On Grey Levels in Random CAPTCHA Generation

Fraser Newton, Michael A. Kouritzin

Department of Mathematical and Statistical Sciences
University of Alberta
Edmonton, Alberta, Canada

ABSTRACT

A CAPTCHA is an automatically generated test designed to distinguish between humans and computer programs; specifically, they are designed to be easy for humans but difficult for computer programs to pass in order to prevent the abuse of resources by automated bots. They are commonly seen guarding webmail registration forms, online auction sites, and preventing brute force attacks on passwords.

In the following, we address the question: How does adding a grey level to random CAPTCHA generation affect the utility of the CAPTCHA? We treat the problem of generating the random CAPTCHA as one of random field simulation: An initial state of background noise is evolved over time using Gibbs sampling and an efficient algorithm for generating correlated random variables. This approach has already been found to yield highly-readable yet difficult-to-crack CAPTCHAs. We detail how the requisite parameters for introducing grey levels are estimated and how we generate the random CAPTCHA. The resulting CAPTCHA will be evaluated in terms of human readability as well as its resistance to automated attacks in the forms of character segmentation and optical character recognition.

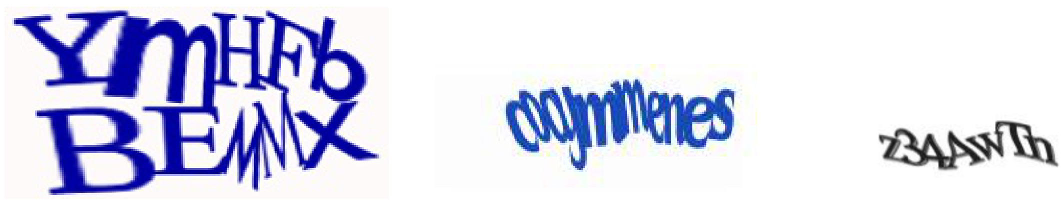
Keywords: Image processing, security, statistical information compression, Markov random field, simulation, detection.

1. INTRODUCTION

A CAPTCHA is a method of separating humans and computer programs; More specifically, a CAPTCHA is an automatically-generated challenge that, given a response, attempts to determine automatically if the response originated from a human or a computer program. CAPTCHAs are used to protect online resources such as webmail from abuse by computer programs, or "bots", and can also be used to prevent brute-force attacks against passwords.[1]

Optical character recognition (OCR) is a hard artificial intelligence (AI) problem which is commonly used as the basis of a CAPTCHA. Contemporary OCR programs continue to struggle with segmentation,[2] and even distorted single characters can be reliably recognized by a computer program with training.[3] In general, an image of a word, called the challenge word, is generated in a fashion that is difficult for a program to recognize while still being easy for a human to recognize. See Figure 1 for examples.

Further author information: (Send correspondence to F.N.)
F.N: E-mail: fnewton@math.ualberta.ca
M.A.K: E-mail: mkouritz@math.ualberta.ca



(a) Windows Live (b) Google (c) Yahoo!

Figure 1. CAPTCHA Examples

Visual Information Processing XX, edited by Zia-ur Rahman, Stephen E. Reichenbach, Mark A. Neifeld,
Proc. of SPIE Vol. 8056, 80560U · © 2011 SPIE · CCC code: 0277-786X/11/$18 · doi: 10.1117/12.884124

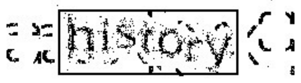
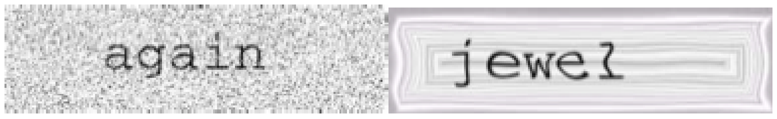


Figure 2. Example of a KNW-CAPTCHA of the word "history", with outline.



(a) (b)

Figure 3. Examples of EZ-Gimpy CAPTCHAs using grey levels.[6]

In Kouritzin *et al.*,[4] a method for generating CAPTCHAs using random field simulation was introduced. A KNW-CAPTCHA is generated using an efficient algorithm for simulating discrete random variables with given pixel-pixel covariances and marginal probabilities (further detailed in Section 2). The procedure is as follows.

1. Generate samples of the challenge word image by combining character images using random per-character vertical and horizontal displacement.
2. Estimate parameters from sample images of the challenge word.
3. Initialize the KNW-CAPTCHA with background noise using a ScatterType CAPTCHA, as developed in.[5]
4. Apply Gibbs-like sampling to re-simulate a given number of sites, based on the estimated parameters and the algorithm; the number of re-simulated sites impacts both attack resistance and human readability.

The hardened KNW-CAPTCHA was found to be very resistant to OCR attacks while maintaining high readability; in particular, the Tesseract OCR engine was unable to recognize any of the KNW-CAPTCHAs, while humans recognized 96.4% of them. The reader is referred to[4] for details of the algorithm as well as how attack resistance and human readability were estimated. See Figure 2 for an example of a KNW-CAPTCHA.

