

**Interweaving Unicode, Color, and Human Interactions
to Enhance CAPTCHA Security**

by

Narges Roshanbin

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Web security has become a critical issue due to the rising reliance of people on diverse types of online transactions. A CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a security mechanism that is widely used to maintain the security of web services by preventing malicious programs from accessing these resources automatically. Despite the existence of several types of CAPTCHAs, many of them have been compromised due to their inherent vulnerabilities and the development of strong artificial intelligence and image recognition algorithms. The vulnerabilities of existing CAPTCHAs coupled with the trend of heavier dependence on the Internet calls for the development of a new generation of CAPTCHAs that are substantially more complex for machines, and yet easy to understand and solve by human users.

In this thesis, we propose, implement and test a new CAPTCHA, which shows resistance against various forms of segmentation and recognition attacks. The multi-layered security approach employed in this CAPTCHA mainly comes from its use of Unicode as an input space, a virtual keyboard as the input device, the use of homoglyphs, the correlated usage of color in the foreground and background, and solution submission through intelligent human interaction. Furthermore, several forms of randomization are employed within different elements of the CAPTCHA which minimize the formation of detectable patterns that can be exploited by machines and make the attacks computationally complex for attackers. Our analyses provide evidence for substantial resistance of the proposed CAPTCHA against major attack types.

Our CAPTCHA's game-like solution procedure is enjoyable and intuitive for human users despite its relatively longer solution time compared to existing CAPTCHAs which comes as the

price for the higher level of security it affords. Our user studies indicate that the CAPTCHA's solving accuracy is comparable to major CAPTCHAs in use today.

The complexity of this CAPTCHA can be further modified based on the security requirements of the resource being protected. Additional security- and usability-enhancing modifications are proposed and tested, which can further improve the CAPTCHA's security or usability as needed.

Preface

This research project received research ethics approval from the University of Alberta Research Ethics Board, Project Name “An investigation to improve CAPTCHAs used for distinguishing humans from machines on the Internet”, ID. Pro00028168, Feb. 2012.

Chapter 1 of this dissertation has been published as N. Roshanbin and J. Miller, “A survey and analysis of current CAPTCHA approaches,” *J. Web Engineering*, vol. 12, no. 1, pp. 1-40, 2013. Sections 2.5 and 2.6 have been published as N. Roshanbin and J. Miller, “Finding Homoglyphs - A Step towards Detecting Unicode-Based Visual Spoofing Attacks,” in *Web Information System Engineering - WISE*, 2011, pp. 1-14. The majority of Chapter 2 is derived from an article submitted for publication as N. Roshanbin and J. Miller, “ADAMAS: Interweaving Unicode and Color to Enhance CAPTCHA Security,” *J. Future Generation Computer Systems*. Chapter 3 and the appendix of this thesis have been submitted for publication as N. Roshanbin and J. Miller, “Enhancing CAPTCHA security using interactivity, dynamism, and mouse movement patterns,” *International Journal of Information Security*. An article has been submitted for publication which uses material from Chapter 5 (N. Roshanbin and J. Miller, “A comparative study of the performance of local feature-based pattern recognition algorithms,” *J. Pattern Recognition*).

Dedication

This dissertation is dedicated to my family without whom I would not be here:

To my husband, Moein, who has been a source of love, encouragement and inspiration to me. Thank you for all the joy you have brought to my life and for patiently bearing with me during the completion of this work and encouraging me in challenging situations.

To my parents, Fatemeh and Mehdi, for their unconditional love and support throughout my life, for teaching me the value of hard work, and for instilling in me the will to pursue my dreams. Thank you for your gifts of life and love.

To my sisters, Asieh and Razieh, whose love, sincerity, intimacy and empathy have always been a source of peace and happiness for me.

Acknowledgements

I would like to express my sincere gratitude to Professor James Miller, for his continuous support and guidance during my years in the PhD program and throughout the completion of this dissertation. Having professor Miller as my supervisor has been a true blessing.

I would also like to thank my committee members Dr. Bruce Cockburn, Dr. John Aycock, Dr. Eleni Stroulia and Dr. John Salmon for their thoughtful and constructive comments and feedback.

In addition, I am thankful to the institutions that granted me scholarships and financial assistance during this research: Alberta Innovates – Technology Futures for iCORE scholarship and University of Alberta for Queen Elizabeth II scholarship.

Contents

INTRODUCTION.....	1
CHAPTER 1 AN OVERVIEW OF EXISTING CAPTCHA SYSTEMS.....	6
1.1 INTRODUCTION.....	6
1.2 CURRENT CAPTCHAS	7
1.2.1 <i>Text-based CAPTCHAs</i>	9
1.2.2 <i>Image-based CAPTCHAs</i>	17
1.2.3 <i>Audio-based CAPTCHAs</i>	22
1.2.4 <i>Motion-based CAPTCHAs</i>	23
1.2.5 <i>Hybrid CAPTCHAs</i>	24
1.3 BREAKING CAPTCHAS	27
1.3.1 <i>Algorithmic approaches to break CAPTCHAs</i>	27
1.3.2 <i>Random guessing attacks</i>	41
1.3.3 <i>Using social engineering to break CAPTCHAs</i>	41
1.4 ROBUSTNESS OF CAPTCHAS	42
1.4.1 <i>Segmentation resistance</i>	43
1.4.2 <i>Recognition resistance</i>	46
1.4.3 <i>Random guessing resistance</i>	47
1.4.4 <i>Security against third party human solver attacks</i>	48
1.4.5 <i>Other security measures</i>	48
1.5 USABILITY OF CAPTCHAS	49
1.5.1 <i>Distortion</i>	49
1.5.2 <i>Content</i>	50
1.5.3 <i>Presentation</i>	51
1.6 CONCLUSION.....	52
CHAPTER 2 THE PROPOSED CAPTCHA.....	58
2.1 INTRODUCTION.....	58
2.2 INTERACTIVE CAPTCHAS	60
2.3 OVERVIEW OF OUR APPROACH.....	64
2.3.1 <i>Major vulnerabilities of current CAPTCHAs</i>	64
2.3.2 <i>An overview of our approach</i>	65
2.4 INTERACTIVITY	80
2.5 UNICODE.....	83
2.5.1 <i>Glyphs and characters</i>	85
2.5.2 <i>Fonts</i>	87
2.5.3 <i>Selection of the characters of the test and the keyboard</i>	93
2.6 HOMOGLYPHS	95
2.6.1 <i>Homoglyphs in Unicode</i>	95
2.6.2 <i>Similarity measurement between glyphs</i>	97
2.7 VIRTUAL KEYBOARD	106
2.8 COLOR REPRESENTATION	116
2.9 USER STUDIES AND RESULTS	126
2.9.1 <i>Design of the user study</i>	127
2.9.2 <i>Results of the experiment</i>	128
2.10 CONCLUSION.....	140

CHAPTER 3	RISK DIVERSIFICATION WITH A MULTI-PROBLEM CAPTCHA	141
3.1	INTRODUCTION.....	141
3.2	INTRODUCING EACH TYPE OF DRAG AND DROP OPERATION	143
3.2.1	<i>Hitting a randomly-moving ball</i>	144
3.2.2	<i>Passing through rotating bars without touching them</i>	145
3.2.3	<i>Passing through moving gates</i>	145
3.3	SECURITY ANALYSIS.....	147
3.4	USER STUDY.....	150
3.4.1	<i>Usability study</i>	151
3.4.2	<i>Differentiating humans from machines by their mouse movements</i>	153
3.5	CONCLUSION.....	158
CHAPTER 4	SECURITY ANALYSIS OF THE PROPOSED CAPTCHA	159
4.1	INTRODUCTION.....	159
4.2	ATTACKS ON EXISTING CAPTCHA SCHEMES.....	159
4.3	SECURITY CONSIDERATIONS IN THE PROPOSED CAPTCHA.....	164
4.3.1	<i>Recognition attack considerations</i>	165
4.3.2	<i>Segmentation attack considerations</i>	171
4.3.3	<i>General security considerations</i>	172
4.4	THE PROPOSED CAPTCHA'S RESISTANCE AGAINST SEGMENTATION AND RECOGNITION ATTACKS	172
4.4.1	<i>Segmentation attack</i>	173
4.4.2	<i>Recognition attack</i>	178
4.5	CONCLUSION.....	188
CHAPTER 5	SECURITY-ENHANCING MODIFICATIONS	190
5.1	INTRODUCTION.....	190
5.2	MODIFICATIONS USED IN THE TESTS	191
5.3	SELECTED PATTERN MATCHING ALGORITHMS	194
5.3.1	<i>SIFT (1999)</i>	197
5.3.2	<i>FAST (2006)</i>	198
5.3.3	<i>SURF (2008)</i>	198
5.3.4	<i>BRIEF (2010)</i>	199
5.3.5	<i>ORB (2011)</i>	201
5.3.6	<i>BRISK (2011)</i>	202
5.3.7	<i>FREAK (2012)</i>	203
5.4	THE EFFECT OF DIFFERENT NOISE ON THE PERFORMANCE OF PATTERN MATCHING ALGORITHMS	204
5.5	USER STUDY.....	207
5.5.1	<i>Results of the user study</i>	208
5.6	CONCLUSION.....	209
CHAPTER 6	USABILITY-ENHANCING CONSIDERATIONS	211
6.1	INTRODUCTION.....	211
6.2	USER STUDY.....	212
6.2.1	<i>Design of the user study</i>	213
6.2.2	<i>Results of the experiment</i>	214
6.3	CONCLUSION.....	216
CHAPTER 7	DISCUSSION AND CONCLUSION	218
BIBLIOGRAPHY	221

APPENDIX A: DESCRIPTION OF MOUSE MOVEMENT FEATURES 230

List of Tables

Table 1: Different types of CAPTCHAs.....	53
Table 2: Attacks on CAPTCHAs.....	53
Table 3: Major vulnerabilities of current CAPTCHAs.....	54
Table 4: A subjective assessment of the vulnerabilities of current CAPTCHAs.....	56
Table 5: Examples of mono-script homoglyphs.....	96
Table 6: Examples of complex-script homoglyphs.....	96
Table 7: Examples of NCD values between some characters.....	106
Table 8: Frequency of the number of homoglyphs at NCD threshold=0.3.....	106
Table 9: Mean solution time of a test.....	129
Table 10: Incorrectly matched characters.....	130
Table 11: Examples of distinguishable characters.....	130
Table 12: Examples of simple-shaped characters.....	130
Table 13: Similar characters distinguished correctly.....	131
Table 14: Users' responses to the survey following the CAPTCHA.....	138
Table 15: User success rates.....	151
Table 16: User solution times in seconds.....	151
Table 17: Solution time and accuracy of expert and non-expert users.....	152
Table 18: Users' responses to the survey following the CAPTCHA.....	152
Table 19: Features extracted form users' mouse movement data.....	156
Table 20: Pre-processing attacks on existing CAPTCHAs.....	160
Table 21: Segmentation attacks on existing CAPTCHAs.....	160
Table 22: Recognition attacks on existing CAPTCHAs.....	162
Table 23: Attacks on existing CAPTCHAs.....	162
Table 24: The results of applying SURF on monochromatic images of 100 CAPTCHAs.....	181
Table 25: The probability of 1, 2, 3 or 4 correct matches in a test (for 100 tests).....	182
Table 26: Results of applying SURF on images segmented by k-means.....	184
Table 27: Results of applying the cross-correlation algorithm on images segmented by color palette.....	187
Table 28: Results of applying a cross-correlation algorithm on images segmented by k-means.....	188
Table 29: Results of the security analysis of the base CAPTCHA.....	189
Table 30: Object recognition methods.....	195
Table 31: Probability of success in correctly matching a character using different pattern recognition algorithms.....	205
Table 32: Ranking of noise types based on their impact on the performance of matching algorithms.....	205
Table 33: Factors tested in the user study.....	207
Table 34: The effect of different noise/distortion types on accuracy and solution time.....	208
Table 35: Pairwise comparison test for solution time.....	209
Table 36: Factors tested in the user study.....	213
Table 37: The effect of familiarity of test characters on accuracy and solution time.....	214
Table 38: Pair-wise comparison test for solution time.....	214
Table 39: The effect of learning on the accuracy and solution time.....	215
Table 40: Regression results for solution time.....	216
Table 41: Speed in different regions of mouse movements.....	234

List of Figures

Figure 1: Steps of an attack on MSN CAPTCHA [69].	38
Figure 2: The proposed CAPTCHA.	65
Figure 3: The proposed CAPTCHA.	77
Figure 4: The proposed CAPTCHA in the palette mode.	79
Figure 5: The proposed CAPTCHA after matching one character.	79
Figure 6: Zoomed test and keyboard.	80
Figure 7: Various glyphs displaying the character ‘a’.	85
Figure 8: Relations between characters and glyphs in Unicode Standard.	86
Figure 9: Ideographic variations of Unicode characters.	86
Figure 10: Characters with context-sensitive glyphs.	87
Figure 11: Representation of character 'A' by some standard fonts.	88
Figure 12: Representing letter ‘A’ with various abstract fonts.	88
Figure 13: Different glyphs for Chinese ideograph ‘字’.	90
Figure 14: Some different glyphs for Arabic character ‘FEH’.	90
Figure 15: The coverage of each Unicode font of the modern script characters [127].	93
Figure 16: Examples of characters removed to increase usability.	94
Figure 17: Examples of homoglyphs in the Unicode [127].	97
Figure 18: Positions of Homoglyph and Anti-homoglyph in the similarity spectrum [127].	98
Figure 19: Histograms of NCD values calculated by different 1-D compressors [127].	102
Figure 20: Histograms of NCD values calculated by different 2-D compressors [127].	103
Figure 21: The histogram of the pair-wise NCD calculation over all characters in our set [127].	105
Figure 22: Cumulative distribution function of pair-wise NCD calculations [127].	105
Figure 23: CDF of results with $0 < \text{NCD} \leq 0.75$ [127].	106
Figure 24: Three first iterations of “random point set selection” algorithm used in this CAPTCHA.	115
Figure 25: The first background of the proposed CAPTCHA.	119
Figure 26: Background of the proposed CAPTCHA.	119
Figure 27: Examples of similarity between background components and characters or parts of the characters.	120
Figure 28: Demonstration of the ability to traverse major colors in HSB by changing hue.	124
Figure 29: Solution time histogram.	129
Figure 30: Characters that users are less inclined to solve.	132
Figure 31: Average time for different solving orders.	132
Figure 32: Average time of different solving orders.	133
Figure 33: Defining a categorical variable for location.	134
Figure 34: Average search time for characters of different colors.	136
Figure 35: The percentage frequency of characters in each order for the four highest-occurring alphabets.	136
Figure 36: Drag and drop operations in the proposed CAPTCHA.	146
Figure 37: SURF applied to a sample test with no additional noise.	167
Figure 38: SURF applied to a test with added character segment and a slightly stretched character match. The character is incorrectly matched with its segment.	167
Figure 39: The effect of including homoglyphs in a test on the accuracy of SURF.	168

Figure 40: SURF applied to a test containing curved noise components.	169
Figure 41: SURF applied to a monochrome image of the proposed CAPTCHA.	170
Figure 42: Cross correlation applied to a monochrome image of the proposed CAPTCHA.	171
Figure 43: Gradients of a color in background noise become similar to a character's color.	173
Figure 44: The proposed CAPTCHA.	174
Figure 45: Histogram of the H component of the CAPTCHA image in Figure 44.	175
Figure 46: An example of a monochromatic image produced by color histogram.	176
Figure 47: An image segmented by k-means clustering algorithm.	177
Figure 48: The results of applying SURF on four monochrome images.	181
Figure 49: The result of applying SURF on images segmented by K-means segmentation or color histogram.	183
Figure 50: The histogram of number of character parts in our selected set of Unicode characters.	185
Figure 51: The result of applying a normalized cross-correlation matching algorithm on a monochrome image segmented by color palette.	185
Figure 52: The result of applying a normalized cross correlation algorithm on images segmented by color histogram or k-means.	188
Figure 53: Security-enhancing modifications.	192
Figure 54: Increasing the amount of background curved noise.	194
Figure 55: The histogram of movement speed for human users.	232
Figure 56: The median speed of movements in each direction.	232
Figure 57: Histogram of length of pauses.	232
Figure 58: Mouse movement directions.	232

List of Abbreviations

AI	Artificial Intelligence
BRIEF	Binary Robust Independent Elementary Features
BRISK	Binary Robust Invariant Scalable Keypoint
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CDF	Cumulative Distribution Function
CIE-LAB	Commission Internationale de l'éclairage – LAB color space
CJK	Chinese-Japanese-Korean
DnD	Drag and Drop
EV	Expected Value
FAST	Features from Accelerated Segment Test
FREAK	Fast Retina Keypoint
HSB	Hue-Saturation-Brightness color space
IM	Instant Messenger
NCD	Normalized Compression Distance
NID	Normalized Information Distance
OCR	Optical Character Recognition
ORB	Oriented FAST and Rotated BRIEF
RBI	Rotating Bars Interaction
RGB	Red-Green-Blue color space
SD	Standard Deviation
SIFT	Scale Invariant Feature Transform
SURF	Speeded-Up Robust Features
UCS	Universal Character Set

Introduction

Almost two and a half decades after the invention of the World Wide Web in 1990, 40% of the world's population has an Internet connection,¹ yet the web is still in its infancy. People and businesses around the world are increasingly relying on the Internet to gain access to resources that require high levels of security, such as conducting monetary transactions or gaining access to sensitive personal or financial information. Due to their high processing power, machines have the ability to use techniques that help them gain illegitimate access to such resources on the web. The first CAPTCHA was developed in 2000 as a type of challenge-response test that can distinguish human users from machines or robots in order to inhibit their access to such resources.

Today, different classes of CAPTCHAs have been developed to address this ever-increasing need for security. However, with improvements in artificial intelligence algorithms, the security of many of these CAPTCHAs has been compromised and most of them exhibit significant vulnerabilities.

In this thesis, we identify and categorize the major issues and vulnerabilities of existing CAPTCHAs through an extensive literature review, and use the resulting security themes to inform us in the development of a more secure CAPTCHA.

We propose, develop, and test a new CAPTCHA that addresses these vulnerabilities in order to achieve higher security by interweaving Unicode, color, intelligent interactivity, and a virtual keyboard. We also extensively use randomization to prevent the formation of machine-detectable patterns in this CAPTCHA. Through multiple security tests and several user studies, we assess

¹ <http://www.internetlivestats.com/internet-users/>

the security and usability of this CAPTCHA and answer several questions regarding the impact of some design factors on the CAPTCHA's usability and security. Eventually, we develop and test further modifications that can be used to enhance and customize the CAPTCHA's security and/or usability for different applications.

Our work in this thesis presents the following contributions to the existing literature on CAPTCHAs:

- We document, classify, and analyze several types of vulnerabilities that plague existing CAPTCHAs and paint a picture of the overall security status of existing CAPTCHAs by showing how major CAPTCHAs exhibit many of these vulnerabilities, any one of which may be sufficient for breaking them. Our findings highlight the need to develop more secure CAPTCHAs through taking a multipronged approach to CAPTCHA security.
- We propose, develop, and test the idea of a CAPTCHA which incorporates a multi-layered security mechanism. Our work provides a comprehensive benchmark against which future multi-layered CAPTCHAs can assess themselves in terms of security and usability. This paves the way for the further development and growth of a new class of CAPTCHAs that exhibit resistance against several classes of attacks.
- We document the first utilization of Unicode as a unique repertoire for automatic CAPTCHA generation which addresses two major limitations in CAPTCHA development: 1) the limitation of text-based CAPTCHAs to characters of a specific script such as Latin, and 2) the challenge of image-based CAPTCHAs in creating a suitable image database. To the best of our knowledge, our work is the first comprehensive analysis of Unicode as a source for CAPTCHA elements. We provide

a thorough analysis of Unicode's strengths and weaknesses for this type of application and provide insights on how it can be used in the development of more secure CAPTCHAs by capitalizing on the idea of homoglyphs and a method for defining homoglyphs using the NCD similarity metric. The use of Unicode requires the use of a virtual keyboard for character input which further enhances security.

- Our work provides insights on how human interactions can be used to strike a double whammy on attacking algorithms by improving security *while* diversifying the risk of the CAPTCHA being broken across two challenges. This is achieved by requiring users to intelligently interact with dynamic elements within the CAPTCHA in order to submit their solutions. This structure allows the first challenge to naturally flow into the second one without introducing any disruption or discontinuity into the solution process.
- In the development of the interactive component of our proposed CAPTCHA, we conduct a comprehensive study of human mouse movement and document the empirical distribution of several unique mouse-related characteristics. Our analysis suggests that any CAPTCHA can be further strengthened by tracking a user's mouse movements and differentiating unsuspecting robots that fail to exhibit human mouse movement characteristics.
- Our user studies document several findings concerning the impact of contextual factors on human users' solution accuracy and solving time. These findings shed light on how user performance in solving the proposed CAPTCHA is affected by factors such as different noise types, character color and location, alphabet,

familiarity with scripts, and practice. These behavioral insights can be used in the development of future CAPTCHAs.

- To the best of our knowledge, our work provides the first thorough analysis of the pros and cons of using color in developing CAPTCHAs and provides several ideas on how color can be effectively used without compromising security.
- In our CAPTCHA's security analysis, in most cases, we go beyond the common random guess success probability estimations and explicitly test the resistance of our CAPTCHA against several major pattern recognition algorithms along with very conservative assumptions on how intelligently these algorithms make final matching decisions.

Our proposed CAPTCHA achieves a high level of security against several forms of attacks by interweaving Unicode, color, CAPTCHA dynamism, human interaction and several forms of randomization. Our CAPTCHA's solution accuracy is comparable to the major CAPTCHAs in use today. Our user studies indicate that humans can comfortably solve this CAPTCHA, which is a sign of its intuitive design and task simplicity for human users.

In Chapter 1 of this thesis, we review the literature on CAPTCHAs and delineate the major security issues that existing CAPTCHAs face. We follow this discussion in Chapter 2 by introducing our proposed CAPTCHA and describing how it addresses the identified vulnerabilities. In this chapter, we conduct our first user studies to elicit information on the CAPTCHA's usability. In Chapter 3, we extend our base CAPTCHA's interactivity task to include a second challenge which further strengthens the CAPTCHA and turns it into an interactive game. We also assess the impact of this added interactivity on the CAPTCHA's security and usability. In Chapter 4, we analyze the security of our base CAPTCHA, i.e., without

the added interactive game. In Chapter 5 and Chapter 6, we propose and test additional provisions to further enhance the CAPTCHA's security and usability in order to customize it for applications that require still higher security or usability. We conclude this thesis in Chapter 7 with a discussion of our main results.

Publications based on this work include:

- Chapter 1 of this dissertation is derived from:
 - o N. Roshanbin and J. Miller, "A survey and analysis of current CAPTCHA approaches," *J. Web Engineering*, vol. 12, no. 1, pp. 1-40, 2013.
- Sections 2.5 and 2.6 are derived from:
 - o N. Roshanbin and J. Miller, "Finding Homoglyphs - A Step towards Detecting Unicode-Based Visual Spoofing Attacks," in *Web Information System Engineering - WISE*, 2011, pp. 1-14.
- The majority of Chapter 2 is derived from an article submitted for publication:
 - o N. Roshanbin and J. Miller, "ADAMAS: Interweaving Unicode and Color to Enhance CAPTCHA Security," *J. Future Generation Computer Systems*, submitted for publication.
- Chapter 3 and the appendix of this thesis has been submitted for publication:
 - o N. Roshanbin and J. Miller, "Enhancing CAPTCHA security using interactivity, dynamism, and mouse movement patterns," *International Journal of Information Security*, submitted for publication.
- An article has been submitted for publication which uses material from Chapter 5:
 - o N. Roshanbin and J. Miller, "A comparative study of the performance of local feature-based pattern recognition algorithms," *J. Pattern Recognition*, submitted for publication.

Chapter 1 An Overview of Existing CAPTCHA Systems

1.1 Introduction

A CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) is a test that attempts to distinguish human users from computers or robots. Such tests are usually based on hard, open artificial intelligence problems such as the recognition of distorted text. The idea of a CAPTCHA comes from the “Turing test” [1]. While a Turing test is designed to act as a measure of progress for AI and aims to assess whether a machine can exhibit intelligent behaviour that is indistinguishable from humans, a CAPTCHA aims to distinguish machines from humans and acts as a security mechanism. Hence, a CAPTCHA is sometimes referred to as a “reverse Turing test”.

CAPTCHA algorithms must have two characteristics. First, the P for Public in the CAPTCHA acronym means that the algorithm used to design the CAPTCHA must be publicly available. This condition was established to demonstrate that breaking the CAPTCHA requires solving a complex artificial intelligence problem rather than having access to the CAPTCHA’s algorithm [1]. Second, different instances of a CAPTCHA need to be substantially different; otherwise, they might be recorded, solved by humans, and then used to answer future challenges. Hence, they should be generated pseudo- randomly from a very large space of distinct challenges [2].

Using a CAPTCHA as a security mechanism can prevent malicious programs from gaining illegitimate access to resources such as signing up for thousands of accounts, posting hundreds of comments in weblogs and so on. Some common types of robot program (a.k.a. bot) abuses include [3]:

- Voting bots can cast thousands of votes masquerading as humans in online polls.
- Account registration bots can sign up for thousands of accounts with different types of service providers;
- Email spam bots can automatically send out thousands of spam messages;
- Weblog bots can post comments in weblogs pointing both readers and search engines to irrelevant sites;
- Search engine bots can automatically register web pages to raise their rankings in a search engine;
- Chat room bots [4].

The applications of CAPTCHAs as a security mechanism are not limited to the above cases and new applications will emerge as novel security threats come into existence by virtue of the increasing processing power of machines. The remainder of this chapter is organized as follows. Section 1.2 provides an overview of several CAPTCHAs which have appeared in the literature. Section 1.3 describes mechanisms or methods to break the existing CAPTCHAs. Sections 1.4 and 1.5, respectively, discuss the robustness and the usability of these CAPTCHAs. Finally, in Section 1.6, based on the discussions in previous sections, existing CAPTCHA types, attack types and CAPTCHA vulnerabilities are summarized.

1.2 Current CAPTCHAs

In general, CAPTCHA methods can be divided into five groups [5]:

- Text-based CAPTCHAs,
- Image-based CAPTCHAs,
- Audio-based CAPTCHAs,
- Motion-based CAPTCHAs, and
- Hybrid CAPTCHAs.

In text-based systems, distorted word characters are rendered as an image and presented to users. Then, the users are asked to type the answer that requires identifying all characters in the correct order. Because the image contains visual effects, it is difficult for a computer to recognize the words. Text-based CAPTCHAs have the weakness of being deciphered by OCR software. In order to overcome this weakness, other types of CAPTCHAs have been introduced. Image-based CAPTCHAs usually take advantage of the superiority of humans over computer vision systems in identifying object types in an image. Although it is more convenient for human to solve image-based rather than text-based CAPTCHAs, the latter requires a large database of images which is hard to prepare automatically.

An audio-based CAPTCHA picks a string, renders it to a sound clip and presents it to users who are asked to recognize the contents of the audio clip. According to a large scale study on the usability of CAPTCHAs, audio-based CAPTCHAs are more problematic than other types [6]. Another category is motion-based CAPTCHAs in which a movie or animation is presented to the users and they are asked to recognize an action, animated word or image in the movie. This CAPTCHA is convenient for users. In addition, since the required processing time in this CAPTCHA is relatively high, it is more secure. However, the high loading time can be a disadvantage from a usability viewpoint. Another disadvantage is the requirement of a large database of animations. Finally, we selected the term “hybrid CAPTCHA” for a CAPTCHA that is a combination of different types or a CAPTCHA that was designed for special purposes.

The remainder of this section introduces current CAPTCHAs that appear in the literature. The order of their introduction is not the same as the chronological order of their development. Introducing a CAPTCHA in a specific group does not mean that it does not have any common features with other groups. Occasionally, a CAPTCHA might belong to more than one category.

Some of the introduced CAPTCHAs are highly developed and used in real world applications, others are just ideas briefly sketched out by their authors. Regardless of their maturity, we present the basic idea behind each CAPTCHA in an effort to illustrate the wide variety of ideas and directions researchers are actively pursuing. Later sections will address the contribution, usability and robustness of these systems. Clearly, CAPTCHAs that have not been developed and are at the idea stage will receive less attention in these later sections than their fully developed counterparts.

1.2.1 Text-based CAPTCHAs

A text-based CAPTCHA is a distorted image of a sequence of characters on which different types of degradation, background clutter and color mixtures are applied to make it more challenging for attackers. We will introduce current text-based CAPTCHAs in six sub-groups:

- CAPTCHAs with “English words” as their CAPTCHA text,
- CAPTCHAs with “random strings” as their CAPTCHA text,
- CAPTCHAs based on handwritten text,
- CAPTCHAs based on linguistic knowledge,
- CAPTCHAs that require user interaction,
- Non-English CAPTCHAs.

CAPTCHAs with “English words” as their CAPTCHA text: In some CAPTCHA systems, such as Gimpy, EZ-Gimpy, CAPTCHAService.org, PessimPrint and reCAPTCHA, the CAPTCHA image contains English word(s).

Gimpy: This CAPTCHA was developed in collaboration with Yahoo! with the aim of protecting chat rooms from spammers to make them unable to post classified ads and write scripts to generate free e-mail addresses. Gimpy picks seven words from a dictionary and renders

a distorted image containing those words. Users are required to type three of the shown words in order to gain entry to the service [7].

EZ-Gimpy: This CAPTCHA, from Carnegie Mellon University, works as follows. First, a word is chosen from a dictionary. In the next step, the word is rendered to an image using various fonts and different types of distortions such as adding black or white lines, background grids and gradients, linear and non-linear deformations, blurring and pixel noise. Then, the user is asked to type the word [8].

CAPTCHAservice.org: In this CAPTCHA, each challenge is a six-letter English word chosen from a set of 6000 words. The distortion method used is random horizontal and vertical shearing [9].

PessimPrint: PessimPrint concentrates on degradations, such as adding noise to or blurring the images to defeat OCR techniques; the designers of this CAPTCHA argue that under the conditions of inferior image quality, there is a significant gap in pattern recognition ability between humans and machines [2]. This CAPTCHA works as follows. First of all, a word is pseudo-randomly selected from a fixed list containing 5-to-8-letter English words. Then, it is rendered with a typeface (from a fixed list of 5 fonts) and a fixed font size (size=8). Finally, a set of image degradations including x-scaling, y-scaling, skewness, blurriness and noise additions are applied to the image.

reCAPTCHA: This CAPTCHA [10] selects its words from old printed material or scanned text that cannot be recognized by OCR programs. This strategy not only increases the security of the CAPTCHA, but also uses the solutions provided by human users to decipher non-digital text. This CAPTCHA shows two words to the user; one that OCRs were not able to decipher and another ‘control’ word with a known answer. If the user enters the control word correctly, he/she

is assumed to be a human and his/her answer to the other word is considered as a correct answer. If a specific number of users' answers to an unknown word match, that word becomes a control word. This type of CAPTCHA, in which there is no specific answer to the question asked from the user, is referred to as 'collaborative filtering CAPTCHA' [11].

CAPTCHAs with “random strings” as their CAPTCHA text: using English words in some current CAPTCHAs makes them vulnerable to dictionary attacks. The solution for this issue is to use random strings instead of words. This technique is utilized by MSN CAPTCHA, Yahoo!, Ticketmaster, Google, etc.:

Hotmail or MSN CAPTCHAs [12]: This CAPTCHA, used in the Hotmail email service registration, selects eight English characters (upper case letters and digits); then, after applying local and global warping, renders the characters with dark blue color on a light gray background. In the next step, three types of arcs are randomly added to make segmentation difficult. The arcs include: “Very thick arcs” (the same as the characters) of foreground color that do not intersect characters, “Thick arcs” of foreground color that intersect characters , and “Thin arcs” of background color that cut characters and remove some of their pixels.

Yahoo! CAPTCHA (Yahoo! version2): Starting in August 2004, Yahoo! introduced its second generation CAPTCHA. Its characteristics include using a string of characters instead of English words, containing only black and white colors, using both letters and digits, and having connected lines and arcs as clutter. [13].

Ticketmaster: www.ticketmaster.com, which is a well-known ticket sales and distribution website, uses a CAPTCHA characterized by crisscrossing lines at random angles [13].

Google/Gmail: The specifications of this CAPTCHA, used by Gmail.com, include: using only image warping for character distortion, having only two colors (one for foreground and the

other for background), locating characters close to each other and following a curved baseline [13].

BaffleText [14]: This CAPTCHA has its roots in Gestalt theory. While computers are not good at recognizing images occluded or interfered by random shapes, humans can distinguish an entire shape or image regardless of its incomplete information according to Gestalt theory [15].

According to the Gestalt laws of perception, the human brain interprets images in their entirety before perceiving their individual parts. Based on this theory, what a human sees when looking at an image is an effect of the whole image, which is more than the sum of its parts. Gestalt principles include proximity, similarity, symmetry, continuity, closure, figure and ground [15, 16]. Proximity refers to how elements located close together tend to be perceived as a group. Similarity occurs when similar objects are grouped together. Symmetry refers to the tendency to group objects according to their symmetry and meaning. Continuity occurs when an eye moves over one object and continues to another. Closure occurs when some shape information is missing, but it can be perceived by the human brain's ability to fill in the missing information. Figure & ground refers to how the human eye differentiates an object from its surroundings. The object is perceived as *figure*; and the surrounding area is perceived as *ground*.

Based on the closure principle of the Gestalt theory, if some portions of characters in a CAPTCHA test are removed, humans are good at inferring the whole picture from only partial information, while machines are not. *BaffleText* takes advantage of this ability of the human brain and uses random masks to obliterate parts of test characters to make a stronger CAPTCHA.

This CAPTCHA chooses non-English pronounceable character strings (instead of English words), selects a font type, produces a mask image (by selecting a mask shape and radius) and a

random mask operation (addition, subtraction and difference) and combines this mask with the character-string image [14].

ScatterType: ScatterType CAPTCHA selects its challenge word from a set of 15,000 English-like nonsense words. Then the algorithm applies a font (from 100 different font types) to it. The image of each character is fragmented using horizontal and vertical cuts and fragments are forced to drift apart until it is difficult to resemble them into characters. In order to achieve the human legibility, some characters with highest confusability ('q', 'c', 'l', 'o', 'u') are removed [17].

Other examples of text-based CAPTCHAs with random strings as their CAPTCHA text include *eBay* [18], *PHP-class*[19] and *MegaUpload* [20] CAPTCHAs.

Sequenced tagged CAPTCHA [21]: In this CAPTCHA, all characters are tagged with numbers and the user is asked to enter the characters based on the logical ordering of their tags. This strategy adds a new layer of security to the CAPTCHA. Variants of this CAPTCHA include characters as base text tagged by numbers, numbers as base text tagged by letters and a hybrid scheme [22].

CAPTCHAs based on handwritten text: While most current text-based CAPTCHAs use machine-printed text, which makes them vulnerable to pattern recognition attacks, there are CAPTCHAs that use handwritten text in their challenges. An example is Handwritten CAPTCHA.

Handwritten CAPTCHA [16]: This CAPTCHA is based on distorted handwritten text. The developers of this CAPTCHA discuss that according to Gestalt laws of perception and the Geon theory of pattern recognition, the interpretation of distorted handwritten text is easy and reliable for humans but difficult for automated programs. Gestalt theory, argues that humans are able to

infer the whole picture from partial information [15] which helps them to recognize distorted images. According to the Geon theory, humans recognize objects by separating them into geons (simple forms such as cylinders, cones and wedges). In human perception of occluded images of words, decomposing a word into known and unknown visual elements allows users to recognize characters by using rules such as: “words contain specific visual elements”, “combination of letters follows specific rules” and “words convey meaning” to recognize the entire word [16].

The designers of this CAPTCHA investigated the effects of different transformations including overlapping, adding occlusions, splitting the image in parts and displacing the parts, changing word orientation, etc. on the usability and security of their CAPTCHA. They designed an algorithm to generate cursive English handwritten text synthetically. They used existing character images and performed auto-scaling, automatic baseline determination, ligature parameterization, ligature joining, skeleton perturbation and skeleton thickening to generate synthetic handwritten words [23].

CAPTCHAs based on linguistic knowledge: Some current CAPTCHA systems combine an OCR problem with linguistic knowledge in order to strengthen their tests. Examples of such CAPTCHAs include *semCAPTCHA*, *odd-words-out*, *number-puzzle-text CAPTCHA*, *SS-CAPTCHA* and *text-domain CAPTCHA*:

SemCAPTCHA [24]: In this CAPTCHA, three words, which are names of animals, are displayed to the user; and the user is asked to click on the one which belongs to a different category from the other two (mammals, reptiles, etc.). The three words are graphically similar, but semantically dissimilar.

‘Odd-words-out’ and *‘Number-puzzle-text’ CAPTCHAs*: These two CAPTCHAs have been proposed by CAPTCHAService.org. “odd-words-out CAPTCHA” presents a list of words to the

user who is asked to select the words that are not related to the general category of the list. “Number-puzzle-text CAPTCHA” provides the user with a textual description of a number and the user is asked to enter the number [9].

‘Strangeness in sentences’ CAPTCHA: This CAPTCHA [25] displays a list of sentences to the user including natural and wrong sentences. The human user is able to detect natural sentences. Natural sentences are collected from newspapers, magazines and books. Wrong sentences are created by getting a machine translator to translate a natural sentence from a mother-tongue language into a non-mother-tongue language and retranslate the result to the mother-tongue language (e.g. Japanese → English → Japanese). This translation strategy usually produces wrong sentences which are strange for human users.

Text-domain CAPTCHA: In this CAPTCHA, a list of sentences is shown to a user who is required to find the sentences that are meaningful replacements of each other [26]. Many of the answers might be semantically correct, but contextually incorrect. Deciding which sentences are meaningful alternatives based on the context is much easier for a human than a computer.

CAPTCHAs that require user interaction: Requiring users interaction with the computer while solving a CAPTCHA can improve the robustness of the CAPTCHA. In this type of CAPTCHA, a user is required to enter the answers using the mouse or other physical devices instead of keyboard. This type of text-based CAPTCHAs is a sub-category of interactive CAPTCHAs which are more extensively discussed in Section 2.2. Drag-n-Drop CAPTCHA, Physical CAPTCHA and iCAPTCHA are examples of this type of CAPTCHA which we briefly introduce here:

Drag-and-drop CAPTCHA [27]: This CAPTCHA uses mouse actions to differentiate between computers and humans. In this CAPTCHA, the test is shown to the users and they are asked to drag and drop character blocks into their respective blank blocks sequentially.

Physical CAPTCHA: Golle and Ducheneaut [28] proposed a physical hardware-based CAPTCHA test to differentiate between human game players and computers based on the fact that only humans are able to interact with the physical world (e.g. pressing a button, tapping on a touchscreen, or moving a joystick). Since different games receive their inputs from various hardware (e.g. keyboard, joystick, etc.) and there is not a general input device, the designers of this CAPTCHA decided to develop a new cheaper generic hardware only for human verification which is called a “CAPTCHA token”. This token, which comes in combination with a game, consists of a keypad, a screen and a CPU.

iCAPTCHA [29]: This CAPTCHA is an interactive CAPTCHA designed to resist third party human attacks. In this CAPTCHA, the user has to submit the solution step by step when prompted by the CAPTCHA. This iterative back and forth traffic between client and server increases the timing difference between a third party human solver and a real user which helps to detect attacks.

Non-English CAPTCHAs: Besides English CAPTCHAs, some CAPTCHAs have been developed in other languages. One reason for producing localized CAPTCHAs is the prevalence of attacks against current English CAPTCHAs. Another reason is that people are more comfortable with solving tests in their own languages. An example is the Arabic CAPTCHA:

Arabic CAPTCHA: Khan et al. [30] proposed an Arabic CAPTCHA to be employed in 21 countries that use the Arabic alphabet. This CAPTCHA exploits the weakness of Arabic OCR systems in segmentation, which results from special characteristics of the Arabic language.

These characteristics include dependency of the shapes of the letters on their positions in the word, the existence of a variety of diacritics in Arabic and different number of dots for letters as well as different positions of dots (many Arabic characters have the same shapes but a different number of dots at different locations). This CAPTCHA picks 4 to 9 Arabic letters, randomly selects a font, and adds noise and background clutter which might be confused with dots and diacritics. Other techniques used to strengthen this CAPTCHA system include adding a shadow of the word, character overlapping, and changing the location of the word in the image. A similar Persian-Arabic CAPTCHA had been designed by Shirali-Shahreza et al. [31].

1.2.2 Image-based CAPTCHAs

In this type of CAPTCHA, an image is displayed to the users and they are asked a question about the contents of the image. A variety of current image-based CAPTCHAs are discussed in this section in 6 sub-groups:

- Object detection CAPTCHAs: based on detecting a certain object among other objects,
- Subject detection CAPTCHAs: based on detecting the common characteristic or the subject of all images,
- Part detection CAPTCHAs: based on detecting a specific part of the test image,
- Swapping task CAPTCHAs: based on swapping the misplaced parts of the test image,
- Orientation task CAPTCHAs: based on fixing the orientation of disoriented objects,
- 3D CAPTCHAs.

Object detection CAPTCHAs: based on detecting a certain object between other objects: In this group of image-based CAPTCHAs, the user is asked to distinguish a certain object between n objects. Examples of this group include Collage CAPTCHA, Asirra, and Imagination.

Collage CAPTCHA: Collage CAPTCHA [32] selects pictures of six different objects, applies distortion effects such as rotation to the images and then merges them to create a single image. This image is presented to users and they are asked to click on a certain picture. For example, an image containing an airplane, a car, an apple, an orange, a pineapple, and a ball are displayed, and the user is asked to click on the image of the car. A similar CAPTCHA is proposed in [33] with the only difference that [33] provides a multilingual user interface.

Asirra: This CAPTCHA [34] asks users to identify cats in a set of 12 images of cats and dogs.

Imagination [35]: Imagination is an image-based CAPTCHA in which the test contains 8 different images located in 8 random orthogonal partitions of a rectangular area. Partitioning and dithering techniques are used to cut the original image tiling and create many false boundaries, which makes segmentation difficult for computers. In the click step, a user is asked to click on the center of one of the 8 images. In annotate step, the chosen image is distorted and displayed to the user to be annotated. The designers of this CAPTCHA have studied the effect of applying a variety of distortions (such as changing luminance, quantization level, dithering levels) on human and machine recognition [36].

Subject detection CAPTCHAs: based on detecting the common characteristic or the subject of all images: In another group of image-based CAPTCHAs including PIX, Bongo, and activity recognition CAPTCHA, a user is asked to distinguish a particular characteristic which is common across all shown objects:

PIX: In this CAPTCHA, a library of pictures with different subjects is prepared. A number of these pictures from within a single category or subject are selected and presented to the user; and the user is asked to select a phrase expressing the common subject of these pictures. For

example if the pictures presented are a globe, volleyball, planet and baseball, the common subject would be “ball” [12].

Bongo: This CAPTCHA uses two sets of images; each set has some particular characteristics. For example, one set is boldface, while the other is not. The system then presents a single image to the users and asks them to specify the set to which the image belongs. Because the number of possible solutions is small, this CAPTCHA is not highly robust to brute-force guessing. However, strategies such as cascaded multiple Bongo CAPTCHAs can improve the security of this CAPTCHA [1].

Activity recognition CAPTCHA: Vimina et al. [37] designed a CAPTCHA in which three distorted images of a common activity are displayed to the user. The user is required to detect the activity and annotate it from a list of activities.

Part detection CAPTCHAs: based on detecting a specific part of the test image:

Another type of CAPTCHA asks users to detect and click on a specific part of the test image. Implicit CAPTCHA, drawing CAPTCHA, Line CAPTCHA, and Artificial are of this type:

Implicit CAPTCHA [38]: This CAPTCHA asks a user to click on a specific part of the image. For example, the user is required to click on the mountain climber’s glasses.

Drawing CAPTCHA: Drawing CAPTCHA [39] displays numerous dots on a screen to the users and asks them to connect dots with a certain shape to each other. The designers of this CAPTCHA argue that computers have difficulty in recognizing the targets from the noise; however, it is easy for a human user to identify the special dots and connect them to each other. An advantage of this CAPTCHA is that it does not require any special knowledge or ability.

Line CAPTCHA: In this system [40], a blurred or randomly segmented line is presented to the user who is asked to drag the mouse along the line.

Artificial: This CAPTCHA is based on human face recognition [41]. In this CAPTCHA, a distorted face embedded in a background that includes face-like clutter is shown to the user. The user must detect the face and click on the eye corners and mouth corners.

Swapping task CAPTCHAs: based on swapping the misplaced parts of the test image:

Some CAPTCHAs ask users to exchange misplaced blocks of the image to reproduce an original image. “Exchanging image blocks CAPTCHA” and Jigsaw puzzle CAPTCHA are examples of this group:

Exchanging-image-blocks CAPTCHA: Liao [42] proposed a CAPTCHA based on swapping the contents of two misplaced non-overlapping regions in an image.

Jigsaw puzzle CAPTCHA: Gao [43] designed a CAPTCHA using jigsaw puzzles in which the user is required to solve a puzzle by swapping the two misplaced pieces.

Orientation task CAPTCHAs: based on fixing the orientation of disoriented objects: In this group of CAPTCHAs, such as Image Flip CAPTCHA, What’s up CAPTCHA and Sketcha, users are required to detect the correct orientation of the objects in the test image:

Image Flip CAPTCHA [44]: This CAPTCHA displays an image composed of several sub-images to the user. The user has to detect all non-flipped images and click on them.

What’s up CAPTCHA: This CAPTCHA, proposed by Gossweiler et al. [45] asks users to detect the orientation of the randomly-rotated test image and use a slider to rotate the image to its upright position.

Sketcha [46]: Sketcha is another orientation-based CAPTCHA that shows the users a set of images which are line drawings of 3D models. The user must detect the upright orientation of each image. Detecting orientation in some images including symmetric images, and images that are typically oriented upside down is difficult (or impossible) for human users. Hence, the designers of this CAPTCHA applied a filter to remove these groups of images from the database of the CAPTCHA. The filtering process is as follows: in a user study, each user is shown some test images. If a certain number of users answer an image inconsistently, that image is considered as a difficult image and removed from the database.

Sub-image orientation CAPTCHA: Kim et al. [47] proposed a CAPTCHA based on the orientation of sub-images. The designers of this CAPTCHA argue that while a whole-size photo contains semantic cues such as sky, grass and dark brown ground that help a machine to detect image orientation, random sub-images do not include meaningful objects that a computer program can easily recognize. In this CAPTCHA, a number of random blocks of an image are cropped, rotated and shown to the user who is asked to find their correct orientations.

3D CAPTCHAs: Working with 3D images instead of 2D ones can reduce the probability of pattern recognition and database attacks. This strategy is used in “2D CAPTCHA from 3D models”, Spamfizzle and 3D CAPTCHA:

2D CAPTCHA from 3D models: This CAPTCHA, designed by Hoque et al. [48], has a database which is populated with several 3D models. To create a new test, this CAPTCHA selects a 3D model, applies different distortions and lighting effects on it and transforms it to a 2D image. This image is displayed to the user to be identified. Since infinite number of 2D images can be gained from a 3D model, the image database for this CAPTCHA is very large.

Spamfizzle [49]: In this CAPTCHA, 3D objects are designed manually and their attributes and behaviors are described and recorded. To produce a new test, the CAPTCHA generation algorithm creates a 3D image by combining different objects and labeling the attributes of each object. The image is displayed to the user who is asked to enter the labels that correspond to a list of attributes.

3D CAPTCHA: Imsamai and Phimoltares [50] propose a 3D CAPTCHA based on the idea that recognizing a sequence of 3D characters is easy for the human, but difficult for computers. This CAPTCHA adds rotation, overlapping, noise, scaling, font variation and background texture and special characters to the image to make it stronger.

1.2.3 Audio-based CAPTCHAs

Audio-based CAPTCHAs are usually used as a complement for text-based CAPTCHAs. Many popular websites such as *eBay*, *Yahoo!* and *Microsoft* use both visual and audio CAPTCHAs [51]. An audio CAPTCHA generally picks a random sequence of letters or numbers; renders them into a sound clip; includes some level of distortion; and then presents the recording to the users. The user is asked to type the contents of the recording [1]. In one type of audio CAPTCHAs, known as spoken CAPTCHA, the users are required to utter the response instead of typing it. This feature makes this CAPTCHA also suitable for blind users [52].

Including noise in audio-based CAPTCHAs can improve their security. Chan [53] discusses that adding background noise to sound CAPTCHAs decreases the accuracy rate of a speech recognizer more than that of a human. Sauer et al. [54] adds silence to the sound in order to discourage checksum/signature based attacks. Kochanski et al. [55] uses 18 different sets of distortion to sound in order to make the problem difficult for machines. However, According to Bursztein et al. [51], among various types of noise that can be incorporated in an audio-based

CAPTCHA, semantic noise, i.e. a noise with similar characteristics of a spoken signal, is the most effective noise in improving security.

1.2.4 Motion-based CAPTCHAs

In this type of CAPTCHA, a movie or animation is shown to the users and they are asked a question about the contents of the clip. Answering the question usually requires recognizing an action, animated string or an image in the clip. Examples of motion-based CAPTCHAs include NUCAPTCHA, HelloCAPTCHA, animation CAPTCHA, and 3D animation CAPTCHA.

NUCAPTCHA: In this CAPTCHA [56], a string is animated from right to left and the user is asked to type the last word.

HelloCAPTCHA: A test in HelloCAPTCHA [57] consists of six letters or digits displayed in an animated GIF image and the user is required to type the characters.

Animation CAPTCHA: In this CAPTCHA [58], a few animated objects are shown to the user who is required to detect and click on one of the objects. The movement of objects along a random path, rather than having a static image, makes a CAPTCHA more secure against random guessing or segmentation attacks.

3D animation CAPTCHA: This CAPTCHA [59] selects three English letters or digits, renders the string to an image in which character area is covered by '1's and background area is covered by '0's. In the next step, the algorithm adds a third dimension to the image using a sinusoidal function and applies waves to this 3D animation. This CAPTCHA also changes colors randomly during the presentation of a test.

1.2.5 Hybrid CAPTCHAs

This section provides examples of CAPTCHAs that combine text and multimedia in their systems. Moreover, CAPTCHAs designed for special purposes such as smartphones or for people with disabilities are discussed in this section.

Dynamic CAPTCHAs: While most existing CAPTCHA systems use one or two pre-defined types of CAPTCHA, e.g. text-based, image-based, etc., a dynamic CAPTCHA selects a type of CAPTCHA among different available CAPTCHA types based on the information the user entered in the first steps of registration or the information provided by the user's web browser. An example of this group of CAPTCHAs is Dynamic CAPTCHA.

Dynamic CAPTCHA [60]: This CAPTCHA selects a type of CAPTCHA based on the limitations of a user's device (in entering characters or numbers) or a user's restrictions and disabilities (such as blindness), the native language of the user, etc.

CAPTCHAs with multiple challenges: Asking users to pass multiple challenges, probably of different types, instead of a single test improves robustness. 'Multiple challenge-response system' [61] is an example of this type of CAPTCHA.

Multiple challenge-response system: Longe et al. [61] designed a CAPTCHA in which the user has to solve multiple challenges instead of a single test. The challenges in this CAPTCHA include: a mathematical test and an image CAPTCHA. The mathematical test consists of alphanumeric characters and symbols; the user is expected to distinguish numbers and add them together. The image CAPTCHA is a distorted image of a random string of characters and digits.

Multi-type CAPTCHAs: A CAPTCHA can be a mixture of different types of CAPTCHA; for example a combination of text- and image-based CAPTCHAs. Question-based CAPTCHA and tree-based handwritten CAPTCHA are examples of this CAPTCHA group.

Question-based CAPTCHA: This CAPTCHA is a hybrid CAPTCHA proposed by Shirali-Shahreza [62]. In this CAPTCHA, the user is shown a test which is a mathematical question created by a combination of text and images; then they are asked to enter the answer.

Tree-based handwritten CAPTCHA: This CAPTCHA, which is a combination of text and graphics, uses the human's ability to read handwritten text and understand tree structure to create a CAPTCHA. In this CAPTCHA, a synthetic handwritten tool creates the words, some random transformations are applied to them, and then, they are combined with a randomly generated tree structure and random test questions. The distorted handwritten words constitute the nodes of the tree and the random symbols are located in the middle of the branches. A random question such as "which word is connected to center by a line marked with a circle?" is asked from the users. It is easy for humans but difficult for computers to answer such a question [63].

CAPTCHAs designed for smartphones or other mobile devices: The rise in the popularity of smartphones and using them for browsing the Internet highlights the need to design new CAPTCHA systems that consider the special characteristics of mobile devices. These characteristics include the presence of a touch screen and the difficulty in using the keyboard, etc. CAPTCHAs developed to be used on smartphones are usually not required to be as robust as those designed for PCs because of the lack of powerful processors and small memories. Examples of this group of CAPTCHA include CAPTCHA zoo and Highlighting CAPTCHA.

CAPTCHA Zoo: This CAPTCHA [64], which is designed for mobile devices, is based on the fact that humans are superior to machines in recognizing similar objects in images. In this CAPTCHA, the challenge image contains two visually similar types of animals: target animals and noise animals. Different colors, lighting, rotation and overlapping are applied to the image. Then the user is asked to recognize the target animals.

Highlighting CAPTCHA: In this system, a random string on a noisy background is shown on the screen of the mobile device and the user is asked to highlight the characters with a stylus [12].

CAPTCHAs designed for people with disabilities: Examples of this group of CAPTCHA include ‘Localized CAPTCHA for linguistically-challenged individuals’, ‘Image/audio CAPTCHA’ and ‘a CAPTCHA for hearing-impaired people’.

Localized CAPTCHA for linguistically-challenged individuals: This CAPTCHA makes a bank of names of different objects; then chooses six of them and searches and finds Yahoo! images for these six names; shows the user one image for each name. Afterwards, this CAPTCHA selects one name from the six alternatives, says it (using speech) to the user and requests the user to click on the related image [65]. The designers argue that this CAPTCHA is easy for a human because it is appropriate for all ages, there is no need for typing abilities, it is in the user’s native language; hence, no English language knowledge is required.

Image/audio CAPTCHA [66]: This CAPTCHA combines pictures of objects with sounds associated with those objects. Since the audio is related to the image, this type of CAPTCHA can be useful for people with vision or hearing impairments. Another CAPTCHA, designed for blind people, contains a simple mathematical problem which is created and converted to speech using a text-to-speech system. The user is asked to listen to the sound and answer the question [67].

CAPTCHA for hearing-impaired people [68]: This CAPTCHA presents a word to the user using sign language and asks the user to recognize the word and choose its name from a list of words.

1.3 Breaking CAPTCHAs

In this section, we will explore attempts to break the CAPTCHAs presented in Section 1.2. However, many of the presented CAPTCHAs are no more than an idea and hence do not provide sufficient details with regard to their realization to allow an analysis of their robustness. This limitation also applies to Sections 1.4 and 1.5 where we discuss the robustness and usability of CAPTCHAs. Hence, the remainder of this chapter will focus on those systems that have actually been implemented.

