Dynamic Cognitive Game CAPTCHA Usability and Detection of Streaming-Based Farming

Manar Mohamed, Song Gao, Nitesh Saxena, and Chengcui Zhang

Computer and Information Sciences
University of Alabama at Birmingham

***Abstract*—CAPTCHAs are a widely deployed mechanism to distinguish a legitimate human user from a computerized program trying to abuse online services. Attackers, however, have devised a clever and an economical way to bypass the security provided by CAPTCHAs by simply relaying CAPTCHA challenges to remote human-solvers. Most existing varieties of CAPTCHAs are completely vulnerable to such relay attacks, routinely executed in the wild.**

Dynamic Cognitive Game (DCG) CAPTCHAs are an upcoming CAPTCHA category which require the user to play a simple moving object matching game. Due to the dynamic and interactive nature of the underlying games, DCG CAPTCHAs may offer resistance to relay attacks. In this paper, we focus on a streaming-based DCG CAPTCHA relay attack whereby the game frames and responses are simply streamed between the attacker and a human-solver. We present a mechanism for detecting such a streaming-enabled game captcha farming based on real-time game statistics, such as play duration, mouse clicks and incorrect drags, fed to machine learning detection algorithms. To demonstrate the feasibility of our detection mechanism, we report on a three-dimensional study measuring: (1) the performance of legitimate DCG CAPTCHA users, (2) the performance of remote human-solvers in a DCG CAPTCHA streaming attack, and (3) the performance of gameplay behavioral features and machine learning classifiers in distinguishing human-solvers in a streaming attack from legitimate users. Our results show that it is possible to detect the streaming-based relay attack against many instances of DCG CAPTCHAs with a high overall accuracy (low false negatives and false positives). Broadly, DCG CAPTCHAs appear to be one of the first CAPTCHA schemes that enable reliable detection of relay attacks.

I. Introduction

The term CAPTCHA (hereafter referred to as "captcha") was first introduced in 2000 [3], 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 [13, 24].

Unfortunately, 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 [18]. These services offer two categories of attacks: *automated attacks* and *relay attacks*. Automated attacks (e.g., [10, 15, 16]) normally utilize image processing algorithms to solve the captcha, while relay attacks [18] 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 [18]. 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 [18]. 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 [18]. 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 [18].

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 (Section II provides the details). In this paper, we focus on an upcoming variety of captchas, called Dynamic Cognitive Game captchas (DCG), which may facilitate relay attack detection capability due to their dynamic and interactive nature. In a DCG captcha, the user has to perform a game-like cognitive task interacting with a series of dynamic images, such as playing a simple

USEC '14, 23 February 2014, San Diego, CA, USA
 ISBN 1-891562-37-1
http://dx.doi.org/10.14722/usec.2014.230021

object matching game (see Figure 1).

Due to their interactive and dynamic nature, DCG captchas may offer some level of resistance to relay attacks. In this paper, we study a specific form of a relay attack against DCG captchas, called *Stream Relay*, whereby the game challenges and responses are synchronously streamed between the attacker and the human-solver.

Our Contributions: We present a three-dimensional study with 120 Amazon Mechanical Turk participants to assess the usability of DCG captchas and the performance of the DCG captcha Stream Relay attack, and show how real-time gameplay characteristics and behavioral features can be used as part of machine learning algorithms to effectively detect the Stream Relay attack. The main contributions of the paper are outlined below:

1. DCG Captcha Usability Study: To evaluate the performance of DCG captchas when solved by legitimate users, we conduct a usability study with 40 users. This study serves as an important component of our Stream Relay detection mechanism. The results show fast response times and low error rates, and good user experience.

2. Stream Relay Attack Study: We formalize, design and implement the Stream Relay attack against DCG captchas, and perform a user study with 80 participants to measure the performance of the streamed version of DCG captchas when solved by remote users (serving as human-solvers in the relay attack). Our study captures three realistic attack settings: *high-latency channel* between attacker and solver, *low-latency channel* between attacker and solver, and *reduced game size*. The results show that the response times and error rates are generally higher when compared to those exhibited by legitimate users in the usability study, highlighting the possibility of relay attack detection.

3. Stream Relay Attack Detection: Based on the data collected from the above components of the study, we design and evaluate a Stream Relay attack detection mechanism. Our detection mechanism utilizes real-time game statistics, such as play duration, mouse clicks and incorrect drags, fed to machine learning algorithms, in order to differentiate legitimate user gameplay from human-solver gameplay in the relay attack. Our results show that it is possible to detect the streaming-enabled relay attack against many instances of DCG CAPTCHAs with a high overall accuracy (low false negatives and false positives).

II. Related Prior Work

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 [23], Baffle [7], handwritten [19] and PayPal, or within a video such as NuCaptcha and emergent captcha[24]. Captchas that challenge the users to recognize or classify objects in images, such as collage captcha [20], implicit captcha [4], Bongo, asirra captcha [8], PIX, ESP-PIX [22] and Google image orientation [11], have also been proposed. Some video-based captchas, such as content-based tagging of YouTube videos [17], and audio based captcha, such as Google, ebay, Yahoo, ReCaptcha, Slashdot and Math-function [12] audio captchas, have also been introduced.

