2016

Human-Machine and Human-Human Authentication Through Active User Interaction

Manar Mohamed
University of Alabama at Birmingham

HUMAN-MACHINE AND HUMAN-HUMAN AUTHENTICATION THROUGH ACTIVE USER INTERACTION

by

MANAR MOHAMED

NITESH SAXENA, COMMITTEE CHAIR
NICOLAS CHRISTIN
SOPHIE JOERG
ALAN P SPRAGUE
CHENGCUI ZHANG

A DISSERTATION

Submitted to the graduate faculty of The University of Alabama at Birmingham,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

BIRMINGHAM, ALABAMA

2016

HUMAN-MACHINE AND HUMAN-HUMAN AUTHENTICATION THROUGH ACTIVE USER INTERACTION

MANAR MOHAMED

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

ABSTRACT

Authentication is a fundamental security component of various critical applications. It is essential to differentiate a human user from a bot (*human-machine authentication*) to prevent against automated mechanisms that attack and abuse the resources of an online entity. Authentication is also essential to differentiate one human user from another (*human-human authentication*) to securely control the access of online accounts and computer terminals. Unfortunately, the security and usability requirements of authentication have not been adequately addressed. The current and almost universally deployed techniques, CAPTCHAs for human-machine authentication, and passwords or biometrics for human-human authentication, all suffer from numerous well-documented usability and security drawbacks.

In this research, we aim to address the security and usability problems of authentication through the use of *active user interaction* in the authentication process. Active user interaction boasts to provide two key advantages. *First*, it can *enhance the security* of the authentication process by adopting multiple rounds of active interactions which serves as a mechanism to prevent against several types of attacks, including replay, spoofing and relay attacks. *Second*, it can *enhance the usability* of the authentication process by actively engaging the user and eliminating the need for highly distorted characters/images commonly used in most currently-deployed CAPTCHAs, which users may find frustrating.

The contribution of this dissertation lies in the realization of interaction-enhanced security approaches that may help in addressing the aforementioned shortcomings with current authentication technologies. In the context of human-machine authentication, we investigate **interactive CAPTCHAs**. These represent interactive games that are easy for the humans, but may be hard for a computer, to play successfully. Unlike existing solutions, interactive CAPTCHAs may be easy, fun, suitable for mobile devices, and resilient to both automated and human-solver relay attacks (due to their dynamic & interactive nature). First, we explore a simplistic form of interactive CAPTCHAs (simple moving-object drag and drop games) and conduct human factor studies

to evaluate their usability and security against automated and relay attacks. The results show that the developed CAPTCHAs are highly usable and can offer some resilience against relay attacks and facilitate relay attack detection. However, these CAPTCHAs were also found to be vulnerable to different forms of automated attacks based on image processing techniques. Second, we study multiple methods to enhance the security of the proposed interactive CAPTCHAs without undermining their usability and resistance to relay attacks. These include incorporating the notions of emergence, image semantics and image orientation, and combinations thereof.

In the realm of human-human authentication, we introduce **interactive biometrics** based on game playing patterns and multi-modal behavioral features. As opposed to most existing biometric systems, game biometrics are software-only, non-invasive and potentially very difficult to impersonate. We design and implement an interactive biometrics system based on simple drag and drop games to capture the unique user interactions. Our system is built using machine learning techniques and extracts a number features from each game challenge solving instance that capture the multiple unique cognitive abilities and the mouse dynamics of the users. We collect data sets in online and lab settings, and show that our system can identify the legitimate users and the zero-effort attackers ("different users") with a high accuracy. We evaluate the security of the proposed game biometrics system against external attacks (e.g., device theft) as well as against a new powerful internal attack framework (malicious code) that can sniff and manipulate touchscreen events. Our analysis suggests that the proposed system can be resistant to these attacks unlike other existing behavioral biometrics schemes.

Keywords: Authentication; CAPTCHAs; Biometrics; Usability; Interaction

DEDICATION

I dedicate this work to my late father, who passed away while I am far away from him busy with this work, may God bless him.

ACKNOWLEDGMENT

I am grateful to my supervisor, **Professor Nitesh Saxena**, for his invaluable guidance, encouragement and support throughout my studies. I owe him great debt for everything I learned in the past few years.

I have also had a great pleasure to work with **Professor Chengcui Zhang**. A significant portion of the work is from joint publication with her. I am also thankful for my other committee members: **Professor Nicolas Christin**, **Professor Sophie Joerg**, and **Professor Alan Sprague**.

Moreover, I am glad to work with Professor Ponnurangam Kumaraguru, Professor Paul C. Van Oorschot, Anders Borg, Wei-Bang Chen, Song Gao, Michael Georgescu, Eric M Lienert, Niharika Sachdeva, Babins Shrestha, Prakash Shrestha and Sandeep Tamrakar.

I am also grateful for many friends and colleagues for their support and comradeship; especially Abhishek Anand, Nardeen Farag, Md Mahmoud Hossain, Mohammad Kamrul Islam, Rasib Khan, Suraj Maharjan, Fadel Megahed, Reed M Milewicz, Dibya Mukhopadhyay, Ajaya Neupane, Shaahid Noor, Maliheh Shirvanian, Prasha Shrestha, Vinaya Shrestha, Shams Zawoad and all current and former members in the Department of Computer and Information Sciences at University of Alabama at Birmingham

I acknowledge the financial support by Comcast, Google and National Science Foundation (NSF).

Finally, I would like to express my deepest gratitude for constant support and encouragement from my parents, sister and brothers during the past years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Authentication is a fundamental security component of many critical applications. It is essential to differentiate a human user from a bot (*human-machine authentication*) to prevent against automated mechanisms that attack and abuse the resources of an online entity. Authentication is also essential to differentiate one human user from another (*human-human authentication*) to securely control access online accounts and computer terminals.

Unfortunately, the security and usability requirements of authentication have not been adequately addressed. The current, almost universally deployed techniques, CAPTCHAs for human-machine authentication, and passwords or biometrics for human-human authentication, all suffer from numerous well-documented usability and security drawbacks. For example, most existing CAPTCHAs are often very difficult for human users to solve, and can be broken using automated attacks or relay attacks based on cheap human labor. Similarly, passwords are only weak secrets prone to guessing and brute forcing attacks of different forms, and most existing biometrics are often deemed invasive by users, have high error rates and are susceptible to replay and spoofing attacks.

This work aims to address the problem of authentication through the use of *active user interaction* in the authentication process. Active user interaction boasts to provide two key advantages. *First*, it can *enhance the security* of the authenticating process as multiple rounds of active interaction would serve as a mechanism to prevent against several types of attacks, including replay, spoofing and relay attacks. *Second*, it can *enhance the usability* of the authentication process by actively engaging the user in the process, and eliminating the need for highly distorted characters/images commonly used in most currently-deployed CAPTCHAs, which users may find frustrating.

1.1 Limitations of Current Authentication Systems

The abuse of the resources of an online (web) service using automated means, such as in the form of spam [3], denial-of-service (DoS) [4] or password dictionary attacks [5], is a common security

problem plaguing today's Internet. To prevent such abuse, one primary defense is a CAPTCHA [6], commonly deployed on most websites these days. A CAPTCHA is a tool for human-machine authentication, to distinguish between a human user and a computer by presenting requesters with a task that is relatively easy for a human but much harder for a computer. This most often takes the form of a word or phrase that has been obscured or distorted. Such textual CAPTCHAs, however, exhibit poor usability and often cause user frustration [7, 8] (more so in the context of mobile devices [9] due to their small form factor). Alternate CAPTCHAs aimed at improving the usability, in contrast, have been subject to several automated attacks. Furthermore, most, if not all, traditional CAPTCHAs are completely vulnerable to low-cost human-solver *relay attacks* [10], whereby the attacker simply recruits a remote human to solve the CAPTCHA challenge in real-time. These limitations with existing CAPTCHAs are reviewed in *Section 2.1*.

User authentication (i.e., human-human authentication) is a classical and one of the most important problems in security. The problem occurs whenever a user, wanting access to a computing device (remote or otherwise), has to prove to the device her possession or ownership of certain credentials, that she has pre-established with that device. The primary goal of user authentication is to ascertain that only a legitimate user, possessing appropriate credentials, is granted access. The increasing popularity of personal devices and Internet websites, and the sensitivity of information they often store, prompts the need for usable authentication mechanisms. However, user authentication still remains to be a challenging problem in practice, with none of the existing authentication primitives providing a good balance between the conflicting requirements of usability and security. In particular, passwords are the most dominant authentication approach deployed today, but suffer from many well-documented security and usability problems. Traditional biometric systems often have high error rates, susceptible to impersonation or spoofing attacks, may require additional hardware, and are deemed invasive by many users. We review the limitations of prior user authentication technologies in *Section 2.2*.

1.2 How Interactivity Can Help?

Active interaction boasts to provide significant advantages in terms of security and usability of the authentication process. *First*, it can significantly help improve the security of authentication

in comparison to existing solutions. In case of human-machine authentication, the requirement for multiple interactions between the CAPTCHA and the user may help in preventing and/or detecting relay attacks. Moreover, the interactive mechanisms do not need to be based solely on the complexity of image segmentation and recognition of distorted objects as in conventional text CAPTCHAs, but on object recognition over wide variety of objects and on finding semantic relationships between the objects. This would add complexity in terms of implementing bots to solve the CAPTCHA automatically.

In case of human-human authentication, the active user interaction may be utilized as a *behavioral biometrics* [11, 12] as this interaction could be unique per user. Adding this biometrics to the process of password entry may further help in preventing impersonation/spoofing attacks as it can serve as a second factor in the authentication process (the attacker should not only know the victim's password to bypass the authentication mechanism but also need to mimic the user interactions with the authentication object). Moreover, by using authentication methods that contain moving objects or objects that are placed at random locations (i.e., the characters are placed randomly on the screen), this randomization may serve as a defense against side-channel attacks, (i.e., attacks that try to deduce the entered password based on the locations the user has pressed on the screen), replay attacks and spoofing attacks.

Second, engaging the user in the authentication process via interactivity may enhance the system usability and user experience. Giving instant feedback to the users and eliminating objects distortion may help in reducing the error rate of CAPTCHA solving. The engagement may also improve the user perception of the CAPTCHA and makes CAPTCHA solving a fun activity rather than a frustrating task. Moreover, interactive solutions will be more suitable for small touchscreen devices, where reading/entering text might be challenging.

1.3 Thesis Statement & Main Contributions

Our thesis is that active user interaction, utilizing multiple rounds of interaction between the user and the authentication constructs represented as simple drag and drop games, can help to enhance the usability and security of human-machine and human-human authentication. Utilizing active user interaction for authentication enhances the usability by engaging the user in the authentication process. Moreover, active user interaction enhances the security of human-machine authentication against relay attacks and enhances the security of human-human authentication against replay and spoofing attacks.

The main contributions of this work lie in (1) the design of interactive authentication mechanisms and (2) the usability and (3) the security evaluation of the proposed interactive authentication mechanisms in contrast to traditional authentication mechanisms.

1.3.1 Human-Machine Authentication

To help meet the problems with existing CAPTCHAs, we designed **interactive CAPTCHAs** based on games that would be easy for humans, but hard for computers. Unlike existing solutions, game CAPTCHAs would be (1) *user-friendly*, (2) *resilient to both automated and relay attacks* (due to their dynamic & interactive nature), and (3) *suitable for mobile devices*. The main contributions of this research related to Human-Machine authentication are listed as follows.

1. We formalized, designed and implemented a simple instance of interactive CAPTCHAs, namely, Simple Dynamic Cognitive Game (S-DCG) CAPTCHAs. An S-DCG CAPTCHA involves objects floating around within an image, and the user's task is to match the objects with their respective target(s) and drag/drop them to the target location(s).

2. We conducted a usability study of these instances on PCs as well as on touchscreen devices, evaluating them in terms of time-to-completion, error rates and perceived usability. Our results indicated the overall usability to be very good.

3. We assessed the security of S-DCG CAPTCHA against relay attacks:

   (a) We explored the security of S-DCG CAPTCHAs against a static relay attack, where the attacker sends only single snapshots of the S-DCG (in line with traditional

CAPTCHA relay attack). This attack reduces to a reaction time task for the solver. We conducted a user study to evaluate the performance of this attack. In general, our results indicate that S-DCG CAPTCHAs offer some level of resistance to relay attacks, differentiating them from other CAPTCHAs.

(b) We formalized, designed and implemented stream relay attack against S-DCG CAPTCHAs, whereby the game is streamed between the attacker's machine and remote solver's machine. We performed a user study to measure the performance of the streamed version of S-DCG CAPTCHAs when solved by remote users (serving as human-solvers in the relay attack). Our study captured three realistic attack settings: low-speed high-latency channel between attacker and solver, high-speed low-latency channel between attacker and solver, and reduced game size. The results showed 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.

(c) Based on the data collected from the usability and stream relay user studies, we designed and evaluated 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).

4. We assessed the security of S-DCG against automated attacks:

(a) We evaluated the security of the developed S-DCG instances against random attack with various types of prior knowledge. Our results show that the S-DCG CAPTCHA is resilient to random attack when the attacker does not have any prior knowledge about the games. However, S-DCG CAPTCHA is vulnerable to random attack if the attacker is able to collect enough knowledge about the games (i.e.. the random attack success rate is up to 50% when the attacker extracts the moving and target objects).

(b) We further developed a novel, fully automated framework to attack these S-DCG CAPTCHA instances based on image processing techniques and principles of unsupervised learning. The attack is computationally efficient and highly accurate, but requires building a dictionary to be effective.

5. We designed and implemented a hybrid attack against S-DCG that combine the strengths of automated and relay attacks and overcomes the limitations of both attacks and evaluated the performance and the usability of the proposed attack.

6. We designed enhanced DCG CAPTCHAs with incremental difficulties against automated attacks. Starting by adding random noises to the S-DCG CAPTCHA, moving background, varying the colors of the moving objects and finally by implementing DCG's based on the notion of Emergent Images (EI-DCG). We evaluated the usability of EI-DCG as well as assessed its security against stream relay attacks. Our study shows that EI-DCG is secure against automated and relay attack, however it has low level of usability.

7. We designed novel ways to integrate S-DCG CAPTCHAs with other categories of CAPTCHAs based on hard AI problems, namely, image orientation-based CAPTCHAs [1] and semantic CAPTCHAs [2]. We evaluated the usability as well as assessed the security of the designed CAPTCHAs against automated attacks and stream relay attacks. The results of our study show that such integration enhance S-DCG security against random-guessing, dictionary-based and shape matching attacks, as well as relay attacks (due to the increased level of interaction), while providing the similar level of usability as S-DCG CAPTCHAs.

1.3.2 Human-Human Authentication

We developed a powerful framework *SMASheD* ("Sniffing and Manipulating Android Sensor Data") that can compromise the security of all well known authentication systems. Then, we attempt to tackle the limitations with existing user authentication mechanisms by introducing, building and studying *Gametrics* ("Game-based biometrics"), an authentication mechanism based on the unique way the user solves such DCG challenges captured by multiple features related to the user's cognitive abilities and mouse dynamics. The main contributions of this research in the topic of Human-Machine authentication are listed as follows.

8. We designed and developed the *SMASheD* framework to sniff and manipulate many restricted Android sensors, and evaluated its effectiveness on multiple Android devices, including phones, watches and glasses.

9. As a significant offensive implication of the *SMASheD* framework, we introduced a broad array of potentially devastating attacks on various user authentication systems.

10. We designed and implemented a *Gametrics* system based on simple DCGs to capture the unique user interactions. Our system is built using machine learning techniques and extracts a total of 64 features from each game challenge solving instance that capture the multiple unique cognitive abilities and the mouse dynamics of the users.

11. We collected a comprehensive data set from a total of 118 users, and showed that *Gametrics* can identify the legitimate users and the zero-effort attackers ("different users") with a high accuracy within a short period of time.

12. We showed that *Gametrics* can thwart active attackers that deliberately attempt to mimic a user's interaction with the challenges in an observation-based attack . Furthermore, we argue that attacking *Gametrics* using automated mechanisms, internal (i.e., *SMASheD*) or external, is also a hard task.

1.4 List of Publications

1. **A Three-Way Investigation of a Game-CAPTCHA: Automated Attacks, Relay Attacks and Usability**. Manar Mohamed, Niharika Sachdeva, Michael Georgescu, Song Gao, Nitesh Saxena, Chengcui Zhang, Ponnurangam Kumaraguru, Paul C. Van Oorschot and Wei-Bang Chen. In ACM Symposium on Information, Computer and Communications Security (AsiaCCS), June 2014.
   This paper covers contributions: 1, 2, 3(a) and 4.

2. **Dynamic Cognitive Game CAPTCHA Usability and Detection of Streaming-Based Farming**. Manar Mohamed, Song Gao, Nitesh Saxena and Chengcui Zhang. In the Workshop on Usable Security (USEC), co-located with NDSS, February 2014.
   This paper covers contributions: 2 and 3(b).

3. **Defeating a Game CAPTCHA with Efficient and Robust Hybrid Attacks**. Song Gao, Manar Mohamed, Nitesh Saxena and Chengcui Zhang. In Security and Forensics Track, IEEE International Conference on Multimedia and Expo (ICME), July 2014.
   This paper covers contributions: 5.

4. **On the Security and Usability of Dynamic Cognitive Game CAPTCHAs**. Manar Mohamed, Song Gao, Niharika Sachdeva, Nitesh Saxena, Chengcui Zhang, Ponnurangam Kumaraguru and Paul C. Van Oorschot. Under submission.
   This paper covers contributions: 1, 2, 3, 4 and 5.

5. **Emerging Image Game Captchas for Resisting Automated and Human-Solver Relay Attacks**. Song Gao, Manar Mohamed, Nitesh Saxena and Chengcui Zhang. In Annual Computer Security Applications Conference (ACSAC), December 2015.
   This paper covers contribution: 6.

6. **Emerging-Image Motion CAPTCHAs: Vulnerabilities of Existing Designs, and Countermeasures**. Song Gao, Manar Mohamed, Nitesh Saxena and Chengcui Zhang. Under submission.
   This paper explores the security of a static Emerging Image CAPTCHA [13, 14].[1]

7. **CAPTCHA Fusion: Mixing Games, Image Orientation and Semantics to Defeat Automated and Human Attacks**. Manar Mohamed and Nitesh Saxena. Under submission.
   This paper covers contribution: 7.

8. **SMASheD: Sniffing and Manipulating Android Sensor Data**. Manar Mohamed, Babins Shrestha and Nitesh Saxena. In ACM Conference on Data and Application Security and Privacy (CODASPY), March 2016.
   This paper covers contributions: 8 and 9.

9. **SMASheD: Sniffing and Manipulating Android Sensor Data for Offensive Purposes**. Manar Mohamed, Babins Shrestha and Nitesh Saxena. Under submission.
   This paper covers contributions: 8 and 9.

10. **Gametrics: Strong Behavioral Authentication with Simple Cognitive Games**. Manar Mohamed and Nitesh Saxena. Under submission.
    This paper covers contributions: 10, 11 and 12.

11. **Slogger: Smashing Motion-based Touchstroke Logging with Transparent System Noise**. Prakash Shrestha, Manar Mohamed and Nitesh Saxena. ACM Conference on Wireless Network Security (WiSec), July 2016.

    This paper explores defensive uses of *SMASheD*. [1]

12. **Curbing Mobile Malware based on User-Transparent Hand Movements**. Babins Shrestha, Manar Mohamed, Anders Borg, Nitesh Saxena and Sandeep Tamrakar. In IEEE International Conference on Pervasive Computing and Communications (PerCom), March 2015.

    This paper is relevant to interactive biometrics for extracting behavioral biometrics from various phone's sensors.[1]

Organization: The remainder of this dissertation is organized as follows. In Chapter 2, we present the preliminary works related to our work. In Chapter 3, we present S-DCG design, and the security as well as the usability evaluation of developed S-DCG instances. In Chapter 4, we present our work in designing various DCG variants with incremental level of difficulties against automated attack, followed by the usability and the security analysis of the final version we proposed. In Chapter 5, we present Mix DCG, a DCG variant that is created by integrating DCG with other CAPTCHA categories, design, usability and security evaluation. Chapter 6 presents *SMASheD*, our developed framework that can subvert the security provided by current authentication systems. Chapter 7 presents game-based biometrics design, implementation and evaluation under benign and active attack settings. Finally Chapter 8 concludes this dissertation and discusses the limitations and possible future extensions to this research.

[1] This paper is not a part of this dissertation.

CHAPTER 2

BACKGROUND

2.1 Human-Machine Authentication

The term CAPTCHA was first introduced in 2000 [6], 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, 15].

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 [16], Baffle [17], handwritten [18] and PayPal, or within a video such as NuCaptcha and emergent CAPTCHA [13]. CAPTCHAs that challenge the users to recognize or classify objects in images, such as collage CAPTCHA [19], implicit CAPTCHA [20], Bongo, Asirra CAPTCHA [21], PIX, ESP-PIX [22] and What's Up CAPTCHA [1], have also been proposed. Some video-based CAPTCHAs, such as content-based tagging of YouTube videos [23], and audio based CAPTCHA, such as Google, eBay, Yahoo, ReCaptcha, Slashdot and Math-function [24] 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 [7, 8]. Challenges based on spoken words suffer from similar issues due to sound distortion [25]. 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 [26]. These services offer

two categories of attacks: *automated attacks* and *relay attacks*. Automated attacks (e.g., [27–29]) normally utilize image processing algorithms to solve the CAPTCHA, while relay attacks [26] 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 [26]. 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 [26]. 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 [26]. 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 [26].

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 [26] 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 [13] 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.

TABLE 2.1: 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

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 2.1 provides a summary of how different categories and instances of existing CAPTCHAs can be subjected to a relay attack.

2.2 Human-Human Authentication

None of the current authentication primitives provide a good balance among the security, usability and efficiency of authentication. Almost universally deployed text-based passwords and PINs are highly efficient, but are either difficult to use (if the password is long and random), or insecure (if the users can choose their own password) [30–32].

Graphical passwords were founded on a psychological principle that the human brain has

superior memory for processing visual rather than textual information (see two excellent surveys [33, 34]). They can be based on recognition, such as those involving Random Arts images [35], objects (PassObjects) [36] and faces (PassFaces) [37], as well as on recall or cued recall, such as those involving drawings [38, 39] and selection of points on an image (PassPoints) [40]. Although preliminary evaluation of graphical passwords suggests that they can be more difficult to break compared to text passwords, using traditional methods such as brute force search or dictionary attacks [33], they can still be riddled with efficiency, memorability and predictability problems. For example, PassFaces [37] login process has been found to be lengthy [41]. Moreover, user-selected PassFaces can be predictable and could be successfully guessed [42]. Similarly, drawings-based passwords [38] may also be predictable, such as involving symmetric drawings with few pen strokes [43]. PassPoints [40] users also have a tendency to select passwords containing popular points (hotspots) or to follow simple patterns [44–47].

Biometrics, e.g., fingerprints [48], are a promising solution, but their universal deployment raises considerable challenges. First, maintaining biometric information databases necessitates users' entrusting a third party with extremely private (physical) information, and demands careful protection of the database itself, because biometric information cannot be revoked easily. Most biometrics also require authentication terminals to be equipped with specialized hardware, which may not be commonly available. Additionally, many biometric systems perform poorly in practical deployments, exhibiting unacceptably high false reject or false accept rates. For instance, voice authentication has significant error rates in noisy environments and basic fingerprint readers can be defeated by fake fingerprints [49]. Another problem is that it may be difficult to capture the biometric data for some users, resulting in a high enrollment failure rate [50, 51].

Perhaps more seriously, many existing biometric designs fail to take the role of users into account, causing them to suffer from severe usability issues. Many individuals consider biometrics to be invasive or "scary". Because they are hard to use and understand, most users find biometrics difficult to accept psychologically [51, 52]. For instance, users often express reluctance at using public fingerprint scanning devices. Notably, they have shown concerns that fingerprint scanners present at border checkpoints could facilitate the spread of contagious diseases (see, for instance, Nobel Prize winner Barry Marshall's blog [53]). There also exists something of a tradeoff between the usability and robustness of existing biometric technology, since the most

accurate forms of biometric authentication are often the most troublesome to operate [51, 54].

Authentication tokens, albeit another promising solution, are very often limited to a single-purpose use, e.g., paying highway tolls, and require users to carry and have access to additional devices.

The behavioral biometrics is used to identify the user based on a specific user's behavior while she performing a task. In most of the use cases of behavioral biometrics, the system measures the user behavior transparently, such as while the user is entering the password to serve as a second factor for authenticating the user based on her password as well as her typing pattern, or while the user interacting with the device/system after the login and throughout the session to continuously check that the legitimate user is the one interacting with the system.

The main aim of behavioral biometrics is to solve the problems associated with the traditional authentication systems, such as password leakage or sharing, requirements for extra hardware in case of traditional biometrics (for example, fingerprint readers or iris scanners). However, most of the proposed behavioral biometrics suffers from various problems, such as low-level of uniqueness among the users, which yields to high acceptance rate of illegitimate users (high False Positive Rate). Moreover, some of the behavioral biometrics require long interaction time to identify/recognize the users (up to 20 minutes [55]), which would allow attackers to interact with the system and may cause harm to the system during that period of time without getting detected.

The most studied approaches for behavioral biometrics are keystroke analysis and mouse dynamics. Keystroke biometrics identifies the user based on her typing characteristics. The verification is performed either based on static text (i.e., password) or random text (i.e., free text to continuously authenticate the users [56]). The features that are mostly used are the timing information of key down/hold/up events, time between the release of a key and the pressing on the next key, overall typing speed, and frequency of errors [57]. Mouse dynamics [58] is another most studied behavioral biometrics. It is mostly used for continuous authentication by recording the user interaction with the device transparently. The user is authenticated based on the general movement, drag and drop, stillness, point and click.

Other recent research studied user authentication based on user's cognitive abilities [59]. In this work, the authors studied the ability of authenticating the users based on their cognitive

process captured by visual search, working memory and priming effect on automatic processing. The game they utilized to capture the users' cognitive abilities provides a challenge-response task. In each instance of the challenge-response, the user is given a challenge, which is an object. The user's task is to drag the challenge object onto the matching object inside the search set. After a valid drop, the user then receives a gold coin as a reward and deposits it in a bank. On a correct deposit, the user is challenged with a new object and the game continues as before. From the interaction with the challenges, the authors extracted several features that captures the cognitive abilities of the users, however, they did not look into the mouse dynamics biometrics of the users.

The authors in [55] proposed a method to solve account hijacking and share problems in an online gaming environment. They propose identifying the user based on her gameplay activities. They show that the idle time distribution is a representative feature of game players. They propose the relative entropy test RET scheme, which is based on the Kullback-Leibler divergence between idle time (i.e., the idle periods between successive moves of a player controlled character) distributions, for user identification. Their evaluations shows that the RET scheme achieves higher than 90% accuracy with a 20-minute detection time given a 200-minute history size.

2.3 Threat Model and Design Choices

The core objective of this dissertation is to improve the usability and the security of Human-Machine and Human-Human authentication process.

In the context of Human-Machine authentication, our aim is to design and develop interactive CAPTCHAs that possess the following properties:

1. **Usability:** The user should be able to solve the proposed CAPTCHA within a short time and with high accuracy.

2. **Security against Automated Attacks**: A bot (automated computer program) must only be able to solve the CAPTCHA challenges with no better than a negligible probability. The automated attacks, we aim to defend in our study.

(a) **Random Guessing Attacks:** The simplest form of automated attack, where the bot tries to solve the CAPTCHA randomly.

(b) **Image-processing Dictionary-based Attacks:** The bot utilizes image processing techniques to learn the knowledge about solving the CAPTCHA challenges, stores the collected knowledge in a dictionary, and later uses the dictionary to attack new challenges.

(c) **Client-Side attacks:** The attacker reverses engineer the challenge code in order to extract information about the solution of the challenge.

3. **Security against Human-Solver Relay Attacks**: we aim to provide security against human-solver relay attacks, either by making it impossible to relay the CAPTCHA or by making it possible to detect such attack with high accuracy.

In the context of Human-Human authentication, our aim is to design and develop an interactive behavioral biometrics system that possesses the following properties:

1. **Usability:** The user has to be identified within a short time and with high accuracy.

2. **Security against Zero-Effort Attacks**: Any biometrics scheme should be able to distinguish between different users. That is, one user (potentially an attacker) should not be able to log in as another user (a victim).

3. **Security against Shoulder-Surfing Attacks**: An external attacker who monitors the user while she is authenticating herself to the system, should not be able to mimic and impersonate the user at a later point of time.

4. **Security against Automated Attacks:** We aim to provide security against sophisticated attacks where the attacker steals a user's authentication template (e.g., by hacking into the device or server that stores this template) and tries to authenticate itself in an automated manner to the system.

5. **Security against Internal Attacks:** We aim to provide security against internal attacks, such as a malware residing on the authentication terminal itself that records the user's valid authentication token/template and replays it later, or tries to learn the template by recording one or multiple valid authentication sessions and then creates an authentication token to authenticate itself as the user. Other forms of behavioral biometrics schemes have been shown to be vulnerable to such attacks [60].

CHAPTER 3

SIMPLE-DYNAMIC COGNITIVE GAME CAPTCHA (S-DCG)

As mentioned in Section 2.1, most existing CAPTCHAs suffer from usability and security problems. 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 chapter, we focus on a broad form of game CAPTCHAs, called *Simple Dynamic Cognitive Game (S-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 S-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" [61] has recently been offering such S-DCG CAPTCHAs.

Besides promising to significantly improve user experience, S-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 [62], while touch screen games are much easier and already popular. Motivated by these unique advantages of S-DCG CAPTCHAs, we set out to investigate their security and usability. Specifically, we pursue a comprehensive study of S-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*.

Chapter Organization: Section 3.1 elaborates on the design and implementation of our S-DCG instantiation. Section 3.2 presents the three usability studies we conducted to evaluate the usability of S-DCG. Two of these are PC-based studies, one "in-lab" and one "online"' using a crowdsourcing platform. The third one is a in-lab mobile-based study. Section 3.3 presents the security analysis of S-DCG against multiple forms of random attack and then describes our novel, fully automated framework to attack these S-DCG CAPTCHA instances based on image processing techniques. Section 3.4 evaluates the security of S-DCG against two types of relay

attacks (static and stream relay attacks) and proposes a viable relay attack detection mechanism. Section 3.5 elaborates on a novel and powerful hybrid attack framework that carefully combines the strength of automated algorithms and relay attacks, and overcomes the limitations of each and then evaluates the performance and the usability of the proposed attack. Finally, Section 3.6 concludes our findings about S-DCG.

3.1 S-DCG Design & Implementation

We use the term Simple Dynamic Cognitive Game (S-DCG) CAPTCHA to define the broad CAPTCHA schemes that form the focus of this chapter. We characterize a S-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 S-DCG, and present the S-DCG categories and associated instances.

3.1.1 S-DCG Design Choices

A pre-requisite for the security of a S-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 S-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 S-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 S-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 [63], 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 model where the attacker has only partial control over the client machine has also been employed in prior work [64].

In our model, we assume that the implementation provides continuous feedback to the user as to whether the objects dragged and dropped to specific target region(s) correspond to correct answers or not. The server also indicates when the game successfully finishes, or times out. This feedback mechanism is essential from the usability perspective otherwise the users may get confused during the solving process. The attacker is free to utilize all of this feedback in attempting to solve the challenges, but within the time-out. We also assume that it is possible for the server to preclude brute force attacks, such as when the attacker tries to drag and drop the regions within the image exhaustively/repeatedly so as to complete the game successfully. Such a detection is possible by simply capping the number of drag/drop attempts per moving object.[1]

3.1.2 S-DCG CAPTCHA Instances and Prototypes

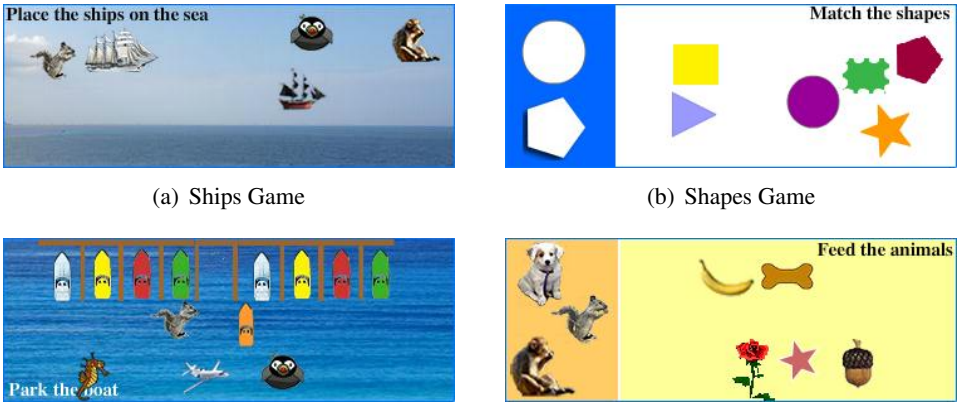
Due to the legal restrictions on attacking commercial S-DCG CAPTCHAs, we proceeded to develop our own animation-based S-DCG prototypes for the purpose of our study. Using Adobe Flash, we implemented four CAPTCHA games that represented a broad class of S-DCGs. These games are 360×130 pixels in size, and seamlessly fit into web pages if used for practical purposes.

