On the Security and Usability of Dynamic Cognitive Game CAPTCHAs

Manar Mohamed[a], Song Gao[b], Niharika Sachdeva[c], Nitesh Saxena[d,*], Chengcui Zhang[d], Ponnurangam Kumaraguru[c], and Paul C. Van Oorschot[e]

[a] *Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA*
[b] *Google, USA*
[c] *IIIT-Delhi, India*
[d] *Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL, USA*
[e] *School of Computer Science, Carleton University, Ottawa, ON, Canada*

Abstract.
Existing CAPTCHA solutions are a major source of user frustration on the Internet today, frequently forcing companies to lose customers and business. Game CAPTCHAs are a promising approach which may make CAPTCHA solving a fun activity for the user. One category of such CAPTCHAs – called *Dynamic Cognitive Game (DCG) CAPTCHA* – challenges the user to perform a game-like cognitive (or recognition) task interacting with a series of dynamic images. Specifically, it takes the form of many objects floating around within the images, and the user's task is to match the objects corresponding to specific target(s), and drag/drop them to the target region(s).

In this paper, we pursue a comprehensive analysis of DCG CAPTCHAs. We design and implement such CAPTCHAs, and dissect them across four broad but overlapping dimensions: (1) usability, (2) fully automated attacks, (3) human-solving relay attacks, and (4) hybrid attacks that combine the strengths of automated and relay attacks. Our study shows that DCG CAPTCHAs are highly usable, even on mobile devices and offer some resilience to relay attacks, but they are vulnerable to our proposed automated and hybrid attacks.

Keywords: CAPTCHA, web-security, relay attack, hybrid attack, visual processing

1. Introduction

The abuse of the resources of online services using automated means, such as denial-of-service or password dictionary attacks, is a common security problem. To prevent such abuse, a primary defense mechanism is CAPTCHA [2] (denoted "captcha"), a tool aimed to distinguish a human user from a computer based on a task that is easier for the former but much harder for the latter.

The most commonly encountered captchas today take the form of a garbled string of words or characters, but many other variants have also been proposed (we refer the reader to [6,22,39] which provide

[*] Corresponding author. E-mail: saxena@uab.edu.

0926-227X/17/$27.50 © 2017 – IOS Press and the authors. All rights reserved

excellent review of different captcha categories). Unfortunately, existing captchas suffer from several problems. *First*, successful automated attacks have been developed against many existing schemes. For example, algorithms have been designed that can achieve character segmentation with a 90% success rate [17,23,25,33]. *Second*, low-cost attacks have been conceived whereby challenges are relayed to, and solved by, users on different web sites or paid human-solvers in the crowd [9,14,19]. In fact, it has been shown that [30] such relay attacks are much more viable in practice than automated attacks due to their simplicity and low economical costs. *Third*, the same distortions that are used to hide the underlying content of a captcha puzzle from computers can also severely degrade human usability [8,40].

Given these problems, there is a need to consider alternatives that place the *human user at the center of the captcha design*. *Game captchas* offer a promising approach by attempting to make captcha solving a fun activity for the users. These are challenges that are built using games that might be enjoyable and easy for humans, but hard for computers.

In this paper, we focus on a broad form of game captchas, called *Dynamic Cognitive Game (DCG)* captchas. This captcha challenges the user to perform a *game*-like *cognitive* task interacting with a series of *dynamic* images. Specifically, we consider a representative DCG captcha category which involves objects floating around within the images, and the user's task is to match (i.e. drag/drop) the objects with their respective target(s). A startup called "are you a human" [4] has recently been offering such DCG captchas.

Besides promising to significantly improve user experience, DCG captchas are an appealing platform for touch screen enabled mobile devices (such as smartphones). Traditional captchas are known to be quite difficult on such devices due to their small displays and key/touch pads [31], while touch screen games are much easier and already popular. Motivated by these unique advantages of DCG captchas, we set out to investigate their security and usability. Specifically, we pursue a comprehensive study of DCG captchas, analyzing them from four broad yet intersecting dimensions: (1) *usability*, (2) *fully automated attacks*, (3) *human-solver relay attacks*, and (4) *hybrid attacks*. Our main contributions are as follows:

1. **DCG CAPTCHA Design:** We formalize, design and implement instances of a representative category of DCG captchas. (*Sections 2*)
2. **Usability Studies:** We conduct three usability studies of DCG captchas. Two of these are web-based studies, one "in-lab" and one online using a crowdsourcing platform. The third one is a mobile-based study. All the studies evaluate the DCG captchas in terms of time-to-completion, error rates and perceived usability. The results of our studies indicate the overall usability to be very good. (*Section 3*)
3. **Automated Attacks:** We develop a novel, fully automated framework to attack these DCG captcha instances based on image processing techniques. The attack is computationally efficient and highly accurate, but requires building a dictionary to be effective. (*Section 4*)
4. **Relay Attacks, and Relay Attack Detection:** We explore *Stream Relay* attack in which the game frames and responses are streamed between the attacker and a human-solver. We conducted user studies for the proposed relay attack and our results shows that Stream Relay is detectable with high accuracy by our proposed machine learning based relay attack detection algorithm. (*Section 5*)
5. **Hybrid Attacks:** We develop a novel and powerful hybrid attack framework that carefully combines the strength of automated algorithms and relay attacks, and overcome the limitations of each. Basically, the bot would send a single snapshot of the game to the human-solver and keep tracking the objects while it is waiting for the human response. The human-solver task is to draw line(s) form the answer object(s) to respective target(s). We evaluated the performance and usability of

the proposed attack. Our results indicate our attack is usable for a remote solver, and efficient in breaking broad categories of DCG captchas. (*Section 6*)

This paper is a combination and consolidation of our prior conference publications [18,28,29], which serves to integrate multiple studies in a single archival journal paper. Moreover, we report on two new usability studies. The **first** new usability study aims to evaluate the usability of DCG on mobile devices. The **second** new usability study is for the underlying human task in the hybrid attacks in order to measure the performance of our proposed attacks, evaluate the participants' performance in the task, and compare the performance and usability in this task with the performance and usability of playing the CAPTCHA games directly by legitimate users in a benign setting.

2. Background

We use the term Dynamic Cognitive Game (DCG) captcha to define the broad captcha schemes that form the focus of our work. We characterize a DCG captcha as having the following features: (1) *dynamic* because it involves objects moving around in image frames; (2) either *cognitive* because it is a form of a puzzle that relates to the semantics of the images or *image recognition* because it involves visual recognition; and (3) a *game* because it aims to make captcha solving task a fun activity for the user. In this section, we discuss the security model and design choices for DCG, and present the DCG categories and associated instances studied in this paper.

2.1. Security Model and Design Choices

The DCG captcha design objective is the same as that of captcha: a bot (automated computer program) must only be able to solve captcha challenges with no better than a negligible probability (but a human should be able to solve with a sufficiently high probability)[1].

A pre-requisite for the security of a DCG captcha implementation (or any captcha for that matter) is that the responses to the challenge must not be provided to the client machine in clear text. For example, in a character recognition captcha, the characters embedded within the images should not be leaked out to the client. To avoid such leakage in the context of DCG captchas, it is important to provide a suitable underlying game platform for run-time support of the implemented captcha. Web-based games are commonly developed using Flash or HTML5 in conjunction with JavaScript. However, both these platforms operate by downloading the game code to the client machine and executing it locally. Thus, if these game platforms were directly used to implement DCG captchas, the client machine will know the correct objects and the positions of their corresponding target(s), which can be used by the bot to construct the responses to the server challenges relatively easily. This will undermine the security of DCG captchas.

The above problem can be addressed by employing encryption and obfuscation of the game code which will make it difficult for the attacker (bot) on the client machine to extract the game code and thus the correct responses. Commercial tools, such as SWF Encrypt [3], exist which can be used to achieve this functionality. This approach works under a security model in which it is assumed that the bot does not have the capability to learn the keys used to decrypt the code and to deobfuscate the code. A similar

[1] For example, target thresholds might limit bot success rates below 0.6% [42], and human user success rates above 90% [11].

model where the attacker has only partial control over the client machine has also been employed in prior work [35].

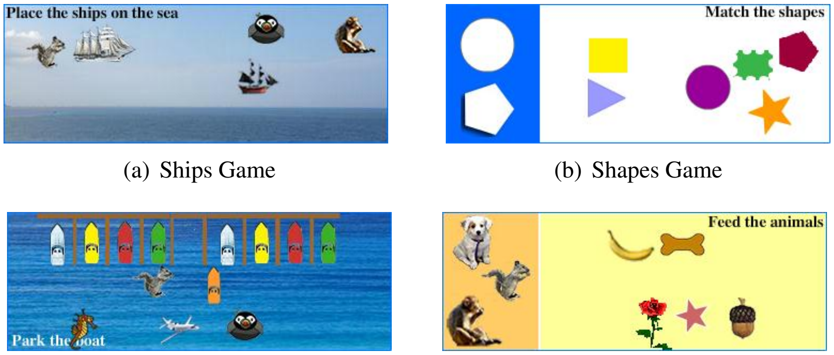
In addition to automated attacks, the security model for DCG captchas (and any other captcha) must also consider human-solver relay attacks [9,30]. In fact, it has been shown that such relay attacks are much more appealing to the attackers than automated attacks currently due to their simplicity and low cost [30]. In a relay attack, the bot forwards the captcha challenges to a human user elsewhere on the Internet (either a payed solver or an unsuspecting user accessing a web-site [15]); the user solves the challenges and sends the responses back to the bot; and the bot simply relays these responses to the server. Unfortunately, most, if not all, existing captcha solutions are insecure under such relay attack model. For example, character recognition captchas are routinely broken via such relay attacks [30]. For DCG captchas to offer better security than existing captchas, they should provide some resistance to such human-solver relay attacks (this is indeed the case as we demonstrate in Section 5).

