



SAPIENZA
UNIVERSITÀ DI ROMA

**Biometric Systems - Academic
Year 2018-2019 – Project Report**

Smart Authentication System for Cars

Prof. Maria De Marsico

Project Members:

Emre Selcuk 1791743
Basar Aksanli 1816789
Kadir Mert Ozcan 1780512
C. Haluk Sumen 1796307

Contents

| | |
|---|----|
| Abstract | 1 |
| 1.0 Introduction | 2 |
| 2.0 Project Description | 2 |
| 3.0 Project Architecture..... | 3 |
| 3.1 Workflow of Smart Authentication System for Car | 3 |
| 3.1.1 Registration to the System..... | 3 |
| 3.1.2 Achieving Authentication..... | 4 |
| 4.0 Technical Aspects..... | 5 |
| 4.1 Mobile Application..... | 5 |
| 4.1.1 Technology Used..... | 5 |
| 4.1.2 Application Structure..... | 5 |
| 4.1.3 Activities | 7 |
| 4.2 Face Detection and Identification Module | 12 |
| 4.3 Fingerprint Enrollment and Verification Module..... | 15 |
| 4.3.1 Libraries Used | 17 |
| 5.0 System Performance Evaluation..... | 21 |
| 5.1 False Acceptance Rate (FAR) and False Recognition Rate (FRR) Evaluation | 21 |
| 5.2 ROC Curve | 22 |
| 6.0 Conclusion..... | 23 |

Abstract

In this report, our idea of developing such a smart authentication system for cars is introduced and detailed in technical aspects. Software and hardware components of the system is described and communication between components is detailed since our authentication system includes multiple hardwares and multiple software implementations where each of them either communicates with other components or cloud software services. Face and fingerprint registration process to the system is described in detailed. In addition, implementations for face and fingerprint identification are described in technical aspects. Then, performance of our system is tested and evaluated in order to increase reliability of the authentication system. Results are plotted and detailed analysis is documented to present optimal value of the threshold which is used in face identification module.

1.0 Introduction

Using a key to unlock vehicles may limit people to use their vehicles in case of key loss. In addition, number of available keys for one car may be limited when we consider crowded families. Key exchange may not be possible each time or become tiring and time wasting process. As a solution, we introduce a smart authentication system for cars. Smart authentication system provides its users with entering and driving their cars without any key. Thus, biometric features of people is considered as a key in order to provide reliable system. Fingerprint feature is used as a key to unlock the car in our system. In order to start the car engine, face identification is required. Face identification also provides us with person information which includes person name, gender, age, emotions etc. This way, people are not limited with certain numbers of keys to enter the car and do not need to exchange keys. In order to create such a reliable system, we designed and implemented embedded fingerprint identification hardware, python code to identify faces and android based mobile application for registering to the system by users.

2.0 Project Description

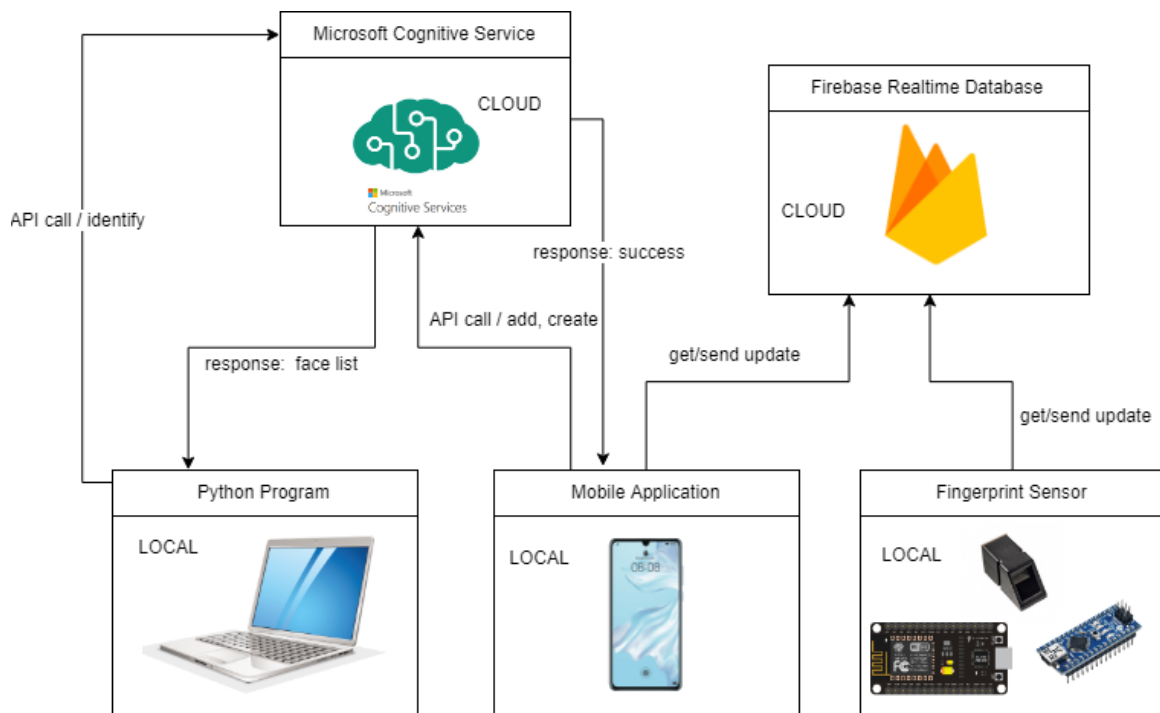
Smart authentication system for cars is composed of 5 different components where 3 of them have been implemented in local and 2 of them runs in cloud as a software service. Mobile application, fingerprint identification hardware and python code for face identification is implemented in local. As a cloud service, Face APIs of Microsoft Cognitive Services are used to create person groups, add persons into person groups, add faces of persons, detect and identify faces. In addition, Firebase Realtime Database, which is provided by Google, is used to store registered user information and registered fingerprint ids.

As a first step, users download the app and register into the system with e-mail and password credentials. Then users register their faces, which will be used during identification phase, to the mobile application. Secondly, users are required to register their fingerprints to the system to unlock their cars. Via mobile application, users send a fingerprint enrollment request to fingerprint hardware. Upon receiving this request, fingerprint hardware goes into enrollment mode and waits users to register their fingerprints. After fingerprint enrollment done by user in success, all authentication credentials are collected and system is ready to be used. In case of entering the car, users show their registered fingerprints to fingerprint hardware and unlock the car. After users sit on the driver seat, python code starts running. Python code activates the frontal camera inside of the car, which is placed in front of driver seat. When users push the start engine button, python code tries to identify driver's faces. If identification is performed successfully, car automatically starts its engine.

3.0 Project Architecture

In order to provide smart authentication system for cars, several software and hardware combinations are required. In our project there are three different high level components are implemented; mobile application for person group creation and face registration to the system, python script for face detection and identification, fingerprint enrollment and identification components have been developed.

Mobile application and fingerprint components are communicating via Google Firebase Realtime Database. Mobile application and python program performing API calls to microsoft cognitive service for face identification and face registration. In Figure 3.1, communication schema of components are illustrated.



In Figure 3.1 Communication Between Components

3.1 Workflow of Smart Authentication System for Car

3.1.1 Registration to the System

First of all, user must register to the mobile application with his/her face photo to be uploaded. Mobile application sends face information of the user to the Microsoft Cognitive Service via API calls. Then, user requests fingerprint registration via mobile application to fingerprint sensor. After fingerprint sensor stores user fingerprint once, registration is completed.

3.1.2 Achieving Authentication

When user wants to enter his/her car, must show his/her enrolled finger to fingerprint sensor. If fingerprint is verified, car unlocks its doors. After, user shows his/her face to the frontal camera which is placed in front of the driver seat and python code starts running to identify shown face. If user's face is identified, car starts its engine. Activity flow of authentication process is shown in Figure 3.1.2.1.

