4-2011

An implementation of a multi-touch Draw-A-Secret password schema for Windows-based computers.

Matthew Lichtenberger
University of Louisville

Recommended Citation
Lichtenberger, Matthew, "An implementation of a multi-touch Draw-A-Secret password schema for Windows-based computers." (2011). *Electronic Theses and Dissertations.* Paper 830.
https://doi.org/10.18297/etd/830

Title Page

AN IMPLEMENTATION OF A MULTI-TOUCH DRAW-A-SECRET PASSWORD SCHEMA FOR WINDOWS-BASED COMPUTERS

By

Matthew Lichtenberger
B.S., University of Louisville, 2009

A Thesis
Submitted to the Faculty of the
University of Louisville
J. B. Speed School of Engineering
as Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Computer Engineering and Computer Science

April 2011

Approval Page

AN IMPLEMENTATION OF A MULTI-TOUCH DRAW-A-SECRET PASSWORD SCHEMA FOR WINDOWS-BASED COMPUTERS

Submitted by:___________________________________
Matthew Lichtenberger

A Thesis Approved On

____
(Date)

by the Following Reading and Examination Committee:

____
Dr. Roman V. Yampolskiy, Thesis Director

____
Professor Michael M. Losavio, A&S Justice Administration

____
Dr. Anup Kumar, Spd-Comp Eng. & Comp Sci.

ACKNOWLEDGEMENTS

The author wishes to thank several people for their assistance along the duration this work was performed:

Roman V. Yampolskiy: For providing succinct and sapient advice at provided research meetings.

Members of Thesis Committee - Dr. Anup Kumar, Professor Michael M. Losavio, and Dr. Roman V. Yampolskiy: For allowing me the time to present my discoveries and for providing suggestions on furthering my research.

Jesse Determann: For assistance with algorithms and theories of operation.

Hak.5 Video Podcast: For the initial work on accepting touch input through webcams.

Community Core Vision: For providing a suite package of the above work.

ABSTRACT

Multi-Touch Draw A Secret Solution Implemented in C# for Microsoft Windows Computing Devices

Author: Matthew Lichtenberger

Draw A Secret password systems have recently come into vogue, primarily in the role of protecting a user’s cellular smart phone from external infiltration in lieu of a password. However, these password systems are only enabled for single touch operation; that is to say, a user may only draw one pattern, with only one finger. Brute forcing these passwords thereby becomes trivial in the context of password complexity. This thesis implements a Draw A Secret system utilizing consumer-grade video hardware to collate multiple patterns synchronously and then authenticate the user at a later point. Care has been given to properly encrypt the results to prevent a malicious third party from infiltrating the authentication stream from a file system perspective.

Table of Contents

List of Figures

I. General Introduction

Digital security measures have always been challenging to implementations, requiring large investments in effort and funding to acquire authentic security, and not simply the illusion of such. Many present day security methods suffer from weaknesses and vulnerabilities. In an attempt to familiarize the reader, a quick review will be given the present state of the field of Information Security. Additional resources can be found in academic texts and papers on the subject; a search for keywords *Information Security* in the IEEE Xplore database returns nearly 36,000 results on the subject.

One of the primary tenants of Information Security is Authentication, or the principle of determining the allowances that should be given to a given user of the system. Authentication schemes come in two flavors: Single-factor and Multi-factor. These schemes operate on a triumvirate of ideologies, that of "something you know" (SYK), "something you have" (SYH), and "something you are" (SYA) (Schneider, 2005). As one would expect, Single-factor authentication focuses on only one of these ideologies, whereas Multi-factor authentication utilizes two or more to provide additional security.

Passwords are a common method of authentication utilized in computer security today. These exhibit the SYK ideology, in that they rely on the user generating some sort of memorable information code that is then stored in the computer. In any successive login attempt the user is queried for this code, and must provide a matching code to the enrolled value he stored during registration in order to proceed. Several different mechanisms have been developed over the years to try to strengthen the SYK ideology, as users have difficulty remembering truly secure systems.

- A standard password is one in which a length of alphanumeric symbols is permitted. Depending on the security required, there may be policies in place to restrict registration acceptance unless certain criteria are filled. Examples of these policies include containing numbers and special characters, or having a mixture of upper and lower case letters. Additionally, there may be a requirement for the length to exceed a certain threshold.

- Formula passwords are very similar to a standard password, except the user may have some algorithm that varies parameters of the password depending on the context the password is utilized (Thomas, 2005). For example, a standard password may be "doggy123", but the user may want to use this password for both their bank account and their email account. By using some defined formula, the user can hash the domain name into the password, as in this example: "dgomgagiyl123" where every other letter is part of the word "gmail", their password is much improved. Quantitatively, Passwordmeter.com ranks the first password at 37%, whereas the hashed password is a full 10% better, at 47%.

- A one-time password scheme involves a remote server backend that is previously authorized to authenticate a user. The user is provided with some way to receive information from the server (sometimes tied in with the Token based SYH authentication systems) that they utilize as their password or, alternatively, append to the beginning or end of their password. Upon submission, the server compares this against their saved value and verifies via the third party server that the addition was valid (Griffin, 2008).

- Similar to the aforementioned Standard Password is the Pattern-based Password, but in this case a graphical depiction is utilized instead of a word or phrase. The most common methodology is known as "Draw A Secret", whereby a user is prevented with a grid of dots. The user traces a pattern in the dots during enrollment, and all subsequent authentication attempts are compared against this enrolled pattern to determine the user's identity (Dunphy & Yan, 2007). This thesis proposes utilizing this method in a novel context to provide additional security.

A Biometric is the next step in authentication systems. Biometrics exhibit the SYA ideology, by reading in one or more measurements from some portion of the user. The name is derived from this measurement… a metric of a biological entity. Biometrics are rapidly beginning to come into their own, with systems implemented in consumer devices such as laptops, cell phones, and tablet PCs to authenticate the user with a value read from some aspect of their personage. Commonly utilized methods of measuring biometrics are provided below, with a brief summary of each.

- The most commonly known of the biometrics is the fingerprint. This system scans in the fingerprint of a user attempting to authenticate and compares it against the database stored entry that the user account was enrolled with. There are three aspects with which the system compares the recorded scan to the enrolled value: ridge ending (a termination of a line), bifurcation (a split of one line into two), and short ridge (a line that starts and stops within the window of comparison) (Thornton, 2000). This methodology is very similar to police operation when collecting and dealing with fingerprints.

- A palm reader utilizes a very similar system to the fingerprint; however, in various security tests it has been shown to be more secure than a simple fingerprint analysis. The palm print does not get pressed against items as readily as the fingerprint, and thus duplication in a viscous compound is more difficult. Furthermore, the palm data read in will also contain texture data, various indentations, and other such marks that can be utilized to make a more detailed (and thus more reliable) comparison (Subcommittee on Biometrics, 2006).

- Another pattern-based identification system, retinal scanning is based on the blood vessels that exist in the back of your eye, where light is focused onto the optic nerve. These scanners are a fundamental aspect of governmental and military security, but require up to fifteen seconds to complete their scan, which prevents them from utilization in more mundane contexts.

- Vocal pattern analysis measures aspects of a user's voice as they state some phrase or speak continuously. In the former case, this is coupled with a password; in the latter, it is a heuristic approach. Analysis of measurements focuses on both anatomic aspects (size and shape of the glottis, for example) and societal/behavioral aspects (pitch and speed of the recitation).

- Commonly used in methods to analyze a crowd and identify one member out of the bunch, this methodology is still a valid security authentication system. Unlike previous methodologies, which focus on one discrete measurement or aspect, this method of authentication compares several, and then collates these measurements into one value that describes the system's confidence in the identity of the user.

- DNA authentication is one of the least commonly used biometrics, reserved for the highest security contexts due to the complexity of equipment necessary to authenticate the users. While providing a margin of error that is closest to 0 in authenticating a user compared to the other methods discussed here, it also suffers most readily from spoofing, as the human body readily sheds cells containing the user's DNA.

- Continual work is being done with Behavioral biometrics in academic circles; however, they have the potential to transform user authentication from an explicit action to implicit and continuous monitoring. Behavioral biometrics measures some portion of the user's action and compares these actions heuristically against previous records. While each aspect may only have 60-70% confirmation rate, combining multiple aspects can result in a very secure authentication system. Common behavioral biometrics include typing analysis (speed, frequency, key hold length, etc.) and pointer behavioral (speed, hard transitions, soft transitions, etc.).

Token-based factors work off the SYH ideology, and require the user to keep an item in their possession that can interact with the security system. While in limited use for information systems at present, most people unknowingly utilize this system every day through their automobiles.

- Similar to a USB Flash Memory Stick, USB tokens communicate to an application and contain a cryptographically secure value that is protected from exfiltration in some manner. Sometimes, the aforementioned One-Time

Use Password system will also be implemented, such that the key will provide a cryptographic hash that can be verified by a remote server.

- Wireless authentication tokens do not have to be plugged into a computer to authenticate the user; instead, they generally communicate with an internal component (or external USB receiver) and perform the authentication conversation when the user is close enough. While they are slowly making inroads into computer authentication, nearly all car keys nowadays utilize this technology to better secure the ignition.

Very shortly after computers were first successfully networked, the first proof of concept attacks began appearing. Initially they focused more on stealing time on time-shared computer resources, which were at the time billed due to their high cost of operation. As the Internet became more integrated in day to day operations of business and individuals alike, these attacks changed focus to better exploit the sensitive data and personal information that was now being stored on these systems. They can be classified as External, Internal, or Hybrid, and are defined by the vector by which they attempt to breach the security.

Externally Facing Vulnerabilities are pieces of software, hardware, and/or "people ware" that accept data from un-trusted sources. As a result, they are often the most targeted by malicious users, and thus where most of the investment in IT Security is spent. These potential threats may include poorly updated software**,** badly designed hardware, and may include social engineering attacks, whereby an illicit user attempts to gain access by convincing an authorized user that the illicit user has access where he does not. As before, once access is gained, the attacks that can be utilized are quite varied, and thus most efforts are to prevent them from gaining access.

Internal Facing Vulnerabilities are the other fundamental class of attacks, and involve users who have been granted some form of authentication into the system. These attacks are much more difficult to protect against, as they usually leverage access that was legitimately granted to perform malicious actions. Risk Management is generally the only counter to these attacks, as the administrative overhead to monitor actions performed by internal actors is often too difficult. Utilizing *The Principle of Least Privilege* is the primary way of limiting the risk, as it limits the rights of a given employee to exactly what is needed to complete their job (Barkley, 1995). When an employee is removed from active duty, however, best practices necessitate Rapid Response paradigms to lock the employee account out of the network and, most importantly, sensitive data before they can do harm. However, these can only do so much, and internal attack mitigation is one of the principle areas of research for information security analysts.

These two meta-vulnerabilities require radically different paradigms for protection, with the former being much more insidious than the latter. In the case of Insider Knowledge attacks, the attack may occur for several months or even years before it is detected. Externally Facing Vulnerabilities are usually more immediate and are more likely to be detected and corrected before significant harm can be done. Several protection paradigms have been implemented in the past two decades to provide countermeasures to both of these threat vectors, with various levels of efficacy versus investment of time and monetary resources.

