AN ORIENTATION BASED IMAGE CAPTCHA

THESIS

Presented in Partial Fulfillment of the Requirements for
the Degree Master of Science in the
Graduate School of The Ohio State University

By
Yashwanth Chanamolu, B.E
Graduate Program in Computer Science and Engineering

The Ohio State University
2009

Thesis Committee:
Professor Dong Xuan, Advisor
Professor Ten H. Lai
ABSTRACT

CAPTCHA is now almost a standard security technology, and has found widespread application in commercial websites. To date, almost all CAPTCHA designs are labeling based. Labeling based CAPTCHAs refer to those that make judgment based on whether the question “what is it?” has been correctly answered. Essentially in Artificial Intelligence (AI), this means judgment depends on whether the new label provided by the user side matches the label already known to the server. Labeling based CAPTCHA designs have some common weaknesses that can be taken advantage of attackers. First, the label set, i.e., the number of classes, is small and fixed. Due to deformation and noise in CAPTCHAs, the classes have to be further reduced to avoid confusion. Second, clean segmentation in current design, in particular character labeling based CAPTCHAs, is feasible. Sometimes, it is even trivial. The state of the art of CAPTCHA design suggests that the robustness of character labeling schemes should rely on the difficulty of finding where the character is (segmentation), rather than which character it is (recognition). However, the shapes of alphabet letters and numbers have very limited geometry characteristics that can be used by humans to tell them yet are also easy to be indistinct. The major contribution of this thesis is that we propose a new direction in CAPTCHA design by focusing on orientation rather than labeling. We introduce an orientation based image CAPTCHA which uses an image set which gets dynamically updated. By focusing on orientation we eliminate problems facing labeling based CAPTCHAs like limited
label classes and object recognition. We built a CAPTCHA web application using Java Sever Pages technology and used AJAX techniques to make the user interface more interactive and user friendly. We deployed our application on a web server and tabulated the response time and success ratio results we got.
This is dedicated to my parents
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Dong Xuan for the guidance and the intellectual support he has provided me. I am also thankful for his unconditional support that he has given me throughout my M.S study. I would like to thank for Dr. Steve Lai for agreeing to be on my Master’s Examination Committee and for all the valuable suggestions he has made.

I am very thankful to Xiaole Bai for being a great mentor and for providing guidance throughout my thesis. I am also thankful for all the inputs and efforts he put for my thesis.

I would like to express my deepest gratitude to my parents who have shown unconditional love and support at every step in my life. I am especially thankful to my cousin Kalyan for all his moral support and for making some of my harder times a lot easier.
VITA

July 5, 1985......................... Born – Hyderabad, India.
2003-2007......................... B.E. Information Technology
Osmania University
Hyderabad, India
2007-present ..................... Graduate Student at the Department of Computer
Science and Engineering, The Ohio State
University

FIELDS OF STUDY

Major Field: Computer Science and Engineering
TABLE OF CONTENTS

Abstract ............................................................................................................................................. ii
Dedication .......................................................................................................................................... iv
Acknowledgements .......................................................................................................................... v
Vita ..................................................................................................................................................... vi
List of Figures ................................................................................................................................... ix

Chapters:

1. Introduction .................................................................................................................................... 1
   1.1 State of the art of CAPTCHA design ....................................................................................... 1
   1.2 Motivation ................................................................................................................................ 12
   1.3 Our method ............................................................................................................................... 19

2. System Overview ......................................................................................................................... 21
   2.1 Architecture ............................................................................................................................ 21
      2.1.1 Image Collection ................................................................................................................ 22
      2.1.2 Image Preprocessing ......................................................................................................... 23
      2.1.3 Server ............................................................................................................................... 23
      2.1.4 User Interface .................................................................................................................... 24
   2.2 Working .................................................................................................................................... 25
      2.2.1 High Pass Filter .................................................................................................................. 25
      2.2.2 Note paper Filter ............................................................................................................... 27
      2.2.3 Threshold ......................................................................................................................... 28
      2.2.4 Noise Addition ................................................................................................................... 29
      2.2.5 Image Selection ............................................................................................................... 30
3. Design Methodology ........................................................................................................... 31
   3.1 Introduction .................................................................................................................. 31
   3.2 Java Server Pages ....................................................................................................... 33
   3.3 JSP Technology and Java Servlets ........................................................................... 33
   3.4 Use of AJAX ............................................................................................................. 38

4. Results .................................................................................................................................. 40
   4.1 GUI Working .............................................................................................................. 40
   4.2 Usability test results ................................................................................................. 42

5. CONCLUSIONS .................................................................................................................. 44

BIBLIOGRAPHY .................................................................................................................. 46
LIST OF FIGURES

Figure 1. GIMPY User Interface......................................................................................... 4
Figure 2. reCAPTCHA........................................................................................................ 5
Figure 3. Examples of various character labeling CAPTCHAs.......................................... 7
Figure 4. ESP-PIX and Assira CAPTCHA systems ......................................................... 10
Figure 5. Question based CAPTCHA ............................................................................... 11
Figure 6. Audio labeling CAPTHCAS (a) Gmail (b) eBay................................................ 12
Figure 7. Confusing characters in Google's CAPTHCA..................................................... 13
Figure 8. System Architecture........................................................................................... 21
Figure 9. Images which are ambiguous with respect to orientation.................................... 22
Figure 10. User Interface of the CAPTCHA application.................................................. 24
Figure 11. Analog high pass filter..................................................................................... 25
Figure 12. Matrix convolution for a high pass filter .......................................................... 26
Figure 13. Image processing using high pass filter............................................................. 27
Figure 14. Image processing using note paper effect.......................................................... 28
Figure 15. Image processing using threshold..................................................................... 29
Figure 16. Applying noise to the image............................................................................. 29
Figure 17. Process flow in Image selection....................................................................... 30
Figure 18. Lifecycle of a servlet...................................................................................... 36
Figure 19. Request/Response flow when calling a jsp

Figure 20. GUI step 1

Figure 21. GUI step 2

Figure 22. GUI step 3

Figure 23. GUI step 4
CHAPTER 1

INTRODUCTION

1.1 State of the Art of CAPTCHA Design

CAPTCHA stands for “Completely Automated Public Turing Test to Tell Computers and Humans Apart” [1]. It is also known as human interaction proofs (HIPs). CAPTCHA systems have been widely adopted nowadays on the internet for protecting free online services for humans from abuse by automated scripts/bots. However, its design is far from trivial since challenges designed are to be easily solved by humans, while remaining too hard for computers to solve.

