CAPTCHA: Using Hard AI Problems For Security

Luis von Ahn[1], Manuel Blum[1], Nicholas J. Hopper[1], and John Langford[2]

[1] Computer Science Dept., Carnegie Mellon University, Pittsburgh PA 15213, USA
[2] IBM T.J. Watson Research Center, Yorktown Heights NY 10598, USA

Abstract. We introduce CAPTCHA, an automated test that humans can pass, but current computer programs can't pass: any program that has high success over a CAPTCHA can be used to solve an unsolved Artificial Intelligence (AI) problem. We provide several novel constructions of CAPTCHAs. Since CAPTCHAs have many applications in practical security, our approach introduces a new class of hard problems that can be exploited for security purposes. Much like research in cryptography has had a positive impact on algorithms for factoring and discrete log, we hope that the use of hard AI problems for security purposes allows us to advance the field of Artificial Intelligence. We introduce two families of AI problems that can be used to construct CAPTCHAs and we show that solutions to such problems can be used for steganographic communication. CAPTCHAs based on these AI problem families, then, imply a win-win situation: either the problems remain unsolved and there is a way to differentiate humans from computers, or the problems are solved and there is a way to communicate covertly on some channels.

1 Introduction

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. Such a program can be used to differentiate humans from computers and has many applications for practical security, including (but not limited to):

– **Online Polls.** In November 1999, *slashdot.com* released an online poll asking which was the best graduate school in computer science (a dangerous question to ask over the web!). As 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 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". MIT finished with 21,156 votes, Carnegie Mellon with 21,032 and every other school with less than 1,000. Can the result of any online poll be trusted? Not unless the poll requires that only humans can vote.

– **Free Email Services.** Several companies (Yahoo!, Microsoft, etc.) offer free email services, most of which suffer from a specific type of attack: "bots" that sign up for thousands of email accounts every minute. This situation can be improved by requiring users to prove they are human before they can get a free email account. Yahoo!, for instance, uses a CAPTCHA of our design to prevent bots from registering for accounts. Their CAPTCHA asks users to read a distorted word such as the one shown below (current computer programs are not as good as humans at reading distorted text).

Fig. 1. The Yahoo! CAPTCHA.

– **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; 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 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. A few companies, such as *www.spamarrest.com* are already marketing this idea.
– **Preventing Dictionary Attacks.** Pinkas and Sander [11] have suggested using CAPTCHAs to prevent dictionary attacks in password systems. The idea is simple: prevent a computer from being able to iterate through the entire space of passwords by requiring a human to type the passwords.

The goals of this paper are to lay a solid theoretical foundation for CAPTCHAs, to introduce the concept to the cryptography community, and to present several novel constructions.

Lazy Cryptographers Doing AI

Note that from a mechanistic point of view, there is no way to *prove* that a program cannot pass a test which a human can pass, since there is a program — the human brain — which passes the test. All we can do is to present evidence that it's hard to write a program that can pass the test. In this paper, we take an approach familiar to cryptographers: investigate state-of-the-art algorithmic developments having to do with some problem, assume that the adversary does not have algorithms for that problem that are are much better than the state-of-the-art algorithms, and then prove a reduction between passing a test and exceeding the performance of state-of-the-art algorithms. In the case of ordinary cryptography, it is assumed (for example) that the adversary cannot factor 1024-bit integers in any reasonable amount of time. In our case, we assume that the adversary cannot solve an Artificial Intelligence problem with higher accuracy than what's currently known to the AI community. This approach, if it achieves widespread adoption, has the beneficial side effect of inducing security researchers, as well as otherwise malicious programmers, to advance the field of AI (much like computational number theory has been advanced since the advent of modern cryptography).

> *A* CAPTCHA *is a cryptographic protocol whose underlying hardness assumption is based on an AI problem.*

An important component of the success of modern cryptography is the practice of stating, very precisely and clearly, the assumptions under which cryptographic protocols are secure. This allows the rest of the community to evaluate the assumptions and to attempt to break them. In the case of Artificial Intelligence, it's rare for problems to be precisely stated, but using them for security purposes forces protocol designers to do so. We believe that precisely stating unsolved AI problems can accelerate the development of Artificial Intelligence: most AI problems that have been precisely stated and publicized have eventually been solved (take chess as an example). For this reason it makes practical sense for AI problems that are used for security purposes to also be *useful.* If the underlying AI problem is useful, a CAPTCHA implies a win-win situation: either the CAPTCHA is not broken and there is a way to differentiate humans from computers, or the CAPTCHA is broken and a useful AI problem is solved. Such is not the case for most other cryptographic assumptions: the primary reason algorithms for factoring large numbers are useful is because factoring has applications in cryptanalysis.

In this paper we will present constructions of CAPTCHAs based on certain AI problems and we will show that solving the CAPTCHAs implies solving the AI problems. The AI problems we chose have several applications, and we will show that solutions to them can be used, among other things, for steganographic communication (see Section 5).

Related Work

The first mention of ideas related to "Automated Turing Tests" seems to appear in an unpublished manuscript by Moni Naor [10]. This excellent manuscript contains some of the crucial notions and intuitions, but gives no proposal for an Automated Turing Test, nor a formal definition. The first practical example of an Automated Turing Test was the system developed by Altavista [8] to prevent "bots" from automatically registering web pages. Their system was based on the difficulty of reading slightly distorted characters and worked well in practice, but was only meant to defeat off-the-shelf Optical Character Recognition (OCR) technology. (Coates et al [5], inspired by our work, and Xu et al [14] developed similar systems and provided more concrete analyses.) In 2000 [1], we introduced the notion of a CAPTCHA as well as several practical proposals for Automated Turing Tests.

This paper is the first to conduct a rigorous investigation of Automated Turing Tests and to address the issue of *proving* that it is difficult to write a computer program that can pass the tests. This, in turn, leads to a discussion of using AI problems for security purposes, which has never appeared in the literature. We also introduce the first Automated Turing Tests not based on the difficulty of Optical Character Recognition. A related general interest paper [2] has been accepted by *Communications of the ACM.* That paper reports on our work, without formalizing the notions or providing security guarantees.

2 Definitions and Notation

Let $\mathcal{C}$ be a probability distribution. We use $[\mathcal{C}]$ to denote the support of $\mathcal{C}$. If $P(\cdot)$ is a probabilistic program, we will denote by $P_r(\cdot)$ the deterministic program that results when P uses random coins r.

