Mestrado em Engenharia Informática
Departamento de Electrónica, Telecomunicações e Informática

Biometrics Report

10/01/2024

Nuno Cunha - 98124
Alexandre Gago - 98123
Bernardo Kaluza - 97521

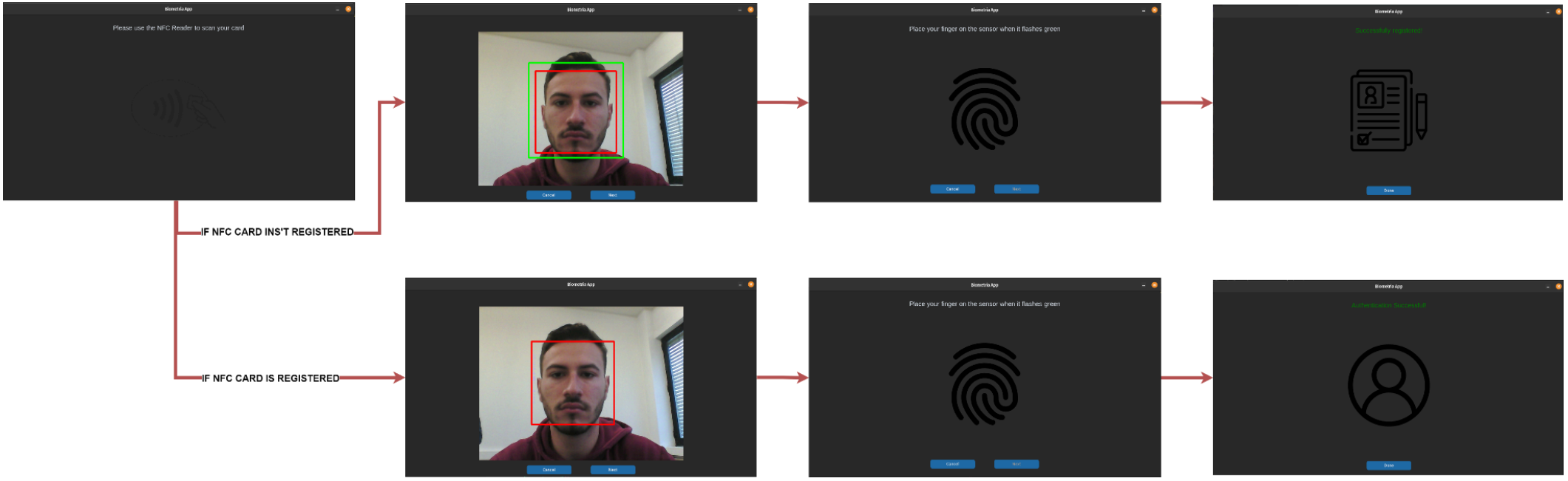
Index

Introduction

User authentication can be done by leveraging the unique physiological and behavioral characteristics of a person, his biometrics, to ensure a secure and personalized access control. In an era where digital interactions dominate various aspects of our lives, the importance of robust and user-friendly authentication systems is gradually increasing.

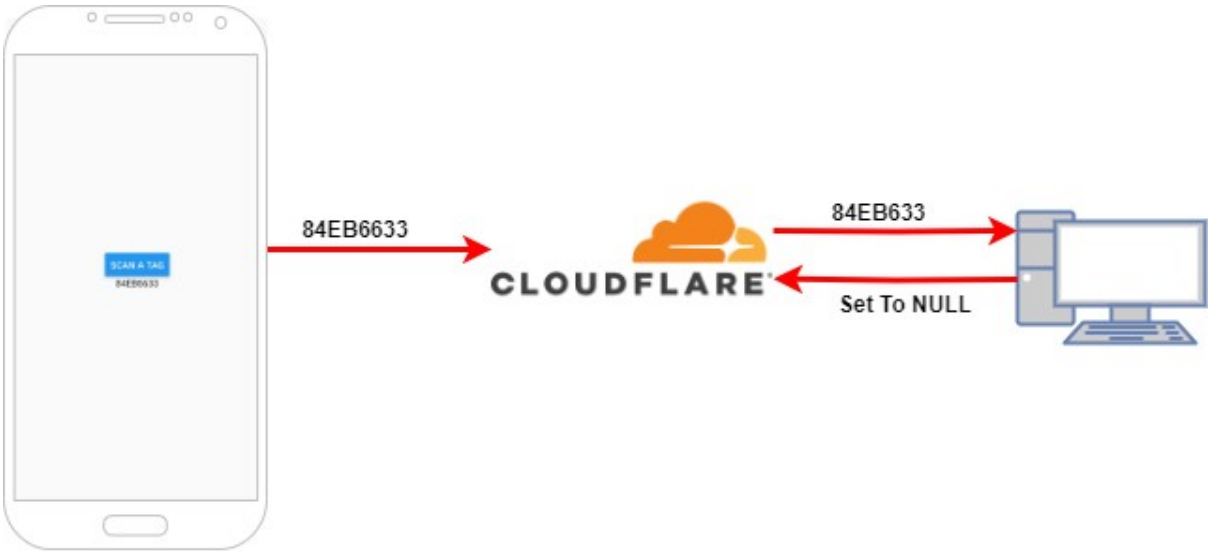
This report will explore our biometric authentication system, comprising a user interface (UI), NFC (Near Field Communication), facial recognition, and fingerprint scanning. This integrated system enhances security measures and offers a seamless and efficient user authentication experience. We will explore each component's functionalities and analyze their collaboration's synergistic effect in improving the authentication process. The system works as follows: the user scans their identification card in the NFC reader. The app will authenticate users if they already exist in the database. Otherwise, it will register the new user.
When authenticating a user, the app will redirect them to the face recognition screen, where they must pass a liveness test and be authenticated using facial recognition. After this step, the user must pass the fingerprint recognition screen before finally being authenticated.

