Politecnico di Torino

Politecnico di Torino

Master's Degree in Computer Engineering
A.y. 2020/2021

Analysis of Webcam Fingerprint for User Authentication

Supervisors:
Enrico Magli
Tiziano Bianchi

Candidate:
Francesco Stagnitta

December 2021

Summary

Webcam fingerprint authentication is a novel password-less authentication method that aims to facilitate the user authentication process in order to make it more usable and still robust. This thesis analyses the extraction of the webcam fingerprint, a noise pattern that identifies each webcam among others, and the process of fingerprint matching using some sets of photos acquired through an acquisition campaign. After a deep bibliographic research, the thesis discusses models behind the construction of a capture script both for Windows and MacOS environments. This script, written in *Python*, was employed in the acquisition campaign to compose two datasets of photos taken from 30 different devices. These photos were later used for the simulation of the webcam fingerprint authentication. In this part, the fingerprint operations were performed using two distinct algorithms, those developed by *Fridrich* and *ToothPic*, respectively used for fingerprint extraction and fingerprint matching. The results of the simulation were analysed by means of ROC curves that assessed the accuracy of the authentication scheme using the two algorithms. In order to obtain the best results, the problem of *Non-Unique Artifacts* had to be resolved. For this purpose, Wiener filtering was applied in the frequency domain as a post-processing operation on the extracted fingerprint. By applying this filter, and observing the related ROC curves, it was demonstrated that the webcam fingerprint authentication process can be used as a robust authentication method both for Windows and MacOS devices.

Acknowledgements

Throughout the work behind this thesis I received a great support. I would like to acknowledge every person that contributed by making computers available for this research and giving useful advice to finish the dissertation in the best possible way.

Moreover, I would like to thank my parents, my sister, and my entire family who have supported me from the beginning of my studies.

Finally, I would like to thank my friends, particularly Giulia and Salvo, who have always been a rock to hold onto in difficult times, during my university years.

"It's fine to celebrate success, but it is more important to heed the lessons of failure"
Bill Gates

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The authentication world has always been a fundamental part of the Internet because it is essential for the users to have a secure experience when accessing the services offered by the web. Basically, authentication is the act of proving something to the system that will decide whether the information provided by the user is enough to pass the authentication process. Authentication relies on three factors which prove different things [1]:

- **Knowledge**: It is something the user knows (e.g., password)
- **Possession**: It is something the user has (e.g., token, identification device)
- **Inherent**: it is something the user is (e.g., biometric)

These factors can be combined in different ways to create *Single-Factor Authentication* or *Multi-Factor Authentication* (MFA) schemes.

1.1 Authentication State-Of-The-Art

The predominant authentication system is *password-only authentication* which is based on the knowledge factor, as the user must prove the knowledge of a password in order to be authenticated by the system. However, over the years, it was demonstrated that single-factor authentication with the use of passwords is extremely weak for many reasons. First, online guessing attacks are becoming more intelligent and effective, given the fact that most users adopt weak and reused passwords. Then, large-scale password database leaks happening in the last few years have revealed the critical problem of password storage.

A solution to this problem has been identified in MFA systems, such as *2-Factor Authentication* or *Risk-Based Authentication*, which rely on additional factors in order to perform the authentication. Nowadays, following the state-of-the-art, MFA is implemented with the use of the knowledge factor, through password authentication, together with the possession factor, using a security token sent via e-mail or SMS. Although this solution is very robust, it lacks usability since the user is required to perform many actions to obtain access. Google reported in January 2018 that 2FA was enabled by less than 10% of their users [2].

Finally, biometric authentication, in which the user is required to use one of his identifying characteristics (i.e., fingerprint, face), is the most secure authentication method because it is very difficult for an attacker to steal the used authentication factor. However, biometric data are regulated in EU by the GDPR [3] and, thus, they cannot be processed. So, biometric authentication can only be performed locally in the user devices.

1.2 Webcam Fingerprint Authentication

In the last few years, considering the growing number of users on the internet and the every day increase in cyber-attacks, many attempts have been made by researchers from all over the world to develop new systems that guarantee the robustness of authentication and, at the same time, are more usable than the solution adopted in recent days.

In this thesis a new password-less authentication scheme, which relies on the possession factor, is introduced. It can be either used as an additional factor for MFA, or even in a single step authentication protocol. Indeed, the possession of a specific webcam is basically verified by extracting its fingerprint. This system has already been patented and tested by ToothPic for the smartphone camera, proving that the camera fingerprint has a good discriminating value in identifying the right device. In this process, the PRNU extracted from the camera sensor was used as a *Physical Unclonable Function* to perform cryptographic operations [4]. This was possible thanks to the characteristics of the PRNU, which is unique for each sensor and stable over its lifetime.

The following analyses try to apply the camera fingerprint authentication system in a PC scenario, using the webcam to capture photos. Hence, it is demonstrated how to extract a good PRNU from the acquired images and whether the extracted webcam fingerprint is able to detect the various PCs, which are part of a large dataset.

When it comes to PCs it is important to note that the user can perform the authentication with either the built-in webcam or an external USB webcam. In the first case the user proves to the system that he is in possession of the PC in which the webcam is incorporated, because all operations in the authentication are done from the same computer. In the second case the authenticated device is the external webcam, since the user can use this webcam in every PC. Therefore, the external USB webcam is used as an authentication token.

1.3 Thesis Objectives

The main purpose of this thesis is to implement webcam fingerprint authentication, in Windows and MacOS devices, and see if the system is as robust as the one developed in the smartphones scenario by ToothPic.

To find these results it was decided to follow a fixed path. Firstly, a deep bibliographic research on the methods to take photos from the selected operating systems is necessary. Frameworks, tools, and libraries were analysed so to find how to access the webcam sensor in order to obtain the least processed images possible. After the selection of the tool, in the acquiring phase, it is needed to build

a script able to acquire the appropriate photos from the PCs webcam. This script can be used in an acquisition campaign so to involve many devices. Then, when the photos acquired in the campaign are collected, the fingerprint extraction and fingerprint matching processes can be applied to simulate the webcam fingerprint authentication scheme. Lastly, the obtained results can be analysed using ROC curves to evaluate the accuracy of the applied authentication method.

Chapter 2

Background

In order to fully understand the experiments explained in this thesis, it is necessary a quick introduction to the concepts of *webcam fingerprint* and how it specifically works for user authentication.

Then, the ToothPic authentication scheme is also presented. This authentication scheme has already been developed for smartphone authentication, using the fingerprint of the smartphone camera.

Finally, one of the most important issues to deal with is the privacy impact of this new authentication method, compared to the existing ones.

2.1 PRNU

The *photo-response non-uniformity* (PRNU) is the main characteristic on which the whole webcam fingerprint authentication system is based. Basically, it is a pixel-to-pixel noise generated by physical imperfections of each pixel; these imperfections are caused by very small variations in the physical dimensions of the pixels, and by the inhomogeneity naturally present in silicon, which contributes in the ability to convert photons to electrons [5].

This noise is deeply interesting for our purpose because it is generated by each pixel, so these imperfections are not correlated. Taking a whole webcam sensor, composed by at least hundreds of thousands pixels, it is possible to obtain a noise pattern such that is very likely assured to be unique for each webcam sensor.

Moreover, PRNU is a systematic part of the noise that can be used as an unclonable physical characteristic of the device because it does not change among the different images taken by the same webcam, and is relatively stable over the webcam life [6].

In order to obtain a good PRNU, it is important to have a good light intensity, since the PRNU term is not visible in saturated areas of the image which contain

dark content. Therefore, it is possible to say that PRNU, extracted in favorable conditions, is a very useful forensic quantity due to its properties of dimensionality, universality, generality, stability, and robustness [5].

2.2 Reference Fingerprint Extraction

The first step required to estimate the webcam fingerprint is the collection of the reference images. From these reference images, the *reference fingerprint*, which is the fingerprint that uniquely represents the webcam, is extracted. Because of this reason, the reference fingerprint must be of the best possible quality, and it is important to extract the PRNU pattern from the most appropriate images.

A set of 20 to 50 images with white and uniform content, as written in 2.1, is necessary to obtain a good PRNU pattern [7]. The purpose of these reference images is to illuminate the webcam sensor with a light intensity as more uniform as possible, and to avoid dark areas in the images, which would decrease the quality of the PRNU.

The obtained fingerprint is, in substance, a matrix of double-precision values with the same dimension as the sensor of the webcam. Each matrix cell represents the PRNU value of one pixel and, together with the other uncorrelated values, it will compose the pattern that identifies the webcam.

The fingerprint can be extracted from different image formats. Thus, if the image is in gray-scale, it is only needed to extract the PRNU pattern; otherwise, if the image is in RGB, firstly the PRNU pattern for each channel must be obtained, and, then, the RGB-to-gray conversion must be applied to recover a "global" PRNU fingerprint [7].

2.3 Fingerprint Compression

The fingerprint in double-precision values brings many usability and technical problems because of its size. Since each cell contains a double-precision value, occupying 64 *bits* in memory, the total fingerprint size can be in the order of some *Megabytes*.

This size can be a problem in terms of network connection, when the fingerprint is sent over the network, and in terms of database size, when many reference fingerprints have to be collected for authentication purposes.

A solution to deal with this problem is the fingerprint compression via *binary random projections*. As demonstrated by Valsesia et al. [7], this compression method is based on the computation of products between the real-valued fingerprint and random vectors, which compose the sensing matrix. Then, it is possible to obtain a binary fingerprint quantizing the measurements while keeping their sign.

It follows that *binary random projections* allow to reduce the fingerprint dimension to one *bit per pixel* with slight information loss when the comparison between the different fingerprints is done.

2.4 Fingerprint Matching

The fingerprint matching problem comes when there is a reference fingerprint for a given webcam, and it is required to decide whether a set of test photos (≈ 100) have been captured by the same webcam. So, a test fingerprint, that needs to be compared to the reference fingerprint, will be extracted from these photos.

As seen in the previous sections, there are two different types of fingerprints:

- *Double-valued* fingerprint
- *Binarized* fingerprint

For each fingerprint type there exists a different method to solve the problem of fingerprint matching. Both methods consist in finding how much the reference fingerprint and the test fingerprint are similar, so a correlation value is calculated.

Then, it is needed to set a proper threshold value that will be compared to the calculated correlation value. From this comparison it will be possible to detect whether the test fingerprint matches the reference fingerprint, or not.

2.4.1 Correlation Coefficient

When we have to compare real-valued uncompressed fingerprints, a correlation coefficient must be computed and, then, compared with the selected threshold. If the calculated correlation coefficient is higher than the threshold, the test fingerprint matches the reference fingerprint.

Representing the fingerprints as real-valued vectors, the correlation coefficient is defined as [4]

$$\boldsymbol{\rho} = \frac{\boldsymbol{k_1}^T \boldsymbol{k_2}}{\| \boldsymbol{k_1} \|_2 \| \boldsymbol{k_2} \|_2}.$$

In this equation $\boldsymbol{k_1}$ is the first fingerprint, $\boldsymbol{k_2}$ is the second fingerprint, and $\boldsymbol{\rho}$ is the correlation coefficient between k_1 and k_2.

2.4.2 Hamming Distance

When dealing with *binarized* fingerprints it is necessary to determine whether the sequence of bits that represents the reference fingerprint is equal to the one representing the test fingerprint.

Given two bit sequences of the same size, the Hamming distance is the number of positions in which the bits are different between the two sequences.

Using this distance it is possible to calculate a comparison term which allows to determine whether the binarized fingerprints are the same or not. If the calculated Hamming distance is higher than the threshold, the test fingerprint does not match the reference fingerprint.

2.5 ToothPic scheme

As mentioned in the introduction, the aim of this thesis is to implement the scheme of camera fingerprint authentication, developed by ToothPic for smartphones, in a PC scenario.

ToothPic [8] is a company working in the field of Cybersecurity that has developed and patented a technology to make each smartphone camera a secure key to perform cryptographic operations.

Its authentication scheme is a solution for multi-factor authentication which ensures a good trade-off between usability and security, as the user is not asked to perform complicated tasks while the authentication mechanism remains highly robust. This is guaranteed because this system obtained the FIDO2 certification [9], meaning that camera fingerprint authentication is compliant to the FIDO Alliance's newest set of specifications.

2.5.1 How it works

ToothPic's technology is based on the concepts of camera fingerprint explained in the previous sections. The fingerprint extracted from the set of photos taken by the smartphone camera is used as the physical secret to implement a physical unclonable function (PUF). In this way it is possible to obfuscate the private key of an asymmetric key pair following the implementation represented in Figure 2.1.

When dealing with *asymmetric cryptography*, the most crucial task is to secure the storage of the private key. At the moment of the creation of a new asymmetric key pair, the private key is firstly obfuscated with the extracted camera fingerprint, and then stored inside the user's device. The obfuscation of the private key does not expose any information about the key itself, so a potential attacker, who steals this obfuscated private key, will not be able to perform any operation.

Whenever the user needs to use the private key to compute a cryptographic operation, the system will capture another set of photos, and will generate again the fingerprint on-the-fly. Then, the private key is de-obfuscated and it is ready to be used to sign a document, or to solve a challenge message.

