ON RANDOM FIELD CAPTCHA GENERATION

MICHAEL A. KOURITZIN*, FRASER NEWTON, AND BIAO WU

Abstract. Herein, we propose generating CAPTCHAs through random field simulation and give a novel, effective and efficient algorithm to do so. Indeed, we demonstrate that sufficient information about word tests for easy human recognition is contained in the site marginal probabilities and the site-to-nearby-site covariances and these quantities can be embedded into KNW conditional probabilities, designed for effective simulation. The CAPTCHAs are then partial random realizations of the random CAPTCHA word: we start with an initial random field (e.g., randomly scattered letter pieces) and use Gibbs resampling to re-simulate portions of the field repeatedly using the KNW conditional probabilities until the word becomes human-readable. The residual randomness from the initial random field together with the random implementation of the CAPTCHA word provide significant resistance to attack. This results in a CAPTCHA which is unrecognizable to modern OCR but is recognized about 95% of the time in a human readability study.

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

1. Introduction

A CAPTCHA is a "Completely Automated Public Turing test to tell Computers and Humans Apart" von Ahn *et al.* [1], widely used to protect online resources from abuse by automated agents. Von Ahn *et al.* [2] suggests that hard artificial intelligence (AI) problems form the test basis and defines a (α, β, η)-CAPTCHA as a test that 1) can be solved by at least α proportion of humans (e.g., the English-speaking adult portion) with a probability of success greater than β; 2) if a computer program can solve it with probability greater than η in fixed time, then the

M.A. Kouritzin, F. Newton, and B. Wu are with the Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, Alberta, T6G 2G1 Canada e-mails: mkouritz@math.ualberta.ca, fnewton@math.ualberta.ca, biao.wu@ualberta.ca.

*The author gratefully acknowledges support from NSERC through a Discovery Grant.

program can be used to solve the hard AI problem (see [2] for details). A common CAPTCHA is an image of (usually alphanumeric) characters to that are easy to identify by English-reading humans yet translate into the hard AI problem of optical character recognition (OCR). Segmentation of characters within a word image is error prone [3], and continues to be difficult for contemporary OCR algorithms [4]. Therefore, segmentation should be hard to ensure an OCR-based CAPTCHA is resistant to computer programs.

Herein, we introduce a general method for generating "KNW-CAPTCHAs" with the view that random CAPTCHA creation is really random field simulation ("KNW-CAPTCHAs" are pronounced KNOW CAPTCHAs, meaning people rather than computers will know what they are, whereas the other computer resistant CAPTCHAs are apparently people resistant too). We simulate random fields with given pixel marginal probabilities and pixel-pixel correlations, which are estimated from a priori samples with random variations in the fonts and placement of letters. This can be thought of as a form of lossy compression: while the complete information is the joint distribution, we store only the marginal probabilities and covariances, from which a (possibly different) joint distribution can be reconstructed. However, we simulate directly from the marginal probabilities and covariances. A KNW-CAPTCHA is initialized as a random field, and the CAPTCHA is then generated via partial Gibbs re-sampling in order to provide enough information to make the test word human-recognizable, yet ensure that OCR remains hard. Perhaps our most important contribution is our method to simulate these random field CAPTCHAs in real time. In contrast to other methods which apply deformations to an initial word image, the KNW-CAPTCHA is a partial evolution from OCR-disruptive noise towards a random word image.

For an effective (α, β, η)-CAPTCHA, β should be high and η should be low. The target population for our KNW-CAPTCHAs is English-readers with better than 20/60 vision (though we have little control over the participants in our readability studies). We establish high β via a readability study and endorse low η via experiments with modern OCR programs.

We begin with an overview of past and present text-based CAPTCHAs. (While there are many alternatives to text-based CAPTCHAs, such as the image-based IMAGINATION [5], which requires users to annotate images, text-based CAPTCHAs continue to be the de facto standard in industry.) The early, now broken PayPal and the Microsoft CAPTCHAs discussed in [6] and [4], respectively, both relied on background noise and random character strings to resist automated attacks but did not employ character crowding, significant distortion, nor sophisticated random field techniques. The background noise (random arcs in [4] - see Figure 1) was trivial to remove due to its distinctiveness.

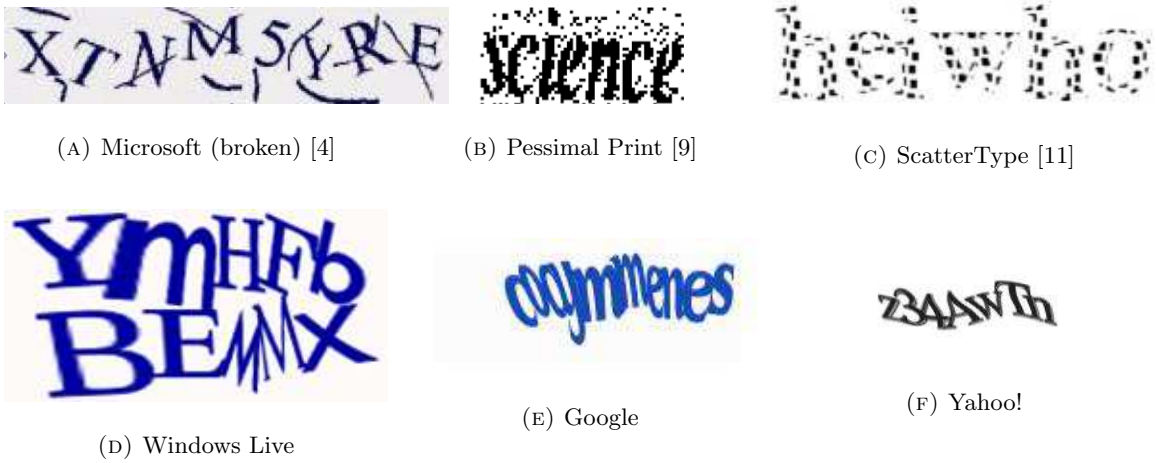
Mori *et al.* [7] successfully attack both EZ-Gimpy and Gimpy CAPTCHAs. EZ-Gimpy uses word images, and employs clutter and character distortion to defend against attacks. However, it does not employ character crowding. The authors of [7] make use of character shape contexts in order to obtain many candidate letter locations and exploit EZ-Gimpy's use of words. Gimpy's clutter is two distorted overlapping word images (chosen from a dictionary of 411). In a CAPTCHA challenge, five pairs of overlapping words are presented. In [7], the authors determine the opening and closing bigrams of each word and use this knowledge to prune the space of possible words. Further pruning is accomplished using word-sized shape contexts. *Moy et al.* [8] break EZ-Gimpy and Gimpy-r. Gimpy-r presents the user with four random, distorted character images from an alphabet of 19 letters against a cluttered background. It does not, however, use character crowding nor random field techniques to impede segmentation. The authors of [8] are able to remove the background clutter and segment the challenge into four character recognition problems, which are solved by determining which template character image requires the least distortion to match the observed character image. (Performance is further improved using additional steps.)

Pessimal Print (see Figure 1b), introduced in Coates *et al.* [9], simulates low-quality print images that challenge OCR. The CAPTCHA generation randomly selects a word, a font, and a set of image degradation parameters to thicken, crowd, fragment, and add noise to character images.

685 word images were generated; all were readable to the ten human volunteers, while almost all were unrecognizable to the Expervision TR, ABBYY FineReader, and IRIS Reader OCR programs. Furthermore, OCR performance was very sensitive to changes in the parameters.

