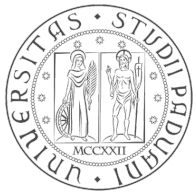


University of Padova

Engineering Departement

Computer Engineering Master Degree

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 2021-2022

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

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, gathering useful information about user credentials or pretending to be a human interaction with Web application. Hence the term *"bot"* is an abbreviation of the words "software robot".
The CAPTCHAs are traditionally used in Web applications for[1]:

- **Online Polls**
  CAPTCHAs prevent the creation and the submission of a large number of votes, favouring a party.
- **Protecting Web Registration**
  CAPTCHAs prevent the creation of free mail account to bot instead of human users. The goal of the use of CAPTCHAs is to remove the possibility that the hacker could take advantages from the large amount of registrations.
- **Preventing comment spam**
  CAPTCHAs prevent the insertion of a large amount of posts made by bot on pages of social platforms or blogs.
- **Search engine bots**
  CAPTCHAs are used to guarantee that a website should be unindexed to prevent the reading of the page through search engine bots. The CAPTCHAs are added because the html tag, used to unindex the web page, doesn't guarantee unindexing.
- **E-Ticketing**
  CAPTCHAs prevent that a big events would sell out minutes after

tickets become available. In fact ticket scalpers that make large number of ticket purchases for big events.

- **Email spam**
  CAPTCHAs are used to verify that a human has sent the email.

- **Preventing Dictionary Attacks**
  CAPTCHAs prevent bot to guess the password of a specific user. The hacker could guess the password, taking it from a dictionary of passwords. 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.

- **Verifying digitized books**
  ReCAPTCHA can verify the contents of a scanned piece of paper analysing responses in CAPTCHA fields. A computer cannot identify all the words from a digital scan.
  The application submits two words to the user in the CAPTCHA challenge: the first one that the machine has already recognized and the other for which it can correctly 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 added to other users' challenges. If the application receives enough responses with the same typed word related to the unknown word, the program extablishes that typed word is the CAPTCHA is related only to the first word and the challenge related to the second word is exploited by the application to scan digitally the paper.

Another useful application of CAPTCHA is the support to the authentication process. This application is going to be analysed in details in the next chapters, looking at the authentication from smartphone.
In **Chapter 2** there is a description of the state of art of CAPTCHA, looking at types of CAPTCHA and the related tests from which this challenge is born.
In **Chapter 4** Invisible CAPPCHA is described in details.
DESCRIPTION OF THE CONTENT OF THE CHAPTERS

Chapter 2

State of the Art

CAPTCHA takes inspiration and is related to three main elements[2]:

1. **Turing test**
   it's used to determine how much a machine can think like a human. The test is made by three figures: a human examiner, an human and a machine. The examiner asks some questions to both other two figures and, after a fixed amount of time, evaluates if the two answers are different or not.
   If they are similar w.r.t. the point of view of the examiner, the machine is an AI (Artificial Intelligence) similar to an human. The test is very important if the answers have many possibilities.

2. **Human-Computer Interaction (HCI)**
   according to cognitive psychology studies, a human process data in a specific way and this test evaluates the interaction between humans and machines. The HCI model is divided into five levels:

   - task level
   - semantic level
   - syntactic level
   - interactive level
   - a level of physical devices

   Then the obtained information is processed by:

   - reasoning
   - problem solving
   - skill acquisition

- error

3. **Human Interactive Proof (HIP)**
   it's used to make differentiation between machine and human users and computer user programs. The test require a type of interaction, that is simple to be done by human instead of bot. The main goals of this type of 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 has the test program that is subjected to the human and the computer. As a result, only a specific group of humans can positively solve the test and then the test results can be validated by the computer.

In order to guarantee a good level of security, a CAPTCHA has to satisfy the following requirements:

- The solution to the CAPTCHA isn't conditional and shouldn't depend on the user's language and/or age.
- The solution of the CAPTCHA must be easy for the humans and hard for the bots. Hence, humans in no longer than 30 seconds with very high success rate
- The creation of the CAPTCHA must not disturb the user privacy (not linked to the user).

2.1 Traditional CAPTCHAs

The traditional CAPTCHAs are based on the knowledge and correct insertion of solution by the user. These CAPTCHA schemes are designed to exploit character recognition, image understanding and speech recognition to guarantee that the challenges will successfully block bots.
The main types of these 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 but the most frustrating CAPTCHA is the text-based one[4]. A summary of usability and security issues is shown in **Table 2.2** .

2.1.1 Audio-based CAPTCHAs

This type of CAPTCHAs asks the user to type the words listened by an audio file (see **Figure 2.1**). It's developed for vision-impaired users. It usually has problems in usability related to the language dictionary, from which words are taken, and the similarity of the sound between several words. It has been proofed that this CAPTCHA is a hard task even for blind users, in fact only 46% of the challenges were solved by participants to an experiment[5].
One of the most popular 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 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].

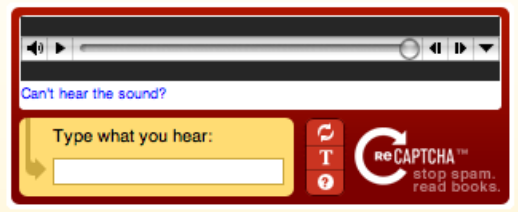


Figure 2.1: Example of audio-based CAPTCHA.

2.1.2 Game-based CAPTCHAs

This type of CAPTCHAs performs the verification of the user nature through a set of several kind of games (see **Figure 2.2**). The strength of this CAPTCHAs is relative to the comprehension phase of the rules that only humans can perform.
This type of CAPTCHAs is called *Dynamic Cognitive Game (DCG)* is usually developed using Flash and HTML5 with JavaScript. These technologies download the game code to the client and execute it locally.
The only difficult for the bot to attack the challenge is the encryption/obfuscation of the code. This strategy prevent the store of the code onto 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.

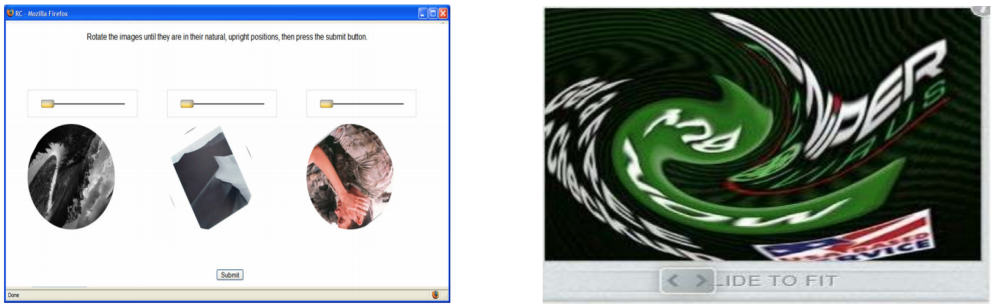
2.1.3 Image-based CAPTCHAs

Image-based CAPTCHAs require to understand a written text describing a task that needs an image evaluation to pass the test. This type of CAPTCHAs can be categorized into the following classes, looking to 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.