The S-DCG CAPTCHAs implemented for the purpose of this study are shown in Figure 3.1. Each S-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 3.1(c), the orange boat, that can be dragged to the empty dock position to complete the game, is the answer object.

[1] The "are you a human" S-DCG CAPTCHA implementation claims to adopt a sophisticated (proprietary) mechanism, based on mouse events, to differentiate human game playing activity from an automated activity. We did not implement such a human-vs-bot behavioral analysis component because our goal is to examine the underlying CAPTCHA scheme only. A behavioral component can be added to other CAPTCHAs also and represents a topic orthogonal to our work. Besides, it is not clear if behavioral analysis would add security; it may instead degrade usability by increasing false negatives.

- *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

FIGURE 3.1: Static Snapshots of 4 Game Instances of a Representative S-DCG CAPTCHA (Targets are Static; Objects are Mobile).

S-DCG CAPTCHAs are classified according to the number of target objects. The Ships game (Figure 3.1(a)), is a one-target S-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 3.1(b)) has a circle and a pentagon placed on the left side of the game as the two target objects. The Animal game (Figure 3.1(d)) is a three-target instance of S-DCG CAPTCHA. The Parking game (Figure 3.1(c)) is a variant of S-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 S-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.

3.2 Usability

In this section, we report our usability studies of our representative S-DCG CAPTCHA category. The first study is student based which we used to measure the usability of the S-DCG CAPTCHA with respect to different parameters as explained in Section 3.1.2. 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 S-DCG CAPTCHAs on mobile devices.

3.2.1 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 3.1). The participants were provided with 20 instances as discussed in Section 3.1.2 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 S-DCG CAPTCHAs was recorded using a survey form. The survey contains the 10 System Usable Scale (SUS) standard questions [65], each with 5 possible responses (5-point Likert scale, where 1 represents strongly disagreement and 5 represents strongly agreement).

TABLE 3.1: Demographics of Participants in the *Usability*, *Relay Attacks* and *Hybrid Attack* Studies

	Usability			**Stream Relay Attack**			**Hybrid Attack**
	Lab	MTurk	Mobile	LSHL		HSLL	
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 and paid each of them \$0.5 for their effort. Each was asked to do similar tasks as of the first study. However, they had only to play only four games (40 FPS, 6 objects) variant of each game instance. The third column of Table 3.1 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 S-DCG usability.

The **third usability study** aims to assess the usability of S-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 3.1). The participants were asked to performed similar task as the MTurk worker but using Mobile device.

Via our study, our goal was to assess the following aspects of the S-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.

3.2.2 Study Results

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

3.2.2.1 Lab-based Usability Study

TABLE 3.2: Drag Error Rates and Completion Time *Lab Usability Study* (Overall Error Rate=0)

Game Type	**Completion Time(s)** *mean (std)*	**Drag Error Rate** *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

Completion Time: Table 3.2 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 ($p = 0.01$).

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 S-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

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

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 3.2. 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.

TABLE 3.3: Drag Error Rates and Completion Time per Object Speeds (Overall Error Rate=0)

Object Speed	**Completion Time (s)** *mean (std)*	**Drag Error Rate** *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

Effect of Object Speed and Number: Table 3.3 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 ($p < 0.001$).

TABLE 3.4: Drag Error Rates and Completion Time per # of Objects (Overall Error Rate=0)

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

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 3.4 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 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 ($p < 0.001$).

3.2.2.2 MTurk Usability Study

TABLE 3.5: Drag Error Rates, Game Error Rate and Completion Time *MTurk Usability Study*

Game Type	**Completion Time (s)** *mean (std)*	**Error Rate** *mean*	**Drag Error Rate** *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 3.5 summarized the results of MTurk usability study. The second column of Table 3.5 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 third column of Table 3.5 shows that the average overall error rate increases from 0% to 3.5% and the average drag error rate increases from 5% to 32% compared to their correspondent in the Lab-based usability study (Table 3.2). 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.

3.2.2.3 Mobile-based Usability Study

TABLE 3.6: 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

Table 3.6 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 3.6 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.

3.2.2.4 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 [66], the usability of our S-DCG game CAPTCHA instances can be rated as high.

3.2.3 Summary of Usability Analysis:

Our results suggest that the S-DCG CAPTCHA representatives tested in this study offer very good usability, resulting in short completion times (less than 15s) on average, very low error rates (0 - 3.5% per game completion)[3], and good user ratings. We found that increasing the object speed and number is likely to degrade the game performance, but up to 6 objects and 40

[3]When contrasted with many traditional CAPTCHAs [8], these timings are comparable but the accuracies are better.

FPS yield a good level of usability. S-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 [62].

3.3 Automated Attacks

Having validated, via our usability study, that it is quite easy for the human users to play our S-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 S-DCG CAPTCHA challenges based on image processing techniques and principles of unsupervised learning. We start by considering random guessing attacks and then demonstrate that our framework performs orders of magnitude better than the random guessing attacks.

3.3.1 Random Guessing Attack

An attacker faced with a S-DCG CAPTCHA challenge can always attempt to perform a random guessing attack based on his prior knowledge about the challenge.

Unknown probable answer objects, unknown target objects: The attacker knows the location of the target area (e.g., the blue region containing the target circle and pentagon in the Shapes game) and the moving object area (e.g., the white region in the Shapes game within which the objects move). However, the attacker (bot) does not have any knowledge of: (1) the moving objects and (2) the target objects. A randomized strategy that the attacker could adopt is to pick a random location on the moving object area and drag/drop it to a random location on the target area. The probability of success will equals

$$ao! \times \begin{pmatrix} a \\ ao \end{pmatrix} \times (\frac{ao_a}{ma_a} * \frac{to_a}{ta_a})^{ao}, \tag{3.1}$$

where ao is the number of answer objects, a is the number of allowed attempts, ao_a the average area of the answer objects, ma_a the area of the moving area, to_a is that average area of the target objects, and ta_a is area of the target area. The success probabilities of random attack for the parking, ships, shape and animals are 0.008, 0.008, 0.001 and 0.00002, respectively when the attacker is allowed 2 extra attempts. These probabilities are much lower than the target

probabilities for a real-world CAPTCHA system security (e.g., 0.6% as suggested by Zhu et al. [67]). Therefore, S-DCG is secure against this form of random attack.

Unknown probable answer objects, known target objects: The attacker knows the location of the target objects, but still does not have knowledge about the answer objects. The attacker can pick a random point on the moving objects and drag/drop it to a randomly picked target object. The probability of success can be calculated as

$$ao! \times \binom{a}{ao} \times (\frac{ao_a}{ma_a} * \frac{1}{to})^{ao}, \tag{3.2}$$

where to is the number of targets. The success probabilities of random attack for the parking, ships, shape and animals are 0.088, 0.008, 0.007 and 0.00008, respectively when the attacker is allowed 2 extra attempts. Only animals game is secure against this random attack.

Known probable answer objects, known target objects: The attacker knows the target objects and the moving objects location, however, he does not know the relationship between the objects and the targets. The attacker picks a random object, drags it and drops it to a random target. The probability of the success of this random attack can be calculated as

$$ao! \times \binom{a}{ao} \times (\frac{1}{mo} * \frac{1}{to})^{ao}, \tag{3.3}$$

where mo is the number of moving objects. The success probabilities of random attack for the parking, ships, shape and animals are 0.5, 0.28, 0.08 and 0.01, respectively when the attacker is allowed 2 extra attempts and the number of moving objects = 6. S-DCG is not secure against this type of random attack.

3.3.2 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 [68], 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 [68], 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 S-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 (or consistent) 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[4], 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

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

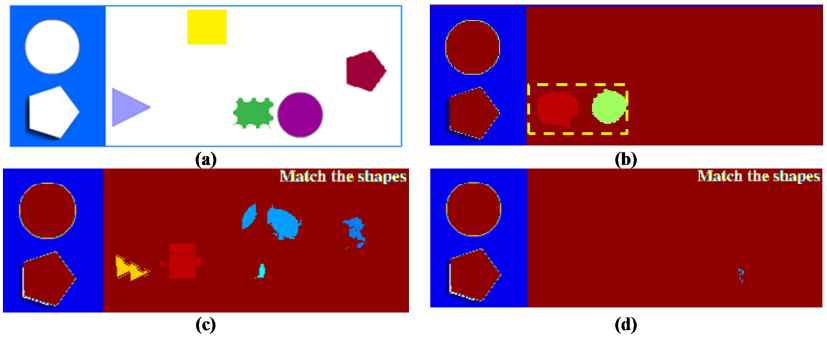


FIGURE 3.2: Detected Backgrounds. (a) Original Frame Image; (b) Detected Background with Pseudo Patches by Using Preliminary Method; (c) Detected Background with Pseudo Patches After Performing the First Step of the Proposed Method; (d) Detected Background with a Minor Patch After Performing the Second Step of the Proposed Method.

instead of by the background. A shadow of foreground objects may appear as pseudo patches in the background image as shown in Figure 3.2(b) for the Shapes game of Figure 3.2(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 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 3.2(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.

Each learned background image is saved in the database. 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.

According to our experimental results, the likelihood of observing such pseudo patches is sufficiently low ($< 7\%$). However, pseudo patches may not pose a big issue. Even though the existence of pseudo patches may result in over-segmented foreground objects when they overlap each other, a partially detected object can still be used to extract visual features and later to locate an object that matches the visual features at the time of attacking.

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)* [69] method and the *Edge-based method*, and compared and contrasted them with regard to detection accuracy and time efficiency.

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 3.3). 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 3.3(c)).

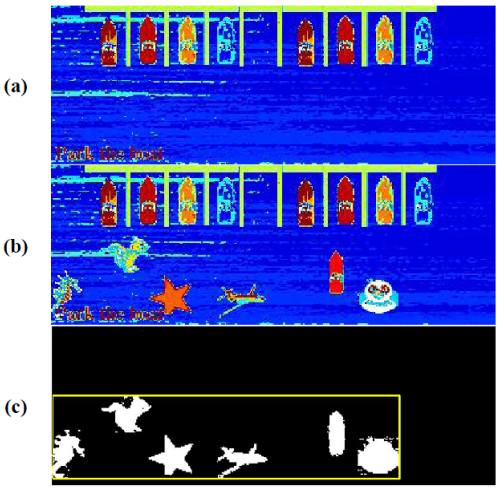


FIGURE 3.3: Target Detection. (a) The Detected Background for the *Parking* Challenge; (b) One Sample Frame Represented in Color Code; (c) Detected Foreground Objects from (b) and their MBR.

After the removal of the entire area bounded by MBR_{final} from the background image, the remaining background is divided into eight sub-areas. as shown in Figure 3.4. The largest sub-area with the largest area (e.g., sub-area #2 in Figure 3.4) 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 [5]

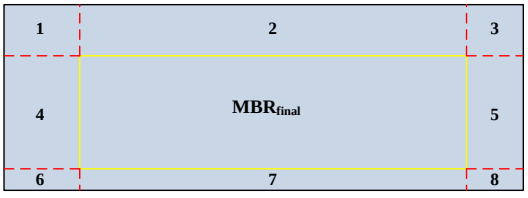


FIGURE 3.4: Eight Sub-Areas Generated According to Moving Area of Foreground Objects.

The Edge-based method employs a different design principle than MBR-based method. It is based on the hypothesis that there are strong edges in the target area because of the likely

[5] We also implemented an alternative design, called the *exclusion method* , which detects the target area by simply removing foreground object pixels accumulated from all the sample frames. However, while this method is slightly faster than the MBR-based method, it is less robust.

presence of objects in the target area, such as the dog and the squirrel in the Animals game. The steps involved in the edge-based method are listed below:

1. Collect a sequence of frames and learn the background image as in the MBR-based method.
2. Detect edge pixels on the background image. Group connected edge pixels into edge segments.
3. Remove trivial edge segments that have too few pixels by a user-input threshold.
4. The mean of all the centroids of remaining segments is used as the target area center.

The comparison results of the MBR-based and Edge-based methods are shown in Figure 3.5. The solid square dot in each game scene in Figure 3.5(a) is the MBR-detected target area center for that challenge. Also displayed in Figure 3.5(a) are the detected foreground object moving areas, namely MBR_{final}, displayed as a black rectangle in each game scene. According to our experimental results, MBR-based method was able to detect the correct target area center in all the challenges. In contrast, for the edge-based method, it is difficult to find a global threshold that works for all the challenges. Rather, we need to adjust the threshold for a specific game in order to achieve "reasonably good" results, and this method is also sensitive to the existence of texts in the background. Figure 3.5(b) shows the "optimal" edge detection result for each challenge with a manually tuned threshold which is different for each challenge. As shown in Figure 3.5(c), some target area centers are incorrectly detected because some edge segments belong to the texts that are part of the background but not of the target area. This means that the accuracy of the edge-based method could be significantly undermined by the presence of strong edges in the background that are not part of the target area (e.g., presence of texts) and the absence of objects in the target area (e.g., the absence of objects in the target area of the Ships Game). As for efficiency, the MBR-based method has a time complexity of $O(MN)$ where M is a constant in the range of 5-8, while the time complexity of the edge-based method is $O(NL+N^2)$ where L is a constant in the range of 3-8 estimated based on the typical time complexity of a non-combining edge detection method [70]. Overall, this shows that the MBR method outperforms the Edge method on several aspects.

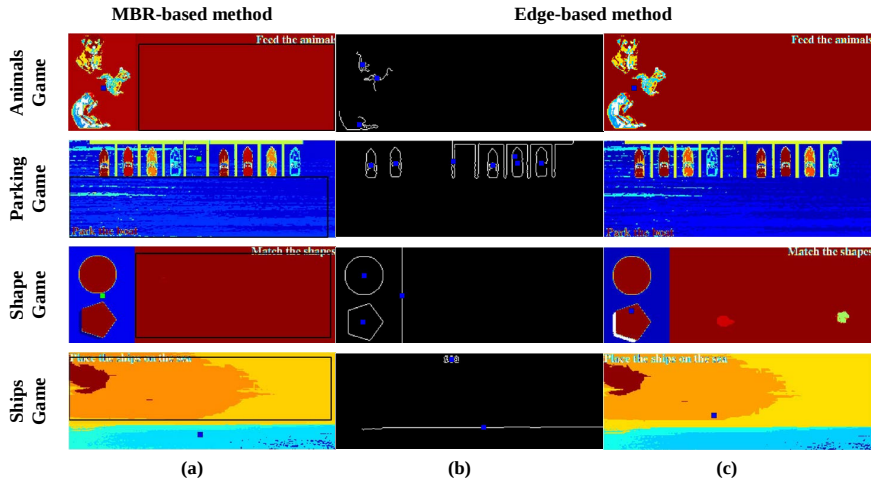


FIGURE 3.5: Comparison of the Target Area Center Detection Results between the MBR-Based and the Edge-Based Methods. (a) Results from MBR-Based Method (Solid Square Dot Represents the Target Area Center and Black Rectangle Represents the Object Moving Area); (b) Centroids of Non-Trivial Edges from the Edge-Based Method; and (v) Final Target Area Centers from the Edge-Based Method.

(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 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.

3.3.3 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 S-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 S-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). It is also important to note that, as per the findings reported in [71], the use of fully automated solving services represent economical hurdles for CAPTCHA attackers. This applies to traditional captchas as well as S-DCG CAPTCHA. Eventually, this may make automated attacks themselves less viable in practice [71], and further motivates the attacker, similar to other CAPTCHAs, to switch to human-solver attacks against S-DCG CAPTCHAs.

3.4 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 [71]. In this section, we assess S-DCG CAPTCHAs w.r.t. to relay attacks.

3.4.1 Difficulty of Relaying S-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. Similarly, video-based character recognition CAPTCHAs [13, 72] can be broken via a relay attack by taking a video of the incoming frames and relaying this video to the human-solver.

In contrast, S-DCG CAPTCHAs offer some level of resistance to relay attacks, as we argue in the rest of this section. In making this argument, we re-emphasize that the primary motivating factors for a human-solver relay attacker are simplicity, low economical cost and practicality. As such, a relay attack that requires sophistication (e.g., special software, complexity and overhead), is likely not viable in practice [71].

There appears to be a few mechanisms using which S-DCG captchas could potentially be subject 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 S-DCG CAPTCHA security model (Section 3.1.1), 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 [63], 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 S-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 on an experiment and the results of an underlying user study in order to evaluate the feasibility of Static Relay attack against our S-DCG CAPTCHA instances. This novel experiment takes the form of a *reaction time* or *reflex action* task for the human-solver. A reaction time task involves performing some operation as soon as a stimulus is provided. A common example is an athlete starting a race as quickly as a pistol is shot. The subject of reaction time has been extensively studied by psychologists (see Kosinski's survey [73]). Then, we report our *Stream Relay* attack study and show the ability of detecting such relay attack using our proposed machine learning detection method.

3.4.2 Reaction Time Static Relay Experiment

Our hypothesis is that S-DCG CAPTCHAs will be resistant to the *Static Relay* attack, and so we give the attacker a strong power in the following sense: our tests eliminate the communication delay between the bot and the human solver, by putting them on the same machine. The focus of the experiment then shifts towards motivating human-solvers to perform at their best by employing meaningful interfaces and by framing the underlying task in a way that is amenable to these solvers. In particular, since attacker's goal is to minimize the delay incurred by the human solver in responding to the challenges, we model human-solver attack as a reaction time [73] task described below. Our Section 3.4.2.2 study further facilitates the attacker with human solvers having low response times and quick reflex actions, such as youths in their 20s [73].

Experimental Steps: The reaction time Static Relay attack experiment consists of the following steps:

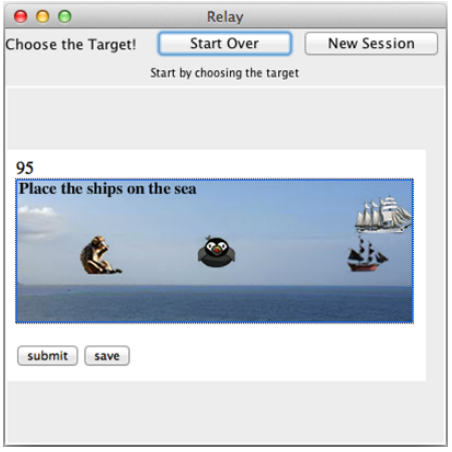
1. A snapshot of the game challenge is extracted by the bot (B), and the human solver (H) is asked to identify/mark a target object for that game challenge (e.g., the dog in the Shapes game).
2. For each target object identified above, H is asked to identify one answer object in the snapshot specific to the game challenge (e.g., bone for the dog in the Shapes game). However, since B wants to minimize the delay between the time the challenge snapshot is given and the response is received from H, a stimulus will be associated with the snapshot. We make use of a combination of (1) a visual stimulus (the border across the game window flashes in Red) and (2) an audio stimulus (a beeping sound). The task for H is to identify an answer object in the image *as soon as* the stimuli are provided.

3. B will emulate the dragging and dropping of the objects based on the response of H (simply use the pixel values provided by H as the coordinates of the objects and respective targets).

4. Steps 2-3 are repeated until all answer objects for a given target object are identified by H and dragged/dropped by B.

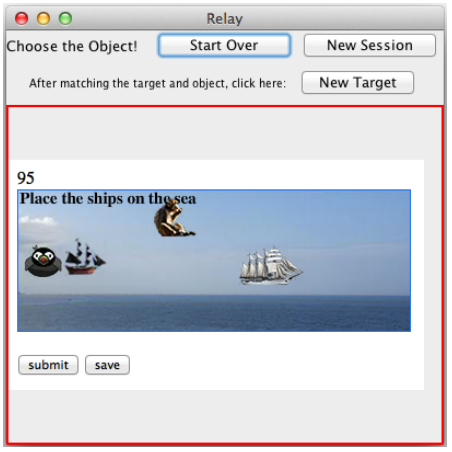
5. Step 1 is repeated until all target objects have been covered.

The experiment succeeds if the CAPTCHA game completes successfully, i.e., if all answer objects are dragged to their respective targets by B per input from H.



(a) The solver is asked to choose a target object

(b) The solver is asked to choose the next answer object, if any



(c) The solver is asked to select a new target object, if any

FIGURE 3.6: User Interface Implementing the Reaction Time Relay Experiment (95 Represents the User ID; The Red Rectangle in (c) Represents Our Visual Stimulus).

Experimental Implementation: Our implementation of the above experiment consists of a user interface (UI) developed in Java that interacts with the human solver and a bot. The core of this implementation is designed using an algorithm following which the screen captures are updated and displayed on the screen as well as an algorithm used to make the mouse drag and drop of the objects.

The game starts by the bot capturing an image of the game challenge from the browser (i.e., the CAPTCHA challenge that the bot received from the server) and displays that image in the UI. The solver is then asked to click on a target object within that image. After selecting the target, the solver is instructed to click a "Next" button, wait for a flashing and a beep (our stimuli), followed by clicking the object that matches with that target. Once the solver has clicked on the object, the bot takes control of the mouse by clicking and dragging the object to the target in the flash game. The solver must be able to identify and choose the correct object before the object has moved too far in the flash game displayed in the browser. Whether the click is successful or not, a new screen capture is retrieved from the game on the browser. If the solver has chosen the object in time on the UI, then he/she can pick a new target if one exists by clicking on the "New Target" button. If the solver has missed clicking on the object fast enough (i.e., if the click was not successful), the solver will automatically get another attempt to choose the correct object followed by the flashing and the beep. Figure 3.6 depicts the UI of our implementation.

3.4.2.1 Static Relay Attack User Study

We now report on a user study of the aforementioned reaction time relay attack experiment presented in Section 3.4.2. Similar to our usability study, we present the relay attack study design, goals, and testing process, as well as study results.

3.4.2.2 Study Design, Goals and Process

In the Static Relay attack study, users were given the task to play our 4 game instances through the UI (described above). The study comprised of **20 participants**, primarily Computer Science university students. This sample represents a near ideal scenario for an attacker, given that young people typically have fast reaction times [73], presumably optimizing the likelihood of the success of the relay attack. The demographics of the participants are shown in the third column

of Table 3.1. The study design was similar to the one used in our usability study (Section 3.2). It comprised of three phases. The *pre-study phase* involved registering and briefly explaining the participants about the protocols of the study and collecting the participant demographics. Then, the participants playing the games via our interface. The participants were told to perform at their best in playing the games. The *post-study phase* required the participants to fill a survey form about their experience.

Each participant was asked to play the relay versions corresponding to each of the 20 variations of the 4 DCG CAPTCHA games as in Section 3.1.2; we used ordering based on 4×4 Latin squares, as in the usability study. The specific goal of our study was to evaluate the reaction time experiment UI in terms of the following aspects:

1. *Efficiency*: time taken to complete the games.
2. *Robustness*: likelihood of not completing the game, and incorrect drag/drops.
3. *User Experience*: quantitative SUS ratings.
4. *Reaction time*: Time delay between the presentation of the stimuli and the response from the participant. This is a fundamental metric for the feasibility of the attack. If reaction time is large, the likelihood of attack success will be low.

Another important goal of our user study was to compare its performance with that of the usability study. If the two differ significantly, the relay attack can be detected based on this difference.

For each game, completion times and errors were automatically logged by our web-interface software. In addition, we maintained "local logs" of the clicks made by the participants on our game interface to measure the reaction timings.

3.4.2.3 Study Results

Completion Time and Error Rates: Table 3.7 shows the time taken and error rates to play the games for each game type by different participants. Unlike our usability study, many game instances timed out, i.e., the participants were not able to always complete the game instances within the time out of 60s. We reported in Table 3.7 only the average times that correspond to the DCGs that were completed successfully.

TABLE 3.7: Error Rates and Completion Time *Static Relay Attack*

Game Type	**Completion time(s)** *mean (std)*	**Error Rate** *mean*	**Drag Error Rate** *mean*
Ships	22.25 (5.04)	0.26	0.17
Animals	37.93 (4.91)	0.40	0.65
Parking	20.45 (5.04)	0.22	0.66
Shapes	22.94 (1.74)	0.09	0.56

All games turned out to be quite slow, and much slower than that of the usability study where the games lasted for less than 15s on an average (Section 3.2). As in our usability study, we found that users took longest to solve the Animals game (37.93s), whereas the other games took slightly less time. This might be due to the presence of 3 target objects. We observed that the error rates were the highest for the Animals game (40%), and the least for the Shapes games (9%) although the corresponding drag error rates were high (56%). The Ships and Parking games had comparable overall error rates between 20-30%. Analyzing the data using Mann-Whitney U test with Bonferroni correction, we found statistically significant difference between the mean time of each of the games and its correspondent in the Lab-based usability study: $p < 0.001$.

To analyze errors better, we investigated drag error rates, i.e. for each drag attempt whether the object being dragged was dropped at the correct position or not. The error rate per click was the least for the Ships game (17%), much lower compared to all other games (50-70%), the latter itself being much higher than observed during the usability study. This suggests that the server could prevent the relay attack against Animals, Parking and Shapes games by simply capping the number of drag/drop attempts.

Reaction Time: We now analyze the reaction time corresponding to different games during the relay attack experiments. We consider two types of reaction times, one corresponding to all clicks made by the participants, and the other corresponding to only the correct clicks (i.e., those that resulted in a correct drag/drop). The averaged results for the two types of reaction times for each game type are summarized in Table 3.8. We can see that the average reaction time (all clicks) for all game categories was more than 2s and the least for the Shapes game (2.17s). The average reaction time (correct clicks) is slightly lower than reaction time (all clicks), but still higher than 1.5s and still lowest for the Shapes game (1.62s). Neither types of reaction times

change significantly across different game categories.

TABLE 3.8: Reaction Times per Game Type

Game Type	**Reaction Time All Clicks(s)** *mean (std)*	**Reaction Time Correct Clicks(s)** *mean (std)*
Ships	2.27 (0.34)	2.06 (0.17)
Animals	2.58 (0.35)	1.85 (0.23)
Parking	2.50 (0.51)	2.00 (0.31)
Shapes	2.17 (0.20)	1.62 (0.11)

User Experience: The average SUS score from the study came out to be only 49.88 (standard deviation = 5.29). This is rather low given that average scores for commercial usable systems range from 60-70 [66], and suggests a poor usability of the system. This means that it would be difficult for human users to perform well at the relay attack task and implies that launching relay attacks against DCG CAPTCHAs can be quite challenging for an attacker. Comparing the SUS score between the lab-based usability study and static relay attack, using Mann-Whitney U test with Bonferroni correction, we found statistically significant difference ($p < 0.001$).

Summary of Static Relay Analysis: All of our analyses above suggest that perpetrating a successful Static Relay attack against the DCG CAPTCHA pose significant challenges. For the attack to succeed, the human-solver needs to perform a reaction time task (average reaction time is more than 2 seconds). That results in increasing the completion time to more than 20 seconds in average, and significantly increasing of the error (error rates more than 20%; per drag error rates more than 50%). Moreover, users found it a hard task (average SUS less than 50).

In real life, where the communication delays between the bot and solver's machine will be non-zero (unlike our attack set-up), launching a real attack would be even more difficult. Participant feedback indicates that the human-solver performance may be improved with training, but this may increase the economical costs for the attacker.

3.4.3 Stream Relay Attack

Under Stream Relay, the attacker obtains the S-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 S-DCG CAPTCHA frames to the human-solver and sending the human-solver's mouse interactions, such as drag/drop, mouse

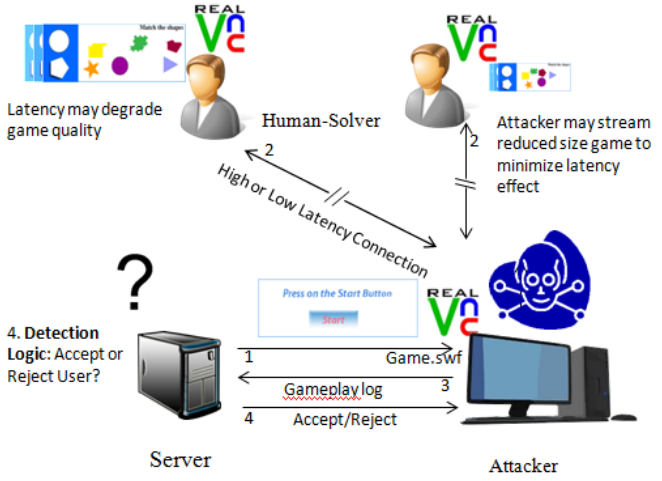


FIGURE 3.7: Human-Solver Gameplay in a *Stream Relay* Attack.

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. The Stream Relay attack flow diagram is shown in Figure 3.7. 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 S-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.

3.4.4 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 [74].

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 [74].

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 [74].

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 [74].

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 [74].

In our study, we utilize RealVNC [75] 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.

3.4.4.1 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. 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, the demographics of the participants are shown in the columns 5 to 7 of Table 3.1, as described below:

1. *Low-Speed High-Latency (LSHL)*: The first scenario involved collecting data from participants residing in a developing country (India), where we expect the users to have a slow Internet connection and they reside on far proximity of the attacker (residing in the USA). 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.

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 $180\times$ 65. To evaluate this scenario, we collected data from 20 participants residing in a developing countries (same as in the previous study).

3. *High-Speed Low-Latency (HSLL)*: In the last scenario, we tested a setting in which the attacker recruits participants from a developed country (USA), where the users have fast Internet connection and they reside in near proximity of the attacker (USA). To evaluate this scenario, we collected data from 20 participants located within US.

3.4.4.2 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 results for the first, LSHL Relay, scenario are shown in first column of Table 3.9. The games played as part of this scenario took significantly longer than that performed with usability study. On average, we found that completing S-DCG CAPTCHAs with LSHL 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 LSHL 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 LSHL Relay attack compared to

TABLE 3.9: Completion Times, and Error Rates in *Stream Relay* Attack Scenarios

	Low-Speed High-Latency (LSHL) Relay			***Small Game Relay***			***High-Speed Low-Latency (HSLL) 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

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 180$\times$ 65 pixels. However, the results, as shown in the second column of Table 3.9, 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, HSLL Relay, tested the Stream Relay attack performance when the attacker and the solver reside relatively nearby (both within the US) and where the solvers have high speed internet connection. The results of this experiment are depicted in the third column of Table 3.9. 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 LSHL 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 HSLL 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 LSHL Relay ($p < 0.0001$) and between usability study and Small Game Relay ($p = 0.004$). HSLL Relay did not turn out to be significantly different from the usability study statistically in terms of user experience.

3.4.4.3 Stream Relay Attack Detection

In the previous section, we have demonstrated that the S-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,$\cdots$).

- *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*.

TABLE 3.10: Class Distribution of Non-Timeout Users

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

Both Support Vector Machine (SVM) [76] and K-Nearest Neighbors (KNN) [77] 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 in order to find the best subset of features and classifier that give the best results.

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 3.10.1 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:

- *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 3.4, 3.5 and 3.6. 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.

$$Precision = TP/(TP + FP) \tag{3.4}$$

$$Recall = TP/(TP + FN) \tag{3.5}$$

$$Accuracy = (TP + TN)/(TP + FP + TN + FN) \tag{3.6}$$

TABLE 3.11: Results of Using the Optimal Feature Subset for Each Game in the Classification of *Legitimate User* and *LSHL* 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 3.12: Results of Using the Common Optimal General Feature Subset for All Games in the Classification of *Legitimate User* and *LSHL* 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**

In the **first classification experiment**, we build a classifier for each of the four game types to distinguish a legitimate user from a LSHL relay attacker (i.e., corresponding to the first scenario of the Stream Relay attack). The average measurement values, as shown in Table 3.11, are calculated from running a 10-fold cross validation 5 times for each test case. The results show that LSHL 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 3.12 indicate

TABLE 3.13: 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 3.14: 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

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 3.11.

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 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 3.13. 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.

TABLE 3.15: 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 3.16: 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**

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 3.14. 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 3.15) when SVM.

Then, we build a model to distinguish between legitimate user and Small Game Relay attacker. The obtained results are shown in Table 3.16. 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.

For the last dataset, using the data collected from HSLL 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 HSLL relay attacker. The average measurement values, as shown in Table 3.17, are calculated from running a 10-fold cross validation 5 times for each test case. The results show that HSLL 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 LSHL Relay – on average 77% for

TABLE 3.17: Results of Using the Optimal Feature Subset for Each Game in the Classification of *Legitimate User* and *HSLL* 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 3.18: Results of Using the Optimal Feature Subset for Each Game in the Classification of *HSLL* 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

HSLL compared to 90% for LSHL. 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 HSLL 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 3.18. 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 3.19, 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 drag-and-drops improves the classification performance. A S-DCG CAPTCHA

TABLE 3.19: Results of Using the Optimal Feature Subset for Each Game in Classification of *Legitimate User* and *(LSHL, HSLL 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**

service may employ this global model without having to remove any of the four games from the game database.

3.5 Hybrid Attack

Relay attacks as shown in section 3.4 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.

3.5.1 Auto-attack with offline learning

Model I of our proposed hybrid attack framework (Figure 3.8), 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.

