) and the user verification (Figure 3). During the enrolment operation, the user actually trains the recognition engine, by taking a series of photos through the device camera. For each photo taken, the application preprocesses it (resizing, face alignment, etc.) and computes an n-dimensional feature vector to represent the face, based on the LBPH features. The feature vectors are then stored as facial templates on the device and the actual photos are deleted. All the feature vectors are stored on the device internal storage, which means that they are private to the application and cannot be accessed by the user or other applications, unless the device is rooted. If the user uninstalls the application all the stored vectors are deleted from the device. At least two facial templates are required in order for the user to be able to authenticate through the face recognition module. The user can retrain the recognition engine at any time, by creating new facial templates or deleting existing ones.

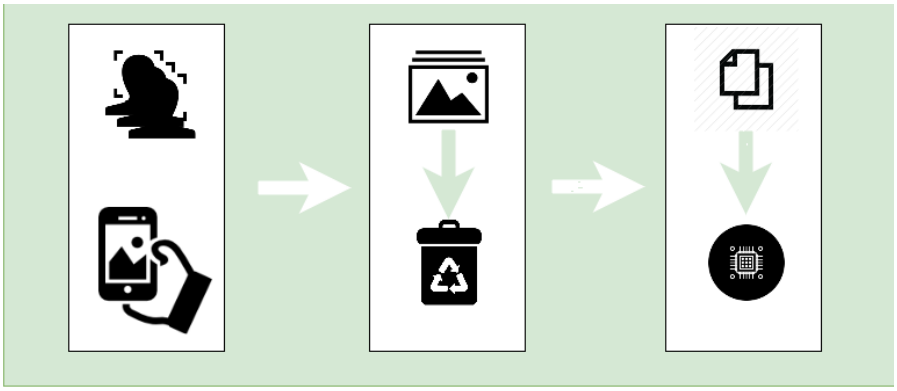

Figure 2 User Enrolment

During the verification operation, the application uses the device camera in order to detect a face. After a face is detected, the application pre-processes the photo and computes its feature vector, which is matched against all the stored feature vectors (facial templates). This matching is based on Fast Approximate Nearest Neighbour Search, using FLANN (Fast Library for Approximate Nearest Neighbours), an open-source library that is incorporated into OpenCV and contains a collection of

algorithms optimized for fast nearest neighbour search in large datasets and for high dimensional features. A matching score is then calculated (a number between 0 and 1 - the higher that number the most confident the application is of the user identity) and compared to a predefined threshold. If the matching score is above the required threshold, then the user identity is considered to be verified. No data is stored during the verification operation and both the actual photo and the feature vector are deleted after the verification is finished.

Finally, the verification operation is used in order to continually authenticate an individual on the device. A verification service is run every time the user unlocks his phone or after a predefined time interval. If no face is detected, a push notification is sent, through which the user can manually trigger a face recognition attempt. A successful face recognition attempt means that the user is authorized for a predefined time period.

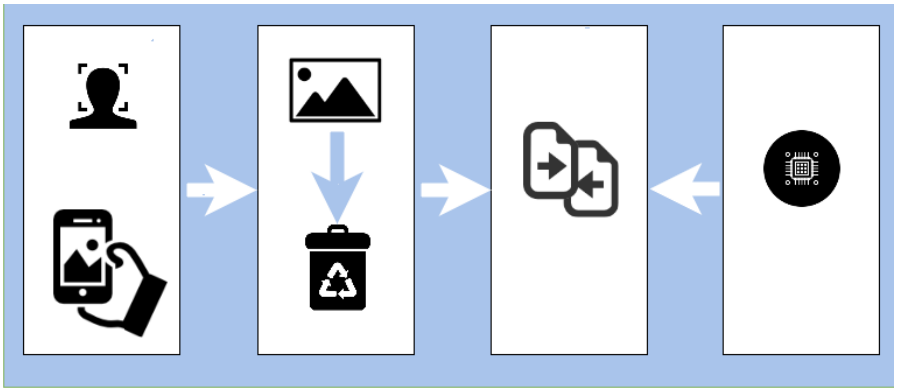

Figure 3 User Verification

2.1.2 Face Recognition Evaluation

A small scale experiment has been conducted in order to evaluate the face recognition module. The purpose of this research is to test the face recognition engine using a sample of users with different characteristics and under different conditions, so that any false positive and/or false negative errors are identified.

A **false positive** error is a result that indicates that an individual identity was verified by the face recognition engine, when it should not be verified (no face templates of that individual have been stored in the templates database).

A **false negative** error is a result that indicates that an individual identity was not verified by the face recognition engine, when it should be verified (there are face templates of that individual stored in the templates database).

Accordingly, **true positives** and **true negatives** indicate that an individual identity was justifiably verified or not verified by the face recognition engine.

In order to identify any false positive errors, many face templates are required, coming from different individuals. For this reason, we initially trained the database with a sample of different face photos

coming from 50 random individuals. Both males and females were included in this sample and for each individual at least two face templates were created, having a diversity of emotions (smile, laugh, frown, etc.) and features (beard, glasses, hats, etc.). After that, a set of 20 testers were identified, also with a diversity of gender and features (some testers were required to wear glasses / sunglasses and/or a hat during the verification attempts). The following table summarizes the characteristics of the tester samples.

Gender	Male	14 out of 20	70%
	Female	6 out of 20	30%
Special Features	Glasses	7 out of 20	35%
	Beard	6 out of 20	30%
	Hat	4 out of 20	20%

During the execution of the experiment, each tester was introduced to the face recognition application and then performed a series of verification attempts, under the guidance of a researcher who would ensure the proper execution of the verification attempts and log the results. For each tester (subject), the following steps were followed:

1. The subject executes a number of verification attempts against the default sample of facial templates (during this step the subject has not trained the face recognition engine with her own face). The purpose of this step is to discover any false positive errors. Since the subject has not trained the face recognition engine yet, any positive result should be a false positive. On the other hand, any negative result should be a true negative. We executed attempts under both good and poor lighting conditions, in order to evaluate the performance of the engine under different conditions. We also executed attempts using different matching score thresholds, in order to evaluate the engine tolerance level. More specifically:

    1.1. One verification attempt is executed under good lighting conditions, with the threshold set at 0.50
    1.2. One verification attempt is executed under poor lighting conditions, with the threshold set at 0.50
    1.3. One verification attempt is executed under good lighting conditions, with the threshold set at 0.75
    1.4. One verification attempt is executed under poor lighting conditions, with the threshold set at 0.75

2. The sample facial templates are erased and the subject captures two face photos, in order to train the face recognition engine. Now the facial templates database includes only templates of the subject.
3. The subject re-executes the verification attempts of the step 1, in order to discover any false negative errors. Since the subject has now trained the face recognition engine, any negative result should be a false negative. On the other hand, any positive result should be a true positive. Again, we executed attempts under both good and poor lighting conditions and using different matching score thresholds.

The following table summarizes the results of the face recognition evaluation.

	Good Lighting				**Bad Lighting**			
	.50 Threshold		**.75 Threshold**		**.50 Threshold**		**.75 Threshold**	
	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative
Before Training	0%	100%	0%	100%	20%	80%	0%	100%
After Training	100%	0%	85%	15%	60%	40%	35%	65%

As already stated, all the negative results **before** training are considered true negatives (indicated by a green background), whereas the positive results are considered false positives (indicated by a red background). Respectively, all the negative results **after** the training are considered false negatives (indicated by a red background), whereas the positive results are considered true positives (indicated by a green background).

The only false positives (20%) appear when the subjects used the app in poor lighting conditions and with the threshold set at 0.50. Therefore, we can confidently conclude that the engine does not produce any significant number of false positives, as long as the threshold is high enough (around .75).

On the other hand, the engine seems to produce a significant number of false negatives when the testers used the app under bad lighting conditions (40% with the threshold set at 0.50 and 65% with the threshold set at 0.75). Under good lighting conditions, no false negatives appeared when the threshold was set at 0.50 and only a small amount (15%) with the threshold set at 0.75. Even these few failed attempts were successful when the testers made some additional attempts, using different angles.

In conclusion, the face recognition engine seems to be most efficient while used under good lighting conditions, since poor lighting tends to hinder the successful verification of the subjects. Moreover, higher thresholds tend to produce less false positives but more false negatives, since the engine is stricter while trying to confirm the identity of the user. Lower thresholds, on the other hand, are more tolerant and, therefore, more successful on verifying the user identity, but may produce some false positives when used under poor lighting conditions.

3 Device to service authentication

3.1 First-factor authentication with FIDO

The FIDO protocol is used in the ReCRED context as first factor authentication (UAF – Universal Authentication Framework). By using public key cryptography, the FIDO protocol provides a passwordless authentication mechanism to an online service. The user selects a local authentication mechanism (biometrics, PIN and/or others) when registering to an online service and repeats this process for every login.

The FIDO software project contains three main modules: the FIDO Client (Android), the FIDO Server (runs on Apache Tomcat) and the FIDO core (a module which contains the FIDO logic, employed by both the client and the server).

The messages exchanged between the server side and the client side modules are JSON serialized.

Figure 4 FIDO Modules

In Figure 4 the FIDO modules are represented along with their deployment environments.

In order to ease the FIDO core and the FIDO Server testing, a command line FIDO client was developed. By deploying a command line FIDO client some tests can be automated after post-build in a continuous-integration environment.

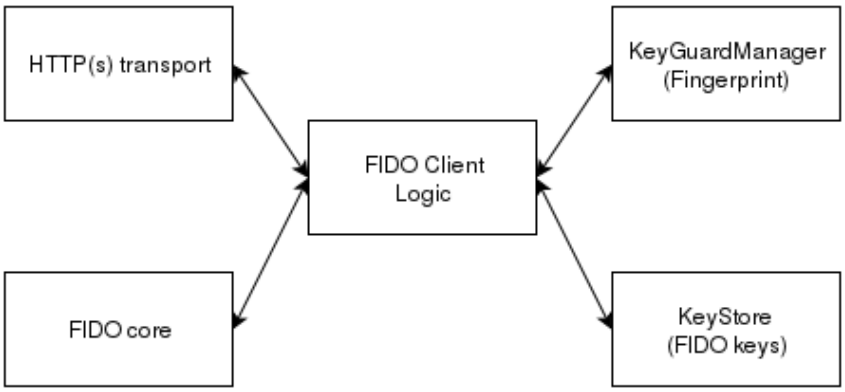

Figure 5 FIDO client

Regarding the structure of the client software modules, there are five main entities as presented in Figure 5.

The FIDO client comprises the following modules:

- FIDO Client Logic – is in charge of the FIDO protocol management on the client side.
- KeyGuardManager – is in charge of user authentication (fingerprint or phone pin). This module also comprises a series of user-interface elements.
- KeyStore – is in charge of key storage and cryptographic operations.
- FIDO core – is in charge of FIDO message computing.
- HTTP(s) transport – is in charge of exchanging FIDO messages with the server.

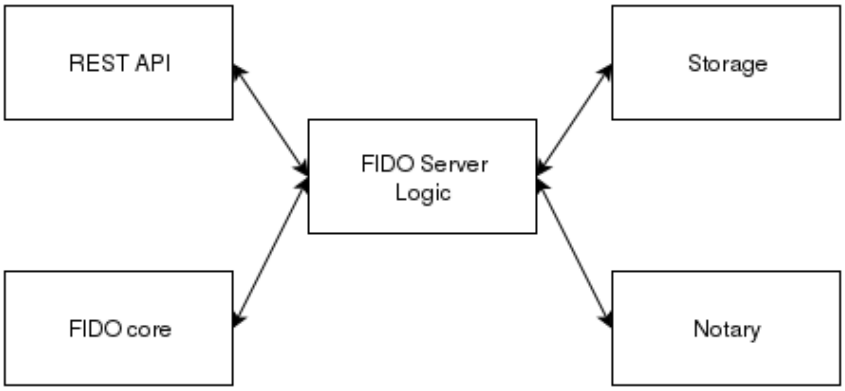

Figure 6 FIDO server

The server side module comprises the following main modules as presented in Figure 6.

- FIDO Server Logic – is in charge of the FIDO protocol management on the server side.
- REST API – is in charge of exchanging FIDO messages with the client
- FIDO core – is in charge of decoding and validating the FIDO messages
- Storage – is in charge of storing the account data
- Notary – is in charge of authenticating the messages

The FIDO client will be integrated into the ReCRED main application as a library project. By doing this, the integration process will be facilitated and a modular design will be obtained. From the user perspective the FIDO component will not be conceived as a different module.

3.1.1 Registration

The purpose of the FIDO registration process is the authentication key agreement between the FIDO server and the FIDO authenticator. The FIDO authenticator is the logic/physical component which stores the FIDO cryptographic keys used in the FIDO authentication process. The user has to unlock the FIDO authenticator (using biometrics or other methods) in order to use the keys stored on this module.

In order for a user to register to the FIDO server a REST call must be done as presented below:

GET /fidouaf/v1/public/regRequest/{username}

The FIDO server computes a registration request message which contains the following:

- An operation header. The operation header specifies a version, an operation type (in this case is **registration**), an application ID and a server data.
- A challenge. This field comprises a timestamp, a nonce generated by the server, the username, and a signature computed over the previous fields with the aid of Notary module.
- A username
- A policy. The policy contains an identifier of the authenticator ID (**AAID**) accepted by the server. The FIDO server currently permits the following AAID:

"EBA0#0001", "0015#0001", "0012#0002", "0010#0001",

"4e4e#0001", "5143#0001", "0011#0701", "0013#0001",

"0014#0000", "0014#0001", "53EC#C002", "DAB8#8001",

"DAB8#0011", "DAB8#8011", "5143#0111", "5143#0120",

"4746#F816", "53EC#3801"

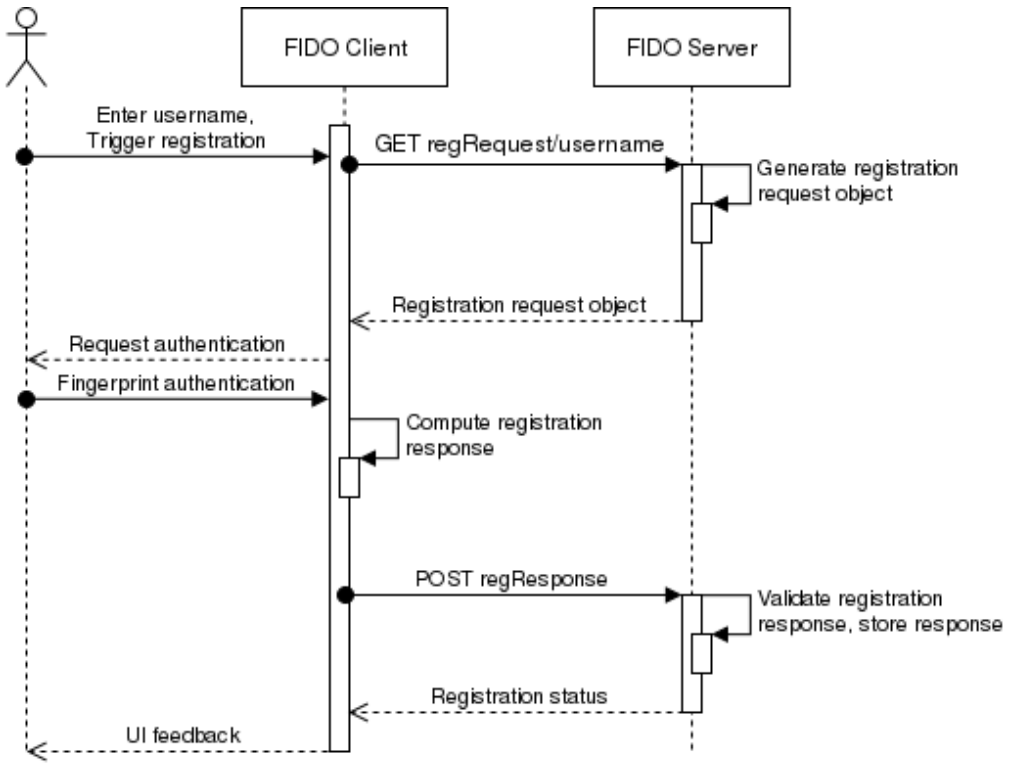

Figure 7 FIDO registration

The registration process is depicted in Figure 7. After the client receives the server generated registration request message, it deserializes it into a Java object in order to be validated. The first validation is performed on the application ID (AppID) field.

The purpose of the AppID is to dictate the audience of the cryptographic material generated during the registration process. If the server does not send an AppID, the client fills this header field with the facet ID of the application. In the FIDO concept an application facet represents an instance of the same application (Eg: an Android and a Web implementation).

If the facet ID does not match the AppID, then the client tries to fetch a trusted facet list from the server and verifies the facets against the facet ID. The trusted facet list is stored on the server side application and can be fetched as a web resource.

In order for the registration response to be completed on the client side, the user must authenticate to the device using the fingerprint or the phone pin. The authentication process is realized by employing the Android KeyGuardManager class. If the user is successfully authenticated the registration response is generated, along with a cryptographic key pair which will be used in subsequent authentications to the relying party.

On the Android client side the cryptographic key pair are generated using the java security KeyPairGenerator class.

In order to generate a registration response, the client computes the final challenge parameters as specified in the FIDO UAF Protocol Specification. The registration assertions are inserted in the registration response along with a basic full attestation. The basic full attestation requires the FIDO authenticator to sign a data with the private key corresponding to the attestation X.509 certificate. This certificate and the associated private key are stored on the FIDO authenticator side and have the purpose to prove to the Relying Party that the generated keys or other processes performed by the authenticator are genuine are are originating from a trusted and certified module.

During this process, the client generates a key ID (key handle) and sends the data to the server.

After the client computes a registration response message, it sends it to the server by making a REST call to the following URL “/fidouaf/v1/public/regResponse”. On the server side, the JSON serialized registration response is converted into a Java class which contains the following fields:

- Operation header
- fcParams
- An array of registration assertions. Each registration assertion consists of an assertion scheme and of an assertion value.

After the message is deserialized, each assertion is processed. If all the assertions are verified according to the FIDO protocol specifications, the registration response is stored in an in-memory storage structure (hash map), by using the following tuple: (AAID | keyID; Registration response).

By storing the whole registration response, the server module has access to the public key and to the key handle for later authentication processes.

The assertions are a byte array encoded in Base64. Each assertion is encoded using a Type-Lengh-Value (TLV) method, the following assertion types being used:

UAF_CMD_STATUS_ERR_UNKNOWN(1)

TAG_UAFV1_REG_ASSERTION(15873)

TAG_UAFV1_AUTH_ASSERTION(15874)

TAG_UAFV1_KRD(15875)

TAG_UAFV1_SIGNED_DATA(15876)

TAG_ATTESTATION_CERT(11781)

TAG_SIGNATURE(11782)

TAG_ATTESTATION_BASIC_FULL(15879)

TAG_ATTESTATION_BASIC_SURROGATE(15880)

TAG_KEYID(11785)

TAG_FINAL_CHALLENGE(11786)

TAG_AAID(11787)

TAG_PUB_KEY(11788)

TAG_COUNTERS(11789)

TAG_ASSERTION_INFO(11790)

TAG_AUTHENTICATOR_NONCE(11791)

TAG_TRANSACTION_CONTENT_HASH(11792)

TAG_EXTENSION(15889)

TAG_EXTENSION_NON_CRITICAL(15890)

TAG_EXTENSION_ID(11795)

TAG_EXTENSION_DATA(11796)

3.1.2 Authentication

The authentication process begin with a client's REST request to an authentication request URL, as follows.

```
GET /fidouaf/v1/public/authRequest
```

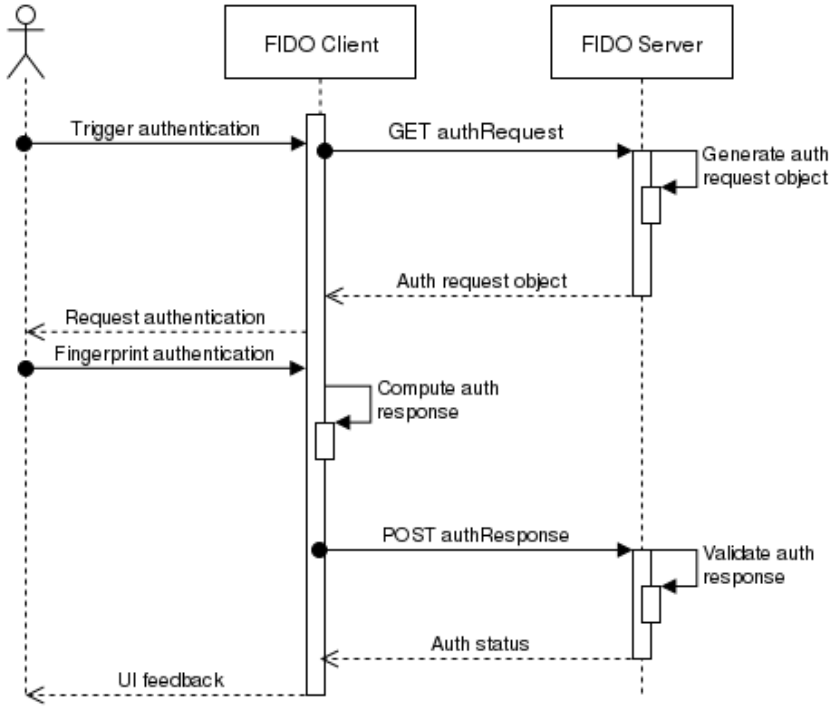

Figure 8 FIDO authentication

The FIDO authentication process is presented in Figure 8. The server generates an authentication request object which contains the following fields:

- Operation header. The operation header contains: the protocol version, the current operation (which is **registration**), an application ID and a server data. The application ID is the FIDO server application URL which returns the trusted FIDO facets (v1/public/uaf/facets).
  In the Android case, the FacetID must be a URI derived from the SHA-1 hash of the APK signing certificate [APK-Signing], such as:

  “android:apk-key-hash:Df+2X53Z0UscvUu6obxC3rIfFyk”. To compute this value, the Android *PackageInfo* class is used.