Let (P, V) be a pair of probabilistic interacting programs. We denote the output of V after the interaction between P and V with random coins u_1 and u_2, assuming this interaction terminates, by $\langle P_{u_1}, V_{u_2}\rangle$ (the subscripts are omitted in case the programs are deterministic). A program V is called a *test* if for all P and u_1, u_2, the interaction between P_{u_1} and V_{u_2} terminates and $\langle P_{u_1}, V_{u_2}\rangle \in \{accept, reject\}$. We call V the *verifier* or *tester* and any P which interacts with V the *prover*.

Definition 1. Define the *success* of an entity A over a test V by

$$\mathbf{Succ}_A^V = \Pr_{r,r'}[\langle A_r, V_{r'}\rangle = accept].$$

We assume that A can have precise knowledge of how V works; the only piece of information that A can't know is r', the internal randomness of V.

CAPTCHA

Intuitively, a CAPTCHA is a test V over which most humans have success close to 1, and for which *it is hard to write a computer program that has high success over V*. We will say that it is hard to write a computer program that has high success over V if *any program that has high success over V can be used to solve a hard AI problem.*

Definition 2. A test V is said to be (α, β)-*human executable* if at least an α portion of the human population has success greater than β over V.

Notice that a statement of the form "V is (α, β)-human executable" can only be proven empirically. Also, the success of different groups of humans might depend on their origin language or sensory disabilities: color-blind individuals, for instance, might have low success on tests that require the differentiation of colors.

Definition 3. An *AI problem* is a triple $\mathcal{P} = (S, D, f)$, where S is a set of problem instances, D is a probability distribution over the problem set S, and $f : S \to \{0,1\}^*$ answers the instances. Let $\delta \in (0, 1]$. We require that for an $\alpha > 0$ fraction of the humans H, $\Pr_{x \leftarrow D}[H(x) = f(x)] > \delta$.

Definition 4. An AI problem $\mathcal{P}$ is said to be (δ, τ)-*solved* if there exists a program A, running in time at most τ on any input from S, such that

$$\Pr_{x \leftarrow D, r}[A_r(x) = f(x)] \geq \delta \ .$$

(A is said to be a (δ, τ) solution to $\mathcal{P}$.) $\mathcal{P}$ is said to be a (δ, τ)-*hard AI problem* if no current program is a (δ, τ) solution to $\mathcal{P}$, and the AI community agrees it is hard to find such a solution.

Definition 5. A (α, β, η)-CAPTCHA is a test V that is (α, β)-human executable, and which has the following property:

> There exists a (δ, τ)-hard AI problem $\mathcal{P}$ and a program A, such that if B has success greater than η over V then A^B is a (δ, τ) solution to $\mathcal{P}$. (Here A^B is defined to take into account B's running time too.)

We stress that V should be a *program* whose code is publicly available.

Remarks

1. The definition of an AI problem as a triple (S, D, f) should not be inspected with a philosophical eye. We are not trying to capture all the problems that fall under the umbrella of Artificial Intelligence. We want the definition to be easy to understand, we want some AI problems to be captured by it, and we want the AI community to agree that these are indeed hard AI problems. More complex definitions can be substituted for Definition 3 and the rest of the paper remains unaffected.
2. A crucial characteristic of an AI problem is that a certain fraction of the human population be able to solve it. Notice that we don't impose a limit on how long it would take humans to solve the problem. All that we require is that some humans be able to solve it (even if we have to assume they will live hundreds of years to do so). The case is not the same for CAPTCHAs. Although our definition says nothing about how long it should take a human to solve a CAPTCHA, it is preferable for humans to be able to solve CAPTCHAs in a very short time. CAPTCHAs which take a long time for humans to solve are probably useless for all practical purposes.

AI Problems as Security Primitives

Notice that we define *hard* in terms of the consensus of a community: an AI problem is said to be hard if the people working on it agree that it's hard. This notion should not be surprising to cryptographers: the security of most modern cryptosystems is based on assumptions agreed upon by the community (e.g., we *assume* that 1024-bit integers can't be factored). The concept of a hard AI problem as a foundational assumption, of course, is more questionable than $P \neq NP$, since many people in the AI community agree that all hard AI problems are eventually going to be solved. However, hard AI problems may be a more reasonable assumption than the hardness of factoring, given the possibility of constructing a quantum computer. Moreover, even if factoring is shown to be hard in an asymptotic sense, picking a concrete value for the security parameter usually means making an assumption about current factoring algorithms: we only *assume* that current factoring algorithms that run in current computers can't factor 1024-bit integers. In the same way that AI researchers believe that all AI problems will be solved eventually, we believe that at some point we will have the computational power and algorithmic ability to factor 1024-bit integers. (Shamir and Tromer [13], for instance, have proposed a machine that could factor 1024-bit integers; the machine would cost about ten million dollars in materials.)

An important difference between popular cryptographic primitives and AI problems is the notion of a security parameter. If we believe that an adversary can factor 1024-bit integers, we can use 2048-bit integers instead. No such concept exists in hard AI problems. AI problems, as we have defined them, do not deal with asymptotics. However, as long as there is a small gap between human and computer ability with respect to some problem, this problem can potentially be used as a primitive for security: rather than asking the prover to solve the problem once, we can ask it to solve the problem twice. If the prover gets good at solving the problem twice, we can ask it to solve the problem three times, etc.

There is an additional factor that simplifies the use of hard AI problems as security primitives. Most applications of CAPTCHAs require the tests to be answered within a short time after they are presented. If a new program solves the hard AI problems that are currently used, then a different set of problems can be used, and the new program cannot affect the security of applications that were run before it was developed. Compare this to encryption schemes: in many applications the information that is encrypted must remain confidential for years, and therefore the underlying problem must be hard against programs that run for a long time, and against programs that will be developed in the future.[1]

We also note that not all hard AI problems can be used to construct a CAPTCHA. In order for an AI problem to be useful for security purposes, there needs to be an *automated* way to generate problem instances along with their solution. The case is similar for computational problems: not all hard computational problems yield cryptographic primitives.

[1] We thank one of our anonymous Eurocrypt reviewers for pointing this out.

Who Knows What?