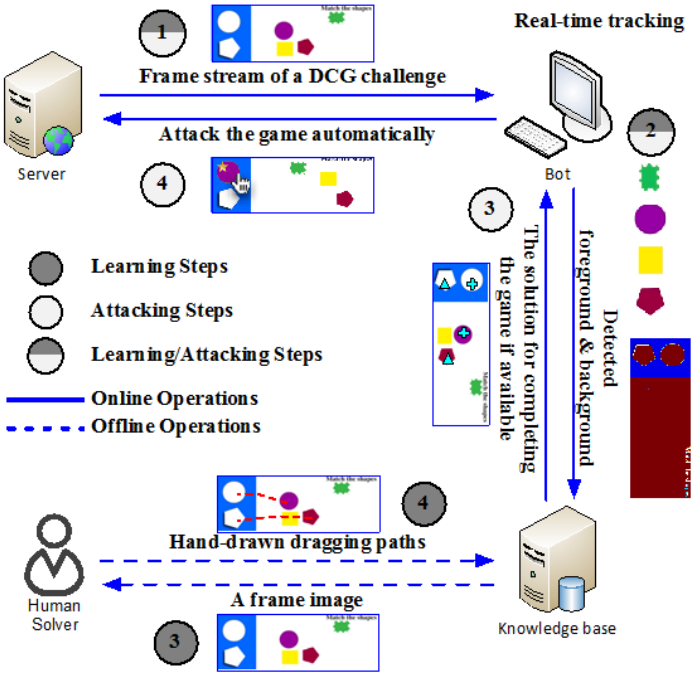


FIGURE 3.8: Hybrid Attack Model I: Auto-Attack with Offline Human Learning

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 3.8). 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 3.3.2. Moreover, the attacking phase is converted into offline learning just in case that brute-force based learning cannot work out the puzzle.

3.5.2 Auto-attack with online learning

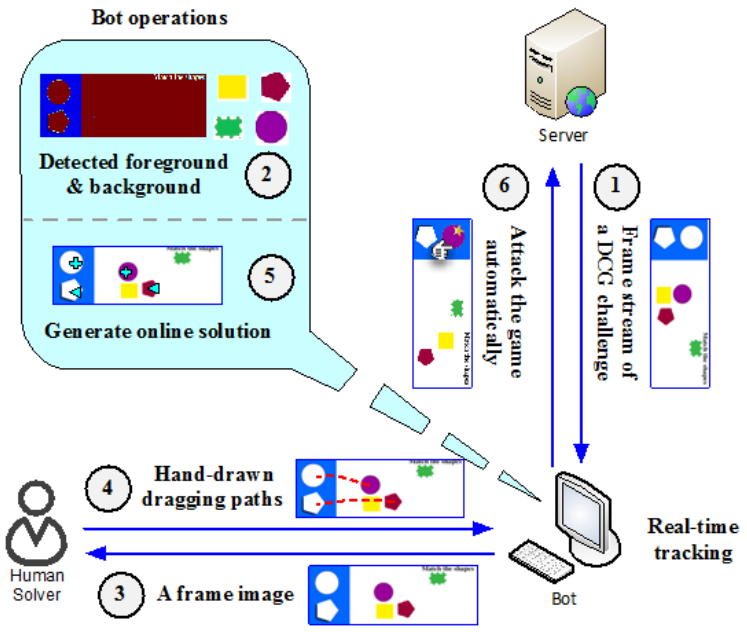


FIGURE 3.9: Hybrid Attack Model II: Auto-Attack with Online Human Learning

Model II of our hybrid attack framework (Figure 3.9), Auto-attack with Online human Learning (AOnL), attacks any game, seen or unseen, with the help of real-time tracking and online knowledge. Compared 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 3.9, 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 S-DCG CAPTCHAs).

3.5.3 Hybrid Attack Usability Study

3.5.3.1 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 S-DCG explained in Section 3.1.2 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 3.1.

3.5.3.2 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 3.20. 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

TABLE 3.20: 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

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 3.20. 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.

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 S-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 S-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.

3.6 Conclusions

In this chapter, we investigated the security and usability S-DCG captchas. Our overall findings are mixed. On the positive side, our results suggest that S-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 S-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 S-DCG CAPTCHAs could concentrate on making these captchas better resistant to automated attacks while maintaining a good level of usability.

CHAPTER 4

SECURITY ENHANCED DCG CAPTCHAs

The results obtained in the previous chapter show that S-DCG CAPTCHAs are highly usable, and resistant to human-solver relay attacks. However, S-DCG CAPTCHAs are rather weak when it comes to automated attacks and known probable answer objects, known target objects random attack. We explored various ways to enhance the security of S-DCG against automated attacks by adding countermeasures to the S-DCG. In this chapter, we outline our novel DCG CAPTCHA variations that aim to resist automated attacks besides relay attacks.

Chapter Organization: Section 4.1 reviews our preliminary designs of various DCG variants with enhanced security. Section 4.2 introduces Emerging Images based DCG CAPTCHA (EI-DCG), a secure DCG instantiation by using the notion of emerging images [78]. Section 4.3 elaborates on the design and implementation of our EI-DCG instantiation. Section 4.4 presents the security analysis of EI-DCG. Section 4.5 evaluates the usability of EI-DCG. Section 4.6 evaluates the security of EI-DCG against relay attack. Finally, Section 4.7 concludes our findings on EI-DCG.

4.1 Preliminary Work

One of the main weaknesses of the S-DCG CAPTCHA instances (Figure 3.1) is that the underlying game background is static, which is utilized by our attack framework (Section 3.3.2) to extract the foreground objects from the game frames thereby undermining the CAPTCHA security. To remedy this problem, we designed DCG CAPTCHAs variations that incorporate different forms of dynamic background, and analyze their security. The variations design involve random noises , objects with dynamically changing color and luminance, object occlusion, and dynamically changing (natural video) background, and combinations thereof, as outlined below (Figure 4.1 shows examples):

- *Noise:* Noise is added to the game in the form of random shapes (e.g., lines) that are moving randomly (Figure 4.1 (a), 4.1(b) and 4.1(c))).

(a) Noise

(b) Noise+Occlusion

(c) Animated Bg

(d) Snow Effect

FIGURE 4.1: The Proposed New Instances of DCG CAPTCHAs

- *Object Occlusion:* Each answer object has an overlapping shape (e.g., rectangle or circle as shown in Figure 4.1(b) and 4.1(c)), and the position of this shape is updated randomly.
- *Animated Background:* A gif file (containing, e.g., random fireworks) is added as the background (Figure 4.1 (c)).
- *Snow Effect:* In each frame, the color of the game background's pixels is set randomly. The objects are represented with boundaries only. Each object has a specific color at the start, and in each frame, the boundary color transitions into a different color (Figure 4.1 (d)).

At a higher level, it may seem that adding those countermeasures could enhance the security of S-DCG CAPTCHAs against automated tracking that utilizes background subtraction method. However, a careful examination suggests that a corresponding automated attack could be built to bypass these countermeasures.

1. *Artificial/Natural scene video background and occlusion randomly roaming over the moving objects:* The foreground objects have a low visual correlation with the background content, which makes them trackable by utilizing the visually distinguishable feature, such as color, thereby being vulnerable to automated attacks.
2. *Random color background and semi-transparent moving objects:* Moving objects could become visible in color and/or luminance spaces in intermittent frames. Their contour could be recovered through cross matching of detected contour information between exposed frames, whereby a drag-and-drop could be launched by the automated attacks.

4.2 Emerging Image based DCG CAPTCHA (EI-DCG)

In order to design a CAPTCHA scheme that resists both automated attacks and relay attacks, we turn to two categories of CAPTCHAs from the current literature – *Emerging Image* (EI) CAPTCHAs [13, 14] (known to be resistant to automated attacks) and *Simple Dynamic Cognitive Game* (S-DCG) CAPTCHAs (known to be resistant to relay attacks). However, *no* current scheme is known to be simultaneously resistant to *both* attacks. We aim to achieve this property via a careful combination of the two categories of CAPTCHAs.

In the work of [78], the authors proposed emerging images of *3D objects* and explained the emergence as "the phenomenon by which we perceive objects in an image not by recognizing the object parts, but as a whole, all at once". The authors indicate [78]: "humans cannot instantaneously detect the object in such images, and can probably recognize it only after several iterations that take into account numerous relationships between hypothetical objects and their context. The computational complexity of this human processing is believed to be extremely high [79], leading us to hypothesize that emergence images are hard for automatic algorithms to segment, identify, and recognize". They further go on to argue that: "Taking into account the complexity of the task, and the lack of a clear understanding of how humans solve the problem, it is highly unlikely, if not impossible, that these types of tasks could be carried out by bots in the near future". A concrete instantiation of an EI CAPTCHA developed in [13, 14] inherits the above-mentioned characteristics, and is demonstrated to be secure against automated attacks. However, such video-based EI CAPTCHAs are completely vulnerable to relay attacks whereby the static video challenge can be simply forwarded to the remote human-solver.

A S-DCG CAPTCHA exhibits certain interesting properties. First, it is based on a cognitive puzzle, which is easy for humans to understand, but may be difficult for a bot without enough clues. Second, the game-based nature enhances the usability of CAPTCHA solving. Third, due to the dynamic and interactive nature of the underlying game, relaying the game to a remote solver might be challenging. As shown in Chapter 3, however, DCG CAPTCHAs based on static background are vulnerable to automated attacks. On the positive side, they were shown to offer resistance to relay attacks.

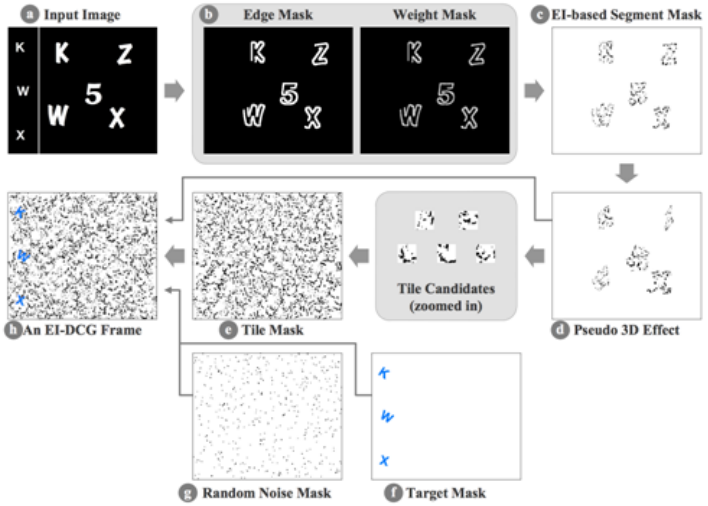


FIGURE 4.2: Generating an EI-DCG CAPTCHA Frame

4.3 DESIGN & IMPLEMENTATION

4.3.1 Design Overview

Our EI-DCG CAPTCHA has following unique features that differentiate it from EI-based Nu CAPTCHA (EI-Nu) [13] and the original EI-based videos [78].

1. Instead of using 2D objects as in EI-Nu CAPTCHA or real 3D objects with shading as in EI videos, EI-DCG CAPTCHA uses a pseudo 3D object (i.e., projecting a 2D object into 3D space, applying necessary transformations, and projecting it back to 2D) to provide a simulated 3D view, and more importantly to lower the possibility of recovering object contours through accumulation of information from consecutive frames, i.e., to protect against auto-decoding.

2. Hiding information in a single frame is more challenging in EI-DCG CAPTCHA since there are usually more foreground objects (e.g., ≥ 5) than there are in EI-Nu or EI videos. Most EI-based videos in [78] contain one single object, making it relatively easy for a human viewer to focus on the object. Meanwhile, having very few moving objects also leaves adequate room for making each object sufficiently large for easy recognition by human eyes. However, it will be too risky for a S-DCG CAPTCHA to use less than 5 objects (keeping in mind the random guessing attacks). As explained in Section 3.3.1, if the locations of moving objects are exposed (e.g., through multi-frame accumulation), a random guess can achieve $\sim$13% success rate given 5 moving objects, 3 target objects,

allowing $\leq$2 drag-and-drop attempts per object. In an EI-DCG CAPTCHA frame, we camouflage moving objects in both a single frame and in the accumulation of consecutive frames by tiling the background with deformed and incomplete edge segments from foreground objects in a way similar to that of the EI videos [78].

3. Both EI-Nu and EI videos play a fixed video clip repeatedly, leading to a constant-time requirement for rendering. However, an EI-DCG CAPTCHA challenge demands real-time user interaction with foreground objects, requiring each frame to be generated on the fly and incurring higher computation. We apply a divide-and-conquer strategy to prepare the information needed for generating a frame in advance, and repeatedly use it to efficiently create more new frames.

Creating an EI-DCG frame requires creating both the foreground object mask as well as the background mask. For an input image with both foreground objects and the target objects (Figure 4.2(a)), the edge mask and the weight mask are computed (Figure 4.2(b)), which are used to generate the EI visual effect (Figure 4.2(c)). Large segments that may leave clue for reconstructing object contour will be further split. The pseudo 3D effect for each object is applied based on current rotation and scaling parameters (Figure 4.2(d)). The remaining area in the frame (i.e., the background) is tiled with segments from foreground objects (Figure 4.2(e)). Finally, an EI-based frame (Figure 4.2(h)) is the Gaussian blur of the combination of the foreground mask, tiled background, target object mask (Figure 4.2(f)), and random noise mask (Figure 4.2(g)). The details will be provided in the following subsections. An EI-DCG CAPTCHA challenge is configured as follows:

- Dimension: 340(height)$\times$400(width).
- 3 target objects and 5 foreground objects, which are all alphanumeric characters.
- Object moving speed: 3 pixels per frame (ppf).
- Frame rate: 40 frames per second (fps).
- N=40 pairs of foreground and background that record the pixel value and location of foreground and background objects in each frame are rendered.

The purpose of using a higher frame rate (e.g., 40 fps) than usual (e.g., 30 fps) is to increase the robustness against the relay attack. The higher the frame rate, the less information about

foreground objects is revealed in one single frame, thus requiring more frames to be read at a time in order to recognize the object. In case of a relay attack, the communication delay between the bot and the human solver's machine could cause loss of synchronization and thus lead to the failure to keep up with the required frame rate, resulting in jittery motion in video play. Since human eyes rely on continuous motion to recognize EI objects, this design choice will make it even harder for remote human-solver to identify the object and play the game effectively.

4.3.2 EI-DCG Configuration Levels

EI-DCG CAPTCHA is configured at three levels of potential difficulty for the human user (easy, medium, and difficult). The more "difficult" the CAPTCHA is, the less susceptible it is supposed to be against computer vision-based auto attack. The pixel density decreases when the difficulty level increases, i.e., decreasing amount of foreground information embedded in one single frame.

4.4 Automated Attack Resistance

Emerging images [78] are known to be robust to automatic object detection using existing image processing and machine learning techniques. One challenge in EI-DCG design is the local density difference caused by the presence of hollow objects and the white dilation surrounding the moving objects (used to make objects more visually distinguishable), which may facilitate automated attack. On one hand, the hollow area and the white dilation area, which accompany the moving object in consecutive frames, may remain white (and thus appear "sparser") in the superimposed mask, indicating possible presence of foreground objects. Also, due to the randomness in the layout of background tiles, the local density of foreground objects in a single frame may occasionally become relatively higher/lower than surrounding areas. In this case, the automated attack could first detect subareas that exhibit such local density anomalies, randomly pick one of them, and drag it to the target. Other automated attacks using object detection in computer vision would be extremely hard, both theoretically and computationally, if not entirely impossible in this case, due to a lack of presence of distinct object visual features such as color, gradient, corner points, edge, or shape, and a lack of prior knowledge of object features. Therefore, neither feature-based nor appearance-based object detection would work well, leaving the best hope with an anomaly-based detection method such as the automated density-based attack

that is also computationally efficient.

To evaluate the robustness of EI-DCG CAPTCHA against such density-based automated attack, we applied the automated attack to 3 different masks, i.e., single binary mask (***single***), superimposition of 3 consecutive masks (***3x***), and the frequency map of 3 consecutive masks (***freq***). Our automated attack assumes that the locations of the target objects are already known (e.g., known via a simple image-based relay attack). First, a screen-captured frame is converted into a binary mask with the target area removed. Second, the remaining area (i.e., activity area of moving objects) is divided into $m \times m$ equal-sized subareas (e.g., 40×40 pixels, Figure 4.3). An $m \times m$ density matrix is created in which each element indicates the black pixel count of the corresponding subarea. The centroids of subareas that correspond to local peak or valley elements in the 2D density matrix are treated as the location candidates of moving objects (Figure 4.3(a)). Third, the automated attack randomly selects a location candidate and performs a drag-and-drop operation from the cell centroid to each of the target objects.

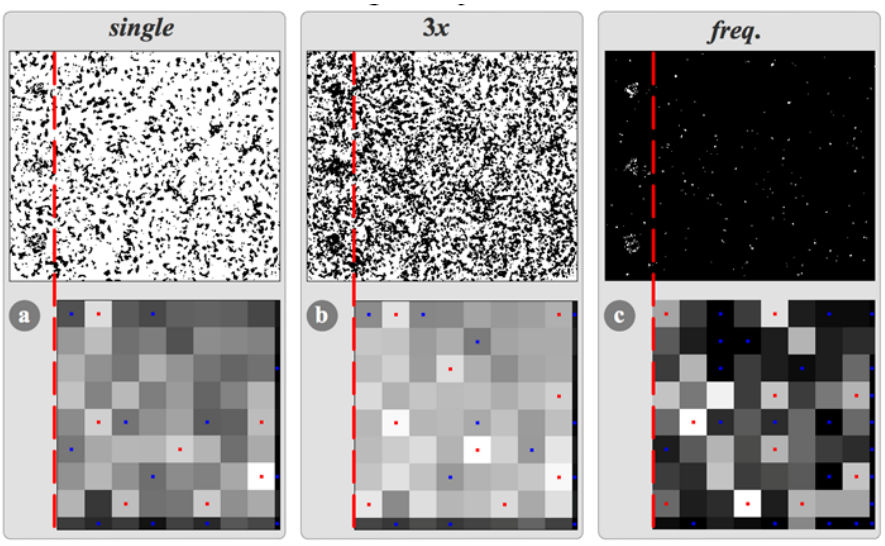


FIGURE 4.3: (a-c) Top: Single Binary Mask, Superimposition of 3 Consecutive Binary Masks, and Binary Mask of Frequency Map with Pixels Having the Highest Possible Frequency (i.e., 3) Shown as White. Bottom: Density Matrix with Grid Interval as 40 Pixels. The Red and Blue Dots Indicate Local Density Peaks and Valleys, Respectively.

A superimposed mask (***3x***) is generated by superimposing the current binary mask with its previous two consecutive masks (Figure 4.3(b)). The value of a pixel in the frequency map is the number of times a black pixel appears at that pixel location across the 3 consecutive masks. The frequency map is further binarized so that only those pixels whose value is 3 (the highest possible frequency count of a black pixel) will be shown as white in the final binary mask (Figure

4.3(c)). The density matrix of such a binary mask records the white pixel count in each subarea (2nd row, Figure 4.3(c)).

We randomly created 100 challenges of EI-DCG CAPTCHAs corresponding to each difficulty level. There are 12 groups of attacks with various parameter settings (Table 4.1) for each difficulty level. Each group contains 500 attacks on randomly selected challenges from the corresponding 100-challenge set. Each attack will perform drag-and-drop at most 50 times ("time out" otherwise). In our first experiment, we set the maximum number of drag-and-drop attempts allowed to be 21, i.e., each target object will receive $\leq$7 drops on average, given 3 target objects. This is the threshold parameter that our EI-DCG CAPTCHA implementation will use in practice (later in, Section 4.5, we will demonstrate that such a thresholdization does not have much impact on the usability of solving EI-DCG challenges by legitimate users).

TABLE 4.1: Parameter Settings for the Density-Based Automated Attack

Parameters	**Values**
Difficulty level	Easy, Medium, Difficult
Density matrix (DM)	Single mask, 3x superimposition, Freq. map
Obj. locations (OL)	Local density peak, Local density valley
Grid interval (GI)	40 pixels, 60 pixels

Our result indicates that the success rate for density-based automated attack is lower than 0.8%, given $\leq$7 drag-and-drops per target object on average. Since this attack is based on local density anomaly, that occurs with randomness, the difficulty level is not necessarily reflected in the success rate. This success rate is well-aligned with the acceptable security level for CAPTCHAs (e.g., as specified by Zhu et al. [67]).

Next, we further experiment with $\leq$50 drag-and-drops in order to gain more insights. As shown in Figure 4.4 (a)(c), localizing foreground objects by using local valleys in the density matrix of *single* and 3*x* provides a higher success rate than that by using local peaks. In a single binary mask (*single*) or the superimposition of 3 consecutive masks ($3x$), the white dilation area surrounding the moving object (for highlighting the object) may form relatively sparse subareas (valleys), thereby making the valley-based attacks more effective than peak-based attacks. On the other hand, since there are not many pixels in the binary mask of frequency map ($freq$), the local peaks more likely correspond to the moving traces left by moving objects. Therefore, the success rate of peak-based attack by using density matrix of $freq$ is higher than the other two. For the same reason, a small grid interval, such as 40 pixels, may result in many meaningless

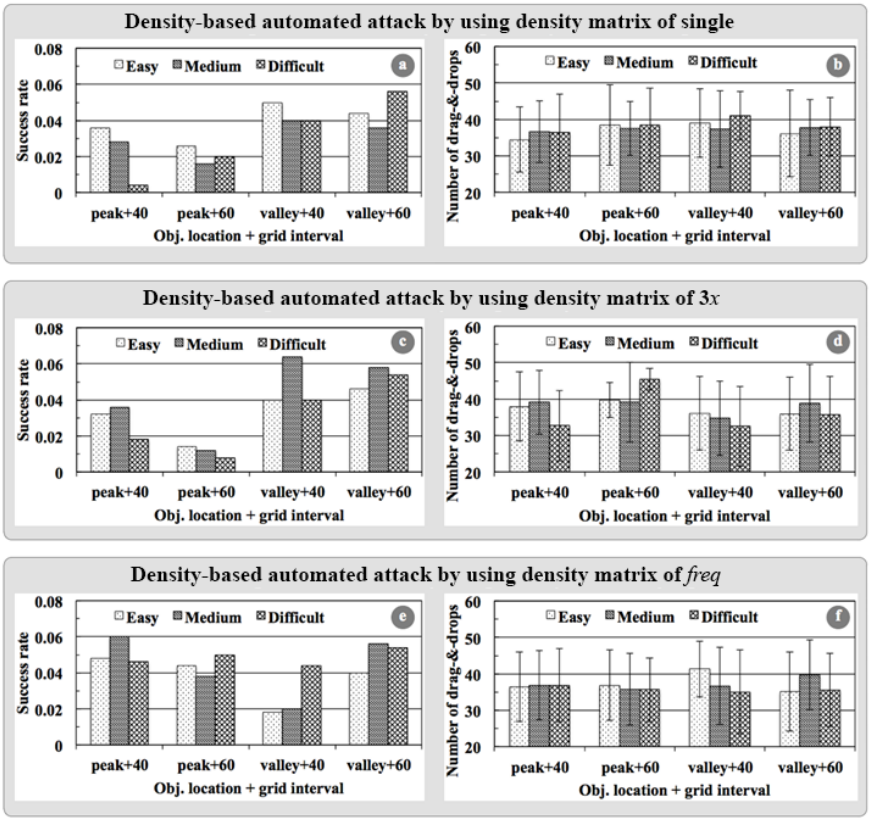


FIGURE 4.4: Success Rate and Number of Drag-And-Drops in Density-Based Automated Attack with $\leq$50 Drag-And-Drops for Each Attack. (a)(c)(e) Success Rates by Using Density Matrices of Single, $3x$, and Freq. (b)(d)(f) the Mean and Standard Deviation of the Number of Drag-And-Drops to Complete an EI-DCG CAPTCHA Challenge.

valleys in the density matrix of $freq$ (e.g., Figure 4.3(c)) and thus likely lead to a lower success rate. This explains why in the experiment for $freq$, 'valley+40' has a lower success rate than the other three configurations i.e., 'valley+60', 'peak+40', and 'peak+60' (Figure 4.4(e)).

Experiment on the number of drag-and-drops (Figure 4.4(b)(d)(f)) indicates that a successful automated attack usually requires an average of 30$\sim$40 drag-and-drops to complete the CAPTCHA. which is much higher than our threshold of 21.

4.5 Usability

In our usability study of EI-DCGs, we use EI-Nu as a baseline – our goal is to compare the usability of EI-DCGs with that of EI-Nu. Basically, we wanted to determine how much usability degradation occurs in EI-DCG over EI-Nu, as a trade-off to enhancing the security against relay attacks. We utilized the Amazon Mechanical Turk (MTurk) service to recruit participants for the study. The study was approved by our University's Institutional Review Board.

4.5.1 Study Design

Each EI-Nu CAPTCHA challenge is of size 285×125 and is displayed as a 6-second video that loops continuously. We asked the participants to type the three characters of the challenge in a textbox and press the "submit" button when they are done. Each EI-DCG challenge is of size 360×400. The user task is to drag-drop the answer objects to their corresponding target objects within 60 seconds (time-out). If the user cannot complete the task within the 60 seconds, the challenge is considered unsuccessful. We generated 100 challenges for each category of tested CAPTCHAs: EI-Nu and EI-DCG (Easy, Medium and Difficult). We employ a within-subjects experimental design, where we ask each participant to solve 5 challenges each for all the four categories. The order of presenting the four categories (EI-Nu, EI-DCG_Easy, EI-DCG_Medium, and EI-DCG_Difficult) followed the standard 4×4 Latin square to reduce the effect of learning biases, while the challenges within each category followed a random order. We recruited 120 MTurk workers, and the experiment took 27 minutes on an average to complete per worker.

We subjected the MTurk workers to a consent form. Then, we asked them to fill-out a demographics form, solve five challenges of one of the tested CAPTCHA categories, and fill-out a survey form about their experience. The survey contains the 10 System Usable Scale (SUS) [80] standard questions, each with 5 possible responses (5-point Likert scale, where strong disagreement is represented by "1" and strong agreement is represented by "5"). We used a similar design to test the rest of the categories. The demographics of the study participants are shown in the second column of Table 4.2.

4.5.2 Study Results

We evaluated the usability of the tested CAPTCHA categories with respect to the measures of: (1) solving time, (2) error rate, and (3) user experiences, as described below. The results are summarized in Table 4.3.

4.5.2.1 Solving time

We calculated the solving time as the time taken by the participants to solve each challenge. In case of EI-Nu, we start measuring the timing from the time the challenge is displayed till the

TABLE 4.2: Demographics of Participants in the *Usability* and *Relay Attack* studies

	Usability	**Stream Relay Attack**	
Participants Study Type		LSHL	HSLL
Participants Size (N=195)	N = 120	N = 27	N = 48
Age (%)			
<18	0	0	2.1
18 - 24	26.6	16.7	8.3
25 - 34	44.2	70.8	58.3
35 - 50	19.2	8.3	27.1
>50	10	4.2	4.2
Gender (%)			
Female	39.2	25	27.1
Male	60.8	75	72.9
Education (%)			
High School	30.8	0	22.9
Bachelor	43.3	62.5	72.9
Master	24.2	33.3	4.2
PhD	1.7	4.2	0

submit button is pressed. Whereas, in case of EI-DCG, we record the timing till the participants drag-drop all the answer objects to their corresponding target objects. We considered in our calculation the time taken only corresponding to the challenges solved successfully. The average solving time is shown in the third column of Table 4.3.

The time taken to solve EI-DCG challenges is about double the time taken to solve EI-Nu challenges. However, it is still less than 25 seconds on average. Moreover, the time for solving EI-DCG increases with the difficulty level of EI-DCG. A Friedman's test showed a statistically significant difference between the solving time of the four tested categories ($\chi^2(3)$ = 500.06, p < 0.001). Further analyzing using pairwise Wilcoxon Signed-Rank test with Bonferroni correction, we found a statistically significant difference between all of the tested pairs ($p < 0.001$).

TABLE 4.3: The Solving Time, Error Rate, Number of Drags, Number of Attempts and SUS Scores for the *Usability* Study

Challenge Type	**SUS**	**Time (sec)**	**#Drags**	**#Attempts**	**Error Rate**
	mean (std. dev.)				
EI-Nu	71.75 (18.39)	10.34 (6.21)	N/A	N/A	0.16
EI-DCG					
Easy	55.94 (19.75)	19.82 (10.20)	3.82 (1.79)	1.17 (1.86)	0.13
Medium	57.15 (18.09)	21.55 (10.55)	3.82 (1.58)	1.51 (2.42)	0.10
Difficult	56.00 (20.08)	23.34 (11.60)	3.84 (1.85)	1.44 (2.03)	0.13

4.5.2.2 Error Rate

The error rate represents the percentage of the challenges that were not solved successfully by the participants. The last column of Table 4.3 shows the error rate for solving each of the tested CAPTCHA categories. All of the categories had low error rate with the minimum error rate for EI-DCG medium as 0.10 and the maximum for EI-Nu as 0.16. The lower error rate in EI-DCGs compared to EI-Nu may be attributed to to the momentary feedback that EI-DCG provides. Whenever the participant drag-drops a correct object to its corresponding target, the object disappears, which informs the user he performed a correct drag-drop. However, EI-Nu does not check the users response until after he submits the whole answer of the challenge.

Further, we analyzed the number of drag-drops performed by the participants, which represents the error rate per drag. We noticed that a minimum of three drag-drops is required to complete any challenge and on an average the users performed less than four drag-drops in all the EI-DCG categories. Finally, we analyzed the number of attempts (clicks that do not correspond to object drag) performed by the users and we found the users performed less than 2 attempts on average for all of the EI-DCG categories. Upon further analysis of the collected logs, we found that some of the participants performed many drags and attempts (up to 37) while solving challenges. However, the fraction of such actions is extremely low, which confirms that we can limit the number of allowed drags and drops to 21 to limit the ability of density-based automated attack (as analyzed in Section 4.4) successfully without impacting the usability much. The overall error rate, after limiting to 21 drags/attempts, becomes 0.14, 0.11, and 0.14 for Easy, Medium and Difficult EI-DCG, respectively. These errors rates are still similar to that of EI-Nu.

4.5.2.3 User Experience (SUS Scores)

The first column of the Table 4.3 shows the SUS scores corresponding to the tested CAPTCHA categories. In our SUS calculations, we ignored the responses of 17 participants as they seem to answer the questionnaire randomly, either by giving the same rating to all the questions or at least answer two contradicting questions with the same answer (i.e., we removed the responses from participants who answer both of “I found the system unnecessarily complex” and “I thought the system was easy to use” with “strongly agree”).

We find degradation in the user experience in EI-DCG compared to EI-Nu. However, the SUS scores for EI-DCG are still above 50.9, which means EI-DCGs have fair usability [81]. Comparing the SUS scores using Friedman's test shows statistical significant difference among the tested CAPTCHA categories ($\chi^2(3)$ = 87.63, p $<$ 0.0001). Further, we used pairwise Wilcoxon Signed-Rank test with Bonferroni correction to assess the difference between each of the pair of the four categories. Significant differences are found (p $<$ 0.01) between EI-Nu and the three categories of EI-DCG. However, no significant difference is found in the SUS score between any pair of the EI-DCG categories.

4.5.3 Summary of Results

The results of the usability study show some degradation of the user experience represented in lower SUS score and higher time taken to solve EI-DCG challenges compared to EI-Nu challenges. However, the average SUS scores for EI-DCG show that they still have fair usability. Moreover, the error rate decreased slightly compared to EI-Nu. Given that EI-DCG offers higher security than EI-Nu, especially against relay attacks, we believe that this degradation in usability may be acceptable.

4.6 Security Against Relay Attack

We conducted two studies to investigate the ability of EI-DCG to resist Stream Relay attack introduced in Section 3.4.3. The studies differs in only the location of the participants. In the first study, tagged Low-Speed High-Latency (LSHL), we recruited participants from a developing country (India), where we expect the users to have a slow Internet connection and they reside on far proximity of the attacker (residing in the USA). In the second study, High-Speed Low-Latency (HSLL), we recruited participants from a developed country (USA), where we expect the users to have a fast Internet connection and they reside in near proximity of the attacker (USA). These two attack models emulate realistic relay attack scenarios. In both models, the human-solvers attempt to solve EI-DCG challenges that are streamed to them in-real time from the attacker's machine using the VNC streaming software.

4.6.1 Study Design

Following the study design in the usability study, we employed a within-subjects experimental design, where we ask each participant to solve 5 challenges for all of the three EI-DCG categories. The order of presenting the three categories followed standard 3×3 Latin square, and the challenges within each category followed a random order. We asked the MTurk workers to connect to a computer which resides in our university (streaming server) via the RealVNC Java applet (streaming client). Then, we subject them to consent an agreement. Next, we ask them to fill-out a demographics form, and solve five challenges of one of the categories. We followed a similar design to test the other categories. The study was approved by our University's Institutional Review Board.

We conducted two separate streaming-based relay attack studies. The two studies differ only in the location of the participants. In the first study (LSHL), we recruited 27 participants in India which simulate the real scenario settings in which the attacker is in USA and hires human-solvers in far and developing countries. In the second study (HSLL), we recruited 48 participants from USA. The second study is to assess how much the performance of the attack will increase when the attacker recruits solvers in near proximity and from developed countries. The demographics of our participants are shown in columns 3-4 of Table 4.2. The participant characteristics in our relay attack studies are in line with that in our usability study, allowing us to fairly compare the two settings in a between-subjects design.