1.3.1 Algorithmic approaches to break CAPTCHAs

The general process in attacking CAPTCHAs is composed of three steps: pre-processing, segmentation, and recognition. The purpose of the first step is to eliminate background noise. In the second step, the location of each object is found and in the third step, each object is recognized. Research suggests that segmentation is more difficult than recognition for machines [69]. Therefore, a CAPTCHA, which is designed to be segmentation-resistant, is less vulnerable to attacks.

In this section, general segmentation and recognition attacks are discussed and then, examples of specific attacks against current CAPTCHAs are explained. As mentioned before, the majority of available CAPTCHAs are text-based CAPTCHAs and most attacks have been designed against this type of CAPTCHA. Hence, most strategies and attacks discussed in this section are related to text-based CAPTCHAs.

Pre-processing attacks

This step is used to separate foreground from the background, remove background patterns, and eliminate noise as much as possible. Background noise includes background mesh, thin or thick arcs, etc. Noise removal can be performed by exploiting the difference between color, size,

shape, location, or motion patterns of noise components and target objects. Examples of strategies used in the pre-processing step are explained in this subsection.

Background noise removal based on the color difference between the noise and targets

A distinction between the color of foreground and background noise can be exploited to detect noise and remove it [70].

Background noise removal based on the difference between the size, shape, or locations of targets and noise

Any pattern in the size, shape, or location of the noise can help remove them. For example, noise components that are smaller (i.e. have a smaller pixel count), line-shaped, and closer to boundaries than the characters, can be identified more easily.

Morphological image processing (erosion, dilation, opening, closing, [71, 72] and skeletonization [73]): This method has been widely used to remove thin arcs or other background noise with different *sizes and shapes* than the targets. For example, after binarization² of the image, the vertical and horizontal line pixels that do not have neighboring pixels are considered as background mesh pixels and can be removed [72].

Vertical and horizontal histograms: This strategy can be used to detect any types of arcs in the image. For example, if in the x-histogram of an image, consecutive columns have only a limited number of foreground pixels; it is very likely that those columns belong to an arc. The same concept is true for a y-histogram [69].

Neural networks: Neural network methods can also be used to distinguish noises that have a specific shape [74].

² Converting the image into a black and white image using thresholding.

Background noise removal based on different moving patterns of the noise and targets

In some animated CAPTCHAs, for usability purposes, there is a distinction between moving patterns of noise components and targets. For example, characters may be displayed at certain fixed locations for a longer time compared to the noise components to get the human attention [75]. The shorter *display period* of noise helps an attacker to find the noise and remove it.

Segmentation attacks

Most current segmentation attacks use information about the positioning, color, size, moving patterns or other features of target objects to segment them. In this subsection, well-known segmentation attacks such as color-filling, vertical histogram and snake segmentation as well as less general methods designed to segment specific CAPTCHAs are briefly explained. Generally, a combination of these methods is utilized to break a CAPTCHA.

Segmentation based on the location of target objects

Information about the location of the target objects, e.g. whether they are connected to each other, located in certain positions of the image, or arranged horizontally or vertically, can help an attacker to segment them. A few examples of this type of attack are introduced in this subsection.

Color filling segmentation: This approach segments all the objects by finding connected components. In order to find a connected component, the algorithm at first detects a foreground pixel, and then traces all of its foreground neighbors until all pixels in this connected component are traversed [69].

Simple segmentation: If the number of characters in a test and their widths are constant, the whole test image can be divided into parts of the same width, each part being a character. This strategy can also be used in combination with other segmentation techniques (for example, when

a histogram is not able to completely segment the characters and instead, returns “chunks of characters”) [69].

Vertical segmentation: Using simple vertical histograms ([69] and [76]), or complex/innovative versions of them [73] to segment an image into its characters. A threshold can be used; histogram bins having values of less than the threshold show the space between characters (segmentation lines). This threshold can be decided statically or dynamically [77].

Projection [78]: This method is similar to the vertical histogram. Horizontal projection of the first and the last character of a CAPTCHA test can help recognize and segment those characters since they only have one neighbor. Sometimes vertical-layered projection can help recognize and extract the characters as well. For example, a character that does not have many pixels in the upper layer and does have a thin layer of pixels in lower level can be an ‘L’.

Vertical slicing [70]: This attack is based on two vulnerabilities: 1) the test image contains only two colors; one for foreground and the other for background; 2) the letters are not vertically overlapped. A vertical slicing process traverses pixels from top to bottom and from left to right. Finding the first pixel with foreground color stops the search and determines the first vertical segmentation line. The process then continues to find the next vertical segmentation lines and segment all characters.

Snake segmentation [70]: This type of segmentation is developed to segment CAPTCHAs whose letters may overlap vertically. A snake moves from top to button and tries not to touch the letters; it can move in four directions (Up, Right, Left, Down) when required.

Middle-axis point separation [79]: The middle-axis points are the white background pixels that are horizontally located in the center of two disconnected black foreground pixels.

Intersecting lines which are formed by connecting these points together can be used to segment characters.

Drawing splitter lines parallel to the angle of the test image [80]: This attack is designed to segment characters in a CAPTCHA with a skewed image of characters. In this attack, segmentation lines are drawn parallel to the angle of the test image or to the side surface of the characters.

Segmentation based on the features of target objects

The distinction between features of noise and target objects is the basis for the approaches introduced in this subsection.

Loop detection [78]: In many text-based CAPTCHAs, contrary to the characters, background noise does not contain loops. The relative positioning of detected loops can help computer programs segment the characters. For example, two loops that are located in the same horizontal place probably belong to an '8' or a 'B'. Also, the probable character-connection loops can be detected using this method.

Edge detection [81]: Edge detection algorithms are used to detect boundaries (sharp intensity changes in the image) of objects to segment them. In some image-based CAPTCHAs, e.g. Imagination [35], the pictures of several images are embedded in a single picture. If the boundary of images is not concealed appropriately, an attacker can use edge detection algorithms to segment them.

Interest points density evaluation [82]: This method was used against a CAPTCHA that has text as noise in background. Noise text was upright while test characters were rotated, distorted and overlapped. In this method, an attacker draws a bounding box around each connected

component and finds interest points of each bounding box by the SIFT (Scale Invariant Feature Transform) algorithm. The density of interest points was more in test characters since they were deformed. This method could help the segmentation of target text from background noise text.

Segmentation based on color information

The color difference between the foreground and background or between the objects can be another clue for an algorithm trying to segment a CAPTCHA. We describe some examples of these approaches here.

Extracting the characters by their colors [75]: If each character in the test has a distinct color which does not appear in the background as well, the characters can be segmented by their colors.

Thresholding [83]: This method is based on separating foreground and background color based on a threshold value for color. Pixels with values beyond this threshold (lower or higher) are removed as background noise.

Segmentation based on motion information

Although animated CAPTCHAs are developed to make segmentation harder for an attacker, motion information can sometimes help attackers distinguish target objects from noise. Examples of such attacks are introduced in this subsection.

Using motion tracking (optical flow) to segment characters [82]: In motion CAPTCHAs, where characters rotate over time, tracking the frames of animation and finding groups of points that move together, helps find the characters.

Using a pixel delay map to extract the characters [75, 84]: In some motion CAPTCHAs, target characters are displayed for a longer period of time in a certain location before they move.

This is because they need to be emphasized to get human attention, which can also help an attacker who tracks animation frames to segment the characters.

Using a catching line to extract the characters [75]: This attack is designed against a CAPTCHA whose characters jump or spring vertically. In this CAPTCHA, the entire character can be easily identified when the highest pixel of the character touches a horizontal line located below the upper image boundary. The attacker uses this feature to find the area which encompasses the whole character image.

Extracting the characters by frame selection [75]: In contrast with static CAPTCHAs that have only one image, animation CAPTCHAs have many frames. An attacker can analyze different frames of the animation and select the “best” frame to process. For example, frames in which characters do not overlap with each other or with the boundaries.

Recognition attacks

After segmenting objects of a test, a recognition step is required to detect each object. These types of attacks, which include object recognition attacks, pixel-count, dictionary and database attacks are discussed in this subsection.

Machine learning object recognition methods

A wide range of artificial intelligence object recognition algorithms including *OCR* systems [75, 85] and *classifiers* [73, 81, 86] can be used to recognize objects based on their features. Another machine learning method used for character recognition is *context shape matching* [87]. This method is based on “description over relative positions”. The context shape vector of one point in the object is a log-polar histogram (2D histogram of distance and angle) of that point relative to the other points of the object. In this method, an object is described as a set of context

shape vectors of a selected group of its points. Comparing this set with those of known templates (i.e., characters) helps recognize the test character.

Pixel-count attack

This attack is applicable to CAPTCHA systems in which each character has a constant pixel count and the pixel count of each character is different from that of other characters. In this attack, after segmentation, the number of foreground pixels of each segment is counted and compared with a previously determined pixel count of the objects used in the CAPTCHA to identify the object/letter in each segment [76]. For letters with the same pixel counts, the algorithm uses a simple geometric analysis to differentiate them. For example, to distinguish between a “P” and a “V”, the algorithm draws a vertical line in the middle of the normalized letter. If the line cuts through the letter in only one point, the character is a “V”; otherwise, it is a “P”. The same process is performed for other similar cases.

Dictionary attack

Using only the words of a specific dictionary in a CAPTCHA limits the number of possible strings. A dictionary attack against such a CAPTCHA searches the whole dictionary to solve the test. Dictionary attacks can also be used in combination with other attacks. For example, when the vertical histogram is not able to segment all of the characters of a test because of overlaps between them, the dictionary is searched for all possible candidates. The word with the same pixel-count as the challenge word is selected as the right answer [76].

Database attack

A database attack is a type of attack in which the entire database of objects used in a CAPTCHA is gradually collected and used to solve the challenge. Every time a challenge is

displayed, a portion of the database is uncovered and by solving enough challenges, the attacker would have access to the whole database [34].

Attacks on current CAPTCHAs

There are several image processing attacks designed against different existing CAPTCHA schemes. We provide examples of these attacks in chronological order: Mori and Malik [87] have broken the EZ-Gimpy (92% success) and Gimpy (33% success) CAPTCHAs with object recognition in 2003. In 2004, Chellapilla and Simard designed an attack against a number of CAPTCHAs using machine learning algorithms [72]. Moy et al. also developed distortion estimation techniques to break gimpy-r (78% success) and EZ-Gimpy (99% success) [88]. In 2007, Yan and El Ahmad proposed methods to break CAPTCHAservice.org CAPTCHAs (94% success) [70]. Golle [86] proposed a machine learning attack (10.3% success) on Asirra CAPTCHA in 2008. In the same year, Yan and El Ahmad designed an attack against Microsoft's CAPTCHA (60% success). In 2009, Decaptcha was designed to break the eBay audio CAPTCHA (75% success) by looking at voice energy spikes [89]. In the same year, Wilkins proposed an attack on reCAPTCHA (17.5% success) [90]. In 2010 more attacks against CAPTCHAs were performed. Another attack against reCAPTCHA was designed (11.8% success) [91]. MegaUpload CAPTCHA was broken (78% success) by El Ahmad and Yan using a segmentation attack. Huang et al. attacked CAPTCHAs with line cluttering and character warping by proposing a new segmentation algorithm (75% success) [79]. Two e-banking CAPTCHAs were attacked using color segmentation and pattern recognition algorithms proposed by Li et al. [71]. Milde attacked reCAPTCHA (11.6% success) using a holistic shape context word recognition algorithm [74]. Attacks against Imagination (4.95% success) and ArtiFacial (18% success) were developed by Zhu et al. [81] in the same year. More audio

CAPTCHAs, i.e. eBay, Authorize, Yahoo! and Microsoft Audio CAPTCHAs, were broken by Bursztein et al. [51] (45.5%-89% success). In 2012, Lorenzi et al. [92] attacked PIX and Asirra CAPTCHAs (respectively, 83.1%, 74.7% success) using neural networks. Yahoo CAPTCHA was attacked by Gan et al. [78] (54.7% success); and, Chandavale and Sapkal proposed an attack against text-based CAPTCHAs with connected or unconnected characters (85% success) [77]. Another attack on reCAPTCHA was designed by Perez et al. [73] (22.8% success rate). A 3D CAPTCHA called Teabag was attacked by Nguyen et al. [80] (76% success); and Lin et al. proposed an attack on drawing CAPTCHA (success 75%) [64]. Since 2012, breaking animated CAPTCHAs has started. Bursztein proposed an attack against NuCAPTCHA (83% success) [82]. Nguyen et al. used simple techniques to break HelloCAPTCHA (16%-100% success) [75] and some other animated CAPTCHAs (18%-100% success rate) [84]. In 2013, Nakaguro et al. proposed a multiple quadratic snakes model to attack against line-noise CAPTCHAs (73% success) [93]. In 2014, Goodfellow et al. broke reCAPTCHA using neural networks (99.8% success) [94]. The extent of attacks on CAPTCHAs and their high success rates highlight the need for developing more secure CAPTCHAs.

Now, we will explore some examples of the above attacks in more detail.

Attacks on current text-based CAPTCHAs

Attacks on EZ-Gimpy CAPTCHA: EZ-Gimpy CAPTCHA, which consists of an English word on a noisy background, has been vulnerable to several attacks. Chellapilla and Simard [72] discussed that the background of this CAPTCHA, which can be categorized into three different types of ‘no mesh’, ‘black mesh’ and ‘white mesh’, can be removed by simple pre-processing algorithms. One of the weaknesses of this CAPTCHA, which is its use of a word, has made this

CAPTCHA vulnerable to an attack designed by Mori and Malik. This attack uses shape context matching to identify the entire test word (rather than individual characters) with a success rate of 92% [87]. Their attack's lack of reliance on detecting individual letters of a test reduces the complexity of the attack. Another attack on EZ-Gimpy based on detecting the whole word instead of its characters has been proposed by Moy et al. [88]. The designers of this attack took advantage of the small set of template images used in this CAPTCHA to break it with a success rate of 99%. In this attack, the attackers collected the small set of template images and solved tests by comparing the challenge image with each of the templates to find the most correlated image.

An attack against Ticketmaster CAPTCHA: The general process of the attack is the same as EZ-Gimpy [72]. However, because of the crisscrossing lines used in the Ticketmaster CAPTCHA, a dilution and erosion step is added in order to remove every thin line in the image. This attack can break Ticketmaster CAPTCHA with a success rate of 4.9%.

An attack on MSN CAPTCHA: The steps of a segmentation attack on this CAPTCHA designed by Yan and Ahmad [69] are summarized in Figure 1. In this CAPTCHA, the little overlap between characters or lack thereof allows vertical histogram and color-filling segmentation attacks to detect connected components (two first steps in Figure 1). In the third step, the different shape, location and pixel-count of arcs compared to those of characters helps remove the arcs. Finally, knowing the fact that each MSN CAPTCHA contains exactly 8 characters and that the width of each test is approximately constant and the direction of the test is always horizontal, the output of a previous step which is in the form of chunks of connected characters can be divided into pieces of the same width to separate characters. The success rate of this attack is 60%.

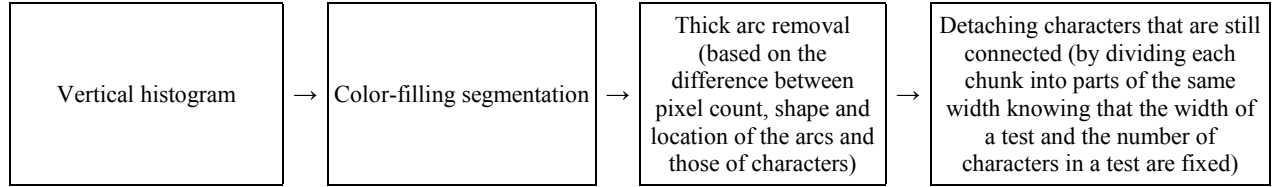


Figure 1: Steps of an attack on MSN CAPTCHA [69].

An attack on CAPTCHAservice.org [70]: To segment the characters, the designers of this attack used two flaws of this CAPTCHA, i.e. using distinct colors for foreground and background and having non-connected characters. After segmentation, the fact that each character in this CAPTCHA had a specific pixel-count helped attackers recognize characters by a pixel-count attack. Finally, in cases that two characters of a test had the same pixel-count, and could not be distinguished by a pixel-count attack, a dictionary attack was used to solve the test. The success rate of this attack was 94%.

Attacks against reCAPTCHA: While reCAPTCHA challenges are selected from old books and newspaper text that cannot be recognized by OCR programs, they might be vulnerable to attacks after a pre-processing step. Two main weaknesses of reCAPTCHA are using English words and allowing one error in answering tests. Beede [91] proposed an algorithm to remove reCAPTCHA's curvature and sharpen the characters. After applying this pre-processing step, OCR can solve this CAPTCHA with a success rate of 11.8%. Wilkins [90] applied a variety of erode/dilate matrices on every CAPTCHA word before OCRing, collected the results given by OCR and selected the longest result with a non-trivial count as the correct answer. The success rate of this attack was 17.5%. The success rate increased slightly after the designers of reCAPTCHA improved its usability by eliminating the horizontal line running through the

challenge word. Nevertheless, more recently this CAPTCHA was broken with a success rate of 99.8% using neural networks [94].

Attacks on current image-based CAPTCHAs

Examples of attacked image-based CAPTCHAs include Drawing CAPTCHA [39], ArtiFacial [41], IMAGINATION [35]; and Asirra [34]. Most segmentation attacks on image-based CAPTCHAs use the vulnerability that the added noise is not similar to the foreground objects. For example, in *Drawing CAPTCHA*, the difference between the size of diamond-shape target dots and that of clutter dots was the basis of an attack designed by Lin et al. [64] with a success rate of 75%. Another example is an attack on *ArtiFacial* in which removing some parts of noise that did not have the features of a human face made segmentation easier [81] (success rate=18%). Another flaw of a CAPTCHA that makes segmentation less costly is having minimal solution requirements. For example, *Imagination* only requires a user to click on the center of one of several pictures provided by the test. Finding the boundaries of only one object, even in a noisy image with many false boundaries is not very difficult for an attacker. Based on this weakness, Zhu et al. attacked Imagination with a success rate of 4.95% [81].

Most recognition attacks on image-based CAPTCHAs use image classifiers to recognize images. For example, Golle [86] used a combination of color and texture features to distinguish dog and cat images in *Asirra* CAPTCHA. This attack had a success rate of 10.3%.

Attacks on current audio-based CAPTCHAs

Audio CAPTCHAs are relatively weaker than visual CAPTCHAs. The reason is probably that the human visual system constitutes a larger portion of the brain as opposed to the human

audio processing system. Bursztein et al. [51] proposed an attack on eBay, Authorize, Yahoo! and Microsoft Audio CAPTCHAs with success rates of 82%, 89%, 45.5% and 49%, respectively. This attack segments the digits using a low-pass RMS (Root Mean Square) filter that eliminates regular and constant background noises and leaves peaks corresponding to the digits. After segmentation, classifiers have been used to recognize digits. The most important weakness of these CAPTCHAs was the difference between background noise and foreground signals. ReCAPTCHA that includes semantic noise, a noise with similar characteristics of a spoken digit, was more resistant against this attack.

Attacks on current motion-based CAPTCHAs

Attacking motion CAPTCHAs can be both harder and easier than image-based or text-based CAPTCHAs. It can be harder because in motion CAPTCHAs, segmentation and recognition, performed by the processing of the frames and by motion tracking, is more complex. It can be easier because the segmentation phase can be more accurate when multiple copies (frames) of the same CAPTCHA are available. The basis of most attacks on motion CAPTCHAs is to extract important information from the animation frames and reduce the animation to a traditional text-based CAPTCHA. Based on this strategy, Bursztein [82] and Nguyen et al. [75] designed attacks on NUCAPTCHA and HelloCAPTCHA respectively. The success rate of the first attack was 83% and for the second one, the success rate was between 16% and 100%. The vulnerabilities of these CAPTCHAs, including having discriminative features between text and the noise, the fixed number and position of the characters, inappropriate use of colors, the constant number and the delay of the frames, guided the attacks. These attacks show that resistance against segmentation in video CAPTCHAs, if not well designed, can be equivalent or less than text-based

CAPTCHAs. One strategy to improve the security of motion CAPTCHAs, suggested by Bursztein [82], is to use a confusing moving background.

1.3.2 Random guessing attacks

In a random guessing attack, also referred to as blind guessing or no-effort attack, an attacker tries to break a CAPTCHA by guessing the answer. In text-based CAPTCHAs, given the character set size, c , the probability of solving an n -character CAPTCHA challenge by blind guessing is $1/c^n$ [95]. In an image-based CAPTCHA that asks a user to detect an object between n candidate objects, the likelihood of a correct guess is $1/n$. Weaknesses of a CAPTCHA that can make it vulnerable to this attack include using a small input space, having a small number of candidate objects in a test, and imposing no limits on the number of attempts to solve a test.

1.3.3 Using social engineering to break CAPTCHAs

Using cheap third party humans is a way to break CAPTCHAs. Kang et al. [96] designed a CAPTCHA phishing attack that is a form of social engineering attack. They deployed a CAPTCHA phishing interface on a webpage (called CAPTCHA carrier) and selected some high-traffic website as phishing areas to publish their phishing messages. The phishing carrier and phishing area can be two different webpages. Alternatively, the phishing carrier can be integrated into the phishing area using Adobe Flash [97]. Since people are less willing to click an unknown hyperlink, the second approach has been more successful.

Truong et al. [29] designed another attack called “Instant Messenger CAPTCHA attack”. The major components of this attack are an attack script and an IM connector. The first component scrapes CAPTCHA images and uses the IM connector to send them to the third party

humans who solve the tests. Since instant messengers allow real-time communication between participants, the attack cannot be detected using timeout values.

1.4 Robustness of CAPTCHAs

A good CAPTCHA must be both easy-to-solve for humans and strong enough to resist attacks. Designing CAPTCHAs that fulfil both usability and robustness criteria is difficult. The key point to design such a CAPTCHA is to exploit the gap in the recognition abilities between humans and computers.

A CAPTCHA method is strong if no automated computer program can solve its challenges with a success rate of higher than 0.01% [98]. The security of a particular CAPTCHA test can be analyzed by investigating its resistance to attacks that might potentially be used to break it. We divide our review of the factors that affect the robustness of CAPTCHAs into the five following subsections and discuss each one in detail:

- Segmentation resistance,
- Recognition Resistance,
- Random guessing resistance,
- Security against third party human solver attacks; and
- Other security measures.

Most techniques are related to text-based CAPTCHAs because this type of CAPTCHA has been studied more than others. Many of the strategies might be mentioned in one category, but be applicable to other categories as well.

1.4.1 Segmentation resistance

A major determinant of CAPTCHA robustness is its resistance against segmentation attempts. Prior studies show that machine learning algorithms are better at solving recognition problems than segmentation problems [72]. Examples of strategies to improve the segmentation-resistance of a CAPTCHA include applying degradations, text distortions, background clutter, making false boundaries, appropriate use of colors, and randomness and dynamism in the presentation of CAPTCHAs. These strategies are discussed in this section.

Applying degradations: One common strategy to increase the security of a CAPTCHA is applying degradations to the test image. Examples of degradations are character fragmentation, masking degradations, crowding letters together, and the addition of arcs:

Character fragmentation: Making horizontal and vertical fragments in characters or using thinned images to break characters into isolated portions makes a CAPTCHA more resistant against segmentation [17]. It can also reduce the success rate of a pixel-count attack.

Masking degradations: Using masks to degrade a CAPTCHA test can improve security [14].

Crowding letters together: Different ways to apply this strategy include using condensed fonts with narrower than usual aspect ratios or italic fonts whose rectilinear bounding boxes overlap their neighbors or thickened fonts that cause characters to merge together. Another strategy is to juxtapose characters in any direction (rather than the x direction only) [13, 69].

Addition of arcs: These arcs may or may not intersect with the original characters. If they intersect with characters, they can make segmentation difficult; and if they do not intersect, the arcs can still baffle attacking algorithms since they might be confused with character parts [12].

Text distortions: Another strategy to increase the CAPTCHAs' resistance to segmentation is applying text distortions [13] including warping, scaling, translation, and rotation:

- *Global warping*: character-level elastic deformations applied to all the characters of a challenge together which can foil template-matching algorithms,
- *local warping*: small ripples, waves, and elastic deformations applied to each character independently which can foil feature-based algorithms,
- *Scaling*: stretching or compressing the text in the x or y direction,
- *Translation*: moving the text either up or down and left or right,
- *Rotation*.

Background clutter: Different types of clutter used to increase the security include blurring, gridlines, mesh, patterns and arcs for visual CAPTCHAs and background noise for audio-based CAPTCHAs. Applying a variety of background clutter in different tests makes it more difficult to extract foreground from the background using pre-processing algorithms. However, many types of degradations, if not applied properly, might reduce usability while having no positive effect on robustness [99]. For example, the effects of blurring, thresholding and salt-and-pepper noise can be removed by many current OCR systems [14]. Any noise that does not resemble challenge text/image/audio is not a good clutter [51]. A good idea is to use shapes similar to test images as background clutter.

Making false boundaries: In an image-based CAPTCHA that contains a number of objects, an attacker can use a boundary detector to segment the objects. Creating false boundaries between objects when designing a CAPTCHA can solve this problem [35].

Appropriate use of colors: Using multiple colors for the background and foreground, and including some foreground colors into the background and vice versa is another technique to make segmentation more difficult. Using two colors, one for background and the other for

foreground or using different colors for adjacent characters helps attackers to separate them easily [83].

Randomness and dynamism in the presentation of CAPTCHA test: Using detectable patterns in positioning CAPTCHAs' characters allows automated systems to find “where each character is” and facilitates their attempts in solving the challenge. The ideal situation is to remove every pattern and use randomness in the presentation of CAPTCHA as far as possible to confuse the machine. For a text-based CAPTCHA, random positioning of the characters, randomly using different fonts and font sizes, or features of other languages such as different writing directions, cursive features and diacritics and random number of characters can improve robustness [95]. For image-based CAPTCHAs, random positioning of the images, boxes with varying shapes and random background elements might be appropriate. For animated CAPTCHAs, using a random number of frames, a random frame delay and random movement paths can reduce the probability of segmentation attacks [58, 75].

In summary, the strategies to make segmentation harder include:

- Applying degradations (character fragmentation, character overlapping, masking operations, additional arcs connected to the letters with the same width as letters),
- Adding background clutter (arcs, images similar to foreground objects, using a variety of clutter for different tests),
- Using colors appropriately,
- Making false boundaries,
- Incorporating randomness and dynamism in CAPTCHA design and presentation (position, direction, size and dimensions of the test image, text length, frame count, frame delay).

1.4.2 Recognition resistance

In order to make a CAPTCHA secure, strategies need to be considered in order to make it resistant against recognition attacks. Such techniques are discussed in this section:

Using random strings instead of words: Using non-English character strings instead of English words in a text-based CAPTCHA can reduce the success probability of dictionary attacks. Inserting special characters will also confuse OCR systems [100].

Making it impossible to distinguish a character by counting its pixels: In text-based CAPTCHAs, making all characters have approximately the same pixel count or making characters have a very different pixel count in different challenges defeats pixel-count attacks [76].

Using different fonts: Using different fonts can defeat pattern recognition attacks. Another suggestion is using handwritten text since humans are superior to machines in recognizing handwritten text [16].

Selecting the words, sentences or images that cannot be recognized by current recognition systems: In a text-based CAPTCHA, before selecting an image as a test image, a good strategy is to make sure that current popular OCR systems cannot solve it (reCAPTCHA [10] implements this strategy). In CAPTCHAs that use sentences instead of words, the source of sentences is usually old books or newspapers and it pays to make sure that sentences are not available on the web [25].

Using a large database: This technique improves a CAPTCHA's security by making random-guess and database attacks less successful. There are different ways to increase the size of database of an image-based CAPTCHA. For example, using image collections of other picture providers [34] or enticing people to label a collection of images while playing a game [101].

Using visually similar but semantically dissimilar characters/images/audio: Similarity can confuse pattern recognition programs. When two objects are visually similar, it is much easier for a human to recognize the slight difference between them than it is for a computer program [64].

Removing or deforming features to defeat pattern-matching programs: an image/text detection tool uses pattern matching to recognize objects. Removing or deforming objects' features can confuse those programs. For example, sky, grass, or the direction of a 'text' can be clues showing image direction. To remove such clues for an attacker that needs to detect an image's direction, one can use a sub-image that consists of fewer clues than the whole image [47]. Another approach is to use global transformations and feature deformations [16].

In summary, some strategies that make recognition more difficult for machines include:

- Using random strings instead of words,
- Making characters have the same pixel count,
- Using different fonts,
- Selecting words or images that cannot be recognized by current recognition systems,
- Using a large database,
- Using visually-similar but semantically-dissimilar characters/images/audio in tests, and
- Removing or deforming objects' features.

1.4.3 Random guessing resistance

In text-based CAPTCHAs, having a large character set will reduce the chance of blind guessing. In many current image-based CAPTCHAs, a test has only a few potential answers which makes random guessing attacks easier for robots. Another strategy is to use mouse actions instead of the keyboard to select the correct answer, For example, if the test utilizes a 200×200 image and displays 6 potential objects, then the user needs to select a number between 1 and 6,

the probability of blind guessing will be 1/6 (or 0.17); however, if she is required to click near the center of the correct answer, e.g. inside a radius of 4 pixels from the center, the success rate of blind guessing will be $\frac{8\pi 4^2}{200 \times 200}$ (or 0.0075) [35].

1.4.4 Security against third party human solver attacks

While no security measure is without its shortcomings, some techniques to obstruct third party attacks include:

- Limiting the number of attempts to solve a CAPTCHA test [90],
- Limiting the time of the validity of a CAPTCHA [90],
- Detecting IP addresses that give successive incorrect answers [89],
- Measuring solution time which might be different for a legitimate user and a third party human [29].
- Making the CAPTCHA image meaningful only on the protected web site [29].

1.4.5 Other security measures

Other strategies to improve the robustness of CAPTCHA systems are discussed in this section. Most of them try to highlight the gap between human and machine abilities in solving problems and recognizing objects. These strategies include:

- Asking users to answer a logical question that requires human thinking [62].
- Using 3D images or characters based on the fact that humans are better than machines at recognizing 3D objects [50].
- Using structures that humans recognize better than machines such as trees [63].
- Combining text and graphics in the tests: most recognition tools are domain specific; they work exclusively with graphics or text. This strategy can confuse those tools [62].
- Requiring more human interaction, e.g. using mouse clicks, dropdown lists or drag and drop for answering CAPTCHA tests will reduce the risk of blind guessing attacks [102].

1.5 Usability of CAPTCHAs

In the development of a CAPTCHA, achieving a balance between usability and security is difficult. This section discusses usability issues that may be considered in the design of CAPTCHAs. Some of these considerations are generic and can help every CAPTCHA to be more usable, some of them are specific strategies which work for specific types of CAPTCHAs.

In this section, the usability issues of CAPTCHAs will be discussed under three dimensions:

- Distortion,
- Content,
- Presentation.

1.5.1 Distortion

The method and level of the distortion should be selected very carefully since many types of distortions not only make CAPTCHAs less usable, but can also compromise security if the system would have to ignore some of the users' mistakes or allow multiple attempts for failed tests [83]. Examples of clutter that confuse humans and do not improve security in text-based CAPTCHAs include:

- Every background noise that is not similar to foreground objects (e.g. EZ-Gimpy [8]),
- Arcs thinner than the characters (e.g. Yahoo! v.2.0 [13]),
- Arcs with the same size and shape as the characters (e.g. MSN [83])

In contrast with the above examples, in CAPTCHA zoo [64] noise elements are similar to the target objects. This CAPTCHA confuses machines, but is still usable.

In audio-based CAPTCHAs, sounds are distorted by background noise. It can affect usability in such a way that characters might be confused with each other. For example the user might not be able to tell 'v' and 'b' apart [103].

1.5.2 Content

An important consideration in selecting the contents of a CAPTCHA should be their impact on the CAPTCHA's usability. Important points about the contents of a text-based CAPTCHA include:

- The size of the character set: Although a large character set improves security, it might have negative effects on usability since a bigger character set implies a higher number of unknown characters and visually similar characters [104].
- Using random strings or words in CAPTCHAs: Using random strings makes the CAPTCHA more difficult for humans, while it increases robustness [83].
- The length of the string: While long strings increase robustness by reducing the success rate of random guessing, they decrease usability [83].
- Using a sequence of characters that might be visually similar to another character (e.g. 'cl' and 'd') might confuse users [83].

Generally, the designer of a CAPTCHA can consider the following points about the contents of the CAPTCHA to make it more usable:

- Being not offensive [83],
- Being independent of a certain language, age or knowledge level [39] (dependency on a certain language is one of the characteristics of audio-based CAPTCHAs),
- Being suitable for all people including those with disabilities [60],
- Being usable not only on PCs, but also on smartphones [64],
- Bringing other social benefits such as helping to read and digitize old scripts (reCAPTCHA [10]), finding homes for pets (Asirra [34]), etc.
- Selecting a type of CAPTCHA among different available CAPTCHA systems based on the user's information [60].

1.5.3 Presentation

The presentation of a CAPTCHA and its user interface should be designed to enhance usability.

In text-based CAPTCHAs, font type, font size and image size affect CAPTCHA usability. In some fonts, some characters might be similar and not easily distinguishable by humans. Large font sizes are usually more convenient for human users since they improve visibility [83].

In image-based CAPTCHAs, the size of the CAPTCHA image is a factor that affects usability. Small images reduce server processing time, accelerate the download process and occupy less space on a webpage [105]. On the other hand, large images are more visible and thus easier for humans to respond to; they also reduce the probability of blind guessing and improve security. The size of the image should be decided in a way that strikes a balance between usability and robustness.

In audio-based CAPTCHAs, the presentation of the CAPTCHA does not generally affect the usability. However, Bigham and Cavender [103] discussed that most current audio CAPTCHAs are frustrating for blind users who use screen readers to access the CAPTCHA. The reason is that using navigation elements to listen to and answer the CAPTCHA distracts them and forces them to miss the beginning of the CAPTCHA. Improving the CAPTCHA interface to solve this issue can increase the usability of audio-based CAPTCHAs for blind users.

In motion-based CAPTCHAs, the load time can be changed according to the limitations of the user's network to enhance usability. This can be achieved by changing the animation quality, the dimensions, or by converting it to grayscale [106].

Other strategies to improve the usability of CAPTCHAs through presentation include:

- Appropriate use of colors: color can facilitate human recognition and confuse OCR systems. However, colors should be used properly in order not to cause negative impacts on security or usability [107].
- CAPTCHAs' user interface: In different CAPTCHA systems, the users are asked to enter their answers by different methods such as typing the answer, selecting from a dropdown list, clicking the answer or dragging and dropping answers to boxes. While the most common method is typing, mouse interaction methods are more usable since they simplify and accelerate the answering process. These methods improve robustness as well [102].
- Partial credit algorithm [34]: Another strategy to enhance usability is to give users partial credit which means if they solve a test almost correctly (e.g. 7/8), they are considered as 'human users' and are shown another test. By passing the second test almost correctly the user is identified as a human and gets access to the protected resource. It is a two-step algorithm that means two almost-correct answers are required. Converting it to a one-step algorithm, which requires only one almost-correct answer (as can be seen in reCAPTCHA [10]), would cause security problems.

1.6 Conclusion

In the development of most human-interaction centric security mechanisms, a trade-off between security and usability is required. A strategy that increases the security in many cases reduces the usability and vice versa. In the evolution of CAPTCHAs, the weaknesses and limitations of many text-based CAPTCHAs have made them vulnerable to attacks. On the other hand, attempts to increase their security have often made them difficult for humans. Hence, CAPTCHA developers have been trying to explore alternative models to design more usable and secure CAPTCHAs. Image-based, audio-based, motion-based and hybrid CAPTCHAs were proposed a long time ago to overcome the restrictions of text-based CAPTCHAs. The drawback of these types of CAPTCHA is that preparing a substantially large database of images, audio files or animation clips that does not require human intervention for categorization or labeling is

close to impossible. These limitations make many of them vulnerable to attacks. To reduce the effects of these limitations, a new generation of CAPTCHAs, namely interactive CAPTCHAs, has been developed to increase the complexity of traditional CAPTCHAs for attackers. Interactive CAPTCHAs involve more human intervention in their tests. They use mouse actions such as clicking or drag and drop to elicit a form of response that is easier to produce for humans and more difficult for robots. They offer a more enjoyable and user-friendly human verification system. In addition, they improve security by adding new layers of complexity required to attack them. Hence, interactive CAPTCHAs seem to be the most promising CAPTCHA type.

In this chapter, we have introduced different types of CAPTCHAs (Table 1) and attacks against them. Many attacks, especially on non-text-based CAPTCHAs, are CAPTCHA specific. However, some of the most well-known strategies in attacking CAPTCHAs are summarized in Table 2. We have also investigated the weaknesses of current CAPTCHAs that make them vulnerable to these attacks. Table 3 summarizes these weaknesses.

Table 1: Different types of CAPTCHAs.

Text-based	“English words” CAPTCHAs
	“Random strings” CAPTCHAs
	Handwritten text CAPTCHAs
	Linguistic knowledge CAPTCHAs
	Interactive Text-based CAPTCHAs
	Non-English CAPTCHAs
Image-based	Object detection CAPTCHAs
	Subject detection CAPTCHAs
	Part detection CAPTCHAs
	Swapping task CAPTCHAs
	Orientation task CAPTCHAs
	3D CAPTCHAs
Audio-based	
Motion-based	
Hybrid	Dynamic CAPTCHAs
	CAPTCHAs with multiple challenges
	Multi-type CAPTCHAs
	CAPTCHAs for smartphones
	CAPTCHAs for people with disability

Table 2: Attacks on CAPTCHAs.

Pre-processing	Noise removal based on noise/target color difference
	Noise removal based on different size, shape, or location of noise and targets
	Noise removal based on different moving patterns of noise and targets
Segmentation attacks	Segmentation based on the location of target objects
	Segmentation based on the features of target objects
	Segmentation based on color information
Recognition attacks	Segmentation based on motion information
	Object recognition
	Pixel-count attack
	Dictionary attack
	Database attack
	Random guessing
	Social engineering attacks

Table 3: Major vulnerabilities of current CAPTCHAs.

Vulnerability source	Vulnerability	Description	Possible attacks	Examples of CAPTCHAs with this vulnerability	Examples of CAPTCHAs addressing this vulnerability
Input space	Small input space	E.g. Latin alphabet	Recognition attacks, Random guessing	Google [13], BaffleText [14]	Asirra [34], Sketcha [46]
	Using limited variations of each object	E.g. limited number of fonts; or limited voices in audio CAPTCHAs	Object recognition, pixel-count attacks	Authorize audio [51]	ScatterType [17], eBay Audio [51]
Similar objects	Only dissimilar objects in a test	Dissimilar in terms of shape, size, etc.	Object recognition, pixel-count	Drawing CAPTCHA [39]	reCAPTCHA audio [51]
	Different pixel count of objects	E.g. each character has a unique pixel-count	Pixel-count attacks	CAPTCHAservice [9]	Handwritten CAPTCHA [16]
Keyboard	Using physical keyboard as the input device	Causes all the limitations of a small input space	Recognition attacks, Random guessing	Most text-based CAPTCHAs	Drag-n-Drop [27]
Randomness	Constant number of objects in a test	The number of objects is known for attackers	Segmentation, random guessing	Collage [32]	Image Flip [44]
	Constant size of each object	Each object's size or with/length ratio is known	Segmentation, Random guessing	CAPTCHAservice [9]	Imagination [35]
	Constant position of the objects	E.g. characters located on a horizontal line, in the middle of the image	Segmentation, Random guessing	Collage [32]	Tree-based handwritten [63]
	Constant direction of the objects	E.g. horizontal, from left to right	Simple segmentation, Vertical histogram	Most text-based CAPTCHAs	Non text-based CAPTCHAs
	Non-random strings	Using words instead of random strings	Dictionary attacks	Gimpy [7]	Google [13]
Color	Inappropriate use of colors	Distinct colors in foreground and background, or giving information by color	Segmentation-preprocessing	MSN [12], Ticketmaster [13]	Imagination [35]
Noise and distortions	Not having noise	Background noise, mesh, arcs, ...	Segmentation attacks	CAPTCHAservice [9]	Ticketmaster [13], yahoo! audio [51]
	Noise dissimilar to test objects	Dissimilar in terms of color, shape, location, ...	Pre-processing, Segmentation attacks	Ticketmaster [13], Yahoo! audio [51]	CAPTCHA zoo [64]
	Having no object distortion	Text/image/audio distortions	Object recognition	Collage [32]	CAPTCHAservice [9], MSN [12]
	Non-fragmented objects	Non-fragmented characters, images without false boundaries	Segmentation attacks	CAPTCHAservice [9]	ScatterType [17], BaffleText [14]
	Not-connected objects	Non-connected characters, images with obvious boundaries	Segmentation attacks, Object recognition attack	Ticketmaster [13]	Google [13]
Design	Having minimal solution requirements	E.g. allowing errors in answering tests; or requiring selection of just one (of N) object voluntarily	Segmentation, Object recognition, Random guessing	Imagination [35]	Google [13]
	No restrictions on 'download limit' or 'error limit'	No limit on the number of CAPTCHAs that can be downloaded, or submitted with an incorrect response	Social Engineering attacks	eBay audio [89]	Asirra [34]

To conclude this chapter we subjectively counted the vulnerabilities of some major CAPTCHAs to paint a picture of the security issues that many of these CAPTCHAs face. We summarize these assessments in Table 4 where an assignment of “1” in a cell signifies that the CAPTCHA in that row might suffer from the vulnerability specified in the particular column. We give equal weights to the vulnerabilities and add up the vulnerabilities to show the extent of vulnerabilities for each CAPTCHA. In reality, some of these vulnerabilities might compromise security more severely than others. Nevertheless, this simple analysis supports our thesis that there remains a need for more secure CAPTCHAs that could be used in applications that need a tighter control of access to valuable resources. In the next chapter, we discuss these vulnerabilities in the development of our proposed CAPTCHA.

Table 4: A subjective assessment of the vulnerabilities of current CAPTCHAs.

CAPTCHA			Vulnerability source	Vulnerability														SUM								
			Input space	Similar objects	Keyboard	Randomness				Color	Noise and distortions	Design														
			Small input space	Using limited variations of each object	Only dissimilar objects in a test	Different pixel count of objects	Physical keyboard as the input device	Constant number of objects in a test	Constant size of each object	Constant position of the objects	Constant direction of the objects	Non-random strings	Inappropriate use of colors	Not having noise	Noise dissimilar to test objects	Having no object distortion	Using text parsers to break the CAPTCHA	Using motion information to break test	Not-automatic CAPTCHA generation	Not-automatic input space generation	Dependency on a specific language	Specific knowledge requirements	Minimal solution requirements	High probability random guessing		
Text-based	English word	Ez-Gimpy	1	1	1			1	1	1			1								1				7	
		Gimpy	1	1	1					1	1											1				6
		CAPTCHAservice	1	1	1	1	1				1	1			1							1				9
		Pessimprint	1	1	1	1					1	1				1						1				7
		reCAPTCHA	1	1	1	1					1	1				1						1				7
	Random string	MSN	1	1	1						1		1		1							1				7
		Yahoo v2.0	1	1	1						1				1							1				6
		TicketMaster	1	1	1						1		1		1							1				7
		Google	1	1	1						1				1							1				6
		ScatterType	1	1	1						1											1				5
		Sequenced tagged	1	1	1						1					1						1				6
		Handwritten CAPTCHA	1	1	1						1	1				1						1				7
	Linguistic knowledge	semCAPTCHA	1	1	1	1					1	1									1	1		1	9	
		Odd words out	1		1							1			1		1					1	1	1	1	8
		Number-puzzle text	1		1							1			1		1					1	1			7
		Strangeness in sentences			1												1					1	1	1	1	5
Text-domain				1												1					1	1	1	1	5	
Non-English	Arabic CAPTCHA	1	1	1						1				1						1				6		
Image-based	Object detection	Collage						1	1					1						1			1	5		
		Asirra						1	1											1			1	4		
		Imagination						1												1	1		1	5		
	Subject detection	PIX												1							1	1	1		4	
		Bongo	1							1				1										1	4	
		Activity recognition						1	1	1										1	1			1	6	
	Part detection	Implicit CAPTCHA																1	1	1					3	
		Line CAPTCHA											1		1										2	
		Artificial													1								1		2	
	Swapping task	Exchanging blocks					1											1	1				1		4	
		Jigsaw puzzle					1											1	1				1		4	
	Orientation task	Image flip													1				1	1					3	
		What's up																		1			1		2	
Sketcha													1						1					2		

			Vulnerability source	Input space	Similar objects	Keyboard	Randomness	Color	Noise and distortions	Design																	
CAPTCHA			Vulnerability	Small input space	Using limited variations of each object	Only dissimilar objects in a test	Different pixel count of objects	Physical keyboard as the input device	Constant number of objects in a test	Constant size of each object	Constant position of the objects	Constant direction of the objects	Non-random strings	Inappropriate use of colors	Not having noise	Noise dissimilar to test objects	Having no object distortion	Using text parsers to break the CAPTCHA	Using motion information to break test	Not-automatic CAPTCHA generation	Not-automatic input space generation	Dependency on a specific language	Specific knowledge requirements	Minimal solution requirements	High probability random guessing	SUM	
	3D	Sub-image orientation																		1	1						2
		2D CAPTCHA from 3D models																				1				1	2
		3D CAPTCHA	1	1	1									1													6
Motion-based		NUCAPTCHA	1	1	1							1	1	1	1				1				1				8
		Hello CAPTCHA	1	1	1							1	1	1	1				1				1				8
		Animation CAPTCHA												1	1						1	1			1		5
		3D Animation CAPTCHA	1	1	1	1						1			1												6
Hybrid		Question-based CAPTCHA	1	1	1							1										1	1				6
		Tree-based handwritten				1										1		1					1				4
Interactive		Drag and drop	1	1								1			1								1				5
		3D Drag and Drop	1	1								1	1										1				5
		iCAPTCHA	1	1								1	1	1									1				6
		Drawing CAPTCHA	1												1												2
		Wordpress KeyCAPTCHA																		1	1						2
		PlayThru																		1	1						2
		ClickSpell	1	1									1											1			5
		3D CAPTCHA																		1	1						2
		CAPTCHA zoo													1							1				1	3

Chapter 2 The Proposed CAPTCHA

2.1 Introduction

In the development of most human-interaction-centric security mechanisms, a tradeoff between security and usability is required. A strategy that increases the security in many cases reduces the usability and vice versa. For example, a password can be very long (e.g. 120 characters); while it is very complex for an attacker to break such a long password; it might be difficult for a human user to memorize it. CAPTCHAs are no exception and there are approaches to make a CAPTCHA extremely secure so that no computer program can solve the test. However, applying these techniques without usability concerns can make it extremely difficult for humans to solve the challenge. Hence, an end goal in designing a CAPTCHA is to make it complex for machines and yet easy to understand and solve for human beings.

The first generation of CAPTCHAs was text-based. Due to the weaknesses and limitations of many text-based CAPTCHAs, they have been vulnerable to attacks that use image processing, pattern recognition and machine learning algorithms. Although distorting the text and/or adding visual noise can make the CAPTCHA stronger, they reduce usability. Hence, CAPTCHA developers are trying to explore alternative models to design more secure CAPTCHAs that are still easy to solve for human beings. Image-based and audio-based CAPTCHAs were proposed a long time ago to overcome the restrictions of text-based CAPTCHAs. The drawback of these types of CAPTCHA is that preparing a substantially large database of images or audio files that does not require human intervention for categorization or labeling is close to impossible. These limitations make many of them vulnerable to attacks [81, 86]. To address these limitations, new generation CAPTCHAs including animated and interactive CAPTCHAs have been developed to go beyond the security offered by traditional CAPTCHAs.

In non-interactive animated CAPTCHAs, the information needed to solve a CAPTCHA can be obtained from all frames of an animation. A user is required to observe the whole animation to be able to solve the CAPTCHA. A main weakness of these CAPTCHAs is that many of them cannot be automatically generated and may be limited by the number of produced animations.

Interactive CAPTCHAs involve more human interaction in their tests. Since this type of CAPTCHA is more intuitive and more appealing for human users and since simulating human interactions by attackers adds new layers of complexity to a CAPTCHA, interactive CAPTCHAs seem to offer a promising approach in designing CAPTCHAs.

Although the new generation of CAPTCHAs seeks to reduce many of the limitations by trying to exploit the gap between the abilities of machines and humans in recognizing objects or interacting with a computer, disregarding core security/usability considerations can make any animated, dynamic or interactive CAPTCHA less secure/usable. For example, NuCAPTCHA – with its target characters concealed in animation – has been successfully attacked because of the existence of discriminative features between its target text and the background [82]. Consequently, in order to achieve high security, multiple sources of vulnerability should be addressed in CAPTCHA design. Within an interactive CAPTCHA design, minimizing discriminative features to a tolerable level for human users, randomizing size, location and the number of objects required to be detected, appropriate use of color, proper design of background noise and using a sufficiently large input space are some strategies to improve the security of every CAPTCHA.

We designed our proposed CAPTCHA to be an interactive CAPTCHA. We utilize Unicode as the input set of our CAPTCHA; the large size of this input space improves the security and provides the resource required for automatic generation of CAPTCHAs. In addition, we included

similar Unicode glyphs in this CAPTCHA that can confuse attackers. We designed a virtual graphical keyboard to be used instead of a physical keyboard in order to require more human interaction in solving tests. Moreover, a background is designed with elements that further increase the security. In designing the elements of this CAPTCHA, we aimed to achieve higher levels of security by making the CAPTCHA increasingly complex for machines while maintaining its intuitiveness for human users. Eventually, our analyses show that the proposed CAPTCHA is substantially secure, has a solving accuracy comparable to most existing CAPTCHAs, but takes longer to solve due to the processes required for solving the CAPTCHA and submitting the solution. We also discuss modifications that can further enhance the CAPTCHA's security or its usability as needed.

The remainder of this chapter is as follows. In Section 2.2, we discuss interactive CAPTCHAs as the basis of our CAPTCHA. Section 2.3 provides an overview of our approach and briefly describes strategies we proposed and implemented to improve the security and usability of the CAPTCHA. These strategies are explained in more detail in Sections 2.5 to 2.8. Section 2.5 describes how our proposed CAPTCHA uses Unicode. In Section 2.6, we discuss how we take advantage of the similarity among Unicode characters to enhance security. Our CAPTCHA's virtual keyboard and color representations are discussed in Section 2.7 and 2.8, respectively. Section 2.9 explains the designed user studies and their results and Section 2.10 concludes the chapter.

2.2 Interactive CAPTCHAs

Interactive CAPTCHAs involve considerable human intervention in their tests. They mainly use mouse actions such as clicking or drag and drop to elicit a form of intelligent response that is easier to produce for humans and more difficult for machines. Contrary to text-based

CAPTCHAs, interactive CAPTCHAs offer a more enjoyable user-friendly human verification system [108]. They can be designed in the form of simple games or puzzles making them more intuitive and more appealing than non-interactive ones. By engaging users, interactive CAPTCHAs make the passage of time less perceivable which is a meritorious advantage as it can lead to less user dissatisfaction. Moreover, their higher reliance on mouse actions is more aligned with the way human users browse the web³. In addition to usability, interaction can improve robustness.

Interactivity in CAPTCHAs enhances security by taking advantage of a human's ability to readily create intelligent patterns of interaction that are computationally intensive for computers to emulate. Requiring intelligent interactions substantially reduces the risk of blind guessing attacks. Moreover, depending on the complexity of the interaction rules, several levels of interaction might need to be simulated in order to break an interactive CAPTCHA. Furthermore, user interaction can be tracked and recorded in order to make a better decision of whether the user is a real human or a machine. The potential for achieving both high security and high usability gives interactive CAPTCHAs a promising outlook in the evolution and development of secure CAPTCHAs.

Examples of interactive CAPTCHAs include Drawing CAPTCHA [39], 3D drag and drop CAPTCHA [102], WordPress KeyCAPTCHA [112], Ince et al. 3D CAPTCHA [113], PlayThru [114], ClickSpell [115] and 3D CAPTCHA [116]. In Drawing CAPTCHA, the user has to find three dots that are different from the others and connect them to each other by drawing lines [39].

³ During web navigation, people usually use the keyboard only for typing. For other tasks, such as selecting menu options, following hyperlinks and selecting targets, they prefer to use positioning devices, e.g. a mouse [109, 110]. One reason for users' unwillingness to switch between input devices is the time loss to travel between devices. For example, a pointing task can be performed in 1,100 ms, while a simple movement from the mouse to the keyboard and back takes approximately 800 ms [109, 111].

3D Drag-n-Drop CAPTCHA asks users to drag and drop a rotated version of Latin characters to their respective blocks as they appear in the image [102]. WordPress KeyCAPTCHA requires users to assemble an image [112]. Ince et al. [113] introduced another 3D CAPTCHA that asks users to enter each character shown on the faces of a cube to the input boxes of the same color. In PlayThru, users have to win a simple drag and drop game to solve the CAPTCHA. In Clickspell, a user has to click on the letters of a word scattered randomly in the CAPTCHA image. The image is covered with another image – to increase the security – and the user clicks on the letters via a moving mask [115]. Finally, 3D CAPTCHA, which is designed based on spatial perspective and human imagination, asks a user to rotate a 3D model and find the correct position of rotation [116].

Interacting with computer input devices is relatively easy for human users. Computer robots generally lack the type of coordinated information acquisition, processing and production capabilities that human beings enjoy and, hence, require more sophisticated algorithms to solve interactive CAPTCHAs [27, 102]. Even if computer programs can generate fake mouse events, they will still have to overcome the challenge of producing such mouse events in a meaningful way. Depending on the nature of the interaction, it can be complex for algorithms to decide where and when to generate what types of mouse events in real time. On the other hand, it is easier and more attractive for human users to use mouse⁴ actions to solve a test. They might find it easier since it eliminates the need to physically move their hand away from the mouse as it is the usual input device used when people are browsing the web. Hence, using an input device, such as mouse, that allows for more complex type of human-computer interaction is intuitively

⁴ We are comparing the mouse only with the keyboard. Since many existing computer systems are still not equipped with touchscreens or other input devices and the most common input devices are still the mouse and keyboard, in this comparison we only consider the mouse and the keyboard.

appealing for users and practically challenging for robot programs that try to break a CAPTCHA. Simulating typically complex patterns of mouse interactions is evidently more challenging for attackers.

The above arguments rest on the fact that all CAPTCHA information is known to the machine. When these data are somehow concealed from the machine, another layer of complexity is added to the CAPTCHA. The machine not only has to have an algorithm for simulating the required interactivity, but it also faces the challenge of detecting informational elements required for solving the CAPTCHA.

It is interesting to note that the added advantage of concealing CAPTCHA information requires interactivity in order to work effectively. The concealed information could be revealed to the users as a response to an action that the user engages in, such as moving the mouse or clicking or drag and drop.

As a result, using human interactions adds dynamism to the system in two dimensions, namely in both the information acquisition and solution submission. The amount of information concealing could vary from invisibility to partial visibility. For example, in Ince et al. 3D CAPTCHA [117], only two sides of the cube are visible at any given time and observing the other sides requires more interaction; or in ClickSpell [115], only by moving the mouse cursor can a person see the original image. In these two examples of invisibility, gaining access to information requires intelligent interactions, which is easy for humans but complex and expensive for machines. Our CAPTCHA entails both interactivity and partial information concealment.