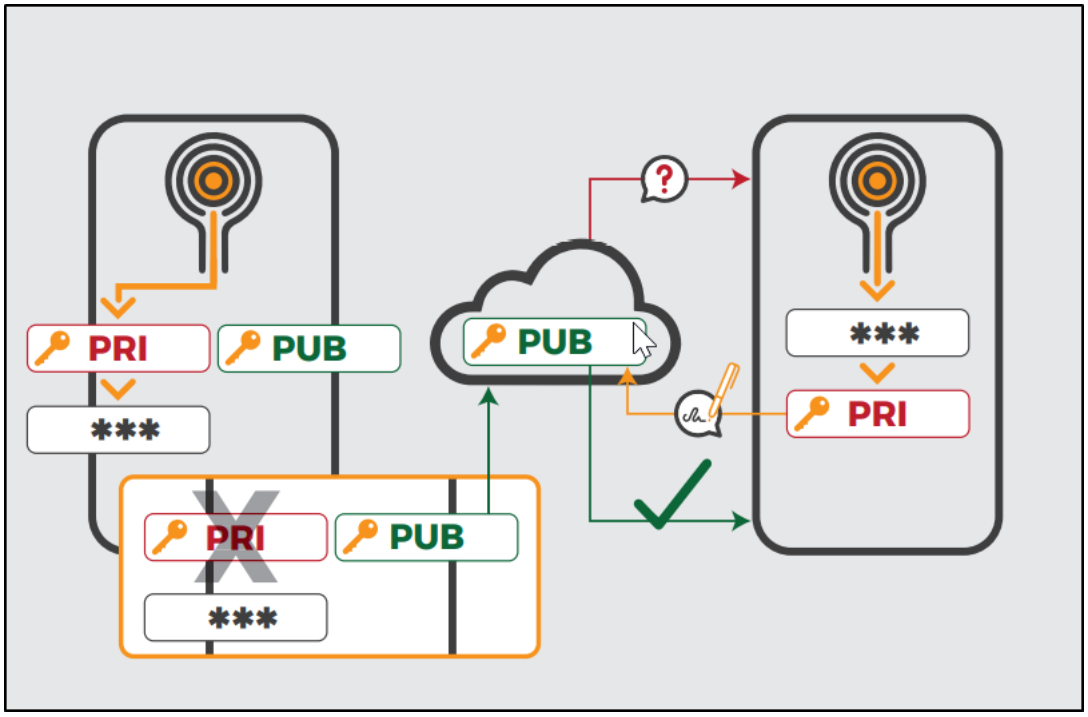


Figure 2.1: ToothPic scheme

2.5.2 RAW Photos

An important consideration in the mechanism explained above is the use of RAW photos [4]. This is necessary because it is possible that a potential attacker could extract the camera fingerprint from a set of images uploaded to the Internet (e.g., to a social network) in a JPEG format.

The usage of unprocessed photos eliminates this problem because they are not affected by JPEG compression. Indeed, JPEG compression deletes some high frequency information from the image. This characteristic is exploited by the ToothPic's system using only the high frequency information present in RAW photos by means of a high-pass filter.

2.6 Privacy Considerations

From the user's privacy point of view it is possible to focus on some considerations about the webcam fingerprint authentication technology.

Firstly, a decisive feature of webcam fingerprint is that it does not need to store any biometric or sensitive personal data. Therefore, it is possible to process webcam fingerprint data without the strictly restrictions defined in Article 9 of

GDPR [3], which comes into force when biometric and sensitive data are processed.

Furthermore, following this system, it is not needed to store any images. In fact, having the fingerprint of the webcam, it is not possible to obtain any information about the image from which it was extracted.

Consequently, it is possible to say that the use of webcam fingerprint allows to have a robust and usable system which does not compromise the privacy of the users.

Chapter 3

Capture Script

This chapter discusses the bibliographic research of possible methods to access the PC webcam in order to capture some photos needed by the webcam fingerprint process.

Afterwards, the chosen library, OpenCV, is deeply analysed by looking at the possible programming languages to use and evaluating the different photos captured with various settings of the webcam.

Finally, the *Python* script, created to take the photos in the campaign, is analysed and explained.

3.1 Software and API Research

In the world of *Personal Computers* there exist many libraries, software and frameworks able to access the webcam to obtain some images. However, heterogeneous environments and different operating systems must be taken into account.

In this thesis the attention is only focused on the most available operating systems on the market, *MacOS* and *Windows*.

3.1.1 Physical Driver Access

The lowest possible level is to directly access the webcam drivers, however this solution was not considered because it is not a scalable solution.

In fact, the PCs, especially those with Windows as operating system, have several webcams which require dedicated drivers. Moreover, it is also imaginable that, in the future, the number of drivers to deal with will grow, and the old ones will be updated with new functionalities.

Hence, it is truly difficult to write a program for each available driver, and to keep the whole capture process up-to-date.

3.1.2 Operating System's Frameworks

Going up to the operating system level, Windows and MacOS make available some frameworks capable of accessing the webcam in order to allow programmers to write camera applications in an easier way. The following multimedia frameworks are available:

- **AVFoundation**: Successor of the Apple's AVKit API and is usable by every Apple device. For the purpose of this thesis, it is available from MacOS 10.7 or later versions. AVFoundation provides an API that gives the possibility to access the Mac's built-in webcam so to write camera application in *Swift*, which is the used programming language. Differently from iOS and iPadOS versions, in MacOS it is not possible to capture photos in an unprocessed format [10].
- **DirectShow**: API created by Windows to support audio and video streams, and to manage multimedia files. DirectShow is designed for *C++* and is distributed by the Windows SDK. Although it supports a lot of formats, there is no mention of a possible RAW format acquisition in the provided documentation [11].
- **Microsoft Media Foundation**: Next generation multimedia platform for Windows, part of the Windows SDK, that will take the place of DirectShow after a period of co-existence. It is available from Windows Vista or later, and requires the use of *C/C++*. Like DirectShow, the Microsoft Media Foundation's documentation does not mention support to an unprocessed format [12].

3.1.3 Cross-Platform Tools and Libraries

Another option to take into consideration is to exploit some third-party tool or library that is able to operate regardless of the operating system installed on the device. In this way it is possible to build a portable script that can be used in both Windows and MacOS environment.

- **OpenCV**: Cross-platform library for computer vision applications, originally developed by Intel. OpenCV is free of charge, since it is open-source, and available for both desktop and mobile operating systems. Thanks to its wide adoption, it is well documented and easy to use. It is natively written in *C++*, but offers also bindings in other programming languages such as *Python*, *Java* and *MATLAB* [13].
- **FFMPEG**: Open-source software for multimedia, able to manage input/output streams, and format conversion. FFMPEG is available as cross-platform

command line tool, or as a set of libraries that can be used by programmers to build applications. These libraries are called *libav* [14] and are written in the *C* programming language [15].

- **Scikit-image**: Open-source and cross-platform collection of algorithms for image processing. It is mainly written in *Python* [16].
- **SimpleCV**: Open-source framework available in *Python* that allows to access high-powered computer vision libraries, such as OpenCV, in an easier way [17].

3.2 OpenCV Properties

The selected library used to access the PC webcam is OpenCV. This choice was made considering the fact that OpenCV is cross-platform and, examining the source-code, does not apply any image processing, directly providing the output made available by the selected operating system framework.

Even if OpenCV does not offer the possibility to get the RAW image, none of the methods discussed in 3.1 seem to be able to retrieve it.

3.2.1 Programming Language

The first version of OpenCV was written in *C* but, nowadays, the older interface is deprecated in advantage to the newer *C++* version. Hence, the entire library is now available through a primary *C++* interface. In any case, OpenCV provides programmers with some bindings in *Python*, *Java*, and *MATLAB*.

Therefore, the selected programming language was *Python*, because it can be easily used as a script. So, the usage of *Python* has speeded up and simplified the construction of the capture application, in which the *opencv-python* [18] library was adopted.

3.2.2 Frameworks

When OpenCV is asked to open the webcam, it is necessary to load the proper library in order to interact with the operating system in the right way. There are three different properties to pass as parameters to the *VideoCapture* function:

- **CAP_AVF**: This represents the AVFoundation framework, and it is needed in the MacOS environment.
- **CAP_DSHOW**: This represents the DirectShow framework used in Windows.

- **CAP_MSMF**: This represents the newer Microsoft's multimedia framework, Microsoft Media Foundation.

If for the MacOS script the choice was forced, for the Windows one there was the possibility to choose between DirectShow and Microsoft Media Foundation. The choice fell on Microsoft Media Foundation because in the future it will be the Microsoft's standard multimedia framework.

3.2.3 Uncompressed Formats

OpenCV makes available some uncompressed image formats, depending on both the properties set by the programmer, and the capabilities of the frameworks. In fact, not all the properties given by OpenCV will be actually set in the camera by the framework, but only the available ones. The output image is an *uint8* array in which the bytes, represented in the cells of Figure 3.1, can be organized in the following ways:

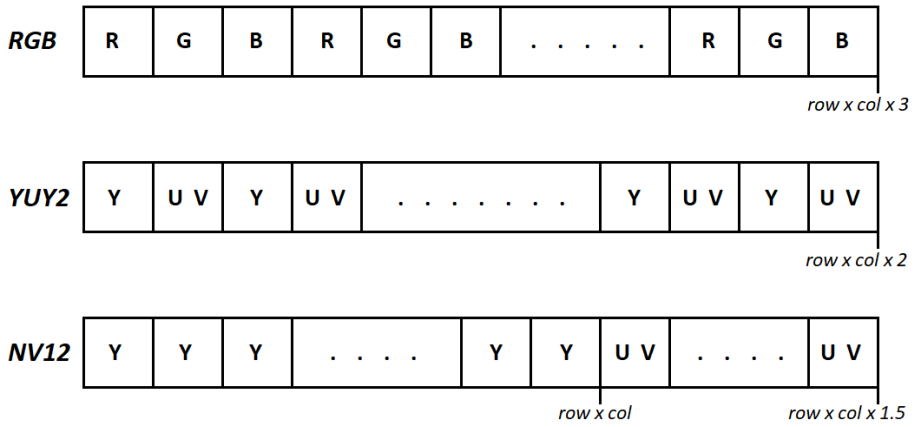


Figure 3.1: Available formats in OpenCV

- **RGB**: In the RGB format the image is represented by 3 channels: red, green, and blue. Every pixel will produce 3 bytes, one for each channel, so the obtained image length is 3 times the number of pixels of the webcam sensor.
- **YUY2**: This format is characterized by two parts: the luminance, represented by the Y channel, and the chrominance, represented by the UV channel. For the purpose of this thesis it is needed to extract only the luminance part.

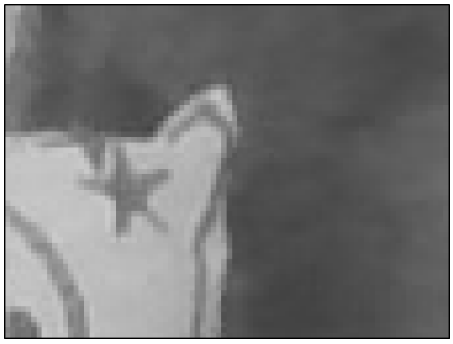
- **NV12**: This format is almost similar as the YUY2 format, but the chrominance part is smaller and the bytes are differently organized.

Thanks to the *PROP_CONVERT_RGB* property, which manages the conversion of the image into RGB format, it is possible to obtain these formats. If it is enabled the output will be an RGB image, if it is disabled a YUY2 or NV12 image can be obtained, depending on the webcam specifications.

While in MacOS it is not possible to obtain neither YUY2 nor NV12, so the only possible format is RGB, in Windows it is possible to obtain all three formats.

3.2.4 JPEG vs Uncompressed Format

(a) *RGB image*



(b) *NV12 image*

(c) *JPEG image*

Figure 3.2: Differences in the pixels between uncompressed images, (a) and (b), and a compressed image, (c)

Taking a photo with the same webcam, it is possible to notice the differences among the various image formats. In particular, as shown in Figure 3.2, the JPEG

compressed image, (c), reveals the classical 8x8 block division used to compress the photo.

Differently, from the uncompressed images, (a) and (b), it is possible to see a better quality of the image that is also reflected in the higher images size.

In this case, as explained in 3.2.3, the NV12 image was obtained disabling the OpenCV property required for RGB conversion. Then, only the luminance component was extracted from the acquired data.

3.2.5 Resolutions

Another aspect to deal with is the photo resolution, since OpenCV gives the opportunity to modify it. Considering that the webcams can capture photos in different resolutions, it is needed to select a unique resolution in order to have fingerprints of the same size. This is important because, as written in 2.4, two fingerprints of the same size are necessary for the fingerprint matching process.

In MacOS devices OpenCV is not able to modify the resolution, so it is possible to capture photos only with $1280x720$ resolution.

In Windows devices it is possible to set the preferred resolution. By default, OpenCV sets the $640x480$ resolution but, in some webcams, it is possible to select the $1280x720$ resolution.

(a) $640x480$ resolution

(b) $1280x720$ resolution

Figure 3.3: Images captured in two different resolutions

From Figure 3.3 it is possible to see that, selecting the two different resolutions, the photo is horizontally cropped and properly scaled, since the two resolutions have a different aspect ratio. Hence, there is no image processing when the resolution is modified in OpenCV. However, the resolution can be a potential problem in some old Windows devices because the maximum one is $640x480$. This characteristic will limit the fingerprint length of every PC in the entire system.

3.3 Photos Analysis

Before the creation of the script, it is needed to select which properties to set in OpenCV for the image format and resolution.

While the choice is forced for the MacOS script, and the retrieved images will be in *RGB* format with $1280x720$ resolution, further image analyses are required for the Windows script.

Even if with OpenCV it is not possible to obtain unprocessed images, it is still important that the uncompressed image captured by OpenCV has not been reconstructed from a JPEG image. This event would have resulted in a loss of information in the image.