2.2. DCG CAPTCHA Instances and Prototypes

Due to the legal restrictions on attacking commercial DCG CAPTCHAs, we proceeded to develop our own animation-based DCG prototypes for the purpose of our study. Using Adobe Flash, we implemented four captcha games that represented a broad class of DCGs. These games are 360x130 pixels in size, and seamlessly fit into web pages if used for practical purposes.

The DCG captchas implemented for the purpose of this study are shown in Figure 1. Each DCG captcha can be characterized by the following distinct components.

- *Answer object* – a moving object that should be dragged to the corresponding target object in order to successfully complete the game. For the parking game shown in Figure 1(c), the orange boat, that can be dragged to the empty dock position to complete the game, is the answer object.
- *Target object* – an object onto which the corresponding answer object should be dragged.
- *Target area* – the area within which the target objects reside.
- *Activity area* – the area within which the foreground objects move.



(a) Ships Game

(b) Shapes Game

(c) Parking Game

(d) Animals Game

Fig. 1. Static snapshots of 4 game instances of a representative DCG captcha analyzed in this paper (targets are static; objects are mobile)

DCG captchas are classified according to the number of target objects. The Ships game (Figure 1(a)), is a one-target DCG type, where the sea is the target object. So the ships, that are the answer objects, can be dropped anywhere within the sea. The Shape game (Figure 1(b)) has a circle and a pentagon placed on the left side of the game as the two target objects. The Animal game (Figure 1(d)) is a three-target instance of DCG captcha. The Parking game (Figure 1(c)) is a variant of DCG captcha where there is no target object but a target area (the empty parking space) onto which the boat should be dragged.

To complete a DCG captcha game, a user has to drag and drop all answer objects to their corresponding target objects. For example, in the Animal game, the user has to drag the bone to the dog, the acorn to the squirrel, and the banana to the monkey. The game is considered incomplete, and the user is rejected in case the game is not completed within 60s.

Each foreground object has an initial pre-specified location in the activity area. The direction of movement of objects is randomly chosen from 8 possible directions – north (N), south (S), east (E), west (W), NE, NW, SE and SW. For horizontal and vertical movements, objects move 1 pixel per frame. For diagonal movements, the objects move 1.414 pixels per frame. The frame rate for the games is set at 40 frames per second. Hence, the foreground objects move at an average speed of $(((1 + 1.414)/2) * 40)$, i.e., 48.28 pixels per second. An object continues moving in its current direction until it collides with either another object or the game border. A collision results in an object moving towards a new random direction.

For each of the 4 games, we set 5 parameterizations, choosing number of moving objects as (4, 5, 6), and object speed as (10, 20, 40) frames per second (FPS) (These frame rates translate into average object speeds of 12.1, 24.2 and 48.4 pixels/second, resp.). For each game, we used 5 combinations of speed and number of objects: (10 FPS, 4 objects); (20 FPS, 4 objects); (20 FPS, 5 objects); (20 FPS, 6 objects); and (40 FPS, 4 objects). This resulted in a total of 20 games in our corpus.

3. Usability

In this section, we report our usability studies of our representative DCG captcha category. The first study is student based which we used to measure the usability of the DCG captcha with respect to different parameters as explained in Section 2.2. Then, to validate our study results we preformed another study on Amazon Mechanical Turk (MTurk) is which we tested only one variant of each game instance. Finally, we present our mobile-based study that we conduct to evaluate the usability of DCG captchas on mobile devices.

3.1. Study Design, Goals, and Process

Our **first user study** involved 40 participants who were primarily students from various backgrounds. (For demographics, see the second column of Table 1). The participants were provided with 20 instances as discussed in Section 2.2 in succession, aimed at understanding how different parameterizations impact users' solving capabilities, and the game completion time, and the number of object-drags were recorded. The order of the games presented to different participants was derived using a standard 20×20 Latin Square to minimize the learning effect.

The participants were subjected to a consent agreement, and demographics form before the experiment. At the end of the experiment, their experience in solving DCG captchas was recorded using a survey form. The survey contains the 10 System Usable Scale (SUS) standard questions [7], each with 5 possible responses (5-point Likert scale, where 1 represents strongly disagreement and 5 represents strongly agreement).

Table 1

Demographics of participants in the usability, relay and hybrid attacks studies

	Usability			**Stream Relay Attack**			**Hybrid Attack**
	Lab	MTurk	Mobile	Outside US		US	
Game Size	360x130			360x130	180x65	360x130	360x130
Number of participants	40	40	20	40	20	20	40
Gender (%)							
Male	50	67.5	80	67.5	80	80	80
Female	50	32.5	20	32.5	20	20	20
Age (%)							
<18	0	2.5	0	2.5	0	0	0
18 - 24	80	40	20	30	45	35	25
25 - 35	20	42.5	75	52.5	35	50	42
35 - 50	0	10	5	12.5	20	10	28
>50	0	5	0	2.5	0	5	5
Education (%)							
High school	45	10	25	0	0	55	18
Bachelor	27.5	60	35	57.5	75	40	67
Masters	22.5	27.5	30	42.5	25	5	10
Ph.D.	5	2.5	10	0	0	0	5

For our **second user study**, we recruited 40 MTurk workers. Each was asked to do similar tasks as of the first study. However, they had to play only four games (40 FPS, 6 objects) variant of each game instance. The third column of Table 1 shows the demographics of the 40 participants of our study. The second study was performed to verify that our results are not limited only to young participants and the game repetition did not impact the DCG usability.

The **third usability study** aims to assess the usability of DCG on mobile devices. For this study, we recruited 20 participants who were primarily students in our university (For demographics, see the forth column of Table 1). The participants were asked to perform similar task as the MTurk worker but using Mobile device.

Via our study, our goal was to assess the following aspects of the DCG captchas:

1. *Efficiency*: time taken to complete each game.
2. *Robustness*: likelihood of not completing the game, and of incorrect drag and drop attempts.
3. *User Experience*: participants' SUS ratings of the games.

3.2. Study Results

In this subsection, we report the study results of the three usability studies we conducted.

3.2.1. Lab-based Usability Study

Completion Time: Table 2 shows the completion time per game type. Clearly, all games turned out to be quite fast, lasting for less than 10s on an average. Users took longest to solve the Animals game with an average time of 9.10s, whereas the other games took almost half of this time. This might have been due to increased semantic load on the users in the Animals game to identify three target objects and then match them with the corresponding answer objects. Moreover, we noticed a decrease in the solving time when the target objects were decreased to 2 (i.e., in the Shapes game), and this time was comparable to games which had 1 target object (Ships and Parking). A Friedman's test showed significant difference[2] in the mean timings of all 4 types of games ($\chi^2(3, N = 200) = 268.83, p < 0.001$). Analyzing further using Wilcoxon signed ranks test with Bonferroni correction, we found significant difference between the mean times of all the games pairs, ($p < 0.001$) for all the pairs except for Ships and Parking ($p = 0.01$).

Table 2
Drag error rates and completion time *Lab Usability Study* (Overall error rate=0)

Game Type	**Completion Time(s)**	**Drag Error Rate**
	mean (std)	*mean*
Ships	4.51 (1.00)	0.04
Animals	9.10 (0.96)	0.05
Parking	4.37 (0.90)	0.09
Shapes	5.26 (0.59)	0.03

Error Rates: An important result is that all the tested games yielded 100% accuracy (*overall error rate of 0%*). In other words, none of the participant failed to complete any of the games within the time out. This suggests our DCG captchas instances are quite robust to human errors.

Next, we calculated the likelihood of incorrect drag and drop attempts (*drag error rate*). For example, in the Animals game, an incorrect attempt would be to feed the monkey with a flower instead of a banana. We define the drag error rate as the number of incorrect objects dragged to the target area *divided by* the total number of objects dragged and dropped. The results are depicted in Table 2. We observe that the Shape game yields the smallest average per click error rate of 3%. This suggests that the visual matching task (as in the Shapes game) is less error prone compared to the semantic matching task (as in the other games). The game challenge which seemed most difficult for participants was the Parking game. Since objects in this game are relatively small, participants may have had some difficulty to identify them.

Effect of Object Speed and Number: Table 3 shows the performance of the game captchas in terms of drag error rates and completion time as per different object speeds. We can see that the maximum number of per drag errors was committed at 10 FPS speed. Looking at the average timings, we find that it took longest to complete the games when the objects move at the fastest speed of 40 FPS, while 20 FPS yielded the fastest completion time followed by 10 FPS. A Friedman's test revealed statistical difference among the mean completion time corresponding to the three speeds ($\chi^2(2, N = 160) = 10.36, p = 0.006$). Further analyzing using Wilcoxon signed ranks test with Bonferroni correction, we found significant difference between the mean timing corresponding to the pair of speeds only: 10 FPS and 20 FPS ($p < 0.001$).

