Telling Humans and Computers Apart (Automatically) or How Lazy Cryptographers do AI

Luis von Ahn Manuel Blum John Langford

February 2002
CMU-CS-02-117

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We introduce the notion of a CAPTCHA: a program that can differentiate between humans and computers based only on interactions over a computer network.

This material is based upon work partially supported by a National Science Foundation Aladdin Grant
Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

Keywords: Automated Turing Tests, CAPTCHA

Introduction

If you try to get a new email account at Yahoo, you'll be asked to prove that you're a human and not a computer. Why? Because a single computer program can get thousands of free email accounts per second. And that's bad for Yahoo. But how do you prove to a computer that you're a human?

Proving that you're a human to *another human* can be done using an idea from the 1950s: the Turing Test [9]. A human judge asks you a bunch of questions and decides, depending on your answers, whether he's talking to a human or a computer.

Proving that you're a human to a *computer* is another matter. It requires a test (or a set of tests) that computers can grade, humans can pass, but paradoxically, computers can't pass. In our lingo, it requires a CAPTCHA.

Gotcha!

Intuitively, a CAPTCHA is a program that can generate and grade tests that:

A) Most humans can pass

But

B) Current computer programs can't pass.[1]

The acronym is a mouthful: CAPTCHA stands for "Completely Automated Public Turing Test to Tell Computers and Humans Apart". The P for Public means that the code and the data used by a CAPTCHA should be publicly available. Thus a program that can generate and grade tests that distinguish humans from computers, but whose code or data are private, is *not* a CAPTCHA.

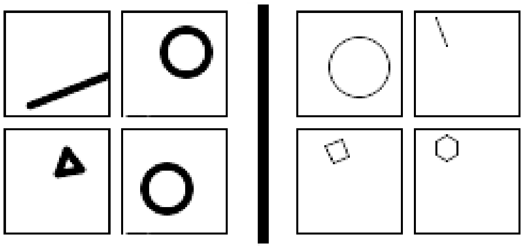
Examples

Let's look at few examples of CAPTCHAs before moving on. When reading, think about why each of these is indeed a CAPTCHA. The code for all the programs described in this section can be found online at *http://www.captcha.net*.

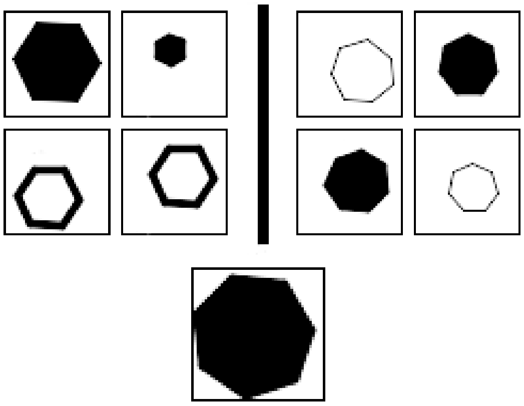
- **Gimpy.** GIMPY works as follows: it picks seven words out of a dictionary, and renders a distorted image containing the words (as shown in the figure below). GIMPY then presents a test to its user, which consists of the distorted image and the directions: *"type three words appearing in the image"*. Given the types of deformations that GIMPY uses, most humans can read three words from the distorted image, while current computer programs can't. (We still haven't *proved* that GIMPY is a CAPTCHA! The issues revolving around such a proof will be discussed later.)

[1]The words "Most" and "Current" can be precisely defined, but the intuitive meaning is good for the purposes of this article.

- **Bongo.** Another example of a CAPTCHA is the program we call BONGO.[2] BONGO asks the user to solve a visual pattern recognition problem. In particular, it displays two series of blocks, the LEFT and the RIGHT. The blocks in the LEFT series differ from those in the RIGHT, and the user must find the characteristic that sets them apart. A possible LEFT and RIGHT series are shown below:



  (These two series are different because everything in the LEFT is drawn with thick lines, while everything in the RIGHT is drawn with thin lines.) After seeing the two series of blocks, the user is presented with a single block and is asked to determine whether this block belongs to the RIGHT series or to the LEFT. The user passes the test if he or she correctly determines the side to which the block belongs to. Try it yourself: to which side does the isolated block belong in the figure below?



- **Pix.** Yet another example of a CAPTCHA is a program that has a large database of labeled images (such databases can be found in [7, 6]). All of these images should be pictures of concrete objects (a horse, a table, a house, a flower, etc). The program picks an object at random, finds 6 images of that object from its database, presents them to the user and then asks the question *"what are these pictures of?"* Current computer programs should not be able to answer this question, so PIX should be a CAPTCHA.