Chew et al.'s [10] BaffleText CAPTCHA relies on a human's Gestalt perception, i.e., the ability to assemble the whole given fragments of an image. BaffleText generates pronounceable non-English random character strings, displayed in a randomly selected font and masked by random circles, squares, and ellipses using one of the pixel-wise boolean operations "or", "not and", or "exclusive or". Character strings are generated using a trigram Markov model to solve the small dictionary problem that can plague English word-based CAPTCHAs; random masks are used over simple additive pixel noise in order to exercise humans' Gestalt perception. Human readability results were collected from 33 volunteers on 1212 BaffleText images, with 79% success. Attack resistance is established by subjecting BaffleText images to the attack described in [7]. The attack succeeded on only 11% of the BaffleText images, lower than both Pessimal Print and EZ-Gimpy. The ScatterType CAPTCHA (see Figure 1c), introduced in Baird *et al.* [11], also relies on Gestalt perception. Pseudo-words are generated using an n-gram Markov model; then each character in the word is cut vertically and horizontally and the resulting fragments are displaced randomly.

Finally, we examine some popular CAPTCHAs in use today. The CAPTCHAs used by Google, Yahoo!, and Windows Live (see Figure 1) all share similar properties: a lack of background noise, distortion of character or word images, and extreme crowding of adjacent characters. Segmentation resistance is largely accomplished by character crowding, notably lacking from earlier, now broken CAPTCHAs such as the captchaservice.org CAPTCHAs in [12], the PayPal CAPTCHA in [6], the Microsoft CAPTCHA in [4], EZ-Gimpy in [7], and Gimpy-r in [8]. However, this extreme crowding also makes human-recognition a challenge. For example, is it obvious what the character string in the Google CAPTCHA is?



(A) Microsoft (broken) [4]

(B) Pessimal Print [9]

(C) ScatterType [11]

(D) Windows Live

(E) Google

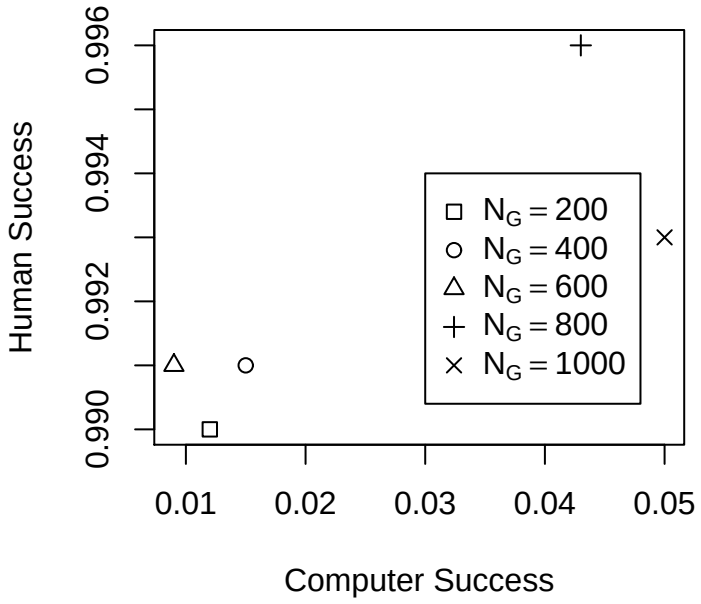
(F) Yahoo!

FIGURE 1. A few CAPTCHA Examples

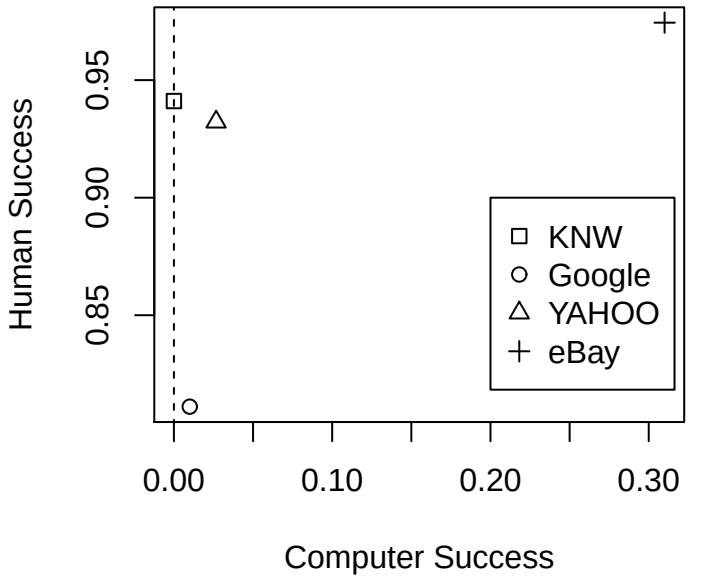
In contrast with the methods covered above, we view CAPTCHA generation as correlated random field simulation. Like Pessimal Print [9], our images provide partial, noisy information. We also leverage Gestalt perception to maintain a human-readable image, as in [10] and [11]. However, our use of randomness is far more fundamental and thereby far harder for computers to deal with than prior methods. We observe that the human readability of random CAPTCHA images is captured by the site, i.e. pixel, marginal probabilities and the site-to-nearby-site covariances; the actual joint distribution of the sites is not so important. Our method begins with a correlated random image that is evolved randomly a site at a time via Gibbs sampling until the random test word is human-readable. Our method of calculating each site's conditional probability mass function given the nearby sites that are either known or already simulated gives us exactly what is required for Gibbs sampling. The initial image can be a simple white background, any correlated random field, or, for strong segmentation resistance, a CAPTCHA generated by the ScatterType algorithm [11] with a different base word. Both the legibility and segmentation-resistance of our KNW-CAPTCHA depends on the number of iterations used in the Gibbs sampling step. The upshot is that we generate flexible, random CAPTCHAs automatically and efficiently and explain exactly how we do it.

In this work, we investigate two variants of the KNW-CAPTCHA: the KNW-CAPTCHA$_{\mathrm{E}}$, an easy variant generated without any background noise, and the KNW-CAPTCHA$_{\mathrm{H}}$, which is generated using character fragments as the background noise. The KNW-CAPTCHA$_{\mathrm{E}}$ is used to investigate how the generation parameters (especially the number of Gibbs iterations, N_G) affect the attack resistance of the resulting CAPTCHA. Figure 2a shows both the attack resistance and human readability of the KNW-CAPTCHA$_{\mathrm{E}}$ for various values of N_G, where computer success is the proportion of CAPTCHAs where either of the OCR programs Tesseract or ABBYY FineReader successfully recognized it, and human success is the proportion of CAPTCHAs where a human successfully recognized it. The KNW-CAPTCHA$_{\mathrm{H}}$ would be used in practice as the background noise provides additional security but the CAPTCHA remains highly readable to humans. Figure 2b compares the human readability and attack resistance of the KNW-CAPTCHA$_{\mathrm{H}}$ with several CAPTCHAs deployed by major corporations. As the correct answers for the comparison CAPTCHAs are unknown, we use optimistic solving accuracy (see Section 4.4) to determine human success; similarly, an OCR program is considered correct if it matches any of the human responses. These graphs clearly illustrate that both the KNW-CAPTCHA$_{\mathrm{E}}$ and KNW-CAPTCHA$_{\mathrm{H}}$ are highly readable and difficult to attack; even the KNW-CAPTCHA$_{\mathrm{E}}$ appears to have resistance to OCR comparable to or surpassing CAPTCHAs currently used by Google, YAHOO, and eBay. There were no computer successes against the KNW-CAPTCHA$_{\mathrm{H}}$, yet it obtained over 94% human success. (None of the other CAPTCHAs went unrecognized by OCR; only the eBay CAPTCHA bested the KNW-CAPTCHA$_{\mathrm{H}}$ in human success, but it also appears to be trivially broken.)

Our notation and random field algorithm are given in Section 2. Section 3 details our CAPTCHA generation, and Section 4 contains our results. We discuss alternative implementations of the KNW-CAPTCHA in Section 5.1. The mathematics behind the methodology in this paper will be published separately (see [13]).