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 [18] 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 [24] 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. Table II.1 provides a summary of how different categories and instances of existing captchas can be subjected to a relay attack.

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," [1] is also available. In this paper, we present an evaluation of the Stream Relay attack against DCG captchas and show how gameplay features and machine learning algorithms can be used to detect this attack.

III. Background and Overview

In this section, we present our implemented DCG captchas, provide an overview of the Stream Relay attack and our attack detection model, and review Virtual Network Computing (VNC), a streaming service used in our attack implementation.

A. DCG CAPTCHA Instances and Prototypes

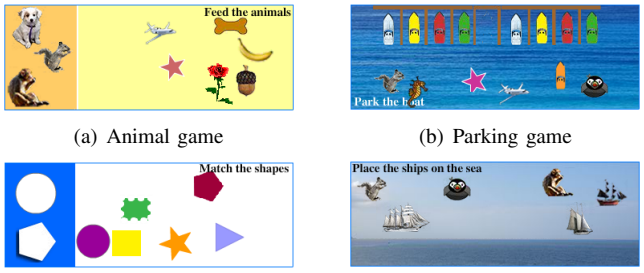
Due to the legal restrictions on attacking commercial DCG CAPTCHAs, we proceeded to develop our own animation-based DCG prototypes for the purpose of our study. Using Adobe Flash, we implemented four captcha games that represented a broad class of DCGs. These games are 360x130 pixels in size, and seamlessly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

TABLE II.1: Relay attacks against various types of captchas

CAPTCHA Category	Example Instances	Static/ Dynamic	User Interaction	Relaying Method	Detection Possible (apart from time-out)
Text	Gimpy, Yahoo, reCAPTCHA, Baffle, Handwritten, PayPal	Static	Type	Transfer challenge image	No
	NuCaptcha	Dynamic	Type	Transfer single snapshot	No
	Emergent captcha	Dynamic	Type	Transfer video file, or create video from snapshots and transfer	No
Image	Asirra, Bongo, collage, implicit	Static	Single/multiple mouse clicks	Transfer image	No
	PIX	Static	Type	Transfer image	No
	ESP-PIX	Static	Select answer	Transfer image	No
	Google image orientation	Static	Move a slider	Transfer image inside same applet used to display the original captcha	No
Video	Content-based tagging of YouTube videos	Dynamic	Type	Transfer video file	No
Audio	Google, ebay, Yahoo, Recaptcha, Slashdot, Math-function	Dynamic	Type	Transfer audio file, or record and send	No
DCG	are you a human	Dynamic	Multiple drag-and-drop	**Stream Relay**	Yes

- *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(b), 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) Animal game (b) Parking game

(c) Shape game (d) Ships game

Fig. 1: DCG captchas used for the study

DCG captchas are classified according to the number of target objects. The Ships game (Figure 1(d)), 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(c)) has a circle and a pentagon placed on the left side of the game as the two target objects. The Animal game (Figure 1(a)) is a three-target instance of DCG captcha. The Parking game (Figure 1(b)) 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.

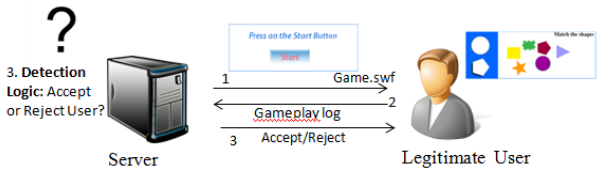
To complete a DCG captcha game, a user has to drag and drop all answer objects to their corresponding target objects. For example, in the Animal game, the user has to drag the bone to the dog, the acorn to the squirrel, and the banana to the monkey. The game is considered incomplete, and the user is 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.

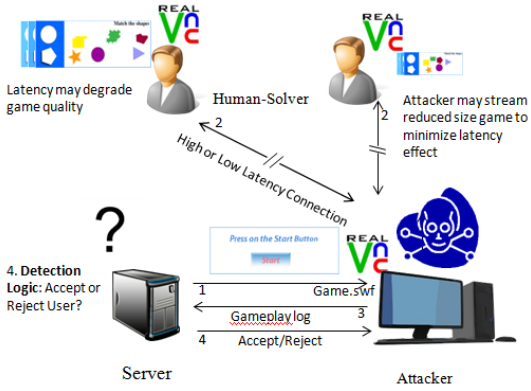
B. Stream Relay Attack, and Study Hypothesis

Web-based games are commonly developed using Flash and HTML5 with JavaScript. Both these platforms operate by downloading the game code to the client machine and executing it locally. To make it difficult for the attacker (bot) to deduce the answer objects and their respective locations from the downloaded code, the security model of DCG captchas must require the server to encrypt or obfuscate the game code. Moreover, the server needs to implement a method to prevent the games from being downloaded and embedded onto different domains.