Our definitions imply that an adversary attempting to write a program that has high success over a CAPTCHA knows exactly how the CAPTCHA works. The only piece of information that is hidden from the adversary is a small amount of randomness that the verifier uses in each interaction.

This choice greatly affects the nature of our definitions and makes the problem of creating CAPTCHAs more challenging. Imagine an Automated Turing Test that owns a large secret book written in English and to test an entity A it either picks a paragraph from its secret book or generates a paragraph using the best known text-generation algorithm, and then asks A whether the paragraph makes sense (the best text-generation algorithms cannot produce an entire paragraph that would make sense to a human being). Such an Automated Turing Test might be able to distinguish humans from computers (it is usually the case that the best text-generation algorithms and the best algorithms that try to determine whether something makes sense are tightly related). However, this test cannot be a CAPTCHA: an adversary with knowledge of the secret book could achieve high success against this test without advancing the algorithmic state of the art. *We do not allow* CAPTCHA*s to base their security in the secrecy of a database or a piece of code.*

Gap Amplification

We stress that *any* positive gap between the success of humans and current computer programs against a CAPTCHA can be amplified to a gap arbitrarily close to 1 by *serial* repetition. The case for parallel repetition is more complicated and is addressed by Bellare, Impagliazzo and Naor in [3].

Let V be an (α, β, η)-CAPTCHA, and let V_k^m be the test that results by repeating V m times in series (with fresh new randomness each time) and accepting only if the prover passes V more than k times. Then for any $\epsilon > 0$ there exist m and k with $0 \leq k \leq m$ such that V_k^m is an $(\alpha, 1-\epsilon, \epsilon)$-CAPTCHA. In general, we will have $m = O(1/(\beta-\eta)^2 \ln(1/\epsilon))$ and sometimes much smaller. Since CAPTCHAs involve human use, it is desirable to find the smallest m possible. This can be done by solving the following optimization problem:

$$\min_m \left\{ \exists k : \sum_{i=k+1}^{m} \binom{m}{i} \beta^i (1-\beta)^{m-i} \geq 1-\epsilon \;\&\; \sum_{i=0}^{k} \binom{m}{i} \eta^i (1-\eta)^{m-i} \leq \epsilon \right\}$$

Notice that amplifying a gap can roughly be thought of as increasing the security parameter of a CAPTCHA: if the best computer program now has success 0.10 against a given CAPTCHA (for example), then we can ask the prover to pass the CAPTCHA twice (in series) to reduce the best computer program's success probability to 0.01.

3 Two AI Problem Families

In this section we introduce two families of AI problems that can be used to construct CAPTCHAs. The section can be viewed as a precise statement of the kind of hardness assumptions that our cryptographic protocols are based on. We stress that solutions to the problems will also be shown to be useful.

For the purposes of this paper, we define an *image* as an $h \times w$ matrix (h for height and w for width) whose entries are *pixels*. A pixel is defined as a triple of integers (R, G, B), where $0 \le R, G, B \le M$ for some constant M. An *image transformation* is a function that takes as input an image and outputs another image (not necessarily of the same width and height). Examples of image transformations include: turning an image into its black-and-white version, changing its size, etc.

Let $\mathcal{I}$ be a distribution on images and $\mathcal{T}$ be a distribution on image transformations. We assume for simplicity that if $i, i' \in [\mathcal{I}]$ and $i \neq i'$ then $T(i) \neq T'(i')$ for any $T, T' \in [\mathcal{T}]$. (Recall that $[\mathcal{I}]$ denotes the support of $\mathcal{I}$.)

Problem Family ($\mathcal{P}1$). Consider the following experiment: choose an image $i \leftarrow \mathcal{I}$ and choose a transformation $t \leftarrow \mathcal{T}$; output $t(i)$. $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ consists of writing a program that takes $t(i)$ as input and outputs i (we assume that the program has precise knowledge of $\mathcal{T}$ and $\mathcal{I}$). More formally, let $S_{\mathcal{I},\mathcal{T}} = \{t(i) : t \in [\mathcal{T}]$ and $i \in [\mathcal{I}]\}$, $D_{\mathcal{I},\mathcal{T}}$ be the distribution on $S_{\mathcal{I},\mathcal{T}}$ that results from performing the above experiment and $f_{\mathcal{I},\mathcal{T}} : S_{\mathcal{I},\mathcal{T}} \to [\mathcal{I}]$ be such that $f_{\mathcal{I},\mathcal{T}}(t(i)) = i$. Then $\mathcal{P}1_{\mathcal{I},\mathcal{T}} = (S_{\mathcal{I},\mathcal{T}}, D_{\mathcal{I},\mathcal{T}}, f_{\mathcal{I},\mathcal{T}})$.

Problem Family ($\mathcal{P}2$). In addition to the distributions $\mathcal{I}$ and $\mathcal{T}$, let L be a finite set of "labels". Let $\lambda : [\mathcal{I}] \to L$ compute the label of an image. The set of problem instances is $S_{\mathcal{I},\mathcal{T}} = \{t(i) : t \in [\mathcal{T}]$ and $i \in [\mathcal{I}]\}$, and the distribution on instances $D_{\mathcal{I},\mathcal{T}}$ is the one induced by choosing $i \leftarrow \mathcal{I}$ and $t \leftarrow \mathcal{T}$. Define $g_{\mathcal{I},\mathcal{T},\lambda}$ so that $g_{\mathcal{I},\mathcal{T},\lambda}(t(i)) = \lambda(i)$. Then $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda} = (S_{\mathcal{I},\mathcal{T}}, D_{\mathcal{I},\mathcal{T}}, g_{\mathcal{I},\mathcal{T},\lambda})$ consists of writing a program that takes $t(i)$ as input and outputs $\lambda(i)$.

