Future Generation Computer Systems 31 (2014) 59–68

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

FaceDCAPTCHA: Face detection based color image CAPTCHA

Gaurav Goswami [a], Brian M. Powell [b], Mayank Vatsa [a,*], Richa Singh [a], Afzel Noore [b]

[a] Indraprastha Institute of Information Technology (IIIT) Delhi, India

[b] Lane Department of Computer Science and Electrical Engineering, West Virginia University, USA

ARTICLE INFO

Article history:
Received 1 February 2012
Received in revised form
29 July 2012
Accepted 27 August 2012
Available online 14 September 2012

Keywords:
CAPTCHA
Face detection
Web security

ABSTRACT

With data theft and computer break-ins becoming increasingly common, there is a great need for secondary authentication to reduce automated attacks while posing a minimal hindrance to legitimate users. CAPTCHA is one of the possible ways to classify human users and automated scripts. Though text-based CAPTCHAs are used in many applications, they pose a challenge due to language dependency. In this paper, we propose a face image-based CAPTCHA as a potential solution. To solve the CAPTCHA, users must correctly identify visually-distorted human faces embedded in a complex background without selecting any non-human faces. The proposed algorithm generates a CAPTCHA that offers better human accuracy and lower machine attack rates compared to existing approaches.

1. Introduction

Completely Automated Public Turing Test to Tell Computers and Humans Apart or CAPTCHA is designed to distinguish between genuine users and automated scripts [1]. The objective of CAPTCHA is to ensure proper service to genuine users while minimizing the attacks by bots. CAPTCHAs are being used for several services including web and financial services, and to provide security against malicious attacks. Research in CAPTCHA has focused on developing tests that are easy for humans to solve and difficult for automated approaches. Several kinds of challenges can be posed by automatic scripts. For instance, scripts or bots can put a heavy load on the servers and enforce a DoS attack, generate multiple fake accounts (in case of registration forms) which are not profitable to both the service provider and the client [2]. Existing CAPTCHA algorithms can be broadly grouped into three classes: (1) text-based, (2) image-based, and (3) video- and audio-based CAPTCHAs.

Text-based CAPTCHAs are the most common and widely used form. These CAPTCHAs require the users to decipher text that has been visually distorted and rendered as an image. AltaVista CAPTCHA, one of the first text CAPTCHAs, was taken from an optical character recognition (OCR) manual. Distortions were incorporated that were known to reduce OCR accuracy [3]. GIMPY CAPTCHA, similar to the AltaVista CAPTCHA [3,4], used English dictionary words. However, Mori and Malik showed that it can be broken and an attack rate of 92% was achieved against EZ-GIMPY [5], a variant of GIMPY. Further variation by Moy et al. [6] boosted the attack rate to 99%. A major shortcoming of these early approaches was vulnerability to segmentation, where each character could be identified in isolation. This greatly simplifies attacks using optical character recognition techniques. One solution was proposed to design the CAPTCHA such that one-to-one mapping between characters and outlines was distorted. For example, two characters might be connected or one might be split into multiple parts. In the ScatterType CAPTCHA, for example, individual characters were segmented into pieces and then systematically scattered so that they are difficult to reassemble [7]. Megaupload CAPTCHA proposed to use overlapping characters whereas MSN CAPTCHA introduced lines connecting individual characters; however, both have high attack rates of 78% or more [3,8–10]. BaffleText's approach of rendering a mottled black-and-white background and then performing different masking operations with overlapping text was more successful, being attacked in only 25% of the attempts [11]. Different masking techniques similar to BaffleText have subsequently been incorporated into other CAPTCHAs [12].

Rather than designing tests to be non-recognizable via OCR, some CAPTCHAs have taken an approach of using handwritten text images already known to fail optical character recognition. A database of text images obtained from handwritten mail addresses that could not be detected automatically were used in such CAPTCHAs. When full city names were used, humans were able to identify the word 100% of the time but the computer success rate was about 9% [13]. Similarly, reCAPTCHA was designed using the text images scanned from book digitization projects [12]. In reCAPTCHA, users were presented with two text images (one of a word that was unknown and one whose text had been previously

* Corresponding author.
E-mail addresses: gauravgs@iiitd.ac.in (G. Goswami), brian.powell@mail.wvu.edu (B.M. Powell), mayank@iiitd.ac.in (M. Vatsa), rsingh@iiitd.ac.in (R. Singh), afzel.noore@mail.wvu.edu (A. Noore).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.08.013

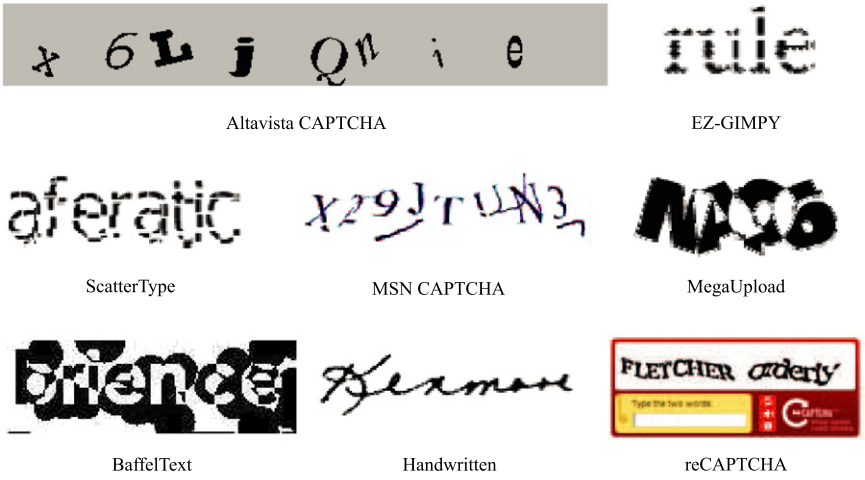

Fig. 1. Example of existing text CAPTCHAs.

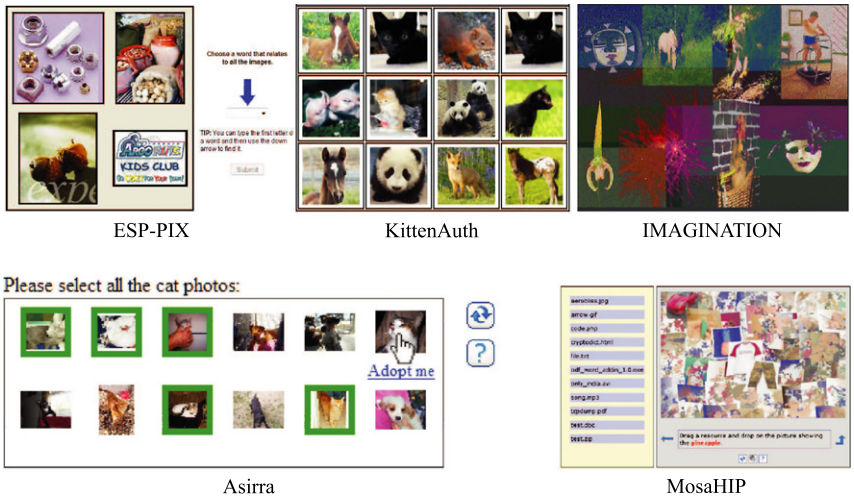

Fig. 2. Example of existing image CAPTCHAs.

determined) and asked to enter both words. The previously-known word served as the test while the currently-unknown word's results were stored to help identify that word for future use. Researchers have shown that the success attack rate for reCAPTCHA is between 5% and 30% [14]. Examples of existing text CAPTCHAs are shown in Fig. 1.