In a normal setting, i.e., when a legitimate user $\mathcal{U}$ is interacting with the web service $\mathcal{W}$, the server $\mathcal{W}$ would send the encrypted/obfuscated DCG code to $\mathcal{U}$, $\mathcal{U}$ would render it



(a) Honest User Gameplay



(b) Human-Solver Gameplay in a Stream Relay Attack

Fig. 2: Honest Setting vs. Attack Setting

locally on his/her machine, and play the game. Since our Stream Relay attack detection strategy employs gameplay features, when $\mathcal{U}$ successfully finishes the game, the log of all of $\mathcal{U}$'s mouse interactions with the game is sent to $\mathcal{W}$. $\mathcal{W}$ then runs a detection algorithm on input of this log, and responds back by accepting (or rejecting) $\mathcal{U}$ as shown in Figure 2(a). All communication between $\mathcal{U}$ and $\mathcal{W}$ takes place over a secure channel (e.g., SSL/TLS).

Under Stream Relay, the attacker $\mathcal{A}$ obtains the DCG captcha challenge from $\mathcal{W}$, just like a legitimate user. The attacker runs a streaming server (such as a VNC server), and the human-solver $\mathcal{S}$ connects to the attacker machine through a streaming client (such as a VNC client embedded within web browser). This streaming software is responsible for delivering the DCG captcha frames to $\mathcal{S}$ and sending $\mathcal{S}$'s mouse interactions, such as drag-and-drop, mouse clicks and positions, to $\mathcal{A}$. $\mathcal{A}$ then simply forwards the log of this interaction between $\mathcal{S}$ and the game to $\mathcal{W}$. Finally, $\mathcal{W}$ would run the detection algorithm on input of this log, and responds back by rejecting (or accepting) $\mathcal{A}$. The Stream Relay attack flow diagram is shown in Figure 2(b). Due to network latency, our hypothesis is that $\mathcal{S}$ may suffer from the 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.

To test our hypothesis, we first measured the performance of legitimate users playing the DCG captcha games (usability study). Then, we measured the performance of the human-solvers in a Stream Relay attack under three different settings: (1) high-latency connection between attacker's machine and solver's machine (attacker in the US and solver outside the US), (2) low-latency connection between attacker's machine and solver's machine (attacker and solver both in the US), and (3) high-latency connection between the attacker's machine and solver's machine with game size down scaled to reduce the effect of latency. In Sections IV and V, we will report on our usability and Stream Relay attack experiments. Then, in Section VI, we will present how the DCG captcha server can utilize gameplay features and machine learning algorithms to detect the Stream Relay attacks on DCG captchas.

C. Virtual Network Computing (VNC) Overview

In our Stream Relay attack experiments, we use VNC as the streaming software. VNC makes it possible to remotely control a computer over a network connection. The VNC system consists of a VNC client, VNC server and VNC protocol. VNC utilizes remote framebuffer protocol (RFB). RFB is a machine independent protocol for remote access to graphical user interface [21].

The VNC client is a simple program. After connecting to the server, it falls into an infinite loop in which it sends requests to the server about a specific on-screen rectangle, and waits for the update. Whenever it receives the update which consists of an encoding changes between now and the last request, it processes the update and redraws the display [21].

The update sent from the server to the client has a header that contains general information about the message, and a series of rectangles, each of them has a header that contains the dimension of the data following it and its encoding. This structure makes it possible for the client to process the updates incrementally, the client does not need to wait till it receives the whole message before starting to process it. When the client has processed as much of the update as it has received, the client utilizes the ideal time to gather the input from the user mouse and keyboard and sends them to the server [21].

VNC server keeps scraping the framebuffer, the area of memory which stores the color value of each screen pixel. Whenever the framebuffer is changed, the server stores the modified region (the representation of the modified area and the modification made to it). The server keeps updating the modified region with the update of the framebuffer till it receives a client's request [21].

The client's request contains only the dimensions of a rectangle and a bit that indicates whether the request is incremental or not. If the request is not incremental, the server will send the whole framebuffer. Otherwise, the server sends the overlapping area between the requested rectangle and the modified region and clears the modification from the modified region [21].

In our study, we utilize RealVNC [2] to stream the DCG captcha from the attacker to the human-solver. RealVNC allows clients to connect to the server from web browser, so the clients do not need to install any additional software.

IV. Study Design and Procedures

In this section, we present the design of our experiments to study the usability of DCG captchas (i.e., legitimate

user gameplay), and the performance of DCG Stream Relay attack (i.e., human-solver gameplay). We utilized the Amazon Mechanical Turk (MTurk) service to recruit workers for the study. Overall, 120 users participated in our study. The project was approved by our University's Institutional Review Board.

A. Usability Study

We provide a detailed description of the usability study that we conducted to measure legitimate user's gameplay performance of our instances of DCG captchas. Forty MTurk workers were recruited, and paid $0.5 for their efforts that took approximately ten minutes. The MTurk workers were provided with four DCG captchas (Figure 1) in succession, and the game completion time, the number of object-drags, and the number of clicks were recorded. The order of the games presented to different participants was derived using a standard 4x4 Latin Square design to minimize learning effects.

The MTurk workers were subjected to a consent agreement, and a 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, each with 5 possible responses (5-point Likert scale, where 1 represents strongly disagreement and 5 represents strongly agreement) [5].

Table IV.1 (second column) shows the demographics of the 40 participants of our study. There were 67.5% males and 32.5% females. These participants are composed largely of young, educated individuals aged 18 - 35 years. A majority of the participants came from computer science, engineering, and business/finance backgrounds. A large percentage of the participants were from India. Our usability study demographics reflects the typical distribution of MTurk workers as explained in [14].