Identifying humanity is a challenging and long lasting task. It dates back from the beginnings of modern computer science when A. Turing presented his theories on the possibility of thinking machines in his famous article “Computing Machinery and Intelligence” [2]. In the original Turing Test, a human judge was allowed to ask a series of questions to two players, one of which was a computer and the other a human. Both players pretended to be human, and the judge had to distinguish between them. CAPTCHAs are similar to Turing Test in that they distinguish humans from computers, but they differ in that the judge is now a computer. Although the goal of the original Turing Test was to serve as measure of progress for Artificial Intelligence—a computer
would be said to be intelligent if it passed the Turing Test—making the judge be a computer allows CAPTCHAs to be useful for other practical applications:

- **Free Email Services**: Several companies (Google, Yahoo!, Microsoft, etc.) offer free email services, most of which previously suffered from a specific type of attack: “bots” that signed up for thousands of email accounts every minute. This situation has been improved by requiring users to prove they are human before they can get a free email account.

- **Search Engine Bots**: Some web sites don’t want to be indexed by search engines. There is an html tag to prevent search engine bots from reading web pages, but the tag doesn’t guarantee that bots won’t read the pages. Search engine bots, since they usually belong to large companies, respect web pages that don’t want to allow them in. However, in order to truly guarantee that bots won’t enter a web site CAPTCHAs are needed.

- **Worms and Spam**: CAPTCHAs also offer a plausible solution against email worms and spam: only accept an email if you know there is a human behind the other computer.

- **Preventing Dictionary Attacks**: Pinkas and Sander [25] have suggested using CAPTCHAs to prevent dictionary attacks in password systems. The idea is simple: prevent a computer from being to iterate through the entire space of passwords by requiring a human to type the passwords.

- **Online Polls**: In November 1999, slashdot.com released an online poll asking which the best graduate school in computer science was. IP addresses of voters
were recorded in order to prevent single users from voting more than once. However, students at Carnegie Mellon found a way to stuff the ballots by using programs that voted for CMU thousands of times. CMU’s score started growing rapidly. The next day, students at MIT wrote their own voting program and the poll became a contest between voting “bots.”

In 1996, based on Turings ideas, M. Naor proposed a theoretical framework that would serve as the first approach in testing humanity automatically [3]. Later, L. Ahn et al. developed a more complete and thorough work on the subject of automated Turing Tests, which is successful formalization and substantiation of Naors conceptual model [1]. To date, almost all CAPTCHA designs are labeling based. Labeling based CAPTCHAs refer to those that make judgment based on whether the question “what is it?” has been correctly answered. Essentially in Artificial Intelligence (AI), this means judgment depends on whether the new label provided by the user side matches the label already known to the server. There are three major categories among them:

- Character labeling
- Image labeling
- Audio labeling and Others

In character labeling based CAPTCHA designs, the computer renders a sequence of letters after distorting them and adding noise. The user is asked to tell what characters they are in order, and will pass the test if the characters typed (new labels) match exactly those known to the server (known labels). Character labeling CAPTCHAs are the most widely used CAPTCHAs. The popularity of such schemes is due to the fact that they
have many advantages [28], for example, being intuitive to users world-wide (the user task performed being just character recognition), having little localization issues (people in different countries all recognize Roman characters), and of good potential to provide strong security (e.g. the space a brute force attack has to search can be huge, if the scheme is properly designed).

![GIMPY User Interface](image.png)

Figure 1. GIMPY User Interface

Gimpy [26] is one of the many CAPTCHAs based on the difficulty of reading distorted text. It was originally built for (and in collaboration with) Yahoo! to keep bots out of their chat rooms, to prevent scripts from obtaining an excessive number of their e-mail addresses, and to prevent computer programs from publishing classified ads. Gimpy is based on the human ability to read extremely distorted and corrupted text, and the inability of current computer programs to do the same. Gimpy works by choosing a certain number of words from a dictionary, and then displaying them corrupted and
distorted in an image; after that Gimpy asks the user to type the words displayed in that image. While human users have no problem typing the words displayed, bots are simply unable to do the same. Figure 1 shows an example of Gimpy.

Another interesting character labeling based CAPTCHA is reCAPTCHA [33]. It is a free CAPTCHA service that helps to digitize books, newspapers and old time radio shows. Their main idea is to channel the effort spent solving CAPTCHAs online into "reading" books. About 200 million CAPTCHAs are solved by humans around the world every day. In each case, roughly ten seconds of human time are being spent. In aggregate CAPTCHAs consume more than 150,000 hours of work each day.