- A challenge. The challenge is a nonce generated by the server and signed using the Notary module.
- A transaction array. A transaction element contains the transaction content type and the transaction content.
- A policy. The policy contains an identifier of the authenticator ID (AAID) accepted by the server.

After the client receives the authentication request object it computes the authentication response according to the FIDO protocol specification.

In order for the client to generate the authentication response, the user must authenticate to the device in order to unlock the private key. The user authenticates to the device using the fingerprint or phone pin. This feature is realized by employing the Android KeyGuardManager class.

The authentication response generation process is triggered by a successful user authentication to the device. The client computes a signature over a server generated challenge using the private key generated during the registration process. The authentication response also contains a nonce (which is comprised in the signed data) generated by the client.

The authentication response assertions generated on the client side comprise the following tags:

TAG_UAFV1_SIGNED_DATA

TAG_AAID

TAG_ASSERTION_INFO

TAG_AUTHENTICATOR_NONCE

TAG_FINAL_CHALLENGE

TAG_TRANSACTION_CONTENT_HASH

TAG_KEYID

TAG_COUNTERS

TAG_SIGNATURE

After the client generates an authentication response it sends it to server by making a REST call to the following URL:

POST /fidouaf/v1/public/authResponse

The server verifies the server data and the assertions contained by the response. The server data verification comprises the timestamp validation (to detect expired authentication requests and thus mitigate some security attack vectors), and the authenticity of the challenge by employing the Notary module.

The server module extracts the device public key from the storage structure and verifies the validity of the signature contained in the authentication response assertions.

3.1.3 Deregistration

The purpose of the deregistration module is to delete the account related material from both server and client side authenticator.

The client initiates the deregistration process by making a REST call to the following URL:

POST /fidouaf/v1/public/deregRequest

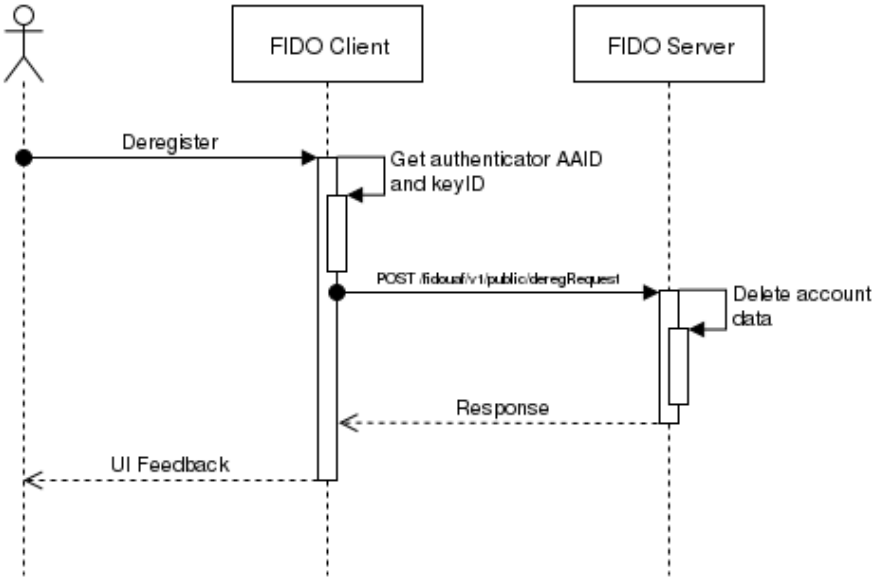

Figure 9 FIDO deregistration

The implemented FIDO deregistration process is depicted in the Figure 9. On the server side the JSON message is parsed, being extracted the following data: authenticator AAID and the key identifier for the FIDO account. After the server obtains this information it deletes the account related information from the storage structure. Any subsequent authentication attempt will fail because there will be no data related with the received key identifier (handle).

3.1.4 Integrating TEE functionality to FIDO

In ReCRED FIDO implementation almost the whole FIDO stack resides in the normal world and only security sensitive functions are executed inside the TEE. These security sensitive functions include:

1. Secure generation and storage of cryptographic keys bound to specific key use authorizations through Android’s Keystore API.
2. Secure enrolment, storage and verification of user-to-device credentials (fingerprint, pin, pattern, and password).
3. Cryptographic functions related to FIDO protocol (Signing and Verification).

In Figure 10, a typical usage example along with a part of the FIDO authenticator implementation is presented to demonstrate the security sensitive functions that are executed in TEE in relation to the user interaction. It should be mentioned that there is only user-device interaction in this example and there is no communication with the FIDO server. More specifically:

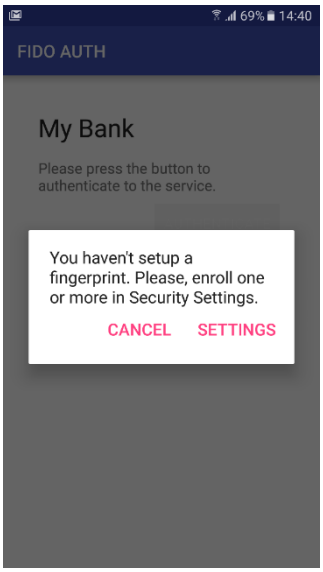

Figure 10 Fingerprint enrolment prompt dialog

1. When a user tries to authenticate to a service if the user has not registered any fingerprints it is prompted to do so in devices settings (Figure 10**Error! Reference source not found.**). E nrolment occurs directly in TEE.
2. A key pair (2048 bits) is created for the service if does not already exist in Keystore assigning specific key use authorizations. Keys are authorized to be used only for signing/verification using SHA-256 digest, RSA PSS signature padding and only if the user has been authenticated.
3. If a key already exists or has successfully generated for the service a SHA256 with RSA/PSS signature object id is initialized and the private key is injected to the object.
4. A fingerprint authentication dialog is initialized (Figure 11). Fingerprint authentication occurs directly in TEE, just as enrolment.
5. Upon authentication, the key can be used for signing / verification purposes. Signing and Verification functions are executed in TEE.

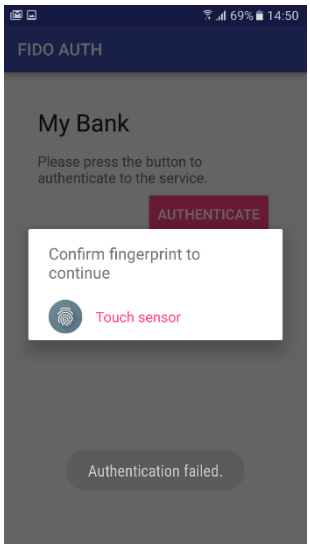

Figure 11 Fingerprint authentication dialog

3.2 Server-side authentication

GateSAFE is a Linux based server-side software product developed by certSIGN. This software product is a reverse SSL proxy server which offloads the back-end servers from the SSL and other security tasks. GateSAFE is a highly configurable software, where multiple back-end servers can be defined in a configuration file. This server supports flexible authorization rules, where only the allowed clients can access a defined URL from a back-end server.

GateSAFE is implemented as a proxy service, so that the entire protocol communication goes through it. The communication between the client (web browser) and the server (gateSAFE) is usually encrypted and authenticated: both sides authenticate each other or just the server authenticates to the client using X509 digital certificates. Further, being a server-side proxy (also known as reverse-proxy), the communication between gateSAFE application server is not encrypted with SSL and, as consequence, not authenticated either. Nevertheless, gateSAFE translates the classic HTTP authentication mechanisms (HTML form, basic/digest authentication) into X509 authentication mechanisms. The general architecture of gateSAFE is shown in Figure 12.

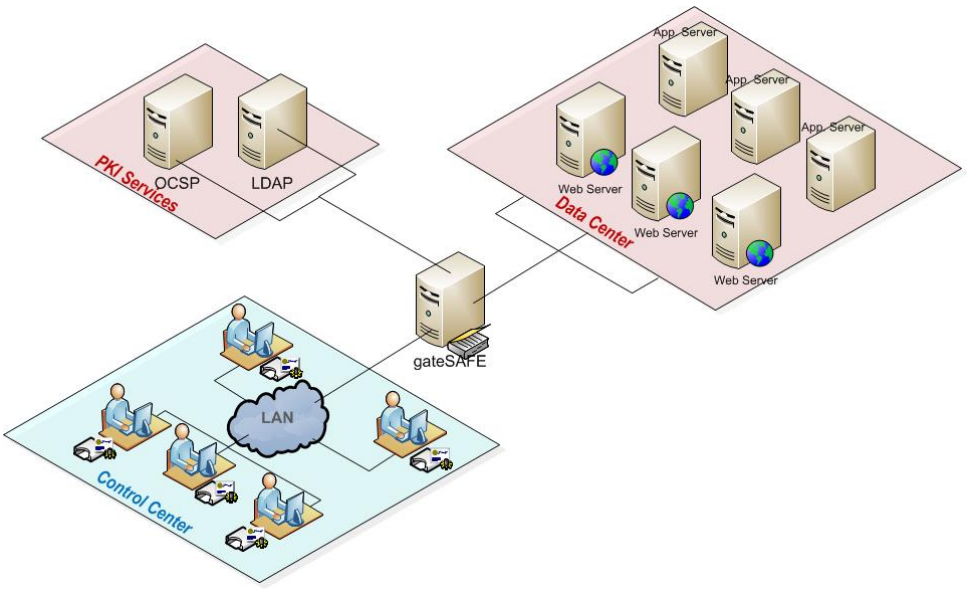

Figure 12 GateSAFE Architecture

Regarding the implementation and the internal architecture, GateSAFE has a modular approach, so that every HTTP extension that the product supports (e.g. authentication mechanisms, application business logic, traffic management) are implemented as separate modules which gives a great extensibility since the design phase.

Within ReCRED, GateSAFE is used to secure server side services (e.g. HTTP web services) by enforcing and at the same time offloading SSL protocol usage. Specialized modules will be developed in the project, modules that are aware of the specific protocol used (e.g. FIDO UAF).

In the ReCRED FIDO client-server scenario, GateSAFE releases the FIDO server from the SSL task, according to the FIDO UAF protocol specifications. The Android FIDO client establishes an SSL connection with the GateSAFE server which redirects the HTTP FIDO messages to the Apache Tomcat

server which hosts the FIDO server application. The FIDO server and GateSAFE are deployed in a Linux container solution called Docker. The communication between the FIDO server and GateSAFE happens locally, by means of a Linux virtual network interface, GateSAFE being the only entry point which can be accessed from the Internet. The deployment of GateSAFE along with the FIDO server is depicted in Figure 13. The FIDO server listens for incoming connections on a local interface IP address and GateSAFE is configured to forward the HTTP packets to this address. The GateSAFE and Docker solution has the advantage of being scalable, modular and easy to control from the security point of view.

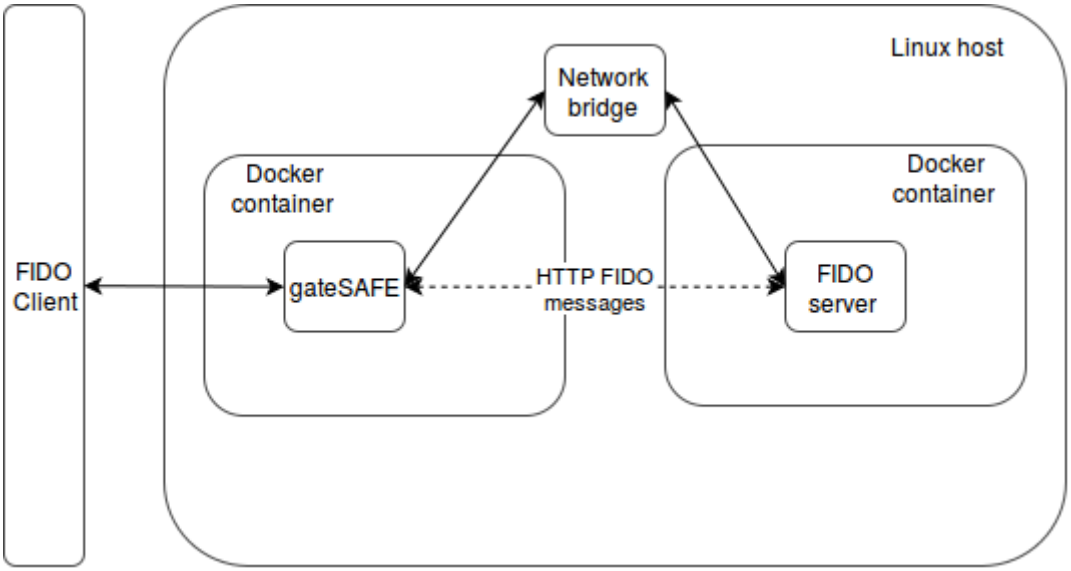

Figure 13 GateSAFE and FIDO Server Integration

4 Behavioural second-factor authentication

4.1 Gait-based Authentication

This chapter reports on the design and the implementation of a service that is capable of authenticate users based on gait dynamics. Our effort so far, has been focused on collecting gait-related data from real users in order to design machine-learning techniques to authenticate users. Most of the current research that authenticate users based on gait dynamics collected via smartphone sensors, is based on data acquired in laboratory environment or during controlled experiments. Our goal is to build a system that could work with inertial data acquired in the wild. We collected real-world gait dynamics from more than 100 users from diverse nationalities, gender, age and physical characteristics like height and weight. Our dataset contains more than 60,000 samples of inertial data. We used this data to design machine-learning based techniques for gait-based authentication and we achieved 80% accuracy by using only a limited number of samples to train the classifier.

4.1.1 Data-collection campaign

We launched a data collection campaign through the website http://gaitcollector.tid.es/. The campaign was announced through colleagues, acquaintances, and mailing lists available to the consortium partners. We asked volunteers to install an Android app that would collect gait dynamics for 2 weeks and to donate the collected data. The app was made available through the Google Play Store.

4.1.2 Gait-collector App

The app was designed according to the following principles:

- **Zero-interaction:** After the first installation of the application, no user interaction should be required. The app should collect data in the background, send the data to our servers automatically and clean up after itself. Users should not be required to perform any activity nor should they be notified about any background process running. The idea behind this the app should be “Install and Forget”.
- **Robustness:** The app should be robust enough to collect inertial data from a range of different (e.g., brand/model) smartphones. It should not be collecting data if the user is not moving. It should start and stop automatically, should be free of any crashes and should start itself even after the device reboots. It should upload data only when Wi-Fi is available, in order not to impact the user data allowance. If there is any error it should fail gracefully yet silently, keeping the zero-interaction design goal in mind. It should send the error reports or logs of the operation to our servers so that they can be analysed.
- **Performance:** While the app is running in the background, it should not affect the performance of other running applications. It should keep the resource consumption to as minimum as possible.
- **Security:** Since the app is collecting user data, it should upload the data over secure channels with encrypted data transfers.
- **Low power consumption:** The app should be battery efficient. It should keep monitoring data collection against the power consumption and optimize for both.
- **Scalability:** The app should be scalable. It should not clog the servers by scheduling data uploads at a fixed time.

- **Simple installation procedure:** The app should be widely available to the users, and installation should be as simple as possible.

In the following we provide more details on how the app works and the challenges we faced during its design.

Data Acquisition

We designed two data-collection techniques based on the sensors available on the phone. The first one uses the significant motion sensor (available on modern phones) while the second is used in devices that do not have such sensor.

Significant Motion Based Approach

- Significant motion is a low powered sensor which triggers when it detects some motion leading to a location change.
- As soon as the app starts, the sensor is registered
- If the sensor triggers itself (i.e., the user is moving), the app starts collecting accelerometer data.
- A timer is triggered to detect if there has not been walking activity in the previous 20 seconds.
- If no walking activity is detected in the previous 20 seconds, data collection stops and the significant motion is registered again.
- If the app is able to successfully collect accelerometer data for 1 minute, it stops and sleeps for 5 minutes.
- At the end of the 5 minutes the significant motion sensor is registered again.

Accelerometer based approach

- The app registers the accelerometer at 5Hz sampling frequency.
- It tries to detect any motion in the next 10 seconds.
- For motion detection, the app uses the magnitude of the accelerometer data.
- If the magnitude is greater than a threshold value (e.g., >12) the app infers motion activity.
- When motion is detected, the app enables the accelerometers at a sampling rate of 50Hz and starts writing to a file.
- If the app cannot detect motion in the next 10 seconds, it stops the service and reschedules it after one minute.
- Rescheduling is based on exponential back-off method. If there is no motion detected at the c-th failed attempt, the service is rescheduled to run after k minutes where k is a random integer between 1 and 2^c – 1.
- Maximum limit of rescheduling is set to 16 minutes and c gets reset after a successful motion detection.

User Data

In addition to the inertial data, a form was provided on the main screen of the app, asking the user for details like height, weight, gender and age (see Figure 14). All data fields were optional.

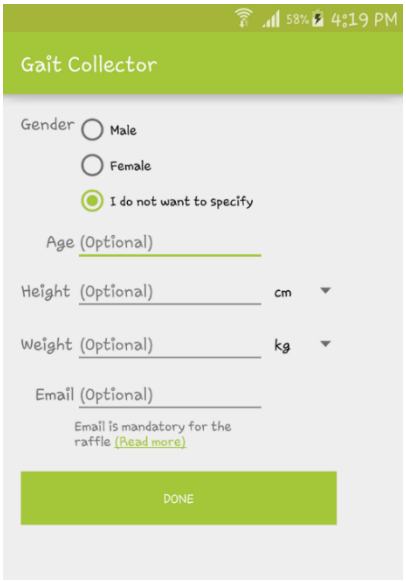

Figure 14 Gait Collector Android app initial screen

Coordinate-space transformation

The Android sensor framework uses a standard 3-axis coordinate system to express data values. The coordinate system is defined relative to the device's screen when the device is held in its default orientation as shown in Figure 15. When a device is held in its default orientation, the X-axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z-axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. We express this acceleration as a vector A with three components:

$$A = [a_x \quad a_y \quad a_z] \qquad [1]$$

where a_x, a_y and a_z represent the magnitude of the acceleration force acting along the three dimensions.

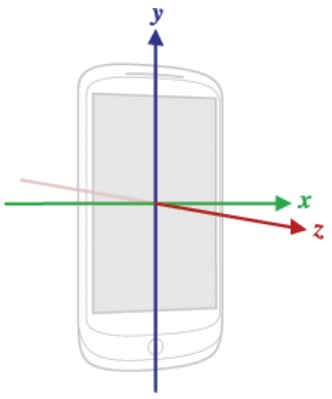

Figure 15 Coordinate system (relative to a device) used by the Android Sensor API

Since the phone orientation changes from time to time, using the default coordinate system presents a challenge in using these signals. Therefore, collected data should be transformed into a coordinate system which is constant throughout the data collection process. Hence, we refer to Earth (see Figure

16) as the fixed coordinate system and each of the component of acceleration is transformed into this coordinate system where

- X is defined as the cross product of Y and Z, tangential to the ground and the device current location and point to the East.
- Y is also tangential to the ground and point to the North Pole.
- Z points toward the sky and is perpendicular to the ground plane.