3.3.1 JPEG compression

The JPEG compression is based on the following steps [19]:

1. First of all the image, converted in grayscale, is divided into 8x8 pixel blocks.
2. Then, for each block, the *Discrete Cosine Transform* (DCT) is calculated
3. The 64 DCT coefficients are uniformly quantized, using a quantization table of 64 elements. The quantization consists in the division between the DCT coefficients and the corresponding elements of the quantization table. The result is rounded to the nearest integer.
4. Finally, the resulting integer matrix is encoded.

3.3.2 JPEG reconstruction

When the uncompressed image is obtained by the reconstruction of a JPEG image, it is applied the reverse process, with respect to the compression one.

The idea is to apply the JPEG compression to the uncompressed image, and find whether the DCT coefficients of a specified spatial frequency are multiple values of a possible quantization element.

If the histogram of the selected frequency shows only the multiple values of the quantization element, the image was reconstructed from a JPEG photo, otherwise, if the histogram is continuous, the image was not reconstructed.

3.3.3 Results

The experiment was done on the images captured with the combination of webcam properties discussed before. Two spatial frequencies are pictured for each image.

The first one is for the DC frequency, in the $[1, 1]$ position of each 8x8 block, where most of the image information is contained. The second one is for a possible AC frequency, in position $[1, 2]$ of each 8x8 block.

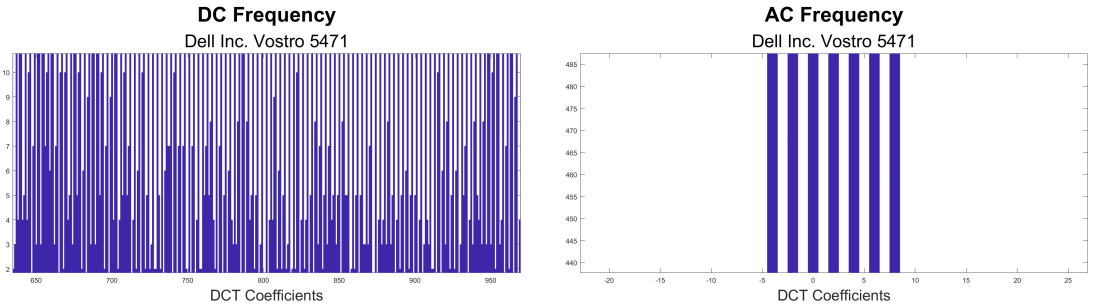


Figure 3.4: RGB image with $1280x720$ resolution in Windows

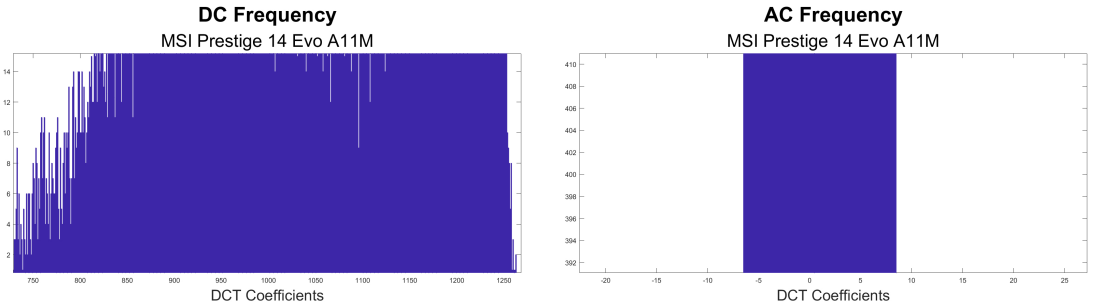


Figure 3.5: RGB image with $1280x720$ resolution in Windows

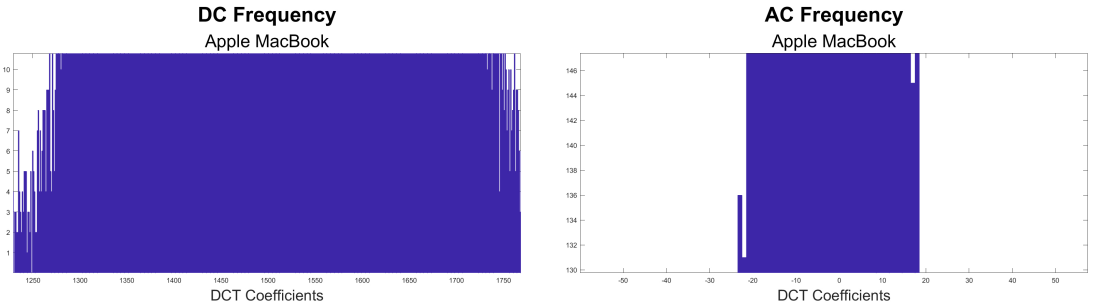


Figure 3.6: RGB image with $1280x720$ resolution in MacOS

Except for the RGB image taken in a particular Windows PC, Figure 3.4, every photo captured with the various webcam properties appears not to have been reconstructed from a compressed JPEG image. It is possible to verify this because

the values of the represented histograms are continuous, and not multiple of a quantization element.

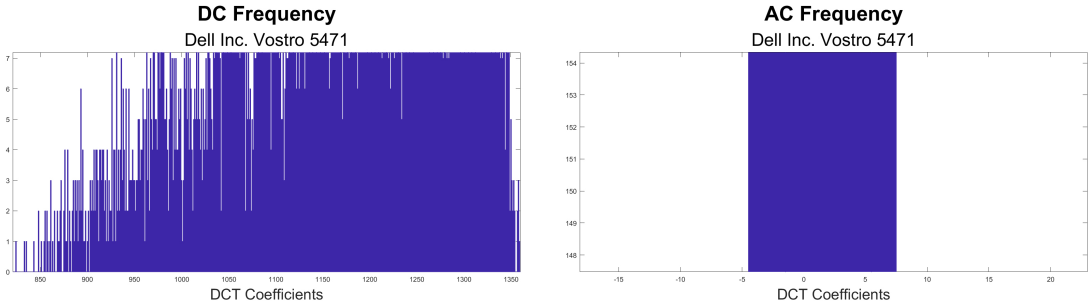


Figure 3.7: RGB image with 640x480 resolution in Windows

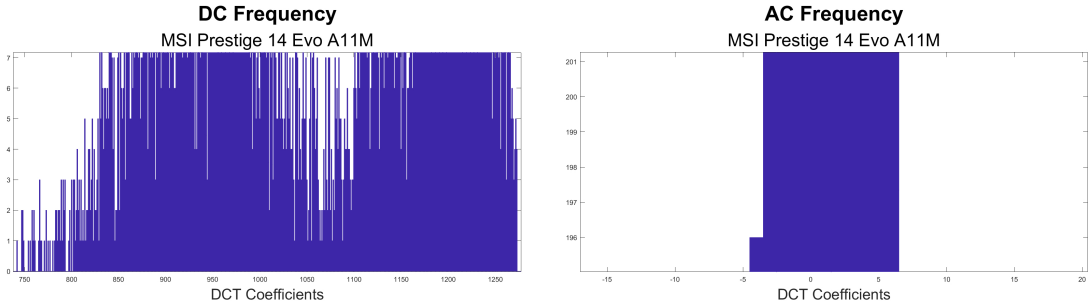


Figure 3.8: NV12 image with 640x480 resolution in Windows

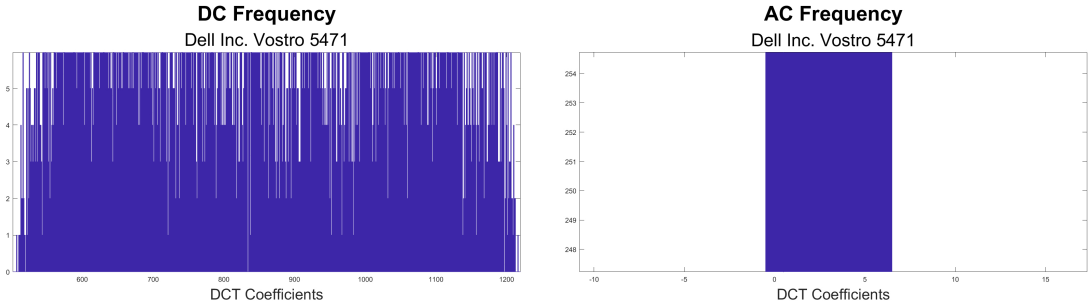


Figure 3.9: YUY2 image with 640x480 resolution in Windows

As written, the JPEG reconstruction can be visible in Figure 3.4, where the graph is not continuous. In this case the photo was acquired by a *Dell Inc. Vostro 5471* with Windows as operating system. Comparing these histograms to those obtained from other images, with different properties, of the same computer (Figure

3.7, and Figure 3.9) it is possible to see that the property, which introduced this event, was the resolution setting to $1280x720$.

It is not possible to say that the RGB image with $1280x720$ resolution will be reconstructed from a JPEG one in every Windows PC. This fact is demonstrated in the histograms obtained by a photo captured in another Windows PC, *MSI Prestige 14 Evo A11M*. As we can see in Figure 3.5, the histogram is continuous, which means that the image was not reconstructed from a JPEG one.

Therefore, since not every Windows PC is capable of acquiring non-reconstructed images in RGB format with $1280x720$ resolution, it was decided to write a script that acquires uncompressed RGB images with the OpenCV default resolution, which is $640x480$ in Windows and $1280x720$ in MacOS.

3.4 Dropbox Saving

At this point, before definitively writing the entire capture script and starting the acquisition campaign, it is necessary a way to collect the photos taken with OpenCV on the various PCs.

Thanks to the Dropbox account offered by the *Politecnico di Torino*, it is possible to create the repository where all data is saved.

3.4.1 Dropbox Apps

Dropbox offers developers the possibility to create *Dropbox Apps* on the DBX platform. Thanks to the SDKs made available by Dropbox, developers can access their own Apps in order to save data inside a dedicated folder within the Dropbox repository. The folder used by the Dropbox App remains private thanks to an authorization token necessary to access the folder.

These SDKs are available in many programming languages such as *HTTP*, *JavaScript*, and *Python* [20]. Since it was previously chosen to use OpenCV *Python* APIs, *Python* was also chosen as the DBX platform SDK. In this way it is possible to write a script, as explained in 3.4.2, that uploads every data inside the private Dropbox folder created in advance.

3.4.2 Saving Script

Following the *Dropbox for Python* documentation [21],the script used to upload data to Dropbox must first import the *dropbox* package. Then, a Dropbox API object, called *dbx* in Figure 3.10, is instantiated using the access token of the Dropbox App. In this way the object is directly connected to the App created in Dropbox and it is able to make API calls.

Next, it is required to find an available subfolder, inside the folder of the Dropbox App, where to load the whole acquired data.

Finally, there is the uploading process in which every file contained by the *data* directory is uploaded to the destination folder inside Dropbox using the *"files_upload()"* API.

```
10  def folder_is_unavailable(log_file, dbx, folder_name):
11      try:
12          result = dbx.files_list_folder(path="")
13          for entry in result.entries:
14              if entry.name == folder_name:
15                  return True
16      except Exception as err:
17          log_file.write("Exception raised in the selection of an available folder!\n\n")
18          raise err
19      return False
20
21  def upload_file(log_file, dbx, filename, source_path, dest_path):
22      try:
23          data = open(source_path + "/" + filename, "rb")
24          dest = "/" + dest_path + "/" + filename
25          dbx.files_upload(data.read(), dest, autorename=True, mute=True)
26      except Exception as err:
27          log_file.write("Failed to upload %s/%s\n\n" % (source_path, filename))
28          raise err
```

Figure 3.10: Dropbox APIs usage

3.5 Creation of the Capture Script

The final capture script will firstly install the required libraries and then call the various *Python* scripts. In order to make it portable it was written in command languages.

For the MacOS version the *bash* programming language was used, so the user has to run it from the Terminal. For the Windows version the *PowerShell* programming language was chosen, available only on Windows, because it is able to generate an executable file that will automatically run the script.

The acquired images will be placed in a dedicated folder, called *data*, together with a log file, *info.log*, which will contain every action performed by the scripts so to record some information and detect possible problems. The script is designed to perform the following tasks:

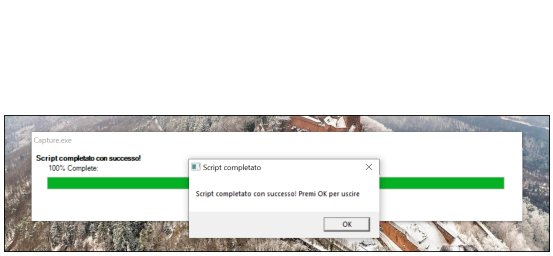
1. Install, if not already present, *Python 3.9.5* using a provided installation executable.

2. Install the needed *Python* packages by means of *pip install*.
3. Obtain the device information such as PC model, OS, and *Python* version.
4. Obtain webcam properties, such as default and maximum resolution, through the use of OpenCV
5. Acquire 30 reference photos running the proper *Python* script
6. Acquire 100 test photos running the proper *Python* scripts
7. Save the entire *data* folder inside the Dropbox repository using the script discussed in 3.4.2

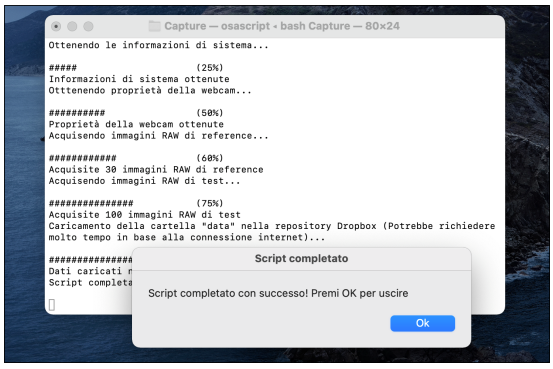
3.5.1 Acquiring the Images

Before calling each *Python* acquiring script, a popup message is displayed, along with a webcam preview, showing the user how to properly take the photos, according to what written in Chapter 2.

