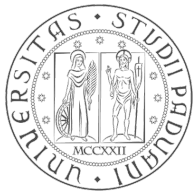


Università Degli Studi Di Padova

Department of Information Engineering

Master Degree in Computer Engineering

AcCAPPCHA:
Design, Development and Security Analysis of an Invisible CAPPCHA based on an acoustic side-channel

Candidate
Di Nardo Di Maio Raffaele

Supervisor
Prof. Migliardi Mauro

DD-MM-2021

Accademic Year 2020-2021

To my parents, that always help
me to be happy doing what I love
and support me reaching my goals.

“*Most people assume that once security software is installed, they're protected. This isn't the case. It's critical that companies be proactive in thinking about security on a long-term basis.*”
Kevin Mitnick

“*You have to learn the rules of the game. And then you have to play better than anyone else.*”
Albert Einstein

“*Si come il ferro s'arrugginisce sanza esercizio, e l'acqua si putrefà o nel freddo s'addiaccia, così lo 'ngegno sanza esercizio si guasta.*”
Leonardo da Vinci

Abstract

Over the years, web services have become part of the daily routine of everyone and a lot of private users'information is uploaded on the network. For this reason, the development of malwares has also grown and is now often endowed with advanced AI techniques. For many years, the CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) has been the protection mechanism against these dangerous programs. It is based on some tasks that the user should complete to be classified as a human. However the bots can also elude it by exploiting machine learning and computer vision. Invisible CAPPCHA overcomes this approach by analysing the physical nature of the user during the authentication web service. The same idea has been used to create AcCAPPCHA (Acoustic CAPPCHA) based on a microphone side-channel. The design of the scheme follows the same strategy of Invisible CAPPCHA but it also introduces deep learning techniques during the classification of the user's activity. AcCAPPCHA gives support to the authentication phase and it has been tested with human users and simple bot programs. The new type of CAPPCHA opens new horizons for a future implementation on mobile devices and voice assistants.

Sommario

Con il passare degli anni, i servizi web sono entrati a far parte della routine quotidiana di chiunque e molte informazioni private degli utenti sono caricate sulla rete. Per questo motivo, lo sviluppo di malware è anche cresciuto ed ora è spesso dotato di tecniche avanzate di Intelligenza Artificiale. Per molti anni, il CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) è stato il meccanismo di protezione contro questi programmi pericolosi. Esso è basato su alcune attività che l'utente dovrebbe completare per essere classificato come un umano. Comunque i bot riescono ad eluderlo sfruttando machine learning e computer vision. Invisible CAPPCHA supera questo approccio analizzando la natura fisica dell'utente durate il servizio web di autenticazione. La stessa idea è stata utilizzata per realizzare AcCAPPCHA (Acoustic CAPPCHA) basato sul side-channel del microfono. La progettazione dello schema segue la stessa strategia di Invisible CAPPCHA ma introduce anche tecniche di deep learning durante la classificazione dell'attività dell'utente. AcCAPPCHA dà supporto alla fase di autenticazione ed è stato testato con utenti umani e semplici bot. Il nuovo tipo di CAPPCHA apre nuovi orizzonti per una futura implementazione su dispositivi mobili e assistenti vocali.

Contents

Chapter 1

Introduction

CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) is a program used to distinguish human users from bots. A bot is a malicious application that automates a task and may be used to gather useful information about user credentials or pretending to do a human interaction with a Web application. In fact the term *"bot"* is an abbreviation of the words "software robot".
The CAPTCHAs are traditionally used in Web applications[1]:

- **To protect online Polls**
  CAPTCHAs prevent the creation and the submission of a large number of votes, favouring a party.
- **To protect Web Registration**
  CAPTCHAs prevent the creation of free mail account for bots instead of human users. The aim of CAPTCHAs is to remove the possibility that the hacker could take advantages from a large amount of registrations.
- **To prevent comment spam**
  CAPTCHAs prevent the insertion of many posts made by a bot on pages of social platforms or blogs.
- **To stop search engine bots**
  CAPTCHAs are used to guarantee that a website would be unindexed and to prevent search engine bots from reading a page. The CAPTCHAs are added because the html tag, used to unindex the web page, doesn't guarantee unindexing.
- **To protect E-Ticketing**
  CAPTCHAs prevent ticket hoarding by resellers so that a big event won't sell out minutes after the tickets become available. In fact

CAPTCHAs reduces the number of scalpers that make many ticket purchases and sell them with higher prices.

- **To prevent email spam**
  CAPTCHAs are used to verify that a human has sent an email.

- **To prevent Dictionary Attacks**
  CAPTCHAs prevent bot from guessing the password of a specific user. The hacker could guess the password, taking it from a dictionary. The use of the CAPTCHA challenge prevents the iteration of the login phase made by the bot using all the words of the dictionary. After a certain number of failures POST requests, the CAPTCHA challenge is shown to the user.

- **To verify digitized books**
  ReCAPTCHA can verify the contents of a scanned piece of paper analysing responses in CAPTCHA fields. A computer can't identify all the words from a digital scan.
  The application submits two words to the user during the CAPTCHA challenge: the first one that the machine has already recognized and the other one for which the application wants to associate a word. If the user types the two words and the first one was correctly detected, it assumes that also the second one is correct.
  In this case the second word is added to a set of words that are going to be choose to create challenges for other users. If the application receives enough responses with the same typed word related to an unknown word, the program establishes that that label would be associated to it. Hence the CAPTCHA scans digitally the paper during the verification of the user identity.

However, over the years, these challenges have become more and more complex because a lot of them easily fail against new machine learning techniques (**Chapter 2**).
Many attacks exploits data, collected by built-in sensors of the victim's machine, to obtain some credentials or useful information (**Chapter 3**). However, at the same time, the very same sensors were also leveraged to increase the strength of CAPTCHAs against bot attacks.
CAPTCHAs are also very used during the authentication process. For example, Invisible CAPPCHA is a new CAPTCHA scheme, developed in 2018, that exploits the physical information obtained by motion sensors of the smartphone[35]. Using this CAPTCHA, the user simply authenticates himself and the application analyses in background if this phase was performed

by a human (**Chapter 4**).
From the ideas developed in Invisible CAPPCHA and leveraging an acoustic side-channel, I designed AcCAPPCHA (**Chapter 5**) to support the authentication from the desktop and the laptop environments.
The pros and the cons of the new invisible challenge, observed during the testing phase, have been described (**Chapter 5**) and some ideas of future improvements are analysed in (**Chapter 7**).
In the appendices of the current work, you can also find the the key map used by AcCAPPCHA (**Appendix A**) and the details about its command line arguments (**Appendix B**).

Chapter 2

State of the Art

CAPTCHA is the acronym of *"Completely Automated Public Turing-test-to-tell Computers and Humans Apart"* and it's a challenge born from these two components[2]:

1. **Human-Computer Interaction (HCI)**
   it is a research field that studies the computer technology, looking at the interfaces between the users and the machines. A human can interact with a machine at several levels:

   - Task level
   - Semantic level
   - Syntactic level
   - Interactive level
   - Physical devices level

   This discipline tries to analyse the ways in which a human can process data, matching the user's needs and interests. According to cognitive psychology studies, the human interaction with the machine is based on:

   - Reasoning
   - Problem solving
   - Skill acquisition
   - Error

2. **Human Interactive Proof (HIP)**
   it's used to classify machines and humans looking at a type of interaction, that is simple to be done by a human instead of a bot. The main goals of the test are:

- To differentiate the humans from the computers
- To differentiate a category of the humans
- To differentiate a specific human from the category of humans

HIP consists of a test program administered to humans and computers. The main problem of this proof is that the complexity of the task is very complex and only a specific group of humans can usually solve the test.
Historically the most widely known version is the cognitive one defined by Alan Turing. The Turing test is used to determine how much a machine is capable of thinking like a human. The experiment involves three figures: a human examiner, a human and a machine. The examiner asks some questions to the other figures and, after a fixed amount of time, he evaluates if the two answers were different or not.
If they were similar, with respect to the point of view of the examiner, the machine is classified as an AI (Artificial Intelligence). The test is very reliable if each question could have many possibilities.

The design of CAPTCHA challenges takes inspiration from the previous components because its goal is to establish if the user is a bot or a human. A bot is usually an advanced program that exploits some AI techniques and pretends to be a human. In the endeavor of foiling bots that are getting every day smarter and more sophisticated, a CAPTCHA challenge sometimes is so complex that a human user can't solve it. For this reason and in order to guarantee an high level of security, a CAPTCHA has to satisfy the following requirements:

- There exists only one solution of a CAPTCHA and it shouldn't depend on the user's language and/or age;
- The solution of a CAPTCHA must be easy for the humans and hard for the bots. In fact humans should solve it in no longer than 30 seconds with very high success rate (30 seconds are usually considered too much time to solve a CAPTCHA);
- The creation of a CAPTCHA shouldn't affect the user privacy.

Over the year, the CAPTCHAs must satisfy more and more requirements and the challenges become more and more complex reducing the usability of them. For this reason, a new generation of CAPTCHAs, that we define modern CAPTCHAs, has been developed in the last years. The new challenges are related to a good exploitation of the biometrics and the sensors of

modern devices. In addition, machine learning techniques and side-channel information are used to evaluate the user behaviour instead of generating challenges based on cognitive capabilities.

2.1 Traditional CAPTCHAs

The traditional CAPTCHAs are based on the knowledge and the correct insertion of the solution by the user. These CAPTCHA schemes are designed to exploit character recognition, image analysis and speech recognition to guarantee that the challenges will successfully block bots.
The main types of traditional CAPTCHAs, namely Audio-based CAPTCHAs, Game-based CAPTCHAs, Image-based CAPTCHAs, Math CAPTCHAs, Slider CAPTCHAs, Text-based CAPTCHAs and Video-based CAPTCHAs, are described in the following sections but the details about specific implementations can be found in the article of Walid Khalifa Abdullah Hasan[3].
With respect to user experience, the most enjoyable traditional CAPTCHAs are usually the game-based and image-based ones while the most frustrating CAPTCHA is the text-based one[4]. A summary of usability and security issues, explained in the following sections, is also reported in **Table 2.1**.

2.1.1 Audio-based CAPTCHAs

This type of CAPTCHAs asks the user to type the words contained in an audio file (see **Figure 2.1**). It's developed for vision-impaired users. Audio-based CAPTCHAs usually have problems in usability related to the language dictionary, from which the words are taken, and the similarity of sound for several words. This CAPTCHA is a hard task even for blind users, in fact during an experiment only 46% of the challenges were solved by the participants[5].
One of the most popular Audio-Based CAPTCHAs is *Audio reCAPTCHA*, developed at Carnegie Mellon University and then bought by Google. In this scheme, the user needs to recognize and write a set of 8 spoken characters from a noisy audio file with several background voices. If the user makes a mistake, the test declares that he's a bot.
Audio-based CAPTCHAs are vulnerable to many Automatic Speech Recognition (ASR) programs[6] but also Deep Learning techniques (e.g. DeepCRACk[7]). A good overview about results, obtained by several classification methods, is described in the work of Jennifer Tam et al.[8].

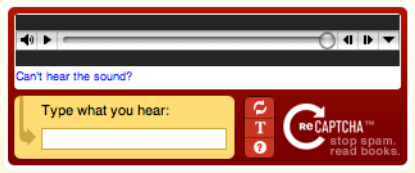


Figure 2.1: Example of audio-based CAPTCHA.

2.1.2 Game-based CAPTCHAs

This type of CAPTCHAs performs the verification of the user identity through a set of several kinds of games (see **Figure 2.2**). The strength of Game-based CAPTCHAs is related to the the rules of the game that only humans can understand.
There exists an implementation of these CAPTCHAs, called *Dynamic Cognitive Game (DCG)*, that is usually developed using Flash, HTML5 and JavaScript. These technologies download the game code to the client and execute it locally.
The code of these challenges must be the obfuscated to prevent that the source file could be stored on different internet domains. However for example, there exists a bot attack, called *Stream Relay Attack*, that obtains good results bypassing these challenges [9] (see **Section 2.3**).

Figure 2.2: Examples of game-based CAPTCHAs.

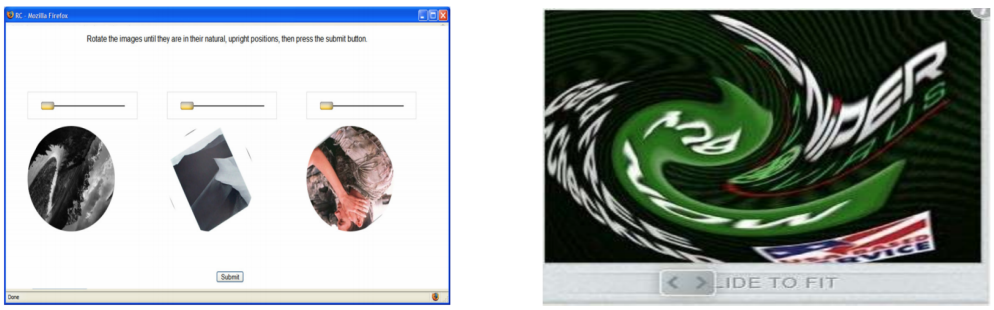
2.1.3 Image-based CAPTCHAs

Image-based CAPTCHAs is based on a written text that describes a task related to the evaluation of some images. The user must understand the action and he passes the test if he performs it in the right way. This type of CAPTCHAs can be categorized into the following classes, looking at the task that the user needs to perform:

- **Click-based CAPTCHAs**
  this type of CAPTCHAs shows an image and a text that explains where the user needs to click (see **Figure 2.3**).

Figure 2.3: Example of click-based CAPTCHA.

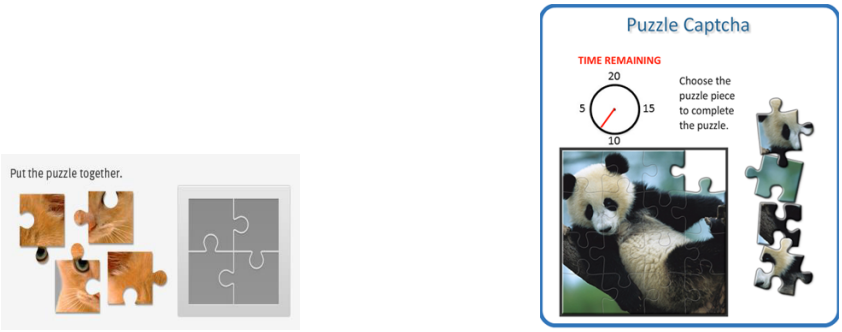
- **Sliding image-based CAPTCHAs**
  this type of CAPTCHAs asks the user to use the slider to solve an image-based challenge such as adjusting the orientation of an image, selecting the correct form of an image, or moving a fragment of an image to the correct location (see **Figure 2.4**).



(a) Orientation based. (b) Form based.

Figure 2.4: Examples of sliding image-based CAPTCHAs.

- **Drag & Drop-based CAPTCHAs**
  this type of CAPTCHAs usually asks the user to complete a visual puzzle, created by dividing a given image in a set of pieces[10] (see **Figure 2.5a**).
  The task isn't easy for users because this type of CAPTCHAs takes more time to solve the puzzle but the security level is very high[10]. To improve the usability of the CAPTCHA, there exists a version of the puzzle-based CAPTCHA in which the user needs to insert only some pieces of the puzzle instead to complete it (see **Figure 2.5b**).



(a) Completing the puzzle. (b) Inserting only some pieces.

Figure 2.5: Examples of puzzle-based CAPTCHAs.