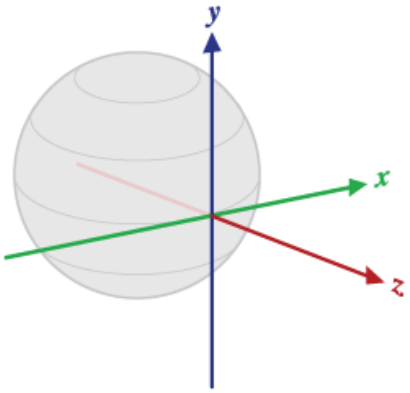

Figure 16 Earth Coordinate system used in the app

In order to transform the raw acceleration vector $A = [a_x \quad a_y \quad a_z]$ into Earth coordinates $A' = [a'_x \quad a'_y \quad a'_z]$, we use the following equation:

$$A' = A.R \qquad [2]$$

where R is the Rotation Matrix calculated from yaw, pitch and roll $[\alpha \quad \beta \quad \gamma]$and is defined as follows:

$$R(\alpha,\beta,\gamma) = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\cos\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \qquad [3]$$

In order to calculate the R matrix, we also need the gravity and magnetometer sensors. Android SDK provides helper methods to calculate these vectors and transform to Earth's coordinates. After transformation, the 3-dimensional accelerometer plots look like in the Figure 17.

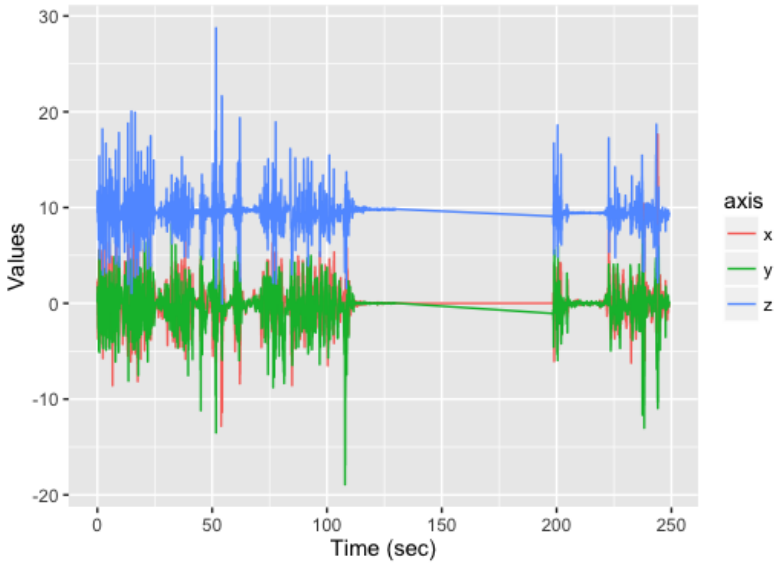

Figure 17 Three dimensional accelerometer signals

Data Formats

We collected the inertial data in csv files labelled with the following format:

“YYYY-MM-DD HH_mm+ZZZZ-data-uuuuuuuu-uuuu-uuuu-uuuu-uuuuuuuuuuuu.csv”

where the first part is the date, time and time zone at the instant data collection process was started, and the later part is a 36 characters long alpha-numeric random identifier (UUID) in order to differentiate one user from the other as well as assign a random unique ID to the user in order to keep him anonymous. The data was stored in the following format:

timestamp from the sensor event, a_x, a_y, a_z

User data was stored in separate files. Device build and software versions were also stored in these files. Files were stored in the user directory of the phone visible to the user (in case they wanted to see the data collected) and later on transferred over a secure connection to the server and deleted from the phone.

Data sync

In order to be scalable, fixed scheduling of network transfers should be avoided. Android SDK provides sync adapter which not only mitigates this problem by randomizing the sync times but also optimizes the network transfer, in order to conserve both battery and network bandwidth. We used this sync adapter to transfer data at a scheduling frequency of two hours. Data transfers were only enabled over WiFi in order not to impact the data allowance of the phone.

4.1.3 Server

Pre-processing

Differences in the way time is represented on different Android devices were mitigated by using a relative timestamp. Since each file accounts for one minute of data collection, we find the minimum

timestamp in the signal and subtract it from all timestamps in the signal. This way the first data point starts at 0 and the last may not be higher than 60 seconds. Hence making it consistent across devices.

Time Interpolation

Mobile sensors are low powered and less stable as compared to commercial grade inertial measurement units which operate at a much higher frequency and with relatively stable frequency. The frequency in our signals was not constant and varied across the signals. In order for them to be analysed uniformly, there was a need to interpolate the signal so that two consecutive data-points had the same time difference throughout the sample. We employed linear interpolation to our signals since it is computationally cheap and quite easy to implement.

Given a set of points$(x_0, y_0)$and (x_1, y_1) the linear interpolant is the straight line between these points. If a point x lies on this straight line, the point can be calculated using the following equation:

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0} \qquad [4]$$

Using the above equation, we interpolated the three dimensional signal separately at a constant sampling frequency of 50 Hz (20ms time period). As can be seen in Figure 18 we were able to restore the lost signal at certain points.

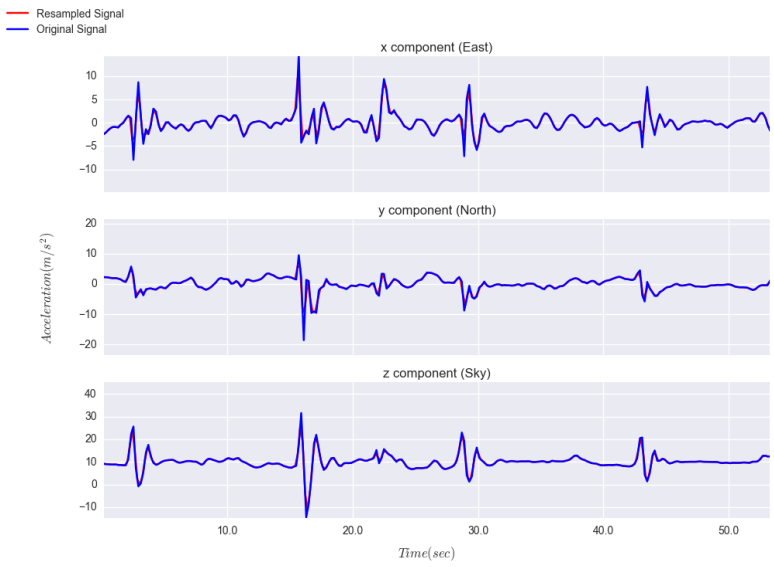

Figure 18 Three-Dimensional Accelerometer Signal w ith Resampled Signal at 50 Hz

Zero Normalization

We transformed our signal into earth’s coordinate system, so in steady state the magnitude of the acceleration along the z-axis (pointing towards sky) should be equal to the gravitational force of earth, while the acceleration along the other two axes should be equal to zero. However, this was not the case because smartphones accelerometer sensors are not stable enough. In order to minimize this problem, all three acceleration signals were zero normalized by subtracting their means from as shown in the following equation

$$A_i'(t) = A_i(t) - \mu_i, i \in \{x, y, z\} \quad [5]$$

This resulted in the signal shape shown in Figure 19.

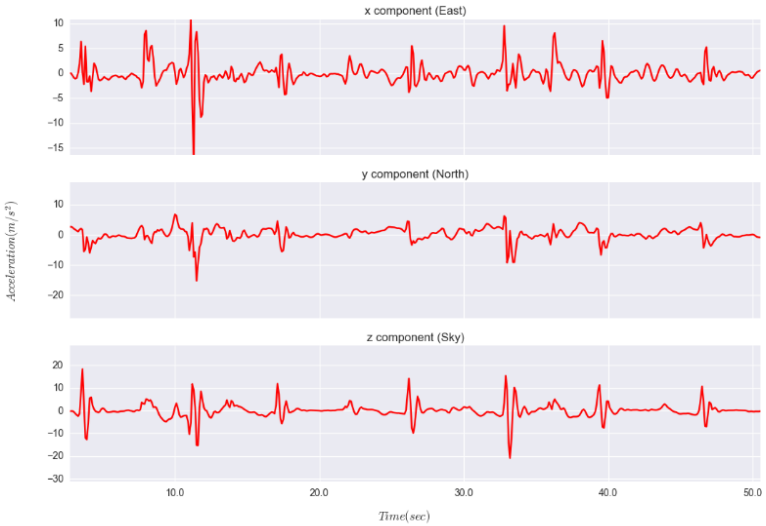

Figure 19 Mean subtracted from each of the three dimensional acceleration signals

Resultant Acceleration

We converted the three dimensional acceleration into its resultant form by calculating the L2 norm of the acceleration vector. The reason for this dimensionality reduction was to minimize any orientation related problems and we had to process only one signal instead of a vector of three components. L2 norm was calculated by the following equation:

$$r(t) = \sqrt{a_x^2(t) + a_y^2(t) + a_z^2(t)}, t = 1,2, \dots, n$$

Walk Annotation

Since we were collecting samples of one minute and part of the signal may have n walking pattern, there was a need to annotate each data point in our sample as being a “walking” or “not walking”. In order to do so, we passed the sample through the following process.

We defined a fixed window of 3 seconds, and rolled it through the signal. If the three second window featured walking, we annotated every data point in that sample as “walking” otherwise it was marked as “not walking”. The time limit of 3 second was chosen so that we had a minimum amount of window to annotate and as it has been shown to yield good accuracy.

Detection of “walking” was accomplished by passing the signal through a band-pass Butterworth filter with cut-off frequencies of 1-3Hz in order to filter out all the noise from the signal. The cut-off frequencies of 1-3 were selected because they are the frequencies at which humans walk according to previous work in the area. We transformed this signal window into frequency domain and checked if the amplitude of the frequency was bigger than a certain threshold in the frequency range of 1-3Hz.

If the maximum value of the amplitude was higher than the threshold value of 3, we regarded it as “walking” otherwise it was marked as “not walking”.

The threshold of 3 was chosen by manually inspecting hundreds of signals which constituted ground truth, then passing them through the filter and checking the signal in the frequency domain, saving the maximum amplitude and then plotting the histogram of all the walking and non-walking amplitudes.

Figure 20 and Figure 21 show a sample of “non walking” signal. No frequency value in the region 1-3Hz was above the amplitude of 3. But as we analysed the walking signals in Figure 22 and Figure 23 we observed that the amplitudes were greater than the threshold value of 3. We evaluated 100s of these signals, selecting a random 3-second window from the signal where we knew it was a walk/non-walk and plotted its amplitude value. As shown inFigure 20, we can clearly see that walking was separated from non-walking at the boundary of 3.

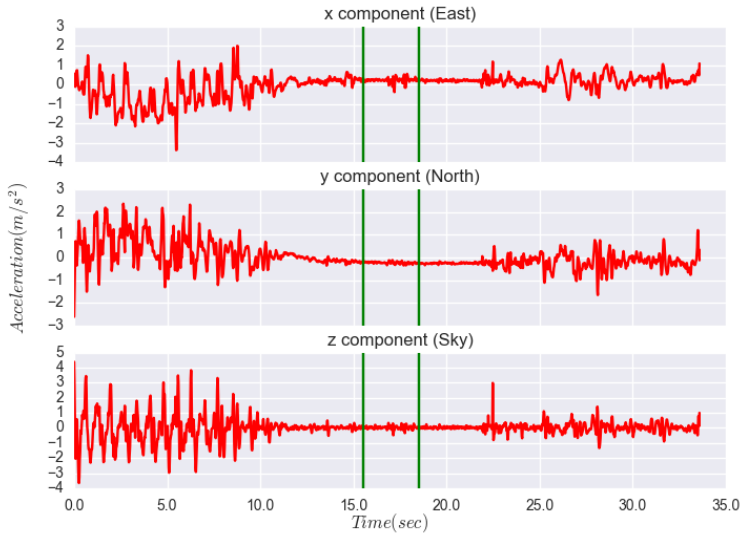

Figure 20 Signal in time domain where in the window (area between green lines) there is no walking activity

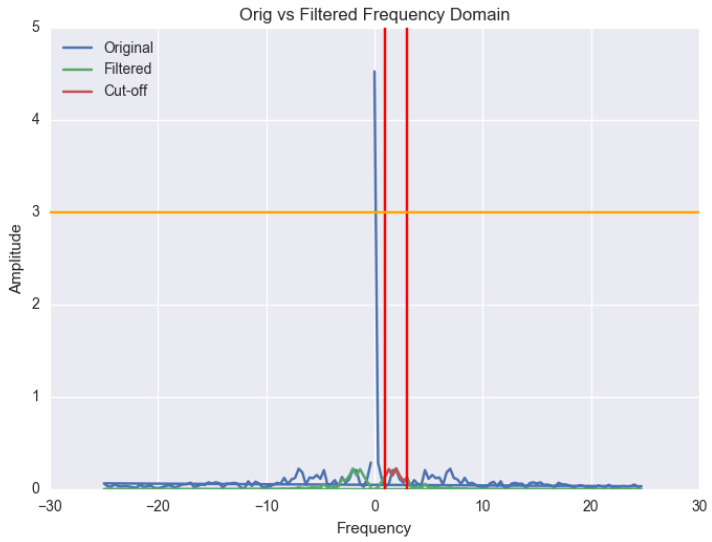

Figure 21 No walking signal in frequency domain. The amplitude in 1-3Hz frequency range is below threshold

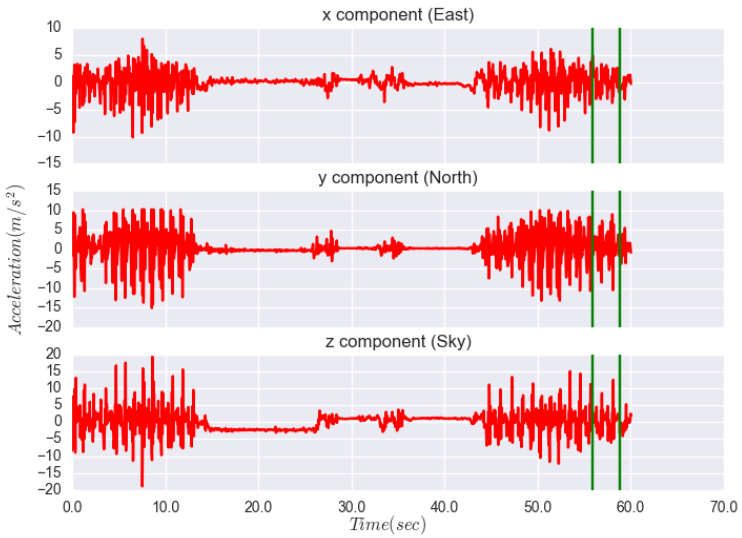

Figure 22 A signal in time domain with both walking and no walking activity. The window in the green region is walking

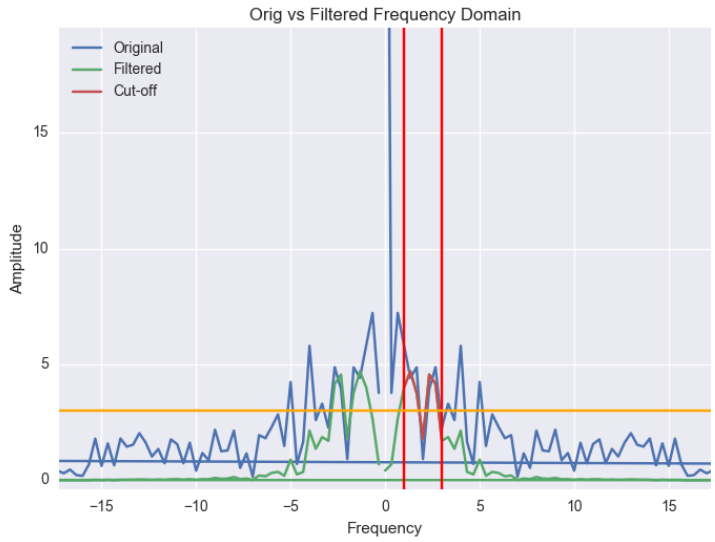

Figure 23 Walking signal in frequency domain, the amplitude in 1-3Hz is above the threshold value

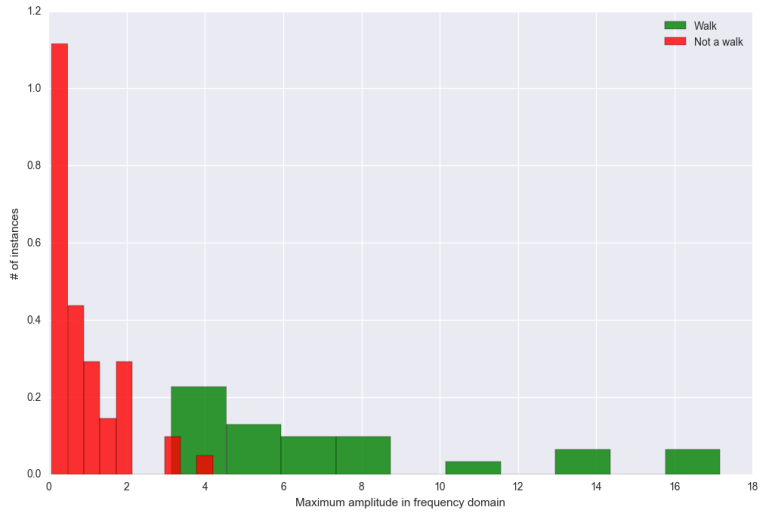

Figure 24 Distribution of maximum amplitudes in 1-3Hz region of frequency domain of ~100 signals. There is a clear threshold at 3

After annotation we had a signal like the one shown in Figure 25, where green area represents “walking”, while red one is “no walking”.

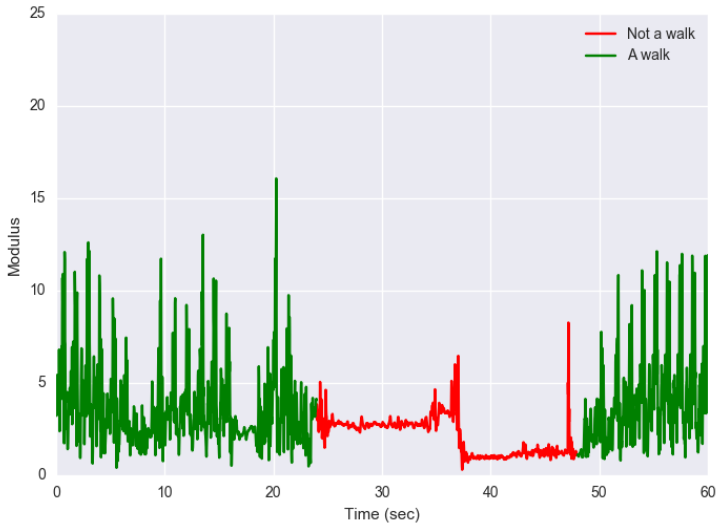

Figure 25 Modulus of the three dimensional axes of acceleration with both walking and no walking data points

File Transformation

After the data was annotated for walking/no walking, we calculated the percentage of data points annotated with walking. Moreover, we also translated the 36 characters random UUID to an integer ID for each user. We saved the cleaned data samples in new files with name YYYY-MM-DD HH-MM+ZZZZ_MAT_uid_per.csv where uid is the integer ID and per is the walking percentage calculated in the sample. The file format was as follows:

x,y,z,mod,walk

-5.61683003493528517e-01, 1.913083062985166,2.531126236593371104,3.222107139761943628,1

Feature Extraction

After we were done with the pre-processing step, the next phase was feature extraction from the clean samples. We extracted the features on only the data samples which had the annotation “walking”. We used a moving window with size of 3 seconds and step of 1.5 seconds. The window only operated on the walking data. For each window we calculated the features described below. For a given sample, we calculated the mean and percentiles 5, 25, 50, 75, 95 of all the windows upon which features were calculated. The following features were calculated.

Time domain features:

Mean $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ [7]

Standard Deviation $\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}|x_i - \overline{x}|}$ [8]

Max $x_{max} = \max(x_i), i = 1..n$ [9]

Min	$x_{min} = \min(x_i), i = 1..n$	[10]
Range	$range = x_{max} - x_{min}$	[11]
Root Mean Square	$RMS = \frac{1}{n}\sum_{i=1}^{n} x_i^2$	[12]
Waveform length	$wl = \frac{1}{n-1}\sum_{i=1}^{n-1}\lvert x_{i+1} - x_i\rvert$	[13]
20-bin histogram distribution	$hist = (n_j)_0^{20}$	[14]
	with $n_j = \sum_{i=1}^{n} x_i$	
	$\frac{j\,\Delta j}{20} \leq x_i \in bin_j < \frac{(j+1)\,\Delta j}{20}$	
	$\Delta j = \max(z) - \min(z),$	
	$z = all\ signals\ concatenated\ together$	

Frequency domain features:

15 first FFT Coefficients	$fft_{15} = (X_k)_1^{15}$	[15]
	$X_k = \sum_{i=0}^{n} x_i e^{-\frac{j2\pi ki}{n}}$	
1st Dominant Power	$P_1 = (\max(fft))^2$	[16]
1st Dominant Frequency	$f_1 = f_{P_1}$	[17]
2nd Dominant Power	$P_2 = (\max_2(fft))^2$	[16]
2nd Dominant Frequency	$f_2 = f_{P_2}$	[17]
Total Power	$P_T = \sum\lvert fft\rvert^2$	[18]
1st Normalized Power	$1np = \frac{P_1}{P_T}$	[19]

All of the above features were calculated on window basis and then average and percentiles were calculated for all windows in one sample. It is to be noted that Fourier transforms only work on fixed window lengths, which was already solved by keeping a window size fixed at 3secs.