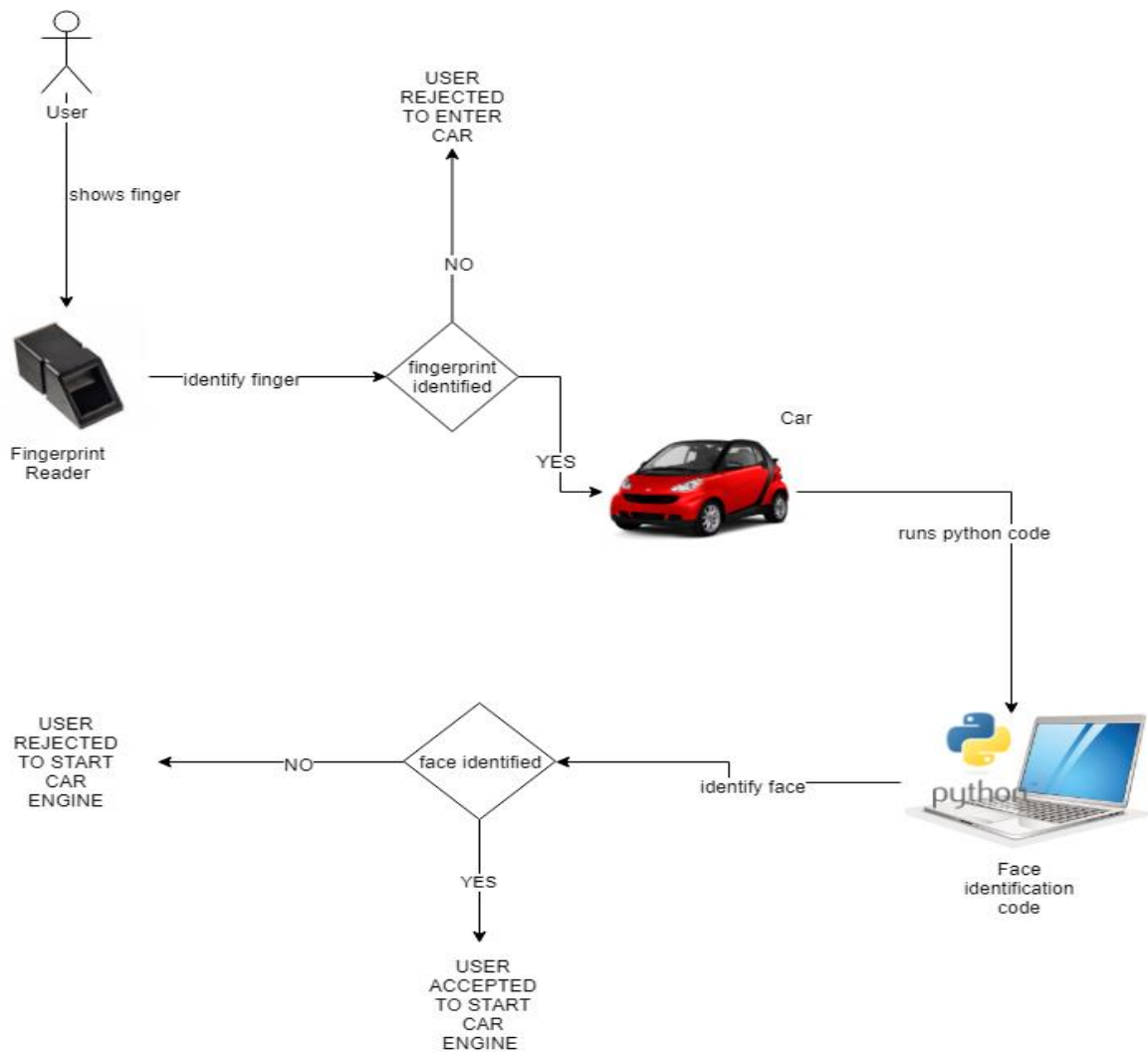


Figure 3.1.2.1 Activity Flow of Authentication Process

4.0 Technical Aspects

4.1 Mobile Application

For our system to function properly, we need to develop a platform for our users, so that they can upload their pictures and link faces with the fingerprints. Since most cars provide very limited interface in terms of usability, we decided for this system to be a mobile application.

4.1.1 Technology Used

We have developed our application using **Android Studio 3.4**.

For creating user groups and detecting faces in the images that users provide, we used **Microsoft Face Api**. This repository contains the Android client library & sample for the Microsoft Face API, an offering within [Microsoft Cognitive Services](#) (MCS), formerly known as Project Oxford.

Finally, to store additional user information, and to provide a bridge between the application and the car's fingerprint device, we used **Firebase Realtime Database**: a cloud hosted database service by Google.

Project Repository: <https://github.com/Kmozcan1/BiometricSystemsAddFace>

Face Api Repository: <https://github.com/Microsoft/Cognitive-Face-Android>

4.1.2 Application Structure

Our mobile application uses Activities to represent each page of our application by inflating the corresponding xml files. Majority of the business logic is also handled in Activities (Section 2.3), with the exception of async tasks that are used to call Face Api services and to access Firebase collections. For each of these Face Api tasks, we have created helper classes and implemented these in the activities. Furthermore, we created another Helper class to make Firebase related calls. A structure of a helper class can be seen in **Figure 4.1.2-1**. **Figure 4.1.2-2** shows the general structure of our mobile application.

```

public class DeletePersonHelper extends AsyncTask<Object, String, UUID> {
    Context context;
    ProgressDialog dialog;

    public DeletePersonHelper(Context context) {
        this.context = context;
        dialog = new ProgressDialog(context);
    }

    @Override
    protected UUID doInBackground(Object... params) {
        publishProgress( ...values: "Deleting Person...");
        String groupId = (String)params[0];
        UUID personId = (UUID)params[1];

        FaceServiceClient faceServiceClient = ConnectionHelper.getFaceServiceClient();
        try {
            faceServiceClient.deletePerson(groupId, personId);
            return personId;
        } catch (Exception e) {
            publishProgress(e.getMessage());
            return personId;
        }
    }

    @Override
    protected void onPreExecute() {
        dialog = new ProgressDialog(context);
        dialog.setTitle("Please Wait");
        dialog.show();
    }

    @Override
    protected void onProgressUpdate(String... progress) { dialog.setMessage(progress[0]); }

    @Override
    protected void onPostExecute(UUID personId) {
        dialog.dismiss();
    }
}

```

Figure 4.1.2-1– A Helper Class

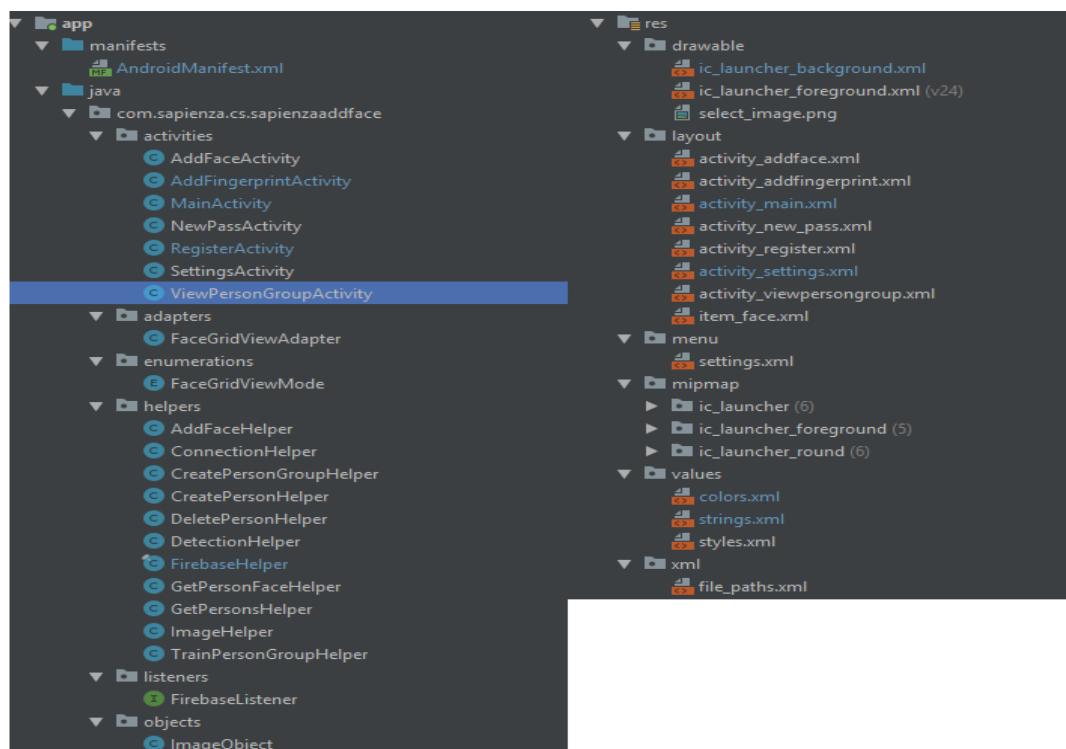


Figure 4.1.2-2 – Structure of the Application

4.1.3 Activities

4.1.3.1 Login Page

Login page (**Figure 4.1.3.1**) is the first page that is shown to users who aren't currently logged in. Here, we give the user the option to create their accounts using a non-Gmail e-mail address that they desire, or a Gmail account that they already use in their android devices. Thanks to Firebase Authentication, login step can be completed with a single click where users choose their desired Gmail account.