UI Scheme


Please use the NFC Reader to scan your card
IF NFC CARD INS'T REGISTERED
IF NFC CARD IS REGISTERED
Place your finger on the sensor when it flashes green
Place your finger on the sensor when it flashes green

NFC

The NFC is the first page in our app. The user is asked to pass his identification card in the sensor in this step. The sensor is an Android smartphone running an app developed by us. The app is very straightforward. It reads the tag ID and sends it to the Cloudflare server to be read by the app on the PC.



The app was made using React Native in the Expo Go environment (see https://reactnative.dev/docs/environment-setup). Unfortunately, the library used to read the information in the NFC didn't allow for debug tests using Expo, so to test, we were obligated to compile the app every time.

To read the NFC card, we used the "react-native-nfc-manager" library (see https://github.com/revtel/react-native-nfc-manager), and to make calls to the Couldflare runner, we used fetch.

In Cloudflare, we set up a runner that only receives a post from the app and makes the data from the post available in the method to be read by the PC.

Here is the code of the runner:

```
const TOKEN = ""
var currentId = null;

var src_default = {
  async fetch(request, env) {
    const { DATABASE } = env;

    if (request.method === "GET"){
      const stmt = DATABASE.prepare("SELECT currentId FROM tagId LIMIT 1");
      const { results } = await stmt.all();

      const json = JSON.stringify(results[0], null, 2);

      return new Response(json, {
        headers: {
          "content-type": "application/json;charset=UTF-8",
          "Access-Control-Allow-Origin": "*",
          "Access-Control-Allow-Methods": "GET,HEAD,POST,OPTIONS",
          "Access-Control-Max-Age": "86400",
        },
      });

    }else if (request.method === "POST"){
      const auth = await request.json()

      if (auth.token === TOKEN){
        await DATABASE.prepare("UPDATE tagId SET currentId =
?1").bind(auth.id).run();
      }

      return new Response(null, {
        headers: {
          "content-type": "application/json;charset=UTF-8",
          "Access-Control-Allow-Origin": "*",
          "Access-Control-Allow-Methods": "GET,HEAD,POST,OPTIONS",
          "Access-Control-Max-Age": "86400",
        },
      });
    }
  }
};

export {
  src_default as default
};
```

Face Recognition

In this section we will talk about our face recognition algorithm.

Image Processing

We tried various techniques of Image Processing that could benefit the Face Recognition models, such as Gamma Correction, applying a soft blur to reduce noise, and automatically adjusting image contrast and brightness. After testing, none of these steps are present in the final version, as we found that the images captured with the camera's automatic parameters were enough and sometimes even performed better in the models than our processed images.

Algorithm

The face recognition works as follows:

1. Detection of the eye's position
2. Liveness Detection (Blinks)
3. Authenticate or Register the face

Liveness

To perform liveness detection, we count the number of blinks the user did in the authentication process. These blinks are gathered without the user's knowledge to avoid attacks on the system's liveness detection.

During 1 second, the system will take several images of the user's face and calculate the average size of the user's eyes. During the 10 seconds after that, the system will take several images of the user's face and, for each of them, check if the eyes are closed, and if they are, we will increment the blink counter. The figure below shows the system's perception of where the eyes are in their natural shape.

To pass the liveness detection, the user must have blinked at least twice during the 10 seconds. On average, a person blinks around 1.6 times per 10 seconds. We consider two or more blinks per 10 seconds sufficient for the liveness detection task; however, in some cases where the user may not blink two times, the system will give a false negative.