(A) KNW-CAPTCHA$_{\mathrm{E}}$ with various N_G



(B) KNW-CAPTCHA$_{\mathrm{H}}$, Google, YAHOO, and eBay

FIGURE 2. Human and Computer Success on Various CAPTCHAs

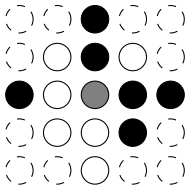


FIGURE 3. Simulation Example

2. Notation, Background, and Probability Computation

We begin by giving the required mathematical background and the equation for the conditional probability of a pixel given the nearby pixels based on correlations and marginal probabilities. Our goal is to randomly turn a pixel on/off given an estimated set of parameters (the marginal probabilities and site-site covariances) and the values of nearby pixels. The parameters capture the fundamental properties of the challenge word and, as pixels are re-simulated, the random image approaches the desired image. Figure 3 illustrates this setup, where the grey node represents the pixel being simulated and the nodes with solid outlines represent the nearby already-simulated pixels.

We consider a rectangular image of $M \times N$ pixels at the sites $S = \{(i,j) : 1 \leq i \leq M, 1 \leq j \leq N\}$, let $\rho(s,t) = \sqrt{(i_2 - i_1)^2 + (j_2 - j_1)^2}$ be the Euclidean distance between $s(i_1, j_1)$ and $t(i_2, j_2)$, and define the neighborhoods of $s = (i,j) \in S$ with radius $\ell \in \mathbb{R}$ as the ℓ-neighborhood

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

Definition 2.1. A point $s = (i,j) \in A$ is *ℓ-connected within* set $A \subset S$ if $\partial_\ell(s) \cap A$ is not empty. A is *ℓ-connected* if for every proper subset $B \subset A$, $\partial_\ell(B) \cap A$ is not empty.

We assume the desired *site marginals* $\{\pi_h\}$ satisfy $\pi_h(1) = 1 - \pi_h(-1) \in (0,1)$, $h \in S$ and $\{\beta_{h,t} : t \in \partial_l(h), h \in S\}$ are *site-site covariances*. Assume the numbers on the RHS of (2.1) are in $[0,1]$ (conditions for this to be true are given in [13]). Then, there is a probability measure Π on $\{-1,1\}^S$

such that for each $h \in S$

$$\Pi(X_h = c) = \pi_h(c),\ \forall\ c \in \{-1, 1\},\ \mathrm{cov}(X_h, X_t) = \beta_{h,t},\ \forall\ t \in \partial_l(h),$$

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

$$\Pi(\, X_h = x_h |\, X_{\partial_\ell(h)} = x_{\partial_\ell(h)}) = \pi_h(x_h) + \frac{\sum_{t \in \partial_\ell(h)} x_h \beta_{h,t} x_t}{\frac{1}{4} d^{|\partial_\ell(h)|+1} \Pi(X_{\partial_\ell(h)} = x_{\partial_\ell(h)})} \tag{2.1}$$

for each $x_h \in \{-1, 1\}$ and $x_{\partial_\ell(h)} \in \mathcal{X}_{\partial_\ell(h)}$, where $|\cdot|$ denotes the cardinality of a set.

Now, we explain how we use the marginals and covariances to determine the conditional probabilities (2.1) for simulating a KNW-CAPTCHA. Suppose we have determined the site pmf's $\{\pi_h\}_{h \in S}$ and the covariances $\{\beta_{h,t} : h, t \in S \text{ and } \rho(h,t) \le \ell\}$ of sites within distance ℓ of each other for random instances of the challenge word. (This is dealt with below.) Then, we start with a random field designed to bait computers into the wrong conclusions. Finally, we resample using $\{\pi_h\}$ and $\{\beta_{h,t}\}$ together with (2.1) until the challenge word is just human-readable yet there is such correlated noise that automated agents are unable to recognize the text.

We resample using Gibbs-like sampling, where we condition only on a large area around a site instead of all sites. The algorithm will randomly select a site $h \in S$ to resample using (2.1) to ensure we keep the desired pmf's and covariances. The joint probability $\Pi(X_{\partial_\ell(h)} = x_{\partial_\ell(h)})$ in the denominator on the RHS of (2.1) can be computed easily in real time by caching and re-using results. Let $\{t_1, \ldots, t_{|\partial_\ell(h)|}\}$ be the sites in $\partial_\ell(h)$ and $B_k = \{t_1, ..., t_k\}$ for $k = 1, \ldots, |\partial_\ell(h)|$ and $B_0 = \emptyset$. Then, we compute $\Pi(X_{\partial_\ell(h)} = x_{\partial_\ell(h)})$ using the multiplication rule

$$\Pi(X_{\partial_\ell(h)} = x_{\partial_\ell(h)}) = \prod_{i=1}^{|\partial_\ell(h)|} \Pi(X_{t_i} = x_{t_i} | X_{B_{i-1}} = x_{B_{i-1}}). \tag{2.2}$$

$\Pi(X_{t_i} = x_{t_i} | X_{B_{i-1}} = x_{B_{i-1}})$, $i = 1, \ldots, |\partial_\ell(h)|$ can be computed directly using (2.1).

Next, based on the conditional probabilities computed using (2.1), we use the following straightforward simulation algorithm to simulate h with the appropriate marginals and covariances.

(1) Compute $\Pi(\, X_h = c^u |\, X_{\partial_\ell(h)} = x_{\partial_\ell(h)})$ for $1 \le u \le d$, using (2.1).

(2) Generate a $[0,1]$-uniform random variable U. If

$$\sum_{u=1}^{w-1} \Pi(\, X_h = c^u |\, X_{\partial_\ell(h)} = x_{\partial_\ell(h)}) \leq U < \sum_{u=1}^{w} \Pi(\, X_h = c^u |\, X_{\partial_\ell(h)} = x_{\partial_\ell(h)})$$

for some $1 \leq w \leq d$, then we set $X_h = c^w$, i.e., the realization of X_h is c^w.

3. The KNW-CAPTCHA

We now present how to estimate the required parameters for a particular KNW-CAPTCHA and use those parameters to generate a novel random CAPTCHA in Sections 3.1 and 3.2, respectively.

3.1. **Parameter Estimation.** We begin by generating the data for the estimation process that consists of many independent instances of a particular word, where each instance varies randomly in many ways. The parameters learned from this data will represent the challenge word; by learning the parameters (site probabilities and site-to-nearby-site covariances) from this data, we can construct the conditional probabilities of the previous section and, thereby, do the Gibbs resampling portion of our CAPTCHA creation.

The algorithm for generating the data consists of selecting a word to serve as the KNW-CAPTCHA's correct response and then generating a number of random images representing this word by varying fonts and placement of characters in the word. The word images will be constructed by joining individual character images. Herein, we select a random word uniformly over a fixed dictionary of common English words with a length of at least three characters.

For each letter in the English alphabet and for each of 18 fonts, we generate character images denoted $\{f_{1,1}, \ldots, f_{1,26}, \ldots, f_{18,1}, \ldots, f_{18,26}\}$, i.e., $f_{i,j}$ is the character image of the j^{th} letter in the i^{th} font. To ensure that forming a word image by joining random character images results in consistent horizontal placement of individual character images, we work with character images that, for a given letter, all have the same width. To accomplish this, we generate trimmed or scaled character images for each letter as appropriate. Let N_j^f denote the maximum width of the

bounding boxes over the character images $\{f_{1,j}, \ldots, f_{18,j}\}$, where a bounding box is the smallest rectangle that encloses the character. For $i = 1, \ldots, 18$, $j = 1, \ldots, 26$,

- if j is one of the letters {i,j,l,r,t}, generate a new character image $f'_{i,j}$ by centering and trimming $f_{i,j}$ so that its width is N_j^f by removing columns outside the bounding box;
- otherwise, generate a new character image $f'_{i,j}$ by scaling $f_{i,j}$ so that $f'_{i,j}$'s bounding box has a width of N_j^f and removing all columns outside the bounding box.