B. Stream Relay Attack Study

MTurk workers were hired for the Stream Relay attack study as well. The MTurk workers (serving the role of human-solvers) were asked to connect to a computer residing at the University of Alabama at Birmingham (UAB) and connected to UAB wireless network through a VNC java applet (serving the role of the attacker's machine). Just like the usability study, the workers were then asked to fill demographics form, play four DCG captchas (ordered based on 4x4 Latin Square), and fill a survey form about their experience. The participants were paid the same amount ($0.5) for their efforts as the usability study participants.

We used three different experiments to test various relay attack scenarios, as described below:

1) *High-Latency Relay*: The first scenario involved collecting data from participants residing outside the US. Since in a typical captcha relay attack [18], 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.
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. In our experiment, it 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.

TABLE IV.1: Demographics of participants in the usability and stream relay attack studies

	Usability	**Stream Relay Attack**		
Participant Location		Outside US		US
Game Size	360x130	360x130	180x65	360x130
Participant Size (N=120)	40	40	20	20
Gender (%)				
Male	67.5	67.5	80	80
Female	32.5	32.5	20	20
Age (%)				
<18	2.5	2.5	0	0
18 - 24	40	30	45	35
25 - 35	42.5	52.5	35	50
35 - 50	10	12.5	20	10
>50	5	2.5	0	5
Education (%)				
High school	10	0	0	55
Bachelor	60	57.5	75	40
Masters	27.5	42.5	25	5
Ph.D.	2.5	0	0	0
Field of Study/Profession (%)				
Computer Science	32.5	52.5	55	15
Engineering	20	20	30	5
Medicine	2.5	5	5	5
Law	2.5	2.5	0	0
Social sciences	5	0	0	5
Journalism	0	0	5	5
Finance	12.5	7.5	0	5
Business	10	0	0	20
Other	15	12.5	5	40
Country (%)				
Afghanistan	2.5	0	0	0
Australia	0	0	5	0
Egypt	2.5	0	2.5	0
India	62.5	92.5	85	0
Ireland	0	2.5	0	0
Italy	0	0	5	0
Macedonia	0	2.5	0	0
Romania	0	2.5	0	0
United Arab Emirates	2.5	0	2.5	0
United States	30	0	0	100
Vietnam	0	0	5	0

Table IV.1 (third, fourth and fifth columns) shows the participant demographics for the three Stream Relay attack scenarios. For the High-Latency Relay scenario, 40 individuals participated in our experiment, of which 67.5% were males and 32.5% were females. These are participants composed largely of young, educated individuals aged 18 - 35 years. A majority of the participants came from computer science, and engineering background. Most of the participants were from India (92.5%). The demographics of this scenario is quite similar to the one of the usability study, which allowed us

to fairly compare the two settings as part of our relay attack detection approach.

For the Small Game Relay scenario, we collected data from 20 participants. Again, the male demographics dominated the female demographics, at 80% males vs 20% females, mostly young. The participants were mostly educated, with 75% with Bachelor's degree, and 25% with a Master's degree. The participants were mostly from India (85%), and mostly from computer science (55%) and engineering (30%). The demographics of this scenario is also much similar to the one in the usability study.

Finally, for the Low-Latency Relay scenario, we collected data from 20 participants inside US. The male to female ratio was the same as for the second scenario, 80% male and 20% female, and the participants were again mostly young. Most of them had high school degree or Bachelor's degree in this case, in contrast to the previous scenarios.

V. Study Results

In this section, we present the results of the usability and Stream Relay attack studies. One goal of our study was to analyze and contrast the efficiency, robustness, and user experience provided by the DCG captchas to the legitimate users and human-solvers in a relay attack. For each of the four games tested in our experiment, game statistics such as completion times and mouse drag-and-drop and clicks were automatically logged by our web-interface software.

The efficiency of the captchas was measured as the game completion time. The likelihood of incorrect drag and drop attempts, incorrect attempts to grab objects, and the game not being completed were used to derive the robustness of the captcha. Finally, we analyzed user experience using the participants' SUS ratings and qualitative feedback.

The experimental results focus on the following metrics:

- *Overall gameplay time* – the gameplay time including both complete and incomplete games.
- *Successful gameplay time* - the gameplay time for games that are successfully completed.
- *Error rate* – the ratio of the number of incomplete games to the total number of games.
- *Drag error rate* – ratio of the number of invalid drags-drops to the total number of drag-drops.
- *Click error rate* – ratio of the number of clicks resulting in invalid object selection to the total number of clicks.

A. Usability Study Results

The results obtained from the experiment performed to assess the usability of the DCG games are shown in Table V.1.

The results indicate that the Animal game took the longest of the four games, followed by the Shape game, the Ships game and the Parking game. We observed that the game completion time was proportional to game complexity, defined by the number of game completion criteria (i.e., number of target objects, or drags necessary). The Animal game, that required three successful drag-and-drops for game completion, took almost twice as long to complete as the Parking game that had only one game completion criteria.