- The reference images have to be composed by white and uniform content, so it is recommended to put a white sheet of paper in front of the webcam.
- The first 50 test images need to portray normal PC usage by the user.
- The last 50 test images have to frame another scene where there are no human faces.



(a)



(b)

Figure 3.11: Examples of the scripts for Windows, (a), and MacOS, (b)

Chapter 4

Dataset Creation

As explained in the previous chapter, the created script is designed to get 30 reference images and 100 test images from every device. These images are taken in RGB format with the default resolution chosen by OpenCV for each different device. Nevertheless, it was later decided to conduct a second acquisition campaign by forcing the resolution of the images to $1280x720$.

This chapter describes a smaller dataset used to check whether the fingerprint, extracted by means of the *Fridrich algorithm*, is visible from the webcam photos or not.

Afterwards, it is given a brief description of the algorithm developed by ToothPic for the camera fingerprint system, discussing the parameters that have to be adapted to the photos captured by webcams and showing the results of the algorithm application.

Lastly, the script application to an external USB webcam is analysed and it is represented the construction and the composition of the two acquisition campaigns datasets.

4.1 First Dataset

The first thing to do was to extract the fingerprint and see whether it was visible or not. It means to examine if the reference fingerprint matches the test fingerprint of the same PC and, at the same time, does not match the test fingerprints of different PCs. For this purpose, a small dataset was created, described in Table 4.1, consisting of 2 Windows devices and 2 MacOS devices.

For the Windows PCs, an *MSI* and a *Dell*, the resolution of the images is $640x480$ because, according to section 3.2.5, that resolution is the default one. Differently, the default photo resolution set by OpenCV in the two Apple devices is $1280x720$.

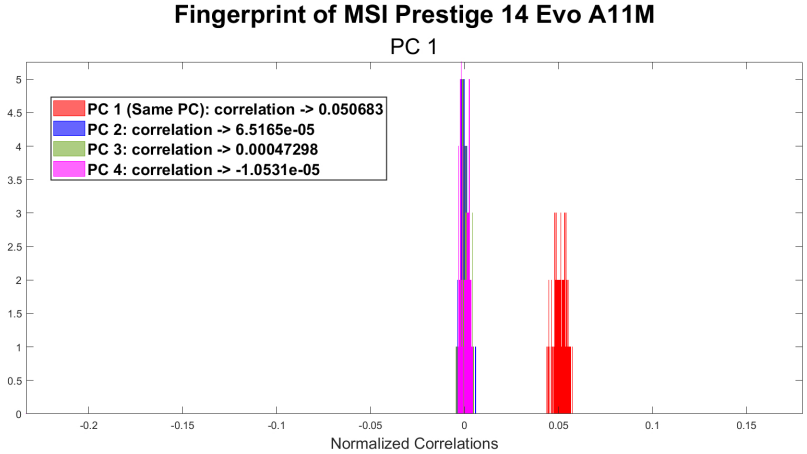
Table 4.1: List of devices in the first small dataset

Model	OS	Resolution	ID
MSI Prestige 14 Evo A11M	Windows 10.0.19043	640x480	PC 1
Dell Inc. Vostro 5471	Windows 10.0.19042	640x480	PC 2
Apple MacBook	MacOS 10.13.6	1280x720	PC 3
Apple MacBook	MacOS 10.16	1280x720	PC 4

4.2 Fridrich Algorithm

At the beginning, the activity performed on these image sets was the extraction of the reference fingerprints using the *Camera Fingerprint Matlab toolbox*, based on the findings of Jessica Fridrich [5]. Then, each reference fingerprint was compared with the four fingerprints obtained from the sets of test images, in order to find out whether that fingerprint, through the correlation value, will match the one of the right device or not.

An important thing is the crop used for the images. Since the fingerprints need to have the same size to be compared and the captured images, as shown in Table 4.1, have different resolutions, every image is cut to the minimum resolution of 640x480. Thus, the fingerprint is represented by 307200 double values, one for each pixel, and its size is 2.458 *MB*.



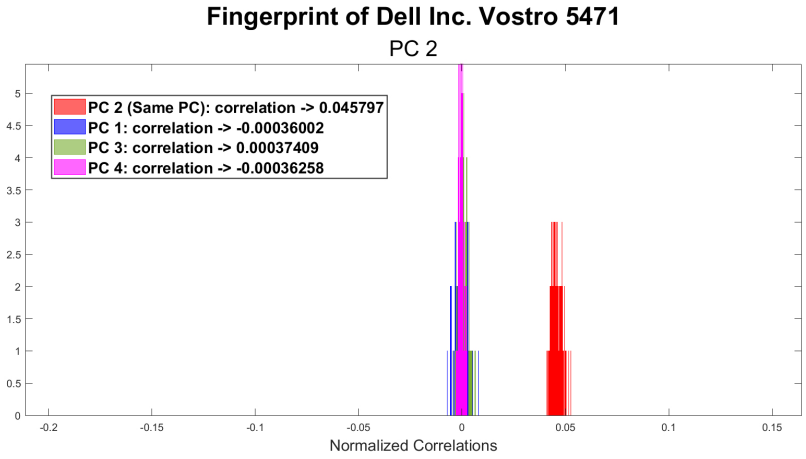
(a)

Figure 4.1: Fingerprint matching PC 1, (a)

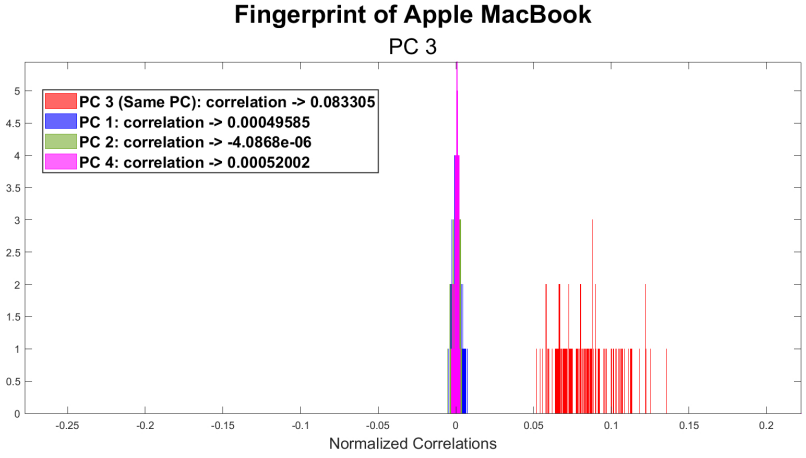
4.2.1 Results

The calculated histograms, represented in Figure 4.1, show the correlation between the reference fingerprint and the fingerprint extracted from each test image. The 4 test-sets are defined by different colours.

From the results it can be seen that the reference fingerprint and the test fingerprint of the same webcam have a significantly higher correlation value than other webcams. Every correlation value written in the legend is calculated as the mean value of the vector pictured in the graph.

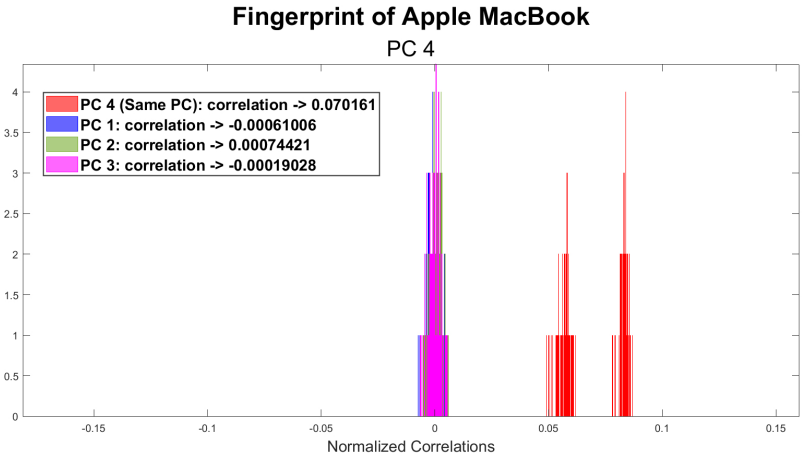


(b)



(c)

Figure 4.1: Fingerprint matching PC 2, (b), and PC 3, (c)



(d)

Figure 4.1: Fingerprint matching PC 4, (d)

4.3 ToothPic Algorithm

The algorithm developed by ToothPic is used for the camera fingerprint system in smartphones. In addition to extracting the fingerprint from the image sets, it performs fingerprint compression, as described in 2.3. Thus, it is possible to obtain both a fingerprint in double-precision values and a binarized fingerprint.

4.3.1 High-Pass Filter

The ToothPic algorithm applies a high-pass filter to the luminance of each photo. In this way, only the high frequencies of the PRNU are considered so to decrease the risk of a potential reconstruction of the fingerprint from JPEG images found by an attacker on the internet. However, the high-pass filter must be adapted to the information contained in the images taken by the PC webcams. It means that, since it is not possible to get a RAW image from the webcam, it is needed to check how many low frequencies can be cut in order to obtain a good fingerprint. The high-pass filter is designed in the following way:

1. The *Discrete Cosine Transform* (DCT) of the image that needs to be filtered is calculated.

2. The filtering matrix is built by means of a special parameter, *diagonal*, that can be manually modified. This matrix is of the same dimension as the selected image block. An example of this matrix is displayed in Figure 4.2.

3. The filter is applied by multiplying the DCT of the image and the filtering matrix cell by cell.

4. Finally, the inverse-DCT of the above output is calculated. This new matrix will be used in the pipeline of the fingerprint extraction.

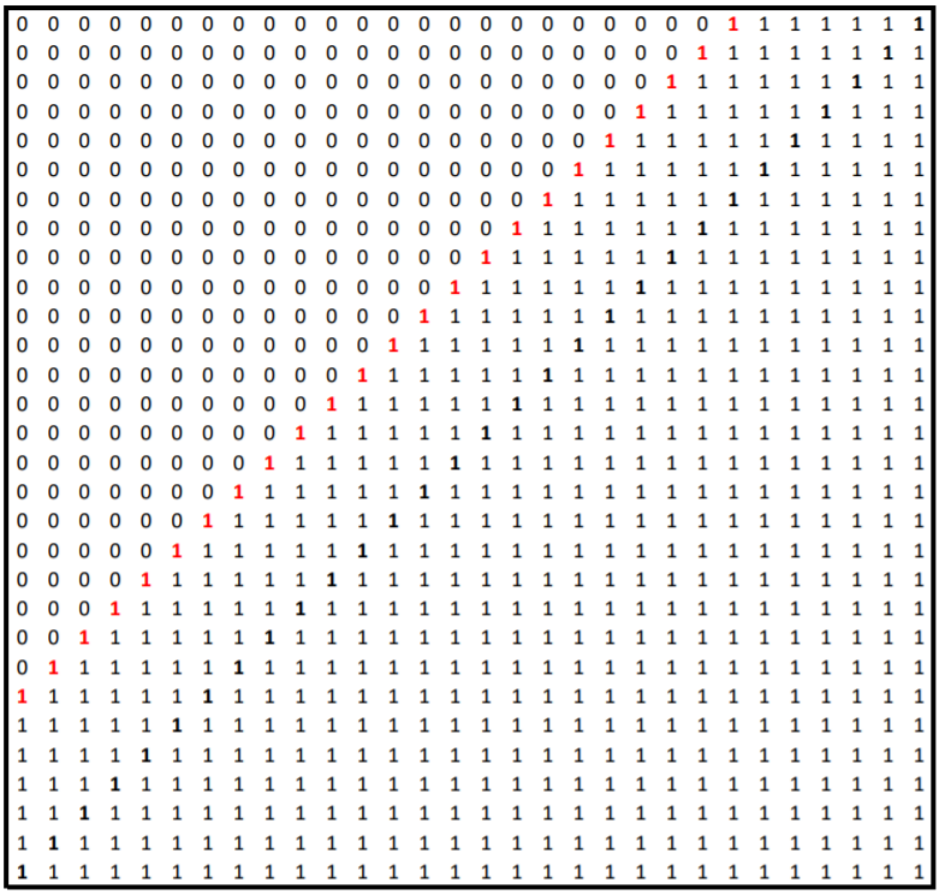


Figure 4.2: Example of the filtering matrix used in the high-pass filter

4.3.2 Images Size

Differently from the Fridrich algorithm, in this algorithm it is necessary to have a square image. This feature is required because of the random projections applied to compress the extracted fingerprint.

In the first dataset the minimum acquiring resolution is $640x480$, so the images must be cut to the square resolution $480x480$. This will produce two $480x480$ fingerprints with the following sizes:

- **Double-precision**: 1.843 *MB*
- **Compressed**: 28.8 *kB*

4.3.3 Results

As written in 4.3.1, the design of the high-pass filter is handled by the *diagonal* parameter. Defining the *blocksize* as the dimension of the photo edges after it has been cropped (in this case 480), the *diagonal* parameter can take the values between $-blocksize$, in which the high-pass filter will filter the whole image, and $+blocksize$, in which no value is filtered out. As example, using the filtering matrix of Figure 4.2, where the *diagonal* has been set to $blocksize * 0.2$, the 80% of the low frequencies, which stay in the upper-left corner of the image's DCT, would be cut off.