The letters {i,j,l,r,t} were chosen for trimming instead of scaling since scaling some of their images results in very tall bounding boxes due to their highly variable character widths.

We then generate K images of the chosen character string with pixel state space $\{-1, 1\} = \{\text{white}, \text{black}\}$, and n_c is the number of characters in the character string using the following algorithm.

(1) The horizontal distance between each adjacent character's bounding box is chosen using a random number selected uniformly over $\{1, 2, 3\}$. This is fixed for all K images.

(2) The vertical displacements of characters are determined using the values $\{v_0, v_1, v_2, \ldots\}$ of a reflecting random walk, moving upward or downward one with probability $\frac{1}{2}$; upon hitting the boundary $\{-25, 25\}$, it reflects. The random walk is initialized randomly over $\{-10, ..., 10\}$. The i^{th} character image, where $i \in \{1, 2, \ldots, n_c\}$, will be placed vertically by centering it according to the vertical center of its bounding box, and then shifting it up or down according to the value $v_{(i-1)\times 6}$ of the random walk. This produces $(n = 6, p = \frac{1}{2})$-binomial shifts before reflection. This is also fixed for all K images.

(3) For $1, ..., K$

   (a) For each letter in the string, a random character image is chosen uniformly over $\{f'_{1,i}, \ldots, f'_{18,i}\}$, where i corresponds to the given letter.

   (b) The string image is generated by positioning each character image according to the above horizontal distance and vertical displacement parameters.

The data generation algorithm is motivated by the following: the horizontal distance is varied randomly to introduce crowding between some adjacent characters and make the horizontal positions of characters unpredictable, both of which make segmentation more difficult; the vertical displacement is varied to ensure the vertical location of the word is unpredictable, but a random walk is used to introduce dependence between adjacent characters and aid the reader in following the flow of the word; and the font is chosen randomly for each character to ensure the estimated parameters represent an "average" character, rather than a particular font, so that feature detection or pattern recognition becomes difficult. Still, it must be remembered that the main sources of defense against automated attacks come from the original correlated random field and the pixel by pixel randomness in simulating the word so we do not rely just on character crowding as other methods do, but rather use it as one more layer of protection.

Returning to estimation, we let $s^{(i)}, t^{(i)}$ denote the value of pixels s and t in the i^{th} image and use the unbiased covariance estimator

$$\beta_{s,t} = \frac{1}{K-1}\sum_{i=1}^{K}(x_{s^{(i)}} - \bar{x}_s)(x_{t^{(i)}} - \bar{x}_t) \text{ for } 0 < \rho(s,t) \leq \ell,$$

where $\bar{x}_s = \frac{1}{K}\sum_{i=1}^{K} x_{s^{(i)}}$ is the empirical mean. We estimate the marginal probabilities as

$$\pi_s(x_s) = \frac{1}{K}\sum_{i=1}^{K} 1_{x_{s^{(i)}}=x_s}, \text{ where } 1_{x_{s^{(i)}}=x_s} = \begin{cases} 1 & \text{if } x_{s^{(i)}} = x_s \\ 0 & \text{otherwise.} \end{cases}$$

3.2. **KNW-CAPTCHA Generation.** We now present the KNW-CAPTCHA generation details, which consists of generating background noise and then simulating the character string, using modified Gibbs sampling with the parameters obtained in Section 3.1, *on top of* the background noise.

Introducing background noise is a common technique when generating CAPTCHAs since it introduces *red herring* character shapes that must be removed or ignored by a computer program. Our view is that the best red herrings are actual character pieces. Background noise also makes

segmentation more difficult since, for example, vertical projection will not detect gaps between adjacent characters bridged by appropriate background noise, and connected components will view two adjacent characters as one if they are connected by background noise (see e.g. Figures 1a, 6c).

We generate background noise via the ScatterType algorithm in [11]. While the original intent of the ScatterType algorithm was to produce CAPTCHAs that were human-readable but difficult to crack, our goal is the reverse: produce ScatterType CAPTCHAs that are clearly unreadable to humans yet "readable" to computers, i.e., the character shapes produced will serve as effective red herrings. By being obviously human-unreadable, the background noise will be visually distinct from the actual character string, serving as a form of stenography. Still, the character pieces are often erroneously detected by computer programs as part of the actual character string. The unreadable background noise is generated using the following algorithm.

(1) Choose a five-letter character string uniformly, with replacement, over the English alphabet.

(2) Apply the ScatterType algorithm using a fixed font and the following parameters:

Cutting Fraction	0.50	Expansion Fraction	0.60
Horizontal Scatter Mean	0.00	Vertical Scatter Mean	0.00
Scatter Standard Error	0.05	Character Separation	0.20

For our purposes, it is sufficient to say that this algorithm cuts each character into large chunks (roughly quadrants), scatters each chunk, and separates each adjacent character by roughly the width of a character. The reader is referred to [11] for a description of the ScatterType algorithm.

Finally, we are ready to generate the KNW-CAPTCHA. We apply Gibbs-like sampling, where we consider background noise as the initial state and use (2.1) to calculate the conditional probabilities of sites in order to re-simulate them. The challenge is to choose and re-simulate the correct sites so that the KNW-CAPTCHA is human-readable but resistant to crack attempts. We consider such a KNW-CAPTCHA to be a "minimally-readable CAPTCHA".

The KNW-CAPTCHA is generated using the following algorithm. See Figure 4 for examples.

(1) Select a character string, generate a data sample of size $K = 30 \times n_c$, where n_c is the number of characters in the string, and estimate the parameters as described in 3.1.

(2) Set R, the sites to re-simulate, as follows:

   (a) $S_p = \{s \in \{1, \ldots, M\} \times \{1, \ldots, N\} : \pi_{x_s}(1) > 0\}$, i.e., the sites that have a non-zero probability of being black.

   (b) $S_p^4 = \{s \in \{1, \ldots, M\} \times \{1, \ldots, N\} : s \in \partial_4(p) \text{ for some } p \in S_p\}$, i.e., the sites that are within a distance of $\ell = 4$ from a site that has a non-zero probability of being black.

   (c) To choose R, select $N_G \times n_c$ sites, where $N_G \in \mathbb{N}$ is constant for all characters, randomly and without replacement from S such that the probability of selecting a site from S_p^4 is ten times greater than selecting a site from $S \setminus S_p^4$. [1]

(3) Generate the random ScatterType-based noise as described above. Select 400 sites in the same manner as choosing R, and re-simulate each of those sites using only the marginal probabilities (i.e., assuming independence). Take this to be the background noise.

(4) Apply the modified Gibbs sampling:

   (a) Take the initial state to be the background noise.

   (b) Re-sample each site in R according to Section 2.[2]

[1] Sites within and near the defining "shape" of a letter are likely to be re-simulated, while others are not, ensuring the background noise is preserved while the character string is sufficiently human-readable. We consider this a Gibbs-like sampler since the goal is not to reach the joint distribution of the KNW-CAPTCHA but to effectively blend the encoded word with the background noise.

[2] Depending on the parameters estimated and the background noise used, we may encounter conditional probabilities outside the bounds of $[0, 1]$. In this paper, we are more concerned with the practical outcome of the algorithm over perfect mathematical sensibility; for this reason, if a probability is encountered outside these bounds, we instead use the marginal probability as a fallback. Please see [13] for a detailed exploration of the constraints on the parameters.