The 20 percentiles were calculated for the whole sample using the equation:

$\Pr[X < x] \leq k\ /q$ [20]

In addition to the above time and frequency domain feature we also selected the hour of day as a feature by transforming into sin and cosine of the hour of the day using the following equations.

sin hour of day $h_{sin} = \sin(\frac{2\pi h}{24})$ [21]

cos hour of day $h_{cos} = \cos(\frac{2\pi h}{24})$ [22]

In total, we calculated 152 features for a given sample. The features were calculated for each sample and formatted as a row in a feature file with the following format:

Filename, date time, sin and cos hour of day, mean (time domain features), percentile (time domain features), mean (frequency domain features), percentile (frequency domain features), 20 percentiles (sample file), walk percent in the sample file, integer based user ID.

Classification

We used the Random Forest classifier, which is an ensemble learning method for classification and regression with decision trees where the trees are constructed using bootstrap aggregation (bagging). For each tree, bagging selects a random sample with replacement from the training set, and creates a decision tree based on this sample. After n number of trees have been trained, the output for the predication is the average of each individual tree.

The primary reason for using Random Forests was its performance. Moreover, it is easy to tune and use, can work at any scale, yields good accuracy, performs auto feature scaling as well as balance the number of positive examples vs negative examples.

We only focused on the authentication of users which translate to a binary classification problem. For each of the users, we divided the whole dataset into two parts. User U with a training and testing set and remaining users U′ again with training and testing sets. We made sure that each user is represented adequately in U′ and the samples were collected at random. For user U we collected a contiguous set of samples ordered by time and the subset selected at random from the sample set of user U.

We then ran the classifier and predicted the outcome for the training examples using the model obtained in the previous step.

We define as *positive class* the samples of the legitimate user whereas the *negative class* is made of the samples of the purported adversary. In order to measure the performance of the classifier, we define *True Positive (TP)* as the number of times we correctly classified the legitimate user. *True Negative (TN)* represents the number of times we correctly classified an adversary. *False Positive (FP)* represents the number of times we classify the adversary as the legitimate user. *False Negative (FN)* represents the number of times we classify the legitimate user as an adversary.

In order to assess the performance of the classifier, we calculated the ROC AUC score and the balanced accuracies for all the predictions. We cross validated the scores by running the same procedure for 20 times (20-time repeated random cross validation) and calculated the average and standard deviation of the AUC score and accuracy.

ROC (Receiver Operating Characteristic) shows the performance of a binary classifier as its discriminant threshold is varied. The curve is created by plotting the True Positive Rate against the False Positive Rate at different threshold values. We used ROC to generate the summary statistic of Area under the curve (AUC) which is the probability with which the classifier would rank a randomly chosen positive sample as higher than a randomly chosen negative sample.

The balanced accuracies are calculated using the following equations:

$$Accuracy_{TN} = \frac{TN}{TN+FP} \text{[23]}$$

$$Accuracy_{TP} = \frac{TP}{FN+TP} \text{[24]}$$

$$Balance\ Accuracy = \frac{Accuracy_{TN} + Accuracy_{TP}}{2} \text{[25]}$$

4.1.4 Experiment and Analysis

The app was distributed through the Google Play Store and we were able to reach a wider audience as can be seen in Figure 26.

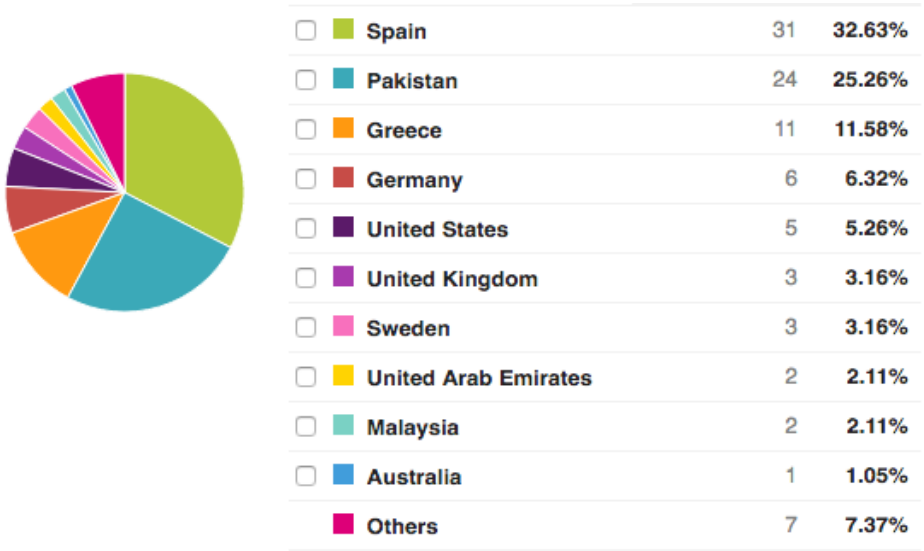

Figure 26 Demographics

At the time of writing, we were able to collect 60,000+ samples from 110 different users with at least 89 males, 15 females and the rest unknown. The age range was from 20 to 60 years with a mean age of 30 years and a standard deviation of 6.8. The mean weight and height were 75kg and 174cm respectively with standard deviations of 12.4 and 8.9 respectively. Figure 27, Figure 28, and Figure 29 show the distributions of the users with respect to age, height, weight and gender.

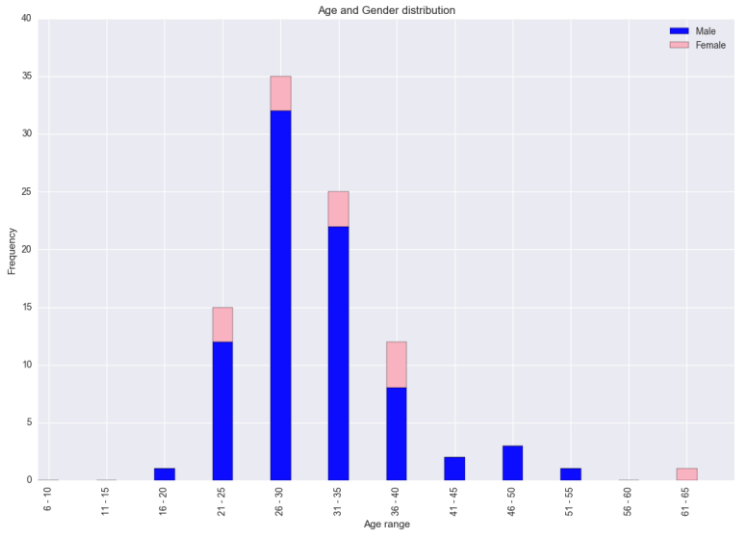

Figure 27 Distribution of users with respect to age and gender

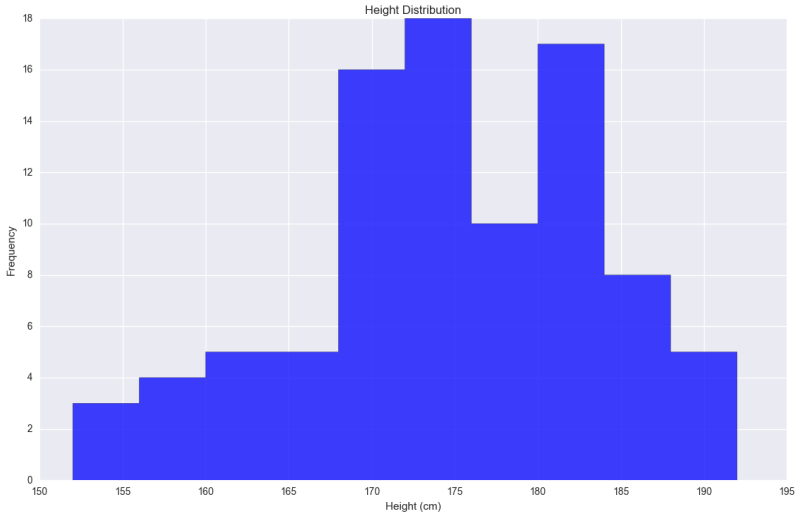

Figure 28 Distribution of users with respect to height

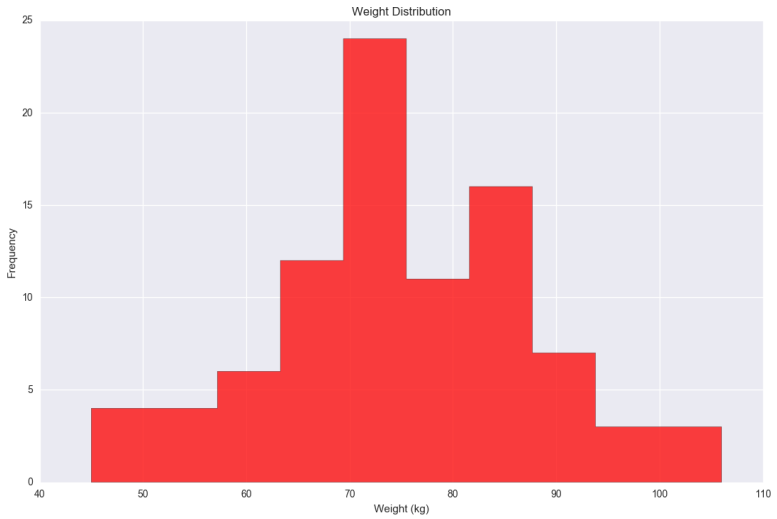

Figure 29 Distribution of users with respect to weight

Experiments

In order to decide the number of per-user samples to be used to train the classifier we first looked at the number of samples that each user provided. As can be seen in Figure 30, where we plotted the cumulative distribution of users with respect to the number of samples, about 80 people had at least 50 samples.

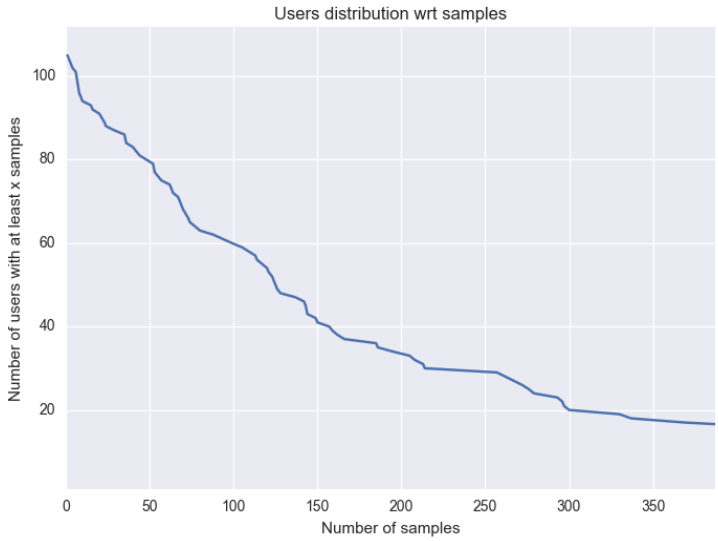

Figure 30 Number of users against number of samples per user

Impact of walking percentage

Recall that each collected samples account for a one-minute signal but that only a part of it could actually feature walking. We therefore decided to investigate the impact of the walking percentage within a sample on our system performance.

In order to carry out this experiment, we divided the users into two groups: one with the actual user U to be authenticated and the other, denoted by U’, with all other users (acting as potential adversaries).

Only users with at least 30 samples of one minute long walk (i.e., 100% walk percentage in the sample) were considered, which reduced the number of eligible users to 29.

We started by selecting 10% of the waling data for each sample. For user U, 25 consecutive samples ordered by time, were randomly chosen as training examples and 5 samples as testing examples. For each user in U’ we randomly chose 25 samples for training and 5 samples for testing.

The classifier was trained for each user U, with a 20 time repeated-random cross validation. We predicted the scores for the testing samples and calculated the ROC AUC score and accuracy as described above.

In the next iterations, we repeated the process by increasing the minimum percentage required in steps of 10%. The result is shown in the Figure 31.

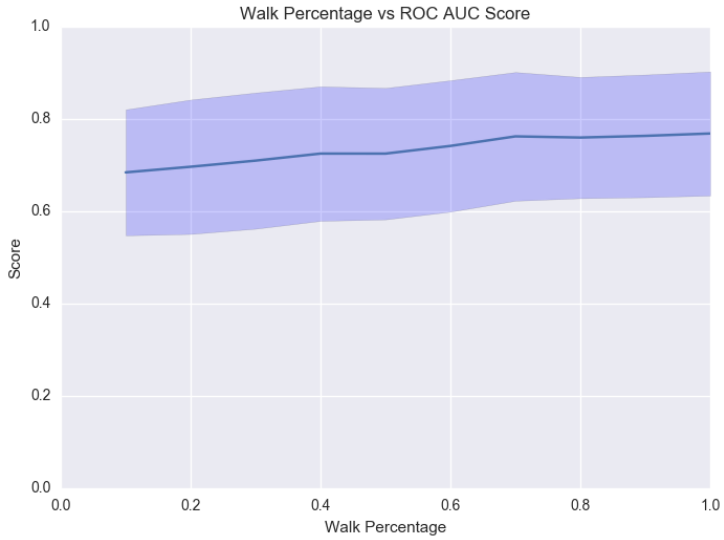

Figure 31 Impact of the walking percentage in a signal on the classifier accuracy

As can be seen in Figure 31, samples higher walking percentage leads to better results up to a score of 0.8. We concluded that samples with 70% of walking data are a good choices to be used in the rest of the experiments.

We also report the confusion matrix for the scenario where we used samples with 100% walking data. Accuracy resulted in 82% for the negative class and 72% for the positive class.

	Predicted U'	Predicted U
Actual U'	TN 66698	FP 14502
Actual U	FN 824	TP 2076

Minimum number of training samples

In this experiment we tried to investigate the minimum number of samples required to train the classifier. From the previous experiment, we set to 70% the percentage of walking data that a sample must have in ordered to be taken into account. This allowed us to select 56 users for the experiment.

We increased the number of training samples from 1 to 25. The samples were again chosen in consecutive time order but with a random starting point. The experiment was repeated 20 times and the mean of the 20 experiments score was calculated. As evident from Figure 32, we required at least 20 samples to reach a score of 7.5.

Figure 32 Increasing training samples, results in improved accuracy

The confusion matrix for this experiment when 25 samples were used, can be found in the following table. We can calculate the accuracies of the true negative and true positive classes using Eq. [24] and [25] which were 74% for both. Hence resulting in a balanced accuracy of 74%.

	Predicted U’	Predicted U
Actual U’	TN 228581	FP 79419
Actual U	FN 1459	TP 4141

We plotted the percentile of users who were wrongly predicted in Figure 33. About 40 percentile of users had error below 20% while 80 percentile of users had error below 30%, as can be seen in Figure 33.

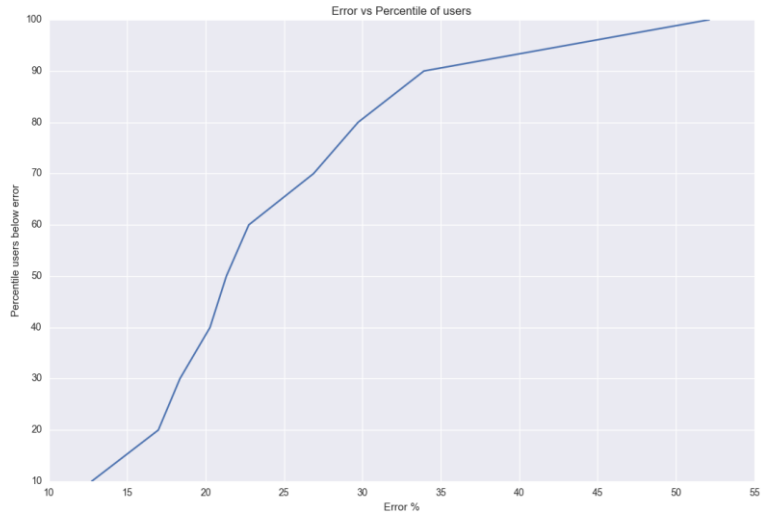

Figure 33 Percentile of users who are below a certain error point

We also plotted the scores for individual users in Figure 34 and noticed that some users always achieve higher accuracies than others with quite low standard deviation.

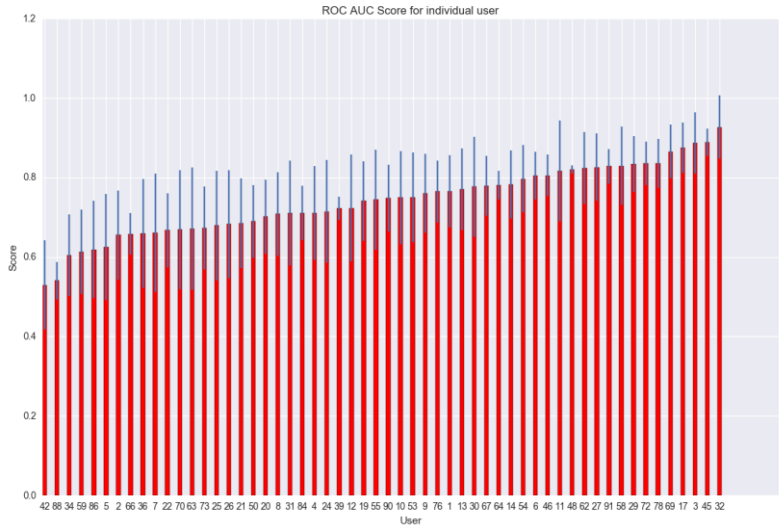

Figure 34 Individual ROC AUC score

Impact of gender, age, height and weight

We used the results obtained in the second experiment and classified the users according to their gender, age, height and weight. The results are explained in the following paragraphs.

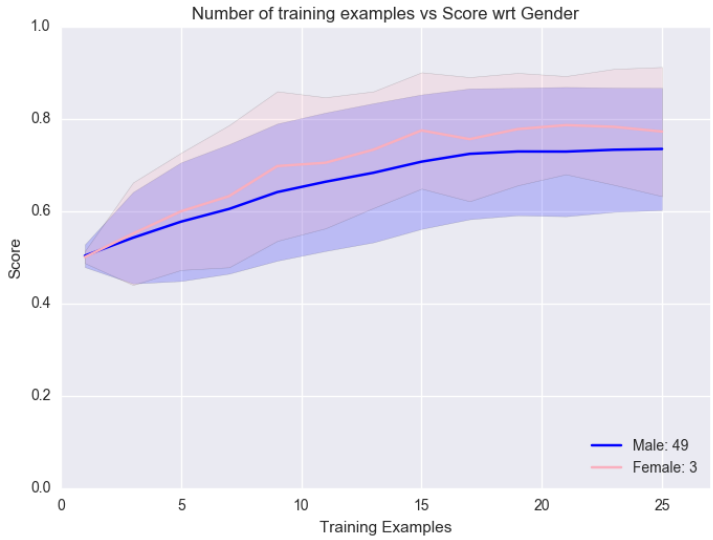

Figure 35 Impact of gender on gait authentication, Female perform differently than male

Despite a skewed user base (only 3 users declared their gender as female) results in Figure 35 suggest that females are more discriminant than male. We only had 3 female subjects all of whom performed better than any of the male subjects.

The average age of our user base was roughly 30 years. We could not see any impact of age on the classifier performance as evident from the following plot (Figure 36).

Figure 36 Age has no real impact on authentication performance

As shown in Figure 37, Figure 38 and Figure 39, height and weight do not heavily impact the performance of the classifier. However, the classifier performs slightly better for users with low height and weight thus low BMI.