In order to find which *diagonal* parameter to choose for the following experiments, 4 different values were tested. The results in Table 4.2, Table 4.3, Table 4.4, and Table 4.5 represent the comparison of the reference fingerprints, in the horizontal rows, with each device's test-set for 4 possible *diagonal* values. Each cell contains two different values:

- The first one is the **correlation value** between the *double-valued* fingerprints.
- The second one is the percentage of the **Hamming distance** between the *compressed* fingerprints, defined in 2.4.2, on the total fingerprint length.

By analysing both the correlation and the Hamming distance between the fingerprints it is possible to see that, as the *diagonal* value increases, there is a better correlation and a lower Hamming distance. A good result for every PC is obtained when the *diagonal* is set to $blocksize * 0.3$ and $blocksize * 0.5$.

Table 4.2: Results of algorithm application with $diagonal = -blocksize * 0.3$

	PC 1 test	**PC 2 test**	**PC 3 test**	**PC 4 test**
PC 1 ref	**-0.0261** **49.54%**	−0.0030 50.02%	0.0004 50.01%	0.0014 50.17%
PC 2 ref	−0.0009 50.09%	**0.0833** **47.20%**	−0.0054 50.11%	0.0053 49.92%
PC 3 ref	0.0009 50.03%	0.0052 49.97%	**0.3008** **40.11%**	0.0004 50.07%
PC 4 ref	−0.0017 50.11%	−0.0003 49.83%	−0.0029 50.08%	**-0.0020** **49.94%**

Table 4.3: Results of algorithm application with $diagonal = blocksize * 0.1$

	PC 1 test	**PC 2 test**	**PC 3 test**	**PC 4 test**
PC 1 ref	**0.0479** **48.38%**	−0.0019 50.07%	0.0020 50.01%	0.0014 50.14%
PC 2 ref	0.0017 49.58%	**0.0931** **47.01%**	−0.0027 50.19%	0.0030 49.86%
PC 3 ref	0.0040 49.87%	0.0009 49.92%	**0.3219** **39.52%**	0.0028 49.97%
PC 4 ref	−0.0002 50.12%	0.0023 49.74%	−0.0041 50.25%	**0.0083** **49.59%**

Table 4.4: Results of algorithm application with $diagonal = blocksize * 0.3$

	PC 1 test	**PC 2 test**	**PC 3 test**	**PC 4 test**
PC 1 ref	**0.0386** **48.89%**	−0.0012 49.95%	0.0031 50.01%	0.0016 50.14%
PC 2 ref	0.0038 49.79%	**0.0940** **47.04%**	−0.0008 50.00%	0.0003 50.08%
PC 3 ref	0.0037 49.91%	−0.0012 50.10%	**0.3366** **39.09%**	0.0022 50.10%
PC 4 ref	0.0035 49.83%	0.0004 50.13%	−0.0004 50.12%	**0.0223** **49.23%**

Table 4.5: Results of algorithm application with $diagonal = blocksize * 0.5$

	PC 1 test	**PC 2 test**	**PC 3 test**	**PC 4 test**
PC 1 ref	**0.0319** **48.82%**	−0.0008 50.04%	0.0032 49.78%	0.0011 50.08%
PC 2 ref	0.0098 49.67%	**0.0888** **47.13%**	0.0004 49.96%	−0.0006 50.12%
PC 3 ref	0.0008 49.80%	−0.0021 50.14%	**0.3448** **38.86%**	0.0011 50.03%
PC 4 ref	0.0017 50.02%	0.0022 49.93%	0.0012 49.95%	**0.0379** **48.99%**

4.4 External USB Webcam

Despite the majority of the webcams nowadays are integrated in the PC, it is also possible to find external USB webcams. These webcams are widely adopted in desktop PCs and can be used even when the built-in webcam is broken.

Since the script is also capable of capturing photos from them, it can be useful to extract the fingerprint from these photos and analyse the results obtained. For this experiment it was used a *Keyteck USB External WebCam*, which was able to take photos with $640x480$ resolution.

4.4.1 Results

Applying the Fridrich algorithm to the images acquired with the USB webcam it is clear that the reference fingerprint has a high correlation with the test fingerprint of the same webcam, while the correlation with the other webcams is near to zero, as pictured in Figure 4.3.

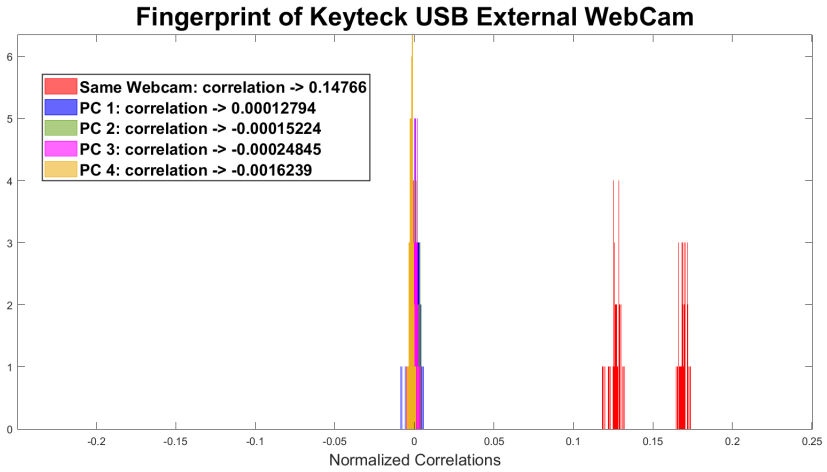


Figure 4.3: Fingerprint matching USB webcam, Fridrich algorithm

Regarding the application of the ToothPic algorithm, the results in Table 4.6, Table 4.7, Table 4.8, and Table 4.9 prove that the correlation and the Hamming distance between the reference fingerprint and the test fingerprint have a good discriminating value to match the right webcam. As seen above, also for this USB webcam the values become better with the increase of the *diagonal* parameter.

Table 4.6: Results, with $diagonal = -blocksize * 0.3$, of ToothPic algorithm application in the USB webcam

	USB test	**PC 1 test**	**PC 2 test**	**PC 3 test**	**PC 4 test**
USB ref	**0.1185** **46.35%**	0.0037 50.02%	0.0013 50.34%	0.0007 50.07%	−0.0009 50.08%

Table 4.7: Results, with $diagonal = blocksize * 0.1$, of ToothPic algorithm application in the USB webcam

	USB test	**PC 1 test**	**PC 2 test**	**PC 3 test**	**PC 4 test**
USB ref	**0.1608** **44.93%**	−0.0019 50.23%	0.0041 49.96%	0.0023 49.76%	−0.0017 50.25%

Table 4.8: Results, with $diagonal = blocksize * 0.3$, of ToothPic algorithm application in the USB webcam

	USB test	**PC 1 test**	**PC 2 test**	**PC 3 test**	**PC 4 test**
USB ref	**0.2008** **43.59%**	0.0001 50.07%	0.0034 50.04%	−0.0033 50.11%	0.0002 49.90%

Table 4.9: Results, with $diagonal = blocksize * 0.5$, of ToothPic algorithm application in the USB webcam

	USB test	**PC 1 test**	**PC 2 test**	**PC 3 test**	**PC 4 test**
USB ref	**0.2124** **43.22%**	−0.0037 49.97%	−0.0023 50.13%	−0.0038 50.21%	0.0006 49.76%

These outcomes demonstrate that even in a USB webcam scenario the webcam fingerprint system can be used with acceptable results. In any case, for authentication purposes, it is important to say that the fingerprint is only able to authenticate the webcam, so the user must always use the same webcam in order to perform the authentication.

4.5 Acquisition Campaigns

After all the required analysis on single photos and individual devices, it was necessary to extend the dataset to conduct more accurate studies. For this reason

two acquisition campaigns were conducted with the aim of building two different datasets. Both datasets, described in Table 4.10 and Table 4.11, are composed by 30 Windows and MacOS devices.

Table 4.10: Devices involved in the first acquisition campaign

Brand	**Resolution**	**N°**
Dell	$640x480$	8
Apple	$1280x720$	8
Asus	$640x480$	4
Lenovo	$640x480$	4
Hewlett-Packard	$640x480$	3
MSI	$640x480$	1
Huawei	$640x480$	1
Acer	$640x480$	1

Table 4.11: Devices involved in the second acquisition campaign

Brand	**Resolution**	**N°**
Dell	$1280x720$	15
Apple	$1280x720$	8
Lenovo	$1280x720$	3
Hewlett-Packard	$1280x720$	2
MSI	$1280x720$	1
Asus	$1280x720$	1

For the first acquisition campaign it was decided to capture the photos with the default resolution given by OpenCV. This choice was made because of two reasons:

- As defined in 3.3, if the resolution is forced it can be possible to obtain RGB photos reconstructed from a JPEG one from Windows devices. With the default resolution this situation should not happen.

- Not all Windows PCs are able to take photos with $1280x720$ resolution. Hence, with this solution, the webcam fingerprint system can involve more devices.

After analysing the results obtained with the data of the first campaign, explained in Chapter 5, it was decided to conduct a second acquisition campaign with new properties. This time the resolution was forced to $1280x720$ in order to obtain a bigger fingerprint which should improve the results.

However, as written above, not all Windows devices were able to acquire photos with the desired resolution, therefore the second dataset is slightly different from the first one.

Chapter 5

Final Results

In this chapter the results of the webcam fingerprint authentication mechanism are presented, taking into consideration the two large datasets created. The authentication process was performed following 3 different pipelines, one for each algorithm explained in the previous chapter plus a modified pipeline. Then the results, saved as correlation values and Hamming distances, were compared to each other in order to construct *ROC curves* by calculating the *true positive rate* and the *false positive rate*.

As will be shown later in this chapter, because of some peculiarities of the second dataset, unexpected results were retrieved from the Fridrich algorithm application. For this reason, after conducting more analyses on the data, a possible explanation of the detected anomaly was presented. In addition, the concept of *Wiener filtering*, a solution to the problem of the second dataset that improves the results, was introduced.

5.1 Analysis Process

The analysis process on the datasets was conducted in the following way. For every PC in the dataset the *reference fingerprint* was firstly extracted, then the correlation between the *reference fingerprint* and the *test fingerprint* of each PC was calculated. These results were saved into a *txt* file which was later used to create the ROC curves. This file starts with a string that contains all the information of the selected device (device ID, model, OS, and the algorithm used). Three different pipelines, described below, have been used to simulate the authentication system.

5.1.1 Fridrich Algorithm Pipeline

The first pipeline is based on the Fridrich algorithm explained in 4.2. After creating the string that identifies the device, the *double-valued reference fingerprint* is computed.

For every device in the dataset it was implemented the fingerprint matching process by calculating the correlation value for each test image and saving these values in a correlation array. The output values in the file, shown in Figure 5.1, represent the *mean* (first value) and the *standard deviation* (second value) of the correlation array. In the first line there are the correlation and the standard deviation of the same device that is under analysis, so the correlation of the first line is higher than the others.

```
PC_77 // Apple MacBook // MacOS 10.1 // Fridrich
0.19569 0.06127
-0.00028 0.00252
0.00023 0.00201
-0.00082 0.00226
-0.00020 0.00175
-0.00010 0.00197
0.00010 0.00214
-0.00075 0.00196
-0.00001 0.00143
```

Figure 5.1: Example of the *txt* file for the Fridrich algorithm pipeline

5.1.2 ToothPic Algorithm Pipeline

The second pipeline is based on the ToothPic algorithm explained in 4.3. As *diagonal*, it was decided to evaluate 4 different values, based on the outcomes obtained before, in order to see the different results:

- $\boldsymbol{diagonal} = 0.2 * blocksize$
- $\boldsymbol{diagonal} = 0.3 * blocksize$
- $\boldsymbol{diagonal} = 0.35 * blocksize$
- $\boldsymbol{diagonal} = 0.4 * blocksize$

Here the *txt* file structure is the same as the one presented above, except for the output values which reflect a difference in the pipeline.

In this pipeline there are the authentication processes with both *double-valued* fingerprints and *binarized* fingerprints. Therefore, the first value in the file, displayed in Figure 5.2, is the correlation between the *double-valued* fingerprints, and the second one is the percentage of the *Hamming distance* on the total length of the *binarized* fingerprint.

```
PC_405 // LENOVO 81EK // Windows 10.0.19043 // ToothPic 0.2
0.00964 49.49%
0.00022 50.01%
-0.00042 50.04%
-0.00101 49.84%
0.00158 49.86%
0.00102 50.06%
0.00056 50.09%
0.00179 49.77%
-0.00137 50.05%
```

Figure 5.2: Example of the *txt* file for the ToothPic algorithm pipeline

5.1.3 Modified Pipeline

The last pipeline is based on the union between the Fridrich algorithm and the ToothPic algorithm. Also in this case the *txt* file structure is the same.

The idea behind this modified pipeline is to apply the *random projections* on the fingerprint obtained by the Fridrich algorithm. Differently from the *binarized* fingerprint calculated with the ToothPic algorithm, this one will be extracted using the whole images information since the Fridrich algorithm does not apply any high-pass filter. In Figure 5.3 it is shown an example of the generated file which contains, for each line, the correlation value of the *double-valued* fingerprints and the Hamming distance between the *binarized* fingerprints.

```
PC_389 // HP Laptop 14s-dq0xxx // Windows 10.0.19043 // Fridrich Binarized
0.02901 46.66%
-0.00274 49.87%
-0.00273 49.78%
0.00045 50.03%
-0.00333 49.98%
0.00001 50.08%
0.00105 49.56%
-0.00084 49.60%
0.00015 49.68%
```

Figure 5.3: Example of the *txt* file for the modified pipeline

5.1.4 ROC Curves

When every *txt* file was obtained, data analysis was performed by means of *Receiver Operating Characteristic* (ROC) curves. These curves represent the accuracy of a binary classifier system at the variation of a threshold, which is in charge to discriminate whether the result is true or false.