Another aspect of the usability analysis included testing the effect of increase in the number of objects (including noisy answer objects) on the overall game performance. Table 4 summarizes the drag error rates and completion time against different number of objects. Here, we can see a clear pattern of increase, albeit very minor, in average completion time and average error rate with increase in the number of objects. This is intuitive because increasing the number of objects increases the cognitive load on the users which may slow down the gameplay and introduce chances of errors. A Friedman's test revealed statistical difference among the mean completion time corresponding to the three number of objects ($\chi^2(2, N = 160) = 37.59, p < 0.001$). Further analyzing using Wilcoxon signed ranks test

[2] All statistical results reported in this paper are at the 95% confidence level.

Table 3

Drag error rates and completion time per object speeds (Overall error rate=0)

Object Speed	**Completion Time (s)**	**Drag Error Rate**
	mean (std)	*mean*
10 FPS	5.74 (2.11)	0.06
20 FPS	4.90 (2.22)	0.05
40 FPS	6.53 (2.87)	0.04

with Bonferroni correction, we found significant difference between the mean timing corresponding to the pair of number of objects: 4 and 5 objects ($p < 0.001$) and 4 and 6 objects ($p < 0.001$).

Table 4

Drag error rates and completion time per # of objects (Overall error rate=0)

# of Objects	**Completion Time (s)**	**Drag Error Rate**
	mean (std)	*mean*
6	6.58 (1.69)	0.06
5	5.30 (2.28)	0.05
4	4.90 (2.22)	0.04

3.2.2. MTurk Usability Study

Table 5

Drag error rates, game error rate and completion time *MTurk Usability Study*

Game Type	**Completion Time (s)**	**Error Rate**	**Drag Error Rate**
	mean (std)	*mean*	*mean*
Ships	10.13 (6.81)	0.03	0.32
Animal	14.97 (8.43)	0.03	0.26
Parking	8.53 (7.01)	0.00	0.54
Shapes	9.42 (6.06)	0.08	0.16

Table 5 summarized the results of MTurk usability study. The second column of Table 5 shows that the average completion time is almost doubled comparing to the lab-based usability study, however, the completion time still less than 15 seconds on average for all the games. A Friedman's test showed significant difference in the mean timings of all 4 types of games ($\chi^2(3, N = 37) = 27.26, p < 0.001$). Analyzing further using Wilcoxon signed ranks test with Bonferroni correction, we found significant difference between the mean times of following pairs: Animals and Parking ($p < 0.001$), Animals and Ships ($p < 0.001$), Animals and Shapes ($p = 0.002$), Parking and Shapes ($p < 0.01$), and Parking and Ships ($p = 0.02$).

The second column of Table 5 shows that the average overall error rate increases from 0% to 3.5% and the average drag error rate increases from 4% to 32%. This is due to the diversity of the MTurk workers over the students who were hired for the first study. Moreover, this variant (40 FPS, 6 objects) is considered harder than the tested variants in the first study, our previous results shows that the completion time increases with increasing the speed and number of the objects.

3.2.3. Mobile-based Usability Study

Table 6 summarized the results of the Mobile-based usability study. The overall error rate is zero – all the participants were able to complete the games successfully. The second column of Table 6 shows the time taken by the participants to complete the games. The time to solve the challenges is in between the time taken by the participants in the lab-based study and Mturk study. Similar to the previous two studies, the maximum time was taken to solve the Animal game. The drag error rate was the minimum for the Shape game and the maximum for Parking game, which is in line with the previous two studies.

Table 6
Drag error rates and completion time *Mobile-based Usability Study* (Overall error rate=0)

Game Type	**Completion Time(s)** *mean (std)*	**Drag Error Rate** *mean*
Ships	9.05(3.64)	0.15
Animals	13.31(6.54)	0.12
Parking	7.24(4.99)	0.85
Shapes	6.32(2.37)	0

3.2.4. User Experience of the Three Studies

Now, we analyze the data collected from the participants during the post-study phase. The average SUS score from the first study came out to be 73.88 (standard deviation = 6.94), for the MTurk based study came out to be 73.25 (standard deviation = 15.07), and for the Mobile based study came out to be 80.69 (standard deviation = 16.07). Considering that the average SUS scores for user-friendly industrial software tend to hover in the 60–70 range [27], the usability of our DCG game captcha instances can be rated as high.

Summary of Usability Analysis: Our results suggest that the DCG captcha representatives tested in this work offer very good usability, resulting in good user ratings, short completion times (less than 15s) on average, and very low error rates (0 - 3.5% per game completion). When contrasted with many traditional captchas, these timings are comparable but the accuracies are better. The average error rate and solving time for text-based CAPTCHAs have been reported to be over 0.13 and around 9.8 seconds, respectively [8]. We found that increasing the object speed and number is likely to degrade the game performance, but up to 6 objects and 40 FPS yield a good level of usability. DCG also seem to have better usability on mobile devices compared to text-based captchas. For example, the error rate for solving reCAPTCHA on mobile devices has been found to be 0.09 and the solving time of 25.2 seconds on average [31].

4. Automated Attacks

Having validated, via our usability study, that it is quite easy for the human users to play our DCG captcha instances, we next proceeded to determine how difficult these games might be for the computer programs. In this section, we present and evaluate the performance of a fully automated framework that can solve DCG captcha challenges based on image processing techniques and principles of unsupervised learning.

4.1. Our Automated Attack and Results

Our attack framework involves the following phases:

1. Learning the background image of the challenge and identifying the foreground moving objects.
2. Identifying the target area. For example, the area that contains all the animals in the Animals game.
3. Identifying and learning the correct answer objects. For example, the ships in the Ships game.
4. Building a dictionary of answer objects and corresponding targets, the background image, the target area and their visual features, and later using this knowledge base to attack the new challenges of the same game.
5. Continuously learning from new challenges containing previously unseen objects.

Next, we elaborate upon our design and matlab-based implementation per each attack phase as well as our experimental results. We note that, on a web forum [1], the author claims to have developed an attack against "are you a human" captcha. However, *unlike our generalized framework*, this method is perfected for *only one simple game* that has one target object and a fixed set of answer objects. It is not known whether or how easily this method can be adapted to handle different games, games with multiple instances that carry different sets of answer objects, and those with multiple target objects. Since only one game is cracked, one needs to keep refreshing the game page until that specific game appears. Since no technical details are provided in [1], we can only doubt if any background learning or object extraction is implemented by observing the short time it takes to finish the attack.

(1) Background & Foreground Object Extraction: To extract the static background of a DCG challenge, the intuitive way is to superimpose some sampling frames that cross a valid period (e.g., 40 frames captured at a fixed time interval (0.2s)), then select the most frequent color value from each pixel as the background color for that pixel. This is based on the assumption that the background image is static and the foreground objects are constantly moving, such that the true background color almost always appears as the most frequent color observed for a pixel. By subtracting the background image from a video frame, the foreground moving objects become readily extractable. To further reduce the computational cost, a 6-bit color code[3], rather than a 3-byte representation of a color value, is used to code the video frame, the learned background image, and the learned foreground objects.

However, one drawback of this preliminary method is that if the moving speed of the foreground objects is too slow, especially when some foreground objects hover over a small area, the dominant color values of most pixels in that area will be contributed by the foreground objects instead of by the background. A shadow of foreground objects may appear as pseudo patches in the background image as shown in Figure 2(b) for the Shapes game of Figure 2(a), indicated by the dashed rectangle. Using more sampling frames for initial background learning could alleviate this problem, but resulting in a time-consuming learning procedure. Our preliminary experiment indicates that an average 30.9s, generated by running the above learning method 15 times per game challenge, is needed for learning a game background completely.

In our new method, we overcome the conflict between the number of sampling frames and the pseudo patch effect by actively changing the location of one moving object per sampling frame. In the first step, a few frames N_1 (e.g., 10 frames captured in 0.3s interval) are collected to generate the initial background that is used to extract the foreground object (through background subtraction) in the next step. Because the number of sampling frames is very limited, pseudo patches may exist. The second step, called active

[3] http://en.wikipedia.org/wiki/Color_code

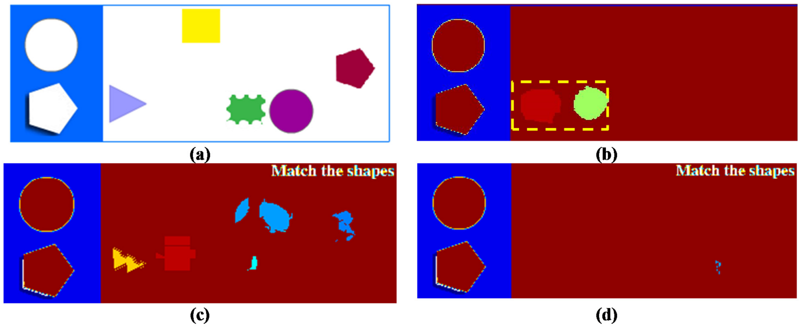


Fig. 2. Detected backgrounds. (a) original frame image; (b) detected background with pseudo patches by using preliminary method; (c) detected background with pseudo patches after performing the 1st step of the proposed method; (d) detected background with a minor patch after performing the 2nd step of the proposed method.

learning, is to actively drag/drop each moving object to a specified destination, which aims to speed up the object movement in order to reduce the pseudo patch effect. Then, N_2 ($N_2 > N_1$) sampling frames are collected whenever a moving object is actively dragged. Because of the high efficiency of the moving object detection and the latter mouse operations, enough sampling frames (e.g., N_2 = (30, 50)) without/with minor hovering effect could be collected within a short period. The new background is detected again based on the dominant color of the collected frames. Figure 2(c, d) show the detected background with non-trivial pseudo patches and with a minor patch, resp. Minor patches could affect the detection of a complete object, but since the affected area is minor, partial matching could still be used in the identification of the answer object.