As an alternative to text, several CAPTCHA applications utilize image classification or recognition tasks as part of their test [15]. One basic image-based CAPTCHA is ESP-PIX in which a collection of images are shown and the user has to select a description from a predefined list of categories [16,17]. KittenAuth, a variant of image CAPTCHA, poses images of cats to the user [18]. Asirra is similar to KittenAuth and uses a closed database to source the images [19]. These image-based CAPTCHAs demonstrate a common weakness—a small number of possible solutions for which random guessing can have a high likelihood of success. A number of other CAPTCHAs rely upon composites of multiple embedded images rather than discrete images as with the previous models. The Scene Tagging CAPTCHA requires identifying relationships and relative placement of different images [20]. On the other hand, MosaHIP requires dragging descriptors and dropping them on top of embedded images in a collage [21]. Recently, a new design technique has been proposed that uses recognition of geometric patterns. The IMAGINATION CAPTCHA combines geometric shape recognition with categorization in a two-step process. Users have to first mark the center point of an embedded image and then select an appropriate category based on a predefined list to describe that image [22]. The results show a human success rate of approximately 70% with a machine random guess rate of about 0.0005% [22]. Fig. 2 shows a sample of existing image CAPTCHAs.

Other than text and image CAPTCHAs, video and audio CAPTCHAs have also been proposed. Video-based CAPTCHAs function by posing the tagged videos with descriptive text. In the tests by Kluever, humans achieved an accuracy of 90% in identifying video descriptions while machine attack rates were approximately 13% [3,23]. To provide access for visually-impaired users, audio CAPTCHAs are used as an alternative to standard visual CAPTCHAs. These work by playing a recording of words or letters which users are then asked to enter. However, these CAPTCHAs have high computer attack rates using a speech recognition approach [24–26]. Specifically, the audio CAPTCHAs used by Digg and Google have a successful attack rate of about 70% [24].

1.1. Research contributions

Making CAPTCHAs resilient to attacks by advanced scripts increases the complexity of the tests and language dependency [15]. In some cases, the difficulty has reached levels that are hard even for humans to solve. Since image CAPTCHAs provide language independence and improved user convenience compared to traditional

approaches, recent research has focused on developing image-based CAPTCHAs [18,19,22]. This research presents a color image CAPTCHA that uses complex face detection as a test. The proposed CAPTCHA is a successor of previously proposed face image-based CAPTCHA [27]. In the preliminary version [27], up to five human and non-human face images were selected and embedded on a complex background to create an image CAPTCHA. To solve the CAPTCHA, users had to select all embedded human faces without any false clicks. On a large scale evaluation, we achieved an average accuracy of 80% by human users whereas face detection algorithms achieved 11% accuracy on correctly breaking the CAPTCHA.

In this paper, we present *FaceDCAPTCHA*—face detection based CAPTCHA, in which four to six distorted face/non-face images are embedded in a complex background and a user has to correctly mark the center of all the face images within a defined tolerance. While generating a CAPTCHA, the proposed algorithm leverages the limitations of state-of-the-art automatic face detection algorithms. The CAPTCHA is created in a way that while it is difficult for automatic face detection algorithms to break, it is easy for human users to solve. Also, by incorporating human response in the parameter optimization process, the performance of the FaceDCAPTCHA is enhanced both in terms of higher human performance and lower attack rates by automatic algorithms. The next section provides the details of the proposed algorithm and Section 3 presents the experimental results and key observations.

2. The proposed FaceDCAPTCHA algorithm

Even after decades of research in face detection, there are several challenges in designing effective and accurate face detection algorithms. For example, as shown in Fig. 3, combinations of rotation, noise, blur, background, and occluding key facial features such as eyes and mouth can cause face detection algorithms to fail. On the other hand, the human mind is very effective in segmenting natural faces and one can easily detect such faces even with a complex background and partial/hidden features. Further, automatic face detection algorithms detect cartoon faces as genuine whereas humans can distinguish between genuine faces and cartoon or fake face images. The proposed FaceDCAPTCHA algorithm utilizes the limitations of automatic algorithms to create image CAPTCHAs. In other words, the proposed algorithm is based on optimizing sets of parameters on which standard face detection algorithms fail but humans can succeed. The process of using FaceDCAPTCHA is as follows:

- The face CAPTCHA image containing distorted and occluded genuine face images and fake images on a random background is shown to a user.
- The user marks the approximate center of all genuine face images present in the CAPTCHA.
- If all the responses are correct (i.e. approximate center of all genuine faces are marked correctly) then the test is solved otherwise not.

The hypothesis is that humans can easily pass the test whereas, with carefully designed FaceDCAPTCHA, an automatic program cannot solve it using the state-of-art face detection algorithms.

2.1. Generating FaceDCAPTCHA

The FaceDCAPTCHA generation process can be written as,

$$C = F(\overline{\phi}) \tag{1}$$

where, F is the function applied on a set of distortion parameters, $\overline{\phi}$, to generate the image CAPTCHA C. The function, F, is a series of image processing operations that are used to generate a secure and robust CAPTCHA. The parameters of these image processing operations can be adjusted according to the desired level of complexity. In the proposed approach, an adversarial gradient learning technique is used to learn the range of parameters for which humans can solve the CAPTCHA but not the adversary (i.e. automatic algorithms). The design of FaceDCAPTCHA generation algorithm can be divided into two stages: *estimating CAPTCHA parameters* and *CAPTCHA generation using trained parameters*.

Fig. 3. Face detection algorithms are not able to detect genuine faces due to noise, rotation, background, and hidden facial features.