Herein, we extend the KNW-CAPTCHA to grey levels. Where we previously generated KNW-CAPTCHAs with only black and white, we now add a third level that will fall in between. The main goal of this work is to determine if adding a grey level to the KNW-CAPTCHA results in a more effective CAPTCHA, where effectiveness is a measure of both the attack resistance and human readability of the generated CAPTCHA. We hypothesize that adding grey levels to the KNW-CAPTCHAs will increase attack resistance by adding yet another dimension to the problem of OCR (i.e., how grey level should be interpreted), while providing more clues to a human reader about the character form and inter-character separation. The addition of grey levels requires significant modification to the KNW-CAPTCHA-generation procedure described in.[4] In particular, the sample word image samples used in parameter estimation require grey levels; this is accomplished by overlapping random character pairs, where the presence of black and grey is determined by the regions of overlap. Furthermore, the generation of background noise using ScatterType is also extended to include grey levels. Parameter estimation and the simulation of the KNW-CAPTCHA using Gibbs-like sampling follows in much the same way as in.[4]

Use of grey levels or colours in CAPTCHAs are well established in practice; for example, see Figure 3. However, we were unable to locate any research on quantifying the impact of the use of multiple colours or grey levels on the effectiveness of the CAPTCHA, which is what we set out to do in this work.

The main goals of this paper are to empirically assess the effectiveness of the KNW-CAPTCHA and determine if black and white or grey-level KNW-CAPTCHAs are the most effective, and to give a sensible approach to selecting KNW-CAPTCHA parameters to balance false positives with false negatives. Furthermore, we aim to provide a detailed analysis of the impact of the parameters in the KNW-CAPTCHA-generation process, including their relation to attack resistance and human readability.

The remainder of this work is laid out as follows. Section 2 details the changes to the work in[4] required to introduce grey levels to the KNW-CAPTCHA; Section 3 explains our procedure for optimizing the effectiveness of the KNW-CAPTCHA, as measured by a flexible cost function; Section 4 gives the results of the optimization procedure; a detailed analysis of the results are given in Section 5; human readability and attack resistance of the hardened grey-level KNW-CAPTCHA, which would be used in practice, is given in Section 6; finally, Section 7 contains our conclusion and discussion of further work.

2. METHOD OF GENERATING THE KNW-CAPTCHA

2.1 Method of Simulating a Random Field

In the following, we explain how we efficiently simulate a random field with given covariances and marginals. The intent is to make the basic idea clear, and the reader is referred to[4] and Kouritzin *et al.*,[7] which handles this in a more abstract setting, for details.

Next, we provide the required definitions in order to make sense of our simulation formula, equation (1) below. In this setting, the random field we want to simulate is a rectangular $M \times N$ image, made up of sites $S = \{(i,j) : 1 \leq i \leq M, 1 \leq j \leq N\}$. We divide the image into an unknown part $H \subset S$ and known part $H^C \doteq S \setminus H$.

We enumerate the sites S column by column: $s_1 = (1,1), s_2 = (2,1), \ldots, s_{MN} = (M,N)$. Now, we let L be the number of sites in H and enumerate the sites in H column by column, i.e.,

$$h_i = s_{m_i} \; \forall i \in \{1, 2, ..., L\},$$

where

$$m_i = \min\{j > m_{i-1} : s_j \in H\} \text{ and } m_0 = 0.$$

The sites in H will be simulated in the above order.

Next, define the ℓ-neighborhood of site $(i,j) \in S$

$$\partial_\ell((i,j)) = \{(u,v) \in S : 0 < \rho((i,j),(u,v)) \leq \ell\},$$

where $\rho((i,j),(u,v)) = \sqrt{(i-u)^2 + (j-v)^2}$. Then, for each k, we let

$$A_{h_k} \doteq \partial_\ell(h_k) \bigcap \left(\{h_1, h_2, ..., h_{k-1}\} \bigcup H^C\right),$$

i.e., A_{h_k} is the set of sites that are in the neighbourhood of h_k and were already known or have already been simulated.

Having established the prerequisite definitions, the following equation gives us exactly how to compute the conditional probability of a site given its neighbouring sites. Let $\mathbf{X} = \{-1, 0, 1\} = \{\text{white}, \text{grey}, \text{black}\}$ be the state space of each site in S, and $\mathcal{X}_A \doteq \prod_{s \in A} \mathbf{X}$ for $A \subset S$ $\mathcal{X} \doteq \mathcal{X}_S$. We construct the random field $X = (X_s)_{s \in S}$ on the canonical space $\mathcal{X}$ and let X_A denote the projection of X onto $\mathcal{X}_A$.