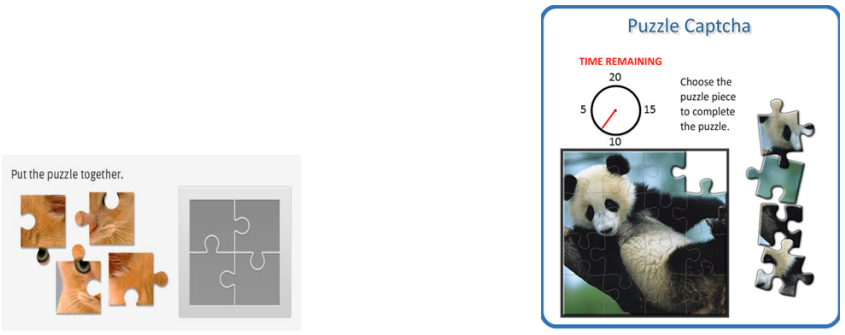
- **Sliding image-based CAPTCHAs**
  this type of CAPTCHAs asks the user to use the slider to solve an image-based challenge such as adjusting the orientation of an image, selecting the correct form of an image, or moving a fragment of an image to the correct location (see **Figure 2.4**).



(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 variant of the puzzle-based CAPTCHA in which needs to insert only some pieces of the puzzle instead of completing the whole puzzle (see **Figure 2.5b**).



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

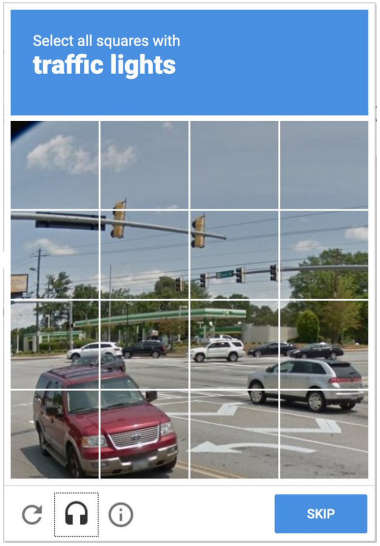
Figure 2.5: Examples of puzzle-based CAPTCHAs.

- **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 implemented in different ways, for example:

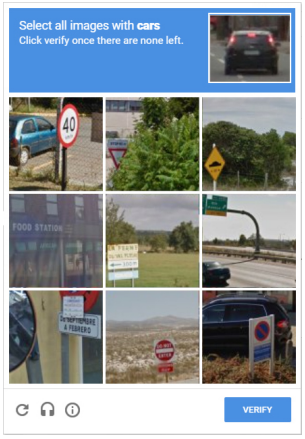
  – An image is divided into a set of sub-squares and each of them is a candidate image (see **Figure 2.6a**

  – 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]. 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. 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 have many problems differentiating real and non-real photographs of faces.
  Face, fingerprint and eye detections in images remain also a difficult challenge to be performed by computers. For this reason these results were used to develop a new variant 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 movement or swiping gesture

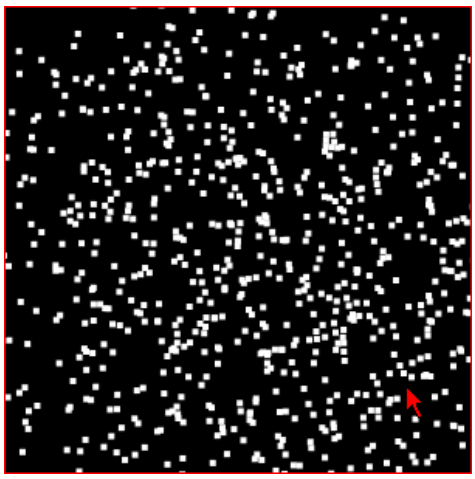


Figure 2.7: Example of interactive-based CAPTCHA.

2.1.4 Math CAPTCHAs

Looking to an 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 Bit Generator Service (QRBGS) sign-up Web Page[14] (see **Figure 2.9**). This type of CAPTCHA asks user to solve an advanced math expression. It prevents the use of free or commercial OCRs because many mathematical symbols are not considered in their detection algorithm. However, it's vulnerable to side-channel attack [14].

Hence many math symbols are wrongly translated by bot programs and the challenge is very secure. The only problem is that this CAPTCHA is very complex for normal users and many of them could not solve the challenge correctly.

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

Figure 2.8: Example of arithmetic CAPTCHAs.

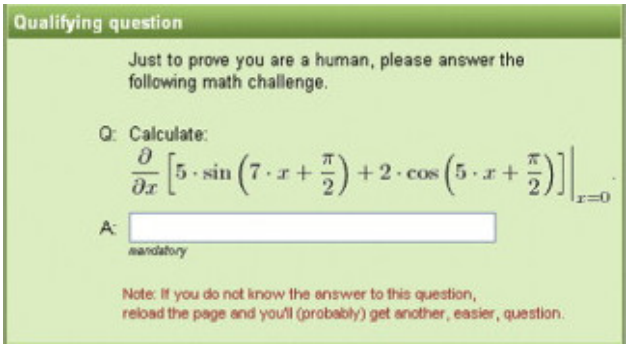


Figure 2.9: Example of Quantum Random Bit Generator Service (QRBGS) sign-up Web Page [14].

2.1.5 Slider CAPTCHAs

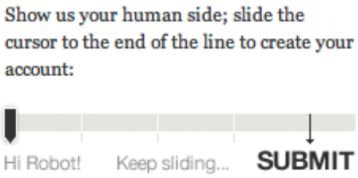
Slider CAPTCHAs only asks users to move the slider across the screen. Hence, image recognition is not part of the challenge to be classified as a human.
The most popular CAPTCHAs are the following:

- **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**).

Different variations 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 wrapped 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 shown sequence and then type them inside a text-field. The text-based CAPTCHAs can be also classified in three main classes looking to the type of wrapped characters:

- **2D**
  the digits are wrapped on a 2D plane, parallel to the screen plane
- **3D**
  the digits are wrapped on a 3D plane oriented in the space and then a 2D image is taken from a specific point of view

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

- 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 techniques, to be considered. For each of them, there is usually a solution in the design phase of the CAPTCHA that reduces the possibility that the challenge would be broken by a bot[17].

2.1.7 Video-based CAPTCHAs

This type of CAPTCHA is not very common because of the weight of the file to be downloaded[3]. The traditional video-based CAPTCHA is composed

Figure 2.11: Example of text-based CAPTCHA.