(A) A KNW-CAPTCHA$_{\mathrm{E}}$. (B) A KNW-CAPTCHA$_{\mathrm{H}}$, with outline.

FIGURE 4. KNW-CAPTCHA examples

This process generates a matrix of black and white pixels saved as a PNM file; in practice, we must use an image format supported by modern web browsers as the CAPTCHAs will typically be deployed on websites. We use ImageMagick [14] to convert the PNM file to a 72 DPI JPEG file, which is used in the following OCR and human readability experiments.

4. RESULTS

In the following, we describe how we measure the properties of the KNW-CAPTCHA and provide results. In Section 4.1, we attack a weak variant of the KNW-CAPTCHA with computer programs to establish a lower-bound to the KNW-CAPTCHAs' attack resistance; in Sections 4.2 and 4.4, we measure the human readability of the hardened KNW-CAPTCHA; finally, in Section 4.3, we measure the attack resistance together with the human readability of the hardened KNW-CAPTCHAs.

We use KNW-CAPTCHA$_{\mathrm{E}}$ to refer to the easy KNW-CAPTCHA variant (Figure 4a). This variant is generated with no background noise and no vertical displacement of individual characters, and is designed to be as easy as possible to attack while maintaining the fundamental properties of the KNW-CAPTCHA. The hardened variant, KNW-CAPTCHA$_{\mathrm{H}}$, generated with both background noise and random vertical displacement, is deployed in practice. See Figure 4b.

4.1. **KNW-CAPTCHA$_{\mathbf{E}}$ Experiments.** Recall that in a (α, β, η)-CAPTCHA, we want η to be low. We now show our η is low by establishing that modern OCR programs are unable to recognize the encoded words. In particular, we use KNW-CAPTCHA$_{\mathrm{E}}$ and design each experiment to give

the attacks the best chance of success. The resistance to attacks in these easy cases is a lower bound for the hardened KNW-CAPTCHA used in practice. However, our results below show even the KNW-CAPTCHA$_{\mathrm{E}}$ is basically unbreakable with contemporary OCR programs.

In order to understand the effect of N_G, the proportion of sites to resimulate in our modified Gibbs sampling, we perform the following experiments over a number of values of N_G and expect N_G to be related to how readable by both computer programs and humans the resulting image is.

We use two OCR programs: Tesseract and ABBYY FineReader. Tesseract is available at `http://code.google.com/p/tesseract-ocr/` (retrieved 2010-09-14). To our knowledge, Tesseract is the best available open-source OCR. An overview of the implementation of Tesseract is given in Smith [15]. ABBYY FineReader is a propriertary OCR program used in, for example, [9].

We proceed in the spirit of giving the OCR programs a "fighting chance" by using KNW-CAPTCHA$_{\mathrm{E}}$. In essence, we make the KNW-CAPTCHA as easy as possible to recognize (while maintaining its fundamental construction). This tactic will provide the most evidence that η is low, i.e., that the KNW-CAPTCHA is difficult to crack. Word accuracy is calculated based on the number of words recognized, and all word comparisons are done ignoring case.

For a particular word, the experiment is as follows.

(1) Generate a KNW-CAPTCHA$_{\mathrm{E}}$ for the word word$_K$ with no background noise and no vertical displacement.
(2) Run the OCR program to obtain word$_O$.
(3) Compare word$_K$ and word$_O$.

We vary N_G and obtain the results over n_T trials under each given value of N_G. Under a given N_G, we model each attempt to recognize the word as i.i.d. (p_w)-Bernoulli random variables, where p_w is the probability of recognizing the word. We use the maximum likelihood estimator $\hat{p}_w$ and provide the 95% confidence interval. We perform the experiment for n_T words, selected without replacement randomly from our dictionary. In order to validate the human readability of

the KNW-CAPTCHA$_{\mathrm{E}}$, we also collect human results via Amazon Mechanical Turk (AMT) [16] (see Section 4.2.2 for details). Results are summarized in Table 1, and an example of a KNW-CAPTCHA$_{\mathrm{E}}$ is provided in Figure 4a. Based on these results, it appears that both OCR programs have great difficulty recognizing the KNW-CAPTCHA$_{\mathrm{E}}$s. In fact, the computer performance on the unhardened KNW-CAPTCHA$_{\mathrm{E}}$ with $N_G = 200$ is similar to the results on the Google CAPTCHA (Table 3), which was the most difficult for OCR to recognize of Google, YAHOO, and eBay. In addition, human performance on the KNW-CAPTCHA$_{\mathrm{E}}$ is very high; taken together, this experiment strongly indicates that η is low while β is high, as desired. As expected, both OCR and human performance generally increase as N_G increases, which indicates that N_G will serve an important role in balancing readability and security.

N_G	n_T	ABBYY	Tesseract	Human
200	1000	0.012 ± 0.007	0.000 ± 0.000	0.990 ± 0.006
400	1000	0.015 ± 0.008	0.000 ± 0.000	0.991 ± 0.006
600	1000	0.007 ± 0.005	0.002 ± 0.003	0.991 ± 0.006
800	1000	0.032 ± 0.011	0.015 ± 0.008	0.996 ± 0.004
1000	1000	0.034 ± 0.011	0.020 ± 0.009	0.993 ± 0.005

TABLE 1. 95% Confidence Interval of Computer and Human $\hat{p}_w$

4.2. **KNW-CAPTCHA$_{\mathbf{H}}$ Experiments.** In the (α, β, η)-CAPTCHA context, our β is high. The KNW-CAPTCHA should be applied to literate English-reading adults with normal eyesight. (In practice, alternative CAPTCHAs, such as an audio CAPTCHA, should be provided others.) Our task is to estimate β and the time to complete the challenge empirically.

The following experiments use a set of 300 KNW-CAPTCHA$_{\mathrm{H}}$ images generated with $N_G = 800$ based on the results of the previous sections along with visual inspection in order to balance attack resistance with readability.

4.2.1. *Online Readability Study.* To collect these results, we set up the website `http://www.knwcaptcha.org`. Volunteers participating in this online study were anonymous. No incentive was provided. The procedure was as follows.

(1) The visitor is presented with information on how the experiment is conducted and how the data will be used. If the user does not accept, the experiment is terminated.
(2) In order to familiarize the visitor with the process, he or she is presented with an example of a KNW-CAPTCHA$_{\mathrm{H}}$ along with the correct response. The example shows a KNW-CAPTCHA$_{\mathrm{H}}$ with the encoded word outlined, and is designed to show the visitor how to recognize the encoded word in noise. See Figure 4b.
(3) The visitor is shown a set of 25 KNW-CAPTCHA$_{\mathrm{H}}$s. A visitor is never shown the same word more than once. Beside each KNW-CAPTCHA$_{\mathrm{H}}$, the visitor enters a response, and submits the entire data set upon completion.

A human's response to a KNW-CAPTCHA$_{\mathrm{H}}$ is marked as correct if it matches the encoded word, ignoring case, and incorrect otherwise. In the analysis, we model the trials as i.i.d. (β)-Bernoulli random variables. The experiment yields n_T responses from humans $y_1, \ldots, y_{n_T}$, where $y_i = 1_{i^{\text{th}} \text{ response was correct}}$. As before, we use the maximum likelihood estimator $\hat{\beta}$ to estimate β. The time to solve each challenge is calculated using the time elapsed from when the user is first presented with the CAPTCHAs to the submission of the responses. The results are summarized in Table 2. Humans succeeded at solving a high proportion of KNW-CAPTCHA$_{\mathrm{H}}$s quickly, helping to establish that β is high and our CAPTCHAs are not onerous.