TABLE V.1: Completion times, and error rates per drag and per click, per game type in *Usability Study*

Game Name	Overall Time (sec.)	Successful Time (sec.)	Error Rate	Drag Error Rate	Click Error Rate
	Mean (Std. Dev.)		Mean		
Animal	16.10 (10.90)	14.97 (8.43)	0.03	0.26	0.45
Parking	8.53 (7.01)	8.53 (7.01)	0	0.54	0.69
Shape	13.22 (14.54)	9.42 (6.06)	0.08	0.16	0.35
Ships	11.46 (10.69)	10.13 (6.81)	0.03	0.32	0.45

TABLE V.2: SUS Scores

	Mean	Std. Dev.
Usability Study	73.25	15.07
High-Latency Relay	59	12.83
Small Game Relay	57	14.97
Low-Latency Relay	65.11	18.42

The error rate was the highest for the Shape game, at 8%, followed by the Animal game and the Ships game, at 3%, and then the Parking game, at 0%. Both the drag error rate and the click error rate were the highest for the Parking game at 54% and 69%, respectively. Upon analyzing the error rates for the Parking game, we found out that approximately 27% of the participants completed the game without any extra drag-drop, while 68% of the participants needed up to 3 extra drag-drops. Only 5% participants needed more than 3 drag-drops to complete the game. Although the drag error rate, at 54%, seems very high, it can be attributed to the erroneous 5% participants, who, on average, required 13 extra drag-drop attempts to complete the game. In a similar manner, the high click error rate of 69% could be attributed to 5% of the users who required 16 extra clicks to complete the game.

We next evaluated the data collected during the post-study phase from the participants. The mean SUS score came out to be 73.25, with a standard deviation of 15.07 (Table V.2). The results from SUS show that the DCG captchas are usable.

B. Stream Relay Attack Study Results

For Stream Relay Attack, we utilized three distinct scenarios, as described previously, to perform our experiments: (1) High-Latency Relay: Normal sized captcha for participants outside the US. (2) Small Game Relay: Small-sized captcha for participants outside the US; and (3) Low-Latency Relay: Normal sized captcha for participants in the US.

The results for the first, High-Latency Relay, scenario are shown in Table V.3. 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.

TABLE V.3: Completion times, and error rates per drag and per click, per game type in *High-Latency Relay* attack scenario

Game Name	Overall Time (sec.)	Successful Time (sec.)	Error Rate	Drag Error Rate	Click Error Rate
	Mean (Std. Dev.)		Mean		
Animal	42.48 (16.48)	34.04 (13.52)	0.33	0.43	0.79
Parking	38.51 (20.82)	25.61 (15.79)	0.38	0.75	0.94
Shape	30.19 (19.62)	21.53 (12.79)	0.23	0.43	0.76
Ships	34.49 (16.08)	29.98 (13.00)	0.15	0.45	0.8

TABLE V.4: Completion times, and error rates per drag and per click, per game type in *Small Game Relay* attack scenario

Game Name	Overall Time (sec)	Successful Time (sec.)	Error Rate	Drag Error Rate	Click Error Rate
	Mean (Std. Dev.)		Mean		
Animal	46.27 (12.90)	37.11 (8.21)	0.4	0.29	0.75
Parking	30.32 (20.61)	20.43 (13.20)	0.25	0.57	0.94
Shape	30.18 (14.32)	26.86 (10.86)	0.1	0.51	0.81
Ships	37.70 (19.29)	22.82 (8.17)	0.4	0.27	0.59

TABLE V.5: Completion times, and error rates per drag and per click, per game type in *Low-Latency Relay* attack scenario

Game Name	Overall Time (sec.)	Successful Time	Error Rate	Drag Error Rate	Click Error Rate
	Mean (Std. Dev.)		Mean		
Animal	26.17 (20.20)	17.54 (11.71)	0.2	0.35	0.67
Parking	18.72 (18.92)	14.13 (13.67)	0.1	0.46	0.88
Shape	17.48 (17.38)	15.25 (14.76)	0.05	0.21	0.57
Ships	18.47 (16.16)	16.28 (13.39)	0.05	0.25	0.54

Furthermore, upon comparing the mean time taken to complete the games (Successful Time) between the usability and High-Latency Relay using Mann-Whitney U test with Bonferroni correction, we found a statistically significant difference, with $p < 0.0001$, for each of the four games[1]. The error rates were also significantly higher than those exhibited in the usability study, on an average of 84%. The drag error rate and click error rate were 40% and 42% higher for High-Latency Relay attack compared to usability study, respectively. The longer game completion time and higher error rates for this Stream Really attack scenario 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 Table V.4, were still comparable to that of stream relay attack with normal game size, with longer gameplay time and higher error rates compared to the 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 and click error rates were 16% and 37% higher than that for usability study, respectively. 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 $p < 0.0001$. 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 Table V.5. 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: Animal game: $p = 2.9929$, Parking game: $p = 0.928$, and Shape game: $p = 0.5695$. 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, drag error rate and click error rate were 41%, 1% and 27% higher for Low-Latency Relay attack compared to usability study, respectively.

The analysis of the SUS scores for the three relay attack scenarios is summarized in Table V.2. The mean of the SUS for the first, second and third relay attack scenarios came to be 59, 57 and 65.11, respectively, which is consistently lower than the mean SUS score obtained from the usability study. Comparing SUS score between the 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 ($p = 0.5695$).