Assume the numbers on the RHS of (1) are in $[0,1]$. (The conditions for this to be true are given in.[7]) Then, there is a probability measure Π on $\mathcal{X}$ such that

$$\Pi(X_h = c) = \pi_h(c), \; \forall h \in H, \; c \in \mathbf{X},$$

and

$$\text{cov}(X_h, X_t) = \beta_{h,t}, \; \forall \; t \in \partial_l(h),$$

i.e., with correct marginals and covariances, and

$$\Pi(\, X_{h_i} = x_{h_i} \,|\, X_{A_{h_i}} = x_{A_{h_i}}) = \pi_{h_i}(x_{h_i}) + \frac{\sum\limits_{t_i \in A_{h_i}} (x_{h_i} - \bar{\mu})\beta_{h_i,t_i}(x_{t_i} - \bar{\mu})}{d^{|A_{h_i}|+1}(\bar{\sigma}^2)^2 \Pi(X_{A_{h_i}} = x_{A_{h_i}} | X_{H^C} = x_{H^C})} \tag{1}$$

for each $x_{h_i} \in \mathbf{X}$ and $x_{A_{h_i}} \in \mathcal{X}_{A_{h_i}}$ $(1 \le i \le n)$, where $\bar{\mu} = \frac{1}{d}\sum_{c\in\mathbf{X}} c$, $\bar{\sigma}^2 = \frac{1}{d}\sum_{c\in\mathbf{X}}(c-\bar{\mu})^2$ and $|A_{h_i}|$ is the cardinality of A_{h_i}.

Simulation of a particular site h_k follows immediately from equation (1) as follows.

1. Compute the value of $\Pi(X_{h_k} = c|X_{A_{h_k}} = x_{A_{h_k}})$ for $c \in \mathbf{X}$
2. Generate a $[0,1]$-uniform random number U to select which value to use for site h_k. In particular, if

$$\sum_{u=1}^{w-1} \Pi(\, X_{h_k} = c^u |\, X_{A_{h_k}} = x_{A_{h_k}}) \le U < \sum_{u=1}^{w} \Pi(\, X_{h_k} = c^u |\, X_{A_{h_k}} = x_{A_{h_k}})$$

for some $1 \le w \le d$, then we set $X_{h_k} = c^w$, i.e., the realization of X_{h_k} is colour $c^w \in \{-1, 0, 1\}$.

The procedure laid out in Section 2.3 requires simulating sites efficiently in a Gibbs-like manner. For the sake of brevity, it is sufficient to say that simulating a site s involves extracting a reduced image consisting of the site s and its neighbourhood. Working with this reduced image, every site is considered to be a neighbour of every other site; this allows for fast computation of the joint probability in the denominator of (1) using the multiplication rule. The reader is referred to[4] for the details of this method.

2.2 Parameter Estimation

In order to apply (1) to generating KNW-CAPTCHAs, we must first estimate the required parameters $\pi_h(c)$ $\forall h \in H$ and $\beta_{h,s}$ $\forall h \in H,\ s \in S$. The procedure is largely the same as in,[4] so we will give a brief description and highlight the differences. For a particular KNW-CAPTCHA and word, we estimate the parameters from a sample of K automatically generated word images. Each word image is constructed by assembling a series of character images. In order to ensure consistent placement of characters, we again work with trimmed images of {i,j,l,r,t} and scaled images of other characters in order to ensure all images of a particular character have the same width. In this context, trimming the character images means extra white space around the character is removed so all images of a particular character have a common width. For a given character string, we generate K $M \times N$ images representing that string. A particular image is constructed in the following way.

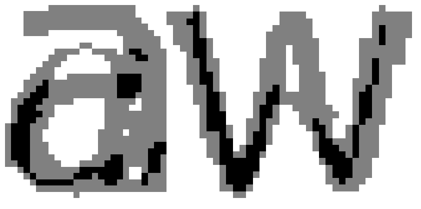
1. The horizontal distance between adjacent characters is randomly selected from $\{1, 2, 3\}$.
2. The vertical distance between adjacent characters is determined by a reflecting one dimensional random walk.
3. Each character image for the string is randomly selected over the available grey-level images of that character. (The grey-level images are described below.)
4. The character images are concatenated into one image according to the random horizontal and vertical placements determined above.

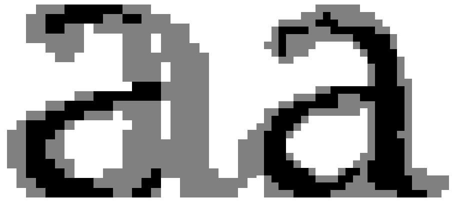
Let s_k^i denote the kth site in the ith word image in this sample. As in,[4] we use the unbiased estimators

$$\beta_{k,t} = \frac{1}{K-1}\sum_{i=1}^{K}(x_{s_k^i} - \bar{x}_{s_k})(x_{s_t^i} - \bar{x}_{s_t}),$$

where $\bar{x}_{s_k} = \frac{1}{K}\sum_{i=1}^{K} x_{s_k^i}$ is the empirical mean, for all $k, t = 1, ..., MN$ and

$$\pi_{s_k}(x_{s_k}) = \frac{1}{K}\sum_{i=1}^{K} 1_{x_{s_k^i} = x_{s_k}},$$


(a) "a" (b) "w"
Figure 4. Examples of grey-level character images.