After removing the extracted background from 5-8 equally distant frames from the collected frames, the objects in each of the selected frame are extracted. The objects below a certain size threshold were discarded as noise. The frame with the maximum number of objects was then selected to extract various objects. Using multiple frames for object extraction also helped us discard the frames in which the objects overlapped each other and were hence detected as a single object instead of distinct individual objects.

As the final step as part of this phase, the visual features, coded as color code histograms (a visual feature commonly used to describe the color distribution in an image), of the foreground objects and the background image, are stored in the database, together with some other meta-data such as the object size and dimensions.

(2) Target Area Detection: Identifying the target area requires analysis of the background extracted in the previous phase. For this purpose, we implemented *Minimum Bounding Rectangle (MBR)* [13] method. The MBR method is based on the observation that the activity/moving area of foreground objects has no or very little overlap with the target area. Therefore, by detecting and removing the foreground moving area from the background image, a reasonable estimate of the target area can be obtained. As the first step of this approach, the selected 5-8 frames and their foreground object masks from the previous phase are used to identify the foreground moving area mask. More specifically, the foreground mask is generated by identifying those pixels that have a different color code value than that of the corresponding pixels in the background image. Then, an MBR is generated that bounds the area where the foreground objects are detected in the current frame (Figure 3). The final estimate of the foreground moving area, denoted as MBR_{final}, is the superimposition of all the MBRs extracted from the sample frames, also represented as a minimum bounding rectangle (see Figure 3(c)).

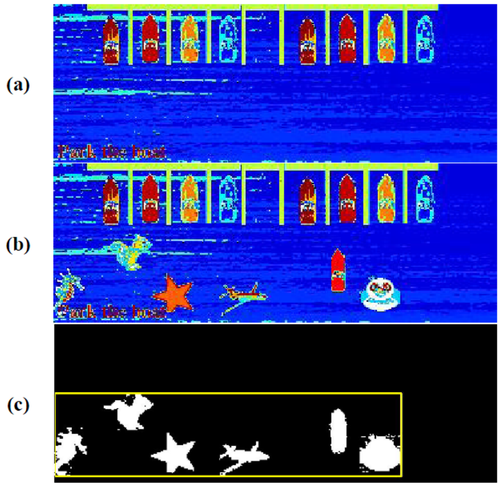


Fig. 3. Target Detection. (a) The detected background for the Parking challenge; (b) One sample frame represented in color code; (c) Detected foreground objects from (b) and their MBR.

After the removal of the entire area bounded by MBR_{final} from the background image, the remaining background is divided into eight sub-areas. The largest sub-area is identified as the target area. It is worth noting that the computational cost of this method is very low ($O(MN)$, where N is the number of pixels in a game scene, M is the number of sample frames, and $M \ll N$) since the foreground object masks are readily available as part of the output from the previous phase. In other words, the most time consuming part is the extraction of foreground objects $O(MN^2)$ from sample frames, which has been covered in the previous phase [4].

(3) Answer Object & Target Location Detection: Once the target area is identified, the next step is to identify the correct answer objects and their respective matching sub-target areas. Since a game can not have too many sub-target areas (otherwise, usability will be compromised), we divide the entire probable target area into 9 equal-sized blocks, each represented by its area centroid, drag each foreground object to each of the 9 centroids, and stop and record the knowledge learned whenever there is a "match." A match occurs when an answer object is dragged to its corresponding sub-target area (e.g., a "bone" dragged onto a "dog"). This is detected by monitoring the change of the area summation of all the foreground objects, since once an answer object is dragged to its correct target location, it will stay in the target area and therefore result in a reduction of the foreground area. In our experiments, this method has proven 100% effective when applied to all four games. As for efficiency, while the worst case upper bound is $O(N)$, where N is the total number of foreground objects, in practice, much less number of drags are required. Our experimental results show that, with 5 foreground objects for each game and 15 training runs for each game, the average number of drags needed for a game is 9. In case the server imposes a strict limit on drag/drop attempts, this process can be repeated over multiple runs.

(4) Knowledge Database Building and Attacking: The background, target area, and learned answer objects as well as their corresponding sub-target areas together constitute the knowledge database for a

[4] We also implemented alternative methods such as Edge-based and exclusion methods, However, MBR method is more robust.

game. After learning about sufficient number of games, whenever a new game challenge is presented, the knowledge base is checked for the challenge. The target area of the currently presented challenge is matched with the target areas present in the database to identify the challenge. If a match is found, the extraction of objects from the foreground follows. The visual features such as the color code histogram of the currently extracted objects are matched with that of the answer objects in the database for that challenge. The extracted objects identified as correct answer objects are then dragged to their corresponding sub-target areas. To measure the performance of our approach, we ran this attacking module 100 times for each game instance, and the average successful attacking time is 6.9s with the number of foreground objects ranging from 4 to 6. The maximum successful attacking time is 9.3s, observed for an instance of the Animal game with 6 foreground objects. These timings are in line with those exhibited by honest users in our usability study, which will make it impossible for the captcha server to time-out our attack.

(5) Continuous Learning: During attacking, if a challenge matches a game in our database but contains previously unseen answer object(s) (e.g., a new ship object in a Ships game instance), the attack will not terminate successfully. Whenever such a situation arises, an answer object learning module that is similar to the aforementioned module is activated, but differs from the latter in that it only needs to drag a potential answer object to each of the previously learned sub-target areas that have matching answer objects in the database. The newly learned answer objects and their corresponding sub-target area centroids are then added to the knowledge base for that game.

4.2. Discussion and Summary

There are two benefits in the background learning. First, the learned background can be used to quickly extract foreground moving objects. Second, the learned background can be used to locate the target area where foreground answer objects need to be dragged to. The proposed active learning is tested on all 36 game challenges (i.e., 3 (speeds), 3 (# of objects), 4 (game prototypes)). N_1 is set to be 10. The shape objects in the Shape Game have larger size than objects in other games, which easily result in pseudo patch effect when 6 moving objects exist in the game window with limited size. Therefore, N_2 is set to be 50 for the Shape Game challenges with 6 objects while 30 frames is used for all the other game challenges. In total, a complete background can be extracted in average 9.04s that is about three times faster than the preliminary method mentioned earlier (i.e., 30.9s).

The adoption of a large image database for each answer object could pose a challenge to our approach since it allows for the creation of many different foreground answer object configurations for the same game. In the worst case, a challenge may contain none of the previously learned answer objects for that particular game. Continuous learning will be activated in such cases and can also be used as a way for auto attacking in the run time. Such cases fall into the category of "known foreground answer objects and known target objects," and the success rate can be estimated using the number of foreground objects (o), number of answer objects (t), and number of drag/drop attempts allowed for each object (a). For example, if $o = 5$, $t = 3$ and $a = 2$, the success rate is approximately $\frac{2^3}{C(5,3)3!} = 13\%$. Though as low as it seems, the rate itself is not affected by the image database size.

During attacking, there is a time lapse between selecting a foreground object and verifying whether it is an answer object. Both feature extraction and database lookup (through feature matching) take time. In our implementation, we chose to click and hold a selected object until a match with an answer object in the database is registered. In doing so, we guarantee that an answer object, once verified, can be readily dragged/dropped, thus to avoid dealing with the issue of constantly moving objects. However, this approach may fail if a constraint is added by the captcha implementation that limits the amount of time one can hold an object from moving. A less invasive attacking method would be to utilize parallel processing, in which one thread is created to perform feature extraction and comparison, and another thread is used to track and predict the movement of the object currently under verification.

Summary of Automated Attack Analysis: Our attack represents a novel approach to breaking a representative DCG captcha category. Attacking captcha challenges, for which the knowledge already exists in the dictionary, is 100% accurate and has solving times in line with that of human users. However, building the dictionary itself is a relatively slow process. Although this process can be sped-up as we discussed, it may still pose a challenge as the automated attack may need to repeatedly scan the different captcha challenges from the server to continuously build an up-to-date dictionary. The defense strategies for the DCG captcha designers may thus include: (1) incorporating a large game database as well as large object image databases for each game; and (2) setting a lower game time-out (such as 20-30s) within which human users can finish the games but background learning does not fully complete. Since our attack relies on the assumption that the background is static, another viable defense would be to incorporate a dynamically changing background (although this may significantly hurt usability).

5. Relay Attacks

Human-solver relay attacks are a significant problem facing the captcha community, and most, if not all, existing captchas are completely vulnerable to these attacks routinely executed in the wild [30]. In this section, we assess DCG captchas w.r.t. to relay attacks.

5.1. Difficulty of Relaying DCG captchas

The attacker's sole motivation behind a captcha relay attack is to completely avoid the computational overhead and complexity involved in breaking the captcha via automated attacks. A pure form of a relay attack, as the name suggests, only requires the attacker to relay the captcha challenge and its response back and forth between the server and a human-solver. For example, relaying a textual captcha simply requires the bot to (asynchronously) send the image containing the captcha challenge to a human-solver and forward the corresponding response from the solver back to the server.