For external attacks, there is an extremely effective security paradigm, known as Defense-In-Depth. Defense-In-Depth security implements several security mechanisms at each tier (hardware, software, and "people ware") in tandem, to proactively censure an attack that does successfully gain access through one medium. An apt analogy for this sort of strategy is a bulkhead or airlock, whereby once a compartment's integrity is compromised, it can be sealed off from the rest of the system until such a time as repairs can be made. This security paradigm is substantially strong against external attacks, but a knowledgeable agent working from within the system, or with expert knowledge as to the system's functionality, may defeat any aspect of the security system at his or her leisure, at least until changes are applied to the system to render their knowledge obsolete. The best Information Security teams utilize a combination of Defense-In-Depth with Risk Management and Least Privilege implementations.

II. Introduction on Touch-aware Devices

Touch-aware devices are tools that attempt to increase usability, largely of electronic computational devices, through a more intuitive interface of direct pressure manipulation. These devices have slowly been working their way into everyday consumer products, and as individuals have become more acclimated towards utilizing them, demand has increased. The following is a brief overview of the various technologies that are utilized to provide touch awareness.

Resistive touch devices measure a change of electrical current to detect pressure and location of touch. These devices generally have a secondary surface mounted above the display screen, with circuitry at the edges driving a current across. When an individual presses against the surface (such as a finger), this electrical current is modified, and the circuitry can detect that something has changed and triangulate this change down to a specific quadrant. Further innovations in this technology have allowed electronics underneath the device to detect pressure and map it as well; primarily for those pursuing digital artistry, but this methodology does allow the potential for increased security.

Capacitive touch devices function very similarly to resistive touch devices, except they function by measuring the distortion of the field across the screen created by the individual's electrostatic field or a change in the proximity of two electrodes. Cheaper implementations involve detectors placed at the four corners of the screen and a uniform broadcast field across the surface; however, this mechanism, known as *Surface Capacitance,* is prone to failures due to parasitic capacitance, and has limited resolution. More specialized implementations, such as *Mutual Capacitance* (Gerpheide, 1992), compare the effects of altered coupling between row and column electrodes, allowing for a much more precise definition of the touch object, and removing the negative effects of unwanted signaling instructions from corrupting the stream of touch data.

SAW[1] devices utilize ultrasonic waves that are emitted across the surface of the display. Anything placed on the display will change the acoustic wave patterns, causing a measurable change in the frequency, pitch, and/or echo. Running a Fourier analysis allows this change to be calculated to form a position matrix of the surface state, which can then map the surface to the screen and provide this data to the computer or device needing positional awareness (Bergstrom).

Infrared devices can be classified as either "Passive" or "Active" devices. Passive Infrared devices have detectors that are staged across the surface, such that a heat source (finger) emits a much higher electromagnetic signature than the surrounding area. An Active Infrared device, on the other hand, may either reflect infrared light emitted from the device off the object in question (in essence making the object a sort of mirror), or the user puts on a glove or other wearable mesh of infrared LEDs that the detectors can track across the 3D space. Active Infrared devices have become numerous in household devices, such as television remotes and video game controllers, and hobbyist projects have taken advantage of this as a cheap source of parts to develop these sorts of touch displays (Lee, 2008).

Strain Gauge devices utilize a mechanical solution that is then digitally converted with digital force-gauge sensors. Springs inside the screen measure the force and deflection the object touching them is applying to the screen. By careful analysis, and with a fine enough mesh of sensors reporting on the strain produced, software can provide a reasonably accurate touch location, as well as extremely accurate pressure

[1] Surface Acoustic Wave

readings of the touch focus. These touch systems are often implemented in industrial settings, as well as in locations where other sensor networks may experience noise or suffer damage from extended use. Furthermore, due to their resilient nature, they are the primary technology utilized for outside displays, such as those implemented in automatic teller machines (IBM, 1993).

APR[2] is a relatively new system where piezoelectric sensors are positioned underneath the screen to translate the vibration of a pressed finger to electronic signaling instructions. These systems are durable and also exceedingly accurate. They are also one of the least susceptible to external noise effects, including environmental hazards. However, due to the nature and size of the sensors implemented within them, they are also extremely expensive to implement and maintain, and thus are still more of a curiosity than a mainstay.

DST[3] focuses more on the mechanical energy imparted into a touch device when something presses against it. It functions by analyzing the bending waves generated when some object imparts a force against the touch system's physical medium. Complex algorithms are then utilized to analyze these waves and generate the relevant information, such as pressure and location, of the touch. While complex, it allows a variety of touch objects to be detected and is also reliable, as scratches and other potential damaging influences that might significantly impair other technologies can effectively be ignored (3M, 2008).

Optical Imaging systems are one of the cheapest systems to implement, and are the most common hobbyist touch devices. A video camera is positioned inside the screen, facing towards the back of the screen/surface. When a finger or other item is

[2] Acoustic Pulse Recognition
[3] Dispersive Signal Touch

pressed against the screen (such as a stylus), it produces a shadow from the ambient background image, which can be analyzed and a shape, or blob, formed. This blob can then be tracked using software, which assigns a cursor to the blob and maps the size of the background field to the size of the display screen (Kaltenbrunner, Bovermann, Bencina, & Constanza, 2005). Thus, touching the surface in the upper left corner would result in the computing system's cursor also being placed in the upper left corner. While extremely cheap and efficient to implement, lighting conditions and video camera quality can introduce several problems to the analysis that requires recalibrating the device fairly often. However, due to the cheap implementation, this system has been chosen for the multi-touch authentication scheme currently being researched. While lighting conditions may negatively affect the data collection, video post-processing can ameliorate the effects, and the video can then be rerun through the algorithms until a generalized normalization algorithm is found.

Each of these devices may be categorized as either "Single Touch" or "Multi Touch" depending on their hardware abilities as well as the internal software's ability to process the raw data and convert it to meaningful information. Single touch devices have been a mainstay in the industry for many years, and are found in everything from sales kiosks to cellular telephones, as well as some specialty electronics such as industrial automation systems. Multi touch devices have only recently come into vogue, with newer "Smartphone" cellular telephony devices utilizing the technology to great effect, as small surfaces need new methodologies of operation to compensate for their poor resolution. This technology is beginning to migrate towards conventional desktop PCs, as users become more comfortable with manipulating data with operations like "Pinch-Zoom" and gesture-driven macros and start to request these intuitive features for every-day tasks.

Despite all the advantages of touch aware devices and applications, adoption in governmental and enterprise markets is extremely slow. Passwords have existed since timeshares and mainframes; thus, it is a comfortable paradigm for both users and IT security administrators. Touch mechanisms, on the other hand, have largely resided in the dominion of research projects and 'novelty' implementations, without being developed for projects considered more 'serious'. Furthermore, it is difficult to make a comparison between traditional security and touch aware security, as there are difficulties in relating one to the other at a conceptual level. As an example thought problem, is a pattern length equally as strong as a password length, or does the graphical format and the various potentials increase security strength beyond that of conventional implementations? These sorts of questions will only be answered as implementations of touch aware security become more numerous, and this is one of the primary motivations for the experiment described below.

III. Instrumentation and Equipment

For experimenting with multi-touch security implementations, a device was constructed that utilizes a consumer grade video camera system[4] secured inside a box made of .5" Medium Density Fiberboard [**FIGURE 1**]. Near the top is a slot [**FIGURE 2**] that allows for two sheets of Lexan polycarbonate resin thermoplastic with a piece of silk paper sandwiched in between [**FIGURE 3**]. [**FIGURE 4**] has been included to demonstrate the overall profile of the device.

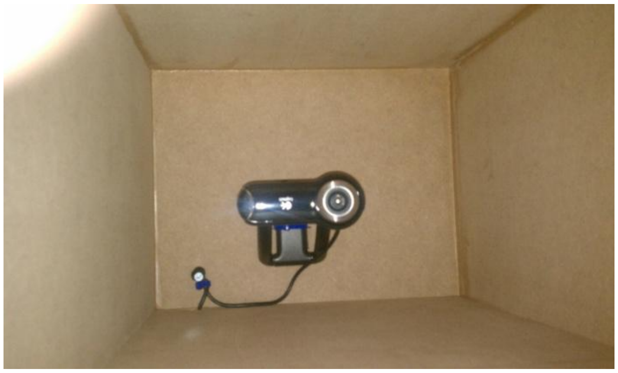


FIGURE 1: Inside View

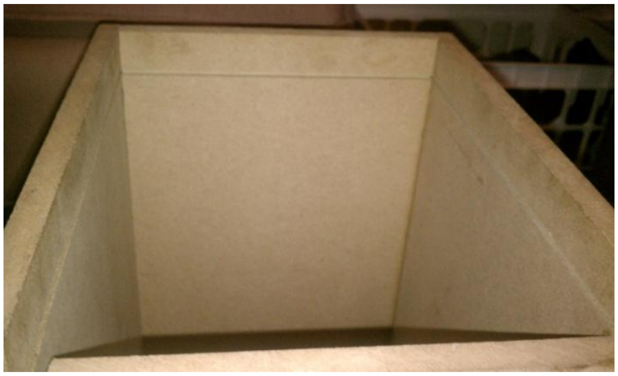


FIGURE 2: Side View of Slot

[4] Logitech Webcam Pro 9000

FIGURE 3: Top View

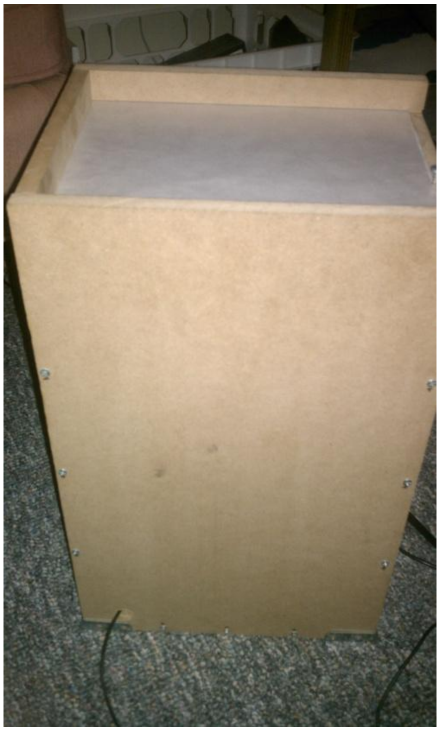


FIGURE 4: Side View of Device

This device is connected via a standardized Universal Serial Bus version 2.0 interface to a computer, which runs several pieces of software to collect the requisite data. First, a program called *Super Webcam* is utilized to split the video feed... it records the video feed utilizing the DivX AVI codec[5]. Audio data is dropped, as it has no relevance to the information collection process. This video file is then converted to QuickTime MOV format utilizing an online file converter, so that it may be played back later. The other feed is accepted by a program called *Community Core Vision*, which is the analysis engine that processes the feed from the Webcam.