![Figure 2. reCAPTCHA](image)

To archive human knowledge and to make information more accessible to the world, multiple projects are currently digitizing physical books that were written before the computer age. The book pages are being photographically scanned, and then transformed into text using "Optical Character Recognition" (OCR). The transformation
into text is useful because scanning a book produces images, which are difficult to store on small devices, expensive to download, and cannot be searched. The problem is that OCR is not perfect. reCAPTCHA improves the process of digitizing books by sending words that cannot be read by computers to the Web in the form of CAPTCHAs for humans to decipher. More specifically, each word that cannot be read correctly by OCR is placed on an image and used as a CAPTCHA. Each new word that cannot be read correctly by OCR is given to a user in conjunction with another word for which the answer is already known. The user is then asked to read both words. If they solve the one for which the answer is known, the system assumes their answer is correct for the new one. The system then gives the new image to a number of other people to determine, with higher confidence, whether the original answer was correct. Figure 3 presents character based CAPTCHAs that can be sampled from the web while signing up for free e-mail accounts with Mailblocks (www.mailblocks.com), MSN/Hotmail (www.hotmail.com), Yahoo! (www.yahoo.com), Google (gmail.google.com), running a whois query at Register.com (www.register.com) or searching for tickets at Ticketmaster (www.ticketmaster.com), etc.
<table>
<thead>
<tr>
<th>Mail blocks</th>
<th>MSN/Hotmail (after May 2004)</th>
<th>Register.com (late 2004)</th>
<th>Yahoo!/EZ-Gimpy (after Aug’04)</th>
<th>Ticketmaster</th>
<th>Google</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Figure 3. Examples of various character labeling CAPTCHAs
Solutions to Yahoo (ver1) CAPTCHAs are common English words, but those for ticketmaster and Google do not necessarily belong to the English dictionary. They appear to have been created using a phonetic generator. Examining the changes in MSN, Yahoo!, and Register.com HIPs, it can be noted that these CAPTCHAs are becoming progressively more difficult. While MSN introduced more arcs as clutter, Yahoo! gave up their language model and replaced simple textures and grids with more random intersecting lines and arcs. Register.com’s update was relatively minor as they simply introduced digits into their character set.

In [11] Chellapilla et al. have discussed the various issues when designing a character labeling based CAPTCHA. They can be summarized as follows:

a) **Character set**: The character set to be used in the CAPTCHA.

b) **Affine transformations**: Translation, rotation, and scaling of characters

c) **Adversarial clutter**: Random arcs, lines, or other simple geometric shapes that intersect with the characters and themselves

d) **Image warp**: elastic deformations of the CAPTCHA Image at different scales i.e., those that stretch and bend the character itself (global warp) and those that simply jiggle the character pixels (local warp)

e) **Background and foreground textures**: These textures are used to form a colored CAPTCHA image from a bi-level or grayscale CAPTCHA mask generated by using a) through d)

f) **Language model**: the language model determines both the conditional and joint probabilities of character co-occurrence in the CAPTCHA. A CAPTCHA can use a) no language model (random equally likely occurrence of all possible combinations – Eg.
Mailblocks, MSN, Register and Yahoo version 2), b) words from a dictionary (Yahoo! version 1), or c) a phonetic generator (Ticketmaster and Google/Gmail).

Each of these choices affects both CAPTCHA security and human-friendliness of the CAPTCHA though commonly to different degrees.

Since image recognition is in general a harder problem than character recognition, people turn to labeled images for CAPTCHA design. M. Chew and J. Tygar were among the pioneers to design image labeling based CAPTCHAs by using labeled photographs [4]. In their system, for each test computer randomly selects six images with the same label that has been pre-set using some common words, and then send them to the user asking for the label. In 2004, L. Ahn et al. designed a new inspiring image labeling focused CAPTCHA, ESP-PIX [5]. PIX has a large database of labeled images each associated with one of 72 concrete object (a bag, a tool, etc). The program presents 4 random images of one object from its database to users and then asks the question “what object are these pictures of?” The user can make selection from a pull-down menu. Its GUI is shown in Fig. 5. Many amateur users of the phpBB forum software (which has suffered greatly from spam) have implemented an open source image recognition CAPTCHA system in the form of an addon called KittenAuth [34] which in its default form presents a question requiring the user to select a stated type of animal from an array of thumbnail images of assorted animals. The images (and the challenge questions) can be customized, for example to present questions and images which would be easily answered by the forum's target user base. In [6], J. Elson et al. formally introduced Asirra (Asirra stands for Animal Species Image Recognition for Restricting Access”), a cat or dog labeling based CAPTCHA design [7]. The beauty of Asirra lies on its cleverly using
the database from Petfinder.com. It asks the user to pick images of “cat” among 12 random chosen pictures. Asirra’s GUI is illustrated in Fig. 4. Audio labeling based CAPTCHA designs were designed as subsidiary services for the blind. We note that there have been some efforts trying to open new CAPTCHA design direction where labeling is abandoned. These works are in the domain of natural language processing [8][9].

Exploiting linguistic cognition aspects of humans, computer asks the user to answer some questions or tell which sentences form a joke. Research towards this direction is preliminary. The well recognized hardness lies on constructing these problems and evaluating their security.

A type of question based CAPTCHA was introduced [35] in which a simple mathematical problem is generated according to a predefined pattern but instead of some
object's name their images are used. Then the whole problem is saved and shown to the user inform of an image to be answered by him. But since answering this problem requires four abilities of understanding text of question, detection of question images, understanding the problem, and solving the problem, only a human user can answer this question. Figure 5 illustrates the GUI of Question based CAPTCHA.

Figure 5. Question based CAPTCHA

Another category of CAPTCHAs called Collaborative filtering CAPTCHAs was proposed by Monica Chew and J.D. Tygar[36] which allowed the CAPTCHA to ask questions that have no absolute answer. Instead the CAPTCHA is graded by comparison to other people’s answers. By observing real-world trends made by human subjects, collaborative filtering CAPTCHAs attempt to extract complex patterns that reflect human choices. For example, humans who like a particular joke, such as a subtle pun, may also enjoy other jokes that incorporate similar patterns of whimsy, word-play, and ironic observation. Research in this direction is still in the beginning stages and these kinds of CAPTCHAs are not ready to be used now.
1.2 Motivation

Labeling based CAPTCHA designs have some common weaknesses that can be taken advantage of attackers. First, the label set, i.e., the number of classes, is small and fixed. (Note that labeling is essentially a classification procedure.) All labeling based CAPTCHA designs require that the label set should be known to the user. Users have to tell exactly “what is it?” since label matching is strict. That is the reason why the set of alphabet letters and numbers is a popular choice. Note that the number of classes for alphabet letters and numbers are very limited. Due to deformation and noise in CAPTCHAs, the classes have to be further reduced to avoid confusion. Distortion often creates ambiguous characters, where users cannot be sure what they are. Although some characters have very different shapes, after distortion, they become hard to tell apart from each other. This problem is common in most character based schemes. We list common confusing character pairs as follows.