Despite the evident benefits of interactivity provisions in CAPTCHA design, there are some restrictions in designing interactive CAPTCHAs. Firstly, the amount of interaction should be

small and the time required for it should be short. While more interaction makes the CAPTCHA more secure, it also tires human users and can encourage them to give up. Although interactions bring more fun to CAPTCHA and appeals to the users even if the test takes a little longer than a typing-based CAPTCHA, the interaction amount should not be higher than a threshold that ensures a sufficient amount of security. Hence, finding a middle ground for the amount of required interaction is important. We have to incorporate a level of interaction that makes the tests difficult enough for attackers and precludes people from stopping to use the CAPTCHA.

2.3 Overview of our approach

In this section, we will briefly review the vulnerabilities of traditional CAPTCHAs and delineate how our proposed CAPTCHA overcomes these weaknesses.

2.3.1 Major vulnerabilities of current CAPTCHAs

The vulnerabilities of existing CAPTCHAs were thoroughly reviewed in the previous chapter. We quickly enumerate them here:

Small-size input set: Many current CAPTCHAs have small input spaces that make object recognition or blind guessing easy for attackers.

Non-similar known objects: Most existing CAPTCHAs use visually-dissimilar elements in their tests. It is more straightforward for a computer program to distinguish between “dissimilar” characters from a known set.

Receiving input from a physical keyboard: Current CAPTCHAs receive their input from a standard keyboard. In the physical keyboard, the number of keys is limited and known. This restriction significantly reduces the set size of usable characters in a CAPTCHA and increases its probability of being solved by an attacker.

Non-randomness in the presentation: The existence of fixed and predictable patterns in the presentation of CAPTCHA elements makes them vulnerable to segmentation attacks.

Improper use of colors: Using colors in a way that creates a detectable distinction between foreground text and background facilitates text extraction for attackers [83].

Lack of proper degradations: The level and type of the distortions have not been decided properly in many current CAPTCHAs.

Simultaneously addressing these vulnerabilities can significantly enhance CAPTCHA security.

2.3.2 An overview of our approach

The proposed CAPTCHA aims to reduce or eliminate the above-mentioned flaws of current CAPTCHAs. Figure 2 displays the proposed CAPTCHA. This CAPTCHA is composed of a test (left rectangle) and a keyboard (right rectangle).

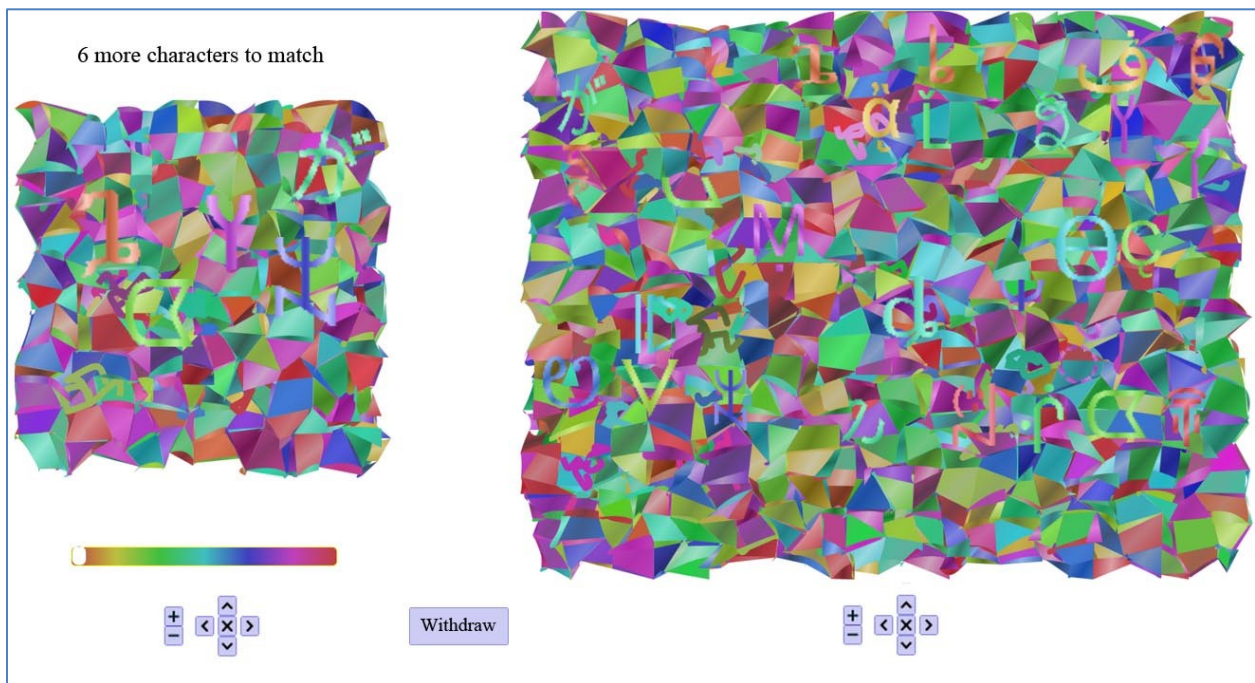


Figure 2: The proposed CAPTCHA.

The user is asked to find the correct match for each Unicode character of the test area in the keyboard. The user can use zooming facilities or the color palette to reduce the complexity of the noisy background and detect the characters more easily. Strategies that are applied in designing this CAPTCHA that seek to reduce the mentioned vulnerabilities include:

- Using Unicode as a large input space,
- Including a number of objects similar to each test character in the keyboard,
- Using a virtual keyboard instead of a physical keyboard,
- Incorporating randomness in the design including random size, number and position of the test and keyboard characters.
- Using a color scheme which uses foreground/text colors in the background as well.

These approaches are discussed in the remainder of this section.

Input Space

One important factor that affects the security of a CAPTCHA is the size of its input space. Most current CAPTCHAs have been created based upon very small input spaces. In all types of CAPTCHAs including text-based, image-based, audio-based and motion-based CAPTCHAs, a small input space reduces the security by allowing easier database attacks, automatic object recognition or blind guessing attacks. With a small input space, database attacks can be more successful. For example, every time a challenge is displayed, a portion of the input space can be revealed and by solving enough challenges, an attacker can uncover the entire database. A small input space makes pattern matching algorithms less costly and considering the small size of the input set, an attacker does not have to find a perfect match. In addition, in a small space, the likelihood of objects having more distinguishable features such as a one-to-one correspondence between pixel counts and specific characters is higher which makes object recognition more promising. Even if pattern recognition is not successful in detecting some objects, there is a high

chance of recovering them by a blind guess. With a small input space, the process of recognition (even by random guessing) can be performed exhaustively.

Almost no text-based CAPTCHA has considered using a larger input space. This unwillingness to incorporate larger character sets in CAPTCHAs might stem from two groups of complications: a) CAPTCHA design implications b) user experience implications. An overview of these implications is in order.

A standard keyboard, which is the main input device for the majority of text-based CAPTCHAs, does not readily support a large number of characters. It only supports Latin characters – or only a few selected character sets at any time. Using a larger character space would require CAPTCHA designers to plan additional character input provisions for the users. Aside from the additional difficulties that a virtual keyboard design would pose, designers face spatial and technical restrictions. Traditionally, CAPTCHAs have been designed to take up a limited amount of space on web pages and add as little extra load to browsers as possible. Avoiding the use of larger character sets might be an example of “thinking within the box of limitations” instead of planning workarounds that could take advantage of the added flexibility offered by a large character set.

The technical implications of using a larger character set might have dissuaded designers from adopting them but it could not single-handedly have stopped them from doing so. Perhaps a major concern in adopting these extensions lies in designers’ prediction of negative user reactions. Users’ unfamiliarity with languages other than their own, and perhaps English, which is an international language, might have inhibited the adoption of larger character sets. Using unfamiliar input spaces would lead to higher cognitive loads on CAPTCHA solvers which could cause dissatisfaction or discomfort for them. Prior research indicates that human beings evade

full information processing in decision making by resorting to simplifying strategies that take advantage of a subset of the available information [118]. These strategies benefit from a person's attention to information that is more salient, familiar or available. In a similar vein, CAPTCHA users experience lower cognitive loads in solving tests whose elements include or resemble their known character sets [104].

Researchers have adopted a number of workarounds to rid themselves of the limitations of a small character set. For example, one group of CAPTCHAs developed to address this issue tends to broaden their input space by asking the user to input a word instead of a sequence of characters ([9], [1] and [17]). This strategy was adopted to shift the input space from the limited Latin character sets to the broader set of English words. However, this purpose is defeated when an attacker breaks the word apart into its characters. In this case, the breakdown re-transforms the problem to one with a small character set. In other words, instead of exploiting one large space, this group of CAPTCHAs is in fact using a series of small spaces. This approach also increases the chance of dictionary attacks.

The limitations of text-based CAPTCHAs have encouraged the development of image-based CAPTCHAs. In these CAPTCHAs, however, a small image repository increases the possibility of database attacks. An attack in which the entire image database is gradually revealed is called a database attack. In order to defeat this attack, the database should be substantial in size. The supporters of image-based CAPTCHAs believe that in contrast to text-based CAPTCHAs that have a small input space, image-based CAPTCHAs can enjoy a large amount of images coming from public or private databases through the Internet. However, populating the database is a major issue with all image-based CAPTCHAs.

An important problem with using images is that the CAPTCHA-producing algorithm does not know the meaning of each image unless it is separately provided by a human. Consequently, unlike text CAPTCHAs which can use any random combination of characters in their challenges, descriptive information for images in image CAPTCHAs must be provided by a human. Mining public image search engines cannot solve this problem as a substantial portion of retrieved images might have irrelevant labels. Such mining attempts could produce small sets of correctly-labeled images. Attackers can take advantage of the small size of such databases and reconstruct the whole database by spending a fixed amount of time and effort. Hence, using human intervention in building the database is inevitable. Labeling the name or category of each image by a human produces extra initial costs in terms of both time and money. In the case of using publicly available images, a human is required to check if the existing image labels are correct, if the image is not offensive, if the image is not vague or irrelevant, etc. Moreover, there may be legal issues in using the crawled images directly. When legal issues in using web images do not exist, Von Ahn and Dabbish [101] propose a strategy to collect a large set of categorized images with less manual work by enticing people to label images while playing a game. However, designing a popular game and attracting people to play the game takes significant effort and tends to be costly. It is not clear how much time is required to populate a sufficiently large database of images by this method. Another drawback is that many images resulted from online search might be hard to label or categorize by game players. Therefore, with image-based CAPTCHAs, manual work is ultimately required to remove abstract or unclear images and label the clear ones. Another approach to gain access to a large database of categorized images is to request and procure them from websites that already have large image datasets. For example, SEMAGE uses e-commerce services to build its CAPTCHA image database [119] and ASIRRA

has formed a partnership with petfinder.com to have access to their images [34]. ASIRRA's growing database, containing more than three million images at the time of this CAPTCHA's introduction, is a very large set. However, each of the three million images, and any other image that is added to the database has to be categorized manually.

In summary, we address the problem that text-based CAPTCHAs have a small input space that limits their amount of achievable security. This problem is partly due to limitations such as the limited number of character slots on standard keyboards and assignment of few character-sets to any given keyboard by the user. On the other hand, the limitations of image-based CAPTCHAs in terms of database size do not make them an easily-implementable and cost-effective candidate to adequately address the security risks posed by small input spaces.

According to the above arguments, the lack of a large input space that can be produced automatically is still an issue. Unicode, with a potential capacity of over 1 million characters and currently approximately 110,000 encoded characters, is our solution for this problem. Using a variety of fonts and font variations (size, style, weight, width, etc.) to represent Unicode characters makes the input space even larger. For example, if each character can be displayed by 10 fonts and 4 font variations can be applied, the size of the input space will be 4,400,000. In this large input space, every element is referable and retrievable without requiring human intervention.

An important advantage of Unicode as a CAPTCHA's input space is that Unicode can produce objects automatically. This is in contrast with many current image-based CAPTCHAs that require human effort to label or categorize their images [34]. It does not matter if we look at the Unicode as a set of characters or as a set of images; what matters is that its elements can be

produced automatically. We believe that this very large input space can be utilized in creating a strong CAPTCHA.

Using such a spacious input set will highly reduce the likelihood of the challenge being solved by either pattern recognition algorithms or random guessing. For example, if a CAPTCHA's character set is the 26-letter English alphabet, it takes a typical computer $26^6 = 308915776$ pattern comparisons or blind guesses to solve a 6-letter challenge. If, on the other hand, the character set is Unicode with approximately 110000 encoded characters, it is much more difficult and pattern recognition or blind guessing can hardly help since $110000^6 = 1.7 \times 10^{30}$ comparisons need to be made. Having an extensive character set will also inhibit pixel-count attacks because many characters will have the same number of pixels. While improving the security, using Unicode as the input space of a CAPTCHA might compromise its usability.

From a usability viewpoint, people are more comfortable with familiar characters [104] and the existence of unfamiliar characters in this CAPTCHA could potentially cause some complications for users. However, in this CAPTCHA, the users do not have to recognize a character and its linguistic meaning: they only need to match some shapes. Moreover, design considerations that might make solving the CAPTCHA easier or even more entertaining can reduce the complications for users.

One of the implications of having Unicode as the input space is the need for a virtual keyboard. Since the standard physical keyboard covers only Latin characters and it can be configured to map only one language at a time, it cannot be used as an input device for the whole Unicode space. See Section 2.7 for discussion. Also, see Section 2.5 for a detailed discussion on our approach using Unicode characters.

Known distinguishable objects

One of the weaknesses of current CAPTCHAs is that their tests include objects that are dissimilar and can be discriminated. Dissimilarity of the objects makes tests easier for both humans and computers. In text-based CAPTCHAs, the small input space of Latin characters implies dealing with known distinguishable characters; this increases the probability of pattern recognition algorithms being successful. In other types of CAPTCHAs, dissimilar items can be distinguished more easily since they have features that are more different. Hence, dissimilarity between elements can be considered as a source of vulnerability for CAPTCHAs.

Instead of using “dissimilar” objects in challenges, which can be differentiated more easily by computer programs using pattern recognition algorithms or pixel count attacks, a designer can employ “similar” objects. Similarity can confuse pattern recognition programs. When two images are visually similar, it is much easier for humans to recognize the slight difference between them than it is for a computer program [64]. Differentiating between similar objects with similar features require more advanced algorithms which makes attacks more expensive and ineffective. A few CAPTCHAs have tried to use this strategy in their tests. For example, CAPTCHA Zoo [64] selects two visually similar kinds of animals and displays them and asks users to identify animals of one group. Another example is semCAPTCHA [24] that includes three words which are graphically similar (e.g. of the same length) and semantically different. The user is asked to select the one which is less related to the other two.

The problem with these CAPTCHAs is that, due to the size restrictions of their input dataset, the number of similar objects/group of objects is limited, and often the similarity of objects/groups is required to be decided manually. Unicode, containing a large number of similar characters solves this problem for us. In fact, Unicode not only gives us an automatically-

generated large input space, but also provides us with many similar objects. An example of similar objects in the Unicode is ㇰ and ㇱ which are two different Hiragana characters.

Although including similar objects in tests leads to stronger CAPTCHAs, solving such tests is also more challenging for humans. When there are similar elements in a test, more human attention is required to distinguish them. Hence, the number of similar objects in a CAPTCHA should be limited. Here again is a tradeoff between security and usability. A few homoglyphs can be used in a CAPTCHA to increase security, but the number of homoglyphs should not be too many to avoid confusing the human solvers. In our CAPTCHA keyboard, we include between 3 and 5 homoglyphs from the pool of homoglyphs available for all 6 to 8 test characters. See Section 2.6 for a detailed explanation of how our proposed CAPTCHA uses similar objects.

Virtual keyboard

Physical keyboards have been the main character input device since the advent of modern computers. Despite the general ease of use that standard keyboards afford computer users, their adoption for solving CAPTCHAs has introduced a major design constraint, namely, the limitation of the input space to the one offered by the keyboard. This constraint has led to the development of most CAPTCHAs as keyboard-dependent puzzles. As a result, most current CAPTCHAs rely on standard physical keyboards for their input. It is conceivable that removing this constraint from the CAPTCHA design process opens new avenues for developing more secure CAPTCHAs. Virtual keyboards offer the required flexibility to remove this design constraint.

A limited known set of characters, as offered by a standard keyboard, is a major weakness in CAPTCHA design. Probabilistically speaking, a CAPTCHA relying on a limited set of input characters is more vulnerable to most types of attacks than a CAPTCHA that does not face this

limitation. Moving from a standard physical keyboard to a virtual keyboard is in fact moving from a *limited* number of *known* keys to the possibility to use a more *diverse* set of both *known* and *unknown* keys. This possibility materializes when we resort to Unicode as the character set of our CAPTCHA. Using any large input space, such as Unicode, necessitates provisions for an input device that can handle a large number of input characters. Using a virtual keyboard can help address this problem. Moreover, virtual keyboards add further improvements to overall CAPTCHA security.

Using virtual keyboards instead of a physical keyboard in a CAPTCHA adds extra burden on any algorithm trying to break it [120]. A virtual keyboard is an image which is a (potentially varying) part of the screen; a high computational complexity is required to process a robust virtual keyboard and segment the characters. In order to break the keyboard, attackers will have to capture the screen and try to analyze the screenshot. From a usability viewpoint, a virtual keyboard does not put extra load on the user. In fact, it is even easier for some users to use the mouse to input information by a virtual keyboard rather than typing on a standard keyboard.

In our proposed CAPTCHA, we employ strategies that can make a virtual keyboard more secure. These strategies include randomizing the number of keyboard characters, their sizes and their positions and having a background with appropriate noise to impede segmentation. Furthermore, we use similar characters to make recognition harder for machines. A detailed discussion on the design of the proposed CAPTCHA's virtual keyboard is provided in Section 2.7.

Randomness in the presentation of CAPTCHA test

A major vulnerability of CAPTCHAs arises from the use of fixed detectable patterns in their design. Increasing the amount of randomness can confuse machines and complicate

segmentation of the CAPTCHA. In text-based CAPTCHAs, the regular rectangular boxes that most current CAPTCHAs use can be replaced with more irregular shapes. Characters can be juxtaposed in any direction or they can be positioned randomly to make segmentation harder. The number of characters can also vary from one challenge to another [95]. For image-based CAPTCHAs, random positioning of the images, different-shaped boxes and random background dimensions and for animated CAPTCHAs, the random number of frames and random frame delay [106] can reduce the vulnerability to segmentation attacks.

The proposed CAPTCHA extensively uses randomization to select characters (see Section 2.5 for more details), their count, size, location and color in both the test and keyboard (details are discussed in Section 2.7), the number of homoglyphs and their selection in the keyboard (explained in Section 2.6) and the design of test and keyboard backgrounds (Section 2.8).

Color representation

An important topic in CAPTCHA design is color representation. Applying color to CAPTCHA tests to paint characters or to design the background can improve security since both OCR programs and segmentation algorithms perform poorly while dealing with color images. However, inappropriate use of colors can have negative effects on both security and usability. In the proposed CAPTCHA, we employ strategies in using colors to strike a balance between usability and security to the extent possible. In this CAPTCHA, multiple colors are used in both the background and foreground of the test and keyboard. We use the *same* colors in both background and foreground to preclude an attacker from telling them apart. To increase randomness, the number of colors and their hue are selected randomly. Section 2.8 discusses color representation in the proposed CAPTCHA in more detail.

Applying degradations properly

One strategy to produce robust CAPTCHAs is to apply appropriate degradations to the test. As mentioned earlier in Chapter 1, various distortions are proposed to degrade text-based CAPTCHAs including blurring, adding random noise, interference by random-shape masks [14], background clutter such as grids and gradients [13]; random shearing, adding intersecting or non-intersecting arcs; and crowding letters together to remove white spaces between them [13]. Similarly, for image-based CAPTCHAs, degradations such as overlapping different images of a test to hide their boundaries, creating false boundaries [35], inserting background noise which is similar to test objects have been proposed. For audio-based CAPTCHAs, adding noise such as white noise, sine waves, cracks, and voices similar to foreground sound is suggested [51]. However, it should be noted that adding distortions directly affects the usability of a CAPTCHA.

Recognizing objects in an over-distorted test is difficult or impossible for humans. Hence, it is very important to determine what distortion method should be applied to the test. Sometimes distortions not only reduce human recognition, but they also do not improve the robustness of the system. Such distortions can be removed by simple image processing methods [83].

In the proposed CAPTCHA, the availability of background noise which can be similar to some of the Unicode characters in terms of shape, color, location, etc. can potentially baffle attackers. In order to improve the usability and to help human users to recognize target objects in the noisy background, they are provided with zooming and color-filtering tools. Section 2.8 provides more details about degradations in the proposed CAPTCHA.

An example of the proposed CAPTCHA system in action

This section illustrates a step by step example of a test in the proposed CAPTCHA system. Figure 3 displays a screenshot of the proposed CAPTCHA. As the figure shows, in this test, the user has to match 6 characters; and she can withdraw at any time if she wishes to.

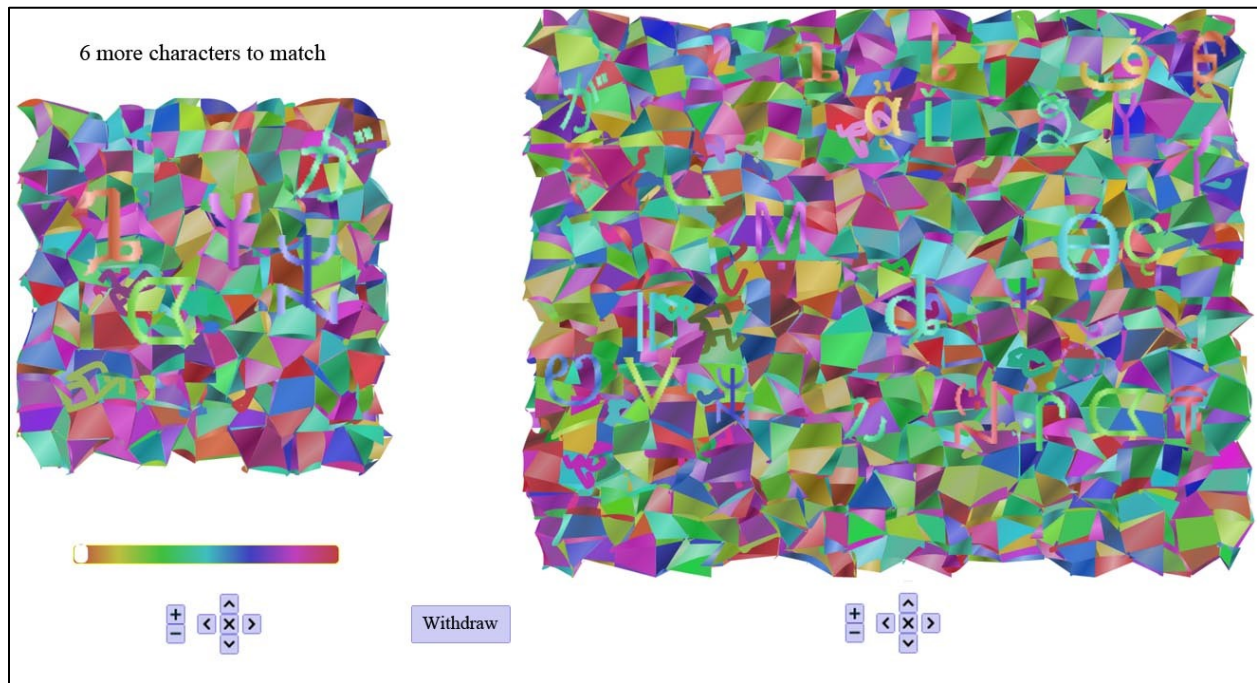


Figure 3: The proposed CAPTCHA.

When the user finds a character in the test (left rectangle) and identifies its match in the keyboard (right rectangle), she can click them in any order, in order to match them. When two characters are matched, regardless of its correctness, the number above the test will reduce by 1, indicating that a match has been received by the CAPTCHA.

The user can also choose to use the color palette by clicking on or dragging the color chooser handle on the palette (See Figure 4). When the color chooser is located on a specific color on the color palette, all colors in the test and keyboard except for the chosen color will be filtered out.

The color chooser can be dragged to the left and right to actively change the chosen filter color to the most current one and apply the relevant filter.

To enhance security, we aim to communicate the least amount of information about the outcome of a mouse click or a matching attempt. Our visible counter drops by one whenever two clicks from the test and the keyboard are received in any order regardless of whether a character or pure background noise has been clicked. In order to avoid user confusion about the location of their last click, the CAPTCHA shows a dashed circle (as seen in Figure 4) around the pixel that is clicked. This strategy helps the users remember which object they are trying to match. It also helps them confirm a click and readily know the direction of the movement for the next click. For example, in Figure 4, the user has first clicked on the left rectangle, which has revealed a dashed circle around the click location. After the user provides the second click on the right rectangle, the dashed circle is replaced with two same-color solid circles around the location of two clicks (at the start and end of the matching) indicating that a pair of matching clicks has been received and the counter drops by one (Figure 5). In the case of multiple consequent clicks on the same rectangle, the last click is always considered. In addition, the solid circles for each matched pair have a different color to allow users to readily identify previously matched pairs.

As a result of the above provisions, showing circles around click points or changing the counter after a pair of clicks does not convey any information about the presence of a character in the clicked region, but helps users remember where they last clicked and how many more matches they are required to achieve.

The user can also choose to use the zooming and panning tools (mouse scroll, drag and drop, or buttons under the test and keyboard) in order to find characters (see Figure 6 for an illustration).

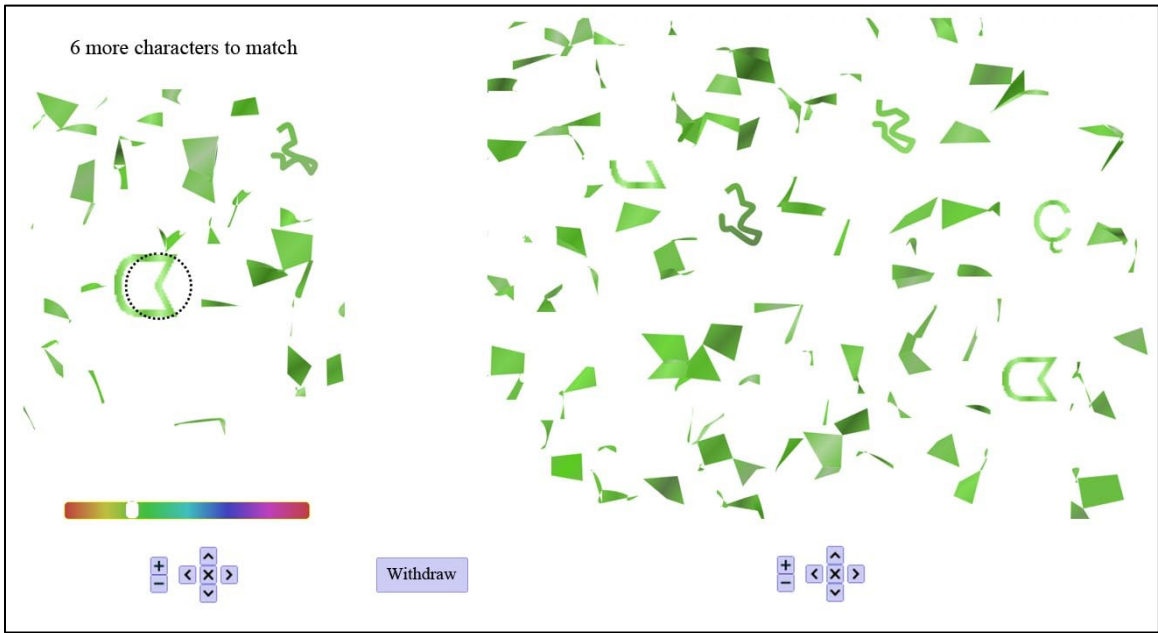


Figure 4: The proposed CAPTCHA in the palette mode.

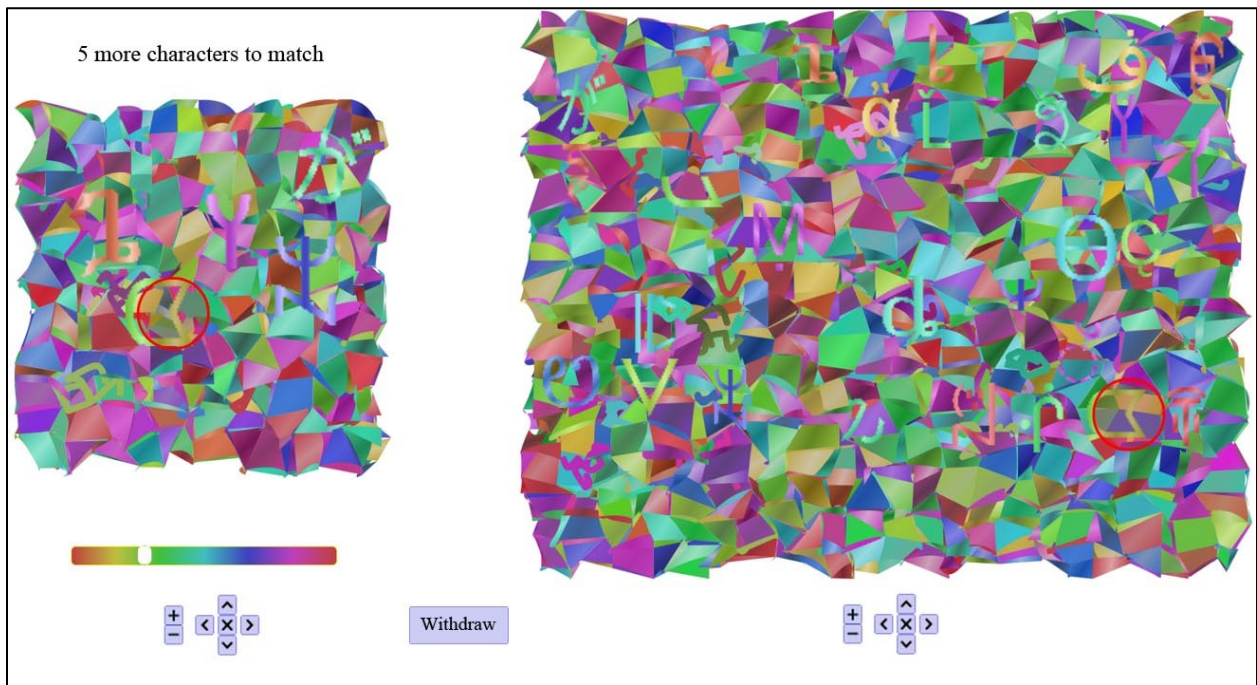


Figure 5: The proposed CAPTCHA after matching one character.

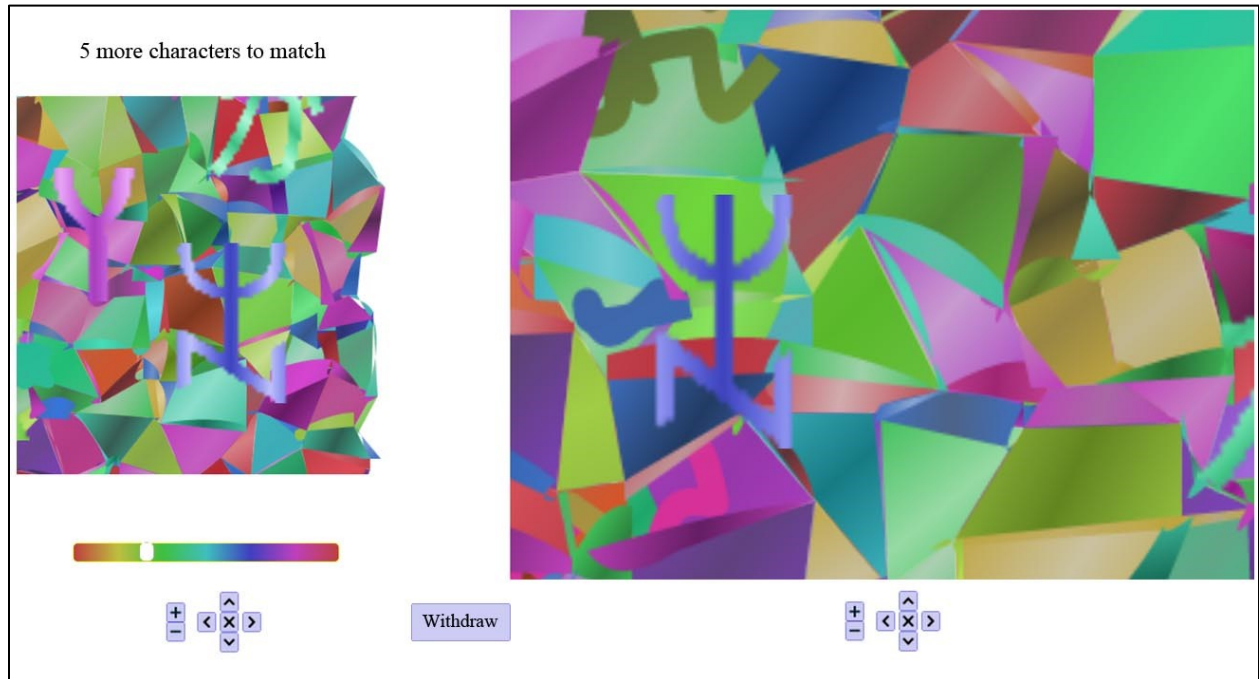


Figure 6: Zoomed test and keyboard.

2.4 Interactivity

The proposed CAPTCHA requires human interaction in answering the tests. This interaction takes the form of mouse actions to solve the CAPTCHA and using two sets of solution aids, i.e. a color palette and zooming facilities.

Interactivity enhances the security of a CAPTCHA by requiring seemingly random actions that need to be intelligently determined. Simulating mouse actions is not as easy for attackers as it is for humans and it adds an extra level of difficulty that challenges attacking algorithms [27, 102]. In fact, while the interactions themselves might not be hard to emulate for a machine, learning the exact parameters that govern the interaction can be computationally intensive. The complexity further increases when the data for inferring the suitable actions are concealed and require more intelligent actions to be unveiled. Concealed information could be revealed in

response to an action by the user, such as moving the mouse or clicking. The proposed CAPTCHA uses both these possibilities and requires calculated mouse actions to learn about the solution and to submit it. Given the high level of information camouflaging in the proposed CAPTCHA, two interactive solution aids, i.e. a color palette and zooming facilities, are provided which will be explained in the remainder of this section.

- Color palette:

Providing a color palette, that allows the user to see the CAPTCHA in one color at a time, facilitates its solution for both humans and attackers but substantially more so for humans. The palette gives the human users clues to see the characters more easily in a noisy colorful background. On the other hand, it allows computer programs to capture a less noisy version of the image that contains target characters. However, the benefit that a machine gets is much smaller than the benefit experienced by the human. The reason is that if the palette is not available, a computer program is able to segment the different colors of an image; hence, the palette just slightly reduces the complexity for attackers. The ability of attackers to segment the colors might seem to undermine the security of our proposed CAPTCHA. However, our CAPTCHA relies on multiple layers of complication, with color being only one. Even after successful color segmentation of the CAPTCHA by a potential attacker, which is by itself a computationally intensive task, the existence of homoglyphs from Unicode in the segmented image is a non-trivial obstacle. Moreover, even after color segmentation, the background contains noise elements that match the target characters in color and are difficult to distinguish. In addition, if the attacker wants to simulate palette clicks, the simulation will increase the complexity of the attack. In short, although the palette makes the problem slightly simpler for the machine, it makes the problem highly manageable for humans.

- Zooming facilities:

Zooming and panning facilities improve CAPTCHA usability by enabling users to magnify the CAPTCHA image for better observation. There are situations in which the users might be confused about whether they have recognized a character correctly or not. In such situations, the users can use zooming and panning facilities to ensure their selections are correct. The confusion can occur because of the unfamiliarity of the characters, the contrast between foreground and background colors, and most importantly, because of the existence of homoglyphs. Increasing the size of the image helps the users ensure that this is a character by finding its match more easily. Another confusing situation is when a part of a character is located on an area of the background that has a similar color. Again, magnifying the image makes distinguishing the character easier in this situation. Zooming can also be useful when deciding between existing homoglyphs. Distinguishing homoglyphs on a noisy background can potentially be difficult for the users, especially when the homoglyphs have a small size. Enlarging the characters helps the users see the details of similar characters (e.g. their diacritics) and distinguish them more easily.

Zooming, while improving usability, does not make the problem easier for attackers since it does not reduce the complexity of the image which is being processed by the machine. Machines might not use this facility at all because zooming addresses a deficiency in human vision and provides little direct benefit to machines understanding of the image, if any.

- Mouse (click, select):

Mouse actions allow users to interact with the proposed CAPTCHA in order to select the target characters and their corresponding match, to use zooming facilities and the color palette. Our CAPTCHA's use of the Unicode, randomized character locations, and matching target characters with their counterparts necessitates the use of mouse actions. As a result, in their

simplest application, mouse actions are imposed by the design of our CAPTCHA. However, they can be further utilized to increase security in addition to the basic function of providing a means for interactivity.

In the basic use of mouse actions, users can match the characters using mouse clicks; they click on related characters as a proof that they have located the target character and its match. They can also use mouse click, scroll and drag to zoom or pan the test or keyboard images or to apply color masks using the color palette. Mouse actions can also be used in implementing further security measures by requiring more intelligent mouse interactions for solving the CAPTCHA.

2.5 Unicode

As discussed in Section 2.3, a large input space could potentially improve CAPTCHA security and we proposed using Unicode as this input space. The Unicode Standard is the universal character encoding standard used for the representation of text which provides a consistent way to encode multilingual plain text. Unicode covers all of the written languages of the world. It includes characters, punctuation marks, mathematical symbols, technical symbols, arrows, diacritics, etc. Unicode has become the dominant scheme for text processing in most operating systems, applications' input methods, web pages and email, etc. Using Unicode has had a dramatic growth and has exceeded all other encoding standards [121].

Unicode assigns a unique value (code point) and a name to each character. Unicode characters, which are assigned code points from 0x000000 to 0x10FFFF, are organized into blocks. Each block is a group of related characters within the Unicode space. Blocks have various sizes. For example its Chinese-Japanese-Korean (CJK) block contains many thousands of code points while the Cyrillic block has only 256 characters.

The potential capacity of the Unicode character set is over 1 million characters (1,114,111 characters). As of Unicode 6.0.0, 249,031 (22.4%) of these code points are assigned; including 109,449 encoded characters; 137,468 reserved for private use; 2,048 for surrogates and 66 designated non-characters. 865,081 (77.6%) are unassigned [122]. The Unicode space is growing; there are languages in the world whose alphabet has yet to be included in Unicode. In Unicode 6.1.0, 732 new characters have been assigned.

This large space size coupled with the added advantage that each object or character within Unicode is basically a referable, retrievable and recognizable object provides a versatile database from which CAPTCHAs can be constructed. This latter feature of the Unicode is an obvious advantage over large image databases that require human intervention for labeling or classifying. With these characteristics, Unicode presents itself as a suitable candidate for the input space of a CAPTCHA.

This large input space can improve the security of a CAPTCHA by making object recognition or blind guessing attacks more effortful for an attacker. In general, if the size of the character set of a text-based CAPTCHA is c and the number of characters in the test is n , the probability of solving the challenge by blind guessing will be $1/c^n$ [95]. For CAPTCHAs comprised of 6 characters that are limited to the Latin character set, the probability of a correct random guess will be 3×10^{-9} while it will be 6×10^{-31} for the same CAPTCHA using Unicode as its input space.

The large object database provided by Unicode can lead to improved CAPTCHA security when used properly. However, a larger input space would translate into user unfamiliarity with a substantial percentage of the characters used in a CAPTCHA. This situation could potentially pose complications as human users might be less comfortable in solving CAPTCHAs with

unfamiliar characters [104]. Nevertheless, this complication can be turned into an opportunity if solving the CAPTCHA can be turned into an entertaining experience. It is conceivable that users become engaged when dealing with unfamiliar experiences. Two of the properties that make a task engaging are the existence of a challenge and the need for problem solving [123]. Both of these properties exist in our CAPTCHA. Consequently, the security benefits of using a large input space can be reaped without a large toll on usability. A review of some important aspects of Unicode and their application in our CAPTCHA is in order.

2.5.1 Glyphs and characters

The elements of the Unicode standard are characters. However, the visual representation of a character, i.e. its glyph, is not specified by the Unicode standard. In other words, Unicode does not deal with the rendering of characters on any type of media such as monitors and printers. To illustrate this point, the character designated as “Latin small letter A” and some of its possible glyphs are displayed in Figure 7. What Unicode knows about this character is merely a numerical code point “0061” and an identifier (Latin Small letter A); a rendering system is responsible for drawing this character as ‘a’, ‘a’, ‘a’, etc. When we press the key assigned to ‘a’ on the keyboard, we specify its code point; but what we see on the display is the representation of that code point by the rendering system.

Character	Glyphs					
Latin small letter A (U+0061)	a	a	a	a	a	a

Figure 7: Various glyphs displaying the character ‘a’.

The relationship between characters and glyphs is not as straightforward as one might expect. For example, there is no one-to-one correspondence between glyphs and characters. This point is illustrated in Figure 8. In order to draw one character, one or more glyphs might be required (row 1) depending on what font is being used. On the other hand, a sequence of

characters might be represented with a single glyph (rows 2 and 3) in one font and with multiple glyphs in another font.

	Character sequence	Single glyph representation	Multiple glyph representation
1	ò	ò	o ^ `
2	f i	fi	f i
3	o ^ `	ò	o ^ `

Figure 8: Relations between characters and glyphs in Unicode Standard.

Since in most cases several glyphs exist for each character, the total number of input elements, i.e. glyphs, offered by the different renderings of Unicode characters is substantially higher than the total number of Unicode characters, especially when character variations and context-sensitive glyphs, as well as the existence of different fonts are taken into account. We briefly examine the first two considerations here and deal with font variations in the next section.

Many Unicode characters, especially CJK characters, have variations which are often merely stylistic and extend the number of available glyphs far beyond that of the character space. These variations are registered in ‘Ideographic variation databases’ of the Unicode standard [124] and have their own variation selector code points. Figure 9 shows representative glyphs for two CJK characters. These variations further increase the size of our input space when using Unicode.

Character	Representative glyphs
U+3689 (Profit)	𠄎 𠄏 𠄐
U+34DE (Engrave)	𠄑 𠄒 𠄓 𠄔

Figure 9: Ideographic variations of Unicode characters.

Furthermore, some scripts use context-sensitive glyphs for most characters. In such scripts, a character’s shape in a word depends on its location within the word and its surrounding characters. For example, in Arabic, many characters have four different forms: initial, medial, final and isolated depending on whether the character is the initial character in a word, the

medial or final character or an isolated character. Figure 10 shows the different glyphs of the same Arabic or Mongolian character as it would appear at different locations within a word.

Character	Different glyphs based on the context			
	initial	medial	final	isplated
ﻩ (U+06D5 : Arabic letter AE)	ﺀ	ﺀ	ﺀ	ﻩ
ᠠ (U+1820 : Mongolian letter A)		ᠠ	ᠠ	ᠠ

Figure 10: Characters with context-sensitive glyphs.

In sum, when we use Unicode as our input space, besides having access to the large database of Unicode characters (i.e., numbers and definitions), we will have a larger repository of glyphs (i.e., images, representation of characters) from which our CAPTCHAs can be constructed. Although ideographic variations and characters' context-based glyphs for some writing systems increase the number of possible glyphs available via Unicode, the existence of different fonts can further broaden the possible representations throughout the entire Unicode space.

2.5.2 Fonts

A font is composed of a set of glyphs used to display or print characters of text. In fact, a font performs a mapping from characters to glyphs. The number of glyphs and characters in a font is not the same; which means there is not a one-to-one isomorphic mapping between characters and glyphs. The number of glyphs in a font is greater than the number of characters. The reason is that different classes of printing, various books, journals or electronic resources require alternate forms of letters that differ in a graphical sense but not in a character sense. It is the responsibility of font designers rather than encoding systems to design different glyphs of characters and include them in a font.

It is important to understand the distinction between a *font* and a *typeface*. Although they are used synonymously, they are technically different. A *typeface* or *font family* is a group of related fonts which vary only in weight, orientation, width, etc., but not in design. A *font* is a collection

of glyphs with one specific weight, style, variant and stretch. For example, “Times” is a typeface that includes TimesRegular, TimesBold, TimesItalic, TimesOblique, TimesBoldItalic and TimesBoldOblique fonts.

Considering different variations of each property of a font, a typical character can be represented in lots of different ways. Figure 11 displays a letter ‘A’ written using a variety of typefaces.

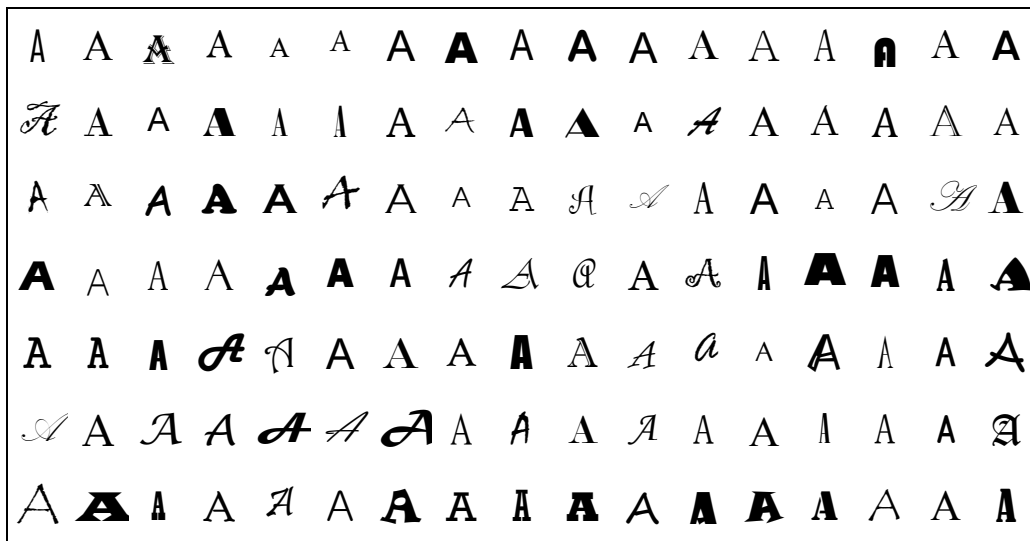


Figure 11: Representation of character 'A' by some standard fonts.

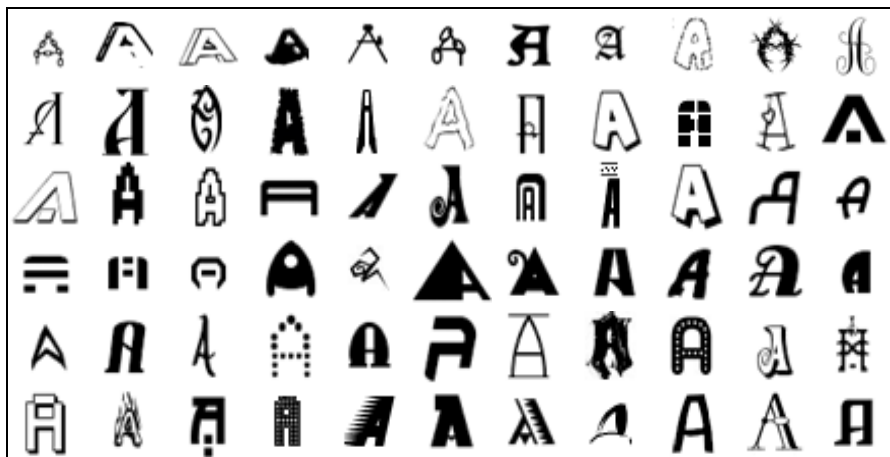


Figure 12: Representing letter 'A' with various abstract fonts.

On a typical Windows 7 machine, there are 257 font families [125]. Considering all 257 font families, three font weights (on average) and three font styles, there are roughly 2300 ways for showing a single letter. For all the 26 Latin characters, this number would be around 60,000. In addition to the fonts provided by operating systems, there is unlimited number of user-defined fonts. Figure 12 displays some examples of letter ‘A’ with a variety of abstract fonts.

Unicode fonts

The continuous evolution of Unicode and its increasing coverage of characters and languages motivate font designers to design a font which maintains a single theme for all different symbols and alphabets and provides a standardized set of glyphs for all Unicode characters. By designing such a font, while keeping the critical differences between disparate alphabets, all noncritical discrepancies amongst them are minimized and they are tuned to work together. However, developing such a font faces its own challenges. It takes much time and effort to design a font for the entire Unicode space. Moreover, font designers who are expert in one language might not appropriately render designs in a different language. Furthermore, some designers believe that harmonization increases the possibility of confusion by reducing distinctive differences between scripts [126].

Currently, there is no single Unicode font that covers all the characters of Unicode. “GNU Unifont” covers the whole Basic Multilingual Plane. A limited number of fonts tend to cover a very large range of Unicode characters and support many Unicode blocks. Such fonts include: “GNU Unifont”, “Code2000”, “Code2001”, “Everson Mano”, “Arial Unicode MS” and “Lucida Sans Unicode”. The Unicode consortium’s suggestion for computer users is to install the fonts for the scripts they generally use as well as a few large-coverage Unicode fonts.

Previously, we talked about the large number of possible ways to show Latin characters using different variations of fonts. This number is quite higher for Unicode characters. Unicode Standard 6.1.0 contains more than 110,000 encoded characters. Using only Unicode fonts and considering variations of each font, such as weight and style, these characters can be displayed in millions of different ways. In addition to Unicode fonts, myriads of fonts have been designed for each script. Regarding the huge number of available font-families and alterations of each font, billions of glyphs exist to display Unicode characters. In Figure 13 and Figure 14, different glyphs for Chinese ideograph ‘字’, which means ‘word’, and Arabic letter ‘FEH’ are displayed.



Figure 13: Different glyphs for Chinese ideograph ‘字’.





Figure 14: Some different glyphs for Arabic character ‘FEH’.

To sum up, as demonstrated by the above arguments, fonts and font variations can additionally extend our input space to include many more different glyphs⁵. However, despite this diversity, it is important not to use multiple fonts of the same script in the same CAPTCHA due to their high similarity which could confuse human users. For example, if the glyphs shown in Figure 14 were used in the same CAPTCHA, many of them could be perceived as identical or close to identical. Hence, in order to effectively use the diversity offered by the myriad of fonts available for a script, a CAPTCHA can use any single one of them but no more.

In the current version of our CAPTCHA, we have selected a group of approximately 6,200 characters from Modern scripts⁶ excluding Hangul scripts. Hangul scripts were excluded due to their large size (11400 characters) in order to reduce the computational time for a number of measures, e.g. similarity measures. Covering 40 different scripts of Unicode, our character set is representative of a variety of modern languages. It covers European scripts such as Armenian, Coptic, Cyrillic; African scripts such as Ethiopic; Middle Eastern languages such as Arabic and Hebrew; South Asian Scripts such as Limbu and Devanagari; Southeast Asian scripts such as Myanmar and Thai; Central Asian scripts such as Mongolian and Tibetan; Philippine scripts such as Hanunoo; East Asian languages such as Japanese and Yi; and American scripts such as Cherokee and Unified Canadian Aboriginal Syllables. With such a variety, our CAPTCHA input space is a smaller version of Unicode. Extending this subset to a larger one can be readily achieved [127].

⁵ Other text-based CAPTCHAs can also extend their input space by using various fonts and font variations. However, since the original input space of most existing CAPTCHAs is usually glyphs of a specific script (usually Latin), it is not very difficult to make OCR systems that are trained to detect the characters of that specific script rendered by a variety of fonts (For English, there already exists many OCR programs that support a large number of fonts).

⁶ Unicode scripts include Modern scripts and Ancient scripts which are not in customary use.

In this version of our CAPTCHA, we use a small set of fonts to render the approximately 6,200 different objects in our object pool. We render each chosen character with a single font to prevent the inclusion of identical glyphs of different fonts from the same script in the same CAPTCHA. The smallest possible set is “GNU Unifont” which can singly represent all modern scripts. However, since its glyphs are not smooth and continuous, we decided to use this font whenever a character was not available in other used fonts. For example 'GUJARATI LETTER AA' can be represented by “GNU Unifont” as  whereas in “Code 2000” it has the smoother shape . Based on this argument, a small set of fonts needs to be defined that can in combination render the whole character set.

In order to select a set of fonts to render the chosen character set in the proposed CAPTCHA, we examined a comprehensive collection of Unicode fonts including: "Code2000", "Arial Unicode MS", "Everson Mono", "Code2001", "Lucida Sans Unicode", "Microsoft Sans Serif", "Times New Roman", "Tahoma", "Arial", "Bitstream Cyberbit", "Cardo", "CaslonRoman", "Charis SIL", "Chrysanthi Unicode", "ClearlyU", "DejaVu Sans", "Doulos SIL", "Free Serif", "Junicode", "Linux Libertine", "Lucida Grande", "TITUS Cyberbit Basic", "Free Mono", "Gentium Regular", "New Gulim", "Y.OzFontN", "Microsoft JhengHei", "sun-extA", "GNU Unifont" and tried to render the characters of modern scripts with each one. The results showed that, ranking after “GNU Unicode” that was able to render the whole 6200 glyphs, “Code 2000” represented approximately 5800 characters. Figure 15 shows how many characters each Unicode font can render [128].

We used “Code 2000” as our primary font due to its smoothness and its relatively high coverage of our character space. From the remaining 400 characters, there are about 200 characters that can only be rendered by “GNU Unifont”; hence we included this font in our

selection, too. "Arial Unicode MS" and "Microsoft Sans Serif" have been selected because they can render more of the remaining characters. The final selected set of fonts comprises “Code 2000”, "Arial Unicode MS", "Microsoft Sans Serif" and “GNU Unifont”.

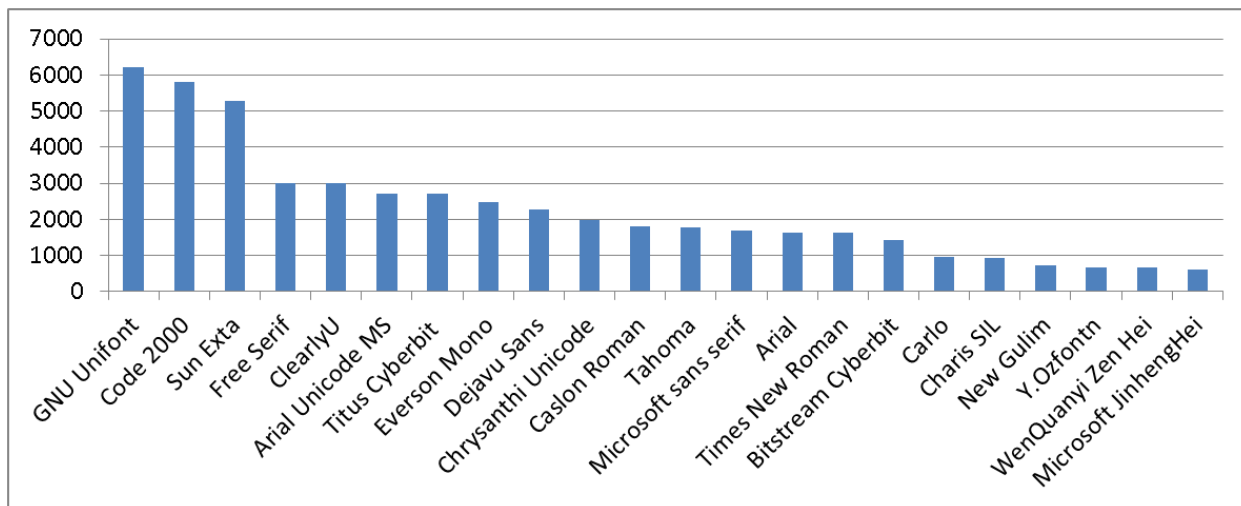


Figure 15: The coverage of each Unicode font of the modern script characters [127].