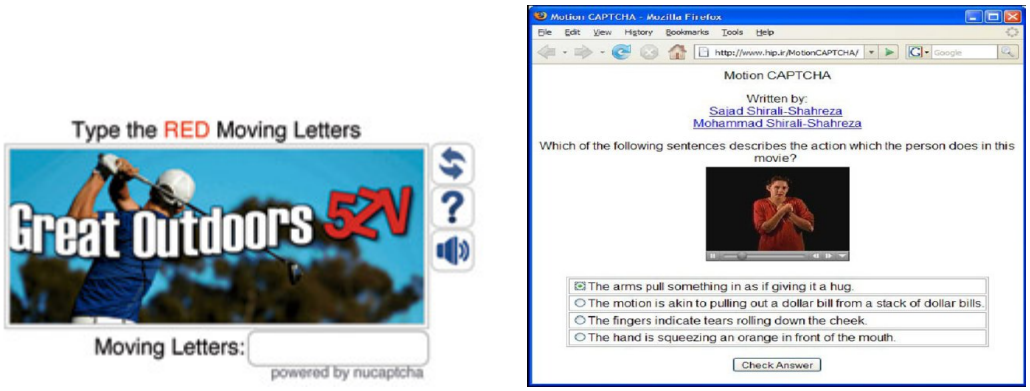
by a video in which a sliding text is shown (see **Figure 2.12a**). The user needs to type this message in a text field to pass the challenge. Some implementations of this type of CAPTCHAs are vulnerable to machine learning attacks.
Another variant of this CAPTCHA is the *Motion CAPTCHA*[18], developed 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 depends on the relationship between the multiple choices submitted to the user[19].
A similar implementation of the previous variant, it's the one developed by Kluever et al. in which the user watches a video and needs to write three words that describe what he sees[20]. The same authors also performed a tag frequency-based attack to evaluate the security of their CAPTCHA schemes achieving a success rate of 13%.

2.2 Modern CAPTCHAs

The type of CAPTCHAs and authentication mechanisms described in the following section are far from traditional CAPTCHAs and aren't based on cognitive knowledge of the human user but on other parameters, such as behavioural analysis and sensors readings. In the following sections there is a summary of the most known CAPTCHA schemes of this type.

2.2.1 Biometrics-based CAPTCHAs

The most known authentication mechanisms, that use biometric parameters of the user, are based on the following CAPTCHA schemes:



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

Figure 2.12: Examples of video-based CAPTCHAs.

- **Bio-CAPTCHA voice-based Authentication**
  This authentication method was developed starting from good results reached in the authentication phase of cloud systems (Alexa for Amazon, Siri for Apple, Cortana for windows)[21]. This particular implementation uses a random voice-based password challenge. This password changes at every login of the user and this method uses CAPTCHA challenge to provide unpredictability and ambiguity to the authentication process. The experiments reveals that unauthorized access probability decreases, while it keeps high usability because it needs only a mic.

- **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 in the file, there is the expected user performing the required action.
  This implementation of CAPTCHA solves many problems of similar CAPTCHAs that are also based on liveness mechanisms and video capture. The attackers can extract patterns or features from existing or captured images and embed them into a new generated video in attack in the compromised device.
  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, looking to all possible Computer Vision attacks.

2.2.2 Behavioural-based CAPTCHAs

In 2014 Google announced that today's Artificial Intelligence can solve 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 in 2015 a new CAPTCHA system that is 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 user as in **Figure 2.13** (or image-based CAPTCHA on mobile devices)

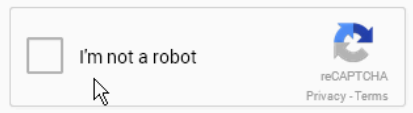


Figure 2.13: Example of Google no CAPTCHA checkbox.

  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 users click, how long they linger over a checkbox). Then the program performs an *advanced risk analysis*, by looking results of first step but also spam traffic and passed/failed CAPTCHAs. It understands in this way if the test is passed or not.
  The tests done confirms that this phase was very inefficient and many times the first layer failed even if a human user was performing correctly the task. A problem of this type of CAPTCHA is that many attacks exploits the image-based CAPTCHA and text-based CAPTCHA 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*, adding the option to bind directly to the form's submit element[24]. It usually requires the use of cookies, used 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 page of app or 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. With constantly analyzing human behavior, mouse movements, typing speed and other features incorporated into NO CAPTCHA technology, Google collected enough sample data to perfectly fine-tune their Google invisible reCAPTCHA v2.0 with this new version. This type of CAPTCHA uses Artificial Intelligence and Machine Learning probability scores, hostname, timestamp and anction validations.
  Google removes image recognition and looking only the score, it evaluates if the user is a human or a bot. The main difference w.r.t. previous versions is that this CAPTCHA returns a probability score (*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 the the site is under attack and what actions need to be performed.
  Some characteristics related to this version of *Invisible ReCAPTCHA* are:
  - 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

This type of devices have natively many sensors, like gyroscope and accelerometer, and the CAPTCHA schemes, described in the following sections, exploit their presence to improve 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*) present in the device. It prevents brute force, side channel and recording attacks. The usability results are good and then some of the comments made by users were 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 for breaking CAPTCHAs is organized into the following phases[27]:

1. **Pre-processing phase**
   In this phase, several techniques are applied to remove background, 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 facilitate 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.

Many CAPTCHAs have yet the following known issues:

- **Session issue**
  Some types of CAPTCHAs have a big issue because they don't destroy the session, after the correct answer is inserted by the user[1].
  Hence, the hacker can crack following 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 CAPTCHA involves the use of remote bot or human solver. Traditional CAPTCHA schemes are vulnerable to this type of attacks.
  Invisible reCAPTCHA and other academic proposals haven't been attacked yet, but works over thousands of different IP addresses and simulate the human behavior. Sensor-based CAPTCHAs are also vulnerable to solver relay attacks. An exception of this issue is the *Invisible CAPPCHA*, that will be analysed in following sections and it's designed to block this type of attacks.

- **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 highlighted in behavioural and sensor-based CAPTCHAs.

- **User's privacy**
  Sensor-based and behavioural CAPTCHAs usually send useful infor-

mation 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 to users' private data.
In some CAPTCHAs, the information are 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].

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 2.2: 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 user private data. This type of extra information is usually: timing information, power consumption, electromagnetic radiations, sound and so on.
The first types of side-channel attacks requires 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[32].
Side-channel attacks can be classified as:

- **active**
  the hacker influences the behaviour of the victim's device
- **passive**
  the attacker only analyses the leaking information

Another categorization can be the following one:

- Local attacks
- Vicinity attacks
- Remote attacks

In the following sections there is a survey of the most popular attacks, organized with respect to the previous classifications (see **Table 3.1**). The main sensors, that are usually exploited by an hacker to obtain side-channel information on mobile devices, are[31]:

- 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)

	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 3.1: Survey of the most popular side-channel attacks[32].

3.1 Local side-channel 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 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 attacks[29]:
  - ***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)***
    these attacks collect data and then makes statistical analysis and error correction techniques from data to extract information correlated to secret keys.
  - ***High Order DPA (HO-DPA)***
    While DPA obtains information across a single event, HO-DPA correlates between multiple cryptographic sub-operations.
- ***Electromagnetic Analysis Attacks***
  the attacker can analyse indirectly the power consumption by accessing electromagnetic. This type of attacks depends on the used instruments (e.g. EM probes) and on the analysed location of the chip, affecting 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. The attacker needs to have full control over the target device and the execution environment. Then using binary instruments, he can control the intermediate state or memory operations (e.g. reading/writing operations)[33].
- ***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 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). To do it, he needs to use an external clock source.

- ***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, causing malfunctioning of the hardware. Temperature higher than the maximum one, supported by hardware, causes faults in memory cells. Temperature too lower 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 the Apple case[?].

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 to exploit reserved data because each web page has a distinct size, loads resources of different sizes. Hence the attacker can fingerprint the page even if HTTPS protocol is used.
  Another cause of these attack 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[28].