VI. Stream Relay Attack Detection

In the previous section, 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. To this end, we explore the following aspects of the human behavior data collected for each gameplay instance:

- *PlayDuration*: overall gameplay time (in seconds) of a game instance for an honest or remote user.
- *IsTimeout*: indicating whether a game is unfinished due to exceeding the maximum time limitation (e.g., 60s).
- *TimeStamps*: a *k*-by-1 numeric vector consisting of timestamps. Each timestamp is a relative time reference in millisecond, contributed by both the mouse-dragging event and the mouse-status event (i.e., left click up/down).
- *DraggingObjs*: a *k*-by-1 binary vector. The drag of an object at the corresponding timestamp is indicated as 1, otherwise 0. A successful drag-and-drop of an answer object to the corresponding target object will have a dragging track ending up with the letter 'y' (e.g., 0,1,1,$\cdots$,y,0,$\cdots$). Otherwise, the track ends up with the letter 'n' (e.g., 0,1,1,$\cdots$,1,n,0,$\cdots$).

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

TABLE VI.1: Class distribution of non-timeout users

Game Name (N)	Usability (40)	HighLatency Relay (40)	SmallGame Relay (20)	LowLatency Relay (20)
Animal	39	27	12	16
Parking	40	25	15	18
Shape	37	31	18	19
Ships	39	34	12	19

- *MouseStatus*: a *k*-by-1 binary vector indicating whether the left key is pressed at the corresponding timestamp (i.e., 1 for down and 0 for up, respectively).

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*: as mentioned above.
2) *Successful drag rate*: the ratio of the number of successful drag-and-drops to the total number of drag-and-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*.

Both Support Vector Machine (SVM) [6] and K-Nearest Neighbors (KNN) [9] 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, namely 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.

The timeout-user records are excluded from the dataset based on the consideration that if a game cannot be completed within a reasonably long game time frame (e.g., 60s), it is reasonable for the game server to reject the user no matter he/she is an honest user or a remote relay attack user. Table VI.1 shows the class distribution of non-timeout users for each game.

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:

TABLE VI.2: Results of using the optimal feature subset for each game in the classification of *legitimate user* and *High-Latency* relay attacker

Game Name	Feature Subset	Method	Average Accuracy	Average Precision	Average Recall
Animal	6	C-SVC linear	0.95	0.95	**0.98**
	6,7	nu-SVC poly	0.95	0.95	**0.98**
Parking	5,6	KNN Euclid	0.86	0.83	**0.98**
	5,6,7	KNN Euclid	0.86	0.83	**0.98**
Shape	4,5,6,7	nu-SVC rad	0.86	0.82	**0.95**
Ships	4,5,6	nu-SVC rad	0.93	0.93	**0.95**

TABLE VI.3: Results of using the common optimal general feature subset for all games in the classification of *legitimate user* and *High-Latency* relay attacker

Game Name	Feature Subset	Method	Average Accuracy	Average Precision	Average Recall
Animal	6	nu-SVC radial	0.95	0.94	**0.98**
Parking	6	nu-SVC radial	0.85	0.83	**0.95**
Shape	6	nu-SVC radial	0.80	0.76	**0.94**
Ships	6	nu-SVC radial	0.92	0.91	**0.94**

- *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 and 2. Of these, recall is more important than precision 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.

$$Precision = TP/(TP + FP),\ Recall = TP/(TP + FN) \quad (1)$$

$$Accuracy = (TP + TN)/(TP + FP + TN + FN) \quad (2)$$

In the **first classification experiment**, we build a classifier for each of the four game types to distinguish a legitimate user from a High-Latency relay attacker (i.e., corresponding to the first scenario of the Stream Relay attack). The average measurement values, as shown in Table VI.2, are calculated from running a 10-fold cross validation 5 times for each test case. The results show that High-Latency relay attack can be detected fairly accurately with a reasonably high precision and a very high recall. The Animal and the Ships game provided the best performance, which is expected as both of them require three drags and drops, which is more than the number of required drags and drops for the Shape and Parking games. In order to find a general feature subset that has the highest average accuracy for all game prototypes, we further ranked the average accuracy of each feature set in all games. The results as shown in Table VI.3 indicate that Feature 6 with SVM gives the highest average accuracy for all games, which makes sense because this feature exists in all optimal feature subsets of each game in Table VI.2.

In our **second classification experiment**, we apply all the models (i.e., total amount: 1016) trained from the first experiment on the Small Game Relay dataset in order to test

TABLE VI.4: Classification results of using the optimal feature subset for each game in the classification of *Small Game* relay attackers using the original model

Game Name	Feature Subset	Method	Avg. Accuracy
Animal	6	nu-SVC poly	1.00
	6,7	nu-SVC poly	1.00
Parking	4	nu-SVC linear	0.73
Shape	1	nu-SVC linear	1.00
	2	nu-SVC linear	1.00
Ships	3	nu-SVC poly	1.00
	2,3	nu-SVC poly	1.00

TABLE VI.5: Results of using the common optimal feature subset for all games in the classification of *Small Game* relay attackers using the original model (Parking game should be discarded)