- **Letter vs digits**: hard to tell distorted O from 0, 6 from G and b, 5 from S/s, 2 from Z/z, 1 from l.
Figure 7. Confusing characters in Google's CAPTCHA

- **Digit vs digit**: 5 is hard to tell apart from 6, 7 is written differently in different countries and often what looks like a 7 may in fact be a 1, and 8 can look like 6 or 9.
- **Letter vs letters**: Under some distortion, “vv” can resemble “w”; “cl” can resemble “d”; “nn” can could resemble “m”; “rn” can resemble “m” ; “rm” can resemble “nn”; “cm” can resemble “an”. Table 1 shows some such confusing examples that we observed in the Google CAPTCHA (used for its Gmail service).
• **Characters vs clutters:** In CAPTCHAs such as the MSN schemes, random arcs are introduced as clutters. Confusion between arcs and characters is often observed in this Microsoft scheme. Figure 7 illustrates a list of cases where the letters in the google CAPTCHA are confusing.

Another issue with character based CAPTCHAs is localization. How friendly are character based CAPTCHAs to foreigners. Luis von Ahn in his world-wide deployed reCAPTCHA system [26] has observed an average success rate of around 97% and 93% for passing reCAPTCHA tests in daytime and at night (both US time), respectively. According to IP addresses of service requests that reCAPTCHA has received, more users from outside of the US (e.g. those in Asia) access this service at night than in the daytime (both US time) – typically evening time in the US is daytime in Asia. This suggests to some extent that people with different first languages do perform differently in decoding distorted Roman characters.

Images can have theoretically infinite classes. However, the classes used still cannot be too many and must be static, otherwise usability will be compromised. In PIX, the class number is 72, while in Asirra the number is 2. Another problem with image based CAPTCHAs is that the image database is always static. The strong point of Assira is the large database of images they have. This was achieved by collaborating with Petfinder which provided them with over three million images of cats and dogs. But however large the image set is it would be static and all the images in the database could be labeled in time. Consider the case for where CAPTCHA sweatshops are used for solving Assira. If the database of images can be downloaded and paying the operators
0.01$ for each image, the whole database of images in Assira could be solved for 30,000$.

Secondly, clean segmentation in current design, in particular character labeling based CAPTCHAs, is feasible. Sometimes, it is even trivial. The state of the art of CAPTCHA design suggests that the robustness of character labeling schemes should rely on the difficulty of finding where the character is (segmentation), rather than which character it is (recognition) [10][11][12][13]. However, the shapes of alphabet letters and numbers have very limited geometry characteristics that can be used by humans to tell them yet are also easy to be indistinct. Hence, limited segmentation proof techniques can be used for character labeling based designs, where segmentation proof is now achieved mainly by connecting characters or adding noise carefully crossing multiple characters. Interestingly, segmentation proof techniques are not popular among image labeling based designs. These weaknesses are critical. The robustness of character-based CAPTCHA has initially been studied mainly just in the computer vision and document analysis and recognition communities. For example, Mori and Malik [29] have broken the EZ-Gimpy (92% success) and the Gimpy (33% success) CAPTCHAs with sophisticated object recognition algorithms. Moy et al [30] developed distortion estimation techniques to break EZ-Gimpy with a success rate of 99% and 4-letter Gimpy-r with a success rate of 78%. Chellapilla and Simard [10] attacked a number of visual CAPTCHAs taken from the web with machine learning algorithms, achieving a success rate from 4.89% to 66.2%. J Yan and A S El Ahmad [31] have broken a number of CAPTCHAs (including those hosted at Captchaservice.org, a web service specialized for CAPTCHA generation) with almost 100% success by simply counting the number of pixels of each segmented
character, although these schemes were all resistant to the best OCR software on the market. This is one of the few examining the robustness of CAPTCHA from the security angle. In contrast to other work that relied on sophisticated computer vision or machine learning algorithms, they used only simple pattern recognition algorithms but exploited fatal design errors that they discovered in each scheme. PWNtcha [32] is an excellent web page that aims to “demonstrate the inefficiency of many CAPTCHA implementations”. It comments briefly on the weaknesses of a dozen visual CAPTCHAs. These schemes were claimed to be broken with a success rate ranging from 49% to 100%. However, no technical detail was publicly available.

In January 2008, Yahoo’s CAPTCHA security had been reportedly broken [14]. In February, it was reported that Google’s CAPTCHA had been cracked by spammers [15]. In May, Microsoft’s CAPTCHA security had been broken [16] and the attack later was published in [17]. Microsoft CAPTCHA, a scheme that is designed to be segmentation resistant, was designed by an interdisciplinary team of diverse expertise in Microsoft including document processing and understanding, machine learning, HCI and security. The Microsoft CAPTCHA has been deployed in many of their online services including Hotmail, MSN and Windows Live for years. Its first version was deployed in Hotmail’s user registration system in 2002, and ever since the scheme has undergone extensive improvement in terms of both robustness and usability. Microsoft has also filed three US patent applications to protect the underlying technology. J Yan and A S El Ahmad [17] have implemented a low-cost segmentation attack that has achieved a success rate of higher than 90% on the latest version of this Microsoft CAPTCHA. They have shown that the Microsoft scheme can be broken with an overall (segmentation and
then recognition) success rate of about 60%. In contrast, its design goal was that “automatic scripts should not be more successful than 1 in 10,000” attempts (i.e., a success rate of 0.01%). In fact, although the Microsoft scheme was believed to be “extremely difficult and expensive for computers to solve” because of the difficulty of segmentation that its designers introduced, it takes only slightly more than 80 ms in average for their attack to completely segment a challenge on a desktop computer with a 1.86 GHz Intel Core 2 CPU and 2 GB RAM. There are many other broken character labeling CAPTCHA systems, e.g., one Chinese hacker’s page [18] gives a list of them and sells some attack software. In these cases, segmentation is always the first breach. In [17], J. Yan and A. Ahmad shows the most efforts are for the segmentation purpose. After being segmented, labeling will achieve desirable accuracy due to small number of classes and rich public training data [19].