TABLE 4.4: The Solving Time, Error Rate, Number of Drags, and Number of Attempts for the *LSHL* and *HSLL* Streaming-Based Relay Attack Studies on EI-DCG

EI-DCG Challenge Type	**LSHL** (human-solver in India)				**HSLL** (human-solver in USA)			
	Time (sec)	**# Drags**	**# Attempts**	**Error Rate**	**Time (sec)**	**# Drags**	**# Attempts**	**Error Rate**
	mean (std. dev.)				*mean (std. dev.)*			
Easy	39.05 (5.47)	8.00 (4.24)	24.00 (1.41)	0.98	18.73 (10.62)	4.35 (3.05)	4.71 (15.83)	0.87
Medium	49.92	4.00	15.00	0.99	22.13 (11.69)	4.17 (2.91)	1.63 (1.88)	0.88
Difficult	-	-	-	1.00	24.05 (11.71)	4.53 (2.37)	1.66 (2.89)	0.86

4.6.2 Study Results

We evaluated the performance of the participants in solving EI-DCG over the streaming channel with respect to the measures of solving time and error rate. The results are summarized in Table 4.4.

In the LSHL relay attack setting, only two participants could complete one of the EI-DCG Easy variant, and only one participant could complete one of the EI-DCG medium variant.

The overall error rate is therefore 0.99 on an average for the three categories of EI-DCG. The average time taken by the participants to complete the challenges was 42.68 seconds. Moreover, the participants performed much higher number of drag-drops and attempts compared to the participants in the usability study. Moreover, if we limit the number of allowed drags/attempts to 21, the overall error rate for the Easy EI-DCG becomes 1.00.

In the HSLL relay attack setting, the error rate decreased from around 0.99 to 0.87 on average when compared to the LSHL relay attack study. However, the error rate is still considerably high. The completion time and number of drags and attempts for the participants who successfully solved the challenges are comparable to the ones in the usability study, except for the number of attempts in the Easy variant. Limiting the number of allowed drags/attempts to our threshold of 21 did not effect the over all error rate. Comparing the successful completion time between each variant of EI-DCG in usability and its corresponding streaming-based relay attack study, using Mann-Whitney test, did not reveal statistically significant differences, however. Further investigation of the collected data shows that only 9 out of the 48 participants (18.75%) were able to complete any EI-DCG challenges. On an average each of them solved around 10.33 out of the 15 challenges.

The above analysis suggests that EI-DCG is very hard to solve via streaming-based relaying, especially when the human-solvers are located far away and have slow Internet connections. The chance of solving EI-DCG challenges seem to increase, but still only marginally, when recruiting the solvers residing in near proximity and having high-speed Internet connections. Even when the solvers can solve the CAPTCHA challenges, we will next show how they can be detected based on different game play patterns compared to legitimate human users.

4.6.3 Relay Attack Detection

We design and implement a machine-learning based EI-DCG Stream Relay attack detection mechanism based on the differences between the participants' performances in completing the challenges in the usability study and the human-solvers' performance in the HSLL relay attack study. We logged all the participants' interactions with the challenges, while they were solving the challenges. We logged the interactions with the challenges regularly and whenever a mouse event is performed. Specifically, at each timestamp, we log the mouse position, mouse status (up or down), and for each object, we log its position and weather it is being dragged or not.

From each of the collected challenge log files, we extract a total of sixteen different features, as described below:

- **Number of Drags (ND)**: Total number of drags.
- **Number of Attempts (NA)**: The number of mouse clicks outside of the moving objects.
- **Distance-based Features**: (1) *Distance Drag (DD)*: The total distance of the mouse movement while it is dragging an object; (2) *Distance Attempts (DA)*: Distance of the mouse movement while the mouse status is down but not dragging an object; and (3) *Distance mouse Up (DU)*: Distance of the mouse movement while the mouse status is up.
- **Time-based Features**: (1) *Completion Time (T):* The total time taken by the participant to complete the challenge; (2) *Time drags (TD):* The total time in which an object is being dragged; (3) *Time Attempts (TA):* Time in which the mouse is down and no object is being dragged, and (4) *Time mouse Up (TU):* Time in which the mouse status is up.
- **Speed-based Features** (Distance/Time): Speed of Drags (SD), Speed of Attempts (SA) and Speed of Mouse Up (SU).
- **Acceleration-based features** (Speed/Time): Acceleration of Drags (AD), Acceleration of Attempts (AA) and Acceleration of Mouse Up (AU).
- **Max Attempt (MA)**: The maximum time taken in a single attempt.

For each of the EI-DCG variants, we picked randomly a number of instances of the successfully completed challenges in the usability study equal to the number of successfully completed challenges in the HSLL relay attack study. Then, we implemented a Java program to check which subset of features along with which classifier provides the best results for each of the challenges categories (Easy, Medium, Difficult). We experimented with different classifiers: SVM (C-SVC and nu-SVN, linear, polynomial and radial kernels), Multilayer Perceptron, Naive Bayes, Random Forest, Random Tree, Simple Logistic and Logistic. As performance measures for our classification task, we used Precision, Recall and F-measure (F1 score).

The results of the best classifier along the best features subset for each of the EI-DCG difficulty level are shown in Table 4.5. The results show that a classifier can be effectively built such that it rejects around 11% of legitimate users and be able to detect around 65% for HSLL relay attackers. As shown in Table 4.4, on average only 13% of HSLL relay attackers could solve the EI-DCG challenges, utilizing our proposed detection method 65% of them could be detected.

TABLE 4.5: Results of Using the Optimal Feature Subset for Each EI-DCG Game in the Classification of *Legitimate User* and *HSLL* Streaming-Based Relay Attacker

Challenge Type	Classifier	Features	Precision	Recall	F-measure
Easy	Multilayer Perceptron	ND, T, TD, TA, SD, SU	0.78	0.90	0.84
Medium	Random Tree	ND, T, DD, DA, DU, TU, MA, AD, AU	0.72	0.87	0.79
Difficult	Multilayer Perceptron	T, TU, AD, AA, AU	0.66	0.91	0.76

Therefore, the success rate for the HSLL relay attack is about 5%. The LSHL relay attack is already prevented with at least 98% probability (Table 4.4). The results show that the user can be authenticated within 4 EI-DCG instances with a probability of almost 1 (0.9978). However, the HSLL relay attacker would need about 20 trials to be authenticated with a probability of 1. Given that the average time for solving an EI-DCG challenge under HSLL relay attacker is 21 seconds, that means the human-solver would need about 7 minutes to be able to break the EI-DCG. This suggests that the streaming-based relay attack in general has a very low chance of succeeding against EI-DCG CAPTCHAs.

4.6.4 Summary of Results

The previous subsection shows that EI-DCGs is resilient to streaming-based relay attacks. Inherently, this ability is due to the fact that the user needs to see the challenge at about 40 fps to be able to recognize the objects and solve the challenge successfully. That requires high data connection speed with low latency between the attacker and the human-solver.

As the game size is 340×400 and the required frame rate is 40 fps, using 2 bit per pixel will require the connection speed of 10.38 Mbit/s between the attacker and the solver, if no compression is performed. However, RealVNC performs some compression on the data sent between the client and the server to enhance the performance. First, RealVNC sends only the modified pixels between the current frame and the previous frame rather than sending the whole frame. Second, the RealVNC client allows the user to adjust the quality of the images, varying from low quality (8 colors) to best quality (all available colors). In order to evaluate the minimum bandwidth required between the attacker and the human-solver to transfer the game with 40 fps, we performed an experiment in which we connected two laptops over a LAN, installed RealVNC server on one of them and RealVNC client on the other. We varied the RealVNC

compression levels starting from the best compression that achieves the best transmission quality. We used Wireshark to capture the packets sent between the two laptops in all of the cases, and then analyzed the average data rate used. The average Mbit/s for the getting all colors with maximum compression is 4.70 Mbit/s, 3.15 Mbit/s for the default settings (256 colors) and 2.22 Mbit/s for the best compression (8 colors). This shows the minimum connection speed required between the attacker and human-solver should be 2.22 Mbit/s to be able to solve the EI-DCG at the human-solver's end. Any slower connection would result in a jittery rendering of the game, and therefore the human-solver will not be able to solve the EI-DCG successfully. This justifies the results of the user studies presented in the previous subsection.

According to [82], the average connection speed is 2.0 Mbit/s in India. Also, since the long distance between the attacker (USA) and the human solver slows down the connection speed, almost all the participants in India could not remain connected to our server with high speed and they failed to successfully solve the challenges. On the other hand, the average connection speed in USA is 11.1 Mbit/s. Also, the distance is much less between the attackers and the human-solvers in the HSLL streaming-based relay attack study. This explains why some of the human-solver were able to successfully solve some of the challenges.

We tested multiple screen-sharing applications, such as TeamViewer and anydesk. All of them require almost similar data rates between the server and the client to transfer the challenges. For example, TeamViewer requires 2.73 Mbit/s for gray-scale, and anydesk requires 2.63 Mbit/sec. Therefore, we conclude that the results of our relay attack studies are not limited to RealVNC.

Small Game Relay whereby the attacker reduces the game size before sending it to the human-solver to lower the data rate required between the attacker and the solvers. However, this scenario cannot be applied to EI-DCG as reducing the game size will make it almost impossible for the human-solver to recognize the moving objects (because of the emerging effect).

4.7 Conclusions

We proposed a new class of CAPTCHAs, EI-DCG, based on the notion of emerging images and Dynamic Cognitive Games. EI-DCG applies a series of countermeasures, such as pseudo 3D rotation, hidden edge segments, segment split-erosion-rotation-translation and tiled background,

to resist automated object recognition based on both single frames and frame superimposition. Since a human can perceive objects in EI-based frames from motion, playing an EI-based video in a relatively slower frame rate decreases the human recognition rate. Based on this property, a faster frame rate (i.e., 40 fps) was applied for the normal playing of an EI-DCG challenge. The higher the network delay is, the lower the frame rate of EI-DCG becomes on the human-solver's side, providing less opportunity for the solver to complete the challenge successfully. The experiments against both automated and relay attacks indicated the robustness of EI-DCG.

CHAPTER 5

CAPTCHA FUSION TO DEFEAT AUTOMATED AND HUMAN ATTACKS

While EI-DCG CAPTCHAs can provide a high level of security, it is clear that their usability are much lower than that of S-DCG CAPTCHAs due to the presence of dynamic background and emergence effect. Moreover, adding background and emergence effect do not secure S-DCG against *known probable answer objects, known target objects* random attack. This motivates us to design secure DCG CAPTCHAs that may offer a usability level comparable to that of S-DCG CAPTCHAs.

Given the current state of CAPTCHA security, it is natural to consider the question: *can a CAPTCHA scheme be designed that simultaneously resists automated attacks, relay attacks and client-side attacks?* To answer this question, we turn to three different forms of CAPTCHAs existing in the literature: (1) image orientation CAPTCHAs (such as "What's up CAPTCHA" [1]), (2) semantic image matching CAPTCHAs (such as SEmantically MAtching imaGEs [2]), and (3) Dynamic Cognitive Game (DCG) CAPTCHAs (e.g., Simple DCG CAPTCHAs). Image orientation CAPTCHAs and semantic image matching CAPTCHAs are known to be secure against automated attacks and client-side attacks, but are vulnerable to relay attacks. Existing DCG CAPTCHAs are known to be resistant to relay attacks, but are vulnerable to both automated and client-side attacks. However, *no* current scheme is known to be simultaneously resistant to *all* three attacks. We aim to achieve this property via a careful fusion of the above three categories of CAPTCHAs. Said differently, we introduce a new form of DCG CAPTCHAs that incorporate the notions of image orientation and image semantics to defeat automate, relay and client-side attacks. We refer to this new class of CAPTCHAs as "Mix DCG" CAPTCHAs.

Chapter Organization: Section 5.1 reviews the various categories of CAPTCHAs relevant to our Mix DCG construction, and explains the security limitations of each of them. Section 5.2 elaborates on the design and implementation of our Mix DCG instantiation. Section 5.3 analyzes the security of Mix DCG against automated attacks. Section 5.4 presents a usability study of the Mix DCG construction. Section 5.5 evaluates the security of Mix DCG against relay attacks and

proposed a viable relay attack detection mechanism. Finally, Section 5.6 and 5.7, respectively, discuss the various aspects relevant to our work and conclude the chapter.

5.1 Background

The design objective of our work is to develop a CAPTCHA scheme that is secure against automated attacks, relay attacks and client-side attacks, while offering a reasonably good level of usability (better than conventional text-based CAPTCHAs). In order to achieve our objective, we integrate three categories of CAPTCHAs: Dynamic Cognitive CAPTCHA, image orientation CAPTCHA (i.e., What's up CAPTCHA [1]), and semantic matching CAPTCHA (i.e. Semantically Matching images [2]). In this section, we review these three forms of CAPTCHAs, highlight their security properties and the vulnerabilities associated with them, and provide an overview as to how integrating all three of these CAPTCHAs can give rise to a secure and usable CAPTCHA scheme.

5.1.1 Utilized CAPTCHA Designs

First, *Dynamic Cognitive CAPTCHA (DCG)* is a simple interactive CAPTCHA that involves objects floating around within an image, and the user's task is to match the objects with their respective target(s) and drag-drop them to the target location(s). As explained in Chapter 3, the dynamic nature of S-DCG as well as the requirement for multiple interactions between the user and the CAPTCHA challenge make S-DCG highly resilient to relay attacks.

Second, *What's up CAPTCHA* is an image orientation CAPTCHA that requires the users to identify the upright orientation of multiple images drawn from web searches (an example of What's up CAPTCHA is shown in Figure 5.1. The security of What's up CAPTCHA against automated attacks is based on the fact that finding the upright orientation of an image is difficult to automate over a wide variety of photographic content (a hard AI problem) [1].

Third, *SEmantically MAtching imaGEs (SEMAGE)* is a semantic CAPTCHA that challenges the user to select *semantically related* images from a set of images. In order to solve a SEMAGE CAPTCHA, the user needs to comprehend the content of the images and find a semantic relationship between a subset of the images. Both the underlying tasks, image recognition and semantic relationship identification, are easy for humans but considered hard for a

FIGURE 5.1: A Snapshot of What's up CAPTCHA [1] (The User Task is to Move the Slider to Rotate the Image to Its Upright Orientation).

computer [2]. Several examples of semantic relationships have been discussed in [2], such as finding the images of animals of the same species from a set of images that contains real and cartoon animal images (see Figure 5.2.

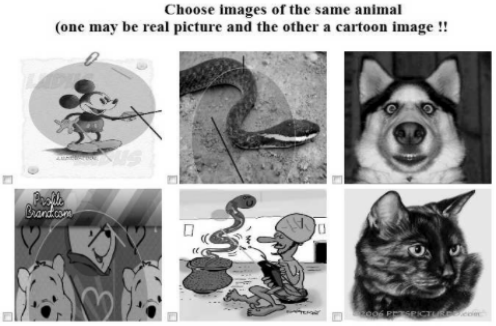


FIGURE 5.2: A Snapshot of SEMAGE [2] (The User Task is to Select the Semantically Related Objects, in this Snapshot the Second Image in the First Row and the Second Image in the Second Row).

5.1.2 Vulnerabilities of the Three Designs

What's up CAPTCHA and SEMAGE CAPTCHA are know to be resistant to automated attacks (based on hard AI problems). They are also secure against client-side attack (since all verification is performed at the server-side). However, both of them are vulnerable to relay attacks. In case of the SEMAGE CAPTCHA, the bot can send a single snapshot of the challenge to a remote human-solver who would respond with the coordinates of the semantically related objects. The bot then solves the challenge by triggering mouse clicks on the locations acquired from the human-solver. In case of What's up CAPTCHA, the bot may send the image of the challenge embedded into a similar interface as provided by What's up CAPTCHA. The human-solver would use the interface to orient the image and respond with the final location of the slider. The bot would adjust the slider on the same positions as it got from the human-solver.

The S-DCG CAPTCHA, on the other hand, has been shown to be resistant to relay attacks. However, it is not secure against automated attacks and client-side attacks. The main security vulnerabilities of S-DCG CAPTCHAs are summarized below:

1. **Random Guessing Attacks:** The S-DCG CAPTCHAs are vulnerable to random guessing attacks (Chapter 3).

2. **Dictionary-based Attacks:** The S-DCG CAPTCHAs are vulnerable to dictionary-based automated attacks (Chapter 3).

3. **Client-Side attacks:** The DCG CAPTCHAs are vulnerable to client-side attacks. In the implementation of S-DCG, the game logic is implemented on the client side, i.e., the check for the correctness of each drag-drop is performed on the client side. Only after the DCG challenge is completed, the log of the user's interaction with the challenge is sent to the server. This implementation suffers from multiple vulnerabilities. First, the attacker may reverse engineer the swf file, and extract the code and resources (i.e., the images of the moving and target objects) of the challenge. The extracted code would let the attacker know the positions of the target objects, the ids of the answer objects and their corresponding targets, and the images of the answer objects. Given all this information, the attacker may attack the game by searching for the answer objects images within the game, and drag and drop them to their corresponding target (a simple image processing program can achieve this functionality); Second, the attacker may generate a response similar to the one the server expects from the game (interaction log, or pass/fail result), and send it to the server. Third, the output of one game can be saved and used as response to a different game instance (replay attack).

4. **Shape Matching Attacks:** A method for selecting the the moving and target objects need to explored. Although some of the S-DCG are based on shape matching and semantic matching, the security of those games was not studied. For example, any game that is based on shape matching can be easily solved after extracting the moving and target objects 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.

5.1.3 Why Mix DCG?

In this chapter, we study approaches to thwart the above-listed security weaknesses of S-DCG CAPTCHAs. The first vulnerability motivates the need for finding a way to increase the solution space to resist random attacks. Increasing the solution space would also solve the second vulnerability as it will slow down building the dictionary. The third vulnerability motivates us to move the game logic to the server side. The fourth vulnerability motivates us to limit the DCG to semantic matching games, and consider a careful selection of the objects such as the shape/color that do not reveal the associations between the moving and target objects. All these vulnerabilities can be addressed by integrating S-DCG with image orientation and semantic matching CAPTCHAs (giving rise to our Mix DCG design), in addition to updating the DCG implementation by migrating the game logic to server-side. Integrating DCG with image orientation CAPTCHA would increase the solution space by a factor of the number of allowed orientations of the objects to the power of the number of answer objects. Following the image selection of semantic matching CAPTCHA in selecting the moving and target objects would prevent shape matching attacks. At the same time, integrating S-DCG with image orientation and semantic matching CAPTCHAs would also thwart the vulnerability of the latter two to relay attacks. Such an integration would also provide a reasonably good level of usability, especially when compared to traditional CAPTCHAs based on distorted characters.

5.2 Design and Implementation

In the section, we elaborate on the design and implementation of our Mix DCG CAPTCHA construction.

5.2.1 Mix DCG Design

In order to provide security against random attacks, we designed our Mix DCG CAPTCHA scheme such that each instance contains three target objects and six moving objects, three of which are the answer objects. To solve a Mix DCG challenge, (1) the user has to understand the content of the images, (2) find the semantic relationship between the answer objects and the target objects, and (3) find the correct upright orientation of the answer objects (by right clicking on the answer object image) and (4) drag the answer objects to their corresponding targets.

The above tasks have to be easy for human users and difficult for computer programs to perform. Therefore, the selection of the images of the moving objects and the targets has to be performed carefully. The answer and target objects should have different physical attributes (i.e., color and shape) to prevent automated programs from solving the challenge based on the similarities between the targets and the answer objects. Moreover, the moving objects should be easy for humans to orient but difficult for computers to orient. The CAPTCHA designer should avoid objects that can be oriented accurately by computer programs, such as faces, cars, outdoor images (for example, images containing sky, grass, and sand), and images that contain text [1].

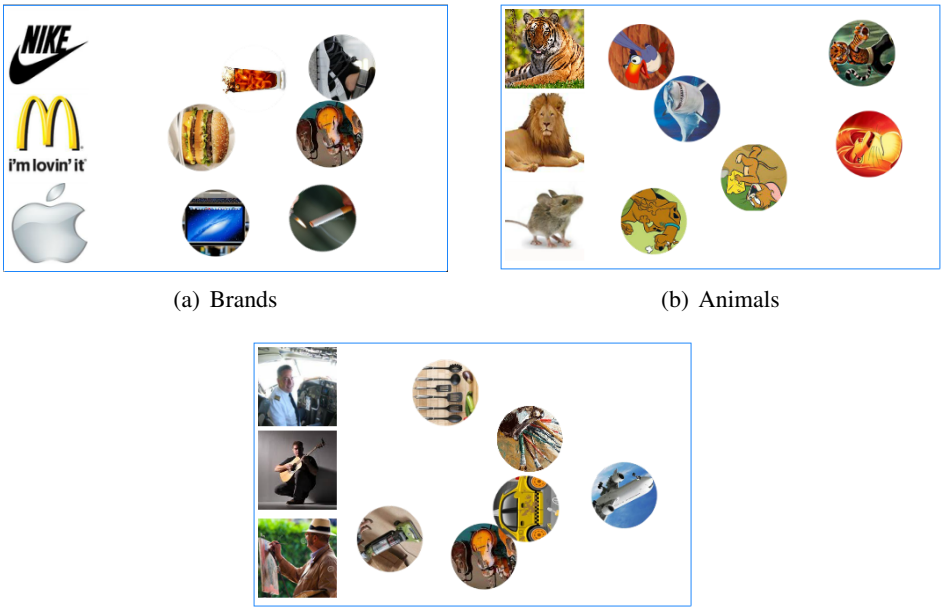
5.2.2 Mix DCG Implementation

We implemented the Mix DCG challenges using Adobe Flash ActionScript3 and the web server using PHP. The game image/frame size is 500 $\times$ 300 pixels, the size of each of the moving object is 75 $\times$ 75 pixels and the size of the target objects is 90 $\times$ 90 pixels. The Mix DCG challenge starts by placing the objects in random locations on the image and with random orientation. Then, each object picks a random direction in which it will move. A total of 8 directions were used, namely, N, S, E, W, NE, NW, SE and SW. If the chosen direction is one of E, W, S, or N, the object will move (across X or Y axis) by 1 pixel per frame in that direction. Otherwise, the object will move $\sqrt{2}$ = 1.414 pixels per frame along the hypotenuse, corresponding to 1 pixel across both X and Y axes. This means that on an average the object moves 1.207 [= (1 $\times$ 4 + 1.414 $\times$ 4)/8] pixels per frame. The object keeps in moving in the current direction until it collides with another object or with the game border, whereupon it moves in a new random direction. The game starts when the user presses a "Start" button on the screen center. The game ends when the user finds the upright orientation of all the correct object and drags them onto their corresponding target(s), in which case a "Game Complete" message is provided.

To find the upright orientation of an object, the user right clicks inside the bounding box across the object, each right click rotates the object 45 degrees clockwise. When the user thinks that the object is in its upright orientation, the user left clicks on the object, drags the object and drops it by releasing it inside the bounding box across the respective target. The game must be successfully completed within a fixed time (we allow 60s); the user gets feedback on the correctness of every drag-drop, as the object disappears if it was in its upright orientation and dropped to its corresponding target.

Our Mix DCG implementation is in line with the general implementation of DCG CAPTCHAs with some modifications:

- The game and objects sizes are bigger to allow the user to comprehend the images.
- Right clicks rotate the objects.
- The check of correctness of each drag and drop is moved to the server. Whenever the user drops an object, the object identifier, its orientation and the drop location are sent to the web server, which is responsible for performing the verification of the correctness of the drop operation (i.e., the check whether the object is on its upright position and dropped to its corresponding target). After the user drags and drops all the answer objects to their corresponding targets, the game code sends to the server the log of the game play which contains objects' locations and orientations, the mouse location and status (up/down) at each time interval to the server. The server utilizes this log to detect relay attack as we will later demonstrate in Section 5.5.



(a) Brands (b) Animals

(c) Professions

FIGURE 5.3: Mix DCG Sample Instances. Targets, on the Left, are Static; Moving Objects, on the Right, are Mobile. The User Task is to Orient (by Right Clicking) and Then Drag-Drop a Subset of the Moving Objects (Answer Objects) to Their Corresponding Targets.

We implemented six instances of Mix DCG that can be categorized into three categories (two instances of each category), described below. A sample of each of the implemented categories is shown in Figure 5.3.

- **Brands:** the targets are popular worldwide brands and the objects are commercial products (e.g., Mac and MacBook Pro) .
- **Animals:** the targets are real animals and the moving objects cartoon animals (e.g., Dog and Scooby Doo).
- **Professions:** the targets are professionals and the moving objects are tools (e.g., Housekeeper and Vacuum Cleaner).

5.3 Security Against Automated Attacks

The security of our Mix DCG CAPTCHA construction is based on the security of DCG CAPTCHA, What's up CAPTCHA and SEMAGE CAPTCHA. To solve Mix DCG CAPTCHA, the bot needs to perform the following tasks:

1. Extract the moving and the target objects from the CAPTCHA challenge,
2. Understand the content of the images of the extracted objects,
3. Solve the semantic relationship in the CAPTCHA challenge (for each of the target object, the bot should find the object from the moving objects that relates to it semantically), and
4. Find the upright orientation of the answer objects.

For the first task, the bot can find the moving area and subtract it from a frame of the CAPTCHA challenge, and then the subarea with the maximum subarea can be recognized as the target area. Subsequently, by subtracting the moving area of any of the challenge's frames, the moving objects can be extracted as discussed [83]. The second and third tasks are similar to the tasks required to be performed by the bot to solve the SEMAGE challenge and the forth task is similar to the task needed to solve the What's up challenge. Image recognition, solving semantic relationships and object orientations are hard AI problems [1, 2]. Thus, performing the second, third and fourth tasks will be challenging for the bot, which will defeat automated attacks against our Mix DCG CAPTCHA. Below we discuss other relevant security considerations against different types of automated attacks:

Security Against Random Guessing Attacks: Assuming the bot can figure out the positions of the moving and target objects, for each of the target objects, the bot needs to pick one of

the moving objects (probability 1/6) and guess the upright orientation of it (probability 1/8). Therefore, the success probability for random attack against Mix DCG challenge with 3 targets and 6 moving objects equals $(1/8)^3 \times (1/P(6,3)) = 1.63e^{-5}$. This probability is extremely low (much lower than the benchmarks established for CAPTCHA security [67]), making Mix DCG CAPTCHAs highly secure against random guessing attacks.

Security Against Dictionary-based Attacks: To build a dictionary, the bot would require a total of 120 drags and drops to make sure the game is successfully completed. To prevent dictionary-based attacks, therefore, the CAPTCHA server can simply limit the number of allowed drags and drops to prevent the bots from building a dictionary of the challenges. This would not affect the legitimate users as our usability study (presented in Section 5.4) shows that legitimate users only require at most 13 drags and drops to play the Mix DCG games successfully. The CAPTCHA provider can further use a large and dynamic dictionary to reduce the attack probability.

Security Against Machine Learning Attacks: Following the strategy mentioned in [1], the Mix DCG CAPTCHA provider should select the images that are hard for machine learning algorithms to identify the underlying correct orientation. The CAPTCHA provider should test the accuracy of the state-of-the-art machine learning algorithms in identifying the upright orientation and only use the images that these algorithms fail at to orient correctly. Building a machine learning mechanism to identify the relationships between the moving objects and targets is a hard problem as the CAPTCHA provider can utilize wide variety of images. The accuracy of any classifier decreases with increase in the number of classes (in this case, different types of images).

5.4 Usability

In this section, we provide a detailed description of the two usability studies that we conducted to measure legitimate user's game play performance of our developed instances of Mix DCG CAPTCHAs. The first study aims to measure the usability of Mix DCG on the traditional (desktop/laptop) platforms, while the second study aims to measure the usability of Mix DCG on mobile devices. Our hypothesis is that Mix DCG will have high accuracy and good user perceptions on both traditional and mobile computers. We utilized the Amazon Mechanical

Turk (MTurk) service to recruit participants for the web-based study, and we recruited students from our University for the mobile-based study. Our University's Institutional Review Board approved the studies. The participation in the studies was voluntary, and standard ethical procedures were fully followed, e.g., participants being informed, given choice to discontinue, and not deceived.

5.4.1 Study Design

Each Mix DCG challenge is of size 500 $\times$ 300 pixels. For each of the answer objects in the game, the user has to keep clicking on the answer object till it is in its upright orientation, each click rotate the object 45 degrees, and then drag-drop it to its corresponding target objects. The user has to complete the task, find the upright orientation of all the answer objects and drag-drop them to their corresponding targets, within a time limit of 60 seconds, otherwise a timeout massage is displayed and the challenge is considered unsuccessful.

We asked each participant to play the six instances of Mix DCG challenges (explained in Section 5.2.2). The order of presenting the instances followed a random order. 100 MTurk workers participated in our web-based study and 20 students from our university participated in the mobile-based study. We paid each of the MTurk workers \$0.5 for their effort. The mobile phone employed in the study was a Samsung Galaxy S1, which was chosen as a representative of a basic smartphone–if the usability were good on this phone, it would probably be better on more current and upcoming phone models having larger touchscreens.

We first subjected the participants to a consent form. Then, we asked them to fill-out a demographics form, solve the six Mix DCG challenges (i.e., filling the demographics form emulated the primary task, while solving the MIX DCG challenges emulated the secondary task). At the end of the study, we asked the participants to rate their experience with Mix DCG CAPTCHA on a scale of 1 to 5 (we term this metric as "personal rating"), where a higher number of points means a better system, and fill-out a survey form about their experience. The survey contains the 10 System Usable Scale (SUS) [80] standard questions, each with 5 possible responses (5-point Likert scale, where strong disagreement is represented by "1" and strong agreement is represented by "5"). SUS is a standard questionnaire to measure the usability of software, hardware, cell phones and websites, and it has been deployed in many prior security usability studies.

TABLE 5.1: Participant Demographics in Our Different User Studies

	Usability		**Relay**	
	Web	Mobile	LSHL	HSLL
Participants' Location		Lab	India	USA
Number of Participants	100	20	20	40
Gender (%)				
Male	54	70	95	62.5
Female	46	30	5	37.5
Age (%)				
18-24	11	30	40	12.5
25-34	54	70	50	52.5
35-50	24	0	10	27.5
>50	11	0	0	7.5
Education (%)				
High school	32	20	10	47.5
Bachelor	48	30	30	37.5
Masters	20	50	60	15
PhD	0	0	0	0

The demographics of the study participants are shown in the second and third columns of Table 5.1. The user pool for the web-based study consisted mostly of educated individuals with almost equal number of female and male (54% male and 46%female), belonging to various backgrounds. Most of the participants were between 25 and 50 years old (11% 18-24, 54% 25-34, 24% 35-50 and 11% >50)[1]. The user pool for the mobile-based study consisted undergraduate and graduate students in our University. The participants were between 18 and 34 years old and the majority of the participants were male (70% male and 30%female). Such participant samples are in line with those reported in many prior CAPTCHA usability studies (e.g., [13, 62]).

5.4.2 Study Results

We evaluated the usability of the Mix DCG CAPTCHA with respect to: (1) error rate, (2) solving time, (3) user experiences (SUS score and personal rating) and (4) learnability, all of which are described below. The results are summarized in Table 5.2, 5.3 and 5.4.

[1]The age distribution of the participants is inline with the distribution of Internet users http://www.statista.com/statistics/272365/age-distribution-of-internet-users-worldwide/.

TABLE 5.2: Web-Based *Usability* Study Quantitative Results

	Overall Solving Time (sec.) mean (std)	**Successful Solving Time (sec.)** mean (std)	**Error rate**	**Number of Drags** mean (std)
Mix DCG	27.03 (13.92)	24.55 (10.96)	0.07	4.34 (2.10)
Brands	25.42 (15.36)	21.99 (11.34)	0.09	4.33 (2.58)
Animals	27.28 (13.00)	25.38 (10.62)	0.06	4.39 (1.86)
Professions	28.40 (13.11)	26.20 (10.45)	0.07	4.30 (1.77)

TABLE 5.3: Mobile-Based Usability Study Quantitative Results

	Overall Solving Time (sec.) mean (std)	**Successful Solving Time (sec.)** mean (std)	**Error rate**	**Number of Drags** mean (std)
Mix DCG	28.03 (10.44)	27.21 (9.21)	0.03	3.44 (0.89)
Brands	24.22 (8.70)	24.22 (8.70)	0.00	3.23 (0.47)
Animals	31.49 (11.70)	29.99 (9.96)	0.05	3.68 (1.10)
Professions	28.37 (9.36)	27.56 (7.97)	0.03	3.41 (0.93)

5.4.2.1 Error Rate

The error rate represents the fraction of the challenges that were not solved successfully by the participants. The forth column of Table 5.2 and 5.3 show the error rate for solving Mix DCG. Mix DCGs had low average error rate (7%) for both the web-based study and (3%) for the mobile-based study. The low error rate in Mix DCG may be attributed to the interactive feedback that DCG provides. Whenever the participant drag-drops a correct object to its corresponding target and if it is in its upright orientation, the object disappears, which informs the user that he performed a correct drag-drop. Also, the low error rate may be due to the high level of engagement and the visual feature of Mix DCG as the objects are not distorted, unlike traditional text CAPTCHAs.

Further, we analyzed the error rates per game category for Mix DCGs. In the web-based study, the minimum error rate was for the Animals game (6%) and the maximum was for the Brands game (9%). The higher error rate for the Brands game may be because some of the participants, being located outside the US, could not recognize some of the brands we used in our instances. However, all the participants in the mobile-based study, being based in the US, were familiar with the brands we used in our study, yielding the error rate for the brands game to decrease to 0%.