Estimating CAPTCHA parameters: Since CAPTCHA generation is dependent on several parameters, the training stage involves learning the useful sets of parameters. Here, a set of parameters is said to be useful for CAPTCHA generation if, during the process, humans can successfully solve the CAPTCHA but the adversary/automatic face detection algorithm fails to detect any of the genuine faces. Let H be the human response and Ad represents the adversary's response, the set of parameters is useful if

$$\overline{\phi}_u = Train_{(C=F(\overline{\phi}_i))}(H = 1, Ad = 0). \tag{2}$$

Here, $H = 1$ represents the correct human response to solve the CAPTCHA and $Ad = 0$ represents the incorrect response by the adversary. There can be several parameters, $\overline{\phi}_i$, however only those parameters, $\overline{\phi}_u$, are chosen that satisfy the condition given in Eq. (2). Parameters associated with other conditions, i.e. ($H = 1$, $Ad = 1$), ($H = 0, Ad = 1$), and ($H = 0, Ad = 0$) are not useful. Further, *Train* represents parameter learning in a gradient descent manner. Let $E_{h,Ad}$ be the objective or error function that minimizes the error caused by four conditions associated with H and Ad, i.e.,

$$E_{H,Ad}(\overline{\phi}_t) = \begin{cases} 0 & \text{if } H = 1 \text{ and } Ad = 0 \\ 1 & \text{otherwise.} \end{cases} \tag{3}$$

In gradient descent, optimal parameters are obtained using Eqs. (4) and (5).

$$\bigtriangledown E_{H,Ad}(\overline{\phi}_t) = \frac{\partial E_{H,Ad}(\overline{\phi}_t)}{\partial \overline{\phi}} \tag{4}$$

$$\overline{\phi}_{t+1} = \overline{\phi}_t - \eta \bigtriangledown E_{H,Ad}(\overline{\phi}_t). \tag{5}$$

Here, $\bigtriangledown E_{H,Ad}(\overline{\phi}_t)$ represents the gradient of error function at the tth learning iteration and η is the learning rate that controls the convergence of parameter learning. Note that this learning procedure involves human response (for H) as well as adversarial response for automatic face detector (Ad) and therefore converges to the case where humans can solve the CAPTCHA but not the adversarial algorithm.

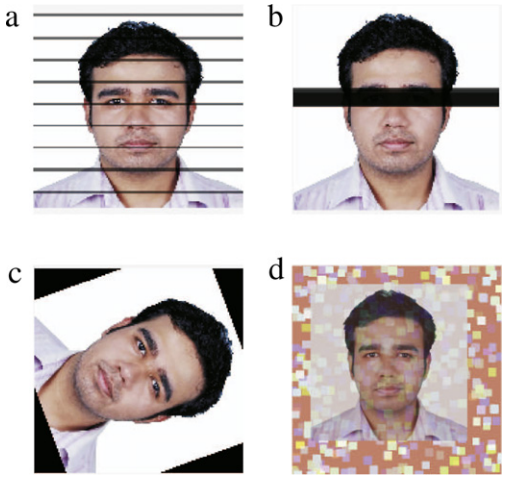

Fig. 4. Illustrating the effect of (a) stripes, (b) strikeout, (c) rotation, and (d) blending with the background.

In the FaceDCAPTCHA design, we choose the following parameters (and related operations):

- The first parameter is the total number of images, both genuine and fake faces, and is represented as n_{total}. Genuine faces are images of real humans collected from different publicly available face databases. Fake faces are images of cartoons and other objects known to generate false positives by automatic face detectors.
- The number of genuine face images in a CAPTCHA, represented as n_{face}, is the second parameter. In a given CAPTCHA,

$$n_{total} = n_{face} + n_{fake} \tag{6}$$

  where n_{fake} is the number of fake images. For a given CAPTCHA, we only need to define n_{total} and n_{face}. Also, randomly changing these parameters in each (new) CAPTCHA can change the content such that only a genuine human user can respond correctly.
- The third parameter, CAPTCHA background **B**, is important to make sure that background has randomness to confuse automatic face detection algorithms. **B** contains parameters such as the number of background shapes to be generated (n_s), the number of dilation operations to be adopted (n_d), and the number of random portions to be placed (n_p).
- Location (x, y) of each constituent image is an important factor. With random location, the segmentation is more difficult than if a static location scheme is used.
- Next, five distortion operations are applied as follows:
  - *Stripes* of three to six pixels width are applied on some constituent images (faces and fake faces) in the CAPTCHA. Note that it is not necessary that this operation is applied uniformly on all face or fake face images in a CAPTCHA. In this operation, the parameters $\mathbf{T_1}$ are the number of stripes and the width of stripes, b. An example of this operation is shown in Fig. 4(a).
  - *Strikeout* operation, as shown in Fig. 4(b), is used to cover key facial features such as eyes and mouth with some transparency. Transparency parameter, $\mathbf{T_2}$, is set such that human users can visualize the facial features but it is challenging for the adversary.
  - *Rotate* operation [28] is used to rotate the constituent face and fake images with θ^0 angle (Fig. 4(c)). Since, for each constituent image, θ may be different, the parameter is represented in vector form such that $\Theta = \{\theta_i\}$ and $i = 1, 2, \ldots, n_{total}$.
  - *Blending operation* [28] is used to smoothly blend the constituent face and fake images with the background, as shown in Fig. 4(d). In this operation, strength of blending S_b is used as the parameter which controls the degree of blending.
  - *Noise addition*. Using the above mentioned parameters and operations, the CAPTCHA image is prepared and then noise is added on the complete CAPTCHA image. In this operation, there are three parameters: $n_{\min}$ and $n_{\max}$ are used as the minimum and maximum percentage of image pixels for application of noise effect and the *type* parameter is used to select the type of noise to be applied (additive, multiplicative or salt & pepper). Collectively, these parameters are referred to as **ns**.

As shown in Fig. 5 and the steps below, these parameters and operations are used to generate the FaceDCAPTCHA.

Step 1: From a set of genuine and false face images, randomly select $n_{face} \geq 2$ (i.e., the number of face images) and $n_{fake} \geq 1$ (i.e., the number of fake images).

Step 2: Each constituent image (both genuine and fake) is processed using the distortion operations (black stripes, strikeouts, and rotate). Images are randomly rotated by θ^0 and using parameters $\mathbf{T_1}$ and $\mathbf{T_2}$, black stripes and strikeout are applied.