In conclusion, Unicode, chosen as the proposed CAPTCHA’s input space, provides us with a large database of objects that can be generated automatically. We also have the option of creating a very large set of Unicode glyphs rendered by different fonts and font variations. While using different fonts and their variations can further enhance security, it would also substantially reduce if different fonts of the same script are used in the same CAPTCHA. This issue can be easily circumvented with the restriction to use no more than a single font for each script in the same CAPTCHA while allowing different CAPTCHAs to use different fonts of the same script.

2.5.3 Selection of the characters of the test and the keyboard

We selected modern scripts and rendered all of their characters using the previously described fonts. The subset selected for our CAPTCHA initially included characters with

features that made them look like the background noise. We removed these characters to improve usability. We performed our filtering operation in three steps:

- Removing characters that are very small and with pixel counts that are less than a threshold (65 pixels). Such characters, e.g. punctuation marks, are hard to see in noisy backgrounds. Figure 16-a shows examples of removed characters.
- Removing characters with a very high aspect ratio (9:1). Such characters resemble horizontal or vertical lines and might be confused with background noise. Figure 16-b shows examples of removed characters.
- Visually inspecting the whole character set to see if there is any hard-to-solve characters that the above algorithms could not detect and remove. Figure 16-c displays examples of removed characters in this step.

These three steps remove approximately 330 characters, which leads to 6200 characters being retained.

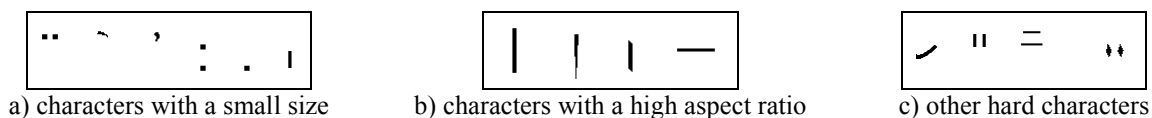


Figure 16: Examples of characters removed to increase usability.

To construct the CAPTCHA, we select 6-8 random characters from the remaining set of characters as our test characters. We make sure that no two characters are identical or homoglyphs⁷. The algorithm of selecting test characters is as follows:

- characterPool \leftarrow all existing modern scripts' characters after filtering
- testCharacterSet \leftarrow empty
- testCharactersHomoglyphs \leftarrow empty
- number of test characters \leftarrow a random number between 6 and 8
- while (testCharacterSet.length < number of test characters)
 - o select a random character from the characterPool (ch_i)
 - o remove the selected character and its homoglyphs from the characterPool
 - o Insert the selected character into testCharacterSet
 - o testCharactersHomoglyphs[i] \leftarrow the selected character's homoglyphs

⁷ Details on homoglyphs are explained in Section 2.6.

The keyboard contains all test characters and a random number of their homoglyphs and other random characters. The algorithm of selecting keyboard characters is as follows:

- keyboardCharacterSet \leftarrow testCharacterSet
- number of keyboard characters \leftarrow a random number between 25 and 30
- number of required homoglyphs \leftarrow a random number between 3 and 5
- while (number of selected homoglyphs < number of required homoglyphs) select a new homoglyph:
 - o $t \leftarrow 1$
 - o if (testCharactersHomoglyphs[t] contains any characters)
 - randomly select one character from testCharactersHomoglyphs[t]
 - remove the selected character from testCharactersHomoglyphs[t]
 - insert the selected character into keyboardCharacterSet
 - o $t \leftarrow t + 1$, if ($t >$ number of test characters) $t \leftarrow 1$
- while (keyboardCharacterSet.length < number of keyboard characters)
 - o select a random character from the characterPool (the pool after excluding test characters and their homoglyphs)
 - o remove the selected character from the characterPool
 - o Insert the selected character into keyboardCharacterSet

The above algorithm creates a test of 6-8 characters. It also creates a keyboard of 25-30 characters which includes the test characters as well as 3-5 homoglyphs per test character.

2.6 Homoglyphs

In our CAPTCHA system, we use homoglyphs of each test character in the virtual keyboard. With this strategy, even if attackers are able to segment the test characters successfully, which is difficult because of using a high level of dynamism and randomness in the presentations, they will still have to find the right glyph among a number of very *similar* glyphs in the keyboard, which is a simpler task for humans than it is for machines.

2.6.1 Homoglyphs in Unicode

There are many visually similar characters in the universal character set (UCS). Two homoglyphs can belong to the same script; For example, Latin lowercase ‘l’ and Latin uppercase ‘i’. Or they might be from multiple scripts. An example could be the Cyrillic letter ‘а’ and Latin ‘a’. Table 5 provides further examples of similar characters in one script. There are also similar characters across different languages. Some examples can be seen in Table 6.

Table 5: Examples of mono-script homoglyphs.

				comment
1	Amharic			(amharic 'h' with 'Salis' vowel) and (amharic 'h' with 'Hamis' vowel)
2				(amharic 'l' with 'Rabe' vowel) and (amharic 's' with 'Rabe' vowel)
3	Sinhala			U+0DB6 and U+0D9B
4				U+0D87 and U+0D88
5				U+0D94 and U+0D95
6	Thai			U+0E05 and U+0E15
7	Balinese			U+1B28 (Balinese letter PA KAPAL) and U+1B58 (Balinese digit Eight)
8	CJK			U+6236 and U+6238 (Chinese ideographs meaning "family")
9	Arabic			U+0643 and U+06A9 have the same Initial and medial forms
10				U+0647 and U+06C1 have the same final forms
11				U+0623 (Arabic letter Alef with Hamza) and U+0672 (Arabic letter Alef with wavy Hamza)
12				U+06A9 (Arabic letter KEHEH) and U+06AA (Arabic letter swash KAF)
13	Tamil			U+0B95 (Tamil letter Ka) and U+0BE7 (Tamil Digit one)
14	Indian			U+A830 (North indic fraction one quarter) U+0964 (Devangari Danda)

Table 6: Examples of complex-script homoglyphs.

1			U+2C69 (Latin Letter K with Descender) and U+049A (Cyrillic Letter K with Descender)
2			U+03C9 (Greek small letter Omega) and U+2375 (APL functional symbol Omega)
3			U+01A9 (Latin capital letter ESH) and U+03A3 (Greek capital letter SIGMA)
4			U+10382 (Ugaritic letter GAMLA) and U+103D1 (Old Persian number One)
5			U+09ED (Bengali digit Seven) and U+0669 (Arabic-Indic digit Nine)
6			U+054F (Armenian capital letter TIWN) and U+0053 (Latin capital letter S)
7			U+0C2C (Telugu letter BA) and U+0DC3 (Sinhala letter Dantaja Sayanna)
8			U+0C8E (Kannada letter E) and U+0DB0 (Sinhala letter Mahaapraana Dayanna)
9			U+0BB5 (Tamil letter VA) and U+0D16 (Malayalam letter KHA)

10	।	।	।	।	U+A830 (North indic fraction one quarter) and U+A8CE (Saurashtra Danda), U+AA5D (Cham punctuation Danda), U+10A56 (Kharoshthi punctuation Danda), U+110C0 (Kaithi Danda)
11	近	近	近	近	U+8FD1 (Glyphs of an ideograph meaning “near” in Chinese, Taiwanese; and Korean from left to right)
12	荒	荒	荒	荒	U+8352 (Glyphs of an ideograph meaning “wasteland” in Korean, Taiwanese, Chinese; and Vietnamese from left to right)

Figure 17 represents some homoglyphs found for the Latin Letter ‘w’, the Arabic letter ‘Yeh’ and the Latin letter ‘o’. Homoglyphs in these examples belong to different scripts (Latin, Greek, Coptic, Cyrillic, Armenian, NKO, Bengali, Gujarati, Tamil, Telugu, Kannada, Malayalam, Unified Canadian Aboriginal Syllables, Thai, Lao, Myanmar, Georgian, Hangul Jamo, Ethiopic, Yi, Lisu, Phags-pa, Japanese, Deseret, etc).

w	ŵ	ɰ	ω	ώ	ẃ	Ẅ	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻	𐌼	𐌽	𐌾	𐌿
0077	0175	026F	03C9	03CE	0461	051D	0561	0bfo	0baf	0CAC	0D27	0E1E	13B3	15EF	164E	1883	198D
ω	Ω	ẁ	Ẃ	ẃ	Ẅ	ẅ	Ẇ	ẇ	Ẉ	ẉ	Ẋ	ẋ	Ẍ	ẍ	Ẏ	ẏ	Ẑ
1B20	1C03	1E81	1E83	1E85	1E87	1E89	1E98	1F60	1F61	1F62	1F63	1F64	1F65	1F7C	1F7D	1FA0	1FA1
Ẓ	ẛ	ẜ	ẝ	ẞ	ẟ	Ạ	ạ	𐄎	𐄏	𐄐	𐄑	𐄒	𐄓	𐄔	𐄕	𐄖	𐄗
1FA2	1FA3	1FA4	1FA5	1FF2	1FF3	1FF4	2CB1	2D0D	A4B3	A4EA	A86D	A99E	AA9D	10436	1D222		
ي	ي	ي	ي	ي	ي	ي	ي	ي	ي	ي	ي	ي	ي	ي	ي	ي	ي
0626	063D	063E	0649	064a	06CD	06D0	06CE	0777	FBFD	FE8A	FEF2						
ò	ó	ô	õ	ō	ơ	ố	ồ	ơ	ớ	ờ	ơ	ớ	ờ	ơ	ớ	ờ	ơ
00F2	00F3	00F4	00F6	014D	01A1	01D2	022F	03BF	03CC	04E7	0585	07CB	09F9	0AE6	0B66	0C66	0ED0
Ō	ō	Ō	Ō	Ō	Ō	Ō	Ō	Ō	Ō	Ō	Ō	Ō	Ō	Ō	Ō	Ō	Ō
1090	101D	110B	12D0	1946	1A14	1A90	1BB0	1C40	1C5B	1ECD	1ECF	1F41	2D54	A4A8	A8D0	AA50	ABF0

Figure 17: Examples of homoglyphs in the Unicode [127].

To utilize the similarity of Unicode characters in our CAPTCHA, we need to measure the degree of similarity between any two glyphs.

2.6.2 Similarity measurement between glyphs

There are many similar glyphs in Unicode space. However, the amount of similarity between two glyphs varies. Our main problem is the definition of similarity and dissimilarity between glyphs. If someone asks: “are these two images similar?”, there is no absolute answer for this question. Instead, a spectrum can show the degree of similarity between two images (Figure 18).

The concept of *similarity* is highly affected by human perception. What one person calls similar might be dissimilar from another person’s viewpoint. Therefore, specifying a boundary between similar and dissimilar is very difficult and changes from person to person [127].

Two images are considered completely dissimilar, if they are random with respect to each other; in other words, if we cannot describe one given the other [129]. Hence, we can define two glyphs as *anti-homoglyphs* (completely dissimilar) if they are random with regard to each other. This is the left-most point in the spectrum of Figure 18. In contrast, a *homoglyph* cannot be defined as a specific point in this spectrum. We can define it as a threshold. Glyphs having a degree of similarity more than that threshold are considered homoglyphs.

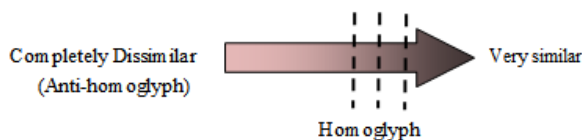


Figure 18: Positions of Homoglyph and Anti-homoglyph in the similarity spectrum [127].

In order to measure similarity, we consider Unicode characters as images comprised of a string of bits and use an operationalization of the Normalized Compression Distance (NCD) metric to calculate similarity scores. This metric is based on the Kolmogorov Complexity theory [130].

The Kolmogorov complexity of an object x , denoted by $K_U(x)$, is defined to be the length of the shortest program that can produce that object on a universal Turing machine. Kolmogorov Complexity theory provides a “viewpoint” on the random content within an object [130]. Kolmogorov defines a string random; if there is no program that can produce it with a length shorter than its own length. The conditional Kolmogorov complexity of a string x given a string

y , denoted by $K(x|y)$, is defined as the length of a shortest program that prints x when y is executed.

The information distance between x and y can be defined as:

$$E(x, y) = \max\{K(x|y), K(y|x)\} \quad \text{Equation 1}$$

$E(x, y)$ is the maximum length of the shortest program that prints x from y and y from x [129]. Since similarity is a relative concept, the above definition needs to be normalized. For example, two strings of 10000 bits differ by 100 bits are considered relatively more similar than two strings of 100 bits that differ by 100 bits. The Normalized Information Distance (NID) between strings x and y is defined as [129]:

$$d(x, y) = \frac{\max\{K(x|y^*), K(y|x^*)\}}{\max\{K(x), K(y)\}} \quad \text{Equation 2}$$

NID is a valid distance metric. However, the Kolmogorov complexity function $K(\cdot)$ is not computable; hence, it also does not have a practical implementation. Cilibrasi and Vitanyi [129] address this issue by replacing $K(\cdot)$ with a practical data compressor function $C(\cdot)$ and casting the metric in terms of this function to introduce the Normalized Compression Distance (NCD) metric [131]:

$$\text{NCD}(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}} \quad \text{Equation 3}$$

In this definition, $C(\cdot)$ gives the compressed length of a string using an arbitrary compression algorithm and xy or yx represent strings produced by concatenating x and y strings in different orders.

We used the above NCD definition to measure the degree of similarity between pairs of glyphs. According to Kolmogorov theory, two homoglyphs must be highly compressed using the

information from each other. However, two “anti-homoglyphs”, are random with respect to each other and hence cannot be compressed using information from each other.

Challenges of realizing the NCD for glyphs

The first step in calculating the NCD is to produce images of Unicode characters. Our images include glyphs of a group of approximately 6200 characters from Modern scripts (see Section 2.5.2 for details). To render the images, we need to decide the font face, the font size and the dimensions of images. In an ideal situation, the same font face should render all the glyphs. However, it is impossible to use a single font to render all the characters since there is no specific font covering the whole Unicode space. Therefore, the least possible number of fonts is utilized for this purpose (i.e., the four fonts as discussed in Section 2.5.2). In order to make two glyphs comparable, we fixed the font size and the dimensions of the images. While arbitrary, the dimensions were set to 1000 pixels to ensure that the compressors had sufficient information to achieve high compression rates where applicable. In addition, Unicode includes many complicated characters, especially in non-Latin scripts. The image representing such complex characters should have sufficient resolution to show all their details.

The selection of a compressor, which is appropriate for our data, is another issue. The most important parameter in NCD estimation is to choose an optimized compression algorithm. The ideal compressor for our purpose is one that if employed in the NCD formula, can lead to NCD scores of close to zero in comparing each glyph with itself. Among compressors that have the above property, those with higher speed are more favorable as we are dealing with a large number of images. Having images (glyphs) as our inputs, we can use either a 1 or 2-Dimensional compressor. While 2-D compression algorithms treat glyphs as rectangular images, 1-D methods consider them as byte-streams. Dealing with an image as a string destroys the spatial

relationships between pixels. A 1-D compressor needs to linearize two-dimensional images into a one-dimensional object.

Another important step in the measurement of NCD is concatenating two images. A 2-D compressor must decide the direction of concatenation, and how to concatenate when the images are of different sizes; a 1-D compressor just concatenates two strings.

Different references use various compressors such as bzip2, gzip, PPMZ and JPEG2000 to measure NCD [131]. Cilibrasi [129] argues that “the NCD minorizes all similarity metrics based on features that are captured by the reference compressor involved”. Chen et al. discuss that although 1-Dimensional compression algorithms destroy the spatial relationships in an image, they produce better results than 2-D algorithms [131]. Because of this uncertainty, we performed an experiment to evaluate a cross-section of compressors for our problem.

For linearization, we have used row-by-row scan of the image. Mortensen et al. compare several types of linearization (e.g., row-by-row, column-by-column, Hilbert-Peano and self-describing context-based pixel ordering) and find that despite producing different NCD values, none of them uniformly outperforms the others [132]. We follow prior researchers in similar applications [131], [133] and use a row-by-row scan of the image for linearization.

Design of experiment

To select a suitable compressor, we used the NCD formula to calculate the similarity of each glyph with itself. A higher NCD value (for (image₁, image₂)) expresses more dissimilarity between them. To select a compressor, we performed an experiment – basically, NCD(image_i, image_i) where i is all the letters in our selected group. We ran this experiment several times for different 2-D compressors (PNG, GIF, JPEG and JPEG2000) and 1-D compressors (Gzip, Bzip2,

PPMD, Deflate and LZMA). While using 2D compressors, we considered both horizontal and vertical concatenation methods to calculate NCD.

Figure 19 and Figure 20 show histograms of the results for a sample of 1-Dimensional and 2-Dimensional compressors. In each graph in Figure 19 and Figure 20, the horizontal axis represents the range of NCD values and the vertical axis indicates the frequency of images having those NCD values when compared to themselves. Since in this experiment, the NCD value of each glyph against itself is measured, the best graph should have a peak around zero. Our results indicate that the NCD measures calculated with the LZMA compressor outperform other alternatives in recovering similarity scores in our application. Consequently, we chose LZMA as our chosen compressor.

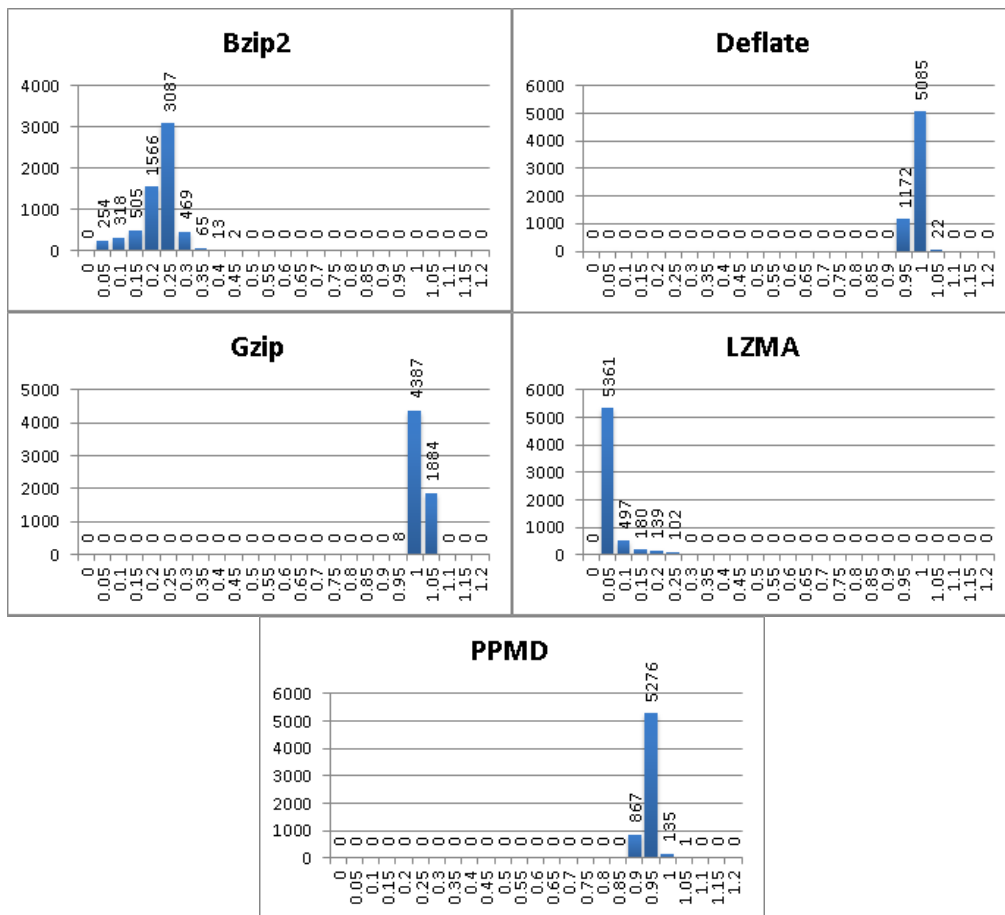


Figure 19: Histograms of NCD values calculated by different 1-D compressors [127].

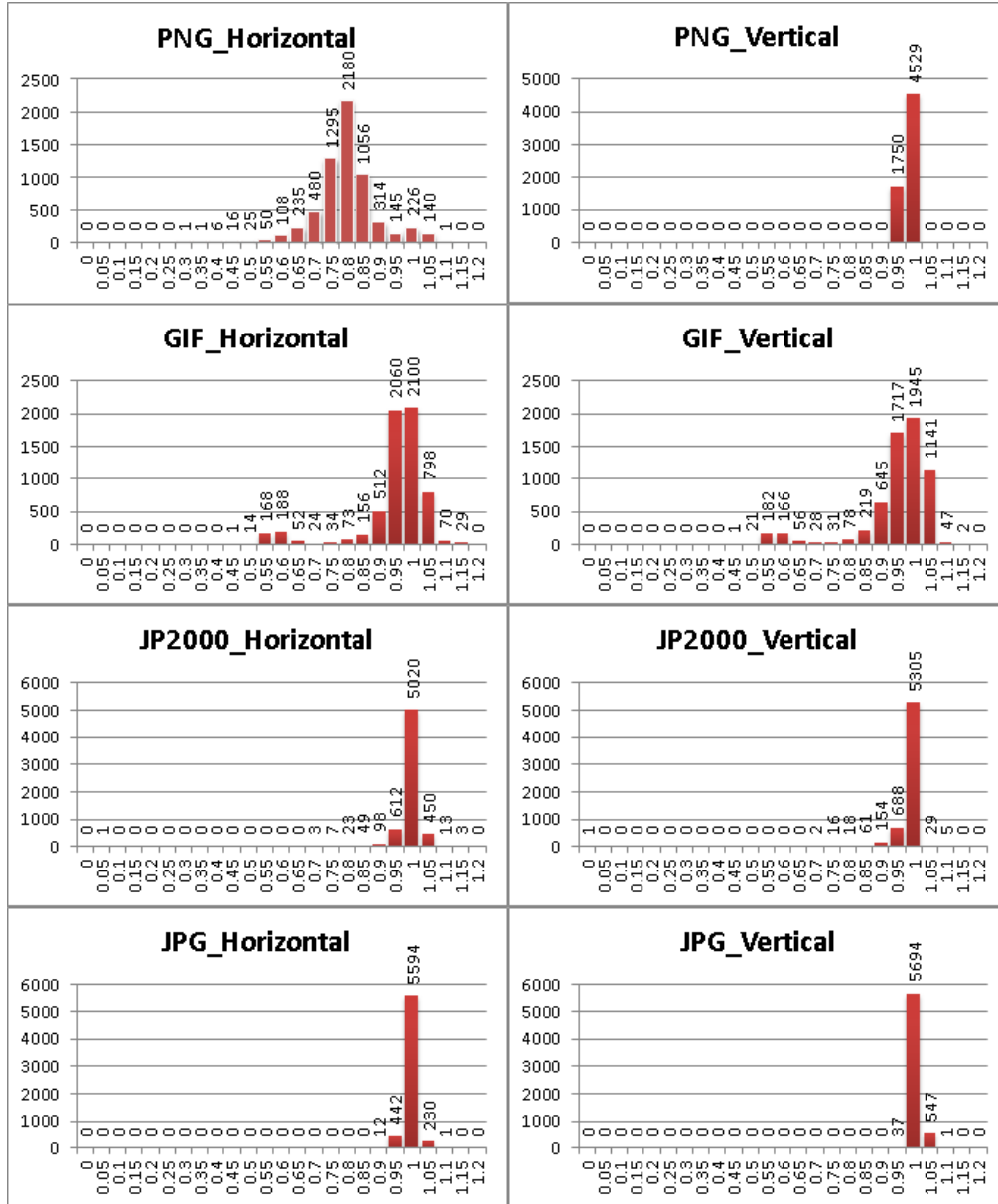


Figure 20: Histograms of NCD values calculated by different 2-D compressors [127].

Experimental Results

We calculate the NCD values between each character and all other characters in our Unicode subset, which amounts to more than 19,000,000 pairwise comparisons. Figure 21 is a histogram of these results. Some example character pairs and their corresponding NCD values are displayed

in Table 7. The two glyphs in the first row (𐄂,𐄃) have a very small NCD and are considered very similar. In contrast, glyphs in the last row (𐄂,𐄃), have a high NCD value of 1.04, and are considered dissimilar. However, specifying the amount of similarity required to call two glyphs *homoglyphs* is very challenging and varies depending on different people’s perspectives. For example, the answer for the question “are the glyphs in row 9 (𐄂 and 𐄃) homoglyphs?” changes from person to person.

Figure 22 represents the cumulative distribution function of the results. As can be seen, many pairs have significant NCD values. This means that most glyphs in the Unicode space are dissimilar. An elbow appears in the CDF at NCD=0.8. If someone decides this point as the similarity threshold, 634,461 out of 19,709,781 pairs are considered homoglyphs; this constitutes 3.2 percent of all records. The remaining 96.8% (19,075,320 pairs) are dissimilar. The existence of such a large number of homoglyphs in Unicode space does not seem reasonable and many counter-examples exist; see Table 7 for example, although two glyphs in row 11 with NCD=0.71 have some similarity (two long vertical lines, etc.) from a compression viewpoint; they do not look very similar from users’ viewpoint. Figure 23 magnifies the left part of CDF graph (where $0 < \text{NCD} \leq 0.75$). Three further elbows can now be detected (0.6, 0.5 and 0.3). If we select the threshold to be 0.6, 0.48% of the pairs will be considered similar and 99.52% will be dissimilar. The ratio of dissimilar to similar pairs in this case is 19614321:95460 (205:1). 95,460 similar pairs is still a large number and again from Table 7, pairs with NCD values around 0.6 (e.g. 𐄂 and 𐄃) are not really visually similar. The ratio increases to 392:1 for a threshold = 0.5; and to 1934:1 for a threshold = 0.3. In the last case, 10189 out of 19,709,781 (0.05%) of pairs are considered similar. This number is probably more realistic. As can be seen in Table 7, pairs with NCD values less than 0.3 (rows 1-7) are visually very similar.

Choosing this value for the threshold leads to homoglyphs for 1844 of our 6200 Unicode characters (approximately 30%). Table 8 shows the frequency of homoglyphs for our Unicode subset given the NCD threshold of 0.30. Raising the threshold to 0.6 would lead to 50% of our glyphs ending up with homoglyphs at the cost of lower similarity for some pairs. However, our CAPTCHA does not require each character to have a homoglyph. In our CAPTCHA keyboard, we use three to five homoglyphs from the combined pool of homoglyphs that exist for any randomly selected six to eight test characters. Having homoglyphs for 30% of the characters adequately addresses this need while keeping similarity levels between homoglyphs high⁸.

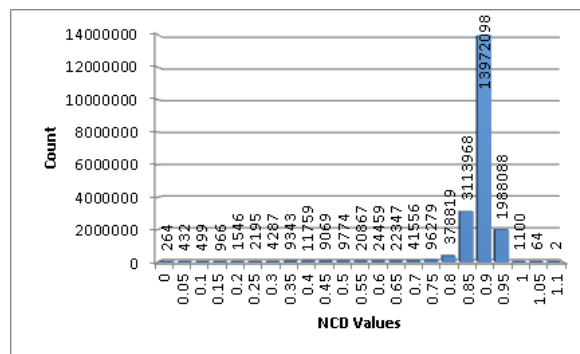


Figure 21: The histogram of the pair-wise NCD calculation over all characters in our set [127].

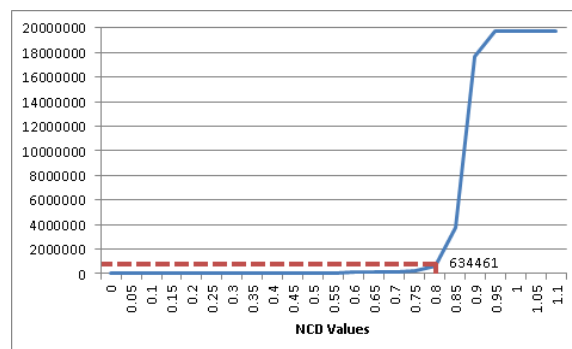


Figure 22: Cumulative distribution function of pair-wise NCD calculations [127].

⁸ As a provision for the less likely cases where none of the six to eight test characters have homoglyphs (Pr=12% to 6%), we choose higher NCD values to define homoglyphs.

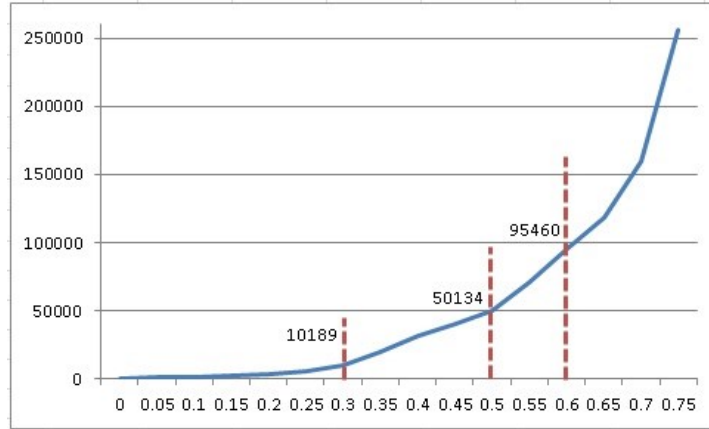


Figure 23: CDF of results with $0 < NCD \leq 0.75$ [127].

Table 7: Examples of NCD values between some characters.

Row	Glyph1	NCD	Glyph2
1	0b95 (Tamil Letter Ka)	0.027	0be7 (Tamil Digit One)
2	A165 (YI Syllable NDUP)	0.134	A167 (YI Syllable NDUR)
3	0100 (Latin Capital A with Macron)	0.154	04D2 (Cyrillic Capital A with Diaeresis)
4	03CE (Greek Small Letter Omega with Tonos)	0.157	0461 (Cyrillic Small Letter Omega)
5	071B (Syriac Letter Teth)	0.224	071C (Syriac Letter Teth Garshuni)
6	06A9 (Arabic Letter Keheh)	0.282	06B1 (Arabic Letter Ngoeh)
7	01D1 (Latin Capital Lette O with Caron)	0.3	1ED6 (Latin Capital O with Circumflex and Tilde)
8	142A (Canadian Syllables Final Down Tack)	0.393	1682 (Ogham Letter Luis)
9	05D3 (Hebrew Letter Dalet)	0.484	A3A8 (YI Syllable NRYP)
10	1701 (Tagalog Letter I)	0.602	1954 (Tai Le Letter SA)
11	0389 (Greek Capital Letter Etta with Tonos)	0.71	1964 (Tai Le Letter I)
12	0A1A (Gurmukhi Letter CA)	0.909	0AA7 (Gujarati Letter DHA)
13	0BB7 (Tamil Letter SSA)	1.00	188F (Mongolian Letter Ali Gali Nna)
14	1696 (Ogham Letter Or)	1.04	0DA3 (Sinhala Letter Mahaapraana Jayanna)

Table 8: Frequency of the number of homoglyphs at NCD threshold=0.3.

# of characters	4356	922	349	157	102	314
# of homoglyphs	0	1	2	3	4	>5

2.7 Virtual keyboard

Our CAPTCHA system requires a virtual graphical keyboard to accommodate the large input space provided by Unicode. Using a virtual keyboard offers additional advantages such as the possibility to dynamically populate the keyboard with different subsets of Unicode, random

layout of characters and random modifications of each character. These possibilities and the specific purpose a CAPTCHA keyboard serves necessitate a closer look at how the virtual keyboard should be developed.

A CAPTCHA keyboard completely differs from the standard keyboard in terms of key configuration and layout, relative key positions, the number of keys and the content of each key. While factors such as relative key positions and fixed layout essentially guide the design of a physical keyboard, they can be entirely ignored in a CAPTCHA keyboard design. Our keyboard does not need to conform to any particular pattern of character usage. Therefore, relative key positions do not matter. Furthermore, a fixed keyboard layout, where the position of each potential character is known, is in direct contrast to the general CAPTCHA design philosophy of avoiding patterns that can be exploited by attacks. With these considerations in mind, the keys, their positions and the symbols they represent are chosen randomly in our virtual keyboard. While the range of the number of keys on the keyboard can be considered a design variable, their exact number is randomly determined from a proposed range. Further design decisions rely on striking a reasonable balance between usability and security.

Most existing research pertains to the design of a virtual keyboard for a specific language and for applications other than CAPTCHAs. For example, designers aim to optimize key sizes and key arrangements to minimize the statistical travel distance between characters and to increase text entry speed. The specifications of the keyboard language such as the frequency of letters or occurrences of digraphs are important in the design of their keyboards. For example, more frequent letters should be located in more convenient positions and more frequent digraphs should be closer together than less frequent digraphs [134]. Moreover, the user is often familiar with the language the keyboard is designed for. However, the above line of research does not

provide much guidance in designing an effective virtual keyboard for our purpose. The reasons are twofold. First, our keyboard contains Unicode characters most of which are not familiar to the users. Second, the purpose of our keyboard is to prevent machines from recognizing the objects it holds (glyphs) while facilitating the same task for the users. This is contrary to the other purpose of keyboards, which is to allow users to type words and sentences efficiently and conveniently. Hence, our keyboard design requires a different set of decisions. Instead of determining the locations of specific characters for our keyboard, we have to make decisions about the number of objects, their size, inter-object space, the number of similar objects or homoglyphs and the objects' color. These features have to be designed in such a way that achieves a reasonable balance between security and usability.

In our CAPTCHA test and keyboard, target characters are surrounded by other non-target elements and background noise. In such a combination of elements, the target might become difficult to detect depending on the degree of similarity between targets and non-targets. This similarity can be along various dimensions such as color, contrast, shape and size. Furthermore, the number of distractors and the spacing between objects might also affect detectability. Each of these dimensions will be discussed in this section. Though the studies and experiments to be discussed here were conducted under specific conditions and assumptions which might not render them immediately generalizable to our setting, most of them can guide our research by offering insights into the human vision system and by highlighting the important factors in humans' visual search for an object surrounded by distractors. Hence, we next discuss these dimensions in turn from both usability and security viewpoints and show how their findings can inform our CAPTCHA design, if at all.

The size of the characters

We define the character size as the total area a character occupies. Prior research indicates that human vision is usually limited by object spacing rather than size [135]. These findings show that object size does not significantly affect recognition⁹. From a security viewpoint, the largeness-smallness of glyphs might not be important as OCR programs or pattern matching algorithms are not designed to find a specific character size. However, changing size from one character to another and from one test to another will confuse an attacking computer program by defying a predictable stable pattern.

According to the above finding and argument for dynamism, random variation of character size seems to be more important than the size itself in designing a CAPTCHA. In the proposed CAPTCHA, the size of the characters should not be very small since we use the large Unicode space and many of the characters included in a test and keyboard are not only unfamiliar to the users, but are also overlaid on a noisy background. As a result, if we choose a very small character size, they could be readily missed by users and usability could be significantly compromised. We chose to use dynamic random sizes for characters from a predetermined comfortable range. The lower bound of this range has been determined in a way that allows characters to be reasonably discernible and the upper bound is jointly determined with the number of characters which will be explained in the next section. Our strategy to randomly determine character sizes within the selected range avoids the formation of a fixed pattern for character size, which can confuse computer programs while not significantly affecting human recognition abilities.

⁹ Evidently, character size should be higher than a lower bound.

The number of characters

Generally, the number of objects directly affects the search time when distinguishing them needs attentive search [136]. Researchers have also shown that set size does not affect search efficiency as much as clutter does [137]. One potential reason for the latter finding might have to do with the fact that the effect of set size on search efficiency might be substantially smaller than clutter. Nevertheless, using a smaller set of characters is desirable from a usability viewpoint.

From a security viewpoint, increasing the number of elements enhances security. However, given space limitations, the maximum number of objects that can be placed in the keyboard without any overlap should be determined jointly with the size of the characters and their spacing. Increasing the size of the characters, so that human users can recognize them more easily in a noisy background leads to a reduction in the amount of space available for keyboard characters which, in turn, makes segmentation easier for a computer program. Hence, a balance is required between the size and the number of characters.

To determine the number of keys on our proposed keyboard, we refer to the purpose that the keyboard serves, namely, acting as a repertoire from which correct CAPTCHA elements can be conveniently picked by human users. This purpose readily determines the minimum number of keys, which is basically equal to the number of elements in the test section of the CAPTCHA. Using this lower bound as the number of keys on our virtual keyboard ensures maximum usability as far as the number of keys is concerned. For a test comprised of 6-8 elements, the minimum number of keys on a keyboard would be 6-8. However, from a security point of view, it is evident that increasing the pool of potential solution characters will reduce the probability of any successful attack. Hence, it is reasonable to let the elements in the keyboard outnumber the

test characters to any extent permissible by other conflicting criteria. In fact, this extent is limited by the time and cognitive resources of CAPTCHA users. As the number of keyboard elements increases, usability is being reduced in favor of security and users might be less willing to engage in CAPTCHA solution or might experience negative feelings.

No prior research provides direct guidance on how to determine an optimum upper bound for the number of CAPTCHA keyboard elements. Given the above discussion, while any decision on this upper bound might not be a definitive one, we use a simple rule of thumb to determine an upper bound. Assuming there is a distance of approximately 50% of a character between two characters, the keyboard in our CAPTCHA can contain between 28 to 42 characters. In order to make the search job easier for human users, we limit the number of keyboard characters to the range of 25 to 30, about four times the number of our chosen test characters of 6 to 8.

In sum, the number of test characters in our CAPTCHA is chosen randomly between 6 and 8 and their sizes vary from 4% to 7% of the test box. The size of the keyboard characters is selected randomly between 1% and 1.5% of the size of the keyboard box.¹⁰ The number of keyboard characters, 25 to 30, is still approximately four times as many as that of test characters, which does seem to achieve the additional security benefits associated with having a higher number of keyboard characters.

In this proposed keyboard, the existence of homoglyphs and a noisy background further complicates search for attackers.

¹⁰ As a security measure against pattern matching algorithms, we also stretch one of the characters of each matching pair in a random horizontal or vertical direction. This strategy slightly changes the relative character dimensions.

The number of homoglyphs

Prior research in human vision studies confirms the intuitive prediction that increasing the number of similar elements [138] as well as increasing target-distractor similarity [139] increases search time. For example, Williams [138] argues that if we measure the similarity between a target and each of its i distractors and denote them by S_i , the search time can be calculated from Equation 4. According to this formula, if the search area contains many similar items, the search time will be higher.

$$t = \frac{\sum S_i}{6} + 1/2 \quad \text{Equation 4}$$

Bloomfield [139] indicates that search time per element increases with target–distracter similarity. He ran an experiment in which four differently-shaped targets: a square, a hexagon, an octagon and a dodecagon had been shown to the user among some disk-shaped distractors. The results of the study indicated that as the shape of the objects became more similar to the disks, more time is required to locate them among non-target disks.

Analogously, similarity between objects makes it harder for computer programs to successfully match objects; more sophisticated algorithms with higher computational complexity will be needed. In contrast, human users are better equipped than robots in distinguishing objects with slight differences [64]. Nevertheless, having so many similar objects in the test or keyboard could make human users frustrated; hence the number of homoglyphs must be limited. There are between 3 and 5 similar objects in our keyboard. Similar characters are selected from the pairs with $NCD > 0.03$ and $NCD < 0.30$. Based on our discussion in Section 2.6, we consider pairs with $NCD < 0.30$ as similar. According to the results, pairs with $NCD < 0.03$ are too similar – almost identical- and a human user might not be able to distinguish them, hence they are not utilized on the keyboard.

Color

Color is one of the most effective factors that can help humans in their visual search. The authors of [140] performed an experiment in which the observer was asked to search for a target among many simply-shaped objects of different shapes, sizes and colors in a field. The result showed that subjects find some characteristics more effective than others and tend to ignore less effective ones in their searches. The most important characteristic was color. Size and shape were respectively in the next places.

From a security viewpoint, color, if not utilized appropriately, can offer clues for segmentation; hence, a secure color design is one that hinders computer programs from segmenting the virtual keyboard. Having foreground objects with distinct colors from the background helps attackers to segment them more easily. For example, making a virtual keyboard with two colors, one for the foreground and the other for the background increases the likelihood of an attacker teasing the foreground apart from the background. A good idea to substantially enhance security of a CAPTCHA by color is to include foreground color into background or vice versa. Security issues related to the color are further discussed in Section 2.8.

In designing our CAPTCHA, 10 to 15 randomly-selected colors are used to paint characters and the backgrounds of both the test and the keyboard. Every test character in our test has a specific color and each test character and its match and homoglyphs in the keyboard have the same color. The reason for this selection is that, as discussed earlier, color helps humans in visual search more than other features. We also included all test foreground colors in the test background to increase security based on the above argument. Since the number of keyboard characters is more than the number of colors, random groups of keyboard characters have the same color. Again, each keyboard foreground color also appears in the keyboard background. All

homoglyphs available in the keyboard have the same color to increase the security (as explained in Section 2.8). The selection of colors is discussed in detail in Section 2.8.

The location of the characters

One of the weaknesses of current CAPTCHAs is that the position of their characters is known. This information singly or coupled with knowing the total number of characters, make CAPTCHAs vulnerable to segmentation attacks. Randomizing character locations makes this task potentially harder for an attacker by removing exploitable locational patterns from the layout.

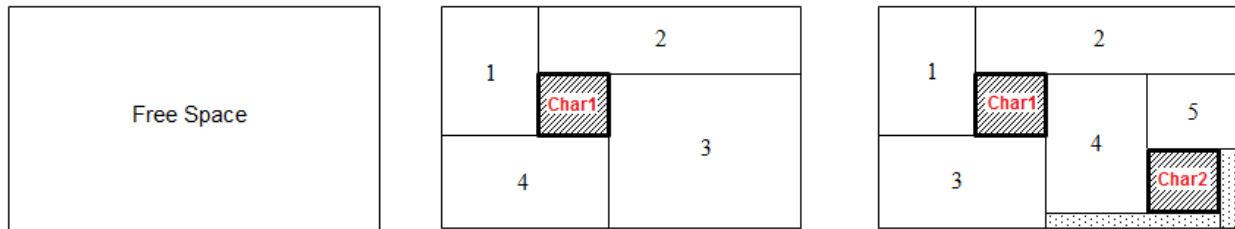
In this CAPTCHA, we sampled from two uncorrelated uniform distributions to determine the random vertical and horizontal locations of characters on the two-dimensional spaces of both the test and the keyboard. Our guiding principles in determining random character locations are to avoid patterns while giving all points an equal chance for inclusion. These considerations led to the following criteria and decisions:

- Different points in the 2-D test and keyboard spaces should be equi-probable, which rules out all non-uniform distributions. Only uniform distributions have this property.
- Correlation between the two dimensions is not necessary and actually is by itself a pattern to be avoided. Hence, x coordinates of characters should be independent from their y coordinates. This means that the samples from the two uniform distributions for the x- and y-coordinates should not be correlated.
- Full coverage of the space is not an issue in our sampling for character locations since it is by itself a pattern to avoid. For example, full coverage in our CAPTCHA would mean that if the space is covered by an appropriately-sized grid, each cell would contain one character. Subsequently, sampling methods that aim to cover the entire space, e.g. stratified sampling methods are not acceptable for our problem.

A simple uniform sampling on each of the two dimensions meets the above requirements.

We used the following algorithm to get a random point set as the location of the characters:

- FreeSpace \leftarrow The whole space (Figure 24-a)
- PointSet \leftarrow Null
- While (FreeSpace is not Empty or pointSet.length < number of the characters), Find a location for a character and subtract this character's area from the FreeSpace:
 - o Randomly select one rectangle from the FreeSpace (using a random number which is weighted based on the size of the rectangles)
 - o Exclude the selected rectangle from the FreeSpace.
 - o Select a random x and a random y in the selected rectangle (condition: the character should totally fit in the rectangle) and locate the character box at that location.
 - o Extend the boundaries of the character box parallel to the boundaries of the rectangle. This will create 4 new rectangles (Figure 24-b).
 - o For each new rectangle, if it is large enough to contain a character, add the new rectangle to the FreeSpace (In Figure 24-c two of the new rectangles are small and hence ignored).
- If (pointSet.length == n) return pointSet;
- If (FreeSpace is Empty) return "unsuccessful".



a) 2D area in the beginning b) After locating the first char c) After locating the second char

Figure 24: Three first iterations of "random point set selection" algorithm used in this CAPTCHA.

The distance between characters:

Vlaskamp et al. [141] showed that at distances smaller than approximately 50% of object size, the search time per element increased with reducing element spacing. At wider spacing, distance did not affect the search time at all. Based on these studies, object spacing is not very important in human visual search and as long as a minimum space of about 50% of an object is retained between the elements, the effect of spacing on search time will be minimal for humans. For an attacker, objects are difficult to segment when they overlap. Many CAPTCHAs use this strategy to secure their tests since humans are good in segmenting connected objects. In cases

where the glyphs do not overlap, the amount of space between them does not seem to affect the difficulty of segmentation for computer programs.

In designing the proposed CAPTCHA, we arranged a minimum random space between characters to ensure they do not overlap. We do not allow characters to overlap because overlapping the unfamiliar multi-part Unicode characters would highly confuse the users. Humans are good at distinguishing connected known characters, but when the characters are unknown, have multiple parts, and some of them are similar, it is more difficult for them to recognize the characters.

2.8 Color representation

Color representation is an important topic in CAPTCHA design. While appropriate utilization of colors in the CAPTCHA background or using colored characters can enhance the CAPTCHA interface, inappropriate usage of color can have negative consequences. In this section, we review the necessary trade-offs in the usage of color in CAPTCHAs and present our approach to using color to enhance our proposed CAPTCHA.

Color is an eye-catching mechanism that can facilitate human recognition and comprehension. Past studies show that color can make a CAPTCHA look more interesting which will make it less intrusive [83]. In addition, color can improve the security since many ordinary OCR software packages do not perform well in segmenting and recognizing color images. However, color could have a negative impact on security and/or usability when used improperly. Examples of improper color usage include the existence of difference between foreground and background colors, the absence of foreground color in the background and the existence of any pattern in the foreground text that distinguishes it from the background.

According to the above argument, there should not be a detectable distinction between the luminance/chrominance of foreground and that of the background in a strong CAPTCHA. One way to reduce such distinctions is to introduce variations in foreground and background colors and include some foreground colors in the background and vice versa. A colorful foreground on a colorful background containing at least some of the foreground colors could be potentially highly confusing for machines. Such color variations can be introduced into the background as random noise to prevent the formation of “detectable patterns” that could possibly defeat the purpose of camouflaging the foreground text. The details of the background generating algorithm are explained later in this section.

Background noise can have a “detectable pattern” if it is comprised of colors, shapes, locations or distribution in the image that differ from the characters. For example, if the noise has a different color than the foreground color, it can be distinguished by luminance/chrominance filters. As another example, when the size of noise elements is significantly different from the size of characters or character elements, a “detectable pattern” will be created since characters have a specific range of width-length ratio which will be different from the same ratio for noise elements. As another example, if characters occupy specific regions of the CAPTCHA, a “detectable pattern” will be formed. Still in other types of CAPTCHAs, the distribution of noise information could be different from that of the characters. For example, the noise may be widespread but with a small amount of information. These weaknesses have been diminished in our proposed CAPTCHA.

In our proposed CAPTCHA, color is used as described here: each character in the test has the same color as its own match and its homoglyphs in the keyboard but is in a different color than the other test characters. All colors of test characters appear in both test and keyboard

backgrounds as well. Provisions are made so that no test or keyboard character lies on the same background color as its own color. Due to considerations that will be explained shortly, the number of colors used is larger than the number of test characters and smaller than the number of keyboard characters. Consequently, there are colors in the test background which do not appear in the test characters and some characters in the keyboard share the same color. The background is a combination of elements of different colors. These elements, which comprise the background noise, have random locations and sizes. In many cases, the shapes of these elements are similar to character elements and are hardly distinguishable from them. The background elements are distributed all over the image just as the characters are. The elements are not too small or too large and have approximately the same width-length ratio as the characters. In order to prevent extreme similarity between noise and character elements, we allow a mixture of various background noise sizes as opposed to all noise element sizes being similar to the size of character elements. Evidently, if the background elements are very similar to the characters, the CAPTCHA becomes more secure but it also becomes much more difficult to solve for human beings. Our currently chosen background noise granulation, which will be shortly discussed, is a result of a few experiments that helps strike a balance between usability and security.

The first background we tested contained noise elements that were more similar to the characters than the current one. The previous background was designed to contain elements which were very similar to the characters and had the same approximate width as the elements that comprise characters. The background was a result of a random walk over the area where each step comprised of creating a polygon with random dimensions – with either straight, convex, or concave sides. As Figure 25 represents, the width of the background elements were very similar to that of the characters. Not surprisingly, this property made the CAPTCHA very

difficult and confusing for human users. The users could hardly detect the characters which made them quickly feel overwhelmed and/or frustrated. Hence, we changed the background generating algorithm to reduce the similarity between background and characters by modifying the size and shape of background noise elements in a step-wise procedure until some test characters became detectable for a few human users who participated in our test.



Figure 25: The first background of the proposed CAPTCHA.
 a) a piece of the keyboard, b) examples of background components, c) characters to compare with background components.

The current background contains elements with a mixture of shapes and sizes. It includes elements with the same width as the character parts to confuse computer programs as well as larger elements that can be relatively-easily differentiated from the characters by human users. The existence of the larger elements has the advantage that users can see a number of the characters at their first glance which prevents immediate user frustration.

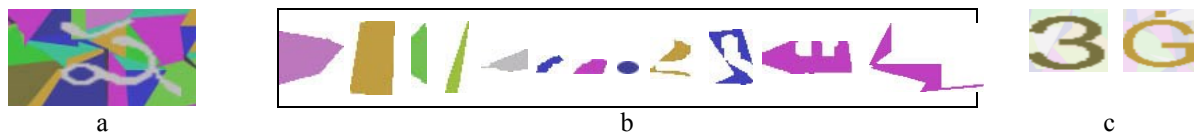


Figure 26: Background of the proposed CAPTCHA.
 a) a piece of the keyboard, b) examples of background components, c) characters to compare with background components.

The background generating algorithm is composed of four phases: The first phase creates large objects; the second phase creates smaller objects that are similar to characters or character parts in terms of shape and width. In the third phase, some randomly-created curved noise components are included into the background. Because most of the characters contain curves, including curved components in the background will confuse computer programs trying to break this CAPTCHA. If an algorithm tries to remove other noise components based on their size and

shape, this type of noise will not be removed because these noise components have the same shape, size, and features as the characters. Finally, in the fourth phase, random parts of a test character are incorporated as noise in the keyboard background. Since character parts have the same features as the characters themselves, this technique would further confuse matching algorithms. The probability of the random part of a character being very similar to another character is very small and it never happened in our user studies. However, even if it occurs, it would not be a problem because these two similar objects would not have the same color (each test character, its match, its homoglyphs and its random part have the same color which is different from other test characters). Even if this similarity happens and the user ignores the colors for some reason, the user can just decide not to match the ambiguous character and solve another character (there is an extra character in each test.)

The random overlap between different noise components creates random shapes some of which are similar to the characters. As Figure 26-b illustrates, background components can be wider than the characters; they also can be as thin as the characters or as small as character parts. Figure 27 shows examples of similarity between background components and characters or character parts.



Figure 27: Examples of similarity between background components and characters or parts of the characters.

The design of the background is further affected by our decision to equalize the usage of all colors in the image. As a result of this color balance, if a computer program uses a color histogram of the image to differentiate foreground from background, no dominant color will be

found and a relatively uniform histogram will emerge that contains little clue, if any, on how the foreground and background differ from one another in terms of color.

An important decision related to color usage is the number of colors to be employed in the CAPTCHA. In choosing the number of colors, there should be a balance between the number of colors and the amount of space painted by each color. Increasing the number of colors can potentially cause more confusion for the attackers in their attempt to differentiate test characters from background noise. However, it can also facilitate CAPTCHA solution for the attacker. In our CAPTCHA, the percentage area of the character in a given instance of the CAPTCHA is fixed. Since the area is being equally distributed among all colors, more colors lead to smaller total areas for each color. Since the total area for each color is comprised of foreground character area and background noise area, with foreground area fixed, the background noise is decreased for each existing color. An example helps further clarify this point. Consider a character that occupies 2% of a rectangular area. When the total area is covered with 5 colors, each color receives a 20% share of the whole area and the share of noise for each color will be 18%. However, when 10 colors are used, each color receives a 10% share of the area and 8% of the area is devoted to noise for each color. Consequently, in such a case, if the attacker uses color segmentation, finding the character of that color will become easier in the latter case as the noise has decreased by more than 50%.

In general, the number of colors to be used in the CAPTCHA can be decided based on the size of the display, the test area, and the test characters¹¹. Keyboard area does not affect this decision because it is larger than the test area. Since the same set of colors is used for both the

¹¹ The size of characters, test and keyboard changes depending on the size of display. In addition, since the number and size of the characters are random, the average size and number has been considered. All of the calculations are approximate since a great degree of dynamism exists in designing our CAPTCHA.

keyboard and the test areas, the test area constraint is a more binding one and acts as the bottleneck in the decision making.

To determine the actual number of colors we try to find a maximum possible range of candidate values and narrow them down to a more realistic range to choose from after taking account various considerations. To begin, we can argue that the number for used colors can be between 8 and 100. The lower limit 8 is the number of test characters. Since we have a maximum of 8 characters in a test and they have different colors, we have to have at least 8 colors. The upper limit of 100 is chosen since each character approximately covers 1% of the test area.¹² In such a case, with 8 test characters that have different colors, there are 92 character-sized micro-areas of character noise and no foreground color can appear in the background. However, having too many colors (approximately 100) or very few colors (e.g. 8) has its own disadvantages.

As demonstrated, with too many colors, many foreground colors might have little, if any, appearance in the background. This happens because the additional used colors take up valuable background space that could be filled with noise of the same color as the foreground text. In the extreme case of no appearance, as exemplified by the 100-color example, also in the case of little appearance, finding the character with color x in between a small amount of background noise with color x would be easier after color x is segmented. A large set of colors can also complicate the task for human users.

With a large number of colors, it is more likely that many selected colors are similar to each other which would substantially confuse human users and easily lead to frustration and

¹² The size of a test character box is approximately 5.5% of the test area. However, approximately 83% of the character box is free background space, as determined from analyzing a sample of characters (by counting the number of black pixels and white pixels in the images of 100 characters), the character itself occupies 17% of the box, and the size of a character will be approximately 1% of the test area ($0.17 * 5.5\% \approx 1\%$).

dissatisfaction. To appeal to human users, the number of colors should be reduced to a number at which human perception can conveniently distinguish between colors¹³. Though human beings are able to perceive many shades of any of the major colors, it will be considerably more complex for them to notice small variations in a colorful noisy background. Hence, a more realistic upper bound for our CAPTCHA would be closer to the number of distinct major colors and farther away from the theoretical upper limit of 100. Moving on the HSB color wheel at 30 degree intervals creates 12 major colors. While finer gradations of major colors might exist that allow for a somewhat larger number of them to be considered as major colors, we choose stay close to 12 as an approximate median of the maximum number of colors in our CAPTCHA.