For the purposes of this thesis it is necessary to evaluate the fingerprint matching process presented in 2.4. So, the correlation values and the Hamming distances previously calculated have been compared to the various thresholds. For each threshold value, 4 parameters have been determined:

- **True Positive (TP)**: The test fingerprint correctly matches the reference

fingerprint of the same device. In this case it would be authenticated the right device.

- **True Negative (TN)**: The test fingerprint does not match the reference fingerprint of a different device. In this case the wrong device will not be authenticated.

- **False Positive (FP)**: The test fingerprint matches the reference fingerprint of a different device. In this case it would be authenticated a wrong device.

- **False Negative (FN)**: The test fingerprint does not match the reference fingerprint of the same device. In this case the right device will not be authenticated.

The analysis of the ROC curves is based on the comparison of a pair of values obtained, for each examined threshold, from the parameters indicated above: *True Positive Rate* (TPR), and *False Positive Rate* (FPR).

The *True Positive Rate*, also called *sensitivity*, shows the probability that the device under analysis successfully passes the authentication process. The ideal *True Positive Rate* is 1.

$$\boldsymbol{TPR} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}}$$

The *False Positive Rate*, is the probability that the device under test passes the authentication process even if it should not. So, it represents the authentication rate of a wrong device, and it is dangerous. The ideal *False Positive Rate* is 0.

$$\boldsymbol{FPR} = \frac{\boldsymbol{FP}}{\boldsymbol{FP} + \boldsymbol{TN}}$$

5.2 First Dataset Results

Taking into consideration the photos acquired in the first acquisition campaign, as written in 4.1, the images were obtained using a default resolution. Consequently the minimum resolution obtained was $640x480$. Since the ToothPic algorithm requires a square crop to apply *random projections*, it was decided to cut every image to the resolution of $480x480$, obtaining the following sizes for the fingerprints:

- **Double-precision**: 1.843 *MB*

- **Compressed**: 28.8 *kB*

The ROC curve used as a reference is the one obtained from the results of the Fridrich algorithm because the images are not filtered, thus the fingerprints do not loose any information. For this reason, the following graphs present the comparison between the Fridrich ROC curve, both for *double-valued* and *binarized* fingerprints, and the curves obtained from the ToothPic algorithm.

In the successive graphs there are the results obtained from the full dataset, and the results obtained from the subset of Windows devices and the subset of MacOS devices.

5.2.1 ROC Curves Correlation Value

In these graphs, shown in Figure 5.4, the analysis of the *double-valued* fingerprints is illustrated. The ROC curve obtained from the Fridrich algorithm is compared to the ROC curves obtained from the ToothPic algorithm before the application of *random projections* to compress the fingerprints. So, the correlation value was used in order to perform the fingerprint matching process.

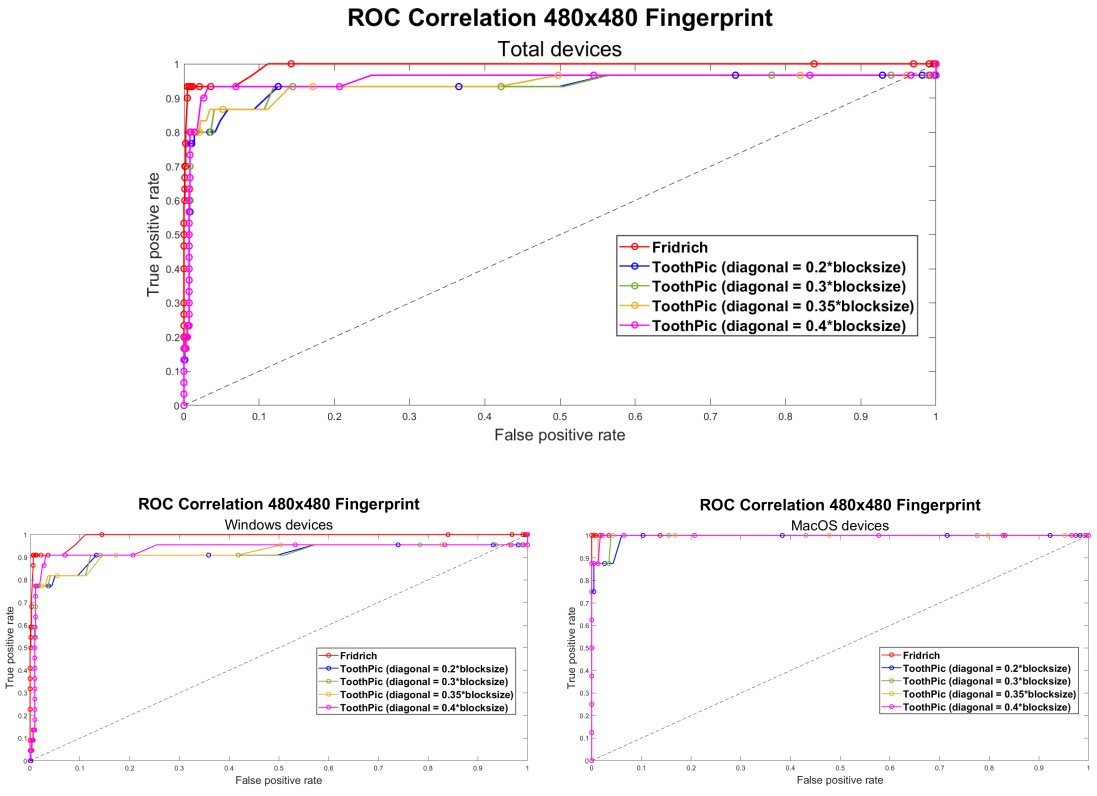


Figure 5.4: ROC curves of the first dataset results using the *double-valued* fingerprints

5.2.2 ROC Curves Hamming Distance

The graphs in Figure 5.5 display the comparison between the results obtained from the modified pipeline of the Fridrich algorithm, which are the *binarized* fingerprints, and the results obtained from the complete ToothPic algorithm pipeline, which includes fingerprint compression. It means that the decision parameter, used to determine the *TPR* and the *FPR*, was the Hamming distance.

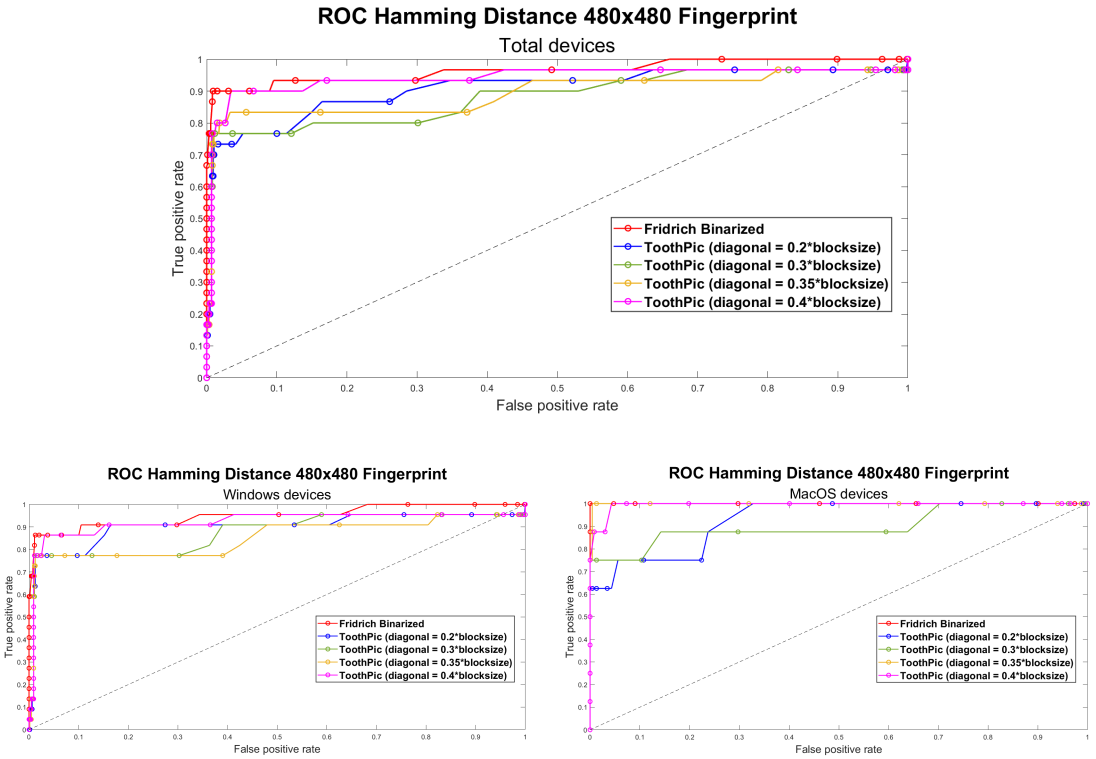


Figure 5.5: ROC curves of the first dataset results using the *binarized* fingerprints

5.2.3 Graphs Explanation

From the ROC curves describing the first dataset, it can be seen that the webcam fingerprint authentication process has overall acceptable results for the Fridrich algorithm. In any case, when the high-pass filter is applied within the ToothPic algorithm, the results seem to get worse when a large portion of the image is filtered (lower *diagonal* value).

This accuracy drop is more visible in the ROC curves of Figure 5.5. In fact, the *binarized* fingerprints carry an additional loss of the information due to the compression process, therefore the ROC curves, especially those obtained from the ToothPic algorithm, show low results.

However, an important thing to note is that there is a big difference between Windows and MacOS results. MacOS devices appear to have a more visible fingerprint than Windows ones. The explanation may be that OpenCV, using the AVFoundation back-end in MacOS environment, is able to get less processed data with respect to Windows environment, keeping more information about the PRNU.

5.3 Second Dataset Results

Given the above results, especially for the ROC curves obtained in the Windows devices, it was decided to conduct a second campaign with the aim of getting a higher-resolution fingerprint. In this way the fingerprint will have more information and the accuracy, shown in the ROC curves, should improve.

As explained in 4.5, the second dataset is composed only by devices capable of taking photos with $1280x720$ resolution. Thus, it was possible to cut the images to the square resolution, needed by the ToothPic algorithm, of $720x720$. With this new image crop the generated fingerprints are of the following sizes:

- **Double-precision**: 4.147 *MB*
- **Compressed**: 64.8 *kB*

While every MacOS device already present in the first dataset was able to get $1280x720$ resolution images, not all Windows devices had the possibility to do that, so the second dataset is composed by different Windows devices. An important characteristic of this new dataset, as seen in the following results, is the presence of many devices of the same model. In particular, it was possible to get the photos from:

- **8** *Dell Inc. Vostro 3591*
- **4** *Dell Inc. Latitude 5510*

5.3.1 ROC Curves Correlation Value

The following graphs, in Figure 5.6, represent the ROC curves of both the Fridrich and ToothPic algorithm considering the *double-valued* fingerprints, using the correlation value between the reference fingerprint and each test fingerprint as decision parameter.

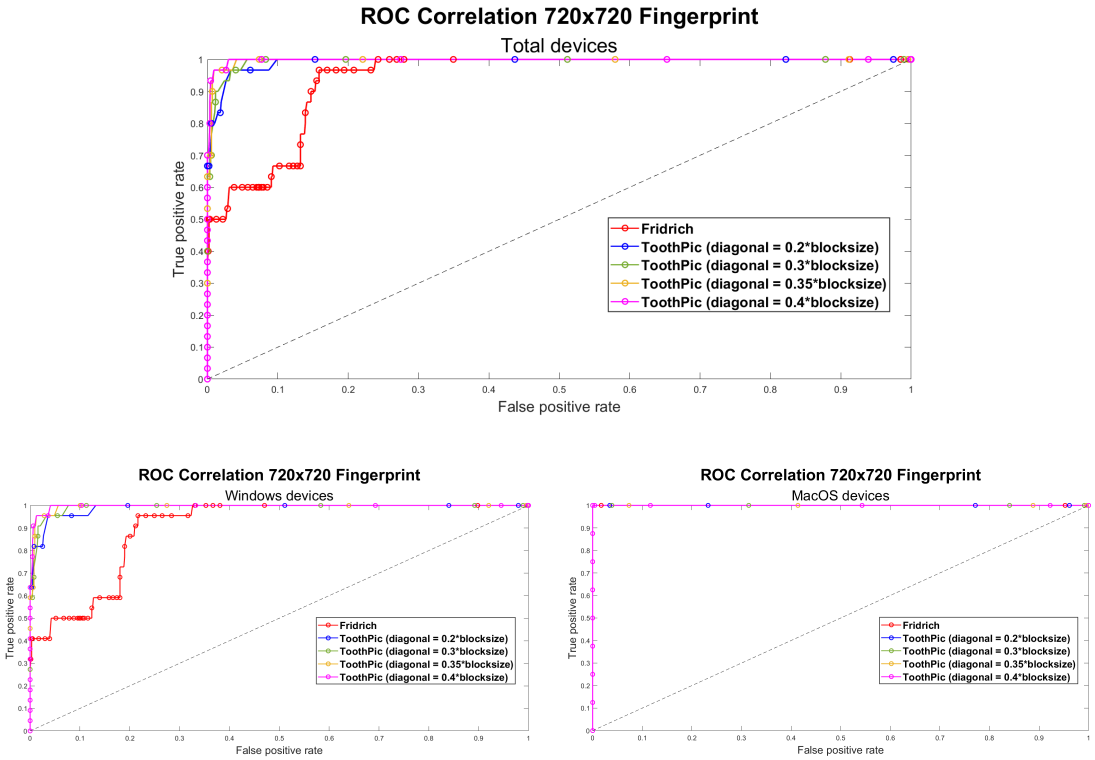


Figure 5.6: ROC curves of the second dataset results using the *double-valued* fingerprints

5.3.2 ROC Curves Hamming Distance

The graphs in Figure 5.7, picture the ROC curves obtained from the results of the modified Fridrich algorithm pipeline and the *binarized* fingerprints obtained from the complete ToothPic algorithm application. Hence, the decision parameter was the Hamming distance between the reference *binarized* fingerprint and each test *binarized* fingerprint.