IV. Procedure

Community Core Vision performs several steps, which are as follows (NUI Community, 2010):

1. Convert the video input source to gray scale: The software utilizes changes in contrast to identify touch, and thus color information simply introduces noise and unwanted artifacts that challenge the engine.

2. Strip out the base background image: The software then takes a snapshot of the state of the camera's charge coupled device at start, and dynamically subtracts this from the new feed, eliminating noise generated by the default state of the device.

3. Smoothing pass: The software runs anti-aliasing algorithms against the incoming feed to remove jagged edges, which assists in preventing sharp corners from defining their own pointer.

4. High-pass: Removes the less focused elements from the video (aka smoothing), as well as removing elements of noise (random specs) from the source.

[5] Codec specifications are as follows: Codec Preset 3, 1-Pass Operation, Bit-rate defined as 1260.501 kbps, no audio data.

5. Amplify: Strengthens weak pixels to assist in a more well-defined 'blob' AKA cursor.

Once these steps are completed, *Community Core Vision* passes the information through an Inter-Process Communication Pipe on UDP port 3333 utilizing a messaging structure known as TUIO, or *Tangible User Interface Objects* to any aware applications listening (TUIO Protocol Specification 1.1). This messaging structure exposes several pieces of data, the most useful being the ID of the cursor in question, the velocity vector of that cursor, the position of the cursor, and the area on the screen the cursor takes up (utilized to extrapolate the 'pressure'). Moreover, the messaging structure exposes items such as rotational velocity vectors, which define how quickly the cursor is turning, which can be utilized to further identify behavior, as some users may do quick swipes and others small precise turns.

One of the primary reasons *Community Core Vision* was chosen as the primary analysis engine is the engineering of the software. The software allows for a live feed to be analyzed, but it also allows the researcher to play video files that have been previously recorded back into the software. It will interpret these 'offline demos' as if they are being fed in real time, so after a user has been recorded enrolling and then authenticating, his experience can be played back at any time in the future with various changes to settings both within the application and elsewhere, which assists greatly in the development of analysis software. Furthermore, as these demos are recorded in a standardized (if relatively proprietary) codec, they can be included as a data set for other researchers who wish to replicate and elaborate on the research done in this experiment.

A novel approach has been implemented that allows constructed software to understand and parse these objects to implement a Draw A Secret scheme, which presents the user with a pattern of nodes that react to a cursor passing over them (Jermyn, Rubin, Mayer, Monrose, & Reiter, 1999). Through this program, the user may enroll a memorable pattern that future authentication attempts will require them to provide. The differentiator is that this program allows multiple input sources to draw at once... a user may take both pointer fingers and draw two separate patterns across the screen simultaneously or they may take their entire hand and swipe it across quickly. The program stores the path the user traces in an array of integer queues, one queue for each registered cursor object. The queues are sorted, in the hopes of ameliorating the issue of TUIO assigning different cursor IDs between enrollment and subsequent authentication attempts.

Once this data is collected, the program stores it in a text file that is encrypted with the *ManagedRijndael* symmetric block cipher encryption scheme in C#, with a 256-bit secret key and a 128-bit initialization vector. Later authentication schemes decrypt the text only long enough to compare it against the drawn and potentially unauthorized pattern, and it is then discarded, maintaining security even in the face of programs such as memory stream sniffers and .NET de-compilers. The incoming array of authentication data is sorted similar to the mechanism implemented when enrollment information is stored, and then the two arrays of integer queues are compared to determine if they match. If a queue is of different size, the program refuses authentication outright. Alternatively, if the queue contains the right node numbers but the wrong arrangement of node numbers, the program refuses authentication. See FIGURE 5 and FIGURE 6 for two examples of authentication.

In order to determine whether multi touch implementations improve the efficacy of analyzing user behavior, two passes were done with each user… one with a grid at 6x3 and another with a grid at 12x6. The program then will map the points, record the locations of the grid nodes they pass, and utilize that for later authentication. This grid represents the tracked position of the user’s actions; as the mouse cursor (or cursors, if multiple fingers are utilized) tracks across a given node, that node triggers that it has been crossed, and the program records this transition in the data it stores.

Vanilla Draw A Secret patterned password systems are not implicitly a behavioral biometric. However, the eventual goal is to shrink the nodes in the grid from their present macroscopic implementation to an extremely dense microscopic grid. While it may appear to still be a Draw A Secret implementation (colloquially, “what you know”), it in effect evolves into a behavioral biometric through the minute tracking of the user’s various touch inputs. Variation can be programmed in that it would allow deviations up to a certain percentage, allowing for the fact that a user may be very haphazard with his or her individual brushes, strokes, and other touch-related motions interacting with the system.

V. Results and Discussion

The primary goal of this experiment was to ascertain whether utilizing a multi touch aware application performed significantly better than a single touch operation. Draw A Secret security paradigms were utilized as an existing method requiring very little adaptation, and the multi touch data was generated utilizing a 6x3 grid of 50x50 pixel nodes, and then again with a 12x6 grid of 50x50 pixel nodes. Precedent for these sizes exists, as a study showed that users utilizing a single touch pattern had approximately a 92% success rate after 36 days (Biddle, Chiasson, & Oorschot, 2009). The success or failure to authenticate once after registering a pattern was recorded along with the video data; further analysis can occur by feeding the video back into Community Core Vision, which will interpret the video data as a user currently operating the system.

While the currently available data sets are limited, additional_research is planned, including dealing with problems with the present implementation creating too high a noise floor for the experiment to generate meaningful experimental data. Please see **Appendix 2: Collected Data** for the collected data and calculations performed on the data.

While a cursory glance at the data provided by this experiment may lead the reader believe that this would be a poor choice for a security system, there are several considerations that lead us to believe we are on a positive track. As this is preliminary viability screening work, various weaknesses inherent in the present implementation were demonstrated during data collection, and addressing these weaknesses would allow for more effective location registration and discrimination. Also, the experiments to date only tested users who had just enrolled in the trials, and the user only had a couple of minutes to familiarize themselves with the system; a real world implementation as part of a security system would involve system training with users interacting with the device on a day to day basis to provide increased accuracy and precision.

VI. Conclusions

This work was performed to determine whether or not Multi Touch password pattern systems are a worthwhile venture to pursue for further development. As the data above shows, this sort of password schema has significant value to increasing the security of a system, and may also assist with the issue of password overload. Once the system is stabilized such that shadows and other external factors are removed, the grid size can be made denser, until the processing requirements of the grid exceed the system's ability to evaluate. It can then be stepped back, and a balance struck between true secure authentication and the needs to provide computing resources for other programs running on the system.

An alternative and novel solution to correct for the shoulder surfing problem mentioned in the introduction to general security is to utilize research done into Gaze-based Password Entry (Kumar, Garfinkel, Boneh, & Winograd, 2007). In this system, the iris of the eye is tracked and the computer draws a path utilizing this information; eye movements could be bound to cursors and the multi touch implementation could be directed to multiple patterns created serially, rather than all at once.

Another interesting note involves users' starting positions. Most users would place their finger in the middle of the screen and then attempt to ascertain their cursor location, drawing from that point forward. However, users who started at a corner of the device were much more likely to remember their pattern more completely and surprisingly had more complexity to their patterns. It is believed that a corner node is more easily remembered than a node somewhere in the center of the grid, and by concentrating less on starting positions they were able to implement larger and more dynamic patterns, without losing the reliability of recalling it for later authentication. The possibility exists to force the user to start at one of the corner nodes, assisting in later recall as well as user precision; at least two of the experiment participants utilized one of the side walls on the top of the device to steady their hand and guide their movements along the grid.

In light of the data to date, it would seem that Multi Touch Draw A Secret password schemes are possible, and more importantly, individuals who have had no prior experience manipulating them can be instructed and will quickly adapt to their methods. Certainly, there is a learning curve, and certainly, some users will find the system onerous and more difficult to work with than a single touch pattern scheme, or a phrase-based password schema, but the future of computing is with touch interfaces, and security should be adapted to function under these new paradigms before they become mainstream.

VII. Recommendations

We have yet to implement an overlay on the device, a planned improvement. As pictured above **[FIGURE 3**], the top of the surface was a blank sheet of silk paper, which did not depict any of the screen state upon itself. As a result, many users were initially confused due to a sensed disconnect between moving their fingers on the surface and the cursors operating on the screen; several complained that they could not even find where their cursor would start, or had difficulty with precisely manipulating the cursors when they were instructed to draw. Future work aims to address this current limitation.

One proposed solution for the 'disconnect' between user interaction with the device and the manipulation of cursors on the screen involves utilizing transparency film, and sandwiching it between the Lexan polycarbonate resin thermoplastic along with the silk paper. The grid that is utilized on the screen will then be printed off onto this transparency film, thus providing a guide on where the operator needs to place their desired touch item to get the desired action on the computer screen. This is an efficient and cheap option for solving the difficulties of manipulating the touch device, at the expense of flexibility. For each 'grid set-up' involved, it will be necessary to print off another piece of transparency film with that layout. Due to the nature of varying resolutions in today's computing applications, it is not believed to be the ideal approach.

Alternatively, a small screen projection device might be mounted inside the box, situated next to the video camera. It would also be driven via a Universal Serial Bus version 2.0 bus, and would project some form of video data back up against the rear of the touch surface. Due to the rather cramped confines of the container [FIGURE 1], and the limits of technology as of the time of this writing, it is believed that a full thirty two bit color projector capable of rendering out the RGB screen data as a whole is infeasible. Rather, the focus is on utilizing a laser projection system, as simple shapes are all that need to be drawn against the surface to provide a more intuitive utilization scheme. Later developments might forgo the use of a projector, and instead build a grid of light emitting diodes powered by a micro controller; such a device would be flexible enough to provide alternating displays of grids dependent on the data the computer feeds, and might also assist in providing a consistent lighting against the surface such that the present light condition calibrations would no longer be necessary.

Similar to the alienation a user experienced with having to manipulate one device while looking at another is the complaint received that the user is offered too much spurious choice. Due to the way the authentication functions, at present the pattern is only a function of which nodes it crosses. However, several users failed authentication due to their belief that as long as they drew the pattern, it did not matter which nodes ended up lighting up, as they were reasonably within their enrolled motions. One suggestion was_to eliminate the cursor trails altogether, relying on linked nodes to illustrate a user's progress through the grid instead. Reflection on the implementation found in Android branded cellular smart phones revealed that this mechanism is indeed implemented without a true cursor. Secondarily, it was suggested that the grid area be less densely populated, with the populated nodes occupying a greater area and therefore allowing the system more error tolerance in users' enrollment and subsequent authentication.

Future experiments may also involve utilizing different touch technologies as described in Section 4, with the hopes of reducing and eventually eliminating entirely the need to calibrate the device before performing tests. Somewhat coupled with this is the desire to switch to a different processing engine, as *Community Core Vision* requires that all videos be of type QuickTime .mov file format, which is a proprietary container file.