  But actually, PIX, as stated, is not a CAPTCHA: it is very easy to write a program that can answer the question *"what are these images of?"* Remember that all the code and data of a CAPTCHA should be publicly available; in particular, the image database that PIX uses should be public. Hence, writing a program that can answer the question *"what are these pictures of?"* is easy: search the database for the images presented and find their label. In order

[2]BONGO is named after M.M. Bongard, who published a book of pattern recognition problems ([2]) in 1951.

for PIX to actually be a CAPTCHA, the images need to be randomly distorted before being presented to the user, in such a way that searching the database for the images is hard for current computer programs.

- **Eco.** ECO is a sound-based CAPTCHA. The program picks a word or a sequence of numbers at random, renders the word or the numbers into a sound clip and distorts the sound clip. It then presents the distorted sound clip to its user and asks them to enter the contents of the sound clip. ECO is based on the gap in ability between humans and computers in recognizing spoken language. Nancy Chan of the City University in Hong Kong has also implemented a sound-based system of this variety [4].

It's still an open problem to create a text-based CAPTCHA (one which doesn't use any images or sound clips).

Online Polls

CAPTCHAs have a wide variety of applications on the web. Here's an exmaple. Almost two years ago, *http://www.slashdot.com* released an online poll asking for the best school in computer science (a dangerous question to ask over the web!). As it is the case with most online polls, IP addresses of voters were recorded in order to prevent single users from voting more than once. However, students at Carnegie Mellon figured out a way to stuff the ballots using programs that voted for CMU thousands of times. CMU's score started growing rapidly. The next day, students at MIT wrote their own program and the poll became a contest between voting "bots". MIT finished with 21,156 votes, Carnegie Mellon with 21,032 and every other school with less than 1,000. And, in fact, any other online poll suffers from this weakness against bots. But CAPTCHAs offer a solution: voters should show they are human before being allowed to vote.

Reductions to Hard AI Problems

But how do we *prove* that computer programs will never pass the tests generated by a CAPTCHA? Unfortunately, we believe there is no way to prove such a strong statement: since humans can pass the tests, proving that computer programs will *never* pass them would show that there exist things that humans can do but computers can't. We (the authors) believe that some day computers will be as good as humans, and possibly better, in every respect.

The best we can do is to present evidence that, given the current state of technology, it is hard to write a program that can pass the tests generated by a CAPTCHA. We can do this by proving the following statement: *Any program that passes the tests generated by a* CAPTCHA *can be used to solve a hard unsolved Artificial Intelligence (AI) problem.* This statement requires some explanation:

- First, by "[a] program that passes the tests generated by a CAPTCHA", we mean a program (possibly probabilistic) that has some (not so small, but precisely defined) probability of passing a given random test generated by the CAPTCHA.

- What is a hard unsolved AI problem? Intuitively, it is an AI problem that remains unsolved despite the fact that the AI community has been working on it for several years. For the proof of this theorem to go through, the AI problem needs to be *precisely* defined. It is surprisingly difficult (but possible) to precisely define an AI problem.

- Why does this give evidence that it's hard to pass the tests generated by a CAPTCHA? If an AI problem remains unsolved after many intelligent people have been working on it for a long time, then it is hard to write a program that can solve this problem. Our statement says that it's at least as hard to write a program that can pass the tests generated by a CAPTCHA.

P Stands for Public. Why?

It might seem like a bad idea to make the code of a CAPTCHA public: if we kept the code hidden, THE ENEMY (the person trying to write a program to pass the tests generated by a CAPTCHA) might have a harder time. But actually there are several reasons for CAPTCHAs to be public:

- **Hackers**. If a CAPTCHA requires private data in order to be secure, then this CAPTCHA is vulnerable to hacker attacks: once a hacker breaks into the system and finds this secret data, the CAPTCHA ceases to be secure forever. We want CAPTCHAs to generate tests that are hard to pass for all computers; not that are hard to pass for computers that don't know a particular secret.

- **Reverse Engineering and Analysis.** If the code weren't public, THE ENEMY could still learn how a CAPTCHA works by simply taking many of the tests generated by it. Moreover, there is no general way to know exactly how much THE ENEMY can learn about how a CAPTCHA works. Hence, we may as well assume that THE ENEMY can learn *everything* and then prove that *even in this case* it is hard for THE ENEMY to write a program that can pass the tests generated by the CAPTCHA.

- **We're doing AI**. We *want* our CAPTCHAs to be broken! If a CAPTCHA is broken, then a previously unsolved AI problem is solved. And this is how lazy cryptographers do AI: we help THE ENEMY as much as we can. This is also why we want several different kinds CAPTCHAs: the more CAPTCHAs that get broken, the more AI problems that get solved.

Other Applications

- **Free Email Services.** Several companies (Yahoo, Microsoft, etc.) offer free email services. Most of these suffer from a specific type of attack: "bots" that sign up for thousands of email accounts every minute. In particular, web pornography companies use such tactics to obtain free email accounts from which they can send junk mail. A solution to this problem is to require users solve a CAPTCHA test before they receive an account. Yahoo has adopted this solution, and, in fact, uses a variation of our program GIMPY.