4.2.2. *Amazon Mechanical Turk.* In addition to collecting responses from volunteers at `knwcaptcha.org`, we used Amazon Mechanical Turk (AMT) [16]. AMT is an online service which allows requesters to submit tasks which will be completed by a pool of workers. The use of AMT for collecting human feedback in research has been established in several works. In Kittur *et al.* [17],

the authors found that high quality responses are achievable when using an appropriate experimental design; for example, in order to be resistant to workers gaming the task, it is important that the task be as much effort to complete incorrectly as correctly. In Sorokin *et al.* [18], the authors successfully use AMT for the purpose of image annotation. In Bursztein *et al.* [19], a number of popular CAPTCHA schemes are evaluated in terms of human readability based on the amount of agreement between three workers on a CAPTCHA image containing an unknown word. (For further discussion of [19], please see Section 4.4.) Our task of evaluating the responses to a known CAPTCHA is relatively straightforward and an appropriate fit for AMT.

Our AMT task design is similar to that of `knwcaptcha.org`. Each task submitted to AMT consisted of a KNW-CAPTCHA$_{\mathrm{H}}$ image and a response field. A batch of tasks is preceded by brief instructions and an example, as on `knwcaptcha.org`. No qualification pre-tests are administered, nor are workers penalized (via, for example, lack of payment) for wrong answers. AMT provides more diverse, international respondents than could be obtained by recruiting local volunteers as in Section 4.2.1. While no demographic information is collected, a comprehensive survey of the AMT worker population conducted by Ross *et al.* [20] found a large population of international, young, educated workers. Furthermore, [19] examines the effect of demographics on CAPTCHA solving ability; of particular interest to us is that native English speakers are able to solve English or pseudo-English CAPTCHAs far faster, which indicates that the KNW-CAPTCHA$_{\mathrm{H}}$ is biased against non-native English speakers.

The responses collected are summarized in Table 2. As before, we model the trials as i.i.d. (β)-Bernoulli random variables, and solving time is calculated as in Section 4.2.1. While there is a drop in accuracy when compared with the results in Section 4.2.1, this is likely explained by the different demographics of the respondents, particularly native language, as well as the lack of incentives for correct responses. Unsurprisingly, the AMT workers, who are incentivized to complete tasks quickly, solve the CAPTCHAs faster than their volunteer counterparts.

4.3. **OCR Attacks on Hardened KNW-CAPTCHAs.** Next, we confirm that the KNW-CAPTCHA$_{\mathrm{H}}$ is an effective separator of humans and computer programs by providing an "apples to apples" comparison of OCR performance against human performance. To obtain these results, we ran Tesseract and ABBYY FineReader on the KNW-CAPTCHA$_{\mathrm{H}}$s for which we have human responses and determined accuracy as before. The results are in Table 2.

As is clear from Table 2, neither OCR program is able to recognize any of the KNW-CAPTCHA$_{\mathrm{H}}$s, while humans perform remarkably well on them. In fact, the OCR programs seldom recognized any of the characters present in the word. These results, taken together with the results in Section 4.1, provide strong evidence that the KNW-CAPTCHA$_{\mathrm{H}}$ defends automated attacks well while also remaining quickly and easily solvable by humans. In particular, we see that the human time to solve the KNW-CAPTCHA$_{\mathrm{H}}$ is low so our CAPTCHAs are not onerous.

	N_G	n_T	95% Confidence Interval	Time to Solve
knwcaptcha.org	800	300	0.960 ± 0.022	6.41s
AMT	800	3319	0.910 ± 0.010	4.98s
Tesseract	800	300	0.000 ± 0.000	N/A
ABBYY	800	300	0.000 ± 0.000	N/A

TABLE 2. KNW-CAPTCHA$_{\mathrm{H}}$ Human and Computer Performance

4.4. **Comparison.** We now compare the KNW-CAPTCHA$_{\mathrm{H}}$ to other popular CAPTCHAs by replicating the procedure used in the excellent CAPTCHA readability study in Bursztein *et al.* [19].

In [19], responses are collected from three distinct AMT workers for each CAPTCHA image. Since the correct answer for each CAPTCHA is unknown, they instead compute "optimistic solving accuracy": for a particular CAPTCHA image, if all three responses agree then all three responses assumed to be correct; if two agree, then two responses are assumed to be correct; otherwise, one response is assumed to be correct. In addition, we collect responses from ABBYY and Tesseract

FIGURE 5. Authorize CAPTCHA Example [19]

as before; in this case, an OCR program is considered correct if it matches any of the three human responses. See Table 3 for optimistic computer and human solving accuracy, where n_C is the number of CAPTCHAs used, and n_H is the number of human responses collected.

	n_C	ABBYY	Tesseract	n_H	$\hat{\beta}$
KNW-CAPTCHA$_{\mathrm{H}}$	300	0.00 ± 0.00	0.00 ± 0.00	900	0.94 ± 0.02
Google	300	0.00 ± 0.00	0.01 ± 0.01	900	0.81 ± 0.03
YAHOO	300	0.01 ± 0.01	0.02 ± 0.01	900	0.93 ± 0.02
eBay	300	0.05 ± 0.02	0.29 ± 0.05	900	0.97 ± 0.01

TABLE 3. 95% Confidence Interval of Optimistic Computer and Human Performance

The KNW-CAPTCHA$_{\mathrm{H}}$ compares favourably with the Google, YAHOO, and eBay CAPTCHAs: it is the only CAPTCHA that was unrecognized by either OCR program, and only the eBay CAPTCHA was more human readable (though the eBay CAPTCHA also appears trivially broken). The Google CAPTCHA was seldomly recognized by OCR, but at significant human readability cost.

The optimistic human solving accuracy on the KNW-CAPTCHA$_{\mathrm{H}}$ also compares favourably to many popular CAPTCHA schemes used in [19], including reCAPTCHA (0.75), Google (0.86), and Yahoo (0.88). A few CAPTCHA schemes achieved higher accuracy, like the Authorize CAPTCHA (0.98). However, this study did not assess attack resistance; the Authorize CAPTCHA example in Figure 5 is straightforward to segment, for example. The reader is referred to [19] for details.

5. Security Discussion

Removal of noise is typically the first step of a CAPTCHA attack, and is often straightforward due to the noise's distinctness from the character images, as in the PayPal CAPTCHA [6] and Gimpy-r [8]. Dictionary knowledge facilitates specific pattern discovery in many text-based CAPTCHAs, as in the bigram-based attack against Gimpy [7]. Font knowledge can be used to determine the most likely character for a particular character image, as in [8]. Finally, segmentation of word images was often a critical step in order to individually attack and recognize characters, as in [6] and [8]. We now examine how the KNW-CAPTCHA$_{\mathrm{H}}$ resists like-minded attacks.

The KNW-CAPTCHA$_{\mathrm{H}}$ uses ScatterType background noise comprised of character fragments difficult to differentiate from the encoded CAPTCHA characters, in contrast to the Microsoft CAPTCHA [4], which relied on random arcs. However, the background still retains distinct qualities; in particular, it does not appear as "noisy" as the encoded characters; which works in our favor for untargeted attacks. If an attack were to target this feature, the amount of degradation done to the initial state could be varied (see Section 5.1 for more details).

The KNW-CAPTCHA$_{\mathrm{H}}$ also uses an English dictionary with a fixed number of words for human readability. However, this does enable attackers to use dictionary knowledge to improve attack effectiveness. If this proves a weakness, there are three straightforward alternatives: increase the dictionary size (reCAPTCHA uses 100,000 words [21]); use pseudo-words as in [11]; or use random character strings.