Thus, a fairly large amount of effort was expended in converting the recorded video of the experiments into a format that could then be played back for analysis.

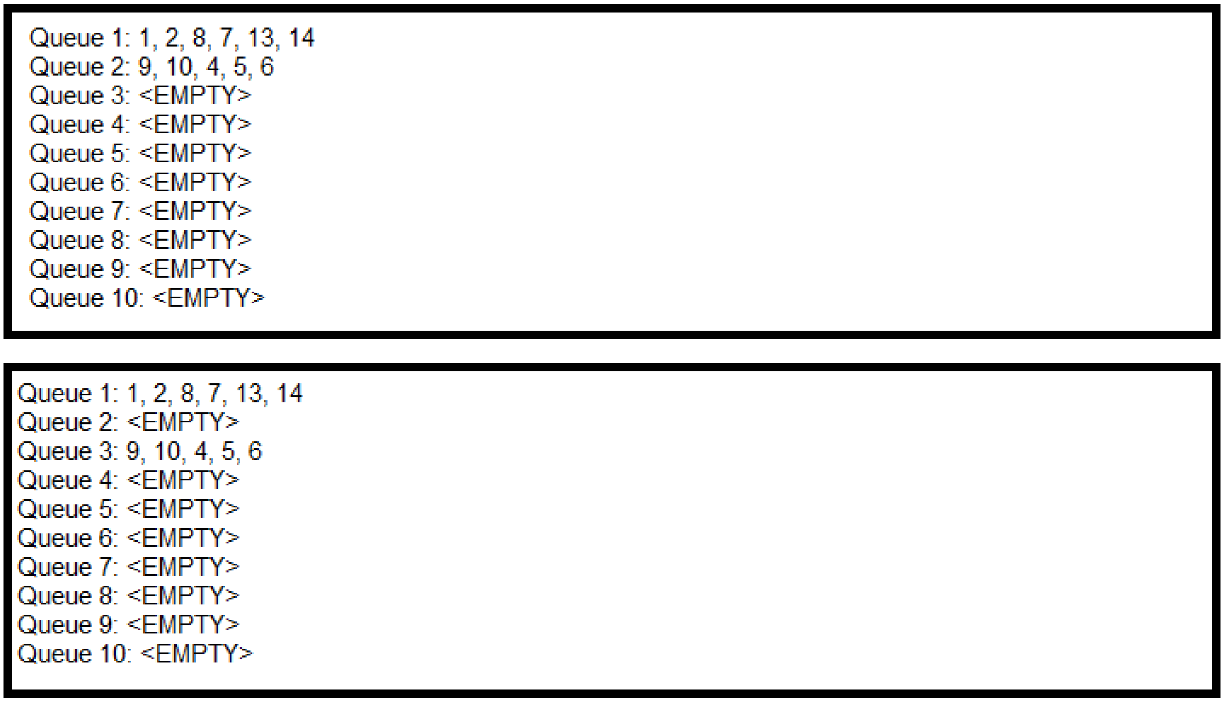
An eventual goal of this project is to develop a cheap and efficient deployable security program for Draw A Secret implementations for Windows and Linux operating systems across the personal computer landscape. No implementation currently exists outside of the one found on cellular smart phones, and it is very likely that if one was developed it would be proprietary and closed source. Furthermore, the implementation on these cellular smart phones is single touch only. Filling this niche would assist with devices such as tablet computers, which have seen large scale deployments across high schools and college campuses in the past several years, and may also make this sort of security more accessible towards others who wish to implement it.

Additional issues existed in the present implementation, where artifacts would confuse the system into creating cursors where none existed. This resulted in nodes triggering outside of the user's effective area of movement, as well as activated nodes registering additional crossings that the user did not commit to. As a result, the user's accuracy fell, and significant challenges presented themselves in the later authentication window, as the user had to replicate these false triggers or else the system would respond negatively.

Ideally, this sort of multi touch aware system would eventually become standard on computer screens and devices that emulate it separately would no longer be required. If or when this comes to pass, it may very well displace the present methods via keyboards and mice, which have long since become inadequate for certain data manipulation tasks. Assuming these systems become standardized across platforms, we believe an opportunity will exist to offer solid security for users of personal systems as well as enterprise and governmental level systems, through a more intuitive and ergonomic interface.

It has been suggested that creating some sort of shooting game in which nodes form and disappear could be advantageous for training. As research has already shown that utilizing a background image improves memory retention for patterns enrolled in the system (Dunphy & Yan, 2007), adding an entertainment factor for the user may further enhance the system. Depending on the image used, Background Draw A Secret systems have equal success rates as vanilla Draw A Secret methodologies, but comparative study of the passwords generated found Background Draw A Secret patterns to be much more complex and effective against brute force attacks (Yampolskiy, 2007). Adding a dimension in the form of time would certainly create a complex biometric in a very short period, hardening the system against all but the most persistent of brute force attacks by increasing the search space necessary to locate a given user's password exponentially.

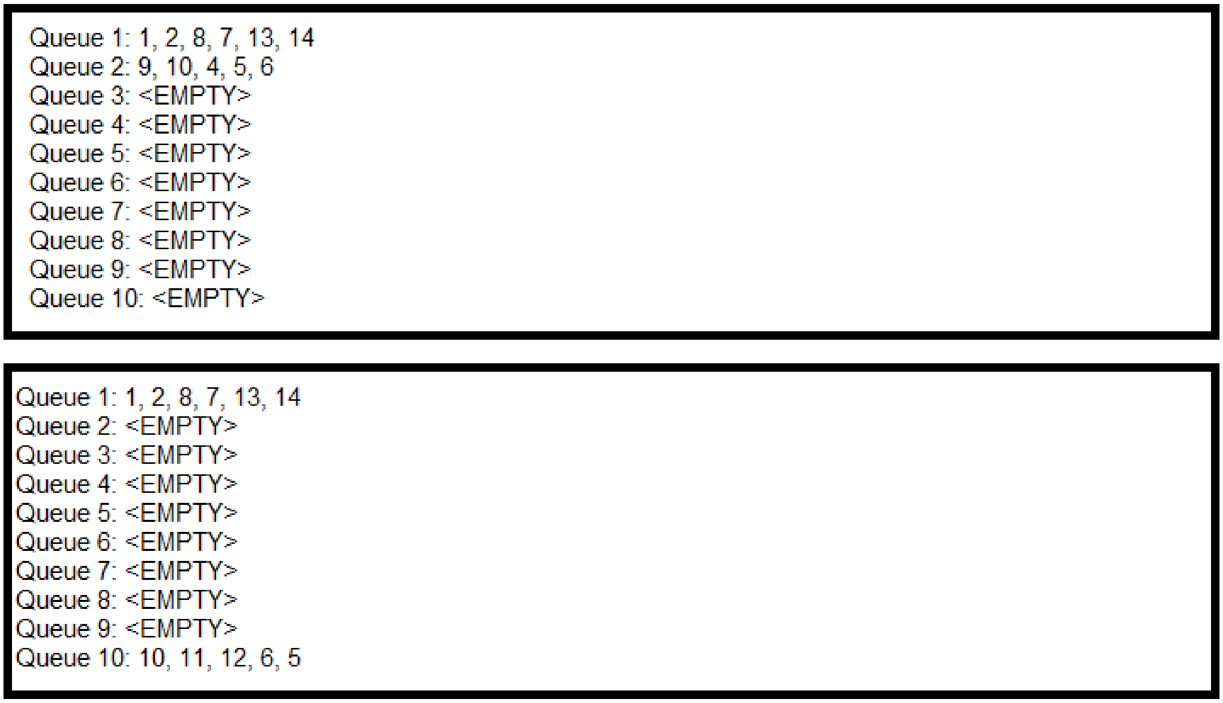
Appendix 1: Authentication Exemplar



```
Queue 1: 1, 2, 8, 7, 13, 14
Queue 2: 9, 10, 4, 5, 6
Queue 3: <EMPTY>
Queue 4: <EMPTY>
Queue 5: <EMPTY>
Queue 6: <EMPTY>
Queue 7: <EMPTY>
Queue 8: <EMPTY>
Queue 9: <EMPTY>
Queue 10: <EMPTY>
```

```
Queue 1: 1, 2, 8, 7, 13, 14
Queue 2: <EMPTY>
Queue 3: 9, 10, 4, 5, 6
Queue 4: <EMPTY>
Queue 5: <EMPTY>
Queue 6: <EMPTY>
Queue 7: <EMPTY>
Queue 8: <EMPTY>
Queue 9: <EMPTY>
Queue 10: <EMPTY>
```

FIGURE 5: Successful Authentication



```
Queue 1: 1, 2, 8, 7, 13, 14
Queue 2: 9, 10, 4, 5, 6
Queue 3: <EMPTY>
Queue 4: <EMPTY>
Queue 5: <EMPTY>
Queue 6: <EMPTY>
Queue 7: <EMPTY>
Queue 8: <EMPTY>
Queue 9: <EMPTY>
Queue 10: <EMPTY>
```

```
Queue 1: 1, 2, 8, 7, 13, 14
Queue 2: <EMPTY>
Queue 3: <EMPTY>
Queue 4: <EMPTY>
Queue 5: <EMPTY>
Queue 6: <EMPTY>
Queue 7: <EMPTY>
Queue 8: <EMPTY>
Queue 9: <EMPTY>
Queue 10: 10, 11, 12, 6, 5
```

FIGURE 6: Failed Authentication

Appendix 2: Collected Data

User	Grid Size	Auth Success[6]	Turn Count[7]	Precision[8]
1	6x3	Yes	2	100%
1	12x6	No	5	73.33%
2	6x3	No	4	40%
2	12x6	No	7	65%

[6] Was the user able to immediately authenticate with their enrolled pattern

[7] The number of times the user changed direction when enrolling their pattern

[8] The number of dots the user 'registered' divided by their traced pattern. Dots that were not in the original pattern are counted against the user

3	6x3	No	4	66.67%
3	12x6	Yes	6	100%
4	6x3	Yes	4	100%
4	12x6	Yes	2	100%
5	6x3	No	4	45%
5	12x6	Yes	8	100%
6	6x3	No	7	83.33%
6	12x6	No	7	30%
7	6x3	Yes	6	100%
7	12x6	No	8	92%
8	6x3	Yes	3	100%
8	12x6	Yes	5	100%
9	6x3	Yes	3	100%
9	12x6	No	7	45%
10	6x3	Yes	2	100%
10	12x6	No	4	66.67%