Remarks

1. Note that a (δ, τ) solution A to an instance of $\mathcal{P}1$ also yields a $(\delta', \tau + \tau')$ solution to an instance of $\mathcal{P}2$ (where $\delta' \geq \delta$ and $\tau' \leq \log |[\mathcal{I}]|$ is the time that it takes to compute λ), specifically, computing $\lambda(A(x))$. However, this may be unsatisfactory for small δ, and we might hope to do better by restricting to a smaller set of labels. Conversely, $\mathcal{P}1$ can be seen as a special case of $\mathcal{P}2$ with λ the identity function and $L = [\mathcal{I}]$. Formally, problem families $\mathcal{P}1$ and $\mathcal{P}2$ can be shown to be isomorphic. Nonetheless, it is useful to make a distinction here because in some applications it appears unnatural to talk about labels.
2. We stress that in all the instantiations of $\mathcal{P}1$ and $\mathcal{P}2$ that we consider, $\mathcal{I}, \mathcal{T}$ and, in the case of $\mathcal{P}2$, λ, will be such that humans have no problem solving $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ and $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$. That is, $[\mathcal{T}]$ is a set of transformations that humans

can easily undo in $[\mathcal{I}]$. Additionally, it has to be possible to perform all the transformations in $[\mathcal{T}]$ using current computer programs.
3. There is nothing specific to *images* about these problem definitions; any other space of objects which humans recognize under reasonable transformations (e.g., organized sounds such as music or speech, animations, *et cetera*) could be substituted without changing our results.
4. It is easy to build a $(\delta_{\mathcal{I}}, \tau_{\mathcal{I}})$-solution to $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$, where $\delta_{\mathcal{I}} = \max\{\Pr_{j \leftarrow \mathcal{I}}[j = i] : i \in [\mathcal{I}]\}$ and $\tau_{\mathcal{I}}$ is the time that it takes to describe an element of $[\mathcal{I}]$, by always guessing the image with the highest probability in $\mathcal{I}$. Similarly, it is easy to build a $(\delta_{\mathcal{I},\lambda}, \tau_{\mathcal{I},\lambda})$-solution to $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$, where $\delta_{\mathcal{I},\lambda} = \max\{\Pr_{j \leftarrow \mathcal{I}}[\lambda(j) = \lambda(i)] : i \in [\mathcal{I}]\}$ and $\tau_{\mathcal{I},\lambda}$ is the time that it takes to describe a label in L. Therefore, we restrict our attention to finding solutions to $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ where $\delta > \delta_{\mathcal{I}}$ and solutions to $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ where $\delta > \delta_{\mathcal{I},\lambda}$.

Hard Problems in $\mathcal{P}1$ and $\mathcal{P}2$

We believe that $\mathcal{P}1$ and $\mathcal{P}2$ contain several hard problems. For example, the CAPTCHA shown in Section 1 (and other CAPTCHAs based on the difficulty of reading slightly distorted text) could be defeated using solutions to $\mathcal{P}2$. To see this, let W be a set of images of words in different fonts. All the images in W should be undistorted and contain exactly one word each. Let $\mathcal{I}_W$ be a distribution on W, let $\mathcal{T}_W$ be a distribution on image transformations, and let λ_W map an image to the word that is contained in it. A solution to $\mathcal{P}2_{\mathcal{I}_W,\mathcal{T}_W,\lambda_W}$ is a program that can defeat a CAPTCHA such as the one that Yahoo! uses (assuming $\mathcal{T}_W$ is the same set of transformations they use). So the problem of determining the word in a distorted image is an instantiation of $\mathcal{P}2$ (it can be easily seen to be an instantiation of $\mathcal{P}1$ too). Reading slightly distorted text has been an open problem in machine vision for quite some time. (For a good overview of the difficulties of reading slightly distorted text, see [12].)

But $\mathcal{P}1$ and $\mathcal{P}2$ are much more general, and reading slightly distorted text is a somewhat easy instance of these problems. In general it will not be the case that the problem is reduced to matching $26 * 2 + 10$ different characters (upper and lowercase letters plus the digits).

The hardness of problems in $\mathcal{P}1$ and $\mathcal{P}2$ mostly relies on $\mathcal{T}$. In particular, it should be computationally infeasible to enumerate all of the elements of $[\mathcal{T}]$, since $\mathcal{I}$ will normally be such that enumeration of $[\mathcal{I}]$ is feasible. Thus we are mainly interested in (δ, τ) solutions where $\tau \ll |[\mathcal{T}]|$, while $\tau > |[\mathcal{I}]|$ may sometimes be acceptable. In addition to the size of the transformation set, the character of the transformations is also important: it is necessary to defeat many simple checks such as color histogram comparisons, frequency domain checks, etc.

Since instantiations of $\mathcal{P}1$ and $\mathcal{P}2$ have never been precisely stated and published as challenges to the AI and security communities, there is no way to tell if they will withstand the test of time. For now we refer the reader to *www.captcha.net* for examples of $\mathcal{I}$'s and $\mathcal{T}$'s which are believed to be good candidates. Any instantiation of $\mathcal{P}1$ and $\mathcal{P}2$ for security purposes requires that the precise $\mathcal{I}$ and $\mathcal{T}$ be published and thoroughly described.

4 Two Families of CAPTCHAs

We now describe two families of CAPTCHAs whose security is based on the hardness of problems in $\mathcal{P}1$ and $\mathcal{P}2$. Notice that if $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ is (δ, τ)-hard then $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ can be used to construct a CAPTCHA trivially: the verifier simply gives the prover $t(i)$ and asks the prover to output i. According to our definition, this would be a perfectly valid CAPTCHA. However, it would also be a very impractical one: if $[\mathcal{I}]$ is large, then humans would take a long time to answer. The CAPTCHAs we present in this section can be quickly answered by humans. The first family of CAPTCHAs, MATCHA, is somewhat impractical, but the second family, PIX, is very practical and in fact several instantiations of it are already in use.

4.1 MATCHA

A MATCHA instance is described by a triple $M = (\mathcal{I}, \mathcal{T}, \tau)$, where $\mathcal{I}$ is a distribution on images and $\mathcal{T}$ is a distribution on image transformations that can be easily computed using current computer programs. MATCHA is a CAPTCHA with the following property: *any program that has high success over $M = (\mathcal{I}, \mathcal{T})$ can be used to solve $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$.*

The MATCHA verifier starts by choosing a transformation $t \leftarrow \mathcal{T}$. It then flips a fair unbiased coin. If the result is heads, it picks $k \leftarrow \mathcal{I}$ and sets $(i, j) = (k, k)$. If the result is tails, it sets $j \leftarrow \mathcal{I}$ and $i \leftarrow U([\mathcal{I}] - \{j\})$ where $U(S)$ is the uniform distribution on the set S. The MATCHA verifier sends the prover $(i, t(j))$ and sets a timer to expire in time τ; the prover responds with $res \in \{0, 1\}$. Informally, $res = 1$ means that $i = j$, while $res = 0$ means that $i \neq j$. If the verifier's timer expires before the prover responds, the verifier rejects. Otherwise, the verifier makes a decision based on the prover's response res and whether i is equal to j:

- If $i = j$ and $res = 1$, then MATCHA accepts.
- If $i = j$ and $res = 0$, then MATCHA rejects.
- If $i \neq j$ and $res = 1$, then MATCHA rejects.
- If $i \neq j$ and $res = 0$, then MATCHA plays another round.