(a) (b)
Figure 5. Examples of excluded grey-level character images.

where

$$1_{x_{s_k^i}=x_{s_k}} = \begin{cases} 1 & \text{if } x_{s_k^i} = x_{s_k} \\ 0 & \text{otherwise.} \end{cases}$$

Next, we establish exactly how the grey-level images are constructed from a set of black and white character images. Given two black and white images of the same character, we create a third grey-level image. Let $x^i_{s_k}$ denote the kth pixel of the ith black and white image and let x_{s_k} denote the kth pixel of the grey-level image. Set

$$x_{s_k} = \begin{cases} 1 & \text{if } x^i_{s_k} = 1 \text{ and } x^j_{s_k} = 1 \\ -1 & \text{if } x^i_{s_k} = -1 \text{ and } x^j_{s_k} = -1 \ , \\ 0 & \text{otherwise} \end{cases}$$

i.e., the new grey-level image will be black where the black and white images are both black, white where both are white, and grey otherwise. See Figure 4 for an example.

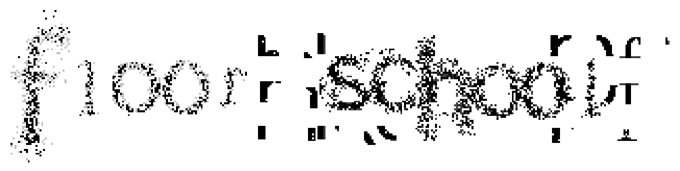
Remark 2.1. *An interesting feature of this process is that we are now working with character images that do not map easily to any particular font, which should have the effect of making feature detection or pattern matching much more difficult.*

Remark 2.2. *We automatically exclude certain grey-level images which may be easily recognized by OCR. For a given grey-level image, set all grey pixels to black; if this new image is recognized by the OCR program Tesseract, available at `http://code.google.com/p/tesseract-ocr/`, exclude the grey-level image from the database. Repeat this procedure with all grey pixels set to white. The intent of excluding these images is to help ensure the introduction of grey levels is not trivially bypassed by setting the grey pixels black or white. See Figure 5 for examples.*

It is clear from Figure 4 that the amount of overlap between the two black and white images will certainly have an effect on both readability and attack resistance. In the following, the amount of overlap for a particular grey-level image is calculated as $\frac{\sum_{s_k \in S} 1_{x_{s_k}=1}}{\sum_{s_k \in S} 1_{x_{s_k} \neq -1}}$, i.e., amount of black/amount not white. The effect of overlap will be analyzed in Section 5.

2.3 Generating a KNW-CAPTCHA

Now that we have estimated the parameters, we describe how to generate a KNW-CAPTCHA for a particular challenge. Again, the procedure is largely the same as in,[4] so we give a summary of the approach and highlight



(a) KNW-CAPTCHA$_{\mathrm{E}}$ (b) KNW-CAPTCHA$_{\mathrm{H}}$
Figure 6. Examples of grey-level KNW-CAPTCHAs.

the differences. Generating a KNW-CAPTCHA begins with initializing the image with background noise using an implementation of the ScatterType CAPTCHA. Background noise is introduced in order to further complicate attacks on the KNW-CAPTCHA by serving as *red herring* character shapes. From this initial state, we simulate the random field from the parameters estimated in Section 2.2. Simulation is accomplished site-by-site for a given number of sites via the Gibbs-like sampling described in 2.1. Two passes of the Gibbs-like sampling are used; the first pass is done with a lag of 0, i.e., all sites are considered independent; the second pass is done with a lag of 4. The first pass is used to help ensure that the site-site correlations introduced in the second pass are between sites representing the actual characters instead of between a character site and a site representing the background noise.

As in,[4] we use ScatterType to generate the background noise. In essence, for each character in a string, ScatterType cuts the character image randomly and scatters the resulting character pieces randomly; these scattered character images are then concatenated into a single word image.[5] In this work, we now use the grey-level character image database generated in 2.2; in fact, we also use grey-level character images that fall into the same target overlap range as our KNW-CAPTCHA. This is intended to ensure that the background noise and simulated character images are not obviously different, and so it is difficult to remove the background noise heuristically.

In the remainder of this paper, we will work with two variants of the KNW-CAPTCHA. The KNW-CAPTCHA$_{\mathrm{E}}$, an easy variant, is generated without background noise or vertical displacement of the characters; the KNW-CAPTCHA$_{\mathrm{H}}$, a hardened variant, is generated with both background noise and vertical displacement. The KNW-CAPTCHA$_{\mathrm{E}}$ is used in cases when we are attempting to measure the relative attack resistance of the generated CAPTCHAs; in our experience, OCR always fails to recognize KNW-CAPTCHA$_{\mathrm{H}}$s, and would provide no information about the relative attack resistance. While the failure rate on the KNW-CAPTCHA$_{\mathrm{E}}$s is still high, it still provides useful results. The KNW-CAPTCHA$_{\mathrm{H}}$ should be used in practice since it is more resistant to attacks See Sections 4 and 5 for results using KNW-CAPTCHA$_{\mathrm{E}}$s, and see Section 6 for readability and attack resistance results of KNW-CAPTCHA$_{\mathrm{H}}$s. See Figure 6 for examples of both.