Mean Accuracy for 6x3: 83.50%

Mean Accuracy for 12x6: 77.2%

Mean Turn Count for 6x3: 3.9 turns

Mean Turn Count for 12x6: 5.9 turns

Success Rate for 6x3: 50%

Success Rate for 12x6: 40%

Appendix 3: Source Code

Encryption.cs

```
//Encryption routines taken from http://waleedelkot.blogspot.com/2009/02/encryption-and-decryption-using-c.html

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Security.Cryptography;

namespace Thesis_Security_App_1._1
{
    class Encryption
    {
        public static string enc(string input)
        {
            return (EncryptString(input));
        }
        public static string dec(string input)
        {
            return (DecryptString(input));
```

```
}

private static string sKey = "UJYHCX783her*&5@$%#(MJCX**38n*#6835ncv56tvbry(&#MX98cn342cn4*&X#&";

protected static string EncryptString(string InputText)

{

    string Password = sKey;

    // "Password" string variable is nothing but the key(your secret key) value which is sent from the front end.

    // "InputText" string variable is the actual password sent from the login page.

    // We are now going to create an instance of the

    // Rihndael class.

    RijndaelManaged RijndaelCipher = new RijndaelManaged();

    // First we need to turn the input strings into a byte array.

    byte[] PlainText = System.Text.Encoding.Unicode.GetBytes(InputText);

    // We are using Salt to make it harder to guess our key

    // using a dictionary attack.

    byte[] Salt = Encoding.ASCII.GetBytes(Password.Length.ToString());

    // The (Secret Key) will be generated from the specified

    // password and Salt.

    //PasswordDeriveBytes -- It Derives a key from a password

    PasswordDeriveBytes SecretKey = new PasswordDeriveBytes(Password, Salt);

    // Create a encryptor from the existing SecretKey bytes.

    // We use 32 bytes for the secret key

    // (the default Rijndael key length is 256 bit = 32 bytes) and
    // then 16 bytes for the IV (initialization vector),
    // (the default Rijndael IV length is 128 bit = 16 bytes)
    ICryptoTransform Encryptor = RijndaelCipher.CreateEncryptor(SecretKey.GetBytes(16), SecretKey.GetBytes(16));
    // Create a MemoryStream that is going to hold the encrypted bytes
    MemoryStream memoryStream = new MemoryStream();
    // Create a CryptoStream through which we are going to be processing our data.
    // CryptoStreamMode.Write means that we are going to be writing data
    // to the stream and the output will be written in the MemoryStream
    // we have provided. (always use write mode for encryption)
    CryptoStream cryptoStream = new CryptoStream(memoryStream, Encryptor, CryptoStreamMode.Write);
    // Start the encryption process.
    cryptoStream.Write(PlainText, 0, PlainText.Length);
    // Finish encrypting.
    cryptoStream.FlushFinalBlock();
    // Convert our encrypted data from a memoryStream into a byte array.
    byte[] CipherBytes = memoryStream.ToArray();
    // Close both streams.
    memoryStream.Close();
    cryptoStream.Close();
    // Convert encrypted data into a base64-encoded string.
    // A common mistake would be to use an Encoding class for that.
    // It does not work, because not all byte values can be
```

```
            // represented by characters. We are going to be using Base64 encoding

            // That is designed exactly for what we are trying to do.

            string EncryptedData = Convert.ToBase64String(CipherBytes);

            // Return encrypted string.

            return EncryptedData;

        }



        protected static string DecryptString(string InputText)

        {

            string Password = sKey;

            try

            {

                RijndaelManaged RijndaelCipher = new RijndaelManaged();

                byte[] EncryptedData = Convert.FromBase64String(InputText);

                byte[] Salt = Encoding.ASCII.GetBytes(Password.Length.ToString());

                PasswordDeriveBytes SecretKey = new PasswordDeriveBytes(Password, Salt);

                // Create a decryptor from the existing SecretKey bytes.

                ICryptoTransform Decryptor = RijndaelCipher.CreateDecryptor(SecretKey.GetBytes(16),
SecretKey.GetBytes(16));

                MemoryStream memoryStream = new MemoryStream(EncryptedData);

                // Create a CryptoStream. (always use Read mode for decryption).

                CryptoStream cryptoStream = new CryptoStream(memoryStream, Decryptor, CryptoStreamMode.Read);

                // Since at this point we don't know what the size of decrypted data
                // will be, allocate the buffer long enough to hold EncryptedData;
                // DecryptedData is never longer than EncryptedData.
                byte[] PlainText = new byte[EncryptedData.Length];
                // Start decrypting.
                int DecryptedCount = cryptoStream.Read(PlainText, 0, PlainText.Length);
                memoryStream.Close();
                cryptoStream.Close();
                // Convert decrypted data into a string.
                string DecryptedData = Encoding.Unicode.GetString(PlainText, 0, DecryptedCount);
                // Return decrypted string.
                return DecryptedData;
            }
            catch (Exception exception)
            {
                return (exception.Message);
            }
        }
    }
}
```

Grid.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;

namespace Thesis_Security_App_1._1
{
    class Grid
    {
        private int x;
        private int y;
        private static int x_mult = 100;
        private static int y_mult = 100;
        private static int x_off = -50;
        private static int y_off = -50;
        private static int img_width = 50;
        private static int img_height = 50;
        private int form_width;
        private int form_height;
        MyPictureBox[,] node;
        private List<int> cursorsExterior;
```

```
public Grid()
{
    node = new MyPictureBox[3, 3];
}
public Grid(int x, int y, TuioDemo f)
{
    node = new MyPictureBox[x, y];
    cursorsExterior = new List<int>();
    drawGrid(f);
}

public bool drawGrid(TuioDemo f)
{
    for (x = 0; x < node.GetLength(0); x++) //x is the column indicator
    {
        for (y = 0; y < node.GetLength(1); y++) //y is the row indicator
        {
            node[x, y] = new MyPictureBox(); //So, node[2][0] is the first row, 3rd element
            ((System.ComponentModel.ISupportInitialize)(node[x, y])).BeginInit();
            node[x, y].Image = global::Thesis_Security_App_1._1.Properties.Resources.Untouched;
            node[x, y].Location = new System.Drawing.Point(((x + 1) * x_mult) + x_off, ((y + 1) * y_mult) + y_off);
            node[x, y].Name = "node:{" + x + "," + y + "}";
            node[x, y].Size = new System.Drawing.Size(img_width, img_height);

            node[x, y].TabIndex = (node.GetLength(1) * y) + x + 4; //The algorithm we are using is
(RowNumber*MaxNumColumns)+currentColumn( + offset so that the buttons are first)

            node[x, y].TabStop = false;

            node[x, y].passes = 0;

            node[x, y].nodenum = (node.GetLength(1) * y) + x;

            f.Controls.Add(node[x, y]);

            ((System.ComponentModel.ISupportInitialize)(node[x, y])).EndInit();

          }

        }

        System.Drawing.Point edge = node[node.GetLength(0)-1, node.GetLength(1)-1].Location;    //Getting bottom
righthand node location

        form_height = edge.Y + 100;

        form_width = edge.X + 100;

        /*form_height = 800;

        form_width = 1280;*/

        //f.ClientSize = new System.Drawing.Size(form_width, form_height);                    //and using it to readjust the
window size

        f.ClientSize = new System.Drawing.Size(1280, 800);

        f.setButtonloc(1, 13, edge.Y + 75);

        f.setButtonloc(2, 105, edge.Y + 75);

        f.setButtonloc(3, 199, edge.Y + 75);

        f.setTextBoxLoc(289, edge.Y + 75);
```

```
    /*f.setButtonloc(1, 13, 700);
    f.setButtonloc(2, 105, 700);
    f.setButtonloc(3, 199, 700);
    f.setTextBoxLoc(289, 700);*/
    return true;
}

public void checkCursor(float cursorX, float cursorY, int cursorID, Queue<int>[] code)
{
    float x_val = (cursorX -(float)x_off) / (float)x_mult;
    float y_val = (cursorY -(float)y_off) / (float)y_mult;
    float wid_max = (float)img_width / (float)x_mult;
    float hth_max = (float)img_height / (float)y_mult;
    int x_id = (int)Math.Floor((double)x_val);
    int y_id = (int)Math.Floor((double)y_val);
    float x_dec = x_val - x_id;
    float y_dec = y_val - y_id;
    MyPictureBox cur_node;

    /*if (x_val > node.GetLength(0) || y_val > node.GetLength(1))
    {
        return;
    }*/

    if (x_dec <= wid_max && y_dec <= hth_max)
    {
        cur_node = node[x_id - 1, y_id - 1];
        if (cur_node.passes < 4 && cursorsExterior.Contains(cursorID))
        {
            cur_node.passes++;
            code[cursorID].Enqueue(cur_node.nodenum);
            updateImage(cur_node);
            cursorsExterior.Remove(cursorID);
        }
    }
    else
    {
        if(!cursorsExterior.Contains(cursorID))
            cursorsExterior.Add(cursorID);
    }
}

public void updateImage(MyPictureBox pb)
{
        switch (pb.passes)
        {
            case 0:
                pb.Image = global::Thesis_Security_App_1._1.Properties.Resources.Untouched;
                break;
            case 1:
                pb.Image = global::Thesis_Security_App_1._1.Properties.Resources.Touched1;
                break;
            case 2:
                pb.Image = global::Thesis_Security_App_1._1.Properties.Resources.Touched2;
                break;
            case 3:
                pb.Image = global::Thesis_Security_App_1._1.Properties.Resources.Touched3;
                break;
            default:
                pb.Image = global::Thesis_Security_App_1._1.Properties.Resources.Bad;
                break;
        }
    }

    public int getWidth()
    {
            return form_width;
        }

        public int getHeight()
        {
            return form_height;
        }

        public MyPictureBox[,] getPBArray()
        {
            return this.node;
        }
    }
}
```

MyPictureBox.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Thesis_Security_App_1._1
```

```
{
    public class MyPictureBox : System.Windows.Forms.PictureBox
    {
        public int passes
        {
            get;
            set;
        }
        public int nodenum
        {
            get;
            set;
        }
    }
}
```

TUIODemo.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Diagnostics;
using System.IO;
using System.Collections;
using System.Threading;
using TUIO;

namespace Thesis_Security_App_1._1
{
    public class TuioDemo : Form, TuioListener
    {
        /// <summary>
        /// Primary logic class for DrawASecret Manipulation
        /// </summary>
        private TuioClient client;
        private Dictionary<long, TuioDemoObject> objectList;
        private Dictionary<long, TuioCursor> cursorList;
        private object cursorSync = new object();
        private object objectSync = new object();
```

```
public static int width;

public static int height;

private int window_width = 640;

private int window_height = 480;

private int window_left = 0;

private int window_top = 0;

private int screen_width = Screen.PrimaryScreen.Bounds.Width;

private int screen_height = Screen.PrimaryScreen.Bounds.Height;



private bool fullscreen;

private bool verbose;



private System.Windows.Forms.Button button1;

private System.Windows.Forms.Button button2;

private System.Windows.Forms.Button button3;

private System.Windows.Forms.TextBox textBox1;



SolidBrush blackBrush = new SolidBrush(Color.Black);

SolidBrush whiteBrush = new SolidBrush(Color.White);



SolidBrush grayBrush = new SolidBrush(Color.Gray);