Face Authentication

For face authentication, we use the DeepFace Python library to check if the face of the user matches the one in the database.

```
result = DeepFace.verify(img1_path = image, img2_path =
f"./db/{user}/user.png", model_name = 'ArcFace')
```

Although the DeepFace authors suggest Facenet512 as the best model of face authentication, based on our testing, we consider ArcFace to be the overperforming model as it was able to recognize faces in different lighting conditions and different facial expressions better than Facenet512.

Face Register

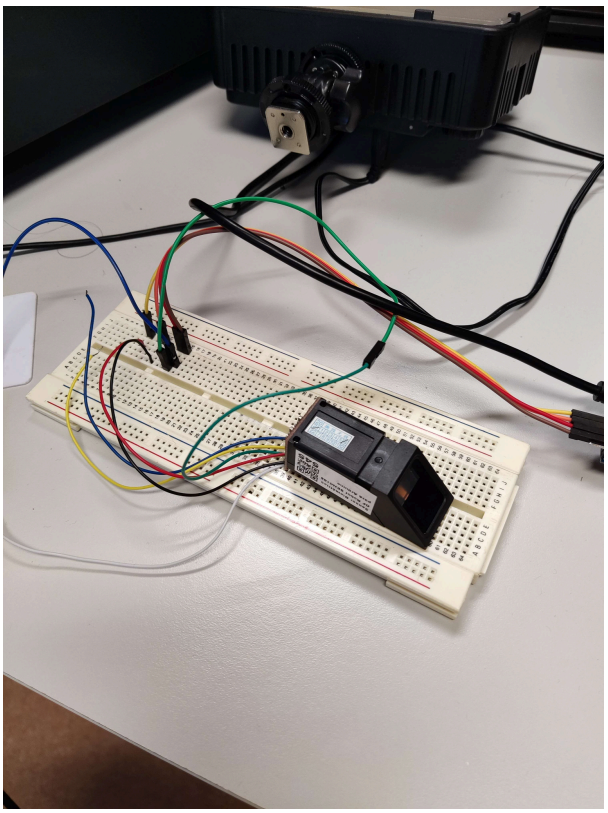
When registering a new face after the user passes the Liveness test, we simply store the user's face in the database so they can be authenticated later.

Fingerprint Recognition

In this section, we will talk about how we implemented fingerprint recognition.

Hardware

To get the fingerprint of a user, we used the DF Robot SEN0188 fingerprint sensor connected to a UART to USB converter, as shown in the following image.



Each cable of the sensor needed to be connected to the correct pin on the UART converter:

- Blue and Yellow off (Vtouch and Sout) are left hanging.
- Red -> Power 5v (Vin)
- Green -> Transmits data from the sensor to the converter (connected to the RX pin on the USB)
- White -> Transmits data from the USB to the sensor (connected to the TX pin on the USB)
- Black -> Groud (GND)

Software

We use the Adafruit Fingerprint python library to communicate with the sensor.
Instead of using the latest version, we use the 2.2.14 version as defined in our requirements file. After our chosen version, there was a lot of refactoring, and all the documentation examples were outdated.
Additionally, this library is supposed to be used for adafruit sensors and not DF Robot ones, but because they follow the same standards, the adafruit library can be used on DF Robot sensors available at the university.

Implementation

In our implementation, we took advantage of the fact that the sensor not only takes images of the fingerprint but also has additional functionalities.
The sensor is able to register fingerprint models in its memory and use that model to authenticate finger reads in a blackbox-like style. The only information we had access to was the image the sensor took of the user's finger as the figure below shows.

After invoking the authentication method, the scanner simply exposes in its interface if it found a match and what template matched the scan, so the authentication algorithm is unknown.

In our implementation, in the same folder where we store the face of a user, we also store a text file that contains the index of the model that their finger corresponds to in the sensor memory. This index is determined at the register time.

Taking this into consideration, our program's fingerprint recognition workflow goes as follows:

- Register:
    - Get a new index based on the number of templates stored in the sensor.
    - Check if the user already has a fingerprint, and if they do, delete the model
    - Create a three-scan model of the user's finger and store it in the sensor, with error checking at each step should the register process need to be restarted.

- Login
    - Scan the user's finger.
    - Try to match any template in the fingerprint sensor's memory.
    - See if the matched template in the fingerprint sensor is the same as the index stored in the database.
    - Authenticate the user if it is, reject the user if not.

User tests

Face Recognition

To test our Face Recognition system. We created different scenarios with different light conditions and with different people.

Normal Scenario: 240 lux of light and 60cm of distance.