Next, we analyzed the number of drag-drops performed by the participants in solving Mix

DCG challenges (fifth column of Table 5.2 and 5.3) . We noticed that on an average the users performed around four drag-drops in the web-based study, a minimum of three drag-drops is required to complete any challenge. Around half of the challenges were solved with exactly 3 drag-drops, however some of the challenges were solved with upto 13 drag-drops. The average number of drag-drops performed by the users in the mobile-based study seem less than its correspondent in the web-based study, this can be because the majority of the mobile-based study participants were young.

5.4.2.2 User Experience

TABLE 5.4: User Experience Results

	SUS (out of 100; (higher the better)) mean (std)	**Personal Rating** (out of 5; (higher the better)) mean (std)
Web-based Study	68.05 (18.46)	4.38 (1.05)
Mobile-based Study	80.00 (22.00)	4.44 (0.58)

The second column of the Table 5.4 shows the SUS scores. In our survey design, in order to assure that the participants read the questions and did not answer them randomly, we added a question asking the participants to select a specific rating. We ignored the responses of 9 users in the web-based study and 3 users in the mobile-based study who did not answer this question correctly. We found the average SUS score for the Mix DCG to be 68.05 in the web-based study and 80 in the mobile-based study. Considering that the average SUS score is 68 [84], our Mix DCG is considered above average for both studies.

The third column of the Table 5.4 shows the personal rating corresponding to Mix DCG CAPTCHA. The results show that the participants gave Mix DCG high personal rating (i.e., on average 4.38 and 4.44 out of 5, respectively) in both the web study and the mobile-based study.

5.4.2.3 Solving Time

We calculated the solving time as the time taken by the participants to solve each challenge. We started measuring the timing from the time the challenge is displayed till the participants find the upright orientation and drag-drop all the answer objects to their corresponding target objects.

The average solving time is shown in the second column of Table 5.2 and the average time corresponding only to the challenges solved successfully is shown in the third column of Table 5.2. The time taken to solve Mix DCG challenges is less than 30 seconds on average. Similar and even higher solving times have been reported for many other CAPTCHA schemes, such as [23, 85].

Furthermore, we checked the average solving time for each of the Mix DCG categories. For the web-based study, the minimum time was for the Brands and the maximum was for the Profession. Comparing the solving times of the three categories using Friedman test shows statistical significant $(\chi^2(2) = 19.813), p < 0.001)$[2]. Further analyzing the solving time with pairwise Wilcoxon Signed-Rank with Bonferroni correction shows statistical significant difference between Brands and Animal and Brands and Profession pairs ($p < 0.001$).

For the mobile-based study, the minimum time was for also for Brands, but the maximum was for Animals. Comparing the solving times of the three categories using Friedman test shows statistical significant $(\chi^2(2) = 7.737), p = 0.021)$. Further analyzing the solving time with pairwise Wilcoxon Signed-Rank with Bonferroni correction shows also statistical significant difference between Brands and Animal and Brands and Profession pairs ($p = 0.001$ and $p = 0.009$, respectively).

5.4.2.4 Learnability

We compared the users' performance in solving the first and last (sixth) challenges (Table 5.5 summarizes the results). The results show improvement in the user performance for both of the studies in terms of solving time. Comparing the solving time of the first and last attempts for web-based study using pairwise Wilcoxon Signed-Rank, we found statistical significant difference ($Z = -4914, p < 0.001$).

Similarly, comparing the solving time of the first and last attempts for the mobile-based study using pairwise Wilcoxon Signed-Rank, we found statistically significant difference ($Z = -2.797, p = 0.005$).

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

TABLE 5.5: Comparing the user performance in solving the first and last challenges

	First Challenge	**Last Challenge**
	Solving Time (sec.) mean (std)	**Solving Time (sec.)** mean (std)
Web-based Study	29.24 (11.35)	21.51 (8.9)
Mobile-based Study	32.36(11.26)	23.34 (5.86)

5.4.3 Summary of Results

Although the participants took slightly long time to solve Mix DCG challenges, they found it to be appealing as reflected in their SUS score and personal ratings. Moreover, the error rate for solving Mix DCG is considerably low, especially when compared to most of deployed text based CAPTCHAs. The average error rate for text-based CAPTCHAs has been reported to be over 0.13 [86]. Mix 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 with a comparable average solving time of 25.2 seconds [62]. Moreover, the error rate of Mix DCG is comparable to that of S-DCG, which is shown to be insecure against automated attacks (the error rate for S-DCG is 0.05 on average. The results also show that the user performance improves with solving multiple Mix DCG challenges, which may result in a significant decrease in the solving time, as users become more and more familiar with solving these CAPTCHAs. These overall positive usability results for Mix DCG CAPTCHAs can be attributed to the level of engagement in the Mix DCG challenges, their visual features, interactive feedback and avoidance of distortion.

5.5 Security Against Relay Attacks

In this section, we explore the security of Mix DCG against relay attacks. We first provide the details about our relay attack study design and present the study results. Then, we explore the ability of detecting relay attack based on the difference in the game play performance between the legitimate users in the web-based usability study (Section 5.4) and human-solvers in the relay attack study. Our hypothesis is that Mix DCG offers a better level of resilience against relay attacks compared to S-DCG, as the former requires more interaction between the users

and the CAPTCHA challenges (the user has to orient the answer objects in addition to drag-drop the answer objects to their corresponding targets).

5.5.1 Study Design

The static relay attack (reviewed in Section 3.4.2) against DCG CAPTCHAs in general has been shown to be ineffective. This result generalizes to Mix DCG CAPTCHAs too. Therefore, we only study the security of Mix DCGs against the stream relay attack. We conducted two studies to evaluate the security of Mix DCG against the stream relay attack introduced in Section 3.4.3).

To investigate the stream relay attack against Mix DCG, we utilized the MTurk platform to recruit the participants in two separate studies. The only difference between the studies is the location of the participants. In the first study, termed Low-Speed High-Latency (LSHL), we recruited participants from a developing country (India), where we expect the users to have a slow Internet connection, and they reside in far proximity of the attacker (residing in the USA). In the second study, termed High-Speed Low-Latency (HSLL), we recruited participants from a developed country (USA), where we expect the users to have a fast Internet connection, and they reside in near proximity of the attacker (USA)[3]. In both studies, the participants were asked to solve Mix DCGs challenges streamed to them in real-time from a server running in Amazon cloud (zone: us-west-2b) using VNC steaming software. We paid each of the MTurk workers $1.5 for their effort. Our University's Institutional Review Board approved the studies.

In the study, each participant was asked to connect to our server on Amazon cloud via VNC viewer, and fill a demographics form and play the six instances of the Mix DCG CAPTCHA. We recruited 20 MTurk workers from India for the first study, and 40 MTurk workers from the US for the second study. The participants in the LSHL study were mostly male (95% male and 5%female), aged between 18-50 (40% 18-24, 50% 25-34 and 10% 35-50). The participants in the HSLL study were majorly male (62.5% male and 37.5% female), aged between 18 and above 50 (12.5% 18-24, 52.5% 25-34, 27.5% 35-50 and 7.5% $>$ 50). The participants in both studies were educated and from various backgrounds, like in our usability study. Such a similarity in the demographics of participants between the two study settings (relay attack and usability) allows us to compare the results of the two settings fairly following a between-subjects design.

[3]The average Internet speed is 2.0 MBit/s in India and 11.1 MBit/s in USA [82].

TABLE 5.6: Relay Attack Study Results (*HSLL* setting). None of the participants in the *LSHL* succeeded in solving any of the challenges.

	Overall Solving Time (sec.) mean (std)	**Successful Solving Time (sec.)** mean (std)	**Error rate**	**Number of Drags** mean (std)
Mix DCG	58.66 (4.59)	47.28(6.99)	0.89	3.31 (0.54)
Brands	58.28(5.53)	44.07 (9.11)	0.88	3.25 (0.46)
Animals	58.64(4.61)	46.10 (6.57)	0.90	3.25 (0.46)
professions	59.07 (3.33)	50.02 (4.54)	0.90	3.50 (0.76)

5.5.2 Study Results

We evaluated the performance of the participants with respect to: (1) solving time and (2) error rate, and compared it with the performance of legitimate users in the web-based usability study. All the participants in the LSHL study failed to solve any of the Mix DCG challenges, i.e., the overall error rate equaled 1. This is attributed to the slow connection speeds and far proximity of these participants which may have introduced jitters and prevented them from orienting and/or dragging the objects correctly. The results for the HSLL study are summarized in Table 5.6 and explained in the following subsections.

5.5.2.1 Error Rate

The fourth column of Table 5.6 shows the error rate committed by the participants in the HSLL relay attack study. The participants failed to solve higher number of Mix DCG compared to the participants in the usability study. 24 participants could not solve any of the challenges, and the other 16 participants could solve 1.63 challenges on an average. The overall error rate was 0.89. The minimum error rate was for the Brands category and the Animals and Professions have equal error rate. The quality degradation of the streamed Mix DCG challenges makes the challenges much harder to solve by the remote human-solvers compared to legitimate users who play the games rendered locally.

Next, we analyzed the number of drag-drops performed by the participants in solving Mix DCG challenges (fifth column of Table 5.6). Using the Mann-Whitney U test, we found statistical difference between the number of drags performed in the relay attack and the usability study ($Z = -2.817, p = 0.005$).

5.5.2.2 Solving Time

The third column of Table 5.6 shows the average time taken by the participants in the (HSLL) relay attack study to successfully solve the Mix DCG challenges. On an average, the participants took around 47 seconds to solve a challenge which is much longer than the time taken by the participants in the usability study (around 25 seconds). Comparing the solving times in the relay attack study with the solving times in the usability study, using the Mann-Whitney U test, we found this difference to be statistically significant ($Z = -3.005, p = 0.003$). This suggests that even when participants are able to solve the challenges in the relay attack setting, they will take much longer to do so.

5.5.3 Relay Attack Detection

Next, we explored the ability of detecting relay attack on Mix DCG based on the differences in the game play performance between the participants in the usability study and relay attack study. As explained in Section 5.2, we logged the participants interaction with the challenges. At each time interval, we recorded the mouse position and status. Moreover, for each object, we recorded whether the object is being dragged or not at each given time.

For each of the log files, we extracted a total of sixteen features. The extracted main features are: the solving time, the number of drags, the number of attempts (i.e., clicks that do not correspond to object drag) and the longest duration of the attempts. The additional features are: the time, the distance, the speed and the acceleration of the mouse movement for each of the mouse status corresponding to drags, attempts and when the mouse button was not pressed.

Then, we built a data file that contains the extracted features from the successfully solved Mix DCG challenges in the (HSLL) relay attack study and a randomly selected equal number of successfully solved Mix DCG challenges in the usability study. We tested various machine learning algorithms on all of the subset of features to find the best classifier along the best sub-features that can be utilized to detect the relay attack against Mix DCG. We developed a Java program that utilizes Weka library to test different classifiers across different features subsets that would result in best accuracy. We tested different machine learning algorithms provided by Weka: *Trees* – Logistic Model Trees, Random Forest and Random Tree; *Functions* – Logistics and Simple Logistic, and *Bayesian Networks* – Naive Bayes, on all features subsets.

In our classification task, the positive class corresponds to the legitimate user and the negative class corresponds to remote human-solver. Therefore, true positive (TP) represents the number of times the legitimate user is accepted, true negative (TN) represents the number of times the remote human-solver is rejected, false positive (FP) represents the number of times the human-solver is accepted and false negative (FN) represents the number of times the correct user is rejected. As performance measures for our classifiers, we used Precision, Recall and F-measure (F1 score).

The classification results, obtained after running a 10-cross validation, are: Recall 0.96, Precision 1.00 and F-measure 0.98. These results suggest that our classifier can detect the HSLL relay attack 100% of the time while rejecting only around 4% of the legitimate users (The success rate of the legitimate user is 0.93 (1 - 4% (falsely rejected as a relay attacker) - 3% (legitimate user error rate))), within 2 trials the success rate is 99.51% and within 3 trials the success rate becomes 99.97%. The best sub features are the number of drags, the distance of attempts, the time drag and the acceleration of mouse movement while the button is not clicked, and the best classifier was Random Tree.

5.5.4 Summary of Results

Mix DCG CAPTCHA offers a high level of resilience against relay attacks. Similar to S-DCG, the dynamic nature of Mix DCG along with its requirement for multiple interaction between the user and the challenge make relaying Mix DCG to remote human-solver a challenging task. Moreover, as the quality of Mix DCG challenges degrades under the streaming-based attack, the remote human-solvers located in far proximity (LSHL) failed to solve any of the challenges and the remote human-solvers located in near proximity (HSLL) failed to solve most of the challenges (error rate around 90%). Furthermore, even when the remote human-solver could successfully solve the Mix DCG challenges, our proposed detection mechanism could detect the relay attack highly accurately as game play performance differs significantly between the remote human-solvers and the legitimate users.

5.6 Discussion

In this chapter, we carefully studied the integration of three categories of CAPTCHAs in order to create a usable and secure CAPTCHA. Image orientation and semantic matching CAPTCHAs

TABLE 5.7: Security of Various Categories of CAPTCHAs, Relevant to This Chapter, against Different Forms of CAPTCHA Attacks. Mix DCG is the First Known CAPTCHA Scheme That Can Resist All Three Forms of Attacks While Still Retaining a Good Level of Usability.

	Automated	Client-Side	Relay
Image Orientation [1]	✓	✓	✗
Semantic Matching [2]	✓	✓	✗
Simple DCG (S-DCG)	✗	✗	✓
Emergent Image DCG (EI-DCG)	✓	✗	✓
Mix DCG	✓	✓	✓

are secure against automated attacks and client-side attacks but vulnerable to relay attacks. The simple form of DCG CAPTCHAs are resistant to relay attacks but vulnerable to automated and client-side attacks.

Although EI-DCG is shown secure against automated and relay attacks, it suffers from low usability and does not offer security against client-side attack[4]. As shown in Chapter 4, the emergent effect added to simple DCG made EI-DCG challenges harder to solve for legitimate users. This was reflected in a low average SUS score of around 56, which is considered much below the benchmark average score for usable computer systems (68) [84], and high error rate of around 12%.

In contrast, our proposed Mix DCG CAPTCHA scheme is secure against all forms of attacks (the comparison among different relevant CAPTCHA categories is outlined in Table 5.7). Its security against automated attacks is based on the security of both image orientation and semantic matching CAPTCHAs (which are based on hard AI problems). In other words, the attacker will have to break *both* notions to be able to break Mix DCG CAPTCHA. Its security against relay attacks relates to the security of simple DCGs against relay attacks. The level of interaction between the user and the Mix DCG challenge is actually higher compared to that in simple DCG, i.e., the user has to orient the objects in addition to dragging and dropping them. This resulted in improving the security against relay attacks. The error rate for the remote human-solvers increased from 22% for S-DCG to 93% for Mix DCG. Also, the accuracy (i.e., F-measure) of attack detection increases from 0.77 to 0.98. Mix-DCG also offers security against client-side attacks since the CAPTCHA verification task is moved to the server.

Security is *not* the only attractive feature of Mix CAPTCHAs – they also offer high accuracy and good overall user experience for legitimate users. Although the participants in our usability

[4]Incorporating image orientation and image semantics features to EI-DCG may not offer any viable usability due to the presence of emergence effect.

study took relatively long time to solve Mix DCG challenges, they were able to solve most of the challenges successfully.

Each Mix DCG challenge is of size 500 $\times$ 300 and the frame rate of the challenges is set to 40 frames per second. Under the relay attack setting, using as less as 2 bit per pixel (black and white) will require the connection speed of 12 Mbit/s between the attacker and the solver if no compression is applied to the data sent between the attacker machine and the human-solve machine. Utilizing RealVNC for transmitting the data allows us to benefit from the data compression it applies on the transmitting packets between the VNC server and VNC viewer. First, RealVNC sends only the modified pixels between the current frame and the previous frame rather than sending the whole frame. Moreover, RealVNC viewer allows the user to picks the compression level applied, varying from best compression (8 bits) to best quality (all available colors with minimum, medium or maximum compression). To calculate the minimum bandwidth required between the attacker and the human-solver to transfer the challenges without any jittery effects, we performed an experiment in which we connected two PCs over a LAN, installed RealVNC server on one of them and RealVNC viewer on the other. Then, we varied the RealVNC compression level and used Wireshark to capture the packets sent between the two PCs in all of the cases. We found that the average Mbit/s for the getting all colors with minimum compression is 15.60 Mbit/s, 2.88 Mbit/s for getting all colors with maximum compression and 2.09 Mbit/s for the default settings (256 colors).

The above experiment shows that to successfully transmit all the game frames between the attacker and the human-solver, the minimum required bandwidth is 2.09 Mbit/sec. If the connection bandwidth is slower, some of the frames would not be sent to the human-solver causing a jittery rendering of the game, and therefore the human-solver will have difficulty in solving the challenges successfully. The average connection speed is 2.0 Mbit/s in India. Moreover, the long distance between the attacker (our server machine is located in USA) and the human-solver slows down the connection speed, all the participants in India could not connect to our server with high speed and therefore failed to successfully solve the challenges. On the other hand, the average connection speed in USA is 11.1 Mbit/s. Moreover, the distance is much less between the attackers and the human-solvers in the HSLL streaming-based relay attack study than its corresponding in the LSHL study. This explains why some of the human-solver were able to successfully solve some of the challenges. This justifies the results of the user studies

presented in the Section 5.5.

We also tested TeamViewer, another screen-sharing application, and obtained similar results as for RealVNC. TeamViewer requires a bandwidth of 4.6 Mbit/s between the attacker and the human-solver machine to transmit all the challenge frames. Therefore, we conclude that the results of our relay attack studies are not limited to RealVNC.

A potential attack strategy against Mix DCGs could be a hybrid approach in which the attacker extracts the moving objects and the targets from the challenge, and sends them to the human-solver in a framework that allows the user to interact with the objects (i.e., rotate and drag-drop the answer objects to their corresponding targets). After the human-solver believes that he is done with solving the challenge, he sends his answer to the attacker. The attacker then has to find the answer objects in the current frame, click on them as many times as the human-solver did and drag-drop them to their corresponding target as provided by the human-solver. This attack strategy is viable but can be effectively defended by adding a simple background to the challenges that would make the extraction of the moving objects from the challenge a time consuming task (the extraction of the foreground objects from the game frames takes on average 30.9 seconds in the attack reported in [83]) without degrading the usability level. Therefore, to solve the challenge, the attacker would need around 30 seconds to extract the objects and send them to the solver, who in turn would take approximately the same time as the time taken by legitimate user since the task is similar (on average around 24 seconds) and then search for the objects in the current frame and solve the challenge. This attack can be detected by checking the time the first drag-drop was performed or reduce the time out slightly, or by utilizing dynamic background which may harden the object extraction.

Recently Google has released "No CAPTCHA reCAPTCHA" as a method to remove the usability burden involved in solving CAPTCHAs. No CAPTCHA reCAPTCHA is based on risk analysis that considers the user's engagement with the CAPTCHA to determine whether the user is a human or not. The task of the user is reduced to clicking on a checkbox that indicates that he is a human. However, whenever the risk analysis cannot confidently predict whether the user is a human or not, it needs to fall-back to a CAPTCHA scheme. Our Mix DCG CAPTCHAs can very well serve the role of this fall-back and can be integrated with this Google system.

One potential practical challenge in deploying Mix DCG CAPTCHA scheme pertains to the selection of the moving and target objects. The moving objects should be hard for machine

learning algorithms to orient but easy for human to do so. The objects and the semantic relationship between the targets and answer objects should be easy for human to comprehend and identify, but hard for bots. The selection of such images would require some manual work to build the images dictionary that is used to create the CAPTCHA challenges, which may slow down the creation of a large number of challenges (e.g., for a web service with huge user base).

5.7 Conclusion

In this chapter, we proposed CAPTCHA fusion to defeat different forms of known CAPTCHA vulnerabilities. We studied the fusion of three CAPTCHA schemes based on dynamic cognitive games (DCG), image orientation and image semantics. Specifically, we introduced a new class of CAPTCHAs, Mix DCG, and instantiated it with a fusion of a simple form of DCG CAPTCHA, What's up CAPTCHA, and SEmantically MAtching imaGEs (SEMAGE). Simple DCGs inherently provide security against relay attacks, but by integrating them with What's up CAPTCHA and SEMAGE CAPTCHA also improved their security against various types of automated attacks.

The Mix DCG CAPTCHA security with respect to automated attacks relies upon the security of What's up and SEMAGE CAPTCHAs, which are both built on fundamental AI problems. We conducted two user studies to evaluate the usability and the security of Mix DCG against relay attacks. The results of the studies show that Mix DCG is usable (significantly more usable than a widely deployed CAPTCHA scheme) and highly resilient to relay attacks. Mix DCG is the first CAPTCHA scheme, to the best of our knowledge, that can simultaneously resist automated attacks, relay attacks and client-side attacks, while providing a reasonably good user experience.

CHAPTER 6

SMASHED: SNIFFING AND MANIPULATING ANDROID SENSOR DATA

In the previous chapters, we show that interactivity can enhance the security and the usability of Human-Machine authentication. To show the importance of adding interactivity and randomization to Human-Human authentication, we developed a framework that can compromise the security of most well-known deployed and proposed Human-Human authentication (i.e., password and behavioral biometrics) on Android devices. Later in Chapter 7, we introduce a behavioral biometrics that can resist the proposed attack.

Chapter Organization: Section 6.1 provides background on Android security Model. Section 6.2 presents our proposed attack *SMASheD* design, implementation and threat model. Section 6.3 and Section 6.4 present *SMASheD* attacks on various authentication systems. Section 6.5 suggests methods for mitigating the attack. Section 6.6 concludes the chapter.

6.1 Background: Android Sensor Security Model

Android's core security principle is to protect user data, system resources and apps from malicious apps [87]. Android utilizes the Linux approach of process isolation to enforce the isolation of apps and operating systems components. This isolation is achieved by assigning each app a unique User Identifier (UID) and Group Identifier (GID) at the app installation time. Therefore, each app is enforced to run in a separate Linux process, called *Application Sandbox*, and the Linux process isolation ensures that an app cannot interfere with other apps or access system resources unless permissions are explicitly granted. In order to allow apps to communicate with each other and access system resources, Android provides a secure Inter-Process Communication (IPC) protocol.

Discretionary Access Control (DAC) is the typical access control employed in Linux. In DAC, the owner/creator of the data sets the access permissions of the data to three types of users: the owner, the users in the same group and all other users. When an app is installed, Android creates a home directory for the app (i.e., /data/data/app-name) and allows only the owner to

read from and write to this directory. The apps signed with the same certificate are able to share the data among each other.

File system permissions are also used to restrict the access of system functionality. For example, /dev/cam permission is set to allow only the owner and the users in the camera group to read and write to the camera sensor as shown in Listing 6.1. When an app requests the CAMERA permission, and if the permission is granted, the app is assigned the camera Linux GID, which would allow it to access /dev/cam. The mapping between the Linux groups and permission labels are set in platform.xml, and ueventd.rc is responsible for setting the owners and groups for various system files.

Some Android resources do not require any permission. In particular, reading motion, position and environmental sensors is globally permitted. Most of the other resources require read-write permissions, and these permissions have four levels:

1. *Normal*: The app needs to request the access, however, the system grants the permission automatically without notifying the user (e.g., vibrate).

2. *Dangerous (protection level 1)*: The system grants the permission to the app only if the user approves granting this permission (e.g., accessing camera, microphone, or GPS).

3. *Signature (protection level 2)*: The system grants the permission to the app only if the requesting app is signed with the same certificate as the app that declared the permission, without notifying the user (e.g., allowing two apps signed by the same developer to access each other components, inject event).

4. *SignatureOrSystem (protection level 3)*: The system grants the permission only to the apps that are in the Android system image or that are signed with the same certificate as the app that declared the permission (e.g., system reboot).

In any Linux system, an executable runs with same permission as the process that has started it. ADB shell is already assigned to several groups (graphics, input, log, adb, sdcard_rw, etc). Therefore, any executable that starts through the ADB shell is granted the same level of access to the resources which belong to any of these groups. As shown in Listing 6.1, since the directory "/dev/input/*" which contains the sensor files, belongs to "input" group, and the ADB shell has read-write access to all the resources associated with "input" group, any executable

that is initiated by ADB shell can read from and write to the "/dev/input/*" resources. This is the key idea upon which our *SMASheD* framework is based, allowing us to sniff and manipulate many of the Android's sensors.

LISTING 6.1: ueventd.rc File

```
...
/dev/input/*   0660   root       input
/dev/eac       0660   root       audio
/dev/cam       0660   root       camera
...
```

6.2 Smashed Design, Implementation and Threat Model

In this section, we explain the design, implementation and threat model of our proposed *SMASheD* framework.

6.2.1 Design Overview

As mentioned in Section 6.1, the current Android security model considers many resources as sensitive and thus limits the access of these resources only to the apps that are signed by the system (protection level 3 for the permissions declared by the system and protection level 4). These protected resources include: injecting user events into any window (INJECT_EVENTS), taking screen shots (READ_FRAME_BUFFER), and reading system log files (READ_LOGS). However, Android allows access to these resources through the ADB shell for development purposes, by assigning the ADB shell to the groups that can access these resources. For example, the ADB shell is assigned to the input group which allows any process with the ADB shell privilege to read from and write to any of the files in the /dev/input/ directory. This directory contains the files associated with user input, motion, position and environmental sensors.

Moreover, Android's current directory structure has the /data/local/tmp/ directory which is assigned to shell user and shell group, and gives read, write and execute permission to the shell user and any user in the shell group. This folder allows the user to run executable files on their Android devices through ADB shell.

Many developers have exploited these capabilities given to the ADB shell to grant permissions to their apps that are not otherwise allowed. This ADB workaround is performed by

developing a native service, pushing it into the /data/local/tmp/ directory and running the service through the ADB shell. This way the native service grants all the permissions that are granted to the shell. Finally, to allow other apps to communicate with the service, both the app and the service open sockets and communicate through it. This approach has been utilized by many apps that are already published in Google Play Store such as apps that allow the users to take screenshots programmatically [88], sync and backup [89], USB tethering [90], and touch record/replay [91].

The above design allows any app with only the INTERNET permission to communicate with the service. Hence, the app with only the INTERNET permission will obtain access to the resources that the service provides *without the user's knowledge*. This vulnerability has been explored in [92], focusing mainly on screenshot apps published in Google Play Store. The authors developed an app, Screenmilker, which communicates with the native services of many screenshots apps. They showed that Screenmilker is able to collect user's sensitive data, such as user's credentials on many banking apps by sending requests to the screenshot's native service to take screenshots while the user is inputting her credentials.

In this chapter, we are exploring and extending this vulnerability further, and with potentially much broader consequences. We focus on INJECT_EVENTS permission. There are already some apps in Google Play Store, such as *FRep* – Finger Replayer [91], which allow users to record their touch interactions with their devices and replay them later. *FRep* has already been installed by 100,000 to 500,000 users[1]. These apps also utilize the ADB workaround, similar to the screenshot apps, in order to gain access to the read and inject touchscreen data. Also, as the communication between the touch repeater app and its native service is done through a socket, the native service becomes accessible to any app installed on the phone with only the INTERNET permission. Therefore, if the user installs any malicious apps with the INTERNET permission, these apps can also communicate with the service and read/inject touch events maliciously.

RERAN [94] presents one of the benign use cases for the apps that record and later replay sensor events by injecting the events into the sensors. Such apps enable the developers to

[1]Although the number of users is not that huge, it can be because the free version of *FRep* offers a limited functionality compared to its paid full version. However, many users seem to be interested in getting touch repeating functionality if it is free, which is apparent based on the number of users installing other touch repeaters that require root (e.g. [93] has 500,000 - 1,000,000) installations).

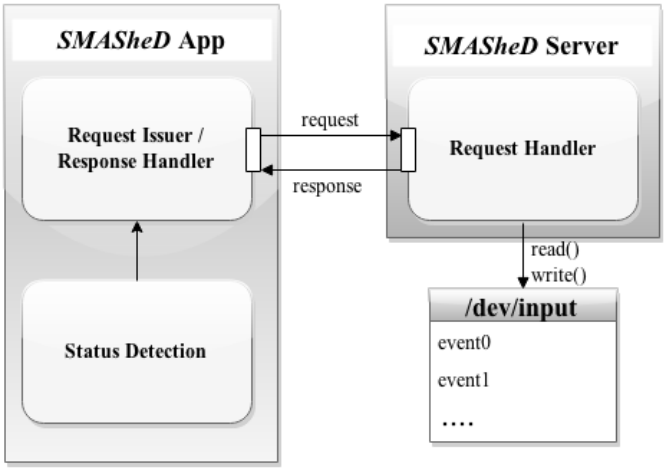


FIGURE 6.1: The Architecture of *SMASheD*

test their apps and reproduce errors. However, rather than implementing a native service for recording and replaying the sensor events, RERAN employed a different methodology for implementing their app. First, the user needs to connect her device to the PC. Then, she launches a terminal and executes the ADB *getevent* command. This command reads the sensor events and stores them in a file. Finally, the user pushes the file along with a native service to her device and runs the service. This service reads the sensor events from the file and injects them to their corresponding sensors. Once the phone is disconnected from USB, the service stops running.

We implement the *SMASheD* framework which encompasses three components: *SMASheD* server: a native service that provides the sensor data reading and injection capabilities, scripts: two simple scripts used to copy the *SMASheD* server to the device and to start the server, and *SMASheD* app: an app that runs a status detection module in the background, and depending on the phone's status and the desired functionality, it sends requests to the *SMASheD* server to read or inject sensor events. Figure 6.1 depicts the overall *SMASheD* architecture.

6.2.2 SMASheD Server

Our system works with the sensors whose events are made available to apps through low-level event interface and have files under the directory /dev/input/, and not through system services (e.g., camera, microphone, and GPS). This includes user input, motion, position and environmental sensors. For each of these sensors, a corresponding file named event$x$ exists in the directory /dev/input/. Android allows reading and injecting sensor events through ADB commands $getevent$ and $sendevent$, respectively.

Each hardware event generates multiple input events. Each input event encompasses $time$, $type$, $code$ and $value$.

- $time$ represents the time at which the event occurred.
- $value$ represents the value of the event.
- $code$ is the event code and it precisely defines the type of the event. For example, REL_X, REL_Y, REL_Z represent relative changes in X, Y and Z axes, respectively.
- $type$ is the event type, which groups the event's codes under a logical input construct. Each event type has a set of applicable event codes. For example, EV_ABS represents the absolute axis value changes, EV_REL represents the relative axis value changes. A special event type, EV_SYN, is used to separate input events into packets of input data changes occurring at the same moment in time.

For a complete list of the applicable events' types and codes, we refer the reader to linux/input.h[2].

LISTING 6.2: Sample Output from Running $getevent$ for a Single Press Release

```
[69934.435503] EV_ABS  ABS_MT_TRACKING_ID   0000038d
[69934.435533] EV_KEY  BTN_TOUCH            DOWN
[69934.435564] EV_ABS  ABS_MT_POSITION_X    000003b2
[69934.435564] EV_ABS  ABS_MT_POSITION_Y    00000607
[69934.435595] EV_ABS  ABS_MT_TOUCH_MAJOR   00000012
[69934.435595] EV_ABS  ABS_MT_TOUCH_MINOR   00000009
[69934.435625] EV_ABS  ABS_MT_WIDTH_MAJOR   00000002
[69934.435625] EV_ABS  003c                 ffffffa6
[69934.435778] EV_SYN  SYN_REPORT           00000000
[69934.452105] EV_ABS  ABS_MT_TOUCH_MAJOR   00000024
[69934.452105] EV_ABS  ABS_MT_TO UCH_MINOR  0000001b
[69934.452135] EV_ABS  ABS_MT_WIDTH_MAJOR   00000008
[69934.452135] EV_ABS  003c                 fffffffd
[69934.452166] EV_SYN  SYN_REPORT           00000000
[69934.462847] EV_ABS  003c                 00000000
[69934.462877] EV_SYN  SYN_REPORT           00000000
[69934.494371] EV_ABS  ABS_MT_TRACKING_ID   ffffffff
[69934.494402] EV_KEY  BTN_TOUCH            UP
[69934.494402] EV_SYN  SYN_REPORT           00000000
```

[2] https://github.com/torvalds/linux/blob/master/include/uapi/linux/input.h

As an example, a simple touchscreen press-release event generates around 19 input events. Listing 6.2 displays a sample output of executing $getevent$ command, pressing on the screen at point (946,1543), and then releasing the touch. BTN_TOUCH DOWN and BTN_TOUCH UP indicate the beginning and the end of the touch, ABS_MT_POSITION_X and ABS_MT_POSITION_Y represent the touch's x and y positions, respectively.

We implemented a native service designed in C with code similar to Android's $getevent$ and $sendevent$ for reading and injecting the sensor events. First, the service scans /dev/input/ directory to find out what sensors are available in the device. Although the file names in the directory are event0, event1, etc, we use *EVIOCGVERSION ioctl* function to retrieve the name of the sensor that corresponds to each file. To read from and write to the sensors' files, we use *read()* and *write()* functions.

To allow other apps to communicate with the service, the service creates a socket. The socket keeps on listening to the incoming requests. In the current implementation, the service accepts three kinds of requests: $read$, $stop$ and $inject$.

- $read$: The service reads the input events from all the sensors. We can limit the read to a subset of sensors to improve the efficiency. The service continues reading until it receives a $stop$ request.
- $stop$: The service stops reading the sensor events. Then, it either writes the events to a file, and sends the file name as a response to the request or sends all the read input events.
- $inject$: Inject needs to have a file name or a list of sensors events as an argument. The service injects the sensors events in the incoming list or in the file to their corresponding sensors files.