In contrast, DCG captchas offer some level of resistance to relay attacks, as we argue in the rest of this section. There appears to be a few mechanisms that can potentially subject DCG captchas to a relay attack. First, if the server sends the game code to the client (bot), the bot may simply ship the code off to the human-solver, who can complete the game as an honest user would. However, in the DCG captcha security model (Section 2.1), the game code is obfuscated and can be enforced to be executable only in a specific domain/host authorized by the server using existing tools [3], which will make this attack difficult, if not impossible.

The second possibility, called *Static Relay* attack, is very simple and in line with a traditional captcha attack (and thus represents a viable and economical relay attack). Here, the bot asynchronously relays a *static snapshot* of the game to a human-solver and uses the responses (locations of answer objects and that of the target objects) from the solver to break the captcha (i.e., drag/drop the object locations provided by the solver to the target object locations provided by the solver). However, it is expected to have poor success rates. The intuitive reason behind this is a natural *loss of synchronization* between the bot and the solver, due to the dynamic nature of DCG captchas (moving objects). In other words, by the time the solver provides the locations of target object and the answer objects within a challenge image (let us call this the n^{th} frame), the objects themselves would have moved in the subsequent, k^{th}, frame ($k > n$), making the prior responses from the solver of no use for the bot corresponding to the k^{th} frame. Recall that the objects move in random directions and therefore it would not be possible for the bot to predict the location of an object in the k^{th} frame given the locations of that object in the n^{th} frame ($n < k$). Such a loss of synchronization will occur due to: (1) communication delay between the bot and human-solver's machine, and (2) the manual delay introduced by the solver him/herself in responding to the received challenge.

The third possibility, called *Stream Relay*, is for the bot to employ a streaming approach, in which the bot can synchronously relay the incoming game stream from the server over to the solver, and then relay back the corresponding clicks made by the solver to the server. Although the *Stream Relay* attack might work and its possibility can not be completely ruled out, it presents one main obstacle for the attacker. Streaming a large number of game frames over a (usually) slow connection between the bot (e.g., based in the US) and the solver's machine (e.g., based in China) may degrade the game performance (similar to video streaming over slow connections), reducing solving accuracy and increasing response time. Such differences from an average honest user gameplay session may further be used to detect the attack.

In the rest of this section, we report our *Stream Relay* attack study and show the ability of detecting such relay attack using our proposed machine learning detection method.

5.2. Stream Relay Attack

Under Stream Relay, the attacker obtains the DCG captcha challenge from the server, just like a legitimate user. The attacker runs a streaming server (such as a VNC server), and the human-solver connects to the attacker machine through a streaming client (such as a VNC client). This streaming software is responsible for delivering the DCG captcha frames to the human-solver and sending the human-solver's mouse interactions, such as drag/drop, mouse clicks and positions, to the attacker. The attacker then simply forwards the log of this interaction between the human-solver and the game to the server. Finally, the server would run the detection algorithm on this log, and responds back by rejecting (or accepting) the attacker. Due to network latency, our hypothesis is that the human-solver may suffer from degradation of the game quality at his/her end. This degradation would decrease the game performance of DCG captcha. More importantly, it would make the solver interaction with the game distinguishable from the interaction between the legitimate user and the game (as in the normal setting), and thereby make it possible for the server to detect the relay attack.

5.2.1. Study Design, Goal and Process

MTurk workers were hired for the Stream Relay attack study. The MTurk workers (serving the role of human-solvers) were asked to connect to a computer residing at our university and connected to our university wireless network through a VNC java applet (serving the role of the attacker's machine). The workers were then asked to fill demographics form, play four games (40 FPS, 6 objects) variant of each game instance (ordered based on 4×4 Latin Square), and fill a survey form about their experience.

We used three different experiments to test various relay attack scenarios, the demographics of the participants are shown in the columns 5 to 7 of Table 1, as described below:

1. *High-Latency Relay*: The first scenario involved collecting data from participants residing outside the US. Since in a typical relay attack, the human-solvers are normally hired from sweatshops in remote countries (e.g., India or China) by an attacker residing in the US, this setting reflects a real-life relay attack scenario. We collected data from 40 participants as part of this scenario.

Table 7

Completion times, and error rates in *Stream Relay* attack scenarios

	High-Latency Relay			***Small Game Relay***			***Low-Latency Relay***		
Game Type	**Successful Time (s)** *mean (std)*	**Error Rate** *mean*	**Drag Error Rate** *mean*	**Successful Time (s)** *mean (std)*	**Error Rate** *mean*	**Drag Error Rate** *mean*	**Successful Time (s)** *mean (std)*	**Error Rate** *mean*	**Drag Error Rate** *mean*
Ships	29.98 (13.00)	0.15	0.45	22.82 (8.17)	0.40	0.27	16.28 (13.39)	0.05	0.25
Animal	34.04 (13.52)	0.33	0.43	37.11 (8.21)	0.40	0.29	17.54 (11.71)	0.20	0.35
Parking	25.61 (15.79)	0.38	0.75	20.43 (13.20)	0.25	0.57	14.13 (13.67)	0.10	0.46
Shape	21.53 (12.79)	0.23	0.43	26.86 (10.86)	0.10	0.51	15.25 (14.76)	0.05	0.21

2. *Small Game Relay*: The second attack scenario involved testing a case when an attacker tries to minimize communication between the attacker and the solvers by reducing the game size. This was achieved by presenting games with 1/4 of the normal size to the subjects, i.e., a game with size 180x65. To evaluate this scenario, we collected data from 20 participants residing outside US.
3. *Low-Latency Relay*: In the last scenario, we tested a setting in which the attacker launches the attack from a machine that is in close proximity to the solvers (e.g., both the attacker and solvers are located in the US). To evaluate this scenario, we collected data from 20 participants located within US.

5.2.2. Study Results

In this subsection, we will report the study results and compare them to the MTurk usability study.

Completion Time and Error Rates: The results for the first, High-Latency Relay, scenario are shown in first column of Table 7. The games played as part of this scenario took significantly longer than that performed with usability study. On average, we found that completing DCG captchas with High-Latency Relay took approximately 61% longer than that for usability study. Furthermore, upon comparing the mean time taken to complete the games (Successful Time) between the MTurk usability and High-Latency Relay using Mann-Whitney U test with Bonferroni correction, we found a statistically significant difference, with $p < 0.001$, for each of the four games. The error rates were also significantly higher than those exhibited in the MTurk usability study, on an average of 84%. The drag error rate was 40% higher for High-Latency Relay attack compared to usability study. The longer game completion time and higher error rates might be attributed to high network latency between the attacker's machine and the human-solvers' machines.

To overcome the issues presented by network latency for the participants outside the US, in the second scenario (Small Game Relay), we reduced the game size by 1/4, to 180x65 pixels. However, the results, as shown in the second column of Table 7, were still comparable to that of stream relay attack with normal game size, with longer gameplay time and higher error rates compared to the MTurk usability study. The successful game completion time was approximately 60% longer, while the error rate was 76% higher on average than that for usability study. The drag error rate was 16% higher than that for usability study. Analyzing the mean time using Mann-Whitney U test with Bonferroni correction, we found statistically significant difference between the mean time between all pairs of games from usability and Small Game Relay with $p < 0.001$. Although reduced size may have resulted faster game transmission, smaller game size may have made it difficult for the users to play the game.

Our last stream relay experiment, Low-Latency Relay, tested the Stream Relay attack performance when the attacker and the solver reside relatively nearby (both within the US). The results of this experiment are depicted in the third column of Table 7. The results show huge improvement over the previous two scenarios. The time taken to complete the game is on average about 40% lower compared to the time taken by participates in High-Latency and Small Game Relay scenario. Analyzing the data using Mann-Whitney U test with Bonferroni correction, we found statistically significant difference between the mean time of the Ships game and its correspondent in the usability study: $p = 0.048$. However, we did not find statistically significant difference between the mean times of the rest of the games and their correspondents in the usability study. It appears that relatively lower latencies between the attacker's machine and solvers' machines in this scenario improved the game performance, but it was still at a lower level compared to that exhibited in the usability study. The error rates and drag error rate were 41% and 1% higher for Low-Latency Relay attack compared to usability study.

User Experience: The mean of the SUS for the first, second and third relay attack scenarios came to be 59 (standard deviation = 12.83), 57 (standard deviation = 14.97) and 65.11 (standard deviation = 18.42), respectively, which is consistently lower than the mean SUS score obtained from the usability study. Comparing SUS score between the MTurk usability study and each of the three relay attack scenarios, using Mann-Whitney U test with Bonferroni correction, we found statistically significant difference between usability study and High-Latency Relay ($p < 0.0001$) and between usability study and Small Game Relay ($p = 0.004$). Low-Latency Relay did not turn out to be significantly different from the usability study statistically in terms of user experience.

5.2.3. Stream Relay Attack Detection

In the previous subsection, we have demonstrated that the DCG captcha game performance (completion timings and error rates) in the usability study setting and the game performance in each of the Stream Relay attack scenarios differs in the average case. In this section, we set out to investigate whether it is possible for the captcha service, based on the different gameplay features and behavioral data, to identify whether an individual gameplay event (captcha solving instance) conducted by a legitimate user or to human-solver in the Stream Relay attack.