Pen fingerPen = new Pen(new SolidBrush(Color.Blue), 1);
```

```
Grid mainGrid;
Queue<int>[] code = new Queue<int>[10];
Queue<int>[] prev_code = new Queue<int>[10];

public TuioDemo(int port)
{
 //  InitializeComponent();
  this.button1 = new System.Windows.Forms.Button();
  this.button2 = new System.Windows.Forms.Button();
  this.button3 = new System.Windows.Forms.Button();
  this.textBox1 = new System.Windows.Forms.TextBox();
  InitializeQueueArray(code);
  InitializeQueueArray(prev_code);
  int x = 12;
  int y = 6;
  mainGrid = new Grid(x, y, this);

  //Button 1
  this.button1.Name = "button1";
  this.button1.Size = new System.Drawing.Size(75, 23);
  this.button1.TabIndex = 0;
  this.button1.Text = "Register";

  this.button1.UseVisualStyleBackColor = true;

  this.button1.Click += new System.EventHandler(this.button1_Click);

  //Button 2

  this.button2.Name = "button2";

  this.button2.Size = new System.Drawing.Size(75, 23);

  this.button2.TabIndex = 1;

  this.button2.Text = "Finish";

  this.button2.UseVisualStyleBackColor = true;

  this.button2.Click += new System.EventHandler(this.button2_Click);

  //Button 3

  this.button3.Name = "button3";

  this.button3.Size = new System.Drawing.Size(75, 23);

  this.button3.TabIndex = 2;

  this.button3.Text = "Authenticate";

  this.button3.UseVisualStyleBackColor = true;

  this.button3.Click += new System.EventHandler(this.button3_Click);

  //Textbox

  this.textBox1.ForeColor = System.Drawing.Color.Red;

  this.textBox1.Name = "textBox1";

  this.textBox1.Size = new System.Drawing.Size(286, 20);

  this.textBox1.TabIndex = 3;
  this.Controls.Add(this.textBox1);

  this.Controls.Add(this.button3);

  this.Controls.Add(this.button2);

  this.Controls.Add(this.button1);


  verbose = false;

  fullscreen = false;

  width = mainGrid.getWidth();       //Used important, used for positioning cursors and such

  height = mainGrid.getHeight();     //Used important, used for positioning cursors and such


  this.ClientSize = new System.Drawing.Size(width, height);

  this.Name = "TuioDemo";     //cut

  this.Text = "TuioDemo";     //cut


  this.Closing += new CancelEventHandler(Form_Closing);//keep

  this.KeyDown += new KeyEventHandler(Form_KeyDown);//keep


  this.SetStyle(ControlStyles.AllPaintingInWmPaint |//keep

          ControlStyles.UserPaint |

          ControlStyles.DoubleBuffer, true);
```

```
    objectList = new Dictionary<long, TuioDemoObject>(128);//keep
    cursorList = new Dictionary<long, TuioCursor>(128);//keep

    client = new TuioClient(port);//keep
    client.addTuioListener(this);//keep
    client.connect();//keep
}

public void InitializeQueueArray(Queue<int>[] toInit)
{
    for (int x = 0; x < toInit.GetLength(0); x++)
    {
        toInit[x] = new Queue<int>();
    }
}

private void Form_KeyDown(object sender, System.Windows.Forms.KeyEventArgs e)
{

    if (e.KeyData == Keys.F1)
    {
        if (fullscreen == false)
```

```
{

    width = screen_width;
    height = screen_height;

    window_left = this.Left;
    window_top = this.Top;

    this.FormBorderStyle = FormBorderStyle.None;
    this.Left = 0;
    this.Top = 0;
    this.Width = screen_width;
    this.Height = screen_height;

    fullscreen = true;
}
else
{

    width = window_width;
    height = window_height;

```

```
            this.FormBorderStyle = FormBorderStyle.Sizable;
            this.Left = window_left;
            this.Top = window_top;
            this.Width = window_width;
            this.Height = window_height;

            fullscreen = false;
        }
    }
    else if (e.KeyData == Keys.Escape)
    {
        this.Close();

    }
    else if (e.KeyData == Keys.V)
    {
        verbose = !verbose;
    }

}

private void Form_Closing(object sender, System.ComponentModel.CancelEventArgs e)
```

```
    {
        client.removeTuioListener(this);

        client.disconnect();
        System.Environment.Exit(0);
    }

    public void addTuioObject(TuioObject o)
    {
        lock (objectSync)
        {
            objectList.Add(o.getSessionID(), new TuioDemoObject(o));
        } if (verbose) Console.WriteLine("add obj " + o.getSymbolID() + " (" + o.getSessionID() + ") " + o.getX() + " " +
o.getY() + " " + o.getAngle());
    }

    public void updateTuioObject(TuioObject o)
    {
        lock (objectSync)
        {
            objectList[o.getSessionID()].update(o);
        }
        if (verbose) Console.WriteLine("set obj " + o.getSymbolID() + " " + o.getSessionID() + " " + o.getX() + " " + o.getY()
+ " " + o.getAngle() + " " + o.getMotionSpeed() + " " + o.getRotationSpeed() + " " + o.getMotionAccel() + " " +
o.getRotationAccel());
    }

    public void removeTuioObject(TuioObject o)
    {
        lock (objectSync)
        {
            objectList.Remove(o.getSessionID());
        }
        if (verbose) Console.WriteLine("del obj " + o.getSymbolID() + " (" + o.getSessionID() + ")");
    }

    public void addTuioCursor(TuioCursor c)
    {
        lock (cursorSync)
        {
            cursorList.Add(c.getSessionID(), c);
        }
        if (verbose) Console.WriteLine("add cur " + c.getCursorID() + " (" + c.getSessionID() + ") " + c.getX() + " " +
c.getY());
    }
```

```
    public void updateTuioCursor(TuioCursor c)
    {
        if (verbose) Console.WriteLine("set cur " + c.getCursorID() + " (" + c.getSessionID() + ") " + c.getX() + " " + c.getY() + " " + c.getMotionSpeed() + " " + c.getMotionAccel());
        mainGrid.checkCursor((c.getX() * TuioDemo.width), (c.getY() * TuioDemo.height), c.getCursorID(), code);
    }


    public void SortCode() //Consider upgrading to level 2 or level 3, also currently won't deal with two patterns starting at the same node.
    {
    int cur = 0;
    int nxt = 0;
    bool flg = true;
    Queue<int> tmp;
    while (flg)
    {
        flg = false;
        for (int x = 0; x < code.GetLength(0) - 1; x++)
        {
            if (code[x].Count < 1 || code[x + 1].Count < 1) break;
            cur = code[x].Peek();
            nxt = code[x + 1].Peek();
            if (cur > nxt)
            {
                tmp = code[x];
                code[x] = code[x + 1];
                code[x + 1] = tmp;
                flg = true;
            }
        }
    }
    }

    public void WriteOutCode(string filepath)
    {
        /// <summary>
        /// Creates the file to store the authentication information in
        /// and encrypts it.
        /// </summary>
        /// <param name="filepath">
        /// The full file path to write out to.
        /// </param>
        FileStream fs = new FileStream(filepath, FileMode.Create);
        StreamWriter sw = new StreamWriter(fs);
        StringBuilder sb = new StringBuilder();
        SortCode();
        for (int x = 0; x < code.GetLength(0); x++)
        {
            while(code[x].Count>0)
            {
                sb.Append(code[x].Dequeue()+",");
            }
            sb.Append("\n,");
        }
        Debug.Write(sb.ToString());
        sw.Write(Encryption.enc(sb.ToString()));
        sw.Close();
    }

    public bool ReadInCode(string filepath)
    {
        /// <summary>
        /// Reads from the file of previously recorded authentication
        /// information and decrypts it.
        /// </summary>
        /// <param name="filepath">
        /// The full file path to read in from.
        /// </param>
        FileStream fs = new FileStream(filepath, FileMode.Open);
        StreamReader sr = new StreamReader(fs);
        if (!fs.CanRead)
        {
            return false;
        }
        else
        {
            string[] comp = Encryption.dec(sr.ReadLine()).Split(',');
            int x = 0; //Used to keep track of which 'cursor' we're reading in.
            foreach (string value in comp)
            {
                if (value.CompareTo("\n")==0)
                {
                    x++;
                }
                else if (value.CompareTo("") == 0)
                {
                    break;
                }
                else
```

```
            {

                prev_code[x].Enqueue(Convert.ToInt16(value));

            }

        }

        return true;

    }

}



public void removeTuioCursor(TuioCursor c)

{

    lock (cursorSync)

    {

        cursorList.Remove(c.getSessionID());

    }

    if (verbose) Console.WriteLine("del cur " + c.getCursorID() + " (" + c.getSessionID() + ")");

}



public void refresh(TuioTime frameTime)

{

    Invalidate();

}
```

```
        protected override void OnPaintBackground(PaintEventArgs pevent)
        {
            // Getting the graphics object
            Graphics g = pevent.Graphics;
            g.FillRectangle(whiteBrush, new Rectangle(0, 0, width, height));

            // draw the cursor path
            if (cursorList.Count > 0)
            {
                lock (cursorSync)
                {
                    foreach (TuioCursor tcur in cursorList.Values)
                    {
                        List<TuioPoint> path = tcur.getPath();
                        TuioPoint current_point = path[0];

                        for (int i = 0; i < path.Count; i++)
                        {
                            TuioPoint next_point = path[i];
                            g.DrawLine(fingerPen, current_point.getScreenX(width), current_point.getScreenY(height),
next_point.getScreenX(width), next_point.getScreenY(height));
                            current_point = next_point;
                        }
                        g.FillEllipse(grayBrush, current_point.getScreenX(width) - height / 100, current_point.getScreenY(height) -
height / 100, height / 50, height / 50);
                        Font font = new Font("Arial", 10.0f);
                        g.DrawString(tcur.getCursorID() + "", font, blackBrush, new PointF(tcur.getScreenX(width) - 10,
tcur.getScreenY(height) - 10));
                    }
                }
            }

            // draw the objects
            if (objectList.Count > 0)
            {
                lock (objectSync)
                {
                    foreach (TuioDemoObject tobject in objectList.Values)
                    {
                        tobject.paint(g);
                    }
                }
            }
        }

        public static void Main(String[] argv)
```

```
{
    int port = 0;
    switch (argv.Length)
    {
        case 1:
            port = int.Parse(argv[0], null);
            if (port == 0) goto default;
            break;
        case 0:
            port = 3333;
            break;
        default:
            Console.WriteLine("usage: java TuioDemo [port]");
            System.Environment.Exit(0);
            break;
    }

    TuioDemo app = new TuioDemo(port);
    Application.Run(app);

}
```

```
private void InitializeComponent()
{
    this.SuspendLayout();
    this.ClientSize = new System.Drawing.Size(292, 266);
    this.Name = "TuioDemo";
    this.ResumeLayout(false);
}


public void setButtonloc(int buttID, int x_loc, int y_loc)
{
    switch (buttID)
    {
        case 1:
            {
                this.button1.Location = new System.Drawing.Point(x_loc, y_loc);
                break;
            }
        case 2:
            {
                this.button2.Location = new System.Drawing.Point(x_loc, y_loc);
                break;
            }
        case 3:

            {

                this.button3.Location = new System.Drawing.Point(x_loc, y_loc);

                break;

            }

    }

}



public void setTextBoxLoc(int x_loc, int y_loc)

{

    this.textBox1.Location = new System.Drawing.Point(x_loc, y_loc);

}



//int m = 0;

//int s = 0;



private void button1_Click(object sender, EventArgs e)

{

    textBox1.ForeColor = System.Drawing.Color.Aqua;

    Random r = new Random();

    /*int t = (int)(r.Next(0, 100));

    if(t<50)
    {

        textBox1.Text = "Registering Multi-Touch. Please draw at least two patterns.";

        m = 1;

    } else

    {

        textBox1.Text = "Registering Single-Touch. Please draw only one patterns.";

        s = 1;

    }*/

    ResetGrid();

    button1.Enabled = false;

    button2.Enabled = true;

    button3.Enabled = false;

}



private void button2_Click(object sender, EventArgs e)

{

    WriteOutCode(@"C:\temp\code.txt");

    button2.Enabled = false;

    button1.Enabled = true;

    button3.Enabled = true;

    textBox1.ForeColor = System.Drawing.Color.Green;

    textBox1.Text = "Your pattern has been registered.";
```

```
ResetGrid();

InitializeQueueArray(code);

/*#region Multi-Touch

if (m == 1)

{

    //Force multi-touch enrollment only

    bool oneTouch = false;

    bool isMulti = false;

    for (int x = 0; x < 10; x++)

    {

        if (code[x].Count() > 0 && !oneTouch)

        {

            oneTouch = true;

        }

        else if (code[x].Count() > 0 && oneTouch)

        {

            isMulti = true;

        }

    }

    if (isMulti)

    {

        WriteOutCode(@"C:\temp\code.txt");
        button2.Enabled = false;

        button1.Enabled = true;

        button3.Enabled = true;

        textBox1.ForeColor = System.Drawing.Color.Green;

        textBox1.Text = "Your pattern has been registered.";

        ResetGrid();

        InitializeQueueArray(code);

        m = 0;

        s = 0;

    }

    else

    {

        textBox1.ForeColor = System.Drawing.Color.Red;

        textBox1.Text = "Error. Must have more than one pattern registered.";

        button2.Enabled = false;

        button1.Enabled = true;

        button3.Enabled = true;

        m = 0;

        s = 0;

    }

}

#endregion
```

```
#region Single-Touch

if (s == 1)

{

    //Force single-touch enrollment only

    bool oneTouch = false;

    bool isSingle = true;

    for (int x = 0; x < 10; x++)

    {

        if (code[x].Count() > 0 && !oneTouch)

        {

            oneTouch = true;

        }

        else if (code[x].Count() > 0 && oneTouch)

        {

            isSingle = false;

        }

    }

    if (isSingle)

    {

        WriteOutCode(@"C:\temp\code.txt");

        button2.Enabled = false;

        button1.Enabled = true;
```

```
                button3.Enabled = true;
                textBox1.ForeColor = System.Drawing.Color.Green;
                textBox1.Text = "Your pattern has been registered.";
                ResetGrid();
                InitializeQueueArray(code);
                m = 0;
                s = 0;
            }
            else
            {
                textBox1.ForeColor = System.Drawing.Color.Red;
                textBox1.Text = "Error. Must not utilize more than one pattern.";
                button2.Enabled = false;
                button1.Enabled = true;
                button3.Enabled = true;
                m = 0;
                s = 0;
            }
        }
        #endregion*/
    }
```

```
        private void button3_Click(object sender, EventArgs e)
        {
            if (button1.Enabled == true) //If the Set button is enabled, we know we are not in the middle of authentication
            {
                ResetGrid();
                InitializeQueueArray(prev_code);
                bool AuthExist = ReadInCode(@"C:\temp\code.txt");
                if (!AuthExist)
                {
                    textBox1.ForeColor = System.Drawing.Color.Red;
                    textBox1.Text = "Error. No pattern set.";
                }
                else
                {
                    button1.Enabled = false;
                    textBox1.ForeColor = System.Drawing.Color.Aqua;
                    textBox1.Text = "Authenticating...";
                }
            }
            else //If the set button is disabled, the Authenticate button has already been pressed once, and now we need to check against our saved value.
            {
                bool valid = true;
```

```
button1.Enabled = true;
button2.Enabled = false;



SortCode();
for (int x = 0; x < code.GetLength(0); x++)
{
    if (code[x].Count != prev_code[x].Count)
    {
        valid = false;
        textBox1.ForeColor = System.Drawing.Color.Red;
        textBox1.Text = "Non-matching Pattern";
    }
    else
    {
        while (code[x].Count > 0)
        {
            if (code[x].Dequeue() != prev_code[x].Dequeue())
            {
                valid = false;
                textBox1.ForeColor = System.Drawing.Color.Red;
                textBox1.Text = "Non-matching Pattern";
```

```
                        break;
                    }
                }
            }
            if (!valid) break;
        }
        if (valid)
        {
            textBox1.ForeColor = System.Drawing.Color.Green;
            textBox1.Text = "Authenticated!";
        }
        InitializeQueueArray(code);
    }
}

public void ResetGrid()
{
    IEnumerable pbEnum = mainGrid.getPBArray() as IEnumerable;
    if (pbEnum != null)
    {
        foreach (MyPictureBox pb in pbEnum)
        {
```

```
                    pb.Image = global::Thesis_Security_App_1._1.Properties.Resources.Untouched;
                    pb.passes = 0;
                }
            }
        }
    }
}
```

TUIODemoObject.cs

```
/*
                TUIO C# Demo - part of the reacTIVision project
                http://reactivision.sourceforge.net/

                Copyright (c) 2005-2009 Martin Kaltenbrunner <mkalten@iua.upf.edu>

                This program is free software; you can redistribute it and/or modify
                it under the terms of the GNU General Public License as published by
                the Free Software Foundation; either version 2 of the License, or
                (at your option) any later version.

                This program is distributed in the hope that it will be useful,
                but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
        GNU General Public License for more details.

        You should have received a copy of the GNU General Public License
        along with this program; if not, write to the Free Software
        Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

using System;
using System.Drawing;
using TUIO;


namespace Thesis_Security_App_1._1
{

    public class TuioDemoObject : TuioObject
    {

        SolidBrush black = new SolidBrush(Color.Black);
        SolidBrush white = new SolidBrush(Color.White);

```

```
public TuioDemoObject(long s_id, int f_id, float xpos, float ypos, float angle)
    : base(s_id, f_id, xpos, ypos, angle)
{
}

public TuioDemoObject(TuioObject o)
    : base(o)
{
}

public void paint(Graphics g)
{
    int Xpos = (int)(xpos * TuioDemo.width);
    int Ypos = (int)(ypos * TuioDemo.height);
    int size = TuioDemo.height / 10;

    g.TranslateTransform(Xpos, Ypos);
    g.RotateTransform((float)(angle / Math.PI * 180.0f));
    g.TranslateTransform(-1 * Xpos, -1 * Ypos);

    g.FillRectangle(black, new Rectangle(Xpos - size / 2, Ypos - size / 2, size, size));
```

```
            g.TranslateTransform(Xpos, Ypos);

            g.RotateTransform(-1 * (float)(angle / Math.PI * 180.0f));

            g.TranslateTransform(-1 * Xpos, -1 * Ypos);



            Font font = new Font("Arial", 10.0f);

            g.DrawString(symbol_id + "", font, white, new PointF(Xpos - 10, Ypos - 10));