6.2.3 Scripts

We wrote two shell scripts to start the service. The first shell script is responsible for pushing the native service and the second script to /data/local/tmp/ folder on the device, and for starting the second script. The second script starts running the service. In this way, the service will run with the same privileges as the shell user. The service will then keep running until the phone is switched off or it gets killed by the user.

6.2.4 SMASheD App

We implemented an Android app, which only requires the INTERNET permission. The app connects to the *SMASheD* server through socket and sends requests to $read$ and $inject$ events. For example, it may send $read$ touch events when a banking app is open to retrieve the password input by the user.

In order to determine whether a specific app that the attacker might be interested in is running, our app has a service that starts when the app is launched and keeps running in the background. The service runs *ps* command periodically, every 100 ms, until the app that the attacker is interested in is launched (status detector shown in Figure 6.1). Once the app under attack is running and on the foreground, *SMASheD* app connects to the *SMASheD* server through socket and sends $read$ request with the list of sensors (e.g., touchscreen data only, all sensors, etc). Once the user exits the app or moves out of the app, *SMASheD* app sends $stop$ request to the *SMASheD* server. In case the purpose of reading is to replay the sensor events later in the same device, to reduce the communication between the *SMASheD* server and app, the *SMASheD* server stores the read events in a file and only sends the file name to the *SMASheD* app. Otherwise, the *SMASheD* server sends all the sensor events.

Also, the *SMASheD* app can send $inject$ request along with a list of sensor events to inject or a file name previously acquired from the service, whenever it wants to inject sensor events.

6.2.5 Threat Model

Our threat model is highly realistic, facilitated under three scenarios:

1. **Already Installed Benign ADB Services**: Apps that read and inject touch events (e.g., FRep [91]) are already available in Google Play Store and installed by many users. Given such an already installed benign app, our attacks that read/inject touch events work under the exact same threat model as [92] by using a malicious app that communicates with the service associated with the already installed app.
2. **Future Benign ADB Services**: Benign developers can publish an app/service that reads/injects sensors events for providing some benign functionality (e.g., debugging or testing). Once such an app is installed, attacker will launch our attacks similar to [92].

3. **Malicious ADB Services**: The attacker can create a benign-looking (malicious) screenshot app, adding read/inject sensor events functionality to its service. The attacker just needs to fool users into installing this app. Note that when user installs a service using ADB, he/she is not notified about the resources the service is accessing. Therefore, the user will not be able to differentiate between services that only take screenshots from services with added malicious functionality. Moreover, if the attacker can gain physical access to an unlocked Android device, the attacker can quickly install the malicious service on the device (e.g., in a lunch-time attack) [95].

The first and second threat model scenarios exploit the vulnerability of the services that expose their ADB functionalities to all the apps installed on the same device with INTERNET permission (same as [92]). The last scenario exploits Android's vulnerability of granting all the shell privileges to any service installed via ADB *without notifying the user*.

SMASheD works on unrooted devices, and does not require an infected PC (unlike [96]) or a constant connection between the device and a PC (e.g., unlike monkeyrunner [3]).

6.2.6 SMASheD Advantages

The *SMASheD* framework has several advantages:

- *Modularity and Expandability to Broad Attacks*: Once the user runs the script via the ADB shell to install the *SMASheD* server, the device becomes vulnerable to the attacks described in Section 6.4. The attacker only needs to modify the *SMASheD* app so as to send the $read$ and $inject$ requests according to the type of attack he wants to perform.

- *Stealthiness*: *SMASheD* is very stealthy and hard to get detected by anti-malware or intrusion detection systems. This is because *SMASheD* does not consume much energy compared to other apps that need to continuously monitor the sensors (such as activity trackers), and it can utilize other benign apps to send the collected information to a remote attacker. Moreover, *SMASheD* can change the phone settings, e.g., decrease screen brightness, mute sound and erase logs/traces to make the attacks "user-invisible", as we will explain in Section 6.4.3.

[3] http://developer.android.com/tools/help/monkeyrunner_concepts.html

- *Mutli-Device Applicability*: *SMASheD* is not limited to Android phones, but rather it works on any Android device. We tested reading and injecting sensor events on Android phones (Samsung S4, Samsung S5 and Motorola Droid X2), smartwatch (Samsung Gear Live) and Google Glass.

6.3 Key Logger

We will demonstrate how *SMASheD* can be used as a TouchLogger in order to sniff a user's sensitive information. According to Android security model, an app cannot read touch events performed by the user on other apps [97]. However, we will show how it is possible to infer the keys that the user has pressed, and therefore extract sensitive information efficiently and with 100% accuracy. We note that *SMASheD* is not only able to detect user key presses but it can also log any interaction of the user with the touchscreen, such as swiping, and zooming.

Many researchers have proposed mechanisms to infer the keys pressed by the users. These attacks require the user to install a malicious app on her phone. These methods range from utilizing motion sensors [98–100] and a combination of camera and microphone [101] to taking screenshot while the user is typing sensitive information, such as PINs or passwords [92]. However, these attacks usually have a relatively low accuracy and may be significantly affected by the way the user types and/or holds the phone. Taking pictures or recording audio may also trigger suspicion.

The *SMASheD* app can send *read* request to the *SMASheD* server to obtain all the events the user performs on the touchscreen. However, getting only the raw touch events is not enough to hamper the user privacy. Moreover, the attacker will be interested only in a small subset of these events. For example, an attacker will be interested in learning the password of the user on banking apps but not the input corresponding to user's interaction with a game.

According to the information the attacker wants to learn about the user, the attacker can modify the service in the *SMASheD* app so it sends *read* request when the app corresponding to the data the attacker wants to collect is launched. Moreover, if the attacker wants only to learn the keys the user presses while the app is running, he can send the *read* request when both the app and the keyboard are on the foreground.

To evaluate the ability of *SMASheD* to extract the username and password from various banking apps, we repurposed the original *SMASheD* app. When the *SMASheD* app is launched, it gets the list of installed apps on the device using *getPackageManager* API. Note that no permission is required to get the list of installed apps. Then, the *SMASheD* app looks for the apps that are already installed and are of interest to the attacker. The *SMASheD* app also finds the soft keyboards installed. The *SMASheD* app starts running its status detection service in the background, which regularly executes ps command to find out the list of running applications. Once any of the apps that the attacker wants to collect user data from appears in the output of the execution ps command, the service gets that app process ID, PID_{app}. The service also gets the process ID of the keyboard, PID_{kb} from the execution of the ps command. The service then executes $ps - tPID_{app}$ command, which returns the list of threads of that process; whenever an app is on the foreground, the list of threads of that app has a thread named "GL updater".

If the app is running the "GL updater" thread, the service also checks if the keyboard is running "GL updater" thread. If both the app and the keyboard app have "GL updater" thread running, *SMASheD* detects that both of them are in the foreground and sends $read$ request to the *SMASheD* server. When user exits the app or the keyboard (which can be detected by checking if the app is no longer in the list returned by executing the ps command, or if "GL updater" is not in the list of the thread running for either the app or the keyboard), the *SMASheD* app sends $stop$ request to the *SMASheD* server. As a response to the $stop$ request, the *SMASheD* server sends all the touch events to the *SMASheD* app. The *SMASheD* app parses the events and extracts the events with event type ABS_MT_POSITION_X and ABS_MT_POSITION_Y between BTN_TOUCH DOWN and BTN_TOUCH UP (Listing 6.2), if between the down and up events, the ABS_MT_POSITION_X or ABS_MT_POSITION_Y is missing, the value of it is same as its correspondent in the previous touch. Finally, it maps the x and y coordinates to keys (keyboard layout can be detected by determining which soft keyboard the user is using, the orientation of the device and the screen resolution) and sends the text typed (if the user is using a soft keyboard that is unknown to *SMASheD*, *SMASheD* can send the name of the soft keyboard app, the x, y coordinates of each touch and the screen resolution and orientation to the attacker and the mapping can performed offline), for example, to the attacker's web service via HTML request, or via other methods as we will discuss in Section 6.4.3.1.

We tested the above attack on some banking applications, such as Wells Fargo and Bank of

America apps, and we could learn the username and password with 100% accuracy. The attack was tested on Samsung Galaxy S5 with Android OS version 5.0 and Samsung Galaxy S1 with Android OS version 4.4.2, and can be easily adapted to other Android phone models and even other devices such as smartwatches. Our attack is an extremely powerful attack since all the input provided by the user can be precisely and stealthily stolen, raising significant concerns for users' security and privacy.

6.4 Attacks Using Smashed

In this section, we present various attacks on various authentication systems that can be performed based on the sensor data reading-writing capability provided by *SMASheD*. The entire spectrum of attacks that *SMASheD* can enable is possibly very broad. As such, our exposition is not exhaustive. However, we introduce some of the most interesting and potentially devastating attacks targeting both real-world and research authentication systems.

6.4.1 Overview of Attacks and Attack Presentation

To present our attacks, we follow an *empirical-analytical* methodology. That is, we show the implementation and evaluation of several of our attacks, and present the rest of the attacks in an analytical way. The attacks that have been implemented and empirically studied (4 in number) are:

- ***Touch-Logger:*** (1) Key-logger that can be utilized to extract banking apps usernames/-passwords, and (2) a general logger that can record user's touchscreen interactions.
- ***Touch Injection***: (3) Phone unlock, and (4) Phone unlock bypassing biometrics security, e.g., [102], by replaying user's unlock-pattern.

The other proposed analytical attacks can be implemented similarly. We believe that it is not necessary to implement all attacks and our empirical-analytical exposition is sufficient to fully demonstrate the impact of the exposed vulnerability.

6.4.2 Sniffing Touchscreen Input (Touchlogger)

We will demonstrate how *SMASheD* can be used as a TouchLogger in order to sniff a user's sensitive information. According to Android security model, an app cannot read touch events

performed by the user on other apps [97]. However, we will show how it is possible to infer the keys that the user has pressed, and therefore extract sensitive information efficiently and with 100% accuracy. We note that *SMASheD* is not only able to detect user key presses but it can also log any interaction of the user with the touchscreen, such as swiping, and zooming.

Many researchers have proposed mechanisms to infer the keys pressed by the users. These attacks require the user to install a malicious app on her phone. These methods range from utilizing motion sensors [98–100] and a combination of camera and microphone [101] to taking screenshot while the user is typing sensitive information, such as PINs or passwords [92]. However, these attacks usually have a relatively low accuracy and may be significantly affected by the way the user types and/or holds the phone. Taking pictures or recording audio may also trigger suspicion.

The *SMASheD* app can send *read* request to the *SMASheD* server to obtain all the events the user performs on the touchscreen. However, getting only the raw touch events is not enough to hamper the user privacy. Moreover, the attacker will be interested only in a small subset of these events. For example, an attacker will be interested in learning the password of the user on banking apps but not the input corresponding to user's interaction with a game.

According to the information the attacker wants to learn about the user, the attacker can modify the service in the *SMASheD* app so it sends *read* request when the app corresponding to the data the attacker wants to collect is launched. Moreover, if the attacker wants only to learn the keys the user presses while the app is running, he can send the *read* request when both the app and the keyboard are on the foreground.

To evaluate the ability of *SMASheD* to extract the username and password from various banking apps, we repurposed the original *SMASheD* app. When the *SMASheD* app is launched, it gets the list of installed apps on the device using *getPackageManager* API. Note that no permission is required to get the list of installed apps. Then, the *SMASheD* app looks for the apps that are already installed and are of interest to the attacker. The *SMASheD* app also finds the soft keyboards installed. The *SMASheD* app starts running its status detection service in the background, which regularly executes ps command to find out the list of running applications. Once any of the apps that the attacker wants to collect user data from appears in the output of the execution ps command, the service gets that app process ID, PID_{app}. The service also gets the process ID of the keyboard, PID_{kb} from the execution of the ps command. The service then

executes $ps - tPID_{app}$ command, which returns the list of threads of that process; whenever an app is on the foreground, the list of threads of that app has a thread named "GL updater".

If the app is running the "GL updater" thread, the service also checks if the keyboard is running "GL updater" thread. If both the app and the keyboard app have "GL updater" thread running, *SMASheD* detects that both of them are in the foreground and sends $read$ request to the *SMASheD* server. When user exits the app or the keyboard (which can be detected by checking if the app is no longer in the list returned by executing the *ps* command, or if "GL updater" is not in the list of the thread running for either the app or the keyboard), the *SMASheD* app sends $stop$ request to the *SMASheD* server. As a response to the $stop$ request, the *SMASheD* server sends all the touch events to the *SMASheD* app. The *SMASheD* app parses the events and extracts the events with event type ABS_MT_POSITION_X and ABS_MT_POSITION_Y between BTN_TOUCH DOWN and BTN_TOUCH UP (Listing 6.2), if between the down and up events, the ABS_MT_POSITION_X or ABS_MT_POSITION_Y is missing, the value of it is same as its correspondent in the previous touch. Finally, it maps the x and y coordinates to keys (keyboard layout can be detected by determining which soft keyboard the user is using, the orientation of the device and the screen resolution) and sends the text typed (if the user is using a soft keyboard that is unknown to *SMASheD*, *SMASheD* can send the name of the soft keyboard app, the x, y coordinates of each touch and the screen resolution and orientation to the attacker and the mapping can performed offline), for example, to the attacker's web service via HTML request, or via other methods as we will discuss in Section 6.4.3.1.

We tested the above attack on some banking applications, such as Wells Fargo and Bank of America apps, and we could learn the username and password with 100% accuracy. The attack was tested on Samsung Galaxy S5 with Android OS version 5.0 and Samsung Galaxy S1 with Android OS version 4.4.2, and can be easily adapted to other Android phone models and even other devices such as smartwatches. Our attack is an extremely powerful attack since all the input provided by the user can be precisely and stealthily stolen, raising significant concerns for users' security and privacy.

6.4.3 Manipulating Touchscreen Sensor

The ability of injecting touch events could be extremely dangerous. In essence, it will allow the malware to do whatever the user can do with her device. The primary challenge for the

attacker is to be stealthy. To do so, the attacker should inject the touch events while the user might not be attending to the phone, such as when the user is asleep or the phone is left inside a pocket or a purse. Such contextual scenarios can be determined by monitoring various motion and environmental sensors on the phone, as shown by prior research [103]. For example, the attacker can monitor the proximity and light sensors to infer when the phone is inside a pocket or placed in dark [104]. Moreover, *SMASheD* can change the phone settings, e.g., decrease screen brightness, mute sound and erase logs/traces to make the attacks "user-invisible."

Following subsections layout some of the attacks that *SMASheD* can perform given its capability to inject touch events.

6.4.3.1 Data Exfiltration

Whenever *SMASheD* needs to send any data to a remote attacker (e.g., previously sniffed passwords, credit card numbers or pictures), it can stealthily transmit this data utilizing other apps, such as email or SMS. As some malware detection mechanisms detect malicious apps based on abnormal data usage, *SMASheD* can remain surreptitious and undetected by such systems. Moreover, *SMASheD* can delete the logs from the email and SMS apps so that users cannot trace back. This simple strategy will prevent *SMASheD* from being detected by either the device user or the anti-virus apps. Such an exfiltration will also avoid the need for doing any data processing on the infected device itself but rather allow the attacker to outsource all processing to a remote machine.

6.4.3.2 Phone Unlock

In order to allow *SMASheD* to access any of the device resources that require the device to be unlocked, *SMASheD* needs to unlock the device first. To do that, *SMASheD* first utilizes the TouchLogger presented in Section 6.4.2 to log the user's PIN or pattern unlock while the user unlocks his phone. Then, whenever *SMASheD* wants to unlock the phone, it will simply inject the recorded PIN or pattern unlock onto the touchscreen. We have built and tested such an auto unlocker. A demo is available at https://androidsmashed.wordpress.com/demos/.

6.4.3.3 Accessing User Accounts

SMASheD can be used to open different apps that require authentication, and log into user's accounts. To do so, *SMASheD* will first extract the user's credentials for the target account by using the TouchLogger described in Section 6.4.2. *SMASheD* will then utilize this credential to log into the user account from her device. Accessing the user accounts from the *SMASheD* infected device is important for several reasons. Many web services and banks implement a second factor authentication approach which may only allow the user to login from a registered device. Similarly, many banks require the user to answer security questions when she logs in from a different device, and others send notification to the user specifying the devices that are used to access her account. After having logged into the user accounts, *SMASheD* can, for example, access the account and perform any kind of the allowed banking transactions, read user's emails, send fake emails, forward the emails to a remote attacker, or read user's private data from or post messages on social media sites — the possibilities are endless.

6.4.3.4 Attacking Biometric Authentication

Recently, a significant amount of research has been done to authenticate a user transparently using biometrics. The touch-based biometrics are applied either as a second factor authentication mechanism during the device unlock or as a continuous authentication mechanism when the user is performing some activity on the device. Among these, some systems analyze the keystrokes of the users to capture the biometrics while others analyze touch gestures provided by the users. We now analyze a variety of these biometrics systems proposed in the literature and provide a systematic methodology to attack them using *SMASheD*.

Keystroke Biometrics: *Campisi et al.* [105] present an approach to authenticate users based on their typing habits on the smartphones. Their approach relies on the analysis of keystroke dynamics. The system acquires and processes the time stamps generated by the mobile phones related to key press and release. Using these, the system further calculates different features such as Manhattan distance, Euclidean distance and statistical features and generate a template for each user. During the authentication, the system computes the normalized distance and compares that with a threshold.

To authenticate against such system, *SMASheD* needs to learn how the user types. During the learning phase, *SMASheD* can record the user's keystroke behavior and compute the features in a similar way to the authentication system. After learning, it can create the keystrokes such that the time interval *SMASheD* presses and releases the keys closely correlates with that of the user. Note that *SMASheD* can simply record and replay the user's keystroke without computing the features and the system may still fail to detect such malicious input. However, creating new keystrokes after learning the features is more detrimental to the user as the attacker can recreate any events or activities he likes.

Touch Gesture Biometrics: *Frank et al.* [97] present "Touchalytics", a continuous touch-based authentication system which utilizes the strokes performed by the user while using her phone. Touchalytics focuses on single touch gestures such as sliding horizontally and vertically. Sliding horizontally is common when navigating between the screens or images, while sliding vertical is common when reading email or documents. To authenticate using touch, Touchalytics records the touch coordinates, finger pressures, the screen areas covered by each finger, and times. Touchalytics extracts 30 different features from these raw inputs. Touchalytics uses these features to build a profile of the user and utilizes it later to identify the user.

Since Touchalytics is monitoring and matching the touch with the trained data for horizontal and vertical slides only but not with other actions, *SMASheD* can perform tap/click and pinch without getting detected. However, to navigate up/down or right/left where *SMASheD* has to provide such horizontal/vertical slides, *SMASheD* needs to record the previous authentic slides from the user, and later inject them as desired. While outsider attacks using robots [106] have previously been reported against Touchalytics, the *SMASheD* attack represents the first known insider attack to our knowledge.

Li et al. [107] present an unobservable re-authentication system for smartphones using finger movement patterns. This system uses machine learning approach to authenticate the user based on touch input. The system monitors user's raw touch event data and preprocesses it, which assembles every single raw data into different gestures and then sends these to the feature extraction component. The predictor component consists of an SVM classifier and multiple classification modules. The system uses five different types of gestures: sliding up, sliding down, sliding left, sliding right and tap. For the sliding gesture, the system considers the properties such as first touch position, first touch pressure, first touch area, first moving direction, moving

distance, duration, average moving direction, average moving curvature, average curvature distance, average pressure, average touch area, max-area portion and min-pressure portion, while for the tap gesture, it considers average touch area, duration and average pressure.

Attacking this system with *SMASheD* is simple as it is only looking for five different gestures as described above. *SMASheD* needs to learn how a user moves his finger for these gestures recording all the finger movements. Similar to attacking Touchalytics, *SMASheD* then repeats the recorded authentic slides and tap gestures from the user whenever it wants to perform certain activity on the phone.

Shahzad et al. [108] present "GEAT" for screen unlocking based on simple gestures. Along with the user touch input, GEAT uses other features such as finger velocity, device acceleration, stroke time, inter-stroke time, stroke displacement magnitude, stroke displacement direction, and velocity direction. GEAT segments each stroke into sub-strokes of different time duration where, for each sub-stroke, the user has consistent and distinguishing behavior. GEAT utilizes these features to train and later identify the user.

Since GEAT is only authenticating when user wants to unlock the screen, *SMASheD* can record all the raw touch and device acceleration data during the legitimate authentication by the user. It can later just replay the touch providing the recorded data such that the features would fully match.

Luca et al. [102] present another transparent authentication approach that enhances password patterns with an additional security layer. They study the touch stroke gestures corresponding to the horizontal slide and the pattern unlock. Their approach uses dynamic time warping for the analysis of touch gestures using different features including XY-coordinates, pressure, size, time, and speed of the touch.

SMASheD cannot only thwart the password pattern to unlock the device but also foil the additional security layer provided by this system. As discussed in the Section 6.4.2, *SMASheD* first simply sniffs the password pattern. In addition, *SMASheD* records the pressure, size, time and speed of the touch when the legitimate user performs the pattern unlock gesture. Now, when the *SMASheD* app needs to unlock the device, it simply injects the previously recorded touch events to circumvent the authentication functionality. As our demo for phone unlock records the user interactions with her device while unlocking the device, and then replays it, it would

also defeat this mechanism.

6.4.4 Manipulating Other Sensors

In this section, we first describe the systems which provide authenticate the user based on the motion, position and environmental sensors. Then, we provide an attack scheme against each system using the sensor event injection capability of *SMASheD*.

Attacking these systems may not be straightforward. The best scenario to manipulate the sensor readings when the current sensor readings are not being altered by the natural events. For example, when the phone is in a pocket, the light sensor may not change or report constant values. Since the sensor file will not be altered by the natural environment in this case, the malware can manipulate the sensor data as it likes. Also if the system is implementing a statistical approach (such as based on mean, standard deviation, etc., of the sensor data), the malware may not need to manipulate the sensors for the whole duration when the system is monitoring the sensors. *SMASheD* can insert some values that significantly changes these statistical features which causes the system to misjudge the sensing context. For the other systems, which implement specialized algorithms based on continuous sensor data, *SMASheD* needs to inject sensor readings at different timestamps that correlate to the sensor values during benign case.

Conti et al. [109] propose a system that transparently authenticates the user by analyzing her hand movement gesture while she is making or answering a phone call. It uses accelerometer and orientation sensor to detect the proposed gesture. The system uses the dynamic time warping distance (DTW-D) algorithm to verify if the authorized user is making or answering the phone call.

To attack this system, *SMASheD* can record the accelerometer and orientation sensor data when the user is making or receiving a valid call. Later, when *SMASheD* wants to make a call (e.g., to premium rate numbers or to user's contacts), it can replay the previously recorded sensor data, thereby fooling the system to believe that the user is making the call.

Gascon et al. [110] present an approach to continuously authenticate users on smartphones by analyzing their typing motion behavior. Along with touch input, it also records the timestamps when the keys are pressed or released. The system uses different motion and position sensors such as accelerometer, gyroscope and orientation sensors to capture behavioral biometrics so as to authenticate the user. It extracts various features leading to a 2376-dimensional

vector representing the typing motion behavior of a user in a given time frame. The system is trained with the linear SVM classifier.

To attack this system, *SMASheD* needs to learn how the user presses each character, and reproduce it. During the learning phase, *SMASheD* continuously records the raw sensor data until it gets necessary information used by the system for all the keys during the legitimate key presses. Once the learning phase is completed, *SMASheD* can provide the touch injects with proper timings and the corresponding sensor readings. Since the motion and position sensors are continuously recording the data from the hardware, *SMASheD* may need to wait for a favorable time, e.g., when the phone is static, otherwise the natural readings may interfere with the injected sensor readings possibly leading to rejection by the system.

6.5 Smashed Mitigation

To protect against the adversarial applications of *SMASheD*, we suggest the following potential mitigation strategies. Although these strategies may not fully prevent the attacks, they may help reduce the impact of the underlying vulnerability.

First, we believe that it is important to raise people's awareness of the possible security risks associated with installing services through the ADB shell. Second, we suggest following the permission models of Android for native services that are executed through the ADB shell. In the current model, any native service that starts through the ADB shell is granted all the permissions that the shell has without notifying the user. These permissions include accessing logs, frame buffer, motion, position, environmental, and user input sensors. An attacker may not reveal all the resources that the service is accessing. For example, the attacker could publish a service as a snapshot service while injecting code that accesses sensor files as well. This may be prevented if the service is only granted permissions after informing the user. Third, we suggest enforcing security policies for the communication between processes running on the device through sockets. We recommend that Android monitors the open sockets on the device and the apps that are accessing those sockets. Whenever an unusual communication is detected, Android should at least inform the user. Whether or not users would pay attention to such notifications is an independent concern. However, we believe that the potential risks should be conveyed to the users.

6.6 Conclusion

In this chapter, we called the Android's sensor security model into question. We exploited Android's ADB workaround to develop a framework that can effectively sniff and manipulate many sensors currently protected by Android's access control model. Our framework can be used to: (1) directly sniff the touchscreen sensor data, (2) directly manipulate the touchscreen, motion, position and environmental sensor data, and (3) indirectly, using the touch inject capability, sniff the audio-visual and navigational sensors. Based on this framework, we introduced a wide spectrum of potentially devastating attacks that can compromise user privacy and subvert many authentication systems that rely upon different sensors. Since the scope of our attacks is extremely broad, we provided demonstrations for a selection of schemes and presented an analytical exposition of several other schemes.

CHAPTER 7

STRONG BEHAVIORAL AUTHENTICATION WITH SIMPLE COGNITIVE GAMES

In the chapters 3, 4 and 5, we show that interactivity helps in enhancing the security and the usability of Human-Machine authentication. Moreover, in Chapter 6 we show that all user authentication mechanisms based on static input can be attacked using our developed framework *SMASheD*. In this chapter, we explore interactivity in the context of Human-Human authentication.

In this chapter, to overcome the limitations of the current behavioral biometrics systems, we propose *Gametrics* (Game-based Biometrics), a novel system of interactive game-based behavioral biometrics. Whenever users wish to authenticate to a device or service, *Gametrics* would simply request them to play a short and simple cognitive game. Once identified, permission to access an account or device can be granted via a back end database as is done with existing behavioral biometric solutions. Games are a good platform for the purpose of authentication since web browsers and touch screen devices fully support them.

Chapter Outline: The rest of this chapter is organized as follows. In Section 7.1, we describe the authentication game object (DCGs) used in our system. This is followed by Section 7.2, where we describe our data collection methodology and procedures. Next, in Section 7.3, we elaborate on our machine learning techniques and feature extraction methods to build the *Gametrics* authentication model, and provide the classification results in benign setting and against zero-effort passive attackers. In Section 7.4, we evaluate *Gametrics* against active adversarial attacks that deliberately attempt to mimic a user's game play pattern to defeat the authentication system. In Section 7.5, we present our small scale study we conducted to evaluate the scheme on touchscreen devices. In Section 7.6, we discuss further aspects of our work and provide future research directions. Finally, in Section 7.7, we conclude our work highlighting the main take away points.

7.1 Cognitive Task

In this section, we elaborate on the design and the implementation of the interactive DCG constructs we utilized in our study.

7.1.1 Cognitive Task Design

We embed the cognitive task in simple web-based games, following the design presented in [83]. In this design, each of the game challenges has three target objects and six moving objects. The user's task is to drag a subset of the moving objects (answer objects) to their corresponding target objects. Solving a challenge require the user to: (1) understand the content of the images, (2) find the semantic relationship between the answer objects and the target objects, and (3) drag the answer objects to their corresponding targets. We impose a time limit of 60 second to complete each challenge.

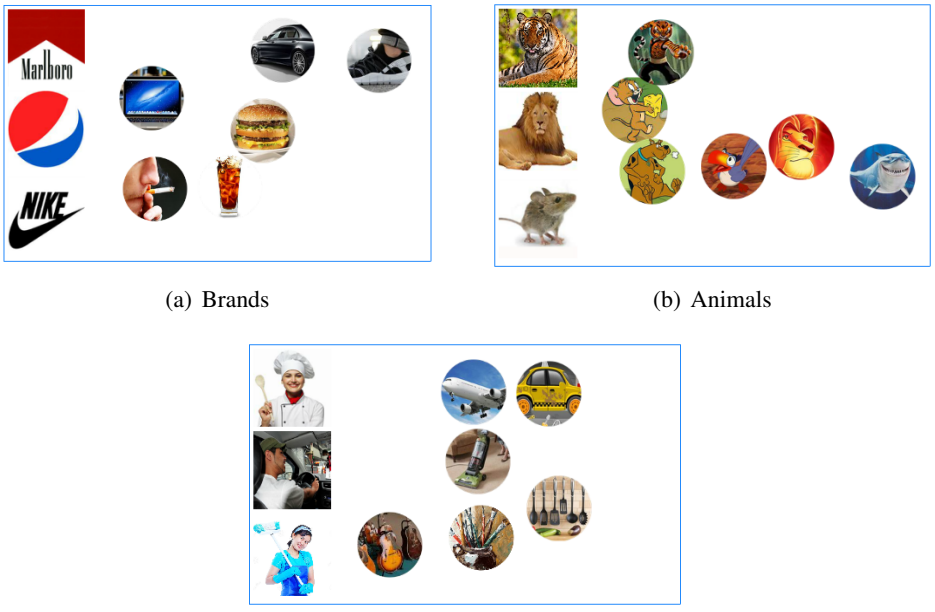
We aim to identify the user based on her interaction with the challenge. Basically, we aim to identify the user based on her cognitive ability (i.e., the time it takes her to recognize the objects and perform the required task) and mouse interaction (i.e., mouse movement characteristics such as mouse movement speed and acceleration).

7.1.2 Cognitive Task Implementation

We implemented the challenges using Adobe Flash ActionScript3 and the web server using PHP. The challenge image/frame size is 500 $\times$ 300 pixels, the size of each of the moving object is 75 $\times$ 75 pixels and the size of the target objects is 90 $\times$ 90 pixels. The challenge starts by placing the objects in random locations on the image. Then, each object picks a random direction in which it will move. A total of 8 directions were used, namely, N, S, E, W, NE, NW, SE and SW. If the chosen direction is one of E, W, S, or N, the object will move (across X or Y axis) by 1 pixel per frame in that direction. Otherwise, the object will move $\sqrt{2}$ = 1.414 pixels per frame along the hypotenuse, corresponding to 1 pixel across both X and Y axes. This means that on an average the object moves 1.207 [= (1 $\times$ 4 + 1.414 $\times$ 4)/8] pixels per frame. We set the number of frames per seconds to 40 FPS. The object keeps moving in its current direction until it collides with another object or with the challenge border, whereupon it moves in a new random direction.

The challenge starts when the user presses a "Start" button on the screen center. The challenge ends when the user drags all the answer objects and drops them onto their corresponding targets, in which case a "Game Complete" message is provided or timeout is reached, in which case a "Time Out" message is provided.

After the user performs an object drag/drop, the challenge code sends to the server the identifier of the object and the drop location. The server checks the correctness of the drag/-drop and gives feedback to the challenge code. If the web server confirms that the object was dropped on its corresponding target, the object disappears giving feedback to the user that he performed a correct action. After the user drags and drops all the answer objects to their corresponding targets, the challenge code sends to the server the log of the gameplay. The gameplay log contains the objects locations, the mouse location and status (up/down) at each time interval to the server. The server utilizes this log to authenticate the user. The timestamps were generated from multiple events listeners: MouseEvent.CLICK, MouseEvent.MOUSE_UP, and MouseEvent.MOUSE_MOVE.



(a) Brands

(b) Animals

(c) Professions

FIGURE 7.1: *Gametrics* Challenges Instances. Targets, on the Left, are Static; Moving Objects, on the Right, are Mobile. The User Task is to Drag-Drop a Subset of the Moving Objects (Answer Objects) to Their Corresponding Targets

For the purpose of our study, we implemented six instances of the explained challenges that can be categorized into three categories (two instances of each category) described below. A sample of each of the implemented categories is shown in Figure 7.1.

- **Brands:** The targets are popular worldwide brands and the objects are commercial products (e.g., Nike and Nike shoes).
- **Animals:** The targets are real animals and the moving objects are cartoon animals (e.g., lion and Lion King).
- **Professions:** The targets are professionals and the moving objects are tools (e.g., taxi driver and taxi).

7.2 Data Collection

As a pre-requisite to building and testing our *Gametrics* system, we pursued data collection from human users, in both online and lab settings. In this section, we elaborate on our data collection methodology, and the characteristics of the collected data set. In Section 7.5, we present a small scale study we conducted on Mobile devices.

The participation in our two studies was voluntary, and standard ethical procedures were fully followed, e.g., participants being informed, given choice to discontinue, and not deceived. The studies was approved by our university's Institutional Review Board. The data collection experiments were divided into four phases. First, we subjected the participants to a consent form. Then, we asked the participants to go through a tutorial and fill up a demographics form. Next, we asked the participants to solve several instances of the game challenges explained in Section 7.1.2. At the end of the study, we asked the participants to fill-out a survey form about their experience. The survey contained the 10 System Usable Scale (SUS) [80] standard questions, each with 5 possible responses (5-point Likert scale, where strong disagreement is represented by "1" and strong agreement is represented by "5"). SUS is a standard questionnaire to measure the usability of software, hardware, cell phones and websites, and it has been deployed in many prior security usability studies. Moreover, specific to our study, we added two questions to the survey in order to measure the easiness and playfulness of the challenges. As the participants played the game challenges, all of their gameplay mouse events were recorded in the background.