In the last case, MATCHA starts over (with a fresh new set of random coins): it flips another fair unbiased coin, picks another pair of images (i, j) depending on the outcome of the coin, etc.

Remarks

1. It is quite easy to write a computer program that has success probability $1/2$ over MATCHA by simply answering with $res = 1$. For most applications, a distinguishing probability of $1/2$ is unacceptable. In order for MATCHA to be of practical use, the test has to be repeated several times.
2. Our description of MATCHA contains an obvious asymmetry: when $i = j$, MATCHA presents the prover with $(i, t(i))$ for $i \leftarrow \mathcal{I}$, and when $i \neq j$ MATCHA presents $(i, t(j))$, where i is chosen *uniformly* from the set $[\mathcal{I}] - \{j\}$. This gives the prover a simple strategy to gain advantage over M: if i seems to

come from $\mathcal{I}$, guess that $t(j)$ is a transformation of i; otherwise guess that it isn't. The reason for the asymmetry is to make the proof of Lemma 1 easier to follow. We note that a stronger CAPTCHA can be built by choosing i from the distribution $\mathcal{I}$ restricted to the set $[\mathcal{I}] - \{j\}$.

3. The intuition for why MATCHA plays another round when $i \neq j$ and $res = 0$ is that we are trying to convert high success against MATCHA into high success in solving $\mathcal{P}1$; a program that solves $\mathcal{P}1$ by comparing $t(j)$ to every image in $[\mathcal{I}]$ will encounter that most of the images in $[\mathcal{I}]$ are different from $t(j)$.
4. In the following Lemma we assume that a program with high success over M always terminates with a response in time at most τ. Any program which does not satisfy this requirement can be rewritten into one which does, by stopping after τ time and sending the response 1, which never decreases the success probability. We also assume that the unit of time that MATCHA uses is the same as one computational step.

Lemma 1. *Any program that has success greater than η over $M = (\mathcal{I}, \mathcal{T}, \tau)$ can be used to $(\delta, \tau|[\mathcal{I}]|)$-solve $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$, where*

$$\delta \geq \frac{\eta}{1 + 2|[\mathcal{I}]|(1-\eta)}.$$

Proof. Let B be a program that runs in time at most τ and has success $\sigma_B \geq \eta$ over M. Using B we construct a program A^B that is a $(\delta, \tau|[\mathcal{I}]|)$-solution to $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$.

The input to A^B will be an image, and the output will be another image. On input j, A^B will loop over the entire database of images of M (i.e., the set $[\mathcal{I}]$), each time feeding B the pair of images (i, j), where $i \in [\mathcal{I}]$. Afterwards, A^B collects all the images $i \in [\mathcal{I}]$ on which B returned 1 (i.e., all the images in $[\mathcal{I}]$ that B thinks j is a transformation of). Call the set of these images S. If S is empty, then A^B returns an element chosen uniformly from $[\mathcal{I}]$. Otherwise, it picks an element ℓ of S uniformly at random.

We show that A^B is a $(\delta, \tau|[\mathcal{I}]|)$-solution to $\mathcal{P}1$. Let $p_0 = \Pr_{\mathcal{T},\mathcal{I},r}[B_r(i, t(i)) = 0]$ and let $p_1 = \Pr_{\mathcal{T},j\leftarrow\mathcal{I},i,r}[B_r(i, t(j)) = 1]$. Note that

$$\Pr_{r,r'}[\langle M_r, B_{r'}\rangle = reject] = 1 - \sigma_B = \frac{p_0}{2} + \frac{p_1 + (1-p_1)(1-\sigma_B)}{2} = \frac{p_0 + p_1}{1 + p_1} \ ,$$

which gives $\sigma_B \leq 1 - p_0$ and $p_1 \leq 2(1 - \sigma_B)$. Hence:

$$\Pr_{\mathcal{T},\mathcal{I},r}[A_r^B(t(j)) = j] \geq \sum_{s=1}^{n} \Pr_{\mathcal{T},\mathcal{I},r}[A_r^B(t(j)) = j \,|\, |S| = s] \Pr_{\mathcal{T},\mathcal{I}}[|S| = s] \tag{1}$$

$$= \sum_{s=1}^{n} \frac{1 - p_0}{s} \Pr_{\mathcal{T},\mathcal{I}}[|S| = s] \tag{2}$$

$$\geq \frac{1 - p_0}{E_{\mathcal{T},\mathcal{I}}[|S|]} \tag{3}$$

$$\geq \frac{1 - p_0}{1 + |[\mathcal{I}]| p_1} \tag{4}$$

$$\geq \frac{\sigma_B}{1 + 2|[\mathcal{I}]|(1 - \sigma_B)} \tag{5}$$

(2) follows by the definition of the procedure A^B, (3) follows by Jensen's inequality and the fact that $f(x) = 1/x$ is concave, (4) follows because

$$E_{\mathcal{T},\mathcal{I}}[|S|] \leq 1 + \sum_{i \neq j} \Pr_{\mathcal{I},r}(B(i, t(j)) = 1)$$

and (5) follows by the inequalities for p_0, p_1 and σ_B given above. This completes the proof.

Theorem 1. *If $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ is $(\delta, \tau|[\mathcal{I}]|)$-hard and $M = (\mathcal{I}, \mathcal{T}, \tau)$ is (α, β)-human executable, then M is a $(\alpha, \beta, \frac{(2-|[\mathcal{I}]|)\delta}{2\delta - |[\mathcal{I}]|})$-CAPTCHA.*

4.2 PIX

An instance $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ can sometimes be used almost directly as a CAPTCHA. For instance, if $\mathcal{I}$ is a distribution over images containing a single word and λ maps an image to the word contained in it, then $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ can be used directly as a CAPTCHA. Similarly, if all the images in $[\mathcal{I}]$ are pictures of simple concrete objects and λ maps an image to the object that is contained in the image, then $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ can be used as a CAPTCHA.