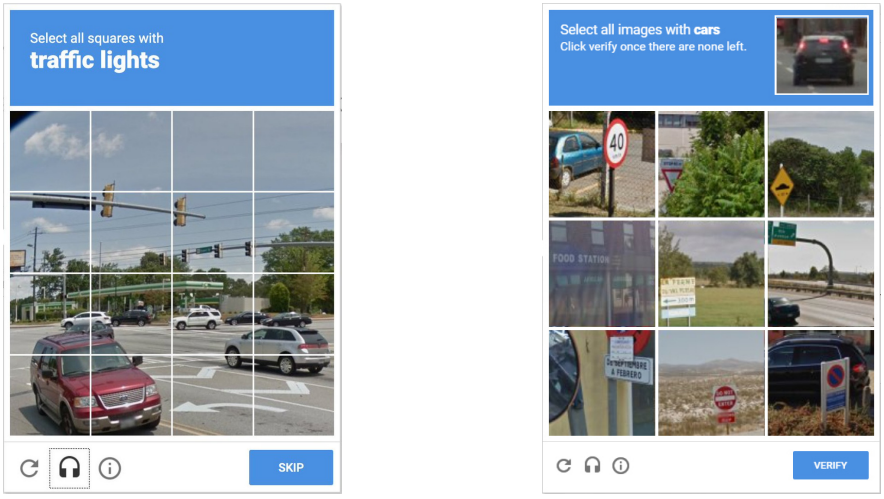
- **Selection-based CAPTCHAs**
  the user usually needs to select the images that contain a requested subject. The set of images, on which the user needs to identify the subject, can be created in several ways, for example:

  – An image is divided into a set of sub-squares and each of them is a candidate image (see **Figure 2.6a**)

  – There are many images, each one with a unique different subject (see **Figure 2.6b**)

  This type of CAPTCHAs is vulnerable to different Object Recognition techniques developed for Computer Vision purposes.
  An extension of this type of CAPTCHAs, called *FaceDCAPTCHA*, has been introduced[11] and it incorporates elements of face detection. The human brain is very effective in the process of natural face segmentation even if there are complex backgrounds. In fact, this approach increases the security efficiency because the Computer Vision programs can easily detect if there is a face, e.g. Viola-Jones algorithm[12], but they have

many problems understanding if the photographs of the faces are real or not.
One of the most difficult challenges, to be performed by a computer, is the detection of faces, fingerprints and eyes in an image. For this reason the previous idea has been used to to develop a new version of image-based CAPTCHA, called *MB CAPTCHA*[13].



(a) With an image divided in sub-squares. (b) With several images.

Figure 2.6: Examples of selection-based CAPTCHAs.

- **Interactive-based CAPTCHAs**
  the user needs to discover a secret position in an image using mouse movements or swiping gestures

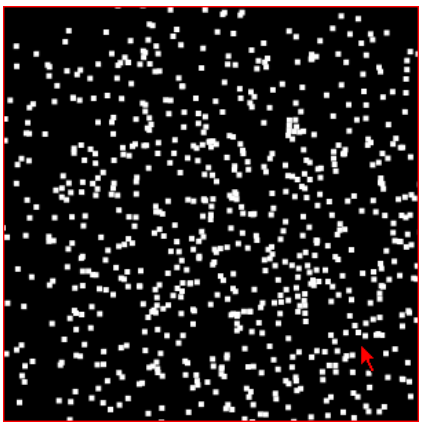
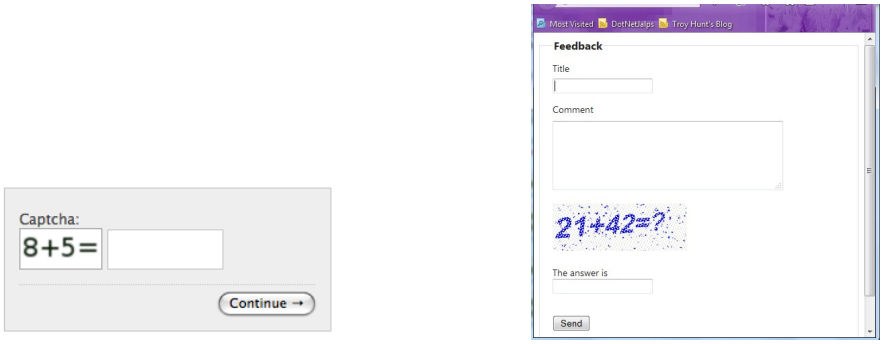


Figure 2.7: Example of interactive-based CAPTCHA.

2.1.4 Math CAPTCHAs

Looking to a math operation, specified in a frame, the user needs to insert the result in a text field. The operation is written in plain text or, to improve the security of this challenge, it's warped like text-based CAPTCHAs (see **Figure 2.8**). These classical math-CAPTCHAs, also known as *arithmetic CAPTCHAs*, are vulnerable to OCR (Optical Character Recognition) techniques.

An advanced version of this CAPTCHA is used in the Quantum Random



(a) With plain text. (b) With wrapped text.

Figure 2.8: Example of arithmetic CAPTCHAs.

Bit Generator Service (QRBGS) sign-up Web Page[14] (see **Figure 2.9**). This type of CAPTCHA asks the user to solve an advanced math expression. It prevents the use of free or commercial OCR techniques because many mathematical symbols are not considered in the traditional detection algorithms. Hence many math symbols are wrongly translated by bot programs and the challenge is very secure. However, this CAPTCHA is vulnerable to side-channel attacks [14] and it's very complex for normal users. In fact many people can't solve it correctly because the required level of math knowledge is very high and not very common.

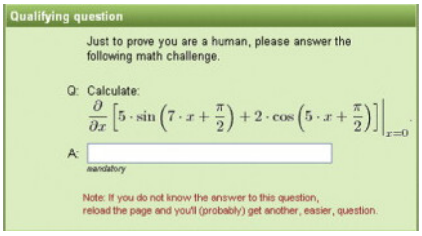


Figure 2.9: Example of Quantum Random Bit Generator Service (QRBGS) sign-up Web Page [14].

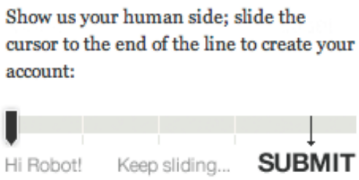
2.1.5 Slider CAPTCHAs

Slider CAPTCHAs ask users to move a slider across the screen. The image recognition is not part of the challenge.
The most popular CAPTCHAs of this type are the following:

- **CAPTCHA used by Taobao.com**
  it asks the user to drag the slider from the start to the end of the sliding bar to verify his identity (see **Figure 2.10a**).
- **CAPTCHA used by TheyMakeApps.com**
  it asks the user to move the slider to the end of the line to submit a form[15] (see **Figure 2.10b**).

Over the years several implementations of this type of CAPTCHAs have been bypassed with a simple JavaScript code and puppeteer.

(a) Example of TaoBao.com CAPTCHA.



(b) Example of TheyMakeApps.com CAPTCHA.

Figure 2.10: Examples of slider CAPTCHAs.

2.1.6 Text-based CAPTCHAs

In text-based CAPTCHA schemes a random series of warped characters and/or numbers is displayed on the screen inside an image (see **Figure 2.11**). The user needs to understand which are the characters that composes the sequence and then he types them inside a text-field. The most known classes of text-based CAPTCHAs are:

- **2D**
  the warped characters of the text are placed and oriented in a 2D plane, parallel to the screen plane. The image of the 2D plane is used in the challenge.
- **3D**
  each warped character of the text is mapped into a 3D model of the character and then is placed and oriented in a 3D plane in the space.

A 2D image of the 3D space is taken from a specific point of view and it's used in the challenge.

This type of CAPTCHAs is vulnerable to several attacks, related to Computer Vision techniques:

- OCR techniques[16]
- Segmentation techniques (e.g. DECAPTCHA[17])
- Machine Learning and Deep Learning techniques

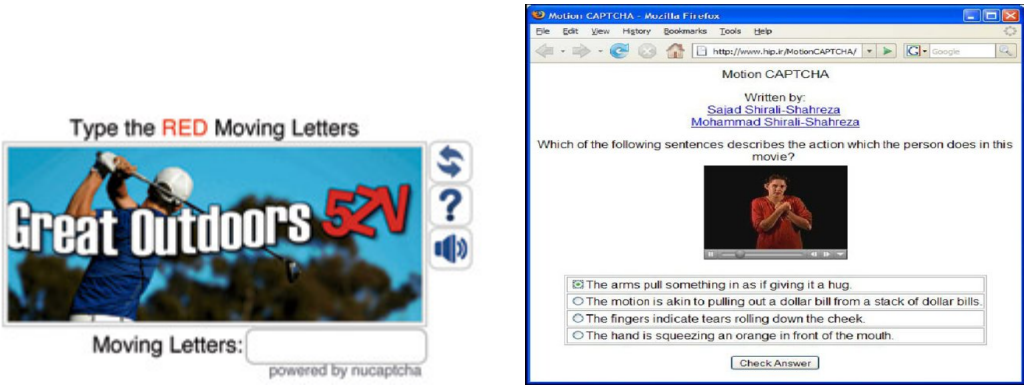
In the design phase of a text-based CAPTCHA there are many issues, related to Computer Vision attacks, to be considered. For each of them, there is usually a solution in the design phase of the CAPTCHA that reduces the probability that the challenge will be broken by a bot[17].

Figure 2.11: Example of text-based CAPTCHA.

2.1.7 Video-based CAPTCHAs

This type of CAPTCHAs isn't very common because it requires the download of a very large file[3]. The traditional video-based CAPTCHA is composed by a video in which a sliding text is shown (see **Figure 2.12a**). The user needs to type this message in a text field to pass the challenge. Some implementations of this type of CAPTCHAs are vulnerable to machine learning attacks.
Another version of the CAPTCHA is *Motion CAPTCHA*[18], developed by M. Shirali-shahreza and S. Shirali-shahreza, in which the user needs to watch a video. Then he needs to select which action was performed in the played file, choosing it from multiple answers (see **Figure 2.12b**). The strength of these implementations of CAPTCHAs is in the relationship between the multiple choices of the answer[19].

A similar implementation of the previous version is the one developed by Kluever et al. in which the user watches a video and then he needs to write three words that describe what he sees[20]. The same authors created a tag frequency-based attack to evaluate the security of their CAPTCHA scheme and they achieve a success rate of 13%.



(a) Example of sliding text in video. (b) Example of Motion CAPTCHA[18].

Figure 2.12: Examples of video-based CAPTCHAs.

2.2 Modern CAPTCHAs

The type of CAPTCHAs and authentication mechanisms, described in the following sections, are different from traditional CAPTCHAs because they don't challenge the cognitive capabilities of a human user. They work leveraging other parameters, such as behavioural analysis and data of the sensors. In the following sections there is a summary of the most known CAPTCHA schemes of this type.

2.2.1 Biometrics-based CAPTCHAs

Biometric-based CAPTCHAs are very often used to support the authentication mechanisms of a user. The most known schemes of this type are the following:

- **Bio-CAPTCHA voice-based Authentication**
  This authentication method was developed starting from good results reached in the authentication phase of cloud systems[21] (Alexa for Amazon, Siri for Apple, Cortana for Microsoft). This particular implementation uses a random voice-based password challenge. This password changes at every login and the user needs to say it out loud. The

strength of the challenge is the peculiarity of the human voice. The experiments reveals that unauthorized access probability decreases, while the usability is very high because it requires only the access to the microphone.

- **rtCAPTCHA**
  this type of authentication method is a Real-time CAPTCHA that asks users to perform some tasks like smile, blink or nod in front of the camera of the mobile phone. The recorded video is sent to the service provider that checks if there is the expected user performing the required action in the file.
  In the work of Erkam Uzun, Simon Pak Ho Chung, Irfan Essa and Wenke Lee[22], there is a detailed comparison between other similar authentication mechanisms and rtCAPTCHA, evaluating all possible Computer Vision attacks.

2.2.2 Behavioural-based CAPTCHAs

In 2014 Google announced that the Artificial Intelligence can solve now even the most difficult variant of text-based CAPTCHAs at 99.8% accuracy[23]. For this reason, the company develops the following CAPTCHA schemes:

- **Google no CAPTCHA**
  Google developed this new CAPTCHA system in 2015. It's simpler than traditional CAPTCHAs in terms of user interaction[24]. This CAPTCHA system is composed by two layers of protection:

  1. Checkbox *"I'm not a robot"* to be clicked by the user as shown in **Figure 2.13** (or image-based CAPTCHA on mobile devices)

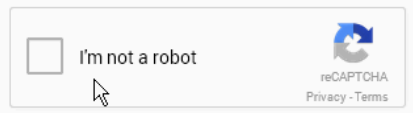


Figure 2.13: Example of Google no CAPTCHA checkbox.

  2. Traditional text-based CAPTCHA with two warped words

  The second layer is reached only if the user doesn't succeed in the first one. For the checkbox step, the application evaluates in background the user's behaviour (e.g. the mouse movement, where the user clicks, how long he lingers over a checkbox). Then the program performs an

advanced risk analysis, by looking at the results of the first step, the spam traffic and the number of passed/failed challenges. In this way the CAPTCHA understands if the test is passed or not.
The experimental results confirm that the first phase is very inefficient and many times the user fails performing this challenge even if he was a human and was performing correctly the task. A problem of this type of CAPTCHA is that many attacks exploits the image-based CAPTCHA (used on mobile phones) and the text-based CAPTCHA of the second phase using attacks based on known Computer Vision techniques or their variants (e.g. CAPTCHA breaker made by Suphannee Sivakorn, Jason Polakis and Angelos D. Keromytis[25]).

- **Google Invisible ReCAPTCHA**
  it's a top layer over the *Google noCAPTCHA v2.0*. It adds the option to bind the CAPTCHA directly to the form's submit element. In this way, the programmer can add layers to reduce the user experience of a bot[24]. It usually requires the use of cookies to track the user's behaviour. There exist two version of this CAPTCHA:

  - **ReCAPTCHA v2.0**
    it was developed in 2017. It's not really invisible because Privacy & Policy badge must be included on every pages of an app or a website in which the CAPTCHA is used. Computer Vision and Artificial Intelligence algorithms can break the challenges by recognizing object in the pictures in the image-based CAPTCHA phase.

  - **ReCAPTCHA v3.0**
    it was developed in 2018. It constantly analyses the human behavior, the mouse movements, the typing speed and the other features in NO CAPTCHA technology until Google collects enough data to tune their Google Invisible reCAPTCHA v2.0. This type of CAPTCHAs uses the probability scores of Artificial Intelligence and Machine Learning, the hostname, the timestamp and the action validation.
    Google removes the image recognition phase and looking only at the score, it evaluates if the user is a human or a bot. The main difference with respect to the previous versions of Google modern CAPTCHAs is that it returns a probability score, called *risk score*, in the range $[0.0, 1.0]$: *0.0* if the user is a bot, *1.0* otherwise. The administrator of the website can decide what range of scores he wants to manage, declaring when the site is under attack and

what actions need to be performed.
Over the year, several peculiarities and problems of this version of *Invisible ReCAPTCHA* were discovered. For example:

* If a user accesses a Web page using incognito mode or private mode, he is classified with a very low score (*high risk*).
* If a human is wrongly classified as a bot, the user can login into its Google account to increase its score. If this doesn't change the classification, you cannot do anything else.

2.2.3 Sensor-based CAPTCHAs

The devices, with access to the net, have natively many sensors nowadays, like the gyroscope and the accelerometer. The CAPTCHA schemes, described in the following sections, exploit the data generated from these sensors to improve the security of the authentication.

- **Completely Automated Public Physical test to tell Computer and Humans Apart (CAPPCHA)**
  this is a way to enforce the PIN authentication phase by mobile phone[26]. The user needs to tilt the device to a specified angle, specified on the screen (see **Figure 2.14**). The CAPPCHA security is based on the *Secure Element* (*SE*) in the device. It prevents brute force, side channel and recording attacks. The usability results are good because many comments, made by the users, were also considered in the implementation.