A continuous key-press track may not correspond to a drag track when the mouse misses to grab a moving object. Such a key-press track is called an invalid mouse drag. When an invalid mouse drag occurs to a legitimate user, he/she can usually realize it immediately and take appropriate corrective actions. Consequently, an invalid mouse drag track will end relatively quickly, resulting in relatively few timestamps on the track. In contrast, when the same situation happens during Stream Relay attack, the remote human-solver may be slow in response due to the network communication delay, which may be reflected as either a longer invalid mouse drag track, or a slow-motion mouse movement that generates many timestamps, or both.

There are 7 features extracted from the users' gameplay data, used as input to train a classifier to differentiate legitimate users from relay attackers and tested with different machine learning methods.

1. *PlayDuration*: overall gameplay time (in seconds).
2. *Successful drag rate*: the ratio of the number of successful drags/drops to the total number of drag/drops.
3. *Number of attempts*: the number of times the mouse status changes from "up" to "down".
4. *Average dragging time*: the sum of time duration of drags divided by the number of drags.
5. *The maximum duration among all invalid mouse drags in a gameplay instance.*
6. *Number of timestamps in the invalid mouse drag with the longest duration.*
7. *The product of Features 5 and 6.*

Table 8

Results of using the optimal feature subset for each game in classification of *legitimate user* and *(High- and Low-Latency and Small Game)* relay attacker

Game Name	**Feature Subset**	**Method**	**Average Accuracy**	**Average Precision**	**Average Recall**
Ships	2,3,4,5,7	C-SVC rad	0.83	0.73	**0.87**
	2,3,4,5,6,7	nu-SVC rad	0.83	0.73	**0.87**
Animal	7	C-SVC rad	0.85	0.75	**0.97**
	7	nu-SVC rad	0.85	0.75	**0.97**
Parking	4,5	C-SVC rad	0.74	0.65	**0.76**
Shape	2,3,4,5	KNN min	0.78	0.66	**0.75**
	2,3,4,5,7	KNN min	0.78	0.66	**0.75**

Both Support Vector Machine (SVM) [10] and K-Nearest Neighbors (KNN) [16] are tested on 127 (2^7-1) feature subsets with 6 (2 SVM types, namely C-SVC, Support Vector Classification, and nu-SVC, with 3 different kernel functions, linear, polynomial, and radial basis functions) and 2 (i.e., Euclidean distance or Minkowski metric) parameter configurations in SVM and KNN, respectively. In total, 1016 (127 × 8) different test cases were tested for each game prototype.

In the classification task, the positive class corresponds to a legitimate user and the negative class corresponds to human-solver relay attacker as denoted below:

- *True Positive (TP):* legitimate user correctly classified as legitimate user.
- *True Negative (TN):* relay attacker correctly classified as relay attacker.
- *False Positive (FP):* a relay attacker misclassified as legitimate user.
- *False Negative (FN):* a legitimate user misclassified as a relay attacker.

Three different measures are used to evaluate the classifier's performance, namely precision, recall, and accuracy, as defined in Equations 1, 2 and 3. Of these, recall is the most important because low recall leads to a high rejection rate of legitimate users, causing user frustrations and compromising usability. The desired classification result should demonstrate a sufficiently high recall and a reasonably high precision.

$$Precision = TP/(TP + FP) \tag{1}$$

$$Recall = TP/(TP + FN) \tag{2}$$

$$Accuracy = (TP + TN)/(TP + FP + TN + FN) \tag{3}$$

To measure the overall classification accuracy and recall of our model, we built a classifier for each of the four game types using all the collected data from usability study and the three relay attack scenarios. The average measurement values are shown in Table 8, suggesting that the best average accuracy and average recall are achieved for the Animal and Ships games, followed by the Shape game and then the Parking game. This suggests that increasing the number of the required drags/drops improves the classification performance.

6. Hybrid Attack

Relay attacks as shown in section 5 are easily detectable. Automated attack is effective, however, it has some weaknesses, such as:

1. The auto-attack framework cannot detect the target area that overlap the activity area.
2. Dragging and dropping a moving object always follows a straight line-path, which doesn't work for more complex paths.
3. The learning phase requires too many drags/drops which may be detectable by the server.

In this section, we propose two models of hybrid attacks for attacking DCG captcha that combine the strengths of both of the automated and relay attack and overcome their weakness. A key component of both models in our framework is robust object tracking that preserves the synchronization between the game and the bot, which we discuss next.

6.1. Real-Time Object Tracking

Real-time tracking on moving objects of a game challenge plays an important role in keeping synchronization between the game and the bot. The tracking efficiency affects whether a timely completion of a game challenge by the bot could be done like human, which becomes one of the key factors that determines the success rate. In this section, we introduce a simple and efficient color code histogram based tracker.

Based on the characteristics of DCG prototypes introduced in Section 2.2, namely, (1) the background is static, and (2) the moving objects have unchanging appearance, we propose a simple and efficient tracking algorithm, named color code histogram based tracker (denoted as CCH) that generates the foreground mask by utilizing the detected background, and associates the track to the same object based on color code features [41]. We used 6-bit color code instead of 8-bit (e.g., gray image) or 24-bit (e.g., RGB/HSV/Lab image), which is an approximate color representation that consists of the top two bits of each RGB channel, in order to reduce the computation cost. Moving objects as well as the game scene are represented as a normalized 64-bin color code histogram. The tracking task is thereby simplified as a histogram matching task in the subsequent frames by using histogram intersection [36] that is known to be robust against scaling and rotation. To alleviate the segment merging problem caused by objects occlusion, CCH will abandon the current foreground mask if fewer segments than the number of tracks can be found. The extent of tolerance to partial occlusion relies on tuning the similarity threshold in histogram intersection.

Such a simple design also exposes two weaknesses: (1) The quality of the foreground mask in each frame completely relies on the quality of the detected background; and (2) Tracking may become inaccurate when multiple objects have similar color histograms. Future research may consider other clues in matching, such as motion, to alleviate this problem.

6.2. Auto-attack with offline learning

Model I of our proposed hybrid attack framework (Figure 4), Auto-attack with Offline human Learning (AOffL), attacks a known game with the help of real-time tracking and offline knowledge. In the learning phase of AOffL, the necessary knowledge related to a game scene is learned in advance from a remote human-solver.

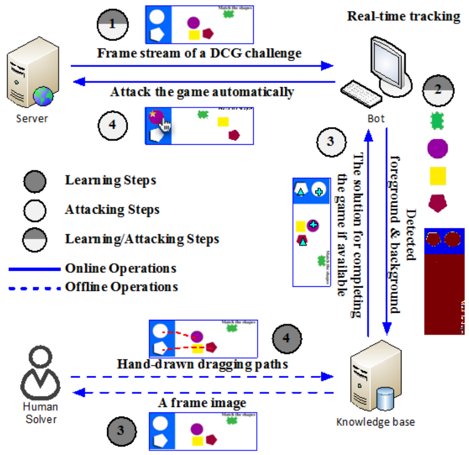

1
Frame stream of a DCG challenge
Real-time tracking
Server
Attack the game automatically
Bot
2
4
3
Learning Steps
Attacking Steps
Learning/Attacking Steps
Online Operations
Offline Operations
The solution for completing the game if available
Detected foreground & background
4
Hand-drawn dragging paths
Human Solver
A frame image
Knowledge base
3

Fig. 4. Hybrid attack model I: Auto-attack with Offline human Learning

First, the bot starts the game and keeps scanning the game frame images. Second, the bot performs initial analysis to detect the game background, moving objects and potential target areas, and keeps tracking the moving objects. Third, similar to the relay attack against a text-based CAPTCHA, only one frame image is sent from the bot to the human-solver. The solver task is draw line(s) from the answer object(s) to its/their corresponding target object(s). These lines provide several clues for attacking the game: (1) The start and the end points of each line label the locations of the answer object and its corresponding target object, respectively; (2) The end portion of each line (e.g., the portion connected to the target object) can also be used as the basic entry path to the target object if straight-line paths are not workable in this game. For example, 40% of each hand-drawn dragging path (starting from the end point) is treated as the basic entry path. Therefore, an answer object can always be dragged to the start point of the entry path first, and follow the entry path to the target finally. If a complex path, is required, a curvature threshold could be defined to identify those critical turning points with curvatures larger than the threshold, which finds the key points that must be passed in turn in a new path. Finally, the above clues together with the initial analysis, i.e., background and foreground detection, will be recorded in the knowledge base. Answer objects as well as the background are represented in color code histograms. A continuous learning from the game server is required to build an up-to-date dictionary.

The attacking phase consists of the following steps (as shown in Figure 4). The initial analysis is performed (Step 1) followed by submitting a query to the knowledge base (Step 2). If a match is found, the bot will drag an answer object from its current location provided by the real-time tracking to its corresponding target object (Step 3). The drag/drop attempt iterates until completing the game (Step 4). If a match is not found, which indicates that the game or the answer objects are completely new, the framework will learn the game as the dictionary based auto-attack framework mentioned in Section 4. Moreover, the attacking phase is converted into offline learning just in case that brute-force based learning cannot work out the puzzle.