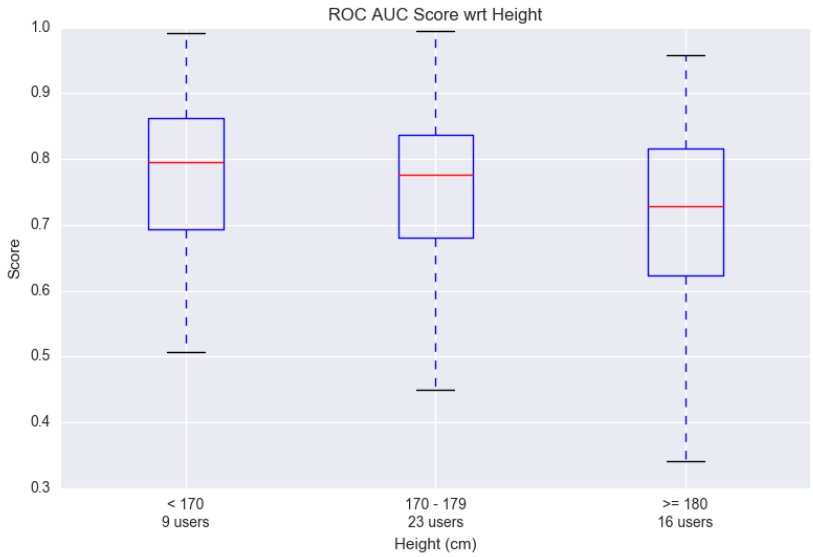

Figure 37 Impact of height on authentication score

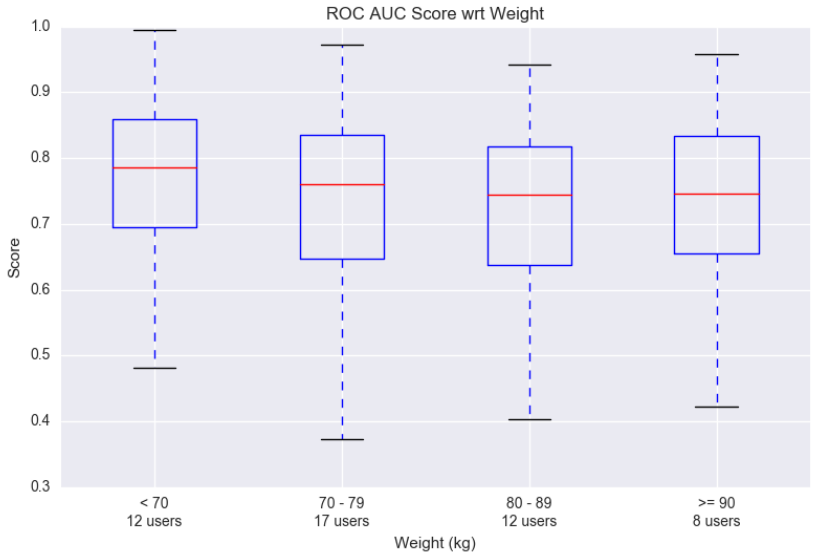

Figure 38 Impact of weight on authentication score

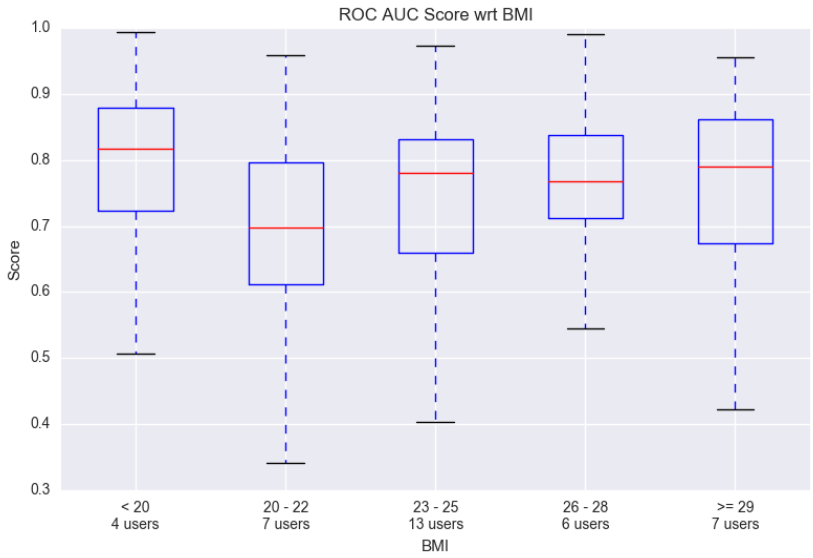

Figure 39 Impact of Body Mass Index (BMI) on authentication score

4.2 Mobility-based Authentication

This chapter reports on the design and implementation of a service that is capable of authenticating users based on their mobility dynamics. Our effort so far has been placed in the following directions:

1) Pre-processing mobile call description records (CDRs) of users and extracting various features regarding the mobility of users from these data.
2) Testing preliminary methods for user identification that can lead to successful authentication of users with a low error margin.

4.2.1 Methodology

We define a methodology to authenticate users based on their mobility patterns as extracted from call records. We should keep in mind that such method should potentially work for millions of users and the authentication should be fast and with as low margin of error as possible.

The proposed service constructs a database of user profiles based on historical data of location visits as extracted from mobile call records of each user during time period T. Then, the user is authenticated if a sample of recent data at time period T' (which we will call fingerprint) matches with some confidence the historical profile on record for the particular user.

Profile build-up.

The user profile is built using data from a mobile call records database, by executing the following steps:

Step 1:

- We extract location features for each user from CDRs for time period T. These features are compiled into two vectors. For these vectors to be properly defined, we use the following notations:
  - $L_a(T)$: the frequency of calls the user did from the location of a cell tower *a* during time period T.
  - A: the set of locations of towers $a \in A_i$ the user performed at least one call during time period T.
  - $TL_{<a,b>}(T)$: the frequency of transitions of the user from location tower *a* to *b*, i.e., the number of times the user did a call from tower *a* and then did a subsequent call from tower *b* during time period T.
  - S: the set of location transitions the user did for outgoing calls during time period T.
- Vector 1: User caller activity. This vector is comprised of the combination of two subvectors: one from the frequency of calls from various locations (i.e., cell towers), and another from the frequency of transitions from one location to another. Analytically,

  $< \{ L_a(T) \mid a \in A_i \}, \{TL_{<a,b>}(T) \mid < a, b > \in S_i\} >_{caller}$

- Vector 2: User callee activity. This vector is similarly defined as the previous vector, but it reflects the activity the user had when other users called him, i.e., when he was the callee. To avoid obvious repetition, we only present its analytical form:

  $< \{ L_a(T) \mid a \in A_i \}, \{TL_{<a,b>}(T) \mid < a, b > \in S_i\} >_{callee}$

Step 2:

- We normalize the two vectors of each user with a tf-idf process:
-

$$u_i = tf_i \times (1 + \ln({}^{N}/_{(df_i + 1)})$$

where

- N is the total number of users in the system,
- tfi is the frequency of the item i that was found in the user's location logs (i.e., La(T) or TL<a,b>(T)),
- dfi is the number of users who have item i (location or location transition) in their location logs

- ui is the tf-idf normalized value of each item i (location or location transition) in the vectors of the user.

Step 3:

The final output of this process is two normalized vectors per user, which comprise his system profile at time T: $P_{user}(T)$.

Fingerprint build-up

While the user is in the system, for every time period T' ≥ T, the service computes a fingerprint which has the same data structure as the profile of the user. The process followed is exactly the same as before for the profile build-up (part A), but the difference is the time period used: T' vs. T). The two vectors produced after this step are the user's current fingerprints, $F_{user}(T')$.

Cosine Similarity Comparison

When the user logs in back to the service, the system can challenge his authentication by comparing the fingerprint of the specific user from the latest period T' and the profile stored in the system for the particular user. For this task, we utilize the well-known cosine similarity. This is a measure of similarity between two vectors of an inner product space. It measures the cosine of the angle between the two vectors. If the cosine similarity is close to 1, then the two vectors are very similar and aligned, whereas if it is close to 0, then the two vectors are almost dissimilar.

In this step, we compute the following similarity:

$$\theta\left(P_{user}(T), F_{user}(T')\right) = \frac{\sum_{i=1}^{n}(u_i \times v_i)}{\sqrt{\sum_{i=1}^{n} u_i^2} \times \sqrt{\sum_{i=1}^{n} v_i^2}}$$

where

- n is the number of items in the vector
- ui is the tf-idf value for item i of the profile of the user at time T
- vi is the tf-idf value for the same item i of the same user from his fingerprint at time T’

Since the profile and the fingerprint of each user are comprised of two vectors, one for the caller and one for the callee, two different cosine similarities are computed per user authentication.

Pairwise comparison for baseline

The performance of the method lies in how many other users could be found "similar" to the user under testing. For this, we produce pair-wise comparisons for a set of R random users from the fingerprinted vectors. That is, we compare via cosine similarity computation of their vectors, each user

in the random set with every other user in the random set. The pairwise comparisons of these random users are performed as a baseline to have a comparison where the similarity of a user's fingerprint with his profile would render the user authenticated or not.

In this step, we compute the following similarity:

$$\theta\left(F_{user1}(T'), F_{user2}(T')\right) = \frac{\sum_{i=1}^{n}(u_i \times v_i)}{\sqrt{\sum_{i=1}^{n} u_i^2} \times \sqrt{\sum_{i=1}^{n} v_i^2}}$$

where

- n is the number of items in the vector
- ui is the tf-idf value for item i of the fingerprint of user1 at time T'
- vi is the tf-idf value for the same item i of the fingerprint of user2 at time T'

Min Rank

Finally, we compute a confidence level on how certain we are of the user being the same one we have stored in the system, by comparing his self-similarity score with the other users' scores.

For this, we use the cosine similarity of the user's profile and current fingerprint (self-similarity), and a set of pairwise comparisons of the same user with the rest of the random users in R. We sort the pairwise comparison similarities in descending order and compute the min rank *k* that the user gets with his self-similarity within this sorted list. If this rank k=1, then the user's self-similarity is top of the list. If the rank k=4, then there are three other users in R with whom the user under testing has higher similarity than with himself in the stored profile.

In general, if we apply a lower threshold of ranking in the system, the number of users who are correctly authenticated (i.e., true positives) can increase, but at the same time the number of other users who may be mistaken as the given user (i.e., false positives).

4.2.2 Evaluation

In order to test our methodology, we used the following experimental setup:

- We used a dataset of mobile call records of ~1.6 million users and ~220 million call records per month. From this dataset we extracted the calling activities of users in a 3-month period between May 2014 and July 2014.
- The time period T for the profile building was defined as 1 month.
- The time period T' for the fingerprint building was defined as 1 day, 1 week, 2 weeks, 4 weeks.
- We repeat the methodology for the 2 consecutive months to confirm that the method is stable.

Implementation

All the modules needed for this experimental setup (i.e., profile and fingerprint build-up, cosine similarity and min-rank comparison) were done in Java for easy portability and deployment in different computing platforms (e.g., server vs. a regular PC). No other special libraries were used.

Results

Next (Figure 40), we computed the variation of vector size of user profiles for the two months, May and June 2014. We notice that the distribution of profile sizes is stable with respect to time. We also note that the median size of the vectors is around 8 locations / location transitions.

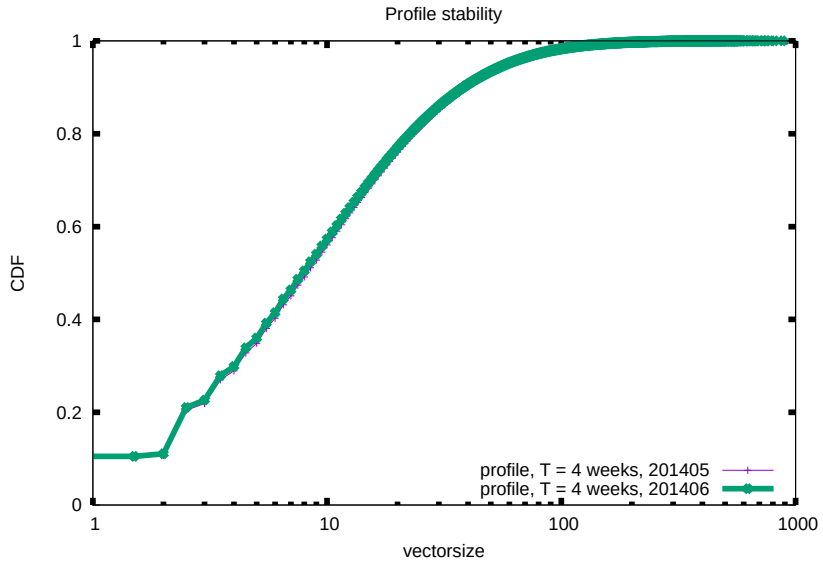

Figure 40 CDF of size of vector with respect to instances of user profile for two testing months.

We also compute a similar comparison for the fingerprints used in the two periods (i.e., June and July of 2014). The result (Figure 41) is similar and confirms that users’ fingerprints are also fairly stable through time. We also note that the median size for a fingerprint of 1 day is about 2 locations, 4-5 locations for 1 week, and around 6 locations for 2 weeks.

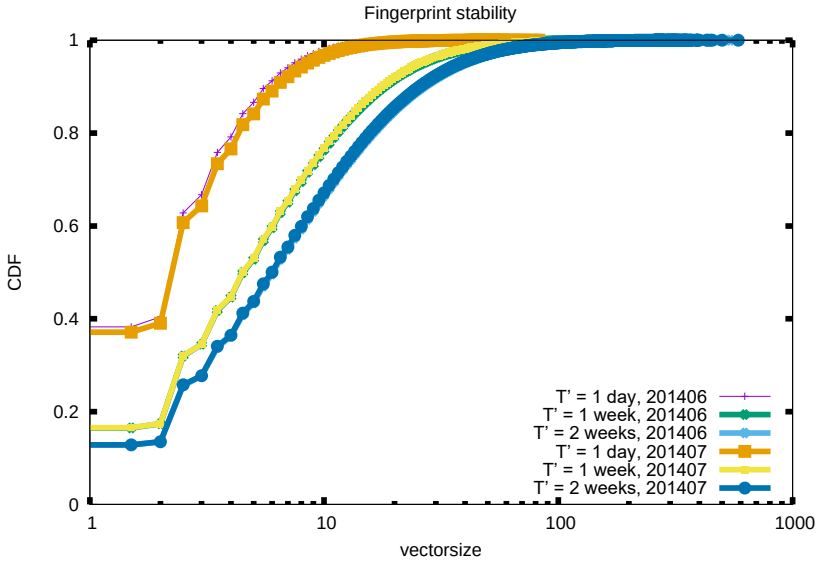

Figure 41 CDF of size of vector with respect to instances of user fingerprints for 3 different time periods and two testing months

Furthermore, we compute the cosine similarity of users between their profile and their current fingerprint, for different time periods T’ and the two testing months. The next plots (Figure 42) show that users’ fingerprints tend to be very similar to their stored profiles, and this is a consistent result through time. Also, the variability of the similarity for different time periods T’ is not very large, even though a larger period usually performs better.

In fact, 70% of users have ~0.7 cosine similarity or better. Even for the 1-day fingerprints, this similarity goes down to 0.5 or better. The results shown were computed on the caller activity vector. However, similar results are extracted with the callee activity vector and are omitted for brevity.

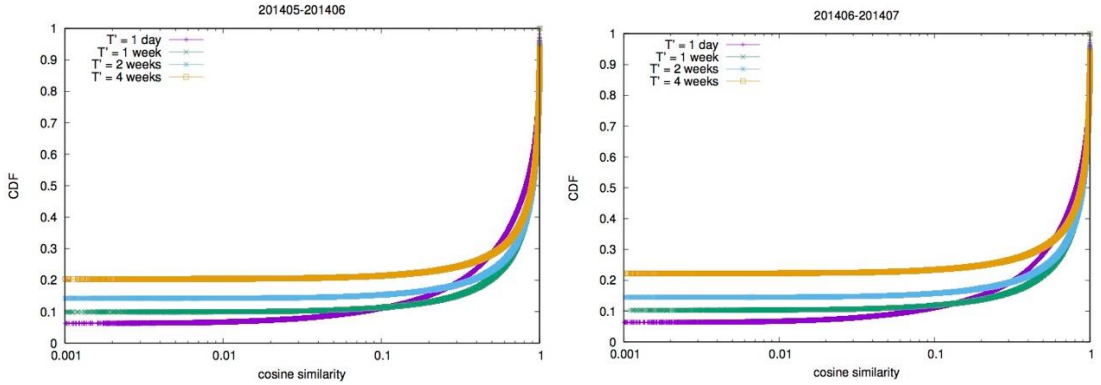

Figure 42 CDF of cosine similarity for comparison of profile and fingerprint vectors for different time periods and two testing months

We also compute the pairwise similarity of a set of random users for different time periods T’, for the two testing months. As we can see (Figure 43), the great majority of user pairs (about 95%) have similarity zero. Only about 5% of pairs have a non-zero similarity and only about 0.5% of pairs have a similarity about 0.1. These results are for caller vector activity but are similarly observed in callee vectors and for the other testing month. Therefore, they are omitted for brevity.

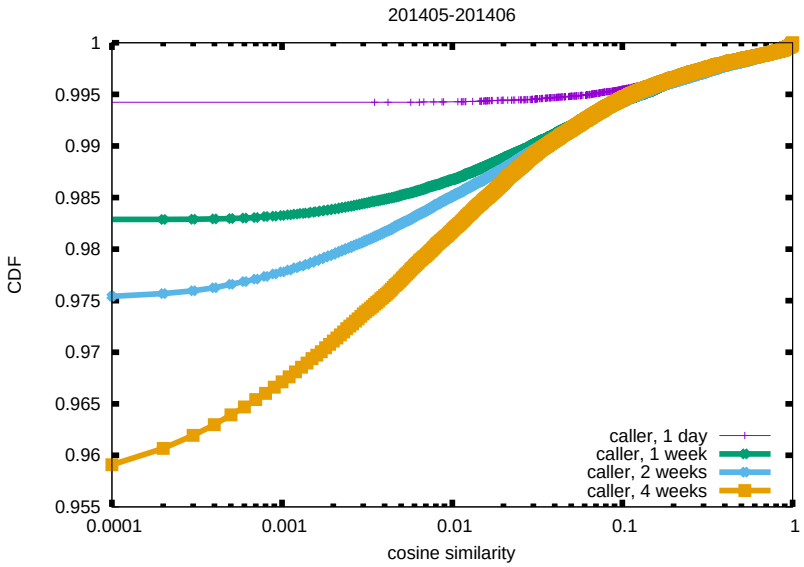

Figure 43 CDF of cosine similarity from the pair-wise comparison of fingerprint vectors of different time periods

Finally, we compute and show the performance of the classification algorithm with respect to best ranking performed per time period considered (Figure 44). We note that the average rate of true positives is fairly high (>0.8) for a very low false positive rate (0.001). We also note that even for a small time period of 1 day, the user fingerprint is capable of delivering about 72% of accuracy with 0% false positives. Also, longer periods do not necessarily mean better performance, as the fingerprint vectors can become similar with other users as well. The results are consistent in both testing months.

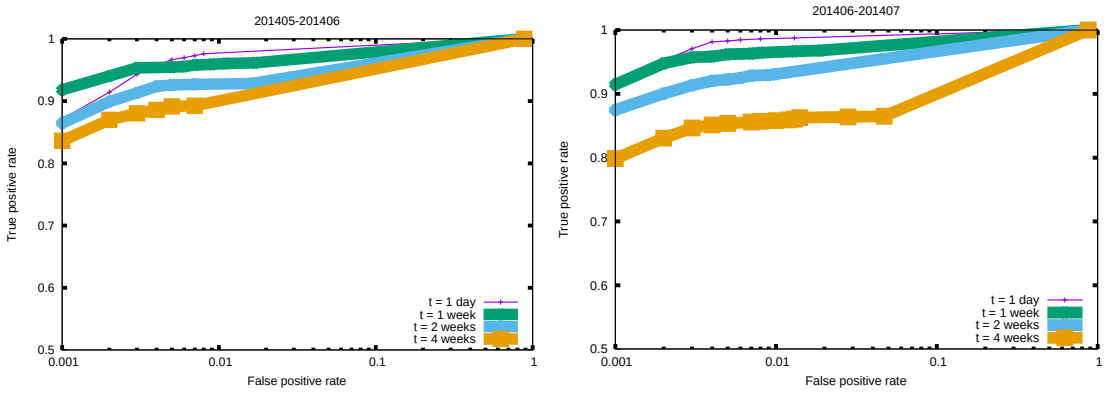

Figure 44 ROC curves of the method for fingerprints computed in different time periods, for two testing months

4.2.3 Next Steps

The experimental evaluation showed that the methodology proposed can perform well in authenticating users, by utilizing their mobile call records for extracting mobility patterns.

In the future we plan to investigate if more features extracted from mobile locations of users can improve the performance of the methodology with respect to true positives and false positives. We would also like to investigate if such new features allow the methodology to authenticate a user when his fingerprint captured is based on data from a smaller time period of a few hours. Finally, we would like to repeat the experimental evaluation to more datasets from different countries, to confirm that the methodology still applies in different cultures and mobility habits.

4.3 Browsing-based Authentication

This section reports on the design and the implementation of a service that is capable of authenticating users based on their web browsing behaviour. So far, we have focused on collecting real user web traces, evaluating how those traces can be used for authentication purposes, determining how stable behavioural signatures are, and the algorithms we will need to efficiently produce the ranking scores used.