Figure 2.14: CAPPCHA and PIN authentication[26].

- **Invisible CAPPCHA**
  It will be described in details in **Chapter 4**.

2.3 CAPTCHA security

The process used to break many traditional CAPTCHAs, based on text or images, is usually organized into the following phases[27]:

1. **Pre-processing phase**
   in this phase, several techniques are applied to remove background, to separate foreground from the background, to delete noise and to remove some particular pattern (e.g. Canny Detection and Scale-Invariant Feature Transform (SIFT) application).

2. **Attack phase**
   the following techniques are usually applied:

   - ***Object Segmentation attacks***
     Segmentation techniques (e.g. vertical histogram, colorfilling, snake segmentation and JSEG) are used to split the CAPTCHA image into segments to simplify the recognition
   - ***Object recognition attacks***
     The most used techniques are pattern matching (e.g. shape context matching, correlation algorithm), OCR recognition, SIFT and machine learning.
   - ***Random Guess Attacks***
     The attacker's program tries to break the CAPTCHA scheme by guessing the correct answer. This attack is effective on CAPTCHAs with few number of different challenges.
   - ***Human Solver Relay Attacks***
     The bot forwards the CAPTCHA challenge to a remote human worker that will solve it if the previous phases don't produce good results. This technique is usually used also to solve other types of CAPTCHAs.

Many CAPTCHA schemes have still several issues:

- **Session issue**
  Some types of CAPTCHAs have a big issue because they don't destroy the session, after a correct answer is inserted by the user[1].
  Hence, the hacker can crack the next accesses using the same session id with the related solution of the challenge, after connecting to the web page of CAPTCHA. In this way the attacker can make hundreds of requests before the session expires and the previous operation must be computed again.

- **Resilience to both automated and human solver relay attacks**
  Many CAPTCHA schemes are designed to be robust against a possible AI attack but the new generation of CAPTCHAs involves the use of remote bot or human solver. Traditional CAPTCHA schemes are vulnerable to this type of attacks.
  Sensor-based CAPTCHAs are often vulnerable to relay attacks. An exception is the *Invisible CAPPCHA*, that will be analysed in **Chapter 4**.

- **Limited number of challenges**
  An issue of sensor-based CAPTCHA schemes is the limited number of challenges because the design of many usable gestures is very hard. This problem could be solved relying on trusted hardware.

- **Trade-off between Friction-heavy and Frictionless CAPTCHAs**
  A trade-off between usability and security aspects is always considered analysing CAPTCHA schemes. This condition is exacerbated in behavioural and sensor-based CAPTCHAs.

- **User's privacy**
  Sensor-based and behavioural CAPTCHAs usually send useful information to a remote server that analyses it to establish if the user was a human or a bot. If an hacker attacks the server side of this application, he can access users' private data.
  In some CAPTCHAs, the information is evaluated on the client side by a trusted hardware and the server receives only the results of the analysis. In this case, we need to be sure that trusted hardware is secure enough to guarantee privacy of user's information.

- **Compatibility with different devices**
  Many CAPTCHA schemes, e.g. behavioural ones, use specific forms factors but a good challenge should be compatible with different factors.

Type	Scheme	Usability issues	Security
Audio	*Audio reCAPTCHA*	Issues of recognition: • Knowledge of English dictionary by the user. • Some character sounds very similar to others.	It's vulnerable to: • ASR programs. • Deep Learning and ML techniques.
Game	*Dynamic Cognitive Game (DCG)*	Comprehension of rules.	Vulnerable to Stream Relay Attack
Image	*Click-based* *Drag & Drop-based* *Sliding-based* *Selection-based* *Interactive-based*	Difficulty in identification of images caused by: • Blur of images. • Low vision condition.	Vulnerable to: • Segmentation techniques • Deep Learning and ML techniques OCR techniques
Math	*Arithmetic* *QRBGS*	It requires basic or advanced math knowledge.	Vulnerable to: • OCR techniques • Side-channel attacks
Slider	Taobao.com TheyMakeApps.com	Simple and intuitive interaction	Simple bypassed by Javascript code and pupeeteer
Text	*2D* *3D*	Many problems have to be solved by user: • Multiple fonts • Font size • Blurred Letters • Wave Motion	It can be identified by: • OCR technique • Segmentation techniques • Deep Learning and ML techniques
Video	*Motion CAPTCHA*	Heavy file to be downloaded	

Table 2.1: Survey of main types of traditional CAPTCHAs[10].

Chapter 3

Side-channel attacks

A side-channel attack is an attack in which the malicious user exploits a side-information of transmitted encrypted data, to give access to the user's private data. This type of extra information could be, as an example, timing information, power consumption, electromagnetic radiations, sound and so on.
In the past the main side-channel attacks require the physical access to the victim's device. Nowadays, side-channel attacks are evolved and can be conducted by remote hackers using malicious code (e.g. cache-timing attacks, DRAM row buffer attacks), even exploiting information from sensors on mobile devices[28].
In fact a hacker can obtain side-channel information on mobile devices, from several sensors[29]:

- Location sensors (e.g. GPS, proximity)
- Motion sensors (e.g. accelerometer, gyroscope, magnetometer)
- Environmental sensors (e.g. for ambient light, temperature, barometer)
- Biometric sensors for wearable devices (e.g. heart rate sensor, ECG)
- Audio sensors (microphone)
- Video sensors (camera)

Side-channel attacks can usually be classified, looking at the action performed by the attacker, in the following categories:

- **active**
  the hacker influences the behaviour of the victim's device

- **passive**
  the attacker only analyses the leaking information

Side-channel attacks can also be classified, looking at the distance between the hacker and the victim during the attack, in the following classes:

- Local attacks
- Vicinity attacks
- Remote attacks

In this chapter there is a summary of the most popular side-channel attacks, organized with respect to the previous classifications (see **Table 3.2**). This study was conducted to find a feasible side-channel to be used in the design phase of a new CAPTCHA scheme, as support to the authentication phase.

3.1 Local side-channel attacks

In local attacks, the attacker needs to get the target device or to be very near to it. In many cases the hacker physically needs to manipulate the device or to obtain access to the chip.

Passive attacks
The following attacks are used to break cryptographic system implementations:

- ***Power Analysis***
  this type of attacks are based on the analysis of the power variations in transistors. There exist several popular attacks[30]:
  - ***Simple Power Analysis (SPA)***
    the attacker analyses the power consumption of the system, that depends on the microprocessor used. This analysis can be useful to understand which operations are performed by different implementations of cryptographic algorithm (e.g. RSA, DES).
  - ***Differential Power Analysis (DPA)***
    this attack collects data and then performs a statistical analysis and some error correction techniques from data to extract information correlated with secret keys.
  - ***High Order DPA (HO-DPA)***
    While DPA obtains information across a single event, HO-DPA correlates with multiple cryptographic sub-operations.

- ***Electromagnetic Analysis Attacks***
  the attacker can analyse indirectly the power consumption by accessing electromagnetic signals of the victim's machine. This type of attacks depends on the used instruments (e.g. EM probes) and on the analysed location of the chip. These dependencies also affect the signal-to-noise ratio.

- ***Differential Computation Analysis***
  the attacker tries to exploit white-box cryptographic implementations. In this model the attacker, even if he has access to code, can't extract the secret key. Hence the hacker needs to have full control over the target device and the execution environment. Then using binary instruments, he can control the intermediate states of the memory or memory operations (e.g. reading/writing operations)[31].

- ***Smudge Attacks***
  the attacker can exploit fingerprints and smudges on the screen of mobile devices to evaluate the user's input.

- ***Shoulder Surfing and Reflections***
  the attacker can exploit the lightness of the device display and obtain the user's activity by its reflection on surfaces such as sunglasses or tea pods.

- ***Hand/Device Movements***
  the attacker exploits the user's movements of fingers and hand to understand the interaction of the victim with the device.

Active attacks
The following attacks require that the hacker physically gets the device for a while:

- ***Clock/Power Glitching***
  in the past the attacker can fault inject on embedded devices by exploiting variations of the clock signal, (e.g. overclocking). However he needs to use an external clock source to do it.

- ***Electromagnetic Fault Injection (EMFI)***
  the attacker uses short (e.g. nanoseconds) high-energy electromagnetic pulses to change the state of memory cells. This attack allows to target specific regions of a microchip by locating the EM probe (e.g. on the instruction memory, the data memory, or CPU registers).

- ***Laser/Optical Faults***
  the attacker needs to decapsulate the chip to obtain access to it and, using a laser beam, it change the state of a transistor (e.g. changing bit value of a memory cell).

- ***Temperature Variation***
  the attacker can change the temperature, in which the target device is working, causing malfunctioning of the hardware. Temperature higher than the maximum one, supported by the hardware, causes faults in memory cells. Temperature too low changes the speed, for which the content of the RAM disappear, after turning the device off.

- ***Differential Computation Analysis***
  the attacker needs to have full control of the white-box environment, manipulating intermediate values in the system computation.

- ***NAND Mirroring***
  the attacker exploits the duplication of the data, usually used to recover data after faults, to restore a previous system state. The hacker can force the reset of the state as demonstrated by Skorobogatov for an Apple device[32].

3.2 Vicinity side-channel attacks

The attacker needs to wiretap or eavesdrop the network communication of the victim or to be in the neighbourhood of the target.

Passive attacks

- ***Network Traffic Analysis***
  the attacker can exploit meta data, related to the encrypted data, transmitted over the network. This information gives access to sensitive information about the traffic.
  For example a Web-application works between two parties: the client and the server. For this reason the communication channel is usually encrypted and the requests made by the user work through the *HTTPS* protocol.
  This solution isn't enough to prevent an attacker from exploiting reserved data because each web page has a distinct size and loads resources of different sizes. Hence the attacker can fingerprint the page even if HTTPS protocol is used.

Another cause of these attacks on Web-services is given by the trend of Web to work on *stateful protocols*, providing better performance to the client by keeping track of the connection information. TCP session for example works on Stateful Protocol because both systems maintain information about the session itself during its life[33].

- ***USB Power Analysis***
  the attacker can modify USB charging stations for mobile devices to obtain an analysis about the power consumption and the related sensitive information.

- ***Wi-Fi Signal Monitoring***
  Wi-Fi devices continuously monitor the wireless channel (channel state information (CSI)) to transmit data. Any environmental variation (e.g. finger motion) affects Wireless signals, generating unique patterns. For example the attacker can exploit these variations to unlock patterns on smartphones[34].

Active attacks

- **Network Traffic Analysis**
  the attacker, after obtaining information about transmitted packets, can interfere the traffic (e.g. delay of packets).

3.3 Remote side-channel attacks

These attacks are software-only based and they depend on the installation of the malicious code on the target device.

Passive attacks

- ***Linux-inherited procfs Leaks***
  the attacker can obtain a lot of information for each running process on Linux File System, looking at the content of the files in **Table 3.1**.

File path	Information
/proc/[pid]/statm	Virtual and physical memory sizes of process with identifier *[pid]*
/proc/[pid]/stat	CPU utilization times of process with identifier *[pid]*
/proc/[pid]/status	Number of context switches of process with identifier *[pid]*
/proc/interrupts	Interrupt counters
/proc/stat	Context switches

Table 3.1: Files used to obtain information about running processes.

- ***Data-Usage Statistics***
  the attacker can access to information, about incoming and outgoing network traffic, of each application without any permission.

- ***Page Deduplication***
  To reduce the overall memory footprint of a system, some operating systems perform deduplication, searching for identical pages within the physical memory and merge them even across different processes. When a process tries to write on a deduplicated page, a copy-on-write fault occurs and the process gets its own copy of this memory region again.

- ***Microarchitectural Attacks***
  By measuring execution times and memory accesses, the attacker can obtain sensitive information from processes running in parallel on the same device. This type of information can be evaluated from CPU caches, that are a big source of information leaks.

- ***Sensor-based Keyloggers***
  in mobile devices, the attacker can exploit information from equipped sensors with any permission. The user's interaction with the device can be evaluated by analysing information from the sensors.

- ***Fingerprinting Devices/Users***
  the attacker can obtain the identity of the device and the user and also the fingerprint by exploiting hardware issues and cookies.

- ***Location Inference***
  the attacker can obtain user's location without using GPS sensor, that requires permission to be accessed. For example, the accelerometer and the gyroscope can be used to infer car driving routes.

- ***Speech Recognition***
  the access to the microphone is protected by permissions. The attacker can also exploit other sensors to obtain information about human speech near to the device. The gyroscope has a sampling rate up to 200 Hz which covers a portion of the audible range of frequencies. In this way this sensor detects some vibrations caused by a human speech near to the device and the attacker can classify them with machine learning techniques.

- ***Soundcomber***
  if the attacker obtains permission of the microphone, he can obtain sensitive information (e.g. credit card numbers) on the automated menu services of phones.

Active attacks

- ***Rowhammer***
  the attacker can induce hardware faults by frequent accesses to main memory. This happens because the size of DRAM cells decreases nowadays to increase the density of memory cells in DRAM. However this causes electromagnetic coupling effects between near cells.

	Local	Vicinity	Remote
Passive	• Power Analysis • Electromagnetic Analysis Attacks • Differential Computation Analysis • Smudge Attacks • Shoulder Surfing and Reflections • Hand/Device Movements	• Network Traffic Analysis • USB Power Analysis • Wi-Fi Signal Monitoring	• Linux-inherited procfs Leaks • Data-Usage Statistics • Page Deduplication • Microarchitectural Attacks • Sensor-based Keyloggers • Fingerprinting Devices/Users • Location Inference • Speech Recognition • Soundcomber
Active	• Clock/Power Glitching • Electromagnetic Fault Injection (EMFI) • Laser/Optical Faults • Temperature Variation • Differential Computation Analysis • NAND Mirroring	• Network Traffic Analysis	• Rowhammer

Table 3.2: Survey of the most popular side-channel attacks[28].

Chapter 4

Invisible CAPPCHA

The *Invisible CAPPCHA* is an evolution of CAPPCHA, in terms of usability[35]. The main difference with respect to CAPPCHA is that the challenge is not explicitly submitted to the user but it is embedded in the PIN authentication phase leveraging a side channel. Invisible CAPPCHA works only on smartphones as its ancestor but it's very important to understand how data, obtained by the sensors, could be leveraged during the authentication phase. Invisible CAPPCHA is a method developed in 2018 and based on motion side-channel of mobile devices and it's very effective as support to Password-based authentication methods. The main steps, followed by this CAPTCHA, are:

1. **Motion detection**
   in this phase the micro-movements of the device, generated by the interaction of the user with the touch-screen, are evaluated by the *Secure Element* (*SE*)

2. **Communication between Client and Server**
   in this phase the credentials are shared with the remote Service Provider

The following sections will analyse this steps highlighting some peculiarities of Invisible CAPPCHA that will be used or modified to develop AcCAPPCHA, a CAPTCHA based on microphone sensors.