3. PROCEDURE

We now present our procedure for determining which method of generating KNW-CAPTCHAs is the most effective. In essence, we want to determine if grey-levels provide us with a way of generating more effective KNW-CAPTCHAs. We conjecture that grey will both increase attack resistance and improve human readability. As stated previously, we work with KNW-CAPTCHA$_{\mathrm{E}}$s in order to capture the relative effectiveness of the CAPTCHAs; we will present results for KNW-CAPTCHA$_{\mathrm{H}}$s in Section 6.

Before proceeding, we must define "effective". How effectiveness of the CAPTCHA will depend on the goals of the user: are false positives or false negatives worse, or are the equally weighted? (Here, a false positive is claiming a human is a computer program; a false negative is claiming a computer program is a human.) The ideal but unrealistic result is to always deny access to computers, but always allow access to humans. In reality, there will be mistakes; furthermore, there will be a penalty associated with mistakenly allowing a program access (it can abuse whatever resource is being provided) as well as a penalty associated with mistakenly denying access to a human (the site becomes less usable, human visitors can become frustrated and less likely to visit or use the site).

Consider the following diverse examples of CAPTCHA usage. First, a CAPTCHA can be shown after 3 unsuccessful attempts at entering a password, which helps prevent brute force attacks. Second, on an online auction site, preventing bots from accessing the site and automatically bidding may be more important than occasionally denying access to a human since a bot can bid faster and more accurately than humans and could ruin the original intent of the site. Third, a CAPTCHA protecting a low-traffic niche forum might favour allowing humans over denying bots since it would be a low-reward target for a spam bot. We adopt a flexible definition of effective to account for the varying needs.

As previously mentioned, we will work with KNW-CAPTCHA$_\mathrm{E}$s, i.e., not hardened (no additional noise, no vertical displacement of characters). For each trial, we generate a KNW-CAPTCHA$_\mathrm{E}$ for a random character pair according to a set of randomized parameters. Let N_I be the number of sites re-simulated during the first pass of the Gibbs-like sampling, N_G be the number of sites re-simulated during the second pass of the Gibbs-like sampling, $g \in [0, 1]$ be the grey colour used (lower is darker), and $[o_{\min}, o_{\max})$ be the acceptable overlap range for the grey character images used. For each black and white KNW-CAPTCHA$_\mathrm{E}$, N_I and N_G will be selected randomly from $\{300, 400, 500\}$. For each grey-level KNW-CAPTCHA$_\mathrm{E}$, N_I and N_G will be selected randomly from $\{300, 400, 500\}$, g will be selected randomly from $\{0.25, 0.5, 0.75\}$, and $o_{\min}$ will be selected randomly from $\{0, 0.5\}$ and set $o_{\max} = o_{\min} + 0.5$.

We work with KNW-CAPTCHA$_\mathrm{E}$ here since we want to be able to select the best "base" CAPTCHA, then harden it; in addition, it is unlikely that Tesseract would recognize any of the KNW-CAPTCHA$_\mathrm{H}$s, which would make the results useless. We work with random character pairs to generate the minimal, interesting CAPTCHA: single characters do not exercise segmentation-based resistance (which is mainly accomplished through character crowding); words introduce other aspects such as linguistic analysis on the part of Tesseract (see Smith[8] for an overview of Tesseract), as well as being more easily recognizable for humans; however, random character pairs exercise segmentation resistance while also not being easier to recognize due to linguistic analysis. We work with a reduced parameter space based partly on prior work in[4] as well as visual inspection of the generated KNW-CAPTCHAs. The selected parameter ranges were determined to yield generally readable KNW-CAPTCHA's without being trivial or impossible to crack (i.e., the generated KNW-CAPTCHAs are interesting).

Selecting the most effective KNW-CAPTCHA is now a matter of minimizing a cost function. We use the cost function

$$f(\theta) = w_t \cdot t(\theta) - w_h \cdot h(\theta),$$

where θ are the parameters, $t(\theta)$ is the probability of a KNW-CAPTCHA generated by parameters θ being recognized by the Tesseract OCR program, and $h(\theta)$ is the probability of that KNW-CAPTCHA being recognized by a human, and $w_t, w_h \in [0, 1]$ are weights to adjust the relative importance of false positives and false negatives. For example, setting $w_t = 1, w_h = 0.5$ would make denying access to a computer program twice as important as successfully allowing access to a human. Our goal is to find the θ that minimizes the cost function, i.e., we would like $t(\theta)$, the probability of the KNW-CAPTCHA being cracked, to be low, and we would like $h(\theta)$ to be high.