Each registration corresponds to a single car that a group of people (most likely family members) use. In Microsoft Cognitive Services structure, these groups are called *PersonGroups*. A *PersonGroups* can contain several *Persons*, which we'll explain in the following pages.

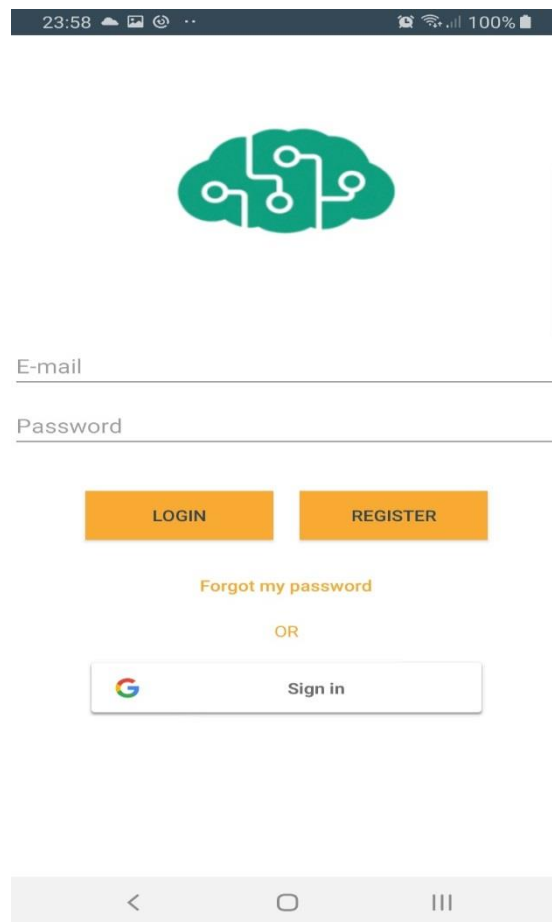


Figure 4.1.3.1 – Login Page

4.1.3.2 Register Page

If a user chooses to register with a non-Gmail e-mail address, they have to go through the Register page. (**Figure 4.1.3.2**). This is a simple step which only requires users to submit their e-mails

and passwords. The e-mail and the encrypted password are stored in the Firebase Database's Authentication section. When a user registers, the application calls FaceApi's *createPersonGroup* with our *CreatePersonGroupHelper* class. A unique id is created for each *PersonGroup* in yyMMddhhmmssMs format of the current time. This Id is also inserted into the *users* table in our Firebase Database for ease of access.

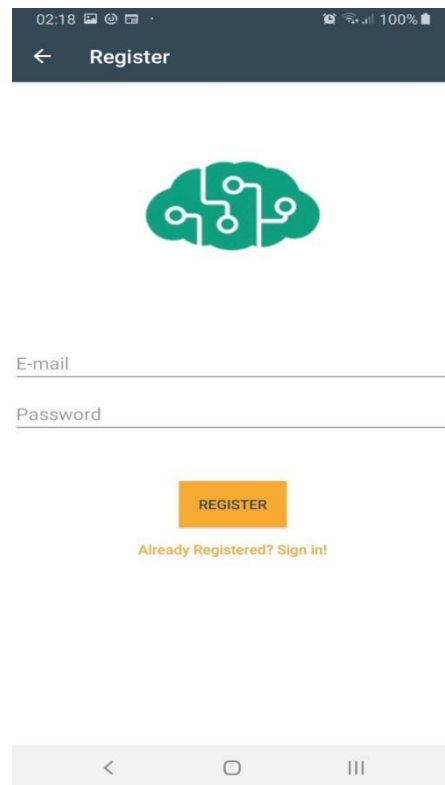


Figure 4.1.3.2 – Registration Page

4.1.3.3 View PersonGroup Page

This page (**Figure 4.1.3.3**) is the first page upon logging in to the application. Already logged in users come upon this page when they launch the application. *View Person Group Page* lists every member of a *UserGroup*. This list is fetched both our Firebase Database; however, the same list can also be fetched from MCS's Azure server by calling FaceApi's *listPersons* method via our *GetPersonsHelper* class. The users can either logout, adds another person to use a car (**Section 4.1.3.4**) or go to the "Add Fingerprint" page (**Section 4.1.3.5**) by tapping a picture. In addition, users can delete a person by long-tapping that person's picture in this page.

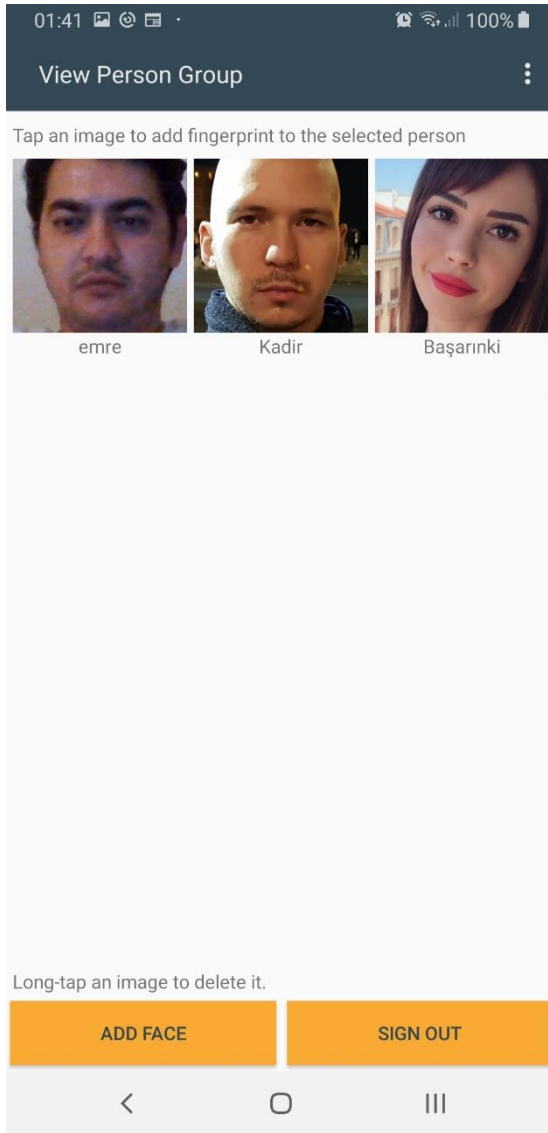


Figure 4.1.3.3 – View PersonGroup Page

4.1.3.4 Add Person Page

This is the page where the user can add a *Person* to the *PersonGroup*. In Microsoft Cognitive Services structure, a *Person* is an entity under a *PersonGroup* that can insert their own images. In our architecture, a *Person* represents an individual who can use a certain car. Users can add the images either using a camera or their phone's gallery. This image is sent to MCS's Azure server using Face Api's *detect* method. MCS then identifies the faces in the image and sends it back to the client.