Step 3: Each constituent face image is placed at a randomly selected location (x, y) on the CAPTCHA background **B**. To generate the background, random patches for constituent face and fake images are selected. The background image is then created using the following two approaches:

- Random colors: In this approach, the background image is created using random shapes such as circles, squares, and crosses with randomly chosen sizes and colors. These shapes are then pasted on the canvas at random co-ordinates to generate the final background image. This background image is

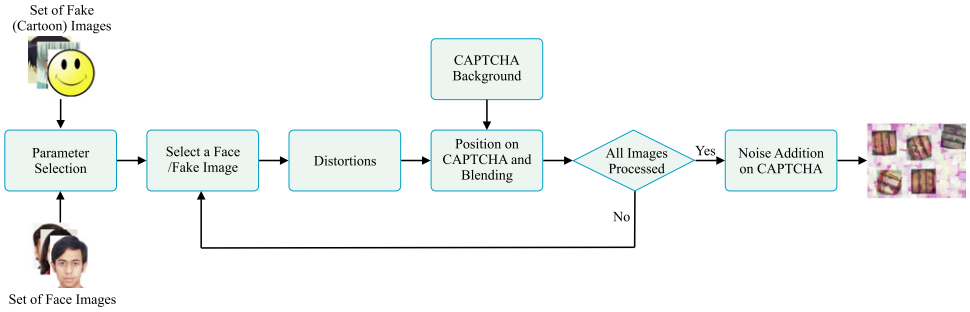

Fig. 5. Illustrating the steps involved in the proposed FaceDCAPTCHA algorithm.

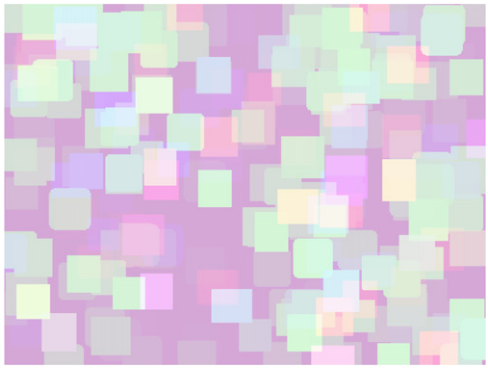

Fig. 6. Example background generated using the random colors approach.

Fig. 7. Example background generated using the random portions approach.

then dilated before being used for CAPTCHA generation. Fig. 6 shows a sample background generated using this approach.

- Random portions: In this approach, the face images selected for generating a particular CAPTCHA are used for its background also. Parts of the face images are randomly picked and stored. These parts are then used along with the random colors algorithm to create the background image. These parts are pasted at random co-ordinates on the background image to include skin color patches on the background that can be useful in confusing skin color based detection algorithms. Fig. 7 shows a sample background generated using this approach.

Step 4: At the end, one of the three noise operations {additive, multiplicative, or salt & pepper}, is applied on the complete CAPTCHA image to generate the final CAPTCHA.

The parameter optimization requires human input as well as a response by automatic face detection algorithms (Eq. (2)). As part of the parameter learning process, a set of 17,000 CAPTCHA images is generated. The images covered 17 different groups of parameter settings and each group has 10 different sets. Therefore, overall, the training is performed with 170 sets of parameters. Furthermore, responses from over 1300 undergraduate students at West Virginia University and IIIT-Delhi collected over a period of three weeks are used for training the CAPTCHA algorithm. Based on their response, out of the 170 sets of parameters, the 10 best-performing sets are used for generating the CAPTCHAs for testing.

CAPTCHA generation using training parameters: Test CAPTCHAs are generated using the best-performing parameter sets (obtained from training) and the four steps described earlier. Fig. 8 shows some FaceDCAPTCHAs and detailed results are presented in Section 3.

2.2. Implementation details

For better understanding and reproducibility, Algorithms 1–8 provide the pseudo-code of the proposed approach (Appendix). The algorithm is developed in C# programming language using Intel's OpenCV library. The main pseudo-code to generate the CAPTCHA is presented in Algorithm 1, *CreateCaptcha*($\overline{\phi}$). Some parameters are initialized, heuristically, for faster convergence, specifically, $n_{face} \geq 2$ and $n_{total} \geq 4$. During training different parameter sets are created with all the combinations, and optimal parameters are estimated using gradient descent algorithm (as described earlier). Further, the OpenCV Haar face detection algorithm is used an an adversary during training, as explained in Algorithm 8.

3. Experimental results and analysis

The proposed FaceDCAPTCHA is evaluated with over 1300 human users and the performance is compared with the automatic face detection algorithm. This section presents the description of images used to generate the CAPTCHA, experimental protocol, and key results.

3.1. Database and protocol

Generating the CAPTCHA images requires human face and cartoon face images. For experimental evaluation, we have selected about 1800 face images from the LFW face database [29] and 300 cartoon images (from photobucket.com). The cartoon images are chosen so that they generate false positives for automatic detection algorithms while they are easy to detect for humans. Each of these images are detected using OpenCV's face detector. 60% of the images are used for FaceDCAPTCHA training and the remaining 40% unseen images are used for testing. For human evaluation, about 1300 volunteers, mostly first year undergraduate students in Science, Mathematics, Arts, and Engineering streams from WVU and IIIT-Delhi contributed to the training and testing of FaceDCAPTCHA. Haar face detector, Google's Picasa, and a commercial face detector are used as the three adversaries to evaluate the robustness of the generated CAPTCHAs.

In both training and testing, each CAPTCHA is evaluated 4–10 times by different human users and once by each automatic face detection algorithm. Therefore, we can compute the average accuracy for human responses (across all CAPTCHAs) using the following equation,

$$Accuracy = \frac{Total\ number\ of\ correct\ responses}{Total\ number\ of\ responses} \times 100 \tag{7}$$

where, a correct response means that in a given CAPTCHA, approximate center of all constituent genuine faces are correctly marked.

3.2. Analysis

With different settings during training, human success rates range from 64% to 86% across various sets. It is observed that the sets with lower computer attack rates also have lower human success rates. Overall, the average success rate for the training phase is 66.81%. It is also observed that many individuals find it difficult to distinguish between actual human faces and comic faces. Users frequently mark the non-human faces as actually being human; after distortions are applied, the life-like images easily become indistinguishable. Based on this finding, the rendered face

Fig. 8. Samples of FaceDCAPTCHA.

Fig. 9. Sample of rendered images that are removed after the training process.

images (as shown in Fig. 9) were removed from the set of fake images. The black stripes placed across some images occluded key facial features and greatly hindered correct recognition; the higher the number of bars and spaces covered, the less likely humans are able to correctly identify the image. This is especially true when stripes are combined with greater transparency on the embedded images so that more background is visible through the image.