Formally, a PIX instance is a tuple $X = (\mathcal{I}, \mathcal{T}, L, \lambda, \tau)$. The PIX verifier works as follows. First, V draws $i \leftarrow \mathcal{I}$, and $t \leftarrow \mathcal{T}$. V then sends to P the message $(t(i), L)$, and sets a timer for τ. P responds with a label $l \in L$. V accepts if $l = \lambda(i)$ and its timer has not expired, and rejects otherwise.

Theorem 2. *If $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ is (δ, τ)-hard and $X = (\mathcal{I}, \mathcal{T}, L, \lambda, \tau)$ is (α, β)-human executable, then X is a (α, β, δ)-CAPTCHA.*

Various instantiations of PIX are in use at major internet portals, like Yahoo! and Hotmail. Other less conventional ones, like ANIMAL-PIX, can be found at *www.captcha.net*. ANIMAL-PIX presents the prover with a distorted picture of a common animal (like the one shown below) and asks it to choose between twenty different possibilities (monkey, horse, cow, *et cetera*).

Fig. 2. Animal-Pix.

5 An Application: Robust Image-Based Steganography

We detail a useful application of (δ, τ)-solutions to instantiations of $\mathcal{P}1$ and $\mathcal{P}2$ (other than reading slightly distorted text, which was mentioned before). We hope to convey by this application that our problems were not chosen just because they can create CAPTCHAs but because they in fact have applications related to security. Our problems also serve to illustrate that there is a need for better AI in security as well. Areas such a Digital Rights Management, for instance, could benefit from better AI: a program that can find slightly distorted versions of original songs or images on the world wide web would be a very useful tool for copyright owners.

There are many applications of solutions to $\mathcal{P}1$ and $\mathcal{P}2$ that we don't mention here. $\mathcal{P}1$, for instance, is interesting in its own right and a solution for the instantiation when $\mathcal{I}$ is a distribution on images of works of art would benefit museum curators, who often have to answer questions such as "what painting is this a photograph of?"

Robust Image-Based Steganography.

Robust Steganography is concerned with the problem of covertly communicating messages on a public channel which is subject to modification by a restricted adversary. For example, Alice may have some distribution on images which she is allowed to draw from and send to Bob; she may wish to communicate additional information with these pictures, in such a way that anyone observing her communications can not detect this additional information. The situation may be complicated by an adversary who transforms all transmitted images in an effort to remove any hidden information. In this section we will show how to use (δ, τ)-solutions to instantiations of $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ or $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ to implement a secure robust steganographic protocol for image channels with distribution $\mathcal{I}$, when the adversary chooses transformations from $\mathcal{T}$. Note that if we require security for arbitrary $\mathcal{I}, \mathcal{T}$, we will require a (δ, τ)-solution to $\mathcal{P}1$ for arbitrary $\mathcal{I}, \mathcal{T}$; if no solution works for arbitrary $(\mathcal{I}, \mathcal{T})$ this implies the existence of specific $\mathcal{I}, \mathcal{T}$ for which $\mathcal{P}1$ is still hard. Thus either our stegosystem can be implemented by computers for arbitrary image channels or their is a (non-constructive) hard AI problem that can be used to construct a CAPTCHA.

The results of this subsection can be seen as providing an implementation of the "supraliminal channel" postulated by Craver [6]. Indeed, Craver's observation that the adversary's transformations should be restricted to those which do not significantly impact human interpretation of the images (because the adversary should not unduly burden "innocent" correspondents) is what leads to the applicability of our hard AI problems.

Steganography Definitions

Fix a distribution over images $\mathcal{I}$, and a set of keys $\mathcal{K}$. A *steganographic protocol* or *stegosystem* for $\mathcal{I}$ is a pair of efficient probabilistic algorithms (SE, SD) where $SE : \mathcal{K} \times \{0,1\} \to [\mathcal{I}]^\ell$ and $SD : \mathcal{K} \times [\mathcal{I}]^\ell \to \{0,1\}$, which have the additional property that $\Pr_{K,r,r'}[SD_r(K, SE_{r'}(K,\sigma)) \neq \sigma]$ is negligible (in ℓ and $|K|$) for any $\sigma \in \{0,1\}$. We will describe a protocol for transmitting a single bit $\sigma \in \{0,1\}$, but it is straightforward to extend our protocol and proofs by serial composition to any message in $\{0,1\}^*$ with at most linear decrease in security.

Definition 6. A stegosystem is *steganographically secret* for $\mathcal{I}$ if the distributions $\{SE_r(K,\sigma) : K \leftarrow \mathcal{K}, r \leftarrow \{0,1\}^*\}$ and $\mathcal{I}^\ell$ are computationally indistinguishable for any $\sigma \in \{0,1\}$.

Steganographic secrecy ensures that an eavesdropper cannot distinguish traffic produced by SE from $\mathcal{I}$. Alice, however, is worried about a somewhat malicious adversary who transforms the images she transmits to Bob. This adversary is restricted by the fact that he must transform the images transmitted between many pairs of correspondents, and may not transform them in ways so that they are unrecognizable to humans, since he may not disrupt the communications of legitimate correspondents. Thus the adversary's actions, on seeing the image i, are restricted to selecting some transformation t according to a distribution $\mathcal{T}$, and replacing i by $t(i)$. Denote by $t_{1\ldots\ell} \leftarrow \mathcal{T}^\ell$ the action of independently selecting ℓ transformations according to $\mathcal{T}$, and denote by $t_{1\ldots\ell}(i_{1\ldots\ell})$ the action of element-wise applying ℓ transformations to ℓ images.

Definition 7. A stegosystem (SE, SD) is *steganographically robust against* $\mathcal{T}$ if it is steganographically secret and

$$\Pr_{t_{1\ldots\ell} \leftarrow \mathcal{T}^\ell, r, r', K}[SD_r(K, t_{1\ldots\ell}(SE_r(K,\sigma))) \neq \sigma]$$

is negligible (in $|K|$ and ℓ) for any $\sigma \in \{0,1\}$.