If more than one face is present, both are listed in the page's gridView (**Figure 4.1.3.4-a**). The user can select one of the faces by tapping to that image. Upon this, a *Person* is created in MCS and the

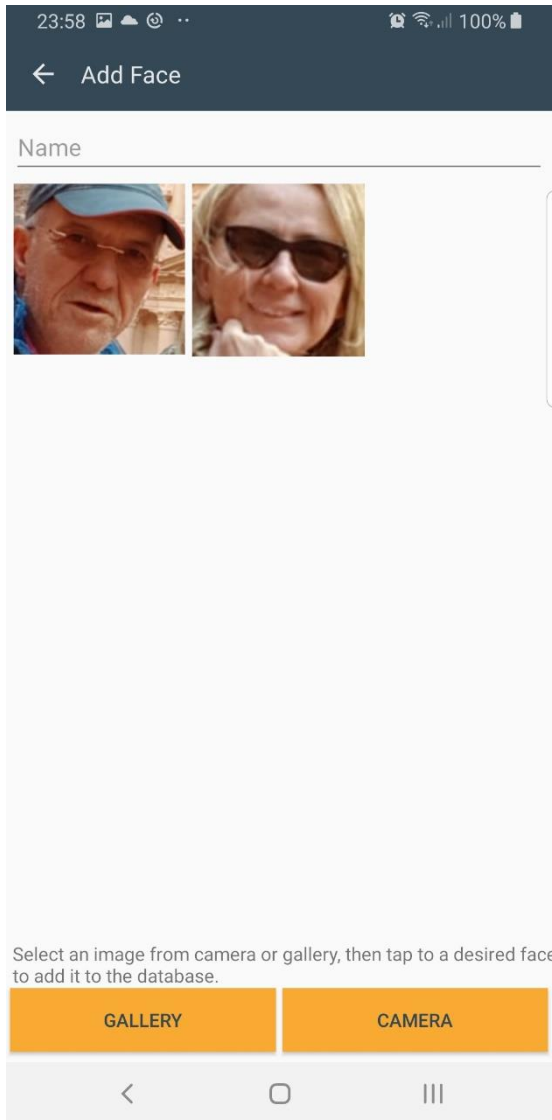


Figure 4.1.3.4-a – Multiple faces were found

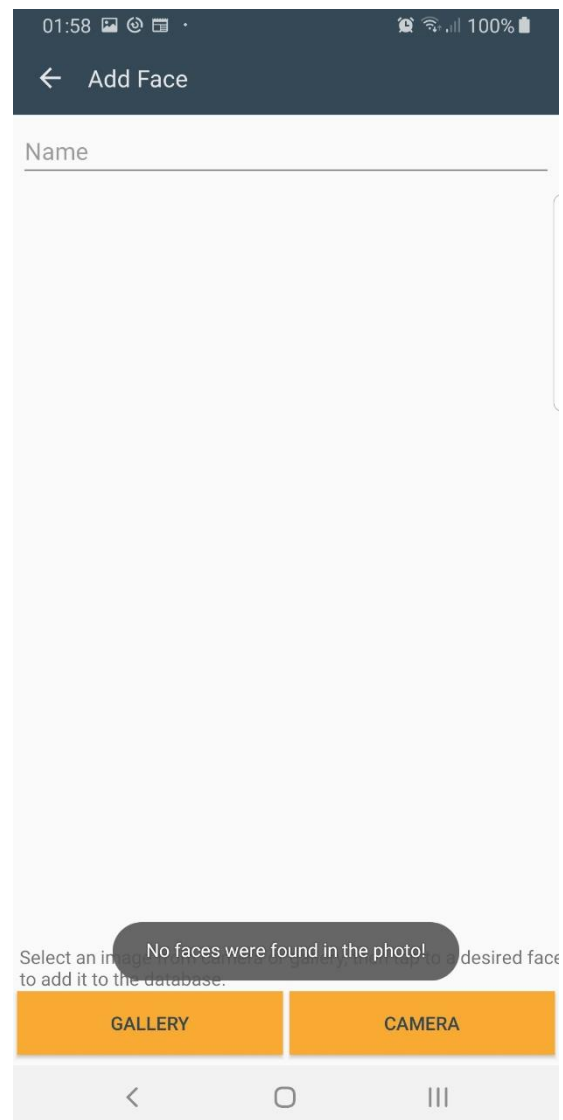


Figure 4.1.3.4-b – No faces were found

face is submitted to the to the Azure Server using Face Api's *addPersonFace* method. The image is also added to Firebase to provide the application easier access. If no faces were found, a warning is displayed and no *Person* is created (**Figure 4.1.3.4-b**).

4.1.3.5 Add Fingerprint Face

Users can access this page (**Figure 4.1.3.5-a**) by tapping an image in the View PersonGroup page. Clearly the phone itself has no ability to read fingerprints. This page's purpose is to let the server know **who** is enrolling their fingerprints using our fingerprint reader. Once the user taps the button in the middle of the screen, they have 30 seconds to put their finger on the Fingerprint reader to enroll their fingerprint. Upon pressing the button in the middle of the screen, the application submits an entry in the Firebase Database as seen in **Figure 4.1.3.5-b**. The fingerprint device in the car learns this

person's identity by reading this table. Each device has its own entry in the database that is named after their ID (remember, Firebase uses NoSql structure), and each device only reads their own table. The application knows which device to connect with thanks to this id, which is submitted to the app via the settings page (see next section for more detail).

If a fingerprint is successfully submitted within 30 second, the fingerprint reader changes the *enrollFinished* field in the database from 0 to 1. The app catches this event with a `FireBase ValueEventListener`, then stops the timer and creates a message to let the user know the enrollment was successful. The user can choose to submit their fingerprint again, in this case they can use the same button. In this case, the fingerprint reader overwrites the previous fingerprint with the new one.

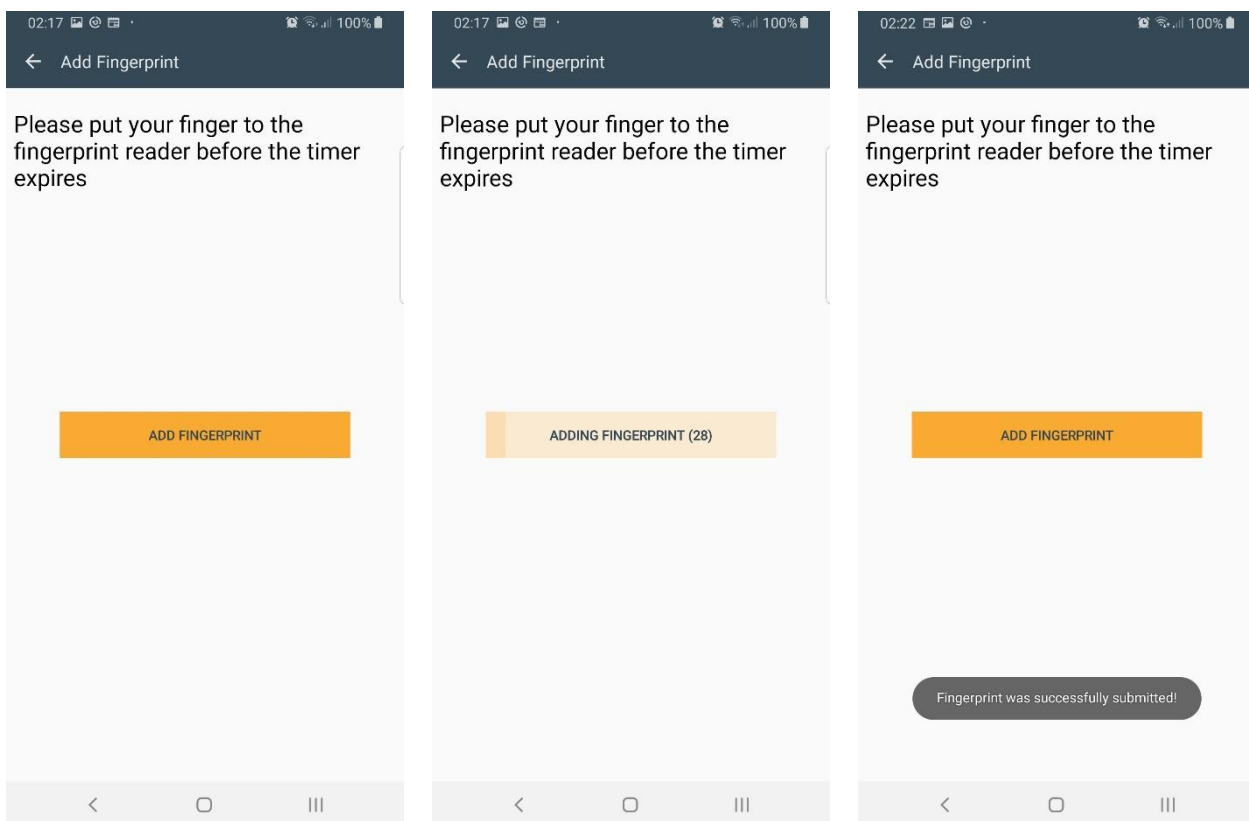


Figure 4.1.3.5-a – Add Fingerprint Page

```
activeTime: 30
activeUserID: "eb3a148"
enrollFinished: 1
needsEnroll: 0
```

Figure 4.1.3.5-b – CurrentFingerprint data in the Database

4.1.3.6 Settings

This page's sole purpose is to add a device ID to the application. This way, the application will know which car to communicate with. Note that this functionality is currently inactive, since we only have one fingerprint reader for this project. In our hypothetical product, this unique device id would be written on the manual for each customer.