After the training phase, responses from human users and automatic algorithms are used to optimize the CAPTCHA generation settings. With optimized parameters, human tests are performed using the final set of 500 CAPTCHA images. The results computed with over 1300 volunteers provide human accuracy of 92.47%. The performance improvements can largely be attributed to the decrease of blending effect on embedded images and addition of noise. In the CAPTCHAs used for testing, embedded images are generally not obscured by the background. Also, the stripes and strikeouts are made translucent as opposed to completely opaque. The reduced use of black stripes to obscure the embedded images also eased the detection process for humans. Key observations are as follows:

- While designing FaceDCAPTCHA, we have considered several image processing operations such as noise, strikeout, stripes, blur, warp, dilation, and erosion as candidate distortions. However, distortions such as blur, dilation, erosion, and warp do not affect automatic face detection performance. Since the objective is to reduce the performance of automatic face detection algorithms while maximizing the human detection performance, the operations that reduce the performance of automatic detection algorithm by over 50% but do not affect human face recognition performance are selected. In the experiments, it is observed that stripes and strikeouts have maximum effect on the performance of automatic algorithms followed by rotation, blending with background, and noise.
- Human users have to mark the approximate center of genuine images. In general, users are not very accurate and therefore a tolerance is provided. During the training stage, evaluation is performed for different tolerance values and it is observed that a tolerance of 80×80 with embedded face images of size 100×100 provides the best human accuracy and still the three automatic face detection algorithms are unable to break the FaceDCAPTCHA.
- To analyze the effect of distortions used during CAPTCHA generation, 11,000 responses are collected from 1300 volunteers. For each distortion, 100 CAPTCHAs are generated totaling 500 CAPTCHAs. Note that in this experiment, only one distortion is used in a CAPTCHA, i.e., if rotation is present in the image, other distortions are not applied on that particular image. Next, the combination of two distortions is also analyzed. With this setup, 100 CAPTCHAs per combination are used, totaling 1000 CAPTCHAs ($5C_2$). The average human success rate for (the above mentioned) 1500 CAPTCHAs is computed and the results in Tables 1 and 2 show that human users can identify face images with all the operations except blending with parameter 0.9 (i.e., more weight on the background).
- In the testing phase with 500 CAPTCHAs, all the volunteers provided responses for 5–12 CAPTCHAs; with a total of over 6000 responses. Also, at least six different volunteers have attempted to solve every CAPTCHA. It is observed that all 500 test CAPTCHAs are successfully solved by at least four volunteers and there were 458 unsuccessful attempts out of the total 6082. This shows that FaceDCAPTCHA has a high human success rate.
- The performance of automatic face detection algorithms is also computed for the same 500 CAPTCHAs and the results show that FaceDCAPTCHA is resilient to automatic algorithms. Picasa yields 0% true positive rates for genuine face images and was therefore not able to successfully solve even a single CAPTCHA. Haar cascade detector included in the OpenCV library is applied on the images after applying full 360° rotation on the images (in steps of 2° at a time) with various parameters and configurations. Haar detected faces in two CAPTCHAs, with one true positive and three false positives in each. Thus, even

Table 1
Effect of individual distortions on the human performance.

Distortion	Parameter	Accuracy (%)
Stripes	4 pixel width	100
	8 pixel width	100
Strike-out	Eye	100
	Mouth	100
Rotate	0°–45°	100
	46°–135°	100
	136°–225°	100
Blending	0.1 (less blend)	100
	0.5	100
	0.9 (more blend)	27
Noise	Additive	100
	Multiplicative	100
	Salt and pepper	100

Table 2
Human performance on combining two distortions.

Distortions	Accuracy (%)
Stripes and strikeout	97
Stripes and rotation	100
Stripes and blending (0.5)	99
Stripes and noise	99
Strike-out and rotation	98
Strike-out and blending (0.5)	95
Strike-out and noise	97
Rotation and blending (0.5)	100
Rotation and noise	100
Blending (0.5) and noise	100

the Haar face detector could not detect any CAPTCHA correctly. A similar result of 0% is obtained with a commercial face detector.

- Image CAPTCHAs such as IMAGINATION (click test), Asirra, and ARTiFACIAL [30] have been attacked by automated approaches [31]. Zhu et al. [31] designed an algorithm for breaking image CAPTCHA and reported an attack accuracy of 74% on IMAGINATION (click test) and 18% for ARTiFACIAL. On applying the same attack approach on 500 CAPTCHAs generated using the proposed CAPTCHA, it yields an accuracy of 0% successful attack rate. The attack algorithm was unable to find all the *genuine* faces in any of the FaceDCAPTCHA images.
- For the random guess attack, i.e. an automatic algorithm randomly guessing the response, we calculate the probability of successfully breaking the FaceDCAPTCHA. Based on the CAPTCHA design, there are $400 \times 300 = 120{,}000$ pixels and there are three cases for correct responses (under the assumption that 80×80 tolerance is given to random guesses also). The probability for breaking the CAPTCHA using a random guess is computed below.
  - *Case* 1: Two genuine face images in the CAPTCHA $\frac{2\times80\times80}{120{,}000} \times \frac{80\times80}{(120{,}000-80\times80)} \times 1/3 = 0.00199$
  - *Case* 2: Three genuine face images in the CAPTCHA $\frac{3\times80\times80}{120{,}000} \times \frac{2\times80\times80}{(120{,}000-2\times80\times80)} \times \frac{80\times80}{(120{,}000-80\times80)} \times 1/3 = 0.00032$
  - *Case* 3: Four genuine face images in the CAPTCHA $\frac{4\times80\times80}{120{,}000} \times \frac{3\times80\times80}{120{,}000-3\times80\times80} \times \frac{2\times80\times80}{(120{,}000-2\times80\times80)} \times \frac{80\times80}{(120{,}000-80\times80)} \times 1/3 = 0.00006$.

  The overall probability of a random guess is therefore 0.00237. However, this calculation only considers the number of genuine images and does not incorporate incorrect guesses due to fake face (cartoon) images. The probability will reduce significantly if we include the incorrect guesses as well. Note that $\frac{1}{3}$ is the probability of guessing the correct number of clicks required for a particular CAPTCHA with the possible values being {2, 3, 4}. We have also attempted to break the CAPTCHA with a random guess based breaking algorithm. However, with over 15 hours of execution on an Intel Core 2 Duo processor running at 2.2 GHz, none of the 500 CAPTCHAs are successfully broken.
- To weigh against existing CAPTCHA, the average human accuracy of Google's text CAPTCHA is compared with the proposed FaceDCAPTCHA. With 25 volunteers, this comparison is performed by randomly downloading 100 Google's CAPTCHAs and randomly selecting 100 test FaceDCAPTCHAs. With 250 responses for both text and face CAPTCHA, the average human accuracy of FaceDCAPTCHA is 8% better than Google's text CAPTCHA. Though this is a very small sample to make any strong conclusions, the volunteers response suggests that FaceDCAPTCHA is easier to solve.
- For a FaceDCAPTCHA of size $N \times M$, the complexity of the proposed algorithm is $O(NM)$. On a 2.2 GHz Core 2 Duo CPU, the current implementation requires only 10–15 ms to generate a FaceDCAPTCHA, depending on the parameter values ($\bar{\phi}$). Further, the proposed CAPTCHA algorithm is scalable. We can use images from several sources, including publicly available face and fake-face databases. Also, the combination of different transformations and parameters further increases the number of CAPTCHAs that can be generated using this algorithm.
- Here we would like to quote a paragraph from Datta et al. [22], "*As reported recently* [32], *humans are being used to solve them, either in a well-organized manner commercially (in low labor cost regions), or by the use of games and other methods whereby humans are unaware that their responses are being used for malicious purposes. These attempts make it futile to create harder AI problems, because in principle, a CAPTCHA should be solvable by virtually all humans, regardless of their intent. Nonetheless, CAPTCHAs are and will continue to remain deployed until alternate, unbreakable, human identity verification methods become practical. Till then, they should, at the very least, serve to impede the intensity of human-guided breaking of CAPTCHAs*".