Game Name	Feature Subset	Method	Avg. Accuracy
Animal	1,3,5,6	C-SVC poly	1.00
Parking	1,3,5,6	C-SVC poly	0.13
Shape	1,3,5,6	C-SVC poly	0.94
Ships	1,3,5,6	C-SVC poly	0.92

whether the current model can also detect the relay attackers who played in a smaller game window). Because the testing dataset contains only Small Game Relay records (i.e., True Negative), calculating precision and recall is not meaningful due to the lack of True Positive data. The classification results for the optimal feature subset of each game are shown in Table VI.4. The proposed feature subsets can achieve 100% accuracy for all games except the Parking game. The optimal feature subset (i.e., Feature 4) for the Parking game can only achieve 73% accuracy, which indicates that few number of answer objects in a game is likely not secure against Small Game Relay attack because the possibility for the relay attackers to generate invalid mouse clicks is low. In this light, the captcha service may choose to remove the Parking game from their game database and use the original training model without compromising the security against relay attacks or usability.

With the exclusion of the Parking game, the optimal general feature subset for all the other three games includes Features '1, 3, 5, 6' when SVM with certain parameter setting is used as shown in Table VI.5. Using Feature '6', the optimal feature subset for predicting relay attack on games with full game window size, can also achieve acceptable accuracies ($>$ 89%) on these three games (Table VI.6) when SVM.

Then, we build a model to distinguish between legitimate user and Small Game Relay attacker. The obtained results are shown in Table VI.7. The average recall is 100% for all of the games expect for the Parking game. The average accuracy and precision for all the games are 91.5% and 90%, respectively.

TABLE VI.6: Results of using Feature '6' for all games in the classification of *Small Game* relay attackers using the original model (Parking game should be discarded)

Game Name	Feature Subset	Method	Avg. Accuracy
Animal	6	nu-SVC poly	1.00
Parking	6	nu-SVC poly	0.07
Shape	6	nu-SVC poly	0.89
Ships	6	nu-SVC poly	0.92

TABLE VI.7: Results of using the optimal feature subset for each game in the classification of *legitimate user* and *Small Game* relay attacker

Game Name	Feature Subset	Method	Average Accuracy	Average Precision	Average Recall
Animal	6,7	C-SVC linear	0.96	0.95	**1.00**
Parking	2,4,5,6,7,8	C-SVC linear	0.83	0.82	**0.97**
Shape	3,5,6,7	KNN Euclid	0.92	0.90	**1.00**
Ships	5,6,8	C-SVC linear	0.95	0.93	**1.00**

TABLE VI.8: Results of using the optimal feature subset for each game in the classification of *legitimate user* and *Low-Latency* relay attacker

Game Name	Feature Subset	Method	Average Accuracy	Average Precision	Average Recall
Animal	3,5,7	KNN min	0.79	0.77	**0.99**
Parking	5,6	nu-SVC rad	0.77	0.76	**0.97**
Shape	1,2,4,5,6,7	nu-SVC rad	0.76	0.75	**0.96**
Ships	5,6,7	C-SVC poly	0.77	0.75	**1.00**
	3,5,6,7	C-SVC poly	0.77	0.75	**1.00**

TABLE VI.9: Results of using the optimal feature subset for each game in the classification of *Low-Latency* relay attackers using the original model

Game Name	Feature Subset	Method	Avg. Accuracy
Animal	2	nu-SVC poly	0.94
	3,4	nu-SVC poly	0.94
Parking	2	nu-SVC poly	1.00
	3	nu-SVC poly	1.00
	2,3	nu-SVC poly	1.00
Shape	4	nu-SVC poly	1.00
Ships	1,3	nu-SVC poly	1.00

For the last dataset, using the data collected from Low-Latency relay, we build a classifier for each of the four game types, which could distinguish a legitimate user (i.e., corresponding to the usability records) from a Low-Latency relay attacker. The average measurement values, as shown in Table VI.8, are calculated from running a 10-fold cross validation 5 times for each test case. The results show that Low-Latency Relay attack can be detected with a high recall for all games, of above 96%. The Ships and Animal games seem to provide the best performance, which is justifiable given the games' complexity (larger number of target objects than the other games). However, the classification accuracy is lower than it is in High-Latency Relay – on average 77% for Low-Latency compared to 90% for High-Latency. This indicates that latency may increase the accuracy of attack detection.

In our **third and final classification experiment**, we used all the models (i.e., total amount: 1016) trained from the first experiment to predict the Low-Latency Game Relay dataset. Because the testing dataset contains only True Negative records, calculating precision and recall is not meaningful (due to the same reason as explained in the second experiment). The classification results using the optimal feature subset of each game are shown in Table VI.9. The proposed feature subsets can achieve accuracy of at least 94% for all games.

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 VI.10, suggesting that the best average accuracy and average recall are achieved for the Animal and Ships games,

TABLE VI.10: 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
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**
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**

followed by the Shape game and then the Parking game. This suggests that increasing the number of the required drag-and-drops improves the classification performance. A DCG captcha service may employ this global model without having to remove any of the four games from the game database.

VII. Discussion, Limitations, and Conclusions

Our study results offer many primary insights about DCG captchas. First, we have shown that DCG captchas are quite usable. Second, and more importantly, we demonstrated that DCG captchas provide a superior ability of detecting relay attacks over all well-known types of captchas.