- ***USB Power Analysis***
  the attacker can modify USB charging stations for mobile devices to obtain analysis about power consumption and 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 pattern in CSI series. For example the attacker can exploit these variations to unlock patterns on smartphones[35].

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 large amount of information for each process running on Linux File System, by looking to the content of the files, reported in **Table 3.2** .

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 3.2: Survey of the most popular side-channel attacks[32].

- ***Data-Usage Statistics***
  the attacker can access to information about incoming and outgoing network traffic for 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 sensors.

- ***Fingerprinting Devices/Users***
  the attacker can obtain identity of the device and the user and 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 gyroscope to obtain information about human speech near to the device.

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

Active attacks

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

Chapter 4

Invisible CAPPCHA

The *Invisible CAPPCHA* is an evolution of CAPPCHA, in terms of usability[36]. The main difference with respect to CAPPCHA is that the challenge isn't explicitly submitted to user but it's hidden behind the PIN authentication phase. This type of challenge works only on smartphones as its ancestor. This CAPTCHA is a method developed in 2018 and based on motion side-channel information, obtained by sensors on mobile device. The main steps that this CAPTCHA follows are:

1. Motion detection
2. Communication between Client and Server

In fact, the micro-movements of the device, generated by the interaction of the user with the touch-screen, are evaluated by the *Secure Element* (*SE*). Then credentials are shared with the remote Service Provider if the input is inserted by a human or not. The Invisible CAPPCHA is very effective as support of Password-based authentication methods.

4.1 Motion detection

The accelerometer of the device 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 acceleration over the z-axis over by comparing them to thresholds and 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 only along Z axis or along more than one axis (see **Figure 4.1**). Instead the finger tap event creates accelerations similar and higher on Z-axis (see **Figure 4.2**).
In Invisible CAPPCHA, the difference between built-in vibration and tap acceleration is evaluated by a simple algorithm. This algorithm relies on negative and positive peaks which are detected by comparing acceleration along Z axis against predefined thresholds. The user's tap cannot be simulated by malware using the vibration motors.
The most important requirement is that Invisible CAPPCHA uses a Secure Element that embedded the accelerometer, blocking the access to the sensor by malicious code. Nowadays there exists a smart card, called SIMSense, that already integrates motion sensor.

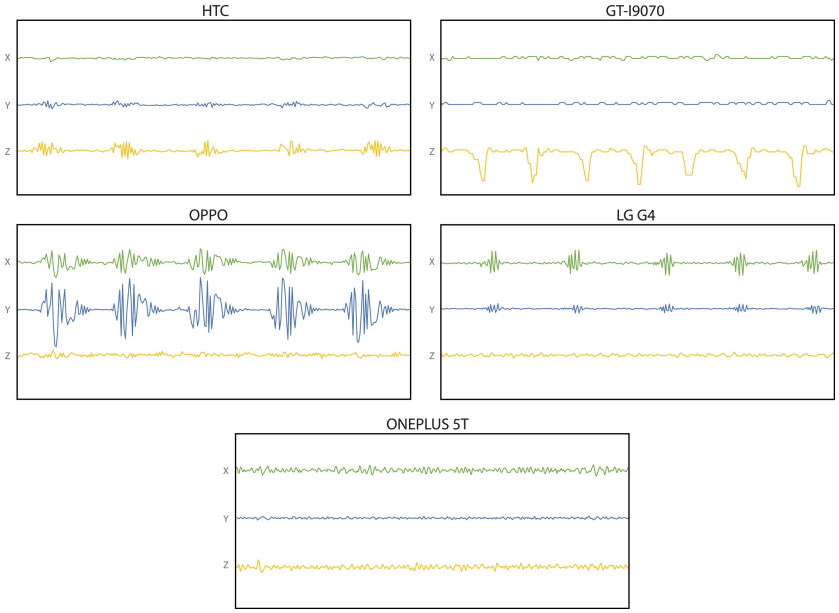


Figure 4.1: Example of accelerations caused by smartphone built-in vibration.

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 con-

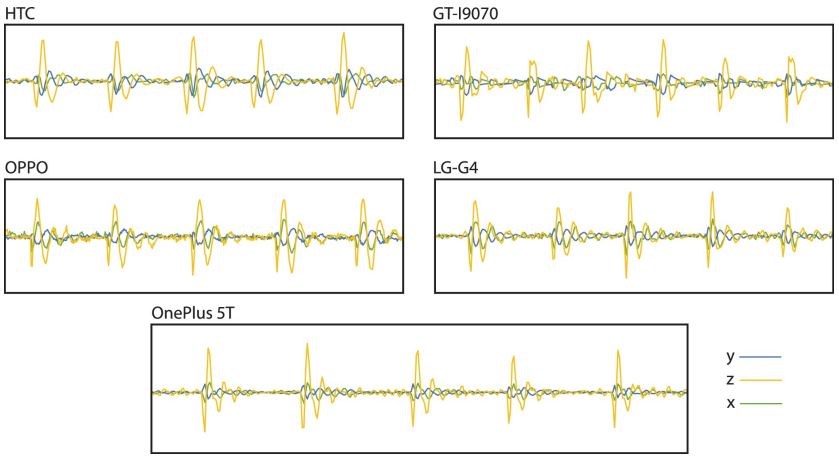


Figure 4.2: Example of accelerations caused by finger tap detection.

sidered valid, generated by a human, otherwise the algorithm tells that the input was generated by a malware.
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 sent message 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)

This type of encryption works similarly to RSA but it's based on elliptic cryptography and uses keys of smaller sizes. The algorithm is divided in two phases:

- **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 the curve, G base point of prime order on the curve and n is the multiplicative order of G for which $n\ x\ G = O$.
  Alice generates a private key d_A in the range $[1, n-1]$ and a public key $Q_A = d_A\ x\ G$, where x is scalar multiplication of a point of the curve. Alice needs to perform **Algorithm 1** to sign a message.

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

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
    $r \leftarrow x_1 \ mod \ n$
    $s \leftarrow k^{-1}(z + rd_A) \ mod \ n$

In Invisible CAPPCHA the message m is bitwise concatenated with a signed unique value, nonce n, so the signed message sent to the server is *(r, s, m, n)* and not *(r, s, m)*.

4.3 Security analysis

After the sign of the verification, 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, it can't brute force the password because of limited number of permitted attempts. If the number of attempts is higher than the maximum one, the attacker is performing a Denial Of Service (DOS) attack. The attacker can also try to steal the password through side-channel information.