5.3.3 Graphs Explanation

Evaluating the *double-valued* fingerprints extracted with the Fridrich and the ToothPic algorithms, in Figure 5.6, it can be seen an unexpected result. While the ROC curves obtained by applying the ToothPic algorithm give a better accuracy compared to the one with lower-resolution fingerprints, the outcomes of the Fridrich algorithm application are really poor. The expectation of the Fridrich algorithm, as demonstrated by the first dataset results, was to have the best accuracy among each obtained outcome because the high-pass filter was not applied to cut some

information from the photo. On the contrary, the ROC curve is even worse than the one obtained with the $480x480$ fingerprint.

Moreover, analysing the subsets of Windows and MacOS devices it is possible to see that the problem is present only in Windows devices since the ROC curves of MacOS devices are perfect for each pipeline used.

The *binarized* fingerprints results of Figure 5.7 are basically in line with the ones of the *double-valued* fingerprints. The Fridrich ROC curve still shows a bad accuracy, while the other ROC curves, especially with a higher *diagonal* value, have excellent accuracy for the fingerprint authentication scheme.

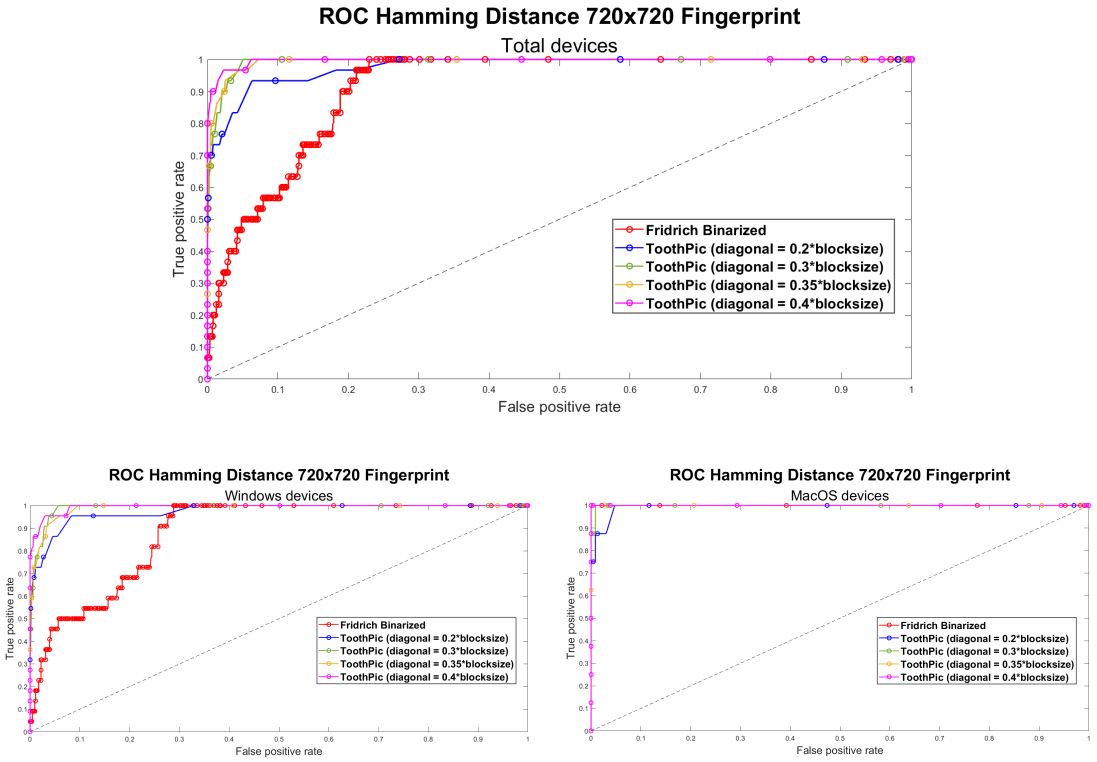


Figure 5.7: ROC curves of the second dataset results using the *binarized* fingerprints

Therefore, the performance drop of the Fridrich algorithm in Windows PCs may be due to the presence of many devices of the same model. Indeed, in some cases, the PRNU pattern noise may contain some *Non-Unique Artifacts* which are equal for PCs of the same model. These artifacts, as evident in the results, affect the fingerprint matching process generating many false positives and false negatives. However, the results of the ToothPic algorithm seem to eliminate this problem since the accuracy, as expected, is greater than the one obtained with

lower-resolution fingerprints.

In the following sections the problem of the Fridrich algorithm ROC curves will be examined in more detail, and a possible solution will be given together with the results of the corrected pipeline.

5.4 Non-Unique Artifacts

The ROC curves extracted from the second dataset show that the Fridrich algorithm, instead of generating the best accuracy, generates very negative results for both *double-valued* and *binarized* fingerprints.

This event could be caused by *Non-Unique Artifacts* (NUA). Basically, these artifacts are components that are systematically present in every photo (e.g., color interpolation artifacts, on-sensor signal transfer, and sensor design). The main feature of NUA is that they are not unique among different sensors, but devices of the same model, or even the same brand, shares them [22]. The consequence is that the PRNU is altered, and ambiguities may arise in the fingerprint matching process.

Taking into consideration the second dataset, it was stated in 4.5 that the dataset consists of multiple Windows devices of the same model and brand, unlike the first dataset which did not raised this problem. Instead, with regard to MacOS devices, it is possible to notice that the extracted PRNU does not contain any NUA as the ROC curves of the MacOS subset show almost perfect results.

Another interesting consideration concerns the application of the two different algorithms. While the Fridrich algorithm results present problems, the ROC curves of the second dataset display that the application of the ToothPic algorithm seems to delete NUA from the fingerprint because the results are very good. This event may be due to the high-pass filter application. It is indeed imaginable that the low-frequencies filtering of the images could potentially lead to NUA removal.

Table 5.1: List of devices in the dataset to detect NUA

Model	OS	Resolution	ID
Dell Inc. Vostro 3591	Windows 10.0.19042	$1280x720$	PC 1
Dell Inc. Vostro 3591	Windows 10.0.19042	$1280x720$	PC 2
Dell Inc. Latitude 5510	Windows 10.0.19041	$1280x720$	PC 3
MSI Prestige 14 Evo A11M	Windows 10.0.19043	$1280x720$	PC 4

5.4.1 Fingerprint Matching with NUA

In order to deeply analyse the NUA problem, a small dataset was chosen with the purpose of executing the fingerprint matching process with the Fridrich algorithm and seeing whether these NUA are evident in the plotted histograms. The devices of this experiment are listed in Table 5.1. The PCs were chosen so to have two devices of the same model, *Dell Inc. Vostro 3591*, a device of the same brand but of a different model, *Dell Inc. Latitude 5510*, and a device of a different brand, *MSI Prestige 14 Evo A11M*.

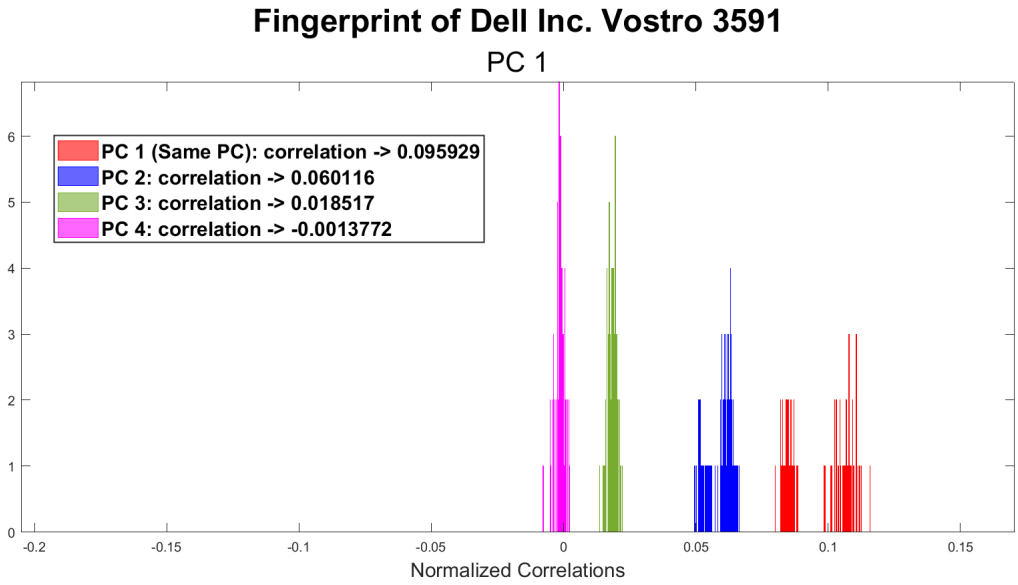


Figure 5.8: Histograms of the fingerprint matching process to detect NUA

In this experiment the reference fingerprint of PC 1 was extracted and, then, compared to the test fingerprints of each device in this small list to check the correlation values.

From the histograms of Figure 5.8 the expected results can be confirmed. The test fingerprint of the different *Dell Inc. Vostro 3591* device has a high correlation with the reference fingerprint because they have many NUA in common. The correlation of the *Dell Inc. Latitude 5510* is much lower compared to the other *Dell* devices, nevertheless it is still a high correlation because the histogram values are not zero. So, NUA can be a problem even in devices of the same brand and of a different model. Considering that the *MSI Prestige 14 Evo A11M* fingerprint is, as expected, completely uncorrelated with the analysed reference fingerprint, it is demonstrated that the fingerprint matching process works well in a scenario of two completely different PCs.

5.5 Wiener Filter

A solution that should solve the NUA problem is post-processing each image by applying the Wiener filter. As well-known in literature, Wiener filtering is able to suppress non-unique artifacts [4] so to improve the estimation of the PRNU in the fingerprint extraction process. In this kind of post-processing the Wiener filter operates in the frequency domain.

For this reason the Fridrich algorithm pipeline was modified with the addition of a function, which calculates the *Discrete Fourier Transform* and applies Wiener filtering, after the extraction of the fingerprint, thus removing any periodical pattern from the extracted PRNU.

Then, the fingerprint matching process was performed again with the devices of Table 5.1 to check whether the histograms of Figure 5.8 have been fixed using this new pipeline.

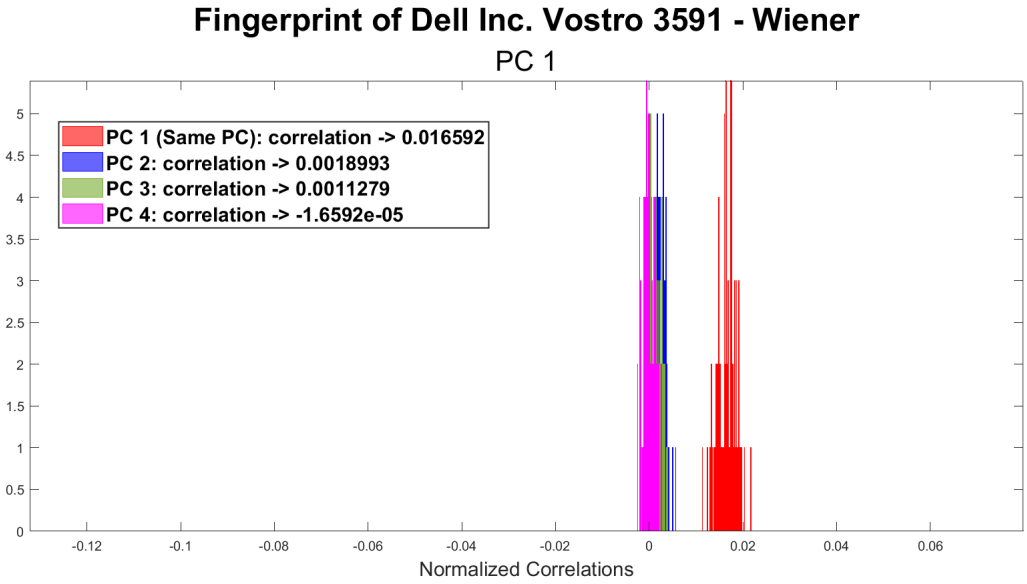


Figure 5.9: Histograms of the fingerprint matching process with the application of Wiener filtering

From Figure 5.9 it is possible to notice that, even if the correlation of the same PC is lower than the process without Wiener filtering, the correlations of the other devices, including those with the same model and brand, are zero. This means that NUA have been removed and the correlation of the same PC is given only by the extracted PRNU.

5.6 Wiener Filter Results

After analysing the problems caused by NUA and applying the Wiener filter, which seems to remove non-unique artifacts, it was decided to execute the Fridrich algorithm again for the whole second dataset. The purpose was to verify whether the promising findings displayed in 5.5 would be confirmed with a larger number of devices. This time, the Fridrich algorithm pipelines were modified, both the one that extracts the *double-valued* fingerprint and the one that extracts the *binarized* fingerprint, with the application of the Wiener filter. In the following images the ROC curves obtained from the Fridrich algorithm and the Wiener filtering are compared with the ROC curves obtained before from the ToothPic algorithm.