Let $F : \mathcal{K} \times \{1,\ldots,\ell\} \times L \to \{0,1\}$ be a pseudorandom function family. We assume that $\lambda : [\mathcal{I}] \to L$ is efficiently computable and $A : S_{\mathcal{P}2} \to L$ is a (δ,τ)-solution to $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ as defined above (recall that $\mathcal{P}1$ is a special case of $\mathcal{P}2$), i.e. A operates in time τ and

$$\Pr_{t,i,r}[A_r(t(i)) = \lambda(i)] \geq \delta \ .$$

Let $c = \Pr_{i,j \leftarrow \mathcal{I}}[\lambda(i) = \lambda(j)]$. We *require* that $c < 1$ (that is, we require that there is enough variability in the labels of the images to be useful for communication). Notice that L can simply be equal to $[\mathcal{I}]$ and λ can be the identity function (in case the images in $[\mathcal{I}]$ have no labels as in $\mathcal{P}1$). We prove in the Appendix that the following construction is an efficient, robust stegosystem for $\mathcal{T}$.

Construction 1

Procedure SE:	**Procedure** SD:
Input: $K \in \mathcal{K}$, $\sigma \in \{0, 1\}$	**Input:** $K \in \mathcal{K}$, $i'_{1\ldots\ell} \in [\mathcal{I}]^\ell$
for $j = 1 \ldots \ell$ do	for $j = 1 \ldots l$ do
draw $d_0 \leftarrow \mathcal{I}$, $d_1 \leftarrow \mathcal{I}$	set $\sigma_j = F_K(j, A(i'_j))$
if $F_K(j, \lambda(d_0)) = \sigma$ then	**Output:** $majority(\sigma_1, \ldots, \sigma_\ell)$
set $i_j = d_0$	
else	
set $i_j = d_1$	
Output: $i_1, i_2, \ldots, i_\ell$	

Proposition 1. *Construction 1 is steganographically secret and robust for* $\mathcal{I}, \mathcal{T}$.

The proof of Proposition 1 is similar in flavor to those of Hopper, Langford and von Ahn [7] and relies on the fact that when A returns the correct solution on received image i'_j, the recovered bit σ_j is equal to the intended bit σ with probability approximately $\frac{1}{2} + \frac{1}{4}(1 - c)$ and otherwise $\sigma_j = \sigma$ with probability $1/2$; therefore the probability that the majority of the σ_j are incorrect is negligible in ℓ. For details, see the Appendix.

Remarks

1. Better solutions to $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ imply more efficient stegosystems: if δ is larger, then ℓ can be smaller and less images need to be transmitted to send a bit secretively and robustly.
2. Since we assume that $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ (or, as it might be the case, $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$) is easy for humans, our protocol *could* be implemented as a cooperative effort between the human recipient and the decoding procedure (without the need for a solution to $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ or $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$). However, decoding each bit of the secret message will require classifying many images, so that a human would likely fail to complete the decoding well before any sizeable hidden message could be extracted (this is especially true in case we are dealing with $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ and a large set $[\mathcal{I}]$: a human would have to search the entire set $[\mathcal{I}]$ as many as ℓ times for each transmitted bit). Thus to be *practical*, a (δ, τ)-solution (for small τ) to $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ or $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ will be required.

6 Discussion and Closing Remarks

Interaction with the AI community

A primary goal of the CAPTCHA project is to serve as a challenge to the Artificial Intelligence community. We believe that having a well-specified set of goals will contribute greatly to the advancement of the field. A good example of this process is the recent progress in reading distorted text images driven by the CAPTCHA in use at Yahoo!. In response to the challenge provided by this test, Malik and Mori [9] have developed a program which can pass the test with probability roughly 0.8. Despite the fact that this CAPTCHA has no formal proof that a program which can pass it can read under other distributions of image transformations, Malik and Mori claim that their algorithm represents significant progress in the general area of text recognition; it is encouraging to see such progress. For this reason, it is important that even Automated Turing Tests without formal reductions attempt to test ability in general problem domains; and even though these tests may have specific weaknesses it is also important that AI researchers attempting to pass them strive for solutions that generalize.

Other AI problem domains

The problems defined in this paper are both of a similar character, and deal with the advantage of humans in sensory processing. It is an open question whether CAPTCHAs in other areas can be constructed. The construction of a CAPTCHA based on a text domain such as text understanding or generation is an important goal for the project (as CAPTCHAs based on sensory abilities can't be used on sensory-impaired human beings). As mentioned earlier, the main obstacle to designing these tests seems to be the similar levels of program ability in text generation and understanding.

Logic problems have also been suggested as a basis for CAPTCHAs and these present similar difficulties, as generation seems to be difficult. One possible source of logic problems are those proposed by Bongard [4] in the 70s; indeed [1] presents a test based on this problem set. However, recent progress in AI has also yielded programs which solve these problems with very high success probability, exceeding that of humans.

Conclusion

We believe that the fields of cryptography and artificial intelligence have much to contribute to one another. CAPTCHAs represent a small example of this possible symbiosis. Reductions, as they are used in cryptography, can be extremely useful for the progress of algorithmic development. We encourage security researchers to create CAPTCHAs based on different AI problems.

Acknowledgments

We are greatful to Udi Manber for his suggestions. We also thank Lenore Blum, Roni Rosenfeld, Brighten Godfrey, Moni Naor, Henry Baird and the anonymous Eurocrypt reviewers for helpful discussions and comments. This work was partially supported by the National Science Foundation (NSF) grants CCR-0122581 and CCR-0085982 (The Aladdin Center). Nick Hopper is also partially supported by an NSF graduate research fellowship.