4.3.1 Strength against popular attacks

The most popular attacks, that have been analysed, are[36]:

- **Replay attack**
  Because the message is signed together with a nonce, an attacker can't

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 ← true
\\Valid coordinates
if Q_A = O then
    check ← false
\\Element of the curve
if Q_A ∈ curve then
    check ← false
\\Correctness of order
if not n x Q_A = O then
    check ← false
```

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 of the attacker.

- **Reverse engineering attack**
  Even if the attacker can de-obfuscate the code of the application running 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 considers it 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 tries more than the maximum amount of attempts of passwords can do a Denial Of Service (DOS) of the Secure Element.

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

To prevent this attack, the Secure Element can block access to itself if three invalid passwords inserted by a human. If an invalid password is inserted by a malware, detected by Invisible CAPPCHA, the Secure Element is blocked.

Chapter 5

AcCAPPCHA

AcCAPPCHA is a verification that works like a key-logger. This type of programs is usually malicious and intended to be used by an attacker to acquire information about the user's activity. This application analyses the sequence of keys inserted by exploiting side-channel information. Its implementation depends on the party, that the hacker wants to attack[37], and that could be:

- **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 analysis of signals coming from 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 this attacks.

- **The network**
  these attacks exploit the packets exchanged in the client-server communication. 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.

Analysing the previous key-logger based on side-channel information, attacks mentioned in **Chapter 3** and the structure of Invisible CAPPCHA in Section **Section 4** , I design this new implementation of CAPTCHA. It exploits acoustic side-channel of microphone to implement a particular type of keylogger that ensures that Authentication phase would be performed by a human user.
The whole implementation was created 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 evaluation of signal, detected by some sensor (motion sensors for Invisible CAPPCHA and microphone for AcCAPPCHA). With respect to Invisible CAPPCHA the program can perform also a classification of the audio signal using neural networks. The two phases of the verification are:

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

In the second phase, the username and the password of the user will be signed through ECDSA and sent by client to the authentication service if and only if the insertion was performed by a human.

5.1 Evaluation of the user's activity

The CAPPCHA records two audio signals: the first one created during the insertion of the password by the user and the second one created before this activity for noise evaluation. The first signal is The second signal is exploited to evaluate a noise threshold useful for the computation of amplitude peaks in the first audio. During the insertion of the password, the instants of the time when each character was typed by the user are stored.

5.1.1 Time correspondence

Before asking the user to insert the password, the program records an audio file of 1 second, called `noise signal`, from the built-in microphone of the laptop. The remaining verification procedure is performed by two threads simultaneously, during password insertion. The first one is continuously waiting for the insertion of a character of the password by the user until he types CARRIAGE RETURN ('\r').
Immediately after a key is pressed, the time instant of this action, related to the Epoch of the PC, is stored. The sequence of time instants stored

by the thread is called $x = (x_0, ..., x_{|password|-1})$. The second thread records an audio signal, called `user activity`, using the same hardware previously mentioned. This task begins its activity before the request of the password to the user and would end after the moment in which the first thread detects a CARRIAGE RETURN. Then the application removes the last 200 ms of this audio signal to be sure that the *CARRIAGE RETURN* peak isn't included in recorded audio file.
From now on, the application has all 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 time instants of the peaks in the signal recorded in parallel to the insertion of the password and the time instants manually stored for each character.
In particular `noise signal` will be analysed by finding its maximum value, called $thresh_N$, and then `user activity` will be analysed by considering only the samples with values higher than $thresh_N$. These sample will be grouped in several disjoint windows of maximum width equal to 5 ms. For each group i, the application finds the sample with the highest value, $peak_i$. For example, given *the sampling period or interval* t_s and a specific group of samples:

$$x_i = (x_t, x_{t+t_s}, ..., x_{t+\lceil \frac{5ms}{t_s} \rceil * t_s})$$

the application computes $t_i = argmax(x_i)$.
Given the sequence of computed time instants relative to peaks of each group $t = (t_0, t_1, ..., t_{n-1})$, n number of windows and $|password|$ the size of the password, there is a **time correspondence** if if there exists a subset of it $t^* = (t*_0, ..., t*_{|password|-1})$ that matches with the sequence of time instants stored during the password insertion. The algorithm used to find a time correspondence is reported next:

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