4.1 Motion detection

In this first phase, Invisible CAPPCHA exploits the accelerometer of the mobile device. It detects the acceleration over the three axis in g-force units, as a sequence of vectors over time:

$$\{A_i\}_{i=1}^{n} = \{(a_1^x, a_1^y, a_1^z), ..., (a_n^x, a_n^y, a_n^z)\}$$

This type of side-channel information from embedded accelerometer has been exploited in different attacks for the single and double tap detection. These attacks analyse the accelerations over the z-axis by comparing them to some thresholds and some timing conditions.
In Invisible CAPPCHA, the side-channel information is stored on the memory of the mobile device. Depending on the device, a smartphone built-in vibration can be generated along Z axis or along more than one axis (see **Figure 4.1**). On the contrary a finger tap event is defined by strong accelerations on Z-axis but also similar one to the other (see **Figure 4.2**).
In Invisible CAPPCHA, the difference between built-in vibration and tap acceleration is evaluated by a simple algorithm. It relies on negative and positive peaks, that are detected by comparing acceleration along Z axis with predefined thresholds. The differences between the tap accelerations and the vibrations is the main characteristic that guarantees user's tap cannot be simulated by a bot using the vibration motors.
Another important requirement, to prevent a bot attack, is that Invisible CAPPCHA uses a Secure Element that embedded the accelerometer of the device. In this way malicious code can't access to the sensor. Nowadays there exists a smart card, called SIMSense, that already integrates motion sensor and embeds it in a Secure Element.

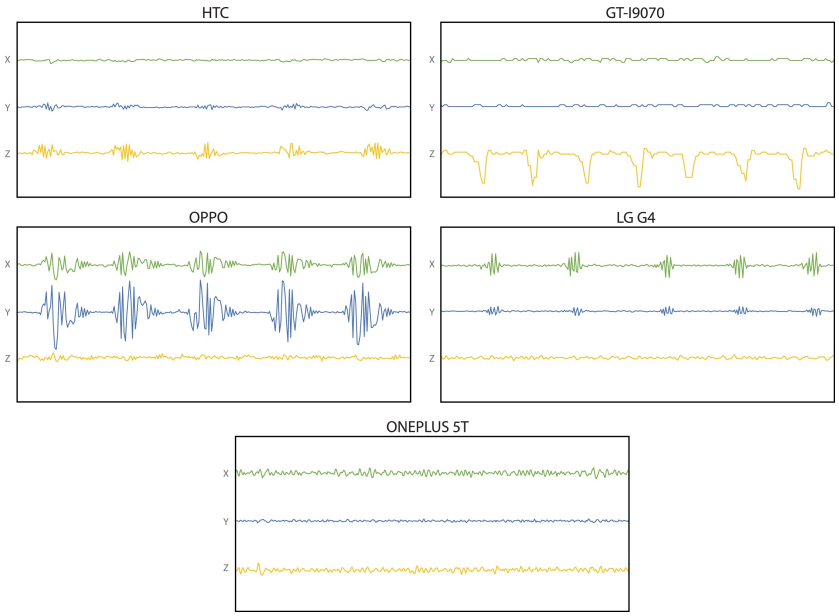


Figure 4.1: Example of accelerations caused by smartphone built-in vibration.

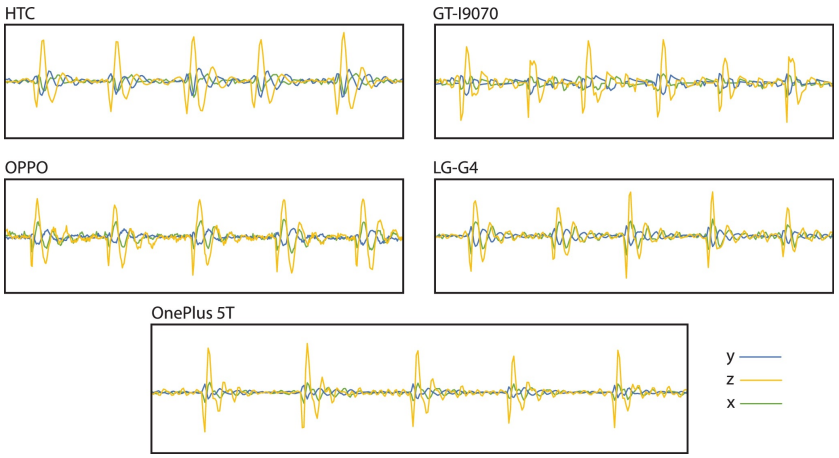


Figure 4.2: Example of accelerations caused by finger tap detection.

4.2 Communication between Client and Server

When the user fills a form or provides other information to a cloud application/service, the Secure Element checks if a micro-movement is measured when a user tap is detect. If this happens the input inserted by user is considered valid, or rather generated by a human, otherwise the algorithm tells that the input was generated by a bot.
An extra message, that tells if the task was performed by a user or not, is sent to the server side. The integrity of this message is guaranteed by the Secure Element, that can be equipped with a digital signature. The identity of the device can be associated to the message sent and then it can be checked and verified. The Secure Element signs the verification message through ECDSA.

4.2.1 Elliptic Curve Digital Signature Algorithm (ECDSA)

The elliptic cryptography works similarly to RSA but it uses smaller keys. The signature algorithm with elliptic curves is divided in two phases, like the one based on RSA. Considering two users, Alice and Bob, ECDSA phases are explained in the following lines:

- **Sign generation**
  If Alice wants to send a message, protected with digital sign, to Bob, they need to share the following parameters *(curve, G, n)*: *curve* is the equation of an elliptic curve, G is the base point of prime order on the curve and n is the multiplicative order of G for which $n\ x\ G = O$ (x is the scalar multiplication of a point of the curve). Alice generates a

private key d_A in the range $[1, n-1]$ and a public key $Q_A = d_A \; x \; G$. Alice needs to perform **Algorithm 1** to sign a message m.

Algorithm 1: Sign generation.

Input: `m`= message to be signed
Output: `(r, s)`= digital sign

$e \leftarrow HASH(m)$ where $HASH$ is an hash function (e.g. SHA-2)
$z \leftarrow$ string composed by the L_n most left bits
    where L_n is the bit length of the group of order n
$r \leftarrow 0$
$s \leftarrow 0$
while $r = 0 \; mod \; n \; or \; r = 0 \; mod \; n$ **do**
    $k \leftarrow RANDOM([1, n-1])$
    $(x_1, y_1) = k \; x \; G$ of the elliptic curve
    $r \leftarrow x_1 \; mod \; n$
    $s \leftarrow k^{-1}(z + rd_A) \; mod \; n$

- **Sign verification**
  Bob wants to verify the digital signature sent by Alice. To do it, he needs to apply in order **Algorithm 2** and **3**.

Algorithm 2: Verification that public key is on the elliptic curve.

Input: $\mathtt{Q_A}$= public key to be verified
Output: `check`= true if public key is correct

`check` $\leftarrow true$
\\Valid coordinates
if $Q_A = O$ **then**
    `check` $\leftarrow false$
\\Element of the curve
if $Q_A \in curve$ **then**
    `check` $\leftarrow false$
\\Correctness of order
if $not \; n \; x \; Q_A = O$ **then**
    `check` $\leftarrow false$

In Invisible CAPPCHA the message m is bitwise concatenated with a signed unique value, the nonce n, and then the signature is computed on their concatenation, $m||n$. Hence the signed message sent to the server is (r, s, m, n).

Algorithm 3: Sign verification.

Input: $(\mathtt{r}, \mathtt{s}, \mathtt{m})$= digital sign and message
Output: $\mathtt{m}$= message to be signed

$e \leftarrow HASH(m)$ where $HASH$ is an hash function (e.g. SHA-2)
$z \leftarrow$ string composed by the L_n most left bits
where L_n is the bit length of the group of order n
if $not\ r\ \in [1, n-1]\ or\ not\ s\ \in [1, n-1]$ **then**
    Invalid sign
$e \leftarrow HASH(m)$
$z \leftarrow$ string composed by the L_n most left bits
$w = s^{-1}\ mod\ n$
$u_1 = zw\ mod\ n$
$u_2 = rw\ mod\ n$
$(x_1, y_1) = u_1\ x\ G\ +\ u_2\ x\ Q_A$ of the elliptic curve
if $r \equiv x_1\ (mod\ n)$ **then**
    Verified sign
else
    Not accepted sign

4.3 Security analysis

After the verification of the signature, the communication must be also encrypted to ensure integrity and authenticity of exchanged messages.
The Secure Element can be accessed only through PIN authentication of the off-card communication party. If the malicious code has enough privileges to access Secure Element, some popular attacks can't be performed by an attacker.

4.3.1 Strength against popular attacks

The most popular attacks, that have been analysed, are[35]:

- **Replay attack**
  Because the message is signed together with a nonce, an attacker can't easily use a message already sent by a client to the server. In fact, the server checks if a nonce was already used by the client and if so, the server refuse the message sent by the attacker.

- **Reverse engineering attack**
  Even if the attacker can de-obfuscate the code of the application run-

ning on the browser, he can access to reserved data on the server only if the verification message for human interaction was correctly signed by the Secure Element. Hence this type of attack can't be performed.

- **Human-solver relay attack**
  The Invisible CAPPCHA is strong to this type of attack because it doesn't require any additional task to be sent to a remote human solver, as in standard CAPTCHAs.

- **Brute force and password replay attacks**
  Invisible CAPPCHA can be used to validate every input before it is considered as a possible attempt for a password. If the password was inserted by a malware or was wrong, the number of attempts decreases. Hence this approach prevents a brute force attack. This also prevents the access to the Secure Element by the attacker in replay attacks.

- **Denial Of Service (DOS)**
  If a malware tried more than the maximum amount of attempts of passwords it could do a Denial Of Service (DOS) of the Secure Element. To prevent this attack, the Secure Element can block access to itself if three invalid passwords are inserted by a human or if an invalid password is inserted by a bot.

Chapter 5

AcCAPPCHA

Analysing the structure of Invisible CAPPCHA in **Chapter 4** and the sensor-based keyloggers in **Section 3.3**, I design a CAPTCHA that works approximately like a keylogger. A keylogger is usually a malicious program, used to acquire information about the user's activity by exploiting side-channel information. Its implementation depends on the party, that the hacker wants to attack[36]:

- **The user**
  these attacks are based on the exploitation of physical information related to the typing state. For example, they can use electroencephalography (EEG), motion of the wrist in the smartwatches, video with keyboard line-of-sight and WiFi signal distortion.

- **The keyboard**
  these attacks are based on the analysis of the signals generated by the keyboard. For example, acoustic emanations can be exploited by using external physical sensors.

- **The host**
  these attacks are based on the physical access of the attacker to the victim machine. For example, the process footprint, the CPU load and other micro-architectural analysis can be exploited in these attacks.

- **The network**
  these attacks exploit the packets exchanged in the communication between the client and the server. For example, a network packet can be related to a keystroke revealing the key press time of the victim and the payload size of the server response.

I design a new CAPTCHA based on a keylogger that analyses the keyboard. Considering also the Bio-CAPTCHA voice-based Authentication described in **Section 2.2.1**, I decide to leverage the audio signal of the microphone of the device. The developed CAPTCHA is called AcCAPPCHA that stands for "Acoustic CAPPCHA". It works in background during the authentication phase, like Invisible CAPPCHA, and it leverages the acoustic side-channel available through the device microphone. This sensor was chosen to simplify the usability of a CAPPCHA, because it requires only the access permissions to it, without affecting the strength of its bot detection.
The whole implementation was made using `Python` language. The structure and the behaviour of AcCAPPCHA are similar to the ones proposed in Invisible CAPPCHA because they are both based on the analysis of some signal, detected by some sensor, during the insertion of the password.
The main difference is that the signal analysed by AcCAPPCHA is generated by the microphone, instead of the accelerometer. However AcCAPPCHA introduces also the analysis of the audio signals using neural networks.
According to the sequence of actions performed by Invisible CAPPCHA, AcCAPPCHA performs two steps for the authentication of a user:

- Evaluation of the user's activity
- Communication with the remote service

A detailed description of them is reported in the following sections.

5.1 Evaluation of the user's activity

During the evaluation of the user's activity, AcCAPPCHA records two audio signals: the first one created during the insertion of the password and the second one created before. The first signal will be analysed to find the audio peaks. The second signal is exploited to define a threshold, that will be used to find significant audio sequences in the first signal. It was introduced to manage background noise during the classification of the audio.
AcCAPPCHA performs two types of verification of the user identity:

- **Time correspondence**
  it always requires that the time instants, in which all the characters of the password were typed by the user, are stored. Then an algorithm checks if there exists a sequence of time peaks, observed in the first audio signal, that overlaps with stored time instants. If the sequence exists, the insertion was performed by a human, otherwise by a bot.

- **Character correspondence**
  it doesn't require to store the time instants like in the previous case because it analyses only the first audio signal by labelling each audio peak with a key of the keyboard. Hence AcCAPPCHA checks if there exists a sequence of labels, sorted by increasing time instants, that is equal to the sequence of characters of the password. If the sequence exists, the insertion was performed by a human, otherwise by a bot. The character correspondence could be also done after the application of the time correspondence. In this case, the only differences are:

  – AcCAPPCHA needs to store the time instants, in which all the characters of the password were typed by the user, to perform time correspondence

  – Time correspondence must return the audio peaks that overlap with the stored time instants

  – If the time correspondence had positive response, the character correspondence will be applied only on overlapping audio peaks. Otherwise, AcCAPPCHA tells that the insertion was performed by a bot.

  The combination of the two verification approaches is more accurate and stronger against false positives (bot detected as human) than the application of only character correspondence.

In the following sections the previous techniques will be explained in details to understand better their pros and cons.

5.1.1 Time correspondence

The first step of the algorithm is the definition of the noise threshold and it's performed before the application asks the user to type the password and after the insertion of the username. During this phase AcCAPPCHA records an audio file of 2 seconds, called `noise signal`, from the built-in microphone of the laptop and analyses it. The program looks for the maximum value of the audio signal, called T_N, that will be used later during the thresholding phase.
The next steps of the verification are performed by two threads simultaneously, during the password insertion. The first thread manages the insertion of the password and it's continuously waiting for the insertion of a character by the user until carriage return (`\r`) is typed. Immediately after a key is pressed, the time instant of this action, related to the Epoch of the PC, is

stored by AcCAPPCHA. The sequence of time instants stored by the thread is called $x = (x_0, ..., x_{|password|-1})$.
Meanwhile the second thread records an audio signal, called `user activity`, using the built-in microphone. The recording session begins immediately after the insertion of the username and it ends after the detection of the carriage return by the first thread. AcCAPPCHA removes the last 200 ms of this audio signal to be sure that the peak, related to carriage return, isn't included in the final audio file.
Once the above described pieces of information have been acquired, the application has everything it needs to understand if the user is a human or not. In fact the verification is performed by looking if there exists a sequence of audio peaks in the signal that overlaps with the series of the time instants stored by the first thread.
First of all, AcCAPPCHA performs the thresholding phase: `user activity` is analysed by keeping only the samples with values higher than $\mathtt{T_N}$. All the previous operations are performed also in the verification of the character correspondence (see **Section 5.1.2**). The samples, survived after the thresholding, will be grouped in several disjoint windows of maximum width equal to 100 ms. For each group W_i, the application looks for the sample with the highest value, $peak_i$. For example, given *the sampling period or interval* t_s and a specific group of samples:

$$W_i = (w_t, w_{t+t_s}, ..., w_{t+\lceil \frac{100\ ms}{t_s} \rceil * t_s})$$

the application computes the time instant of the sample related to $peak_i$, $t_i = argmax(W_i)$.
Given the sequence of computed time instants $t = (t_0, t_1, ..., t_{n-1})$, related to peaks of all the windows, n the number of windows obtained after thresholding and $|password|$ the size of the password, there is a **time correspondence** if there exists a subset of t, called $t^* = ({t^*}_0, {t^*}_1 ..., {t^*}_{|password|-1})$, that matches with the sequence of time instants stored during the password insertion. **Algorithm 4** is used to find a time correspondence and to establish if the password was inserted by a human or not. The procedure evaluates the distances between the time instant of the first audio peak and the time instants of the other ones. AcCAPPCHA finds a time correspondence if there exists an overlay of the time instants of the audio peaks and the stored time instants. After several tests, I decided to add a tolerance *threshold* of 100 ms during the search of the sequence.