  Similar to Datta et al., we also agree that CAPTCHAs are intended for humans to solve and any human in the loop attack may break any CAPTCHA, including the proposed one. Since the FaceDCAPTCHA generation process involves (i) randomly choosing genuine and fake face images, (ii) applying distortions, and (iii) generating random background using the constituent images, we believe that it may be challenging to break FaceDCAPTCHA.

4. Conclusion

This paper presents the FaceDCAPTCHA[1] algorithm that utilizes the difference between face detection capabilities of humans and automated algorithms. By combining face detection with visual distortions optimized through a training–testing process, it is possible to create a test that is simple for human users to solve while effectively eliminating automated attacks. The proposed methodology offers major benefits over traditional text-based CAPTCHAs, most notably language independence.

[1] http://research.iiitd.edu.in/groups/iab/facedcaptcha.html

By incorporating the proposed FaceDCAPTCHA into existing online authentication schemes, developers can substantially reduce the likelihood of credentials-based attacks. In requiring users to solve the CAPTCHA in addition to providing a username and password, an additional dimension of complexity can be added that requires human effort. The FaceDCAPTCHA's point-and-click-based implementation adds this additional stage with minimum difficulty for users. It can be readily used on mobile devices since it has no language requirements and does not require a keyboard for data entry.

Appendix

A.1. FaceDCAPTCHA algorithm

Algorithm 1 *CreateCaptcha*($\overline{\phi}$)

Input: $\overline{\phi} \in \{n_{total}, n_{face}, n_{fake}, \mathbf{B}, (x, y), \mathbf{T_1}, \mathbf{T_2}, \Theta, S_b, n_s\}$.
Process: Initialize n_{total} and n_{face}. For representation, $\overline{\phi}$ is used as a common parameter set for all the functions. However, when a function is invoked, only relevant parameters are passed.
G = *init_grids*(n_{total}) /*Initializes the set of grids (each grid has top-left corner coordinates and height, width) based on the number of grids needed on a 400 × 300 image */
S = *SelectFiles*($\overline{\phi}$) (Algorithm 2) /* Select files associated to face and fake images */
r = *Random*(0, 2) /* Generate a random number within the given limits */
C = *newImage*(400, 300) /* Workspace for generating CAPTCHA C */
if ($r = 0$) **then**
  C = *Background1*($\overline{\phi}$) (Algorithm 3) /*Background generation - Random portions approach */
else
  C = *Background2*($\overline{\phi}$) (Algorithm 4) /*Background generation - Random colors approach */
while $S \neq null$ **do**
  i = *Random*(0, n_{total})
  $I = S[i]$
  I = *Stripes*($I, \overline{\phi}$) (Algorithm 5) /* Stripes in the constituent face */
  I = *Strikeout*($I, \overline{\phi}$) (Algorithm 6) /*Strikeout key facial facial regions */
  I = *Rotate*(I, θ_i) /* Rotate the image */
  j = *Random*(0, n_{total})
  $x = G[j].x$ + *Random*(0, $width(G[j]) - width(I)$)
  $y = G[j].y$ + *Random*(0, $height(G[j]) - height(I)$)
  C = *Blend*($C, I, x, y, \overline{\phi}$) /* Smoothly blends the constituent face and fake face images with the background using blending strength s_b */
end
C = *noise*($C, \overline{\phi}$) (Algorithm 7) /* Adding noise to the CAPTCHA */
return C
Output: The CAPTCHA image C containing n_{total} images out of which n_{face} are genuine faces.

Algorithm 2 *SelectFiles*(DB_{face}, DB_{fake}, $n_{total,}n_{face}$)

Input: DB_{face} denotes the set of all genuine face images. DB_{fake} denotes the set of all the fake face (cartoon) images. n_{total} is the total number of images in the CAPTCHA and n_{face} is the number of genuine human faces in the CAPTCHA.
Process: S = *new List*(n_{total})
for $k = 1$ to n_{face} **do**
  i = *random*(0, *size*(DB_{face}))
  add $DB_{face}[i]$ to S
for $k = 1$ to ($n_{total} - n_{face}$) **do**
  i = *random*(0, *size*(DB_{fake}))
  add $DB_{fake}[i]$ to S
return S

Algorithm 3 *Background1*(n_s, n_d, n_p, I, M, N, M_p, N_p)

Input: n_s denotes the number of shapes to be generated for the background. n_d denotes the number of dilations to be applied to the background image after generation. n_p denotes the number of random portions to be placed. I denotes a face image. M, N is the size of the CAPTCHA for which the background has to be generated and M_p, N_p is the size of a random portion.
Process: B = *newImage*(M, N)
for $k = 1$ to n_s **do**
  c = *random*()
  S_p = *Shape*(c) /* Shape(c) is a method that generates a randomly sized shape with color c */
  x_r = *random*(0, M)
  y_r = *random*(0, N)
  for $x = 0$ to $width(S_p)$ **do**
    for $y = 0$ to $height(S_p)$ **do**
      $B(x + x_r, y + y_r) = S_p(x, y)$
for $k = 1$ to n_p **do**
  x_r = *random*(0, $width(I)$)
  y_r = *random*(0, $height(I)$)
  P = *newImage*(M_p, N_p)
  for $x = 0$ to M_p **do**
    for $y = 0$ to N_p **do**
      $P(x, y) = I(x + x_r, y + y_r)$
for $k = 1$ to n_p **do**
  x_r = *random*(0, M)
  y_r = *random*(0, N)
  blend(B, P, x_r, y_r)
return B
Output: The background image B.

Algorithm 4 *Background2*(n_s, n_d, I, M, N)

Input: n_s denotes the number of shapes to be generated for the background. n_d denotes the number of dilations to be applied to the background image after generation. I denotes a face image and M, N is the size of the CAPTCHA for which the background has to be generated.
Process: B = *newImage*(M, N)
for $k = 1$ to n_s **do**
  x_r = *random*(0, $width(I)$)
  y_r = *random*(0, $height(I)$)
  $c = I(x_r, y_r)$
  S_p = *Shape*(c) /* Shape(c) is a method that generates a randomly sized shape with color c */
  x_r = *random*(0, M)
  y_r = *random*(0, N)
  for $x = 0$ to $width(S_p)$ **do**
    for $y = 0$ to $height(S_p)$ **do**
      $B(x + x_r, y + y_r) = S_p(x, y)$
B = *Dilate*(B, n_d)
return B
Output: The background image B.