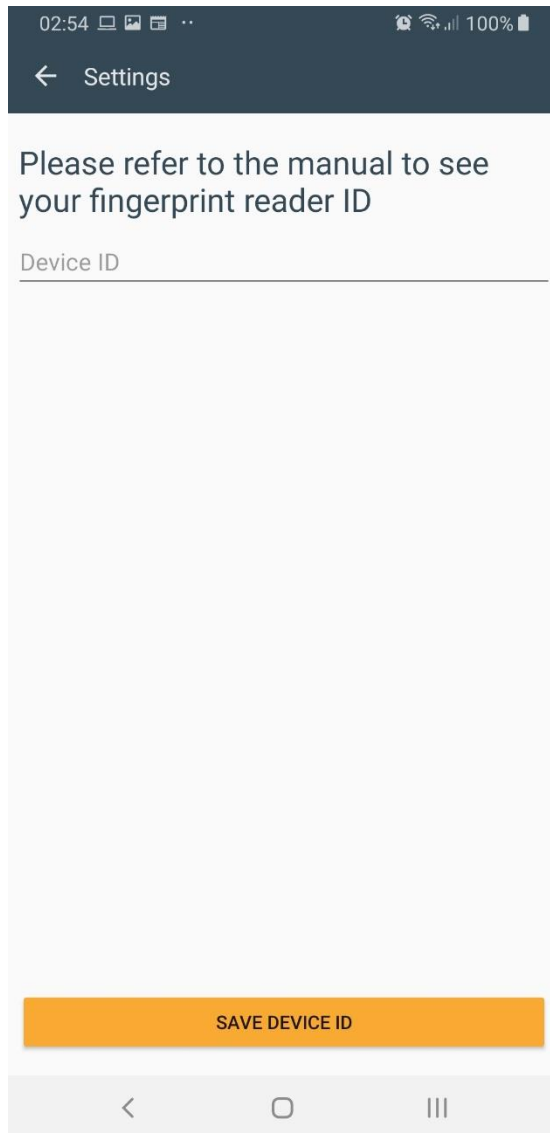


Figure 4.1.3.6 – Settings Page

4.2 Face Detection and Identification Module

Face detection and identification module is implemented to verify user if he/she is authenticated to start the car engine. This part is implemented in python programming language and works like a embedded frontal camera program and it is placed in front of the driver seat. After fingerprint verification is achieved and door of the car is unlocked, and driver sits on the seat, program starts the video capturing. When user wants to start the car engine he/she is requested to push start

button. When button clicked face identification code starts running and tries to identify user face. After, face recognition is achieved, car engine starts running.

Before starting identification, program also check whether there is single frontal face in the captured video or not. If there is no face found or multiple faces are detected, program wait until it finds only single face in the captured video.

Face API as Microsoft cognitive service is used to identify face. In order to detect frontal face, haarcascade_frontalface_cascade is used as OpenCV library. In order to identify faces by using Face API, its structure should be known well. Microsoft cognitive service keeps face information in a hierarchical model. There are person groups which contains different person and each person in this group has his/her face. Registration of faces are performed by using this hierarchy.

```
2 import cognitive_face as cf
3 from PIL import Image, ImageDraw
4 import cv2
5
6
7 KEY = '2582b9f1e2c24202b8de06cc4126a485'
8 cf.Key.set(KEY)
9
10 detectedFaceId = []
11 BASE_URL = 'https://westeurope.api.cognitive.microsoft.com/face/v1.0'
12 cf.BaseUrl.set(BASE_URL)
```

The main part of the program is VideoCapture() function which contains face detection and identification code parts. It opens video capturing and runs detecting faces until driver pushes the start engine button which is defined as button“q” on the keyboard of the computer.

```
def videoCaptureFunc():
    faceCascade = cv2.CascadeClassifier(r"C:\Users\EmreSelcuk\PycharmProjects\biometricProjectDefault\venv\Lib\site-p
    video_capture = cv2.VideoCapture(0)
    faceCnt = 0
    while True:
        # Capture frame-by-frame
        ret, frame = video_capture.read()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        facesDetect = faceCascade.detectMultiScale(
            gray,
            scaleFactor=1.1,
            minNeighbors=5,
            minSize=(30, 30)
        )
        print("LENGTH : "+str(len(facesDetect)))
        print('\n')
        if len(facesDetect) != 0:
            if faceCnt < facesDetect.shape[0]:
                faceCnt = faceCnt + 1
                print("New Face Detected")
                print('\n')
                print("Number of face on camera : " + str(facesDetect.shape[0]))
                print('\n')
```

```

if faceCnt > facesDetect.shape[0]:
    print("Face lost")
    faceCnt = faceCnt - 1
    print('\n')
    print("Number of face on camera : " + str(facesDetect.shape[0]))
    print('\n')
else:
    print("No face detected !")
    print('\n')
    faceCnt = 0

# Draw a rectangle around the faces
for (x, y, w, h) in facesDetect:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
# Display the resulting frame
cv2.imshow('Video', frame)
# Write frame in file
cv2.imwrite('imageCaptured1.jpg', frame)

```

After button is pushed by user, identification part of the program starts running.

```

84 for ids in detectedFaceId:
85     identifiedFaces = cf.face.identify(detectedFaceId, person_group_id='190428054518418', large_person_group_id=None,
86                                     max_candidates_return=1,
87                                     threshold=0.5)
88 # When everything is done, release the capture

```

From Face API, we call “identify()” function which tries to identify detected faces on the video frame. It takes detected face list, person group id information which cognitive service will search by, large person group id if exists, max candidate return which is set to 1 because we need only one person on driver seat, and threshold value as parameters. Then, it returns identified candidates list where each candidate in the list has several information such as face id, confidence value, and person id where each candidate has confidence values greater than defined threshold. After code gets the id of the candidate person, it needs to search this person in the person group which he/she is belong to.

```

for facesKnown in identifiedFaces:
    print(facesKnown['faceId'] + '\n')
    print(' : is the face ID \n')
    for candidates in facesKnown['candidates']:
        print(candidates['confidence'])
        print(' : is the confidence \n')
        print(candidates['personId'])
        print(' : is the person ID \n')
        person_id_found= candidates['personId']

personFound = cf.person.get('190428054518418', person_id_found)

print("You are : " + personFound['name']+'\n')
print("Welcome to the car ! ")
video_capture.release()
cv2.destroyAllWindows()

```

After candidate person is found in the group id, it means that identification is achieved and car is ready to start its engine. Functions supported by face API : https://github.com/microsoft/Cognitive-Face-Python/tree/master/cognitive_face

4.3 Fingerprint Enrollment and Verification Module

Fingerprint enrollment and verification module consists of one database communication and 3 different hardware components; one esp6288 nodemcu, one arduino nano and one fingerprint sensor. All components are connected in a logical way for infotmarion share. Communication flow of the module is illustrated in Figure 4.3.1.