Algorithm 4: Time correspondence

Input: $\mathtt{x} = (\mathtt{x}_0, \mathtt{x}_1, ..., \mathtt{x}_{|\mathtt{password}|-1})$ = time instants stored by first thread
$\mathtt{t} = (\mathtt{t}_0, \mathtt{t}_1, ..., \mathtt{t}_{n-1})$ = time instants relative to peaks of each group
`threshold` = threshold with respect to stored time instant

Output: `true` if human, `false` otherwise

$\mathtt{y} = (\mathtt{y}_0, \mathtt{y}_1, ..., \mathtt{y}_{|\mathtt{password}|-1})$ where $y_i = x_i - x_0$
if $n < |password|$ **then**
    //Number of found peaks lower than number of characters of the password
    return *false*
//Search of sequence
for $i \leftarrow 0$ **to** $n - 1$ **do**
    if $(n - i) < |password|$ **then**
        //Not enough peaks after t_i to be analysed to find the time correspondence
        return *false*
    //t_i already verified
    $\mathtt{j} \leftarrow i + 1$
    $\mathtt{count} \leftarrow 1$
    while $count < |password| \wedge j < n$ **do**
        if $(n - j) < (|password| - count)$ **then**
            //Not enough peaks after t_i to be analysed to find the time correspondence
            break
        if $(\mathtt{t_j} - \mathtt{t_i}) < (y_{count} - threshold)$ **then**
            //Too less time between the time instant t_i and t_j to obtain time correspondence ($t_j = t_{j+1}$)
            $\mathtt{j} \leftarrow j + 1$
        else if $(\mathtt{t_j} - \mathtt{t_i}) < (y_{count} + threshold)$ **then**
            //Time correspondence
            $\mathtt{count} \leftarrow count + 1$
            $\mathtt{j} \leftarrow j + 1$
        else
            //Too much time between the time instant t_i and t_j to obtain time correspondence ($t_i = t_{i+1}$)
            break
        if $\mathtt{count} = |password|$ **then**
            //Time correspondence found
            return *true*

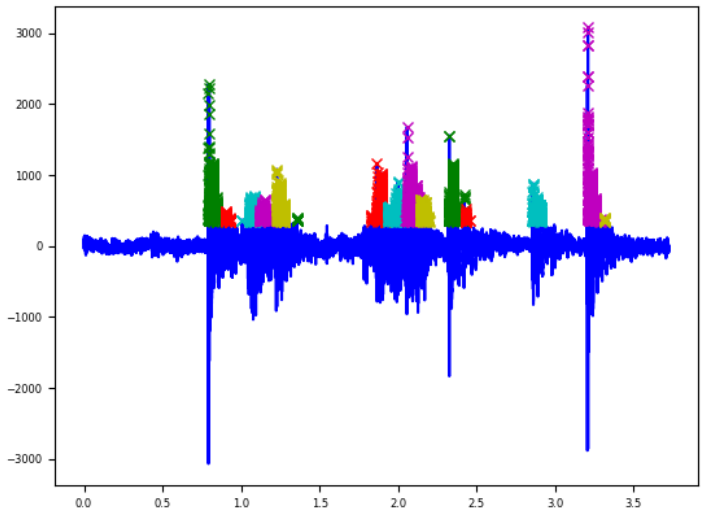


Figure 5.1: Audio during insertion of password `hello35` all sub-windows highlighted.

5.1.2 Character correspondence

The character correspondence also looks for audio peaks during the insertion of the password. The identification of the peaks is performed by using two threads as mentioned in time correspondence (see **Section 5.1.1**). The thresholding parameters are obtained in the same way, by exploiting the noise peak, but character correspondence also analyses the shape of the audio signal recorded during the insertion of the password.
When a key is pressed by a user on the keyboard, it produces a variation of the audio signal, called *press peak*, for a time window of about 8-10 ms[37]. This signal shape in each window can also be divided in three consecutive areas:

- **touch peak**
  peak in a window of 2-3 ms, caused by the user's finger touching the key
- **background noise**
- **hit peak**
  peak in a window of 2-3 ms at the end of the 8-10 ms window, caused by the finger and the key hitting the keyboard supporting plate.

The first goal of character correspondence is to obtain the character related to each key pressed by the user. AcCAPPCHA extract information from the touch peak, that is the most significant, and the hit peak.
Following the idea of Asonov and Agrawal, I exploit deep learning to classify each pressed key. In the following sections, there is a detailed explanation of the main phases of the classification method:

- Data acquisition
- Classification
- Verification

The first two phases were performed only once and were not executed by the CAPTCHA. On the contrary the last step is performed at every insertion of a password by AcCAPPCHA. In the first step I record an audio file for each key press. During the second phase, I extract information from the peaks of the audio files and I train a neural network using them. The verification procedure is based only on: the detection of the audio peaks during the insertion of the password, the extraction of the information from every audio peaks and the classification of them through the neural network trained at the previous step.

5.1.2.1 Data acquisition

The dataset acquisition was performed using two tasks running in parallel: the first one is a key-logger that is used to organize all the recorded audio files in several directories and the second one that records an audio file for each key typed. The communication between the two threads is performed through the private members of the class, in which the thread are created, and the use of the mutual exclusion (mutex). The keylogger guarantees that the acquisition of the audio files continues even if a special key is pressed (for example F3 button) and terminates only if `CTRL+C` is pressed.
The choice of running two tasks in parallel is because the recording session must start before the key press and it must end a little bit later than the acquisition of the character. In this way the shape of the audio peak in the signal, related to the key press, is not going to be cut. The data acquisition program can also acquire audio files for a series of key presses but in this case I decided to record one audio file for each key press.
In details the key-logger waits for the insertion of a single key by the user and then reports it to the thread that performs audio recording. This last task closes the audio stream and stores the audio signal into a *wav* file named

with a progressive number. All the audio files are dynamically organized into a set of subfolders of an output directory, each one with the name of the respective typed key.
The recording phase was performed using the built-in Realtek microphone and the keyboard of my MSI GL63 8RD laptop. The names of the subfolders (labels), in which each audio file of a key press is inserted, are reported in **Appendix A**.
Looking at the Table in the Appendix, the name of the label, related to a key press, is highlighted. The keylogger maps each key to an ASCII string because otherwise many keys would be mapped into invalid names of folders (for example, the key . is now mapped into the label `POINT`). Another observation about the table in **Appendix A** is that there are two columns of labels: the first one related to the label seen by the key-logger, the second one related to the label manually assigned by me to each key. Sometimes these labels differ for the same entry because:

- **I want to highlight the spatial distribution of the keys on the keyboard**
  for example, `INSERT` and `0_INSERT` (with Num lock on) would be mapped into `INSERT` by the keylogger but they are considered different thanks to my map;

- **I want to improve the classification of the keys made by the keylogger**
  for example, `ALT` label is wrongly mapped into `SHIFT` by the keylogger.

- **I want to add keys that are mapped only on the hardware level**
  `FN` is the only key with this problem. The keylogger doesn't detect any key, when `FN` is typed by the user. Hence I had to type `FN`, followed by another key `a`, so the audio files will be stored in `a` subfolder. Then I rename the subfolder `FN` and I create a python script to resize all the audio signals inside it and remove the useless second peak, related to `a` press.

The last two reasons are very important because they highlight also the power of the acoustic side-channel. If an attacker implements an high-level keylogger exploiting also microphone information and remapping wrong labels, the accuracy of its software can increase very much with respect to a traditional keylogger without the audio analysis. In fact the hacker could collect a dataset of recordings of pressed keys on the same type of the victim's keyboard and then it could train a Neural Network to be used for malicious

purposes.
However I record 200 audio signals for each key of my keyboard obtaining a dataset of 20400 audio files. I performed also Data Augmentation on them trying to improve the accuracy of the prediction for the neural network, using two approaches:

- **Time-shift**
  from each audio signal I created four new audio signals obtained by applying a time-shift respectively of 0.5, 1.0, 1.5 and 2.0 seconds.

- **Introduction of Gaussian noise**
  from each audio signal I created four new audio signals by adding a sequence of random samples, taken from a Gaussian distribution with the standard deviation equal to 150 and the mean equal 0.

Using these approaches I obtained a training set of 1800 audio signals for key, composed respectively by the following datasets:

- 200 audio signals manually recorded by me
- 800 audio signals obtained by time-shift technique
- 800 audio signals from introduction of Gaussian noise

The accuracy of the Neural Network trained on audio signals of both first and second datasets is higher than the one related to the Network trained on only the first dataset. The efficiency of the Neural Network trained on both the first and the third datasets is worst than the one related to the network trained on only the first dataset.
I notice that the reason of the previous observation is that the third dataset has many signals, related to different keys, with FFT coefficients similar one to the other. Hence I used only the network trained on the first dataset and both on the first and the second dataset as prediction models. Considering that my keyboard is composed by 102 keys, I used respectively a dataset of 20400 and 102000 audio files to train my neural network.

5.1.2.2 Classification

During the classification phase, I decided how to extract the features from each audio peak, that will be used as input of the neural network. I tried to use three different types of features:

- the FFT coefficients of the touch peak

- the FFT coefficients of the hit peak and the touch peak

- the features obtained from the hit peak and the touch peak using a deep learning pre-trained model

The first two approaches come from the idea of Asonov and Agrawal's work and the last one was based on the modern sound classification techniques. In the first two cases, the FFT coefficients are extracted from the 3 ms windows around the peaks and then they are normalized in floating point values in range $[0, 1]$ (see **Figure 5.2**).

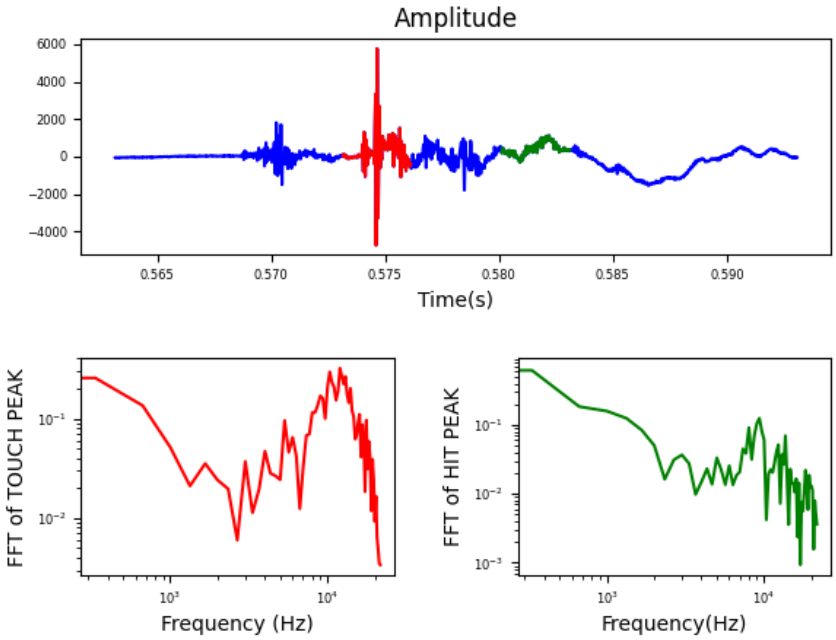


Figure 5.2: Example of normalized FFT coefficient of the touch peak and the hit peak for an audio file of key `0`.

In the third case, the touch peak and the hit peak samples, taken by the 3 ms windows, are concatenated and then the spectrogram is computed over these samples (see **Figure 5.3**). From the spectrogram, I extract a feature composed by 512 values through the use of the VGG16 convolutional neural network.

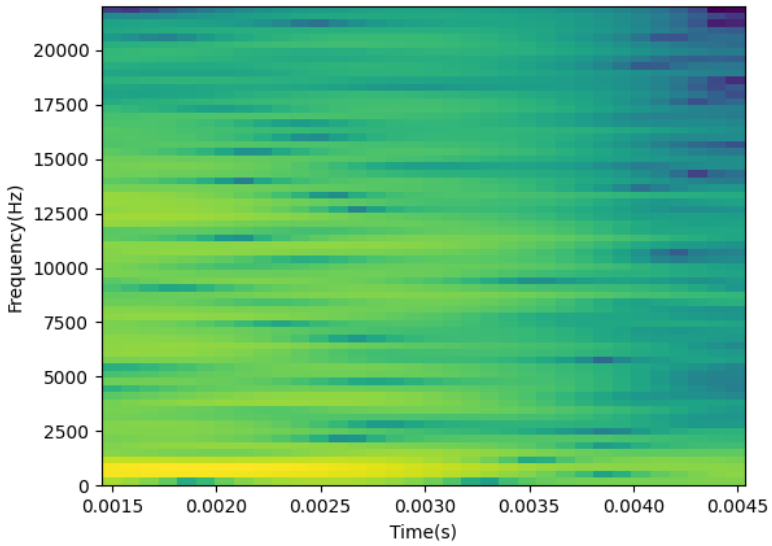


Figure 5.3: Example of spectrogram for an audio file of key `0`.

VG16 was pre-trained on the images of ImageNet database and I used the intermediate results from neurons, before the last fully connected layers, as feature. The reason of this approach is that a pre-trained network already extracts very well features for good classification of many labels and it can extract features better than a Convolutional Neural Network, for image classification, created from scratch.
The features obtained from the audio files are organized into two dataset: the training set and the test set. After the extraction of the features with one of the previous methods, I created a deep neural network from scratch. Its number of input neurons is equal to the size of a feature and the number of output neurons is equal to the number of keys to be mapped. I tried to insert several hidden layers composed by different number of neurons but after changing many parameters, I decided to use only one hidden layer of 1024 neurons if the feature was obtained using the spectrograms and 100 otherwise.
This choice comes from the best trade-off between the accuracy (83-86%) of the network on the test set and the correct prediction of the labels for new audio peaks obtained later during the insertion of the password. Each label associated to a key press (see **Appendix A**), is mapped into an integer number in $[0, n-1]$ where n is the number of the output neurons.
Each time the neural network receives a feature as input, it would return a series of floating point values (probabilities) in $[0, 1]$, one for each output

neuron. The neuron, related to the highest probability, is the label predicted from the input feature. The training phase of the neural network is performed looking at the real label of each audio file and the predicted one. Thanks to the back propagation procedure, the weights of the network are modified and hence the network becomes trained.

5.1.2.3 Verification

The verification phase is performed by AcCAPPCHA. The audio signal, recorded during the insertion of the password, is analysed and then the verification can be based on:

- every audio peaks obtained using the threshold from the noise evaluation
- audio peaks related to the time instants of the time correspondence

After the selection of the previous peaks, AcCAPPCHA analyses the touch and the hit peaks and it computes the features using the approach chosen by the user at the start-up of the program, from the ones described in **Section 5.1.2.2**. The selection of the features type is only for testing purpose and, if the CAPTCHA is released, it will be removed by using only the feature with the best results. I perform the prediction using the Neural Network trained during the classification phase. For the feature of each audio peak f_i, I collect the 10 most probable predicted labels in the set $Y_i = \{{y_i}^0, {y_i}^1, ..., {y_i}^9\}$.
Then AcCAPPCHA evaluates if there exists a character correspondence by using **Algorithm 5**. If there exists a sequence of sets that "overlaps" with the characters of the password, inserted by the user, the algorithm decrees that a human performed the action. Otherwise, the user was a bot. If the verification is performed using every audio peaks, the results are very weak because it could introduce also false positives.
This could happens when the audio, recorded during the noise evaluation, is very flat and the audio, related to the insertion of the password, is very noisy. In this case the sequence of audio peaks, found by the character correspondence, could be composed by peaks that aren't related to all the real key presses. The main problem of this approach is caused by the missing time correspondence between a character, belonging to the final sequence, and the instant in which the same character was physically inserted by the user.
However, as previously mentioned, the character correspondence could also be applied after the time correspondence. In this way, **Algorithm 5** would be applied on the sets of labels predicted for the audio peaks, obtained by

the time correspondence. In this case, AcCAPPCHA verification becomes more accurate even if in practice character correspondence isn't very efficient because the neural network often fails in single key prediction, causing some false negatives (a human is classified as a bot).