For image labeling based designs, recognition is relatively easy since only limited classes are used and no segmentation proof techniques are adopted. Recently, it was reported in [20] that Asirra can be broken. The Asirra CAPTCHA [7] relies on the problem of distinguishing images of cats and dogs. An Asirra challenge consists of 12 images, each of which is of either a cat or a dog. To solve the CAPTCHA, the user must select all the cat images, and none of the dog images. According to [6], Asirra can be solved by humans 99.6% of the time in under 30 seconds". The usability of Asirra is a significant advantage compared to CAPTCHAs based on recognizing distorted strings of letters and numbers. The security of Asirra is based on the presumed difficulty of classifying images of cats and dogs automatically. As reported in [6], evidence from the 2006 PASCAL Visual Object Classes Challenge suggests that cats and dogs are
particularly difficult to tell apart algorithmically. A classifier based on color features, described in [6], is only 56.9% accurate. The authors of [6] conjecture that based on a survey of machine vision literature and vision experts at Microsoft Research; they believe classification accuracy of better than 60% will be difficult without a significant advance in the state of the art. With a 60% accurate classifier, the probability of solving a 12-image Asirra challenge is only about 0.2%. Philippe Colle [20] in his paper described a classifier which is 82.7% accurate in telling apart the images of cats and dogs used in Asirra. This classifier allows us to solve a 12-image Asirra challenge with probability 10.3%. This success probability is significantly higher than the 0.2% estimate given in [6] for machine vision attacks. The classifier is a combination of two support-vector machine (SVM) classifiers trained on color and texture features of images. The classifier is entirely automatic, and requires no manual input other than the one-time labeling of training images. Using 15,760 color features, and 5,000 texture features per image, the classifier is 82.7% accurate. The classifier was trained on a commodity PC, using 13,000 labeled images of cats and dogs downloaded from the Asirra website [7]. Note even audio labeling based designs also suffer the fate of being broken due to same weaknesses [21][22].

Another trend that was reported was that CAPTCHA breaking groups combining cheap labor with the Internet's capacity for making quick, anonymous global transactions. CAPTCHA solvers work piecemeal in Internet cafes or in sweatshops full of Internet-connected PCs in developing countries. They work long hours, solving CAPTCHAAs sent by an unseen coordinator. Initially a code is written which fills out a form like registering for an email account. When the program gets to the CAPTCHA it would relay it to the
operators to solve it. The solvers would be paid 2.50$ per hour and at an average rate they solve about 720 CAPTCHAs. Given this each CAPTCHA would just cost 1/3 of a cent.

1.3 Our Method

The major contribution of this thesis is that, we have shown an image based and orientation based CAPTCHA design. The images we used are directly from Google’s search engine. Thus our image set is huge and gets dynamically updated every time. By emphasizing orientation instead of exactly labeling, the user does not need to know explicitly the label set. Hence, the size of label set used can be theoretically infinite and dynamically updated. Since the user now focuses on object orientation, there is no need to tell exactly what the object is. Thus, more segmentation proof techniques can be used without worrying too much about recognition accuracy.

Humans know the correct orientation of a shape through daily experience with objects of that shape. It is through previous interaction that we know, for example, that the correct orientation for a horse is one in which the legs are down, instead of up. Such information is difficult, if not impossible, to compute from the shape itself [23]. E. Cohen and M. Singh in [24] point out that, the essential cause for such difficulty lies in the fact that homogeneous computation where all points are treated equally cannot guarantee to produce results similar to human perception. Their paper [24] exhibited large and systematic deviations of perceived orientation from the principal axis of the shape obtained by homogeneous computation. These results thus argue against the homogenous computation hypothesis for automatic orientation. The human visual system does not
treat all points within a shape uniformly in computing its orientation. Rather, it explicitly takes into account the structural decomposition of the shape into parts and treats points within the two parts differently. Hence, in any image where there is no way to do the decomposition and weighing (which requires common sense), computing orientation which is consistent with human perception is very difficult.

Introducing the concept of orientation into CAPTCHA design helps overcome two well known difficulties when designing image labeling CAPTCHA systems. First since no exact labels are needed from the user, the class number of objects that can be used for orientation is theoretically infinite and can be huge in practice. The classes have no need to be static. And they can be dynamically updated even within a test session. Second, in image labeling designs once an image is correctly labeled, it should not be used again. It is difficult for a small site to acquire a huge dictionary of images which an attacker does not have access to and without a means of automatically acquiring new labeled images. While when orientation is the focus, the image of objects can be repeatedly used as long as the object has not been identified. Image consumption can be much slower.
Chapter 2

SYSTEM OVERVIEW

2.1 Architecture

The CAPTCHA system consists of essentially four major components: (a) Image Collection (b) Image Preprocessing (c) Server (d) User Interface

Figure 8. System Architecture
2.1.1 Image Collection:

Initially images are collected from the web by using one of the many search engines that are available today. Care has to be taken in collecting the right kind of images i.e. images which have no ambiguity whatsoever with respect to their orientation. We use only those images which have their orientation fixed and not vary at different times. For example if we use an image of a dog, the user would know its correct orientation, which is the side with its legs facing downwards. Whereas if we use an image of a ball or a pipe, the orientation would depend on the object’s surroundings and not on the object itself. Figure 9 shows some images which are ambiguous with respect to orientation. After collecting the images they are sent for processing. Another case is when we have two objects in an image both having different orientations. Then the user would be in doubt which objects in the image to properly orientate.