Esp6288 nodemcu and Firebase database communicates over the internet by using WiFi module on esp6288 nodemcu. Esp6288 is connected to the internet and takes periodic updates from database. Exchanged information consists of “**activeTime**” of enrollment process, “**activeUserID**” , “**needsEnroll**” value to understand if active user wants to enroll his/her fingerprint into the system or not, and “**enrollFinished**” value which is updated by esp6288 when successful enrollment process is performed by fingerprint reader.

```

..... activeTime: 30
..... activeUserID: "eb3a1483-3b3c-4c9d-bd31-ff1c1f29a6
..... enrollFinished: 0
..... needsEnroll: 0

```


“needsEnroll” value is set to 1 by mobile application if user wants her/his fingerprint is to be enrolled to the system. Esp6288 notifies arduino nano (via serial port communication) that a user wants to enroll her/his fingerprint into the system. Then, arduino nano changes fingerprint sensor mode to enrollment mode(note that fingerprint sensor always runs in verification mode unless arduino nano changes its mode to enrollment mode). After user enrolls her/his finger successfully, arduino nano notifies esp6288 that enrollment is finished and returns back to verification mode. Esp6288 updates database by setting ”needsEnroll” value to 0 and “enrollFinished” value to 1. Finally, Esp6288 updates user information on database by adding his/her fingerprint id as “enrolledFinger”.

```

.....enrolledFinger: 1
.....groupId: "190428054518416
.....image: "iVBORw0KGgoAAAANSUhEUgAAAQkAAAEJCAIAAAAIckUzAA/
.....personName: "Başarınki

```

Esp6288 nodemcu and arduino performs serial communication over serial port. They exchange information about if enrollment needed or if enrollment is performed successfully. As a default, Esp6288 makes arduino nano run in verification mode.

Fingerprint sensor and arduino nano performs serial communication. Fingerprint reader returns the id value of verified finger if it runs in verification mode or returns the id value of enrolled finger if it runs in enrollment mode.

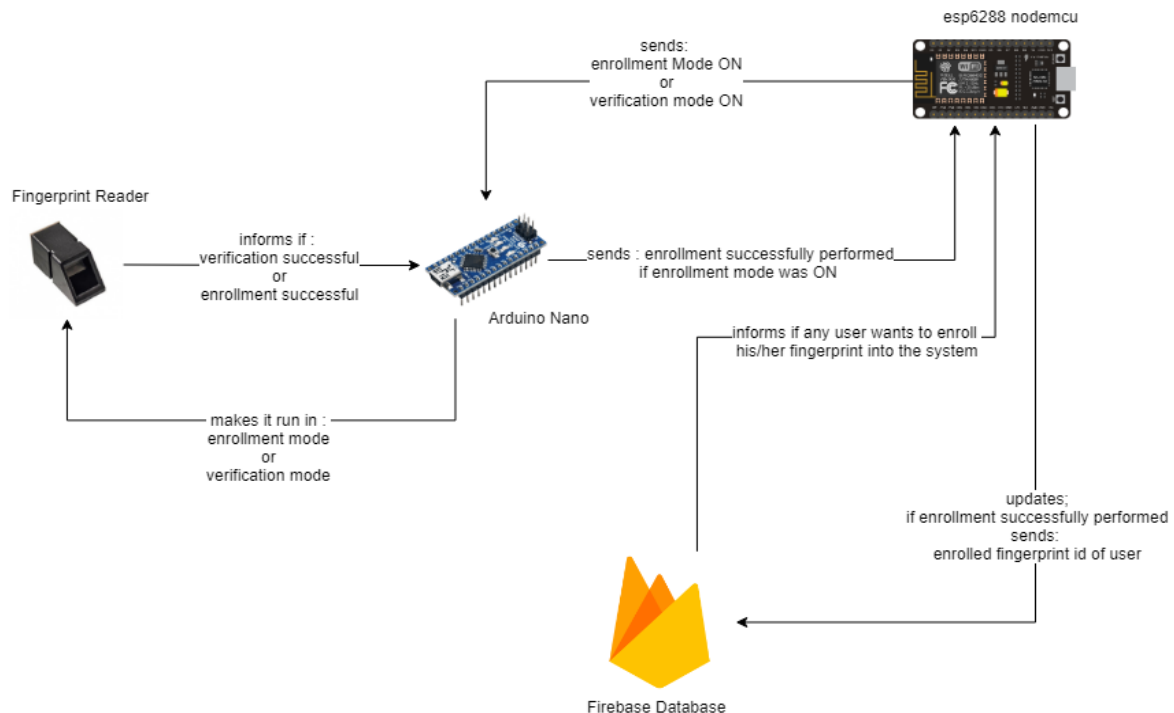


Figure 4.3.1 : Communication flow of Fingerprint enrollment and verification

4.3.1 Libraries Used

In arduino nano, Adafruit Fingerprint library is used. It provides functions while performing enrollment and verification of fingerprints. In order to reach functions, “Adafruit_Fingerprint.h” header file must be included in the code.

finished_biometric_fingerprint_nano

```
1 #include <Adafruit_Fingerprint.h>
2 SoftwareSerial mySerial(2, 3); //FINGERPRINT SERIAL COMMUNICATION DEFINED WITH PIN NUMBERS
3 Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);
4 ...
```

In verification part, “getImage()”, “image2Tz()”, “fingerFastSearch()” functions are used from the library respectively. “getImage()” function takes the image of shown finger from the sensor. Then we perform a check if finger could be read properly. Since library provides us with pre-defined Macros, we can check the return value of “p” and decide to continue to next step of finger identification.

```
91 uint8_t p = finger.getImage();
92
93 switch (p) {
94     case FINGERPRINT_OK:
95         //Serial.println("Image taken");
96         break;
97     case FINGERPRINT_NOFINGER:
98         //Serial.println("No finger detected");
99         return p;
100    case FINGERPRINT_PACKETRECEIVEERR:
101        //Serial.println("Communication error");
102        return p;
103    case FINGERPRINT_IMAGEFAIL:
104        // Serial.println("Imaging error");
105        return p;
106    default:
107        // Serial.println("Unknown error");
108        return p;
109 }
110
111 // OK success!
...
```

```
28 #define FINGERPRINT_OK 0x00
29 #define FINGERPRINT_PACKETRECEIVEERR 0x01
30 #define FINGERPRINT_NOFINGER 0x02
31 #define FINGERPRINT_IMAGEFAIL 0x03
32 #define FINGERPRINT_IMAGEMESS 0x06
33 #define FINGERPRINT_FEATUREFAIL 0x07
34 #define FINGERPRINT_NOMATCH 0x08
35 #define FINGERPRINT_NOTFOUND 0x09
36 #define FINGERPRINT_ENROLLMISMATCH 0x0A
37 #define FINGERPRINT_BADLOCATION 0x0B
38 #define FINGERPRINT_DBCLAREFAIL 0x0C
39 #define FINGERPRINT_UPLOADFEATUREFAIL 0x0D
40 #define FINGERPRINT_PACKETRESPONSEFAIL 0x0E
41 #define FINGERPRINT_UPLOADFAIL 0x0F
42 #define FINGERPRINT_DELETEFAIL 0x10
43 #define FINGERPRINT_DBCLEARFAIL 0x11
44 #define FINGERPRINT_PASSFAIL 0x13
45 #define FINGERPRINT_INVALIDIMAGE 0x15
46 #define FINGERPRINT_FLASHERR 0x18
47 #define FINGERPRINT_INVALIDREG 0x1A
48 #define FINGERPRINT_ADDRCODE 0x20
49 #define FINGERPRINT_PASSVERIFY 0x21
```

If “p==FINGERPRINT_OK”, we call “image2Tz()” function to convert image to the feature template. Again, we check the returned value of the function to decide if conversion is done successfully.

```

113 p = finger.image2Tz();
114
115 switch (p) {
116     case FINGERPRINT_OK:
117         // Serial.println("Image converted");
118         break;
119     case FINGERPRINT_IMAGEMESS:
120         //Serial.println("Image too messy");
121         return p;
122     case FINGERPRINT_PACKETRECEIVEERR:
123         // Serial.println("Communication error");
124         return p;
125     case FINGERPRINT_FEATUREFAIL:
126         // Serial.println("Could not find fingerprint features");
127         return p;
128     case FINGERPRINT_INVALIDIMAGE:
129         // Serial.println("Could not find fingerprint features");
130         return p;
131     default:
132         // Serial.println("Unknown error");
133         return p;
134 }

```

If “p==FINGERPRINT_OK”, it means that conversion is finished in success and we call another librar function which is “fingerFastSearch()”. By using this fnction, we ask the sensor to search the current fingerprint features to match saved templates.

```

137 p = finger.fingerFastSearch();
138 if (p == FINGERPRINT_OK) {
139     //Serial.println("Found a print match!");
140     digitalWrite(BUZZER, 1);
141     delay(50);
142     digitalWrite(BUZZER, 0);
143 }
144 else if (p == FINGERPRINT_PACKETRECEIVEERR) {
145     //Serial.println("Communication error");
146     return p;
147 }
148 else if (p == FINGERPRINT_NOTFOUND) {
149     Serial.println(-9);
150     digitalWrite(BUZZER, 1);
151     delay(50);
152     digitalWrite(BUZZER, 0);
153     delay(50);
154     digitalWrite(BUZZER, 1);
155     delay(50);
156     digitalWrite(BUZZER, 0);
157
158     return p;
159 }
160 else {
161     //Serial.println("Unknown error");
162     return p;
163 }

```

If “p==FINGERPRINT_OK”, it means that shown fingerprint is matched with a stored template and verification step is finished successfully.