The lower limit of 8 must also be improved. Having a few more colors than the colors of characters causes confusion for an attacker who is trying to distinguish characters from background noise. If there are only 8 colors in the test, an attacker can segment each color and, knowing that there is a character with each color, employ pattern recognition algorithms on each segmented color to find the character. On the other hand, if, there are more colors than the number of test characters, segmentation and pattern recognition should be used on more images some of which merely contain noise. This can increase the amount of effort required to break the test.

Eventually, we randomly select the number of colors between 10 and 15. By this selection, the number of colors is more than that of test characters, and each color has a considerable presence in the test background. For example, with 12 colors, each color can appear 7 times as large as the size of a character in the background. In this case, about 2.25 characters in the

¹³ Color-blind people will not have any problem solving this CAPTCHA because they can match characters based on their shapes (similar shapes). Moreover, when they use color palette, it shows them only one color at a time (whose hue is not important in a monochrome image).

keyboard have the same color and that color constitutes a large portion of the keyboard background noise. To make segmentation even harder for a computer program by involving more colors in the proposed CAPTCHA, we can paint every character or background component with a gradient of the selected color. The gradient can be created by changing the saturation and/or the lightness of that color. After selecting the number of colors, a decision about the method of color selection is necessary.

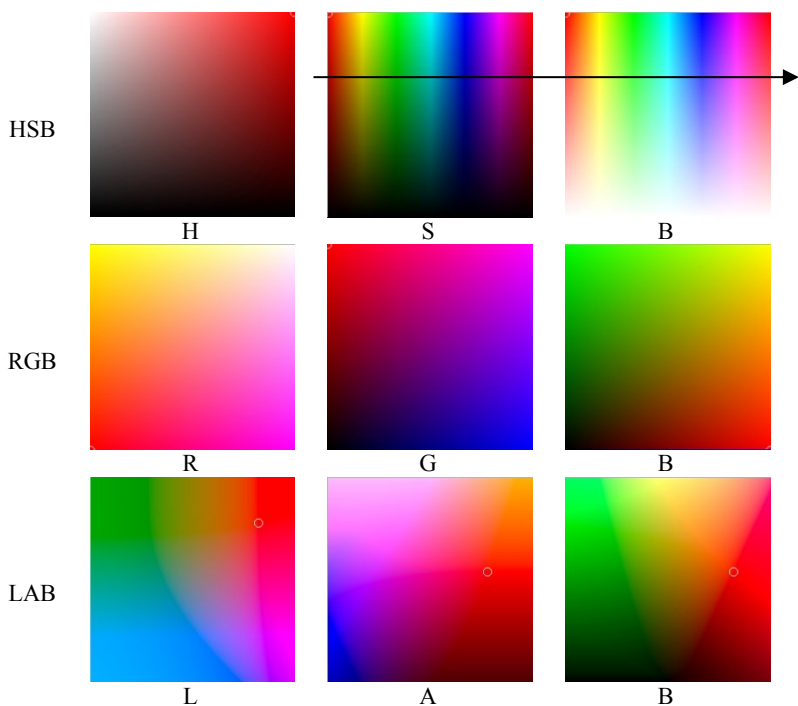


Figure 28: Demonstration of the ability to traverse major colors in HSB by changing hue.

Colors have been selected using systematic random sampling. We used the HSB color space to produce distinct colors while allowing sampling from the whole range of hues. Systematic color creation is much easier in HSB than RGB and CIE-LAB. In fact, holding the saturation and brightness parameters constant and varying hue in HSB leads to more predictable color changes than using a similar process in either the RGB or CIE-LAB. This is due to the fact that neither

RGB nor CIE-LAB includes a parameter whose variation traverses the whole spectrum of hues. Intuitively, varying the hues in HSB while holding saturation and lightness constant leads to a more diverse set of distinct colors than varying the R, G, or B parameters in the RGB or varying L, A, or B in the CIE-LAB. In Figure 28, each letter under each square indicates the dimension that is fixed for the square above it. It can be seen that changing hue (going from left to right) when S (saturation) or B (brightness) are fixed in the HSB space enables us to traverse the whole range of major colors whereas none of the other color spaces show this characteristic for any of their elements.

Security and usability considerations

In the color representation of this CAPTCHA, we balanced considerations of usability and security. We painted each test character and its matching keyboard character with the same color because, according to the previous arguments, color helps humans in visual search more than other features such as size and shape. On the other hand, to improve the security, character colors are also included in the background. Moreover, similar characters available in the keyboard, i.e. homoglyphs, are in the same color; therefore, even if a computer program is able to segment a color, they will still have a hard time distinguishing between very similar characters. What makes the decision more difficult for a machine is that many Unicode characters consist of more than one part. Recognizing different parts of a character and deciding if a component belongs to a character or to the background noise is a challenge which is easier for the human to tackle than it is for the machine. Another security strategy is that a test character and its matching keyboard character –and their homoglyphs in the keyboard- do not have the same size. This strategy makes pattern matching more effortful. For example, we may have *é* in the test and *e*, *é* and *ê* in the keyboard, none of them with the same size. The background noise contains many elements

similar to the ‘dot’ and ‘acute’ which will substantially confuse a computer program. Another superiority of humans over machines in solving this CAPTCHA is that according to gestalt theory, humans see an object in its entirety before perceiving its individual parts; therefore, if a character intermingles with its surrounding noisy background and some parts of it are not readily detectable, humans are inherently better at detecting such characters than machines due to their more holistic visual processing system.

In order to substantially increase the usability of the CAPTCHA, while not significantly affecting the security, a color palette has been designed that further assists users to solve the CAPTCHA by filtering out additional colors when they select a specific color on the palette. This palette, by reducing the complexity of the noisy background, increases the user-friendliness of the CAPTCHA. To further enhance the security of our CAPTCHA, we paint every character or background component with a gradient of a color rather than a simple color which will convert segmentation from a discrete task to a continuous one for the machine.

2.9 User studies and results

In order to examine the usability of the proposed CAPTCHA, we conducted a user study and measured some relevant performance indicators in order to establish how our CAPTCHA fares. In this study, we asked users to solve tests and recorded relevant data, such as the correctness of their solutions and solving time. Each test was also followed by a short questionnaire that measures the users’ evaluation of major aspects of the proposed CAPTCHA. In this section, the details of the study and its findings are presented.

2.9.1 Design of the user study

In this study, we asked 120 students at the University of Alberta to each solve our CAPTCHA once. They had different first languages including English, French, German, Romanian, Spanish, Korean, Vietnamese, Chinese, Assamese, Sinhala, Farsi and Arabic. We made a decision not to collect much demographic information to allow the test to be as quick as possible to facilitate user participation. We used a laptop with a 12.5” matte monitor for this experiment. The main purpose of this data collection was to measure the amount of time it took users to solve the test as well as the number of tests and characters they could solve correctly.

Our CAPTCHA test consisted of six to eight random characters from the chosen Unicode subset. The users were required to solve all the displayed characters except one in order for their solution to be considered correct¹⁴. The keyboard included 25 to 30 characters of which three to five would be homoglyphs of the test characters.

An additional purpose of this data collection was to realize what type of characters are difficult for users, whether “more difficult” characters have a specific shape, belong to a specific writing script, have a specific color or are located in a specific position in the test area, etc. Our interest in these factors is rooted in the expectation that character detectability could be affected by character size, the location of the test character or the matching keyboard character, the shape of the character, color contrast, etc. For this reason, we recorded the solution time of each character as well as for the whole test, test characters’ solving order, color, location, keyboard characters’ color and location, palette clicks, usage of zooming facilities, etc.

¹⁴ The reason for allowing one character to remain unsolved is discussed in question 4 of the results.

Besides the general metrics such as *solving time*¹⁵, *solving accuracy*¹⁶, and *rate of giving up*¹⁷, we explored the data to search for answers to a series of questions in order to find clues or draw inferences that could help improve our CAPTCHA. These questions include:

1. What types of test characters are being more frequently mismatched?
2. What type of test characters are people good at detecting?
3. What characters are users less likely or less inclined to solve as revealed by solutions?
4. Is there a relation between a character's solution order and its solution time?
5. Is there a relation between the location of a test character and its correct/incorrect answer?
6. Is there any relation between color and solution time, order or correctness of the answer?
7. Is there any relation between alphabet and solution time, order or correctness of the answer?
8. Do people use the palette and/or zooming tools?
9. Do users enjoy solving the CAPTCHA?

2.9.2 Results of the experiment

From a total of 120 tests, 93 were solved correctly, 25 were answered incorrectly (failed) and two tests were withdrawn from. Hence, solving accuracy was 77.5% , which is reasonable in comparison with reCAPTCHA's 75% and Google CAPTCHA's 86% [6]. The rate of giving up was 1.7%. Interactive CAPTCHAs have longer solution times¹⁸ than simpler CAPTCHAs because of the added processes involved to find and submit the solution. For example, 3D CAPTCHA has a solution time of 45.9 seconds [113]. The average solving time for our CAPTCHA was 111 seconds which is relatively longer. However, a fair comparison of these

¹⁵ For measuring efficiency.

¹⁶ A measure of correctness with which users can respond to a CAPTCHA challenge without making mistakes.

¹⁷ The number of CAPTCHAs given up divided by the number of all CAPTCHAs.

¹⁸ We use "solving time" and "solution time" interchangeably.

solution times must take into account the level of security a CAPTCHA affords. Unfortunately, no standard metric for comparing solution time per security level exists. A summary of the results for our experiment is displayed in Table 9. Figure 29 displays the histogram of user solution times.

Table 9: Mean solution time of a test.

	total	Grouped		
		Correctly solved	Incorrectly answered	Withdrawn
Number of tests	120	93	25	2
Average solving time	111 s	102 s	138 s	189 s
Solving time SD	30.5	22.1	33.5	4.9

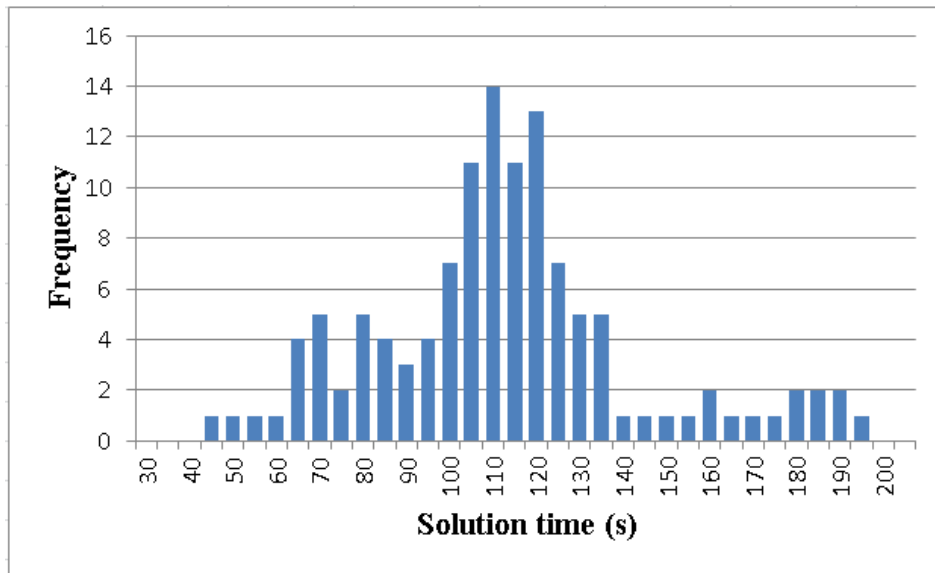


Figure 29: Solution time histogram.

Subjects used the palette in 94% of the tests and zooming in 15% of the tests. These results show that subjects' usage of the color palette was higher than the zooming tool which might indicate that the color palette is more helpful than the zooming tool in solving the tests. The low usage of zooming is consistent with our expectation that zooming may not substantially enhance detectability. The observed 15% usage was in most cases due to single trials rather than consistent usage. Only in 2% of the tests, was zooming used more than once.

In the following part, we discuss the insights derived from our user study pertaining to the above questions.

1. What types of test characters are being more frequently mismatched? And what keyboard characters are more frequently being mismatched for each test character?

From all 120 tests, only 25 tests failed. A sample of mis-identified characters is displayed in Table 10.

Table 10: Incorrectly matched characters.

	Test character solved incorrectly	Selected match	Available similar characters
1	9	9̄	9̄9̄9̄9̄9̄
2	8	8	8
3	ŵ	ŵ	ŵŵ
4	Ä	A	ÄÄ
5	10	10	1010
6	×	ƒ	**

** The selected match is not considered similar to the test character based on our NCD calculation thresholds.

As observed in Table 10, similarity between characters seems to be the major cause behind most of the mistakes. In 23/25 wrong tests, the mistakes were due to similarity between the target character and the incorrectly-selected character (see Table 10-rows 1-5). Only in two cases, the real match and the selected match were not similar (see Table 10, row 6). Consequently, similarity seems to be the dominant cause of wrong answers.

2. What type of test characters are people good at detecting?

People more easily solved characters that have distinguishable shapes as compared to simple shapes that look like background noise. Table 11 represents a few of these distinguishable characters and Table 12 shows some examples of characters with simple shapes.

Table 11: Examples of distinguishable characters.

ㄋ	𐄂	𐄃	𐄄	𐄅	𐄆
𐄇	𐄈	𐄉	𐄊	𐄋	𐄌

Table 12: Examples of simple-shaped characters.

𐄍	𐄎	𐄏
---	---	---

Aside from the above observed pattern, we are especially interested to study users' solutions of similar characters or homoglyphs. Though a few people failed to solve the *similar* characters correctly, in many cases (86%) they chose the correct match (see Table 13 for some examples). These results indicate that people are nevertheless good at distinguishing similar characters.

Table 13: Similar characters distinguished correctly.

Test character	ý	Ū	Ɖ	𐄂	フ	ó	ǎ	9	⌘	ᄁ	ğ	⌘
Similar characters in the keyboard	ÿ ÿ̂ y	Ū U̇	Ɖ	𐄂 𐄃 𐄄	フ	ö "O	ǎ	9 9̇ 9̈	⌘	ᄁ	ğ ğ̇ ğ̈	⌘ ⌘̇

Results show that in many cases (58%), users correctly solved characters with similar glyphs sooner rather than later (e.g., as the first, second, or third character in their solution order). This observation of a rather balanced occurrence at the early versus the late solution orders suggests that test characters with similar glyphs might not be substantially different from other characters in term of detectability.

3. *What characters are users less likely or less inclined to solve as revealed by solutions?*

Every test in this experiment included six to eight characters and users were asked to solve five to seven of them. Hence, one character is not solved by each user. Figure 30 displays examples of these characters. Evidently, users were less likely or less inclined to solve simple-shaped characters that may be more similar to background noise and less detectable than the other competing characters in the test. Although we removed characters with very small sizes and high aspect ratios, there still are some characters similar to the background noise. Applying a stricter threshold to eliminate these smaller characters can further enhance usability.

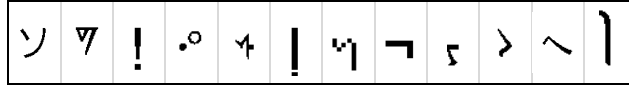


Figure 30: Characters that users are less inclined to solve.

4. Is there a relation between a character’s solution order and its solution time?

We do expect to observe a relationship between a character’s solution order and its solution time. Conceivably, users start solving the CAPTCHA by matching the characters that are more salient from the background, e.g. by starting from characters that attract their attention and look more promising in terms of being readily matched. The salience could come from dissimilarity to other characters, higher contrast from the background, color, or merely larger character size. If the above prediction were true, we would be able to observe substantially longer solution times for the last solved characters as compared to the first solved characters.

We expected to detect the easy-to-hard solution pattern from the characters’ solution times; if the last characters are the hardest to solve, the solution time for the last characters should be noticeably higher than the solution time for the first characters. The results provide support for this prediction (Figure 31) and are consistent with our intuitive expectation that users tend to spend a longer time on the last characters as they initially attempt to solve the more salient characters.

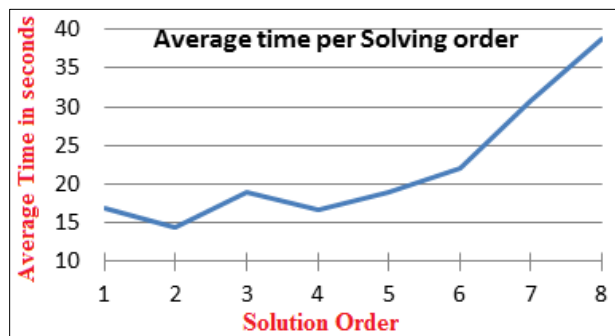


Figure 31: Average time for different solving orders.

The easy-to-hard solution pattern can be used to further enhance CAPTCHA usability. This can be achieved by allowing users to solve m characters from the pool of n test characters where $m < n$. This provision will allow users to avoid the hardest-to-detect character(s) and still achieve a correct overall solution. As a result, overall solution times will decrease without posing a significant threat to security. In our CAPTCHA, we used this effect to some extent by allowing users to solve one fewer character than those existing in the test. Such provisions improve usability by decreasing overall solution time.

As a result of the above finding, we required the remaining 86 users to solve 7 out of 8 characters in their tests in order for their submissions to be accepted. This modification improves usability by decreasing overall solution time, as indicated in Figure 32.

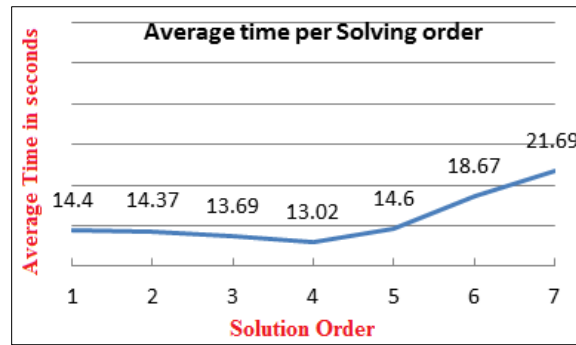


Figure 32: Average time of different solving orders.

5. Is there a relation between the location of a character and its correct/incorrect solution?

It is conceivable to expect that the location of the characters on the keyboard might have an effect on their correct solution. In their search of the keyboard to find matching test characters, users might scan the whole keyboard area but focus more heavily on different areas of the keyboard at different periods during their search. Given the existing background noise in our CAPTCHA, it is conceivable that these more prospective areas of the keyboard are generally

determined by the unique combination of the foreground and background colors in those areas. For example, a user solving this CAPTCHA is more likely to initially focus on areas of the keyboard where a sharper contrast between the foreground text and the background noise makes some character salient from the background. After checking these sharper contrast areas, if the CAPTCHA is still unsolved, the users might extend their search to areas that are harder to search, i.e. locations with slighter contrast or color variations between the foreground and the background.

As the variations between foreground and background colors are randomly determined, we expect easy-to-search and hard-to-search areas to be randomly dispersed on our keyboard across its different realizations for different users. This random dispersion is expected to lead to an on average insignificant impact of character location on the solution of the CAPTCHA. Hence, it is less likely that the mere Cartesian coordinates of an area on the screen determine the intensity of search in that area. We tested this prediction by running a logistic regression with *correctly_solved* (0/1) as the dependent variable and *location on test* and *location on keyboard* (central, middle and outer) as categorical explanatory variable. Location was determined based on what region a character was located on the test area as illustrated in Figure 33.

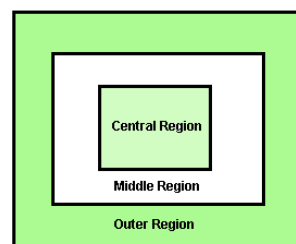


Figure 33: Defining a categorical variable for location.

The results of the logistic regression indicated that none of the three levels of the two location variables had a significant effect on the correct solution of characters suggesting that location did not affect correct vs. incorrect matching of individual characters.

6. *Is there any relation between color and solution time, order or correctness of the answer?*

In this CAPTCHA, the contrast between foreground and background color is a more effective factor than the hue of the character colors. As Figure 34 shows, the average search time for characters of different colors does not change sharply. Also, our analysis did not indicate any specific pattern in the solution order, solution time, or correctness for different colors.

7. *Is there any relation between alphabet and solution time, alphabet and solution order or alphabet and correctness of the answer?*

The results of our user study show that there is no specific relation between alphabet and solution time or the correctness of the answers. However, we do observe a slight preference in first solving the characters of the more familiar languages such as Latin and Greek (see Figure 35). Results do not show any sign of users' higher ability to solve the characters of any specific language, even their own first language or other familiar languages. In 8 out of 25 incorrectly-solved tests, the false character belonged to the user's native language or Latin or Greek alphabets which were familiar for the users. The incorrectly-solved characters are from different alphabets; this means there is no specific alphabet that is difficult for most users. Figure 35 shows the percentage frequency of characters solved in orders 1 to 7, for four scripts with the highest occurrences in our sample. A regression analysis of the solution order (treated as an interval-scaled variable) on these four script types (treated as a four-level categorical variable) shows that the script type does not have a significant relationship with the solution order (F-statistic (3,407)=0.81, p-value=0.49, $R^2=0.01$).

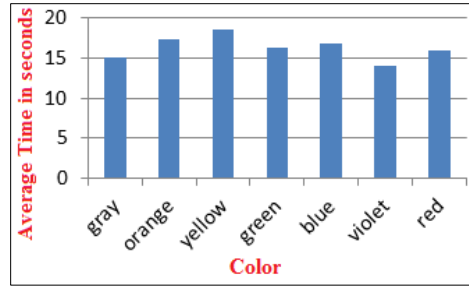


Figure 34: Average search time for characters of different colors.

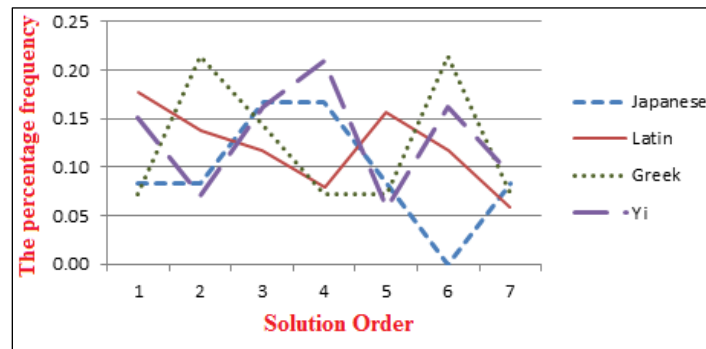


Figure 35: The percentage frequency of characters in each order for the four highest-occurring alphabets.

8. Do people use the palette and/or zooming tools?

In order to assess the usefulness of the palette and zooming tool, we studied:

- a) How often people used them;
- b) If there is a relation between the frequency of palette usage and the number of correct answers and solution time;
- c) What colors in the palette have been selected more often; and
- d) If they have been clicked in a particular order or randomly.

Did people use these tools? In contrast with *zooming*, which was tried in only 15% of the tests and was re-used in only 2% of the tests, the *color palette* was used in 94% of the tests and its usage averaged 13.7 clicks per test. This observation suggests that users found the palette

more helpful than zooming. Although it takes time from users to click on the palette, they would prefer to use it to reduce the complexity of the image.

Is there a relation between the number of palette usage and the number of correct answers and solution time? No relation is observed between the number of palette clicks and test time or the correctness of the answers. However, the average number of palette clicks in most failure and withdrawal cases is relatively higher than successful cases. This observation suggests that people were trying to use the palette more often when they had more difficulty finding the characters.

Which colors in the palette were clicked more often? Were they clicked in any particular order? We do not find any evidence of a specific color being clicked more frequently than others; the order seems to be random.

9. Do users enjoy solving the CAPTCHA? Do they experience any negative feelings (e.g. frustration, etc.)?

In order to collect user feedback in this study, we provided users with a questionnaire after the test. This questionnaire contained five questions and a textbox in which the users could enter their comments. The questions were presented using a five-point Likert scale. Table 14 shows a summary of user answers to these questions. According to this table, 91% of the users felt comfortable in solving this CAPTCHA. 72% of the users believed that the problem was not hard to solve; and only 3% found this CAPTCHA difficult. 92% of the users agreed that this CAPTCHA was easy to learn. Finally, 14% of the subjects found the zooming facilities useful and 82% agreed that the color palette was helpful.

In most of the comments, users mentioned that the test was straightforward, especially using the color palette. One person mentioned that informing the users about the high amount of

similarity between some characters (homoglyphs) would prompt them to be more attentive to details.

Table 14: Users' responses to the survey following the CAPTCHA.

	strongly agree	agree	neutral	disagree	strongly disagree	no answer
I felt comfortable in solving this problem.	56%	35%	8%	0%	0%	1%
The problem was hard to solve.	0%	3%	21%	35%	37%	4%
It is difficult to learn to use this system.	0%	2%	2%	48%	44%	4%
The zooming tool was helpful.	6%	8%	60%	17%	8%	1%
The color palette was helpful.	67%	15%	10%	0%	6%	2%

In addition to the questionnaire, we asked the users after each test to explain their experience and their feelings during the search operation. Most subjects found solving this CAPTCHA to be an interesting experience and some of them were interested in solving more tests after they failed/succeeded their assigned tests.

Summary of user study results

To sum up, based on the results of this user study, we identified factors that are more/less effective in the usability of the proposed CAPTCHA. According to the results, character color, alphabet and location do not seem to have a significant impact on solving the test. The most noticeable cause of confusion in solving this CAPTCHA seems to be the similarity between characters or between characters and background noise; however, the results show that this level of complexity for users leads to an accuracy rate of 77.5%, which is comparable with currently used CAPTCHAs [6]. This outcome suggests that the overall complexity of our CAPTCHA is not higher than most current CAPTCHAs. Including a color palette and allowing people to solve $n-1$ out of n test characters had a significant role in increasing accuracy and decreasing the solution time. However, our tests were on average solved in 111 seconds which is relatively high compared to many current CAPTCHAs. Two major factors that contribute to this higher solution

time include the interactive process required to solve our CAPTCHA and our use of unfamiliar Unicode characters. In our CAPTCHA, these features address several sources of vulnerability alongside other measures in order to achieve a higher level of security. Consequently, our CAPTCHA's solution time cannot be directly compared with the available data on most current CAPTCHAs that are not interactive and exhibit these vulnerabilities.

While shorter solution times are inarguably preferred to longer solution times, acceptable solution times for a CAPTCHA should depend on the required amount of security, which will vary from application to application. For example, CAPTCHAs that protect access to sensitive information such as banking or credit card accounts require a high level of security. In addition to such business to consumer (B2C) transactions, many organizations in the business to business (B2B) sector, including financial institutions and supply chains, rely on Electronic Data Interchange (EDI) systems that operate through the World Wide Web and require highly secure CAPTCHAs to control access to sensitive or proprietary business data or to verify transactions. However, many CAPTCHAs currently used in these applications exhibit major vulnerabilities. In e-banking alone, more than 40 CAPTCHA schemes that are used by a large number of financial institutions worldwide have recently been broken with either close to or equal to 100% success rates [38, 39]. This underlines the need to adopt more secure CAPTCHAs in these applications. Achieving high security with longer solution times is a viable option in many such applications provided the users remain engaged in the task.

The relatively longer solution times of our CAPTCHA can be used to further enhance its security. For our CAPTCHA, a lower bound on solution time can be determined after accounting for the effect of learning on solution time. For example, a human being may not be able to solve our CAPTCHA in less than, e.g. 20 seconds. This lower bound can be used as a restriction to

disqualify any solution attempts that are completed in less than this lower bound which can be attributed to non-human users. This feature alone enormously increases the solution cost for any intelligent algorithm.

One question regarding the usability of our CAPTCHA is yet unanswered. It is not clear how usability measures would change over a prolonged period of usage. It is conceivable that familiarity with the CAPTCHA over several usage occasions and the resulting learning would significantly improve our usability measures over time. This issue will be explored in future trials.

2.10 Conclusion

In this chapter, we introduced our proposed interactive CAPTCHA. In this CAPTCHA, we aim to achieve a high level of security against several prevalent CAPTCHA vulnerabilities by using the large Unicode space as our CAPTCHA's input set, using a virtual keyboard instead of a physical keyboard, incorporating homoglyphs, character segments, and curved noise components in the virtual keyboard, using a balanced color representation, and requiring human interaction for solution submission.

Several design considerations allow us to prevent the CAPTCHA from becoming too complex for human users. The results of our user studies discussed in Section 2.9 indicate that users solved this CAPTCHA with accuracy comparable to existing CAPTCHAs. In the following two chapters, we further explore the issue of security while keeping usability in check.

Chapter 3 Risk Diversification with a Multi-Problem CAPTCHA

3.1 Introduction

Our current CAPTCHA poses a single challenge to a potential attacker. Hence, it can be vulnerable to improvements in matching algorithm technologies. Hence, in order to further increase its security, we convert it from a single-problem to a multi-problem CAPTCHA by a simple modification to how the matching task is performed. In fact, recent research that documents breaking reCAPTCHA with 99.8% accuracy suggests that solving an image puzzle should not be the only factor for distinguishing humans from machines [94]. From a security viewpoint, a multi-problem CAPTCHA is more robust against attacks because an attacker needs to solve multiple distinct problems and the risk of the CAPTCHA being broken is diversified across these problem types. From a usability viewpoint, solving more than one problem might take more time and/or be more difficult than solving just one problem. Hence, it is important to reduce the time and difficulty of each step in a multi-problem CAPTCHA for human users to keep the CAPTCHA appealing for human users.

Our base CAPTCHA requires a user to click two characters in order to match them. While the clicking task might be difficult for machines to emulate, it does not pose an intellectual challenge to them. To take advantage of human users' versatile information processing ability, we modified the matching task from simply clicking the identified characters to a more interactive drag-and-drop task which makes the CAPTCHA amenable to the inclusion of extra problems. For example, the drag and drop task can include sub-problems that can be easy and intuitive for human users to solve while being difficult for machines.

With the above modification, our CAPTCHA's proverbial security eggs are not in one basket anymore. If an algorithm can find an efficient way to identify potential matches within the CAPTCHA, it still has to be able to solve additional problems in order to solve the CAPTCHA.

In order to address the potential decrease in usability, we reduced the number of UNICODE characters users need to match from 6 to 4, which makes the test shorter for users.¹⁹ In our security analysis, we will show that this change will not reduce security while it affords us the advantage of diversifying the risk of the CAPTCHA being broken.

Our modification of the matching task allows us to use interactive dynamic operations that are difficult to solve for robots. We designed and implemented three different interaction types in the tests. One of these interaction types is randomly selected for each test. A user/attacker will have to correctly identify matching character pairs, and also pass the interactive test in order to solve the CAPTCHA.

With the addition of the new interactions, the order of users' operations to solve our CAPTCHA will be as follows. Users can use the zooming facilities and the color palette to find four pairs of matching characters (one in the test and the other in the keyboard). To match each character, they will have to click and drag either of the two characters from the test or keyboard onto the other one. However, when a user clicks on a character to start dragging, the area between the test and keyboard is populated with moving elements that are either obstacles or targets. When faced with *obstacles*, the user must prevent the mouse cursor from touching the moving objects in their matching attempt whereas when faced with *targets* they should ensure that the mouse cursor touches an object with a specific characteristic, e.g. a red colored circle

¹⁹ The actual number of characters in the proposed CAPTCHA is randomly chosen from 5 to 7 with 6 being the average.

from a bunch of moving circles. This simple interactive game enhances the security of this CAPTCHA by making its successful solution dependent on intelligent mouse movements²⁰.

The three types of interactions that we designed help pose a combination of challenges to robot attackers. First of all, they are designed to have many random properties in terms of time, location, size, color, movement directions and speed. Furthermore, these interactive games are displayed when a user clicks on the test or keyboard, and they disappear when the user releases the mouse button or after a fixed amount of time. When a user clicks again, the random properties of the interactive game lead to the creation of an entirely new version of it. The type of the interactive game does not change within each instance of the CAPTCHA.

The above properties make it complex for an attacker to process the interactive game and find the solution to simulate. Even if an attacker is able to find a solution, the dynamism in the interactive game requires careful timing and thoughtful movements of the mouse, which are difficult for a machine to simulate in real time. Furthermore, the machine has to solve the problem and simulate the solution for each character pair separately. While improving security, the addition of the proposed interactions did not compromise usability. Interestingly, most users enjoyed the game-like drag and drop operations.

In this section, the designed interactive operations are introduced.

3.2 Introducing each type of Drag and Drop operation

We use three sample interactive operations that are designed and embedded in this CAPTCHA:

²⁰ The attacker needs to perform all steps of the attack *online*. This means the attacker has a limited time to process the images of a CAPTCHA test and pass each interactive game. It is not possible for an attacker to download the CAPTCHA and process it offline since every time a completely new CAPTCHA is created and interactive games have many random properties.

- Hitting a randomly-moving ball (i.e., a target-touching task),
- Passing through rotating bars without touching them (i.e., an obstacle-avoidance task),
- Passing through moving gates (i.e., an obstacle-avoidance task).

A short introduction of these operations follows below.

3.2.1 Hitting a randomly-moving ball

In this interaction type, when a user clicks on the CAPTCHA, a random number of balls (between 3 and 6) are displayed in the empty space between the test and the keyboard areas. These balls move randomly and the user is prompted to touch a ball of a specific color. This target color is also selected randomly and is announced to the users in a box (Figure 36-a). When a user clicks on the display or keyboard and starts dragging, the moving balls appear and the other area, i.e. test or keyboard, becomes inaccessible by “barbed wire”. The “barbed wire” and the moving balls disappear when the user touches the target ball with the mouse pointer making the target area accessible for dropping the character.

From a security viewpoint, the random selection of the balls’ color, speed, direction of movement, initial location, and size make it difficult for machines to track them. The existence of other non-target balls forces a robot to conduct a search operation rather than simply tracking any moving object. Machines cannot identify ball colors based on the descriptive test that informs users of the target ball’s color. For example, *red* could be any shade from the red range rather than RGB (255, 0, 0).

In order to pass this interactive game, a machine has to distinguish the target object, predict its subsequent location by capturing a few screenshots, which is difficult given the balls’ random walk movement, and mimic human behavior in touching it with a mouse pointer. The machine

has to capture screenshots and process those more frequently, which imposes additional computational complexity. In addition, machines cannot circumvent this mechanism as skipping the task and moving the mouse pointer to the target area would create jumps in mouse movements that are easily detectable.

From a usability aspect, human users can easily understand and handle this interaction type.

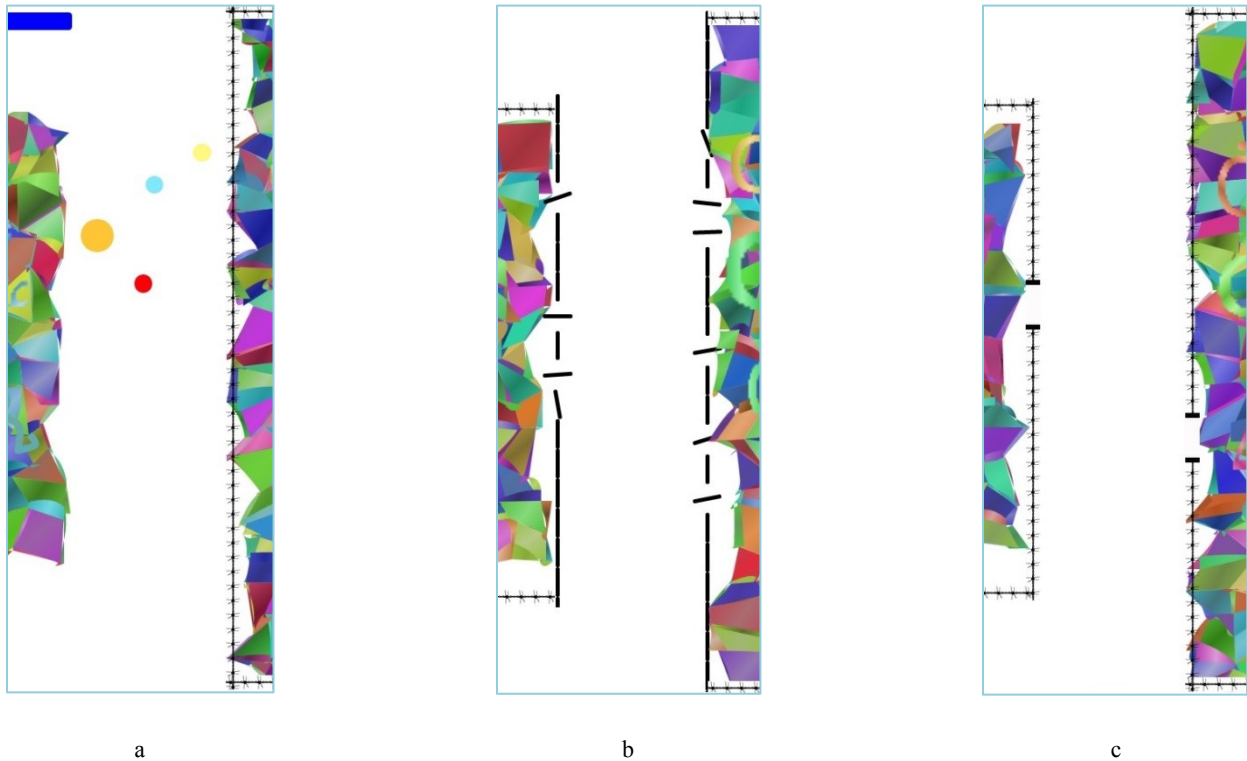
3.2.2 Passing through rotating bars without touching them

In another type of interactive operation in this CAPTCHA, the user has to successfully pass through two columns of rotating bars on the adjacent vertical sides of the test and keyboard. The other three sides of the test and keyboard are closed and the user must pass through the rotating bars in order to complete a match (Figure 36-b). The direction and the rotation speed of each bar are determined randomly, is different for each bar, and changes at any random time. The users have to move the mouse pointer through the dynamic gaps that are generated between the bars as they rotate. Our user study shows that it is easy for users to pass through the bars; users have to appropriately time their movement so as to avoid touching the bars. From a security viewpoint, it is computationally expensive for a robot to identify the pattern of changing movements, predict safe zones for passing through, and time a successful passage. Similar to the previous interaction type, clicking on the test or keyboard makes this interaction visible, releasing the mouse button makes it disappear, and subsequent clicks of the mouse produce new instances of the interaction that vary in their local features, such as the distribution of the bars, their rotational direction and speed.

3.2.3 Passing through moving gates

In this interaction type, all four sides of the test and keyboard rectangles are closed and the only way to exit one and enter the other is through two moving gates, one for the keyboard and

one for the test. Each gate has a random size and moves vertically with a random speed that changes frequently. The gate stops moving at random times for a random duration which allows the user to pass through the gate (Figure 36-c). The size, moving time, and stopping duration of the gates are designed in such a way that human users have no difficulty passing through them. While it is easy for a user to appropriately time a successful passage, it is difficult for a machine to achieve this goal. The machine has to take many screenshots to know where the door stops. That is because everything about the gates is random. Similar to the previous two types, this interaction type appears and disappears by clicking and releasing the mouse button, and subsequent clicks create new instances of the interaction.



a b c
Figure 36: Drag and drop operations in the proposed CAPTCHA.
a) Hitting a moving ball, b) passing through rotating bars, and c) passing through moving gates.

3.3 Security analysis

In this section, we calculate the effect of adding interactive games on the security of our base CAPTCHA. If the net security effect is zero, these modifications will only diversify the risk of the CAPTCHA being broken across two challenges. However, if the net security effect is positive, the differential impact of adding the proposed interactions on security is of interest. For comparison purposes, we will refer to the base CAPTCHA with the interactive matching tasks as the extended CAPTCHA.

In addition to the image processing attack, breaking the extended CAPTCHA requires passing the interactive matching task, i.e. drag and drop operation.

In the extended CAPTCHA, one interaction type is randomly displayed when users click the CAPTCHA. We first calculate the probability of passing each of these interactive tasks by random guessing when these interactions are static. For the moving gates and rotating bars challenges, we use the ratio of the permissible passage gaps over the total length of the walls to determine the successful passage probability. In reality, the gaps are dynamic which further reduces these calculated probabilities. Due to the difficulty of calculating dynamic probabilities given the randomness of the dynamism, we resort to calculating the basic static probabilities:

$$P_{gates} = \left(\frac{\text{the length of the display gate}}{\text{the length of the display barbed wire}} \right) * \left(\frac{\text{the length of the keyboard gate}}{\text{the length of the keyboard barbed wire}} \right) \approx 0.0066$$

$$P_{bars} = \left(\frac{\text{Sum of average gap lengths between bars in display}}{\text{the length of the display vertical side}} \right) * \left(\frac{\text{Sum of average gap lengths between bars in keyboard}}{\text{the length of the keyboard vertical side}} \right) \approx 0.017$$

$$P_{balls} = \left(\frac{\text{Average area of a target ball}}{\text{Area of the movement space}} \right) \approx 0.0045$$

The rotating bars interaction has the highest random guess solving probability because having several rotating bars increase the number of gaps. This interaction type can be modified to include a single rotating bar. However we do our calculations with and without the current implementation of this interaction type.

In order to effectively use the interactive task, there must be a limit on the number of attempts that a person or a machine can have. If a user fails beyond this limit, they will be disqualified and fail the CAPTCHA test. Data from our user study indicate that from all drag and drop attempts, human users never failed in 82.8% of the cases, failed once in 8.6% of the cases, twice in 5.4% of the cases, and three times in 3.2% of the cases. Failing four times in any matching task was not observed in our data. Hence, we allow a user or an algorithm a maximum of four times for each character to fail in the matching task. For a four-character test, this condition allows 16 total matching failures.

The joint probability of solving the image processing and the interactive game challenge by random guess with and without the rotating bars interaction (RBI) in a CAPTCHA for a 4-character test can be calculated as:²¹

$$\begin{aligned}
p_{\text{success-in-both-problems}_{\text{with 4 characters}}} &= \max(P_{\text{imageProcessing for 4 characters}}) \times EV^4(P_{DnD}) \\
&= p_{\text{SURF-palette-semiRandom}_{\text{with 4 characters}}} \times EV^4(P_{DnD}) \\
&= \begin{cases} (4 \times 10^{-5}) \times \left(\frac{1}{3} \times 4 \times (0.0066 + 0.017 + 0.0045)\right)^4 = 7.9 \times 10^{-11} & \text{with RBI} \\ (4 \times 10^{-5}) \times \left(\frac{1}{2} \times 4 \times (0.0066 + 0.0045)\right)^4 = 9.7 \times 10^{-12} & \text{without RBI} \end{cases}
\end{aligned}$$

²¹ The process for calculating $p_{\text{SURF-palette-semiRandom}_{\text{with 4 characters}}}$ is explained in Section 4.4. This is the highest probability of success in a recognition attack using the SURF algorithm assuming that the CAPTCHA image is perfectly segmented.

In the above calculations, P_{DnD} refers to the probability of successfully solving a drag and drop interactive task by random guessing and EV is the expected value operator. The above calculated probabilities show that the addition of the interactive tests reduces the probability of successfully breaking the extended CAPTCHA to a millionth of the success probability for the base CAPTCHA. However, a user might use additional image processing algorithms and spend more resources to solve the interactive tasks. In order to make such attempts less effective, the movement characteristics of the dynamic elements, such as direction, speed, and location change dynamically by design. Even when a machine manages to decipher the characteristics of an element's motion and predict the formation of gaps, these characteristics lose their relevance as these parameters change. Despite these strategies, we make a more stringent comparison by assuming that an intelligent machine is able to process the game and narrow down the location of the gaps to $1/3$ of the sample space. With this assumption, we recalculate the impact of the interactive tasks on the probability of success for the extended CAPTCHA:

$$EV(P_{DnD}) =$$

$$\left\{ \begin{array}{l} \left(\frac{1}{3} \times 4 \times \left(\left(\frac{1}{1/3} \right)^2 \times (0.0066 + 0.017) + \left(\frac{1}{1/3} \right) \times 0.0045 \right) \right)^4 = 8 \times 10^{-3} \text{ with RBI} \\ \left(\frac{1}{2} \times 4 \times \left(\left(\frac{1}{1/3} \right)^2 \times (0.0066) + \left(\frac{1}{1/3} \right) \times 0.0045 \right) \right)^4 = 4 \times 10^{-4} \text{ without RBI} \end{array} \right.$$

And the overall probability of success will be:

$$P_{\text{success-in-both-problems}_{\text{with 4 characters}}}$$

$$= \left\{ \begin{array}{l} (4 \times 10^{-5}) \times (8 \times 10^{-3}) = 3.2 \times 10^{-7} \text{ with RBI} \\ (4 \times 10^{-5}) \times (4 \times 10^{-4}) = 1.6 \times 10^{-8} \text{ without RBI} \end{array} \right.$$

We compare the above probabilities with the probability of solving our original CAPTCHA without the added interaction types with the requirement of solving 6 characters,²² which is:

$$\begin{aligned}
 p_{\text{success-in-one-problem}_{\text{with 6 characters}}} &= p_{\text{SURF-palette-semiRandom}_{\text{with 6 characters}}} \\
 &= (0.444 \times 0.286 + 0.150 \times 0.000)^6 = 4.2 \times 10^{-6}
 \end{aligned}$$

Given the above calculations, adding the interactions and reducing the number of characters required to solve the CAPTCHA does not compromise security. The above calculations entirely ignore the random dynamism in the different elements of the interactive component. Nevertheless, the addition of the interactive tasks still significantly enhances the security of the base CAPTCHA. The high amount of random dynamics employed in the interactive component makes it extremely difficult for an algorithm to emulate the interactive behaviour required for solving the CAPTCHA. The advantages offered by the interactive component in diversifying the risk of the CAPTCHA failure has immeasurable positive implications for security. In addition, employing interactions adds substantial computational cost for an attacker.

In the next section, we conduct a user study to assess the CAPTCHA's usability and also develop a profile of user mouse movements that can be utilized to further enhance CAPTCHA security.

3.4 User study

In this user study, one hundred people of different nationalities, ages, and education levels solved a single CAPTCHA. We used a sample comprised of friends and their families, university students and staff. We asked the users if they use the Internet regularly (expert users) or

²² Similar to the calculations for $p_{\text{SURF-palette-semiRandom}_{\text{with 4 characters}}}$ in Section 4.4.

infrequently (non-expert users). We collected data on the CAPTCHA’s usability and the users’ performance as well as their mouse movements.

3.4.1 Usability study

Table 15 represents the success rate of users in solving the CAPTCHA and Table 16 shows their average, minimum, and maximum solution time in seconds.

Our analysis of the overall accuracy and solution times of the 4-character CAPTCHA with interactions and comparing them with the same metrics for the non-interactive CAPTCHA reported in Section 2.9.2 shows that usability has not significantly changed with the added interactions.

Table 15: User success rates.

	Total CAPTCHAs solved	Percentage solved correctly	Percentage solved incorrectly	Percentage Withdrawal
All three tasks	100	79.0%	21.0%	0%
CAPTCHA with moving balls	33	78.8%	21.2%	0%
CAPTCHA with moving gates	33	81.8%	18.2%	0%
CAPTCHA with rotating bars	34	76.5%	23.5%	0%

Table 16: User solution times in seconds.

	Mean Time	Minimum time	Maximum time	SD time
All three tasks	110	44	187	31.9
CAPTCHA with moving balls	101	44	151	25.1
CAPTCHA with moving gates	117	57	187	37.9
CAPTCHA with rotating bars	113	49	172	30.3

The ANOVA test for time showed no significant difference in solution time across the three types of interactive operations ($F(2,97)=2.29$, $p\text{-value}=0.11$). We also divided users into two groups: expert and non-expert users. Expert users were mostly university students or staff who use the Internet regularly; and non-expert users include middle-aged male and female users who

were familiar with computers, but use it infrequently. Table 17 displays solution time and accuracy for these groups. Although average time seems to be lower for expert users, no significant difference was found in solution times across the two groups ($F(1,98)=1.22$, p -value=0.27). This could be due to the small sample size of the non-expert group.

Table 17: Solution time and accuracy of expert and non-expert users.

	Number of users	Solution time		Solution accuracy
		Mean Time	SD time	
Overall	100	110	31.9	79.0%
Expert	82	108	30.8	81.7%
Non-expert	18	118	36.8	66.7%

We also asked participants in the user study to answer a few questions after they solved the CAPTCHA. These questions and user responses are summarized in Table 18.

Table 18: Users' responses to the survey following the CAPTCHA.

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree	No answer
I felt comfortable in solving this problem.	52%	37%	6%	1%	0%	4%
The problem was hard to solve.	0%	2%	10%	44%	41%	3%
It is difficult to learn to use this system.	0%	1%	4%	37%	56%	2%
The color palette was helpful.	56%	17%	8%	7%	5%	1%

In general, 89% of the participants felt comfortable solving this CAPTCHA. A small percentage of the participants felt that the problem was hard to solve (2%), or difficult to learn (1%). Taken together, our results indicate that security increases while perceptions of task complexity, solution accuracies, and solving times remain unchanged with the addition of the interactive games and requiring users to solve 4 characters instead of 6. The substantial increase in security is desirable where higher security levels are required.

Now, we provide our synthesis of human mouse movements based on the data collected in the previous user study.

3.4.2 Differentiating humans from machines by their mouse movements

A specific goal of this user study was to record and analyze users' mouse movements. The analysis of mouse movements would allow us to find patterns in mouse movements that can help distinguish human users from a robot. Human users' distinct mouse movement patterns can be used to identify machines or at least make attacks harder for a machine that tries to emulate them.

Machines can emulate any type of mouse movement. However, such emulation puts additional burden on an attacking robot which makes the attack more difficult and expensive. Nevertheless, the inherent differences between mouse movements of a human user and those of an unsuspecting machine can be used to identify or flag machines as potentially non-human. In order to find these differences, we studied the biometrics of our user study participants' mouse movements.

Physical (e.g. fingerprints) and behavioral (e.g. keystrokes) biometrics have become popular tools to enhance security in applications such as user authentication and intrusion detection. Behavioral biometrics data can be readily collected using standard input devices, such as mouse and keyboard and can be effectively used to enhance our CAPTCHA's security as well. Mouse dynamics refer to the features of a human user's mouse movements. Collecting mouse dynamics data would allow dynamic and passive user monitoring without their cooperation [142].

Researchers have mostly studied mouse dynamics biometrics in the context of user identification [142-144]. In these applications, features from a person's mouse movements are extracted to create a dynamic signature for that user. According to [145], a mouse action can be classified as either a "movement" or a "silence". A *mouse movement* has features such as type, direction, time, travelled distance, speed, and acceleration of the movement [143]. A *silence* can

be described by its duration. After profiling a user's entire mouse actions using the mentioned parameters, the mouse dynamics signature of a user can be created by extracting features from the mouse movement data. This signature can later be used to identify a specific user with a neural network.

In contrast to the above line of research, our goal is to utilize the similarities and shared characteristics of human users' mouse movements rather than their dissimilarities. To the best of our knowledge, there is little work that attempts to find common properties between human user signatures. Akif Nazar [142] is an exception that studied several of the above factors in an attempt to create a synthetic mouse action generator that can produce mouse movement data similar to a human. Patterns he found in mouse movements of human users include the following:

- There should be at least one action in each direction,
- There should be at least one action for each action type,
- Distances have to be within screen resolution,
- Single movement times cannot be greater than 7 seconds,
- The average speed in each direction has to be in a certain range,
- The average speed for each action type has to be within a certain range.

In this work, we studied additional movement features in addition to some features that are specific to our CAPTCHA. For analysis purposes, we defined specific actions and features for mouse movements. While these features vary from one human user to another, they exhibit characteristics that are common across different users. The empirical distribution of some of these features allows us to identify anomalies and extreme values which could result from automated handling of the mouse. A caveat is that our findings would not be as effective for a machine that knows the distribution of these parameters and emulates a "human-like" mouse movement.

In order to find common mouse movement patterns in our user study, we defined different types of actions.

Definition of actions

Silence: ‘Silence’ is defined as the time spent by the user performing no mouse actions.

Short silence: Any silence that lasts less than 120 milliseconds.

Pause: We define ‘pause’ as a long silence that is approximately one standard deviation longer than the average silence. Hence, we define a pause as a silence that is longer than 500 milliseconds.

Movement: We define a ‘movement’ as a mouse movement between two pauses, a pause and an action (click, drag-start, drop, touching a barrier, etc.); or between two such actions. In order to reduce noise, movements shorter than 100 pixels are not considered a *movement*.

Short movement: A ‘short movement’ is a movement between two pauses with route length shorter than 100 pixels.

Matching movement: A ‘matching movement’ is a mouse movement that starts when a user starts dragging the first item of the match; and finishes when the user drops it on the second item of the match.

Route length: Total length of a movement route.

Direct distance: The direct distance between two points, which can differ from route length.

We extracted a long list of features for mouse actions. However, not all of them could be effectively used to identify users from machines. Hence, we mainly report features that can be used for identifying non-human users. A detailed list of the extracted features and their distributional properties as well as their potential usefulness in identifying machines is provided in the Appendix.

Summary of findings on human mouse movements

Our analysis of human users' mouse movement data reveals the distributional properties and some patterns that can be used to identify machines. Table 19 summarizes the distributional properties of several features that have been introduced in the Appendix. Different criteria can be employed to use these data for identifying non-human users. For example, if the values of a given number of these features for a CAPTCHA solver fall beyond specific thresholds and are considered to be non-representative of human users, the user can be flagged as potentially non-human. In some cases, the value of even one of these variables can readily identify a machine. For example, no human user shows a direct distance to route length ratio equal to 100%. If such perfect movements are observed for a user, it is very likely that the user is a machine, as human mouse movements exhibit imperfection and imprecision.

Table 19: Features extracted from users' mouse movement data.

Behavioral factor	Median	Min	Max	SD	
Number of short movements in a test	14	3	34	6.85	
Overall mouse movement speed	313.57	24	3332	398.04	
Length of pauses (ms)	960	504	4920	906.68	
Number of pauses	23	4	60	12.36	
Number of short silences during movements	2.6	0.6	9.2	1.41	
Time length of short silences during movements (ms)	72.5	35.3	119.3	17.78	
Number of direction changes in a movement	6.5	2	16	2.54	
Direct distance to route length ratio of a movement	56.47	7.83	99.78	27.91	
Average speed per movement direction	R	513.58	50.23	1447.8	414.88
	RU	459.99	36.03	1475.12	398.10
	U	323.34	112.99	1749.87	443.51
	LU	370.21	70.81	2862.96	825.78
	L	409.81	40.7	2915.17	712.83
	LD	250.42	56.49	942.53	218.57
	D	318.47	92.81	1020.97	239.86
RD	293.63	79.86	1725.11	442.57	

Our results provide the following specific findings which can be used to flag unsuspecting machines:

- The lack of short movements or consistently exhibiting less than 5 short movements (i.e. one SD from the mean),
- Consistently exhibiting high mouse movement speeds,
- Similar directional speeds in the 8 specified directions,
- Exhibiting few/no pauses,
- Exhibiting no long pauses,
- Exhibiting no short silences or an extremely low number of short silences, e.g., short silences below 35 milliseconds,
- Exhibiting a value of 100% or consistently exhibiting high values for the “Direct distance to route length ratio”, e.g. values above 97.5,
- Lack of direction changes in a movement.