The ability of DCG captchas in detecting streaming-enabled relay attack is based on two main properties. First, the dynamic nature of DCG captchas makes streaming the underlying game a bit tricky, especially over high-latency networks (e.g., when the attacker is in the US and the human-solver is in India) but also in low-latency networks (e.g., when both the attacker and solvers are located within the US). Second, DCG captchas require multiple interactions between the user and captcha. Increasing the number and complexity of the required interaction increases the ability of relay attack detection. For example, the accuracy of our relay attack detection algorithm is much better in case of the Animal and Ships games, which require three drags-and-drops, than it is in the Parking game which requires a single drag-and-drop.

The proposed relay detection is quite efficient. For example, when using SVM, it requires 0.224 msec training with 66 training records, and about 0.087 msec for testing a single record. Although the training time will increase with the increase of the number of training instances, the testing that requires instant response (accept or reject to the user) keeps the same efficiency. This validates that our proposed method could be used in an online real-time setting without incurring noticeable delays.

Our work utilized MTurk system to recruit participants for the study. On one hand, it allowed us to collect data outside of the lab from users with varying demographic characteristics. On the other hand, it limited our ability to test different screen sharing applications (such as Microsoft Remote Desktop Protocol, Apple Remote Desktop and TeamViewer) because apart from RealVNC, all applications require the participants to install additional client software on their machine. Furthermore, the amount of money we paid to the participants might not be the same as what is typically paid to human-solvers in a real captcha relay attack. This may have implications in terms of participant motivation in solving the games.

Our study suggests increasing the interaction between the users and the captcha improves the ability to detect relay attack. This suggests the need to generate more games with larger number of target objects to verify our finding, and to assess the impact of asking the user to drag-and-drop more objects on the usability of DCG captchas, and on the security against automated attacks.

The small size of the game, simplicity of the underlying cognitive task, and the requirements of simple interactions (drag-and-drop) suggest that DCG captcha would have good usability on touch screen devices, such as smart phones and tablets. However, we have not considered these devices in our current study. Further research is necessary to validate the usability of DCG captchas and the accuracy of our streaming attack detection mechanism after incorporating data from touch screen devices to our training/testing models.

Acknowledgments: This work was supported in part by the NSF CNS-1255919 grant. We are thankful to: Alexander De Luca (our shepherd), Matthew Smith and anonymous USEC'14 reviewers for their valuable feedback, and Maliheh Shirvanian for her comments on a draft version of this paper.

References

[1] Are You a Human. http://areyouahuman.com/.
[2] Realvnc. http://www.realvnc.com/.
[3] L. Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *Advances in Cryptology, EUROCRYPT*. 2003.
[4] H. S. Baird and J. L. Bentley. Implicit captchas. In *Electronic Imaging*, 2005.
[5] J. Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189:194, 1996.
[6] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3), 2011.
[7] M. Chew and H. S. Baird. Baffletext: A human interactive proof. In *Electronic Imaging*, 2003.
[8] J. Elson, J. R. Douceur, J. Howell, and J. Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. In *ACM Conference on Computer and Communications Security*, 2007.
[9] 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.
[10] G. Keizer. Spammers' Bot Cracks Microsoft's CAPTCHA. In *Computer World*, Available at: http://www.computerworld.com/s/article/9061558/Spammers_bot_cracks_Microsoft_s_CAPTCHA_, 2008.
[11] R. Gossweiler, M. Kamvar, and S. Baluja. What's up captcha?: a captcha based on image orientation. In *World wide web*, 2009.
[12] J. N. Gross. Captcha using challenges optimized for distinguishing between humans and machines, 2009. US Patent App. 12/484,800.
[13] J. M. G. Hidalgo and G. Alvarez. Captchas: An artificial intelligence application to web security. *Advances in Computers*, 83, 2011.
[14] P. Ipeirotis. Demographics of mechanical turk. 2010.
[15] Jeff Yan and Ahmad Salah El Ahmad. A Low-cost Attack on a Microsoft CAPTCHA. In *ACM Conference on Computer and Communications Security*, 2008.
[16] K. Kluever. Breaking the PayPal.com CAPTCHA. Available at: http://www.kloover.com/2008/05/12/breaking-the-paypalcom-captcha/, 2008.
[17] K. A. Kluever and R. Zanibbi. Balancing usability and security in a video captcha. In *Symposium on Usable Privacy and Security*, 2009.
[18] 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.
[19] 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.
[20] M. Shirali-Shahreza and S. Shirali-Shahreza. Collage captcha. In *Signal Processing and Its Applications*, 2007.
[21] C. Taylor and J. Pasquale. Improving video performance in vnc under high latency conditions. In *Collaborative Technologies & Systems*, 2010.
[22] L. Von Ahn and L. Dabbish. Labeling images with a computer game. In *SIGCHI conference on Human factors in computing systems*, 2004.
[23] 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.
[24] Y. Xu, G. Reynaga, S. Chiasson, J. Frahm, F. Monrose, and P. Van Oorschot. Security and usability challenges of moving-object captchas: decoding codewords in motion. In *USENIX Security*, 2012.