Algorithm 5 *Stripes*($I, \mathbf{T_1}$)

Input: I is the input image on which the stripes effect is to be applied. Parameter $\mathbf{T_1}$ consists of h_{min} and h_{max}, the lower and upper bounds for the height of one stripe respectively, w_{orig} and w_{str}, the weights assigned to actual image pixel and $c_{stripes}$ colored pixel respectively when taking the weighted average for the final output pixel, and f, the spacing between consecutive stripes (represented as a fraction of the image height, e.g. 20 means the space between two consecutive stripes in an image of height h would be $h/20$) and $c_{stripes}$ is the color of the stripes.
Process: for $y = height(I)/f$ **to** $height(I)$ **do**
  h = *random*(h_{min}, h_{max})
  for $i = y$ to $(y + h)$ **do**
    for $j = 0$ to $width(I)$ **do**
      $I(j, i) = w_{orig} * I(j, i) + w_{str} * c_{stripes}$
  $y = y + h + height(I)/f$
return I
Output: The output image I with stripes effect applied.

Algorithm 6 *Strikeout*$(I, \mathbf{T_2})$

Input: I denotes the input face image. $\mathbf{T_2}$ contains c_{strike}: the color of the strikeout strips, w_{orig} and w_{str}: the weights assigned to actual image pixel, and c_{strike}: colored pixel respectively when taking the weighted average for the final output pixel.
Process: $result = detectEyePair(I)$ /* Result contains the pixels belonging to the detected eye pair (which is a subset of the pixels of I). If more than one pair of eyes is detected then we take just the first eye pair */
if $(result \neq null)$ **then**
  for each pixel in *result* **do**
    pixel = pixel$^*w_{orig}$ + $c_{strike} * w_{str}$
else
  $result = detectMouth(I)$ /* Result contains the pixels belonging to the detected mouth (which is a subset of the pixels of I). If more than one mouth is detected then we take just the first mouth */
if $result \neq null$ **then**
  for each pixel in *result* **do**
    pixel = pixel$^*w_{orig}$ + $c_{strike} * w_{str}$
return I
Output: The input image I with the strikeouts applied on detected eye pair or mouth.

Algorithm 7 *Noise*$(C, \mathbf{n_s})$

Input: C is the input image which has to be imbued with noise. $\mathbf{n_s}$ contains n_{min} and n_{max} which denote the minimum and the maximum percentage of image pixels for application of noise effect respectively, and *type* which denotes the type of noise to be applied (additive, multiplicative or salt and pepper).
Process: $n_{perc} = random(n_{min}, n_{max})$
$n_{pixels} = \frac{(n_{perc} * total\ pixels\ in\ I)}{100}$
for $k = 1$ to n_{pixels} **do**
  $x_r = random(0, width(C))$
  $y_r = random(0, height(C))$
  if $(type = addictive)$ **then**
    add additive noise to $C(x_r, y_r)$
  if $(type = multiplicative)$ **then**
    add multiplicative noise to $C(x_r, y_r)$
  if $(type = salt\&pepper)$ **then**
    add salt and pepper noise to $C(x_r, y_r)$
return I
Output: Image C with added noise.

Algorithm 8 *CAPTCHA_Break*

Input: The CAPTCHA Image to break, C, the number of genuine face images in the CAPTCHA, n_{face}, and the set of coordinates of genuine face images for the particular CAPTCHA, $(\mathbf{X}, \mathbf{Y})$.
Process: $CAPTCHA\_Break = false$
  result = detectHaarCascade() /* OpenCV's Haar face detector used as an adversary */
for each point in result **do**
  if point lies in $(\mathbf{X}, \mathbf{Y})$ **then**
    $C_{correct}++$;
if $C_{correct} == n_{face}$ **then**
  $CAPTCHA\_break = True$;
Output: CAPTCHA break: True/False