Algorithm 5: Character correspondence

Input: $\texttt{password} = (\texttt{x}_0, \texttt{x}_1, ..., \texttt{x}_{|\texttt{password}|-1})$ = sequence related to the password, where x_i is the label related to i-th character of the password typed by the user
$\texttt{Y} = (\texttt{Y}_0, \texttt{Y}_1, ..., \texttt{Y}_{\texttt{n}-1})$ = sets of labels related to all the audio peaks

Output: `true` if human, `false` otherwise

```
1  if n < |password| then
2      //Number of found peaks lower than number of characters of the password
3      return false
4  //Search of sequence
5  i ←0
6  for j ← 0 to n − 1 do
7      if x_i ∈ Y_j then
8          //Character x_i found in the set of labels Y_j
9          i ←i + 1
10         j ←j + 1
11     if i == n then
12         //Character correspondence found (all the characters of the password
             were found in Y)
13         break
14 return i==n
```

5.2 Communication between client and server

After the evaluation of the user activity, the response will be signed through ECDSA and the credentials of the client will be sent to the server if and only if the insertion of the password was performed by a human. The evaluation of the user's activity (see **Section 5.1**) is performed at the client side but the server is the one that establishes if the user was a human.
The response of the evaluation of the user's activity, concatenated with a nonce and signed through ECDSA, is sent to the server (see **Section 4.2**).

The use of the nonce, a unique and random generated sequence, is very important to guarantee that no replay attacks would be performed. In fact after the reception of a message from the client, the server checks if the same client has sent the same nonce before and in this case it declines the message of the client. In this way, any attacker can't reuse a message sent by a human client before.
The ECDSA signature can be also useful to sign HTTP data, for example a POST request, to increase the integrity of the messages and the security of the communication between the client and the server. In the testing phase I performed, I designed and implemented also a simplified version of the communication between the client and the server during the authentication service. The application was tested on the local network and the actions performed by the involved parties are described in details in the following sections (see **Figure 5.4**).

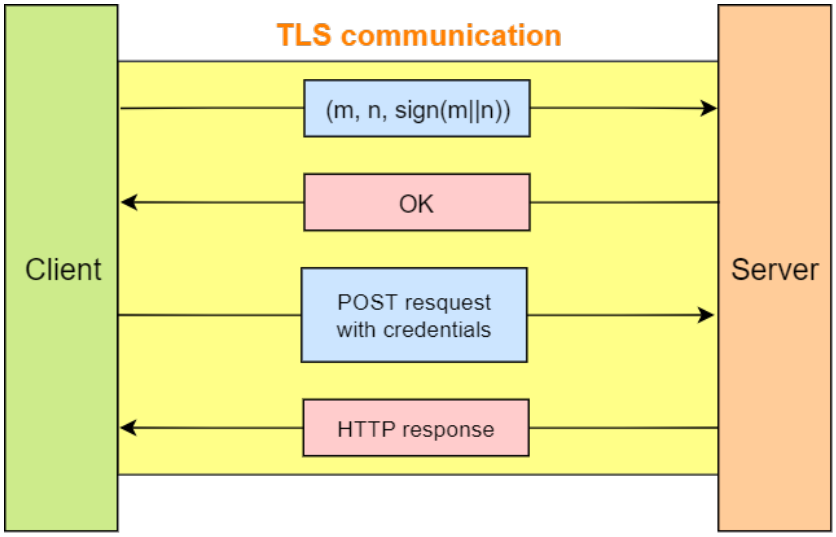


Figure 5.4: Authentication between client and server using AcCAPPCHA.

5.2.1 Client

The client performs the authentication following these steps:

1. The client establishes the connection with the server using the Transport Layer Secure (TLS) protocol to increase the security strength of the communication between the parties;

2. The client sends the message $(m, n, sign(m||n))$ to the server where:

- m is the string with the response of the evaluation of the user's activity on the client side (`True` if Human, `False` if bot)
- n is the client nonce
- $sign(m||n)$ is the ECDSA signature of the concatenation $m||n$ of the response and the nonce. I decided to use SHA256 as hashing function exploited by ECDSA.

In practice, I format the message $(m, n, sign(m||n))$ in the following way to easily distinguish the portions of the message:

```
m CRLF n sign(m||n)
```

According to the basic rules in the grammar of **HTTP/1.1** (see Section 2.2 of RFC 2616), *CR* is the carriage return (\r) and LF is the line feed (\n). The spaces in the format string aren't inserted in the message, if not explicitly specified with format string **SP**. The introduction of \r \n is intended to easily identify m because it can be a message composed by 4 or 5 characters. However the nonce is a Universally Unique Identifier (UUID) and it has a fixed length of 16 bytes. In this way, I emulated the uniqueness of the nonce.

3. The client waits for the response of the server, that has the following format:

```
response CRLF
```

If the answer is equal to OK \ r \ n, the user was a human and AcCAPPCHA will go on with the authentication step. Otherwise the user is classified as a bot and the client-side application performs again the verification, asking the user to insert the password. The maximum number of trials for a specific user is 3 and if the client reaches it, the client-side will block the next accesses to AcCAPPCHA for a fixed amount of time. In this case AcCAPPCHA can't run and it terminates immediately the execution by writing on the standard output that the user can't access to the application.

4. If the user was classified as a human, the client sends the credentials (username, password) to the resource `/cgi-bin/auth` of the server, using an HTTP POST request, and performs the next phases. The name of the folder `/cgi-bin` is the name of the standard path that

identifies function calls. In fact *auth* is the name of the function that server will call during the authentication phase. This naming approach was used very much in the past to separate functions code from pure HTML code. The password is not sent directly but it's hashed using SHA512. The POST request sent by the client has the following format:

```
POST /cgi-bin/auth HTTP/1.1 \r \n
Host:  SP foo.example CRLF
Content-Type:  SP application/x-www-form-urlencoded CRLF
Content-Length:  SP SIZE CRLF CRLF
user = USERNAME & pwd = HASHEDPWD
```

where **SIZE**, **USERNAME** and **HASHEDPWD** are replaced respectively with the size of the HTTP body, the username of the client and its password hashed with SHA512. The format of the POST request follows again the grammar of HTTP/1.1, specified in RFC 2616).

5. The client waits for the HTTP response of the server, containing HTML code as body. Then the client saves the code on the file system and opens the default web browser to show the HTML page received. The HTML code is intended to show 3 possible scenarios: the user was correctly logged in, the user inserted wrong password, the username wasn't already stored on the server database. The client can insert wrong credentials for at most 3 times. If the limit is reached, the access of the user to the client side of AcCAPPCHA is blocked.

When the user is blocked because the client inserted wrong credentials or a bot was detected, the client-side of AcCAPPCHA stores the datetime in a file called `block.txt`. At every start-up of AcCAPPCHA on the client side, the program checks if a fixed amount of time was passed from the stored datetime. If the block period is already elapsed, the file `block.txt` is removed by the client, otherwise it isn't touched and it will be used for future checks. During the testing phase, I used a block period of 100 seconds but in practice it should be bigger.

5.2.2 Server

The server performs the authentication of the client following these steps:

1. The server establishes the connection with the client, after his request, using the TLS protocol to increase the security strength of the communication between the parties;

2. The server receives the message $(m, n, sign(m||n))$ and it checks the integrity of the message. To do it, the server decrypts the ECDSA signature using the client's ECDSA public key and it compares the result with $m||n$. If the comparison fails, the server replies `No integrity \ r \ n` to the client.

3. If the previous comparison has a positive response, the server checks if the nonce was already used by the same client. If this happens, the server thinks that an attacker is performing a replay attack and the server replies `NO \ r \ n` to the client. If the nonce wasn't already used by the client, it will be stored in a dictionary to monitor clients activity. Each entry of the dictionary is composed by:

   - **Key: IP address**
     It is the IP address of the client and it is a simplification of the information that could identify a client. For example the client could be associated also to port number used to make the request, the Operating System on which the AcCAPPCHA was running on the client-side or other useful parameters.
   - **Value: list of nonces**
     Every time a client performs a new verification request on the server, the nonce used is added to the list related to its key in the dictionary.

4. If the nonce was used for the first time by the client, the server checks the value of the response received by the client. If the response is `True`, the server replies `OK \ r \ n` otherwise `NO \ r \ n`. If some error occurs it sends `ERROR \ r \ n` to the client. In the last two cases, the server doesn't perform the following steps.

5. If the server doesn't terminate, it waits for the POST request from the client and analyses it to perform the authentication service. The server replies to the client with an HTTP 1.1 response with several possible status codes:

   - **501** (Not implemented)
     If the request is not a POST (e.g. GET)

- **400** (Bad Request)
  If the number of the parameters in the POST body is different from 2 (username and password).
- **200** (OK)
  If the number of the parameters in the POST body is equal to 2, the server will reply with an HTTP response with a body content depending on several cases:
  - **The user isn't in the database**
    the server sends the following HTML code if the specified username isn't already stored in the database.

```
<!DOCTYPE html>
<html>
   <body>
      <h1>Failure</h1>
      <p>First sign up.</p>
   </body>
</html>
```

Failure

First sign up.

  - **The password is wrong**
    the server sends the following HTML code if the hashed password, sent by the client, isn't equal to the one stored in the database for the username, specified by the client.

```
<!DOCTYPE html>
<html>
   <body>
      <h1>Failure</h1>
      <p>Wrong password.</p>
   </body>
</html>
```

Failure

Wrong password.

  - **The user correctly logged in**
    the server sends the following HTML code if the specified username exists in the database and the hashed password stored in the database is equal to the one received in the POST request.

```
<!DOCTYPE html>
<html>
   <body>
      <h1>Logged in</h1>
      <p>Welcome back.</p>
   </body>
</html>
```


Logged in

Welcome back.

5.2.3 Database

The database, created to simulate the search of a username by the server, was made using PostgreSQL. I could store all the information in a simple text file or a csv file but I decided to use this approach to be more flexible and to emulate the integration of AcCAPPCHA in more complex web applications. The database is composed by a single table `CloudUser` that stores information about the users identities, that are usually asked to the clients during the sign up phase. The creation of the database table was performed using the following instructions:

```
-- Database Creation
CREATE DATABASE cloudservice OWNER POSTGRES ENCODING = 'UTF8';

-- Connect to cloudservice db to create data for its 'public'
    schema
\c cloudservice

-- Create new domains
-- Correct password format
CREATE DOMAIN pwd AS char(128)
  CONSTRAINT properpassword CHECK (((VALUE)::text ~* '[a-f0-9]'::
      text));

-- Correct password format
CREATE DOMAIN userformat AS char(128)
  CONSTRAINT properpassword CHECK (((VALUE)::text ~* '[A-Za-z0-9]
      '::text));

--Correct mail format
CREATE DOMAIN mail AS character varying(254)
  CONSTRAINT propermail CHECK (((VALUE)::text ~* '[A-Za-z0-9._
      %-]+@[A-Za-z0-9._%]+$'::text));
```

```
--Create new data type
CREATE TYPE gendertype AS ENUM (
    'Male',
    'Female'
);

--Create tables
CREATE TABLE CloudUser (
    Name VARCHAR NOT NULL,
    Surname VARCHAR NOT NULL,
    Username userformat NOT NULL,
    Email mail NOT NULL,
    Sex gendertype,
    Password pwd NOT NULL,
   PRIMARY KEY (Username)
);
```

I created several domains to manage the format of some information of the user. For example the password is a string of 128 characters because each password is hashed using SHA512 and then stored as a string of hexadecimal digits. Then I populated the dataset with some entries, related to fake users, only for testing purpose. An example of the query that I use to insert information about a user is reported next:

```
INSERT INTO CloudUser (Name,
                       Surname,
                       Username,
                       Email,
                       Sex,
                       Password)

VALUES ('Raffaele',
        'Di Nardo Di Maio',
        'RaffaDNDM',
        'example1@gmail.com',
        'Male',
       HASHPASSWORD);
```

During the insertion, `HASHPASSWORD` was replaced by the string of 128 characters corresponding to the hashed password in hexadecimal format.

5.2.4 Encryption Keys

The creation of the TLS socket for the communication between the client and the server is done by using keys and certificates created thanks to the following bash instructions:

```
openssl req -new -x509 -days 365 -nodes -out client.pem
        -keyout client.key
```

```
openssl req -new -x509 -days 365 -nodes -out server.pem
        -keyout server.key
```

OpenSSL is a open-source implementation of TLS/SSL protocols and, thanks to the option `-x509`, you can display certificates and also access to many signing protocols. In particular, in the previous bash instructions, a X.509 Certificate Signing Request (CSR) is generated and signed for both the parties.
Thanks to `-nodes` the private key is created and not encrypted. The certificates are stored respectively in *server.pem* for the server side and *client.pem* for the client and they are valid for 365 days. The private keys are stored, using the `-keyout` option, in the files *client.key* and *server.key*.

The keys, used in ECDSA signing and verification, were created as follow from the Python language instead of using a bash tool:

```
from ecdsa import *
from hashlib import sha256

PRIVATE_KEY = SigningKey.generate(curve=SECP256k1,
                                  hashfunc=sha256)

with open('ecdsa.key', 'w') as private_pem:
    private_pem.write(PRIVATE_KEY.to_pem().decode())

PUBLIC_KEY = PRIVATE_KEY.get_verifying_key()

with open('ecdsa.pem', 'w') as public_pem:
    public_pem.write(PUBLIC_KEY.to_pem().decode())
```

In this case the private key, used to sign a message from the client, was computed on the curve `SECP256k1` usually used in Bitcoin applications.

Chapter 6

Experimental results

All the tests were performed on a MSI GL63 8RD laptop, using the version 2004 of Windows 10 Home as Operating System. AcCAPPCHA results are described in the following sections, analysing two aspect: the security efficiency and the usability.

6.1 Security analysis

The main tests were performed by analysing the response of AcCAPPCHA when the user is a human and when it is a bot. All the techniques, described in **Section 5.1**, are tested and the strength against the attacks mentioned in Invisible CAPPCHA (see **Section 4.3.1**) are analysed again for the new CAPPCHA.

6.1.1 Human detection

The responses of AcCAPPCHA for a human user were tested in all the three possible operating modes (see **Section 5.1**):

- Time correspondence
- Character correspondence
- Both time and character correspondence

The first method is the most efficient and AcCAPPCHA usually detects the human activity of a user at first or second trial using it. Sometimes the correspondence isn't found at first trial because of background noise. The last two approaches based on deep learning require the reduction of the number of labels for the classification of a key.

In this way I limit the possible characters that user could use. Hence the password can be composed of lower case alphabetic characters, accented vowels or numbers (see **Figure 6.1**). For this reason, I trained again the network using less labels and only the audio files related to these labels. However the user can also type the backslash key \b but the corresponding character isn't analysed in the final verification.
AcCAPPCHA still considers the 10 most probable labels, predicted by the neural network, for each audio peak during the insertion of the password. The prediction apparently seems to work better after the simplification but this happens only because the correct key of an audio peak has higher probability to be predicted on a subset of all the keys of the keyboard.
At the contrary of D. Asonov, R. Agrawal's conclusions[37], I notice that, using the deep learning techniques, there are many false negatives given by the similar sound produced by several keys. The reasons of their presence could be:

- different parts of my keyboard plate produce similar sounds
- the built-in microphone of my laptop is more affected by the noise than an external microphone
- the movements of the user's hands influence the noise during the password insertion
- some keys are more worn out than other

AcCAPPCHA was executed three times for both the passwords `bye5` and `hello35` and for each type of used features. As explained in **Section 5.2.1**, a human user has at most 3 attempts to be correctly classified as human during an execution of AcCAPPCHA. For each execution of AcCAPPCHA, the **Table 6.1** reports the number of the step in which a human user is correctly detected as a human using both the character and the time correspondences. If the number of the step is replaced by the string NO, it means that AcCAPPCHA wasn't able to understand that the user was a human during the three trials. For each entry of the table where the number of the step is different from one, I also reported the number of characters correctly predicted in all the previous steps.
According to the results in the table, the worst deep learning method is the one that uses the spectrograms and a pre-trained deep learning model to extract the features of a press peak.