```
1  y = (y_0, y_1, ..., y_{|password|-1}) where y_i = x_i − x_0
2  if n < |password| then
3
4      Number of found peaks lower than number of characters of the password
5      return false
6  //Search of subsequence
7  for i ← 0 to n − 1 do
8      if (n − i) < |password| then
9          //Not enough peaks from t_i on to be analysed to find the time
             correspondence
10         return false
11     //t_i already verified
12     j ← i + 1
13     count ← 1
14     while count < |password| ∧ j < n do
15         if (n − j) < (|password| − count) then
16             //Not enough peaks from t_i on to be analysed to find the time
                 correspondence
17             break
18         if (t_j − t_i) < (y_count − threshold) then
19             //Too less time between the time instant of the first character and
                 the time instant of the count-th character
20             j ← j + 1
21         else if (t_j − t_i) < (y_count + threshold) then
22             //Time correspondence
23             count ← count + 1
24             j ← j + 1
25         else
26             //Too much time between the time instant of the first character and
                 the time instant of the count-th character.
27             break
28         if count =|password| then
29             //Time correspondence found
30             return true
```

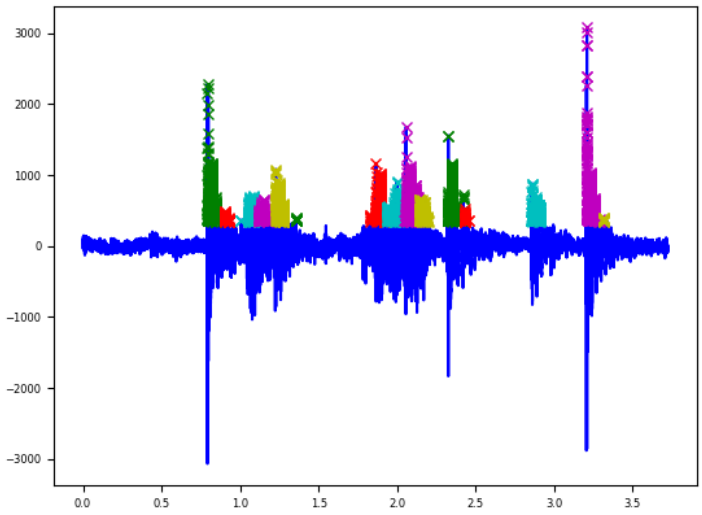


Figure 5.1: Audio during insertion of password `hello35` all sub-windows highlighted.

5.1.2 Character correspondence

When pressed each key of the keyboard produces a variation of the signal, called *press peak*, for a time window of about 8-10 ms[38]. This signal trend can also be divided in three consecutive and distinctive areas:

- **touch peak**
  peak in a window of 2-3 ms, caused by the finger touching the key
- **noisy meaningless area**
- **hit peak**
  peak in a window of 2-3 ms, caused by the finger and the key hitting the keyboard supporting plate.

To obtain a prediction of each key pressed by the user, we can extract information from the touch peak, that is the most significant, and the hit peak, that increases the information related to the pression.
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 required and implemented for this classification method:

- Data acquisition

- Extraction of features
- Neural network
- Verification

5.1.2.1 Data acquisition

To create a program that record audio while user type something, I created `DatasetAcquisition.py` source file containing the relative class. After instantiating an object of the class `AcquireAudio`, it applies `record` method to this instance.
Inside this method, two different program are run in parallel: the first one is a key-logger that is used to classify all the recorded audio files in some directories and the second one that records an audio file during keys typing. The update of private members of the class is guaranteed through the use of the mutual exclusion (mutex) management. The keylogger in the first task requires the access to the operating system signals generated by typing a key on the keyboard. It has the only purpose to continue the acquisition of the audio files even if a special key is pressed (for example F3 button).
The choice of running two different tasks in parallel was given by the need of recording audio before the start and after the end of password insertion by the user. Each recorded audio can contain several audio peaks related to multiple insertion of the same key but, during the acquisition of training and test set, I record one audio file for each key pressed.
Hence in this particular case, 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 also 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 the output directory, each one with the name of the respective typed key.
The recording phase was performed using directly 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 pressed key is inserted, are reported in **Appendix A** .
Looking at Table, we can see that the keylogger changes its behaviour mapping each key to an ASCII string of upper or lower alphabetic characters because otherwise many keys would be mapped into invalid names of folders (for example, the key *'.'* is now mapped into the label *'POINT'*). In the table, 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 assigned by me to each key. The reason why these labels differ for some entry are:

- **higher accuracy for 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 key-logger but they are considered different thanks to the final mapping;

- **improve the classification of keys made by key-logger**
  for example, *'ALT'* label is wrongly mapped into *'SHIFT'* by key-logger.

- **solve the problem of keys mapped only by hardware**
  FN is the only key with this problem. The key-logger doesn't detect any pressed key, when *FN* is inserted. Hence, I needed to typed it and then another key to be sure that recording for *FN* was performed. Then I made another python script to resize the audio signal and remove the useless second peak.

The last two reasons are very important because they highlight also the power of acoustic side-channel. If an attacker implements an high-level key-logger exploiting also microphone information, the accuracy of its software can increase very much.
In fact the hacker could collect a dataset of recordings of pressed keys on the same type of the victim's keyboard and then could train a Neural Network, that will be add in its malicious code. For each key, I record 200 audio signals obtaining a dataset of 20400 audio files. To improve the accuracy of the prediction for the neural network, I performed also Data Augmentation of the audio signals used for the training phase. I tried two approaches:

- **Time-shift**
  from each audio signal I created 4 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 added a sequence of random samples from a Gaussian distribution, with standard deviation equal to 150 and mean 0, generating four new audio signals.

Using these approaches I have a training set of 2000 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 first dataset only. The efficiency of the Neural Network trained on first and third dataset is worst than the one related to the network trained on first dataset only because the third dataset introduces many sequences of FFT coefficients that are very similar even if they are related to different keys. Hence I used only the network trained on the first dataset and both on the first at the second dataset as prediction model. So having 102 keys, I had respectively a dataset of 20400 and 102000 audio files.

5.1.2.2 Classification

I used three different approaches, the first two were taken from the work of Asonov and Agrawal and the last one was based on modern sound classification techniques.
Respectively to the method used, the feature for each key was composed by:

- FFT coefficients of the touch peak
- FFT coefficients of the hit peak and the touch peak
- Features obtained from the hit peak and the touch peak using a deep learning pre-trained model

In the first two cases, the coefficients are extracted from a window of 3 ms around the peaks and then they are normalized in floating point values in range $[0, 1]$ (see **Figure 5.2**).
In the third case, the touch peak and the hit peak samples were concatenated, creating a new signal on which the spectrogram is computed. From the spectrogram, I extract a feature composed by 512 values through the use of VGG16 pre-trained convolutional neural network.
In this way, I remove the last fully connected layers, used for classification of other task, and take the intermediate results as feature. The reason of this approach is that a pre-trained network already extracts very well features for classification of a lot of common immages and so it can extract features better than a NN created from scratch.

5.1.2.3 Verification

The audio signal taken, during the insertion of the password, is analysed and then organized in windows as specified in the **Section 5.1.1** , but the verification is based on:

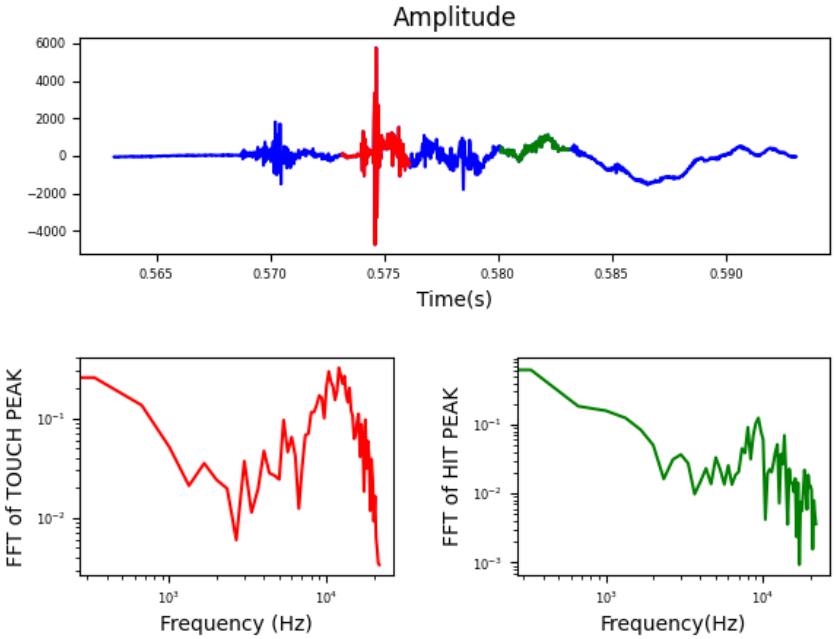


Figure 5.2: Example of normalized FFT computation of the touch peak and the hit peak for an audio file of key *'0'*.

- every window previously computed
- the windows that contains the time instants related to the time correspondence

In the first approach the application uses the maximum value of each window as the touch peak and looks for the related hit peak, taking it about 10 ms after the previous peak. After the computation of the features for the two peaks, with one the methods described in the previous section, I perform the prediction using the Neural Network. I collect the most probable predicted keys and I repeat the procedure for every window initially computed on the audio. This method is very weak because after this phase, the algorithm tries to find an ordered sequence of characters, one from each window, that corresponds with the password inserted by the user. If this exists, AcCAPPCHA declares that the user was a human, otherwise a bot. The main problem of this approach is that there is no correspondence in time between a character belonging to the final sequence and the moment in which the same character was inserted physically by the user. In fact there can be false positives caused by the prediction from peak that are not related to the absolute maximum one. In other terms, in the set of the maximuma of all the windows there can

be someone that is not related to the touch peak but to a local maximum. The second approach solves the previous problem because the windows, where the maxima are looked for, are obtained by the correspondence time approach. In this case, AcCAPPCHA verification becomes more accurate in theory even if in practice the deep learning technique is not very efficient in prediction for a single key.

5.2 Communication between client and server

The algorithm that perform the evaluation of user activity (see **Section 5.1**) is performed at client side but the value returned by it is evaluated at server-side. The response of the evaluation of user activity concatenated with a nonce and then signed through ECDSA, is sent to the server (see **Section 4.2**). The use of the nonce, unique and random generated sequence, is very important to guarantee that no reply attacks would be performed. In fact the server, after the reception of a message from the client, the server can check if the client has already 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 that previously establishes a client was human. This type of procedure can be also useful to sign HTTP data, for example data sent using POST request (as insertion of password during authentication phase). In the testing phase I performed, I designed and implemented also a simplified version of the communication between the client and the server for an authentication service.
The application was tested on local network and the actions performed by involved parties are described in details in the following sections (see Figure).

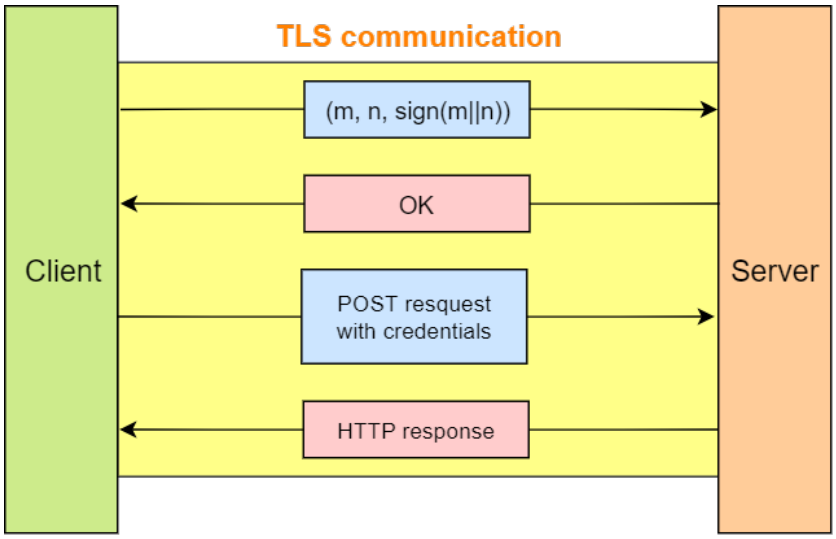
5.2.1 Client

The client performs the authentication following these steps:

1. It establishes a connection with the server over TLS layer to increase the security strength of the communication between the parties;

2. It sends the message $(m, n, sign(m||n))$ where:

   - m is the string with the response of AcCAPPCHA algorithm on client side (*'True'* if Human, *False* if bot)
   - n is the the nonce



- $sign(m||n)$ is the ECDSA signature of the concatenation $m||n$ of the response and the nonce. For ECDSA was used SHA256 algorithm.

From a practical point of view, I format the message in the following way:

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

According to basic rules in grammar of **HTTP/1.1** (see Section 2.2 of RFC 2616), *CR* is the carriage return ('$\backslash r$') and LF is the line feed ('$\backslash n$'). The spaces in the message aren't considered. In this way, I can separate easily m looking at '$\backslash r \setminus n$'. The nonce has a fixed length of 16 bytes.

3. The client waits for response of the server with format:

```
response CRLF
```

If the answer is equal '$OK \setminus r \setminus n$', AcCAPPCHA will go on with the authentication step, otherwise the client-side application performs again the verification, asking again to user to insert the password. The maximum number of trials for a particular user is 3.

4. If everything goes well in the previous step, The client sends the credentials (username, password) to the server through a POST request

to '/cgi-bin/auth' resource. The name of folder '/cgi-bin' comes from the standard name of the folder with functions and *auth* is the name of the function that server will call. 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 before using SHA512. The POST request used 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 everything follows as before the grammar in RFC 2616). In fact also **SP** represents the space character as in the documentation. **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.

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 only to show. 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.

5.2.2 Server

The client performs the authentication following these steps:

1. It establishes a connection with the client, after his request, over TLS layer to increase the security strength of the communication between the parties;

2. It receives the message $(m, n, sign(m||n))$ and check the integrity of the message. To do it, the server decrypts the ECDSA signature using the client's ECDSA public key and compares the result with $m//n$.

3. If compared messages were the same, the server checks if the nonce was already used by the same client. If so, the server thinks that there

was an attacker that is performing a replay attack. 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 identifies 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 client-side or other useful parameters.
- **Value: list of nonces**
  Every time a client performs a new verification request on the server, the nonce is added to the list related to its IP address 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 '$OK \setminus r \setminus n$' otherwise '$NO \setminus r \setminus n$'. If some error occurs it sends '$ERROR \setminus r \setminus n$' to the client. In the last two cases, the server terminates.

5. If the server doesn't terminate, it waits for the POST request from the client and analyses it to perform authentication service. The server replies to client with several 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:
     - **User not 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.

– **Wrong password**
the server sends the following HTML code if the hashed password received from the client isn't the same with respect to the one stored in database for the specified username.

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


Failure

Wrong password.

– **User logged in**
the server sends the following HTML code if the specified username exists in the database and its hashed password stored in the database is the same of the one received through 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 decide to use this approach to be more flexible to future integration to more complex database.
The database is composed by only a table `CloudUser` to store information about user identity usually stored during the sign up phase. The creation of the database was performed through 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 domain to manage 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 formatting the result as a string of hexadecimal values. Then I populated the dataset with some entries, related to fake user, only for testing purpose. An example of inserted users is the following one:

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

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

In practice, `HASHPASSWORD` is 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 thanks to `-keyout` option in the *client.pem* and *server.key*.

The keys, used in ECDSA signing and verification, were created as follow from 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())
```

The *ecdsa* module gives access to the management of operations performed by signing and verification phase. 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 Windows 10 Home version 1909 as Operating System.

6.1 Human detection

The application response was tested over all the three possible scenarios (see **Section 5.1**):

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

The first method is the most efficient and usually at first or second trial already finds a human activity. Sometimes the correspondence isn't found at first trial because of background noise.
The second approach is very powerful with short password, because of less noise given by the user interaction with keyboard. In fact using PIN passwords of 4 digits, AcCAPPCHA easily detects the human activity. For the last method, the same results of the previous approach are observed.
Deep learning approaches require the reduction of number of labels to obtain good results. In this way I limit the possible characters that user could use and the should be composed of lower case alphabetic characters, accented vowels or numbers (see **Figure 6.1**) but considering again the 10 high probable labels predicted for each peak.
In any case the last method that uses both the time and the character correspondence speed up the search of the correct peak related to the insertion and reduces the possibility of having a false positive. However there are many

false negatives given by the similar sound produces by several keys opposite to D. Asonov, R. Agrawal's conclusions[38].
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 that other During the insertion of the password, the user can obviously type backslash key but then '\b' character isn't considered in the final verification. The worst deep learning method is the one that uses the spectrograms to extract the features of a press peak. This approach difficulty predict all the keys even for password of 4 characters.

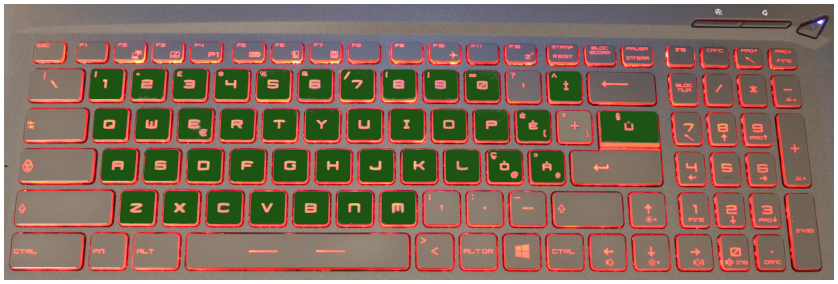


Figure 6.1: Keys to be used in the password (highlighted in green).