The following general insights were also derived from the analysis of the data:

- Human mouse movements are not very precise and straight. Human users make many unnecessary movements and direction changes while moving the mouse.
- Users have silence periods during their mouse movements. These silence periods include long silences (we call them pauses), and short silences.
- Users’ movement speed is range-bound.
- Human users do not exhibit jumps in their mouse movements.

The above findings provide some insights that can be used to identify unsuspecting machines. However, a smart computer program is still able to emulate the above mouse dynamic patterns. Therefore, the least advantage of using mouse actions to flag non-human users is that adopting such a technique increases the computational complexity for an attacking program. In addition, emulating human-like behavior takes time which eliminates a machine’s ability to conduct tasks at incredibly high speeds. A machine that is forced to exhibit time-consuming imperfect human behaviour is denied its main advantage in exploiting resources, i.e. speed.

3.5 Conclusion

In this section, we explained the conversion of our proposed CAPTCHA from a one-problem to a multi-problem test. This strategy improves the security by diversifying the risk of CAPTCHA being broken and forces an attacking program to have the capability to solve an interactive problem with a totally different nature. Our security analysis shows that addition of the proposed interactions as well as reducing the number of characters required to match does not compromise security. Furthermore, a user study shows that the net effect of the proposed modification and a reduction in the number of matches required for solution does not compromise the usability of the proposed CAPTCHA as well. The second problem added to this CAPTCHA is a simple interactive game that is intuitive and comfortable for human users²³ [108], yet hard for computer programs to solve.

We also used our user study to extract distributional characteristics of several features of human mouse movements. These characteristics can be used to flag potential non-human solvers when they exhibit extreme characteristics on any or all of the described features. Such considerations can help flag less-smart algorithms as non-human or further increase the computational cost for smarter algorithms which attempt to emulate human behaviour. In addition, employing a mouse movement analysis mechanism removes machines' main advantage over human users which is speed.

²³ Many users mentioned in their comments that they would prefer such game-like CAPTCHAs to many existing text-based CAPTCHAs that include distorted text.

Chapter 4 Security Analysis of the Proposed CAPTCHA

4.1 Introduction

In Chapter 3, we analyzed the implications of adding a second challenge to our proposed CAPTCHA for security and usability. However, resistance to image processing attacks is an essential requirement of a CAPTCHA system. Hence, in this chapter, we explore our base CAPTCHA's security without the second challenge against image processing attacks. In the first section, we introduce different preprocessing, segmentation, and recognition attacks on existing CAPTCHAs (in general and as a part of attacks against other CAPTCHAs) and discuss their chance of success in attacks against the proposed CAPTCHA. Subsequently, security considerations of the proposed CAPTCHA are described; and we conclude this chapter with testing the proposed CAPTCHA's robustness and resistance against two segmentation and recognition attacks.

4.2 Attacks on existing CAPTCHA schemes

Most attacks on CAPTCHAs are composed of three steps: pre-processing, segmentation, and recognition. These steps are discussed in more detail in Section 1.3.1 and are summarized in Table 20, Table 21, and Table 22, respectively. Table 23 summarizes specific attacks on existing CAPTCHAs that have employed different combinations of the pre-processing, segmentation, and recognition attacks. In addition, in the rightmost column of all these tables, we briefly explain the reasons why our proposed CAPTCHA might be resistant against each attack.

Table 20: Pre-processing attacks on existing CAPTCHAs.

Pre-processing attack	Description/Details/Examples	Examples of vulnerable CAPTCHAs	Why our CAPTCHA is resistant against this attack
Noise removal based on the color difference between the noise and targets.	Using foreground/background color difference to remove noise [70].	CAPTCHAservice.org [70]	In the proposed CAPTCHA, noise components have the same color as characters.
Noise removal based on the difference between the size, shape, or location of the noise and those of the targets.	Identifying connected components as noise based on their different size (e.g. with a smaller pixel count), shape (line-shaped), and location (e.g. closer to boundaries) [69, 75].	MSN CAPTCHA [69]	In the proposed CAPTCHA, noise does not have one specific shape. There are also noise elements with features similar to target objects.
	Using morphological image processing [71-73] (e.g. erosion, dilation, opening, closing) to remove noise.	Yahoo! version2 [72]	
	Using vertical and horizontal histograms [79] to remove arcs; knowing that they have specific size or location.	MSN CAPTCHA [79]	
	Using neural networks to distinguish noise with a specific shape [74].	reCAPTCHA [10]	
Noise removal based on different moving patterns of noise and targets in animated CAPTCHAs.	E.g. using the difference between the length of display period of noise components and that of target objects [75] to remove noise.	HelloCAPTCHA [75]	This technique does not apply to our CAPTCHA as noise and target elements are static.

Table 21: Segmentation attacks on existing CAPTCHAs.

Segmentation attack	Description/Details/Examples	Examples of vulnerable CAPTCHAs	Why our CAPTCHA is resistant against this attack
Segmentation based on the location of target objects (knowing that the direction is horizontal).	Color filling segmentation [69]: Using flood-filling of connected components in the image by different colors to detect every connected component in the image.	Microsoft CAPTCHA [69]	In the proposed CAPTCHA, there are many connected components, finding all connected components cannot help.
	Simple segmentation [69]: Dividing the image into parts of the same width to segment the characters, knowing that characters are arranged horizontally and have the same width.	Microsoft CAPTCHA [69]	In the proposed CAPTCHA, the location of the characters is completely random.
	Vertical segmentation [69, 73, 77]: Using a vertical histogram to segment the objects.	Microsoft CAPTCHA [69]	In the proposed CAPTCHA, the characters are not located horizontally.
	Projection [78] Using horizontal projection to detect the first and last character.	Yahoo! [78]	In the proposed CAPTCHA, the large input space makes this attack ineffective as the projection of many characters might become similar.
	Vertical slicing [70]: Traversing pixels from top to bottom and from left to right to find vertical slicing lines in CAPTCHAs whose objects do not overlap horizontally.	CAPTCHAservice.org[70]	In our CAPTCHA, characters are not arranged horizontally; they are distributed randomly in the area. In addition, our use of multiple colors makes this CAPTCHA ineffective.
	Snake segmentation [70]: Using lines to separate objects of a test that may overlap. The lines are programmed to move such as snakes trying not to collide with the objects.	CAPTCHAservice.org[70]	This method is not applicable to our CAPTCHA because of the use of colors and background noise that has the same color as the characters.
	Middle-axis point separation [79]: Drawing segmentation lines between	Yahoo! [79]	In the proposed CAPTCHA, the characters are not arranged horizontally.

	objects by connecting points that are horizontally located in the center of two disconnected black foreground pixels.		
	Drawing splitter lines parallel to the angle of the test image (when test image is skewed) [80].	Teabag 3D [80]	The proposed CAPTCHA does not contain skewed characters.
Segmentation based on the features of target objects.	Loop detection [78]: Identifying the location of characters with the help of the relative positioning of detected loops (e.g. in '8' or 'p').	Yahoo! [78]	In the proposed CAPTCHA, the large input space makes this attack ineffective.
	Edge detection [81]: Detecting boundaries (sharp intensity changes in the image) of objects and segment them.	Imagination [35]	In the proposed CAPTCHA, the presence of a complex background inhibits the effectiveness of an edge detection algorithm because edge detection algorithm detects the edges of background noise as well as the edges of target objects. Moreover, because of the existence of color gradients, sometimes there is no boundary between a character and its surrounding background.
	Interest points density evaluation [82]: Using the difference between the density of interest points of (detected by SIFT) noise components and that of targets to segment them.	NuCAPTCHA [82]	The proposed CAPTCHA includes noise components that have features similar to characters. Hence, the density of interest points that the SIFT algorithm finds in the characters is not always more than those in noise components.
Segmentation based on color information.	Extracting the characters by their colors [75]: Segmenting characters by their colors if they have distinct colors.	HelloCAPTCHA A [75]	In the proposed CAPTCHA, each character has a distinct color from other characters; however, this color also exists in the background.
	Thresholding [83]: Adjusting threshold values to segment different colors in the test image or to segment foreground from the background.	Gimpy-r [83]	Color gradients existing in the proposed CAPTCHA would split a character or merge background noise and characters if this algorithm were used.
Segmentation based on "motion" information.	Using motion tracking (optical flow) to segment objects [82]: Tracking frames of an animation and finding groups of points that move together to find target objects.	NuCAPTCHA [82]	In our proposed CAPTCHA, characters do not move.
	Using pixel delay map to extract the targets [75, 84]: Using the difference between length of motion/stop period of target objects and that of noise to segment them.	Animierte CAPTCHA [84]	In our proposed CAPTCHA, characters do not move.
	Using a catching line to extract the characters [75]: Using a horizontal line located below the upper image boundary to catch the whole image of jumping characters.	AmourAngels [84]	In our proposed CAPTCHA, characters do not move.
	Extracting the characters by frame selection [75]: Selecting best frames of an animation, where characters do not overlap, for segmentation.	HelloCAPTCHA A [75]	In our proposed CAPTCHA, characters do not move.

Table 22: Recognition attacks on existing CAPTCHAs.

Recognition attack	Description/Details/Examples	Examples of vulnerable CAPTCHAs	Why our CAPTCHA is resistant against this attack
Machine learning object recognition methods.	Using OCR [75] for character recognition.	HelloCAPTCHA [75]	There is no effective OCR for the whole Unicode space so far. For example, there is still a need for more effective OCRs for Indian scripts [146, 147], Arabic [148] and Bangla [149].
	Using classifiers [73, 81, 86] to recognize objects based on their features.	ArtiFacial [81]	The existence of homoglyphs, character parts as background noise, stretching one of the characters of a matching pair; and, having semi-similar curved noise components confuses feature-based algorithms.
	Using Context shape matching to recognize characters [87]: Describing each object by a set of context shape vectors; and matching objects by comparing context shape vectors of an object with those of known templates.	EZ-Gimpy [87]	
Pixel-count attack [70].	Pixel-count attack [70]: Exploiting the distinction between pixel counts of different characters to recognize them.	CAPTCHAservice.org [70]	The input space of the proposed CAPTCHA is Unicode with a large number of characters, many of which can have the same pixel counts. Moreover, in the proposed CAPTCHA the size of the characters vary randomly from one test to another which changes their pixel count.
Dictionary attack [70].	Dictionary attack [70]: Knowing that a CAPTCHA is based on a specific dictionary, e.g. English words, Using that dictionary to break the CAPTCHA.	CAPTCHAservice.org [70]	The proposed CAPTCHA does not contain words.
Database attack.	Database attacks [34]: Gaining access to the whole database of a CAPTCHA by solving enough challenges.	EZ-Gimpy [88]	In the proposed CAPTCHA system, the number of challenges is not limited. The tests are created from a random combination of Unicode characters.

Table 23: Attacks on existing CAPTCHAs.

Attacked CAPTCHA	Attack details	Why this attack fails on the proposed CAPTCHA
EZ-Gimpy [87]	<ul style="list-style-type: none"> Context shape matching to recognize each character and estimate its location in the image, Extracting the possible words based on recognized letters and their location, Choosing the most likely word by evaluating a matching score for each of these words. 	- This attack is based on knowing that the CAPTCHA contains English words; but the proposed CAPTCHA does not contain words.
Gimpy [87]	<ul style="list-style-type: none"> Holistic word recognition using context shape matching. 	- The proposed CAPTCHA does not contain English words.
Gimpy, Yahoo!, MailBlocks, etc. [72]	<ul style="list-style-type: none"> Morphological operations to remove background grid and noise, Segmenting connected components. 	- The proposed CAPTCHA includes noise with the same features as the target objects.
Gimpy-r [88]	<ul style="list-style-type: none"> Background removal (based on the difference between the color of foreground and background), Word template matching (by collecting the whole set of input images). 	<ul style="list-style-type: none"> It is not possible to collect all of the images of the proposed CAPTCHA, Characters are positioned randomly, Background color is not different from foreground color.
Microsoft CAPTCHA [69]	<ul style="list-style-type: none"> Vertical segmentation and color filling segmentation, Arc removal, Locating connected characters (after segmentation, if there still exists any chunk with more than one character, use simple segmentation to segment the chunk). 	<ul style="list-style-type: none"> The characters are positioned randomly, There are many character-shape and character-size noise components in the image which are distributed all over the image.

CAPTCHAservice.org [70]	<ul style="list-style-type: none"> • Vertical slicing segmentation, • Pixel-count + dictionary attack. 	<ul style="list-style-type: none"> - Characters may be collinear (i.e., lie on the same vertical or horizontal line), - Each character does not have a specific pixel count to differentiate it from the other characters.
BotCheck [70] and HumanVerify [70]	<ul style="list-style-type: none"> • Preprocessing (background and noise removal), • Vertical histogram, • Pixel count attack. 	<ul style="list-style-type: none"> - Characters may be collinear (i.e., lie on the same vertical or horizontal line), - Each character does not have a specific pixel count to differentiate it from the other chars.
CAPTCHAservice.org [70]	<ul style="list-style-type: none"> • Snake segmentation, • Pixel-count + Geometric analysis. 	<ul style="list-style-type: none"> - The existence of background noise with the same color as the foreground, - Each character does not have a specific pixel count to differentiate it from the other chars, - The input space is Unicode with a large number of characters with different geometric features.
GeCAPTCHA [71]	<ul style="list-style-type: none"> • morphological image processing, • K-means color segmentation, • Cross-correlation or structural similarity (SSIM) for pattern recognition. 	<ul style="list-style-type: none"> - It will be shown that K-means does not work well on our CAPTCHA because of the color gradients, - This CAPTCHA includes noise components with the same features as the target objects in terms of size, color, etc.
MegaUpload [150]	<ul style="list-style-type: none"> • Segmentation attack: locating black components, locating white components; and, merging shared white components to both of the left and right characters to form complete characters. 	<ul style="list-style-type: none"> - The input space of the proposed CAPTCHA is larger, - The type and color of the noise in the proposed CAPTCHA is different.
CAPTCHAs with line cluttering [79]	<ul style="list-style-type: none"> • Removing arcs using a histogram, • Axis-middle point segmentation. 	<ul style="list-style-type: none"> - Both characters and noise components are located randomly all over the image.
3 rd generation reCAPTCHA [74]	<ul style="list-style-type: none"> • Feature-based classification to remove elliptic noise, • Holistic shape context word recognition. 	<ul style="list-style-type: none"> - The proposed CAPTCHA does not contain elliptic noise, - It does not contain words.
reCAPTCHA [73]	<ul style="list-style-type: none"> • Pre-processing: morphological image processing (after estimating the orientation of the word), • Heuristic segmentation: using morphological analysis of characters to group them to certain categories (characters with circles, thin characters, etc.), detecting circles (big closed areas) and finding “characters with circles”, then using a proposed version of vertical segmentation (called three-color bar character encoding) to segment the characters, • Recognition: SVM (Support Vector Machine) neural network. 	<ul style="list-style-type: none"> - This method is based on the shape of English letters and characters’ horizontal location, which does not apply to our CAPTCHA.
Yahoo! [78]	<ul style="list-style-type: none"> • Using specific features of Yahoo! CAPTCHA (all characters are located at a guide line and have a cosine curve shape) to segment the characters using projection, loop detection, and even cut, • OCR for recognition. 	<ul style="list-style-type: none"> - In the proposed CAPTCHA, the characters are randomly positioned, - Most Unicode characters cannot be recognized by projection or loop detection.
CAPTCHAs with connected or disconnected characters [77]	<ul style="list-style-type: none"> • Using projection (vertical histogram) with a dynamic threshold to segment a CAPTCHA image to a few chunks, • Using snakes to find the connections between characters and remove them. 	<ul style="list-style-type: none"> - The random positioning of characters makes vertical segmentation and snake segmentation methods ineffective.
Teabag [80]	<ul style="list-style-type: none"> • Pre-processing: detecting side surface and front surface of characters (the larger boxes and the black areas of the image respectively) and filling the space between the side and front surface of each character, • Segmentation: vertical projection and drawing splitter 	<ul style="list-style-type: none"> - This attack is highly dependent on the shape of English characters, the horizontal positioning of the characters, and features of this specific CAPTCHA.

	<ul style="list-style-type: none"> lines parallel to the angle of the test image or to the side surface of the characters, • Post-processing: connecting the side and front sides of each character together, re-skewing, resizing, and refining, • Character recognition: OCR. 	
Asirra [86]	<ul style="list-style-type: none"> • Using support vector machine classifier to classify dog and cat images based on their color and texture features. 	<ul style="list-style-type: none"> - Objects in the proposed CAPTCHA need to be segmented before recognition (while in Asirra they are already segmented and only require a recognition attack), - Color and texture features are not useful in the recognition of objects in the proposed CAPTCHA.
Imagination [81]	<ul style="list-style-type: none"> • Edge detection to find the center of just one of the objects in the picture (only one is required), • Random guess to select the name of that object from a short list of names. 	<ul style="list-style-type: none"> - Imagination needs solving just one out of N objects; this is a weakness which does not exist in the proposed CAPTCHA, - In the proposed CAPTCHA, objects do not have distinguishable frames to be detected by edge detection algorithms.
ArtiFacial [81]	<ul style="list-style-type: none"> • A Feature-based machine learning method (to detect a real, complete face in the image which contains six facial corner points). 	<ul style="list-style-type: none"> - Our CAPTCHA does not consist of a specific item with particular features.
NuCAPTCHA [82]	<ul style="list-style-type: none"> • Using bounding box and interest point density evaluation to segment test characters from the textual background noise, • Motion tracking (optical flow) to segment characters. 	<ul style="list-style-type: none"> - The existence of homoglyphs, deforming one of the characters of each matching pair, and similar, curved noise components cause problems for this strategy, - Motion tracking to segment the characters does not help because characters in our CAPTCHA do not move.
HelloCAPTCHA [75]	<ul style="list-style-type: none"> • Preprocessing using the distinction between the color of the foreground and background, • Segmentation by pixel delay map, catching line, color selection, or frame selection, • Character recognition using OCR. 	<ul style="list-style-type: none"> - The characters in the proposed CAPTCHA do not move, - The characters' colors also exist in the background, - No effective OCR recognizes all Unicode characters [146-149].

In the next section we discuss the security considerations that guided the design of our proposed CAPTCHA in detail.

4.3 Security considerations in the proposed CAPTCHA

We employ several security considerations to make the proposed CAPTCHA resistant against different types of attacks. These considerations include:

Recognition attack considerations:

- A large input space of Unicode characters makes pattern recognition algorithms less useful,
- Including character segments as noise in tests, and slight unidirectional stretching of one of the characters of a matching pair to change its aspect ratio, which confuses pattern matching algorithms,
- Including homoglyphs in the keyboard, which confuses pattern matching algorithms,
- Including randomly-created curved noise components in the test and keyboard that confuses pattern matching algorithms.

Segmentation attack considerations:

- Coloring the characters and background noise with color gradients makes it difficult for an attacker to segment the image using a color histogram,

General security considerations:

- Using a substantial amount of randomness in tests (the location and size of the characters, background noise, and interaction components) makes it more computationally complex for a machine to solve the tests.
- Requiring human interactions makes attacks more expensive by forcing machines to emulate human actions.
- Modifying the CAPTCHA's matching task to include a second interactivity problem can help diversify the CAPTCHA's breaking risk across two complex problems.
- Recording and comparing users' mouse movement data against the distributional features of human mouse movement data forces a machine to emulate human-like mouse behaviour in order not to be recognized as an attacker. Such a provision increases the computational complexity for an attacker.

We will further explore these security considerations in the following sections.

4.3.1 Recognition attack considerations

Strategies employed to make this CAPTCHA resistant against recognition attacks include:

Including character segments in the keyboard and slight uni-directional stretching of one of the characters of a matching pair:

For many feature-based algorithms, such as SIFT [151], matching is more difficult if any change happens in the object's internal geometry. These algorithms operate best when the relative positions between the points of the object, which are being matched, do not change from one image to another. Cross-correlation-based algorithms are also vulnerable to changes in relative dimensions or aspect ratio. In order to take advantage of this vulnerability, we stretched one of the characters of each matching pair randomly in the vertical or horizontal direction. This strategy will change the internal geometry of the shape and will confuse pattern matching algorithms. The user study shows that this strategy does not affect the usability of the CAPTCHA. To further make the CAPTCHA resistant against feature-based algorithms, we included one random part of each test character as noise in the keyboard. We assess the effect of these considerations using SURF (Speeded-Up Robust Features) [152], which is a robust local feature detector used in computer vision tasks such as object recognition (see Section 5.3 for a detailed review).

Figure 37 shows the result of applying SURF on a test with no additional security considerations. The probability of correctly matching the character by machine is:

$$\frac{\# \text{ of correct point matches}}{\text{total \# of point matches}} = \frac{10}{70} = 14.3\% , \quad \frac{\# \text{ of correct symbol matches}}{\text{total \# of symbol matches}} = \frac{1}{45} = 2.2\%$$

Figure 38 shows the results when the match is slightly stretched and a half-character noise is included. In Figure 38, the algorithm has found 69 *point* matches and 39 *symbol* matches from which, only one is the correct answer; which means:

$$\frac{\# \text{ of correct point matches}}{\text{total \# of point matches}} = \frac{2}{69} = 2.9\% , \quad \frac{\# \text{ of correct symbol matches}}{\text{total \# of symbol matches}} = \frac{1}{39} = 2.6\%$$

Another point in this figure is that the algorithm has found more matching points between the test character and its segment than it did between the character and its deformed match in the keyboard. Hence, if the algorithm were to choose the best matching symbols based on the number of their matching points, it would select the character segment as a better match – which would be a wrong answer.

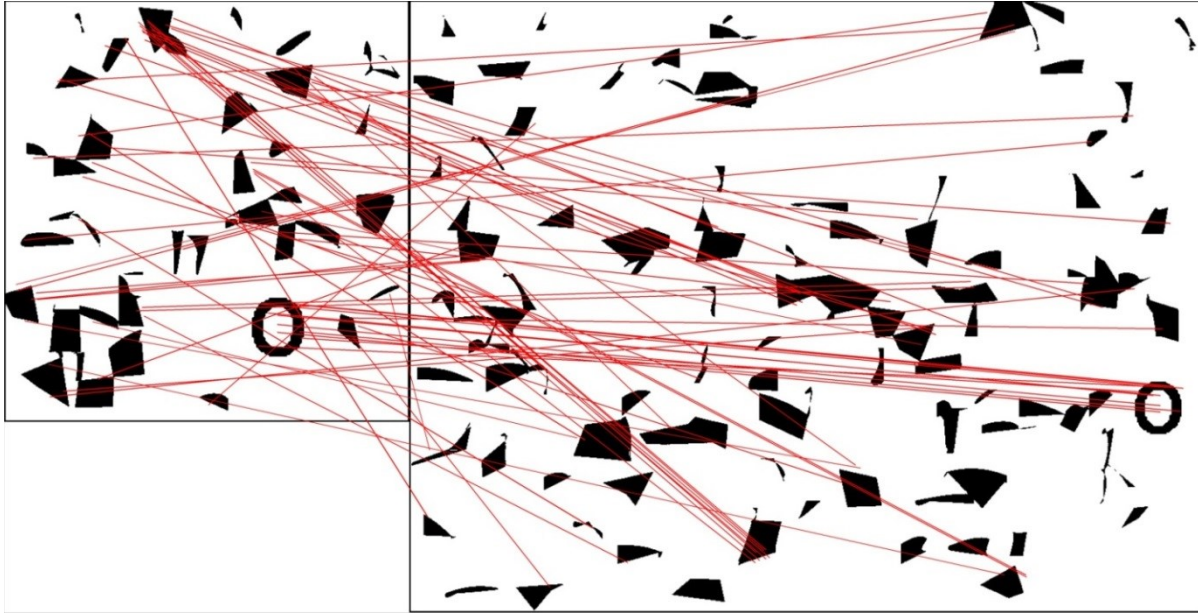


Figure 37: SURF applied to a sample test with no additional noise.

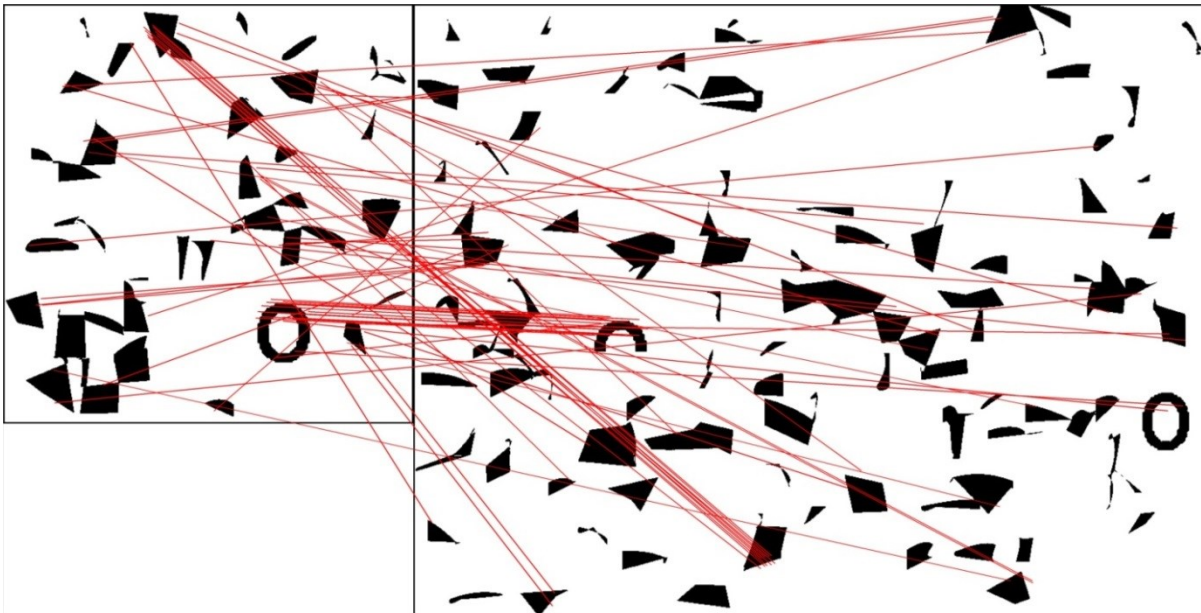


Figure 38: SURF applied to a test with added character segment and a slightly stretched character match. The character is incorrectly matched with its segment.

Including homoglyphs in the keyboard

Including homoglyphs in tests will confuse pattern matching algorithms as well. Figure 39 represents the result of applying SURF on a test that contains homoglyphs. In this figure, there are many false positives and the ratios of the correct matches to the total number of matches are:

$$\frac{\# \text{ of correct point matches}}{\text{total \# of point matches}} = \frac{1}{57} = 1.8\% , \quad \frac{\# \text{ of correct symbol matches}}{\text{total \# of symbol matches}} = \frac{1}{41} = 2.4\%$$

As demonstrated in Figure 39, the algorithm is unable to distinguish between a character's real match and its homoglyphs. Disregarding the strongly matched noise in this example, if the algorithm were to choose the best matching symbol based on the number of their matching points, it would choose the wrong match.

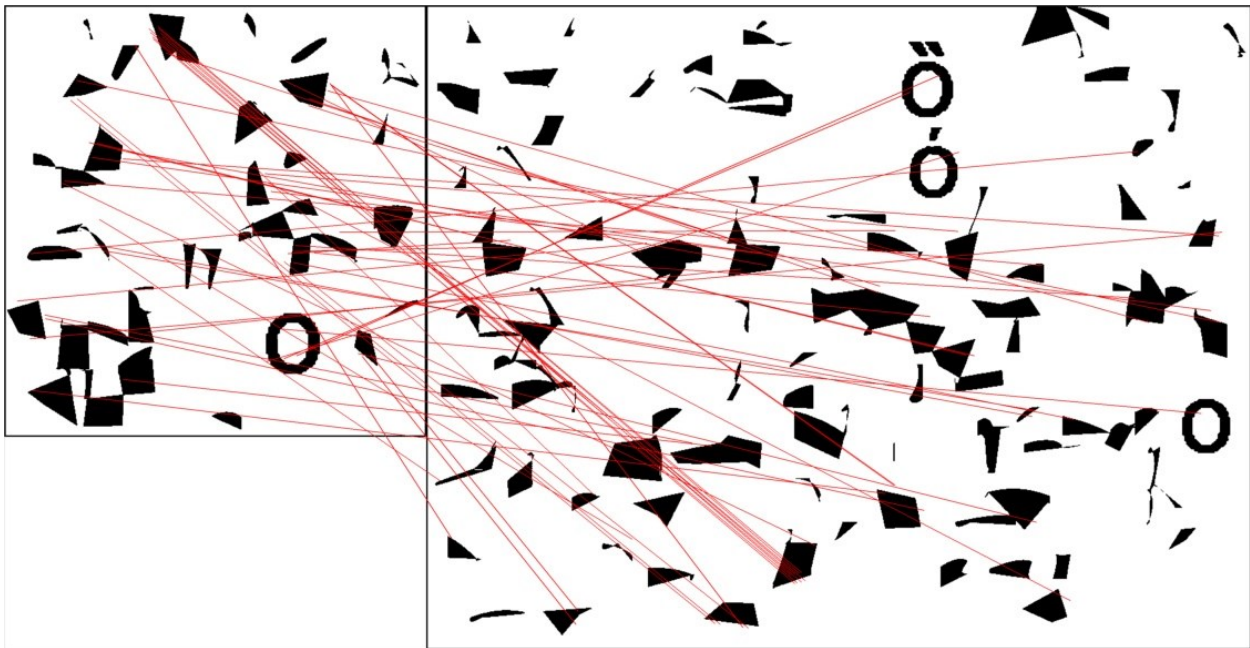


Figure 39: The effect of including homoglyphs in a test on the accuracy of SURF.

Including randomly-created curved noise components

Since most characters contain curves, including curved components in the background will confuse computer programs trying to break this CAPTCHA. Figure 40 represents an example. The advantage of these curved noise components is that they are created algorithmically; we

create a few semi-similar noise components, which vary in only one or two parameters, then include some of them in the test and some in the keyboard. The partial similarity between these noise components confuses pattern matching algorithms. In addition, if no matching characters exist in a segmented monochromatic image as is the case in Figure 40, an attacking algorithm might select a pair of curved noise components as a match.

In this figure, there are 65 point matches and 39 symbol matches. Evidently, all of the matches are wrong which means that the ratio of false point/symbol matches to the total number of point/symbol matches equals to 1.

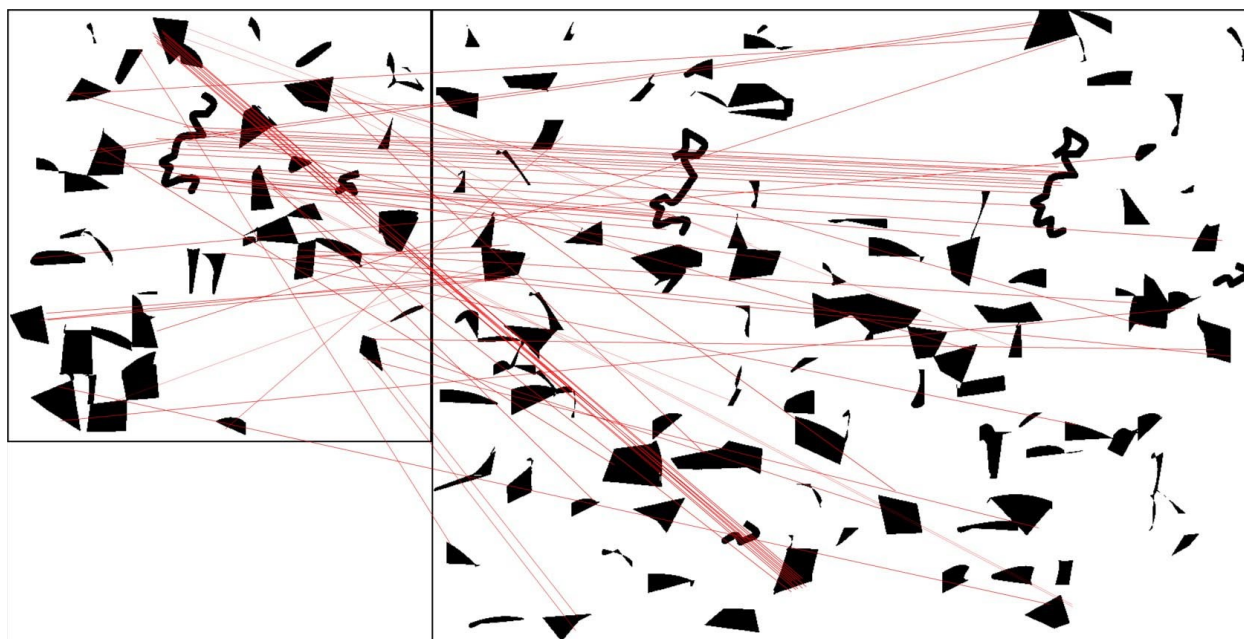


Figure 40: SURF applied to a test containing curved noise components.

As a test of the combined effect of the above considerations, we implemented all these considerations in a test and applied the SURF and cross-correlation algorithms to them. The results of this exercise are displayed in Figure 41 and Figure 42, respectively.

As demonstrated in these figures, both algorithms find many false positive matches when these security considerations are implemented. Figure 41 represents a monochromatic image that

contains a character in the test area and its match in the keyboard. The keyboard also contains noise components including the test character's homoglyphs, and a random piece of the test character as noise. After applying SURF on this image, the ratios of true matches to the total number of matches are:

$$\frac{\# \text{ of correct point matches}}{\text{total \# of point matches}} = \frac{0}{79} = 0\%, \quad \frac{\# \text{ of correct symbol matches}}{\text{total \# of symbol matches}} = \frac{0}{43} = 0\%.$$

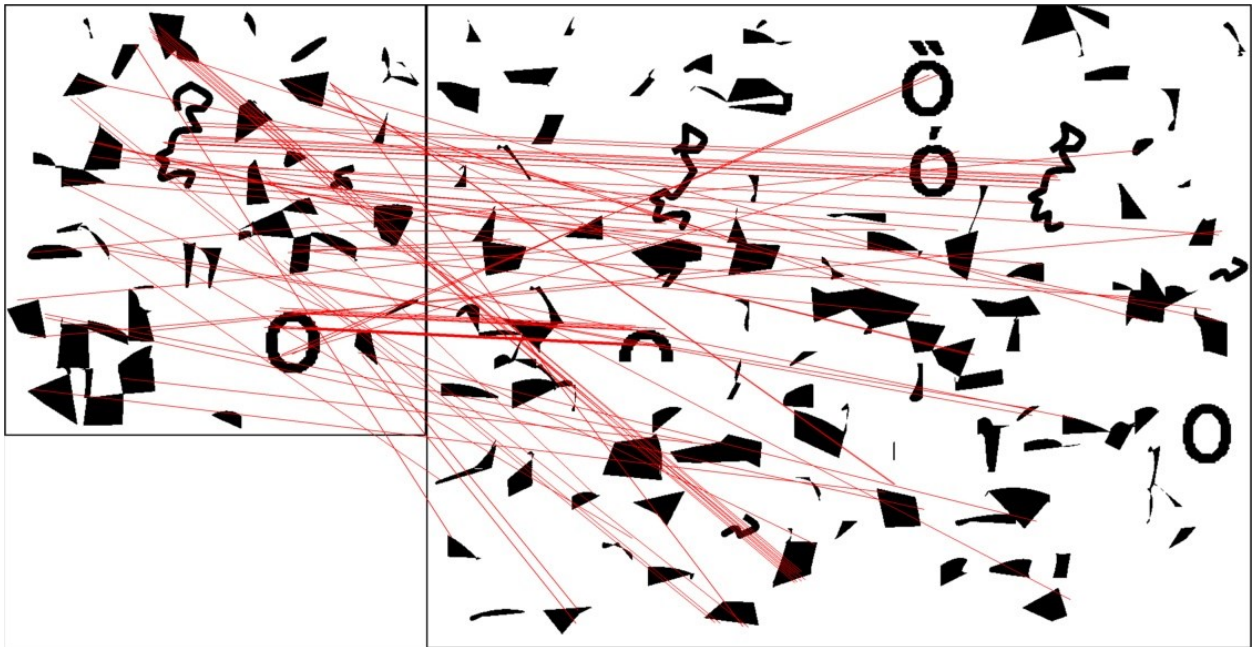


Figure 41: SURF applied to a monochrome image of the proposed CAPTCHA. This image includes homoglyphs, stretched character match, a character segment as noise, and curved semi-similar noise components. SURF failed to find the match.

In Figure 42, a normalized cross correlation algorithm is applied to the same image, and matches with correlation coefficient of more than 0.7 are displayed. In this figure, the algorithm found 13 incorrect matches and it did not find the correct match:

$$\frac{\# \text{ of correct symbol matches}}{\text{total \# of symbol matches}} = \frac{0}{13} = 0\%.$$

The above examples illustrate how our security considerations might affect the performance of recognition attacks. We will explore this issue further in our security analysis.

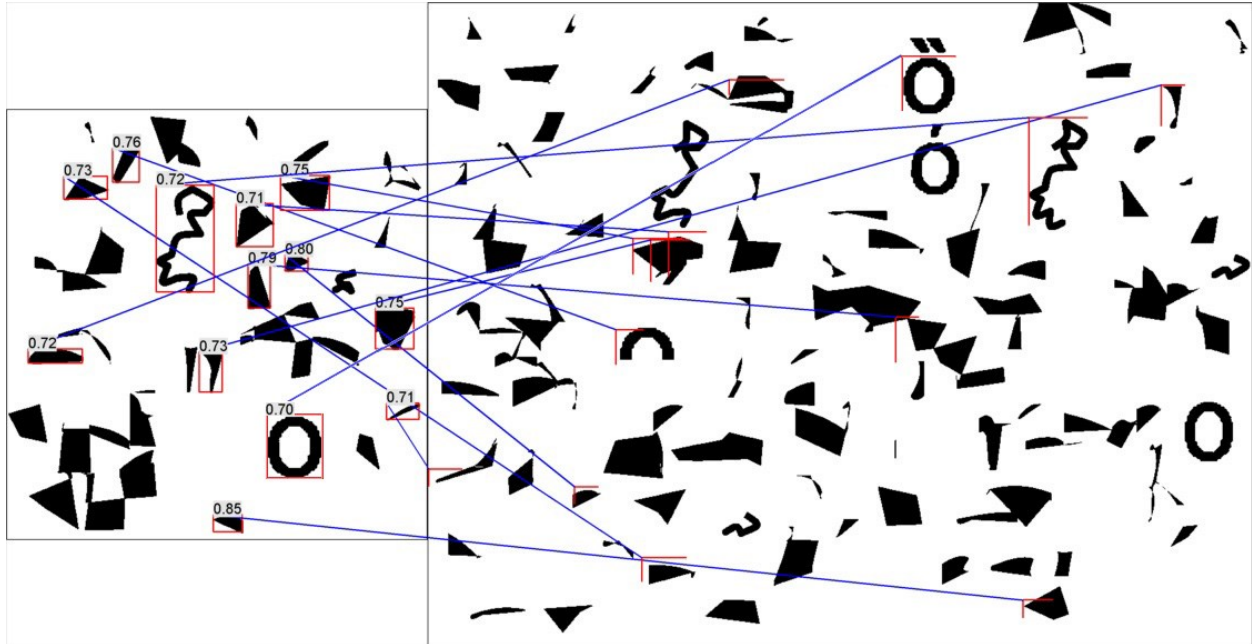


Figure 42: Cross correlation applied to a monochrome image of the proposed CAPTCHA. This image includes homoglyphs, stretched character match, a character segment as noise, and curved semi-similar noise components. The cross-correlation algorithm failed to find the match.

4.3.2 Segmentation attack considerations

Coloring the characters and background noise with color gradients

Coloring characters and background noise with color gradients can cause problems for segmentation algorithms that use a color histogram to segment the colors. For example, when the background of a character shares some shades of the colors used in the character's gradient, these same-color segments will merge in a monochromatic image which leads to the unification of character segments and noise segments. As another example, when a character is comprised of multiple segments, these segments could appear in different monochromatic images after segmentation as a result of the color gradient. These effects are demonstrated in Section 4.4.1.

4.3.3 General security considerations

Using a substantial amount of randomness in tests

If the characters have a fixed size, location, or color, it would be easier for an attacker to segment the image or solve the test by random guessing. The randomness of the character and noise location, size, and color as well as and size, movement direction, and speed of interactive components make the test more computationally complex for an attacker.

Requiring human interaction

In order to solve the CAPTCHA, the machine not only needs to identify the right character matches, but it also needs to interact with the computer via mouse actions to complete each matching task. This task is easier for human users than it is for computer programs.

In the next section, we test the proposed CAPTCHA's resistance against image processing attacks.

4.4 The proposed CAPTCHA's resistance against segmentation and recognition attacks

The simplest possible attack on our proposed CAPTCHA²⁴ is a random guessing attack in which random points on the test and keyboard are clicked. The probability of successfully matching the characters using this attack can be calculated from:

$$P_{\text{success-randomGuessing}} = \frac{m! (n - m)!}{n!} (rs)^{2m} \quad \text{Equation 5}$$

In this equation, m is the number of test characters, n is the number of keyboard characters, each test character takes up on average $r\%$ of the test area, and each keyboard character on average covers a small square whose side length equals $s\%$ of the keyboard's length. For $m=4$, $n=27$, $r=0.23$ and $s=0.11$, the probability of success by a blind attack equals $9.6e-18$.

²⁴ "The proposed CAPTCHA" or "our CAPTCHA" in this section refer to our base CAPTCHA without the second interactive game challenge and without mouse movement tracking.

Evidently, an attacker would develop more intelligent algorithms using image processing techniques to improve its probability of success. An image processing attack on most current CAPTCHAs can be summarized in two steps: 1) segmentation of the image and 2) recognition of each object. In addition, solving our CAPTCHA requires the attacker to emulate human mouse clicks in order to match identified pairs.

In the next subsections, we describe two segmentation and two recognition attacks on our CAPTCHA and assess their ability to solve it.

4.4.1 Segmentation attack

To perform segmentation, an attacker might either use the color palette or an image processing algorithm. Segmenting by image processing algorithms is more complex in the presence of color gradients. Using color gradients makes it likely that a shade of one color becomes very similar to the shade of another color (Figure 43). Hence if a machine uses segmentation by a color histogram, it might consider two different colors as one, or parts of noise components or characters from one color might end up in the monochromatic image of another color. Color gradients do not pose any difficulty for human users and they even facilitate character recognition for them. Since the same gradient is applied to the parts of a character, human users can more easily identify the entire shape of the character by following the parallel color patterns of the gradient on a single character. For example, in Figure 43, the same gradient pattern can be observed on the lower and upper arcs of the circular segment of the character.

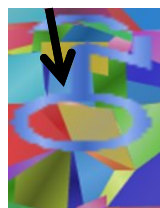


Figure 43: Gradients of a color in background noise become similar to a character's color.

We used a color histogram method and k-means clustering to segment images of this CAPTCHA. An elaboration of these procedures follows.

Segmentation attack 1: Using a color histogram to segment the colors

Histogram-based methods are one of the most efficient image segmentation methods. In this method, a histogram of the colors in the image is produced, and the peaks of the histogram are used to identify the main colors and segment the image.

We captured a screenshot of the CAPTCHA, converted it into an HSB image, and produced the histogram of the H component. Figure 45 represents the histogram of the sample CAPTCHA in Figure 44. The peaks of this histogram identify the colors with the highest presence in this CAPTCHA.

Next, we segmented the image into n colors, with n being the number of peaks of the histogram. Each pixel p with color $C(p)$ in the image is included in the monochromatic image of color i , if $|C(p)-i|$ is smaller than a threshold. Figure 46 shows a monochromatic image resulting from this algorithm.

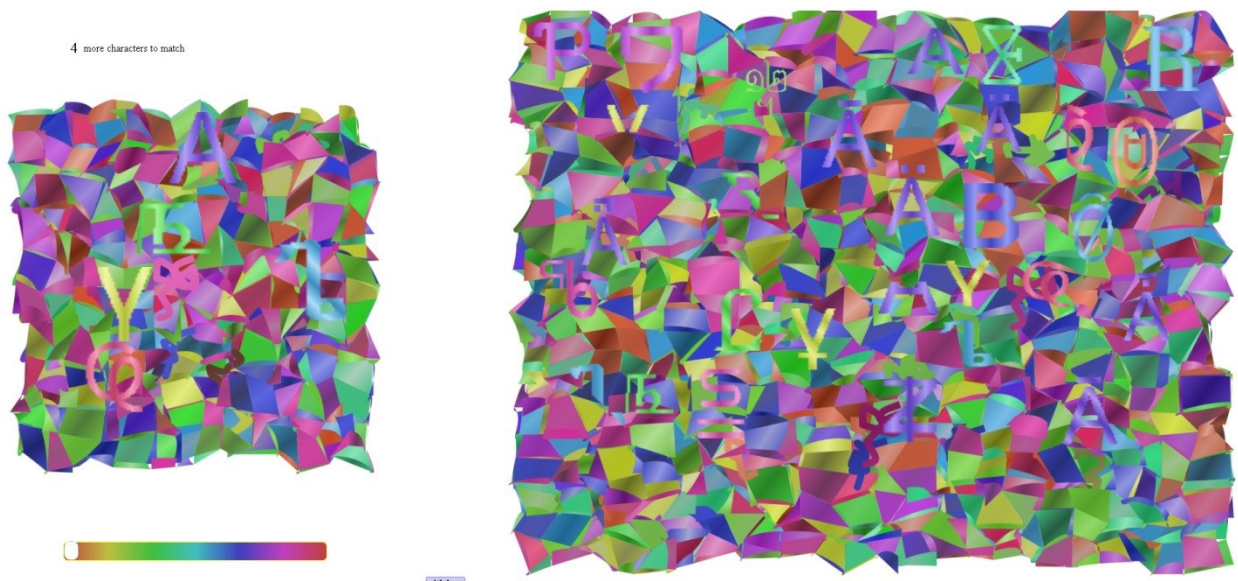


Figure 44: The proposed CAPTCHA.

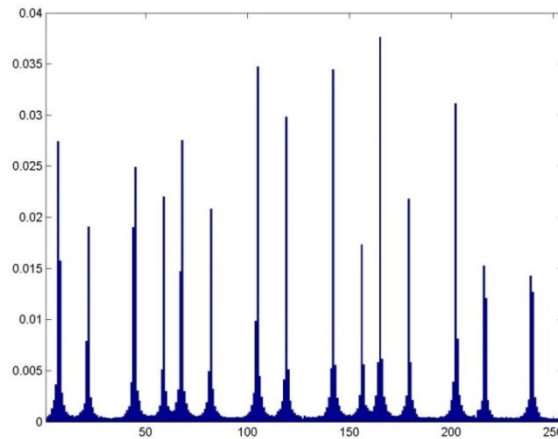


Figure 45: Histogram of the H component of the CAPTCHA image in Figure 44.

Our observations indicate that in many cases, some of the peaks (even the highest peaks) produced by the color histogram do not represent the base colors used to create the color gradients in the CAPTCHA. Instead, these peaks represent random shades of those base colors. Hence, an attacker cannot successfully color-segment the CAPTCHA using the peaks of the histogram since this would lead to the omission of some character segments. Furthermore, increasing the segmentation threshold to include more shades than the mere peak color would increase the amount of noise in the CAPTCHA. In this latter scenario, some parts of characters and their similar-color background noise can merge together in a monochromatic image, making it difficult for the machine to differentiate the character from the noise behind it. An instance of this problem can be seen in Figure 46. Based on the above arguments, color segmentation might not lead to reliable and effective separation of target characters from noise elements.



Figure 46: An example of a monochromatic image produced by color histogram. In this image there is a matching pair which is connected to its surrounding similar-color background noise and was not detected by the recognition algorithm.

Segmentation *attack 2*: Using k-means to segment the image to its colors

We also used k-means clustering to segment images into their colors. Sample results of applying k-means to our CAPTCHA images are shown in Figure 47. As this figure shows, the segmented output created by k-means suffers from the same issues that plagued the color histogram method. Gradients may cause some problems for the attack algorithm including:

- Characters and their match might be segmented into two different monochromatic images,
- Characters might merge with background noise with similar color (Figure 46),
- Characters might be divided into a few pieces of different colors (Figure 47),

The above problems make k-means clustering ineffective in successfully separating characters from background noise by causing the following situations for the attack algorithm:

- The character cannot be matched because the character and its match do not exist in the same image,
- The character is deformed and cannot be matched because it is segmented into multiple pieces with different colors so that the whole character does not exist in any single monochromatic image. Although in Figure 47 the character is segmented into only two

colors and parts of the character appearing in the monochromatic images of each of the two colors still have some features of the original character, in many cases the character is segmented into more than two colors and the resulting character parts might not have any similarity to the original character at all,

- The character is deformed because it is connected to the background noise and it cannot be matched.

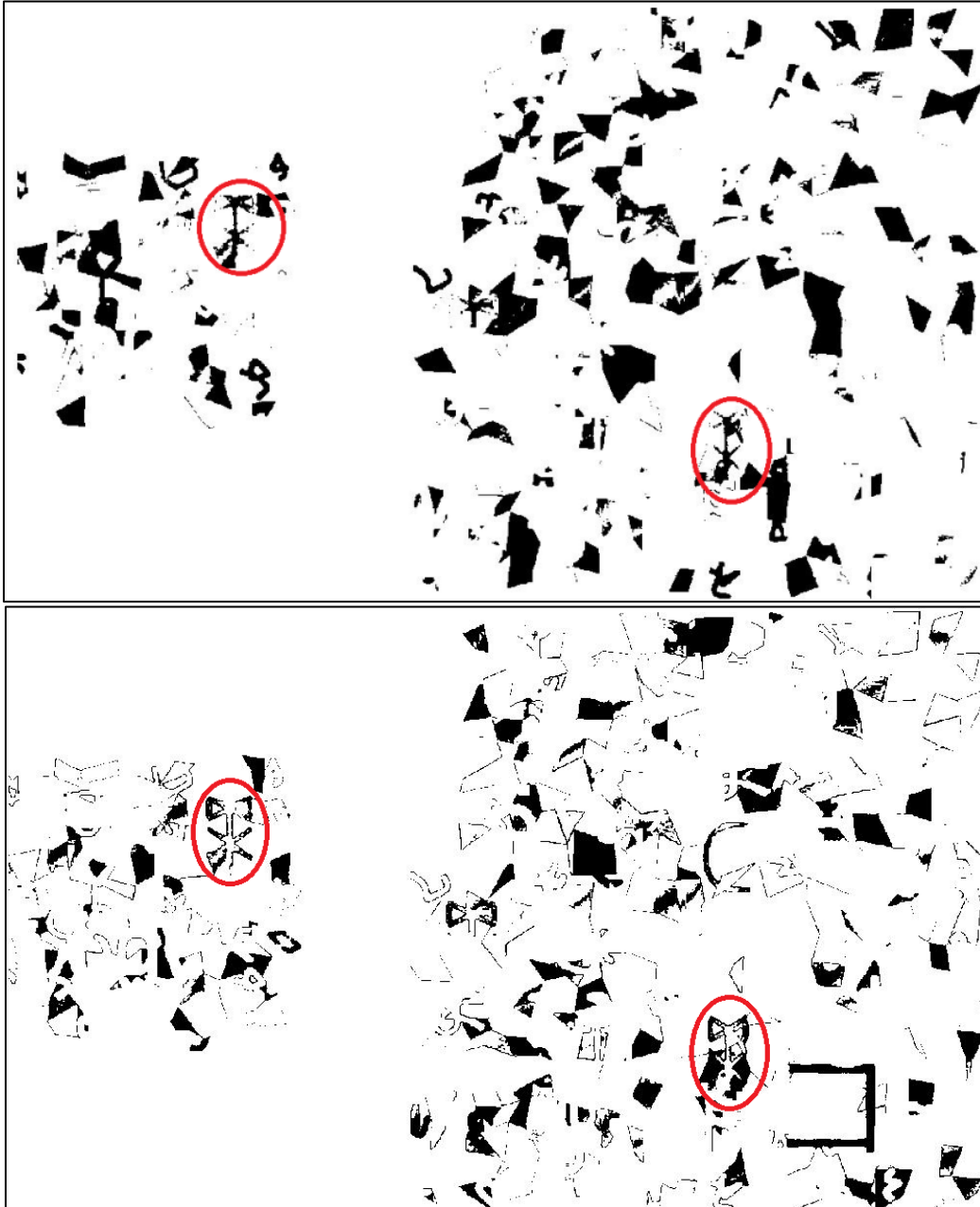


Figure 47: An image segmented by k-means clustering algorithm.

Some parts of the matching character are segmented in monochromatic image of color a ; and some other parts in color b .

Eventually, the attack algorithm will have to solve these rather complex issues in order to be able to carry out an effective segmentation.

4.4.2 Recognition attack

Two of the most fundamental approaches in pattern matching are template-based and feature-based methods. Template-based algorithms compare a replica of an object of interest to all unknown objects in the image field [153]. These algorithms calculate a correlation coefficient between two objects to determine if there is a strong resemblance between them or not. Feature-based approaches are divided into two groups: global image representations and local feature representations. While most global feature representation techniques are based on the comparison of entire images or entire image windows, methods based on local invariant features match local structures between images with partial occlusion, images with viewpoint changes or with deformations [154].

In the past decade, significant progress has been made in the development of local invariant feature detectors and descriptors. One of the most popular local descriptors is SIFT [155]. Following SIFT and motivated by it, SURF [152] was later developed as an efficient alternative to SIFT. SURF extracts interesting points of every object in the image and provides a feature description of the objects. Like SIFT, SURF can recognize objects even if they are under scaled transformations and surrounded by clutter. A more detailed description of these algorithms is provided in Section 5.3.

We tested the proposed CAPTCHA with a feature-based matching approach: SURF; and with a template-based matching algorithm: normalized cross-correlation (an implementation of the Lewis cross-correlation algorithm [156]). We tested these two algorithms on: 1-images

segmented by k-means clustering and color histogram; 2-images perfectly segmented using the color palette. A description of these tests follows.

Recognition attack 1: Matching by SURF

In this attack, we used the SURF feature detector and descriptor to match characters in monochromatic images.

When matching using the SURF algorithm, we consider strategies to reduce the number of false positives. The SURF algorithm can match rotated objects. However, since we do not rotate the characters in our tests, we disable the algorithm's ability to identify rotated objects which reduces the probability of finding false matches. Second, according to Lowe's finding [157], best matching pairs of points (p_1, p_2) in an image are the ones for which the ratio of the distance between point p_1 and its most similar point (p_2) to the distance between p_1 and its second-most-similar point (p_3) is less than a threshold (Equation 6). In other words, in order to achieve reliable matching, the similarity of a point to its most similar match must be significantly higher than its similarity to the second most similar match. Hence, we only accept those matching points that follow the inequality in Equation 6.

$$r = \frac{\text{distance of the descriptor of a point from that of its nearest neighbor}}{\text{distance of the descriptor of a point from that of its second nearest neighbor}} < 0.6 \quad \text{Equation 6}$$

Performance of SURF when applied to images segmented by the color palette

We applied SURF to the color-palette-segmented monochromatic images of 100 CAPTCHAs. Each CAPTCHA includes 10-15 monochromatic images depending on the random number of colors used to create it. Figure 48 shows a few examples of these monochromatic images. In this figure, images (a-c) contain matching characters while image (d) does not. As it can be seen in these images, there are always false positives in the results. In all cases, SURF

detects matches between background noise components. In Figure 48-a, the algorithm is confused between homoglyphs of the test character and in Figure 48-b the confusion is between the character and a piece of the character that is added as noise. In Figure 48-c, the character pair is matched correctly. In Figure 48-d, there is no matching character pair; however, the algorithm found a few matches between background components.