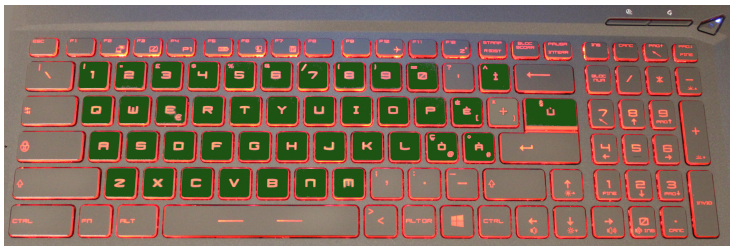


Figure 6.1: Keys to be used in the password (highlighted in green).

Feature type	*bye5*	*hello35*
FFT of touch peak	2 (3)	1
	2 (3)	1
	1	1
FFT of touch and hit peaks	2 (3)	2 (6)
	2 (3)	1
	3 (3, 3)	3 (5, 6)
Spectrogram of touch and hit peaks	NO (3, 3, 3)	NO (5, 5, 6)
	NO (2,3,3)	NO (6,6,6)
	NO (3,2,3)	NO (5,4,5)

Table 6.1: Step number when the human user is correctly identified as a human by AcCAPPCHA (3=maximum).

6.1.2 Bot detection

I've tested the bot activity analysing the response generated by AcCAPPCHA. I emulated it using several approaches and using the username `RaffaDNDM` and its password **hello35**:

- **Python program with `popen` communication**
  this approach opens a subprocess and a communication through pipes with stdin and stdout streams of AcCAPPCHA. Supposing that the hacker obtained the credentials in some way, the bot communicates the username and the password to running CAPPCHA with only the time correspondence selected. In practice the strength of AcCAPPCHA against this attack is very high because the insertion of the password is managed by AcCAPPCHA using consecutive calls to `getwch()` in the Windows Operating system.

This function belongs to `msvcrt` module and takes one character at the time. This module guarantees also that the console I/O routines are not compatible with stream I/O or low-level I/O library routines. In the Windows operating systems, the output from these functions is always directed to the console and cannot be redirected through any kind of pipes.
The program easily accepts the username, because AcCAPPCHA acquires it using the standard `input()` function, and then it waits for the insertion of the password. Hence the CAPPCHA waits forever the insertion of the first character of the password because `popen` can't access to the stream analysed by `getwch()`. I didn't iterate the insertion of the password for the maximum number of possible trials because even at the first insertion, AcCAPPCHA doesn't see any action performed by the bot.

```
from subprocess import Popen
import sys
import msvcrt
from time import sleep

def popen_bot(username, password):
    #Subprocess that redirects pipes
    process = Popen('python3 AcCAPPCHA.py -t -plot',
                    shell=True,
                    stdin=subprocess.PIPE,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.STDOUT)

    #Wait until username could be inserted
    sleep(4)
    #Write username and password
    credentials = username.encode() + b'\r\n' + \
                  password.encode() + b'\r\n'
    output = process.communicate(credentials)[0]

    print(output.decode())
```

Listing 6.1: Bot using popen.

- **Python program with `pynput` module**
  Using this module, the bot can access directly the console, bypassing the stream limits of **msvcrt** character acquisition. To emulate the user, the bot program should start the execution and immediately after the hacker must open the working terminal with AcCAPPCHA running. Then each character of the password is inserted by the bot on the

selected terminal, by emulating the press and the release of the corresponding key of the keyboard.
This scenario isn't realistic because it requires the management of terminal windows by a human user. However it was useful to establish if the insertion of the password by a malware is correctly classified as a bot activity.

```
from pynput.keyboard import Key, Controller
from time import sleep

def input_bot(username, password):
    #Object for control of keyboard events
    keyboard = Controller()

    def press_release(char):
        keyboard.press(char)
        keyboard.release(char)

    #Wait that username could be inserted
    sleep(4)

    #username insertion
    for x in username:
        press_release(x)

    press_release(Key.enter)

    #Trials for password insertion
    count = 0
    while(count<3):
        sleep(5)

        for x in password:
            press_release(x)

        press_release(Key.enter)
        count += 1
```

Listing 6.2: Bot using pynput module.

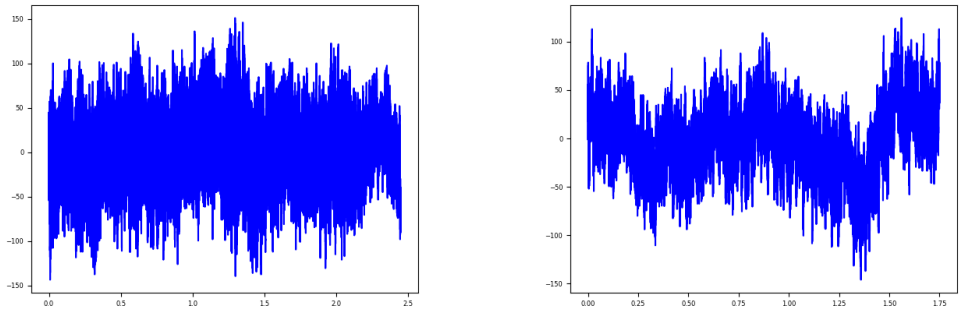
- **Remote control of the PC**
  the last test was performed by using the program `Team Viewer` and accessing directly the terminal on which the remote user will run AcCAPPCHA. This type of attack isn't realistic but it has only a testing purpose like the previous mentioned attack with the Python bot.

In the last two approaches the audio files recorded by AcCAPPCHA are very

similar and highlights the two most probable situations:

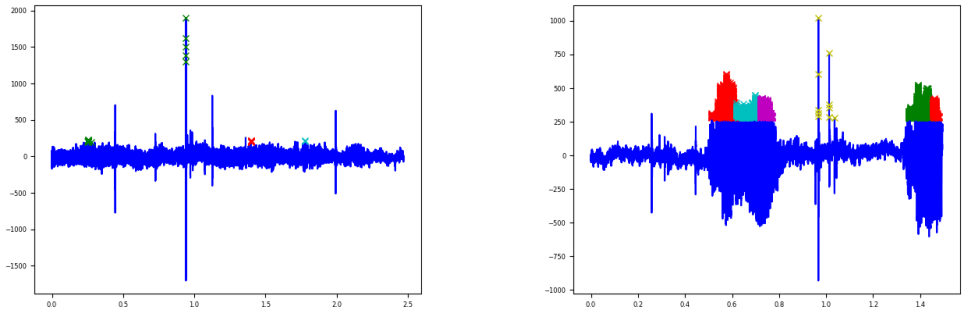
- **The noise during the noise evaluation is very high**
  If this happens, no audio peaks can be found (see **Figure 6.2**).
- **The noise during the noise evaluation is very low**
  If this happens, some audio peaks can be found, even if the user was a bot, because there is some noise during the password insertion. However these audio peaks don't have a time correspondence with the time instants stored during the password insertion (see **Figure 6.3**). The only case, in which a bot could authenticate it self, is when there is a sequence of audio peaks caused by the noise and the time between them is the same of the stored ones.
  However this event isn't very probable because it is hard that there would be an high noise only during the insertion of the password and not during the noise evaluation. If no noise evaluation was performed, the attacker could also analyse the background noise of the user and find a periodic sequence of peaks. Then the hacker could type the characters with the same time intervals discovered in the previous noise pattern. This highlights the strength and the importance of the noise evaluation performed before the insertion of the password.

The bots activity was tested only using time correspondence of AcCAPPCHA because this is the best method for the analysis of the user's activity (see **Section 6.1.1**). The tests were not performed using only character correspondence because many false positives can be found. The character correspondence were not applied after the time correspondence because the bots already fail the first verification based on time instants.



(a) Bot with `pynput` module. (b) Team Viewer.

Figure 6.2: Plot of audio during the password insertion with high noise during noise evaluation.



(a) Bot with `pynput` module.

(b) Team Viewer.

Figure 6.3: Plot of audio during the password insertion with low noise during noise evaluation.

6.1.3 Strength against known attacks

Following the analysis performed for Invisible CAPPCHA[35], I analyse the strength of AcCAPPCHA against the following attacks:

- **Replay attack**
  The message is still concatenated with a nonce and then they are signed to guarantee a client would use a nonce only once. The server still prevents this type of attacks by refusing the second message from a client with a nonce already used by him.

- **Reverse engineering attack**
  The code working on the client must be kept secure by the File System. Even if the attacker could reverse code on server, the signature of the message and the TLS communication guarantee that this attack cannot be performed.

- **Human-solver relay attack**
  AcCAPPCHA is strong against this attack because there aren't additional tasks to be performed as in classical CAPTCHAs. Hence no challenges can be sent to remote human solver.

- **Brute force and password replay attacks**
  A single call to AcCAPPCHA performs at most 3 attempts of password insertion. Hence this type of attack couldn't be performed because if invalid credentials were inserted by the user for 3 times, AcCAPPCHA immediately blocks the user.

- **Denial Of Service (DOS)**
  If a bot was detected by AcCAPPCHA for 3 times during the execution, the program will block also the next trials on the client side. Hence the server won't be overloaded, because the client will refuse next attempts without communicating any message to the server.

After the analysis of side-channel attacks, I decided also to modify the messages transmitted between the client and the server (see **Section 5.2**). All the messages, included the POST requests, are padded with space characters such that an hacker cannot understand which string was sent in every step of the communication. In fact if padding isn't applied, as explained in **Section 3.2**, the attacker could analyse the size of the packets to understand which data were exchanged by the parties.

6.1.4 Other security considerations

I analyse also other important aspects that guarantee the security of AcCAPPCHA. The first one is the choice of the HTTP POST method instead of HTTP GET method. I choose the first one because it's more secure to exchange sensible information than the second one. In fact, the HTTP GET method is associated to an URL that could be easily stored in the Browser History, even if in my application I didn't exploit the browser while the client sends the HTTP request.
Another important aspect to be considered is that the File Systems of the client and the server must guarantee that the file `block.txt`, used to manage the block period of a user, the encryption keys and the certificates must be kept secure. The File System has to guarantee the protection of the spectrogram images because the application generates them, during the insertion of the password, and it stores them in a folder. Then AcCAPPCHA accesses the images in the path, using the pre-trained network, and it extracts the features from them. Hence the File System must guarantee that the folder with spectrogram images couldn't be accessed by other applications. A path, for which the same type of protection must be ensured, is the one containing the model of the trained neural network, used for the classification of the key presses.
Another consideration is about the microphone permission that the Windows Operating System asks the user at only the first execution of AcCAPPCHA. The programs that have access to the microphone, as AcCAPPCHA, are listed in **Settings** > **Privacy** > **Permissions** > **Microphone** on Windows 10. The user could also analyse the list to find a bot that was installed by an attacker on the Operating System and it's trying to access the micro-

phone (e.g. trying to elude AcCAPPCHA verification).
I suggest every user to enable the microphone icon in **Settings** > **Personalisation** > **Taskbar** > **Turn system icons on or off** even if an attacker could disable it. In this way, every time that an application accesses the microphone resource, the user could see the icon on the taskbar if the option was previously enabled on the settings. If an attacker has remote control of the pc on which client-side AcCAPPCHA runs, the user could understand on Windows 10 that he's under attack by looking at the icon because the microphone is used by AcCAPPCHA.

6.2 Usability

The verification with deep learning techniques would be more strong if the predictions would be more accurate. A problem of this methods is that the neural network must be trained using the audio files related to the keys of a specific keyboard. This method limits the usability of the neural network to the type of keyboard used to record the training set.
To use the character correspondence on several keyboard, each producer of keyboards should record audio files produced by the key presses and he should share them. In this way the neural network could be easily trained on several keyboards, generating a trained set of weights for each of them. Then AcCAPPCHA could be extended to download the set of weights, related to the keyboard of the user, at the first execution of the application. This action could also bring to a malicious use of the downloaded data (e.g. an attacker could use them to produce an advanced key-logger). Hence the access to these datasets should be limited to AcCAPPCHA application.
I used also other input devices to test the usability of the best method used to very the identity of the user (time correspondence). At first I tested AcCAPPCHA using an external wireless keyboard (Logitech K480) and positioning it in front of my laptop. AcCAPPCHA was tested also considering the interaction of the user with the touch keyboard of Windows 10 using:

- **Mouse** (Zelotes T90)
  the time correspondence exploits the sound of each mouse click needed to select a key on the keyboard.

- **Touchpad**
  this device also guarantees to perform the time correspondence exploiting the sound of each click on the touchpad, used to select a key. The

only condition, required by AcCAPPCHA to work correctly, is that the user clicks the physical left button at the bottom side of the touchpad.

Using these three devices, the results of the time correspondence were equal to the ones obtained using the keyboard of the laptop. Hence the time correspondence is very efficient and it works well in several scenarios because it's not related to the input hardware but only to the microphone. At the contrary character correspondence is limited to the hardware characteristics of the input device.

Chapter 7

Future work

In the future, AcCAPPCHA could be tested on several Operating Systems to make sure that it works correctly on all of them. The application could still be improved by collecting more audio files, with different types of background noise, and increasing the accuracy of the classification, in particular for the one based on the spectrograms. The classification with the neural network technique could also be improved by adding a validation set to perform some advanced techniques. The security of AcCAPPCHA could be improved by managing the spectrogram images from the application without the involvement of the File System.
The main work, that could be done in the future, is the implementation of the time correspondence approach on the smartphones. We can exploit the analysis of audio signals recorded by the microphones of the mobile phones. In fact, many smartphones are now equipped with two microphones and these can be used to increase the accuracy during the research of the audio peaks. To classify a key press, the application could exploit the shape of the waves, discovering some general pattern of an audio peak, as done in the Asonov and Agrawal's work[37]. Instead of using the previous technique, AcCAPPCHA could also exploit the time difference between the audio files of the two microphones to classify a key press[38].
Moreover the voice assistants, like Amazon Echo and Google Home, are becoming increasingly widespread and they are equipped with several microphones. These devices record human activity continuously and only if the user says a specific keyword, they turn on speakers and reply to user. Hence, the audio signals could be exploited to develop new attacks and to analyse input on physical or virtual keyboards[39]. However the acoustic side-channel of the voice assistant, could also be exploited to implement a new version of AcCAPPCHA.

Appendix A

Key Map

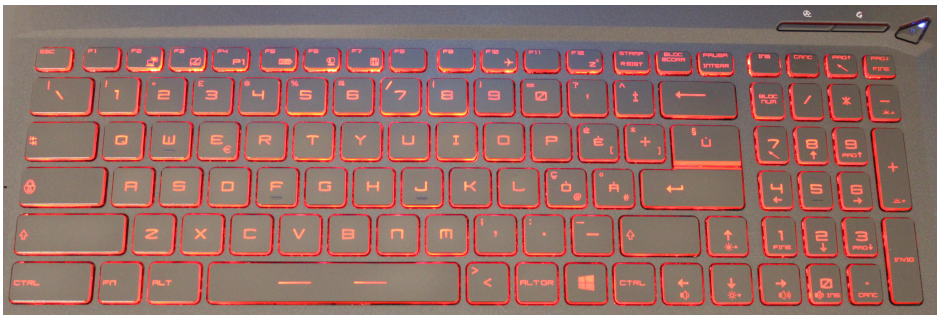
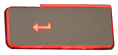


Figure A.1: Layout of the keyboard of my MSI GL63 8RD laptop.