Of course, we don't know the true functions $t(\theta)$ and $h(\theta)$, so we work with the estimates of the functions $\hat{t}(\theta)$ and $\hat{h}(\theta)$, respectively. First, we estimate $t(\theta)$. For a particular set of parameters θ, model the experimental trials as independent and identically distributed (i.i.d.) $t(\theta)$-Bernoulli random variables. Let

$$y_i = \begin{cases} 1 & \text{if Tesseract's } i^{\text{th}} \text{ response was correct} \\ 0 & \text{otherwise} \end{cases},$$

where the response correctness is determined using a case-insensitive comparison. For a particular set of parameters θ, the experiment yields n_T results, $\{y_1, y_2, \ldots, y_{n_T}\}$. We use parameter-by-parameter maximum likelihood estimator $\hat{t}(\theta)$ for $t(\theta)$, i.e.,

$$\hat{t}(\theta) = \frac{1}{n_T} \sum_{i=1}^{n_T} y_i.$$

We determine $\hat{h}(\theta)$ in exactly the same manner.

Remark 3.1. *We use the MLE for each unique set of parameters to avoid making any assumptions about the shape of $f(\theta)$ during the optimization process.*

Now we optimize the estimated cost function

$$\hat{f}(\theta) = w_t \cdot \hat{t}(\theta) - w_h \cdot \hat{h}(\theta),$$

i.e., we want to find θ^*, the set of parameters which minimizes $\hat{f}(\theta)$. For each unique set of parameters $\theta \in \Theta$, where Θ is the parameter space, determine $\hat{f}(\theta)$ and choose the parameters which minimize it, i.e.,

$$\theta^* = \text{argmin}_{\theta \in \Theta} \hat{f}(\theta)$$

4. RESULTS

We now present the results of the procedure laid out in Section 3. We give the most effective black and white KNW-CAPTCHA$_\text{E}$ and the most effective grey-level KNW-CAPTCHA$_\text{E}$ for the following cost functions: $w_t = 1, w_h = 1$, where human success and attack failure are considered equally important (Table 1); $w_t = 0.5, w_h = 1$, where human success is considered more important than attack failure (Table 2); and $w_t = 1, w_h = 0.5$, where attack failure is considered more important than human success (Table 3). Similarly, we also include results for $w_t = 0.25, w_h = 1$ (Table 4), $w_t = 1, w_h = 0.25$ (Table 5), $w_t = 0, w_h = 1$ (Table 6), $w_t = 1, w_h = 0$ (Table 7); the final two sets of weights are the edge cases that we only care about human readability and only care about attack resistance, respectively. These cost functions all balance the trade-offs of security and human usability differently.

KNW-CAPTCHA$_\text{E}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 400, N_G = 400$	0.0082	0.9714	-0.9633
Grey-level	$N_I = 500$, $N_G = 400$, $g = 0.75$, $o_{\min} = 0.50$, $o_{\max} = 1.00$	0.0000	0.9860	-0.9860

Table 1. Optimization results with $w_t = 1.00, w_h = 1.00$

KNW-CAPTCHA$_\text{E}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 500, N_G = 400$	0.0298	0.9915	-0.9766
Grey-level	$N_I = 500$, $N_G = 400$, $g = 0.75$, $o_{\min} = 0.50$, $o_{\max} = 1.00$	0.0000	0.9860	-0.9860

Table 2. Optimization results with $w_t = 0.50, w_h = 1.00$

KNW-CAPTCHA$_\text{E}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 400, N_G = 400$	0.0082	0.9714	-0.4776
Grey-level	$N_I = 500$, $N_G = 400$, $g = 0.75$, $o_{\min} = 0.50$, $o_{\max} = 1.00$	0.0000	0.9860	-0.4930

Table 3. Optimization results with $w_t = 1.00, w_h = 0.50$

Section 5 contains a detailed analysis of results; however, it is immediately clear that the cost function offers a flexible way of selecting a CAPTCHA-generation method, and that the exact choice of w_t, w_h affects the recommended CAPTCHA; in particular, there does not appear to be a universally better KNW-CAPTCHA$_\text{E}$ (i.e., one which maximizes attack resistance and human readability simultaneously). Also, in most cases, the recommended CAPTCHA was the grey-level KNW-CAPTCHA$_\text{E}$; in fact, the sole exception is the edge case $w_t = 0, w_h = 1$, which would not normally be deployed in practice. Furthermore, the grey-level KNW-CAPTCHA$_\text{E}$ was recommended over the black and white KNW-CAPTCHA$_\text{E}$ in the case that $w_t = 1, w_t = 1$; indeed, in this case, the grey-level version outperformed the black and white version in both readability and attack resistance.