To measure SURF's ability to find the right matches in each image, we assume that the algorithm is capable of choosing the two objects with the highest number of matching points as "the right answer". We also assume that the attacking algorithm is able to group all parts of a multi-part character as one object, which is not an easy task because of the similarity between noise components and some character parts.

Each CAPTCHA has on average 12 monochromatic images (based on the discussion in Section 2.8). Of these, five images contain a matching character pair and seven images are without a matching pair. Of those images containing a matching character pair, on average two images contain homoglyphs and three are without homoglyphs. In images that contain homoglyphs, SURF was not able to correctly detect any matches. In such cases, the algorithm always matched the display character with one of its homoglyphs or character parts included in the noise instead of the real match in the keyboard. On the other hand, characters without homoglyphs were matched correctly in 28.6% of the cases. This means in 28.6% of monochromatic images without homoglyphs, two objects with the largest number of matching points were the matching character pair. Table 24 presents the results of applying SURF on monochromatic images of the 100 tested CAPTCHAs.

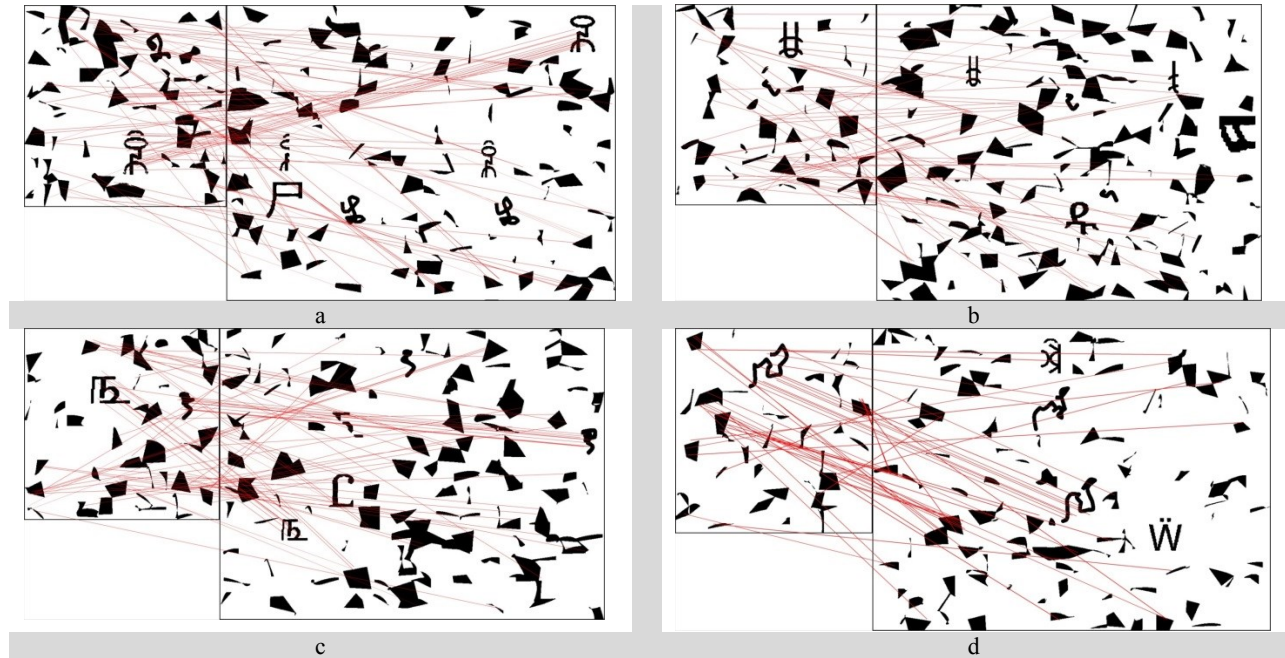


Figure 48: The results of applying SURF on four monochrome images.
 In a-e there is a matching pair in the image; while f is just noise.

Table 24: The results of applying SURF on monochromatic images of 100 CAPTCHAs.

	Monochromatic images with a matching pair without homoglyphs in the keyboard	Monochromatic images with a matching pair with homoglyphs in the keyboard	Monochromatic images without a matching pair
Percentage of each image type	27.8%	11.1%	61.1%
Percentage of correct matches in each type*	28.6%	0%	0%
Mean number of false positive point matches	22	23	20
Mean number of false positive symbol matches	10	11	19
The probability of detecting any matching point between the character and its real match	88.6%	42.3%	0%

* A match is considered to be correct when the matched character pair has the largest number of matching points.

Based on these results, if the attacker randomly selects four monochrome images out of the (on average) 12 images and selects *the best match* (a pair of symbols with largest number of matching points) in each image as ‘the right answer’, the probability of correctly matching all four characters can be calculated as follows:

$$p_{SURF-palette-semiRandom} = (0.278 \times 0.286 + 0.111 \times 0.000 + 0.611 \times 0.000)^4 = 4 \times 10^{-5}$$

Table 24 also shows that, on average, 20 false positive point matches and 11 false positive symbol matches are found in each image. If the attacker’s algorithm does not choose the matches based on the number of the connected points, it would have to randomly select one of the suboptimal matches as ‘the answer’. This is not an unlikely scenario as our study shows that in many cases, the ‘right match’ is not ‘the pair of symbols with the largest number of matching points’. If this scenario for choosing the right answer is chosen, the probability of the correctness of the answer would be:

$$p_{SURF-palette-Random-points} = \left(0.278 \times 0.886 \times \frac{1}{22} + 0.111 \times 0.423 \times \frac{1}{23}\right)^4 = 3 \times 10^{-8}$$

$$p_{SURF-palette-Random-symbols} = \left(0.278 \times 0.886 \times \frac{1}{10} + 0.111 \times 0.423 \times \frac{1}{11}\right)^4 = 7 \times 10^{-7}$$

In a more intelligent scenario, the algorithm first applies SURF on all images taken using the color palette. In the next step, the algorithm sorts the images based on the maximum number of matching points between two objects in the image; then it picks the first four images (since each CAPTCHA includes four pairs to match). Table 25 shows the probability of correctly matching 1, 2, 3 or 4 characters in 100 tests by this algorithm. In 71.3% of the tests, only one correct match was found by the attacking algorithm. In 12.6% of the cases, 2 out of four selected matches were correct; the algorithm was not able to find more than two correct matches in a test. Hence, the probability of the success of this attack, defined as correctly matching all four characters, was 0. As this result shows, this seemingly more intelligent match determination technique would fail to break the proposed CAPTCHA.

Table 25: The probability of 1, 2, 3 or 4 correct matches in a test (for 100 tests).

	1 correct match	2 correct matches	3 correct matches	4 correct matches
The probability of finding i ($i:1-4$) correct match in a test by a more intelligent attacker	71.3%	12.6%	0%	0%

Performance of SURF when applied to images segmented by color histogram or k-means:

Figure 49 represents the result of applying SURF on monochromatic images segmented by color histogram or k-means clustering. Table 26 summarizes the results of applying SURF on the images of 100 tests segmented by k-means clustering. The results of k-means were better than color histogram from an attacker's viewpoint. Hence, we mainly report the k-means clustering results. The major difference between these images and the images taken by color palette is that in these images, in almost all (88.2%) of the images, characters are connected to the background noise after segmentation. After applying SURF on these images, no strong match is usually found. The maximum number of matching points between the two most similar objects in a monochromatic image was 3 (in 3.7% of cases), 2 (in 8.6% of cases), 1 (in 49.4% of cases) and 0 (in 38.3% cases). Because of the lack of strong matches (with many matching points), the attacking algorithm might select *randomly* between objects matched by SURF.

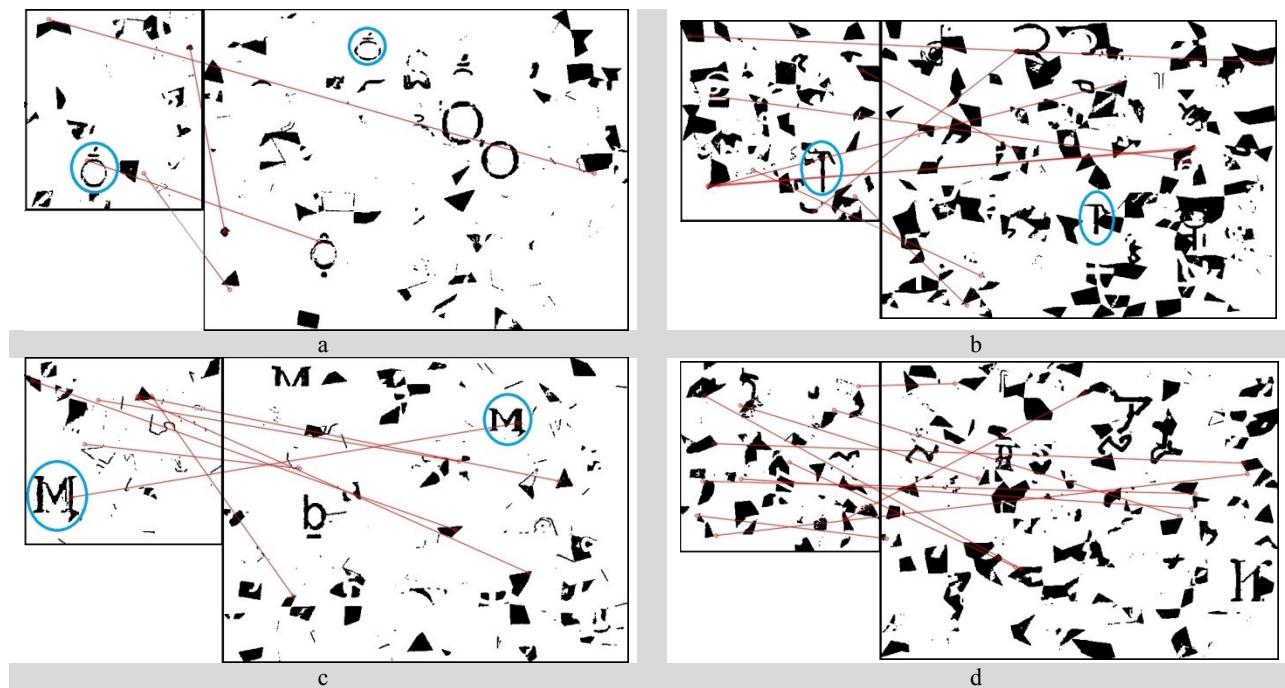


Figure 49: The result of applying SURF on images segmented by K-means segmentation or color histogram.

Table 26: Results of applying SURF on images segmented by k-means.

	Monochromatic images without homoglyphs in the keyboard	Monochromatic images with homoglyphs in the keyboard	Monochromatic images without a matching pair
Percentage of each image type	32.1%	9.9%	58%
Percentage of correct matches in each type	13.5%	0%	0%
The probability of detecting any matching point between the character and its real match	19.2%	0%	0%

Based on the information in Table 26, the probability of success for this attack is:

$$p_{SURF-Kmeans} = (0.321 \times 0.135 + 0.099 \times 0.000 + 0.58 \times 0.000)^4 = 3.5 \times 10^{-6}$$

With a similar calculation, using a color histogram to segment the colors, leads to a probability of success equal to:

$$p_{SURF-colorHistogram} = (0.289 \times 0.021 + 0.108 \times 0.000 + 0.603 \times 0.000)^4 = 1.3 \times 10^{-9}$$

Recognition attack 2: Matching by normalized cross-correlation

In this attack, we detect all connected components in the test and then use a normalized cross-correlation algorithm to find the best match for each connected component of the test in the keyboard and record the correlation coefficient. Our algorithm calculates cross-correlation in the spatial and the frequency domains. Subsequently, the algorithm selects the connected components with the highest correlation coefficient as matching pairs.

Performance of the normalized cross-correlation algorithm when applied to images segmented by color palette

In most cases, the best matches with the cross-correlation algorithm were small noise components. An attacker may try to remove the quadrilateral or small noise components. However, such a procedure would also omit many parts of the characters. Many characters in Unicode are composed of two or more parts. Figure 50 shows a histogram of the number of

character parts in the selected character set of the proposed CAPTCHA. Based on this histogram, approximately 2850 out of 6000 characters (47.5%) have two parts or more. Due to the existence of a large number of multi-segment characters in the Unicode, removing segments of the characters (e.g. by using a noise reduction algorithm) would exacerbate character recognition for an attacker.

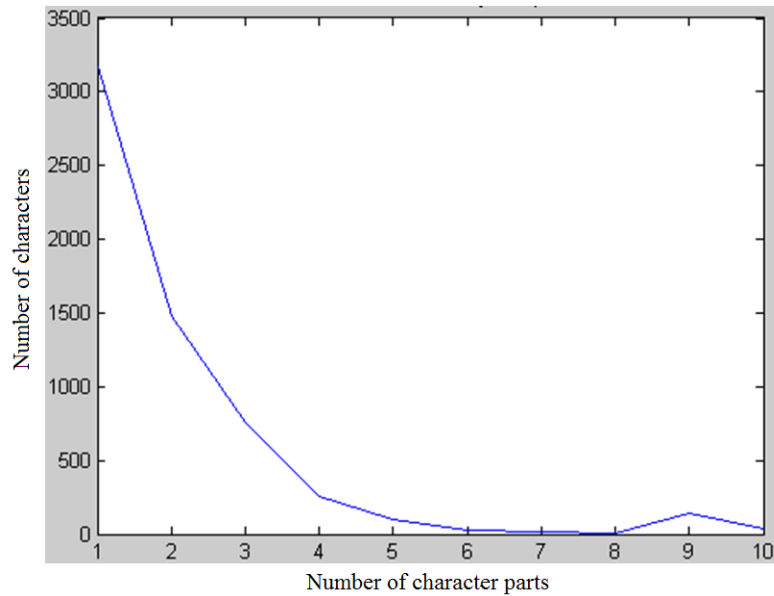


Figure 50: The histogram of number of character parts in our selected set of Unicode characters.

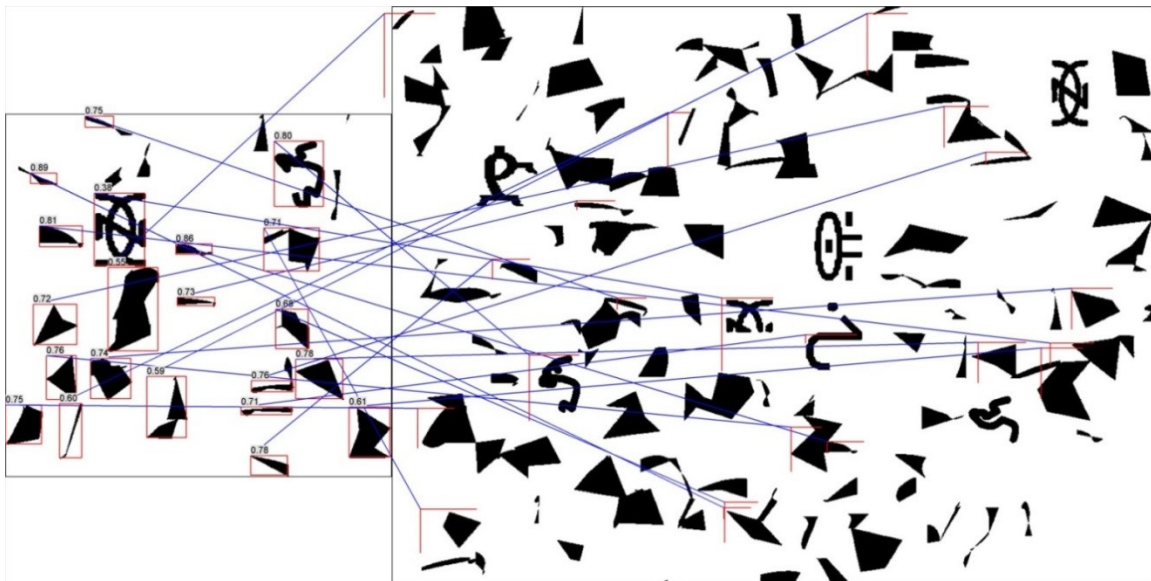


Figure 51: The result of applying a normalized cross-correlation matching algorithm on a monochrome image segmented by color palette.

After applying the cross-correlation algorithm on the monochromatic images segmented using the color palette, it was observed that in most cases (e.g. Figure 51), the first matching pairs in the images are curved noise components, and the character is incorrectly matched. Three strategies used in this CAPTCHA that reduce the effectiveness of cross-correlation algorithms are 1) including similar curved noise components; and 2) changing the aspect ratio of one of the characters of a matching pair, and 3) including character parts as noise in images. Curved noise components, which have many features of the characters, are semi-similar by design; hence they have a high amount of correlation. In addition, stretching a character of each matching pair in only one direction reduces their correlation.

The results of applying this algorithm on images of 100 CAPTCHAs are summarized in Table 27. This table shows that in monochromatic images containing a matching character pair without homoglyphs, the algorithm finds a match between a character and its match in 34.37% of the cases. However, it also finds 5 other matches on average. The strength (correlation coefficient) of the correct match (between the matching character pair) is not always greater than the other matched items. The strength randomly varies between rank 1 and 6. Hence, if the attacker relies on this algorithm to decide which one is the correct match, it has to choose randomly between the 6 found matches. Moreover, the strongest correlation coefficient does not vary in three different types of monochromatic images (i.e. in images containing a matching character pair, images containing a matching character pair with homoglyphs; and, noise images). Hence, the attacking algorithm does not know which image contains matching characters; and it has to decide randomly.

Table 27: Results of applying the cross-correlation algorithm on images segmented by color palette.

	Monochromatic images without homoglyphs in the keyboard	Monochromatic images with homoglyphs in the keyboard	Monochromatic images without a matching pair
Percentage of each image type	27.6%	10.3%	62.1%
The probability of detecting any match between the character and its real match	34.37%	0%	0%
Mean number of false positive symbol matches*	5	6	5
Percentage of correct answers of each type	6.87%	0%	0%

* After removing small rectangular noise

Based on the preceding argument and the information in Table 27, if the attacker selects 4 images randomly, and chooses one of the matches of each image randomly, the probability of success will be:

$$\begin{aligned}
 p_{NormCorr-palette-random} &= \left(0.276 \times \frac{0.344}{5} + 0.103 \times \frac{0.000}{6} + 0.621 \times \frac{0.000}{5} \right)^4 \\
 &= 1.3 \times 10^{-7}
 \end{aligned}$$

Performance of normalized cross-correlation algorithm when applied to images segmented by color histogram or k-means

Detecting character pairs in this scenario is more difficult than the previous one for the attacker because in images segmented by k-means, characters and background noise are sometimes connected together. In addition, after segmenting images using k-means or a color histogram, some noise components are connected together and it is hard to remove noise based on its shape or size (Figure 52). For the same reason, the number of false positives increases. The results of applying cross-correlation algorithm on images of 100 tests segmented by k-means clustering are summarized in Table 28.

Based on this information, the probability of success is:

$$p_{NormCorr-Kmeans-random} = \left(0.321 \times \frac{0.192}{10} + 0.099 \times \frac{0.000}{11} \right)^4 = 1.4 \times 10^{-9}$$

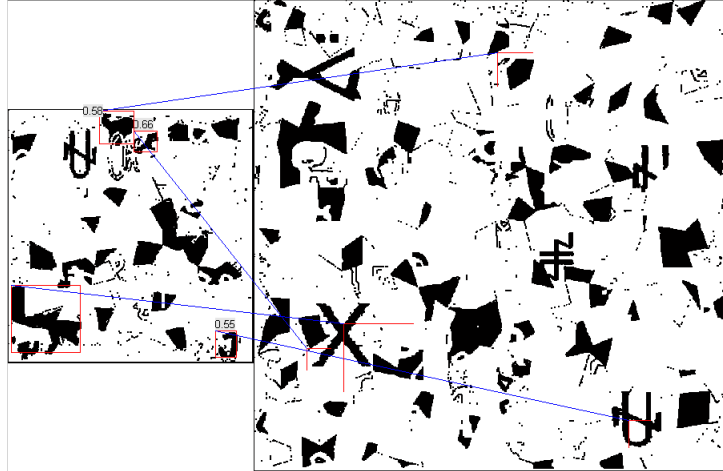


Figure 52: The result of applying a normalized cross correlation algorithm on images segmented by color histogram or k-means.

Table 28: Results of applying a cross-correlation algorithm on images segmented by k-means.

	Monochromatic images without homoglyphs in the keyboard	Monochromatic images with homoglyphs in the keyboard	Monochromatic images without a matching pair
Percentage of each image type	32.1%	9.9%	58%
The probability of detecting any match between the character and its real match	19.2%	0%	0%
Mean number of false positives	10	11	11
Percentage of correct answers of each type	1.92%	0%	0%

4.5 Conclusion

In this section, we described two segmentation and recognition attacks on the proposed base CAPTCHA without the interactive matching challenge and without mouse movement tracking. The segmentation attacks proved to be ineffective due to their inability to reliably separate characters from background noise. Consequently, in order to achieve a relatively conservative test of the CAPTCHA's resistance, we assumed that an attacker can emulate human behavior and use the color palette to successfully segment the CAPTCHA. Then, we tested recognition attacks on images of 100 CAPTCHA challenges and reported the results. The highest successful solution probabilities for each attack are summarized in Table 29.

Table 29: Results of the security analysis of the base CAPTCHA.

Attack type	Highest probability of success	Lowest probability of success
Random guess	1.3×10^{-32}	
SURF applied to images segmented by the color palette	4×10^{-5}	3×10^{-8}
SURF applied to images segmented by color histogram or k-means	3.5×10^{-6}	1.3×10^{-9}
Normalized cross-correlation algorithm when applied to images segmented by color palette	1.3×10^{-7}	
Normalized cross-correlation algorithm applied to images segmented by color histogram or k-means	1.4×10^{-9}	

A common goal for CAPTCHA design is that automated computer programs should not be able to solve CAPTCHA challenges with a success rate of higher than 0.01% [69, 95, 98]. Probabilities of success for all our attack scenarios are lower than 0.01% even when we assume that the attacker has perfectly segmented the CAPTCHA images using the color palette. Actual success probability for breaking a well-known CAPTCHA such as reCAPTCHA is considerably higher than 0.01% [91].

Our results show that image processing algorithms such as SURF do not perform well in breaking this CAPTCHA. The presence of homoglyphs, the existence of hard-to-remove curved noise components, and using random pieces of characters as noise in the images make this CAPTCHA more resistant against a strong local feature matching algorithm such as SURF. The above results ignore the impact of requiring intelligent interactions for the matching task, which was explored in Section 3.3 of Chapter 3.

In the next two chapters, we explore ways to further enhance security and usability for applications where higher levels of security or usability are desirable.

Chapter 5 Security-Enhancing Modifications

5.1 Introduction

Previous research shows that small changes in the internal structure of characters should not highly affect human recognition according to the gestalt laws of perception [158]. However, prior research indicates that changing the internal structure of characters will reduce a computer algorithm's ability to match characters, especially if these algorithms rely on relative local characteristics of character points that are likely to change when the internal structure of characters slightly change [14].

In order to utilize the above distinction between human and machines, we used several variations of three different types of modifications to create small changes in internal character structures or their immediate surroundings. These modifications include overlaying both the characters and background with specific noise types, juxtaposing characters and background noise, or creating the characters using a combination of smaller elements. We further compare the effect of our proposed modifications on both usability and security with two baseline cases: baseline case 1 which is our initial CAPTCHA without the proposed modifications, and baseline 2 which includes additional curved noise elements in the background.

The above mentioned modifications aimed to improve security without significantly affecting usability. In order to achieve this purpose, we first tested the effect of the proposed modifications on the matching ability of several major algorithms. Based on the results, we selected two types of modifications that most significantly impeded the algorithms' ability to match characters. These modifications were adopted in a user study to gauge a CAPTCHA's usability when these modifications are used. Modifications that do well on both security and usability would be implemented in the final CAPTCHA.

5.2 Modifications used in the tests

We used several variations of three categories of modifications:

- 1- Adding masking circles or curves (four variations)
- 2- Creating the characters using a combination of small circles (two variations)
- 3- Juxtaposing characters and noise (two variations)

The effects of the above modifications were compared with the control condition of having *additional* curved noises in the background.

We now describe these modification types in more detail.

- 1- *Adding masking circles or curves:*

We used four variations of this noise type which involved overlaying both the character and the background with 1) single small circles, 2) small circles in groups of three, 3) high density larger circles, and 4) random curves (Figure 53-a:d). The first mask is selected to see how small noise which is hardly visible to human users affects the performance of recognition algorithms. The idea behind the second mask is to realize the effect of a mask which is still hardly visible for humans and also has a more complex pattern (circles in groups of three). The third mask, high density large circles, was selected to create more internal changes within characters. Finally, masking curves are presumed to reduce the performance of matching algorithms by dividing characters into several parts and changing the relative characteristics of a character's internal points. This last strategy can especially affect algorithms that detect connected components first and use global methods for matching.

The above modifications make character recognition more difficult for computer programs. One reason is that masking circles or curves change the internal structure of the characters. These

changes are random and different in two characters of a matching pair. The first consequence of such modifications is the generation of more edges, corners or generally, more details in the characters. As a result, more false keypoints are selected by keypoint detectors. Another effect of changing the structure is on keypoint descriptors. Since the intensity of pixels in the area around the keypoints is disturbed, the descriptors are not as effective as they are in noiseless images. Moreover, because the areas around a keypoint in two characters of a matching pair are now more different, their descriptors are supposed to be more different and matching may not be as precise as before.

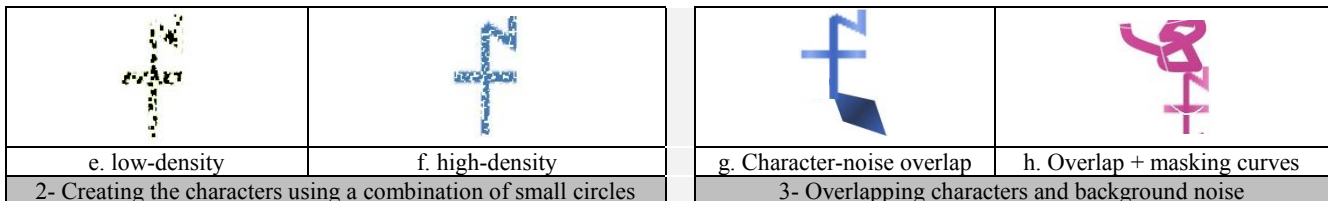
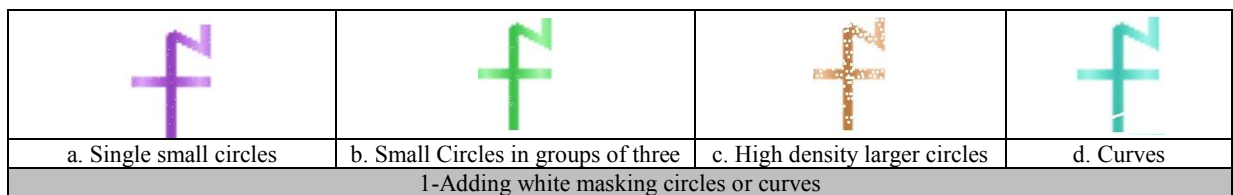


Figure 53: Security-enhancing modifications.

1-Masking circles or curves: a. single small circles (radius=1), b. small circles in groups of three (radius=1), c. high-density larger circles (radius=2), d. curves, 2- Creating the characters using a combination of small circles: e. sparse, d. dense, 3- Overlapping characters and background noise, g. overlap, h. overlap + masking curves.

2- Creating the characters using a combination of small circles:

In this modification strategy, instead of using a solid color (or a gradient) to paint the characters, the characters are created by painting the character area by small colored circles (Figure 53-e:f). Rather than noise being added to a character, noise is created by small patches of the character area that remain unfilled after painting the character area with circles. Two variations, a low- and a high-density variation, were used with differences in the density of

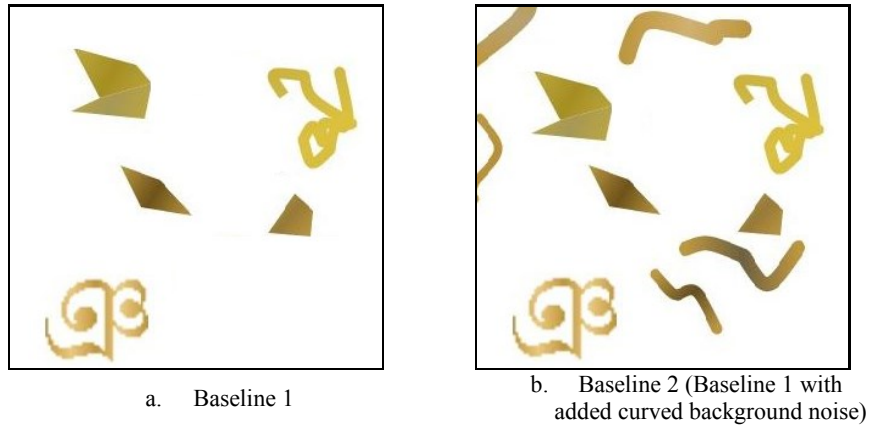
circles used to paint the characters. This type of distortion divides the character into several parts and changes the structure of the character. Moreover, some circles connect to each other and hence, the pattern (i.e. the circles) cannot be detected and removed by attacking algorithms. The size of the circles and their density may be changed.

3- Juxtaposing characters and noise:

Letting characters have a common boundary or a slight overlap with background noise (Figure 53-g:h) turns them into one connected component. This modification will have a negative impact on the ability of global pattern recognition programs because overlapping characters are harder to segment. This strategy may also reduce the performance of local feature representation algorithms in detecting keypoints and keypoint descriptors for the same reasons described for our first modification type. However, the amount of overlap should not be so much that it affects the usability. We used two variations of this modification which included a simple overlap and an overlap plus masking curves.

Baseline cases

We compared CAPTCHAs employing our proposed modifications with one without these modifications, called baseline 1, and one with additional curved noises in the background, called baseline 2. In the latter case, we added a random walk of curved noise to the background noise. Because background curved noise is more similar to characters, it can confuse computer programs. Increasing the amount of this type of noise decreases the performance of matching algorithms by raising the number of false positives. Figure 54 depicts examples of the baseline 1 and baseline 2 cases.



a. Baseline 1
 b. Baseline 2 (Baseline 1 with added curved background noise)

Figure 54: Increasing the amount of background curved noise.
 a. before, b. after increasing.

In the next section, we will describe the major matching algorithms used in testing the above proposed modifications.

5.3 Selected pattern matching algorithms

Several pattern recognition schemes have been developed in the literature. We have provided brief descriptions for the major schemes along with their advantages and disadvantages in Table 30 [159]. Most of these schemes are designed to match an identified object with another identified object or images that contain the identified object. However, our images include substantial amounts of noise, occasional overlaps, or multi-part characters which makes character segmentation and extraction more difficult.

As a result, most of these schemes face a limitation in solving our CAPTCHA as the identification of the unique objects in our CAPTCHA poses an additional challenge to them. However, descriptor-based algorithms are an exception among these schemes and do not face this limitation as they rely on local features of an image to match similar areas in two images.

Table 30: Object recognition methods.

	General explanation	Examples/Categories	Advantages	Disadvantages
Global methods	In global methods, the object to be recognized is considered as a whole. Global characteristics of the object such as area, perimeter, etc. are used for object recognition.	2D correlation: 2D cross correlation of an image with a template (a prototype representation of the object that needs to be found).	- Easy to implement, - Generic.	- Not invariant to rotation and scaling, - Only robust with respect to linear illumination changes, - Sensitive to clutter and occlusion.
		Global feature vectors: To extract a set of global features from the image (e.g., moments, FFT, color histogram, etc.) and to compare its feature vector to the one of the model.	Compared to correlation: - Faster, - Can be more invariant to changes (depending on the selected features).	- Requires segmentation of the objects from the background clutter, - A few global features may not be able to distinctively characterize complex objects.
Transformation-search-based methods	Transformation parameters between an image and all models are estimated (by means of maximizing the similarity between the image point set (e.g. edge points) and the transformed model point set). Then, pattern matching is performed by finding a parameter combination with maximum similarity.	Generalized Hough Transform (uses a voting system to perform pattern matching), The Hausdorff Distance (uses a nonlinear metric for the proximity of the points between two point sets).	- More robust to occlusion and clutter.	- Still cannot tolerate high amounts of clutter and occlusion or object deformations, - Hough Transform: substantial computational and storage requirements, - Hausdorff: Erroneous in densely populated areas.
Geometric correspondence-based approaches	The object is described by a set of primitives called features (e.g. corners, curves, specific shapes detected by geometric filters, etc.). The search is based on a one-to-one correspondence between an image feature and a model feature. This method is usually used in industrial applications where the geometry of the objects is usually known before recognition.	Graph-based matching (uses graphs to model the geometry of objects), Interpretation Trees (uses trees instead of graphs, to accelerate the search among features; each level of the tree represents possible pairings of a particular feature).	- Can handle partial occlusion or clutter. Tree: - Fast.	- Best for industrial applications, - A-priori knowledge about geometry of the shapes is necessary, - Not good where there are considerable deviations in the shapes of objects of the same class. Graph: - Computational cost rises exponentially with the number of features.
Flexible shape matching	These techniques use parametric curves, which offer enough degrees of freedom, to approximate the object contour as accurate as possible.	Active contour models (snakes): A snake is such as an elastic band which can be pulled to the contours of an object.	- Modeling arbitrarily shaped objects, - Quite insensitive to contour interruptions or local distortions.	- Relies on reasonable starting point of the curve (the possibility of getting stuck in local minima), - Some models may have high computational complexity.
Interest Point Detection and Region Descriptors	These methods try to identify objects with the help of descriptors of the object appearance in a local neighborhood around keypoints (keypoints: points selected by a suitable detector due to their saliency, e.g. corners).	SIFT: SIFT is a representative for all algorithms that perform pattern recognition by describing an object with the help of regional descriptors around its keypoints.	- Increased robustness to clutter and occlusion, - Good performance.	- Relies heavily on the keypoint detector performance.
		Shape context descriptors: Descriptors are derived from the shape information of an image region.	- Insensitive to local distortions, - Good performance.	- Enough contour information is needed.

Consequently, we find descriptor-based algorithms to be more capable in challenging the security of our CAPTCHA. Selecting the other schemes that are inherently unsuitable for challenging our CAPTCHA would be a futile effort. Hence, we select descriptor-based algorithms to test our CAPTCHA. Major representative algorithms from the schemes described in Table 30 are compared in [159] along several security dimensions such as computation speed, suitable object complexity, suitable intra-class object variation, invariance to viewpoint change and invariance to occlusion and clutter. According to this comparison, SIFT-based algorithms, which are a major group of descriptor-based algorithms, have a very high tolerance of noise and work on both simple and complex shapes, which makes them very suitable for challenging our CAPTCHA.

As a result, we selected several recent versions of SIFT-based algorithms and tested their performance on breaking our CAPTCHA with different types of noises, modifications and clutter. These algorithms are comprised of two sub-algorithms: a keypoint detector sub-algorithm and a descriptor sub-algorithm. A matching process for these paired algorithms starts with the detector which finds all keypoints in both images. Next, the descriptor algorithm produces a feature description vector for each keypoint. Eventually, feature vectors of the two images are compared to find points that have the same features. These points will be matched together.

We employed the following detector-descriptor algorithm pairs for our tests:

- 1- SIFT-SIFT
- 2- SURF-SURF
- 3- FAST-BRIEF
- 4- ORB-ORB

5- BRISK-BRISK

6- FAST-FREAK

Except for BRIEF and FREAK, which are merely descriptor algorithms, and FAST, which is merely a detector algorithm, the other algorithms, namely SIFT, SURF, ORB and BRISK, can serve as both detectors and descriptors.

SIFT and SURF descriptors rely on histograms of intensity gradients around a keypoint to form a feature vector for that keypoint. While SURF is a modified version of SIFT to reduce the computational complexity of this process, both these algorithms rank high in terms of required computations. The next group of descriptors, namely BRIEF, ORB, BRISK, and FREAK, was developed with the reduction of computational complexity in mind and achieves this purpose by employing binary feature vectors for keypoints. We provide a description of each individual algorithm as a detector, a descriptor or both in chronological order of their development.

5.3.1 SIFT (1999)

SIFT (Scale Invariant Feature Transform) [157] introduces an interest point detector and a region descriptor.

Detector: The detector is based on the Difference of Gaussians (DoG). Image information is not only a function of the spatial directions x and y but also a function of scale s (scale space). For example, fine scale analysis of the image reveals detailed information, such as texture information, whereas coarse scale analysis displays general information. DoG detects points in the image that are invariant to scale changes. In other words, keypoints introduced by DoG are the local maxima and minima in scale space. To find the maxima and minima, the $D(x,y,\sigma)$ of images at nearby scales is calculated by convolution of the image with difference of Gaussians (DoG) function:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \quad \text{Equation 7}$$

where $I(x,y)$ is the image, and k is a constant (typically between 1.1 and 1.4). The scale space can be explored by variation of σ . Local maxima and minima are identified by comparing location (x,y,σ) of $D(x,y,\sigma)$ with its 8 neighbors of the same scale (σ) and the 3×3 regions centered at (x,y) of the two neighboring scales.

Descriptor: a keypoint-region resulting from a DoG detector is partitioned into 4×4 sub-regions. Afterwards, the intensity gradient magnitude and direction at each image sample point of these sub-regions is calculated and accumulated in an 8-bin histogram for each sub-region separately. Finally, depending on the gradient magnitude and distance to the center of the region, a weighting is applied. The descriptor vector for one region has $4 \times 4 \times 8 = 128$ elements.

5.3.2 FAST (2006)

FAST (Features from Accelerated Segment Test) [159, 160] is a corner detector that is used for keypoint detection (hence, FAST is a detector, not a descriptor). This approach reduces the number of calculations required to detect a keypoint; hence, it is efficient and suitable for real-time applications. This algorithm places a circle consisting of 16 pixels around the pixel p which is being investigated. To decide whether pixel p belongs to a corner or not, only gray value differences between each of these 16 pixels and pixel p are evaluated. Hence, this method is very fast.

5.3.3 SURF (2008)

SURF (Speeded-Up Robust Features) [152] is a detector and descriptor that improves the speed and performance of descriptor-based schemes by relying on integral images for image convolution and by simplifying its selected detector and descriptor (Hessian detector and SIFT descriptor).

Detector: keypoint detection is performed by a basic Hessian matrix approximation. The Hessian matrix at point $\mathbf{x}=(x,y)$ of image I at scale σ is defined as:

$$H(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{yx}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \quad \text{Equation 8}$$

where $L_{xx}(\mathbf{x},\sigma)$ is the convolution of the Gaussian second-order derivative with the image I in point \mathbf{x} ; and the same for $L_{xy}(\mathbf{x},\sigma)$, $L_{yx}(\mathbf{x},\sigma)$ and $L_{yy}(\mathbf{x},\sigma)$. Because Gaussian filters are not optimal for practical use, and they need to be discretized and cropped, the developers of SURF used box filters that are approximations of Gaussian derivatives and their evaluation can be performed at a very low cost. Using integral images [152] in this step drastically reduces the computation time.

Descriptor: The first step is to find the dominant orientation of the interest region (to make SURF rotation invariant²⁵). Afterwards, a square region around the keypoint is selected and rotated along the selected orientation. This area is divided into 4x4 sub-regions; each of which is further divided into 5x5 squares. For each square, the wavelet responses, representing d_x and d_y , are computed. Finally, for each sub-region, all d_x and d_y are collected and hence, each sub-region is represented by four features: $\sum d_x$, $\sum |d_x|$, $\sum d_y$, $\sum |d_y|$. The length of the descriptor vector of each keypoint is 4x4x4=64 (which is smaller than SIFT).

5.3.4 BRIEF (2010)

BRIEF (Binary Robust Independent Elementary Features) [161] is a standalone descriptor based on binary strings. The idea behind this descriptor is that a descriptor vector of a few bits computed using simple intensity difference tests can be highly discriminative, while it is light-weight and fast at the same time.

²⁵ Many applications do not need rotation invariance; they can use an upright version of SIFT which is faster.

Descriptor: This approach first defines a test pattern (a group of points and their connections) in the interest region²⁶. Then, the region's descriptor vector is calculated by pairwise intensity comparisons between test pairs for a given adopted pattern. The comparison between two points of each pair is performed using the following formula:

$$\tau(p; x, y) := \begin{cases} 1 & \text{if } p(x) < p(y) \\ 0 & \text{otherwise} \end{cases} \quad \text{Equation 9}$$

where $p(x)$ is the pixel intensity at point x ; and x and y are the two points of a test pair. The next step is to measure the similarity of descriptors of two regions. This can be performed simply and quickly by Hamming distance. BRIEF is not orientation invariant due to the nature of its design.

Test patterns in BRIEF are not restricted to a specific model and pairs can be chosen at any location in the interest region. The authors of [161] suggested five different patterns including:

- i. x and y locations²⁷ are chosen randomly based on a uniform distribution,
- ii. x and y locations are selected randomly based on a Gaussian distribution,
- iii. x and y locations are randomly chosen using a Gaussian distribution in two steps: first, x is sampled using a Gaussian centered at the region center. Secondly, y is chosen using another Gaussian centered at x . Consequently, test pairs are more local.
- iv. x and y locations are randomly selected from discrete locations of a coarse polar grid.
- v. x locations are all (0,0); and y takes all possible values on a coarse polar grid [161].

The authors showed that pattern ii outperforms the other patterns.

²⁶ A region around a key point.

²⁷ X s are one end and Y s are the other end of the lines.

5.3.5 ORB (2011)

ORB (Oriented FAST and Rotated BRIEF) [162] is an orientation-invariant, noise-insensitive version of BRIEF. ORB introduces a detector and a descriptor which are explained in order.

Detector: ORB detector, oFAST²⁸, is based on the FAST detector. oFAST proposes a strategy to select corners more precisely and to calculate the orientation of each selected corner. FAST has considerable number of responses along edges. In order to choose corners more accurately, oFAST uses a Harris corner measure to sort FAST keypoints and picks the top N points (N : the target number of keypoints). To measure the orientation of the corner, first, the intensity centroid of the region is calculated using the moments:

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \quad \text{Equation 10}$$

$$c = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \quad \text{Equation 11}$$

where $I(x,y)$ is the region of interest. After constructing a vector from the center of the corner, O , to the centroid, \overrightarrow{OC} , the orientation of the region is:

$$\theta = \text{atan2}(m_{01}, m_{10}) \quad \text{Equation 12}$$

Descriptor: ORB descriptor, rBRIEF, is an orientation-invariant version of BRIEF. This descriptor first steers the test pattern according to the orientation of the key-point measured by the detector. Rublee et al. [162] showed that steered BRIEF significantly reduces the variance of features which makes them less discriminative. To recover from this loss, rBRIEF uses a

²⁸ FAST keypoint orientation.

learning method to choose a good subset of binary tests and to select pairs with lower correlation and higher variance.

The main difference between BRIEF and ORB is that:

- ORB is rotation-invariant,
- While BRIEF uses randomly selected sampling pairs, ORB benefits from a training method that chooses optimal sampling pairs.

5.3.6 BRISK (2011)

Brisk (Binary Robust Invariant Scalable Keypoint) [163] is a rotation and scale-invariant detector/descriptor.

Detector: The detector is a scale-space enabled version of FAST. The scale-space pyramid layers in this model consist of n octaves and n intra-octaves (typically, $n=4$). Each octave c_i is a half-sampling of its previous octave c_{i-1} ; and each intra-octave d_i , located between two octaves c_i and c_{i+1} , is obtained by down-sampling c_i by a factor of 1.5. c_0 is the original image. To find the key-points, first of all, FAST is applied to each layer. Each point, to be considered as a key-point, not only needs to fulfill the maximum condition with respect to its eight neighboring points in the same layer, but it also needs to achieve saliency with regard to its immediately-neighboring layers.

Descriptor: The pattern used for sampling of the key-point region consists of N locations equally spaced on circles concentric with the key-point. In order to avoid aliasing effects, a Gaussian smoothing filter is applied on an area around each sample point. Sampling-point pairs include:

$$A = \{(p_i, p_j) | i < N, j < i\} \tag{Equation 13}$$

These pairs are divided into two groups of short-distance and long-distance pairings:

$$S = \{(p_i, p_j) \in A | \|p_j - p_i\| < \delta_{max}\} \tag{Equation 14}$$

$$S = \{(p_i, p_j) \in A | \|p_j - p_i\| > \delta_{min}\}$$

The thresholds δ_{min} and δ_{max} are set to: $\delta_{max} = 9.75t$, and $\delta_{min} = 13.67t$. Long-distance pairs are used to determine the orientation of the region (by which the sampling pattern is rotated before calculating the descriptor) and short-distance pairs are used to build the descriptor by pairwise intensity comparisons. The result of each comparison is, the same as BRIEF, either 0 or 1, depending on which point in the pair has the higher intensity. Finally, matching of the patterns can be performed by a bitwise XOR followed by a bit count.

5.3.7 FREAK (2012)

FREAK (Fast Retina Keypoint) [164] is a standalone rotation-invariant descriptor which is inspired by the human visual system or more specifically by the retina.

Descriptor: Similar to BRISK, FREAK suggests a circular sampling pattern. However, in FREAK's pattern, the density of points near the center is higher. This pattern, which is biologically inspired by the retinal pattern in the eye, consists of 43 weighted Gaussians around the key-point. These areas overlap. In order to select sampling pairs, an algorithm similar to ORB is used to select pairs that are more discriminative and less correlated²⁹ [164].

To be rotation-invariant, FREAK also calculates the orientation of the region; and rotates the sampling pairs by the measured angle. In order to measure the orientation, in contrast with

²⁹ The authors stated that BRISK's method (selecting short-distance pairs as sampling pairs) results in choosing highly correlated pairs.

BRISK that uses long-distance pairs; FREAK uses pairs with symmetric receptive fields with respect to the center.

5.4 The effect of different noise on the performance of pattern matching algorithms

In the previous sections, we described our modification/noise types and the pattern matching algorithms used to compare their effect on CAPTCHA security. We implemented the selected modifications/noises on our CAPTCHAs and used each algorithm to solve a hundred CAPTCHAs comprised of 12 monochromatic image pairs from which 5 included target characters. Table 31 shows the mean probability of success in correctly matching a character. Since these algorithms match image keypoints rather than objects, the success or failure of the matching process depends on how identified keypoint matches are interpreted to create potential object matches. To address this interpretational issue, we calculated three probabilities, P1, P2, and P3 to consider main potential interpretations. P1 is the probability of success if the objects with highest number of matching points are selected as the matched character pair. P2 is the probability of success if one random pair of matched *objects* (connected components) is selected as the final match; and P3 is the probability of success if one pair of matched *points* is randomly selected as the match.

Next, we tabulated the maximum and the average probability of success across all algorithms for each modification type within each of the three probability groups P1, P2, and P3 in Table 32. In addition, we ranked these noise types in terms of their ability to reduce an algorithm's success probability as defined by P1. P1 represents the most intelligent method for an algorithm to map matched points onto matched objects. Using P1 for comparison of the algorithms' success probability should allow a more rigorous test of the algorithms in solving this CAPTCHA.

Table 31: Probability of success in correctly matching a character using different pattern recognition algorithms.

Matching algorithms ▶	SIFT-SIFT			SURF-SURF			FAST-BRIEF		
Modification types ▼	p1	p2	p3	p1	p2	p3	p1	p2	p3
Original	0.0917	0.0619	0.0711	0.0795	0.0289	0.0132	0.0792	0.0347	0.0518
Masks – single small circles	0.0542	0.0350	0.0425	0.0750	0.0250	0.0284	0.0367	0.0214	0.0267
Masks – small circles in groups of 3	0.0875	0.0417	0.0553	0.0472	0.0254	0.0341	0.0589	0.0256	0.0385
Masks – curves	0.0750	0.0486	0.0563	0.0458	0.0264	0.0334	0.0567	0.0331	0.0433
Masks – high density larger circles	0	0	0	0.0667	0.0583	0.0611	0.0333	0.0097	0.0167
painting w colored circles–low density	0.0111	0.0125	0.0119	0	0	0	0.0242	0.0117	0.0142
painting w colored circles–high density	0.0458	0.0569	0.0576	0.0514	0.0387	0.0379	0.0167	0.0167	0.0154
Character noise overlap	0.0667	0.0581	0.0629	0.0333	0.0104	0.0145	0.0500	0.0306	0.0441
Overlap + masking curves	0.0250	0.0278	0.0258	0.0315	0.0100	0.0145	0.0200	0.0117	0.0144
More curved noise in background	0.0708	0.0347	0.0431	0.0306	0.0077	0.0122	0	0.0056	0.0056

Matching algorithms ▶	ORB-ORB			BRISK-BRISK			FAST-FREAK		
Modification types ▼	p1	p2	p3	p1	p2	p3	p1	p2	p3
Original	0.0292	0.0109	0.0139	0.0333	0.0167	0.0200	0.0057	0.0140	0.0113
Masks – single small circles	0.0265	0.0207	0.0213	0.0267	0.0200	0.0211	0.0083	0.0083	0.0083
Masks – small circles in groups of 3	0.0304	0.0130	0.0134	0.0200	0.0142	0.0133	0.0264	0.0297	0.0292
Masks – curves	0.0641	0.0227	0.0333	0.0100	0.0100	0.0100	0	0.0042	0.0033
Masks – high density larger circles	0.0222	0.0256	0.0250	0.0200	0.0100	0.0100	0	0	0
painting w colored circles–low density	0	0	0	0	0	0	0	0	0
painting w colored circles–high density	0.0190	0.0165	0.0172	0	0	0	0.0167	0.0167	0.0167
Character noise overlap	0.0258	0.0127	0.0148	0	0.0083	0.0067	0.0089	0.0089	0.0089
Overlap + masking curves	0.0167	0.0090	0.0094	0.0033	0.0075	0.0061	0.0056	0.0111	0.0097
More curved noise in background	0.0032	0.0054	0.0052	0	0	0	0	0.0012	0.0018

According to Table 32, “painting with colored circles-low density” and “overlap plus masking curves” have the largest negative effect on the performance of our tested algorithms regardless of using the maximum or average success probability for comparison. Next in line is the “painting with colored circles – high density” noise that ranks third in terms of the maximum probability of success which is more relevant than the average probability of success as an attacker would ideally use the best-performing algorithm to challenge a CAPTCHA.

Table 32: Ranking of noise types based on their impact on the performance of matching algorithms.

Modification	Rank (based on P1 _{max})	Rank (based on P1 _{avg})	Maximum probability			Average probability		
			P1	P2	P3	P1	P2	P3
painting with colored circles – low density	1	1	0.0242	0.0125	0.0142	0.0059	0.0040	0.0043
Overlap + masking curves	2	2	0.0315	0.0278	0.0258	0.0171	0.0128	0.0133
painting with colored circles – high density	3	5	0.0514	0.0569	0.0576	0.0249	0.0242	0.0241
Character noise overlap	4	6	0.0667	0.0581	0.0629	0.0308	0.0215	0.0253
Masks – high density larger circles	5	4	0.0667	0.0583	0.0611	0.0237	0.0173	0.0188
More curved noise in background (Baseline2)	6	3	0.0708	0.0347	0.0431	0.0174	0.0091	0.0113
Masks – single small circles	7	7	0.0750	0.0350	0.0425	0.0379	0.0217	0.0247
Masks – curves	8	8	0.0750	0.0486	0.0563	0.0419	0.0242	0.0299
Masks – small circles in groups of 3	9	9	0.0875	0.0417	0.0553	0.0451	0.0249	0.0306
Original (Baseline1)	10	10	0.0917	0.0619	0.0711	0.0531	0.0279	0.0302

Hence, we considered the modifications that ranked highest in challenging the matching algorithms as determined by P1, i.e. the maximum probability of solution success, and considered to use them in our test with human users. If these modification types do not substantially decrease usability, we can implement them in our CAPTCHA to improve security. With a more secure CAPTCHA, we can further simplify the tests for human users by reducing the number of characters they need to match in order to solve the CAPTCHA.

In our CAPTCHA, the probability of solving a test that needs matching n characters can be calculated from:

$$P_{\text{solving a CAPTCHA}} = (P_{\text{solving one character}} \times EV(P_{DnD}))^n$$

Where P_{DnD} is the probability of an interactive game type being successfully solved and EV is the expected value operator for this probability across the three different interactive game types. The above probability has to be less than 0.01% or 0.0001 in order for a CAPTCHA test to be considered strong against attacks [98]. In our calculation of the probabilities, we also have to account for the limitation that 4 matching attempt failures are allowed for each character as explained in Section 3.3.

For the first three top ranking modification types, the probability of success in breaking the CAPTCHA with 2 test characters ($n=2$) is:

$$\begin{aligned} P_{\text{solving a CAPTCHA}} &= \left(\text{Max}(P1, P2, P3)_{\text{top 3 modifications}} \times EV(P_{DnD}) \right)^2 \\ &= \left(0.0576 \times \frac{1}{3} \times 4 \times (0.0066 + 0.017 + 0.0045) \right)^2 \\ &= 4.7 \times 10^{-6} \end{aligned}$$

Hence, with these security modifications a user can be required to solve two characters only in order to pass the test.

5.5 User study

In order to examine the usability of the CAPTCHA after adding security-enhancing modifications, we conducted a user study in which we asked a sample of 40 users (university students) to solve our CAPTCHA with the implemented modifications and recorded relevant data such as solution times and accuracy.

The two modification types with the highest ranks in our security-enhancing test of the previous section were considered in this user study. However, sample renderings of the top-ranking modification, the “Painting with colored circles-low density” modification, showed that this modification significantly compromised the integrity of the characters, making them hardly distinguishable. Due to its low face validity and potentially low usability, we replaced this modification in our user study with the next top ranking modification of Table 32. Table 33 shows the final two security-enhancing modifications studied in our experiment.

Table 33: Factors tested in the user study.

Factor	Settings
Modification type	Overlapping characters and noise, plus masking curves.
	Painting with colored circles-high density

In this study, the effect of modification types on solution time and accuracy was planned to be compared with a Baseline 1 version of the CAPTCHA that had no security-enhancing modification and comprised our control condition. We also intended to compare the effect of these modifications with the Baseline 2 modification, i.e. “More curved noise in background”. However, the initial tests of this modification showed that additional curved noise in the background made the CAPTCHA difficult for users to solve leading to low face validity for

usability. Eventually, we only conducted the user study on CAPTCHAs that had either of the two modification types of Table 33 (our treatment conditions) and a CAPTCHA without these modifications, i.e. Baseline 1 (control condition).

In our study, the users were asked to solve the three above versions of the CAPTCHA: one original CAPTCHA, one CAPTCHA with ‘Overlapping characters and noise, plus masking curves’, and one CAPTCHA with ‘Painting with colored circles-high density’. The order of these three cases was random for each user to neutralize learning and other carryover effects on solution accuracy and timing across users.