- **Search Engine Bots.** You may not want to have your web page indexed. It's sometimes surprising how much others can learn about you by just entering your name into Google or Altavista. There is an *html* tag to prevent bots from reading your web page. The tag, however, doesn't guarantee that bots won't read your web page; it only serves to say "no bots, please". Search engine bots, since they usually belong to large companies, respect web pages that don't want to allow bots. However, if you want to *guarantee* that bots won't enter your web-site, you need a CAPTCHA.

- **Shopping Agents.** There are many online stores that offer extremely low prices. Part of the reason these companies can maintain their prices is because of advertisements: users see various products or brands being advertised while they shop. "Shopping Agents" are programs that visit many online stores, retrieve the best prices from all of them, and present a table containing the best prices to the user. These shopping agents are bad for the online stores because users don't get to see the advertisements. An online store which uses a CAPTCHA test before revealing its prices will ensure that only humans visit the site.

Stealing Cycles From Humans

Now, CAPTCHAs aren't always a good solution the the Shopping Agent problem. Why? Because if an online store displays a CAPTCHA test before allowing a user to see the prices, then a shopping agent can simply ask the human behind the computer to solve the CAPTCHA test.

This same strategy can be used to defeat CAPTCHAs in many cases. We call it *stealing cycles from humans*. Consider web porn companies, for instance. These companies can still obtain thousands of free email accounts even if the email provider uses a CAPTCHA. How? The porn companies can have bots that try to get free email accounts, and as soon as the bots encounter a CAPTCHA test, they simply send the test to the porn site. Back at the porn site, there are several thousand users seeing pornographic pictures, and one of them is told: *"please solve the following test before we can show you the next picture". Voila.* Solving this problem is still a largely open question.

Notice how *stealing cycles from humans* works: humans are used in order to expand the computational abilities of bots. This leads to an interesting digression: in general, how can we use humans to expand our computational abilities? Currently, there are many things that humans can do but computers can't: recognize facial expressions from any angle, understand language, read distorted text, recognize which animal is in a picture. Can we use humans in some way to make computers solve these problems in a distributed manner? Can we have a kind of SETI@home project in which web users help to classify all books in the library of congress?

Imagine a program that tells you which is the best haircut for you. It seems nearly impossible to write such a program given the current state of technology. But we can steal cycles from humans! If you visit *http://www.hotornot.com*, you'll get a picture of a person (man or woman) and you're supposed to rate how attractive you think the person is (on a scale from 1 to 10). After you rate the person you learn what other users have said about them and you also get to rank another person. This site is very popular: sometimes entire offices of graduate students are found rating these pictures. The "Best Haircut" program should now be obvious: you give the program a picture of yourself and it posts several pictures of you, all with different haircuts, on *http://www.hotornot.com.* Whichever picture is rated the highest is the "best" haircut for you.

Notice that CAPTCHAs are the ultimate tool for stealing cycles from intelligent people: malicious, intelligent programmers (writing programs for getting thousands of free email accounts, etc.) can be put to work on hard unsolved AI problems.

Acknowledgements and Credits

The ideas and the theory behind CAPTCHAs were developed at Carnegie Mellon University in the fall of 2000, when Udi Manber, chief scientist of Yahoo, started working in collaboration with us. We'd like to thank Udi for his helpful motivation and feedback. We'd also like to thank Henry Baird and his research group at PARC (formerly Xerox PARC) for the enthusiasm about our project and all the support and ideas.

As we have discovered over the last few months, CAPTCHAs have been around for quite some time (though not under that name). Andrei Broder and his team at Altavista developed a system similar to GIMPY in 1998 [3]. Another system of this kind can be found in [5]. Moni Naor also mentions ideas related to ours in [8]. To the best of our knowledge, however, we were the first to conduct a serious investigation of Automated Turing Tests.

Finally, we'd like to thank the National Science Foundation for the Aladdin Grant.

References

[1] Luis von Ahn, Manuel Blum and John Langford. The Captcha Project homepage: *http://www.captcha.net.*

[2] Mikhail M. Bongard. (1970). *Pattern Recognition.* Rochelle Park, N.J.: Hayden Book Co., Spartan Books.

[3] Andrei Broder. The Altavista Ransom Note. *Unpublished.*

[4] Nancy Chan. Program Byan: *http://drive.to/research*

[5] A. L. Coates, H. S. Baird, and R. J. Fateman. Pessimal Print: A Reverse Turing Test. *Proc., ICDAR 2001 Int'l Conf. on Document Analysis and Recognition*, Seattle, WA

[6] Corbis: *http://www.corbis.com.*

[7] The Getty Images: *http://www.gettyimages.com.*

[8] Moni Naor. Verification of a human in the loop or Identification via the Turing Test. *Unpublished.*

[9] Alan M. Turing. Computing Machinery and Intelligence. In: *Mind, Vol. 59, No. 236, pp. 433-460.* 1950.