Ethnicity and Gender	Light	Distance		Normal Scenario
	120 lux	30 cm	100 cm	
Caucasian Male (Nuno) Age 22	Success	Success	Success	Success
Caucasian Male (Kaluza) Age 22	Success	Success	Success	Success
Caucasian Male (Alexandre) Age 22	Success	Success	Success	Success
Imposter of Nuno	Failed Liveness	Failed Liveness	Failed Liveness	Failed Liveness
Caucasian Female (Lúcia) Age 23	Success	Success	Success	Success
Caucasian Male (Antonio) Age 44	Success	Success	Success	Success

We also tried to attack the system by using a real person blinking to pass the liveness test and then quickly switch to a photo of the user to authenticate. However, we didn't have enough time to switch to the photo immediately after the second blink.

Fingerprint Recognition

To test our Fingerprint Recognition, we created different scenarios with different positions of the finger.

Test subject	Position of the finger			Normal Scenario
	0°	45°	90°	
Caucasian Male (Nuno) Age 22	Success	Success	Success	Success
Caucasian Male (Kaluza) Age 22	Success	Success	Success	Success
Caucasian Male (Alexandre) Age 22	Success	Success	Success	Success
Caucasian Female (Lúcia) Age 23	Success	Success	Success	Success
Caucasian Male (Antonio) Age 44	Success	Success	Success	Success

Normal Scenario: Position of the finger 0° and the finger without grease.

With the results obtained in the previous section, we can conclude that the biometric systems implemented are difficult to deceive and good at identifying people.
As final results, we calculated a false acceptance rate (FAR) of 0% and a false rejection rate (FRR) of 0%.
With this, we can conclude that the system is pretty robust against attackers and very good at identifying users.

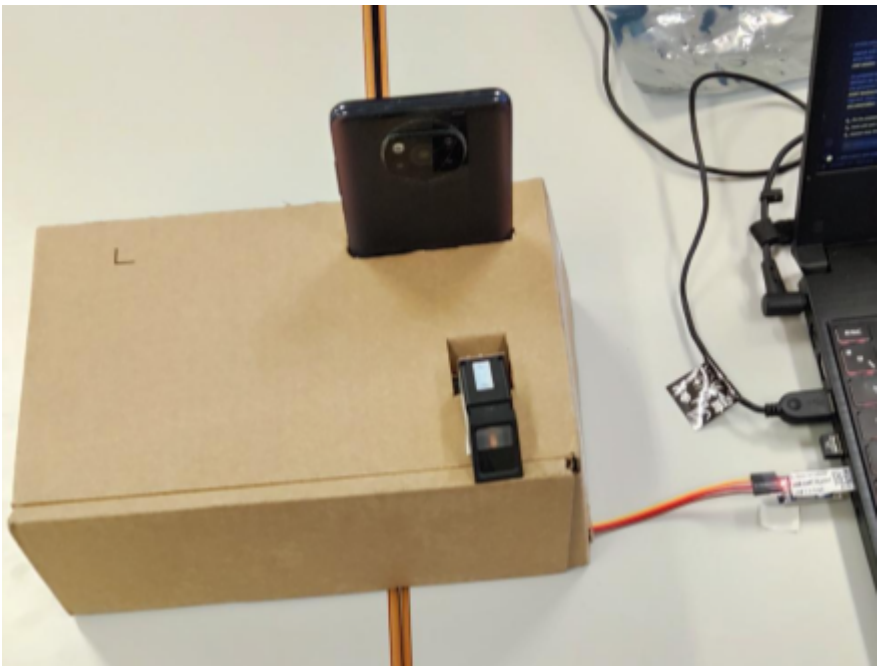
Tests procedure:

1. Mesure light using https://play.google.com/store/apps/details?id=com.doggoapps.luxlight
2. Measure distance using a measuring tape
3. Users do the usual authentication (follow the steps of the app)
4. Do this for the different distances and light conditions

Note: The imposter was made using a picture of the actual user

Final project setup

The final project setup is shown in the figure below, the box has 3 cutouts and is open in the back to easily access wiring. 2 cutouts in the top are used for the fingerprint sensor and the phone, the other is for the cables connecting to the UART-USB converter.



The image below shows the side view of the whole system, including the camera and computer which contains the instructions to the user.

Solved problems / Difficulties

- To test the mobile app we were obligated to compile it every time
- We tried to use the fingerprint in our Android smartphone, but the Android system doesn't allow us to access the fingerprint scanner and memory.
- We had several problems setting up the fingerprint device in connecting wires, welding and connecting everything to the PC. We even had to use a multimeter with the sensor hooked to a breadboard to measure the current in order to troubleshoot.
- We also had problems reading the data from the USB port, we didn't have permissions, but this was later fixed.
- To read data from the fingerprint device we used a library from a different brand because the original library was for Arduino. We ended up using the Adafruit library.
- We had some light problems in the face detection algorithm which obligated us to make a gamma correction. Later on, we removed gamma correction and changed our detection model.