KNW-CAPTCHA$_\text{E}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 500, N_G = 400$	0.0298	0.9915	-0.9840
Grey-level	$N_I = 500$, $N_G = 300$, $g = 0.50$, $o_\text{min} = 0.50$, $o_\text{max} = 1.00$	0.0058	0.9884	-0.9870

Table 4. Optimization results with $w_t = 0.25, w_h = 1.00$

KNW-CAPTCHA$_\text{E}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 400, N_G = 300$	0.0040	0.9622	-0.2366
Grey-level	$N_I = 500$, $N_G = 400$, $g = 0.75$, $o_\text{min} = 0.50$, $o_\text{max} = 1.00$	0.0000	0.9860	-0.2465

Table 5. Optimization results with $w_t = 1.00, w_h = 0.25$

5. ANALYSIS

In the following, we set out to understand how the parameters controlling the generation of the KNW-CAPTCHA$_\text{E}$ impact both the human readability and attack resistance of the resulting CAPTCHA. The following analysis uses logistic regression. Let

$$y_i = \begin{cases} 1 & \text{if } i^\text{th} \text{ response was correct} \\ 0 & \text{otherwise} \end{cases},$$

and we model y_i as independent (μ_i)-Bernoulli trials. We assume that

$$\log(\frac{\mu_i}{1-\mu_i}) = \beta_0 + \beta_1 x_{1,i} + \cdots + \beta_k x_{k,i}.$$

Our task is to estimate β_j, $j = 1, \ldots, k$ and determine the covariates which have a significant impact on the probability of the response being correct, i.e., μ_i. In the following, $x_{1,i}, \ldots, x_{1,k}$ will be the values of the parameters, and we will consider any coefficient with a p-value less than 0.05 to be significant. We will perform the analysis on the responses from both Tesseract and humans.

As is clear from Tables 8 and 10, the effect of increasing N_I and N_G increases the probability that Tesseract will recognize a given KNW-CAPTCHA$_\text{E}$. However, as Tables 9 and 11 show, the effect of increasing N_I and N_G is to also increase the probability that a human will recognize a given KNW-CAPTCHA$_\text{E}$. (While N_G is not significant at a 0.05 level in the case of human responses on grey-level KNW-CAPTCHA$_\text{E}$s, we suspect that it would become significant with more data.) Thus, N_I and N_G increase the probability of both Tesseract and humans recognizing the KNW-CAPTCHA$_\text{E}$s, and the effects appear to be roughly the same magnitude. As such, this parameter does not appear to aid in distinguishing between humans and computer programs.

Interestingly, increasing g (i.e., using a lighter shade of grey) appears to decrease the probability of Tesseract recognizing the grey-level KNW-CAPTCHA$_\text{E}$, yet does not have a significant effect on human readability, as can be seen in Tables 10 and 11, respectively. This indicates that adjusting the grey level parameter should allow us to design KNW-CAPTCHAs with higher attack resistance without significantly impacting human readability.

Similarly, it appears that the choice of overlap range significantly impacts human readability (i.e., the more overlap between characters, the higher the readability), but does not significantly impact Tesseract's ability to recognize the KNW-CAPTCHA$_\text{E}$s, as seen in Tables 10 and 11, respectively. This indicates that adjusting the acceptable overlap range can also serve to further differentiate between computer programs and humans.

In summary, it appears that grey-level KNW-CAPTCHAs introduce two new parameters, the grey level and the acceptable overlap range, that allow for greater flexibility and power when compared with the black and white KNW-CAPTCHA.

6. KNW-CAPTCHA$_\text{H}$

Next, we present the attack resistance and readability results of the grey-level KNW-CAPTCHA$_\text{H}$. We generate the KNW-CAPTCHA$_\text{H}$ using the parameters for the most effective grey-level KNW-CAPTCHA$_\text{E}$ on the cost

KNW-CAPTCHA$_\mathrm{E}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 500, N_G = 500$	0.0824	0.9945	-0.9945
Grey-level	$N_I = 500$, $N_G = 300$, $g = 0.50$, $o_{\min} = 0.50$, $o_{\max} = 1.00$	0.0058	0.9884	-0.9884

Table 6. Optimization results with $w_t = 0.00, w_h = 1.00$

KNW-CAPTCHA$_\mathrm{E}$ Type	Parameters θ^*	$\hat{t}(\theta^*)$	$\hat{h}(\theta^*)$	$\hat{f}(\theta^*)$
Black and White	$N_I = 400, N_G = 300$	0.0040	0.9622	0.0040
Grey-level	$N_I = 300$, $N_G = 400$, $g = 0.50$, $o_{\min} = 0.00$, $o_{\max} = 0.50$	0.0000	0.8361	0.0000

Table 7. Optimization results with $w_t = 1.00, w_h = 0.00$

function with $w_t = 1, w_h = 1$. We present these results in order to compare the KNW-CAPTCHA$_\mathrm{H}$ effectiveness with that of the black and white KNW-CAPTCHA$_\mathrm{H}$ presented in.[4] Results are given in Table 12.