References

1. Luis von Ahn, Manuel Blum, Nicholas J. Hopper and John Langford. The CAPTCHA Web Page: `http://www.captcha.net`. 2000.
2. Luis von Ahn, Manuel Blum and John Langford. Telling Humans and Computers Apart (Automatically) or How Lazy Cryptographers do AI. To appear in *Communications of the ACM.*
3. Mihir Bellare, Russell Impagliazzo and Moni Naor. Does Parallel Repetition Lower the Error in Computationally Sound Protocols? In *38th IEEE Symposium on Foundations of Computer Science (FOCS' 97)*, pages 374-383. IEEE Computer Society, 1997.
4. Mikhail M. Bongard. *Pattern Recognition.* Spartan Books, Rochelle Park NJ, 1970.
5. A. L. Coates, H. S. Baird, and R. J. Fateman. Pessimal Print: A Reverse Turing Test. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR' 01)*, pages 1154-1159. Seattle WA, 2001.
6. Scott Craver. On Public-key Steganography in the Presence of an Active Warden. In *Proceedings of the Second International Information Hiding Workshop*, pages 355-368. Springer, 1998.
7. Nicholas J. Hopper, John Langford and Luis von Ahn. Provably Secure Steganography. In *Advances in Cryptology, CRYPTO' 02*, volume 2442 of *Lecture Notes in Computer Science*, pages 77-92. Santa Barbara, CA, 2002.
8. M. D. Lillibridge, M. Adabi, K. Bharat, and A. Broder. Method for selectively restricting access to computer systems. Technical report, US Patent 6,195,698. Applied April 1998 and Approved February 2001.
9. Greg Mori and Jitendra Malik. Breaking a Visual CAPTCHA. Unpublished Manuscript, 2002. Available electronically: `http://www.cs.berkeley.edu/~mori/gimpy/gimpy.pdf`.
10. Moni Naor. Verification of a human in the loop or Identification via the Turing Test. Unpublished Manuscript, 1997. Available electronically: `http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human.ps`.
11. Benny Pinkas and Tomas Sander. Securing Passwords Against Dictionary Attacks. In *Proceedings of the ACM Computer and Security Conference (CCS' 02)*, pages 161-170. ACM Press, November 2002.
12. S. Rice, G. Nagy, and T. Nartker. *Optical Character Recognition: An Illustrated Guide to the Frontier.* Kluwer Academic Publishers, Boston, 1999.
13. Adi Shamir and Eran Tromer. Factoring Large Numbers with the TWIRL Device. Unpublished Manuscript, 2003. Available electronically: `http://www.cryptome.org/twirl.ps.gz`.
14. J. Xu, R. Lipton and I. Essa. Hello, are you human. Technical Report GIT-CC-00-28, Georgia Institute of Technology, November 2000.

A Proof of Proposition 1

Lemma 1. *Construction 1 is steganographically secret.*

Proof. Consider for any $1 \leq j \leq \ell$ and $x \in [\mathcal{I}]$ the probability ρ_x^j that $i_j = x$, i.e. $\rho_x^j = \Pr[i_j = x]$. The image x is returned in the jth step only under one of the following conditions:

1. D_0: $d_0 = x$ and $F_K(j, \lambda(d_0)) = \sigma$; or
2. D_1: $d_1 = x$ and $F_K(j, \lambda(d_0)) = 1 - \sigma$

Note that these events are mutually exclusive, so that $\rho_x^j = \Pr[D_0] + \Pr[D_1]$. Suppose that we replace F_K by a random function $f : \{1, \ldots, \ell\} \times L \to \{0, 1\}$. Then we have that $\Pr_{f,d_0}[D_0] = \frac{1}{2}\Pr_{\mathcal{I}}[x]$ by independence of f and d_0, and $\Pr_{f,d_1}[D_1] = \frac{1}{2}\Pr_{\mathcal{I}}[x]$, by the same reasoning. Thus $\rho_x^j = \Pr_{\mathcal{I}}[x]$ when F_K is replaced by a random function. Further, for a random function the ρ_x^j are all independent. Thus for any $\sigma \in \{0, 1\}$ we see that $SE(K, \sigma)$ and $\mathcal{I}^\ell$ are computationally indistinguishable, by the pseudorandomness of F.

Lemma 2. *Construction 1 is steganographically robust for $\mathcal{T}$.*

Proof. Suppose we prove a constant bound $\rho > \frac{1}{2}$ such that for all j, $\Pr[\sigma_j = \sigma] > \rho$. Then by a Chernoff bound we will have that $\Pr[majority(\sigma_1, \ldots, \sigma_\ell) \neq \sigma]$ is negligible, proving the lemma.

Consider replacing F_K for a random K with a randomly chosen function $f : \{1, \ldots, \ell\} \times L \to \{0, 1\}$ in SE, SD. We would like to assess the probability $\rho_j = \Pr[\sigma_j = \sigma]$. Let A_j be the event $A(i'_j) = \lambda(i_j)$ and $\overline{A_j}$ be the event $A(i'_j) \neq \lambda(i_j)$, and write $\lambda(d_b^j)$ as l_b^j. We have the following:

$$\Pr_{f,i_j,t}[\sigma_j = \sigma] = \Pr[\sigma_j = \sigma | A_j] \Pr[A_j] + \Pr[\sigma_j = \sigma | \overline{A_j}] \Pr[\overline{A_j}] \tag{6}$$

$$= \delta \Pr[\sigma_j = \sigma | A_j] + \frac{1}{2}(1 - \delta) \tag{7}$$

$$= \delta(\Pr[f(j, l_0^j) = \sigma \text{ or } (f(j, l_0^j) = 1 - \sigma \text{ and } f(j, l_1^j)) = \sigma)] + \frac{1 - \delta}{2} \tag{8}$$

$$= \delta(\frac{1}{2} + \Pr[f(j, l_0^j) = 1 - \sigma \text{ and } f(j, l_1^j) = \sigma \text{ and } l_0^j \neq l_1^j]) + \frac{1 - \delta}{2} \tag{9}$$

$$= \delta(\frac{1}{2} + \frac{1}{4}(1 - c)) + \frac{1 - \delta}{2} \tag{10}$$

$$= \frac{1}{2} + \frac{\delta}{4}(1 - c) \tag{11}$$

Here (7) follows because if $A(i'_j) = l \neq \lambda(i_j)$ then $\Pr_f[f(j, l) = \sigma] = \frac{1}{2}$, and (8) follows because if $A(i'_j) = \lambda(i_j)$, then $f(j, A(i'_j)) = \sigma$ iff $f(j, \lambda(i_j)) = 0$; the expression results from expanding the event $f(j, \lambda(i_j)) = 0$ with the definition of i_j in the encoding routine.

For any constant $\delta > 0$, a Chernoff bound implies that the probability of decoding failure is negligible in ℓ when F_K is a random function. Thus the pseudorandomness of F_K implies that the probability of decoding failure for Construction 1 is also negligible, proving the lemma.