Figure 9. Images which are ambiguous with respect to orientation
### 2.1.2 Image Preprocessing:

Image preprocessing is a critical step in our application. Research has been going on in developing a reverse image search engine i.e. a search engine for images which use an image as the input. TinyEye [27] is one of the first search engines where we can submit an image to find out where it came from, how it is being used, if modified versions of the image exist, or to find higher resolution versions. When we submit an image to be searched, TinEye creates a unique and compact digital signature or 'fingerprint' for it, then compares this fingerprint to every other image in their index to retrieve matches. TinyEye can find modified versions of an image but the term ‘modified’ only includes images which have been cropped, edited or resized. We use an elaborate process of preprocessing images which essentially converts an image to a line drawing. This ensures that an image used by our application cannot be searched on the web in order to find out its correct orientation.

### 2.1.3 Server:

Server indicates the back end logic which implements the following functions. The server has to select a fixed number of images randomly from our set of processed images. In those images the server selects some images for rotation and it also selects the angle at which each image should be rotated. Then the server would send this set of images to the user interface for the user to take the test. After this step a communication path is set up with the user interface to take care of test validation and also reloading a different test. The backend logic for the server was implemented using Java Server Pages technology.
2.1.4 User Interface:

The user interface essentially contains a table populated with twelve images, some of which are rotated and some of which are not. Figure 10 shows the GUI of our CAPTCHA application. Two buttons are provided under the table for submitting the answer to the server and to reload a different set of images. The user has to select the images which are rotated by clicking on the image once which highlights the border of the image selected. The user can deselect the image by clicking on the image again. A zoom in feature is provided for each image for the user to make the image clearer. The user can access this feature by placing the mouse pointer on the image that he wishes to zoom in on. Once the user moves the mouse pointer away from the image the image would automatically zoom out. If the user would like a new set of images he would use the reload button to request the server. Once all the rotated images are selected by the user, he would use the submit button to send his answer to the server for validation. The server would then send back the result of the test, i.e. a Pass or a Fail, along with the response time taken for the whole test. The scoring of the test and also the refresh images part are implemented such that the requests to the server are sent asynchronously to the server without reloading the page every time.

Figure 10. User Interface of the CAPTCHA application
2.2 Working

The images are initially preprocessed before using them for the CAPTCHA test. The preprocessing steps consist of passing the images thru a high pass filter, applying a note paper effect, applying a threshold and finally adding some noise to the image. We used the Adobe Photoshop CS4 to process our images.

2.2.1 High Pass Filter

A high-pass filter is a filter that passes high frequencies well, but attenuates (reduces the amplitude of) frequencies lower than the cutoff frequency. The basic high-pass filter consists of a capacitor in series with the signal path in conjunction with a resistor in parallel with the signal path. The resistance times the capacitance (R×C) is the time constant (τ); it is inversely proportional to the cutoff frequency, at which the output power is half the input (−3 dB):

\[ V_{\text{out}} \text{ and } V_{\text{in}} \text{ are related in time domain as follows:} \]

\[ V_{\text{out}}(t) = C \left( \frac{dV_{\text{in}}}{dt} - \frac{dV_{\text{out}}}{dt} \right) \quad R = RC \left( \frac{dV_{\text{in}}}{dt} - \frac{dV_{\text{out}}}{dt} \right) \]

Figure 11. Analog high pass filter
Discretizing this equation, assume that samples of the input and output are taken at evenly-spaced points in time separated by \( \Delta_T \) time. Let the samples of \( V_{\text{in}} \) be represented by the sequence \((x_1, x_2, \cdots, x_n)\), and let \( V_{\text{out}} \) be represented by the sequence \((y_1, y_2, \cdots, y_n)\), which correspond to the same points in time.

\[
y_i = \alpha y_{i-1} + \alpha (x_i - x_{i-1}) \quad \text{where} \quad \alpha \triangleq \frac{RC}{RC + \Delta_T}
\]

With images a high pass filter is essentially mathematical transformation, also called a convolution product, which allows the value of a pixel to be modified according to the values of neighboring pixels, with coefficients, for each pixel of the region to which it is applied. The filter is represented by a table (matrix), which is characterized by its dimensions and its coefficients, whose centre corresponds to the pixel concerned.

\[
\begin{array}{ccc}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{array}
\]

Figure 12. Matrix convolution for a high pass filter

The table coefficients determine the properties of the filter. Figure 12 shows a typical matrix convolution for a high pass filter. The matrix convolution of the high pass filter is designed to increase the brightness of the center pixel relative to neighboring pixels.
We used the Adobe Photoshop to apply the effects of the high pass filter to our image processing. A parameter called the Radius value is provided to increase or decrease the intensity of the filter. As the radius value is increased more sharpening would be added to the image. But if the value is set too high, a halo effect would be seen around the edges of the image. To avoid this we fix the radius value just when these effects are observed. For our processing we used a radius value of 0.3. Figure 13 shows the input image and the output got after applying the high pass filter.

![Figure 13. Image processing using high pass filter](image)

2.2.2 Note Paper Filter

We used the Note Paper Filter to redraw the photo as textured light and dark tones. This is based on the current selection of fore-and background colors. There are three parameters present that control the look and strength of the filter. The Image Balance parameter adjusts which areas are toned the foreground color and which are converted to the background hue. Low values favor the background color and high settings color more of the picture with the foreground color. We have used a value of 25
for the Image Balance. The second parameter called Graininess determines the amount of
grain that is added to the picture. We have not used any Graininess for our image
processing. The third parameter called the Relief adjusts the strength of the simulated
side lighting falling on the paper surface. With higher values the texture is more
pronounced. Changing the foreground and background colors away from their defaults
greatly alters the nature of the filtered results. We have used the default value of 0 for the
Relief. Figure 14 shows the input image and the output got after applying a note paper
filter.