4.3.1 Datasets

We acquired traces of real user web behaviour from two different datasets.

The first dataset (D1) is derived from traffic logs of an opt-in proxy for mobile users operated by a large telecom provider in Europe. This dataset represents mobile traffic that passes through the proxy but does not include HTTPS traffic. Users are identified with an internal proxy identifier that is not linked to any personal information.

The second dataset (D2) was collected via a browser plugin installed by a cohort of real users. The plugin captured all URLs that were entered into the address bar of the browser. User ids are fully

anonymized, and we only collect the fully qualified domain name of the visited page. We are also able to distinguish between HTTP and HTTPS requests.

Dataset	Number of Users	Number of Requests	Time Period
D1	330	2,272,479	Sept. 1, 2015 - Sept. 30, 2015
D2	67	799,539	Oct. 1, 2015 - Nov. 30, 2015

4.3.2 Evaluation of Suitability for Authentication

For web browsing behaviour to be useful for authentication, it has to be a viable form of identity. That is, users must be distinguishable, uniquely so ideally, from each other when comparing their browsing behaviour. One way of doing this is to compare the distribution of visits to particular sites of different users. We thus convert the raw web traces to frequency distributions. I.e., each user has associated with him a list of (site, visit count) tuples.

Next we looked for suitable metrics to compare these distributions. The most commonly used metrics in user similarity studies are Jaccard index and cosine similarity. The problem with Jaccard index is that it operates solely on set membership and therefore cannot capture properties like frequency/popularity. For this reason, and similar to previous work, we use the Vector Space Model as a representation of users' signatures and cosine similarity with tf-idf weights to calculate similarity. Tf-idf is widely used in information retrieval and it manages to reduce the impact of very popular terms.

Given two users u and v, the similarity of their browsing behaviour is calculated as follows:

$$\theta(sig_u, sig_v) = \frac{\sum_{i=1}^{n} u_i \times v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \times \sqrt{\sum_{i=1}^{n} v_i^2}}$$

where the values of the attributes u_i, v_i are the corresponding tf-idf values. For example:

$$u_i = tf_i \times (1 + ln(\frac{N}{df_i + 1}))$$

Where tf_i is the number of times user u has visited web page i, N is the total number of users and df_i is the number of users who have web page i in their browsing signature.

Next, we looked at the distribution of pair-wise similarity scores between users. By focusing in on D2, we are able to see the impact of HTTP vs. HTTPS traffic as well. The results are plotted in Figure 45.

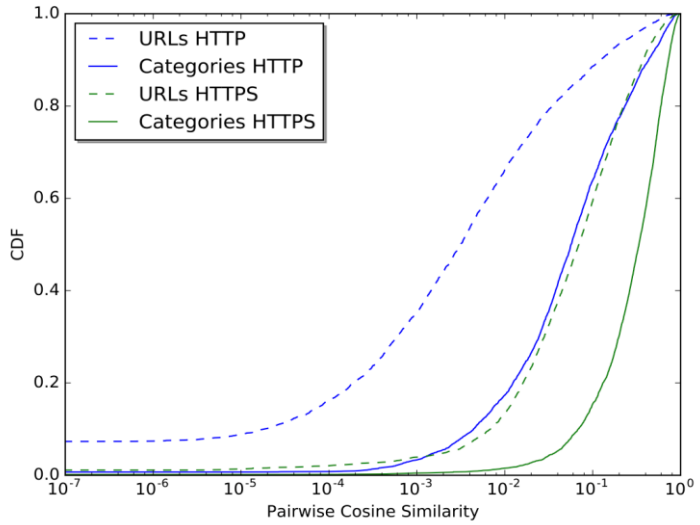

Figure 45 Pairwise similarity scores for users in D2.

From the figure we observe that the median similarity score between users, when considering only their HTTP traffic is roughly 0.001; when comparing the similarity of the online behaviour of any two randomly selected users, there is a 50% chance it will be less than or equal to 0.001. In other words, users are quite dissimilar from each other in terms of the frequency of the sites they visit, and thus this behaviour is suitable for use in an authentication system.

Next, we need to determine the stability of the browsing behaviour signatures. Here we need to answer two major questions: 1) over what period of time must a signature be captured, and 2) for how long is a signature “valid” for identifying a user. To answer these two questions we measured the effect of the signature capture period (which we call “quarantine”) on D1. We first set the quarantine period to be 1 day (the first day of September). I.e., the signature is built from a single day of user activity. We then compared the rank of the user as time progressed when compared to his signature for the month of September, the first week of October, and the second week of November. We also did the same for a 1 week quarantine period (the first week of September). The results can be seen in Figure 46.

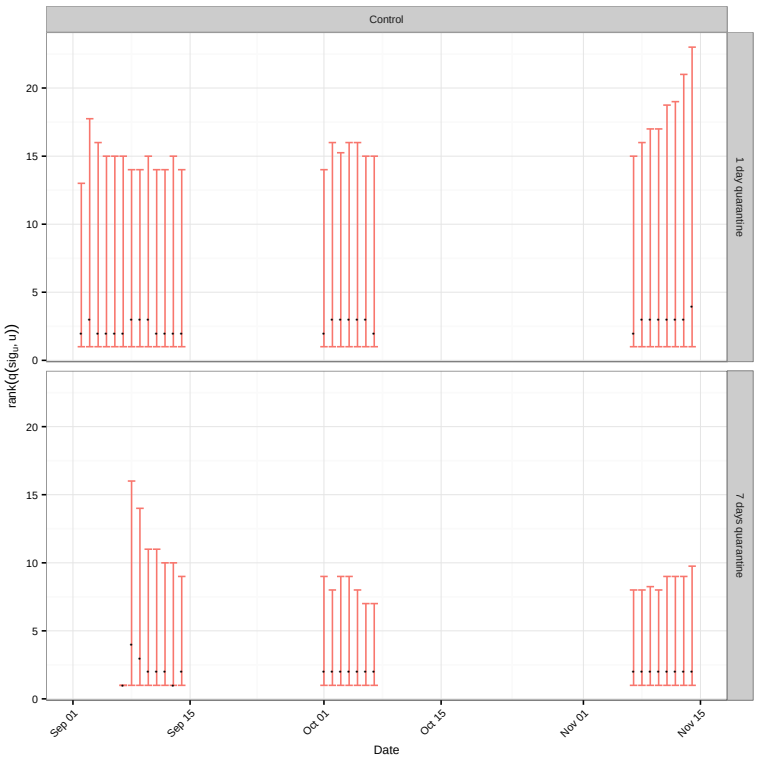

Figure 46 Effect of the signature collection period and stability over time

From the figure we make several observations. First, we note no significant difference between a 1 and 7 day quarantine period. I.e., the signature is able to identify a user about the same if it is only built off of a day’s worth of activity vs. a week’s worth. Second, we note that the signatures are quite stable over time: a signature taking a month in the past is generally sufficient to identify a user.

In the future, we will extend our experimentation to determine exactly at what point the signature stability breaks down. From here, we will devise a decay system, whereby a user’s signature is “rebuilt” by adding new information and removing old information.

Ranking Algorithm

To ensure the scalability of the authentication system in terms of computational resources, carefully designed data organization and query processing techniques are necessary. A naive approach to handle ranking is an exhaustive search algorithm. In an exhaustive search, given an internal browsing signature int_u, we simply enumerate through the entire list of external browsing signatures and compute rank(theta (int_u, ext_v)), returning the top-K results (i.e., the users with ranks 1 through K). While straightforward and easy to implement, exhaustive search necessarily requires O(n) rank(theta(int_u, ext_v)) calculations, which in turn can entail an excessive number of cosine similarity calculations. We thus focus our data management efforts on reducing the number of similarity computations performed, in order to reduce the total processing cost incurred.

Internal Data Storage: All data and indices are stored in main memory, although they are persisted to a database for fault tolerance; lightweight structures are used, with space requirements well within the memory available in basic commodity servers (in our experiments processing 10M requests results

in less than 2GB memory). User signatures are stored in an array in the form of vectors. The vector for u's signature comprises a unique user identifier and the respective u_i values. The index we use is similar to a (frequency-ordered) inverted file. That is, for every website of interest i, we maintain a list L_i which includes for every user signature a pair of the form <u, f_i> where u is the identifier of the user and

$$f_i = \frac{u_i}{\sqrt{\sum_{j=1}^{j=n} u_j^2}},$$

i.e., his tf-idf value for website i, normalized by the magnitude of u's signature. The list only contains pairs for signatures that include accesses to website i (i.e., where f_i > 0).

Query processing: A key observation is that we do not need to compute the exact rank(theta(int_u, ext_v)). Instead, it suffices to compute the K external signatures with the largest cosine similarity to int_u, and intervene only when ext_u is among them. Given int_u, we compute the K most similar external signatures to it by an algorithm similar to the threshold algorithm in [1]. Our objective is to save processing time by computing the similarity score (to int_u) of as few signatures as possible, by considering only the top parts of the sorted lists L_i (and ignoring the rest).

Specifically, in the first round of the algorithm, we consider the first pair in each sorted list, in a round-robin fashion. When considering list L_i, we evaluate its top pair < v, f_i >, i.e., we (access the entire vector of v from the signature array and) compute theta(int_u, ext_v). In an interim top-K result we maintain the K signatures evaluated so far that are most similar to nt_u, in decreasing similarity order; we denote by theta_K the cosine similarity (to int_u) of the K-th of them.

When all lists have been considered, we commence the second processing round, where we evaluate the second pair in each list (in a round-robin fashion, again), and so on. We define as the local threshold t_i for list L_i the value f_i in the last pair evaluated in that list. Due to the sorting in the lists, it holds that every non-evaluated pair in the unexplored part of the lists has a similarity
(to int_u) that is upper bounded by global threshold.

$$T = \frac{\sum_{j=1}^{j=n} ext_{v,j} \times t_j}{\sqrt{\sum_{j=1}^{j=n} ext_{v,j}^2}}$$

Thus, when T becomes smaller than or equal to theta_K, the interim result is guaranteed to contain the K most similar signatures to int_u (without loss of generality, we assume no ties). When that condition is met, the algorithm terminates and the interim result finalized.

4.4 Keystroke-based Authentication

ReCRED also implements a behavioural second factor authentication mechanisms based on typing style (Keystroke Dynamics) used in conjunction with local device-based password-less protocols in order to address key security and privacy issues such as resilience to device loss, theft and impersonation.

While the basic concepts of the keystroke dynamics approach in ReCRED are described in Deliverable D3.1, in this deliverable some more specific to the implementation issues are discussed.

The scope of the b-verifier application is to use the unique way a user is typing to the keyboard of their device as a second factor authentication addressing the threat of the unauthorized access to a service from a stolen device (even if the attacker knows the credentials to unlock the device). Its main purpose is to confirm the identity of the user, rather than uniquely identify it.

The application uses a client-server architecture. The client resides on user's device while the server component resides at the Behavioural Authentication Authority. Both components take part in the two phases that the application supports, the training and the verification mode. The client application is an Android custom keyboard application that replaces system-wise the device's default keyboard.

Training phase

During the training mode/phase, the mobile application captures and transmits in real-time to the server the typing pattern of the user. If the device does not have internet connectivity, the patterns are not sent neither stored to the device for security reasons.

The capturing of the typing patterns is done from every application that uses the ReCRED custom keyboard as a user might type in different ways depending the application (messaging, email, comments to sites, chatting, search or inserting data in web forms, writing documents etc). For this reason except from the typing pattern of the user, the application id is also transmitted and stored in the BAA's database.

Keystroke dynamics is a well-studied area and a lot of approaches have been proposed. We have studied the literature and found that the majority of the approaches use the timings of the key-press (key-press, key-release), the timing between non-consecutive key-presses, word per minute ratios etc.

Apart from these features we have experimented on using features that identify users style of writing, i.e. error rate (user presses the backspace key) and use of punctuation. We aim to combine these features to propose a novel authentication scheme. Thus, as typing pattern, we define the set of features that the application captures. Such features currently include

- Keystroke interval
- Words per minute
- Characters per minute
- Punctuation percent
- Backspace percent
- Mean word length

Regarding the keystroke interval, timings between key-down to key-down and key-up to key-up are recorded.

This feature set is giving us the opportunity to test various combinations and choose the one that has the better performance in terms of correct verification.

A vector of this feature set along with the application id that the custom keyboard is running, the device ID and a timestamp is sent to the server side where the server component receives it and performs a set of operations.

- Determines if the received vector is to be used for training or for verification (matching). This is done by checking an indicator in the database. This indicator is set to “training” mode if the number of the received typing patterns for the specific device and application is below the threshold that is needed to create a valid typing signature for the user. This threshold will be defined after conducting several experiments on a large set of users and using various clustering algorithms and parameters.
- The vector is stored to the database.
- After the necessary number of vectors have been gathered, a clustering algorithm clusters the features of the same device and application and produces a unique signature. This signature is stored to the database and will be used for the verification (matching) process.
- The training phase has been concluded for the specific device and application and the training indicator is off.
- Following this, each new vector that arrives from the device is compared with the user signature using a distance measure (Euclidean distance). This matching score is stored along with the newly arrived vector.
- Shall a new vector arrive from the same device id and the same application, this step will be repeated and the new score and vector will be stored. Thus, the database will contain the latest score for every application for the specific device.

Verification phase

When a user tries to access a service (e.g. e-banking service) he provides a pin or password to the service. If this is correct the service can also ask the Behavioural Authentication Authority (BAA) if the user behaves as normally, regarding his typing style.

The service provider contacts the BAA sending the device id that has requested the access.

The BAA queries the database and retrieves all the related to this device id scores for every application.

If there are no typing signatures yet, the BAA responds that it is not possible to answer about that device id.

If there are more than one signatures, the BAA will have either to send them all to the service provider, or, preferably to calculate an average score from all the applications. Weights may be applied to this calculation to promote the recently calculated scores against the older ones.

The BAA sends this final score (a percentage) to the service provider.

The service provider decides if this score is acceptable based on a threshold. This threshold depends on the criticality of the service or the specific action the user is requesting and is for the service provider to decide.

The authentication attempts are stored in a log in the Identity Consolidator.

All typing signatures have an expiration date after which a new training phase shall took place while old data are erased.

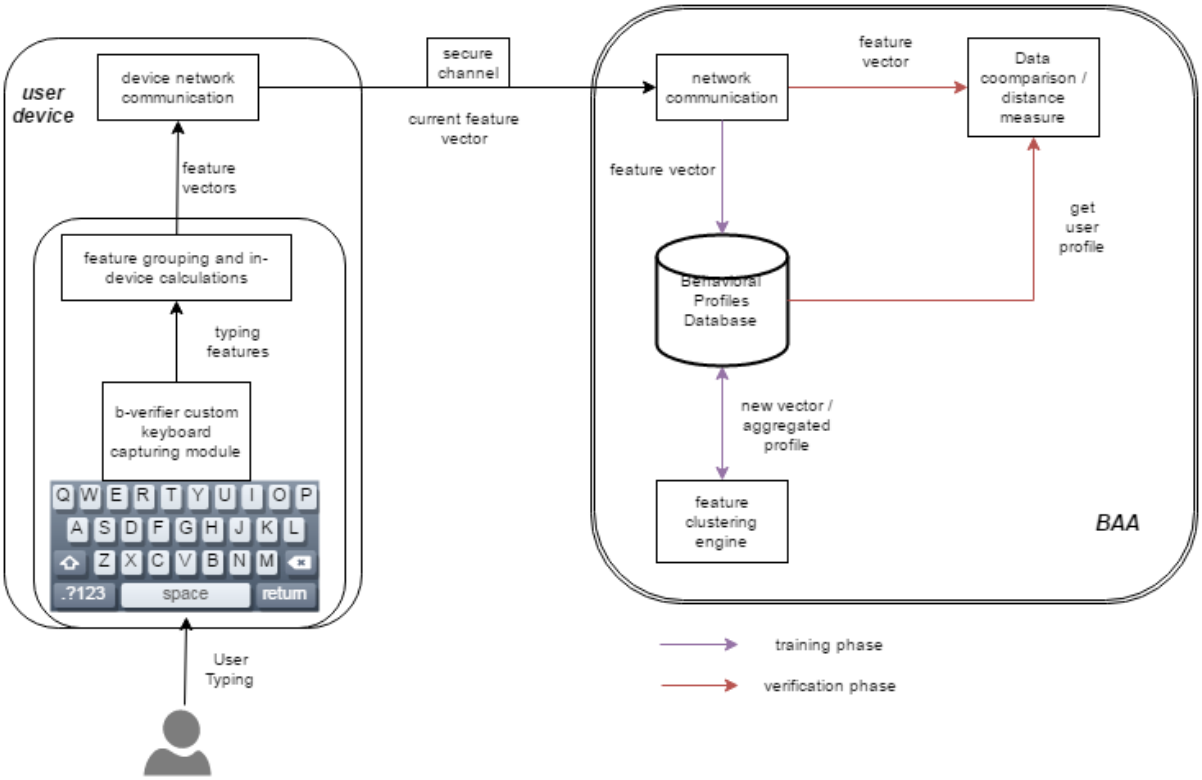

Figure 47 b-verifier architecture (mobile device and BAA server)

Database

The database that stores the typing signatures (Behavioural Profiles Database) is hosted by UPCOM and the related API is used for all the actions (insert, update, delete). Currently a temporary database is used in WEDIA servers with a table that holds the described features.

Methodology

In this section we describe the methodology we are following towards the implementation of the b-verifier application.

In order to define the correct set of features that will be captured and used for the application and test its efficiency we follow the below steps:

1. Create javascript code that is used in web browser and captures a set of features. In this step a basic source code evaluation is made to identify errors in the algorithm and bugs.

2. Use this javascript at a test environment (test web page) for capturing validation. In this step we collect a first set of data and quickly try to see if the application works as expected. (live at: http://bverifier.vmlab.wedia.gr/).

3. Use this javascript to a live page to collect real user's input. We incorporate the developed javascript in the comments box of WEDIA's site www.gazetta.gr in order to collect data from real users.

4. The data collected are stored in a database on the server. The database is a temporary version of the final database.

5. Evaluate the data collected using various data mining algorithms that cluster the features. This step requires data cleaning and post-processing. After that samples are tested with WEKA data mining tool to check the efficiency.

6. Decide on the features that will be used after studying the output of the data mining algorithms. Decide on the best clustering algorithm that will be used to create the user profile.

7. Appropriately change the application code to capture the correct set of features and repeat all the steps in order to conclude to the set of features that best fits our RECRED requirements.

8. Decide the best distance measure to perform the user verification phase. This step requires extended study and experimentation.

9. In parallel with step 8, port the javascript code to an Android mobile application that is a custom keyboard. Setup the communication with the server and address all issues related to the development of an android application.

4.4.1 Current status - experiments

Apart from studying the literature and defining the technologies to be used, we have performed a first round of our methodology, step 1 through 5. Our initial set of features used for five test users and roughly evaluated with the use of Bayesian classification with the WEKA tool, shows promising results. Although, a first sample from real users collected from the gazetta.gr and 500000 feature vectors uncovered some bugs and issues that we had to resolve. Thus, a second version of the code deployed in the live environment and collected a smaller set of 80000 records to re-evaluate the captured features.

In parallel the development of the custom keyboard application has started and we are porting the code to the android application.

No experiments have been performed in the touch screen environment of the android application. Various parameters and issues have to be considered such as the device screen size, touch screen sensitivity and responsiveness, OS compliance, communication with the server etc.

Next steps include the evaluation of the second data set collected from the real users, the final definition of the feature vectors, the evaluation of the distance measures, the connection to the UPCOM database server and the completion of the implementation of the custom keyboard

application. Furthermore, the test and experiments with several different devices and a set of real users will take place.

While small scale experiments to this point have shown that the selected features are sufficient to distinguish different users using Bayesian classification algorithms, we will conduct experiments with a larger volume of real users in order to validate the set and combination of features.

Several verification experiments will be also conducted to validate the efficiency of the clustering algorithm (K-means and EM) and the distance measure.

Technologies

For the client application of the mobile device, the open source keyboard *AnySoftKeyboard*[2] is used as a basis. In order to capture the key interactions the *date4j*[3] library is used. The *okHttp*[4] library is used for securely transmitting the data to the server.

On the server side, the *Laravel*[5] PHP framework is used.

For the classification and clustering algorithms the WEKA[6] open source data mining tool is used.

[2] https://github.com/AnySoftKeyboard/AnySoftKeyboard
[3] http://www.date4j.net/
[4] http://square.github.io/okhttp/
[5] https://laravel.com/
[6] http://www.cs.waikato.ac.nz/~ml/weka/

5 Mobile Connect

Identity Federation allows a user to use the same set of credentials to obtain access to multiple resources, distributed across multiple service providers. Connecting ReCRED platform to Mobile Connect service is a good demonstration of internetworking of multiple identity ecosystems.