6.3. Auto-attack with online learning

Model II of our hybrid attack framework (Figure 5), Auto-attack with Online human Learning (AOnL), attacks any game, seen or unseen, with the help of real-time tracking and online knowledge. Compared

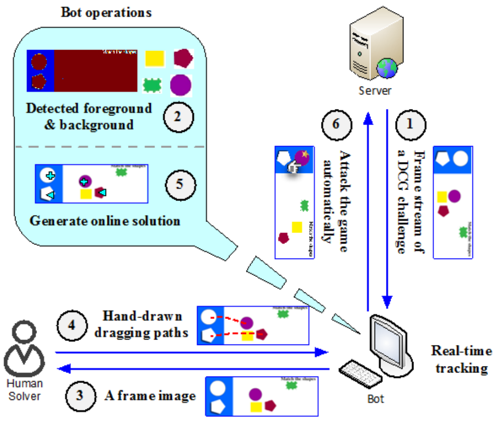

Bot operations
Detected foreground & background
2
Generate online solution
5
Server
6
Attack the game automatically
1
Frame stream of a DCG challenge
4
Hand-drawn dragging paths
Real-time tracking
Human Solver
3
A frame image
Bot

Fig. 5. Hybrid attack model II: Auto-attack with Online human Learning

with AOffL, a human-solver must be available when the game starts, similar to what is required in relay attacks. Moreover, there is no knowledge base for future attacking as the remote solver provides the required knowledge in real-time.

As indicated in Figure 5, when attacking a game challenge, the bot keeps receiving frames from the server (Step 1), and performs initial analysis (i.e., detect background and foreground) on the game without drag/drop attempts (Step 2). Meanwhile, it sends one frame image to the solver once the game starts (Step 3). The solver performs the same operation as the learning phase in AOffL (Step 4). Once the solver submits his/her responses, the bot can learn the answer objects and the dragging paths for this particular challenge based on the initial analysis and the solver's response (Step 5), and complete the game automatically with the help of real-time tracking (Step 6). One concern in AOnL is that the success rate for completing a game relies heavily on correctness and efficiency of the solver's response (the same concern underlies the relay attack on text-based and DCG captchas).

6.4. Hybrid Attack Usability Study

6.4.1. Study Design, Goal and Process

In order to evaluate the performance for the users' drawing operation and thereby the performance of our hybrid attack, we conducted a user study in which we recruited 40 MTurk workers. We asked the participants to fill a demographics form, then provide them with 4 static images that represent snapshot to the four DCG explained in Section 2.2 and asked them to draw lines from the answer object(s) to its/their corresponding target(s), the order of representing the images to the participants followed the standard 4x4 Latin square. The interface has two buttons "undo", which clears the previously drawn lines, and "done", which the participant should press after drawing all the lines from the answer objects to the targets. Finally, the participants were asked to fill a survey form consists of the 10 standard System Usable Scale (SUS) questions. The demographics of the participants is presented in the last column of Table 1.

6.4.2. Study Results

In this subsection, we will report the study results and compare them to the MTurk usability study.

Completion Time and Error Rates: The average time taken by the participants and the error rates are shown in Table 9. The shortest average completion time and the minimum error rate were for the Shapes game. The average time taken by the participants to complete the challenges is higher than its correspondence in the usability study, however it is still considerably short (around 13s on average). Comparing the completion time of all the games with their respective in MTurk based usability study using Mann-Whitney U test with Bonferroni correction, we found significant difference between each of the games and its correspondence in the usability study Animal ($p = 0.0035$), Parking ($p < 0.001$), Shape ($p = 0.0163$) and Ships ($p = 0.0215$). Some of the participants drew extra lines, which are represented in the drag error rate in Table 9. The drag error rate on average is less than the drag error rate committed by the participants in the usability study, this can be due that the participants are allowed to delete the previously drawn lines by pressing the "undo" button in case they committed errors. However, the overall error rate is higher than its correspondence in the usability study as there is no instant feedback if the drawn lines are correct or incorrect and whether the game is completed successfully or not.

User Experience: The SUS score came to be on average 68.00 (standard deviation = 16.17), which is lower than the SUS score for the usability study but better that all the tested variants of relay attacks.

Table 9

Drag error rates, game error rate and completion time *Hybrid attack*

Game Type	**completion Time(s)** *mean (std)*	**Error Rate** *mean*	**Drag Error Rate** *mean*
Ships	11.78 (5.39)	0.28	0.00
Animals	19.65 (10.48)	0.08	0.30
Parking	12.56 (7.93)	0.20	0.31
Shapes	10.57 (4.77)	0.05	0.45

Summary of Hybrid Attack Analysis: The two models of the hybrid attack framework have their respective advantages. AOffL can complete a known game efficiently and effectively, but it requires continuously updating the knowledge base for unseen games or answer objects. The delay issue in the manual learning phase is not a problem in AOffL due to its offline nature. Therefore, AOffL is a significant threat to DCGs that do not have a large database (e.g., manually extended database), but has a low tolerance on completion time. On the other hand, AOnL is insensitive to the database size. That is, it is possible for AOnL to complete a game challenge even if the game has never been seen before, largely attributed to the instant solution provided by the solver. However, response delay from the solver may be a bit of a bottleneck for AOnL (as shown in the hybrid attack usability study, users took more time to complete the drawing task comparing to the playing time in the usability studies). In the current settings, the bot wait till the human-solver draws all the lines and sends them to the attacker. The bot then starts dragging/dropping the answer objects to their corresponding targets according to the response of the human-solver. Therefore, according the collected data in the user study explained above the bot would wait in average 13s before it starts playing the game). Therefore, AOnL could be a significant threat to those DCGs that has a relatively high tolerance on playing time. Using the gameplay features other than the play duration to detect hybrid attacks is a challenging task as the bot is the one who is playing the games and it would be able to mimic the human mouse interactions with the games.

7. Related Prior Work

The term CAPTCHA was first introduced in 2000 [2], describing a test that can differentiate humans from malicious computer programs. CAPTCHAs are deployed by many online services, such as account registration, ticket selling, and search engines, to limit the scale of different types of attacks (e.g., denial-of-service or password dictionary attacks) involving automated bots. Most CAPTCHAs are largely based on visual challenges, such as involving users to identify alphanumeric characters in distorted images, but many other variants have also been proposed [22,39].

A wide variety of CAPTCHAs have been proposed over the last decade or so. The most commonly utilized CAPTCHA involves challenging the users to recognize alphanumeric characters embedded within an image, such as Gimpy, Yahoo, reCapthca [38], Baffle [12], handwritten [32] and PayPal, or within a video such as NuCaptcha and emergent CAPTCHA [39]. CAPTCHAs that challenge the users to recognize or classify objects in images, such as collage CAPTCHA [34], implicit CAPTCHA [5], Asirra CAPTCHA [24], PIX, and ESP-PIX [37], have also been proposed. Some video-based CAPTCHAs, such as content-based tagging of YouTube videos [26], and audio based CAPTCHA, such as Google, eBay, ReCaptcha, Slashdot and Math-function [21] audio CAPTCHAs, have also been introduced.

Unfortunately, existing CAPTCHA technology suffers from several problems. The distortions that are used to hide the underlying content of a puzzle from computers can also severely degrade human usability [8,40]. Challenges based on spoken words suffer from similar issues due to sound distortion [20]. Moreover, CAPTCHAs are not foolproof, and many CAPTCHAs used in real-world have been successfully attacked. The task of solving CAPTCHA has been made easier by commercial solving services that attackers often utilize [30]. These services offer two categories of attacks: *automated attacks* and *relay attacks*. Automated attacks (e.g., [17,23,25]) normally utilize image processing algorithms to solve the CAPTCHA, while relay attacks [30] utilize the human intelligence of third-party, remotely located human-solvers.

Relay attack involves outsourcing the CAPTCHA solving process to human labor, either opportunistically or via sweatshops [30]. An attacker could launch a website that attracts visitors by providing some free service, and then opportunistically engage them in solving third-party CAPTCHAs. Alternatively, an attacker could hire people to solve CAPTCHA and pay them a certain amount of money per successful attack. A relay attack against a text CAPTCHA, for instance, involves the attacker to forward the image that contains the CAPTCHA to a human-solver; the solver then solves the CAPTCHA in real-time, and provides the solution, which the attacker relays back to the server.

Although automated attacks seem to be a natural option to bypass the security offered by CAPTCHAs, developing programs to solve CAPTCHA with human-like accuracy is often very complicated and costly [30]. In contrast, paid solvers are willing to solve as many as 1000 CAPTCHAs for just $1, making relay attack an overall more attractive, effective and economical option [30]. While the traditional CAPTCHA research has focused mainly on developing, or preventing, automated CAPTCHA attacks, attackers in the wild have gone on to break existing CAPTCHA schemes via relay attacks [30].

Most, if not all, existing CAPTCHAs are vulnerable to relay attacks, and do not provide a reliable mechanism to distinguish a remote human-solver from a legitimate user. Subjecting textual CAPTCHAs to relay attacks is very simple – the attacker simply forwards the challenge image to the solver, who provides the response which the attacker simply forwards to the service. Effectively detecting such attacks does not seem feasible. One way to avoid them is to set a timeout for solving the CAPTCHA. However, timing alone is not a robust method for detecting an attack. A comprehensive study on CAPTCHA-solving services presented in [30] concludes that 70% of the CAPTCHA submissions are correct, and are submitted within 30 seconds, which is well within the CAPTCHA timeout set by most websites.

Attacking video CAPTCHA [39] is straightforward as well. The CAPTCHA video file can be forwarded to a human-solver. Alternatively, a new video can be created by taking multiple snapshots of the video CAPTCHA, and sent to the human-solver.