As expected, Tesseract is unable to recognize any of the generated KNW-CAPTCHA$_\mathrm{H}$s. Surprisingly, however, the human readability is significantly lower than the results on the black and white KNW-CAPTCHA$_\mathrm{H}$ (which obtained a $\hat{h}(\theta)$ of 0.96). It appears that the introduction of background noise is more likely to interfere with the readability when working with the grey-level KNW-CAPTCHAs; this may be due to the increased complexity of the background noise (i.e., the introduction of grey levels into the ScatterType CAPTCHA) when compared with the black and white KNW-CAPTCHA$_\mathrm{H}$s. However, we expect that the readability of the grey-level KNW-CAPTCHA$_\mathrm{H}$ can be increased significantly by adjusting the acceptable overlap range, N_G, and N_I. For now, those wishing to favour human readability over attack resistance may prefer to deploy the KNW-CAPTCHA$_\mathrm{E}$.

7. CONCLUSION

- KNW-CAPTCHAs are effective distinguishers of humans over computer programs.
- Adding a grey level to a KNW-CAPTCHA can increase the CAPTCHA's power.
- The actual grey level as well as the amount of grey compared to black are two of the most significant parameters to KNW-CAPTCHA power.
- Adding grey-level background noise makes both human readability and computer recognizability harder over black and white.
- Even grey-level KNW-CAPTCHAs are easy to implement in real time using the given algorithm and they provide effective protection.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous volunteers who participated in the readability study found at `http://www.knwcaptcha.org`.

REFERENCES

[1] Von Ahn, L., Blum, M., Hopper, N., and Langford, J., "CAPTCHA: Using hard AI problems for security," *Lecture notes in computer science* , 294–311 (2003).

[2] Yan, J. and El Ahmad, A., "A Low-cost Attack on a Microsoft CAPTCHA," in [*Proceedings of the 15th ACM conference on Computer and communications security*], 543–554, ACM (2008).

[3] Chellapilla, K., Larson, K., Simard, P., and Czerwinski, M., "Computers beat humans at single character recognition in reading based human interaction proofs (HIPs)," in [*Proceedings of the Second Conference on Email and Anti-Spam*], 21–22, Citeseer (2005).

Covariate	Estimate	p-value	Significant?
Intercept	$-1.03e+01$	$8.29e-20$	Yes
N_G	$8.67e-03$	$2.77e-08$	Yes
N_I	$6.83e-03$	$3.69e-04$	Yes

Table 8. Tesseract Responses on Black and White KNW-CAPTCHA$_{\mathrm{E}}$

Covariate	Estimate	p-value	Significant?
Intercept	$1.14e+00$	$1.55e-01$	No
N_G	$2.67e-03$	$2.12e-02$	Yes
N_I	$3.42e-03$	$4.05e-02$	Yes

Table 9. Human Responses on Black and White KNW-CAPTCHA$_{\mathrm{E}}$

[4] Kouritzin, M., Newton, F., and Wu, B., "On Random Field CAPTCHA Generation," *Submitted* .

[5] Baird, H. and Riopka, T., "ScatterType: A reading CAPTCHA resistant to segmentation attack," in [*Proc. SPIE*], **5676**(1), 197–201, Citeseer (2005).

[6] Mori, G. and Malik, J., "Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA," (2003).

[7] Wu, B., Newton, F., and Kouritzin, M., "On Simulating Correlated Random Fields," *In Final Preparation* .

[8] Smith, R., "An overview of the Tesseract OCR engine," in [*Ninth International Conference on Document Analysis and Recognition, 2007. ICDAR 2007*], **2** (2007).

Covariate	Estimate	p-value	Significant?
Intercept	$-2.66e+01$	$9.74e-01$	No
N_G	$7.19e-03$	$1.14e-03$	Yes
N_I	$9.56e-03$	$1.64e-02$	Yes
g	$-3.45e+00$	$1.53e-04$	Yes
$I_{[o_{\min},o_{\max})=[0.5,1.0)}$	$1.65e+01$	$9.84e-01$	No

Table 10. Tesseract Responses on Grey-level KNW-CAPTCHA$_{\mathrm{E}}$

Covariate	Estimate	p-value	Significant?
Intercept	$4.58e-01$	$3.94e-01$	No
N_G	$9.56e-04$	$1.79e-01$	No
N_I	$3.92e-03$	$6.10e-05$	Yes
g	$-4.27e-01$	$2.62e-01$	No
$I_{[o_{\min},o_{\max})=[0.5,1.0)}$	$1.14e+00$	$1.34e-19$	Yes

Table 11. Human Responses on Grey-level KNW-CAPTCHA$_{\mathrm{E}}$

Parameters	$\hat{t}(\theta)$	$\hat{h}(\theta)$
$N_I = 500, N_G = 400, g = 0.75, o_{\min} = 0.50, o_{\max} = 1.00$	0.0000	0.8639

Table 12. Human Readability of KNW-CAPTCHA$_{\mathrm{H}}$