![Figure 14. Image processing using note paper effect](image)

2.2.3 Threshold

Threshold essentially converts grayscale or color images to high-contrast, black-
and-white images. We can specify a certain level as a threshold. All pixels lighter than
the threshold are converted to white; all pixels darker are converted to black. A threshold
level of 180 was sufficient for our image processing. Figure 15 shows the input image
and the output got after applying a threshold.
2.2.4 Noise Addition

The Noise filter applies random pixels to an image, simulating the effect of shooting pictures on high-speed film. We use a uniform noise filter which distributes color values of noise using random numbers between 0 and plus or minus the specified value, creating a subtle effect. We apply a noise level of 20% for our image processing. Figure 16 shows the input image and the output got after adding noise to the image.
### 2.2.5 Image Selection

After the initial processing of the images, a set of twelve images are selected randomly from the entire set of processed images. Out of these twelve images, a random number \( n \) between 1 and 12, number of images are selected for rotation.

These selected images are each rotated 90, 180 or 270 degrees randomly. The set of images now contain \( n \) number of rotated images and \( (12-n) \) non-rotated images. Finally this entire set of images is resized to a length of 100X100 pixel length for uniformity. Figure 17 illustrates the process flow during image selection.
CHAPTER 3

DESIGN METHODOLOGY

3.1 Introduction

We built a browser-based client, which had several advantages over traditional client/server based applications. These include nearly unlimited client access and greatly simplified application deployment and management. We need a technology for interfacing the external application software with a web server and create interactive pages as part of a Web-based application. It should separate programming logic from page design through the use of components that are called from the page itself. The most common available technologies are Java Server Pages (JSP), Active Server Pages (ASP) and PHP. We chose Java Server Pages in implementing our application. We discuss a few advantages JSP has over ASP and PHP.

Firstly ASP pages are written in VBScript and JSP pages are written in the Java programming language. Therefore, JSP pages are platform-independent and ASP pages are not. Secondly although ASP pages are cached, they are always interpreted. By contrast, JSP pages are compiled into Java servlets and loaded into memory the first time they are called, and executed for all subsequent calls. This gives JSP pages a speed and scalability advantage over ASP pages. Thirdly JSP pages have an advanced feature known as extensible tags. This mechanism enables developers to create custom tags. In
other words, extensible tags allow us to extend the JSP pages tag syntax which cannot be done with ASP pages. Also ASP pages work only with Microsoft IIS and Personal Web Server. Using ASP pages requires a commitment to Microsoft products, while JSP pages do not tie us up to any specific web server or operating system. Although ASP technology is available on other platforms through third-party porting products, to access components and interact with other services, the ActiveX objects must be present on the selected platform. If not present, a bridge to a platform supporting them is required.

Coming to PHP, JSP is much more powerful than PHP since it has access to all the Java libraries whereas PHP only has access to PHP libraries. JSP is Object-Oriented, so leads to cleaner code that's easier to debug, maintain, and improve. PHP also allows objects, but the object model is more primitive, and most scripted pages ignore PHP objects and just use normal variables. With JSP, if the code inside a page gets too big, or if we want to use it elsewhere we can make it into a Java class, and invoke it from anywhere in our application. With PHP, we are stuck inside the HTML box. JSP's concept of state management and persistence is more explicit and powerful than PHP's. With JSP, we can specify whether a variable persists for the page, the request, the session, or the application (or if it's just local to the function). The JSP engine automatically does the right thing with cookies so we have access to the variable on later requests. With PHP, we just have "global" and "not global", we don't have automatic session management, and have to do our state thing manually with cookies or hidden variables.
3.2 Java Server Pages

Java Server Pages (JSP) technology enables to rapidly develop and easily maintain, information-rich, dynamic Web pages that leverage existing business systems. As part of the Java technology family, JSP technology enables rapid development of Web-based applications that are platform independent. JSP technology separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

The main advantages with using Java Server Pages are

- Extend the JSP language: Java tag library developers and designers can extend the JSP language with "simple tag handlers," which utilize a new, much simpler and cleaner, tag extension API. This spurs the growing number of pluggable, reusable tag libraries available, which in turn reduces the amount of code needed to write powerful Web applications.

- Easily write and maintain pages: The Java Server Pages Standard Tag Library (JSTL) expression language is now integrated into JSP technology and has been upgraded to support functions. The expression language can now be used instead of scriptlet expressions.

3.3 JSP Technology and Java Servlets

JSP technology uses XML-like tags that encapsulate the logic that generates the content for the page. The application logic can reside in server-based resources (such as JavaBeans component architecture) that the page accesses with these tags. Any and all
formatting (HTML or XML) tags are passed directly back to the response page. By separating the page logic from its design and display and supporting a reusable component-based design, JSP technology makes it faster and easier than ever to build Web-based applications.

Java Server Pages technology is an extension of the Java Servlet technology. Servlets are platform-independent, server-side modules that fit seamlessly into a Web server framework and can be used to extend the capabilities of a Web server with minimal overhead, maintenance, and support. Unlike other scripting languages, servlets involve no platform-specific consideration or modifications; they are application components that are downloaded, on demand, to the part of the system that needs them. Together, JSP technology and servlets provide an attractive alternative to other types of dynamic Web scripting/programming by offering: platform independence; enhanced performance; separation of logic from display; ease of administration; extensibility into the enterprise; and ease of use.

Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional CGI and than many alternative CGI-like technologies. With traditional CGI, a new process is started for each HTTP request. If the CGI program does a relatively fast operation, the overhead of starting the process can dominate the execution time. With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are \( N \) simultaneous request to the same CGI program, then the code for the CGI program is loaded into memory \( N \) times. With
Servlets, however, there are $N$ threads but only a single copy of the servlet class. Servlets also have more alternatives than do regular CGI programs for optimizations such as caching previous computations, keeping database connections open, and the like. Servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities. Java servlets lets us easily do several things that are difficult or impossible with regular CGI. Servlets can talk directly to the Web server while regular CGI programs can't. This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations. Servlets are written in Java and follow a well-standardized API. Consequently, servlets written for one server can run virtually unchanged on any other Web server.