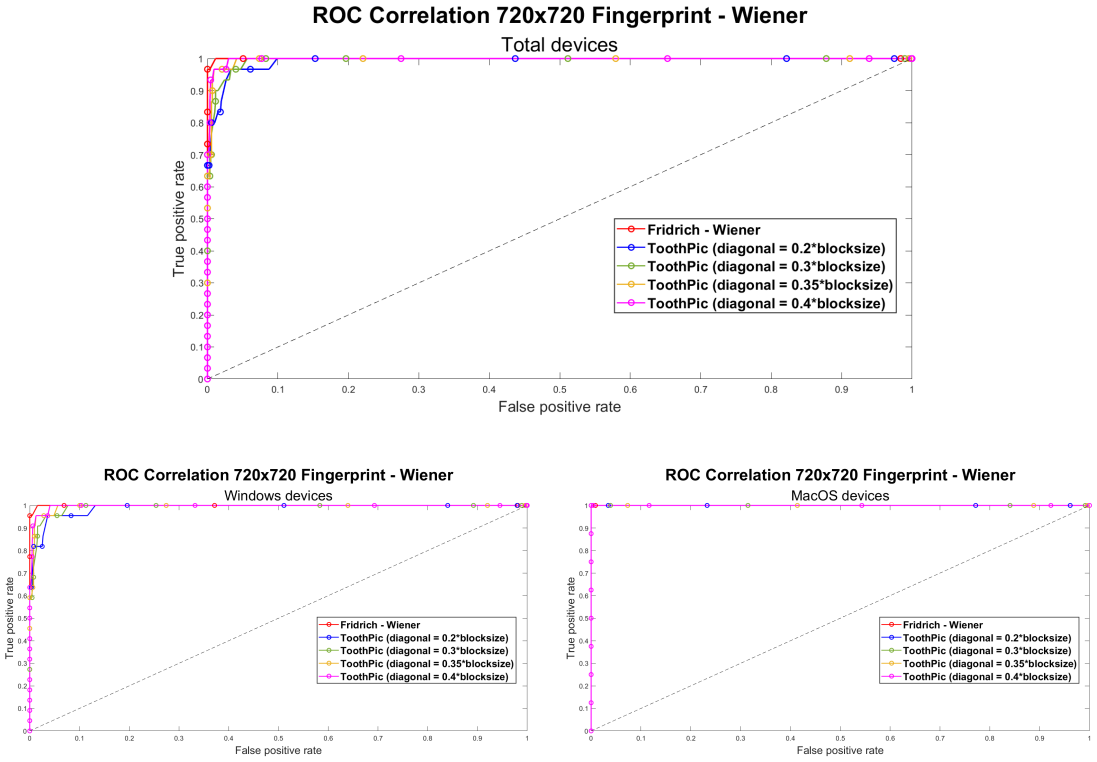


Figure 5.10: ROC curves of the second dataset results using the *double-valued* fingerprints. In the Fridrich algorithm pipeline Wiener filtering was applied

5.6.1 ROC Curves Correlation Value

The ROC curves of the fingerprint matching process with *double-valued* fingerprints in Figure 5.10 show that the Wiener filter has an excellent effectiveness in deleting NUA from photos, since the curve representing the Fridrich algorithm is the one

with the best level of accuracy, as expected at the beginning.

Furthermore, the application of the Wiener filter does not affect the perfect results obtained in the MacOS subset with the Fridrich algorithm because the extracted ROC curve is, once again, the best possible.

5.6.2 ROC Curves Hamming Distance

In Figure 5.11 the outcomes of the fingerprint matching process with *binarized* fingerprints are displayed. From the Fridrich ROC curve it can be noticed that, even with the application of the Wiener filter and the compression of the fingerprints, it is possible to obtain a great robustness in the fingerprint matching process.

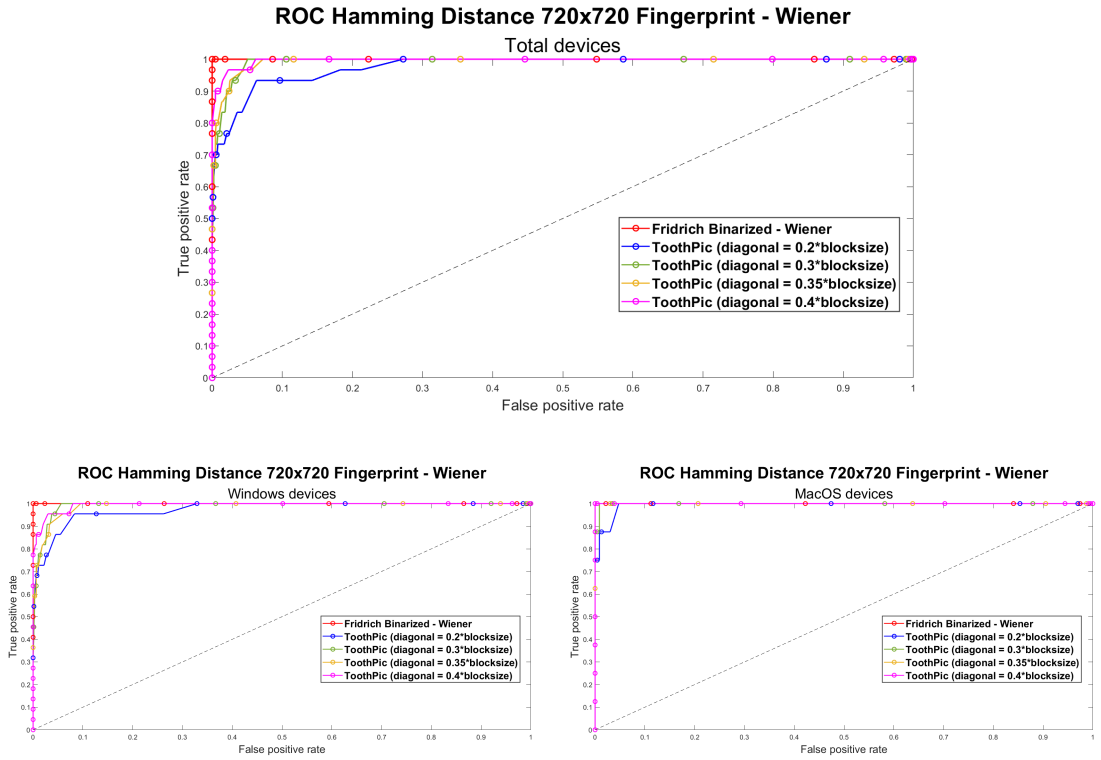


Figure 5.11: ROC curves of the second dataset results using the *binarized* fingerprints. In the Fridrich algorithm pipeline Wiener filtering was applied

Chapter 6

Conclusions

The webcam fingerprint, if extracted from photos taken in controlled situations, is confirmed as a valid mechanism from which a robust authentication scheme can be built. The results of these studies demonstrate that ROC curves can be obtained with the best accuracy from both Windows and MacOS PCs, even with the differences in the photo acquisition pipeline introduced by the two frameworks used by OpenCV. In particular, MacOS devices had very good outcomes from the beginning even with a smaller fingerprint, while Windows devices needed further actions in order to achieve the same accuracy level.

Another important issue solved, was the presence of *Non-Unique Artifacts* in the PRNU of the PCs of the same model. While the first dataset did not give the opportunity to see that, due to its composition, the second dataset made possible a deeper analysis of NUA presence in the fingerprint extracted from the acquired photos. This problem was managed with the application of Wiener filtering as a post-processing operation on the fingerprint. Therefore, the better overall results were retrieved using the higher-resolution fingerprints extracted from the second dataset. A possible weakness on designing the webcam fingerprint authentication system on $1280x720$ images is that not all Windows PCs are capable of capturing them, so these PCs cannot use the potential authenticator built from the findings of this thesis. The biggest improvement in accuracy achieved by a larger fingerprint is visible when dealing with *binarized* fingerprints. In fact, in a real-world scenario, the *double-valued* fingerprints will not be used because of the considerations in 2.3.

In conclusion, the last aspect to evaluate is which pipeline should be used to extract the reference fingerprints and perform the fingerprint matching process. For the ToothPic algorithm, the best ROC curve is the one obtained when the *diagonal* value has been set to $blocksize * 0.4$. This means that the high-pass filter cuts the 60% of the low frequencies that stay above the principal diagonal of each image's *Discrete Cosine Transform*. Instead, the *binarized* fingerprint obtained from the Fridrich algorithm, after the application of the Wiener filter, has a perfect accuracy level. However, there is no high-pass filter in the pipeline to extract this fingerprint, so the full image is considered. This could lead to a potential problem if many images captured by the PC webcam can be found on the Internet. In any case, the probability to upload such images online is very low, differently from photos taken with a smartphone camera. Thus, the choice of which pipeline to use for future work is only technological, considering the pros and cons of these two possibilities.

6.1 Future Work

Webcam fingerprint authentication may represent a considerable upgrade to solve the usability problem of *Multi-Factor Authentication*. So, starting from the studies

of this thesis, an authenticator which implements these pipelines in a real-world scenario can be built. The authenticator can be a good support for MFA systems such as *2-Factor Authentication*, which always requires an additional authentication factor in addition to passwords, or *Risk-Based Authentication*, a newer MFA system which assesses the risk of every login attempt and requires an additional authentication factor when the risk is high [23]. It might also be possible, in some controlled situations where no critical systems are involved, to use webcam fingerprint as *Single-Factor Authentication*.

Another imaginable future work could be the implementation of this system for Linux devices. In this thesis they were not considered because it was first decided to analyse the most used operating systems, Windows and MacOS. As the obtained results demonstrate the reliability of this mechanism, the study can be expanded to Linux devices. This is also favoured by the usage of OpenCV, since it is available in Linux environments.

Bibliography

[1] National Institute of Standards and Technology (NIST). *Authentication Factor*. 2021. URL: https://csrc.nist.gov/glossary/term/authentication_factor (visited on 11/23/2021).

[2] Grzergor Milka. *Anatomy of Account Takeover. In Enigma 2018 (Santa Clara, CA). USENIX Association*. 2018. URL: https://www.usenix.org/node/208154 (visited on 11/23/2021).

[3] EUR-Lex. *General Data Protection Regulation*. 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN (visited on 10/26/2021).

[4] Diego Valsesia, Giulio Coluccia, Tiziano Bianchi, and Enrico Magli. «User authentication via PRNU-based physical unclonable functions». In: *IEEE Transactions on Information Forensics and Security* 12.8 (2017), pp. 1941–1956.

[5] Jessica Fridrich. «Digital image forensics». In: *IEEE Signal Processing Magazine* 26.2 (2009), pp. 26–37.

[6] Jan Lukas, Jessica Fridrich, and Miroslav Goljan. «Determining digital image origin using sensor imperfections». In: *Image and Video Communications and Processing 2005*. Vol. 5685. International Society for Optics and Photonics. 2005, pp. 249–260.

[7] Diego Valsesia, Giulio Coluccia, Tiziano Bianchi, and Enrico Magli. «Compressed fingerprint matching and camera identification via random projections». In: *IEEE Transactions on Information Forensics and Security* 10.7 (2015), pp. 1472–1485.

[8] ToothPic S.r.l. *ToothPic*. 2020. URL: https://www.toothpic.eu/ (visited on 10/25/2021).

[9] FIDO Alliance. *FIDO2*. 2021. URL: https://fidoalliance.org/fido2/ (visited on 10/26/2021).

[10] Apple Inc. *AVFoundation*. 2020. URL: https://developer.apple.com/documentation/avfoundation (visited on 10/29/2021).

[11] Microsoft Corporation. *DirectShow*. 2021. URL: https://docs.microsoft.com/en-us/windows/win32/directshow/directshow (visited on 10/29/2021).

[12] Microsoft Corporation. *Microsoft Media Foundation*. 2021. URL: https://docs.microsoft.com/en-us/windows/win32/medfound/microsoft-media-foundation-sdk (visited on 10/29/2021).

[13] OpenCV team. *OpenCV*. 2021. URL: https://opencv.org/about/ (visited on 10/29/2021).

[14] FFMPEG. *libav*. 2021. URL: https://trac.ffmpeg.org/wiki/Using%20libav* (visited on 10/29/2021).

[15] FFMPEG. *FFMPEG*. 2021. URL: https://ffmpeg.org/about.html (visited on 10/29/2021).

[16] scikit-image development team. *Scikit-image*. 2021. URL: https://scikit-image.org/ (visited on 10/29/2021).

[17] Inc. Sight Machine. *SimpleCV*. 2021. URL: http://simplecv.org/ (visited on 10/29/2021).

[18] Inc. GitHub. *OpenCV Python*. 2021. URL: https://github.com/opencv/opencv-python (visited on 10/29/2021).

[19] Gregory K Wallace. «The JPEG still picture compression standard». In: *IEEE transactions on consumer electronics* 38.1 (1992), pp. xviii–xxxiv.

[20] Inc. Dropbox. *Dropbox Developers Documentation*. 2021. URL: https://www.dropbox.com/developers/documentation (visited on 11/04/2021).

[21] Inc. Dropbox. *Dropbox for Python Documentation*. 2019. URL: https://dropbox-sdk-python.readthedocs.io/en/latest/ (visited on 11/04/2021).

[22] Mo Chen, Jessica Fridrich, Miroslav Goljan, and Jan Lukás. «Determining image origin and integrity using sensor noise». In: *IEEE Transactions on information forensics and security* 3.1 (2008), pp. 74–90.

[23] Stephan Wiefling, Markus Dürmuth, and Luigi Lo Iacono. «More Than Just Good Passwords? A Study on Usability and Security Perceptions of Risk-based Authentication». In: *Annual Computer Security Applications Conference*. 2020, pp. 203–218.