In verification part, only these three functions are used from the library. Rest of the codes are implemented by us. Our implementation includes controlling information exchange between nodes,

checking if verification steps are done in success and overcoming possible failures during fingerprint reading.

In finger enrollment part, we developed “getFingerprintEnroll” function by using “getImage()”, “image2Tz()”, “createModel()” and “storeModel()” functions which are provided by the library. In enrollment, first two steps are the same as verification part. User is asked to show his/her finger to the sensor to get image of the finger and convert it to the template. In order to enroll fingerprint into the system, fingerprint template must be stored in the sensor as fingerprint model. Thus, “createModel()” function is called to create the model. Before calling this function, user is asked to show his/her finger again to the sensor in order to have two different templates of the same finger. “createModel()” functions tries a match between these two templates. If mismatch is detected, user is asked to return back to the first step of enrollment. If fingerprints are matched, “storeModel(id)” function is called to store the fingerprint with its “id” and enrollment is performed in success.

```

330 p = finger.storeModel(id);
331 if (p == FINGERPRINT_OK) {
332     Serial.println("Stored!");
333     fingerSuccess=1;
334     digitalWrite(BUZZER, 1);
335     delay(50);
336     digitalWrite(BUZZER, 0);
337     digitalWrite(BUZZER, 1);}
338     delay(50);
339     digitalWrite(BUZZER, 0);
340     digitalWrite(BUZZER, 1);
341     delay(50);
342     digitalWrite(BUZZER, 0);
343     digitalWrite(BUZZER, 1);
344     delay(50);
345     digitalWrite(BUZZER, 0);
346     digitalWrite(BUZZER, 1);
347     delay(400);
348     digitalWrite(BUZZER, 0);
349     delay(4000);
350 } else if (p == FINGERPRINT_PACKETRECEIVEERR) {
351     Serial.println("Communication error");
352     return p;
353 } else if (p == FINGERPRINT_BADLOCATION) {
354     Serial.println("Could not store in that location");
355     return p;
356 } else if (p == FINGERPRINT_FLASHERR) {
357     Serial.println("Error writing to flash");
358 }

297 p = finger.createModel();
298 if (p == FINGERPRINT_OK) {
299     Serial.println("Prints matched!");
300     digitalWrite(BUZZER, 1);
301     delay(200);
302     digitalWrite(BUZZER, 0);
303     delay(1000);
304 } else if (p == FINGERPRINT_PACKETRECEIVEERR) {
305     Serial.println("Communication error");
306     return p;
307 } else if (p == FINGERPRINT_ENROLLMISMATCH) {
308     Serial.println("Fingerprints did not match");
309     digitalWrite(BUZZER, 1);
310     delay(50);
311     digitalWrite(BUZZER, 0);
312     delay(50);
313     digitalWrite(BUZZER, 1);
314     delay(50);
315     digitalWrite(BUZZER, 0);
316     digitalWrite(BUZZER, 1);
317     delay(50);
318     digitalWrite(BUZZER, 0);
319     delay(50);
320     digitalWrite(BUZZER, 1);
321     delay(50);
322     digitalWrite(BUZZER, 0);
323     return p;
324 }

```

In esp8266 nodemcu, Firebase-ESP8266 library is used to create database communication and information exchange. ESP8266WiFi library is used to establish wifi communication.

finished_biometric_fingerprint_esp

```

1 #include "FirebaseESP8266.h"
2 #include <ESP8266WiFi.h>
3 #define FIREBASE_HOST "biometricproject-23d0f.firebaseio.com"
4 #define FIREBASE_AUTH "OKR28iD1743OXcKhE86LMb5JBBX41VX3AUAg6Jkj"
5 #define WIFI_SSID "Vodafone-34946245"
6 #define WIFI_PASSWORD "hakunamatata"
7 FirebaseData FB;

```

During setup part of the code, we implemented wifi communication code by using “WiFi.begin()” with giving wifi name and its password as parameter. Node waits to start its functionalities until communication is established. Then, firebase communication is established by using “Firebase.begin()” with giving firebase host name and its authentication key.

```

34 void setup() {
35
36   Serial.begin(115200);
37   delay(300);
38   WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
39   delay(300);
40   Serial.print(F("Connecting to Wi-Fi"));
41   while (WiFi.status() != WL_CONNECTED) {
42     Serial.print(F("."));
43     delay(300);
44   }
45   swSer.begin(115200);
46   Serial.println();
47   Serial.print(F("Connected with IP: "));
48   Serial.println(WiFi.localIP());
49   Serial.println();
50   pinMode(outPin, OUTPUT);
51   Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
52   Firebase.reconnectWiFi(true);
53

```

In order to read data from firebase database, we used “Firebase.getInt()” function by giving firebase data instance, path information where we will find the data. In this case, we are reading integer type data. For other primitive types there are different functions such as “Firebase.getString()”.

```

83 // 3. get if enrol needed
84 Firebase.getInt(FB,pathCurrentFingerRequest+"/needsEnroll");
85 needsEnroll = FB.intData();

```

In order to write data to firebase database, we used “Firebase.setInt()” function by giving firebase data instance, path information where we want to add the data, and the variable of data which we want to add or update in to the database as a parameter.

```

121 Serial.println("Enrollment Successfully Finished !");
122 Firebase.setInt(FB,pathCurrentFingerRequest+"/enrollFinished",enrollSuccesFinished);

```

Rest of the codes are implemented by us which contains information control and exchange between both firebase database and arduino nano.

Firebase library link for esp8266 : <https://github.com/mobizt/Firebase-ESP8266>

5.0 System Performance Evaluation

In order to evaluate the performance of our product, we applied tests over datasets which contains different people with positive and negative faces.

5.1 False Acceptance Rate (FAR) and False Recognition Rate (FRR) Evaluation

Evaluation 1 : First of all, in one person group, we trained 20 person face (1 face for each person). We used a test data that contains 42 face in total (20 positive and 22 negative). Then we collected each confidence value of faces in test data. For each threshold values starting from 0.01 to 1, we calculated TP, TN, FP and FN results according to corresponding threshold. For each threshold values we also used FAR and FRR equations to plot Figure 5.1.1-a.

$$\text{Equation FAR} = \text{FPR} = \text{FP} / (\text{FP} + \text{TN})$$

$$\text{Equation FRR} = \text{FNR} = \text{FN} / (\text{TP} + \text{FN})$$

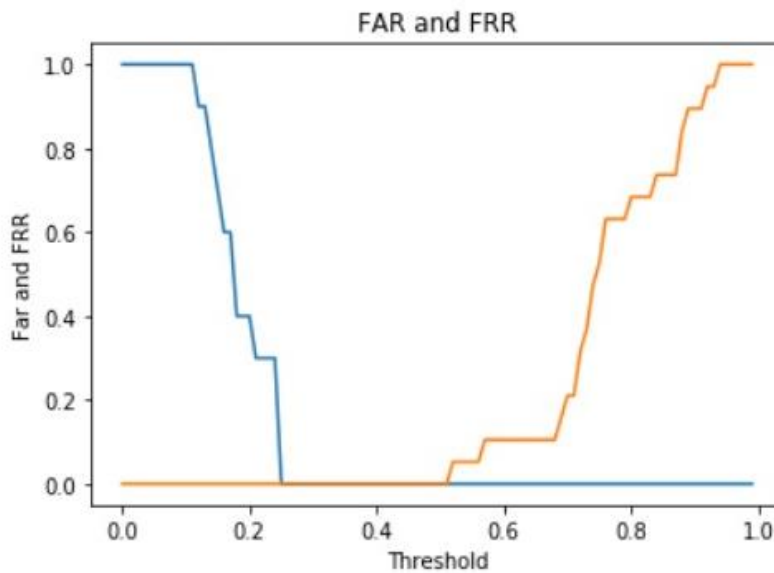


Figure 5.1.1-a FAR and FRR – Threshold

From the figure above, we found out that threshold value from 0.25 to 0.5, we get the same FRR and FAR results. We can also figure out that resulting EER value is 0 in this case. Since we were limited to have quite less number of data, we can't define the accurate threshold. Thus, we decided to use much larger dataset for more FAR and FRR results to evaluate optimal threshold.