Table 7.1 summarizes the characteristics of the data collected during the two studies. The total number of participants is 118 (98 in online study and 20 in lab study). The participants

TABLE 7.1: Summary of the Collected Data Sets

		# Users	Solving Time(s) Mean (std)	Completed Challenges
Online Study	**Day 1**	98	7.39 (3.55)	5839
	Day 2	62	7.23 (2.77)	2209
	Day 3	29	7.65 (2.98)	1028
Lab Study		20	7.66 (3.45)	1200

successfully completed a total of 10276 challenges (9076 in online study and 1200 in lab study). The average time to complete a game challenge was around 7.5 seconds.

For our online data collection study, we utilized the Amazon Mechanical Turk (MTurk) service to recruit the participants. The aim of our online study was to evaluate the applicability of identifying the user based on the way she interacts with the posed game challenges. Moreover, we wanted to determine how our system would perform in a longitudinal setting, over multiple sessions/days. Therefore, we created a total of three Human Intelligence Tasks (HITs) distributed over three days. The first HIT was created with 100 assignments to have 100 unique workers. We gathered 98 valid submissions until the HIT expired. The workers were directed to the website hosting the study. They were required to solve a tutorial, fill a demographics form and play 60 instances of our challenges. The order of presenting the challenges to the participants was random. Finally, the participants filled out the survey. On the next two days, we sent the participants emails asking them to participate in the follow-up study. However, we asked them to solve 36 challenges rather than 60 challenges in this round. 62 participants performed the study on the second day and 29 performed the study on the third day. We paid each participant \$1.0 for the first HIT, and \$0.5 each for the second and third HIT.

The participants in our online study were from various age groups, education levels and backgrounds. Age group: 1% $<$ 18, 20.4% 18-24, 38.8% 25-34, 32.7% 35-50 and 7.1% $>$ 50. Gender: 58.2% male and 41.8% female. Education: 26.5% high school graduate, 58.2% hold bachelor degree, 14.3% hold master degree and 1% hold a PhD degree. The participants were from various backgrounds such as Computer Science, Engineering, Medicine, Law, Social Science, Finance, Business, Mathematics, Art, etc. (detailed demographics information is populated in Table 7.2)

For our lab-based study, we collected data from some volunteers recruited from our University. It followed a similar protocol as the online study, but using a lab computer. We asked

TABLE 7.2: Participants Demographics

	MTurk	Lab
# Participants	**98**	**20**
Age (%)		
<18	1.0	0
18-24	20.4	40
25-34	38.8	45
35-50	32.7	15
>50	7.1	0
Gender (%)		
Female	41.8	35
Male	58.2	65
Education (%)		
High School	26.5	25
Bachelor	58.2	40
Masters	14.3	30
PhD	1.0	5

the volunteers to perform a similar task as the task performed by the MTurk workers on the first day. A total of 20 undergraduate and graduate students as well as some employees participated in the study. The age of the participants ranged between 19 and 50, 13 of them are male and 7 are female, 5 are high school graduate, 8 have bachelor degree and 7 have master degree. The majority of the participants are from Computer Science background (Table 7.2). We asked the volunteers to play 60 instances of the challenges using the same computer and same setting. The aim of this study was to validate the results of the MTurk study. In particular, we mainly wanted to ensure that the acquired results are not based on the platform and the setting used in performing the experiment rather than the different characteristics of an individual's unique way of interacting/solving the game challenges.

7.3 System Design & Results

In order to evaluate the applicability of the *Gametrics* as an authentication scheme, we utilized the machine learning approach. In this section, we present the features we extracted from the user's gameplay logs collected during our data collection campaign. Then, we discuss the classification models and the classifier employed. Finally, we present the classification results for the benign setting and the zero-effort attack.

7.3.1 Feature Extraction

TABLE 7.3: The Features Utilized for Classification

	Feature		Description
Cognitive	Time	number	Time taken to complete the challenge
	Time first action	number	The timestamp of the first mouse event after the game start
	Time first drag	number	The timestamp of the first drag
	Time between drags	mean, std, min, max	Times between drops and start of drags
Mouse interaction	Speed drag	mean, std, min, max	Speed while dragging
	Speed move	mean, std, min, max	Speed while moving
	Acceleration drag	mean, std, min, max	Acceleration while dragging
	Acceleration move	mean, std, min, max	Acceleration while moving
	Difference timestamp	mean, std, min, max	The difference between each consecutive recorded timestamps
	Move silence	mean, std, min, max	The times between consecutive timestamps while the mouse is moving
	Drag silence	mean, std, min, max	The times between consecutive timestamps while dragging
	Pause and drag	mean, std, min, max	The times between approaching the object and click on it
	Pause and drop	mean, std, min, max	The times between approaching the target and drop
	Angle	mean, std, min, max	The angles between each three consecutive points
Mixed	Drag distance to real distance	mean, std, min, max	The difference between the distance traveled while dragging and the straight line connecting the start and end points of the drag
	Move distance to distance	mean, std, min, max	The difference between the distance traveled while moving and the straight line connecting the start and end points of the move
	Distance click object center	mean, std, min, max	Distances of the clicks and objects' centers
	Distance drop target center	mean, std, min, max	Distances of the drops and targets' centers
	Total distance	number	Total distance

From each instance of the gameplay logs we collected during the data collection phase, we extracted a total of 64 features that captures the cognitive abilities as well as the mouse interaction characteristics of the participants while they are interacting with the challenges. (The extracted features are summarized in Table 7.3.)

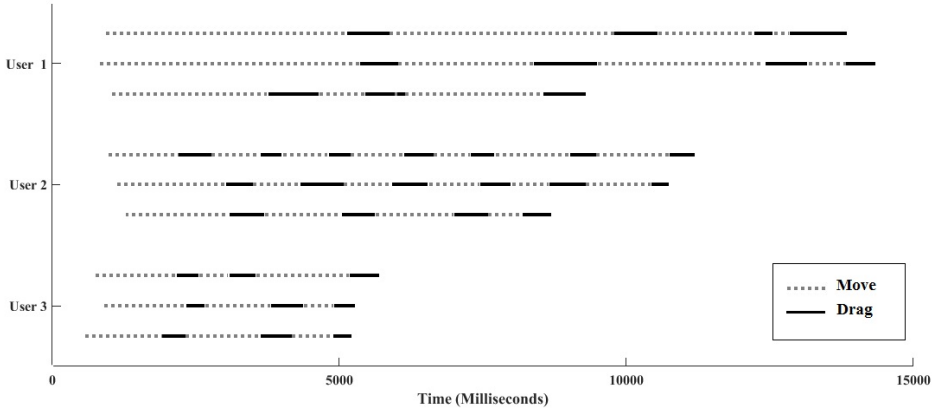


FIGURE 7.2: An Example for Illustration of Different Cognitive Characteristics among Different Users While Playing the Game Challenges: The Time for Completing the Games and the Time Spent in Drag and Time Spent in Moving the Mouse Around. We Can See That User 1 Took a Long Time to Understand the Game (Long Move Segment before the Start of the First Drag), Also Took on Average a Long Time to Locate Each of the Answer Objects and to Start Dragging. User 2 Took Shorter Time to Complete the Challenges but Committed Many Mistakes (the User Performed Exactly 3 Drags and Drops to Complete Each Challenge, However, User 2 Performed on Average More Than 5 Drags), User 3 Completed the Games in Short Time with Shorter on Average Times to Locate the Answer Objects

As described in Section 7.1.1, in order to solve a challenge, the user has to match the answer objects to their corresponding targets. In order to do that, the user has to understand the content of the images representing the targets and the moving objects, find the relationship between the moving objects and the target objects, and then select a subset of the moving objects (the answer objects) and finally drag/drop them to their corresponding targets. By monitoring the users while solving the challenges (lab study), we found different users take different approaches to solve the challenges. For example, some users start by trying to comprehend the whole challenge and then start the object matching, while some try to find the answer objects corresponding to the target in certain order (i.e., always try to search for the answer object that corresponds to the top most target, and then the second and so on), while some try to pick the object closest to the mouse cursor and then check if it matches with any of the targets. For visualization purposes, these differences in the cognitive characteristics of different users are illustrated in Figure 7.2.

These different mechanisms of solving the game challenges are related to the cognitive characteristics of individuals. We capture these characteristics based on the following features:

1. The time between the user pressing on the start button and the first recorded mouse event and the time of the first click/drag. These timing measures capture the time the participates take to comprehend the challenge and start solving it.

2. The times between each of the drops and the start of the next drag (these capture the time the user takes to find the next answer object).

3. The total time taken by the user to complete the challenge.

The mouse movement characteristics of the users are captured by following features:

1. The speed and acceleration while the user is searching for an answer object and while the user is dragging the object.

2. The duration between each two consecutively generated timestamps and the "silence" during move and during drag.

3. The time duration between reaching an object and clicking on it, and the time duration between approaching a target object and dropping an answer object on it.

4. The angles between the lines that connect each 3 consecutive points in the mouse movement trajectory.

Other mixed features are also extracted that relate to both cognitive and mouse movement characteristics of the participants such as the total distance the mouse moved within a game challenge, the difference between the straight line connecting the start and the end of a move or a drag and the real distance traveled. The distance between a click and the object center, and a drop and the target center.

TABLE 7.4: MTurk Study Results: Performance for the classifier for three different classification models. The first part shows the performance of the classifier using all the features. The next part shows the results of using the features subset that provides the best average results. The last part shows the result of using the best features subset for each user. For each of the models, we show the results of using a single challenge and merging of two challenges. Highlighted cells emphasize the most interesting results.

			FPR	FNR	Precision	Recall	F-Measure
			Mean (Std)				
All features	**Single**	**Day 1**	0.12 (0.10)	0.12 (0.16)	0.88 (0.09)	0.88 (0.16)	0.87 (0.12)
		Day 2	0.11 (0.09)	0.25 (0.31)	0.81 (0.24)	0.75 (0.31)	0.76 (0.28)
		Day 3	0.10 (0.07)	0.22 (0.27)	0.86 (0.14)	0.78 (0.27)	0.80 (0.24)
	Merge	**Day 1**	0.10 (0.13)	0.11 (0.18)	0.91 (0.11)	0.89 (0.18)	0.88 (0.14)
		Day 2	0.09 (0.11)	0.20 (0.30)	0.85 (0.23)	0.80 (0.30)	0.80 (0.27)
		Day 3	0.08 (0.10)	0.22 (0.30)	0.86 (0.25)	0.78 (0.30)	0.80 (0.27)
Average overall best	**Single**	**Day 1**	0.11 (0.09)	0.11 (0.15)	0.89 (0.08)	0.89 (0.15)	0.89 (0.11)
		Day 2	0.18 (0.13)	0.17 (0.15)	0.83 (0.15)	0.83 (0.15)	0.82 (0.12)
		Day 3	0.10 (0.07)	0.19 (0.26)	0.85 (0.18)	0.81 (0.26)	0.82 (0.23)
	Merge	**Day 1**	0.10 (0.13)	0.09 (0.16)	0.91 (0.11)	0.91 (0.11)	0.90 (0.13)
		Day 2	0.12 (0.12)	0.19 (0.20)	0.88 (0.12)	0.81 (0.20)	0.83 (0.14)
		Day 3	0.12 (0.18)	0.18 (0.18)	0.88 (0.10)	0.82 (0.18)	0.84 (0.13)
User specific	**Single**	**Day 1**	0.06 (0.06)	0.02 (0.04)	0.95 (0.05)	0.98 (0.04)	0.96 (0.04)
		Day 2	0.09 (0.09)	0.07 (0.10)	0.91 (0.09)	0.93 (0.10)	0.92 (0.09)
		Day 3	0.07 (0.06)	0.07 (0.10)	0.93 (0.06)	0.93 (0.10)	0.93 (0.07)
	Merge	**Day 1**	0.02 (0.05)	0.02 (0.05)	0.98 (0.05)	0.98 (0.05)	0.98 (0.04)
		Day 2	0.05 (0.09)	0.04 (0.09)	0.96 (0.08)	0.96 (0.09)	0.96 (0.08)
		Day 3	0.04 (0.06)	0.03 (0.05)	0.96 (0.05)	0.97 (0.05)	0.96 (0.04)

7.3.2 Classifier and Metrics

In our analysis, we utilized the Random Forest classifier. Random Forest is an ensemble approach based on the generation of many classification trees, where each tree is constructed using a separate bootstrap sample of the data. In order to classify a new input, the new input is run down all the trees and the result is determined based on majority voting. Random Forest is efficient, can estimate the importance of the features, and is robust against noise [111]. Several other classifiers were tested during the course of study such as SVM, Bayes Network, Neural Networks, but Random Forest outperformed all of them.

In our classification task, the positive class corresponds to the gameplay of the legitimate user and the negative class corresponds to the impersonator (other user / zero-effort attacker). Therefore, true positive (TP) represents the number of times the legitimate user is granted access, true negative (TN) represents the number of times the impersonator is rejected, false positive (FP) represents the number of times the impersonator is granted access and false negative (FN) represents the number of times the correct user is rejected.

As performance measures for our classifier, we used false positive rate (FPR), false negative rate (FNR), precision, recall and F-measure (F1 score), as shown in Equations (7.1) to (7.5). FPR and precision measure the security of the proposed system, i.e., the accuracy of the system in rejecting impersonators. FNR and recall measure the usability of the proposed system as high FNR leads to high rejection rate of the legitimate users. F-measure considers both the usability and the security of the system. To make our system both usable and secure, ideally, we would like to have FPR and FNR to be as close as 0, and recall, precision and F-measure to be as close as 1.

$$FPR = \frac{FN}{TN + FN} \tag{7.1}$$

$$FNR = \frac{FN}{TP + FN} \tag{7.2}$$

$$precision = \frac{TP}{TP + FP} \tag{7.3}$$

$$recall = \frac{TP}{TP + FN} \tag{7.4}$$

$$F\text{-}measure = 2 * \frac{precision * recall}{precision + recall} \tag{7.5}$$

7.3.3 Classification Models & Feature Selection

We studied various models of classifications. In the first model, we utilized all the features explained in Table 7.3 for training and later testing the classifier. Second, in order to improve the accuracy of the classification, we ran a program to find the subset of features that produces the best classification results, as using many features can cause over fitting of the classifier

and therefore reduce the accuracy of the future prediction, thus removing some features may improve the accuracy. Therefore, we report, in the next subsection, the results obtained by using the subset of features that produces the best average results across all the participants (users being authenticated) in the study. Third, we find the best subset of features that produces the best classification results per user.

For each of the three classification models, we study the identification of the user based on a single game challenge as well as on combining two challenges. As the average time for solving a challenge is around 7.5 seconds, we believe that utilizing two instances of the game challenges to identify the user is not much of an overhead. However, it may improve the accuracy by doubling the amount of captured interactions between the user and the challenges. In a real-life authentication application, posing the user with two consecutive game challenges captures this scenario.

7.3.4 Classification Results

TABLE 7.5: Lab-Based Study Results: Performance for the classifier for three different classification models. The first part shows the performance of the classifier using all the features. The next part shows the results of using the features subset that provides the best average results. The last part shows the result of using the best features subset for each user. For each of the models, we show the results of using a single challenge and merging of two challenges. Highlighted cells emphasize the most interesting results.

		FPR	FNR	Precision	Recall	F-Measure
		Mean (Std)				
All features	**Single**	0.20 (0.12)	0.23 (0.14)	0.80 (0.10)	0.77 (0.14)	0.78 (0.10)
	Merge	0.15 (0.18)	0.16 (0.16)	0.87 (0.15)	0.84 (0.16)	0.84 (0.13)
Average overall best	**Single**	0.18 (0.13)	0.22 (0.14)	0.82 (0.11)	0.78 (0.14)	0.80 (0.10)
	Merge	0.14 (0.15)	0.16 (0.14)	0.88 (0.12)	0.84 (0.14)	0.85 (0.10)
User specific	**Single**	0.11 (0.09)	0.08 (0.09)	0.90 (0.08)	0.92 (0.09)	0.91 (0.06)
	Merge	0.04 (0.08)	0.05 (0.08)	0.97 (0.06)	0.95 (0.08)	0.95 (0.05)

Inter-Session Analysis: As mentioned in Section 7.2, we collected data from 98 MTurk workers during the first day of our data collection experiment. Each of them completed 60 challenges. We divided the collected data into 98 sets based on the users' identities (ids). In order to build a classifier to authenticate a user based on her gameplay biometrics, we defined two classes. The first class contains the gameplay data from a given user (to be identified), and the other class contains randomly selected gameplay data from other users.

Then, we divided the data into two sets, one for training and the other for testing. The first 40 gameplay instances of each participant and 40 gameplay instances of the randomly selected

set were used to train the classifier, while the other 20 are used for testing. We have tested our three classification models in two settings to evaluate our system. In the first setting, we used a single gameplay instance to authenticate the user while in the second setting, we used two instances of the gameplay to authenticate the user. The merging is done by averaging all the features from the two instances.

The results are shown in the first row ("Day 1") of each block in Table 7.4. We see that utilizing two gameplay instances is consistently better than using a single instance. Also, we find that the user-specific model outperforms both the other models (using all the features and using the features that provide the best average over all results). Thereby, the best results are acquired by using the user-specific model and merging two challenge instances in which both the false positive rate and false negative rate = 2%.

Intra-Session Analysis: Our other main goal was to check the accuracy of the classifier over multiple sessions. As mentioned in Section 7.2, 62 MTurk workers participated in the study in the second day and 36 participated in the study in the third day. For each of these users, we used the data of the gameplay of the previous day(s) to train the classifier and then we tested the classifier with the data collected in the next day(s).

The results are shown in the second and third rows ("Day 2" and "Day 3") in each block in Table 7.4. We find that the performance of the classifier degrades slightly compared to the first day, inter-session analysis. Also, we still found that merging two instances provides better results than using a single instance. The best results are again acquired by using the user-specific model and merging 2 instances. For the second day, False Positive Rate = 0.05 and False Negative Rate = 0.04 and for the third day False Positive Rate = 0.04 and False Negative Rate = 0.03.

Lab-based Study Analysis: Our lab experiment involved 20 participants who were asked to perform the study in controlled settings. All of the participants were asked to solve 60 challenges using the same PC and same setting with minimal distraction. The results of the lab based study are summarized in Table 7.5. The results indicate that merging two challenges and using the user specific model can identify the user with high accuracy (0.05 False Negative Rate) and reject the zero effort attackers with high accuracy (0.04 False Positive Rate). The results are in line with the results acquired from the MTurk study, which show that the performance of the

TABLE 7.6: Shoulder-Surfing Impersonation Attack Results

		FPR
All features	**Single**	0.15
	Merge	0.07
Average overall best	**Single**	0.20
	Merge	0.10
User specific	**Single**	0.31
	Merge	0.03

classifier was related to the ability of the classifier to distinguish users' unique way of solving the challenges rather than the platform and the settings they used while solving the challenges.

7.3.5 Summary of Results

The results obtained from the classification models show that *Gametrics* is a viable form of behavioral biometrics. The results show that the classifier can identify the users and reject a zero effort attacker with a high overall accuracy, especially when user-specific models are employed and two game instances are merged together.

7.4 Impersonation Attack

In Section 7.3.4, we demonstrated that *Gametrics* is robust against zero-effort attacks, reflected in the low False Positive Rate. In this section, we analyze the security of *Gametrics* against deliberate impersonation attacks

We first considered shoulder-surfing impersonation attacks. During the lab-based study's data collection, a researcher in our group served the role of an attacker, and monitored, through video recording, the participants while they were solving the challenges. For the impersonation attack analysis, the attacker picked one of the participants who had the most similar features, such as the time duration and mouse movement speed, as that of the attacker, and tried to mimic that participant by solving the challenges in a similar way as the participant did for 60 times. Making a selection in this fashion is representative of a powerful scenario where the attacker targets victims who are easier to attack. If we can show that our *Gametrics* system can be resistant to such a powerful attacker, it may be even more resistant to other weaker, more realistic attackers who may not have the capability to make such selections.

The performance of this attack is enumerated in Table 7.6. For the user-specific model,

the attack success rate came out to be 0.31 when single instance of the challenges was used by the classifier, and decreased drastically to 0.03 when merging of two instances is used by the classifier. There are two main reasons for the increase in security when merging two instances. First, the features that were used for the classification in the single instance model (i.e., the features subset that yielded the highest classification accuracy in the benign and zero-effort case) all related to the mouse movement characteristics, namely, the features used were the drag speed, the move and the drag acceleration and the drag silence. However, in the merged instance model, more features were used by the classifier that relate to both of the cognitive as well as the mouse movement characteristics of the user, which made mimicking the victim much harder. Second, the classifier performs better as using two challenges involve more interaction between the user and the challenges, and make the mimicking task much harder for the attacker. In all the other classification models, we found that the security provided by merging two challenges was also much higher than its corespondent in using a single challenge. This suggests that our *Gametrics* system can defeat powerful shoulder-surfing attacks with a high probability when two game instances are merged and when user-specific model is used.

In practice, it is possible that the attacker resorts to an automated strategy, for example, the use of robots, rather than manual shoulder-surfing (which may be a tedious attack anyway). A robotic attack to compromise behavioral authentication schemes, specifically touchscreen dynamics, has been proposed in [112]. Such robots can be built to mimic the user's way of interacting with the authentication construct based on the leaked authentication template. These attacks have been shown to be able to significantly decrease the performance of touch-based authentication systems. In contrast to tradition behavioral biometrics where the authentication construct is static (i.e., PIN or pattern unlock), *Gametrics* involves randomization in the object movements as well as solving a game-based CAPTCHA (DCG) [83]. Thereby, to build a robot that is able to mimic the user's interaction with the games, the robot is required to not only repeat a previously recorded interaction between the user and the authentication construct, but also to understand the underlying challenge as fast as a human user and then try to mimic the user's interaction with the challenge. Although it is shown in [83] that DCG CAPTCHA can be attacked using a *dictionary-based attack*, if the server incorporates a large database of the challenges and display the challenges randomly to the user, this task would become hard for the bot as the dictionary search and the matching between each of the moving objects and the

stored answer objects in the database would significantly slow down this process. Furthermore, matching each of the answer objects with the answer objects stored in the dictionary requires some amount of time, for instance in [83] the authors proposed to click on the object to hold it while performing the object matching. This would make it hard to mimic the user's "pause and drag" feature. Based on this analysis, we therefore conclude that even automated shoulder-surfing attacks against *Gametrics* will not be effective.

The authors of [60] showed that most of the currently proposed behavrioral biometrics schemes (including keystroke and touchscreen dynamics) are vulnerable to internal, malware-based attacks. Malware installed on the device (authentication terminal/phone) can record the user's valid authentication template and replay it later to authenticate itself as the user (e.g, replay a "pattern unlock" biometrics [102]), or learn from multiple interactions between the user and the device, and then reproduce the new data that has similar features to the user's valid interactions with the device in order to fool the authentication system (e.g., learn the user's typing pattern and then enter another text mimicking the user's typing style). In contrast to other behavioral biometrics schemes, the multi-round randomization embedded in the *Gametrics* challenges as well as the requirement of solving the underlying game-based CAPTCHA will make *Gametrics* robust against such attacks. That is, even having access to the authentication template or a prior authentication session data will not be sufficient for the attacker to impersonate the user in the *Gametrics* system.

To sum up, *Gametrics* promises to address many of the attacks that are known to be a significant concern for traditional password-based authentication systems as well as existing behavioral biometrics systems, including:

- *User-side attacks*, where the attacker observes the victim as she logs in, through manual or automated mechanisms, to learn the user's input (password in password system) or learn the way the user provides the input (biometrics data from the current session in behavioral biometrics systems). The attacker then attempts to replay the information in an authentication session at a later point of time.

- *Server-side attacks*, where the attacker hacks into the web server databases to learn the stored authentication token (e.g., hash of passwords in password systems and biometrics template in behavioral biometrics systems). The attacker then uses this information to

run an offline dictionary attack against passwords, or reproduce the biometric data that matches with the template.

- *Client-side attacks*, where the attacker hacks into the authentication terminal using which the user is logging in and learn the user's input. The attacker then attempts to replay the information in an authentication session at a later point of time.

7.5 Mobile Study

The aim of the study is to check (1) the accuracy of verifying the users on touch screen devices, (2) check the enhancement in the accuracy of verifying the user by fusing the cognitive abilities of the user, touch screen data and the motion sensors on the phone and (3) compare the proposed biometrics with pattern unlock biometrics [102](the most widely used method for point of entry for Android devices). For the purpose of the study, we recruited 12 users and we asked them to solve 25 DCG challenges, and perform 25 pattern unlock. The data collection was performed over 5 days where at each day the user has to perform 5 challenges of each of the tasks. The DCG challenges were randomly ordered to the participants. The static pattern used in our study is the one shown in Figure 7.3 one of the most used pattern by Android users [113].

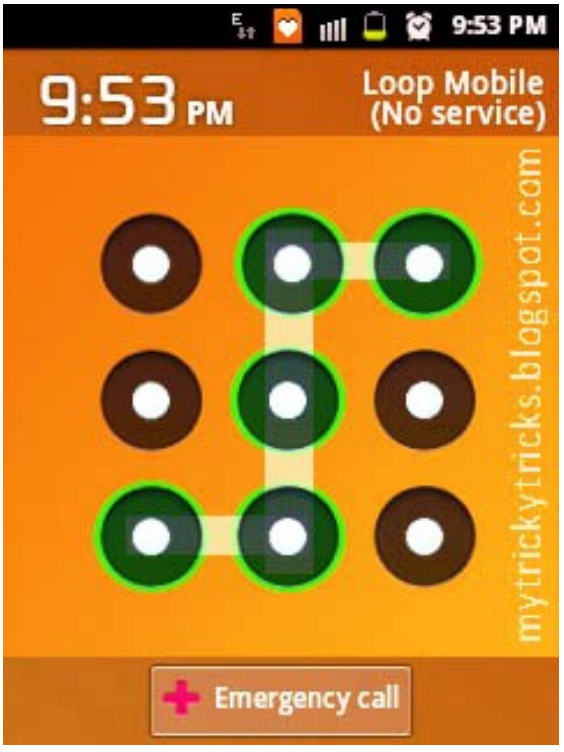


FIGURE 7.3: The Pattern Lock Used in Our Study

TABLE 7.7: Mobile-Based Study Results

	FPR	FNR	Precision	Recall	F-measure
	Mean (Std)				
Pattern	0.05 (0.04)	0.02 (0.03)	0.95 (0.03)	0.98 (0.03)	0.96 (0.03)
DCG	0.07 (0.05)	0.03 (0.05)	0.93 (0.05)	0.97 (0.05)	0.95 (0.04)

All the collection are performed on one smartphone to eliminate the confounding factors introduced by changing software and hardware environments. The smartphone was Samsung S3 (720 $\times$ 1280 resolution) with a 4.8-inch screen, and 1.5 GHz dual-core processor. While the users performed each of the tasks, we recorded their interactions with the device captured by the touch screen interaction (x, y position of the touch, and touch size) and motion and position sensors on the device (Accelerometer, Magnetic, Gyroscope, Magnetic, Orientation, Rotation Vector, Gravity, Linear Acceleration).

For each of the used sensor, we calculated the square root of the sum of squares for that instance's components of all the instances in the sample that corresponds to a single challenge solving. From the touch data, we calculated several features. For pattern lock, we calculate the start, average and end touch sizes, time, distance, speed and acceleration of the swipe, and number of generated time stamps. For DCG, we calculated the time, time of first touch, average, standard deviation, min and max of times between a drop and next drag, start, end and average touches sizes, drag speed and acceleration and difference between the drag distance and the distance between the start and end points of the drag.

As a classification method we used Random Forest as we did for the Web based study. As the data we had per user is less than its correspondent in the Web based study, we use 10-fold cross validation rather than train test model. We ran our program for finding the best subset of features per user that gives the best classification accuracy for both of the tested tasks. The obtained results are summarized in Table 7.7. We found both methods can identify the user with high accuracy. We argue the two methods are equally good in identifying the user, however utilizing DCG rather than pattern would prevent against replay attack (i.e., the attack explained in Section 6.4.3.2), robot attacks [112] and smudge attack.

7.6 Discussion

Efficiency: The proposed *Gametrics* system can fit well for many applications noting the short time the user took to solve the challenges (around 7.5 seconds for a single challenge and 15 seconds for two challenges). Moreover, the enrollment phase consisted of 40 challenges (around 5 minutes on average) and provided a reasonably high identification accuracy. In short, building the classifier model, updating the model with the new data over time (e.g., as the user logs in by playing new game instances) and testing a new instance, all take a short amount of time.

User Experience: The *Gametrics* system also seems to offer high usability, as the average SUS score came to be 86.11 (standard deviation = 14.12) in the lab-study and 73.95 (standard deviation = 17.14) for the web study. SUS scores above 70 are indicative of good overall usability. The score for the playfulness of the challenges came to be 3.36 (standard deviation = 1.40) and the easiness of the challenges was 4.58 (standard deviation = 0.77). This suggests that the participants found the game challenges to be very easy (although not necessarily playful). These results overall bode well for the user experience of *Gametrics*.

Application Scenarios: *Gametrics* can be utilized as a point-of-entry mechanisms, such as to authenticate the user to a web server.

Graphical passwords were founded on a psychological principle that the human brain has superior memory for processing visual rather than textual information (see two excellent surveys [33, 34]). They can be based on recognition, such as those involving Random Arts images [35], objects (PassObjects) [36] and faces (PassFaces) [37], as well as on recall or cued recall, such as those involving drawings [38, 39] and selection of points on an image (PassPoints) [40]. *Gametrics* can be integrated with graphical passwords as a second factor authentication, which would enhance the security of graphical passwords against shoulder surfing and spoofing attacks. Further work is needed to realize such two-factor designs.

Gametrics can be also used as a fall-back authentication mechanism. In such use case, multiple instances of the challenges can be presented to the user, since fall-back does not happen frequently. However, in order to build an up-to-date classification model for the user, the system may need to ask the user to solve challenges periodically. Future investigation is necessary to study *Gametrics* in the context of such fall-back authentication applications.

Given the popularity of touchscreen games, *Gametrics* would fit well on touchscreen devices. Here, *Gametrics* can utilize the various sensors, such as accelerometer and gyroscope, available on these devices to measure the users' implicit interactions with these devices, which when combined with other explicit touchscreen interaction features may enhance the overall classification accuracy and resistance to attacks. In our future work, we will study the effectiveness of *Gametrics* for authenticating the users on such devices.

Recently, Google has announced a plan to eliminate passwords by introducing a Trust API that uses a fusion of multiple biometrics indicators to verify the user's identity, such as face recognition and voice patterns, and other behavioral biometrics such as the gait biometrics [114]. In the future, we hope that *Gametrics* can be added to the Trust API by asking the user to play a game challenge on a periodic basis.

7.7 Conclusion

In this chapter, we introduced *Gametrics*, an interactive biometrics system based on the gameplay pattern of the users embedded in very simple game constructs. We showed that incorporating the mouse dynamics with the cognitive mechanisms can identify the users with high accuracy within a short period of time. Moreover, *Gametrics* provides security against multiple forms of vulnerabilities ranging from zero-effort attacks to expert attacks who try to mimic the user, even those who hack authentication templates and employ automated mechanisms such as robots and malware. The time taken for enrollment and authentication are both reasonably short. The system seems to provide good user experience as reflected in the participants' responses to the survey.

CHAPTER 8

CONCLUSIONS AND FUTURE WORKS

In this dissertation work, we have investigated the impact of using interactivity in the context of human-machine and human-human authentication.

8.1 Human-Machine Authentication

First, we designed and developed simple interactive CAPTCHAs (S-DCG) and evaluated their security and usability. We showed through our study that such simple interactive constructs are highly usable, even on mobile devices where most deployed CAPTCHAs suffer from low usability. Moreover, we showed the dynamic and interactive nature of S-DCG provides security against relay attacks. However, we also demonstrated that S-DCG is insecure against dictionary-based automated attacks.

Then, we explored various variants aiming to secure S-DCGs against automated attacks. First, we utilized various countermeasures in order to prevent foreground objects extraction. Our final DCG design was built on the emerging image notion (EI-DCG). Although EI-DCG is shown to be secure against automated attacks based on image processing techniques, and offers high security against human-solver relay attack, it was found that EI-DCG suffers from low usability due to the presence of the emergence effect.

Finally, to address the usability limitation of EI-DCG, we introduced Mix DCG. Mix DCG is designed by carefully combining DCG, image semantic CAPTCHA and image orientation CAPTCHA. Mix DCG has been shown to be secure against automated attacks and relay attacks, and offers a high level of usability. Mix DCG achieves the various CAPTCHA design goals and is resistant against the attacks underlying the standard CAPTCHA threat model.

The security analysis for the studied DCGs against human-solver relay attack showed that we were able to detect the relay attack with high accuracies. Although DCGs cannot prevent rely attack 100% of the time, it delays such attack and makes it expressive for the attacker.

The future work in this line of research may look at various ways for speeding up the process of building Mix DCG such as utilizing the line drawing of 3D objects as introduced in [115].