Key	Key-logger label	Final label
	0	0
	• INSERT (Num lock on) • None (Num lock off)	0_INSERT
	1	1
	• END (Num lock on) • None (Num lock off)	1_END
	2	2
	• DOWN (Num lock on) • None (Num lock off)	2_DOWN

	3	3
	• PAGE_DOWN (Num lock on) • None (Num lock off)	3_PAGE_DOWN
	4	4
	• LEFT (Num lock on) • None (Num lock off)	4_LEFT
	5	5
	• None (Num lock on) • None (Num lock off)	5_NUM
	6	6
	• RIGHT (Num lock on) • None (Num lock off)	6_RIGHT
	7	7
	• HOME (Num lock on) • None (Num lock off)	7_HOME
	8	8
	• UP (Num lock on) • None (Num lock off)	8_UP
	9	9
	• PAGE_UP (Num lock on) • None (Num lock off)	9_PAGE_UP
	a	a
	SHIFT	ALT
	CTRL	ALT_GR
	APOSTROPHE	APOSTROPHE
	b	b

	BACKSLASH	BACKSLASH
	BACKSPACE	BACKSPACE
	c	c
	CAPS_LOCK	CAPS_LOCK
	COMMA	COMMA
	CTRL	CTRL
	CTRL_R	CTRL_R
	d	d
	DELETE	DELETE
	DOWN	DOWN
	e	e
	ENTER	ENTER
	ENTER	ENTER_R
	ESC	ESC
	f	f
	F1	F1
	F2	F2
	F3	F3
	F4	F4
	F5	F5
	F6	F6

	F7	F7
	F8	F8
	F9	F9
	F10	F10
	F11	F11
	F12	F12
		FN
	g	g
	h	h
	i	i
	INSERT	INSERT
	j	j
	k	k
	l	l
	LEFT	LEFT
	LOWER	LOWER
	m	m
	MINUS	MINUS
	MINUS	MINUS_R
	n	n
	NUM_LOCK	NUM_LOCK

	o	o
	p	p
	PAGE_DOWN	PAGE_DOWN
	PAGE_UP	PAGE_UP
	PAUSE	PAUSE
	PLUS	PLUS
	PLUS	PLUS_R
	POINT	POINT
	• DELETE (Num lock on) • None (Num lock off)	POINT_DELETE
	PRINT_SCREEN	PRINT_SCREEN
	q	q
	r	r
	RIGHT	RIGHT
	s	s
	SCROLL_LOCK	SCROLL_LOCK
	SHIFT	SHIFT
	SHIFT	SHIFT_R
	MINUS	SLASH
	SPACE	SPACE
	STAR	STAR

	t	t
	TAB	TAB
	u	u
	UP	UP
	v	v
	w	w
	SHIFT	WINDOWS
	x	x
	y	y
	z	z
	à	à
	è	è
	ì	ì
	ò	ò
	ù	ù

Table A.1: Labels seen by the keylogger and final labels manually assigned for every key press.

Appendix B

Program

In the following sections, I describe the command line parameters of the main programs, developed to perform AcCAPPCHA verification. As explained in the previous chapters, all the code was developed using the `Python` programming language.

B.1 DatasetAcquisition.py

`DatasetAcquisition.py` does several actions: the recording of the audio during the insertion of a key, the plotting of the graphics of the audio files, the extraction of the features from the audio peak in each audio files. The audio recording terminates after a number of key presses. The label related to the first key press defines the name of the subfolder of the output path where the wav file will be stored. By default the program waits only for sequences of 1 key and then stores it.

Parameter	Description
`-plot`	If specified, the program plots existing audio files
`-p`	
`-record`	If specified, the program records audio while the keylogger is going on
`-r`	
`-extract`	If specified, the program extracts features from audio already recorded
`-e`	

`-zoom` `-z`	If specified, the program plots the audio peaks of the input files in graphic images, showing only the signal in the neighbourhood of the peak.
`-file` *path* `-f` *path*	It specifies the *path* of the file that you want to plot or from which you want to extract features
`-dir` *path* `-d` *path*	It specifies the *path* of the folder where: • **-r option** there will be stored audio files acquired from the work of both the recorder and the keylogger • **-p option and -e option** there are already the recorded audio files on which the extraction of the features or the plotting will be performed
`-out` *path* `-o` *path*	It specifies the *path* of the output folder where the graphics will be stored as images (if `-p/-plot` selected)

B.2 AcCAPPCHA.py

`AcCAPPCHA.py` runs at the client side and it performs the verification of the user identity, using either time or character correspondence. Then it sends the results to the server and communicates with it during the authentication phase.

Parameter	Description
`-dir` *path* `-d` *path*	It specifies the *path* of the folder with the 3 subfolders: `touch/`, `touch_hit/` and `spectrum/`. Each of them contains a folder called `model/` that contains the info about the pre-trained network that classifies the keys of the keyboard. It's mandatory if -deep/-dl option is specified.
`-time` `-t`	If specified, AcCAPPCHA performs the time correspondence.
`-deep` `-dl`	If specified, AcCAPPCHA performs the character correspondence.

`-plot` `-p`	If specified, AcCAPPCHA plots the graphics of the peaks in the audio signal recorded during the password insertion.
`-debug` `-dbg`	If specified, AcCAPPCHA shows debug information about the verification phases

B.3 Authentication.py

`Authentication.py` runs forever at the server side. It performs the verification of the message received by the client and it accesses the database during the authentication phase.

Parameter	Description
`-debug` `-dbg`	If specified, the program shows debug information about the remote client

Bibliography

[1] Sarika Choudhary, Ritika Saroha, Yatan Dahiya, and Sachin Choudhary, "Understanding CAPTCHA: Text and Audio Based Captcha with its Applications" in *International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3(6), pp. 106-115, 2013.*

[2] Darko Brodić, Alessia Amelio, Radmila Janković, "Exploring the influence of CAPTCHA types to the users response time by statistical analysis" in *Multimedia Tools and Applications, vol. 77, pp. 12293–12329, 2017.*

[3] Walid Khalifa Abdullah Hasan, "A Survey of Current Research on Captcha" in *International Journal of Computer Science Education in Schools (IJCSES), vol. 7, pp. 141–157, 2016.*

[4] Ruti Gafni, Idan Nagar, "CAPTCHA – Security affecting user experience" in *Issues in Informing Science and Information Technology, vol. 13, pp. 63-77, 2016.*

[5] G. Sauer, J. Holman, J. Lazar, H. Hochheiser, and J. Feng, "Accessible privacy and security: A universally usable human-interaction proof tool" in *Univers. Access Inf. Soc., vol. 9, no. 3, p. 239–248, Aug. 2010.*

[6] Jennifer Tam, Jiri Simsa, David Huggins-Daines, Luis von Ahn, Manuel Blum, "Improving Audio CAPTCHAs" in *Symposium On Usable Privacy and Security (SOUPS), 2008.*

[7] William Aiken, Hyoungshick Kim, "POSTER: DeepCRACk: Using Deep Learning to Automatically CRack Audio CAPTCHAs" in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS), 2018.*

[8] Jennifer Tam, Jiri Simsa, Sean Hyde, Luis von Ahn, "Breaking audio CAPTCHAs" in *Advances in Neural Information Processing Systems, pp. 1625–1632, 2009.*

[9] Manar Mohamed, Song Gao, Nitesh Saxena, Chengcui Zhang, "Dynamic cognitive game captcha usability and detection of streaming-based farming" in *The Workshop on Usable Security (USEC), co-located with NDSS, 2014.*

[10] Ved Prakash Singh, Preet Pal, "Survey of Different Types of CAPTCHA" in *International Journal of Computer Science and Information Technologies (IJCSIT), vol. 5(2), 2014.*

[11] Goswami G, Powell BM, Vatsa M, Singh R, Noore A, "FaceDCAPTCHA: face detection based color image CAPTCHA" in *Future Generation Computer Systems, vol. 31(2), pp. 59–69, 2014.*

[12] Wen-yao Lu, Ming Yang, "Face Detection Based on Viola-Jones Algorithm Applying Composite Features" in *International Conference on Robots & Intelligent System (ICRIS), pp. 82-85, 2019.*

[13] Brian M. Powell, Abhishek Kumar, Jatin Thapar, Gaurav Goswami, Mayank Vatsa, Richa Singh, Afzel Noore, "A multibiometrics-based CAPTCHA for improved online security" in *IEEE 8th International Conference on Biometrics Theory, Applications and Systems, 2016.*

[14] Carlos Javier Hernandez-Castro, Arturo Ribagorda, "Pitfalls in CAPTCHA design and implementation: The Math CAPTCHA, a case study" in *Computers & Security, vol. 29(1), pp. 141-157, 2010.*

[15] Luke Wroblewski, "A Sliding Alternative to CAPTCHA?", *2010.*

[16] Silky Azad, Kiran Jain,, "Captcha: Attacks and weaknesses against OCR technology" in *Global Journal of Computer Science and Technology, 2013.*

[17] E. Bursztein, M. Martin, J. Mitchell, "Text-based CAPTCHA strengths and weaknesses" in *Proc. 18th ACM Conference on Computer and Communications Security (CCS), pp. 125–138, 2011.*

[18] M. Shirali-Shahreza and S. Shirali-Shahreza, "Motion CAPTCHA" in *2008 Conference on Human System Interactions, Krakow, pp. 1042-1044, 2008.*

[19] Kameswara Rao Kavya Sri, Gnana Sai, "A Novel Video CAPTCHA Technique to Prevent BOT Attacks" in *International Conference on Computational Modeling and Security (CMS 2016), Procedia Computer Science, vol. 85, pp. 236–240, 2016.*

[20] K. A. Kluever, R. Zanibbi, "Balancing usability and security in a video captcha" in *Proceedings of the 5th Symposium on Usable Privacy and Security, 2009.*

[21] Omar Ahmed Hedaia, Ahmed Shawish, Hana Houssein, Hala Zayed, "Bio-CAPTCHA Voice-Based Authentication Technique for Better Security and Usability in Cloud Computing" in *International Journal of Service Science Management Engineering and Technology, vol. 11(2), pp. 59-79, 2020.*

[22] Erkam Uzun, Simon Chung, "rtCaptcha: A Real-Time Captcha Based Liveness Detection System" in *The Network and Distributed System Security Symposium (NDSS), Georgia Institute of Technology, 2018.*

[23] Vinay Shet, "Are you a robot? Introducing "No CAPTCHA reCAPTCHA"", *2014.*

[24] TehnoBlog, "Google no Captcha + INVISIBLE reCaptcha–First Experience Results Review", *2019.*

[25] Suphannee Sivakorn, Jason Polakis, Angelos D. Keromytis, "I'm not a human: Breaking the Google reCAPTCHA" in *Black Hat, pp. 1–12, 2016.*

[26] Meriem Guerar, Alessio Merlo, Mauro Migliardi, "Completely automated public physical test to tell computers and humans apart: A usability study on mobile devices" in *Future Generation Computer Systems, vol. 82, pp. 617–630, 2018.*

[27] Narges Roshanbin, James Miller, "A survey and analysis of current CAPTCHA approaches" in *Journal of Web Engineering, vol. 12, issue 1–2, pp. 1–40, 2013.*

[28] R. Spreitzer, V. Moonsamy, T. Korak and S. Mangard, "Systematic Classification of Side-Channel Attacks: A Case Study for Mobile Devices" in *IEEE Communications Surveys & Tutorials, vol. 20, no. 1, pp. 465-488, 2018.*

[29] A. Nahapetian, "Side-channel attacks on mobile and wearable systems" in *13th IEEE Annual Consumer Communications & Networking Conference (CCNC), pp. 243-247, 2016.*

[30] Paul Kocher, Joshua Jaffe, Benjamin Jun, "Introduction to Differential Power Analysis and Related Attacks" in *Cryptography Research, 1998.*

[31] J. W. Bos, C. Hubain, W. Michiels, and P. Teuwen, "Differential Computation Analysis: Hiding Your White-Box Designs is Not Enough" in *Cryptographic Hardware and Embedded Systems – CHES, ser. LNCS, vol. 9813, Springer, pp. 215-236, 2016.*

[32] S. Skorobogatov, "The Bumpy Road Towards iPhone 5c NAND Mirroring" in *arXiv ePrint Archive, Report 1609.04327, 2016.*

[33] Shuo Chen, Rui Wang, XiaoFeng Wang, Kehuan Zhang, "Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow" in *2010 IEEE Symposium on Security and Privacy, pp. 191-20, 2010.*

[34] J. Zhang, X. Zheng, Z. Tang, T. Xing, X. Chen, D. Fang, R. Li, X. Gong, and F. Chen, "Privacy Leakage in Mobile Sensing: Your Unlock Passwords Can Be Leaked through Wireless Hotspot Functionality" in *Mobile Information Systems, vol. 2016, pp. 8793025:1–8793025:14, 2016.*

[35] M. Guerar, A. Merlo, M. Migliardi, F. Palmieri, "Invisible CAPPCHA: A usable mechanism to distinguish between malware and humans on the mobile IoT" in *Computers & Security, vol. 78, pp. 255–266, 2018.*

[36] J. V. Monaco, "SoK: Keylogging Side Channels" in *2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2018, pp. 211-228.*

[37] D. Asonov, R. Agrawal, "Keyboard acoustic emanations" in *IEEE Symposium on Security and Privacy. Proceedings. 2004, Berkeley, CA, USA, 2004, pp. 3-11.*

[38] I. Shumailov, L. Simon, J. Yan, R. Anderson, "Hearing your touch: A new acoustic side channel on smartphones" in *arXiv, 2019.*

[39] A. Zarandy, I. Shumailov, R. Anderson, "Hey Alexa what did I just type? Decoding smartphone sounds with a voice assistant" in *arXiv, 2020.*

Acknowledgements

I would like to express my great appreciation to Professor Migliardi for the availability every time I need some help and to accept me to be my supervisor. I learnt so much thanks to his tips due to his big experience in the software development and security threat. Thanks to our conversations I fell in love with every aspect of my work. I tried to go in deep on every concept he mentioned because of the curiosity he leaves me every time we talk about cyber security.

I would like to express my great appreciation to Professor Nanni for the hints he gives me during the development of the application.

I would like to express my gratitude to Dr. Guerar to introduce me in the world of the CAPTCHAs and for her availability every time I need. I learnt very much from her works about cyber security.

I would like to express my great appreciation to the University of Padova for the study path I follow. The uncertainty about the future and the idea of being far from needed cyber security skills have become a stimulus to improve myself. I learn a lot and I got hooked on the programming, starting from zero level of it, thanks to the professors' professionalism and knowledge. During the last five years, I've changed my self and now I spend a lot of my free time programming. Thanks to the University because professors follow my thirst of knowledge and I grew up during the last five years.

I would like to express my gratitude to my family that taught me to never give up.

Thanks to Cristina, to taught that no goals can be reached without someone, near to you, that believes in you and makes you happy.

Thanks to Francesca, because even if we had many commitments we have

always found 5 minutes to stay together and to support the other one in his/her choices.

Thanks to Davide to help me improving my IT skills, with new stimulating ideas, and to be near me when I needed it.

Thanks to everyone that has stayed near to me and that I meet during these years. Everyone left me something and helped me to delete every worry and to have fun outside the study hours.