        }



    }

}
```

List of References

3M. (2008). Touchscreen Technologies. Retrieved September 24, 2010, from 3M United States: Products, Brands and Technologies: http://multimedia.3m.com/mws/mediawebserver?mwsId=66666UuZjcFSLXTtmxTXoxfaEVuQEcuZgVs6EVs6E666666--

Ahmed, A. A., & Traore, I. (July-Sept 2007). A New Biometric Technology Based on Mouse Dynamics. IEEE Transactions on Dependable and Secure Computing , 165-179.

Barkley, J. (1995, January 9). Role Based Access Control: Principle of Least Privilege. Retrieved September 22, 2010, from National Institute of Standards and Technology: Computer Security Division: http://hissa.ncsl.nist.gov/rbac/paper/node5.html

Gerpheide, G. E. (1992). Patent No. 5,305,017. United States.

Hanna, D. (2006, October 3). Day 18 - Mouse Heat Map. Retrieved September 21, 2010, from An App A Day: http://www.anappaday.com/downloads/2006/10/day-18-mouse-heat-map.html

IBM. (1993, December 1). Strain Gauge for Touch-Screen Sensing. Prior Art Database: IPCOM000106753D.

Jermyn, I., Rubin, A. D., Mayer, A., Monrose, F., & Reiter, M. K. (1999). The Design and Analysis of Graphical Passwords. Proceedings of the 8th Usenix Security Symposium, (pp. 1-14). Washington, D.C.

Kaltenbrunner, M., Bovermann, T., Bencina, R., & Constanza, E. (2005). TUIO - A Protocol for Table-Top Tangible User Interfaces. Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation. Vannes.

Kumar, M., Garfinkel, T., Boneh, D., & Winograd, T. (2007). Reducing Shoulder-surfing by Using Gaze-based Password Entry. Symposum On Usable Privacy and Security (SOUPS). Pittsburgh.

Lee, J. C. (2008, August 20). Johnny Chung Lee - Projects - Wii. Retrieved September 24, 2010, from Johnny Chung Lee - Human Computer Interaction Research: http://johnnylee.net/projects/wii/

Monrose, F. (2000, March). User authentication. Retrieved September 21, 2010, from Department of Computer Science, UNC-Chapel Hill: http://cs.unc.edu/~fabian/Authentication.html

National Security Agency. (n.d.). Defense in Depth: A Practical Strategy for Achieving Information Assurance. Retrieved September 21, 2010, from National Security Agency Central Security Service: http://www.nsa.gov/ia/_files/support/defenseindepth.pdf

NSTC Subcommittee on Biometrics. (2006, August 7). Biometrics.gov - Introduction to Biometrics. Retrieved September 21, 2010, from Biometrics.gov: http://www.biometrics.gov/Documents/SpeakerRec.pdf

NUI Community. (2010, July 9). Getting Started with CCV. Retrieved September 21, 2010, from NUI Group Community Wiki: http://wiki.nuigroup.com/Getting_Started_with_tbeta#Overview_Diagram

TUIO Protocol Specification 1.1. (n.d.). Retrieved September 21, 2010, from TUIO.org: http://www.tuio.org/?specification

Griffin, D. (2008, May). Safer Authentication with a One-Time Password Solution. MSDN Magazine, pp. http://msdn.microsoft.com/en-us/magazine/cc507635.aspx.

Schneider, F. B. (2005, September). Something You Know, Have, or Are. Retrieved 3 23, 2011, from CS 513 System Security: http://www.cs.cornell.edu/courses/cs513/2005fa/nnlauthpeople.html

Subcommittee on Biometrics. (2006, August 7). Palm Print Recognition. Retrieved March 24, 2011, from Biometrics.gov - Introduction to Biometrics: http://www.biometrics.gov/Documents/PalmPrintRec.pdf

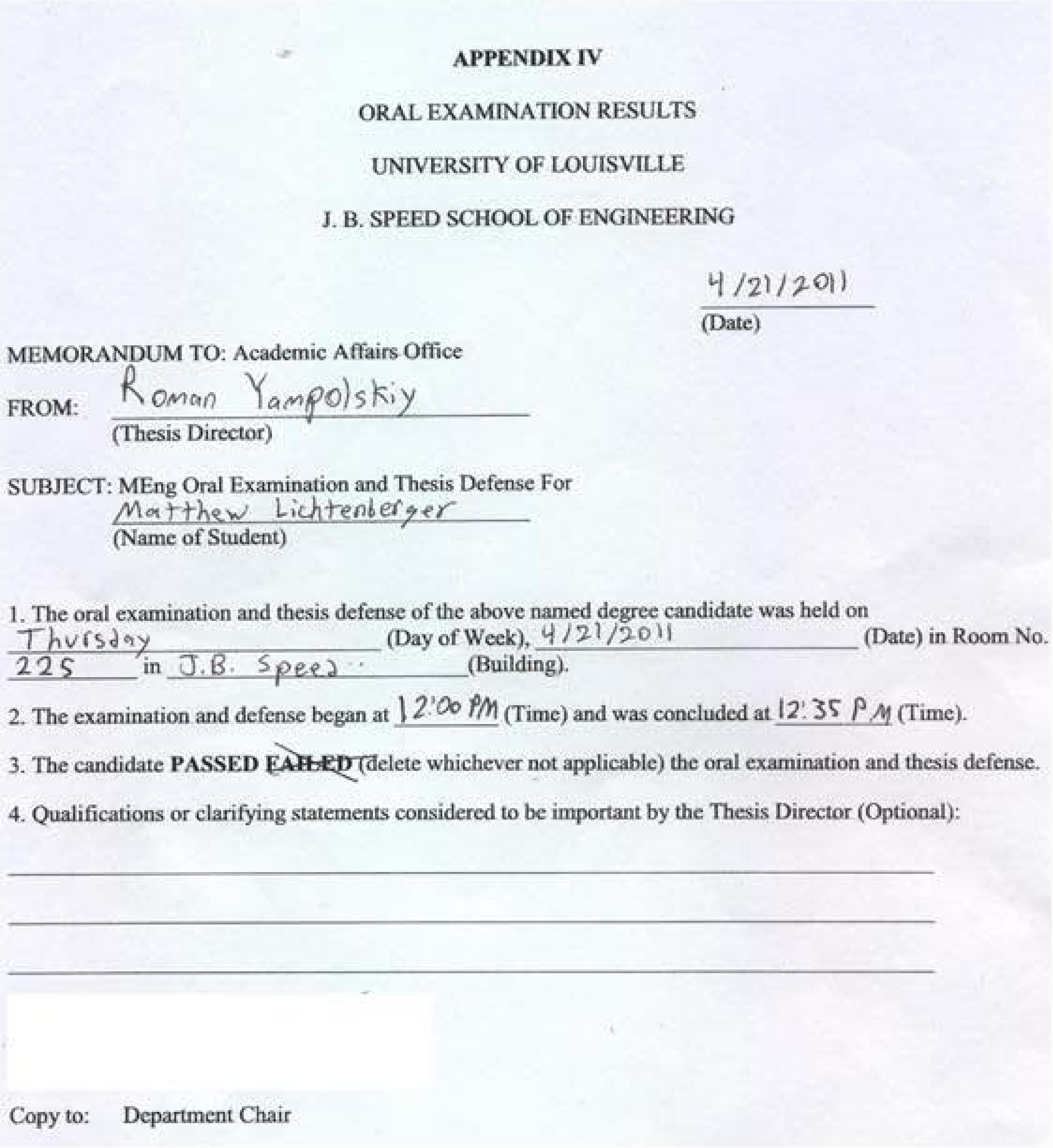
Thomas, B. (2005, May 17). Simple Formula for Strong Passwords (SFSP). Retrieved March 24, 2011, from SANS InfoSec Reading Room: http://www.sans.org/reading_room/whitepapers/authentication/simple-formula-strong-passwords-sfsp-tutorial_1636

Thornton, J. (2000). Setting Standards In The Comparison and Identification. 84th Annual Training Conference of the California State Division of IAI. Laughlin: IAI.

Yampolskiy, R. (2007). User Authentication via Behavior Based Passwords. IEEE Explore.

Vita

Matthew Lichtenberger is a Computer Engineering graduate from the Speed School of Engineering in Louisville, KY. He has studied computers for most of his life, and finds a particular passion in developing practical solutions for problems in business. Currently, he is pursuing a business initiative with his colleague Paul Frederick (also an alumnus of Speed School) in assisting Non-Profit community businesses with their IT infrastructure. His hobbies include video games, collecting gadgets, and writing science-fiction prose.



APPENDIX IV

ORAL EXAMINATION RESULTS

UNIVERSITY OF LOUISVILLE

J. B. SPEED SCHOOL OF ENGINEERING

4/21/2011
(Date)

MEMORANDUM TO: Academic Affairs Office

FROM: Roman Yampolskiy
(Thesis Director)

SUBJECT: MEng Oral Examination and Thesis Defense For
Matthew Lichtenberger
(Name of Student)

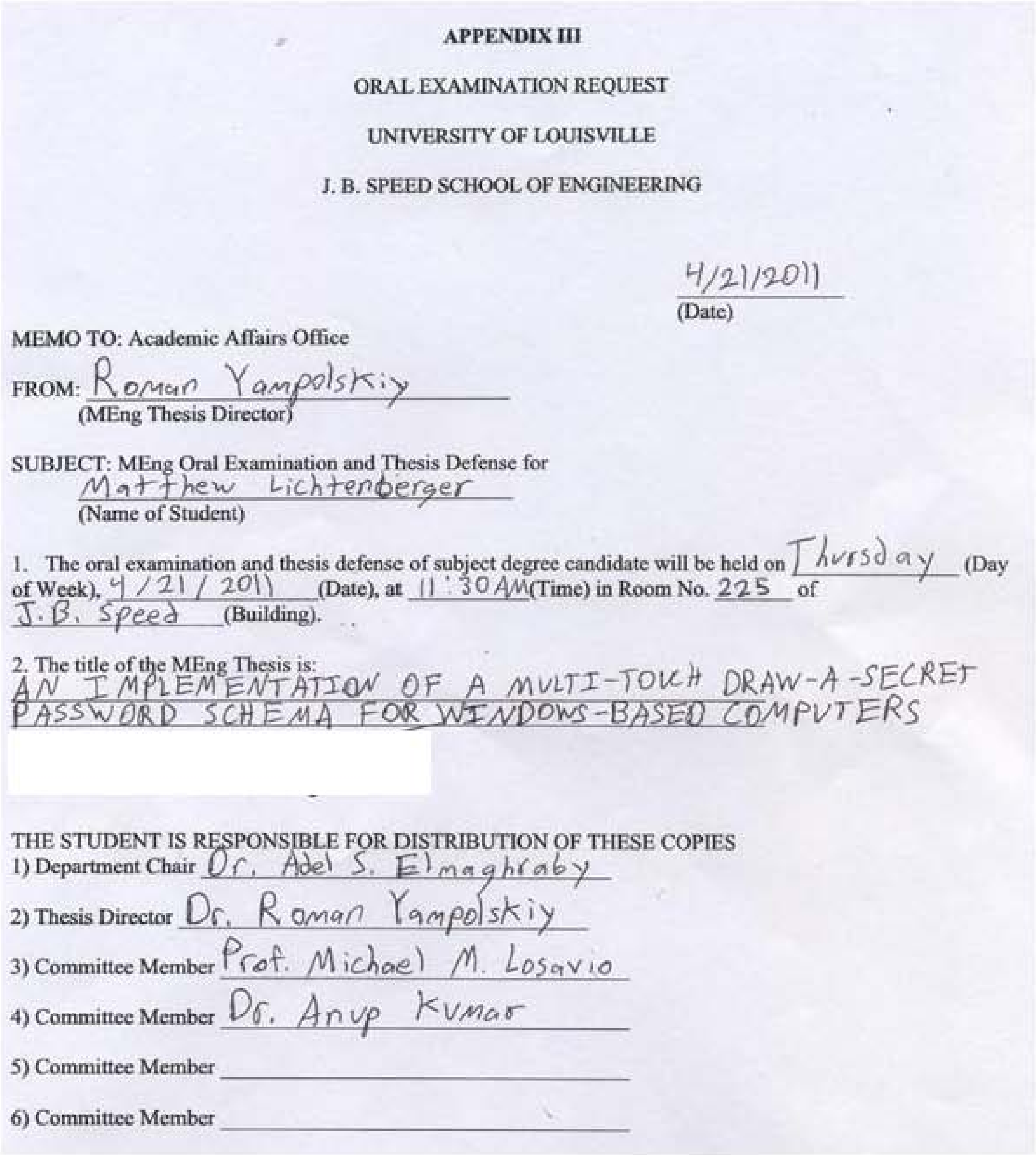
1. The oral examination and thesis defense of the above named degree candidate was held on Thursday (Day of Week), 4/21/2011 (Date) in Room No. 225 in J.B. Speed (Building).

2. The examination and defense began at 12:00 PM (Time) and was concluded at 12:35 PM (Time).

3. The candidate **PASSED** ~~**FAILED**~~ (delete whichever not applicable) the oral examination and thesis defense.

4. Qualifications or clarifying statements considered to be important by the Thesis Director (Optional):

Copy to: Department Chair



APPENDIX III

ORAL EXAMINATION REQUEST

UNIVERSITY OF LOUISVILLE

J. B. SPEED SCHOOL OF ENGINEERING

4/21/2011
(Date)

MEMO TO: Academic Affairs Office

FROM: Roman Yampolskiy
(MEng Thesis Director)

SUBJECT: MEng Oral Examination and Thesis Defense for
Matthew Lichtenberger
(Name of Student)

1. The oral examination and thesis defense of subject degree candidate will be held on Thursday (Day of Week), 4 / 21 / 2011 (Date), at 11:30 AM (Time) in Room No. 225 of J.B. Speed (Building).

2. The title of the MEng Thesis is:
AN IMPLEMENTATION OF A MULTI-TOUCH DRAW-A-SECRET PASSWORD SCHEMA FOR WINDOWS-BASED COMPUTERS

THE STUDENT IS RESPONSIBLE FOR DISTRIBUTION OF THESE COPIES

1) Department Chair Dr. Adel S. Elmaghraby

2) Thesis Director Dr. Roman Yampolskiy

3) Committee Member Prof. Michael M. Losavio

4) Committee Member Dr. Anup Kumar

5) Committee Member ______________________

6) Committee Member ______________________