One of the limitations of the developed mechanisms is that they cannot be solved by blind and impaired users. A backup mechanism, such as audio CAPTCHA, would be needed to allow such users to access the system. However, this might become the Achilles' heel of the authentication system as the attacker can easily relay such CAPTCHAs to remote human solver and thereby eliminate our ability to detect such the relay attack. Dealing with this problem is out of scope for this thesis.

Another limitation of our work is that we did not collect the usability data from real systems, but we conducted usability studies where we recruited MTurk workers and volunteers from our university. Such types of usability studies can be affected by Hawthorne effect which may make the results biased. However, we tried our best to minimize such effects by not observing closely the participants while they are performing the tasks, notifying the participants that the collected data are anonymized and asking them to provide honest responses to help us to improve the system.

8.2 Human-Human Authentication

In the context of Human-Human authentication, we introduced *Gametrics*. *Gametrics* identifies the user based on her unique way of interaction with simple DCGs captured by mouse dynamics and cognitive abilities of the user. We showed that incorporating interactivity in the user authentication helps in achieving our design goals and prevents the attacks underlying a strong threat model of user authentication. First, *Gametrics* offers good usability as it can identify the user with high accuracy and within a short period of time. Moreover, the DCG constructs utilized in our study were well perceived by the users shown in the high SUS scores. Second, we showed that *Gametrics* are hard to be attacked by zero-effort attackers and by expert attackers.

Finally and perhaps most importantly, we showed that the requirement of multiple interaction between the DCG and the user, and the dynamic nature of the DCGs make attacking *Gametrics* by automated means, such as *SMASheD*, hard. As such an attack would require solving the underlying DCG CAPTCHA as well as mimicking the user's way of interaction with the game.

We introduced multiple application in which *Gametrics* can be utilized, ranging from point of entry to a method of fall-back authentication.

Gametrics has some limitations, which can be explored in future work. *Gametrics* is similar to any other behavioral biometrics in that, we believe, it will suffer from a degradation in the accuracy of user identification when the user's behavior is undergoing a significant variation, such as changing emotions [116], falling sick or getting injured. The effect of behavioral changes on the performance of *Gametrics* should be subject to future work.

Future work may also need to be conducted to test the accuracy of the *Gametrics* classification models when switching devices or hardware (e.g., different kinds of mouse or screens).

The results obtained from our study are promising, however, we believe that further work is needed in order to improve the overall accuracy of user identification. One avenue in this direction is to add a little more complexity to the game challenges in order to increase the level of interaction between the challenge and the user (e.g., similar to Mix DCG games), and thereby improving the overall usability (False Negative Rate) and security (False Positive Rate) of the interactive authentication.

LIST OF REFERENCES

[1] Rich Gossweiler, Maryam Kamvar, and Shumeet Baluja. What's up captcha?: a captcha based on image orientation. In *Proceedings of the 18th international conference on World wide web*, pages 841–850. ACM, 2009.

[2] Shardul Vikram, Yinan Fan, and Guofei Gu. Semage: a new image-based two-factor captcha. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 237–246. ACM, 2011.

[3] Carlos Castillo, Debora Donato, Luca Becchetti, Paolo Boldi, Stefano Leonardi, Massimo Santini, and Sebastiano Vigna. A reference collection for web spam. *ACM Special Interest Group on Information Retrieval (SIGIR) Forum*, 40(2):11–24, 2006.

[4] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher. *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. ISBN 0131475738.

[5] Benny Pinkas and Tomas Sander. Securing passwords against dictionary attacks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 161–170, New York, NY, USA, 2002. ACM. ISBN 1-58113-612-9. doi: http://doi.acm.org/10.1145/586110.586133.

[6] L von Ahn, M Blum, NJ Hopper, and J Langford. The captcha web page, 2000.

[7] Yan, Jeff and El Ahmad, Ahmad Salah. Usability of CAPTCHAs Or usability issues in CAPTCHA design. In *Symposium On Usable Privacy and Security*, 2008.

[8] Elie Bursztein, Steven Bethard, Celine Fabry, John Mitchell, and Dan Jurafsky. How Good Are Humans at Solving CAPTCHAs? A Large Scale Evaluation. *SP '10*, pages 399–413, 2010.

[9] Chow, Richard and Golle, Philippe and Jakobsson, Markus and Wang, Lusha and Wang, XiaoFeng. Making CAPTCHAs Clickable. In *Workshop on Mobile Computing Systems and Applications*, 2008.

[10] Cory 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.

[11] Fabian Monrose and Aviel Rubin. Authentication via keystroke dynamics. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 48–56. ACM, 1997.

[12] Francesco Bergadano, Daniele Gunetti, and Claudia Picardi. User authentication through keystroke dynamics. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):367–397, 2002.

[13] Y Xu, G Reynaga, S Chiasson, JF Frahm, F Monrose, and PC Van Oorschot. Security and usability challenges of moving-object captchas: decoding codewords in motion. In *USENIX Security*, 2012.

[14] Yi Xu, Gerardo Reynaga, Sonia Chiasson, J Frahm, Fabian Monrose, and Paul Van Oorschot. Security analysis and related usability of motion-based captchas: Decoding codewords in motion. *Transactions On Dependable And Secure Computing*, 2013.

[15] José María Gómez Hidalgo and Gonzalo Alvarez. Captchas: An artificial intelligence application to web security. *Advances in Computers*, 83, 2011.

[16] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321 (5895), 2008.

[17] Monica Chew and Henry S Baird. Baffletext: A human interactive proof. In *Electronic Imaging*, 2003.

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

[19] Mohammad Shirali-Shahreza and Sajad Shirali-Shahreza. Collage captcha. In *Signal Processing and Its Applications*, 2007.

[20] Henry S Baird and Jon L Bentley. Implicit captchas. In *Electronic Imaging*, 2005.

[21] Jeremy Elson and John R. Douceur and Jon Howell and Jared Saul. Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization. In *Conference on Computer and Communications Security*, 2007.

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

[23] Kurt Alfred Kluever and Richard Zanibbi. Balancing usability and security in a video captcha. In *Symposium on Usable Privacy and Security*, 2009.

[24] John Nicholas Gross. Captcha using challenges optimized for distinguishing between humans and machines, 2009. US Patent App. 12/484,800.

[25] Graig Sauer and Harry Hochheiser and Jinjuan Feng and Jonathan Lazar. Towards a Universally Usable CAPTCHA. In *Symposium On Usable Privacy and Security*, 2008.

[26] Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. Re: Captchas-understanding captcha-solving services in an economic context. In *USENIX Security*, pages 435–462, 2010.

[27] Jeff Yan and Ahmad Salah El Ahmad. A Low-cost Attack on a Microsoft CAPTCHA. In *Conference on Computer and Communications Security*, 2008.

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

[29] 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.

[30] Robert Morris and Ken Thompson. Password security: a case history. *Commun. ACM*, 22(11):594–597, 1979.

[31] Jeff Yan, Alan Blackwell, Ross Anderson, and Alasdair Grant. Password memorability and security: Empirical results. *IEEE Security and Privacy*, 2(5):25–31, 2004.

[32] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42 (12):40–46, 1999.

[33] Xiaoyuan Suo, Ying Zhu, and G. Scott Owen. Graphical passwords: A survey. In *ACSAC*, 2005.

[34] Robert Biddle, Sonia Chiasson, and Paul Van Oorschot. Graphical passwords: Learning from the first generation. In *Technical Report TR-09-09, School of Computer Science, Carleton University*, 2009.

[35] Adrian Perrig and Dawn Song. Hash visualization: a new technique to improve real-world security. In *CrypTEC*, 1999.

[36] Susan Wiedenbeck, Jim Waters, Leonardo Sobrado, and Jean-Camille Birget. Design and Evaluation of a Shoulder-surfing Resistant Graphical Password Scheme. In *Proceedings of the working conference on Advanced visual interfaces (AVI)*, 2006.

[37] The Science Behind Passfaces. http://www.realuser.com/. Last access, December 2008.

[38] Ian Jermyn, Alain Mayer, Fabian Monrose, Michael K. Reiter, and Aviel D. Rubin. The design and analysis of graphical passwords. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, 1999.

[39] Paul Dunphy and Jeff Yan. Do background images improve "draw a secret" graphical passwords? In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 36–47. ACM, 2007.

[40] Susan Wiedenbeck, Jim Waters, Jean-Camille Birget, Alex Brodskiy, and Nasir D. Memon. PassPoints: Design and Longitudinal Evaluation of a Graphical Password System. In *International Journal of Human Computer Studies*, 2005.

[41] Sacha Brostoff and M. Angela Sasse. Are passfaces more usable than passwords? a field trial investigation. In *HCI 2000: Proceedings of People and Computers XIV - Usability or Else*, pages 405–424, 2000.

[42] Fred Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13:319–340, 1989.

[43] P. C. van Oorschot and Julie Thorpe. On predictive models and user-drawn graphical passwords. *ACM Trans. Inf. Syst. Secur.*, 10(4):1–33, 2008.

[44] Krzysztof Golofit. Click passwords under investigation. In *ESORICS '07: Proceedings of the 12th European symposium on Research In Computer Security*, pages 343–358, 2007.

[45] Ahmet Emir Dirik, Nasir Memon, and Jean-Camille Birget. Modeling user choice in the passpoints graphical password scheme. In *SOUPS '07: Proceedings of the 3rd symposium on Usable privacy and security*, pages 20–28, 2007.

[46] Julie Thorpe and P. C. van Oorschot. Human-seeded attacks and exploiting hot-spots in graphical passwords. In *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 1–16, 2007.

[47] Amirali Salehi-Abari, Julie Thorpe, and P. C. van Oorschot. On purely automated attacks and click-based graphical passwords. In *ACSAC '08: Proceedings of the 2008 Annual Computer Security Applications Conference*, pages 111–120, 2008.

[48] Umut Uludag, and Anil K. Jain. Attacks on Biometric Systems: A Case Study in Fingerprints. In *Conference on Security, Steganography, and Watermarking of Multimedia Contents*, 2004.

[49] Tsutomu Matsumoto, Hiroyuki Matsumoto, Koji Yamada, and Satoshi Hoshino. Impact of Artificial Gummy Fingers on Fingerprint Systems. In *Conference on Optical Security and Counterfeit Deterrence Techniques*, 2002.

[50] Yunhong Wang, Tieniu Tan, and Anil K. Jain. Combining Face and Iris Biometrics for Identity Verification. In *Audio- and Video-Based Biometric Person Authentication Conference*, 2003.

[51] Andrew Patrick. Usability and Acceptability of Biometric Security Systems. In *Financial Cryptography*, Available at: http://www.andrewpatrick.ca/biometrics/NATO-BiometricsAbstract.pdf, 2004.

[52] E. Eugene Schultz, Robert W. Proctor, Mei-Ching Lien and Gavriel Salvendy. Usability and Security: An Appraisal of Usability Issues in Information Security Methods. *Computers and Security*, 20(7), 2001.

[53] Barry Marshall. Does Fingerprint ID at Entry Portals Spread Swine Flu? Available at http://barryjmarshall.blogspot.com/2009/06/does-fingerprint-id-at-entry-portals.html.

[54] H. Proenca. Towards Non-Cooperative Biometric Iris Recognition. Available at: http://www.di.ubi.pt/~hugomcp/doc/TesePhD_HugoMCP.pdf, 2007.

[55] Kuan-Ta Chen and Li-Wen Hong. User identification based on game-play activity patterns. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 7–12. ACM, 2007.

[56] Arik Messerman, Tarik Mustafic, Seyit Ahmet Camtepe, and Sahin Albayrak. Continuous and non-intrusive identity verification in real-time environments based on free-text keystroke dynamics. In *Biometrics (IJCB), 2011 International Joint Conference on*, pages 1–8. IEEE, 2011.

[57] Yu Zhong, Yan Deng, and Anubhav K Jain. Keystroke dynamics for user authentication. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 117–123. IEEE, 2012.

[58] Nan Zheng, Aaron Paloski, and Haining Wang. An efficient user verification system via mouse movements. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 139–150. ACM, 2011.

[59] Asadullah Al Galib and Reihaneh Safavi-Naini. User authentication using human cognitive abilities. In *Financial Cryptography and Data Security*, pages 254–271. Springer, 2015.

[60] Manar Mohamed, Babins Shrestha, and Nitesh Saxena. Smashed: Sniffing and manipulating android sensor data. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 152–159. ACM, 2016.

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

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

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

[64] Deian Stefan and Danfeng Yao. Keystroke-dynamics authentication against synthetic forgeries. In *CollaborateCom*, 2010.

[65] John Brooke. SUS: a "quick and dirty" usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, and A. L. McClelland, editors, *Usability Evaluation in Industry*. Taylor and Francis, London, 1996.

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

[67] Bin B Zhu, Jeff Yan, Qiujie Li, Chao Yang, Jia Liu, Ning Xu, Meng Yi, and Kaiwei Cai. Attacks and design of image recognition captchas. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 187–200. ACM, 2010.

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

[69] Shu ching Chen, Mei ling Shyu, Chengcui 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.

[70] Slawo Wesolkowski. Stochastic nested aggregation for images and random fields. In *Ph.D. Thesis, University of Waterloo*, 2007.

[71] Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. Re: Captchas-understanding captcha-solving services in an economic context. In *USENIX Security Symposium*, 2010.

[72] Nucaptcha. http://www.nucaptcha.com.

[73] Robert Kosinski. A literature review on reaction time. Available at: http://biology.clemson.edu/bpc/bp/Lab/110/reaction.htm, 2012.

[74] Cynthia Taylor and Joseph Pasquale. Improving video performance in vnc under high latency conditions. In *Collaborative Technologies & Systems*, 2010.

[75] Realvnc. http://www.realvnc.com/.

[76] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3), 2011.

[77] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3), 1977.

[78] Niloy J. Mitra, Hung-Kuo Chu, Tong-Yee Lee, Lior Wolf, Hezy Yeshurun, and Daniel Cohen-Or. Emerging images. *ACM Transactions on Graphics*, 28(5), 2009. to appear.

[79] John K Tsotsos. On the relative complexity of active vs. passive visual search. *International journal of computer vision*, 1992.

[80] John Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189: 194, 1996.

[81] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 2009.

[82] Akamai. Akamai's state of the internet: Q4 2014 report. http://www.stateoftheinternet.com/resources-connectivity-2014-q4-state-of-the-internet-report.html.

[83] Manar Mohamed, Niharika Sachdeva, Michael Georgescu, Song Gao, Nitesh Saxena, Chengcui Zhang, Ponnurangam Kumaraguru, Paul C van Oorschot, and Wei-Bang Chen. A three-way investigation of a game-captcha: automated attacks, relay attacks and usability. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 195–206. ACM, 2014.

[84] Jeff Sauro. Measuring usability with the system usability scale (sus). http://www.measuringu.com/sus.php. Accessed: 2015-10-12.

[85] Mauro Conti, Claudio Guarisco, and Riccardo Spolaor. Captchastar! a novel captcha based on interactive shape discovery. *arXiv preprint arXiv:1503.00561*, 2015.

[86] Elie Bursztein, Steven Bethard, Celine Fabry, John C Mitchell, and Dan Jurafsky. How good are humans at solving captchas? a large scale evaluation. In *2010 IEEE Symposium on Security and Privacy*, pages 399–413. IEEE, 2010.

[87] Android. Android security. https://goo.gl/ihIwi3.

[88] Edward Kim. No root screenshot it. https://play.google.com/store/apps/details?id=com.edwardkim.android.screenshotitfullnoroot, 2013.

[89] ClockworkMod. Helium. https://play.google.com/store/apps/details?id=com.koushikdutta.backup.license, 2013.

[90] ClockworkMod. Clockworkmod tether (no root). https://goo.gl/qg2e80.

[91] strAI. Frep - finger replayer. https://play.google.com/store/apps/details?id=com.x0.strai.frep&hl=en, 2015.

[92] Chia-Chi Lin, Hongyang Li, Xiaoyong Zhou, and X Wang. Screenmilker: How to milk your android screen for secrets. In *Network and Distributed System Security Symposium*, 2014.

[93] ProHiro.com. Hiromacro auto-touch macro. https://goo.gl/1T5pnx.

[94] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. Reran: Timing-and touch-sensitive record and replay for android. In *Software Engineering (ICSE)*, 2013.

[95] [David Rogers. *Mobile Security: A Guide for Users*. lulu.com, 2013. ISBN 978-1-291-53309-5.

[96] Sungjae Hwang, Sungho Lee, Yongdae Kim, and Sukyoung Ryu. Bittersweet adb: Attacks and defenses. In *ACM Symposium on Information, Computer and Communications Security*, 2015.

[97] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *IEEE Transactions on Information Forensics and Security*, 2013.

[98] Adam J Aviv, Benjamin Sapp, Matt Blaze, and Jonathan M Smith. Practicality of accelerometer side channels on smartphones. In *Computer Security Applications Conference*, 2012.

[99] Liang Cai and Hao Chen. *On the practicality of motion based keystroke inference attack*. Springer, 2012.

[100] Zhi Xu, Kun Bai, and Sencun Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *ACM conference on Security and Privacy in Wireless and Mobile Networks*, 2012.

[101] Laurent Simon and Ross Anderson. Pin skimmer: Inferring pins through the camera and microphone. In *ACM workshop on Security and privacy in smartphones & mobile devices*, 2013.

[102] Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. Touch me once and i know it's you!: Implicit authentication based on touch screen patterns. In *SIGCHI Conference on Human Factors in Computing Systems*, CHI, 2012.

[103] Jun-Ki Min, Afsaneh Doryab, Jason Wiese, Shahriyar Amini, John Zimmerman, and Jason I Hong. Toss'n'turn: smartphone as sleep and sleep quality detector. In *ACM conference on Human factors in computing systems*, 2014.

[104] Jun Yang, Emmanuel Munguia-Tapia, and Simon Gibbs. Efficient in-pocket detection with mobile phones. In *ACM conference on Pervasive and ubiquitous computing adjunct publication*, 2013.

[105] P Campisi, E Maiorana, M Lo Bosco, and A Neri. User authentication using keystroke dynamics for cellular phones. *IET Signal Processing*, 3(4), 2009.

[106] Abdul Serwadda and Vir V. Phoha. When kids' toys breach mobile phone security. In *Conf. on Computer & Communications Security*, 2013.

[107] Lingjun Li, Xinxin Zhao, and Guoliang Xue. Unobservable re-authentication for smartphones. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[108] Muhammad Shahzad, Alex X. Liu, and Arjmand Samuel. Secure unlocking of mobile touch screen devices by simple gestures: You can see it but you can not do it. In *Mobile Computing & Networking*, 2013.

[109] Mauro Conti, Irina Zachia-Zlatea, and Bruno Crispo. Mind how you answer me!: transparently authenticating the user of a smartphone when answering or placing a call. In *ACM Symposium on Information, Computer and Communications Security*, 2011.

[110] Hugo Gascon, Sebastian Uellenbeck, Christopher Wolf, and Konrad Rieck. Continuous authentication on mobile devices by analysis of typing motion behavior. In *Sicherheit*, 2014.

[111] Roy A Maxion and Kevin S Killourhy. Keystroke biometrics with number-pad input. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 201–210. IEEE, 2010.

[112] Abdul Serwadda and Vir V Phoha. When kids' toys breach mobile phone security. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 599–610. ACM, 2013.

[113] Sanket Misal. Common lock patterns of mobiles phones screen lock, 2013. "http://mytrickytricks.blogspot.com/2013/07/commonlockpattern.html".

[114] Alex Hern. Google aims to kill passwords by the end of this year. https://www.theguardian.com/technology/2016/may/24/google-passwords-android.

[115] Steven A Ross, J Alex Halderman, and Adam Finkelstein. Sketcha: a captcha based on line drawings of 3d models. In *Proceedings of the 19th international conference on World wide web*, pages 821–830. ACM, 2010.

[116] Clayton Epp, Michael Lippold, and Regan L Mandryk. Identifying emotional states using keystroke dynamics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 715–724. ACM, 2011.

APPENDIX: IRB Approval

OMB No. 0990-0263
Approved for use through 3/31/2015

Protection of Human Subjects
Assurance Identification/IRB Certification/Declaration of Exemption
(Common Rule)

Policy: Research activities involving human subjects may not be conducted or supported by the Departments and Agencies adopting the Common Rule (56FR28003, June 18, 1991) unless the activities are exempt from or approved in accordance with the Common Rule. See section 101(b) of the Common Rule for exemptions. Institutions submitting applications or proposals for support must submit certification of appropriate Institutional Review Board (IRB) review and approval to the Department or Agency in accordance with the Common Rule.

Institutions must have an assurance of compliance that applies to the research to be conducted and should submit certification of IRB review and approval with each application or proposal unless otherwise advised by the Department or Agency.

1. Request Type
[X] ORIGINAL
[ ] CONTINUATION
[ ] EXEMPTION

2. Type of Mechanism
[ ] GRANT [X] CONTRACT [ ] FELLOWSHIP
[ ] COOPERATIVE AGREEMENT
[ ] OTHER:_____

3. Name of Federal Department or Agency and, if known, Application or Proposal Identification No.

4. Title of Application or Activity
Usable Security via Extrinsic Motivation: An Investigation Into Rewards, Games and Game Elements

5. Name of Principal Investigator, Program Director, Fellow, or Other
SAXENA, NITESH

6. Assurance Status of this Project *(Respond to one of the following)*

[X] This Assurance, on file with Department of Health and Human Services, covers this activity:
Assurance Identification No. FWA00005960, the expiration date 01/24/2017 IRB Registration No. IRB00000196

[ ] This Assurance, on file with *(agency/dept)*_____, covers this activity.
Assurance No._____, the expiration date_____ IRB Registration/Identification No._____*(if applicable)*

[ ] No assurance has been filed for this institution. This institution declares that it will provide an Assurance and Certification of IRB review and approval upon request.

[X] Exemption Status: Human subjects are involved, but this activity qualifies for exemption under Section 101(b), paragraph 2.

7. Certification of IRB Review (Respond to one of the following IF you have an Assurance on file)

[ ] This activity has been reviewed and approved by the IRB in accordance with the Common Rule and any other governing regulations.
by: [ ] Full IRB Review on (date of IRB meeting) _____ or [ ] Expedited Review on (date)_____
[ ] If less than one year approval, provide expiration date _____
[ ] This activity contains multiple projects, some of which have not been reviewed. The IRB has granted approval on condition that all projects covered by the Common Rule will be reviewed and approved before they are initiated and that appropriate further certification will be submitted.

8. Comments
Protocol subject to No renewal continuing review.

Title E130115003
Usable Security via Extrinsic Motivation: An Investigation Into Rewards, Games and Game Elements

IRB Approval Issued: 2/13/13

9. The official signing below certifies that the information provided above is correct and that, as required, future reviews will be performed until study closure and certification will be provided.

10. Name and Address of Institution
University of Alabama at Birmingham
701 20th Street South
Birmingham, AL 35294

11. Phone No. *(with area code)* (205) 934-3789

12. Fax No. *(with area code)* (205) 934-1301

13. Email: irb@uab.edu

14. Name of Official
Cari Oliver

15. Title
Assistant Director, IRB

16. Signature

17. Date 2/13/13

Authorized for local Reproduction

Sponsored by HHS

According to the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number. The valid OMB control number for this information collection is 0990-0263. The time required to complete this information collection is estimated to average 30 minutes per response. If you have comments concerning the accuracy of the time estimate(s) or suggestions for improving this form, please write to: U.S. Department of Health & Human Services, OS/OCIO/PRA, 200 Independence Ave., S.W., Suite 336-E, Washington D.C. 20201, Attention: PRA Reports Clearance Officer.

IRB Exemption Review Application

- **To complete the form**, click the underlined areas and type or paste in your text; double-click checkboxes to check/uncheck. For more tips, see www.uab.edu/irb/forms.
- **Mail or deliver all materials to AB 470**, 701 20th Street South, Birmingham, AL 35294-0104.

Indicate the type of review:
☒ **New** ☐ **Continuing Review** ☐ **Final Report**

1. Project Identification

a. Title of Project: **Usable Security via Extrinsic Motivation: An Investigation into Rewards, Games and Game Elements**

b. Principal Investigator (PI):**Nitesh Saxena, PhD** PI's BlazerID or E-Mail Address:**saxena@uab.edu**
If the PI is a student, fellow, or resident, provide the name, number, and email of the faculty advisor or course instructor as contact information and obtain the person's signature.
Advisor/Instructor's Name:_____ Telephone Number:_____ BlazerID:_____
Advisor/Instructor's Signature:_____

c. PI's Address (on-campus or home)
On-Campus: Department: **Computer and Information Sciences** Building: **Campbell Hall** Room:**115** UAB Zip:**1170**
Phone:**205 934-2213** FAX:_____
-OR-
Home Address:_____ Street:_____ City:_____ State:_____ ZIP:_____
and Campus Affiliation:_____

d. List all staff who will be involved with the research, their degree(s) and job title, and any additional qualifications. Include individuals who will be involved in the consent process. *Repeat the table below for each individual.*
Note. For studies involving investigational drugs, include all investigators who will be listed on FDA Form 1572 and attach a copy, if applicable. Send the IRB a copy of Form 1572 anytime you update the form with the FDA.

Role:	☐Co- -*OR*- ☒Other
Full Name:	**Manar Mohamed**
Primary UAB Dept.: (Employer if not UAB)	**Computer and Information Sciences**
Degree(s) / Job Title:	**Graduate Student Researcher**

Role:	☐Co- -*OR*- ☐Other
Full Name:	_____
Primary UAB Dept.: (Employer if not UAB)	_____
Degree(s) / Job Title:	_____
Additional Qualifications pertinent to the study:	_____

e. Is this activity funded in any way? ☒Yes ☐No
If yes, attach 1 copy of completed application and complete (i)-(iv):

i. Grant or Contract Title: Usable Security via Extrinsic Motivation: An Investigation into Rewards, Games and Game Elements (pending)

ii. PI of Grant or Contract: Nitesh Saxena, PhD
iii. OGCA Tracking Number: OSP Link Number: 425911
iv. Funding Source
☒Gov't Agency or Agencies: **NSF**

☐UAB Departmental Funds:_____
☐Other:_____

2. Mark the category or categories below that describe the proposed research:

☐1. Research conducted in established or commonly accepted educational settings, involving normal educational practices, such as (i) research on regular and special education instructional strategies, or (ii) research on the effectiveness of or the comparison among instructional techniques, curricula, or classroom management methods. The research is not FDA regulated and does not involve prisoners as participants.

☒2. Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures or observation of public behavior, unless: (i) Information obtained is recorded in such a manner that human subjects can be identified, directly or through identifiers linked to the subjects; and (ii) any disclosure of the human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation. Attach questionnaire(s) and/or surveys. If the research involves children as participants, the procedures are limited to educational tests and observation of public behavior where the investigators do not participate in the activities being observed. The research is not FDA regulated and does not involve prisoners as participants.

☐3. Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures, or observation of public behavior that is not exempt under category (2), if: (i) the human subjects are elected or appointed public officials or candidates for public office; or (ii) federal statute(s) require(s) without exception that the confidentiality of the personally identifiable information will be maintained throughout the research and thereafter. Attach to this application a copy of any questionnaire or survey to be used. The research is not FDA regulated and does not involve prisoners as participants.

☐4. Research involving the collection or study of existing data, documents, records, pathological specimens, or diagnostic specimens, if these sources are publicly available or if the information is recorded by the Investigator in such a manner that subjects cannot be identified, directly or through identifiers linked to the subjects. Attach a specimen release form if applicable. (*Specimens must be preexisting.*) The research is not FDA regulated and does not involve prisoners as participants.

☐5. Research and demonstration projects which are conducted by or subject to the approval of department or agency heads, and which are designed to study, evaluate, or otherwise examine: (i) public benefit or service programs; (ii) procedures for obtaining benefits or services under those programs;(iii) possible changes in or alternatives to those programs or procedures; or (iv) possible changes in methods or levels of payment for benefits or services under those programs. The protocol will be conducted pursuant to specific federal statutory authority; has no statutory requirement for IRB review; does not involve significant physical invasions or intrusions upon the privacy interests of the participant; has authorization or concurrent by the funding agency and does not involve prisoners as participants.

☐6. Taste and food quality evaluation and consumer acceptance studies, (i) if wholesome foods without additives are consumed or (ii) if a food is consumed that contains a food ingredient at or below the level and for a use found to be safe, or agricultural chemical or environmental contaminant at or below the level found to be safe, by the Food and Drug Administration or approved by the Environmental Protection Agency or the Food Safety and Inspection Service of the U.S. Department of Agriculture. The research does not involve prisoners as participants.

3. Briefly describe the proposed research: **The research procedures involve subjects playing several computerized games and filling out questionnaires about their experiences with these approaches.** Specifically, these games challenge the user to perform a cognitive task interacting with a series of dynamic images and take the form of many objects floating around within the images, and the user's task is to match the objects corresponding to given target(s) and drag-and-drop them to the target region(s). It is believed that such games can be easily played by human users but will be difficult for computer programs to

solve and serve as a tool to prevent attack programs from exploiting web services. This research aims at formally evaluating the usability of such game-based mechanisms. Here is the testing process:

Our study will be conducted at an on-campus (at our university) location, specifically in our graduate student lab of our department. Same computer terminal will be used for the tests with all participants.

An overview of the testing process will be first given to each respondent prior to the actual tests.
First, the respondents will be asked to fill out the the Pre-Test questionnaire. Next, the respondents will be asked to several versions of the games described above. Finally, the respondents will be required to fill out the Post experiment questionnaire.

The PI will be himself running and scheduling the sessions with the participant based on communication with the test participants. A walk-in session will also be conducted where the participants will just come in and perform the tests. The sessions will take place in a graduate student lab in our department. These will be one-to-one sessions.

4. Describe how subjects/data/specimens will be selected. If applicable, include the sex, race, and ethnicity of the subject population: **Volunteers will be solicited by e-mail and asked, if they are over the age of 19, if they are interested in participating in a study to evaluate some aspects of modern technology. Sex, race, and ethnicity of the subject population are not factors in the selection of subjects.** We will recruit from student populations. The student participants will be all university students, studying towards undergraduate, Master's and Ph.D. degrees in Computer Science or closely related fields. We will recruit by emailing to the appropriate mailing lists (such as student lists in the CS department). In addition, we will recruit via word of mouth and personal communications.

5. Does the research involve deception? ☐Yes ☒No

6. Describe why none of the research procedures would cause a subject either physical or psychological discomfort or be perceived as harassment above and beyond what the person would experience in daily life: **Subjects who volunteer are asked to perform normal functions on computer terminals and to give their feedback on their experiences. Subjects are told to stop the process at anytime they wish. There is no physical harm and absolutely no discomfort involved in participating to this study.**

7. Describe the provisions to maintain confidentiality of data: **All information collected will be kept strictly CONFIDENTIAL and will only be referenced with a code.** This code is a random one and will be pre-assigned to each participant (i.e., the questionnaires). In our dataset, we will reference the recorded responses with this code.. **We do not collect any data that can directly identify participants and any information that can help to personally identify anyone will not be voluntarily released or disclosed.**

8. Describe the provisions included in the research to protect the privacy interests of participants (e.g., others will not overhear your conversation with potential participants, individuals will not be publicly identified or embarrassed): **Participants will be playing computer games and will not engage meaningfully with other participants. After their evaluation is complete, participants will be asked to fill out a questionnaire and to skip any questions they do not want to answer. The questionnaires do not contain any participant indentifying information.**

9. Will the research involve interacting with the subjects? ☒Yes ☐No

If yes, describe the consent process and information to be presented to subjects, including:

- That the activities involve research.
- The procedures to be performed.
- That participation is voluntary.
- Name and contact information for the investigator.

The study activities and procedures are described in the handouts. The voluntary nature of the participation is clearly stated in the Study Information Sheet. The interaction with the

participants consists of a background questionnaire and a posttest questionnaire, and possibly some questions about user experience as the evaluation is underway. Neither the background questionnaire, nor the posttest questionnaire/questions ask for any information that could in any way be linked to the study participants. Volunteers are given a one-sheet handout explaining the research and the evaluation process and their role as participants. The handout explains that participation is strictly voluntary and that subjects may cease participation at any time they choose. Researcher name and contact information are on the handout.

10. Additional Information

In the space below, provide any additional information that you believe may help the IRB review the proposed research, or enter "None."

None

11. Findings? **(applicable for Continuing Review or Final Report only**)

State both the positive and negative results received to date:____

Since the last IRB review, have any of the following occurred?

a. Have participants experienced any harms (expected or unexpected)? ☐Yes ☐No
If yes, attach Problem Summary Sheet, and briefly describe here the harms (serious and/or non-serious) experienced by participants:____

b. Have there been any unanticipated problems involving risks to participants or others? ☐Yes ☐No
If yes, attach Problem Report, and briefly describe here the unanticipated problems involving risks to participants or others:____

c. Have you have any problems obtaining informed consent? ☐Yes ☐No ☐N/A
If yes, briefly describe the problems here:____

d. Have any participants or others complained about the research? ☐Yes ☐No
If yes, briefly describe the number and nature of the complaints:____

e. Have any participants withdrawn from the research? ☐Yes ☐No
If yes, indicate the number of withdrawals and include the reason for each:____

f. Have any obvious, study-related benefits occurred for participants? ☐Yes ☐No
If yes, briefly describe the benefits here:____

g. Have the risks or potential benefits of this research changed? ☐Yes ☐No
If yes, briefly describe the changes here:____

h. Has there been any published literature? ☐Yes ☐No
If yes, attach a copy and summarize the published findings here:____

Principal Investigator's Signature:_____ Date:**January 11, 2013**