The life cycle of a servlet can be categorized into four parts:

1. **Loading and Instantiation:** The servlet container loads the servlet during startup or when the first request is made. The loading of the servlet depends on the attribute `<load-on-startup>` of web.xml file. If the attribute `<load-on-startup>` has a positive value then the servlet is loaded with loading of the container otherwise it load when the first request comes for service. After loading of the servlet, the container creates the instances of the servlet.
2. **Initialization**: After creating the instances, the servlet container calls the init() method and passes the servlet initialization parameters to the init() method. The init() must be called by the servlet container before the servlet can service any request. The initialization parameters persist until the servlet is destroyed. The init() method is called only once throughout the life cycle of the servlet. The servlet will be available for service if it is loaded successfully otherwise the servlet container unloads the servlet.

3. **Servicing the Request**: After successfully completing the initialization process, the servlet will be available for service. Servlet creates separate threads for each
request. The servlet container calls the service() method for servicing any request. The service() method determines the kind of request and calls the appropriate method (doGet() or doPost()) for handling the request and sends response to the client using the methods of the response object.

4. **Destroying the Servlet:** If the servlet is no longer needed for servicing any request, the servlet container calls the destroy() method. Like the init() method this method is also called only once throughout the life cycle of the servlet. Calling the destroy() method indicates to the servlet container not to sent the any request for service and the servlet releases all the resources associated with it. Java Virtual Machine claims for the memory associated with the resources for garbage collection. Figure 18 illustrates the life cycle of a servlet.

When the jsp page is called, it will be compiled by the JSP engine into a java servlet. At this point the servlet is handled by the servlet engine just like any other servlet. The servlet engine then loads the servlet class using a class loader and executes it to create dynamic HTML to be sent to the browser, as shown in Figure 19. The next time the page is requested, the JSP engine executes the already-loaded servlet unless the JSP page has changed, in which case it is automatically recompiled into a servlet and executed. Figure shows a typical request/response flow when a jsp is called.
3.4 Use of AJAX

AJAX (shorthand for asynchronous JavaScript + XML), is a group of interrelated web development techniques used to create interactive web applications or rich Internet applications. With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. The use of AJAX was critical for the development of the user interface for the following reasons:

- The captcha application is not a standalone application and would have to be embedded in another html page. So any communication with the server, like submission of results or refreshing the set of images, has to be done asynchronously without reloading the entire page every time.
• Using traditional methods, the entire application would have to be reloaded on every request to the server. However, using Ajax, the web application can request only the content that needs to be updated, thus drastically reducing bandwidth usage and load time.

• The use of asynchronous requests allows the client's Web browser UI to be more interactive and to respond quickly to inputs, and sections of pages can also be reloaded individually. Users may perceive the application to be faster or more responsive, even if the application has not changed on the server side.

• The use of Ajax can reduce connections to the server, since scripts and style sheets only have to be requested once.

• State can be maintained throughout a web site. JavaScript variables will persist because the main container page need not be reloaded.
CHAPTER 4
RESULTS

4.1 GUI Working

Figure 20 shows a set of twelve images displayed in a table. Some of these are rotated and some are not. Two buttons, the Submit and Reload, are provided under the table.

Figure 20. GUI step 1
Figure 21 shows an image zoomed out by placing the mouse pointer on the image.

![Figure 21. GUI step 2](image1)

Figure 21. GUI step 2

Figure 22 shows an image selected by the user. The selected image is highlighted in green in order to indicate that the image is selected.

![Figure 22. GUI step 3](image2)

Figure 22. GUI step 3
Figure 23 shows that the user has selected all the rotated images and has pressed the submit button in order to validate the test. A response is given back by the server indicating the result of the test and the response time needed for the validation of the test.

![Figure 23. GUI step 4](image)

4.2 Usability Test Results

We implemented a prototype system of our application and carried out real user experiments based on it. Our purpose is to check usability and to explore the ways to improve it. We set up a web server with Apache Tomcat 6.0 on a PC with a 2.66 GHz Intel Pentium Core Duo CPU and 2 GB RAM. We have collected an image set of 7000 images and used these images for our application. Each image is of size 100X100 pixel in
JPG format. The test was carried out 200 times and the average response time for successfully passing a test was 14 seconds and the success ratio was 92%.
CHAPTER 5

CONCLUSIONS

We have presented a novel CAPTCHA system that requires users to choose the images which are rotated. This is a task that will be familiar to many people as it is analogous to the simple act of sorting through physical photographs. Our system further improves traditional text-based CAPTCHAs in that it is language and written-script independent, and supports keyboard-difficult environments. It is important that random images are not chosen for this task; they must be carefully selected. After selection the image preprocessing steps ensure that our CAPTCHA cannot be defeated by state-of-the-art orientation detection systems. Some of the major pitfalls associated with other proposed image based CAPTCHA systems do not apply to our CAPTCHA system. A priori knowledge of the image’s label is not needed, which makes examples for our system easier to automatically generate than other image-based CAPTCHA systems. Since no exact labels are needed from the user, the class number of objects that can be used for orientation is theoretically infinite and can be huge in practice. The classes have no need to be static. And they can be dynamically updated even within a test session. Also in image labeling designs once an image is correctly labeled, it should not be used again. It is difficult for a small site to acquire a huge dictionary of images which an attacker does not have access to and without a means of automatically acquiring new labeled images.
While when orientation is the focus, the image of objects can be repeatedly used as long as the object has not been identified. Image consumption can be much slower. Furthermore since we use an image set which is not static but gets dynamically updated, an attack which can manually classify all images is ruled out. Finally, our system provides opportunities for a number of interesting extensions. First, the set of images selected can be chosen to be more interesting or valuable to the end-user by displaying those that are related to the overall theme of the website. Second, a kind of social-correction scheme could be implemented where the users select those images which seem to be ambiguous so that those images could be removed from future tests.
[22] http://vorm.net/captchas
[35] Collaborative Filtering CAPTCHAs, Monica Chew and J.D. Tygar, University of California, Berkeley