5.5.1 Results of the user study

Table 34 shows the average solution time and accuracy of 40 users for each tested modification type in addition to the original CAPTCHA which comprise our three test conditions. According to this table, while condition 2 does not highly affect the accuracy of the CAPTCHA, condition 3 substantially reduces the accuracy. The average solution time values show differences across noise types. However, in order to test if these differences are statistically significant, we ran an ANOVA model. In this model, we used solution time as the dependent variable and tested for a difference in the mean of solution time across the three test conditions. The test showed a significant difference of solution times across the three groups ($F(2,117)=3.23$, $p\text{-value}=0.04$).

Table 34: The effect of different noise/distortion types on accuracy and solution time.

Noise/distortion type	Solving Accuracy			Average solution time (s)
	T	F	W	
Condition 1: Original	82.5%	17.5%	0%	101.87
Condition 2: Overlapping characters and noise + masking curves	77.5%	22.5%	0%	106.59
Condition 3: Painting with colored circles – high density	72.5%	27.5%	0%	120.18

The ANOVA test shows that a significant difference exists across conditions in term of solution time. However, it does not make any between-group comparisons to pinpoint which group is significantly different from the others. In order to make pairwise comparisons across groups, we further conducted a pairwise comparison test. The results of this test are shown in Table 35.

Table 35: Pairwise comparison test for solution time.

Dependent variable: Solution time	Difference in group means	t-statistic	P-value
Condition 2 vs. 1	4.72	0.63	0.804
Condition 3 vs. 1	18.31	2.45	0.042
Condition 3 vs. 2	13.59	1.82	0.169

As the results indicate, condition 3, “Painting with colored circles – high density” exhibited a significantly higher solution time compared with the two other conditions. However, no significant difference was found between condition 2 and condition 1.

5.6 Conclusion

In this chapter, we proposed and tested several security-enhancing modifications for our CAPTCHA. Our security analysis showed that “painting with colored circles – low density”, “overlapping characters and noise + masking curves”, and “painting with colored circles – high density” had the highest positive impact on security among the proposed modifications. The first of these three modifications made the CAPTCHA very difficult for human users and was not explored further. The next two modifications were chosen for a test of usability in a user study.

The results of our user study indicate that from the two main security-enhancing modifications tested, “painting with colored circles – high density” significantly compromises usability by both increasing solution time and reducing accuracy. However, despite some change in average solution time and accuracy, the impact of our other security-enhancing modification,

namely “overlapping characters and noise + masking curves” on usability was not considerable. This finding suggests that our CAPTCHA might become more secure with this modification without paying a high price in terms of usability.

The addition of security-enhancing modifications to our CAPTCHA allows us to reduce the number of characters users need to match while keeping the security at the pre-enhanced levels. This possibility could further increase usability as users finish the solution process by solving fewer characters and find them among the more visible/convenient/familiar ones.

This CAPTCHA’s security and usability can always be adjusted based on the importance of the resource being protected. For higher security resources, more types of noise/distortion may be added to the tests and for lower security, less noise can be used or matching of fewer numbers of characters may be required.

Chapter 6 Usability-Enhancing Considerations

6.1 Introduction

In this section, we study the effect of two important variables, namely familiarity with characters and learning on the usability of our CAPTCHA. First, we review findings on the impact of text familiarity on user performance.

Prior research indicates that in visual search with unfamiliar distractors, familiar targets result in shorter response times than unfamiliar targets [165]. Goonetilleke et al. [166] found that Chinese subjects could find target Chinese characters among other Chinese characters faster and with higher accuracy than non-Chinese subjects.

However, familiarity may not always have a positive effect on performance. Other research shows that familiarity with text can indeed cause complacency and have a negative impact on error detection [167]. The likelihood of predicting textual elements in advance, rather than observing them in full detail, increases with text familiarity. In addition, people are more likely to ignore small details when text is familiar.

Given these contradictory findings, it is not clear how inclusion of familiar characters, i.e. characters from a user's mother tongue, might affect usability variables such as solution time and accuracy. In order to empirically study the effect of text familiarity on these variables, we have conducted a user study to examine their effect.

Another factor that can affect a CAPTCHA's usability is learning through experience and practice. According to the learning curve theory [168], repeated engagement in a task can decrease the required time and cost of that task due to the accumulation of experience and the

continuous improvement of proficiency. This learning effect stabilizes over time and reaches a plateau.

One possible explanation for the learning effect is the fact that people might be overwhelmed by available information when they do a task for the first time. As a result, they might not know what part of that information is useful and how to use it. As experience builds up, they learn how to pay selective attention to the relevant data in a more efficient matter and to extract meaningful information more effectively. Another explanation is that for first-time problem solvers, their knowledge is not meaningfully organized and conditionalized. As they solve more problems, they learn how to sort their knowledge about that problem and to organize it around core concepts and to follow a specific solution approach. Furthermore, due to a high cognitive load, a first time problem solver may consider only the first solution that comes to their mind and may not look for possible flaws of that solution. After solving the problem repeatedly, the solver gains the ability to avoid common mistakes.

We expect our CAPTCHA's usability to significantly increase with a slight amount of practice as a user becomes familiar with its dynamics. In order to empirically test this prediction, we perform a user study to examine the effect of practice on the solution time and accuracy for our proposed CAPTCHA.

6.2 User study

In order to examine the effect of character familiarity and practice on the usability of the proposed CAPTCHA, we conducted two user studies in which we asked users to solve our CAPTCHA under two conditions and recorded relevant data such as solution times and accuracy. In this section, the details of these user studies and their results are presented.

6.2.1 Design of the user study

In this study, we examined the effects of two factors on users:

- 1- The familiarity of the characters to be matched, and
- 2- The practice effect.

First, to test for the effect of familiarity, we show the users a few characters of their own language in each test. It is conceivable that human users can recognize the characters of their mother tongue easier and faster. Hence, we expect familiarity to improve usability by increasing accuracy and reducing the time of the test.

Secondly, we explore how user learning influences their accuracy and solution times with our CAPTCHA. We define user learning as experience obtained in solving a few instances of our CAPTCHA. Table 36 shows the factors studied in this experiment.

Table 36: Factors tested in the user study.

Factor	Levels
Familiarity	0 test characters from users' mother tongue
	2 test characters from users' mother tongue
	5 test characters from users' mother tongue
Practice effect	Solving 5 CAPTCHAs by the same user.

Forty university students participated in a study to test the effect of familiarity with characters; they solved three tests each: one with 5 test characters from their mother tongue, one with 2 test characters from their mother tongue; and one with no characters from their mother tongue. The order of these three cases was random for each user.

In order to study the effect of practice, 40 additional users were asked to solve 5 tests each and their solution time and accuracy was recorded for each solution.

6.2.2 Results of the experiment

Table 37 shows how users' average solution time and accuracy changes with the number of test characters coming from a user's mother tongue. In terms of accuracy, the results do not show any considerable change when the number of familiar characters increases. This could be due to the fact that familiarity with characters could lead to user complacency and decreased attention to subtle differences between test characters and potential homoglyphs, which would in turn inhibit potential improvements in accuracy in the presence of familiar characters.

While some differences can be observed across our three test groups for solution time, direct comparison of these values would not be informative. Hence, we ran an ANOVA model to compare the mean solution time over the distribution of responses from these three groups.

Table 37: The effect of familiarity of test characters on accuracy and solution time.

Condition	# of characters from user language	Solving Accuracy			Average solution time (s)
		T	F	W	
1	0	80.0%	20.0%	0%	118.41
2	2	77.5%	22.5%	0%	101.04
3	5	80.0%	20.0%	0%	102.10

The ANOVA test showed significant difference in mean solution times across the groups ($F(2,117)=4.13$, $p\text{-value}=0.01$).

In order to identify the differences of the three conditions in solution time, a pairwise comparison test was conducted. The results (Table 38) show a significant contrast between the solution times of all groups. The results indicate that solution time does decrease when familiar characters are used in the CAPTCHA.

Table 38: Pair-wise comparison test for solution time.

Dependent variable: Time	Difference in group means	t-statistic	P-value
Condition 2 vs. 1	-17.37	-2.56	0.03
Condition 3 vs. 1	-16.32	-2.41	0.04
Condition 3 vs. 2	1.05	0.16	0.98

The mother tongue of most of the subjects in this user study fell within one of three script types. There were 15 people with Latin, 13 people with Chinese, and 6 people with Arabic as their mother tongue script. Comparison of the effect of familiarity on solution time and accuracy across these three groups did not indicate considerable difference among these three groups. This is to be viewed with caution as the small sample sizes of the groups do not allow for a meaningful comparison.

In our second user study, we studied the effect of practice on solution time and accuracy. Table 39 displays the average solution time and accuracy for each of 5 ordered trials across the 40 subjects. The results show a reduction in solution time as the number of trials increases. Furthermore, a slight increase in accuracy is observed. We ran a regression model to further test the impact of solution order on solution time.

Table 39: The effect of learning on the accuracy and solution time.

Solution order	Solution Accuracy			Average solution time (s)
	T	F	W	
1	77.5%	22.5%	0%	116.62
2	80.0%	20.0%	0%	97.39
3	82.5%	17.5%	0%	72.23
4	82.5%	17.5%	0%	66.31
5	85.0%	15.0%	0%	64.15

Solution time and solution order were respectively used as the dependent variable and the independent variable in our regression model. In addition, a quadratic term for solution order was also used as a second independent variable. Based on the law of diminishing marginal returns, it is likely that the marginal impact of practice on solution time decreases as more tests are solved. In such a case, improvements in solution time are expected to become smaller and smaller as

people solve more and more CAPTCHAs. If this effect exists in our sample, we can capture it through the quadratic term.

Introduction of a quadratic term can lead to multi-collinearity in the regression model. We solved this issue by mean-centering the independent variable, i.e. the solution order. To mean-center a variable, the mean of the variable is subtracted from each value of that variable. Mean-centering changes the interpretation of the intercept in the regression model. In the results reported in Table 40, the intercept should be interpreted as the solution time for the average value of solution order, which is 3 in our data.

Table 40: Regression results for solution time.

Dependent Variable: Time	Coefficient	Std. Err.	t-statistic	P-value	Sig.
Intercept	75.71	2.52	29.94	0.000	***
Solution order	-13.6	1.14	-11.85	0.000	***
Solution order ²	3.81	0.97	3.93	0.000	***
Model was significant with $F(2,197)=77.97$; $R\text{-squared}=0.44$					

Our results indicate that solution order has the expected negative effect on solution time. We also observe a diminishing marginal effect of solution order on solution time.

6.3 Conclusion

In this chapter, we investigated the effect of learning and familiarity with characters on CAPTCHA solution times and accuracy. The results of our first user study indicate that accuracy did not change substantially within our sample with the addition of users' mother tongue characters in the test. This result might be partly due to users becoming less careful and paying less attention to minor details when they identify characters from their own mother tongue. This potential scenario could lead to the selection of homoglyphs due to the resulting complacency.

With respect to solution times, our user studies show that solution times slightly improve with the addition of 2 or 5 characters from a person's mother tongue. However, the effect of learning on solution times is much stronger as demonstrated by the results of our second study. This latter finding suggests that solution times can substantially improve when users solve a few instances of our CAPTCHA. In addition, occasional use of characters from a person's mother tongue script can slightly improve solution times, which could be a useful strategy in some applications.

Chapter 7 Discussion and Conclusion

In this thesis, we review the extant CAPTCHA literature to identify and classify the vulnerabilities of existing CAPTCHAs. Table 3 and Table 4 illustrate the extent of these vulnerabilities and highlight the need for the development of more secure CAPTCHAs.

Informed by our review of the literature, we proposed and developed a CAPTCHA that aims to address these vulnerabilities by taking advantage of Unicode and the homoglyphs within it, noise components, color, intelligent interactivity, and a virtual keyboard. We conducted several tests to assess the security and usability of the proposed CAPTCHA.

From a security point of view, a goal for CAPTCHA design is that automated computer programs should not be able to solve CAPTCHA challenges with a success rate of higher than 0.01% [98]. Our base CAPTCHA successfully meets this criterion by a good margin. It is noteworthy that actual success rates for breaking some famous CAPTCHAs are substantially higher than the 0.01% level. For example, reCAPTCHA has been broken with success rates ranging from 5% to 99.8% [94].

Furthermore, additional modifications to our base CAPTCHA, e.g. the addition of an intelligent matching task in Chapter 3, or the security-enhancing modifications tested in Chapter 5, further reduce the success rate of automated programs and allow us to achieve the same level of security even when we require users to solve fewer characters. The addition of an intelligent matching task also allows us to diversify the risk of the CAPTCHA being broken across two complex problems. The high amount of random dynamics employed in this interactive matching task makes it difficult for an algorithm to find predictable patterns that could help it simulate the required interactive behaviour.

From a usability perspective, two dimensions are noteworthy: user comfort and solving time. A large proportion of the people who solved our CAPTCHA indicated that they were comfortable solving it and expressed interest in solving more tests. Hence, our CAPTCHA's usability is reasonably good on this dimension. This is partly due to the fact that, despite the CAPTCHA's complexity for machines, it is still intuitive and straightforward to solve for a human user. User accuracy in solving our CAPTCHA is comparable to other existing CAPTCHAs, which supports the notion that the CAPTCHA is not too complex for users.

Regarding the second usability dimension, i.e. solving time, our CAPTCHA takes longer than some other CAPTCHAs. However, direct comparison between our CAPTCHA's solution time and other CAPTCHAs' may not be appropriate for a few reasons. First, interactive CAPTCHAs generally take longer to solve than non-interactive CAPTCHAs due to the additional solution steps they involve. Second, a fair comparison of solving times across CAPTCHAs must take into account the level of security that these CAPTCHAs offer.

The findings of our user study on the learning effect suggest that a large proportion of our CAPTCHA's solving time comes from user unfamiliarity with the new CAPTCHA. We observe that solving the CAPTCHA for five times reduces average solving times by as much as 45%. In addition, the game-like nature of the CAPTCHA can reduce the psychological perception of time as people enjoy solving the CAPTCHA.

Our user studies shed some light on the impact of several design factors on the usability of a CAPTCHA. In our studies, we find that users exhibit an expected from-easy-to-hard solution pattern. This feature allowed us to require users to match fewer character pairs in the highly interactive version of the CAPTCHA introduced in Chapter 3 without reducing the CAPTCHA's security level. In addition, we did not find any effect of character color or location on users'

solving accuracy. Using characters from a person's mother tongue script was shown to slightly improve solution times without any substantial impact on solving accuracy.

In sum, in this thesis we proposed, developed and tested a new CAPTCHA that addresses observed vulnerabilities exhibited by existing CAPTCHAs. In addition, we investigated the impact of several factors on CAPTCHA usability and security. Our work contributes to the literature on CAPTCHAs by highlighting the need for the development of more secure CAPTCHAs and provides encouraging results from the first utilization of Unicode and its characteristics in addressing many of the challenges faced by existing CAPTCHAs. We documented, classified, and analyzed several types of vulnerabilities and proposed a multipronged approach to CAPTCHA security in order to address them. Our work provides a benchmark that can be used in the development of future multilayered CAPTCHAs. We also developed a human mouse movement profile by documenting the empirical distribution of human mouse movement characteristics that can be used to further enhance security. Our work also provides insights on the effect of several contextual factors on human users' performance in solving CAPTCHAs.

Security has become increasingly critical as the World Wide Web matures and protecting online resources will be an important topic as the world population becomes more and more reliant on the Internet. The work presented in this thesis takes a step in the direction of achieving higher levels of security in online applications.

Bibliography

- [1] L. Von Ahn, M. Blum, and J. Langford, "Telling humans and computers apart automatically," *Communications of the ACM*, vol. 47, pp. 56-60, 2004.
- [2] H. S. Baird, A. L. Coates, and R. J. Fateman, "PessimPrint: a reverse Turing test," *International Journal on Document Analysis and Recognition*, vol. 5, pp. 158-163, 2003.
- [3] J. Yan, "Bot, cyborg and automated Turing test," in *Security Protocols Workshop*, 2006, pp. 190-197.
- [4] H. Baird and K. Popat, "Human interactive proofs and document image analysis," presented at the The 5th IAPR International Workshop on Document Analysis Systems (DAS 2002), 2002.
- [5] N. Roshanbin and J. Miller, "A Survey and Analysis of Current CAPTCHA Approaches," *Journal of Web Engineering*, vol. 12, pp. 001-040, 2013.
- [6] E. Bursztein, S. Bethard, C. Fabry, J. C. Mitchell, and D. Jurafsky, "How good are humans at solving CAPTCHAs? a large scale evaluation," in *2010 IEEE Symposium on Security and Privacy (SP)*, 2010, pp. 399-413.
- [7] C. Pope and K. Kaur, "Is it human or computer? Defending e-commerce with CAPTCHAs," *IT Professional*, vol. 7, pp. 43-49, 2005.
- [8] M. Blum, L. Von Ahn, J. Langford, and N. Hopper, "The CAPTCHA project, "Completely automatic public Turing test to tell computers and humans apart,"" *Dept. of Computer Science, Carnegie-Mellon University, www.captcha.net*, 2000.
- [9] T. Converse, "CAPTCHA generation as a web service," *Human Interactive Proofs*, vol. 3517, pp. 82-96, 2005.
- [10] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, "reCAPTCHA: Human-based character recognition via web security measures," *Science*, vol. 321, pp. 1465-1468, 2008.
- [11] M. Chew and J. Tygar, "Collaborative filtering CAPTCHAs," *The 2nd International Conference on Human Interactive Proofs (HIP 2005)*, pp. 66-81, May 2005.
- [12] M. Shirali-Shahreza, "Highlighting CAPTCHA," in *2008 Conference on Human System Interactions*, 2008, pp. 247-250.
- [13] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski, "Designing human friendly human interaction proofs (HIPs)," in *ACM Conference on Human Factors in Computing Systems (CHI 05)*, 2005, pp. 711-720.
- [14] M. Chew and H. S. Baird, "BaffleText: A human interactive proof," presented at the 10th Document Recognition & Retrieval Conference (SPIE), 2003.
- [15] G. Kepes, "Language of vision," Chicago: P. Theobald, 1944.
- [16] A. Rusu, A. Thomas, and V. Govindaraju, "Generation and use of handwritten CAPTCHAs," *International Journal on Document Analysis and Recognition*, vol. 13, pp. 49-64, 2010.
- [17] H. S. Baird, M. A. Moll, and S. Y. Wang, "ScatterType: A legible but hard-to-segment CAPTCHA," in *8th International Conference on Document Analysis and Recognition*, 2005, pp. 935-939.
- [18] (2012, Oct. 8). *ebay*. Available: www.ebay.ca
- [19] (2012, Oct. 8). *PHP Class CAPTCHA*. Available: <http://www.nogajski.de/priv/php/captcha/>

- [20] (2012, Jan. 8). *MegaUpload*. Available: www.megaupload.com
- [21] A. Gupta, A. Jain, A. Raj, and A. Jain, "Sequenced tagged CAPTCHA: generation and its analysis," in *IEEE International Advance Computing Conference 2009 (IACC 2009)*, 2009, pp. 1286-1291.
- [22] A. Raj, A. Jain, T. Pahwa, and A. Jain, "Analysis of tagging variants of Sequenced Tagged CAPTCHA (STC)," in *IEEE Toronto International Conference on Science and Technology for Humanity (TIC-STH 2009)*, 2009, pp. 427-432.
- [23] A. O. Thomas, A. Rusu, and V. Govindaraju, "Synthetic handwritten CAPTCHAs," *Pattern Recognition*, vol. 42, pp. 3365-3373, 2009.
- [24] P. Lupkowski and M. Urbanski, "SemCAPTCHA—user-friendly alternative for OCR-based CAPTCHA systems," in *International Multiconference on Computer Science and Information Technology (IMCSIT 2008)*, 2008, pp. 325-329.
- [25] T. Yamamoto, J. Tygar, and M. Nishigaki, "CAPTCHA using strangeness in machine translation," in *The 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, 2010, pp. 430-437.
- [26] R. Bergmair and S. Katzenbeisser, "Towards human interactive proofs in the text-domain: Using the problem of sense-ambiguity for security," presented at the The 7th International Information Security Conference (ISC 2004), 2004.
- [27] A. Desai and P. Patadia, "Drag and Drop: A Better Approach to CAPTCHA," in *Annual IEEE India Conference (INDICON)*, 2009, pp. 1-4.
- [28] P. Golle and N. Ducheneaut, "Keeping bots out of online games," in *The 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology (ACE '05)*, 2005, pp. 262-265.
- [29] H. D. Truong, C. F. Turner, and C. C. Zou, "iCAPTCHA: the next generation of CAPTCHA designed to defend against 3rd party human attacks," in *IEEE International Conference on Communications (ICC)*, 2011, pp. 1-6.
- [30] B. Khan, K. Alghathbar, M. Khan, A. AlKelabi, and A. AlAjaji, "Using Arabic CAPTCHA for Cyber Security," in *Security Technology, Disaster Recovery and Business Continuity*. vol. 122, Springer Berlin Heidelberg, 2010, pp. 8-17.
- [31] M. S. Shahreza, "Verifying Spam SMS by Arabic CAPTCHA," in *2nd IEEE International Conference on Information and Communication Technologies (ICTTA '06)*, 2006, pp. 78-83.
- [32] M. Shirali-Shahreza and S. Shirali-Shahreza, "Collage CAPTCHA," in *9th International Symposium on Signal Processing and Its Applications (ISSPA 2007)*, 2007, pp. 1-4.
- [33] M. H. Shirali-Shahreza and M. Shirali-Shahreza, "Multilingual CAPTCHA," in *5th IEEE International Conference on Computational Cybernetics (ICCC 2007)*, 2007, pp. 135-139.
- [34] J. Elson, J. R. Douceur, J. Howell, and J. Saul, "Asirra: a CAPTCHA that exploits interest-aligned manual image categorization," *14th ACM conference on Computer and Communications Security (CCS 2007)*, pp. 366-374, Oct.-Nov. 2007.
- [35] R. Datta, J. Li, and J. Z. Wang, "IMAGINATION: a robust image-based CAPTCHA generation system," in *13th ACM International Conference on Multimedia (Multimedia 05)*, 2005, pp. 331-334.
- [36] R. Datta, J. Li, and J. Z. Wang, "Exploiting the Human-Machine Gap in Image Recognition for Designing CAPTCHAs," *IEEE Transactions on Information Forensics and Security*, vol. 4, pp. 504-518, Sep 2009.

- [37] E. Vimina and A. U. Areekal, "Telling computers and humans apart automatically using activity recognition," in *IEEE International Conference on Systems, Man and Cybernetics (SMC 2009)*, 2009, pp. 4906-4909.
- [38] H. S. Baird and J. L. Bentley, "Implicit CAPTCHAs," in *SPIE-IS&T Electronic Imaging, Document Recognition and Retrieval*, 2005, pp. 191-196.
- [39] M. Shirali-Shahreza and S. Shirali-Shahreza, "Drawing CAPTCHA," in *28th International Conference on Information Technology Interfaces (ITI 2006)*, Cavtat, Dubrovnik, Croatia, 2006, pp. 475-480.
- [40] A. Karunathilake, B. Balasuriya, and R. Ragel, "User friendly line CAPTCHAs," in *International Conference on Industrial and Information Systems (ICIIS 2009)*, 2009, pp. 210-215.
- [41] Y. Rui and Z. Liu, "ARTiFACIAL: automated reverse Turing test using FACIAL features," *Multimedia Systems*, vol. 9, pp. 493-502, 2004.
- [42] W. H. Liao, "A CAPTCHA mechanism by exchange image blocks," in *18th International Conference on Pattern Recognition (ICPR 2006)*, 2006, pp. 1179-1183.
- [43] H. Gao, D. Yao, H. Liu, X. Liu, and L. Wang, "A Novel Image Based CAPTCHA Using Jigsaw Puzzle," in *13th IEEE International Conference on Computational Science and Engineering (CSE)*, 2010, pp. 351-356.
- [44] M. Banday and N. Shah, "Image flip CAPTCHA," *ISC International Journal of Information Security (ISecure)*, vol. 1, pp. 105-123, 2009.
- [45] R. Gossweiler, M. Kamvar, and S. Baluja, "What's up CAPTCHA?: a CAPTCHA based on image orientation," in *18th International Conference on World Wide Web 2009*, pp. 841-850.
- [46] S. A. Ross, J. A. Halderman, and A. Finkelstein, "Sketcha: a CAPTCHA based on line drawings of 3D models," in *19th International Conference on World Wide Web*, 2010, pp. 821-830.
- [47] J. W. Kim, W. K. Chung, and H. G. Cho, "A new image-based CAPTCHA using the orientation of the polygonally cropped sub-images," *The Visual Computer*, vol. 26, pp. 1135-1143, 2010.
- [48] M. E. Hoque, D. J. Russomanno, and M. Yeasin, "2D CAPTCHAs from 3D models," in *IEEE SoutheastCon 2006*, 2005, pp. 165-170.
- [49] (Jan. 1, 2012). *Spamfizzle CAPTCHA*. Available: <http://spamfizzle.com/CAPTCHA.aspx>
- [50] M. Imsamai and S. Phimoltares, "3D CAPTCHA: A next generation of the CAPTCHA," in *International Conference on Information Science and Applications (ICISA)*, 2010, pp. 1-8.
- [51] E. Bursztein, R. Beauxis, H. Paskov, D. Perito, C. Fabry, and J. Mitchell, "The Failure of Noise-Based Non-continuous Audio CAPTCHAs," in *2011 IEEE Symposium on Security and Privacy (SP)*, 2011, pp. 19-31.
- [52] S. Shirali-Shahreza, H. Abolhassani, H. Sameti, and M. H. Shirali-Shahreza, "Spoken CAPTCHA: A CAPTCHA system for blind users," in *ISECS International Colloquium on Computing, Communication, Control, and Management (CCCM 2009)*, 2009, pp. 221-224.
- [53] T. Y. Chan, "Using a test-to-speech synthesizer to generate a reverse Turing test," in *IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2003)*, 2003, pp. 226-232.
- [54] G. Sauer, H. Hochheiser, J. Feng, and J. Lazar, "Towards a universally usable CAPTCHA," in *4th Symposium On Usable Privacy and Security (SOUPS '08)*, Pittsburgh, 2008.
- [55] G. Kochanski, D. Lopresti, and C. Shih, "A reverse Turing test using speech," in *7th International Conference on Spoken Language Processing*, 2002, pp. 1357-1360.
- [56] (2010, October 8, 2012). *NUCAPTCHA*. Available: <http://www.nucaptcha.com/>

- [57] (October 8, 2012). *HelloCAPTCHA*. Available: <http://www.hellocaptcha.com/>
- [58] E. Athanasopoulos and S. Antonatos, "Enhanced CAPTCHAs: Using animation to tell humans and computers apart," in *10th International Conference on Communications and Multimedia Security (CMS 2006)*, 2006, pp. 97-108.
- [59] J. S. Cui, J. T. Mei, W. Z. Zhang, X. Wang, and D. Zhang, "A CAPTCHA implementation based on moving objects recognition problem," in *International Conference on E-Business and E-Government (ICEE)*, 2010, pp. 1277-1280.
- [60] M. Shirali-Shahreza and S. Shirali-Shahreza, "Dynamic CAPTCHA," in *International Symposium on Communications and Information Technologies (ISCIT)*, 2008, pp. 436-440.
- [61] O. Longe, A. Robert, and U. Onwudebelu, "Checking Internet masquerading using multiple CAPTCHA challenge-response systems," in *The 2nd International Conference on Adaptive Science & Technology (ICAST 2009)*, 2009, pp. 244-249.
- [62] M. Shirali-Shahreza and S. Shirali-Shahreza, "Question-based CAPTCHA," in *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*, 2007, pp. 54-58.
- [63] A. Rusu, R. Docimo, and A. Rusu, "Leveraging cognitive factors in securing WWW with CAPTCHA," in *The 2010 USENIX conference on Web application development (WebApps'10)*, 2010.
- [64] R. Lin, S. Y. Huang, G. B. Bell, and Y. K. Lee, "A new CAPTCHA interface design for mobile devices," in *Australasian User Interface Conference, Australasian Computer Science Week (ACSW2011)*, 2011.
- [65] M. H. Shirali-Shahreza and M. Shirali-Shahreza, "Localized CAPTCHA for illiterate people," in *International Conference on Intelligent and Advanced Systems (ICIAS)*, 2007, pp. 675-679.
- [66] J. Holman, J. Lazar, J. H. Feng, and J. D'Arcy, "Developing usable CAPTCHAs for blind users," in *9th international ACM SIGACCESS conference on Computers and accessibility*, 2007, pp. 245-246.
- [67] M. Shirali-Shahreza and S. Shirali-Shahreza, "CAPTCHA for blind people," in *7th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2007)*, 2007, pp. 995-998.
- [68] S. Shirali-Shahreza and M. Shirali-Shahreza, "A new human interactive proofs system for deaf persons," in *5th International Conference on Information Technology: New Generations (ITNG 2008)*, 2008, pp. 807-810.
- [69] J. Yan and A. S. El Ahmad, "A Low-cost Attack on a Microsoft CAPTCHA," in *15th ACM Conference on Computer and Communications Security (CCS 08)*, 2008, pp. 543-554.
- [70] J. Yan and A. S. El Ahmad, "Breaking visual CAPTCHAs with naive pattern recognition algorithms," in *The 23rd Annual Computer Security Applications Conference (ACSAC 07)*, 2007, pp. 279-291.
- [71] S. Li, S. Shah, M. Khan, S. A. Khayam, A. R. Sadeghi, and R. Schmitz, "Breaking e-banking CAPTCHAs," 2010, pp. 171-180.
- [72] K. Chellapilla, Simard, P., "Using machine learning to break visual human interaction proofs (HIPs)," *Advances in Neural Information Processing Systems*, vol. 17, pp. 265-272, 2004.
- [73] C. Cruz-Perez, O. Starostenko, F. Uceda-Ponga, V. Alarcon-Aquino, and L. Reyes-Cabrera, "Breaking reCAPTCHAs with Unpredictable Collapse: Heuristic Character Segmentation and Recognition," in *Pattern Recognition*. vol. 7329, Springer Berlin Heidelberg, 2012, pp. 155-165.

- [74] B. Milde, "On the Security of reCAPTCHA," Bachelor, Department of Computer Science, Darmstadt University of Technology, 2010.
- [75] V. Nguyen, Y. W. Chow, and W. Susilo, "Breaking an Animated CAPTCHA Scheme," in *The 10th International Conference on Applied Cryptography and Network Security (ACNS'12)*, 2012, pp. 12-29.
- [76] J. Yan and A. S. El Ahmad, "CAPTCHA Security: A Case Study," *IEEE Security and Privacy*, vol. 7, pp. 22-28, Jul-Aug 2009.
- [77] A. Chandavale and A. Sapkal, "A New Approach towards Segmentation for Breaking CAPTCHA," in *Recent Trends in Computer Networks and Distributed Systems Security*. vol. 335, Springer Berlin Heidelberg, 2012, pp. 323-335.
- [78] G. Haichang, W. Wei, and F. Ye, "Divide and Conquer: An Efficient Attack on Yahoo! CAPTCHA," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, 2012, pp. 9-16.
- [79] S. Y. Huang, Y. K. Lee, G. Bell, and Z. Ou, "An efficient segmentation algorithm for CAPTCHAs with line cluttering and character warping," *Multimedia Tools and Applications*, vol. 48, pp. 267-289, 2010.
- [80] V. Nguyen, Y.-W. Chow, and W. Susilo, "Breaking a 3D-Based CAPTCHA Scheme," in *Information Security and Cryptology - ICISC 2011*. vol. 7259, Springer Berlin Heidelberg, 2012, pp. 391-405.
- [81] B. B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, *et al.*, "Attacks and design of image recognition CAPTCHAs," in *The 17th ACM conference on Computer and communications security (CCS '10)*, 2010, pp. 187-200.
- [82] E. Bursztein. (2012, October 8). *How we broke the NuCaptcha video scheme and what we propose to fix it*. Available: <http://elie.im/blog/security/how-we-broke-the-nucaptcha-video-scheme-and-what-we-propose-to-fix-it/#.T-tDK7VfGIA>
- [83] J. Yan and A. S. El Ahmad, "Usability of CAPTCHAs or usability issues in CAPTCHA design," in *The 4th symposium on Usable privacy and security (SOUPS)*, 2008, pp. 44-52.
- [84] V. Nguyen, Y.-W. Chow, and W. Susilo, "Attacking Animated CAPTCHAs via Character Extraction," in *Cryptology and Network Security*. vol. 7712, Springer Berlin Heidelberg, 2012, pp. 98-113.
- [85] C. W. Lin, Y. H. Chen, and L. G. Chen, "Bio-Inspired Unified Model of Visual Segmentation System for Captcha Character Recognition," *2008 IEEE Workshop on Signal Processing Systems: Sips 2008, Proceedings*, pp. 158-163, 2008.
- [86] P. Golle, "Machine learning attacks against the Asirra CAPTCHA," in *The 15th ACM conference on Computer and communications security (CCS 2008)*, 2008, pp. 535-542.
- [87] G. Mori and J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2003, pp. 134-141.
- [88] G. Moy, N. Jones, C. Harkless, and R. Potter, "Distortion estimation techniques in solving visual CAPTCHAs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2004, pp. 23-28.
- [89] E. Bursztein and S. Bethard, "Decaptcha: breaking 75% of eBay audio CAPTCHAs," in *The 3rd USENIX conference on Offensive technologies (WOOT'09)*, 2009.
- [90] J. Wilkins. (2009, Oct. 8, 2012). *Strong CAPTCHA guidelines*. Available: <http://bitland.net/captcha.pdf>

- [91] R. Beede, "Analysis of reCAPTCHA effectiveness," University of Colorado at Boulder Dec. 2010.
- [92] D. Lorenzi, J. Vaidya, E. Uzun, S. Sural, and V. Atluri, "Attacking Image Based CAPTCHAs Using Image Recognition Techniques," in *Information Systems Security*, Springer, 2012, pp. 327-342.
- [93] Y. Nakaguro, M. N. Dailey, S. Marukatat, and S. S. Makhanov, "Defeating Line-Noise CAPTCHAs with Multiple Quadratic Snakes," *Computers & Security*, 2013.
- [94] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnaud, and V. Shet, "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks," *arXiv preprint arXiv:1312.6082*, 2013.
- [95] J. Yan and A. S. El Ahmad, "CAPTCHA Robustness: A Security Engineering Perspective," *Computer*, vol. 44, pp. 54-60, Feb 2011.
- [96] L. Kang and J. Xiang, "CAPTCHA phishing: a practical attack on human interaction proofing," in *The 5th International Conference on Information security and cryptology (Inscrypt)*, 2010, pp. 411-425.
- [97] (October 8, 2012). *Adobe Flash*. Available: <http://get.adobe.com/flashplayer/>
- [98] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski, "Building segmentation based human-friendly human interaction proofs (HIPs)," presented at the The 2nd International Workshop on Human Interactive Proofs (HIP 2005), 2005.
- [99] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski, "Computers beat humans at single character recognition in reading based human interaction proofs (HIPs)," in *The 2nd Conference on Email and Anti-Spam*, 2005.
- [100] J. Bentley and C. Mallows, "CAPTCHA challenge strings: Problems and improvements," in *The 18th SPIE-IS&T Electronic Imaging, Document Recognition and Retrieval*, 2006.
- [101] L. Von Ahn and L. Dabbish, "Labeling images with a computer game," in *The SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*, 2004, pp. 319-326.
- [102] S. K. Chaudhari, A. R. Deshpande, S. B. Bendale, and R. V. Kotian, "3D drag-n-drop CAPTCHA enhanced security through CAPTCHA," in *The International Conference and Workshop on Emerging Trends in Technology*, Mumbai, Maharashtra, India, 2011, pp. 598-601.
- [103] J. P. Bigham and A. C. Cavender, "Evaluating existing audio CAPTCHAs and an interface optimized for non-visual use," in *The SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*, 2009, pp. 1829-1838.
- [104] B. R. Chiswick and P. W. Miller, "Linguistic distance: A quantitative measure of the distance between English and other languages," *Journal of Multilingual and Multicultural Development*, vol. 26, pp. 1-11, 2005.
- [105] M. Tariq Banday and N. Shah, "A Study of CAPTCHAs for Securing Web Services," *IJSDIA International Journal of Secure Digital Information Age*, vol. 1, pp. 66-74, December 2009.
- [106] M. Shirali-Shahreza and S. Shirali-Shahreza, "Motion CAPTCHA," in *Conference on Human System Interactions*, 2008, pp. 1042-1044.
- [107] A. Kolupaev and J. Ogijenko, "CAPTCHAs: Humans vs. bots," *IEEE Security & Privacy*, vol. 6, pp. 68-70, 2008.
- [108] M. Mohamed, N. Sachdeva, M. Georgescu, S. Gao, N. Saxena, C. Zhang, *et al.*, "A three-way investigation of a game-CAPTCHA: automated attacks, relay attacks and usability," in

- Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 195-206.
- [109] T. Jastrzembski, N. Charness, P. Holley, and J. Feddon, "Input devices for web browsing: age and hand effects," *Universal Access in the Information Society*, vol. 4, pp. 39-45, 2005.
- [110] L. D. Catledge and J. E. Pitkow, "Characterizing browsing strategies in the World-Wide Web," *Computer Networks and ISDN systems*, vol. 27, pp. 1065-1073, 1995.
- [111] M. Schrepp and P. Fischer, "A GOMS model for keyboard navigation in web pages and web applications," in *Computers Helping People with Special Needs*, Springer, 2006, pp. 287-294.
- [112] *Wordpress KeyCaptcha*. Available: <https://www.keycaptcha.com/>
- [113] I. F. Ince, Y. B. Salman, M. E. Yildirim, and Y. Tae-Cheon, "Execution Time Prediction for 3D Interactive CAPTCHA by Keystroke Level Model," in *Computer Sciences and Convergence Information Technology, 2009. ICCIT '09. Fourth International Conference on*, 2009, pp. 1057-1061.
- [114] (Feb 20). *PlayThru CAPTCHA*. Available: <http://areyouahuman.com/site-owners/playthru/>
- [115] K. F. Hwang, C. C. Huang, and G. N. You, "A Spelling Based CAPTCHA System by Using Click," in *2012 International Symposium on Biometrics and Security Technologies (ISBAST)*, 2012, pp. 1-8.
- [116] Juraj Rolko. *3D CAPTCHA*. Available: <http://www.3dcaptcha.net/>
- [117] I. F. Ince, I. Yengin, Y. B. Salman, H. G. Cho, and T. C. Yang, "Designing captcha algorithm: splitting and rotating the images against OCRs," 2008, pp. 596-601.
- [118] S. M. Shugan, "The cost of thinking," *Journal of Consumer Research*, pp. 99-111, 1980.
- [119] S. Vikram, Y. Fan, and G. Gu, "SEMAGE: a new image-based two-factor CAPTCHA," 2011, pp. 237-246.
- [120] Y. Wu and Z. Zhao, "Enhancing the Security of On-line Transactions with CAPTCHA Keyboard," in *Information Security and Privacy Research*. vol. 376, Springer Berlin Heidelberg, 2012, pp. 531-536.
- [121] M. Davis. (2010). *Unicode nearing 50% of the web*. Available: <http://googleblog.blogspot.ca/2010/01/unicode-nearing-50-of-web.html>
- [122] U. Consortium. (2012). *Unicode 6.0.0*. Available: <http://www.unicode.org/versions/Unicode6.0.0/>
- [123] M. Prensky, "Fun, play and games: What makes games engaging," in *Digital Game-Based Learning*, St. Paul: Paragon House, 2007.
- [124] U. Consortium. (2012). *Ideographic Variation Database*. Available: http://www.unicode.org/ivd/data/2012-03-02/IVD_Charts.pdf
- [125] Microsoft. *Fonts supplied with Windows 7*. Available: <http://www.microsoft.com/typography/fonts/product.aspx?pid=161>
- [126] C. A. Bigelow and K. Holmes, "The design of a Unicode font," *Electronic Publishing*, vol. 6, pp. 289-305, 1993.
- [127] N. Roshanbin and J. Miller, "Finding Homoglyphs-A Step towards Detecting Unicode-Based Visual Spoofing Attacks," in *Web Information System Engineering–WISE 2011*, Springer, 2011, pp. 1-14.
- [128] *Unicode Font*. Available: http://en.wikipedia.org/wiki/Unicode_font

- [129] R. Cilibrasi and P. M. B. Vitányi, "Clustering by compression," *Information Theory, IEEE Transactions on*, vol. 51, pp. 1523-1545, 2005.
- [130] M. Li and P. M. B. Vitányi, *An introduction to Kolmogorov complexity and its applications*: Springer, 2008.
- [131] T. C. Chen, S. Dick, and J. Miller, "Detecting visually similar Web pages: Application to phishing detection," *ACM Transactions on Internet Technology (TOIT)*, vol. 10, p. 5, 2010.
- [132] J. Mortensen, J. J. Wu, J. Furst, J. Rogers, and D. Raicu, "Effect of image linearization on normalized compression distance," *Signal Processing, Image Processing and Pattern Recognition*, pp. 106-116, 2009.
- [133] N. Tran, "The normalized compression distance and image distinguishability," in *Human Vision and Electronic Imaging XII*, 2007, p. 64921D.
- [134] S. Sarcar, S. Ghosh, P. Saha, and D. Samanta, "Virtual keyboard design: State of the arts and research issues," in *IEEE Students' Technology Symposium (TechSym)*, 2010, pp. 289-299.
- [135] D. G. Pelli and K. A. Tillman, "The uncrowded window of object recognition," *Nature neuroscience*, vol. 11, pp. 1129-1135, 2008.
- [136] H. Pashler, "Target-distractor discriminability in visual search," *Perception & Psychophysics*, vol. 41, pp. 285-292, 1987.
- [137] M. Beck, M. Lohrenz, J. G. Trafton, and M. Gendron, "The role of local and global clutter in visual search," *Journal of Vision*, vol. 8, pp. 1071-1071, 2008.
- [138] L. Williams, "A study of visual search using eye movement recordings," DTIC Document, 1966.
- [139] J. R. Bloomfield, "Experiments in visual search," in *Visual Search*, Washington, D.C.: National Academy of Sciences, 1973, pp. 1-25.
- [140] L. G. Williams, "Studies of extrafoveal discrimination and detection," in *Visual Search*, Washington, D.C.: National Academy of Sciences, 1973, pp. 77-92.
- [141] B. N. S. Vlaskamp, E. A. B. Over, and I. T. C. Hooge, "Saccadic search performance: the effect of element spacing," *Experimental brain research*, vol. 167, pp. 246-259, 2005.
- [142] A. Nazar, "Synthesis & Simulation of Mouse Dynamics," M.A.Sc. thesis, University of Victoria, 2007.
- [143] A. Weiss, A. Ramapanicker, P. Shah, S. Noble, and L. Immohr, "Mouse movements biometric identification: A feasibility study," *Proc. Student/Faculty Research Day CSIS, Pace University, White Plains, NY*, 2007.
- [144] A. A. E. Ahmed and I. Traore, "A new biometric technology based on mouse dynamics," *Dependable and Secure Computing, IEEE Transactions on*, vol. 4, pp. 165-179, 2007.
- [145] A. A. E. Ahmed and I. Traore, "Mouse Dynamics Biometric Technology," in *Behavioral Biometrics for Human Identification: Intelligent Applications*, Medical Information Science Reference, 2010.
- [146] D. Arya, C. Jawahar, C. Bhagvati, T. Patnaik, B. Chaudhuri, G. Lehal, *et al.*, "Experiences of integration and performance testing of multilingual OCR for printed Indian scripts," in *Proceedings of the 2011 joint workshop on multilingual OCR and analytics for noisy unstructured text data*, 2011, p. 9.
- [147] S. S. Kumar, K. Manjusha, and K. P. Soman, "Novel SVD Based Character Recognition Approach for Malayalam Language Script," in *Recent Advances in Intelligent Informatics*. vol. 235, Springer International Publishing, 2014, pp. 435-442.

- [148] S. A. Sattar and S. Shah, "Character recognition of Arabic script languages," in *Proceedings of the International Conference on Computer and Information Technology (ICCIT'12)*, 2012.
- [149] F. Y. Omee, S. S. Himel, M. Bikas, and A. Naser, "A Complete Workflow for Development of Bangla OCR," *arXiv preprint arXiv:1204.1198*, 2012.
- [150] A. S. El Ahmad, J. Yan, and L. Marshall, "The robustness of a new CAPTCHA," in *The third European workshop on system security (Eurosec 2010)*, New York, 2010, pp. 36-41.
- [151] D. G. Lowe, "Object recognition from local scale-invariant features," in *The Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, pp. 1150-1157.
- [152] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, vol. 110, pp. 346-359, 6// 2008.
- [153] W. K. Pratt, *Digital image processing*: John Wiley & Sons, Inc., 1978.
- [154] K. Grauman and B. Leibe, *Visual object recognition*: Morgan & Claypool Publishers, 2011.
- [155] D. G. Lowe, "Object recognition from local scale-invariant features," in *The Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, pp. 1150-1157 vol.2.
- [156] J. Lewis, "Fast normalized cross-correlation," in *Vision interface*, 1995, pp. 120-123.
- [157] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91-110, 2004.
- [158] J. D. McCafferty, *Human and machine vision: computing perceptual organisation*: Ellis Horwood, 1990.
- [159] M. Treiber, "An introduction to object recognition," *An Introduction to Object Recognition: Selected Algorithms for a Wide Variety of Applications, Advances in Pattern Recognition, Volume 0. ISBN 978-1-84996-234-6. Springer-Verlag London Limited, 2010*, vol. 1, 2010.
- [160] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision–ECCV 2006*, Springer, 2006, pp. 430-443.
- [161] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision–ECCV 2010*, Springer, 2010, pp. 778-792.
- [162] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011, pp. 2564-2571.
- [163] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary robust invariant scalable keypoints," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011, pp. 2548-2555.
- [164] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, pp. 510-517.
- [165] Q. Wang, P. Cavanagh, and M. Green, "Familiarity and pop-out in visual search," *Perception & psychophysics*, vol. 56, pp. 495-500, 1994.
- [166] R. S. Goonetilleke, W. C. Lau, and H. M. Shih, "Visual search strategies and eye movements when searching Chinese character screens," *International journal of human-computer studies*, vol. 57, pp. 447-468, 2002.
- [167] M. Pilotti and M. Chodorow, "Does familiarity with text breed complacency or vigilance?," *Journal of Research in Reading*, vol. 35, pp. 204-214, 2012.
- [168] G. Huang and W. Man, "Learning curve: principle, application and limitation," in *2010 International Conference on E-Business and E-Government*, 2010, pp. 1840-1843.

Appendices

Appendix A: Description of Mouse Movement Features

In this appendix, we describe the features that were extracted from users' mouse movement data from the user study described in Section 3.4. The definitions of movement features were given in Section 3.4.2.

1. Number of short movements in a test

Human users frequently exhibit short mouse movements which we define as movements with route lengths shorter than 100 pixels between pauses. Machines do not have the physiological and psychological characteristics that give rise to such redundancies. This feature alone can be used to identify unsuspecting machines who are not emulating such patterns. The median of the number of short movements was 14 (SD = 6.85, minimum = 3, maximum = 34). This variable has an approximately normal distribution. The lack of short movements or consistently exhibiting less than five short movements (i.e. one SD from the mean) can be used as a criterion to flag a user as “potentially a robot”.

2. Overall mouse movement speed

Machines can potentially move the mouse pointer at very high speeds while human users' mouse movement speed is expected to be range-bound. The histogram of the speed of all movements of all users is displayed in Figure 55. The median of these values is 313.57 pixels per second (SD = 398.04, minimum = 24, maximum = 3332).³⁰ Based on these data, it is reasonable

³⁰ The number and size of pixels depend on the screen resolution. Hence, variables that use pixel counts in their calculation have to be converted to their equivalent values for screen resolutions other than the one used in our test. We used a resolution of 1366x768 pixels in our test.

to expect a human's mouse movement speeds to be generally lower than 1500 p.p.s. A user that consistently exhibits high mouse movement speeds can be flagged as a machine.

3. Average speed per movement direction

We considered eight movement directions which are displayed in Figure 58. In this figure, for example, "R" represents all actions performed with angles between -22.5 and 22.5 degrees. Figure 56 shows the median speed of users' mouse movements in these eight directions. An analysis of variance (ANOVA) on these numbers showed significant variation among directions ($F(7,792) = 5.31, p < .05$). We also conducted a Tukey's HSD pairwise comparison test. The results indicated that the "Right (R)" and "Right-Up (RU)" directions had significantly faster speeds than the other directions. In addition, movements in the "Down (D)" and "Left-Down (LD)", and "Right-Down (RD)" directions were significantly slower than the rest of the movements, all at the 5% significance level. These findings can easily help to identify machines, as they do not have the restrictions imposed by humans' wrist on their mouse movements and are likely to exhibit equal speeds in all directions.

4. Length of pauses

Silence time is defined as the time spent by the user performing no mouse actions [145]. We measured the duration of fixed mouse positions. A user might be searching for an object (e.g. a character) on the screen, or contemplating their next action during such pauses. These silence periods are usually less than 5000 milliseconds. Longer silence times were rarely observed in our user data and were removed as outliers.

Figure 57 displays the "lengths of pauses" histogram for all users. According to this figure, most pause periods fall in the 0.5 to 1.5 second range. The median of the time of pauses is 960 milliseconds (SD=906.68, minimum = 504, maximum = 4920).

Human users exhibit a considerable number of short pauses. In addition, the distribution for this variable has a long tail which signifies that human users also have a number of longer pauses. Machines are less likely to exhibit short pauses or very long pauses. Hence, the lack of short pauses and/or a consistent lack of longer pauses can be used as a sign to flag an unsuspecting machine.

5. Number of pauses

The users in our study exhibited between 4 and 60 pauses with an approximately normal distribution (median=23, SD=12.36) while solving a single CAPTCHA test. Machines are not likely to exhibit any or many pauses (i.e. silences longer than 500 ms) in their matching behavior. Hence, the number of pauses can also be used to flag them.

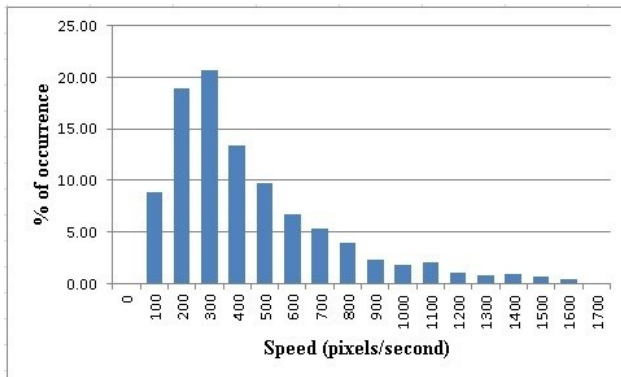


Figure 55: The histogram of movement speed for human users.

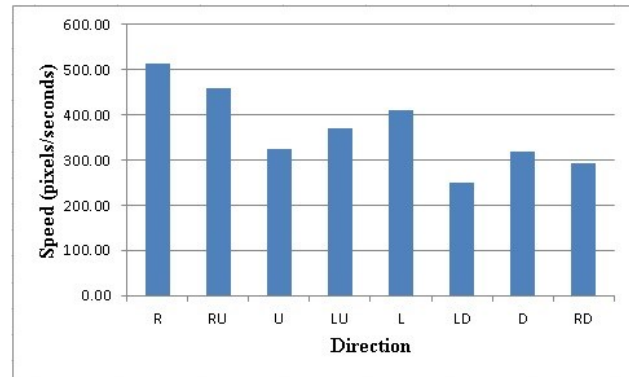


Figure 56: The median speed of movements in each direction.

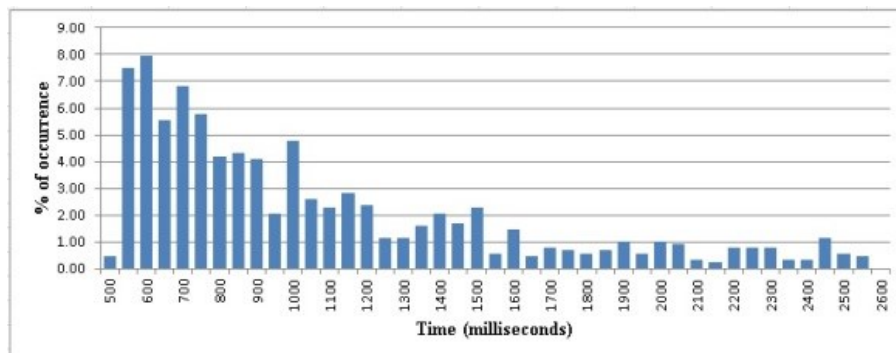


Figure 57: Histogram of length of pauses.

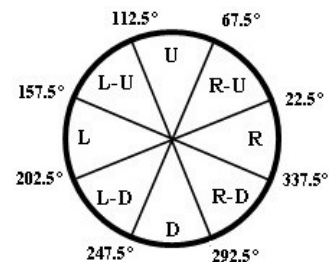


Figure 58: Mouse movement directions.

6. Number and time length of short silences

Computers are precise, while humans are imperfect. Hence, the former are expected to exhibit precise mouse movements whereas the latter's mouse movements are not very precise. Humans make tiny direction changes and tiny pauses in their mouse movements. To study these imperfections, we recorded users' mouse movements with a very low sampling rate of 5ms for the first 30 seconds of every test. We calculated the average number of small silence periods per each 100 pixels of mouse movement for each person. Our users had 2.6 silences (median) in each 100 pixels of mouse movements (minimum³¹ = 0.6, maximum = 9.2, SD=1.41, normally distributed). The time length of small pauses was approximately normally distributed (median=72.5 milliseconds, minimum =35.3, maximum =119.3, SD=17.78). Observing no short silences, an extremely low number of short silences, e.g., short silences below 35 milliseconds can be taken as a sign that a machine is carrying out the operation.

7. Number of direction changes in a movement

A human user's mouse movement usually consists of direction changes even when they are moving in the same general direction. In our study, a user on average changes the direction of movement 6.5 times during a 100-pixel mouse movement (minimum= 2, maximum= 16, SD=2.54, approximately normally distributed). A machine that is not emulating such behavior can be easily flagged as non-human.

8. Direct distance to route length ratio of a movement

Human users don't move their mouse pointer in a straight line when they are moving from point A to point B. In our data, the direct distance of a movement is, on average, 56.47% of its

³¹ Minimum of the "average number of silences for each user".

route length (min = 7.83%; max = 99.78%, SD=27.91). The value of this variable for a machine that makes precise movements would be 100%. Hence, consistently exhibiting high values on this variable, e.g. values above 97.5, can be taken as a sign that a machine is carrying out the operations.

9. Average speed in matching movements

We divided users' movements in their matching attempts into three sections: from the start of dragging to the start of the first barrier ('start'), the movement in the area between the test and the keyboard barriers ('middle') and the movement after the second barrier ('end'). We measured the average movement speed for each of these three sections across users and tests. The results indicate that the middle speed is relatively lower than the start and end speeds which is due to the existence of barriers in the middle area. The end speed is also consistently lower than the start speed which is perhaps due to the higher focus of users to find the exact dropping location in order to release the mouse. These regional speed characteristics can also help identify machines, as machines are not expected to exhibit such variations.

Table 41: Speed in different regions of mouse movements.

	Start speed (Pixels/second)	Middle speed (Pixels/second)	End speed (Pixels/second)
Median	185	134	179
Min	78	37	56
Max	516	351	338
SD	139	101	64