Mobile Connect is the GSMA Personal Data programme focused on positioning Mobile Network Operators (MNOs) as trusted providers of identity services to third party service providers. The programme identifies a set of propositions (including authentication, validated identity, enhanced profile, attribute brokerage) that collectively are referred to as Mobile Connect.

In Mobile Connect, operator discovery is the process of finding out the correct IDP (home operator for the user) to which the user belongs to and getting the Mobile Connect endpoints details to connect to the MNO for identity services. Mobile Connect is fundamentally based on the OpenID Connect standard, where the MNOs are primarily responsible to authenticate end-users and to provide user attributes to the service providers.

5.1 APIGEE OneAPI Exchange Discovery Service

In order to facilitate the federation process, APIGEE OneAPI Exchange provides a discovery service which helps service providers to find out the correct mobile network operator for a given user. There are three steps in the discovery process:

- Operator Discovery: APIGEE OneAPI provides a mechanism to discover the home/serving operator. The Operator Discovery is achieved based on the available context e.g. MCC+MNC read from the device SDK, IP address of the mobile access point or in the case when there is no context available then asking the user to either provide the MSISDN or select the Operator from the Operator Selection Interface.

- Operator End-point: Once the discovery is done, it is then the responsibility of APIGEE OneAPI to provide the Mobile Connect end-points for the discovered MNO to the respective service provider.

- SP/Client Credential management for Federation: One of the key aspects of the Discovery and Federation architecture needs to be the secure usage of credentials across participating Operators so that the SP may register with one Operator (the developer Operator) and use the credentials to get access to the Identity Services from other participating Operators. APIGEE OneAPI provides client credentials to the on-boarded service providers which allow them to access other MNOs.

5.2 Identity Consolidator as a Discovery Service

In order to support discovery service in the identity consolidator the following two options can be considered:

Option 1: Identity Consolidator with APIGEE OneAPI Exchange for Discovery

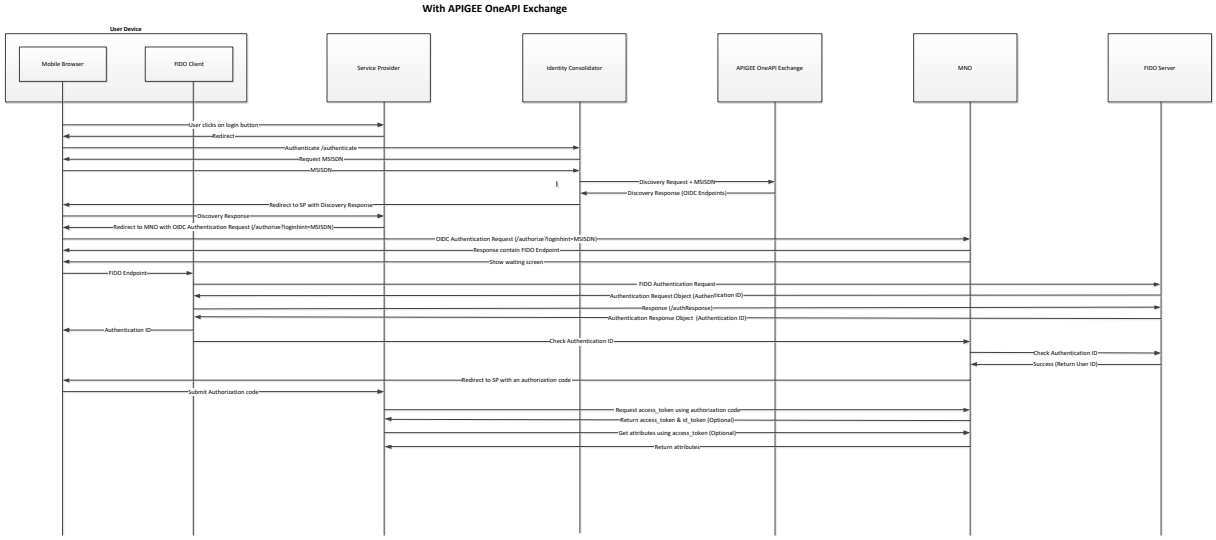

Figure 48 Interoperation between Identity Consolidator and APIGEE OneAPI Exchange

Steps:

1. User accesses service provider application using mobile browser.
2. He/she then clicks on Login button on the service provider application.
3. Service provider redirects the user to the identity consolidator.
4. Identity consolidator asks the user to provide mobile number (MSISDN).
5. User provides his/her mobile number (MSISDN).
6. Identity consolidator makes a discovery request to APIGEE OneAPI Exchange and passes this mobile number in the request.
7. APIGEE OneAPI Exchange identifies the respective MNO based on the mobile number provided and returns a discovery response containing MNO OIDC endpoints to the identity consolidator.
8. Identity consolidator then redirects the user browser to the service provider along with the discovery response.
9. Service provider makes OIDC authentication request to OIDC endpoint of MNO and provides MSISDN in the request.
10. In response, MNO sends a response containing FIDO endpoint to the browser and also shows the waiting page.
11. Browser sends this response to the FIDO client.
12. FIDO client initiates FIDO authentication towards that FIDO endpoint.
13. FIDO server then sends FIDO authentication request (containing authentication ID) to the FIDO client.
14. FIDO client coordinates with the respective FIDO authenticator.
15. FIDO authenticator asks user to scan his/her finger.
16. FIDO authenticator signs FIDO response with the private key and sends this response to the FIDO client.
17. FIDO client sends this response to the FIDO server.
18. FIDO server verifies and validates the response.
19. If the validation is successful, then FIDO server provides successful authentication response to the FIDO client.
20. FIDO client then sends the authentication ID to the mobile browser.
21. Mobile browser in turn sends this authentication ID to the MNO
22. MNO checks this authentication ID with the FIDO server.

23. FIDO server sends success response along with the user id to the MNO
24. MNO then provides authorization code to the service provider
25. Mobile browser submits this authorization code to the service provider.
26. Service provider using this authorization code asks MNO for the access and id token.
27. MNO provides access and id token to the service provider.
28. Service provider then asks MNO for the user attributes using an access token.
29. MNO provides user attributes to the service provider.

Option 2: Identity Consolidator without APIGEE OneAPI Exchange for Discovery

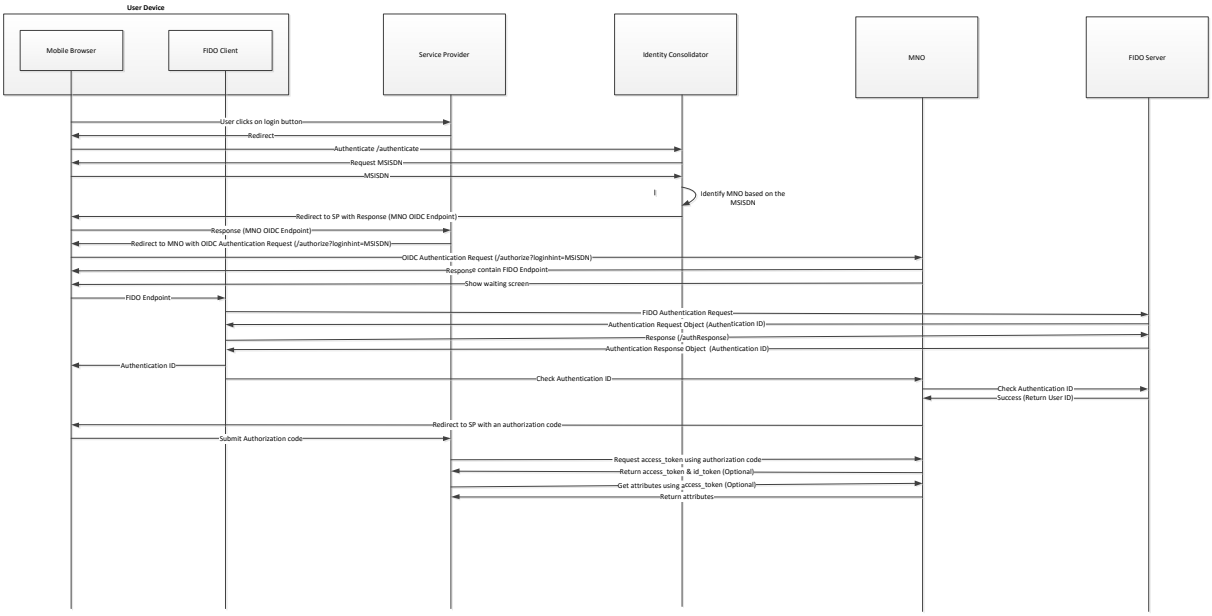

Figure 49 Replacement of the APIGEE OneAPI Exchange with the Identity Consolidator

Steps:

1. User clicks on Login button on the service provider.
2. Service provider redirects the user to the identity consolidator.
3. Identity consolidator asks the user to provide mobile number (MSISDN).
4. User provides his/her mobile number (MSISDN).
5. Identity consolidator checks MSISDN and corresponding MNO internally and sends a response containing MNO OIDC endpoints to the SP via browser.
6. Service provider makes OIDC authentication request to OIDC endpoint of MNO and provides MSISDN in the request.
7. In response, MNO sends a response containing FIDO endpoint to the browser and also shows the waiting page.
8. Browser sends this response to the FIDO client.
9. FIDO client initiates FIDO authentication towards that FIDO endpoint.
10. FIDO server then sends FIDO authentication request (containing authentication id) to the FIDO client.
11. FIDO client coordinates with the respective FIDO authenticator.
12. FIDO authenticator asks user to scan his/her finger.
13. FIDO authenticator signs FIDO response with the private key and sends this response to the FIDO client.
14. FIDO client sends this response to the FIDO server.
15. FIDO server verifies and validates the response.
16. If the validation is successful, then FIDO server provides successful authentication response

to the FIDO client.
17. FIDO client then sends the authentication ID to the mobile browser.
18. Mobile browser in turn sends this authentication ID to the MNO
19. MNO checks this authentication ID with the FIDO server.
20. FIDO server sends success response along-with the user id to the MNO
21. MNO then provides authorization code to the service provider
22. Mobile browser submits this authorization code to the service provider.
23. Service provider using this authorization code asks MNO for the access and id token.
24. MNO provides access and id token to the service provider.
25. Service provider then asks MNO for the user attributes using an access token.
26. MNO provides user attributes to the service provider.

The latter option contradicts Mobile Connect specification, because identity consolidator does not provide Mobile Connect Discovery APIs. Discovery service is one of the features of the mobile connect via OneAPI Exchange. The objective of mobile connect integration with ReCRED project is to demonstrate the interoperability between ID consolidator and mobile connect solution provider (this is called Mobile Connect Accelerator - MCX). With the adoption of the latter option, including discovery service in the ID consolidator, the ReCRED framework will include the role of a Mobile Connect Solution provider. This means, ReCRED also has to comply with all the technical specifications that are part of the mobile connect solution prescribed by GSMA. In addition, as an MCX provider, there is a requirement for providing APIs and SDK to on-board service providers and operators through a developer portal (e.g., *https://developer.mobileconnect.io/*, *https://github.com/Etalio*). We therefore adopt option one and dictate the Identity Consolidator to use the APIGEE OneAPI Exchange, in order to achieve a more flexible and interoperable integration of the ReCred framework with Mobile Connect.

6 QR Login

QR Login permits the identity transfer between the authenticated smartphone and a desktop. The identity transfer process is realized by using the device camera to scan a QR code and by performing a cryptographic protocol with a server entity (QR Authenticator). The QR Login architecture was described in Deliverable 3.1 (Description of device centric authentication protocols and technology support).

The QR Login comprises two main modules: the client (deployed as an Android application) and the server (QR Authenticator). The client and the server communicate through REST API, using JSON encoded messages. The JSON encoding is realized by employing the Gson[7] library.

The client software components are the following:

- Transport – is in charge with exchanging messages with the server through an already authenticated channel. This module exposes a robust interface in order to abstract the logic which authenticates the user.
- QR scanning – is in charge with scanning the QR code displayed on the desktop side
- User interface

The server entity exposes REST API in order to exchange messages with the Android client and with the service provider. The messages exchanged between the QR Authenticator and the Service provider are encoded as JWT tokens. For the JWT implementation the library auth0/java-jwt[8] was used.

The QR Authenticator the main modules are:

- REST API – exchanges messages with the Service Provider and with the Android client
- Storage – stores QR session related data
- Logging/Audit
- QR Authenticator Logic

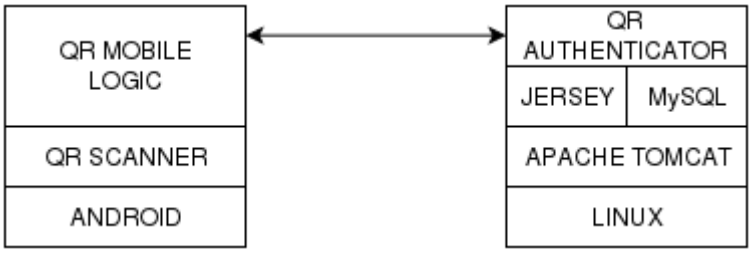

Figure 50 Interaction between client and QR Authenticator

The interaction between the Android client and the QR Authenticator, along with the deployment environment is described in Figure 50.

[7] https://github.com/google/gson
[8] https://github.com/auth0/java-jwt

7 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 and queries the status label of user account. Each user account has a global status label – “locked” or “unlocked” – that defines whether the Online Service accepts 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.

7.1 Account locking functionality in ReCRED

Account locking functionality is an integral part of Account Management Module (AMM) inside an Identity Consolidator. The Account Management Module communicates with an outer world via a 3rd party API, defined in Deliverable D4.1.

7.1.1 Account locking principle

The Account locking functionality starts at the Online Service’s side. Online Service has to register with AMM in order to obtain credentials that identify Online Service within ReCRED ecosystem. Those are used to authenticate requests made to the AMM.

A user’s name and domain of the Online Service define user’s account at AMM’s side. This information will be collected during Identity Consolidation process.

Once a user tries to log in to the Online Service at the web page, the Online Service sends a request to AMM with user’s login and service credentials. This request is used to check the global status label for that service and particular user. The AMM respond contains global status label state -locked/unlocked - and the service behaves accordingly: it either allows the user to enter the service, or it stops the authentication process.

The user can check the locking status of his account by querying the AMM. In addition to it, he can change the status of any of his accounts.

The following paragraphs describe particular operations needed for Account locking functionality. Please note that the authentication to the calls is not depicted for brevity. The authentication process is describe in Section 7.1.3.

Register Online Service

A third-party Online Service registers its application with Account Management Module (AMM). An application in this context means a functionality that allows user to log in in the Online Service. The service receives an applicationID that identifies the Online Service and the server receives also an applicationSecret that is used to authenticate requests made to the AMM. Hooks in the Online Service’s code must be implemented in order to communicate with AMM and to pairing user accounts. The registering of an Online Service happens on ReCRED’s web page. A developer of an Online Service has to register, login and create a new application for which he/she obtains the applicationID and applicationSecret.

Pair account

A user has to pair his account to create a verifiable link between an Online Service and a user’s account. Figure 51 depicts the process.

The user initiates pairing on the Online Service webpage after successful login to the Online Service (1). Under his online account, the Online Service offers a possibility to use ReCRED’s Account locking functionality, which the user selects and is requested to enter a pairing token (2). To obtain a pairing token, the user must open the ReCRED mobile app and trigger the request for pairing a new Online

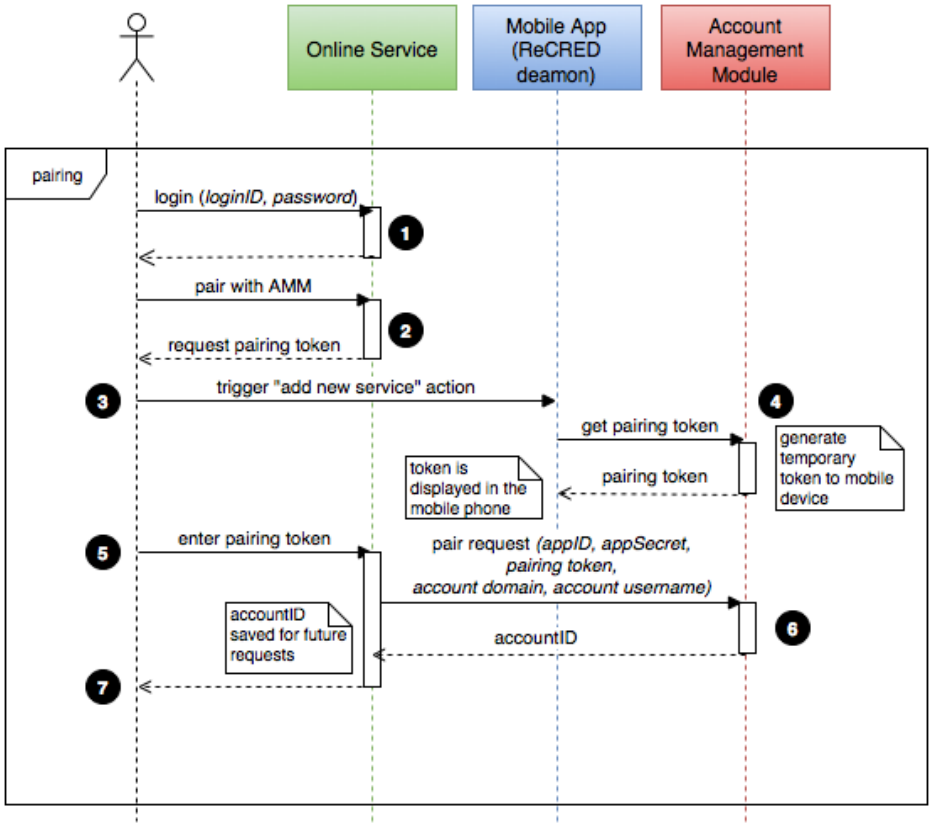

Figure 51 Pair account message flow

Service with Account Management Module of ID consolidator (4). The AMM generates a token for that particular user and returns it back to the mobile app, which shows the token on the mobile phone

display. The user fills the token in the form on Online Service web page (5), which then requests pairing of Online Service with AMM (6). If the token corresponds, a domain of the Online Service is stored in AMM’s database along with user’s account username. An accountID is generated for the user that is returned to Online Service and serves for future requests on user’s behalf user. Finally, if the process ended with success, it is reported back to the user (7).

Once the pairing is completed, the user can lock or unlock his/her accounts as described in Section 0.

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 52 are executed.

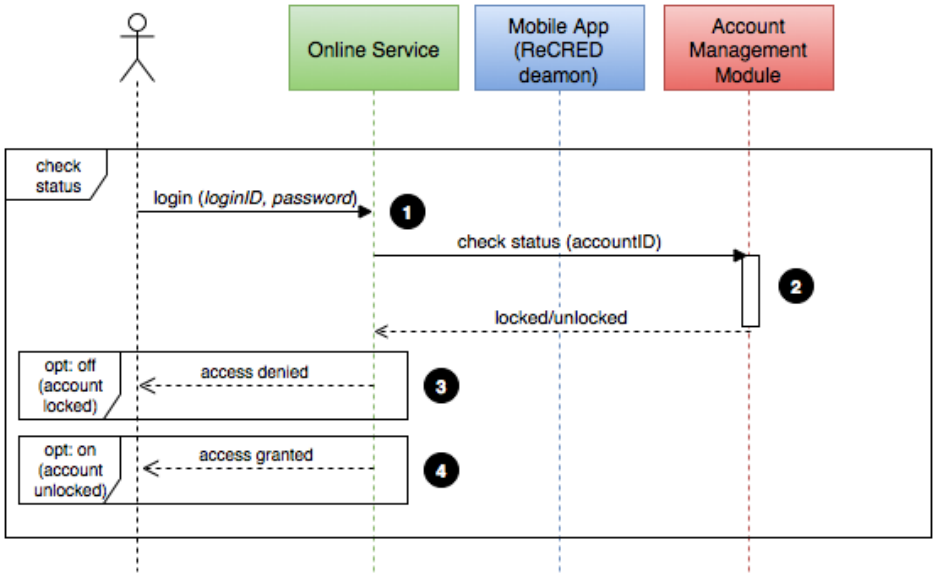

Figure 52 Check status message flow

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 53 depicts the process.