It will prove very difficult for attackers to leverage font knowledge against the KNW-CAPTCHA$_{\mathrm{H}}$. The KNW-CAPTCHA algorithm learns the parameters to simulate a character image from many fonts; no one particular font is used, and the set of fonts used can be changed easily. Instead, each character image in the KNW-CAPTCHA is randomly simulated, leading to a partial, noisy image such that no two realizations are the same nor do they match any of the fonts. This contrasts

particular font distortions, as in Gimpy-r, which yielded to distortion estimation techniques [8], or to particular font noise obscurations, as in the Pessimal Print CAPTCHA [9].

Segmentation resistance is critical for CAPTCHA design since a trained computer program can outperform humans at recognizing distorted, cluttered single character images [22]. Segmentation continues to be error-prone for OCR, but several CAPTCHAs have been broken via segmentation attacks (see Section 1). As several modern CAPTCHAs, the KNW-CAPTCHA$_{\mathrm{H}}$ uses character crowding (in addition to random images). The KNW-CAPTCHA$_{\mathrm{H}}$ crowds using adjustable random spacing, as in [11], which typically leads to some adjacent character images overlapping.

To illustrate, we implement and execute a vertical projection segmentation attack on the easier KNW-CAPTCHA$_{\mathrm{E}}$. The vertical projection attack, at its simplest, calculates the total number of "on", or black, pixels in each column in an image. The image is then segmented at columns where there are few or no black pixels. This method is very fast since only one pass of the image is required, making it a useful tool for attempting to crack large numbers of CAPTCHAs. We implement a more sophisticated variant of the vertical projection attack to identify segmentation candidates as described by Tsujimoto *et al.* [23], which is designed to segment touching characters as in KNW-CAPTCHAs. (Casey *et al.* [3] provides an excellent overview of segmentation methods.)

Tsujimoto *et al.* [23] define their algorithm for finding segmentation candidates as follows.

(1) For each adjacent pair of columns, perform an AND operation and determine the number of black pixels in the resulting column (i.e., the number of pixels that were black in both columns); this number is called the *break cost*.
(2) Smooth the break costs obtained in the previous step.
(3) Identify break candidates as local minima in the smoothed break costs.

Herein, we smooth using a moving average.

We use the following experimental procedure for measuring the segmentation performance.

(1) Generate an easy KNW-CAPTCHA for a character string consisting of two random (i.e., chosen uniformly over the English alphabet), lower case letters, with no background noise (i.e., the modified Gibbs sampling is initialized with a white image).

(2) Find the global smoothed break point minimum within the boundaries (i.e., in the horizontal region between the first and last black pixels) of the generated KNW-CAPTCHA using the above sophisticated vertical projection segmentation attack. In the case of a tie, select the point randomly amongst the global minima. Use this as the point of segmentation.

(3) If the bounding boxes of the two characters overlap, and the segmentation point is within two pixels of the middle of the overlapping region, then consider the segmentation correct. If the bounding boxes do not overlap, then consider the segmentation point correct if it lies anywhere in the region between the bounding boxes.

The determination of whether the segmentation point is correct is slightly modified from the work done by Hoffman *et al.* [24], which sought to isolate measures of segmentation performance from recognition engines. Since we are attempting to evaluate only segmentation resistance at this point, rather than recognition resistance, ours was an appropriate technique to adopt.

We estimate p_s, the probability of successful segmentation, using the maximum likelihood estimator $\hat{p}_s$. The results are summarized in Table 4. The segmentation performance is low despite targeting character crowding and presenting simplified two character images without scatter noise. Furthermore, the segmentation performance does not vary significantly with N_G indicating the vertical projection algorithm has difficulty with the basic construction of the KNW-CAPTCHA.

Collectively, the security mechanisms in the KNW-CAPTCHA$_{\mathrm{H}}$ will prove difficult to circumvent. However, should it be successfully attacked, other variants may take its place.

5.1. **Variants.** The mechanism described in Section 3 is more general than the particular example we study in this work: it can easily be extended to counter new attacks. For example, if an attacker is able to remove the background noise, one can use striped correlated noise (see Figure 6c); if a

N_G	n_T	95% Confidence Interval of $\hat{p}_s$
600	1000	0.094 ± 0.018
800	1000	0.092 ± 0.018
1000	1000	0.088 ± 0.018

TABLE 4. Segmentation Performance

dictionary-based attack succeeds, one can use pseudo-words; if a segmentation attack succeeds, one can increase character crowding or decrease N_G (see Figure 6a). In fact, one could deploy several variants simultaneously, effectively reducing the reward for successfully attacking any particular variant. We now provide a high-level view of potentially useful variants; furthermore, we will discuss how a variant can be selected by a user of the KNW-CAPTCHA.

The algorithm given in Section 3 consists of the following steps.

(1) Sample generation: many random instances of a character string image are generated.
(2) Parameter estimation: simulation parameters are estimated from image samples.
(3) Initial state: an initial state for the KNW-CAPTCHA is generated.
(4) Simulation: re-simulate random pixels of the KNW-CAPTCHA until word appears.

This is a template method pattern [25], meaning that we have given a high-level description of the algorithm while allowing variants to define the details of how each step is accomplished.

Within, we studied two variants, the KNW-CAPTCHA$_{\mathrm{E}}$ and the KNW-CAPTCHA$_{\mathrm{H}}$. The KNW-CAPTCHA$_{\mathrm{E}}$ was deliberately designed to be vulnerable to attack by eschewing random vertical displacement in the sample generation step, and background noise. In contrast, the KNW-CAPTCHA$_{\mathrm{H}}$ does use random vertical displacement and the initial state is generated using a ScatterType CAPTCHA. These two relatively simple differences produce significantly different CAPTCHAs, yet the overall algorithm remains the same.

Now we introduce several variants to illustrate the flexibility of the KNW-CAPTCHA algorithm.

Low N_G: We use a less cluttered initial state and alter the simulation step by using a lower N_G, $N_G = 100$ say. The intent is quite different from the KNW-CAPTCHA$_{\mathrm{H}}$: instead of relying on background noise to defend against attack, we are relying on using only partially-formed character shapes to ensure segmentation is difficult. See Figure 6a.

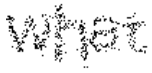
Clustered Correlated Noise: Instead of the ScatterType character fragments, we generate the initial state using the simulation algorithm detailed in Section 2, with pair-wise covariances set to the Euclidean distance from the pixel being simulated with $\ell = 2$. The effect is an initial state with clustered random shapes. See Figure 6b.

Striped Correlated Noise: We generate the initial state using one pass of the simulation algorithm with $\ell = 2$. Let the pixel p being simulated have the coordinates (x, y), and let pixel p_i have the coordinates (x_i, y_i). For each pixel p_i in the neighbourhood of p, set the pair-wise covariance to 0 if $x < x_i$ and $y < y_i$, or if $x > x_i$ and $y > y_i$; otherwise set the pair-wise covariance according to the Euclidean distance between p and p_i. This generates striped correlated noise. See Figure 6c.

Simulated Characters: generate the initial state using the KNW-CAPTCHA$_{\mathrm{E}}$ algorithm with a lower N_G and a random character string. This produces a background noise that is distinct to humans but difficult to eliminate automatically due to the similarity in form to the CAPTCHA word. A random character string is used instead of a word to prevent confusion between the background noise and the CAPTCHA word. See Figure 6d.

5.1.1. *Variant Selection.* While it is clear that it is easy to generate varied CAPTCHAs using the methods laid out in this work by modifying parameters or the steps in the CAPTCHA algorithm, we have not yet discussed how these variants can be compared and selected. In the following, we will present an idea of how to accomplish this automatically.