Image-based CAPTCHAs require users to perform an image recognition task, e.g., selecting only the images of cats in a grid of images. This kind of CAPTCHA can also be easily attacked by relaying. The image can be transferred to a solver, and solver can send back the coordinates of the mouse clicks to the attacker. The attacker's bot can then replicate the action performed by the solver.

The fact that most captchas are static and do not require multiple interactions from a user makes attacking them by relaying to a human-solver an easy task. DCG captcha is perhaps the first step towards creating interactive and dynamic captcha capable of defeating relay attacks. A commercial implementation of DCG captcha, called "are you a human," [4] is also available.

8. Conclusions and Future Work

We investigated the security and usability of game-oriented captchas in general and DCG captchas in particular. Our overall findings are mixed. On the positive side, our results suggest that DCG captchas, unlike other known captchas, offer *some* level of resistance to relay attacks. We believe this to be a primary advantage of these captchas, given that other captchas offer no relay attack resistance at all. Furthermore, the studied representative DCG captcha category demonstrated high usability. On the negative side, however, we have also shown this category to be vulnerable to a dictionary-based automated attack and hybrid attacks.

An immediate consequence from our study is that further research on DCG captchas could concentrate on making these captchas better resistant to automated attacks while maintaining a good level of usability. One of the main weaknesses of the DCG captcha instances is that the underlying game background is static, which is utilized by our attack framework to extract the foreground objects from the game frames thereby undermining the captcha security. One direction to remedy this problem is to design DCG CAPTCHAs variations that incorporate different forms of dynamic background. The variations design may involve random noises , objects with dynamically changing color and luminance, object occlusion, and dynamically changing background, and combinations thereof. Such variations would resist our proposed attack, however they may be prune to other more sophisticated attacks as well they may have low level of usability. Further work is needed to investigate the security as well as the usability of such variations.

Further study is also needed to find the best/optimal method for selecting the moving and target objects. For example, an attack can be developed on the challenges that is based on shape matching that can solve the challenges by finding the similarities between the moving and target objects, and then for each of the target objects the moving object with highest similarity could be considered to be its corresponding answer object.

Acknowledgement

We thank: the team of "are you a human" for creating the CAPTCHAs that inspired our work; Wei-Bang Chen for his help in the implementation of the automated-attack framework; John Grimes, John

Sloan and Anthony Skjellum for guiding us regarding the ethical aspects of our work; Sonia Chiasson, Fabian Monrose and Gerardo Reynaga regarding early discussions; and various members of the SPIES group at UAB and PreCog group at IIITD for helpful suggestions throughout the study. The work of Mohamed, Georgescu, Gao and Saxena is partially supported by a grant on "Playful Security" from the National Science Foundation CNS-1255919. Van Oorschot holds the Canada Research Chair in Authentication and Computer Security and acknowledges NSERC for funding the chair and a Discovery Grant.

References

[1] Cracking the areyouahuman captcha. `http://spamtech.co.uk/software/bots/cracking-the-areyouhuman-captcha/`.

[2] L. V. Ahn, M. Blum, N. J. Hopper, and J. Langford. Captcha: using hard ai problems for security. In *EUROCRYPT*, 2003.

[3] Amayeta. Swf encrypt: Encrypt, obfuscate & protect your flash swf actionscript & resources from decompilers. `http://www.amayeta.com/software/swfencrypt/`.

[4] Are You a Human. `https://www.areyouahuman.com/`.

[5] H. S. Baird and J. L. Bentley. Implicit captchas. In *Electronic Imaging*, 2005.

[6] A. Basso and F. Bergadano. Anti-bot strategies based on human interactive proofs. In *Handbook of Information and Communication Security*. 2010.

[7] J. Brooke. SUS: a "quick and dirty" usability scale. In *Usability Evaluation in Industry*. 1996.

[8] E. Bursztein, S. Bethard, C. Fabry, J. Mitchell, and D. Jurafsky. How Good Are Humans at Solving CAPTCHAs? A Large Scale Evaluation. In *IEEE Symposium on Security and Privacy*, 2010.

[9] C. Doctorow. Solving and Creating CAPTCHAs with Free Porn. In *Boing Boing*, Available at: `http://www.boingboing.net/2004/01/27/solving_and_creating.html`, 2004.

[10] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011.

[11] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski. Designing human friendly human interaction proofs (hips). In *ACM CHI*, 2005.

[12] M. Chew and H. S. Baird. Baffletext: A human interactive proof. In *Electronic Imaging*, 2003.

[13] S. ching Chen, M. ling Shyu, C. Zhang, and R. L. Kashyap. Identifying overlapped objects for video indexing and modeling in multimedia database systems. In *Multimedia Database Systems, International Journal on Artificial Intelligence Tools*, 2001.

[14] Dancho Danchev. Inside india's captcha solving economy. Available at: `http://blogs.zdnet.com/security/?p=1835`.

[15] M. Egele, L. Bilge, E. Kirda, and C. Kruegel. Captcha smuggling: hijacking web browsing sessions to create captcha farms. In *ACM Symposium on Applied Computing*, 2010.

[16] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3), 1977.

[17] G. Keizer. Spammers' Bot Cracks Microsoft's CAPTCHA. In *Computer World*, Available at: `http://www.computerworld.com/article/2536901/security0/spammers--bot-cracks-microsoft-s-captcha.html`, 2008.

[18] S. Gao, M. Mohamed, N. Saxena, and C. Zhang. Gaming the game: Defeating a game captcha with efficient and robust hybrid attacks. In *IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2014.

[19] C. Gentry, Z. Ramzan, and S. Stubblebine. Secure distributed human computation. In *ACM conference on Electronic commerce*, EUROCRYPT, 2005.

[20] Graig Sauer and Harry Hochheiser and Jinjuan Feng and Jonathan Lazar. Towards a Universally Usable CAPTCHA. In *SOUPS*, 2008.

[21] J. N. Gross. Captcha using challenges optimized for distinguishing between humans and machines, 2009. US Patent App. 12/484,800.

[22] J. M. G. Hidalgo and G. Alvarez. CAPTCHAs: An Artificial Intelligence Application to Web Security. *Advances in Computers (editor: Marvin V. Zelkowitz)*, 2011.

[23] Jeff Yan and Ahmad Salah El Ahmad. A Low-cost Attack on a Microsoft CAPTCHA. In *ACM CCS*, 2008.

[24] Jeremy Elson and John R. Douceur and Jon Howell and Jared Saul. Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization. In *ACM CCS*, 2007.

[25] K. Kluever. Breaking the PayPal.com CAPTCHA. Available at: `http://www.kloover.com/2008/05/12/breaking-the-paypalcom-captcha`, 2008.

[26] K. A. Kluever and R. Zanibbi. Balancing usability and security in a video captcha. In *Symposium on Usable Privacy and Security*, 2009.

[27] J. Lewis and J. Sauro. The factor structure of the system usability scale. In *Human Computer Interaction International Conference (HCII)*, 2009.

[28] M. Mohamed, S. Gao, N. Saxena, and C. Zhang. Dynamic cognitive game captcha usability and detection of streaming-based farming. In *the Workshop on Usable Security (USEC), co-located with NDSS*, 2014.

[29] M. Mohamed, N. Sachdeva, M. Georgescu, S. Gao, N. Saxena, C. Zhang, P. Kumaraguru, P. C. van Oorschot, and W.-B. Chen. A three-way investigation of a game-captcha: automated attacks, relay attacks and usability. In *ACM symposium on Information, computer and communications security*. ACM, 2014.

[30] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: Captchas-understanding captcha-solving services in an economic context. In *USENIX Security Symposium*, 2010.

[31] G. Reynaga. *THE USABILITY OF CAPTCHAS ON MOBILE DEVICES*. PhD thesis, CARLETON UNIVERSITY Ottawa, 2015.

[32] A. Rusu and V. Govindaraju. Handwritten captcha: Using the difference in the abilities of humans and machines in reading handwritten words. In *Frontiers in Handwriting Recognition*, 2004.

[33] S. Prasad. Google's CAPTCHA Busted in Recent Spammer Tactics. Available at: `http://securitylabs.websense.com/content/Blogs/2919.aspx`, 2008.

[34] M. Shirali-Shahreza and S. Shirali-Shahreza. Collage captcha. In *Signal Processing and Its Applications*, 2007.

[35] D. Stefan and D. Yao. Keystroke-dynamics authentication against synthetic forgeries. In *CollaborateCom*, 2010.

[36] M. J. Swain and D. H. Ballard. Indexing via color histograms. In *Active Perception and Robot Vision*. 1992.

[37] L. Von Ahn and L. Dabbish. Labeling images with a computer game. In *SIGCHI conference on Human factors in computing systems*, 2004.

[38] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895), 2008.

[39] Y. Xu, G. Reynaga, S. Chiasson, J.-M. Frahm, F. Monrose, and P. C. van Oorschot. Security and usability challenges of moving-object captchas: Decoding codewords in motion. In *USENIX Security*, 2012.

[40] Yan, Jeff and El Ahmad, Ahmad Salah. Usability of CAPTCHAs Or usability issues in CAPTCHA design. In *SOUPS*, 2008.

[41] C. Zhang, W.-B. Chen, X. Chen, R. Tiwari, L. Yang, and G. Warner. A multimodal data mining framework for revealing common sources of spam images. *Journal of Multimedia*, 2009.

[42] B. B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, M. Yi, and K. Cai. Attacks and design of image recognition captchas. In *ACM CCS*, 2010.