Evaluation 2 : For FAR and FRR evaluation, in one person group, we trained 50 person. Then we choosed 300 negative person. We tested our identification function with 200 positive faces and 800 neegative faces by getting their corresponding confidence values. For each threshold values starting

from 0.01 to 1, we calculated TP, TN, FP and FN results according to corresponding threshold. For each threshold values we also used FAR and FRR equations to plot Figure 5.1.1-b.

$$\text{Equation FAR} = \text{FPR} = \text{FP} / (\text{FP} + \text{TN})$$

$$\text{Equation FRR} = \text{FNR} = \text{FN} / (\text{TP} + \text{FN})$$

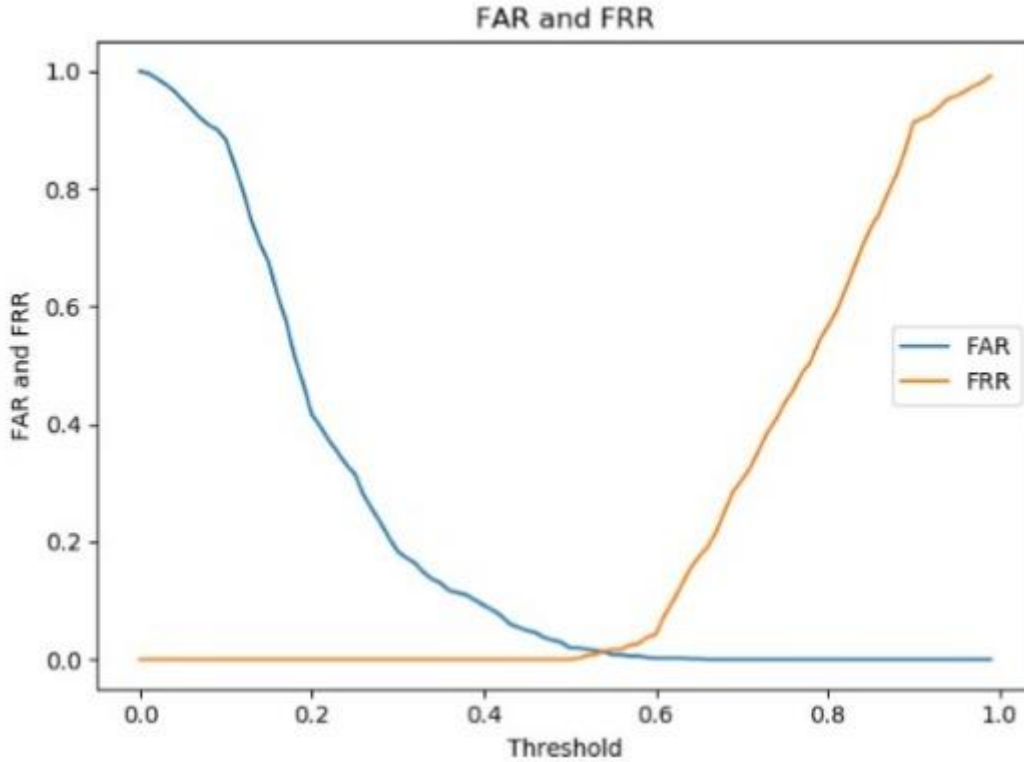


Figure 5.1.1-b FAR and FRR – Threshold

Since face identification module shouldn't allow any person who is out of defined person group, our FAR should be as less as possible. Otherwise unauthorized person could start the car engine. FRR should also be minimum as much as possible. As optimal, resulting EER is around 0.02 which is indicated where the proportion of false acceptances is equal to the proportion of false rejections. Thus corresponding threshold is 0.57 with respect to EER.

As a result, when we compare our first and second evaluations, we found out that since evaluation 2 has more number of test data, we could select our optimal threshold value.

5.2 ROC Curve

ROC curve is plotted to show that GAR (Genuine Acceptance Rate) increasing while threshold decreasing from 1.0 to 0.57 where FAR stays at 0. When the threshold value start decreasing from 0.57, FAR started to increase from 0. For threshold values less than 0.57, we inspect that while GAR value slowly increasing, FAR value also starts to increase from 0. Since GAR is increasing

between approximately 0.96 and 1 while threshold start decreasing from a certain point, this threshold value deduction causes FAR to start increasing. Hence, optimal threshold should be set to 0.57 as it is seen from Figure 5.1.1-b and Figure 5.1.2.

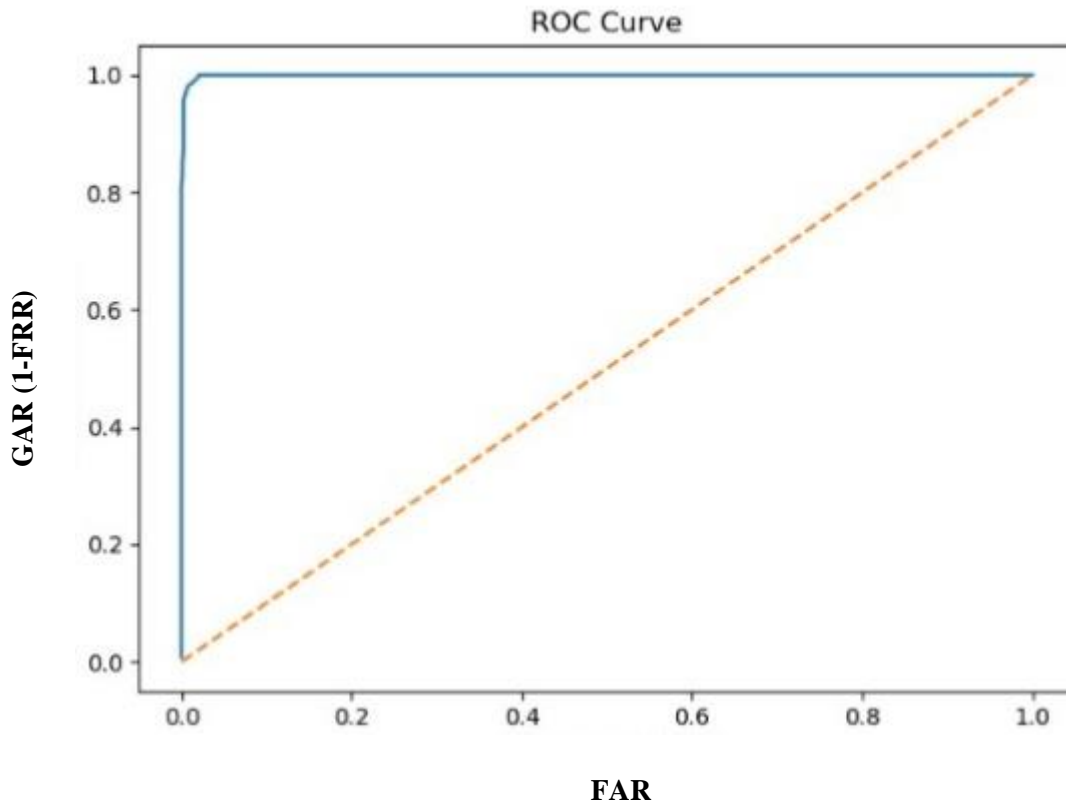


Figure 5.1.2 ROC Curve

6.0 Conclusion

As a conclusion, we have started implementing the idea of smart authentication system for cars in case of key loss, in order to prevent consuming time during key exchange and to provide flexible number of abstract keys for car users. In order to provide such reliable system we used biometric features of person such as fingerprint and face. Since person is authenticated only when fingerprint and face identification is performed respectively in succes, this two level biometric feature identification increased relaibility of the system. In order to measure performance of our system, we performed evaluations on the system by using different datasets with different faces. Upon getting results of the performance evaluation, we could define our optimal threshold value which is used during face identification function. By performing evaluation over the face identification part, we could ensure that non-authorized users cannot be able to start the car engine. As a result, smart authentication system for cars project can be used as a reliable product.