6.2 Bot detection

I've tested both human and bot activities analysing the response generated by AcCAPPCHA. The bot activity was emulated 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 (see). Assuming the hacker obtained the credentials in some way, the bot communicates the username and the password to running AcCAPPCHA with only time correspondence

option selected. In practice the strength of AcCAPPCHA against this attack is very high because the insertion of the password is managed through `getwch()` call 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 king of pipes.
The program easily accepts the username, because AcCAPPCHA acquires it using standard `input()` function. Then AcCAPPCHA waits for characters of password, until ENTER key is pressed, but popen doesn't have 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 the insertion of 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, I pretended to be a bot and access directly the console, bypassing the stream limits of **msvcrt** character acquisition. To emulate the user the bot program should start the execution run

and immediately after the hacker must open the working terminal with AcCAPPCHA running.
This scenario isn't very feasible because requires management of terminal windows but was useful to establish if the insertion of the password by malicious software is correctly classified as a bot activity. Each character of the password is inserted emulating press and release of the corresponding key of the keyboard.

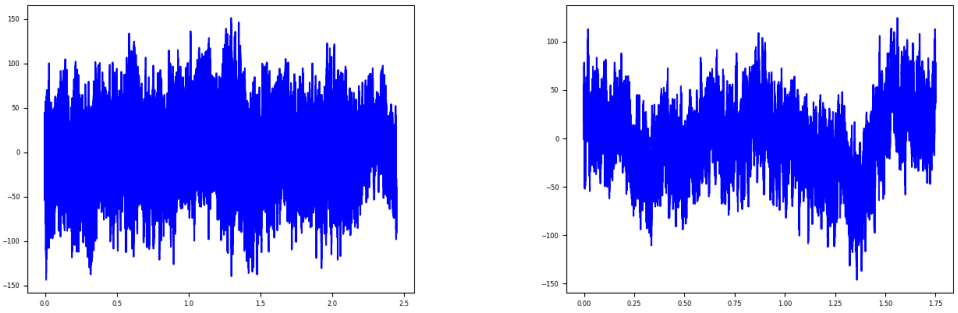
```
from pynput.keyboard import Key, Controller
from time import sleep

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. For this reason, this type of attack isn't feasible in practice as the previous mentioned attack with Python bot.

In the last two approaches the audio recorded by AcCAPPCHA are very similar and highlights the two most probable situations:

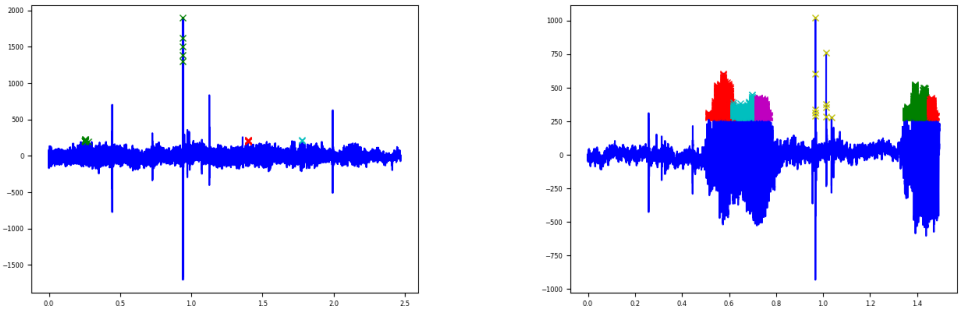
- **The noise during noise evaluation is very high**
  If this happens, no audio peaks can be found (see **Figure 6.2**).
- **The noise during noise evaluation is normal or very low**
  If this happens, some audio peaks can be found if there is some noise during password insertion. However they don't have time correspondence with time instants stored during the password insertion (see **Figure 6.3**). The only case, in which a bot can 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 too high noise only during the insertion of the password and not during the noise evaluation. For example, if the attacker would analyse the background noise finding a sequence of peaks and defining a model for amount of time between each couple of them, he couldn't use information from it. In fact it could insert a character of the password after an amount time previously modelled but in practice, the peaks of background noise would be discarded through noise evaluation.



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

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

6.3 Security analysis



(a) Bot with `pynput` module.

(b) Team Viewer.

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

Chapter 7

Future

Appendix A

Key Mapping

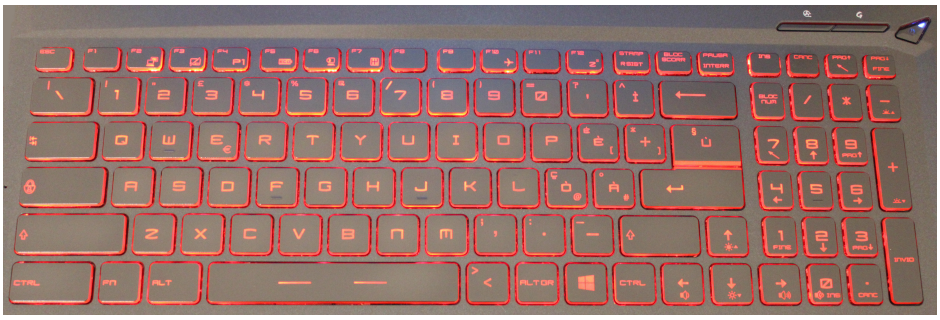
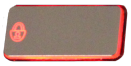
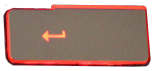


Figure A.1: Layout of the keyboard of my MSI GL63 8RD laptop.

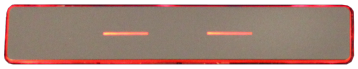
Key	Key-logger label	Final label
	0	0
	• INSERT (Num lock on) • None (Num lock off)	0_INSERT
	1	1
	• END (Num lock on) • None (Num lock off)	1_END
	2	2
	• DOWN (Num lock on) • None (Num lock off)	2_DOWN

	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: Map of pressed key performed by key-logger and then manually modified in final label.

Appendix B

Program

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, "FaceD-CAPTCHA: 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] 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.*

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

[30] Kazuo Sakiyama, Takanori Machida, Arisa Matsubara, Yunfeng Kuai, Yu-ichi Hayashi, Takaaki Mizuki, Noriyuki Miura, Makoto Nagata, "Authentication Using Side-Channel Information." in *IACR Cryptol. ePrint Arch., 2015, 834.*

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

[32] 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.*

[33] 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.*

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

[35] 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.*

[36] Meriem Guerar, Alessio Merlo, Mauro Migliardi, Francesco 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.*

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

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

Acknowledgements

Professor Migliardi

I would like to express my very great appreciation to Dr Guerar

I would like to express my gratitude to University of Padova for the study path I performed. 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 and now I spend programming all my free time. Thanks to University because professors follow my thirst of knowledge and I grew up, living alone and really becoming an adult.

Thanks to staff,

Thanks to other company,

I would like to express my gratitude to my family that taught me to never give up. In particular thank to my sister that, with her great experience in University course, has been a reference for my study attitude and perseverance.

Thanks to my grandmother Concetta that, with her smiles, has left to me happiness during my darker and difficult periods.

Thanks to my grandmother Maria that, was always worried for my health issues and tries always to give me up using her food.

Thanks to Cristina,

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,

Thanks to Giuseppe, Aurora, Alessia and Sara,

Thanks to Elia,

Thanks to Lorenzo,