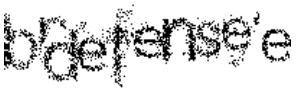
Any comparison should naturally take into account both attack resistance and human readability. However, different users of CAPTCHAs will place different on each quality and a CAPTCHA should



(A) Low N_G

(B) Clustered Correlated Noise

(C) Striped Correlated Noise



(D) Simulated Characters

FIGURE 6. Variants of the KNW-CAPTCHA

be able to balance the two qualities. More precisely, let $f(\theta) = w \times a(\theta) - (1 - w) \times h(\theta)$, where θ is the set of parameters determining the variant of KNW-CAPTCHA generated, $a(\theta)$ is the probability of an attack succeeding on an instance of the variant, $h(\theta)$ is the probability of a human being able to read an instance of the variant, and $w \in [0, 1]$ is a weight balancing the two qualities. Then $f(\cdot)$ is a cost function, and the goal of a CAPTCHA should be to minimize it. The role of w is to allow the user of a CAPTCHA to balance attack resistance and readability. $a(\theta)$ and $h(\theta)$ are unknowable and can only be estimated. One method of estimating $a(\theta)$ would be to attempt to attack the many instances of the CAPTCHA with several OCR engines and consider it a success if any of them succeed (similar to how the reCAPTCHA project determines if a word image should be used as a CAPTCHA challenge [21]), i.e., $\hat{a}(\theta) = \frac{1}{n}\sum_{i=1}^{n} y_i$ where $y_i = 1$ if any of the OCR engines recognize the word, and $y_i = 0$ otherwise. $h(\theta)$ could be estimated in a similar fashion, using human readability experiments. Then, selecting the appropriate CAPTCHA variant becomes a matter of minimizing the cost function over the evaluated variants, i.e., $\theta* = \min_{\theta \in \Theta} \hat{f}(\theta)$.

6. Conclusion and Future Work

We developed and implemented a new method of generating random CAPTCHAs, called KNW-CAPTCHAs, using random field simulation that outperforms popular CAPTCHAs in use today.

First, we estimated the marginal probabilities of sites and site-to-site covariances of the KNW-CAPTCHA based on randomly generated samples; second, we used an efficient algorithm to simulate a new KNW-CAPTCHA based on these parameters in a Gibbs-like manner.

Furthermore, we established that the KNW-CAPTCHA is an effective separator of computer programs and humans. We provided evidence that the KNW-CAPTCHA is difficult for computer programs to crack through an analysis of its resistance to segmentation attacks and OCR attacks. We also established that the KNW-CAPTCHA is very readable to humans.

Finally, we discussed targeted attacks against the KNW-CAPTCHA and several implementation variants, as well as how to select a variant automatically based on empirical results.

There are several methods of further hardening the KNW-CAPTCHA, which we explored in part in Section 5.1. Characteristics of the generated CAPTCHA can be varied within the CAPTCHA. For example: the number of sites to re-simulate per character could increase with each character in a KNW-CAPTCHA; the colors used for the background noise and the CAPTCHA could change from left to right; or the amount of noise could be increased or decreased vertically. The intent of these changes would be to effectively add another dimension to the problem, further confusing an attacker without compromising readability.

Finally, in this work we used only black and white when generating samples; however, this method can be readily extended to generate CAPTCHAs with one or many grey levels. As above, the intent would be to increase the dimensionality of the problem for the attacker without decreasing readability; for example, grey levels could be used to make distinguishing between the background and the letters themselves more difficult, or to make the shapes of the characters themselves less obvious. The main challenge would be to adjust the random sample generation and parameter estimation methods used in this paper in such a way that maintains or improves readability.

Acknowledgment

The authors thank the anonymous volunteers who partook in the readability study at `http://www.knwcaptcha.org`.

References

[1] M. Blum, L. Von Ahn, J. Langford, and N. Hopper, "The CAPTCHA Project, Completely Automatic Public Turing Test to tell Computers and Humans Apart," *Dept. of Computer Science, Carnegie-Mellon Univ., http://www. captcha. net*.

[2] L. von Ahn, M. Blum, N. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*. Springer-Verlag, 2003, vol. 2656, pp. 294–311.

[3] R. Casey and E. Lecolinet, "A survey of methods and strategies in character segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 18, no. 7, pp. 690–706, 1996.

[4] J. Yan and A. El Ahmad, "A Low-cost Attack on a Microsoft CAPTCHA," in *Proc. 15th ACM Conf. Computer and Communications Security (CCS 08)*. ACM Press, 2008, pp. 543–554.

[5] R. Datta, J. Li, and J. Wang, "Imagination: A robust image-based captcha generation system," in *Proceedings of the 13th annual ACM international conference on Multimedia*. ACM, 2005, pp. 331–334.

[6] K. Kluever, "Breaking the PayPal HIP: A Comparison of classifiers," Rochester Institute of Technology Document and Pattern Recognition Lab, https://ritdml.rit.edu/handle/1850/7813, 2008.

[7] G. Mori and J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA," *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 134–141, 2003.

[8] G. Moy, N. Jones, C. Harkless, and R. Potter, "Distortion estimation techniques in solving visual captchas," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 23–28, 2004.

[9] A. Coates, H. Baird, and R. Faternan, "Pessimal print: a reverse turing test," in *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*. IEEE, pp. 1154–1158, 2001.

[10] M. Chew and H. Baird, "Baffletext: a human interactive proof," in *Document Recognition and Retrieval X, Proceedings of SPIE*, vol. 5010, pp. 305–316, 2003.

[11] H. Baird and T. Riopka, "ScatterType: A reading CAPTCHA resistant to segmentation attack," in *Document Recognition and Retrieval XII, Proceedings of SPIE*, vol. 5676, pp. 197–201, 2005.

[12] J. Yan and A.S. El Ahmad, "Breaking Visual CAPTCHAs with Naive Pattern Recognition Algorithms," in *Proceedings of the 23rd Annual Computer Security Applications Conference*. IEEE, pp. 279–291, Dec. 2007.

[13] M. Kouritzin, F. Newton, and B. Wu, "Properties of Quick Simulation Random Fields," *In Preparation*.

[14] ImageMagick Studio LLC. ImageMagick. http://www.imagemagick.org/.

[15] R. Smith, "An overview of the Tesseract OCR engine," in *Proceedings of Ninth International Conference on Document Analysis and Recognition (ICDAR)*, pp. 629–633, 2007.

[16] "Amazon mechanical turk," http://www.mturk.com/.

[17] A. Kittur, E. Chi, and B. Suh, "Crowdsourcing user studies with mechanical turk," in *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*. ACM, 2008, pp. 453–456.

[18] A. Sorokin and D. Forsyth, "Utility data annotation with amazon mechanical turk," in *First IEEE Workshop on Internet Vision, CVPR*, 2008, pp. 1–8.

[19] E. Bursztein, S. Bethard, C. Fabry, J. Mitchell, and D. Jurafsky, "How good are humans at solving captchas? a large scale evaluation," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10, IEEE. Washington, DC, USA: IEEE Computer Society, 2010, pp. 399–413.

[20] J. Ross, L. Irani, M. Silberman, A. Zaldivar, and B. Tomlinson, "Who are the crowdworkers?: shifting demographics in mechanical turk," in *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2010, pp. 2863–2872.

[21] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, "reCAPTCHA: Human-Based Character Recognition via Web Security Measures," *Science*, vol. 321, no. 5895, pp. 1465–1468, September, 2008.

[22] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski, "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*. Citeseer, 2005, pp. 21–22.

[23] S. Tsujimoto and H. Asada, "Major components of a complete text reading system," *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1133–1149, 1992.

[24] R. Hoffman and J. McCullough, "Segmentation methods for recognition of machine-printed characters," *IBM Journal of Research and Development*, vol. 15, no. 2, pp. 153–165, 1971.

[25] E. Gamma, R. Helm, R. Johnson, J. Vlissides *et al.*, *Design patterns*. Addison-Wesley Reading, MA, 2002, vol. 1.