References

[1] S. Shirali-Shahreza, M.H. Shirali-Shahreza, Bibliography of works done on CAPTCHA, in: Proceedings of the 3rd International Conference on Intelligent System and Knowledge Engineering, 2008, vol. 1, pp. 205–210.
[2] J. Mirkovic, P. Reiher, A taxonomy of ddos attack and ddos defense mechanisms, ACM SIGCOMM Computer Communication Review 34 (2) (2004) 39–53.
[3] K.A. Kluever, Evaluating the usability and security of a video CAPTCHA, Master's Thesis, Rochester Institute of Technology, 2008.
[4] H.S. Baird, K. Popat, Human interactive proofs and document image analysis, in: Document Analysis Systems, 2002, pp. 531–537.
[5] G. Mori, J. Malik, Recognizing objects in adversarial clutter: breaking a visual CAPTCHA, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003, vol. 1, pp. 134–141.
[6] G. Moy, N. Jones, C. Harkless, R. Potter, Distortion estimation techniques in solving visual CAPTCHAs, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004, vol. 2, pp. 23–28.
[7] H.S. Baird, T. Riopka, Scattertype: a reading CAPTCHA resistant to segmentation attack, in: Proceedings of the SPIE Conference on Document Recognition and Retrieval, 2005, pp. 16–20.
[8] A.S. El Ahmad, J. Yan, L. Marshall, The robustness of a new CAPTCHA, in: Proceedings of the 3rd European Workshop on System Security, 2010, pp. 36–41.
[9] J. Yan, A.S. El Ahmad, A low-cost attack on a Microsoft CAPTCHA, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, 2008, pp. 543–554.
[10] P.S. Richard, R. Szeliski, J. Benaloh, J. Couvreur, I. Calinov, Using character recognition and segmentation to tell computer from humans, in: Proceedings of the IAPR International Conference on Document Analysis and Recognition, 2003, pp. 418–423.
[11] M. Chew, H.S. Baird, Baffletext: a human interactive proof, in: Proceedings of the Document Recognition and Retrieval, 2003, pp. 305–316.
[12] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, M. Blum, ReCAPTCHA: human-based character recognition via web security measures, Science 321 (2008) 1465–1468.
[13] A. Rusu, V. Govindaraju, Handwritten CAPTCHA: using the difference in the abilities of humans and machines in reading handwritten words, in: Proceedings of the 9th International Workshop on Frontiers in Handwriting Recognition, 2004, pp. 226–231.
[14] P. Baecher, N. Buscher, M. Fischlin, B. Milde, Breaking reCAPTCHA: a holistic approach via shape recognition, Future Challenges in Security and Privacy for Academia and Industry 354 (2011) 56–67.
[15] J. Yan, A.S. El Ahmad, Usability of CAPTCHAs or usability issues in CAPTCHA design, in: Proceedings of the 4th Symposium on Usable Privacy and Security, 2008, pp. 44–52.
[16] The official CAPTCHA site, http://www.captcha.net.
[17] ESP-pix, Carnegie Mellon University, http://www.captcha.net.
[18] O. Warner, The cutest human-test: Kittenauth, http://thepcspy.com/read/the_cutest_humantest_kittenauth/.
[19] J. Elson, J.R. Douceur, J. Howell, J. Saul, Asirra: a CAPTCHA that exploits interest-aligned manual image categorization, in: Proceedings of the 14th ACM Conference on Computer and Communications Security, 2007, pp. 366–374.
[20] P. Golle, Machine learning attacks against the Asirra CAPTCHA, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, 2008, pp. 535–542.
[21] A. Basso, S. Sicco, Preventing massive automated access to web resources, Computers and Security 28 (3–4) (2009) 174–188.
[22] R. Datta, J. Li, J.Z. Wang, Exploiting the human–machine gap in image recognition for designing CAPTCHA, IEEE Transactions on Information Forensics and Security 4 (2009) 504–518.
[23] K.A. Kluever, R. Zanibbi, Balancing usability and security in a video CAPTCHA, in: Proceedings of the 5th Symposium on Usable Privacy and Security, 2009, pp. 14:1–14:11.
[24] J. Tam, J. Simsa, S. Hyde, L. von Ahn, Breaking audio CAPTCHAs, in: Proceedings of the Neural Information Processing Systems, 2008, pp. 1625–1632.
[25] E. Bursztein, S. Bethard, DeCAPTCHA: breaking 75% of ebay audio CAPTCHAs, in: Proceedings of the 3rd USENIX Conference on Offensive Technologies, 2009.
[26] R. Santamarta, Breaking gmail's audio CAPTCHA, http://blog.wintercore.com/?m=200803.
[27] B.M. Powell, A.C. Day, R. Singh, M. Vatsa, A. Noore, Image-based face detection CAPTCHA for improved security, International Journal of Multimedia Intelligence and Security 1 (2010) 269–284.
[28] R.C. Gonzalez, R.E. Woods, Digital Image Processing (3rd Edition), Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.
[29] G.B. Huang, M. Ramesh, T. Berg, E. Learned-Miller, Labeled faces in the wild: a database for studying face recognition in unconstrained environments, Technical Report 07-49, University of Massachusetts, Amherst, October 2007.
[30] Y. Rui, Z. Liu, Artifacial: automated reverse Turing test using facial features, in: Proceedings of the 11th ACM International Conference on Multimedia, 2003, pp. 295–298.
[31] B.B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, M. Yi, K. Cai, Attacks and design of image recognition CAPTCHAs, in: Proceedings of the 17th ACM Conference on Computer and Communications Security, 2010, pp. 187–200.
[32] How CAPTCHA was foiled: are you a man or a mouse?, http://www.guardian.co.uk/technology/2008/aug/28/internet.captcha.

Gaurav Goswami received his Bachelor in Technology degree in Information Technology in 2012 from the Indraprastha Institute of Information Technology (IIIT) Delhi, India where he is currently pursuing a Ph.D. His main areas of interest are image processing, computer vision and their application in biometrics.

Brian M. Powell received his M.S. in Computer Science from West Virginia University, USA in 2006. He is currently a Doctoral student in the Lane Department of Computer Science and Electrical Engineering at West Virginia University. His areas of interest are human interactive proofs, human computation, user interface design and computer science education. He is a member of the IEEE, Computer Society and the Association for Computing Machinery. He is also a member of the Phi Kappa Phi, Upsilon Pi Epsilon and Sigma Zeta honor societies. He was the recipient of the West Virginia University Foundation Distinguished Doctoral Fellowship.

Mayank Vatsa received his M.S. and Ph.D. degrees in computer science in 2005 and 2008, respectively from West Virginia University, Morgantown, USA. He is currently an Assistant Professor at the Indraprastha Institute of Information Technology (IIIT) Delhi, India. He has more than 100 publications in refereed journals, book chapters, and conferences. His research has been funded by the UIDAI and DIT. He is the recipient of FAST award by DST, India. His areas of interest are biometrics, image processing, computer vision, and information fusion. Dr. Vatsa is a member of the IEEE, Computer Society and Association for Computing Machinery. He is also a member of the Golden Key International, Phi Kappa Phi, Tau Beta Pi, Sigma Xi, Upsilon Pi Epsilon, and Eta Kappa Nu honor societies. He is the recipient of 11 best paper and best poster awards in international conferences. He is also an area editor of IEEE Biometric Compendium.

Richa Singh received her M.S. and Ph.D. degrees in computer science in 2005 and 2008, respectively from West Virginia University, Morgantown, USA. She is currently an Assistant Professor at the Indraprastha Institute of Information Technology (IIIT) Delhi, India. Her research has been funded by the UIDAI and DIT, India. She is a recipient of FAST award by DST, India. Her areas of interest are biometrics, pattern recognition, and machine learning. She has more than 100 publications in refereed journals, book chapters, and conferences. She is also an editorial board member of Information Fusion, Elsevier. Dr. Singh is a member of the CDEFFS, IEEE, Computer Society and the Association for Computing Machinery. She is also a member of the Golden Key International, Phi Kappa Phi, Tau Beta Pi, Upsilon Pi Epsilon, and Eta Kappa Nu honor societies. She is the recipient of 11 best paper and best poster awards in international conferences.

Afzel Noore received his Ph.D. in Electrical Engineering from West Virginia University, USA. He was a Digital Design Engineer with Philips, India. From 1996 to 2003, he was the Associate Dean for Academic Affairs and Special Assistant to the Dean in the College of Engineering and Mineral Resources, West Virginia University. He is currently a Professor in the Lane Department of Computer Science and Electrical Engineering. His research interests include CAPTCHA based security, computational intelligence, biometrics, software reliability modeling, machine learning, hardware description languages and quantum computing. His research has been funded by NASA, the National Science Foundation, Westinghouse, General Electric, Electric Power Research Institute, the US Department of Energy, the US Department of Justice and the Army Research Lab. He serves on the Editorial Boards of Recent Patents on Engineering, the Open Nanoscience Journal and the International Journal of Multimedia Intelligence and Security. He has over 100 publications in refereed journals, book chapters and conferences. He has received several outstanding teacher and outstanding researcher awards. He is a Senior Member of the IEEE and member of Phi Kappa Phi, Sigma Xi, Eta Kappa Nu and Tau Beta Pi honor societies. He is the recipient of seven best paper and best poster awards in international conferences.