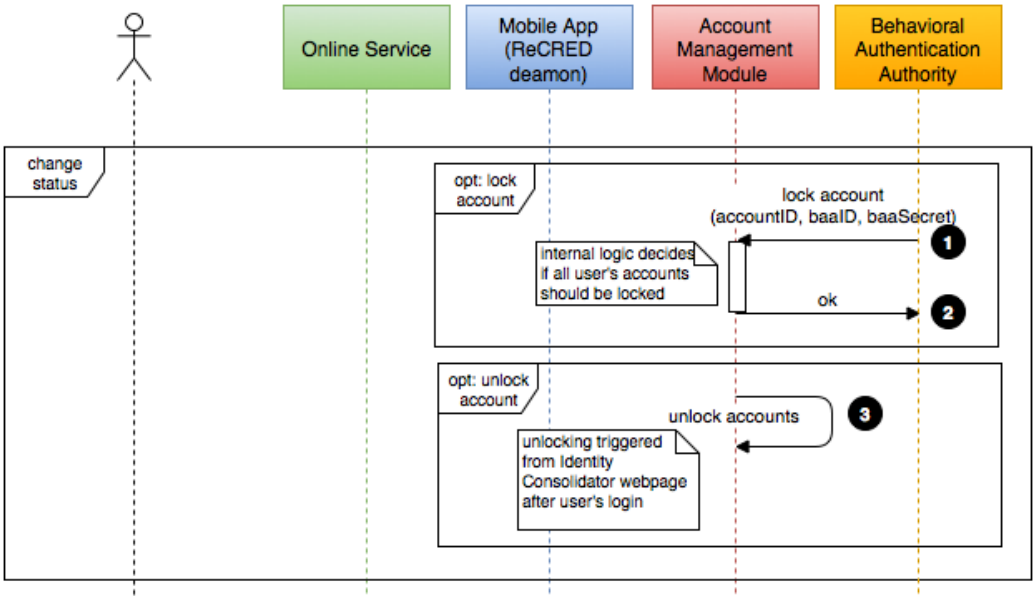

Figure 53 Change status message flow

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.

Unpair account

The user may decide to unpair some of his accounts from ReCRED’s Account locking mechanism. Figure 54depicts the process.

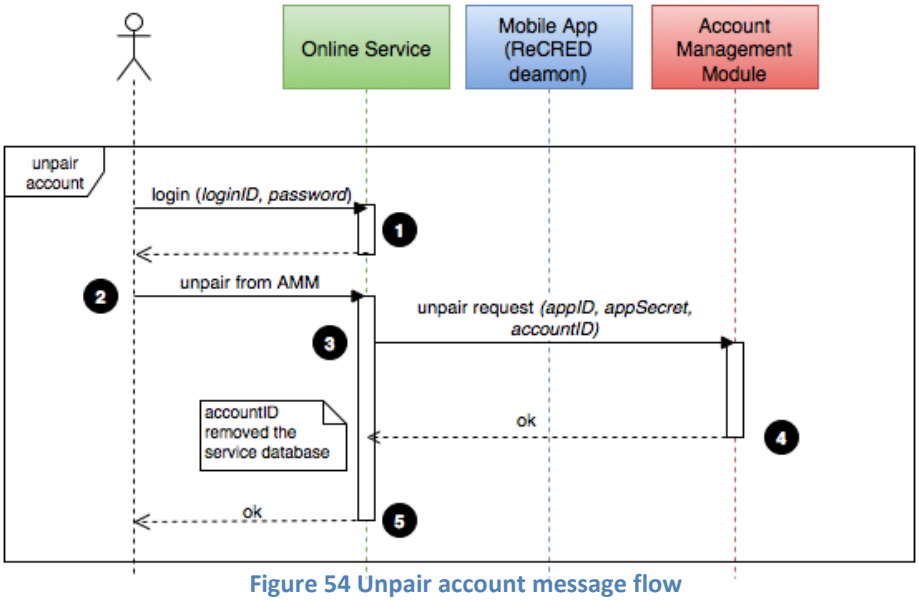

Figure 54 Unpair account message flow

First, the user logs in to the Online Service he/she wants to unpair from AMM (1). Then the user triggers an action “unpair from ReCRED”, which must be available on the webpage of the Online Service (2). The Online Service then sends an unpair request, authenticated by its applicationID and applicationSecret (3). The user’s account on Online Service is identified by the accountID, which is stored in Online Service database. After a response from AMM (4) the user’s accountID should be discarded rom Online Service database, because any subsequent pairing of the same user with the same Online Service will result in different accountID anyway. The last step is to report successful unpairing action to the user (5).

7.1.2 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 with Online Services, Behavioral Authentication Authorities and Identity Consolidator storage. Figure 55 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 if not related to Account locking mechanism.

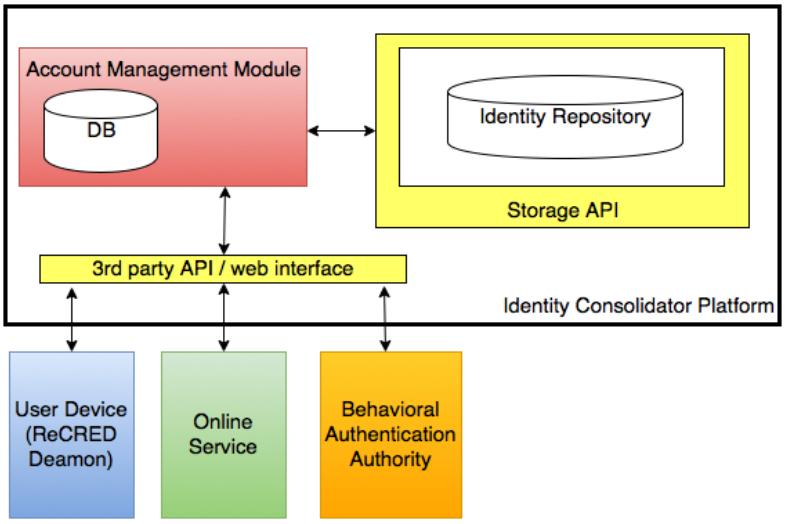

Figure 55 Account locking architecture

The AMM communicates with the outer worlds through a 3rd party API, which is a REST API defined in Deliverable D4.1. The entry points, particular methods and parameters are detailed in Section 7.1.4. The subset of 3rd party API, related to Account locking functionality, will be referred to as Account locking API further in the rest of this chapter.

Internally, user-related information such as 3rd Party Accounts are stored in the Identity Repository. As defined in Deliverable D4.1, 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 Behavioural Authentication Authority 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 D4.1.

7.1.3 Authentication

Every request to Account locking API must be signed and accompanied with two authentication headers: Authorization and Date.

The Authorization header consists of requestID and requestSignature. The requestID is either applicationID, userID or baaID, depending which party communicates with the AMM. An applicationID is used for communication for communication between Online Service and the AMM, a userID for

communication with ReCRED daemon in user device and the AMM, and a baaID serves as an identifier of a Behavioral Authentication Authority.

The requestSignature is a concatenation of request URL, request method (POST, PUT, GET, DELETE), and date of the request and all parameters of the request call. The requestSignature must be signed using the HMAC-SHA1 algorithm and the secret that was obtained during registration. A user has its own userSecret, each Online Service its own application Secret and each BAA its own baaSecret.

The Date header contains the value of the current UTC date and time and must be the same as in the Authorization header.

7.1.4 API description

The following paragraphs detail the Account locking API. For description purposes we expect that the endpoint of Identity Consolidator will have the following URL:

https://amm.recred.com/api/v1/

This address serves only for demonstration purposes and might be different after all integration work later during the project.

Obtain pairing token

To pair user’s account from the Online Service web page, the user has to request a pairing token.

Authentication

Authentication of the request has to be performed as described in Section 7.1.3. The requestID is userID in this case and the Authorization header is signed with userSecret.

Request

Perform a HTTP GET request on the following endpoint:

https://amm.recred.com/api/v1/pair

Response

The response from the API is the following:

```
{"data":{"token":"..."}}
```

where token is an 8-characters token that a user has to enter on the Online Service web page.

Errors

The following is a list of possible error conditions that can occur when trying to use this API:

- Error 205: Token already issued. This error will be returned if the user already asked for a pairing token and it still has not expired. The expiration period for a token is 60 seconds.
- Error 401: Missing parameter in API call. This error will be returned for all wrongly formatted requests.

Pair account

To pair the user's account and Online Service, the Online Service has to call Pair account API. The same procedure applies to pair user's account with a Behavioural Authentication Authority. The BAA then acts as an Online Service with restricted access to operations of the user's account.

Authentication

Authentication of the request has to be performed as described in Section 7.1.3. The requestID is applicationID, the identifier of the Online Service. The Authorization header is signed with applicationSecret. In case of BAA requesting the pairing process, a baaID and corresponding baaSecret are used.

Request

Perform a HTTP GET request on the following endpoint:

https://amm.recred.com/api/v1/pair/{token}

where token is 8-characters long token that has been returned in previous API call for obtaining pairing token. The request has to be accompanied with the following data content:

```
{"data":{"account_address":"...",     "account_username":"..."}}
```

where account_address is the top-level domain of the Online Service and the account_username is user's login for that particular Online Service.

Response

The correct response from the API is the following:

```
{"data":{"accountID":"..."}}
```

where accountID is an alphanumeric 64 characters long token that can be used to uniquely identify the account that is being paired with this application. This value should be stored to be used later in subsequent calls such as status or unpair.

Errors

The following is a list of possible error conditions that can occur when trying to use this API:

- Error 205: Account and application already paired. This error will be generated when a user, who has already paired the account, has generated the token.
- Error 206: Pairing token not found or expired. This error will be returned when the pairing token has expired. The expiration period for a token is 60 seconds. A user has to enter a pairing token in the Online Service web page within this time. This error also indicates, that the pairing token does not exist. Such situation can happen when someone randomly generate a token and tries to pair an account with user's name.
- Error 401: Missing parameter in API call. This error will be returned for all wrongly formatted requests.

Check status

Authentication

Authentication of the request has to be performed as described in Section 7.1.3. The requestID is applicationID, the identifier of the Online Service. The Authorization header is signed with applicationSecret.

Request

Perform a HTTP GET request on the following endpoint:

`https://amm.recred.com/api/v1/status/{accountID}`

where accountID is an alphanumeric 64 characters long token that can be used to uniquely identify the account which status is requested. The accountID has to be obtained in advance during the Pair account procedure.

Response

The correct response from the API is in the following format:

```
{"data":{"operations":{"applicationId":{"status":"..."}}}}
```

where the applicationID is the applicationID from request Authentication header and the status is one of the possible values of status field: locked or unlocked.

Errors

The following is a list of possible error conditions that can occur when trying to use this API:

- Error 201: Account not paired. This error occurs when the requested accountID has not been paired before with the application requesting the status check.
- Error 401: Missing parameter in API call. This error will be returned for all wrongly formatted requests.

Change status

The status on an account can be changed by the user or by any Behavioral Authentication Authority the user chose to keep track of his behavior and act as a countermeasure in case the mobile device is stolen.
The user changes account status from Identity Consolidator web interface after login.
BAA calls the Account locking API anytime it detects suspicious behavior that might mean the identity of a user has been compromised.

Authentication

Authentication of the request has to be performed as described in Section 7.1.3. The requestID depend on the party which requests the status change:

- If the user changes status of some of his accounts, the request ID is userID. The Authorization header is then signed with userSecret.
- If a BAA requests the status change of a user, the requestID is baaID. The Authorization header is then signed with baaSecret.

Request

Perform an HTTP POST request on the following endpoints, depending on what action is requested:

https://amm.recred.com/api/v1/lock/{accountID}
https://amm.recred.com/api/v1/unlock/{accountID}
where accountID is an alphanumeric 64 characters long token that is used to uniquely identify the account which status change is requested. The accountID has to be obtained in advance during the Pair account procedure.

Response

The response to change status request is empty in case everything goes well.

Errors

The following is a list of possible error conditions that can occur when trying to use this API:

- Error 111: BAA not authorized. This error occurs if the BAA is not allowed to change user’s account status, because the user has not approved the BAA to act so.
- Error 201: Account not paired. This error occurs if the account is not paired with the BAA or when the accountID is wrong.
- Error 401: Missing parameter in API call. This error will be returned for all wrongly formatted requests.

Unpair account

The Account can be unpaired from an Online Service on a Behavioral Authentication Authority.

Authentication

Authentication of the request has to be performed as described in Section 7.1.3. The requestID is applicationID, the identifier of the Online Service, or the baaID, the identifier of the BAA. The Authorization header is signed with applicationSecret or baaSecret, respectively.

Request

Perform an HTTP GET request on the following endpoints:

https://amm.recred.com/api/v1/unpair/{accountID}

where the accountID is an alphanumeric 64 characters long token that is used to uniquely identify the account which is to be unpaired. The accountID has to be obtained in advance during the Pair account procedure.

Response

The response to unpair account request is empty in case everything goes well.

Errors

The following is a list of possible error conditions that can occur when trying to use this API:

- Error 201: Account not paired. This error occurs when the accountID is wrong or if the account is not paired with the particular Online Service application or BAA.
- Error 401: Missing parameter in API call. This error will be returned for all wrongly formatted requests.
-

List accounts

To provide a user with web interface to manage his accounts, the List accounts request becomes useful. Requesting a list of users accounts returns all accounts of the user and their status.

Authentication

Authentication of the request has to be performed as described in Section 7.1.3. The requestID is the userID in this case and the Authorization header is signed with the userSecret.

Request

Perform a HTTP GET request on the following endpoint:

https://amm.recred.com/api/v1/list

Response

The response from the API is the following:

```
{"data":{"account_address":"...",     “account_username”:”...”,
“status”:”...”, “accountID”:”...”}}
```

where

- account address contains the top level domain text string of the Online Service. In case of BAA this field represents the top level domain or the name of the BAA.
- account_username contains the user name on the Online Service web page. In case of AA this field represents the name under which BAA recognizes the user.
- status field contains the global status label of the account: locked or unlocked.

- accountID contains the accountID which the Online Service or BAA may use to handle user’s Account locking status.

Errors

The following is a list of possible error conditions that can occur when trying to use this API:

- Error 401: Missing parameter in API call. This error will be returned for all wrongly formatted requests.

User history

User history of all Account locking-related operation can be listed using the User history part of Account locking API.

Authentication

Authentication of the request has to be performed as described in Section 7.1.3. The requestID is the userID in this case and the Authorization header is signed with the userSecret.

Request

Perform a HTTP GET request on the following endpoint:

https://amm.recred.com/api/v1/history/{from}/{to}

with optional parameters:

- from denotes the unix timestamp to list the user’s history from
- to denotes the unix timestamp to lit the user’s history to

Response

The response from the API is the following:

```
{"data":{“userID”:”...”,“count”:”...”,  “history”:{“t”:”...”,
“id”:”...”,  “source”:”...”,  “action”:”...”,  “was”:”...”,
“is”:”...”}}}
```

where

- userID field contains the identifier of the user as stated in Authentication header.
- count contains number of records in history
- history contains history of all events
  - t field contains the unix timestamp of time at which the action happened
  - id is the identifier of a party that initiated the action (applicationID, baaID, userID)
  - action contains the action identifier of action performed. The value is one of the following:
    - pair
    - status

        - lock
        - unlock
        - unpair
        - list
        - history
    - was field contains the previous value of the global status label: locked or unlocked
    - is field contains the value of the global status label after the action is performed: locked or unlocked

Errors

The following is a list of possible error conditions that can occur when trying to use this API:

- Error 401: Missing parameter in API call. This error will be returned for all wrongly formatted requests.

7.1.5 Implementation

The Account locking solution is being implemented in Python 2.7 programming language. The Account Management Module 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 decision 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 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.

8 Fail-over authentication mechanism

In case a mobile device is lost, stolen, or simply stops working, ReCRED users must be able to re-establish connectivity with the ReCRED infrastructure using a new device. According to the ReCRED architecture, users first authenticate themselves to the mobile device and then the device authenticates to the ReCRED server. Therefore the failover mechanism involves the re-registration of the user with a new mobile device, and the restoration of the user’s profile, i.e., credentials (provided that the user has earlier performed a backup of the credentials at the ReCRED server), user settings etc.

8.1 Scenario

There are several reasons why a user may need to exchange a mobile device for a new one:

- A mobile device is stolen.
- A mobile device is misplaced or lost.
- A mobile device is accidentally destroyed.
- A mobile device’s hardware is faulty e.g. faulty battery, faulty fingerprint reader, etc.
- A mobile device has to be replaced with a newer model in order to accommodate more demanding mobile applications.
- A mobile device must be replaced because the security of the device has been compromised e.g. a hardware hack/bug/exploit has been published that renders the particular model insecure.

In all of the above scenarios a ReCRED user must be able to re-establish in the new device the ReCRED profile stored in the old device, in order to be able to seamlessly continue to use ReCRED-based services and infrastructure.

The steps that ReCRED users need to follow involve:

- Connecting to the ReCRED server with the new device
- Restoring the user’s credentials from the ReCRED server to the new device

Then the user will be able to gain access to all Online Services that run the ReCRED software. Of course it is necessary that the user has at some earlier point saved a backup of the ReCRED credentials at the ReCRED server.

8.2 Credentials Backup & Restore

According to the ReCRED architecture, Identity Providers may issue credentials that ReCRED users store in their mobile devices. These credentials are stored in the *Trusted Execution Environment* (TEE), or encrypted by the TEE and stored in the local storage of the device to secure them against tampering or stealing.

The ReCRED suite of mobile applications includes a *Credentials Backup & Restore* application dedicated to the handling of ReCRED credentials.

In addition, ReCRED users may also use a Credentials Management web application that offers similar but limited functionality.

8.2.1 Mobile Application

Backup

The ReCRED user authenticates to the mobile device using a biometric key such as fingerprint signature in order to run the *Credentials Backup & Restore* application.

The main screen of the application displays all the credentials encrypted by the TEE. At the same time the application connects to the pre-configured ReCRED server using a secure connection and retrieves a list of all the user’s credentials found in the backup storage of the server.

Each row in the list of credentials displays:

- The attributes included in the credential
- The date the credential was issued
- The expiration date of the credential (if any)
- The name of the authority that issued the credential
- An indication whether a backup of the credential was found in the ReCRED server
- The date that the backup was stored in the ReCRED server.
- The name of the mobile device from which the backup originates.

The user may select one or more items from the list to backup to the ReCRED server. A pop-up dialog warns that old backups will be overwritten.

The user may also select to remove one or more of the credentials from the mobile device. Removal from the mobile device does not affect any copies stored in the backup storage of the ReCRED server.

Restore

The *Restore* screen of the *Credentials Backup & Restore* mobile application displays a list of all user’s credentials retrieved from the pre-configured ReCRED server. Each row of the list displays:

- The attributes included in the credential
- The date the credential was issued
- The expiration date of the credential (if any)
- The name of the authority that issued the credential
- The date that the backup was stored in the ReCRED server
- The name of the mobile device from which the backup originates

The user may select one or more credentials to restore to the mobile device.

The user may also select to remove one or more of the credentials from the ReCRED server for maintenance purposes or in case the user does not trust the ReCRED server anymore.

8.2.2 Web Application

There is also a ReCRED Web application that users can use to review their list of credentials saved in the backup storage of the ReCRED server. The information displayed for each credential is the same as in the Restore screen of the mobile application. However, there is no restore functionality. Users may only review, or remove one or more of the stored credentials from the ReCRED server.

8.3 ReCRED Elements involved

The Credentials Backup & Restore functionality involves communication between the mobile device and the ReCRED Server:

- Mobile Device
  - Runs biometric authentication to allow user access
  - Runs the ReCRED software stack
  - Retrieves the user credentials encrypted by (or stored in) the *Trusted Execution Environment*
  - Runs the *Credentials Backup & Restore* application
- ReCRED Server
  - Utilizes the ReCRED authentication module to allow remote connections from registered ReCRED users
  - Connects to the Identity Repository Service to access user credentials
  - Offers the *Credentials Management* API that provides Retrieve, Backup, Restore, and Delete functionality.
  - Runs the ReCRED *Credentials Management* web application which is a front-end to the *Credentials Management* API for users that wish to administer their credentials from a computer connected to the internet.

8.4 Security and Privacy Considerations

It is important to ensure that malicious users cannot tamper, steal or expose the credentials of ReCRED users. In order to achieve this, the Backup and Restore of Credentials was designed according to the following security and privacy considerations:

8.4.1 Mobile Device

- Access to the mobile device and in particular to the *Credentials Backup & Restore* application is allowed only after biometric authentication.
- The user's credentials are encrypted by (or stored in) the *Trusted Execution Environment* so that they may not be tampered with, even if the device is stolen or compromised.
- Communication with the ReCRED server is encrypted.
- The mobile application allows users to remove credentials from the ReCRED server's backup storage in case the user does not trust the server anymore.

8.4.2 ReCRED Server

- Access to the user's credentials is provided only via the ReCRED *Credentials Management* API.
- Access to the ReCRED *Credentials Management* API is allowed only to register ReCRED users.
- The *Credentials Management* Web application is offered only as an https service ensuring the encryption and safety of the communications channel.
- The *Credentials Management* API allows users to remove unwanted credentials in case they do not trust the server.